МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Пензенский государственный университет» (ПГУ)

Н. П. Вашкевич, Р. А. Бикташев

НЕДЕТЕРМИНИРОВАННЫЕ АВТОМАТЫ И ИХ ИСПОЛЬЗОВАНИЕ ДЛЯ РЕАЛИЗАЦИИ СИСТЕМ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ИНФОРМАЦИИ

Монография

Пенза
Издательство ПГУ
2016

УДК 519.713
В23

Р е ц е н з е н т ы:
кафедра «Вычислительная техника»
Национального исследовательского университета «МЭИ»;
Заслуженный деятель науки Республики Татарстан
и Российской Федерации, доктор технических наук, профессор кафедры
«Компьютерные системы» Казанского национального исследовательского
технического университета им. А. Н. Туполева – КАИ
В. А. Песошин

Вашкевич, Н. П.

В23 Недетерминированные автоматы и их использование для реализации систем параллельной обработки информации : моногр. / Н. П. Вашкевич, Р. А. Бикташев. – Пенза : Изд-во ПГУ, 2016. – 394 с.

ISBN 978-5-906831-93-4

Рассмотрен основной математический аппарат, связанный с формальным представлением алгоритмов управления параллельной обработки информации с использованием модели недетерминированных автоматов (НДА), и даны эквивалентные преобразования такой модели. В качестве основной базовой модели НДА принята система рекуррентных канонических уравнений (СКУ), описывающая все реализуемые в системе управления частные события и язык граф-схем алгоритмов с параллельными ветвями (ГСАП). Эквивалентные преобразования модели НДА базируются на использовании операций детерминизации, минимизации, кодирования и компиляции НДА. Рассмотрены некоторые начальные языки представления управляющих алгоритмов и их использования для формального описания алгоритмов и методы их преобразования на язык СКУ. Формальное представление алгоритмов взаимодействия параллельными процессами иллюстрируется на базе решения классических задач управления процессами при обращении к общему ресурсу. Рассмотрены также методы структурной реализации и верификации алгоритмов управления процессами и ресурсами в многопроцессорных системах, представленных автоматными моделями НДА и ДА.

Книга предназначена для широкого круга специалистов, а также может быть использована студентами направления 09.03.01 «Информатика и ВТ» и направления 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения», студентами бакалавриата, магистрантами и аспирантами всех специальностей направления «Информатика и ВТ» при изучении ими родственных дисциплин и при выполнении выпускных и научных работ.

УДК 519.713

Рекомендовано к изданию научно-техническим советом Пензенского государственного университета (*протокол № 14 от 17.12.2015 г.*)

ISBN 978-5-906831-93-4

ПРЕДИСЛОВИЕ

Повышение производительности вычислительных машин и систем автоматики на их основе во многом определяется эффективностью организации логического управления в системах, учитывающих реализацию параллелизма на всех уровнях управления. Особенно это относится к мультипроцессорным системам, в том числе к системам реального времени, предназначенным для управления в робототехнических системах, когда возникают задачи управления параллельными взаимодействующими процессами и ресурсами. При создании таких систем одной из актуальных задач является разработка формальных методов описания алгоритмов управления параллельными взаимодействующими процессами, их эквивалентных преобразований и методов их структурной реализации на разных уровнях.

Содержание монографии посвящено вопросам синтеза таких систем логического управления, которые базируются на использовании моделей недетерминированных конечных автоматов (НДА) и их стандартной аналитической форме представления в виде системы рекуррентных канонических уравнений, описывающих все реализуемые в системе управления частные события.

В такой математической модели НДА, в отличие от модели детерминированного автомата (ДА), предложено использовать представления функций переходов в автомате не в терминах состояний, а в терминах частных событий, реализуемых в автомате. При этом каждому НДА может быть поставлен в соответствие эквивалентный ему детерминированный автомат, в котором состояния в общем случае будут представлены совокупностью событий, имеющих место в один и тот же момент времени в НДА, т.е. ДА является частным случаем НДА, в котором не соблюдаются условия однозначности переходов, проявляющейся в том, что под действием одного и того же входного сигнала возможен переход от некоторого исходного события к нескольким событиям, совокупность которых и будет представлять собой одно из состояний ДА.

Таким образом, «недетерминированность» автомата нельзя смешивать со «случайностью», характеризующей вероятностный автомат, в котором он может перейти в одно из следующих состояний с фиксированными вероятностями [6]. Указанная особенность модели НДА и определяет «скрытый» в ней параллелизм.

Такой подход к формализации сложных параллельных алгоритмов управления с взаимодействующими ветвями позволяет создать набор методов и средств для компактного формального описания алгоритмов управления и их эквивалентных преобразований для построения возможных реализаций с учетом требований повышения производительности, надежности и уменьшения аппаратурных затрат.

Математические модели устройств управления, принятые в работе и основанные на использовании концепции параллелизма и недетерминизма, являются достаточно простыми и универсальными, позволяющими их использовать для широкого круга применения как по управлению процессами и ресурсами, так и для моделирования событий, начиная от простейших устройств и кончая функциями управления операционных систем.

В первых трех главах монографии рассмотрены основные понятия и определения из теории недетерминированных автоматов (НДА) и методы их эквивалентных преобразований на основе операций детерминизации, минимизации, кодирования и композиции.

В четвертой главе рассматриваются начальные языки, используемые для представления управляющих алгоритмов параллельной обработки. К ним относятся: язык регулярных выражений алгебры событий (РВАС), язык исчисления предикатов первого порядка с ограниченными кванторами и его связь с языком РВАС, язык операторных схем алгоритмов с параллельными ветвями. Рассматриваются также и обычные (непараллельные) языки – например, язык граф-схем алгоритмов (ГСА), так как они являются основой для построения параллельных языков. Для каждого начального языка рассматривается методика преобразования описания на начальном языке в описание на стандартном языке в виде системы канонических уравнений для всех событий, реализуемых в алгоритме управления.

Отличительной особенностью формализации управляющих алгоритмов на начальных языках заключается в том, что в качестве входного алфавита используется множество элементарных двоичных сигналов. При этом реализуемые в автомате события могут зависеть либо от всех, либо от части элементарных входных сигналов и эта зависимость событий от элементарных входных сигналов на разных шагах работы алгоритма управления может быть различной. Поэтому в качестве входных сигналов на каждом

шаге работы алгоритма управления используются частные входные входы, которые образуются сочетаниями только тех элементарных двоичных входных сигналов, которые действуют на данном переходе. Это обстоятельство, наряду с использованием для описания управляющих алгоритмов частных событий, а не состояний, позволяет значительно уменьшить критичность формального описания к его размерности, определяемой как произведение состояний системы и числа входных сигналов.

В пятой главе рассматриваются вопросы структурной реализации алгоритмов логического управления, заданных моделью НДА. При этом отмечаются недостатки классических методов структурного синтеза систем управления на основе использования моделей детерминированных конечных автоматов. Обращается особое внимание на структурную реализацию систем управления на основе разбиения частных событий, реализуемых в автомате, на группы несовместимых событий, что позволяет построить распределенные системы параллельной обработки. Рассматриваются также методы одноуровневой и двухуровневой структурной реализации систем управления для унитарного способа кодирования частных событий.

Шестая глава посвящена вопросам формализации алгоритмов управления взаимодействующими процессами для организации параллельной обработки информации. В главе вначале представлена методика формализации простейших базовых структур управления взаимодействующими процессами с использованием стандартного языка НД СКУ. На основе базовых структур управления показана методика формализации главной функции управления взаимодействующими процессами – взаимоисключение критических интервалов (участков), обеспечивающих доступ к общим разделяемым данным (общему ресурсу) для двух и n процессов. В последующих разделах главы рассмотрены методы формализации алгоритмов управления взаимодействующими параллельными процессами, связанные с решением задач синхронизации процессов и обменом сообщений. Методы формализации алгоритмов управления процессами иллюстрируются на примерах решения известных «классических» задач: обращение к общему ресурсу для n процессов, производители-потребители, читатели-писатели, обедающие философы. По каждой задаче представлены все основные уравнения, определяющие реализуемые в алгоритме

частные события, которые могут быть взяты за основу для практической структурной реализации системы управления.

В седьмой главе рассматриваются вопросы использования событийных недетерминированных автоматов (СНДА) для формального описания основных свойств систем управления взаимодействующими параллельными процессами и ресурсами, обеспечивающих надежность и эффективность таких систем с использованием различных механизмов синхронизации (в том числе согласующего кольцевого буфера, монитора, «рандеву»).

В восьмой главе рассматриваются вопросы структурной реализации алгоритмов управления процессами и ресурсами в многопроцессорных системах, которые основаны на формальном описании алгоритмов с использованием логики СНДА, и трансформации этих описаний в функциональные модели на языке VHDL. Представлены результаты экспериментов, и сделаны выводы, главным из которых является принципиальная возможность аппаратной реализации на современной элементной базе трудоемких функций ядра многопроцессорных операционных систем, таких как планирование и диспетчеризация задач, каналы межпроцессного обмена и управление ресурсами с использованием механизма критических интервалов. Другой вывод заключается в том, что значительно повышаются показатели производительности ядра многопроцессорных операционных систем, что особенно важно для систем реального времени.

В девятой главе рассматриваются вопросы верификации алгоритмов управления процессами и ресурсами в многопроцессорных системах, основанные на моделях событийных недетерминированных автоматов. Представлены способ верификации НД СКУ, основанный на прямой таблице переходов (ПТП), и программная система ПСВАУ, автоматизирующая создание и анализ ПТП. Приведен пример верификации моделей решения задачи взаимоисключающего доступа к общему ресурсу с использованием указанных средств. Также даны примеры верификации автоматных моделей алгоритмов планирования и диспетчеризации задач с использованием известных средств верификации и отладки типа систем SMV, Stateflow Matlab, CPN Tools.

В разделе «Заключение» приведены наиболее важные достоинства способов формализации алгоритмов логического управления на разных уровнях преобразования информации в вычислительных системах и сетях и их структурной реализации на основе

использования модели НДА и ее аналитического представления на языке НД СКУ в виде систем канонических уравнений, описывающих все реализуемые в алгоритме управления частные события.

При разработке монографии предполагалось, что читателю известны основные положения булевой алгебры, теории множеств, а также методы минимизации булевых функций и основные понятия конечных цифровых автоматов, способы их задания и синтеза с использованием систем канонических уравнений (СКУ). Читатели, которым неизвестны указанные сведения, могут пользоваться разделом «Приложение».

Монография разрабатывалась по материалам научных работ авторов в области теории НДА и ее использования для формального описания и структурной реализации алгоритмов управления процессами и ресурсами в параллельных системах обработки цифровой информации в вычислительных системах и сетях и систем управления промышленной автоматики.

Разработка монографии базировалась также на материалах работ авторов в области создания учебных дисциплин, связанных с изучением теории цифровых автоматов и их использования для разработки систем управления процессами и ресурсами вычислительных систем, учитывая при этом решение задач повышения производительности и надежности таких систем.

Данная книга базируется на учебном пособии «Недетерминированные автоматы в проектировании систем параллельной обработки», изданном в Пензенском государственном университете в 2004 г. В монографии представлены новые сведения в области формального описания и структурной реализации систем параллельной обработки информации с учетом обеспечения надежности и эффективности таких систем, полученные в ходе проведения исследований в течение последних десяти лет. Необходимо также отметить, что гл. 1–5 написаны Н. П. Вашкевичем, гл. 6–9 написаны Н. П. Вашкевичем совместно с Р. А. Бикташевым. Авторы выражают глубокую благодарность и признательность коллегам по работе, которые критически прочитали различные части рукописи и внесли полезные предложения. Особая благодарность преподавателям кафедры ВТ: Д. В. Пащенко, Н. Н. Коннову, С. А. Зинкину, В. Н. Дубинину, Е. И. Калиниченко, Е. И. Гурину.

Также выражаем благодарность магистрантам и аспирантам кафедры ВТ: А. И. Меркурьеву, В. В. Кутузову, Д. А. Левину, Е. А. Кизилову и С. В. Киселеву – за разработку макетов и прове-

дение многочисленных экспериментов, ставших важной частью гл. 8 и 9.

Очень признательны сотрудникам кафедры О. Н. Таньковой, А. В. Прошкину, а также студенту А. В. Калачеву за техническую подготовку рукописи.

Глава 1
ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ ИЗ ТЕОРИИ НЕДЕТЕРМИНИРОВАННЫХ АВТОМАТОВ

1.1. Определение и понятия недетерминированного автомата

Недетерминированные автоматы являются общим случаем конечных детерминированных цифровых автоматов, основные понятия и определения которых рассмотрены во многих источниках, в том числе в [1, 2] и в разделе «Приложение».

Под НДА понимается математическая модель устройства переработки цифровой (дискретной) информации, для которой не выполняются условия однозначности функций переходов. Кроме того, для НДА, как и для детерминированного конечного цифрового автомата, могут не соблюдаться также и условия полноты переходов. Термин «недетерминированный», как было отмечено в [3], наводит на мысль как о чем-то случайном, однако в НДА ничего случайного нет. В данном случае под НДА понимается просто формализм для определения (распознавания) множества цепочек (слов), а слово «автомат» в названии НДА присутствует, потому что формализм является обобщением формализма, используемого для определения обычного детерминированного конечного автомата. Таким образом, НДА не следует смешивать с вероятностным автоматом, в котором переходы осуществляются в любое из состояний с некоторой вероятностью.

Понятие НДА было впервые введено в работах американских ученых М. О. Рабина и Д. Скотта [4]. В дальнейшем понятие НДА успешно использовалось в работах, связанных с решением теоретических вопросов информатики, например, в [5, 6, 7]. Однако в этих работах НДА рассматривались в основном с точки зрения использования такой математической модели для построения различных распознавателей и компиляторов. В частности, в работах [5, 8] НДА были представлены под названием «источники». При этом под источником понимается ориентированный граф, однозначно определяющий событие S в некотором алфавите $[Z]$, порождаемое множеством всех путей из начальных вершин в заключительные.

В работах авторов [9] и [10] НДА был представлен системой предикатных формул на языке исчисления предикатов 1-го порядка, а также системой бескванторных предикатных формул, формализующих все реализуемые частные события алгоритма функционирования цифрового автомата, полученных после спуска кванторов.

В дальнейших работах авторов [2, 11, 51, 54, 77–79] модель НДА, представленная в виде стандартной и универсальной системы рекуррентных бескванторных предикатных формул, названная системой канонических уравнений недетерминированного автомата (НД СКУ), стала основой для синтеза систем логического управления процессами и объектами.

Использование понятий НДА как математической модели в виде НД СКУ, как будет показано ниже, является эффективным средством для формализации описания сложных алгоритмических процессов управления, в том числе алгоритмов управления с параллельными взаимодействующими ветвями. Используя различные эквивалентные преобразования такого описания, можно построить различные структуры устройств управления для систем параллельной обработки, позволяющие увеличить их производительность и уменьшить аппаратурные затраты.

Наиболее простое определение НДА основывается на языке графов. НДА называется произвольная диаграмма B над входным алфавитом, в которой выделены множества начальных и финальных вершин. Под диаграммой B понимается направленный граф, вершины которого помечены буквами из алфавита $B = [b_0, b_1,..., b_m]$ (или цифрами). Каждая вершина графа отмечена соответствующими выходными сигналами из абстрактного выходного алфавита $W = [w_0, w_1, ..., w_y]$ или совокупностью выходных двоичных элементарных сигналов из структурного алфавита $Y = [y_1, y_2,..., y_N]$. Ребра графа помечены буквами из абстрактного входного алфавита $Z = [z_0, z_1,..., z_F]$ или частными входными сигналами, являющимися сочетаниями двоичных элементарных сигналов из структурного алфавита $X = [x_1, x_2,..., x_L]$.

Понятие «произвольный» здесь означает, что граф может не удовлетворять условиям автоматности [5]. В частности, допускаются следующие особенности:

а) из некоторой вершины может выходить несколько ребер, помеченных одной и той же буквой входного алфавита [Z] или

одним и тем же частным входным сигналом из алфавита [X] (нарушение однозначности);

б) из ряда вершин может вообще не выходить ни одно ребро, помеченное некоторой буквой входного алфавита (нарушение полной определенности);

в) имеется несколько начальных вершин.

Автоматная интерпретация НДА предполагает следующее его функционирование. Если в некоторый момент времени (t) возбуждена вершина с номером $i(b_i)$ и имеет место входной сигнал z_k, то в момент времени ($t + 1$) будут возбуждены все вершины, в которые ведут ребра с пометкой z_k. Если таких ребер нет, то не будет возбуждена ни одна вершина.

Так как в каждый момент времени (t) может быть возбуждено несколько вершин, то для определения вершин, которые будут возбуждены в момент времени ($t + 1$), необходимо объединить множество вершин, возбуждаемых из каждой возбужденной вершины. В момент времени $t = 0$ возбуждены все начальные вершины.

В каждом такте будут выдаваться выходные сигналы, которыми были отмечены одновременно возбужденные вершины.

Таким образом, недетерминированный автомат Мура – это пятерка:

$$M = [S, X_{i,j}, \delta, S_0, Y_j],$$

где $[S] = [S_1, S_2, ..., S_m]$ – множество частных событий; $[X_{i,j}]$ – множество частных входных сигналов, образуемых сочетаниями элементарных двоичных входных сигналов из алфавита $[X]$; $[Y_j]$ – множество выходных сигналов, отмечающих частные события S_j и образуемых совокупностью двоичных элементарных выходных сигналов из алфавита $[Y]$; S_0 – начальное событие (возможно существование нескольких начальных событий); δ – функция переходов НДА, которую в обобщенном виде можно представить так:

$$\delta[(S_{i,1}, ..., S_{i,k}), X_{i,j}](t) = [(S_{j,1}, ..., S_{j,l}),(Y_{j,1}, ...,Y_{j,l})](t + 1),$$

откуда видно, что функция переходов δ определяет переходы от i-го множества частных событий ($S_{i,1}$, ..., $S_{i,k}$) к j-му множеству частных событий ($S_{j,1}$, ..., $S_{j,l}$), одновременное существование которых в автомате возможно, под действием частного входного сигнала $X_{i,j}$. Необходимо иметь в виду, что в некоторых случаях i-е и/или j-е множества частных событий для отдельных переходов могут содержать в себе только один элемент.

Отметим, что ***недетерминированность модели автомата Мура*** будет проявляться лишь в возможном переходе под действием входного сигнала от какого-либо частного события S_i (или событий) ***к нескольким частным событиям*** (или событию S_j).

В то время как в ***детерминированном автомате*** (если под событием понимать переход автомата в определенное состояние a_j) всегда реализуются переходы под действием сигнала $X_{i,j}$ от события S_i только к ***одному*** событию S_j.

По НДА может быть построен эквивалентный ему конечный детерминированный автомат с помощью алгоритма детерминизации. Один из вариантов алгоритма детерминизации был предложен в работе [12], где он представлен в виде алгоритма синтеза таблицы переходов детерминированного цифрового автомата Мура, заданного моделью НДА в виде системы рекуррентных бескванторных предикатных формул, описывающих все реализуемые в автомате частные события. В дальнейшей работе [2] алгоритм детерминизации усовершенствовался применительно к синтезу микропрограммных управляющих автоматов. Алгоритмы детерминизации НДА были также рассмотрены, например, в работах [5, 6, 8].

В процессе детерминизации каждому возможному множеству одновременно возбужденных вершин (событий) ставится в соответствие вполне определенное состояние автомата и определяются переходы между этими состояниями. НДА является более компактным представлением автомата, поскольку при детерминизации НДА с m реализуемыми в нем событиями (состояниями) эквивалентный ему детерминированный автомат может иметь не более $M \le 2^m$ состояний, что будет доказано в дальнейшем. С другой стороны, можно утверждать, что детерминированный автомат с M состояниями может быть представлен НДА с $[\log 2\ M]$ состояниями. Получение такого НДА возможно, например, путем кодирования детерминированного автомата состояниями НДА. Отсюда следует, что операция детерминизации над НДА и операция кодирования эквивалентного ему детерминированного автомата являются операциями, обратными друг другу.

В том случае, когда в графе автомата выделено единственное начальное состояние (событие) и при любых входных последовательностях сигналов никогда не будут одновременно возбуждены две и более вершин (или одновременно существовать два или более событий), такой автомат будет являться детерминированным. В этом случае каждой вершине графа или событию, реализуемому

в автомате, будет соответствовать свое вполне определенное состояние автомата.

Над НДА возможно выполнение большого количества операций, из которых, кроме отмеченных выше операций детерминизации и кодирования, можно выделить: операцию стягивания начальных и финальных вершин, которые отмечены одним и тем же выходным сигналом; операции минимизации, отрицания, конкатенации, суммирования, объединения и др. Указанные операции над НДА рассматриваются как операции над СКУ, описывающими поведение НДА. Они используются для преобразования СКУ с целью построения структуры автомата, удовлетворяющей определенным требованиям. Это оказалось возможным благодаря использованию модели НДА в виде СКУ, представляющей собой простую стандартную и универсальную систему рекуррентных бескванторных предикатных формул, правая часть которых представляет собой ДНФ булевых функций.

1.2. Общие сведения о выборе языков для представления недетерминированных автоматов

НДА так же, как и конечные цифровые автоматы, используются в качестве математической модели для формального описания алгоритмов логического управления в системах обработки дискретной информации. При этом наиболее эффективно они могут быть использованы в системах, реализующих параллельную обработку информации. Для формального описания таких математических моделей используются языки двух типов: начальные и стандартные, которые отличаются друг от друга способом задания функций переходов.

Начальные языки характеризуются тем, что функции переходов автоматов в описании их закона функционирования на этих языках представлены не в явном виде. К числу таких языков относятся, например, такие известные, как язык регулярных выражений алгебры событий (РВАС), язык логики одноместных предикатов, языки операторных схем алгоритмов (ОСА), языки операторных схем алгоритмов с параллельными ветвями (ОСАП) и др.

Стандартные языки, в отличие от начальных, характеризуются тем, что в описании закона функционирования автоматов

на этих языках функции переходов представлены в явном виде. К числу таких языков относятся: различные типы таблиц переходов и выходов, матрицы переходов, направленные графы и системы канонических уравнений.

Стандартный язык, представляя функции переходов автомата в явном виде, позволяет непосредственно перейти к структурной реализации управляющего устройства (автомата). С другой стороны, на стандартном языке в большинстве случаев затруднительно выполнять «перевод» неформального словесного описания алгоритма функционирования автомата на формальное описание, т.е. говорят, что стандартный язык обладает небольшими выразительными возможностями.

В связи с этим для первоначального описания алгоритма логического управления по его словесному неформальному описанию, представленному в техническом задании, целесообразно использовать начальные языки, к которым предъявляются следующие основные требования [8, 13, 14]:

– язык должен обладать достаточно широкими выразительными возможностями и быть удобным не только для перевода на этот язык часто встречающихся словесных описаний алгоритма управления, но и должен быть достаточно удобным переход от начального формального описания к каноническим уравнениям стандартного языка. Кроме того, язык должен также содержать необходимые средства и методы для эквивалентных преобразований и выполнения контрольных процедур описания на начальном языке;

– описание алгоритма управления, полученное на основе использования начального языка и представленное в виде системы уравнений, формализующих все реализуемые в алгоритме события, должно быть выражено в виде регулярных формул. Это требование следует из известной теоремы С. К. Клини [15], в которой утверждается, что ***для того, чтобы события были представлены в конечном автомате***, ***необходимо и достаточно***, ***чтобы они были регулярными***, т.е. описание таких событий должно быть получено из описания элементарных событий с использованием только трех операций алгебры событий: дизъюнкции, конкатенации и итерации. Таким образом, когда задание на реализацию алгоритма управления будет выражено на языке регулярных формул, то говорят, что проблема распознавания и синтеза управляющего устройства в соответствии с заданным алгоритмом не возникает [13].

Выбор того или иного начального языка для формального описания алгоритмов управления зависит, в первую очередь, от двух основных факторов: от сферы применения (области применения) языка и от традиций и привычек проектировщика в описании условий работы автомата [8, 14]. Например, для решения задач, связанных с распознаванием и преобразованием языков, целесообразно использовать язык РВАС, а различные варианты языков операторных схем алгоритмов (ОСА) наиболее целесообразно использовать при решении вопросов проектирования систем логического управления объектами и процессами.

В нашей работе в качестве базового языка для формального представления управляющих алгоритмов (по аналогии с выбором базового языка в [14]) предлагается использовать язык граф-схем алгоритмов с параллельными ветвями (ГСАП) совместно с языком НД СКУ, являющимся его аналитической интерпретацией [16–21].

Такой выбор языков для формализации алгоритмов управления позволяет наглядно представить общую структуру модели алгоритма управления в виде графа НДА, в котором предусмотрены, кроме четырех типов вершин, имеющихся в обычном языке ГСА, разветвительные и соединительные вершины, обеспечивающие разветвление алгоритмического процесса и соединение параллельных ветвей. При этом, изменяя логические условия выхода алгоритмического процесса за соединительную вершину, можно решать многие вопросы синхронизации и взаимодействия параллельных ветвей.

Язык НД СКУ, являясь аналитической интерпретацией графического представления общей структуры алгоритма управления, позволяет детально представить аналитическое описание всех реализуемых в алгоритме частных событий в виде стандартной системы канонических бескванторных рекуррентных уравнений, в том числе частных событий, которые обеспечивают организацию взаимодействия параллельных ветвей алгоритма управления и их синхронизацию.

В связи с тем, что правая часть уравнений канонической системы НД СКУ представляет ДНФ булевых функций двух типов переменных, то над системой уравнений НД СКУ для целей ее структурной реализации (аппаратно, программно или аппаратно-программно) могут быть выполнены все необходимые эквивалентные преобразования, в том числе и преобразования в соответствии с законами булевой алгебры.

Кроме того, учитывая, что управляющий алгоритм аналитически представлен системой канонических уравнений НД СКУ, формализующих все частные события НДА (а не состояния детерминированного автомата), вопросы структурной реализации систем логического управления менее критичны к размерности решаемых задач, так как представление алгоритма управления в виде модели НДА является самым минимальным по сложности по сравнению с представлением алгоритма управления на основе использования модели детерминированного автомата.

Следует также отметить, что выбор указанных языков представления алгоритма управления предопределил также широкое использование для структурного синтеза систем управления прямых таблиц переходов для частных событий (НД ПТП), реализуемых в алгоритме управления (в виде списочной структуры). Использование НД ПТП позволяет выполнять при необходимости преобразование описания алгоритма управления с одного языка на другой, избавляться от недостижимых событий, проверку алгоритма управления на корректность и многие другие эквивалентные преобразования алгоритма управления, связанные с минимизацией, детерминизацией, кодированием и другими операциями над событиями.

В связи с тем, что язык НД СКУ является базовым, на который осуществляется перевод алгоритмов управления со всех начальных языков, то его рассмотрению посвящены два следующих раздела.

1.3. Аналитическое представление недетерминированных автоматов на стандартном языке

Если с каждой вершиной направленного графа НДА связать предикатную переменную $S_j(t)$, с каждым входным сигналом – предикатную переменную $X_{i,j}(t)$, а с каждым выходным сигналом – предикатную переменную $Y_j(t)$, где t пробегает ряд целых неотрицательных чисел (t = 0, 1, 2,...), то функционирование НДА может быть задано системой рекуррентных бесквантворных предикатных формул вида (1.1), описывающих все реализуемые в автомате частные события S_j:

$$S_j^{Yj}(t+1) = \bigvee_{i,j} X_{i,j}(t) \,\&\, S_i(t) \vee \bigvee_{j} X_{j,j}(t) \,\&\, S_j(t), \qquad (1.1)$$

$$S_0(0) = 1,\ j = \overline{0,m},$$

где $(m+1)$ – число вершин графа (число всех событий, реализуемых НДА); $S_i(t)$ – сокращенное обозначение выражения, описывающего событие S_i, непосредственно предшествующее событию S_j, при этом каждая предикатная формула $S_i(t)$ должна быть либо тождественно истинной предикатной формулой, либо удовлетворять условию $S_i(t) \neq S_j(t)$ и определяться также выражением вида (1.1); $X_{i,j}$ – частный входной сигнал, представляющий собой сочетание элементарных входных сигналов из алфавита $[X]$ (взятых с отрицанием или без отрицания), в результате воздействия которых осуществляется переход от вершины графа НДА с номером i к вершине графа с номером j (в новых обозначениях переход от события S_i к событию S_j); Y_j – частный выходной сигнал, представляющий собой сочетание элементарных выходных сигналов из алфавита $[Y]$, отмечающих вершину графа b_j (или событие S_j).

Систему уравнений вида (1.1) будем называть в дальнейшем ***простой канонической формой*** или сокращенно – системой канонических уравнений (СКУ). При этом если система уравнений описывает НДА, то сокращенное обозначение ее будет – НД СКУ. Кроме того, так как способы представления цифровых автоматов принято называть также и языками представления автоматов, то и в нашей работе, когда это целесообразно, будем называть систему уравнений вида (1.1) сокращенно – язык СКУ.

Как видно из (1.1), СКУ определяет все переходы в НДА и является отмеченной НД СКУ автомата Мура, так как каждое событие S_j отмечено своим частным выходным сигналом Y_j. При этом первая часть уравнений (1.1) позволяет формализовать описание условий первоначального появления или зарождения события S_j, а вторая – определить условие возобновления или сохранения его. Следует иметь в виду, что описание некоторых событий S_j системы (1.1) может и не содержать вторую часть уравнений, определяющую сохранение события S_j, в то время как присутствие первой части в каждом уравнении системы (1.1) обязательно.

Из системы уравнений (1.1) также следует, что введенное понятие «событие» принципиально ничем не отличается от понятия события, принятого в теории конечных цифровых автоматов [1].

Действительно, следуя [1], в нашем рассмотрении событие, например, S_j – это также множество слов, каждое из которых состоит из последней буквы ($X_{i,j}$) этого слова и начального отрезка слова до этой буквы, обозначенного сокращенно S_i, и определяемое как событие, непосредственно предшествующее событию S_j.

Представление функционирования НДА системой уравнений (1.1) соответствует описанию, которое является результатом преобразования регулярных выражений алгебры событий (РВАС) произвольной формы к виду, представленному в развернутой форме в виде системы канонических уравнений. Действительно, используя правило развертывания итерации и закона коммутативности для итерации, в алгебре событий [1] выражение для события вида

$$S_\alpha = S_\beta \{R\}, \alpha \neq \beta \qquad (1.2)$$

преобразуется в выражение с отсутствием итерационных скобок

$$S_\alpha = S_\beta \vee S_\alpha R, \qquad (1.3)$$

откуда следует, что выражения (1.2) и (1.3) являются различными формами определения одного и того же события S_α. При этом выражение (1.2) называют свернутой формой, а выражение (1.3) – развернутой формой определенного регулярного выражения алгебры событий.

В связи с этим каждую предикатную формулу $S_j(t + 1)$ системы (1.1) можно привести к следующей свернутой форме регулярного выражения, имеющего вид:

$$S_j(t) = \bigvee_{i,j} S_i(t-1) \,\&\, X_{i,j}(t) \vee \bigvee_{i,j} S_j(t-1) \,\&\, X_{j,j}(t) =$$

$$= \bigvee_{i,j} S_i X_{i,j} \vee \bigvee_{i,j} S_j X_{j,j} = \bigvee_{i,j} S_i X_{i,j} \{X_{j,j}\}. \qquad (1.4)$$

Выполняя все необходимые операции свертывания в уравнениях системы (1.1), которые содержат вторую часть описания, определяющую условия сохранения события, и выполняя m раз операции подстановки свернутых выражений вместо непосредственно предшествующих событий S_i в выражения типа (1.4), получим представление исходной системы (1.1) в виде выражений алгебры событий, выраженных через частные входные сигналы и начальное событие S_0 с использованием только трех операций алгебры событий: дизъюнкции, конкатенации и итерации.

Таким образом, учитывая изложенное, можно утверждать, что система уравнений (1.1), представляющая простую каноническую форму, описывающая все реализуемые в автомате частные события, на основании теоремы С. К. Клини представима в конечном цифровом автомате, а следовательно, и в НДА.

П р и м е ч а н и е. В гл. 4, посвященной начальным языкам, будет подробно рассмотрен алгоритм преобразования регулярных выражений алгебры событий в описание событий в виде системы канонических уравнений.

Для построения управляющего автомата в виде модели автомата Мура необходимо, кроме функций переходов, представленных уравнениями (1.1), получить и систему выходных функций (СВФ). Она может быть получена из (1.1) путем объединения тех событий S_j, в состав отмеченных частных выходных сигналов которых входит рассматриваемый элементарный выходной сигнал, т.е. СВФ для автомата Мура будет иметь вид

$$y_k(t+1) = \bigvee_{\forall S_j\left(Y_j \subset y_k\right)} S_j^{Y_j}(t+1),\ k = \overline{1,N} \tag{1.5}$$

или

$$y_k(t) = \bigvee_{\forall S_j\left(Y_j \subset y_k\right)} S_j^{Y_j}(t),\ k = \overline{1,N}, \tag{1.6}$$

где переменные в левой и правой частях уравнений взяты для одного и того же момента времени.

Система выходных функций для модели автомата Мили эквивалентного ему автомата Мура может быть получена из отмеченной НД СКУ вида (1.1) путем объединения выражений для тех событий S_j, в состав отмеченных частных выходных сигналов которых входит рассматриваемый элементарный выходной сигнал, т.е. для автомата Мили СВФ будет иметь вид

$$y_k(t) = \bigvee_{\forall S_j\left(Y_j \subset y_k\right)} S_j^{Y_j}(t+1),\ k = \overline{1,N}, \tag{1.7}$$

где в правую часть выражения (1.7) необходимо сделать подстановку из (1.1).

Сравнивая выражения (1.5) и (1.7), подтверждаем тот факт, что выходной сигнал автомата Мура отстает на один такт по срав-

нению с выходным сигналом эквивалентного ему автомата Мили. Кроме того, следует иметь в виду, что выходные сигналы автомата Мура отличаются от выходных сигналов эквивалентного ему автомата Мили по времени действия, а именно: выходной сигнал автомата Мура действует от момента появления события и до его исчезновения, а выходной сигнал автомата Мили действует от момента появления входного сигнала и до его исчезновения.

Система функций переходов для автомата Мили эквивалентного ему автомата Мура может быть получена из системы уравнений (1.1) путем объединения тех событий НД СКУ автомата Мура, переходы из которых полностью совпадают.

П р и м е р 1.1. Пусть НДА Мура задан графом (рис. 1.1), в котором в качестве входного и выходного алфавитов используются структурные алфавиты:

$$X = [x_0, x_1, x_2, x_3, x_4],\ Y = [y_0, y_1, y_2, y_3, y_4],$$

где x_0 – сигнал приведения автомата в исходное состояние.

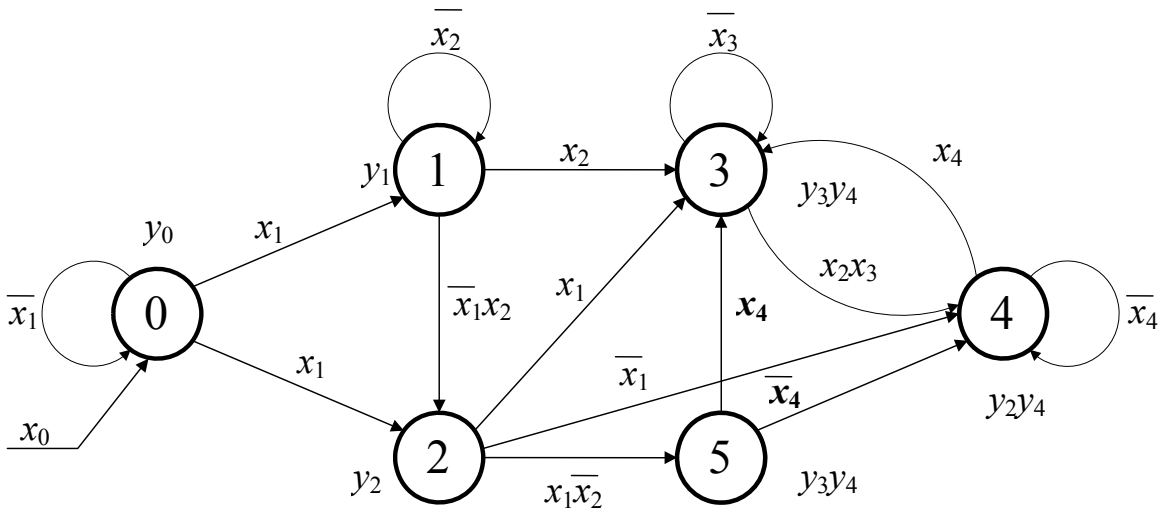


Рис. 1.1. Граф недетерминированного автомата Мура

Построить для этого графа НД СКУ и СВФ для моделей автомата Мура и Мили.

Как видно из рис. 1.1, представленный графом НДА Мура является также и не полностью определенным, так как из вершины 3 не удовлетворяются условия полноты переходов.

Поставив в соответствие каждой вершине графа событие перехода в эту вершину, получим следующую отмеченную недетерминированную СКУ не полностью определенного НДА Мура:

$$
\begin{aligned}
&S_0^{y_0}(t+1) = S_0(t)\,\&\,\overline{x}_1(t) \vee x_0(t);\\
&S_1^{y_1}(t+1) = S_0(t)\,\&\,x_1(t) \vee S_1(t)\overline{x}_2(t);\\
&S_2^{y_2}(t+1) = S_0(t)\,\&\,x_1(t) \vee S_1(t)\,\&\,\overline{x}_1(t)x_2(t);\\
&S_3^{y_3y_4}(t+1) = S_1(t)\,\&\,x_2(t) \vee S_2(t)\,\&\,x_1(t) \vee S_3(t)\,\&\,\overline{x_3}(t) \vee\\
&\vee(S_4 \vee S_5)(t)\,\&\,x_4(t);\\
&S_4^{y_2y_4}(t+1) = S_2(t)\,\&\,\overline{x}_1(t) \vee S_3(t)\,\&\,x_2(t)\,\&\,x_3(t) \vee\\
&\vee(S_4 \vee S_5)(t)\,\&\,\overline{x}_4(t);\\
&S_5^{y_1y_3}(t+1) = S_2(t)\,\&\,x_1(t)\,\&\,\overline{x}_2(t).
\end{aligned}
\qquad (1.8)
$$

Примечание. В дальнейшем, если позволяет контекст, знаки & и время t в правой части СКУ для простоты будем опускать.

По НД СКУ автомата Мура в соответствии с (1.6) получим СВФ не полностью определенного НДА Мура:

$$
\begin{aligned}
&y_0 = S_0 \qquad y_1 = S_1 \vee S_5 \qquad y_2 = S_2 \vee S_4\\
&y_3 = S_3 \vee S_5 \qquad y_4 = S_3 \vee S_4
\end{aligned}
\qquad (1.9)
$$

НД СКУ автомата Мили, эквивалентного автомату Мура, получим из (1.8) путем объединения тех событий НД СКУ автомата Мура, переходы из которых полностью совпадают. К таким событиям относятся события S_4 и S_5, переходы из которых совпадают: $(S_4 \vee S_5)x_4 \to S_3$, $(S_4 \vee S_5)\overline{x}_4 \to S_4$. При этом для всех букв левой части уравнений для НД СКУ автомата Мили опускаются отме-чающие их выходные сигналы. Делая замену переменных $S_4 \vee S_5 = S_4$ и их подстановку в (1.8), получим НД СКУ не полностью определенного автомата Мили:

$$
\begin{aligned}
&S_0(t+1) = S_0\overline{x}_1 \vee x_0;\\
&S_1(t+1) = S_0x_1 \vee S_1\overline{x}_2;\\
&S_2(t+1) = S_0x_1 \vee S_1\overline{x}_1x_2;\\
&S_3(t+1) = S_1x_2 \vee S_2x_1 \vee S_3\overline{x}_3 \vee S_4x_4;\\
&S_4(t+1) = S_2(\overline{x}_1 \vee \overline{x}_2)S_3x_2x_3 \vee S_4\overline{x}_4.
\end{aligned}
\qquad (1.10)
$$

По НД СКУ автомата Мура, в соответствии с (1.7) и учитывая замену $S_4 \vee S_5 = S_4$, получим СВФ не полностью определенного НДА Мили:

$$
\begin{aligned}
&y_0 = S_0\overline{x}_1 \vee x_n;\\
&y_1 = S_0x_1 \vee S_1\overline{x}_2 \vee S_2\overline{x}_2;\\
&y_2 = S_0x_1 \vee S_1\overline{x}_1x_2 \vee S_2\overline{x}_1 \vee S_3x_2x_3 \vee S_4\overline{x}_4; \qquad (1.11)\\
&y_3 = S_1x_2 \vee S_2x_1 \vee S_3\overline{x}_3 \vee S_4x_4;\\
&y_4 = S_1x_2 \vee S_2 \vee S_4 \vee S_3\left(x_2 \vee \overline{x}_3\right).
\end{aligned}
$$

1.4. Расширение выразительных возможностей языка недетерминированных систем канонических уравнений и его отличительные особенности

Следует отметить, что в ряде случаев исходный алгоритм функционирования автомата целесообразно сразу записать в форме НД СКУ, так сказать, использовать СКУ как начальный язык. В этом случае НД СКУ, представленную в простой канонической форме уравнениями вида (1.1), целесообразно представить в более обобщенной форме с целью увеличения «выразительных» возможностей языка СКУ. В частности, такую НД СКУ можно представить в виде системы рекуррентных предикатных формул в следующем виде:

$$
\begin{aligned}
&S_j^{Y_j}(t+1) = \bigvee_{i,j} X_{i,j}(t)\,\&\,R_i(t)\,\&\,S_i(t) \vee \bigvee_{j} X_{j,j}(t)\,\&\,R_j(t)\,\&\,S_j(t), \qquad (1.12)\\
&S_0(0) = 1,\ j = \overline{0,m},
\end{aligned}
$$

где $R_i = \tilde{S}_{i,1}\,\&\cdots\&\,\tilde{S}_{i,k}$, $R_j = \tilde{S}_{j,1}\,\&\cdots\&\,\tilde{S}_{j,r}$; «~» означает, что переменные в R_i и R_j могут быть взяты с отрицанием или без него.

Из структуры уравнений системы (1.12) следует, что расширение выразительных возможностей для формального описания событий достигается за счет описания более сложных условий зарождения и сохранения событий S_j, которые могут зависеть не только от значений внешних частных входных сигналов $X_{i,j}$, но и от значений совокупности некоторых частных событий или внутренних переменных, непосредственно предшествующих событию S_j. К числу таких частных событий или внутренних переменных, образующих комбинационные события R_i и R_j, могут

относиться, например, такие, которые определяют условия готовности исходных данных для их обработки в операционных блоках и их готовности к приему результатов обработки исходных данных, а также переменные, отражающие временной фактор, определяющий начало или конец выполнения отдельных операций (микроопераций) и др.

При формализации частных событий системы (1.12) необходимо иметь в виду, что описание каждого частного события S_j должно в общем случае состоять из трех частей [21]:

– описание события в момент его наступления;

– описание условий, при которых событие сохраняется;

– описание события, непосредственно предшествующего событию S_j.

Отметим также, что описания некоторых событий могут и не иметь описаний условий его сохранения, тогда как две другие части описания события S_j должны быть обязательно. В противном случае описание события S_j может преобразоваться в булеву функцию, реализуемую простой комбинационной схемой.

Полученная система уравнений (1.12) должна быть проверена на отсутствие недостижимых событий. Для этой цели строится прямая таблица переходов недетерминированного автомата Мура (НД ПТП) для всех частных событий, представленных уравнениями (1.12) и реализуемых в автомате.

НД ПТП Мура в общем случае может иметь пять основных колонок (рис. 1.2).

Шаг алгоритма	$S_i(t)$	$R_i(t)$	$X_{i,j}(t)$	$S_j(t+1)$	$Y_j(t+1)$
1	2	3	4	5	6
1	S_0				

Рис. 1.2. Прямая таблица переходов для событий НДА Мура (НД ПТП)

Построение НД ПТП должно выполняться по следующему правилу: построение первой строки таблицы начинается с исходного события S_i, которое соответствует событию S_0, а для всех последующих строк в качестве исходного события S_i использует-

ся то, которым уже была отмечена колонка (5) в предыдущих строках таблицы. Такой порядок построения НД ПТП выявляет недостижимые события, т.е. те, которые не имеют событий предшественников.

Построенная НД ПТП может быть в дальнейшем использована для преобразования системы уравнений (1.12), связанных с выполнением операций минимизации, детерминизации, кодирования и др.

Отличительные особенности языка НД СКУ:

а) это язык, определяющий в явном виде функции переходов в автомате. Эти функции выражаются в виде переходов от события к событию (или событиям) или переходов от событий к событию (или событиям), причем эти переходы для каждого входного сигнала являются ***однозначными***;

б) недетерминированная СКУ представляется системой **«рекуррентных»** формул. Это означает, что каждое событие S_j из множества реализуемых в автомате событий, может быть определено через событие S_i (события), имеющее место в предшествующий момент автоматного времени. При этом в начале работы автомата может быть истинным то событие (события), для которого в предшествующий момент времени было истинным событие S_0;

в) в названии системы НД СКУ присутствует слово **«каноническая»**. Этим названием подчеркивается тот факт, что все переменные в правой части уравнений НД СКУ взяты для одного и того же момента времени (t);

г) учитывая, что правая часть формул НД СКУ представляет собой ДНФ булевых функций от двух типов переменных, представляющих как входные сигналы, так и события, то к этим формулам могут быть применены все законы и методы преобразования булевых функций;

д) учитывая также, что для систем (1.1) и (1.12) можно было бы вместо знака равенства поставить знак $\eqslantless$, являющийся сокращенным обозначением, означающим, что предикат, стоящий слева от этого знака, является сокращенным обозначением предикатной формулы, расположенной от него справа [15, с. 222, 226], возможно применять к формулам НД СКУ все законы и методы преобразования, принятые в исчислении высказывания и предикатов;

е) исходя из изложенного, целесообразно математическую модель НДА, представленную на языке НД СКУ, называть в даль-

нейшем ***автоматной моделью НДА***, хотя во многих работах по теории цифровых автоматов слово **«автоматность»** всегда отождествлялось с понятием детерминированного автомата. При введении такого названия автор исходил из того, что в названии математической модели автомата словом **«автоматная»** в первую очередь подчеркивается ***задание функций переходов в автомате в явном виде***;

ж) в связи с тем, что НДА является обобщением конечного детерминированного автомата, математическую модель детерминированного автомата (ДА), представленную на языке СКУ, будем называть ***автоматной моделью ДА***. Для такого автомата СКУ будет представлять собой функции переходов в автомате, когда переход в любое состояние интерпретируется как событие.

В заключение отметим, что в ряде работ автора приведены алгоритмы синтеза автоматных моделей управляющих автоматов, заданных на некоторых начальных языках: операторных схем алгоритмов (ОСА), в том числе с параллельными ветвями (ОСАП) [2, 16–19], на языке регулярных выражений алгебры событий [2, 20], на языке исчисления предикатов 1-го порядка [9].

1.5. Иерархия входных сигналов и событий, реализуемых в системах логического управления

В данном разделе уточняются и систематизируются введенные ранее некоторые понятия и определения и вводится ряд новых понятий и определений, связанных с представлениями входных сигналов и событий, реализуемых в системах управления. Эти понятия и определения будут в дальнейшем использоваться в последующих разделах монографии при рассмотрении вопросов, связанных с эквивалентными преобразованиями НДА, формализацией алгоритмов управления и структурного синтеза систем управления.

Частный входной сигнал – сигнал, определяемый сочетанием (конъюнкцией) тех элементарных двоичных входных сигналов из алфавита [X], взятых с отрицанием или без него, которые действуют ***в одной из любых ветвей алгоритма управления на данном его шаге*** и которые определяют вполне определенное (част-

ное) событие. Будем обозначать эти сигналы буквой $X_{i,j}$, означающей, что под действием этого сигнала происходит переход от события S_i к событию S_j в пределах одного шага алгоритма управления. Под шагом алгоритма понимается путь от одной операторной вершины к другой, проходящей только через логические условия.

Полный входной сигнал в алфавите [X] – сигнал, определяемый сочетанием (конъюнкцией) тех элементарных двоичных входных сигналов, взятых с отрицанием или без него, действующих ***во всех ветвях на данном шаге алгоритма управления***, которые определяют вполне определенное состояние автомата (полное событие). Будем обозначать этот сигнал символом $X(a_m, a_s)$, означающим, что под действием этого сигнала происходит переход ДА из состояния a_m в состояние a_s в пределах одного шага алгоритма управления.

Полный входной сигнал в алфавите [Z]. Этому сигналу будет соответствовать полный входной сигнал в алфавите [X], определяемый сочетанием всех L элементарных двоичных входных сигналов из алфавита [X], взятых с отрицанием или без него, действующих ***во всех ветвях и на всех шагах алгоритма управления***. Максимально возможное число таких сигналов равно 2^L. Будем обозначать такие сигналы символом $Z(a_m, a_s)$, означающим, что под действием этого сигнала происходит переход ДА из состояния a_m в состояние a_s в пределах одного шага алгоритма управления.

Пустой входной сигнал – специально введенный абстрактный входной сигнал, называемый пустой буквой e (пустое слово нулевой длины), позволяющий считать, что абстрактный автомат (математическая модель автомата) получает входные сигналы во все множества времени $t = 0, 1, 2, ...$, несмотря на то, что реальный автомат в какие-то моменты времени может и не получать никаких реальных входных сигналов [1]. Таким образом, появление буквы e на входе автомата свидетельствует о том, что на вход автомата в действительности ничего не подается. Это нашло отражение в законе нейтральности пустого слова: $eS = Se = S$.

Запрещенный входной сигнал – сигнал, который на вход автомата никогда не подается. Выражение для такого сигнала может быть определено как отрицание от дизъюнкции всех входных сигналов $X_{i,j}$, действующих на переходе. Для запрещенного входного сигнала событие перехода будет неопределенным.

Выражение для запрещенных входных сигналов может быть определено как отрицание от дизъюнкции всех входных сигналов, действующих на переходах алгоритма управления.

В зависимости от типа математической модели автомата можно выделить три вида запрещенных входных сигналов:

– ***частный запрещенный входной сигнал*** в одной из ветвей алгоритма управления на определенном его шаге в алфавите [*X*]. Например, пусть задан фрагмент ПТП автомата, для которого не выполняются условия полноты переходов.

$S_i(t)$	$x_2(t)$ $x_1\overline{x}_2(t)$	$S_j^1(t+1)$ $S_j^2(t+1)$

Для этого фрагмента ПТП $\overline{x}_{i,j}^1 \vee x_{i,j}^2 \neq 1$. Полнота переходов может быть обеспечена вводом запрещенного входного сигнала, для которого событие перехода будет неопределенным.

$$x_{\text{запр.}} = \overline{x_1 \vee x_1\overline{x}_2} = \overline{x}_1 \ \overline{x}_2$$

Скорректированный фрагмент ПТП будет иметь вид

$S_i(t)$	$x_2(t)$ $x_1\overline{x}_2(t)$ $(\overline{x}_1\overline{x}_2)^*(t)$	$S_j^1(t+1)$ $S_j^2(t+1)$ $S(-)$

Здесь знаком * будем обозначать запрещенный входной сигнал, а знаком $S(-)$ – неопределенное событие;

– ***полный запрещенный входной сигнал*** для всех ветвей алгоритма управления на определенном его шаге в алфавите [*X*];

– ***полный абстрактный запрещенный входной сигнал*** для всех ветвей алгоритма управления на всех его шагах в алфавите [*Z*].

Покажем на примере взаимосвязь между отдельными видами запрещенных входных сигналов и пустым входным сигналом.

Пусть алфавит абстрактных входных сигналов содержит следующие пять сигналов $[Z] = [z_0, z_1, z_2, z_3, z_4]$. Тогда при построении структурной схемы автомата абстрактные входные сигналы должны кодироваться тремя двоичными входными сигналами из алфавита $[X] = [x_1, x_2, x_3]$.

Для произвольного кодирования получаем

$$z_0 = \overline{x}_1\ \overline{x}_2\ \overline{x}_3,$$
$$z_1 = \overline{x}_1\ \overline{x}_2\ x_3,$$
$$z_2 = \overline{x}_1\, x_2 \overline{x}_3,$$
$$z_3 = \overline{x}_1\ x_2\ x_3,$$
$$z_4 = x_1\ \overline{x}_2\ \overline{x}_3.$$

Неиспользуемые кодовые группы имеют вид

$$z_5 = x_1\ \overline{x}_2\ x_3,\ z_6 = x_1\ x_2\ \overline{x}_3,\ z_7 = x_1\, x_2\, x_3.$$

Эти неиспользуемые кодовые группы и составят полные запрещенные входные сигналы. Они определятся также как отрицание от дизъюнкции действующих входных сигналов:

$$e_з = \overline{z_0 \vee z_1 \vee z_2 \vee z_3 \vee z_4} = x_1\overline{x}_2x_3 \vee x_1x_2\overline{x}_3 \vee x_1x_2x_3.$$

Пустой абстрактный входной сигнал для данного алфавита входных сигналов определится как отрицание от дизъюнкции всех абстрактных входных сигналов, включая запрещенные:

$$e = \overline{z_0 \vee z_1 \vee z_2 \vee z_3 \vee z_4 \vee e_3} = 1 = 0.$$

S-событие – это частное событие, определяемое частными входными сигналами, отмеченное соответствующей совокупностью выходных сигналов из алфавита [*Y*] и формализуемое в соответствии с формулами (1.1) и (1.12). Множество *S*-событий определяет функции переходов в НДА Мура.

a-событие – это полное событие, определяемое полными входными сигналами в алфавите [*X*], которому соответствует возможное сочетание частных событий, одновременное существование которых в автомате возможно. Этому сочетанию частных событий ставится в соответствие одно вполне определенное состояние ДА Мура, отмеченное совокупностью выходных сигналов, отмечающих соответствующие частные события. Множество *a*-событий определяет функции переходов ДА Мура.

Q-событие – это событие, определяющее переходы элементарного автомата (триггера), входящего в состав памяти автомата. Модель автомата, представленная системой уравнений для *Q*-событий (СКУ для *Q*-событий), будет недетерминированной относительно *Q*-событий, хотя система, полученная в результате кодирования полных *a*-событий, представляет реальный ДА. Отметим, что детерминизация СКУ для *Q*-событий даст нам СКУ для *a*-событий.

Промежуточное событие – это событие, обычно не отмеченное реальным выходным сигналом, которое играет вспомогательную роль при определении отмеченных и реализуемых в автомате событий.

Пустое (или тупиковое) событие S* – это событие, из которого не может быть получено ни одно из реализуемых в автомате событий, кроме начального события S_0, являющегося исходным для всех событий в автомате. Пустое событие может быть определено выражением, являющимся отрицанием от дизъюнкции всех реализуемых в автомате событий. Пустому событию может быть поставлено в соответствие одно состояние автомата, которое по аналогии будем называть пустым (или тупиковым) состоянием.

Неопределенное событие – это событие, которому соответствует на переходе запрещенный входной сигнал, в результате чего функция переходов на этом шаге алгоритма будет не определена и может быть доопределена совершенно произвольно. Это позволит выполнять упрощение формул, определяющих функции переходов автоматов. Необходимо иметь в виду, что $S_i \& S(-) \neq 0$, тогда как $S_i \& S_* = 0$.

Недостижимое событие – это событие, которое не может быть получено из начального события при любой последовательности входных сигналов, т.е. такое событие не имеет событий-предшественников.

Глава 2
ДЕТЕРМИНИЗАЦИЯ НЕДЕТЕРМИНИРОВАННЫХ АВТОМАТОВ

В предыдущей главе было отмечено, что над НДА возможно выполнение большого количества операций, основными из которых являются операции детерминизации, минимизации и кодирования. Эти операции над НДА рассматриваются как операции над системами канонических уравнений (СКУ), описывающими поведение НДА. Перечисленные основные операции над СКУ и некоторые дополнительные используются для эквивалентных преобразований НДА с целью построения структур управляющих автоматов, удовлетворяющих заданным определенным требованиям.

Будем рассматривать алгоритм детерминизации НДА Мура, имея в виду, что для НДА Мили алгоритм детерминизации в принципе будет такой же.

Максимально возможное число состояний детерминированного автомата, полученного в результате детерминизации эквивалентного ему НДА, представленного m частными событиями, не может быть больше, чем 2^m, и определится из выражения:

$$M \le \sum_{k=0}^{m} C_m^k = 2^m, \qquad (2.1)$$

где C_m^k – число сочетаний из m элементов по k.

Величина M может быть определена так же, как определяется в булевой алгебре максимальное число наборов значений двоичных переменных, на которых задается булева функция m переменных. В нашем случае под булевыми переменными будем понимать сокращенные обозначения частных событий S_j, число которых равно m. Если каждому набору значений переменных S_j будет соответствовать состояние детерминированного автомата, то $M \le 2^m$.

Основу алгоритма детерминизации НД СКУ составляет процесс отыскания на каждом шаге работы алгоритма функционирования автомата множества сочетаний частных событий, одновременное существование которых в автомате возможно. Такому

сочетанию частных событий, названному в гл. 1 полным событием (*a*-событием), ставится в соответствие вполне определенное состояние детерминированного автомата, отмеченное совокупностью выходных сигналов, отмечающих соответствующие частные события.

Для более эффективного решения задачи детерминизации НДА, более наглядного представления этого процесса исходную НД СКУ целесообразно представить в виде прямой таблицы переходов НДА, на основании которой будем строить таблицу переходов детерминированного автомата по алгоритму, который, как было отмечено ранее, базируется на алгоритме синтеза цифровых автоматов Мура, предложенном в [12].

2.1. Алгоритм детерминизации НДА

Алгоритм построения прямой таблицы переходов детерминированного автомата Мура включает следующие этапы:

1. Строится прямая таблица переходов НДА Мура, и заготавливается прямая таблица переходов детерминированного автомата Мура. По прямой таблице переходов исходного НДА (ПТП НДА) отыскивается совокупность (сочетание) исходных – начальных событий и/или частных событий выводимых из них. Этой совокупности (сочетанию) начальных событий ставится в соответствие начальное состояние детерминированного автомата, а их обозначения записываются во второй столбец в первой строке прямой таблицы переходов детерминированного автомата Мура (ПТП ДА Мура).

2. Для каждого исходного (вначале начального) частного события, входящего в совокупность (сочетание) частных событий, записанных во втором столбце ПТП ДА Мура, выписываются в третий столбец подмножества частных входных сигналов, которые вызывают переходы из этих исходных событий.

В каждом из подмножеств частных входных сигналов находят все не эквивалентные между собой и не эквивалентные нулю сочетания.

3. Выполняют анализ и корректировку состава каждого из полученных подмножеств сочетаний частных входных сигналов,

вызывающих переходы из исходного события, на основании которых:

а) оставляют для дальнейшего рассмотрения сочетания, не являющиеся выводимыми ни из какого одного из сочетаний рассматриваемого подмножества, и те из выводимых, для которых их истинность не влечет истинность хотя бы одного из сочетаний, из которого это сочетание само выводимо;

б) исключают из дальнейшего рассмотрения те из выводимых сочетаний частных входных сигналов рассматриваемого подмножества, истинность которых всегда влечет истинность хотя бы одного из сочетаний, из которых они выводимы;

в) выполняют проверку состава частных входных сигналов рассматриваемого подмножества на полноту состава. Нарушение полноты состава частных входных сигналов будет иметь место в том случае, если логическая сумма всех частных входных сигналов данного подмножества не равна единице. Такой состав частных входных сигналов не удовлетворяет условиям полноты переходов в автомате для рассматриваемого исходного события.

Для устранения неполноты состава частных входных сигналов определяются дополнительные частные входные сигналы, которые для рассматриваемого подмножества будут запрещенными. Для них соответствующие события перехода будут неопределенными. Запрещенные входные сигналы определятся из выражения, представляющего собой отрицание от дизъюнкции всех частных входных сигналов рассматриваемого подмножества.

4. Для всех откорректированных подмножеств частных входных сигналов, действующих на данном шаге алгоритма работы ДА Мура, находится множество всех не эквивалентных между собой и не эквивалентных нулю сочетаний частных входных сигналов.

5. Выполняются анализ и корректировка полученного множества сочетаний частных входных сигналов, действующих на данном переходе ДА Мура, на основании которых:

а) оставляют для дальнейшего рассмотрения и записывают в третий столбец ПТП ДА сочетания, не являющиеся выводимыми ни из какого одного из сочетаний рассматриваемого множества, и те из выводимых, для которых их истинность не влечет истинность хотя бы одного из сочетаний, из которого это сочетание само выводимо;

б) исключают из дальнейшего рассмотрения те из выводимых сочетаний, истинность которых влечет истинность хотя бы одного из сочетаний, из которых они выводимы;

в) к оставленным выводимым сочетаниям применяют операцию развертывания, умножая их на выражение вида $(x_i \vee \overline{x}_j)...(x_j \vee \overline{x}_j)$, где $x_i, ..., x_j$ – элементарные входные сигналы, которые не вошли в выводимое сочетание элементарных входных сигналов по сравнению с сочетанием, из которого они выводятся. Такая операция необходима для обеспечения условий однозначности на рассматриваемом переходе ДА (шаге алгоритма). Из полученных в результате выполнения операции развертывания частных входных сигналов удаляются те из них, которые эквивалентны ранее полученным. В том случае, если выводимое сочетание частных входных сигналов выводимо из нескольких сочетаний частных входных сигналов, при операции развертывания необходимо использовать те элементарные входные сигналы $x_i, ..., x_j$, которые входят в сочетание с наибольшим числом букв.

В результате выполнения п. 5 алгоритма детерминизации будут получены все полные входные сигналы $X(a_m, a_s)$, действующие на рассматриваемом шаге работы алгоритма ДА.

Примечание. В том случае, если детерминизация НДА используется только с целью определения состава всех полных событий, то операцию развертывания к выводимым сочетаниям частных входных сигналов можно не применять, так как новых полных событий не получим. Для таких входных сигналов в столбце 4 будем ставить дополнительно знак (–).

6. Для каждого полного входного сигнала $X(a_m, a_s)$, записанного в третьем столбце, определяют, используя исходную ПТП НДА, совокупность (сочетание) частных событий, которые выводимы из каждого исходного (вначале начального) события. Этой совокупности (сочетанию) частных событий ставится в соответствие определенное состояние ДА (полное событие a_s), а их обозначения записываются в четвертом столбце таблицы переходов ДА.

В том случае, если на данном шаге работы автомата в какой-либо строке ПТП ДА имеет место запрещенный входной сигнал, в четвертом столбце в рассматриваемой строке ставится черточка (или событие $S(-)$), означающая, что событие или совокупность

событий (полное событие) на этом переходе является неопределенной.

Если для какого-нибудь сочетания частных входных сигналов (полного входного сигнала) и для исходных частных событий, входящих в рассматриваемую совокупность, не выводимо ни одно из частных событий НДА, то четвертый столбец в рассматриваемой строке отмечается обозначением пустого события *S**.

7. Каждое из полученных новых сочетаний частных событий (полное событие) записывается во второй столбец очередной строки таблицы переходов автомата, и вновь повторяются этапы по пп. 2, 3, 4, 5, 6, 7 до тех пор, пока не будут исчерпаны все новые сочетания частных событий, записываемые в четвертый столбец.

8. Отмечаются все совокупности (сочетания) частных событий (полные события) соответствующими обозначениями частных выходных сигналов.

Сочетания частных промежуточных событий (состояния автомата), которые не отмечены никакими выходными сигналами, отмечаются знаком (–). Так как знак (–) не противоречит никакому частному выходному сигналу, то при необходимости можно дополнительно отметить эти состояния ДА любым (в том числе и пустым) выходным сигналом.

9. После окончания построения ПТП ДА ее еще раз проверяют на однозначность и полноту переходов. Для однозначности переходов должно выполняться условие ***равенства нулю*** всех попарных произведений входных сигналов на каждом шаге работы алгоритма ДА, а для полноты переходов логическая сума всех полных входных сигналов на каждом шаге работы алгоритма ДА должна ***равняться единице***.

По полученной прямой таблице переходов детерминированного автомата Мура нетрудно построить детерминированную СКУ по следующему алгоритму:

1. В прямой таблице переходов, полученной после детерминизации, отыскиваются все пути, которые оканчиваются одинаковыми полными событиями (состояниями) $a_s(t+1)$, и составляются для них конъюнкции из обозначений полных событий (состояний) $a_m(t)$, стоящих в начале пути перехода, с соответствующими соче-

таниями частных входных $X(a_m, a_s)$ сигналов, действующих на переходе от a_m к a_s.

2. Полученные конъюнкции $a_m(t)\&X(a_m, a_s)(t)$ объединяют знаком дизъюнкции. Такую операцию выполняют для всех полных событий (состояний).

3. Все полные события (состояния), представленные в левой части уравнений СКУ, отмечаются соответствующими сочетаниями выходных сигналов.

П р и м е р 2.1. Сформировать полные входные сигналы для одного из шагов алгоритма работы ДА.

Пусть на данном шаге алгоритма работы автомата заданы следующие два подмножества частных входных сигналов:

$$[X]_1 = \left[x_1\overline{x}_3, \overline{x}_1x_2, x_3, \overline{x}_3\right],$$

$$[X]_2 = \left[x_1, \overline{x}_1\right].$$

В соответствии с п. 2 алгоритма детерминизации получим следующие сочетания частных входных сигналов в подмножестве $[X]_1$:

$$\left[x_1\overline{x}_3, \overline{x}_1x_2x_3, \overline{x}_1x_2\overline{x}_3, \overline{x}_1x_2, x_3, \overline{x}_3\right].$$

Из анализа полученных сочетаний частных входных сигналов следует, что в данном сочетании имеют место:

– сочетания, которые не являются выводимыми ни из какого одного из сочетаний подмножества $[X]_1$. Для нашего примера к таким сочетаниям относятся: $x_1\overline{x}_3$, $\overline{x}_1x_2x_3$, $\overline{x}_1x_2\overline{x}_3$; такие сочетания остаются для дальнейшего рассмотрения;

– сочетание $\overline{x}_1x_2$, которое выводимо из сочетаний $\overline{x}_1x_2x_3$ и $\overline{x}_1x_2\overline{x}_3$. Так как истинность сочетания $\overline{x}_1x_2$ всегда влечет истинность одного из двух сочетаний, из которых выводим частный входной сигнал $\overline{x}_1x_2$, то данный входной сигнал исключается из дальнейшего рассмотрения;

– элементарные сочетания – входной сигнал x_3, выводимый из сочетания $\overline{x}_1x_2x_3$, и входной сигнал $\overline{x}_3$, выводимый из сочетаний $x_1\overline{x}_3$ и $\overline{x}_1x_2\overline{x}_3$. Истинность сигнала x_3 еще не означает истинность сочетания $\overline{x}_1x_2x_3$, а истинность сигнала $\overline{x}_3$ еще не означает всегда истинность хотя бы одного из сочетаний $x_1\overline{x}_3$ и $\overline{x}_1x_2\overline{x}_3$. Поэтому сигналы x_3 и $\overline{x}_3$ остаются для дальнейшего рассмотрения.

Таким образом, откорректированное подмножество частных входных сигналов $[X]_1$ будет иметь вид

$$[X]_1 = \left[x_1\overline{x}_3, \overline{x}_1x_2x_3, \overline{x}_1x_2\overline{x}_3, x_3, \overline{x}_3\right].$$

Сочетания всех частных входных сигналов для двух заданных подмножеств $[X]_1$ и $[X]_2$ после корректировки подмножества $[X]_1$ будут представлены следующим множеством:

$$[X]_{1,2} = \left[x_1\overline{x}_3,\ x_1x_3,\ \overline{x}_1x_2x_3,\ \overline{x}_1x_2\overline{x}_3,\ \overline{x}_1x_3,\ x_3,\ \overline{x}_3,\ x_1,\ \overline{x}_1\right].$$

В соответствии с пп. 5,а и 5,б алгоритма детерминизации из полученного множества сочетаний частных входных сигналов остаются для дальнейшего рассмотрения следующие шесть сочетаний:

$$\left[x_1\overline{x}_3,\ x_1x_3,\ \overline{x}_1x_2x_3,\ \overline{x}_1x_2\overline{x}_3,\ \overline{x}_1x_3,\ \overline{x}_1,\ \overline{x}_3\right].$$

К двум последним сочетаниям частных входных сигналов в соответствии с п. 5,в алгоритма детерминизации необходимо применить ***операцию развертывания***, так как каждое из них является выводимым из другого сочетания рассматриваемого множества и их истинность не влечет истинность сочетаний, из которых они выводимы. Для нашего примера сочетание $\overline{x}_1x_3$ выводимо из сочетания $\overline{x}_1x_2x_3$, а сочетание $\overline{x}_1\overline{x}_3$ выводимо из сочетания $\overline{x}_1x_2\overline{x}_3$. Операции развертывания будут иметь вид

$$\overline{x}_1x_3(x_2 \vee \overline{x}_2) \text{ и } \overline{x}_1\ \overline{x}_3(x_2 \vee \overline{x}_2).$$

В результате имеем после учета эквивалентных ранее полученных сигналов следующие ***полные входные*** сигналы, которые действуют на данном шаге алгоритма работы ДА:

$$\left[x_1\overline{x}_3,\ x_1x_3,\ \overline{x}_1x_2x_3,\ \overline{x}_1x_2\overline{x}_3,\ \overline{x}_1(\overline{x}_2)x_3,\ \overline{x}_1(\overline{x}_2)\overline{x}_3\right],$$

где переменная в скобках свидетельствует о применении операции развертывания.

П р и м е р 2.2. Сформировать полные входные сигналы для одного из шагов алгоритма работы ДА, когда на данном шаге алгоритма работы автомата заданы два подмножества частных входных сигналов, одно из которых не удовлетворяет условиям полноты.

$$[X]_1 = [\overline{x}_2, x_2,\ \overline{x}_1\ x_2],$$
$$[X]_2 = [\overline{x}_3, x_2x_3].$$

Операции определения сочетаний частных входных сигналов в заданных подмножествах новых частных входных сигналов не дают, поэтому в соответствии с п. 5,в алгоритма детерминизации выполняем проверку заданных подмножеств на полноту состава входных сигналов. Из двух подмножеств второе подмножество $[X]_2$ не удовлетворяет условиям полноты, так как дизъюнкция входных сигналов не равна единице:

$$\overline{x}_3 \vee x_2 x_3 = \overline{x}_3 \vee x_2 .$$

Запрещенный входной сигнал для подмножества $[X]_2$, определяемый как отрицание от дизъюнкции входных сигналов, имеет вид

$$\overline{\overline{x}_3 \vee x_2} = \overline{x}_2 x_3 .$$

Таким образом, скорректированные подмножества частных входных сигналов примут следующий вид:

$$[X]_1 = [\overline{x}_2, x_2,\ \overline{x}_1 x_2\ ],$$

$$[X]_2 = [\overline{x}_3, x_2 x_3, (\overline{x}_2 x_3)^*].$$

Сочетания всех частных входных сигналов для откорректированных подмножеств $[X]_1$ и $[X]_2$ будут представлены следующим множеством:

$$[X]_{1,2} = \left[\overline{x}_2\overline{x}_3, x_2\overline{x}_3, \overline{x}_1 x_2\overline{x}_3, x_2 x_3, \overline{x}_1 x_2 x_3, \overline{x}_1 x_2, (\overline{x}_2, x_3)^*, \overline{x}_2, x_2, \overline{x}_3 \right].$$

После удаления выводимых сочетаний, истинность которых всегда влечет истинность одного из сочетаний, из которых эти сочетания выводимы, для дальнейшего рассмотрения остаются следующие шесть сочетаний:

$$\left[x_2 x_3,\ x_2\overline{x}_3,\ \overline{x}_1 x_2\overline{x}_3, \overline{x}_2\overline{x}_3,\ \overline{x}_1 x_2 x_3,\ (\overline{x}_2, x_3)^* \right].$$

После применения к первым двум выводимым сочетаниям частных входных сигналов операции развертывания получим следующие полные входные сигналы, действующие на данном шаге алгоритма работы ДА:

$$\left[(x_1) x_2 x_3,\ (x_1) x_2\overline{x}_3,\ \overline{x}_1 x_2\overline{x}_3, \overline{x}_2\overline{x}_3,\ \overline{x}_1 x_2 x_3,\ (\overline{x}_2, x_3)^* \right].$$

2.2. Пример детерминизации недетерминированных автоматов с построением систем канонических уравнений и систем выходных функций для автоматов Мура и Мили

В качестве примера детерминизации используется НДА Мура, заданный графом (см. рис. 1.1). В результате детерминизации необходимо получить:

– прямую таблицу переходов ДА Мура, эквивалентного заданному НДА;

– системы канонических уравнений ДА Мура и ДА Мили;

– системы выходных функций (СВФ) ДА Мура и ДА Мили.

Порядок построения:

а) по графу НДА Мура (см. рис. 1.1) строим прямую таблицу переходов исходного НДА (ПТП НДА) Мура, которая будет иметь вид (табл. 2.1), где для запрещенного входного сигнала $(\overline{x}_2 x_3)^*$ принято событие перехода S_4. В нашем примере при построении ПТП ДА событие, принятое в качестве неопределенного, будем заключать в скобки, т.е. имеем: $S_{(-)} = (S_4)$;

Таблица 2.1

Шаг алгоритма	Исходное частное событие и отмечающий его выходной сигнал $S_i(t)(Y_i)$	Частный входной сигнал на переходе $X_{i,j}(t)$	Событие перехода $S_j(t+1)$
1	$S_0(y_0)$	$\overline{x}_1$	S_0
		x_1	S_1, S_2
2	$S_1(y_1)$	$\overline{x}_2$	S_1
		x_2	S_3
		$\overline{x}_1 x_2$	S_2
3	$S_2(y_2)$	$\overline{x}_1$	S_4
		x_1	S_3
		$x_1 \overline{x}_2$	S_5
4	$S_3(y_3, y_4)$	$\overline{x}_3$	S_3
		$x_2 x_3$	S_4
		$(\overline{x}_2 x_3)^*$	$S_{(-)} = S_4$
5	$S_4(y_2, y_4)$	$\overline{x}_4$	S_4
		x_4	S_3
6	$S_5(y_1, y_3)$	$\overline{x}_4$	S_4
		x_4	S_3

б) в соответствии с алгоритмом детерминизации на основании табл. 2.1 получим следующую прямую таблицу переходов ДА Мура (табл. 2.2);

Таблица 2.2

Шаг алгоритма	Сочетание (конъюнкция) частных исходных событий в момент времени (t)	Подмножества частных входных сигналов на переходе $[X_{i,j}]$	Сочетание (конъюнкция) частных событий в момент времени ($t+1$)
	Полное событие $a_m(t)(Y_m)$	Сочетание (конъюнкция) частных входных сигналов на переходе (полный входной сигнал) $X(a_m,a_s)(t)$	Полное событие на переходе $a_s(t+1)$
1	2	3	4
1	$\underline{S_0(y_0)}$ a_0	$\overline{x_1}$ x_1	S_0 / a_0 S_1S_2 / a_1
2	$\underline{S_1S_2(y_1y_2)}$ a_1	$[\overline{x_2}, x_2, \overline{x_1}x_2]$, $[\overline{x_1}, x_1, x_1\overline{x_2}]$	
		$\underline{\overline{x_1}\overline{x_2}}$	S_1S_4 / a_2
		$\underline{\overline{x_1}x_2}$	$S_2S_3S_4 / a_3$
		$\underline{x_1\overline{x_2}}$	$S_1S_3S_5 / a_4$
		$\underline{x_1x_2}$	S_3 / a_5
3	$\underline{S_1S_4(y_1y_2y_4)}$ a_2	$[\overline{x_2}, x_2, \overline{x_1}x_2]$, $[\overline{x_4}, x_4]$	
		$\overline{x_2}\overline{x_4}$	S_1S_4 / a_2
		$\overline{x_2}x_4$	S_1S_3 / a_6
		$(x_1)x_2\overline{x_4}$	S_3S_4 / a_7
		$(x_1)x_2x_4$	S_3 / a_5
		$\overline{x_1}x_2\overline{x_4}$	$S_2S_3S_4 / a_3$
		$\overline{x_1}x_2x_4$	S_2S_3 / a_8

Продолжение табл. 2.2

1	2	3	4
4	$\frac{S_2S_3S_4(y_2y_3y_4)}{a_3}$	$[\overline{x_1}, x_1, x_1\overline{x_2}]$, $[\overline{x_3}x_2x_3, (\overline{x_2}x_3)^*], [\overline{x_4}, x_4]$	
		$\underline{\overline{x_1}\,\overline{x_3}}$	S_3S_4 / a_7
		$x_1(x_2)\overline{x_4}$	S_3S_4 / a_7
		$x_1\overline{x_2}\,\overline{x_4}$	$S_3S_4S_5 / a_{10}$
		$\overline{x_1}x_3\overline{x_4}$	S_4 / a_9
		$x_1(x_2)\overline{x_3}x_4$	S_3 / a_5
		$x_1\overline{x_2}\,\overline{x_3}x_4$	S_3S_5 / a_{11}
		$x_2x_3x_4$	S_3S_4 / a_7
		$x_1\overline{x_2}x_3x_4$	$S_3(S_4)S_5 / a_{10}$
		$\overline{x_1}\,\overline{x_2}x_3x_4$	$S_3(S_4) / a_7$
5	$\frac{S_1S_3S_5(y_1y_3y_4)}{a_4}$	$[\overline{x_2}, x_2, \overline{x_1}x_2]$, $[\overline{x_3}, x_2x_3, (\overline{x_2}x_3)^*], [\overline{x_4}, x_4]$	
		$\overline{x_2}\,\overline{x_3}\,\overline{x_4}$	$S_1S_3S_4 / a_{12}$
		$\overline{x_1}x_2\overline{x_4}$	$S_2S_3S_4 / a_3$
		$x_1(x_2)\overline{x_4}$	S_3S_4 / a_7
		$\overline{x_2}\,\overline{x_3}x_4$	S_3 / a_6
		$(x_1)x_2\overline{x_3}x_4$	S_3 / a_5
		$\overline{x_1}x_2\overline{x_3}x_4$	S_2S_3 / a_8
		$\overline{x_1}x_2x_3x_4$	$S_2S_3S_4 / a_3$
		$(x_1)x_2x_3x_4$	S_3S_4 / a_7
		$\overline{x_2}x_3\overline{x_4}$	$S_1(S_4) / a_2$
		$x_2x_3\overline{x_4}$	$S_1S_3(S_4) / a_{12}$

Продолжение табл. 2.2

1	2	3	4
6	$\underline{S_3(y_3y_4)}$ a_5	$[\overline{x_3}, x_2x_3, (\overline{x_2}x_3)^*]$	
		$\underline{\overline{x_3}}$	S_3 / a_5
		$\underline{x_2x_3}$	S_4 / a_9
		$\underline{(\overline{x_2}x_3)}$	$(S_4) / a_9$
7	$\underline{S_1S_3(y_1y_3y_4)}$ a_6	$[\overline{x_2}, x_2, \overline{x_1}x_2], [\overline{x_3}, x_2, x_3, (\overline{x_2}x_3)^*]$	
		$\overline{x_2}\,\overline{x_3}$	S_1S_3 / a_6
		$(x_1)x_2\overline{x_3}$	S_3 / a_5
		$\overline{x_1}x_2\overline{x_3}$	S_2S_3 / a_8
		$(x_1)x_2x_3$	S_3S_4 / a_7
		$\overline{x_1}x_2x_3$	$S_2S_3S_4 / a_3$
		$\overline{x_2}x_3$	$S_1(S_4) / a_2$
8	$\underline{S_3S_4(y_2y_3y_4)}$ a_7	$[\overline{x_3}, x_2x_3, (\overline{x_2}x_3)^*], [\overline{x_4}, x_4]$	
		$\overline{x_3}\,\overline{x_4}$	S_3S_4 / a_7
		$\overline{x_3}x_4$	S_3 / a_5
		$x_3\overline{x_4}$	S_4 / a_9
		x_3x_4	S_3S_4 / a_7
9	$\underline{S_2S_3(y_2y_3y_4)}$ a_8	$[\overline{x_1}, x_1, x_1\overline{x_2}], [\overline{x_3}, x_2x_3, (\overline{x_2}x_3)^*]$	
		$\overline{x_1}\,\overline{x_3}$	S_3S_4 / a_7
		$x_1(x_2)\overline{x_3}$	S_3 / a_5
		$x_1\overline{x_2}\,\overline{x_3}$	S_3S_5 / a_{11}
		$\overline{x_1}x_3$	S_4 / a_9
		$x_1x_2x_3$	S_3S_4 / a_7
		$x_1\overline{x_2}x_3$	$S_3(S_4)S_5 / a_{10}$
10	$\underline{S_4(y_2y_4)}$ a_9	$\overline{x_4}$	S_4 / a_9
		x_4	S_3 / a_5

Окончание табл. 2.2

1	2	3	4
11	$\frac{S_3S_4S_5(y_1y_2y_3y_4)}{a_{10}}$	$[\overline{x_3}, x_2x_3, (\overline{x_2}x_3)^*], [\overline{x_4}, x_4]$	
		$\overline{x_3}\,\overline{x_4}$	S_3S_4 / a_7
		$\overline{x_3}x_4$	S_3 / a_5
		x_3x_4	S_3S_4 / a_7
		$x_3\overline{x_4}$	$(S_4) / a_9$
12	$\frac{S_3S_5(y_1y_3y_4)}{a_{11}}$	$[\overline{x_3}, x_2x_3, (\overline{x_2}x_3)^*], [\overline{x_4}x_4]$	
		$\overline{x_3}\,\overline{x_4}$	S_3S_4 / a_7
		$\overline{x_3}x_4$	S_3 / a_5
		x_3x_4	S_3S_4 / a_7
		$x_3\overline{x_4}$	$(S_4) / a_9$
13	$\frac{S_1S_3S_4(y_1y_2y_3y_4)}{a_{12}}$	$[\overline{x_2}, x_2, \overline{x_1}x_2],$ $[\overline{x_3}, x_2x_3, (\overline{x_2}x_3)^*], [\overline{x_4}, x_4]$	
		$\overline{x_2}\,\overline{x_3}\,\overline{x_4}$	$S_1S_3S_4 / a_{12}$
		$(x_1)x_2\overline{x_4}$	S_3S_4 / a_7
		$\overline{x_1}x_2\overline{x_4}$	$S_2S_3S_4 / a_3$
		$\overline{x_2}\,\overline{x_3}x_4$	S_1S_3 / a_6
		$(x_1)x_2\overline{x_3}x_4$	S_3 / a_5
		$\overline{x_1}x_2\overline{x_3}x_4$	S_2S_3 / a_8
		$\overline{x_1}x_2x_3x_4$	$S_2S_3S_4 / a_3$
		$(x_1)x_2x_3x_4$	S_3S_4 / a_7
		$\overline{x_2}x_3\overline{x_4}$	$S_1(S_4) / a_2$
		$\overline{x_2}x_3x_4$	$S_1S_3(S_4) / a_{12}$

в) отметим некоторые особенности, связанные с оформлением табл. 2.2. Для отдельных шагов алгоритма работы автомата (шаги алгоритма 3–5, 7, 9, 13) к частным выводимым входным сигналам была применена операция развертывания. Результат этой операции после удаления входных сигналов, эквивалентных ранее полученным, отмечен заключением элементарных входных сигналов в круглые скобки.

В том случае, если детерминизация НД СКУ выполняется в соответствии с примечанием п. 5 алгоритма детерминизации, то операция развертывания к выводимым входным сигналам не применяется, и для таких входных сигналов ставится в столбце 4 дополнительный знак (–). Например, для двух переходов третьего шага алгоритма работы автомата табл. 2.2 имела бы вид:

a_2	x_2x_4 x_2x_4	$S_3S_4(-)/a_7$ $S_3(-)/a_5$

Для некоторых переходов (шаги алгоритма 4–8, 10–12) сочетания частных входных сигналов включают и запрещенный входной сигнал $(\overline{x}_2x_3)^*$, для которого событие перехода может быть определено произвольно. В нашем случае в качестве события перехода для такого сигнала принято событие S_4. В общем случае вместо неопределенного события перехода, обычно отмечаемого черточкой, принимается такое событие, которое может привести к упрощению выражений СКУ и СВФ;

г) по полученной прямой таблице переходов детерминированного автомата Мура (см. табл. 2.2), пользуясь рассмотренным ранее алгоритмом построения СКУ по прямой таблице переходов и учитывая указанные выше замечания по оформлению табл. 2.2, получим отмеченную детерминированную СКУ, определяющую все переходы в автомате Мура, заданном графом НДА (см. рис. 1.1). После преобразования этой СКУ с целью ее упрощения за счет минимизации булевых функций, представленных дизъюнкцией частных входных сигналов при одних и тех же состояниях, отмеченная СКУ полностью определенного детерминированного автомата Мура примет следующий вид:

$$a_0^{y_0}(t+1) = a_0\overline{x}_1 \vee x_0;$$

$$a_1^{y_1y_2}(t+1) = a_0x_1;$$

$$a_2^{y_1y_2y_4}(t+1) = a_1\overline{x}_1\overline{x}_2 \vee a_2\overline{x}_2\overline{x}_4 \vee (a_4 \vee a_{12})\overline{x}_2x_3\overline{x}_4 \vee a_6\overline{x}_2x_3;$$

$$a_3^{y_2y_3y_4}(t+1) = a_1\overline{x_1}x_2 \vee a_2\overline{x}_1x_2\overline{x}_4(a_4 \vee a_{12})(\overline{x}_1x_2\overline{x}_4 \vee \overline{x}_1x_2x_3) \vee a_6\overline{x_1}x_2x_3;$$

$$a_4^{y_1y_3y_4}(t+1) = a_1x_1\overline{x}_2;$$

$$a_5^{y_3y_4}(t+1) = a_1x_1x_2 \vee a_2x_1x_2x_4 \vee (a_4 \vee a_{12} \vee a_3)x_1x_2\overline{x}_3x_4 \vee a_5\overline{x}_3 \vee$$

$$\vee(a_6 \vee a_8)x_1x_2\overline{x}_3 \vee (a_7 \vee a_{10} \vee a_{11})\overline{x}_3x_4 \vee a_9x_4;$$

$$a_6^{y_1y_3y_4}(t+1) = a_2\bar{x}_2x_4 \vee (a_4 \vee a_{12})\bar{x}_2\bar{x}_3x_4 \vee a_6\bar{x}_2\bar{x}_3;$$

$$a_7^{y_2y_3y_4}(t+1) = a_2x_1x_2\bar{x}_4 \vee a_3(\bar{x}_1\bar{x}_3 \vee \bar{x}_1x_4 \vee x_1x_2x_3 \vee x_1x_2\bar{x}_4) \vee$$
$$\vee(a_4 \vee a_{12})(x_1x_2\bar{x}_4 \vee x_1x_2x_3) \vee a_6x_1x_2x_3 \vee (a_7 \vee a_{10} \vee a_{11}) \wedge$$
$$\wedge(\bar{x}_3\bar{x}_4 \vee x_3x_4) \vee a_8(\bar{x}_1\bar{x}_3 \vee x_1x_2x_3);$$

$$a_8^{y_2y_3y_4}(t+1) = a_2\bar{x}_1x_2x_4 \vee (a_4 \vee a_{12})\bar{x}_1x_2\bar{x}_3x_4 \vee a_6\bar{x}_1x_2\bar{x}_3; \qquad (2.2)$$

$$a_9^{y_2y_4}(t+1) = a_3\bar{x}_1x_3\bar{x}_4 \vee a_5x_3 \vee (a_7 \vee a_{10} \vee a_{11})x_3\bar{x}_4 \vee a_8\bar{x}_1x_3 \vee a_9\bar{x}_4;$$

$$a_{10}^{y_1y_2y_3y_4}(t+1) = a_3(x_1\bar{x}_2x_3 \vee x_1\bar{x}_2\bar{x}_4) \vee a_8x_1\bar{x}_2x_3;$$

$$a_{11}^{y_1y_3y_4}(t+1) = a_3x_1\bar{x}_2\bar{x}_3x_4 \vee a_8x_1\bar{x}_2\bar{x}_3;$$

$$a_{12}^{y_1y_2y_3y_4}(t+1) = (a_4 \vee a_{12})\,(\bar{x}_2\bar{x}_3\bar{x}_4 \vee \bar{x}_2x_3x_4);$$

д) система канонических уравнений детерминированного полностью определенного автомата Мили, эквивалентного автомату Мура, может быть получена из отмеченной СКУ (2.2) путем объединения тех состояний (см. табл. 2.2), для которых все частные события на переходах полностью совпадают. Для нашего примера это соответствует двум группам состояний: (a_4, a_{12}) и (a_7, a_{10}, a_{11}). Делая необходимые замены и подстановки переменных вида $a_4 \vee a_{12} = a_4$ и $(a_7 \vee a_{10} \vee a_{11}) = a_7$ в (2.2), получим следующую СКУ автомата Мили:

$$a_0(t+1) = a_0\bar{x}_1 \vee x_0;$$

$$a_1(t+1) = a_0x_1;$$

$$a_2(t+1) = a_1\bar{x}_1\bar{x}_2 \vee a_2\bar{x}_2\bar{x}_4 \vee a_4\bar{x}_2x_3\bar{x}_4 \vee a_6\bar{x}_2x_3;$$

$$a_3(t+1) = a_1\bar{x}_1x_2 \vee a_2\bar{x}_1x_2\bar{x}_4 \vee a_4(\bar{x}_1x_2\bar{x}_4 \vee \bar{x}_1x_2x_3) \vee a_6\bar{x}_1x_2x_3;$$

$$a_4(t+1) = a_1x_1\bar{x}_2 \vee a_4(\bar{x}_2\bar{x}_3\bar{x}_4 \vee \bar{x}_2x_3x_4);$$

$$a_5(t+1) = a_1x_1x_2 \vee a_2x_1x_2x_4 \vee (a_3 \vee a_4)x_1x_2\overline{x_3}x_4 \vee a_5\bar{x}_3 \vee$$
$$\vee a_7\bar{x}_3x_4 \vee (a_6 \vee a_8)x_1x_2\bar{x}_3 \vee a_9x_4; \qquad (2.3)$$

$$a_6(t+1) = a_2\bar{x}_2x_4 \vee a_4\bar{x}_2\bar{x}_3x_4 \vee a_6\bar{x}_2\bar{x}_3;$$

$$a_7(t+1) = a_2x_1x_2\bar{x}_4 \vee a_3(\bar{x}_1x_4 \vee \overline{x_3x_4} \vee x_1\bar{x}_2 \vee x_1x_3) \vee$$
$$\vee a_6x_1x_2x_3 \vee a_7(\bar{x}_3\bar{x}_2 \vee x_3x_4) \vee a_8(\bar{x}_1\bar{x}_3 \vee x_1x_3 \vee x_1\bar{x}_2) \vee a_4(x_1x_2x_3 \vee x_1x_2\bar{x}_4);$$

$$a_8(t+1) = a_2\bar{x}_1x_2x_4 \vee a_4\overline{x_1}x_2\overline{x_3}x_4 \vee a_6\bar{x}_1x_2\bar{x}_3;$$

$$a_9(t+1) = a_3\bar{x}_1x_3\bar{x}_4 \vee a_5x_3 \vee a_7x_3\bar{x}_4 \vee a_8\bar{x}_1x_3 \vee a_9\bar{x}_4;$$

е) система выходных функций для детерминированного полностью определенного автомата Мура в соответствии с (2.2) и (1.3) будет иметь вид

$$
\begin{aligned}
&y_0 = a_0;\\
&y_1 = a_1 \vee a_2 \vee a_4 \vee a_6 \vee \bigvee_{k=10}^{k=12} a_k;\\
&y_2 = \bigvee_{k=1}^{k=3} a_k \vee \bigvee_{k=7}^{k=10} a_k \vee a_{12};\\
&y_3 = \bigvee_{k=3}^{k=8} a_k \vee \bigvee_{k=10}^{k=12} a_k;\\
&y_4 = \bigvee_{k=2}^{k=12} a_k;
\end{aligned}
\qquad (2.4)
$$

ж) систему выходных функций для детерминированного полностью определенного автомата Мили, эквивалентного автомату Мура, получим, исходя из (2.2), (1.4) и принятой системы замены переменных. Она будет иметь такой вид:

$$
\begin{aligned}
&y_0 = a_0\overline{x_1} \vee x_n;\\
&y_1 = a_0x_1 \vee (a_1 \vee a_2 \vee a_4 \vee a_6)\overline{x_2} \vee (a_3 \vee a_8)x_1\overline{x_2};\\
&y_2 = a_0x_1 \vee a_1\overline{x_1} \vee a_2(\overline{x_1}x_2 \vee \overline{x_4}) \vee a_3(\overline{x_1} \vee \overline{x_4} \vee x_3) \vee\\
&\vee a_4(\overline{x_1}x_2 \vee \overline{x_4} \vee x_3) \vee a_6(x_3 \vee \overline{x_1}x_2) \vee a_5x_3 \vee\\
&a_7\overline{x_4} \vee a_8(\overline{x_1} \vee x_3) \vee a_9\overline{x_4};\\
&y_3 = a_1(x_1 \vee x_2) \vee a_2(x_2 \vee x_4) \vee a_3(x_1 \vee \overline{x_3} \vee x_4) \vee\\
&\vee a_4(x_2 \vee \overline{x_3} \vee x_4) \vee a_5\overline{x_3} \vee a_6(x_2 \vee \overline{x_3}) \vee a_7(\overline{x_3} \vee x_4) \vee\\
&\vee a_8(x_1 \vee \overline{x_3}) \vee a_9x_4;\\
&y_4 = \bigvee_{k=2}^{k=12} a_k.
\end{aligned}
\qquad (2.5)
$$

Глава 3
МИНИМИЗАЦИЯ СИСТЕМ КАНОНИЧЕСКИХ УРАВНЕНИЙ

В данной главе рассматриваются вопросы минимизации систем канонических уравнений, являющихся математической моделью как детерминированных, так и недетерминированных автоматов.

При формализации алгоритма функционирования автомата на начальных языках, в первую очередь, составляется формальное описание словесного алгоритма и проверяется его адекватность с помощью моделирования. При этом не особенно обращается внимание на возможную избыточность описания. Так, при использовании языка ОСА в описании могут быть, например, избыточные операторные и условные вершины. При реализации алгоритма желательно устранение этой избыточности, так как она ведет, как правило, к увеличению стоимости реализации. В связи с этим после синтеза СКУ ее необходимо минимизировать.

Минимизация СКУ включает два этапа. Один из них используется для минимизации каждого из уравнений СКУ за счет минимизации булевых функций, представляющих собой ДНФ частных входных сигналов, входящих в уравнения СКУ перед одинаковыми предшествующими событиями. Такая минимизация может быть выполнена с использованием известных методов минимизации булевых функций. В примере детерминированной СКУ, построенной в предыдущей главе, такая минимизация выполнялась. Второй этап минимизации СКУ связан с минимизацией числа событий, представленных в СКУ. В качестве основного метода минимизации числа событий СКУ в монографии рассматривается метод минимизации на основе определения эквивалентного разбиения событий. Кроме этого метода, будет рассмотрен еще один метод минимизации СКУ, который базируется на использовании дополнительной информации о входных сигналах, называемой «учет распределения сдвигов» [23–26]. Этот метод в ряде случаев приводит как к минимизации числа событий, так и минимизации частных входных сигналов в СКУ.

В том случае, если СКУ не детерминирована, ее минимизацию целесообразно выполнить до операции детерминизации, так как в противном случае детерминизация может потребовать большой объем работ. По этой же причине, если функционирование автомата задано на языке ОСАП, то минимизацию НД СКУ

целесообразно выполнять отдельно для каждой частной СКУ, формализующей события в соответствующей ветви алгоритма.

Прежде чем приступить к выполнению основных этапов минимизации СКУ, необходимо убедиться, что исходная СКУ не содержит ***недостижимых событий***. Для этого необходимо по исходной СКУ построить прямую таблицу переходов, которая должна строиться по такой же процедуре, что и прямая таблица переходов при детерминизации НДА, т.е. построение прямой таблицы переходов должно начинаться с исходного начального события. При этом очередное событие может быть занесено в первый столбец таблицы в качестве исходного только в том случае, если оно имело место в один из ***предшествующих шагов*** алгоритма работы автомата (было зафиксировано в качестве события перехода). Указанная процедура построения прямой таблицы переходов позволит избавиться от всех ***недостижимых событий***. В то же время такая таблица является и исходной информацией для выполнения последующих основных этапов минимизации СКУ.

3.1. Минимизация систем канонических уравнений на основе определения эквивалентного разбиения событий

3.1.1. Некоторые понятия и определения

Для минимизации числа событий СКУ можно воспользоваться одним из методов, предложенных для минимизации числа состояний автомата, заданного классической таблицей переходов и выходов, основанного на определении эквивалентного разбиения автомата [27]. При использовании таких методов возникает необходимость в их коррекции, которая обусловлена применением частных входных сигналов при описании событий. Прежде чем приступить к рассмотрению методики определения эквивалентного разбиения автомата, введем некоторые понятия и определения, полезные при дальнейшем рассмотрении вопросов минимизации СКУ. Эти понятия и определения базируются на заимствовании аналогичных понятий, применительно к минимизации внутренних состояний автомата [27].

Эквивалентность событий. События S_α и S_β эквивалентны, если при переходе из них под воздействием любой входной после-

довательности сигналов формируются одинаковые выходные последовательности сигналов. Эквивалентность событий S_α и S_β будем обозначать равенством $S_\alpha = S_\beta$. Отношения эквивалентности обладают свойствами: рефлексивности ($S_\alpha = S_\alpha$), симметричности (если $S_\alpha = S_\beta$, то $S_\beta = S_\alpha$) и транзитивности (если $S_\alpha = S_\beta$ и $S_\beta = S_\gamma$, то $S_\alpha = S_\gamma$).

***K*-эквивалентность событий**. События S_α и S_β k-эквивалентны, если при переходе из них под воздействием любой входной последовательности сигналов длины k формируются одинаковые выходные последовательности сигналов. Если события S_α и S_β не являются k-эквивалентными, то они называются k-различимыми.

Преемники (или последователи) событий. K-м преемником события S_α по отношению к входной последовательности сигналов длины k называют событие, которое формируется из события S_α под воздействием входной последовательности сигналов длины k.

Определение эквивалентности одних событий при установленной эквивалентности других событий:

а) если события S_α и S_β являются k-эквивалентными и если их k-е преемники по отношению к любой входной последовательности сигналов длины k являются эквивалентными, то $S_\alpha = S_\beta$. С другой стороны, можно утверждать, что, если события S_α и S_β являются k-эквивалентными, а их k-е преемники по отношению к любой входной последовательности сигналов длины k являются k-различимыми, $S_\alpha \neq S_\beta$;

б) если события S_α и S_β являются эквивалентными, то их k-е преемники по отношению к любой входной последовательности сигналов длины k и для любого k являются эквивалентными.

***K*-эквивалентное разбиение событий СКУ на классы.**

Все множество событий может быть разбито на классы по признакам:

а) все события, принадлежащие одному классу, должны быть k-эквивалентны, их называют также смежными;

б) все события, принадлежащие к различным классам, должны быть k-различимыми.

Такое разбиение называют k-эквивалентным разбиением событий СКУ.

Эквивалентные разбиения событий СКУ. Эквивалентным разбиением событий СКУ называется k-эквивалентное разбиение их, когда в каждом из классов этого разбиения смежные события эквивалентны. Такое разбиение в принципе может быть, если последовательно получать k-эквивалентные разбиения событий, начиная с $k = 1$ для автомата Мили и $k = 0$ для автомата Мура. Та-

кие разбиения последовательно получают до тех пор, пока первый раз не получается разбиение, которое совпадает с предыдущим. Это разбиение событий СКУ и будет эквивалентным разбиением. Для определения такого разбиения событий могут быть использованы различные методы. Рассмотрим один из них, который основан на использовании так называемой таблицы пар [27].

3.1.2. Определение эквивалентного разбиения событий систем канонических уравнений на основе использования таблицы пар

Такое разбиение выполняется последовательным изменением таблицы пар путем вычеркивания по определенному алгоритму тех пар событий, которые не являются эквивалентными. Оставшиеся не вычеркнутые пары будут образовывать все пары эквивалентных событий.

Методику построения таблицы пар и ее использования для определения эквивалентного разбиения событий СКУ будем рассматривать на основе примера автомата Мура, заданного прямой таблицей переходов (табл. 3.1), в которой все события, как исходные, так и события перехода, отмечены выходными сигналами.

Таблица 3.1

Шаг алгоритма	Исходное событие $S_i(t)(Y_i)$	Входной сигнал $X_{i,j}(t)$	Событие перехода $S_j(t+1)(Y_j)$
1	2	3	4
1	$S_0(y_0)$	1	$S_1(y_1)$
2	$S_1(y_1)$	x_1 $\overline{x}_1$	$S_2(y_2)$ $S_3(y_3)$
3	$S_2(y_2)$	1	$S_3(y_3)$
4	$S_3(y_3)$	1	$S_4(y_1)$
5	$S_4(y_1)$	x_1 $\overline{x}_1 x_2$ $\overline{x}_1 \overline{x}_2$	$S_5(y_2)$ $S_6(y_3)$ $S_7(y_3)$
6	$S_5(y_2)$	1	$S_8(y_1)$
7	$S_6(y_3)$	x_3 $\overline{x}_3$	$S_9(y_4)$ $S_{10}(y_5)$

Окончание табл. 3.1

1	2	3	4
8	$S_7(y_3)$	x_3 $\overline{x}_3$	$S_9(y_4)$ $S_{10}(y_5)$
9	$S_8(y_1)$	x_1 $\overline{x}_1$	$S_5(y_2)$ $S_7(y_3)$
10	$S_9(y_4)$	1	$S_k(y_k)$
11	$S_{10}(y_5)$	x_3 $\overline{x}_3$	$S_9(y_4)$ $S_{10}(y_5)$

Такое представление исходной прямой таблицы переходов ЦА Мура позволит использовать ее для построения таблицы пар при определении эквивалентного разбиения событий как для ЦА Мура, так и для эквивалентного ему автомата Мили. Разница будет заключаться лишь в построении основного столбца таблицы пар: для автомата Мура в основном столбце размещают все пары 0-эквивалентных событий S_α и S_β (для $\alpha \neq \beta$), а для автомата Мили – все пары 1-эквивалентных событий.

Напомним, что под ***0-эквивалентными событиями*** понимаются те события, которые отмечены во втором столбце прямой таблицы переходов одинаковыми выходными сигналами, а под ***1-эквивалентными событиями*** понимают события (во втором столбце), при переходе из которых под действием одного и того же входного сигнала формируются одинаковые выходные сигналы (в четвертом столбце).

Для нашего примера имеются три класса 0-эквивалентных событий: (S_1, S_4, S_8), (S_2, S_5), (S_3, S_6, S_7), и три класса 1-эквивалентных событий: (S_1, S_4, S_8), (S_0, S_3, S_5), (S_6, S_7, S_{10}).

Определение эквивалентного разбиения событий для построения минимального ЦА Мили.

Рассматриваемая методика определения эквивалентного разбиения событий на основе использования ПТП ЦА Мура позволяет, наряду с процедурой минимизации числа событий, выполнить также и одновременное преобразование автомата Мура в эквивалентный ему автомат Мили с минимальным числом событий.

Для более компактного представления таблицы пар ее целесообразно создавать в виде подтаблиц, число которых определится для ЦА Мили числом классов 1-эквивалентных событий.

Как было отмечено выше, для автомата Мили основной столбец таблицы пар будет содержать все пары 1-эквивалентных событий. Во все последующие колонки таблицы пар записываются пары событий, которые являются первыми преемниками по отношению к конкретному входному сигналу. Для этой цели колонки таблицы пар обозначают теми входными сигналами, которые не являются выводимыми ни из каких других входных сигналов.

Для их определения необходимо:

– сначала найти по исходной ПТП (см. табл. 3.1) все возможные сочетания входных сигналов для каждого конкретного класса 1-эквивалентных событий;

– из полученных сочетаний входных сигналов использовать для обозначения колонок соответствующей подтаблицы те из них, которые не являются выводимыми ни из каких других сочетаний сигналов. Например, сигнал $\overline{x}_1$ выводим из сигналов $\overline{x}_1 x_2$ и $\overline{x}_1 \overline{x}_2$, поэтому сигналом $\overline{x}_1$ не отмечается столбец таблицы пар.

Предусматривается также одна колонка, для которой переходы к первым преемникам не зависят от входного сигнала. Учитывая изложенное, таблица пар будет иметь следующий вид (табл. 3.2).

В каждой клетке для упрощения таблицы записываются только индексы, определяющие номера событий.

Таблица 3.2

Классы 1-эквивалентных событий	Пары 1-эквивалентных событий	$\overline{x}_1 x_2$	$\overline{x}_1 \overline{x}_2$	x_1	x_3	$\overline{x}_3$	1
S_1, S_4, S_8	**1–4**	3–6	3–7	2–5			
	1–8	3–7	3–7	2–5			
	4–8	6–7	7–7	5–5			
S_0, S_3, S_5	**0–3**						1–4
	0–5						1–8
	3–5						4–8
S_6, S_7, S_{10}	6–7				9–9	10–10	
	6–10				9–9	10–10	
	7–10				9–9	10–10	

Алгоритм нахождения эквивалентного разбиения событий будет включать следующие этапы:

1) последовательно по строкам отыскиваются отличающиеся пары событий (S_α и S_β, где $\alpha \neq \beta$), которые отсутствуют в первом основном столбце таблицы пар. Если в какой-либо строке имеется хотя бы одна такая пара, то в этой строке зачеркивается пара в первом столбце. Такие строки, в которых зачеркнуты пары в первом столбце таблицы, называют выделенными строками. Для нашего примера это пары (3–6), (3–7), (2–5). Для них зачеркиваем в первом столбце таблицы пары (1–4) и (1–8), они выделены в таблице жирным шрифтом;

2) отыскиваются невыделенные строки, в которых имеются пары, зачеркнутые в первом столбце на предыдущем этапе. Если такие строки имеются, то для них зачеркиваются пары в первом столбце. Для нашего примера это строки 4 и 5, для них зачеркиваем пары в первом столбце (0–3) и (0–5). Такой процесс повторяется до тех пор, пока на очередном этапе не обнаруживаются невыделенные строки, в которых имеются пары, зачеркнутые в первом столбце на предыдущем этапе. Оставшиеся не зачеркнутые пары образуют все пары эквивалентных событий. Для нашего примера это пары: (S_4,S_8), (S_3,S_5), (S_6,S_7), (S_6,S_{10}), (S_7,S_{10}).

Рассмотренная методика определения пар эквивалентных событий, основанная на последовательном зачеркивании пар неэквивалентных событий, базируется на использовании введенных ранее свойств (а) и (б) по определению эквивалентности одних событий при установленной эквивалентности других событий.

Учитывая свойства транзитивности для эквивалентных событий, а также состав событий, которые не вошли в пары эквивалентных событий, получим для нашего примера следующее эквивалентное разбиение событий ЦА Мили: (S_0), (S_1), (S_2), (S_3,S_5), (S_4,S_8), (S_6,S_7,S_{10}), (S_9), (S_k).

Определение эквивалентного разбиения событий для построения минимального ЦА Мура.

В том случае, если требуется построить минимальный ЦА Мура, для определения эквивалентного разбиения событий, как было отмечено ранее, необходимо по исходной ПТП ЦА Мура (см. табл. 3.1) найти все классы 0-эквивалентных событий, которые будут использованы для построения основного столбца таблицы пар.

Дальнейшая методика построения таблицы пар для автомата Мура ничем не отличается от рассмотренной методики для автомата Мили.

Для нашего примера таблица пар для определения эквивалентного разбиения событий для построения минимального ЦА Мура будет иметь следующий вид (табл. 3.3).

Таблица 3.3

Классы 0-эквивалентных событий	Пары 0-эквивалентных событий	$\overline{x}_1 x_2$	$\overline{x}_1 \overline{x}_2$	x_1	x_3	$\overline{x}_3$	1
S_1, S_4, S_8	1–4	3–6	3–7	2–5			
	1–8	3–7	3–7	2–5			
	4–8	6–7	7–7	5–5			
S_2, S_5	2–5						3–8
S_3, S_6, S_7	3–6				4–9	4–10	
	3–7				4–9	4–10	
	3–7				9–9	10–10	

На первом шаге алгоритма отыскания эквивалентного разбиения событий находим пары (3–8), (4–9), которые отсутствуют в основном столбце. Для таких пар зачеркиваются соответствующие им пары в основном столбце, к ним относятся: (2–5), (3–6), (3–7).

На втором шаге алгоритма отыскиваются строки, в которых имеются пары, зачеркнутые в основном столбце на первом шаге алгоритма. Для нашего примера это строки 1- и 2-я, для них зачеркиваются пары (1–4) и (1–8).

Для нашего примера на этом шаге алгоритм отыскания эквивалентного разбиения событий свою работу заканчивает, так как больше не обнаруживаются невыделенные строки, в которых имеются пары, зачеркнутые в основном столбце на предыдущем шаге алгоритма. Оставшиеся незачеркнутые пары событий образуют пары эквивалентных событий, к ним относятся: (S_4, S_8) и (S_6, S_7).

Учитывая состав событий, которые не вошли в пары эквивалентных событий, получим следующее эквивалентное разбиение событий для ЦА Мура для нашего примера:

$$(S_0),(S_1),(S_2),(S_3),(S_4, S_8),(S_6, S_7),(S_9),(S_k),(S_5),(S_{10}).$$

3.1.3. Определение минимальной прямой таблицы переходов и минимальной системы канонических уравнений для автоматов Мили и Мура

Цифровой автомат Мили.

Обозначив произвольно каждый класс эквивалентного разбиения событий ЦА Мили своим символом и заменив в исходной ПТП (см. табл. 3.1) каждое обозначение событий, входящих в класс, обозначением введенного символа для этого класса и выполнив зачеркивание повторяющихся строк, получим минимальную прямую таблицу переходов ЦА Мили. Примем для нашего примера следующие обозначения для классов эквивалентного разбиения:

$(S_0) = a_0$; $(S_1) = a_1$; $(S_2) = a_2$; $(S_3,S_5) = a_3$;

$(S_4,S_8) = a_4$; $(S_6,S_7,S_{10}) = a_5$; $(S_9) = a_6$; $(S_k) = a_7$.

После подстановки введенных обозначений в табл. 3.1 и вычеркивания повторяющихся строк получим следующую минимальную прямую таблицу переходов для автомата Мили (табл. 3.4).

Таблица 3.4

Шаг алгоритма	$a_i(t)$	$X_{i,j}(t)$	$a_j(t+1)$	$Y_{i,j}(t)$
1	a_0	1	a_1	y_1
2	a_1	x_1	a_2	y_2
		$\overline{x}_1$	a_3	y_3
3	a_2	1	a_3	y_3
4	a_3	1	a_4	y_1
5	a_4	x_1	a_3	y_2
		$\overline{x}_1$	a_5	y_3
6	a_5	x_3	a_6	y_4
		$\overline{x}_3$	a_5	y_5
7	a_6	1	a_7	y_k

Минимальную СКУ для автомата Мили и соответствующую СВФ можно получить одним из двух способов. **В первом случае**, если минимальная прямая таблица переходов уже построена, то по

ней нетрудно построить минимальную СКУ и СВФ. **Во втором случае**, если минимальная прямая таблица переходов не строилась, а исходный автомат был задан СКУ и СВФ, то минимальную СКУ и СВФ можно построить путем склеивания (объединения) правых частей уравнений для одинаково обозначенных событий и подстановки новых обозначений в исходные СКУ и СВФ. Для нашего примера имеем:

$$S_0 = a_0;\ \ S_1 = a_1; \qquad S_2 = a_2; \qquad S_3 \vee S_5 = a_3; \qquad S_4 \vee S_8 = a_4;$$

$$S_6 \vee S_7 \vee S_{10} = a_5; \qquad S_9 = a_6; \qquad S_k = a_7,$$

откуда получим минимальные СКУ и СВФ автомата Мили:

$$a_1(t+1) = a_0; \qquad y_1 = a_0 \vee a_3;$$

$$a_2(t+1) = a_1 x_1; \qquad y_2 = (a_1 \vee a_4)x_1;$$

$$a_3(t+1) = a_2 \vee a_1\overline{x}_1 \vee a_4 x_1; \qquad y_3 = a_1\overline{x}_1 \vee a_2 \vee a_4\overline{x}_1;$$

$$a_4(t+1) = a_3; \qquad y_4 = a_5 x_3;$$

$$a_5(t+1) = a_4\overline{x}_1 \vee a_5\overline{x}_3; \qquad y_5 = a_5\overline{x}_3;$$

$$a_6(t+1) = a_5 x_3; \qquad y_k = a_6.$$

$$a_7(t+1) = a_6.$$

Цифровой автомат Мура.

Минимальная таблица переходов автомата Мура для нашего примера с учетом эквивалентности двух пар событий и введенных обозначений для классов эквивалентного разбиения:

$$(S_0) = a_0, \qquad (S_1) = a_1, \qquad (S_2) = a_2, \qquad (S_3) = a_{3,1}, \qquad (S_4, S_8) = a_4,$$

$$(S_6, S_7) = a_{5,1}, \quad (S_9) = a_6, \quad (S_k) = a_7, \quad (S_5) = a_{3,2}, \quad (S_{10}) = a_{5,2},$$

будет иметь следующий вид (табл. 3.5).

Таблица 3.5

Шаг	$a_i(t)$	$X_{i,j}(t)$	$a_j(t+1)(Y_j)$
1	2	3	4
1	a_0	1	$a_1(y_1)$
2	a_1	x_1	$a_2(y_2)$
		$\overline{x}_1$	$a_{3,1}(y_3)$
3	a_2	1	$a_{3,1}(y_3)$
4	$a_{3,1}$	1	$a_4(y_1)$

Окончание табл. 3.5

1	2	3	4
5	$a_{3,2}$	1	$a_4(y_1)$
6	a_4	x_1	$a_{3,2}(y_2)$
		$\overline{x}_1$	$a_{5,1}(y_3)$
7	$a_{5,1}$	x_3	$a_6(y_4)$
		$\overline{x}_3$	$a_{5,2}(y_5)$
8	$a_{5,2}$	x_3	$a_6(y_4)$
		$\overline{x}_3$	$a_{5,2}(y_5)$
9	a_6	1	$a_7(y_k)$

В соответствии с табл. 3.5 минимальная отмеченная СКУ и СВФ автомата Мура имеют вид

$$a_1^{y_1}(t+1) = a_0; \qquad y_1 = a_1 \vee a_4;$$

$$a_2^{y_2}(t+1) = a_1 x_1; \qquad y_2 = a_2 \vee a_{3,2};$$

$$a_{3,1}^{y_3}(t+1) = a_1 \overline{x}_1 \vee a_2; \qquad y_3 = a_{3,1} \vee a_{5,1};$$

$$a_{3,2}^{y_2}(t+1) = a_4 x_1; \qquad y_4 = a_6;$$

$$a_4^{y_1}(t+1) = a_{3,1} \vee a_{3,2}; \qquad y_5 = a_{5,2};$$

$$a_{5,1}^{y_3}(t+1) = a_4 \overline{x}_1; \qquad y_k = a_7.$$

$$a_{5,2}^{y_5}(t+1) = (a_{5,1} \vee a_{5,2})\overline{x}_3;$$

$$a_6^{y_4}(t+1) = (a_{5,1} \vee a_{5,2}) x_3;$$

$$a_7^{y_k}(t+1) = a_6.$$

В заключение отметим, что в случае, если структуру управляющего автомата необходимо строить на основе модели ЦА Мили с минимальным числом событий, а исходный алгоритм управления задан моделью ЦА Мура, минимизацию числа событий ЦА Мили можно выполнить на основе использования модели исходного ЦА Мура без предварительного преобразования его в эквивалентный ЦА Мили. Это обеспечивается тем, что рассмотренная методика минимизации числа событий автоматов на основе использования ПТП ЦА Мура позволяет объединить события как за счет

непосредственно минимизации событий ЦА Мура (для нашего примера: объединение событий $S_4 \vee S_8 = a_4$ и $S_6 \vee S_7 = a_5$), так и за счет одновременного преобразования ЦА Мура в эквивалентный ему ЦА Мили (для нашего примера: объединение событий $a_{5,1} \vee a_{5,2} = a_5 = S_6 \vee S_7 \vee S_{10}$ и $a_{3,1} \vee a_{3,2} = a_3 = (S_3 \vee S_5)$).

3.2. Минимизация систем канонических уравнений на основе учета распределения сдвигов

3.2.1. Методика минимизации систем канонических уравнений

В реальных цифровых управляющих автоматах во многих случаях в каждом такте работы автомата могут изменяться не все элементарные входные сигналы по сравнению со значениями их в предшествующем такте. Некоторые элементарные входные сигналы в соответствии с алгоритмом функционирования автомата каждый раз после реализации определенного события S_α могут сохранять свое значение, полученное в предыдущем такте. Такую дополнительную информацию о работе автомата при описании алгоритма его функционирования на языке ОСА, называемую распределением сдвигов, используют для упрощения записи ОСА [23–26].

В тех случаях, когда по условиям распределения сдвигов после появления любого события возможно изменение всех элементарных входных сигналов, никакого упрощения записи СКУ рассматриваемым способом получить нельзя.

Дальнейшее рассмотрение методики минимизации СКУ будет базироваться на работе [28].

Если условия распределения сдвигов заданы, то минимизация СКУ с учетом распределения сдвигов может быть достигнута или за счет сокращения числа реализуемых в автомате событий или их отдельных ветвей, или за счет сокращения записи частных входных сигналов, когда некоторые элементарные входные сигналы, входящие в частный входной сигнал, окажутся равными единице.

Будем предполагать, что СКУ состоит из нескольких уравнений, каждое из которых описывает событие, состоящее из нескольких ветвей:

$$S_\alpha = \bigvee_{i=1}^{k} S_\alpha^i,$$

где α – номер события, входящего в СКУ; k – число ветвей в событии S_α.

Любая i-я ветвь события S_α может быть представлена в следующем виде:

$$S_\alpha^i(t+1) = X_\alpha^i(t) \,\&\, S_{\beta,i}(t), \qquad (3.1)$$

где X_α^i – частный входной сигнал в i-й ветви события S_α; $S_{\beta,i}$ – сокращенное обозначение события, непосредственно предшествующего событию S_α в i-й ветви.

Событие $S_{\beta,i}$, в общем случае также может содержать несколько ветвей, каждая из которых может быть представлена в виде формулы, аналогичной формуле (3.1), т.е. имеем:

$$S_{\beta,i} = \bigvee_{j=1}^{\tau} S_{\beta,i}^j, \quad S_{\beta,i}^j(t+1) = X_{\beta,i}^j(t) \,\&\, S_{\varepsilon,j}(t),$$

где τ – число ветвей в событии $S_{\beta,i}$.

Пусть распределение сдвигов для любого события $S_{\beta,i}$ будет задано в виде следующего множества элементарных входных сигналов:

$$\Delta_{\beta,i} = [x_t, ..., x_l, ..., x_p, ...]. \qquad (3.2)$$

Это означает, что после выполнения события $S_{\beta,i}$ указанные в множестве $\Delta_{\beta,i}$ элементарные входные сигналы сохраняют свое значение в следующем такте, т.е. в данном случае для удобства дальнейшего изложения множество $\Delta_{\beta,i}$ интерпретируется по-иному, чем в [24, 25].

Для анализа заданной СКУ с целью ее минимизации необходимо для каждой ветви события S_α, например, для S_α^i, определить ту составляющую часть входного сигнала X_α^i, значение ко-

торой остается неизменным после выполнения события $\Delta_{\beta,i}$. Здесь следует иметь в виду, что если $X_\alpha^i = 1$, то дальнейший анализ с целью упрощения S_α^i не производится.

Первый этап анализа S_α^i может быть выполнен с помощью следующей операции пересечения:

$$\left[X_\alpha^i\right] \bigcap \left[\Delta_{\beta,i}\right], \tag{3.3}$$

где $\left[X_\alpha^i\right]$ – множество элементарных входных сигналов, входящих в частный входной сигнал X_α^i; $\Delta_{\beta,i} = [\tilde{x}_t, ..., \tilde{x}_l, ..., \tilde{x}_p, ...]$ – множество элементарных входных сигналов, входящих в (3.2), включающее также, кроме переменных, и их инверсии.

Переменные, полученные в результате выполнения операции пересечения (3.3), объединим знаком конъюнкции. В результате получим часть входного сигнала X_α^i, которая остается в соответствии с заданным $\Delta_{\beta,i}$ неизменной после выполнения события $S_{\beta,i}$, являющегося непосредственно предшествующим событию S_α^i. Обозначим этот частный входной сигнал символом $X_\alpha^i(\Delta_{\beta,i})$.

Второй этап анализа с целью упрощения записи S_α^i производится, если выражение (3.3) не является пустым, т.е. не равно $\varnothing$. На этом этапе необходимо отыскать общую часть частных входных сигналов для событий всех ветвей, образующих событие $S_{\beta,i}$. Для этого может быть использована следующая операция пересечения:

$$\left[X_{\beta,i}^1\right] \bigcap \left[X_{\beta,i}^2\right] \bigcap ... \bigcap \left[X_{\beta,i}^j\right] \bigcap ... \bigcap \left[X_{\beta,i}^\tau\right], \tag{3.4}$$

где $\left[X_{\beta,i}^j\right]$ – множество элементарных входных сигналов, входящих в сигнал $X_{\beta,i}^j$, который, в свою очередь, входит в описание *j*-й ветви события $S_{\beta,i}$. Необходимо при этом иметь в виду, что если $X_{\beta,i}^j = 1$, то множество $\left[X_{\beta,i}^j\right]$ включает в себя все элементарные входные сигналы, как с отрицанием, так и без отрицаний (так как дизъюнкция всех 2^L конституент единицы равна единице).

Дальнейший анализ может идти в двух направлениях: первое направление, когда число параллельных ветвей в событии $S_{\beta,i}$ не менее двух, причем не менее одного входного сигнала $X_{\beta,i}^{j} \neq 1$; второе направление, когда для всех ветвей в событии $S_{\beta,i}$ все входные сигналы $X_{\beta,i}^{j} = 1$.

Для первого направления анализа переменные, полученные в результате выполнения операции (3.4) и объединенные знаком конъюнкции, обозначим символом $X_{\beta,i}^{1,2,\ldots,\tau}$. Этот входной сигнал является общей частью входных сигналов всех ветвей события $S_{\beta,i}$.

Если выражение (3.4) не является пустым, то в результате пересечения и конъюнкции двух полученных выше входных сигналов найдем условия упрощения записи исследуемого события:

$$\left[X_{\alpha}^{i}(\Delta_{\beta,i}) \right] \cap \left[X_{\beta,i}^{1,2,\ldots,\tau} \right]; \qquad (3.5,\text{а})$$

$$X_{\alpha}^{i}(\Delta_{\beta,i}) \,\&\, X_{\beta,i}^{1,2,\ldots,\tau}, \qquad (3.5,\text{б})$$

где оба множества в (3.5,а) включают в себя элементарные входные сигналы как с отрицанием, так и без отрицания.

Если выражение (3.5,а) равно $\varnothing$, то запись события S_{α}^{i} не упрощается, в противном случае могут быть два варианта: если конъюнкция (3.5,б) равна нулю, то входной сигнал $X_{\alpha}^{i} = 0$, а следовательно, и событие $S_{\alpha}^{i} = 0$; если конъюнкция (3.5,б) представляет некоторое выражение, не равное нулю, то входящие в это выражение элементарные входные сигналы с отрицанием или без отрицания заменяются единицей в записи входного X_{α}^{i} сигнала события S_{α}^{i}.

Для второго направления анализа необходимо найти пересечение

$$\left[X_{\alpha}^{i} \right] \cap \left[\Delta_{\beta,i} \right] \cap \left[\Delta_{\varepsilon,j} \right], \qquad (3.6)$$

где $\Delta_{\varepsilon,j}$ – распределение сдвигов для события $S_{\varepsilon,j}$, являющегося для события $S_{\beta,i}^{j}$ непосредственно предшествующим.

Для простоты дальнейшего анализа будем рассматривать событие $S_{\beta,i}$ состоящим только из одной ветви. Переменные, полученные в результате выполнения операции пересечения (3.6), объединим знаком конъюнкции и обозначим символом $X_{\alpha}^{i}(\Delta_{\beta,i};\Delta_{\varepsilon,j})$. В дальнейшем находим выражение $X_{\varepsilon,j}^{1,2,\ldots,\gamma}$, аналогичное выражению $X_{\beta,i}^{1,2,\ldots,\tau}$, но полученное из анализа следующего пересечения:

$$\left[X_{\varepsilon,j}^{1}\right]\bigcap\left[X_{\varepsilon,j}^{2}\right]\bigcap\ldots\bigcap\left[X_{\varepsilon,j}^{p}\right]\bigcap\ldots\bigcap\left[X_{\varepsilon,j}^{\gamma}\right], \qquad (3.7)$$

где $\left[X_{\varepsilon,j}^{p}\right]$ – множество элементарных входных сигналов, входящих в сигнал $X_{\varepsilon,j}^{p}$, который, в свою очередь, входит в описание p-й ветви события $S_{\varepsilon,j}$; γ – число ветвей события $S_{\varepsilon,j}$.

Если выражение (3.7) не является пустым, то аналогично выражениям (3.5,а) и (3.5,б) определим пересечение и конъюнкцию:

$$\left[X_{\alpha}^{i}(\Delta_{\beta,i};\Delta_{\varepsilon,j})\right]\bigcap\left[X_{\varepsilon,j}^{1,2,\ldots,\gamma}\right], \qquad (3.8,а)$$

$$\left[X_{\alpha}^{i}(\Delta_{\beta,i};\Delta_{\varepsilon,j})\right]\&\left[X_{\varepsilon,j}^{1,2,\ldots,\gamma}\right]. \qquad (3.8,б)$$

В дальнейшем, используя (3.8,а) и (3.8,б), поступают аналогично рассмотренному выше. В том случае, если найдено, что событие S_{α}^{i} равно нулю, то необходимо произвести коррекцию СКУ, т.е. вычеркнуть строку, описывающую это событие, и приравнять нулю ветви тех событий, в которые S_{α}^{i} входит в качестве непосредственно предшествующего события.

3.2.2. Пример минимизации систем канонических уравнений

Проиллюстрируем рассмотренную методику минимизации СКУ с учетом заданного распределения сдвигов на примере СКУ, полученной по МСА из [23].

$$
\begin{aligned}
&S_1(t+1) = \overline{x}_1 S_0 \vee \overline{x}_1 \overline{x}_4 S_3 \vee \overline{x}_1 x_2 \overline{x}_3 S_1;\\
&S_2(t+1) = S_5;\\
&S_3(t+1) = x_1 x_2 S_1 \vee x_1 \overline{x}_2 S_0;\\
&S_4(t+1) = \overline{x}_2 x_4 S_2 \vee \overline{x}_1 x_4 S_3;\\
&S_5(t+1) = x_1 x_2 S_0 \vee x_2 x_4 S_2 \vee x_2 x_4 S_4 \vee \overline{x}_1 x_2 x_3 S_1;\\
&S_6(t+1) = \overline{x}_4 S_2 \vee \overline{x}_2 S_1 \vee x_1 S_3 \vee (\overline{x}_2 \vee \overline{x}_4) S_4.
\end{aligned}
\qquad (3.9)
$$

Распределение сдвигов для заданной СКУ:

$$
\begin{aligned}
&\Delta_0 = [\varnothing]; && \Delta_4 = [\tilde{x}_4];\\
&\Delta_1 = [\tilde{x}_1]; && \Delta_5 = [\tilde{x}_2, \tilde{x}_3];\\
&\Delta_2 = [\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \tilde{x}_4]; && \Delta_6 = [\varnothing].\\
&\Delta_3 = [\tilde{x}_1, \tilde{x}_2, \tilde{x}_4];
\end{aligned}
\qquad (3.10)
$$

Произведем поочередной анализ событий исходной СКУ (3.9):

1. Событие S_1 состоит из трех ветвей:

а) ветвь $S_1^1(t+1) = \overline{x}_1 S_0$. Для этой ветви имеем: $\left[X_1^1\right] \cap [\Delta_0] = [\overline{x}_1] \cap [\varnothing] = \varnothing$. Поэтому запись этой ветви остается без изменения;

б) ветвь $S_1^2(t+1) = \overline{x}_1 \overline{x}_4 S_3$. Для этой ветви имеем: $\left[X_1^2\right] \cap [\Delta_3] = [\overline{x}_1, \overline{x}_4]$, $X_1^2(\Delta_3) = \overline{x}_1 \overline{x}_4$.

Для события S_3 находим $\left[X_3^1\right] \cap \left[X_3^2\right] = [x_1], X_3^{1,2} = x_1$, откуда $\left[X_1^2(\Delta_3)\right] \cap \left[X_3^{1,2}\right] \neq [\varnothing], X_1^2(\Delta_3) \,\&\, X_3^{1,2} = 0$, поэтому $X_1^2 = 0$ и $S_1^2 = 0$;

в) ветвь $S_1^3(t+1) = \overline{x}_1 x_2 \overline{x}_3 S_1$ Для этой ветви имеем: $\left[X_1^3\right] \cap [\Delta_1] = [\overline{x}_1], X_1^3(\Delta_1) = \overline{x}_1$.

Для события S_1 находим $\left[X_1^1\right] \cap \left[X_1^2\right] \cap \left[X_1^3\right] = [\overline{x}_1], X_3^{1,2,3} = \overline{x}_1$, откуда $\left[X_1^3(\Delta_1)\right] \cap \left[X_3^{1,2,3}\right] \neq [\varnothing], X_1^3(\Delta_1) \,\&\, X_3^{1,2,3} = \overline{x}$, поэтому новая запись входного сигнала для события S_1^3 будет иметь вид $X_1^3 = x_2 \overline{x}_3$.

2. Событие S_2 имеет одну ветвь, для которой $X_2^1 = 1$, поэтому дальнейший анализ не производится.

3. Событие S_3 состоит из двух ветвей:

а) ветвь $S_3^1(t+1) = x_1 x_2 S_1$. Для этой ветви имеем: $\left[X_3^1\right] \cap [\Delta_1] = = [x_1], X_3^1(\Delta_1) = x_1$.

Для события S_1 в п. 1,в получено $X_1^{1,2,3} = \overline{x}_1$, откуда $\left[X_1^3(\Delta_1)\right] \cap \left[X_3^{1,2,3}\right] = [\varnothing], X_3^1(\Delta_1)\,\&\, X_1^{1,2,3} = 0$, поэтому $X_3^1 = 0$ и ветвь $S_3^1 = 0$;

б) ветвь $S_3^2(t+1) = x_1 \overline{x}_2 S_0$. Для этой ветви имеем: $\left[X_3^2\right] \cap [\Delta_0] = \varnothing$, поэтому дальнейший анализ не производится.

4. Событие S_3 состоит из двух ветвей:

а) ветвь $S_4^1(t+1) = \overline{x}_2 x_4 S_2$. Для этой ветви имеем: $\left[X_4^1\right] \cap [\Delta_2] = = [\overline{x}_2, x_4], X_4^1(\Delta_2) = \overline{x}_2 x_4$.

Для события S_2 имеем $X_2^1 = 1$. Поэтому рассматриваем событие S_5, являющееся непосредственно предшествующим событию S_2.

В этом случае согласно операции (3.6) получим

$$\left[X_4^1\right] \cap [\Delta_2] \cap [\Delta_5] = [\overline{x}_2],\ X_4^1(\Delta_2, \Delta_5) = \overline{x}_2.$$

Далее для всех ветвей события S_5 имеем:

$$\left[X_5^1\right] \cap \left[X_5^2\right] \cap \left[X_5^3\right] \cap \left[X_5^4\right] = [x_2],\ X_5^{1,2,3,4} = x_2,$$

откуда

$$\left[X_4^1(\Delta_2, \Delta_5)\right] \cap \left[X_5^{1,2,3,4}\right] \neq \varnothing,\ X_4^1(\Delta_2, \Delta_5)\,\&\, X_5^{1,2,3,4} = 0,$$

поэтому $X_4^1 = 0$ и ветвь $S_4^1 = 0$;

б) ветвь $S_4^2(t+1) = \overline{x}_1 x_4 S_3$. Для этой ветви имеем

$$\left[X_4^2\right] \cap [\Delta_3] = [\overline{x}_1, x_4],\ X_4^2(\Delta_3) = \overline{x}_1 x_4.$$

Для события S_3^1 в п. 1,б получено $X_3^{1,2} = x_1$, откуда $\left[X_4^2(\Delta_3)\right] \cap \left[X_3^{1,2}\right] \neq \varnothing, X_4^2(\Delta_3)\,\&\, X_3^{1,2} = 0$, поэтому $X_4^2 = 0$ и $S_4^2 = 0$.

5. Событие S_5 состоит из четырех ветвей:

а) ветвь $S_5^1(t+1)=x_1x_2S_0$. Для этой ветви имеем $\left[X_5^1\right]\bigcap\left[\Delta_0\right]=\varnothing$. Поэтому запись этой ветви остается без изменения;

б) ветвь $S_5^2(t+1)=x_2x_4S_2$. Для этой ветви имеем $\left[X_5^2\right]\bigcap\left[\Delta_2\right]=$ $=\left[x_2,x_4\right], X_5^2(\Delta_2)=x_2x_4$.

Для события S_2, как и в п. 4,а, находим $\left[X_5^2\right]\bigcap\left[\Delta_2\right]\bigcap\left[\Delta_5\right]=$ $=\left[x_2\right], X_5^2(\Delta_2,\Delta_5)=x_2,$ откуда $\left[X_5^2(\Delta_2,\Delta_5)\right]\bigcap\left[X_5^{1,2,3,4}\right]\neq\varnothing,$ $X_5^2(\Delta_2,\Delta_5)\,\&\,X_5^{1,2,3,4}=x_2$, поэтому запись входного сигнала X_5^2 будет иметь вид $X_5^2=x_4$;

в) ветвь $S_5^3(t+1)=x_2x_4S_4$. Для этой ветви ранее было найдено $S_4=0$, поэтому $S_5^3=0$;

г) ветвь $S_5^4(t+1)=\overline{x}_1x_2x_3S_1$. Для этой ветви имеем $\left[X_5^4\right]\bigcap\left[\Delta_1\right]=$ $=\left[\overline{x}_1\right], X_5^4(\Delta_1)=\overline{x}_1$. Для события S_1 в п. 1,в было найдено, что $X_1^{1,2,3}=\overline{x}_1,$ откуда $\left[X_5^4(\Delta_1)\right]\bigcap\left[X_1^{1,2,3}\right]\neq\varnothing, X_5^4(\Delta_1)\,\&\,X_1^{1,2,3}=\overline{x}_1,$ поэтому новая запись входного сигнала для события S_5^4 будет иметь вид $X_5^4=x_2x_3$.

6. Событие S_6 будет состоять из трех ветвей, так как было ранее найдено, что $S_4=0$:

а) ветвь $S_6^1(t+1)=\overline{x}_4S_2$. Для этой ветви имеем: $\left[X_6^1\right]\bigcap\left[\Delta_2\right]=\left[\overline{x}_4\right],$ $X_6^1(\Delta_2)=\overline{x}_4$.

Дальнейший анализ выполняем, как в п. 4,а. Так как $\left[X_6^1\right]\bigcap\left[\Delta_2\right]\bigcap\left[\Delta_5\right]=\varnothing$, то запись этой ветви остается без изменения;

б) ветвь $S_6^2(t+1)=\overline{x}_2S_1$. Для этой ветви имеем $\left[X_6^2\right]\bigcap\left[\Delta_1\right]=\varnothing,$ поэтому запись этой ветви остается без изменения;

в) ветвь $S_6^3(t+1)=x_1S_3$. Для этой ветви имеем $\left[X_6^3\right]\bigcap\left[\Delta_3\right]=\left[x_1\right],$ $X_6^3(\Delta_3)=x_1$.

Для события S_3 в п. 1,б получено $X_3^{1,2} = x_1$, откуда $\left[X_6^3(\Delta_3)\right] \bigcap \left[X_3^{1,2}\right] \neq \varnothing$, $X_6^3(\Delta_3)\,\&\,X_3^{1,2} = x_1$, поэтому новая запись входного сигнала для события S_6^3 будет иметь вид $X_6^3 = 1$.

После произведенного анализа исходной СКУ с целью ее минимизации за счет учета распределения сдвигов ее новая запись, где введена замена $S_i = a_i$, будет иметь следующий вид:

$$
\begin{aligned}
&a_1(t+1) = \overline{x}_1 a_0 \vee x_2 \overline{x}_3 a_1;\\
&a_2(t+1) = a_5;\\
&a_3(t+1) = x_1 \overline{x}_2 a_0;\\
&a_4(t+1) = 0;\\
&a_5(t+1) = x_1 x_2 a_0 \vee x_4 a_2 \vee x_2 x_3 a_1;\\
&a_6(t+1) = \overline{x}_4 a_2 \vee \overline{x}_2 a_1 \vee a_3.
\end{aligned}
\qquad (3.11)
$$

Глава 4
НАЧАЛЬНЫЕ ЯЗЫКИ, ИСПОЛЬЗУЕМЫЕ ДЛЯ ПРЕДСТАВЛЕНИЯ УПРАВЛЯЮЩИХ АЛГОРИТМОВ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ИНФОРМАЦИИ

В качестве начальных языков для формального представления параллельных алгоритмических процессов и их отдельных ветвей в данной работе рассматриваются:

– язык регулярных выражений алгебры событий (РВАС);

– язык исчисления предикатов первого порядка с ограниченными кванторами и его связь с языком РВАС;

– язык операторных схем алгоритмов с параллельными ветвями (ОСАП), базирующийся на его графической интерпретации – языке граф-схем алгоритмов с параллельными ветвями (ГСАП). Кроме того, рассматриваются и обычные непараллельные языки – язык граф-схем алгоритмов (ГСА) и язык логических схем алгоритмов (ЛСА), так как они являются основой для построения параллельных языков ОСАП.

Для каждого начального языка, кроме основных понятий и определений, рассматривается методика его использования для представления алгоритмических процессов и методика преобразования описания алгоритмических процессов на начальном языке к описанию на стандартном языке в виде системы канонических уравнений для всех событий, реализуемых в алгоритме управления преобразованием информации.

4.1. Язык регулярных выражений алгебры событий

4.1.1.Основные понятия и определения языка регулярных выражений алгебры событий

Язык регулярных выражений алгебры событий (РВАС) нашел применение при построении различных управляющих и операционных устройств, например, для построения синтаксических анализаторов и устройств преобразования цифровой информации, для описания протоколов в компьютерных сетях и др.

Язык РВАС базируется на использовании алгебры событий, что позволило представлять бесконечные события, реализуемые в устройствах преобразования информации, конечными выражениями. В данном случае под преобразованием информации понимается процесс отображения множества слов в некотором конечном входном алфавите в множество слов в конечном выходном алфавите. Особенно удобен язык РВАС для решения задач распознавания цепочек слов и их принадлежности к тому или иному языку.

В связи с тем, что в соответствии с известными теоремами С. К. Клини [15] в конечном цифровом автомате представимы только регулярные события, язык РВАС имеет исключительно важное значение в теории и практике разработки способов описания законов функционирования цифровых автоматов на других начальных языках, когда решаются вопросы о представлении таких описаний в цифровых автоматах.

Язык РВАС и его использование для синтеза цифровых автоматов были впервые фундаментально представлены в монографии академика В. М. Глушкова «Синтез цифровых автоматов» [1]. Следуя [1], введем некоторые понятия и определения из теории языка РВАС.

Событием в любом конечном входном алфавите [Z] называют произвольное множество слов в этом алфавите.

Событие $S_{\alpha}^{W_{\beta}}$, имеющее номер α, будет представленным буквой W_{β} в любом заданном отображении φ с алфавитами [Z] и [W], если для каждого входного слова $p(t)$ этого события из алфавита [Z] соответствующее ему выходное слово $q(t)$ будет оканчиваться буквой W_{β}, входящей в алфавит [W], т.е. имеем $q(t) = \varphi[p(t)]$. Такому определению события не противоречит определение события, данное в гл. 2 данной книги.

Элементарными событиями в алфавите $Z = [z_1, z_2, ..., z_F]$ называются события, состоящие из одной буквы алфавита [Z], и событие $S = e$, где e – пустое слово.

Множество всех событий, каждое из которых отмечено одним из выходных сигналов алфавита [W], называют каноническим множеством событий, соответствующих отображению φ.

Для всякого автоматного отображения φ определяемое им множество событий должно удовлетворять условиям автоматности, которые формулируются таким образом:

1) события, входящие в автоматное множество событий, попарно не пересекаются и не содержат пустого слова (условие однозначности);

2) начальные отрезки слов, принадлежащие какому-нибудь событию из автоматного множества событий, за исключением пустого слова, принадлежат каким-либо событиям из того же самого множества (условие полноты).

Под ***алгеброй событий*** в алфавите [Z] понимается множество всех событий в этом алфавите, на котором определены две двухместные операции – дизъюнкция и умножение и одна одноместная операция, называемая итерацией.

Дизъюнкция двух событий $S_1 \vee S_2$ – это теоретико-множественное объединение этих событий.

Произведение (или конкатенация) двух событий S_1 S_2 – это событие, состоящее из всех слов вида k_1k_2, где k_1 – любое слово из S_1, а k_2 – любое слово события S_2. При этом $S_1S_2 \neq S_2S_1$, т.е. эта операция отличается от операции логического умножения. В дальнейшем, если такая операция встречается в каком-либо выражении одновременно с операцией логического умножения, последнюю операцию будем обозначать символом &.

Итерация события S, записываемая в виде $\{S\}$, – это событие, состоящее из пустого слова e и всевозможных слов вида $p_1, p_2, \cdots, p_n$, где $p_1, p_2, \cdots, p_n$ – произвольные слова события S, а n – любое натуральное число $n = 1, 2, \ldots$

$$\{S\} = e \vee S \vee SS \vee SSS \vee \ldots .$$

В алгебре событий при отсутствии в выражении обычных скобок, изменяющих обычный порядок действий, сначала должны выполняться итерации, затем конкатенация и потом дизъюнкция.

Регулярное событие – это событие, которое можно получить из элементарных событий в результате применения конечного числа раз трех основных операций алгебры событий. Необходимо иметь в виду, что не все события являются регулярными. Например, не является регулярным событием множество всех последовательностей $z_1z_1 \cdots z_1z_2$, таких, что их длина выражается простым числом [44].

К числу не основных операций в алгебре событий относят операцию дополнения события и двуместную операцию пересечения событий.

Дополнением $\overline{S}$ события S в некотором алфавите $[Z]$ называется множество всех слов в алфавите $[Z]$, которые не вошли в событие S.

Пересечением S_1 & S_2 событий S_1 и S_2 называется событие, состоящее из всех слов, входящих одновременно в оба события S_1 и S_2.

Регулярное выражение алгебры событий – это всякое представление регулярного события через элементарные события и указанные операции алгебры событий, включая и не основные операции.

Так как одно и то же регулярное выражение допускает много различных представлений, то в ряде случаев может возникнуть проблема эквивалентных преобразований регулярных выражений. Такие преобразования могут быть выполнены с помощью следующих тождественных соотношений [1]:

1. $P \vee Q = Q \vee P$ – коммутативность дизъюнкции;
2. $P \vee (Q \vee S) = (P \vee Q) \vee S$ – ассоциативность дизъюнкции;
3. $P \vee P \vee P \vee ... \vee P = P$ – идемпотентность дизъюнкции;
4. $P\ (Q\ S) = (P\ Q)\ S$ – ассоциативность умножения;
5. $P(Q \vee S) = PQ \vee PS$ – левая и правая дистрибутивность, $(P \vee Q)S = PS \vee Q\,S$ – умножения по отношению к дизъюнкции;
6. $\{\{P\}\} = \{P\}$ – идемпотентность итерации;
7. $\{P\} = e \vee P\{P\}$ – правило развертывания итерации;
8. $P\ \{P\} = \{P\}\ P$ – закон коммутативности для итерации;
9. $\{P\}\{P\} = \{P\}$ – закон мультипликативного поглощения для итерации;
10. $\{P\} \vee P = \{P\}$ – закон дизъюнктивного поглощения для итерации;
11. $eS = Se = S$ – закон нейтральности пустого слова;
12. $\{e\} = e$.

В этих соотношениях P, Q, S – обозначают произвольные события, а e – событие, состоящее из одного пустого слова.

Приведем некоторые дополнительные тождественные соотношения алгебры событий, полученные в [30]:

1. $\{\{P\}\{Q\}\} = \{P \vee Q\}$;
2. $\{P \vee \{Q\}\{S\}\} = \{P \vee Q \vee S\}$;
3. $\{P \vee Q\} \vee Z\left(P,Q\right) = \{P \vee Q\}$;

где $Z(P,Q)$ – регулярное выражение, построенное из P и Q с помощью трех основных операций алгебры событий;

4. $\{P \vee \{Q\}\} = \{P \vee Q\}$;

5. $\{P \vee Q \vee Z(P,Q)\} = \{P \vee Q\}$;

6. $\{P \vee Q\}\{Z(P,Q)\} = \{P \vee Q\}$.

Можно доказать следующее важное соотношение, позволяющее раскрывать итерационные скобки:

если $P = S\{Q\}$, то $P = S \vee PQ$. (4.1)

Действительно, используя тождество $\{Q\} = e \vee Q\{Q\}$ и закон нейтральности пустого слова, получим

$$P = S(e \vee Q\{Q\}) = Se \vee SQ\{Q\} = S \vee SQ\{Q\}.$$

Применяя правило $Q\{Q\} = \{Q\}Q$, будем иметь: $P = S \vee S\{Q\}Q$, где заменяя $S\{Q\} = P$, получим искомое соотношение (4.1). С другой стороны, если $e \notin Q$, то уравнение $P = QP \vee S$ имеет решение $P = S\{Q\}$.

Здесь уместно еще раз обратить внимание на фундаментальное значение для теории конечных цифровых автоматов известных теорем С. К. Клини [13, 15], в которых было впервые доказано, что класс событий, представимых в конечных цифровых автоматах, совпадает с классом регулярных событий. Это означает, что:

1) событие, представленное в произвольном конечном автомате Мили или Мура некоторым множеством выходных сигналов или состояний, обязательно регулярно;

2) любое регулярное событие может быть представлено в конечном автомате Мили или Мура некоторым множеством выходных сигналов или состояний.

Приведем примеры регулярных выражений для входного алфавита $[Z] = [z_1 z_1, \cdots, z_F]$ некоторых характерных событий, которые могут быть полезными для описания законов функционирования автоматов [31]:

1. Всеобщее или универсальное событие – это событие, состоящее из всех возможных слов входного алфавита [Z]:

$$I = \{z_1 \vee z_2 \vee \cdots \vee z_F\}. \quad (4.2)$$

2. Событие, включающее все возможные слова, состоящие из букв $z_i,...,z_j$:

$$S = \{z_i \vee ... \vee z_j\}, \tag{4.3}$$

где $\left(z_i,...,z_j\right) \in \left[Z\right]$.

3. Событие, содержащее все слова, оканчивающиеся отрезком p. Например, для $p = z_i z_j$ имеем:

$$S = \{z_1 \vee z_2 \vee ... \vee z_i \vee z_j \vee ... \vee z_F\} z_i z_j = I z_i z_j . \tag{4.4}$$

4. Событие, содержащее все слова, в которых хотя бы один раз встречается отрезок слова p в любом месте. Например, для $p = z_i z_j$ имеем

$$S = IpI = Iz_i z_j I. \tag{4.5}$$

5. Событие, состоящее из всех слов, имеющих начальные и конечные отрезки p_1 и p_2, соответственно. Например, для $p_1 = z_i$, $p_2 = z_j z_k$ имеем

$$S = p_1 I p_2 = z_i I z_j z_k . \tag{4.6}$$

6. Событие, содержащее все слова длины r:

$$S = S_1 S_2 ... S_j ... S_r , \tag{4.7}$$

где $S_j = z_1 \vee z_2 \vee ... \vee z_F , j = \overline{1,r}$.

7. Событие, содержащее все слова длиной, кратной r:

$$S = \{S_1 S_2 ... S_j ... S_r\}, \tag{4.8}$$

где $S_j = z_1 \vee z_2 \vee ... \vee z_F , j = \overline{1,r}$.

8. Событие, состоящее из всех слов алфавита $Z = \left[z_1, z_2\right]$, не содержащих серии из r букв z_1 и оканчивающихся буквой z_2:

$$S = \left\{ z_2 \vee z_1 z_2 \vee z_1 z_1 z_2 \vee ... \vee \underbrace{z_1 z_1 \cdots z_1}_{(r-1)\ \text{букв}} z_2 \right\}. \tag{4.9}$$

Как было отмечено выше (см. гл. 1), для многих практических реальных автоматов в качестве входного (выходного) алфавитов целесообразно использовать множество элементарных

входных (выходных) сигналов. В этом случае при формализации закона функционирования автомата на языке регулярных выражений алгебры событий под входной буквой будем понимать также частный входной сигнал, который равен конъюнкции (комбинации) тех элементарных входных сигналов, которые действуют в данный момент автоматного времени одновременно. Такую конъюнкцию будем заключать в круглые скобки. Например, для события, записанного в виде выражения

$$S_{\alpha}^{W_{\beta}} = S_0(x_1x_2)\{(x_2\overline{x}_3x_4)\}(\overline{x}_4),$$

элементарные конъюнкции (x_1x_2), $(x_2\overline{x}_3x_4)$, $(\overline{x}_4)$ представляют собой частные входные сигналы, действующие на входе автомата в соответствующие такты его работы, и могут быть обозначены одной буквой, например: $\chi_i = x_1x_2$; $\chi_j = x_1\overline{x}_3x_4$; $\chi_k = \overline{x}_4$.

Приведем пример составления регулярного выражения, определяющего закон функционирования цифрового автомата.

П р и м е р 4.1. Пусть необходимо описать автомат, выдающий сигнал W_k в течение одного такта всякий раз, когда происходит изменение входной буквы с z_1 на z_2. Формулируя это условие по-другому, можно сказать, что сигнал W_k должен выдаваться в ответ на любые входные последовательности, кончающиеся последовательностью z_1z_2. Фраза «любые входные последовательности» формализуется так называемым всеобщим или универсальным событием, состоящим из всех возможных слов в алфавите $Z = [z_1, z_2]$. Тогда событие, в ответ на которое должен выдаваться сигнал W_k, на основании (4.2) и (4.4) будет описываться регулярным выражением $S^{W_k} = \{z_1 \vee z_2\} z_1z_2$.

4.1.2. Построение СКУ по регулярному выражению алгебры событий и ее детерминизация

Предлагаемый способ построения СКУ по регулярному выражению алгебры событий основывается на следующих предложениях [32, 33]. Любое регулярное выражение алгебры событий, представляющее в цифровом автомате событие $S_{\alpha}^{W_{\beta}}$, в общем случае может быть записано в виде

$$S_{\alpha}^{W_{\beta}} = \bigvee_{i=1}^{k} S_{\alpha}^{i}, \tag{4.10}$$

где α – номер события; k – число параллельных ветвей алгоритма, определяющих событие S_{α} (число дизъюнктивных членов регулярного выражения); S_{α}^{i} – обозначение события, описывающего i-ю ветвь алгоритма.

Событие S_{α}^{i}, в свою очередь, может быть сведено к произведению элементарных одноэлементных событий. Действительно, если произвести замену итерации в описании события S_{α}^{i} там, где она есть, то получим

$$S_{\alpha}^{i} = S_0 \underbrace{z_{\varepsilon} \cdots z_p \cdots z_r z_{\gamma}}_{n \text{ букв}}. \tag{4.11}$$

Исходя из интерпретации регулярного выражения алгебры событий, выражение (4.11) может быть представлено в виде следующей системы регулярных выражений:

$$\begin{gathered} S_{\alpha,0}^{i} = S_{\alpha,1}^{i} z_{\gamma}, \\ S_{\alpha,1}^{i} = S_{\alpha,2}^{i} z_{r}, \\ \cdots \\ S_{\alpha,\beta}^{i} = S_{\alpha,\beta+1}^{i} z_{p}, \\ \cdots \\ S_{\alpha,n-1}^{i} = S_0 z_{\varepsilon}, \end{gathered} \tag{4.12}$$

где β – порядковый номер буквы z_p, начиная с конца выражения (4.11) (β называется рангом события и используется в качестве второго нижнего индекса); S_0 – обозначение начального события, равного $\{e\}$; поскольку $\{e\} = e$ и $eP = Pe = P$ (закон нейтральности пустого слова), то событие S_0 может быть всегда добавлено в регулярное выражение; $S_{\alpha,\beta+1}^{i}$ – обозначение события, непосредственно предшествующего событию $S_{\alpha,\beta}^{i}$; n – число входных букв в выражении (4.11).

Если в выражении (4.11) была произведена замена вида

$$z_p = \{R\},$$

где R – произвольное регулярное выражение, то, используя тождество (4.1), получим:

$$S^i_{\alpha,\beta} = S^i_{\alpha,\beta+1}\{R\} = S^i_{\alpha,\beta+1} \vee S^i_{\alpha,\beta} R. \qquad (4.13)$$

Применяя рассмотренные операции к выражению (4.13), можно также свести его к выражениям типа (4.10), (4.11) и (4.12).

Исходя из изложенного, алгоритм построения СКУ по заданным регулярным выражениям исходной системы событий будет содержать следующие этапы:

1. В каждом из регулярных выражений исходной системы событий $S_1, S_2, ..., S_\alpha, ..., S_M$, отмеченных соответствующими выходными сигналами, выделяют параллельные ветви, описывающие события $S_\alpha \left(\alpha = \overline{1, M}\right)$.

2. Для каждой параллельной ветви $S^i_\alpha \left(i = \overline{1, k}\right)$ каждого из регулярных выражений исходной системы событий выполняют операцию замены итерационных скобок (внешних) там, где они имеются, одной из вспомогательных переменных, которые будут играть роль дополнительных входных букв. Например, $\left\{R^i_{\alpha,p}\right\} = z^i_{\alpha,p}$, где $R^i_{\alpha,p}$ – регулярное выражение, входящее в итерационные скобки i-й ветви события S_α, p – порядковый номер итерационных скобок в i-й ветви.

3. Каждое событие S^i_α преобразуют в систему рекуррентных регулярных соотношений вида (4.12).

4. Выполняют подстановку вместо буквы типа $z^i_{\alpha,p}$ соответствующих итерационных скобок $\left\{R^i_{\alpha,p}\right\}$ и осуществляют их раскрытие в соответствии с тождеством (4.1).

5. Если содержимое итерационных скобок какого-либо частного события из полученной системы соотношений вида (4.12) включает более одной буквы, в том числе и итерационные скобки (внутренние), то для этого события повторяют все операции, предусмотренные пп. 1–5 и т.д. до тех пор, пока не будут раскры-

ты все итерационные скобки в регулярных выражениях всех событий, входящих в исходную систему.

П р и м е ч а н и е. Для уменьшения числа итерационных скобок целесообразно использовать дополнительные тождественные соотношения алгебры событий.

6. Для каждого события S_α исходной системы выполняют подстановку вместо обозначения событий S_α^i соответствующих им выражений.

Аналогичную подстановку делают и в том случае, если в процессе преобразований описания некоторых частных событий будут содержать обозначения непосредственно предшествующих событий без входных букв. В этом случае необходимо сделать подстановку вместо обозначения таких событий соответствующих им выражений.

7. Во всех описаниях полученных частных событий заменяют знаки конкатенации на знаки конъюнкции и, вводя дискретное время, как принято в (2.3), получают искомую СКУ, в которой обозначения событий исходной системы отмечаются соответствующими выходными сигналами.

П р и м е ч а н и е. Принятый способ нумерации событий и букв удобен при выполнении всех приведенных выкладок и не является обязательным. При небольшом числе событий может оказаться более удобным нумеровать события с использованием одного индекса.

П р и м е р 4.2. Рассмотрим без дополнительных пояснений пример построения СКУ по следующему регулярному выражению:

$$S_\alpha^W = S_0 z_2 \{z_1 \vee z_2\} z_2 z_1 \vee S_0 \{z_1 z_2\} = S_{\alpha,1}^W \vee S_{\alpha,2}^W,$$

$$S_{\alpha,1}^W = S_0 z_2 \{z_1 \vee z_2\} z_2 z_1 = S_3 z_1,$$

$$S_3 = S_0 z_2 \{z_1 \vee z_2\} z_2 = S_4 z_2,$$

$$S_4 = S_0 z_2 \{z_1 \vee z_2\} = S_0 z_2 \vee S_4 (z_1 \vee z_2),$$

$$S_{\alpha,2}^W = S_0 \{z_1 z_2\} z_2 = S_5 z_2,$$

$$S_5 = S_0 \{z_1 z_2\} = S_0 \vee S_5 z_1 z_2 = S_0 \vee S_6 z_2,\ S_0 \Rightarrow S_5,$$

$$S_6 = S_5 z_1,$$

где $\Rightarrow$ – знак выводимости.

Окончательно СКУ будет иметь вид

$$
\begin{aligned}
&S_{\alpha}^{W}(t+1) = S_3(t)\,\&\,z_1(t) \vee S_5(t)\,\&\,z_2(t),\\
&S_3(t+1) = S_4(t)\,\&\,z_2(t),\\
&S_4(t+1) = S_0(t)\,\&\,z_2(t) \vee S_4(t)\,\&\left[z_1(t) \vee z_2(t)\right],\\
&S_5(t+1) = S_0(t) \vee S_6(t)\,\&\,z_2(t),\\
&S_6(t+1) = S_5(t)\,\&\,z_1(t).
\end{aligned}
\qquad (4.14)
$$

В общем случае СКУ, полученная по регулярным выражениям алгебры событий, не детерминирована. Это относится не только к СКУ, полученной по системе регулярных выражений с общими начальными событиями, но и к СКУ, полученной по одному регулярному выражению, включающему только одну ветвь с итерационной скобкой. В связи с этим для таких СКУ в целях дальнейшего синтеза автомата может потребоваться их детерминизация.

Если в качестве входных сигналов в исходных регулярных выражениях алгебры событий используются частные входные сигналы из структурного алфавита, то детерминизацию СКУ необходимо проводить в соответствии с алгоритмом, предложенным в гл. 2.

Если в качестве входных сигналов в исходных регулярных выражениях алгебры событий используются, как обычно принято, полные входные сигналы из абстрактного алфавита [Z], то алгоритм детерминизации упрощается, так как на входе автомата в любой момент автоматного времени может появиться только одна из букв входного алфавита [Z], но не их сочетание.

В связи с отмеченными замечаниями при построении прямой таблицы переходов детерминированного автомата для каждой строки таблицы в столбце 3 выписываются все входные сигналы из абстрактного алфавита [Z], а для обычной таблицы переходов детерминированного автомата каждая строка должна отмечаться своим абстрактным входным сигналом, а число строк таблицы определится числом всех входных сигналов.

Для упрощения процесса детерминизации при построении таблиц переходов автомата также целесообразно представить исходную СКУ в виде вспомогательной прямой таблицы переходов для событий.

При этом следует иметь в виду, что при построении по СКУ вспомогательной прямой таблицы переходов (НД ПТП) и таблицы переходов (ТП) детерминированного автомата первый столбец в первой строке таблиц должен обозначаться следующим образом:

а) если S_0 входит в качестве предшествующего события в какое-либо из уравнений СКУ и из S_0 не выводимо ни одно из событий СКУ, то $a_0 = S_0$;

б) если S_0 входит в качестве предшествующего события в какое-либо из уравнений СКУ и из S_0 выводимы события S_i, …, S_j, то $a_0 = S_0 \,\&\, S_i \,\&\ldots\&\, S_j$;

в) если S_0 не входит в качестве предшествующего события ни в одно из уравнений СКУ и из S_0 выводимы события S_i, …, S_j, то $a_0 = S_i \,\&\ldots\&\, S_j$.

П р и м е р 4.3. Выполним детерминизацию СКУ, полученной в предыдущем примере, путем построения обычной таблицы переходов детерминированного автомата.

По СКУ (4.14) строим вспомогательную прямую таблицу переходов для частных событий (табл. 4.1), в которой в столбец 3 каждой строки записываются те входные сигналы, которые имеют место в исходной СКУ на рассматриваемом переходе.

Таблица 4.1

1	2	3	4
1	$S_0 \,\&\, S_5$	z_1 z_2	S_6 $S_4 \,\&\, S_\alpha$
2	S_6	z_2	S_5
3	S_4	z_1 z_2	S_4 $S_3 \,\&\, S_4$
4	S_5	z_1 z_2	S_6 S_α
5	S_3	z_1	S_α
6	S_α	z_1 z_2	* *

В соответствии с общими положениями алгоритма детерминизации, а также замечаниями отмеченными выше, и на основании табл. 4.1, таблица переходов детерминированного автомата Мура будет иметь вид (табл. 4.2).

Таблица 4.2

w	e	e	w	e	e	e	w	e
a / z	S_0 & S_6	S_6	S_4 & S_α	S_5	S_4	S_4 & S_3	S_α	*
z_1	S_6	*	S_4	S_6	S_4	S_4 & S_α	*	*
z_2	S_4 & S_α	S_5	S_4 & S_3	S_α	S_4 & S_3	S_4 & S_3	*	*

П р и м е р 4.4. Предположим, что необходимо описать множество всех последовательностей из нулей и единиц, которые имеют три подряд стоящие единицы, но не кончающиеся на 01 и не состоящие лишь из единиц. По полученному описанию построить отмеченную таблицу переходов (ТП) цифрового автомата Мура.

В данном примере рассматривается представление двоичных слов в виде событий на языке РВАС с использованием дополнительных операций алгебры событий с построением СКУ и ТП ЦА Мура, реализующего представляющее событие.

Решение:

1) закодируем исходные двоичные цифры следующим образом:

$$0 = x_0,\ 1 = x_1;$$

2) множество всех последовательностей, которые имеют подряд три единицы, можно записать так:

$$\{x_0 \vee x_1\} x_1 x_1 x_1 \{x_0 \vee x_1\};$$

3) множество всех последовательностей, кончающееся на 01, запишется таким образом:

$$\{x_0 \vee x_1\} x_0 x_1;$$

4) множество всех последовательностей, состоящее лишь из единиц, имеет вид:

$$x_1 \{x_1\};$$

5) таким образом все искомое множество двоичных последовательностей можно представить в виде следующего события:

$$S_\alpha^y = \{x_0 \vee x_1\} x_1 x_1 x_1 \{x_0 \vee x_1\} \,\&\, \left(\overline{\{x_0 \vee x_1\} x_0 x_1 \vee x_1 \{x_1\}}\right) = S_1 \,\&\, \overline{S}_2;$$

6) на основе рассмотренного алгоритма преобразования РВАС описание события S_α^y преобразуется в следующую СКУ:

$$S_1(t+1) = S_3(t) \,\&\, x_1(t) \vee S_1(t) \,\&\, \left[x_0(t) \vee x_1(t) \right],$$
$$S_3(t+1) = S_4(t) \,\&\, x_1(t),$$
$$S_4(t+1) = S_5(t) \,\&\, x_1(t),$$
$$S_5(t+1) = S_0(t) \vee S_5(t) \,\&\, \left[x_0(t) \vee x_1(t) \right],$$
$$S_2(t+1) = S_0(t) \,\&\, x_1(t) \vee S_6(t) \,\&\, x_1(t) \vee S_7(t) \,\&\, x_1(t),$$
$$S_6(t+1) = S_5(t) \,\&\, x_0(t),$$
$$S_7(t+1) = S_0(t) \,\&\, x_1(t) \vee S_7(t) \,\&\, x_1(t);$$

7) для построения отмеченной таблицы переходов ЦА Мура по СКУ строится вспомогательная НД ПТП ЦА Мура, в которой первый столбец отмечается совокупностью событий S_0 и S_5, так как событие S_5 выводимо из события S_0, так как имеем $S_0 \Rightarrow S_5$ (табл. 4.3);

Таблица 4.3

1	2	3	4
1	S_0 & S_5	x_0 x_1	S_5 & S_6 S_5 & S_7 & S_2 & S_4
2	S_5	x_0 x_1	S_5 & S_6 S_5 & S_4
3	S_6	x_1	S_2
4	S_7	x_1	S_2 & S_7
5	S_4	x_1	S_3
6	S_3	x_1	S_1
7	S_1	x_0 x_1	S_1 S_1

8) искомая отмеченная детерминированная ТП ЦА Мура, построенная на основе НД ПТП, будет иметь следующий вид (табл. 4.4).

Таблица 4.4

w	e	e	e	e
a / z	S_0 & S_5	S_5 & S_6	S_5 & S_7 & S_4 & S_2	S_5 & S_4 & S_2
z_1	S_5 & S_6	S_5 & S_6	S_5 & S_6	S_5 & S_6
z_2	S_5 & S_4 & S_7 & S_2	S_5 & S_4 & S_2	S_5 & S_4 & S_7 & S_2 & S_3	S_5 & S_4 & S_3

Окончание табл. 4.4

w	e	e	e
z \ a	$S_5 \,\&\, S_4 \,\&\, S_7 \,\&\, S_2 \,\&\, S_3$	$S_5 \,\&\, S_4 \,\&\, S_3$	$S_5 \,\&\, S_4 \,\&\, S_3 \,\&\, S_2 \,\&\, S_7 \,\&\, S_1$
z_1	$S_5 \,\&\, S_6$	$S_5 \,\&\, S_6$	$S_5 \,\&\, S_6 \,\&\, S_1$
z_2	$S_5 \,\&\, S_4 \,\&\, S_3 \,\&\, S_2 \,\&\, S_7 \,\&\, S_1$	$S_5 \,\&\, S_4 \,\&\, S_3 \,\&\, S_1$	$S_5 \,\&\, S_4 \,\&\, S_3 \,\&\, S_1 \,\&\, S_2 \,\&\, S_7$
w	y	y	e
z \ a	$S_5 \,\&\, S_4 \,\&\, S_3 \,\&\, S_1$	$S_5 \,\&\, S_6 \,\&\, S_1$	$S_5 \,\&\, S_4 \,\&\, S_2 \,\&\, S_1$
z_1	$S_5 \,\&\, S_6 \,\&\, S_1$	$S_5 \,\&\, S_6 \,\&\, S_1$	$S_5 \,\&\, S_6 \,\&\, S_1$
z_2	$S_5 \,\&\, S_4 \,\&\, S_2$	$S_5 \,\&\, S_4 \,\&\, S_2 \,\&\, S_1$	$S_5 \,\&\, S_4 \,\&\, S_3 \,\&\, S_1$

Цифровой автомат, построенный в соответствии с полученной ТП ЦА Мура, при подаче на вход последовательности двоичных цифр, представленных событием S_α^y, будет выдавать выходной сигнал y = 1; для всех других последовательностей, не соответствующих событию S_α, ЦА будет выдавать пустой сигнал.

4.2. Язык исчисления предикатов

Как следует из [29, 34], язык исчисления предикатов имеет ряд преимуществ по сравнению с другими начальными языками, в том числе и с языком РВАС.

Во-первых, язык исчисления предикатов естественно описывает словесные алгоритмы функционирования устройств цифровой вычислительной техники.

Во-вторых, язык исчисления предикатов значительно шире, т.е. выразительнее языка РВАС.

В-третьих, алгоритмические проблемы проектируемых цифровых устройств могут формулироваться и решаться на том же языке, на котором решается проблема синтеза абстрактного автомата.

В связи с тем, что любой новый предлагаемый для использования начальный язык требует доказательств представимости в цифровом автомате событий, описываемых на этом языке, то в начале этого раздела будет показано, что все регулярные выражения алгебры событий, в том числе элементарное одноэлементное событие и событие с итерацией сложного регулярного выражения, могут быть описаны вполне определенным частным видом фор-

мул исчисления предикатов, не имеющих кванторов по функциональной переменной, т.е. частным видом формул исчисления предикатов первого порядка с ограниченными кванторами [34].

Полученный частный вид формул исчисления предикатов первого порядка, не имеющих кванторов по функциональной переменной, будет принят в качестве основы для описания событий, реализуемых в алгоритме управления преобразованием информации [22].

4.2.1. Описание регулярных выражений алгебры событий рекурсивными предикатами

При описании событий в алфавите $[Z]=[z_1, z_2, ..., z_F]$ будем учитывать два условия:

1. Всякий абстрактный автомат получает входные сигналы во все моменты времени t = 1, 2, 3, … . Если же реальный автомат в какие-то моменты времени не получает сигналов из алфавита $[Z]$, то для описания такого случая будем использовать пустой входной сигнал. В исчислении предикатов пустой входной сигнал (пустое слово) $e(\tau)$ может быть описан предикатной формулой

$$e(\tau) = \overline{z_1(\tau) \vee z_2(\tau) \vee ... \vee z_F(\tau)}. \qquad (4.15)$$

2. На вход абстрактного автомата в любой момент времени может поступать только один сигнал z_i. Это условие в исчислении предикатов может быть описано предикатной формулой $\chi_i(\tau)$, определяемой выражением

$$\chi_i(\tau) = z_i(\tau) \,\&\, e_{z_i}(\tau), \qquad (4.16)$$

где $e_{z_i}(\tau)$ есть результат подстановки нуля вместо предиката $z_i(\tau)$ в формулу для пустой буквы $e(\tau)$. Использовав обозначение подстановки, принятое в [35], получим

$$e_{z_i}(\tau) = S_0^{z_i(\tau)} e(\tau). \qquad (4.17)$$

Описание события S_0, заданное в алгебре событий пустым словом, формулами исчисления предикатов.

Событие S_0 на основании закона нейтральности пустого слова в алгебре событий можно записать так:

$$S_0 = \{e\}. \qquad (4.18)$$

Выполнив описание выражения (4.18) в исчислении предикатов, получим:

$$S_0(t) = (\forall\tau)\left[\tau \le t \to e(\tau)\right]. \qquad (4.19)$$

Выражение (4.19) определяет предикат $S_0(t)$ для всех значений вещественной переменной τ из области натуральных чисел, удовлетворяющих неравенству $0 < \tau \le t$.

Вводя сокращенное обозначение ограниченных кванторов [34], окончательно получим:

$$S_0(t) = \underset{\tau \le t}{(\forall\tau)} e(\tau). \qquad (4.20)$$

Описание элементарного одноэлементного события $S = z_i$.

Учитывая закон нейтральности пустого слова, исходное выражение можно переписать так:

$$S = \{e\}\, z_i\, \{e\}. \qquad (4.21)$$

Выражение (4.21) в исчислении предикатов, с учетом 2-го условия, может быть описано следующим образом:

$$S(t) = (\exists\tau)[\tau \le t \,\&\, z_i(t) \,\&\, e_{z_i}(\tau) \,\&\, (\forall\tau_1)\left[\tau < \tau_1 \le t \to e(\tau_1)\right] \,\&$$
$$\&\,(\forall\delta)\left[\delta < \tau \to e(\delta)\right]]. \qquad (4.22)$$

Легко видеть, что полученная предикатная формула является утверждением того, что для $\tau \le t$ существует единственное τ, такое, что $z_i(\tau)$ [29], и утверждением того, что для всех τ, удовлетворяющих неравенству $\tau \le t$, $e_{z_i}(\tau)$.

Использовав сокращенное обозначение ограниченных кванторов и выражение (4.16), перепишем (4.22) так:

$$S(t) = \underset{\tau \le t}{(\exists\tau)}\chi_i(\tau) \,\&\, \underset{\tau < \tau_1 \le t}{(\forall\tau_1)} e(\tau_1) \,\&\, \underset{\delta < \tau}{(\forall\delta)} e(\delta), \qquad (4.23)$$

или, использовав дополнительно (4.20) и учтя, что неравенство $\delta < \tau$ эквивалентно неравенству $\delta \le \tau - 1$, получим:

$$S(t) = \underset{\tau \le t}{(\exists\tau)}\chi_i(\tau) \,\&\, \underset{\tau < \tau_1 < t}{(\forall\tau_1)} e(\tau_1) \,\&\, S_0(\tau - 1). \qquad (4.24)$$

Из выражения (4.24) следует, что описание события S состоит из трех частей:

1) описание события в момент его наступления;

2) описание условия, при котором событие *S* сохраняется;

3) описание события, непосредственно предшествующего событию *S*.

Описание события *S*, заданного дизъюнкцией элементарных одноэлементных событий $S = z_i \vee z_j$**.**

На основании закона нейтральности пустого слова перепишем исходное выражение в следующей эквивалентной форме:

$$S = \{e\} z_i \{e\} \vee \{e\} z_j \{e\}. \tag{4.25}$$

Поступив так же, как и в предыдущем случае, получим:

$$S(t) = (\underset{\tau \le t}{\exists \tau}) \chi_i(\tau) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall \tau_1}) e(\tau_1) \,\&\, S_0(\tau - 1) \vee$$

$$\vee (\underset{\tau \le t}{\exists \tau}) \chi_i(\tau) \,\&\, (\underset{\delta < \delta_1 < t}{\forall \tau_1}) e(\delta_1) \,\&\, S_0(t - 1). \tag{4.26}$$

Выполнив в полученном выражении переименование связанных переменных [29, 39] и производя необходимые преобразования, можно выражению (4.26) придать следующий вид:

$$S(t) = (\underset{\tau \le t}{\exists \tau}) [\chi_i(\tau) \vee \chi_j(\tau)] \,\&\, (\underset{\tau < \tau_1 \le t}{\forall \tau_1}) e(\tau_1) \,\&\, S_0(\tau - 1). \tag{4.27}$$

Описание события *S*, заданного регулярным выражением алгебры событий $z_i z_j$, т.е.

$$S = z_i z_j,$$

которое на основании нейтральности пустого слова в алгебре событий можно переписать так:

$$S = \{e\} z_i \{e\} z_j \{e\}. \tag{4.28}$$

Выполнив описание выражения (4.28) в исчислении предикатов, получим:

$$S(t) = (\underset{\tau \le t}{\exists \tau}) [\chi_j(\tau) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall \tau_1}) e(\tau_1) \,\&\, (\underset{\delta < \tau}{\exists \delta}) [\chi_i(\delta) \,\&$$

$$\&\, (\underset{\delta < \delta_1 < \tau}{\forall \delta_1}) e(\delta_1) \,\&\, (\underset{\sigma < \delta}{\forall \sigma}) e(\sigma)]]. \tag{4.29}$$

Если не пользоваться скобками, определяющими область действия кванторов существования, считая при этом, что действие квантора распространяется на всю конъюнкцию предикатов, рас-

положенных правее квантора, и учесть (4.20), то выражение (4.29) примет вид

$$S(t) = (\underset{\tau \le t}{\exists \tau})\chi_i(\tau)\,\&\,(\underset{\tau < \tau_1 \le t}{\forall \tau_1})e(\tau_1)\,\&\,(\underset{\delta < \tau}{\exists \delta})\chi_i(\delta)\,\&$$

$$\&(\underset{\delta < \delta_1 < \tau}{\forall \delta_1})e(\delta_1)\,\&\,S_0(\delta - 1). \qquad (4.30)$$

Обозначив на основании (4.24) связную группу предикатов

$$(\underset{\delta < \tau}{\exists \delta})\chi_i(\delta)\,\&\,(\underset{\delta < \delta_1 < \tau}{\forall \delta_1})e(\delta_1)\,\&\,S_0(\delta - 1)$$

через $S_1(\tau - 1)$, получим следующую форму записи выражения (4.30):

$$S(t) = (\underset{\tau \le t}{\exists \tau})\chi_i(\tau)\,\&\,(\underset{\tau < \tau_1 \le t}{\forall \tau_1})e(\tau_1)\,\&\,S_1(\tau - 1). \qquad (4.31)$$

Из (4.31) следует, что описание выражения $S = z_i z_j$ так же, как и описание выражения $S = z_i$, состоит из аналогичных трех частей.

Описание события *S*, заданного регулярным выражением алгебры событий $\{z_i\}$, т.е.

$$S = \{z_i\}.$$

На основании правила развертывания итерации, закона коммутативности для итерации и закона нейтральности пустого слова исходное выражение в алгебре событий может быть переписано в следующей эквивалентной форме:

$$S = \{e\} \vee \{z_i\} z_i \{e\}. \qquad (4.32)$$

Учитывая исходную формулу описываемого выражения, можно (4.32) переписать и так:

$$S = \{e\} \vee S z_i \{e\}. \qquad (4.33)$$

Выполнив описание выражения (4.33) формулами исчисления предикатов, получим:

$$S(t) = S_0(t) \vee (\underset{\tau \le t}{\exists \tau})\chi_i(\tau)\,\&\,(\underset{\tau < \tau_1 \le t}{\forall \tau_1})e(\tau_1)\,\&\,S(\tau - 1). \qquad (4.34)$$

Описание дополнения $\overline{S}$ события *S*.

Дополнение $\overline{S}$ события S в исчислении предикатов описывается обычным образом на основании правил исчисления предикатов. Так, если

$$S(t) = (\underset{\tau \le t}{\exists \tau})\chi_j(\tau) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall \tau_1})e(\tau) \,\&\, S_1(\tau - 1),$$

то

$$\overline{S(t)} = (\underset{\tau \le t}{\forall \tau})[\overline{\chi_j}(\tau) \vee (\underset{\tau < \tau_1 \le t}{\exists \tau_1})\overline{e(\tau_1)} \vee \overline{S_1(\tau - 1)}]. \qquad (4.35)$$

Описание регулярных выражений алгебры событий произвольной формы формулами исчисления предикатов.

В разделе 4.1.2 было показано, что любое РВАС, заданное произведением элементарных одноэлементных событий, может быть представлено в виде системы регулярных выражений (4.12), в которой любое входящее в систему регулярное выражение представляется произведением двух переменных:

$$S^i_{\alpha,\beta} = S^i_{\alpha,\beta+1} z_p, \qquad (4.36)$$

где $S^i_{\alpha,\beta+1}$ – сокращенное обозначение события, являющегося непосредственно предшествующим событию $S^i_{\alpha,\beta}$, а z_p – буква входного алфавита [*Z*].

Выражения типа (4.36) на основании предыдущих результатов могут быть представлены формулами исчисления предикатов в следующем виде:

$$S^i_{\alpha,\beta}(t) = (\underset{\tau \le t}{\exists \tau}) z_p(\tau) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall \tau_1}) e(\tau_1) \,\&\, S^i_{\alpha,\beta+1}(\tau - 1). \qquad (4.37)$$

На основании полученных в разделе 4.2 результатов можно сформулировать следующее предложение:

любое регулярное выражение алгебры событий, заданное произведением элементарных одноэлементных событий $z_1, z_2, ..., z_F$, *может быть описано в алфавите [Z] системой одноместных формул исчисления предикатов первого порядка с ограниченными кванторами вида (4.37), общее число формул системы определяется числом букв входного алфавита [Z], входящих в исходное произведение [22].*

В том случае, когда в исходном выражении для произведения элементарных одноэлементных событий (4.11) была выполне-

на замена, например, вида $z_p = \{R\}$, используя правила развертывания итерации, на основании (4.34) получим:

$$S^i_{\alpha,\beta}(t) = S^i_{\alpha,\beta+1}(t) \vee (\underset{\tau \le t}{\exists\tau}) R(\tau) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall\tau_1}) \,\&\, S^i_{\alpha,\beta}(\tau - 1), \qquad (4.38)$$

где R – любое регулярное выражение алгебры событий.

В том случае, когда выражение для R представлено не в виде одной буквы, используют операции подстановок, позволяющие получить описания событий, заданных произвольными регулярными выражениями алгебры событий. Например, пусть

$$R = z_i \vee z_k z_r . \qquad (4.39)$$

Требуется построить предикатную формулу $S^i_{\alpha,\beta}(t)$, получающуюся из (4.38) путем подстановки вместо группы предикатов с переменными в называющей форме, описывающих левую часть формулы (4.39), группу предикатов в называющей форме, описывающих правую часть формулы (4.39). Тогда событие $S^i_{\alpha,\beta}(t)$ запишется так:

$$S^i_{\alpha,\beta}(t) = S^i_{\alpha,\beta+1}(t) \vee (\exists\tau)[[\tau \le t \,\&\, \chi_i(\tau) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall\tau_1}) \,\&\, e(\tau_1) \vee$$

$$\vee (\underset{\delta \le t}{\exists\delta}) \chi_r(\delta) \,\&\, (\underset{\delta < \delta_1 \le t}{\forall\delta_1}) e(\delta_1) \,\&\, \tau < \delta \,\&\, \chi_k(\tau) \,\&\, (\underset{\tau < \tau_1 \le \delta}{\forall\tau_1}) e(\tau_1)] \,\&\, S^i_{\alpha,\beta}(\tau - 1)].$$

Выполнив необходимые преобразования, в том числе частичное переименование связанной переменной, окончательно получим:

$$S^i_{\alpha,\beta}(t) = S^i_{\alpha,\beta+1}(t) \vee (\underset{\tau \le t}{\exists\tau}) \chi_i(\tau) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall\tau_1}) e(\tau_1) \,\&$$

$$\&\, S^i_{\alpha,\beta}(\tau - 1) \vee (\underset{\delta \le t}{\exists\tau}) \chi_r(\tau) \,\&\, (\underset{\delta < \delta_1 \le t}{\forall\tau_1}) e(\tau_1) \,\&\, (\underset{\sigma < \tau}{\exists\tau}) \chi_k(\sigma) \,\&$$

$$\&\, (\underset{\sigma < \sigma_1 \le \tau}{\forall\sigma_1}) e(\sigma_1) \,\&\, S^i_{\alpha,\beta}(\sigma - 1). \qquad (4.40)$$

Обозначив на основании (4.24) связную группу предикатов

$$(\underset{\sigma < \tau}{\exists\sigma}) \chi_k(\sigma) \,\&\, (\underset{\sigma < \sigma_1 \le \tau}{\forall\sigma_1}) e(\sigma_1) \,\&\, S^i_{\alpha,\beta}(\sigma - 1) \qquad (4.41)$$

через $S_1(\tau - 1)$, получим следующую форму записи выражения (4.40):

$$S^i_{\alpha,\beta}(t) = S^i_{\alpha,\beta+1}(t) \vee (\underset{\tau \le t}{\exists \tau}) \chi_i(\tau) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall \tau_1}) e(\tau_1) \,\&\, S^i_{\alpha,\beta}(\tau - 1) \vee$$

$$\vee (\underset{\tau \le t}{\exists \tau}) \chi_r(\tau) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall \tau_1}) e(\tau_1) \,\&\, S_1(\tau - 1), \qquad (4.42)$$

откуда следует, что событие $S^i_{\alpha,\beta}$ будет представлено двумя предикатными формулами, одна из которых (4.42), а другая, определяемая связной группой предикатов (4.41), после переименования связанной переменной будет иметь вид:

$$S_1(t) = (\underset{\tau \le t}{\exists \tau}) \chi_k(\tau) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall \tau_1}) e(\tau_1) \,\&\, S^i_{\alpha,\beta}(\tau - 1). \qquad (4.43)$$

Таким образом, исходя из полученных результатов, справедливо следующее утверждение:

любое регулярное выражение алгебры событий S_α, заданное в алфавите [Z], может быть рекурсивно описано в этом же алфавите системой (A_α) предикатных формул, полученных из выражений типа (4.37) при помощи необходимого количества подстановок. При этом любые предикатные формулы системы имеют одинаковую структуру и представляются в виде выражений, состоящих из трех частей, как это было показано, например, в (4.37).

Из сформулированного утверждения и изложенного выше можно вывести *следующее следствие*:

если в исчислении предикатов первого порядка некоторый предикат P(t) можно эквивалентными преобразованиями в алфавите [Z] привести к виду (A_α), то предикат P(t) в этом же алфавите [Z] описывает вполне определенное регулярное выражение алгебры событий и на основании первой теоремы С. К. Клини [13] представим в конечном цифровом автомате.

4.2.2. Формализация словесных алгоритмов, определяющих реализуемые в цифровом автомате события, на основе использования языка исчисления предикатов

Для удобства и упорядочения формы записи исходных предикатных формул, формализующих словесные алгоритмы, введем

понятие номера и ранга рассматриваемого события и номера ветви алгоритма, определяющей это событие [12].

Номером события S будем называть натуральное число 0, 1, 2, …, α, …, приписываемое к обозначению события в качестве первого нижнего индекса.

Рангом события S_α будем называть натуральное число 0, 1, 2, …, β, …, приписываемое к обозначению события в качестве второго нижнего индекса.

Событием нулевого (старшего) ранга относительно описываемого события S_α будем называть самое это событие S_α, а также события, непосредственно определяющие событие S_α как произвольную двузначную функцию [36].

Событием первого ранга относительно события S_α будем называть каждое событие, непосредственно предшествующее событию S_α.

Событием (β+1)-го ранга относительно события S_α будем называть каждое событие, непосредственно предшествующее любому из событий β-го ранга относительно события S_α.

Если событие $S_{\alpha,\beta}$ непосредственно определяется логической суммой других событий, то для отличия таких дизъюнктивных членов введем первый верхний индекс, который будем называть номером ветви алгоритма, определяющего событие S_α. При этом различные ветви алгоритма должны иметь различные номера, т.е. обозначения событий, порожденных различным образом из исходного, при одинаковых нижних индексах должны иметь различные первые верхние индексы.

Для формального описания на языке исчисления предикатов словесных алгоритмов, определяющих реализуемые в цифровом автомате события, можно воспользоваться результатами, получаемыми в разделе 4.2.1 относительно представления любого регулярного выражения алгебры событий в виде системы рекурсивно определяемых предикатных формул, имеющих одинаковую структуру вида (4.37).

Такая система в расширенном исчислении предикатов может быть преобразована в более удобную для формализации словесных алгоритмов системы $(A)_p$. Под расширенным исчислением предикатов в данном случае понимается исчисление предикатов с добавлением к нему двух аксиом, формализующих закон нейтральности пустого слова алгебры событий [9, 22].

Аксиома 1.

$$F(t) \approx (\underset{\tau \le t}{\exists \tau}) F(\tau) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall \tau_1}) e(\tau_1). \qquad (4.44)$$

Аксиома 2.

$$(\underset{\tau < \tau_1 \le t}{\forall \tau}) F(\tau_1) \approx (\underset{\tau < \tau_1 \le t}{\forall \tau_1}) e(\tau_1) \vee \tau < t \,\&\, F(t) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall \tau_1}) F(\tau_1), \qquad (4.45)$$

где ≈ – знак эквивалентности в расширенном исчислении предикатов; $F(t)$ – предикатная формула.

Для построения новой системы $(A)_p$ введем понятие рода событий и ветвей алгоритма.

Все события β-го ранга $S_{\alpha,\beta}^{[i,j]}$, непосредственно определяющие некоторое событие $S_{\alpha,\beta}^{i}$, разделим на две принципиально разные группы событий. К одной группе отнесем все события, которые определяют зарождение, т.е. первичное появление события $S_{\alpha,\beta}^{i}$. Будем называть такие события событиями 1-го рода.

К другой группе отнесем все события $S_{\alpha,\beta}^{[i,j]}$, которые определяют возобновление события $S_{\alpha,\beta}^{i}$, т.е. повторное его появление. Будем называть такие события событиями 2-го рода.

Аналогично ветви алгоритма, описывающие события 1-го (2-го) рода, будем называть ветвями 1-го (2-го) рода.

В связи с изложенным выражение вида (4.37), описывающее в исчислении предикатов событие $S_{\alpha,\beta}^{i}$, может быть записано так:

$$S_{\alpha,\beta}^{i}(t) = \bigvee_{j=1}^{j=k_1} S_{\alpha,\beta}^{[i,j]}(t) \vee \bigvee_{j=1}^{j=k_2} S_{\alpha,\beta}^{[i,k_1+j]}(t), \qquad (4.46)$$

где k_1 – число ветвей 1-го рода; k_2 – число ветвей 2-го рода.

Выполнив подстановку вместо предикатов выражения (4.46) обозначаемых ими предикатных формул, получим:

$$S_{\alpha,\beta}^{i}(t) = \bigvee_{j=1}^{j=k_1} (\underset{\tau \le t}{\exists \tau}) \chi_{\alpha,\beta}^{[i,j]}(\tau) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall \tau_1}) e(\tau_1) \,\&\, S_{\alpha,\beta+1}^{[i,j]}(\tau - 1) \vee$$

$$\vee \bigvee_{j=1}^{j=k_2} (\underset{\tau \le t}{\exists \tau}) \chi_{\alpha,\beta}^{[i,k_1+j]}(\tau) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall \tau_1}) e(\tau_1) \,\&\, S_{\alpha,\beta}^{i}(\tau - 1), \qquad (4.47)$$

где $\chi_{\alpha,\beta}(\tau)$ – предикат, формализующий частный входной сигнал.

На основании аксиом, формализующих закон нейтральности пустого слова (4.44), (4.45), нетрудно доказать, что выражение (4.47) в расширенном исчислении предикатов эквивалентно выражению

$$S^i_{\alpha,\beta}(t) = \bigvee_{j=1}^{j=k_1} (\exists\tau)_{\tau\le t} \chi^{[i,j]}_{\alpha,\beta}(\tau) \,\&\, (\forall\tau_1)_{\tau<\tau_1\le t} q^i_{\alpha,\beta}(\tau_1) \,\&\, S^{[i,j]}_{\alpha,\beta+1}(\tau-1), \quad (4.48)$$

где $q^i_{\alpha,\beta}(\tau_1) = \bigvee_{r=1}^{r=k_2} \chi^{[i,k_1+r]}_{\alpha,\beta}(\tau_1)$.

На основании изложенного при формализации словесных алгоритмов функционирования цифровых автоматов можно использовать систему $(A)_p$, построенную на основе предикатных выражений вида (4.48), каждое из которых представляет событие, реализуемое в автомате. При этом описания таких событий имеют одинаковую структуру, состоящую из упомянутых ранее трех частей (4.24): описание наступления события и его сохранения и описание непосредственно предшествующего события.

4.2.3. Построение СКУ по описанию алгоритма обработки информации, представленного на языке исчисления предикатов

Система канонических уравнений, являющаяся основой для построения структуры цифровых устройств обработки информации, функционирование которых представлено на начальном языке исчисления предикатов первого порядка, может быть получена из системы $(A)_p$ как способом, изложенным в [34], так и на основании аксиом (4.44) и (4.45).

На основании аксиом (4.44) и (4.45) выражение вида (4.48) преобразуется в каноническое уравнение по следующему алгоритму [37]:

1. Применить один раз аксиому 2 (4.45) к предикату $(\forall\tau_1)_{\tau<\tau_1\le t} q^i_{\alpha,\beta}(\tau_1)$ выражения (4.48). В результате чего получим:

$$S^i_{\alpha,\beta}(t) = \bigvee_{j=1}^{j=k_1} (\exists\tau)_{\tau\le t} \chi^{[i,j]}_{\alpha,\beta}(\tau) \,\&\, (\forall\tau_1)_{\tau<\tau_1\le t} e(\tau_1) \,\&\, S^{[i,j]}_{\alpha,\beta+1}(\tau-1) \vee \bigvee_{j=1}^{j=k_1} (\exists\tau)_{\tau\le t} \chi^{[i,j]}_{\alpha,\beta}(\tau) \,\&$$

$$\&\,\tau < t \,\&\, q^i_{\alpha,\beta}(t) \,\&\, (\forall\tau_1)_{\tau<\tau_1\le t} q^i_{\alpha,\beta}(\tau_1) \,\&\, S^i_{\alpha,\beta+1}(\tau-1). \quad (4.49)$$

Перейти к выполнению п. 2.

2. Учитывая, что $q^i_{\alpha,\beta}(t)$ не содержит свободно анонимную переменную τ и формулу $\tau \le t \,\&\, \tau < t \equiv \tau < t$, записать выражение (4.49) в следующей эквивалентной форме:

$$S^i_{\alpha,\beta}(t) = \bigvee_{j=1}^{j=k_1} (\exists\tau)_{\tau\le t} \chi^{[i,j]}_{\alpha,\beta}(\tau) \,\&\, (\forall\tau_1)_{\tau<\tau_1\le t} e(\tau_1) \,\&\, S^{[i,j]}_{\alpha,\beta+1}(\tau-1) \vee q^i_{\alpha,\beta}(t) \,\&$$

$$\&\Big[\bigvee_{j=1}^{j=k_1} (\exists\tau)_{\tau\le t} \chi^{[i,j]}_{\alpha,\beta}(\tau) \,\&\, (\forall\tau_1)_{\tau<\tau_1\le t} q^i_{\alpha,\beta}(\tau_1) \,\&\, S^{[i,j]}_{\alpha,\beta+1}(\tau-1)\Big]. \quad (4.50)$$

Перейти к выполнению требований п. 3.

3. Выражение (4.50) путем применения к первому дизъюнктивному члену аксиомы 1 (4.44) и сокращенного обозначения на основании (4.44) соответствующей связной группы предикатов (обозначена квадратными скобками) преобразовать к виду

$$S^i_{\alpha,\beta}(t) = \bigvee_{j=1}^{j=k_1} \chi^{[i,j]}_{\alpha,\beta}(t) \,\&\, S^{[i,j]}_{\alpha,\beta+1}(t-1) \vee q^i_{\alpha,\beta}(t) \,\&\, S^i_{\alpha,\beta}(t-1). \quad (4.51)$$

На этом работа алгоритма заканчивается.

4.2.4. Пример синтеза абстрактного цифрового автомата с использованием языка исчисления предикатов первого порядка

Требуется построить цифровой автомат Мура, заданный следующим словесным алгоритмом.

1. На вход автомата могут подаваться восемь элементарных двоичных входных сигналов:

$$[X] = x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7.$$

2. Цифровой автомат должен функционировать циклически. Начало цикла должно определять событие S_0, отмеченное выходным сигналом y_0.

Событие S_0 должно иметь место при поступлении на вход автомата элементарного входного сигнала x_0 или x_7 независимо от наличия в этот момент времени других элементарных входных сигналов, а также в том случае, если после зарождения события S_0 другие входные сигналы на вход автомата еще не подавались.

3. В цифровом автомате должно иметь место событие S_1, отмеченное выходным сигналом y_1, если в интервале времени между поступлением, соответственно, сигналов x_5 и x_6 (не включая моменты времени поступления сигналов x_5 и x_6) на вход автомата было подано четное количество сигналов x_2, которое (это четное количеством сигналов x_2) может чередоваться с произвольным количеством сигналов x_1, при условии, что для всех моментов времени этого интервала другие сигналы, кроме, быть может, сигнала x_4, на вход автомата не поступают.

Все сигналы, кроме сигналов x_0 и x_7, поданные на вход автомата после поступления сигнала x_6, на событие S_1 не влияют.

Одновременно с сигналами x_5 и x_6 на вход автомата может поступать, не влияя на событие S_1, только сигнал x_3.

Прежде чем производить формализацию этого словесного алгоритма, введем для сокращения записи следующие обозначения входных сигналов:

$$e_{2,4}(\tau) = \overline{x}_0(\tau) \,\&\, \overline{x}_1(\tau) \,\&\, \overline{x}_3(\tau) \,\&\, \overline{x}_5(\tau) \,\&\, \overline{x}_6(\tau) \,\&\, \overline{x}_7(\tau),$$

$$e_{1,4}(\tau) = \overline{x}_0(\tau) \,\&\, \overline{x}_2(\tau) \,\&\, \overline{x}_3(\tau) \,\&\, \overline{x}_5(\tau) \,\&\, \overline{x}_6(\tau) \,\&\, \overline{x}_7(\tau),$$

$$e_{3,5}(\tau) = \overline{x}_0(\tau) \,\&\, \overline{x}_1(\tau) \,\&\, \overline{x}_2(\tau) \,\&\, \overline{x}_4(\tau) \,\&\, \overline{x}_6(\tau) \,\&\, \overline{x}_7(\tau),$$

$$e_{3,6}(\tau) = \overline{x}_0(\tau) \,\&\, \overline{x}_1(\tau) \,\&\, \overline{x}_2(\tau) \,\&\, \overline{x}_4(\tau) \,\&\, \overline{x}_5(\tau) \,\&\, \overline{x}_7(\tau),$$

$$e_4(\tau) = \overline{x}_0(\tau) \,\&\, \overline{x}_1(\tau) \,\&\, \overline{x}_2(\tau) \,\&\, \overline{x}_3(\tau) \,\&\, \overline{x}_5(\tau) \,\&\, \overline{x}_6(\tau) \,\&\, \overline{x}_7(\tau).$$

Для упрощения процесса формализации предикатных формул, описывающих исходные события, воспользуемся временными схемами их формирования (рис. 4.1).

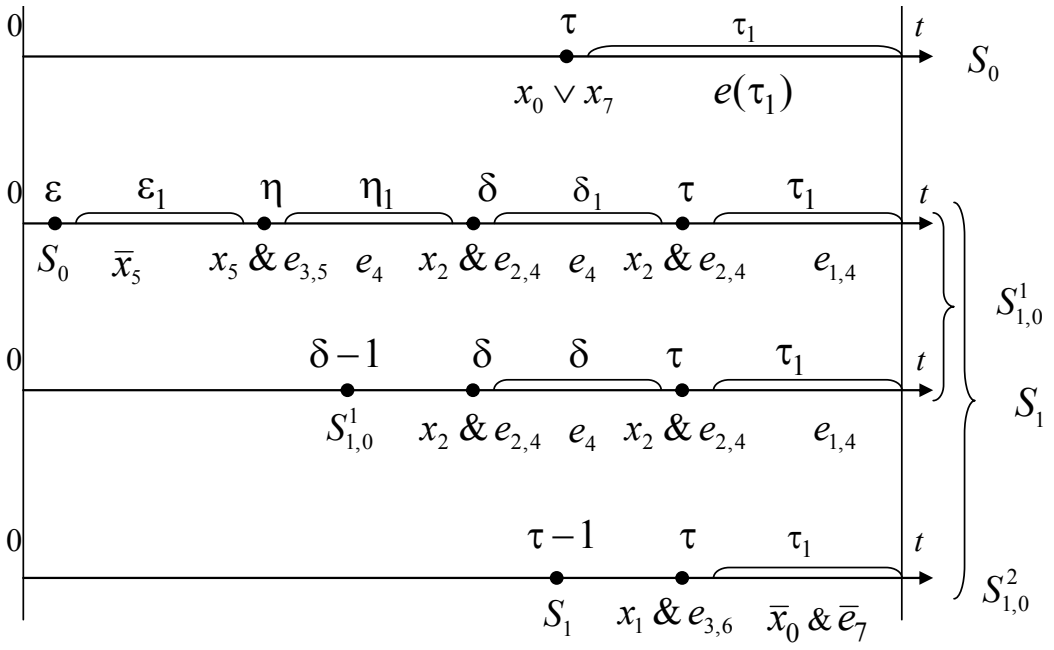


Рис. 4.1. Временные схемы формирования событий S_0 и S_1

П р и м е ч а н и е. При построении временных схем формирования событий временные интервалы, в течение которых действуют входные сигналы, обозначаются буквами с индексами, а моменты времени поступления входных сигналов – буквами без индексов.

Выполняя формализацию словесного алгоритма с учетом (4.48) и временных схем формирования событий (см. рис. 4.1), получим систему $\left(A\right)_p$, представляющую заданные события S_0 и S_1, она имеет вид

$$
\begin{aligned}
&S_0^{y_0}(t) = (\underset{\tau\le t}{\exists\tau})\left[x_0(\tau)\vee x_7(\tau)\right] \,\&\, (\underset{\tau<\tau_1\le t}{\forall\tau_1})e(\tau_1),\\
&S_1^{y_1}(t) = (\underset{\tau\le t}{\exists\tau})x_2(\tau)\,\&\,e_{2,4}(\tau)\,\&\,(\underset{\tau<\tau_1\le t}{\forall\tau_1})e_{1,4}(\tau_1)\,\&\,(\underset{\delta<\tau}{\exists\delta})x_2(\delta)\,\&\\
&\&\,e_{2,4}(\delta)\,\&\,(\underset{\delta<\delta_1\le\tau}{\forall\delta_1})e(\delta_1)\,\&\,(\underset{\eta<\delta}{\exists\eta})x_5(\eta)\,\&\,e_{3,5}(\eta)\,\&\\
&\&\,(\underset{\eta<\eta_1<\delta}{\forall\eta_1})e_4(\eta_1)\,\&\,(\underset{\varepsilon<\eta}{\exists\varepsilon})S_0(\varepsilon)\,\&\,(\underset{\varepsilon<\varepsilon_1<\eta}{\forall\varepsilon_1})\bar{x}_5(\varepsilon_1)\vee\\
&\vee(\underset{\tau\le t}{\exists\tau})x_2(\tau)\,\&\,e_{2,4}(\tau)\,\&\,(\underset{\tau<\tau_1\le t}{\forall\tau_1})e_{1,4}(\tau_1)\,\&\\
&\&\,(\underset{\delta<\tau}{\exists\delta})x_2(\delta)\,\&\,e_{2.4}(\delta)\,\&\,(\underset{\delta<\delta_1<\tau}{\forall\delta_1})e(\delta_1)\,\&\,S_1(\delta-1)\vee\\
&\vee(\underset{\tau\le t}{\exists\tau})x_6(\tau)\,\&\,e_{3,6}(\tau)\,\&\,(\underset{\tau<\tau_1\le t}{\forall\tau_1})\overline{x_0(\tau_1)}\,\&\,\overline{x_7(\tau_1)}\,\&\,S_1(\tau-1).
\end{aligned}
\qquad (4.52)
$$

Выполняя преобразование уравнения системы (4.52), определяющего событие S_1, за счет введения сокращенного обозначения связных групп предикатных выражений, получим систему рекурсивных предикатных выражений, каждое из которых имеет одинаковую структуру, соответствующую предикатному выражению (4.48) и состоящую из упомянутых ранее трех частей.

$$
\begin{aligned}
&S_1(t) = S^1_{1,0} \vee S^2_{1,0},\\
&S^1_{1,0}(t) = (\underset{\tau \le t}{\exists \tau}) x_2(\tau) \,\&\, e_{2,4}(\tau) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall \tau_1}) e_{1,4}(\tau_1) \,\&\, S^1_{1,1}(\tau - 1),\\
&S^1_{1,1}(t) = (\underset{\delta \le \tau}{\exists \delta}) x_2(\delta) \,\&\, e_{2,4}(\delta) \,\&\, (\underset{\delta < \delta_1 < \tau}{\forall \delta_1}) e(\delta_1) \,\&\\
&\qquad \&\, [S^1_{1,2}(\delta - 1) \vee S^1_{1,0}(\delta - 1)],\\
&S^1_{1,2}(t) = (\underset{\eta < \delta}{\exists \eta}) x_5(\eta) \,\&\, e_{3,5}(\eta) \,\&\, (\underset{\eta < \eta_1 < \delta}{\forall \eta_1}) e_4(\eta_1) \,\&\, S^1_{1,3}(\eta - 1),\\
&S^1_{1,3}(t) = (\underset{\varepsilon < \eta}{\exists \varepsilon}) S_0(\varepsilon) \,\&\, (\underset{\varepsilon < \varepsilon_1 < \eta}{\forall \varepsilon_1}) \overline{x}_5(\varepsilon_1),\\
&S^2_{1,0}(t) = (\underset{\tau \le t}{\exists \tau}) x_6(\tau) \,\&\, e_{3,6}(\tau) \,\&\, (\underset{\tau < \tau_1 \le t}{\forall \tau_1}) \overline{x_0(\tau_1)} \,\&\, \overline{x_7(\tau_1)} \,\&\, S_1(\tau - 1).
\end{aligned}
\tag{4.53}
$$

Выполняя спуск кванторов в уравнениях системы (4.53) на основании аксиом (4.44) и (4.45), получим следующую систему бескванторных канонических уравнений, которая имеет вид

$$
\begin{aligned}
&S_0(t) = x_0(t) \vee x_7(t),\\
&S_1(t) = S^1_{1,0} \vee S^2_{1,0},\\
&S^1_{1,0}(t) = x_2(t) \,\&\, e_{2,4}(t) \,\&\, S^1_{1,1}(t-1) \vee e_{1,4}(t) \,\&\, S^1_{1,0}(t-1),\\
&S^1_{1,1}(t) = x_2(t) \,\&\, e_{2,4}(t) \,\&\, \left[S^1_{1,2}(t-1) \vee S^1_{1,0}(t-1) \right] \vee\\
&\vee e_4(t) \,\&\, S^1_{1,1}(t-1),\\
&S^1_{1,2}(t) = x_5(t) \,\&\, e_{3,5}(t) \,\&\, S^1_{1,3}(t-1) \vee e(t) \,\&\, S^1_{1,2}(t-1),\\
&S^1_{1,3}(t) = S_0(t) \vee \overline{x_5}(t) \,\&\, S^1_{1,3}(t-1), \qquad S_0(t) \Rightarrow S^1_{1,3},\\
&S^2_{1,0}(t) = x_6(t) \,\&\, e_{3,6}(t) \,\&\, S^1_{1,0}(t-1) \vee \overline{x_0}(t) \,\&\, \overline{x_7}(t) \,\&\, S^2_{1,0}.
\end{aligned}
\tag{4.54}
$$

Для наглядности при проверке правильности формально представленных событий можно использовать графы, реализующие эти события. Для нашего примера граф, построенный на основе системы (4.54) и реализующий событие S_1, имеет вид (рис. 4.2).

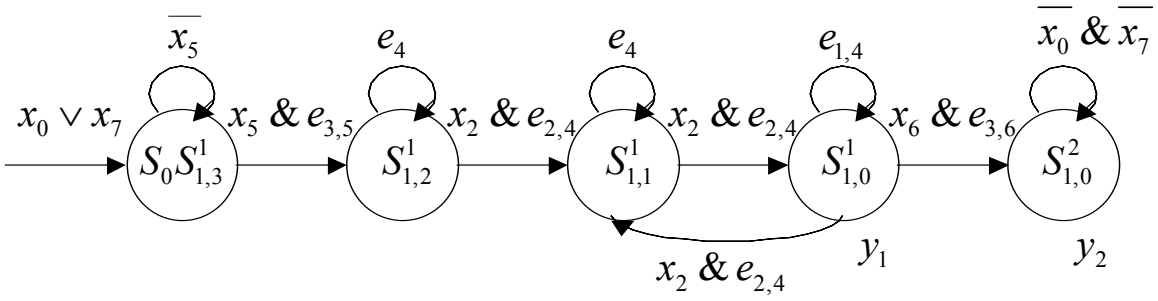


Рис. 4.2. Граф, реализующий событие S_1

В заключение рассмотрим представление события S_1 на языке РВАС и построение на его основе системы канонических уравнений. Для этого воспользуемся временными схемами формирования события S_1 (см. рис. 4.1), на основании которых событие S_1 на языке РВАС может быть представлено следующим выражением:

$$S_1 = S_0\{\overline{x}_5\}(x_5 e_{3,5})\{e_4\}(x_2 e_{2,4})\{e_4\}(x_2 e_{2,4})\{e_{1,4}\}\times$$

$$\times\{(x_2 e_{2,4})\{e_4\}(x_2 e_{2,4})\{e_{1,4}\}\vee(x_6 e_{3,6})\{(\overline{x}_0\ \overline{x}_7)\}\}. \qquad (4.55)$$

Выполнив в выражении (4.55) раскрытие итерационных скобок, в которых представлено описание условия сохранения события S_1, и некоторые предварительные преобразования, получим описание события S_1 в следующем виде:

$$S_1 = [S_0\{\overline{x}_5\}(x_5 e_{3,5})\{e_4\}(x_2 e_{2,4})\vee S_1(x_2 e_{2,4})\{e_4\}](x_2 e_{2,4})\{e_{1,4}\}\vee$$

$$\vee S_1(x_6 e_{3,6})\{(\overline{x}_0\ \overline{x}_7)\} = S^1_{1,0}\vee S^2_{1,0}. \qquad (4.56)$$

Полученное регулярное выражение (4.56) на основании методики, представленной в разделе 4.1.2, преобразуется в следующую каноническую систему регулярных выражений для всех событий, реализуемых в исходном алгоритме для события S_1:

$$
\begin{aligned}
&S^1_{1,0} = S^1_{1,1}(x_2 e_{2,4})\{e_{1,4}\} = S^1_{1,1}(x_2 e_{2,4}) \vee S^1_{1,0}\, e_{1,4},\\
&S^1_{1,1} = S^1_{1,2}(x_2 e_{2,4})\{e_4\} \vee S_1(x_2 e_{2,4})\{e_4\} = (S^1_{1,2} \vee S_1)e_4,\\
&S^1_{1,2} = S_0\{\overline{x}_5\}(x_5 e_{3,5})\{e_4\} = S^1_{1,3}(x_5 e_{3,5})\{e_4\} =\\
&\qquad = S^1_{1,3}(x_5 e_{3,5}) \vee S^1_{1,2}\, e_4,\\
&S^1_{1,3} = S_0\{\overline{x}_5\} = S_0 \vee S_{1,3}\overline{x}_5 \ , \qquad S_0 \Rightarrow S^1_{1,3},\\
&S^2_{1,0} = S_1(x_6 e_{3,6})\{(\overline{x}_0\ \ \overline{x}_7)\} = S_1(x_6 e_{3,6}) \vee S^2_{1,0}\{(\overline{x}_0\ \ \overline{x}_7)\}.
\end{aligned}
\tag{4.57}
$$

Система уравнений (4.57) полностью соответствует канонической системе уравнений (4.54), полученной по описанию события S_1 на языке исчисления предикатов.

4.3. Язык граф-схем алгоритмов (ГСА)

4.3.1. Общие сведения о языке ГСА

Граф-схемы алгоритмов (ГСА) нашли широкое применение для описания функционирования управляющих микропрограммных устройств в различных отраслях техники, в том числе и в вычислительной технике. Это объясняется их хорошей обозримостью, наглядностью и простотой конструкции, а также возможностью преобразований и формального перехода к таблицам перехода и к аналитическому представлению алгоритмов в виде СКУ.

Язык ГСА совместно с языком логических схем алгоритмов (ЛСА) иногда называют языком операторных схем алгоритмов (ОСА).

Язык ГСА – графический язык, поэтому символы, применяемые в нем, имеют определенное геометрическое начертание, определяемое ГОСТом. В языке ГСА применяются четыре основных символа для обозначения начальной, конечной, операторных и логических (условных) вершин. Для указания связи между прерванными линиями внутри одного листа или между разными листами могут быть использованы два дополнительных символа. Любой алгоритм должен начинаться и заканчиваться символами начальной и конечной вершин. Начальная вершина A_0 имеет только одну выходящую линию, а конечная A_k – только входящие

линии. Операторной вершине сопоставляется вполне определенный оператор A_i, символизирующий определенные действия. На первом этапе проектирования алгоритма функционирования цифрового автомата используют содержательные ГСА, когда внутри операторных вершин записывается содержательное обозначение управляющих действий. Например, для алгоритма выполнения каких-либо операций в операционном устройстве ЭВМ внутри операторных вершин ГСА управления выполнением таких операций записываются микрооперации в виде оператора присваивания или совокупности таких операторов. Внутри условных вершин в содержательный ГСА записывается некоторое логическое выражение, принимающее значение I или 0, или "Истина" или «Ложно».

При кодировании содержательной ГСА внутри операторных вершин записываются символы из множества выходных сигналов структурного алфавита [*Y*], которые в МПА инициируют выполнение соответствующих микроопераций в операционном автомате, а внутри условных вершин – из множества входных сигналов структурного алфавита [*X*], которые для МПА принято называть осведомительными сигналами.

Операторная вершина имеет одну входящую и одну выходящую линии, причем входящая линия может быть образована слиянием нескольких линий. Условная вершина имеет одну входящую линию и две выходящих. Входящая линия также может быть образована слиянием нескольких линий. Выходящие линии помечаются цифрами I и 0 или словами «Да» и «Нет».

Запись алгоритма функционирования цифрового автомата на языке ГСА сводится к начертанию графических символов указанных типов и соединению их в определенном порядке между собой с помощью линий. При этом должны соблюдаться следующие условия [23]:

1. Входы и выходы вершин соединяются друг с другом с помощью линий, направленных от выхода к входу.

2. Каждый выход соединен точно с одним входом.

3. Любой вход соединяется по крайней мере с одним выходом.

4. Для любой вершины графа существует по крайней мере один путь из этой вершины к конечной вершине.

5. Один из выходов условной вершины может соединяться с ее входом, что недопустимо для операторной вершины. В случае соединения выхода условной вершины с ее входом в цепь обрат-

ной связи вводится пустая операторная вершина, отмеченная пустым выходным сигналом y_e.

Рассмотрим выполнение алгоритма по ГСА, представленной на рис. 4.3.

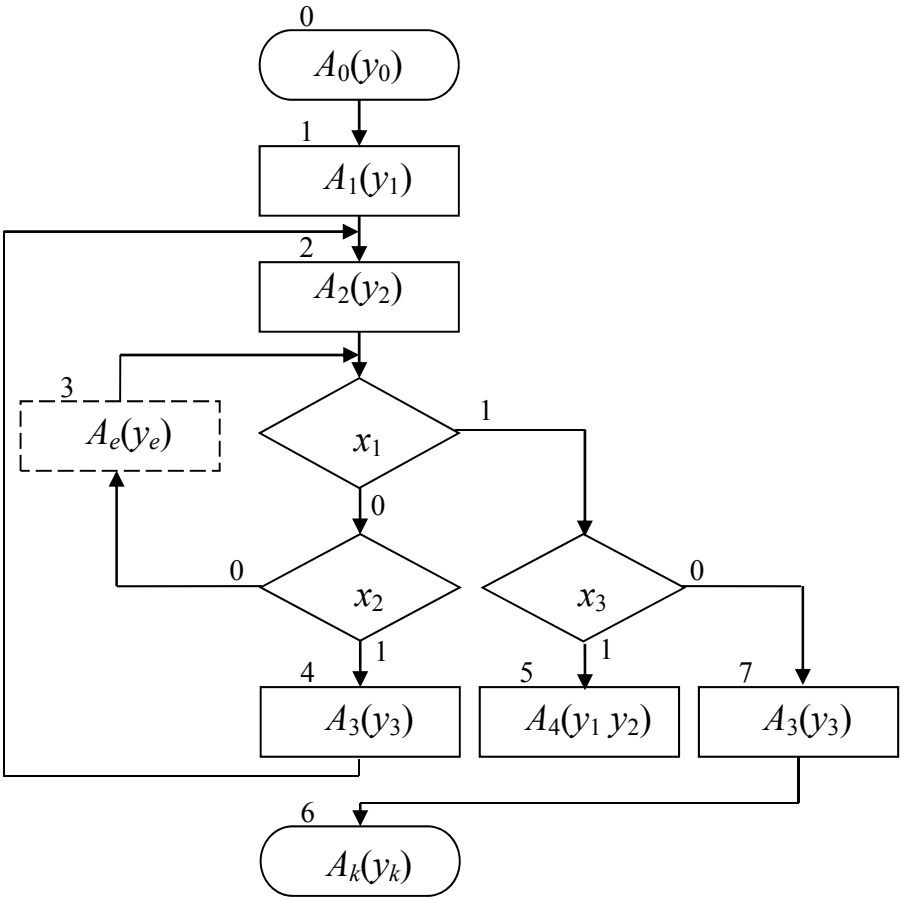


Рис. 4.3. Граф-схема алгоритма произвольного автомата

Выполнение алгоритма начинается с оператора A_0. На следующих двух шагах выполняются последовательно операторы A_1 и A_2. На третьем и четвертом шагах может выполняться или один из трех операторов A_3, A_4 и A_k, или выполняется пустой оператор A_e в зависимости от значений логических условий x_1, x_2 и x_3. Если $x_1 = x_2 = 0$, то независимо от значения сигнала x_3 будет выполняться пустой оператор A_e, т.е. автомат будет находиться в режиме ожидания, выдавать пустой сигнал y_e до тех пор, пока x_1 или x_2 не станет равным единице. Если $\overline{x}_1 = x_2 = 1$, то выполняется оператор A_3, если $x_1 = 1$, то выполняется или A_4 при $x_3 = 1$, или

A_3 при $x_3 = 0$. Предположим, что после выполнения A_2 имеет место $\overline{x}_1 = x_2 = 1$. Тогда независимо от значения x_3 будут последовательно выполняться A_2 и A_3 столько раз, пока после очередного выполнения оператора A_2 и проверки логических условий логическое условие x_1 не станет равным единице. После этого выполняется оператор A_4 (или как было отмечено выше). Выполнение алгоритма закончится оператором A_k.

4.3.2. Построение таблиц переходов и СКУ по ГСА

СКУ в принципе можно построить сразу непосредственно по ГСА, поставив в соответствие каждой операторной вершине событие СКУ, учитывая при этом, что одинаковым операторам, стоящим в разных местах ГСА, должны соответствовать разные события. Однако для облегчения процесса построения СКУ целесообразно сначала по ГСА построить прямую таблицу переходов для событий, которая будет соответствовать прямой таблице переходов автомата, так как рассматриваемый вид ГСА не допускает параллельных ветвей алгоритма. По прямой таблице переходов строится в дальнейшем СКУ, тем более, что во многих случаях прямая таблица переходов автомата используется для структурного синтеза автомата и самостоятельно. Таким образом, построенная по ГСА СКУ будет детерминированной всегда и каждому событию СКУ будет соответствовать свое вполне определенное состояние автомата.

Алгоритм построения СКУ для автомата Мура будет включать следующие этапы:

1. Разметить (пронумеровать) все вершины ГСА, предварительно введя пустые операторные вершины, если в ГСА имеются циклы из условных вершин. Пустые операторные вершины размещают при выходе из условной вершины, определяющей начало пути в цикле из условных вершин.

2. Каждому оператору ГСА поставить в соответствие вполне определенное событие (состояние автомата), присвоив ему индекс, соответствующий номеру вершины ГСА. При этом необходимо иметь в виду, что одинаковым операторам, стоящим в разных местах ГСА, должны соответствовать разные события (состояния автомата). Это обеспечивается сквозной нумерацией

вершин исходной ГСА. Начальной вершине ГСА будет соответствовать начальное событие S_0 (начальное состояние автомата a_0). Часто начальную и конечную вершины ГСА совмещают, тогда автомат после выполнения программы алгоритма переходит в начальное состояние.

3. Заготовить прямую таблицу переходов, как это выполнялось в гл. 2, и отметить каждую строку в столбце 2 обозначением одного из исходных событий (состояний автомата), начиная с начального события. Осуществить просмотр всех путей перехода от исходного события (состояния автомата) в момент времени (t) к событиям (состояниям автомата) в конце пути в момент времени $(t+1)$. Каждый такой путь должен соответствовать шагу алгоритма и состоять только из логических условий (условных вершин).

Каждому пути из логических условий от одного оператора к другому сопоставить конъюнкцию (сочетание) входных сигналов, т.е. частный входной сигнал автомата $X_{i,j}$, причем в эту конъюнкцию элементарный входной сигнал войдет без отрицания, если выход из условной вершины отмечен 1, и с отрицанием, если – 0.

Для каждого исходного события $S_i(t)$ (состояния автомата) для всех путей, ведущих из него, записать в третьем столбце выражения для частных входных сигналов $X_{i,j}$, а в четвертом столбце записать соответствующие им обозначения событий $S_j(t+1)$ перехода (состояний автомата).

Примечание. Частные входные сигналы $X_{i,j}$, определяемые по ГСА, соответствуют условиям так называемого универсального распределения сдвигов, когда логические условия могут изменяться в любом такте работы автомата. Другие случаи распределения сдвигов рассматриваются, например, в [23–26].

4. Отметив каждое событие (состояние автомата) соответствующими выходными сигналами, получим прямую таблицу переходов автомата Мура, которую необходимо проверить на удовлетворение условиям автоматности: для каждого исходного события (состояния автомата) любое попарное произведение частных входных сигналов для всех путей из каждого из исходных событий должно равняться нулю, а логическая сумма всех частных входных сигналов должна равняться единице.

5. Построение СКУ осуществляется путем отыскания в прямой таблице переходов всех путей, которые оканчиваются одина-

ковыми событиями $S_j(t+1)$. Для них составляются конъюнкции из обозначений событий $S_i(t)$, стоящих вначале пути перехода, с соответствующими частными входными сигналами $X_{i,j}$, вызывающими этот переход. Полученные конъюнкции $X_{i,j}(t)\,\&\,S_i(t)$ объединяют знаком дизъюнкции. Такую операцию выполняют для всех событий $S_j(t+1)$, которые отмечают соответствующими выходными сигналами.

П р и м е р 4.5. Построить прямую таблицу переходов и СКУ по ГСА, представленной на рис. 4.3, для автомата Мура. Введя сквозную нумерацию операторных вершин и следуя предложенному алгоритму, получим искомую прямую таблицу переходов автомата Мура (табл. 4.5).

Таблица 4.5

	Исходное событие (состояние) $S_i(t)$	Частный входной сигнал $X_{i,j}(t)$	Событие (состояние перехода), отмеченное совокупностью выходных сигналов $S_j(t+1)(Y_j)$
1	S_0	1	$S_1(y_1)$
2	S_1	1	$S_2(y_2)$
3	S_2	$\overline{x}_1\,\overline{x}_2$	$S_3(y_e)$
		$\overline{x}_1 x_2$	$S_4(y_3)$
		$x_1\overline{x}_3$	$S_7(y_3)$
		$x_1 x_3$	$S_5(y_1y_2)$
4	S_3	$\overline{x}_1\,\overline{x}_2$	$S_3(y_e)$
		$\overline{x}_1 x_2$	$S_4(y_3)$
		$x_1\overline{x}_3$	$S_7(y_3)$
		$x_1 x_3$	$S_5(y_1y_2)$
5	S_4	1	$S_2(y_2)$
6	S_5	1	$S_6(y_k)$
7	S_7	1	$S_6(y_k)$

По табл. 4.5 в соответствии с п. 5 алгоритма получим следующую СКУ для автомата Мура:

$$S_1^{y_1}(t+1) = S_0,$$

$$S_2^{y_2}(t+1) = S_1 \vee S_4,$$

$$S_3^{y_e}(t+1) = \overline{x}_1 \overline{x}_2 (S_2 \vee S_3),$$

$$S_4^{y_3}(t+1) = \overline{x}_1 x_2 (S_2 \vee S_3),$$

$$S_5^{y_1 y_2}(t+1) = x_1 x_3 (S_2 \vee S_3),$$

$$S_6^{y_k}(t+1) = S_5 \vee S_7,$$

$$S_7^{y_3}(t+1) = x_1 \overline{x}_3 (S_2 \vee S_3).$$

В том случае, если требуется построить автомат на основе модели автомата Мили с использованием прямой таблицы переходов, необходимо в таблице переходов автомата Мура объединить те события (состояния) автомата Мура, переходы из которых полностью совпадают. Для рассмотренного выше примера это соответствует группам событий (состояний): S_2, S_3; S_1, S_4 и S_5, S_7. Делая необходимые подстановки и замены переменных вида: $S_0 = a_0$; $S_1 \vee S_4 = a_1$; $S_2 \vee S_3 = a_2$; $S_5 \vee S_7 = a_3$; $S_6 = a_4$, получим прямую таблицу переходов автомата Мили, эквивалентного автомату Мура (табл. 4.6), дополненную столбцом 5, отмечающим переходы выходными сигналами.

Таблица 4.6

	Исходное состояние $a_i(t)$	Частный входной сигнал $X_{i,j}(t)$	Состояние перехода $a_j(t+1)$	Комбинация выходных сигналов $Y_{i,j}(t)$
1	a_0	1	a_1	y_1
2	a_1	1	a_2	y_2
3	a_2	$\overline{x}_1\,\overline{x}_2$	a_2	y_e
		$\overline{x}_1\,x_2$	a_1	y_3
		$x_1\overline{x}_3$	a_3	y_3
		x_1x_3	a_3	y_1y_2
4	a_3	1	a_4	y_k

Прямой таблице переходов (см. табл. 4.6) будут соответствовать следующие СКУ и СВФ автомата Мили:

$$
\begin{aligned}
a_1(t+1) &= a_0 \vee \overline{x}_1 x_2 a_2, \\
a_2(t+1) &= a_1 \vee \overline{x}_1\ \overline{x}_2\ a_2, \\
a_3(t+1) &= x_1 a_2, \\
a_4(t+1) &= a_3.
\end{aligned}
\qquad (4.58)
$$

$$
\begin{aligned}
y_1(t) &= a_0 \vee x_1 x_3 a_2, \\
y_2(t) &= a_1 \vee x_1 x_3 a_2, \\
y_3(t) &= (\overline{x}_1 x_2 \vee x_1 \overline{x}_3) a_2, \\
y_e(t) &= \overline{x}_1\ \overline{x}_2\ a_2, \\
y_k(t) &= a_3.
\end{aligned}
\qquad (4.59)
$$

4.4. Язык операторных схем алгоритмов с параллельными ветвями (ОСАП)

4.4.1. Общие сведения о языке ОСАП

Возможность формального описания параллельно выполняемых во времени ветвей алгоритма значительно способствует увеличению выразительных средств любого начального языка. Однако параллельные алгоритмы обладают большим разнообразием по сравнению с обычными, что существенно затрудняет осмысливание и правильный учет всех факторов, связанных с параллельно протекающими процессами и их взаимодействием.

Предлагаемый в данном разделе язык ОСАП базируется на расширении синтаксиса известных языков ГСА и ЛСА, а также на использовании модели НДА и стандартного языка СКУ для описания всех частных событий. Основные идеи рассматриваемого языка ОСАП изложены в работах автора в [2, 16–19]. Выбор такого базиса для языка ОСАП вытекает из следующих предпосылок. Использование модели НДА позволяет описывать параллельные ветви алгоритма в виде частных СКУ независимо друг от друга на основе обычных ГСА или ЛСА, а взаимодействие параллельных ветвей и их синхронизацию можно учитывать при построении общей НД СКУ путем надлежащей формализации событий, опре-

деляющих выход алгоритмического процесса за вершину объединения ветвей и коррекции частных событий СКУ, полученных для каждой параллельной ветви.

Применение НД СКУ для формализации взаимодействия параллельных ветвей, а при необходимости и детерминированной СКУ позволяет снять многие ограничения на виды и условия взаимодействия параллельных ветвей, отмеченные в ряде работ. Например, в [43] конечные члены каждой параллельной ветви формализуемого алгоритма должны оканчиваться одновременно; в [39] алгоритмы не должны содержать циклы; в [40] ограничиваются рассмотрением параллельных ветвей, представленных двумя частными видами ГСА; в [41] предлагаемый язык характеризуется ограниченными возможностями по организации взаимодействия параллельных ветвей.

Предлагаемый язык ОСАП, обладая сравнительно широкими выразительными возможностями, позволяет также использовать многие результаты по синтезу цифровых автоматов, полученные до настоящего времени, например, в [26, 42] и др.

Дальнейшее изложение основных положений языка ОСАП будем вести на основе использования языка ГСА в связи с его большей наглядностью по сравнению с языком ЛСА. Учитывая это обстоятельство, будем сокращенно обозначать язык ОСАП на основе языка ГСА через ГСАП.

4.4.2. Основные конструкции, вводимые в язык ГСАП

Для описания параллельных ветвей алгоритма и их взаимодействия в язык ГСА вводятся дополнительные вершины, указывающие места разветвления и объединения параллельных ветвей: разветвители и соединители.

Разветвителю в ГСАП будет соответствовать разветвительная вершина (рис. 4.4), обозначаемая буквой F_α, которая имеет один вход и n выходов, α – номер вершины.

Соединителю в ГСАП будет соответствовать соединительная вершина (рис. 4.5), обозначаемая буквой J_β, имеющая столько входящих линий, сколько параллельных ветвей она объединяет, β – номер вершины.

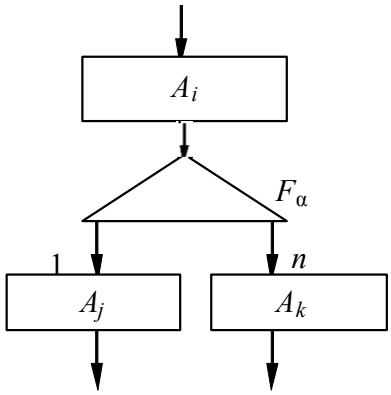


Рис. 4.4. Фрагмент ГСАП с разветвительной вершиной

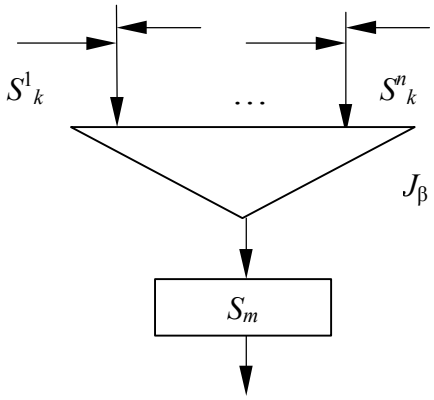


Рис. 4.5. Фрагмент ГСАП с соединительной вершиной

Каждая входящая линия может быть образована слиянием нескольких линий, но все компоненты такой линии должны принадлежать одной и той же параллельной ветви. Выходящая линия из соединителя может быть только одна.

Объединение параллельных ветвей в соединителе осуществляется с учетом определенных условий, т.е. при наступлении определенных ситуаций в объединяемых параллельных ветвях. В самом общем виде условия объединения могут быть заданы в виде булевой функции над обозначениями частных событий, имеющих место в объединяемых параллельных ветвях:

$$S_{\text{в}} = f_{\text{в}}(\tilde{S}_k^1,...,\tilde{S}_k^n,\tilde{S}_\alpha^1,...,\tilde{S}_\beta^n), \qquad (4.60)$$

где $\tilde{S}_k^1,...,\tilde{S}_k^n$ – сокращенное обозначение событий, отмечающих окончание алгоритмического процесса в соответствующих ветвях с 1-й по n-ю; $\tilde{S}_\alpha^1,...,\tilde{S}_\beta^n$ – сокращенное обозначение частных событий, реализуемых в ветвях с 1-й по n-ю;

$\sim$ – знак, означающий, что переменная может быть взята с отрицанием или без отрицания.

В соответствии с выражением (4.60) алгоритмический процесс распространяется за вершину соединения, если будет истинным выражение (4.60), т.е. имеет место соотношение:

$$S_m(t+1) = S_{\text{в}}(t), \tag{4.61}$$

где S_m – первое событие, реализуемое в последовательной части алгоритма после соединителя.

Практический интерес представляет ряд частных случаев задания условий для объединения параллельных ветвей, которые формализуются с использованием соединителей трех типов: конъюнктивного соединителя, дизъюнктивного соединителя и конъюктивно-дизъюнктивного соединителя.

Конъюнктивный соединитель *J*(&).

При использовании такого соединителя алгоритмический процесс распространяется за вершину соединителя в том случае, если имеет определенное сочетание (конъюнкция) событий, отмечающих окончание алгоритмического процесса в объединяемых ветвях, взятых с отрицанием или без отрицания. При этом возможны два частных случая. ***Первый случай*** соответствует алгоритму, когда алгоритмический процесс распространяется за вершину соединителя при условии окончания алгоритмического процесса во всех объединяемых ветвях одновременно (в один и тот же момент времени). Для такого соединителя имеет место соотношение

$$S_{\text{в}} = \tilde{S}_k^1 \cdots \tilde{S}_k^i \cdots \tilde{S}_k^n. \tag{4.62}$$

Второй случай соответствует условию окончания алгоритмических процессов во всех объединяемых ветвях, но не обязательно в один и тот же момент времени. В этом случае факт окончания алгоритмического процесса в каждой из параллельных ветвей должен запоминаться, т.е. этот факт должен найти отражение при формализации условий сохранения событий, отмечающих окончание алгоритмического процесса в каждой из параллельных ветвей. Тогда для любой *i*-й ветви событие S_k^i должно иметь вид

$$S_k^i(t+1) = S_k^i(0) \vee S_k^i \cdot \overline{S}_{\text{в}}, \tag{4.63}$$

где $S_k^i(0)$ – сокращенное обозначение события, определяющего зарождение события S_k^i (его первоначальное появление).

Из соотношения (4.63) следует, что событие S_k^i после своего зарождения сохраняется до тех пор, пока не будет истинным событие $S_в$.

Дизъюнктивный соединитель *J*(v).

Для таких соединителей алгоритмический процесс распространяется за соединитель при окончании его в любой из соединяемых ветвей, т.е. имеем

$$S_в = S_k^1 \vee \cdots \vee S_k^i \vee \cdots \vee S_k^n . \qquad (4.64)$$

Из выражения (4.64) следует, что если по инициативе какой-либо ветви процесс распространился за вершину соединения, а в это время в другой ветви процесс достигает соединителя, то возможна неоднозначность выполнения алгоритма. Если это недопустимо, то должны быть предусмотрены меры устранения такой неоднозначности. Для этой цели выполняют коррекцию алгоритма таким образом, чтобы при достижении алгоритмическим процессом в любой ветви соединителя он распространялся бы за соединитель с одновременной ликвидацией алгоритмических процессов во всех параллельных ветвях. Это обеспечивается коррекцией описания частных событий во всех параллельных ветвях с учетом того, что все частные события в любой *i*-й ветви должны быть не совместимы с событиями, отмечающими окончание алгоритмического процесса во всех других ветвях. В общем случае коррекцию частных событий в любой *i*-й ветви можно представить таким образом:

$$S_\alpha^i(t+1) = S_\alpha^i(0) \cdot \overline{S}_в , \qquad (4.65)$$

где $S_\alpha^i(0)$ – сокращенное обозначение события S_α^i без учета коррекции.

Соединитель комбинированный *J*(&,v).

Для комбинированного соединителя функция $S_в$ имеет вид

$$S_в = \mathrm{V}\, \tilde{S}_k^1 \cdots \tilde{S}_k^i \cdots \overline{S}_k^n . \qquad (4.66)$$

Из (4.66) следует, что $S_в$ есть дизъюнктивная нормальная форма от переменных типа S_k^i.

Для такого типа соединителя однозначность выполнения алгоритма также требует коррекции всех частных событий во всех

параллельных ветвях, как и для дизъюнктивного соединителя. Это означает, что для всех частных событий всех параллельных ветвей правая часть их описания должна быть умножена на $\overline{S}_{\text{в}}$ в соответствии с выражением (4.65).

В тех случаях, когда выход за соединитель $J(\&,\text{v})$ в соответствии с алгоритмом может осуществляться необязательно при одновременном появлении событий типа S_k^i, эти события после их зарождения следует сохранять до тех пор, пока не будет истинным комбинационное событие $S_{\text{в}}$. В связи с этим в описаниях событий типа S_k^i должны быть предусмотрены условия сохранения событий S_k^i в соответствии с выражением (4.63).

4.4.3. Построение таблиц переходов и СКУ для цифровых автоматов, заданных на языке ГСАП

Цифровые автоматы, заданные на языке ГСАП, являются недетерминированными, так как отдельные параллельные ветви алгоритма управления имеют общую вершину, от которой выполняется разветвление алгоритмического процесса. В связи с этим прямая таблица переходов (ПТП), соответствующая такому автомату, также будет недетерминированной, хотя она и будет представлять собой совокупность нескольких детерминированных ПТП, к числу которых относятся: детерминированные ПТП, соответствующие отдельным параллельным ветвям исходной ГСАП с общей начальной вершиной, и детерминированная ПТП, которая соответствует последовательной части алгоритма управления, состоящей из начальной части до разветвительной вершины и конечной части после соединительной вершины.

Недетерминированная ПТП для автомата, заданного на языке ГСАП, будет строиться в два этапа. Сначала строят независимо друг от друга ПТП для отдельных параллельных ветвей и последовательной части алгоритма управления так же, как и для обычных ГСА, а затем выполняют необходимую коррекцию частных событий, реализуемых алгоритмом управления. Коррекция част-

ных событий выполняется с учетом условий выхода алгоритмического процесса за вершину соединения параллельных ветвей.

Исходя из основных положений языка ГСАП, алгоритм построения детерминированной ПТП для управляющего автомата, заданного на языке ГСАП, будет включать следующие этапы:

1. Построение недетерминированной ПТП управляющего автомата, состоящей из совокупности отдельных детерминированных ПТП для всех частных событий, реализуемых в параллельных ветвях и последовательной части алгоритма управления. Построение таких ПТП выполняется так же, как и для обычных ГСА, без учета связи между отдельными ветвями.

2. Выполнение коррекции всех частных событий, полученных на предыдущем этапе, в соответствии с заданными условиями выхода алгоритмического процесса за вершину объединения параллельных ветвей. В результате получим общую скорректированную недетерминированную ПТП управляющего автомата.

3. Выполнение детерминизации недетерминированной ПТП, полученной на предыдущем этапе, на основе алгоритма детерминизации, предложенного в гл. 2.

П р и м е р 4.6. Для управляющего автомата, заданного на языке ГСАП (рис. 4.6), построить недетерминированную и детерминированную ПТП и соответствующие им СКУ автомата Мура при условии, что выход алгоритмического процесса за вершину объединения параллельных ветвей осуществляется после окончания алгоритмических процессов во всех ветвях, но необязательно одновременно. Для заданной исходной ГСАП для простоты примера начальная и конечная последовательные части алгоритма управления включают только по одной операторной вершине.

В соответствии с заданными условиями выхода алгоритмического процесса за вершину объединения в исходную ГСАП вводятся промежуточные вершины S_k^1, S_k^2, S_k^3 и соответствующие им события, отмеченные сигналами y_k^1, y_k^2, y_k^3, которые свидетельствуют об окончании алгоритмических процессов в соответствующих параллельных ветвях. Вводится также в первой ветви пустая операторная вершина в петлю из логического условия x_3, отмеченную пустым выходным сигналом y_e.

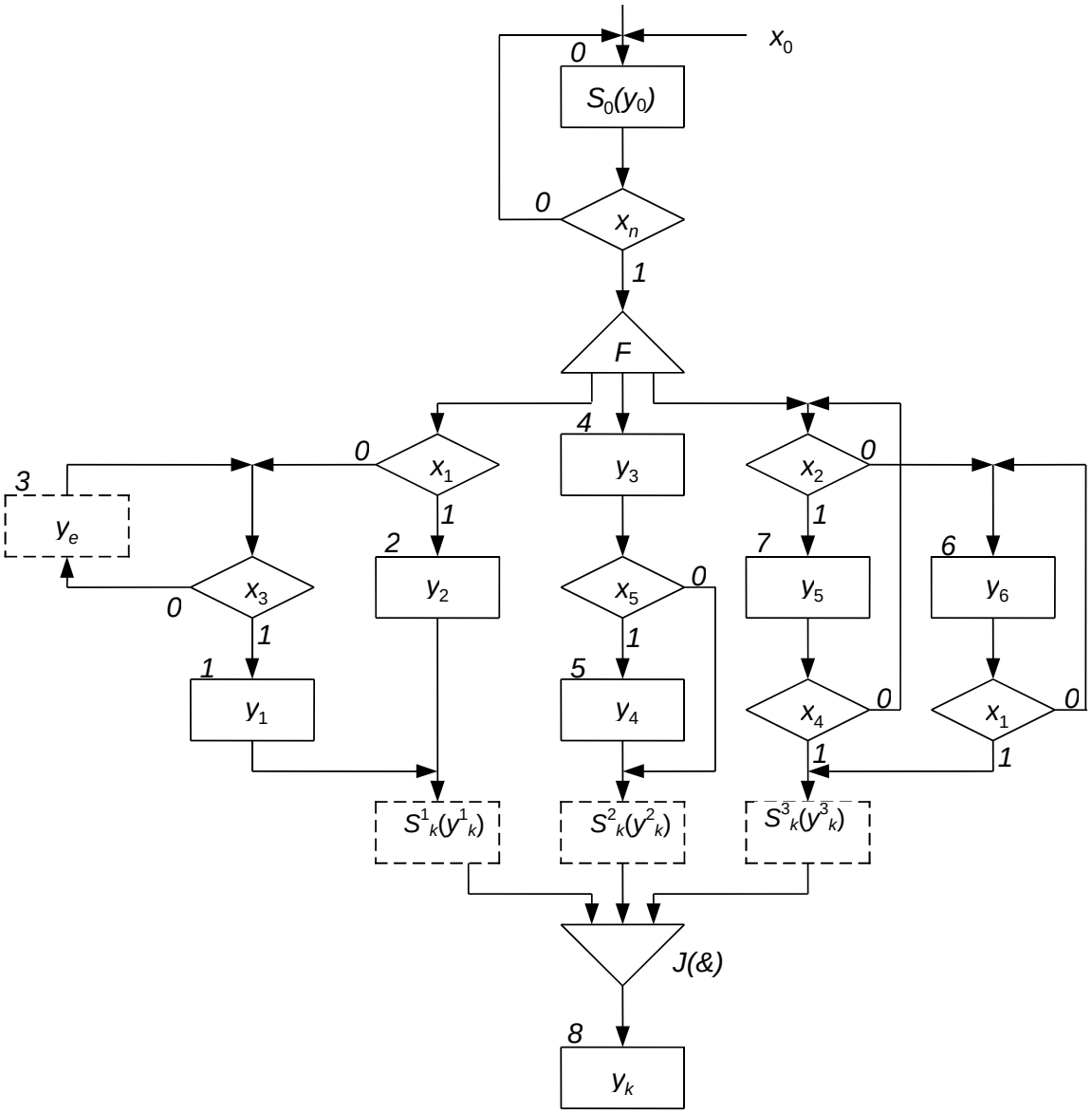


Рис. 4.6. ГСАП с тремя параллельными ветвями и конъюнктивным соединителем (x_n – пусковой сигнал)

Выполнив сквозную нумерацию операторных вершин и учитывая заданное условие объединения ветвей $S_{\text{в}} = S_k^1 S_k^2 S_k^3$, получим недетерминированную ПТП для всех частных событий, реализуемых в управляющем автомате (табл. 4.7).

В табл. 4.7 строки с 1-й по 8-ю представляют собой детерминированные ПТП для трех параллельных ветвей, которые строятся так же, как и ПТП для простых ГСА; строки с 10-й по 12-ю в соответствии с заданным типом соединителя формализуют условия сохранения событий типа S_k^i (коррекция событий типа S_k^i), а

строка 9-я в соответствии с заданным типом соединителя формализует выход алгоритмического процесса за вершину объединения параллельных ветвей.

Таблица 4.7

Шаг алгоритма	Совокупность исходных событий $R_i(t)$	Частный входной сигнал $X_{i,j}(t)$	Событие перехода $S_j(t+1)(Y_j)$	Примечание
1	S_0	$x_n x_1$	$S_2(y_2)$	Для 1-й ветви
		$x_n \overline{x}_1 x_3$	$S_1(y_1)$	
		$x_n \overline{x}_1 \overline{x}_3$	$S_3(y_e)$	
		x_n	$S_4(y_3)$	Для 2-й ветви
		$x_n x_2$	$S_7(y_5)$	Для 3-й ветви
		$x_n \overline{x}_2$	$S_6(y_6)$	
2	S_1	1	$S_k^1(y_k^1)$	Для 1-й ветви
3	S_2	1	$S_k^1(y_k^1)$	
4	S_3	x_3	$S_1(y_1)$	
		$\overline{x}_3$	$S_3(y_e)$	
5	S_4	x_5	$S_5(y_4)$	Для 2-й ветви
		$\overline{x}_5$	$S_k^2(y_k^2)$	
6	S_5	1	$S_k^2(y_k^2)$	
7	S_7	x_4	$S_k^3(y_k^3)$	Для 3-й ветви
		$\overline{x}_4 x_2$	$S_7(y_5)$	
		$\overline{x}_4 \overline{x}_2$	$S_6(y_6)$	
8	S_6	$\overline{x_1}$	$S_6(y_6)$	
		x_1	$S_k^3(y_k^3)$	
9	$S_в = S_k^1 S_k^2 S_k^3$	1	$S_8(y_k)$	Выход за соединитель
10	$S_k^1 \overline{S}_в$	1	$S_k^1(y_k^1)$	Сохранение событий S_k^1, S_k^2, S_k^3
11	$S_k^2 \overline{S}_в$	1	$S_k^2(y_k^2)$	
12	$S_k^3 \overline{S}_в$	1	$S_k^3(y_k^3)$	
13	S_8	1	$S_0(y_0)$	Возврат в исходное событие

Недетерминированная скорректированная СКУ, соответствующая ПТП (см. табл. 4.7), будет иметь вид

$$
\begin{aligned}
&S_1^{y_1}(t+1) = S_0 x_n \overline{x}_1 x_3 \vee S_3 x_3,\\
&S_2^{y_2}(t+1) = S_0 x_n x_1,\\
&S_3^{y_e}(t+1) = S_0 x_n \overline{x}_1 \overline{x}_3 \vee S_3 \overline{x}_3,\\
&S_k^1 y_k^1 (t+1) = (S_1 \vee S_2) \vee S_k^1 \overline{S}_{\text{в}},\\
&S_4^{y_3}(t+1) = S_0 x_n,\\
&S_5^{y_4}(t+1) = S_4 x_5,\\
&S_k^2 y_k^2 (t+1) = S_5 \vee S_4 \overline{x}_5 \vee S_k^2 \overline{S}_{\text{в}},\\
&S_6^{y_6}(t+1) = S_0 x_n \overline{x}_2 \vee S_7 \overline{x}_4 \overline{x}_2 \vee S_6 \overline{x}_1,\\
&S_7^{y_5}(t+1) = S_0 x_n x_2 \vee S_7 \overline{x}_4 x_2,\\
&S_k^3 y_k^3 (t+1) = S_7 x_4 \vee S_6 x_1 \vee S_k^3 \overline{S}_{\text{в}},\\
&S_8^{y_k}(t+1) = S_{\text{в}} = S_k^1 S_k^2 S_k^3,\\
&S_0^{y_0}(t+1) = x_0 \vee S_8 \vee S_0 \overline{x}_n.
\end{aligned}
\tag{4.67}
$$

На основании табл. 4.7 и алгоритма детерминизации (гл. 2) ПТП детерминированного автомата будет представлена табл. 4.8.

Таблица 4.8

Шаг алгоритма	Сочетание частных <u>исходных событий</u> Полное событие $a_m(t)(Y_m)$	Сочетание частных входных сигналов на переходе $X(a_m, a_s)(t)$	Сочетание частных событий в момент времени $(t+1)$ $a_s(t+1)$
1	2	3	4
1	$\frac{S_0}{a_0(y_0)}$	$\overline{x}_n$	S_0 / a_0
		$x_n x_1 x_2$	$S_2 S_4 S_7 / a_1$
		$x_n x_1 \overline{x}_2$	$S_2 S_4 S_6 / a_2$
		$x_n \overline{x}_1 x_2 x_3$	$S_1 S_4 S_7 / a_3$
		$x_n \overline{x}_1 \overline{x}_2 x_3$	$S_1 S_4 S_6 / a_4$
		$x_n \overline{x}_1 x_2 \overline{x}_3$	$S_3 S_4 S_7 / a_5$
		$x_n \overline{x}_1 \overline{x}_2 \overline{x}_3$	$S_3 S_4 S_6 / a_6$

Продолжение табл. 4.8

1	2	3	4
2	$\frac{S_2S_4S_7}{a_1(y_2y_3y_5)}$	x_4x_5	$S_k^1S_5S_k^3 / a_7$
		$x_4\overline{x}_5$	$S_k^1S_k^2S_k^3 / a_8$
		$\overline{x}_4x_2x_5$	$S_k^1S_5S_7 / a_9$
		$\overline{x}_4x_2\overline{x}_5$	$S_k^1S_k^2S_7 / a_{10}$
		$\overline{x}_4\overline{x}_2x_5$	$S_k^1S_5S_6 / a_{11}$
		$\overline{x}_4\overline{x}_2\overline{x}_5$	$S_k^1S_k^2S_6 / a_{12}$
3	$\frac{S_2S_4S_6}{a_2(y_2y_3y_6)}$	$\overline{x}_1\overline{x}_5$	$S_k^1S_k^2S_6 / a_{12}$
		$\overline{x}_1x_5$	$S_k^1S_5S_6 / a_{11}$
		$x_1\overline{x}_5$	$S_k^1S_k^2S_k^3 / a_8$
		x_1x_5	$S_k^1S_5S_k^3 / a_7$
4	$\frac{S_1S_4S_7}{a_3(y_1y_3y_5)}$	x_4x_5	$S_k^1S_5S_k^3 / a_7$
		$x_4\overline{x}_5$	$S_k^1S_k^2S_k^3 / a_8$
		$\overline{x}_4x_2x_5$	$S_k^1S_5S_7 / a_9$
		$\overline{x}_4x_2\overline{x}_5$	$S_k^1S_k^2S_7 / a_{10}$
		$\overline{x}_4\overline{x}_2x_5$	$S_k^1S_5S_6 / a_{11}$
		$\overline{x}_4\overline{x}_2\overline{x}_5$	$S_k^1S_k^2S_6 / a_{12}$
5	$\frac{S_1S_4S_6}{a_4(y_1y_3y_6)}$	$\overline{x}_1\overline{x}_5$	$S_k^1S_k^2S_6 / a_{12}$
		$\overline{x}_1x_5$	$S_k^1S_5S_6 / a_{11}$
		$x_1\overline{x}_5$	$S_k^1S_k^2S_k^3 / a_8$
		x_1x_5	$S_k^1S_5S_k^3 / a_7$
6	$\frac{S_3S_4S_7}{a_5(y_ey_3y_5)}$	$x_4\overline{x}_3\overline{x}_5$	$S_3S_k^2S_k^3 / a_{13}$
		$x_4\overline{x}_3x_5$	$S_3S_5S_k^3 / a_{14}$
		$x_4x_3\overline{x}_5$	$S_1S_k^2S_k^3 / a_{15}$
		$x_4x_3x_5$	$S_1S_5S_k^3 / a_{16}$
		$\overline{x}_4x_2\overline{x}_3\overline{x}_5$	$S_3S_k^2S_7 / a_{17}$
		$\overline{x}_4x_2\overline{x}_3x_5$	$S_3S_5S_7 / a_{18}$
		$\overline{x}_4x_2x_3\overline{x}_5$	$S_1S_k^2S_7 / a_{19}$
		$\overline{x}_4x_2x_3x_5$	$S_1S_5S_7 / a_{20}$
		$\overline{x}_4\overline{x}_2\overline{x}_3\overline{x}_5$	$S_3S_k^2S_6 / a_{21}$
		$\overline{x}_4\overline{x}_2\overline{x}_3x_5$	$S_3S_5S_6 / a_{22}$
		$\overline{x}_4\overline{x}_2x_3\overline{x}_5$	$S_1S_k^2S_6 / a_{23}$
		$\overline{x}_4\overline{x}_2x_3x_5$	$S_1S_5S_6 / a_{24}$

Продолжение табл. 4.8

1	2	3	4
7	$\frac{S_3 S_4 S_6}{a_6(y_e y_3 y_6)}$	$\overline{x}_1\overline{x}_3\overline{x}_5$	$S_3 S_k^2 S_6 / a_{21}$
		$\overline{x}_1\overline{x}_3 x_5$	$S_3 S_5 S_6 / a_{22}$
		$\overline{x}_1 x_3\overline{x}_5$	$S_1 S_k^2 S_6 / a_{23}$
		$\overline{x}_1 x_3 x_5$	$S_1 S_5 S_6 / a_{24}$
		$x_1\overline{x}_3\overline{x}_5$	$S_3 S_k^2 S_k^3 / a_{13}$
		$x_1\overline{x}_3 x_5$	$S_3 S_5 S_k^3 / a_{14}$
		$x_1 x_3\overline{x}_5$	$S_1 S_k^2 S_k^3 / a_{15}$
		$x_1 x_3 x_5$	$S_1 S_5 S_k^3 / a_{16}$
8	$\frac{S_k^1 S_5 S_k^3}{a_7(y_k^1 y_4 y_k^3)}$	1	$S_k^1 S_k^2 S_k^3 / a_8$
9	$\frac{S_k^1 S_k^2 S_k^3}{a_8(y_k^1 y_k^2 y_k^3)}$	1	S_8 / a_{25}
10	$\frac{S_k^1 S_5 S_7}{a_9(y_k^1 y_4 y_5)}$	x_4	$S_k^1 S_k^2 S_k^3 / a_8$
		$\overline{x}_4 x_2$	$S_k^1 S_k^2 S_7 / a_{10}$
		$\overline{x}_4\overline{x}_2$	$S_k^1 S_k^2 S_6 / a_{12}$
11	$\frac{S_k^1 S_k^2 S_7}{a_{10}(y_k^1 y_k^2 y_5)}$	x_4	$S_k^1 S_k^2 S_k^3 / a_8$
		$\overline{x}_4 x_2$	$S_k^1 S_k^2 S_7 / a_{10}$
		$\overline{x}_4\overline{x}_2$	$S_k^1 S_k^2 S_6 / a_{12}$
12	$\frac{S_k^1 S_5 S_6}{a_{11}(y_k^1 y_4 y_6)}$	$\overline{x}_1$	$S_k^1 S_k^2 S_6 / a_{12}$
		x_1	$S_k^1 S_k^2 S_k^3 / a_8$
13	$\frac{S_k^1 S_k^2 S_6}{a_{12}(y_k^1 y_k^2 y_6)}$	$\overline{x}_1$	$S_k^1 S_k^2 S_6 / a_{12}$
		x_1	$S_k^1 S_k^2 S_k^3 / a_8$
14	$\frac{S_3 S_k^2 S_k^3}{a_{13}(y_e y_k^2 y_6)}$	x_3	$S_1 S_k^2 S_k^3 / a_{15}$
		$\overline{x}_3$	$S_3 S_k^2 S_k^3 / a_{13}$
15	$\frac{S_3 S_5 S_k^3}{a_{14}(y_e y_4 y_k^3)}$	x_3	$S_1 S_k^2 S_k^3 / a_{15}$
		$\overline{x}_3$	$S_3 S_k^2 S_k^3 / a_{13}$
16	$\frac{S_1 S_k^2 S_k^3}{a_{15}(y_1 y_k^2 y_k^3)}$	1	$S_k^1 S_k^2 S_k^3 / a_8$
17	$\frac{S_1 S_5 S_k^3}{a_{16}(y_1 y_4 y_k^3)}$	1	$S_k^1 S_k^2 S_k^3 / a_8$

Окончание табл. 4.8

1	2	3	4
18	$\frac{S_3 S_k^2 S_7}{a_{17}(y_e y_k^2 y_5)}$	$x_4 x_3$	$S_1 S_k^2 S_k^3 / a_{15}$
		$x_4 \overline{x}_3$	$S_3 S_k^2 S_k^3 / a_{13}$
		$\overline{x}_4 x_2 x_3$	$S_1 S_k^2 S_7 / a_{19}$
		$\overline{x}_4 x_2 \overline{x}_3$	$S_3 S_k^2 S_7 / a_{17}$
		$\overline{x}_4 \overline{x}_2 x_3$	$S_1 S_k^2 S_6 / a_{23}$
		$\overline{x}_4 \overline{x}_2 \overline{x}_3$	$S_3 S_k^2 S_6 / a_{21}$
19	$\frac{S_3 S_5 S_7}{a_{18}(y_e y_4 y_5)}$	$x_4 x_3$	$S_1 S_k^2 S_k^3 / a_{15}$
		$x_4 \overline{x}_3$	$S_3 S_k^2 S_k^3 / a_{13}$
		$\overline{x}_4 x_2 x_3$	$S_1 S_k^2 S_7 / a_{19}$
		$\overline{x}_4 x_2 \overline{x}_3$	$S_3 S_k^2 S_7 / a_{17}$
		$\overline{x}_4 \overline{x}_2 x_3$	$S_1 S_k^2 S_6 / a_{23}$
		$\overline{x}_4 \overline{x}_2 \overline{x}_3$	$S_3 S_k^2 S_6 / a_{21}$
20	$\frac{S_1 S_k^2 S_7}{a_{19}(y_e y_k^2 y_5)}$	x_4	$S_k^1 S_k^2 S_k^3 / a_8$
		$\overline{x}_4 x_2$	$S_k^1 S_k^2 S_7 / a_{10}$
		$\overline{x}_4 \overline{x}_2$	$S_k^1 S_k^2 S_6 / a_{12}$
21	$\frac{S_1 S_5 S_7}{a_{20}(y_1 y_4 y_5)}$	x_4	$S_k^1 S_k^2 S_k^3 / a_8$
		$\overline{x}_4 x_2$	$S_k^1 S_k^2 S_7 / a_{10}$
		$\overline{x}_4 \overline{x}_2$	$S_k^1 S_k^2 S_6 / a_{12}$
22	$\frac{S_3 S_k^2 S_6}{a_{21}(y_e y_k^2 y_6)}$	$\overline{x}_1 \overline{x}_3$	$S_3 S_k^2 S_6 / a_{21}$
		$\overline{x}_1 x_3$	$S_1 S_k^2 S_6 / a_{23}$
		$x_1 \overline{x}_3$	$S_3 S_k^2 S_k^3 / a_{13}$
		$x_1 x_3$	$S_1 S_k^2 S_k^3 / a_{15}$
23	$\frac{S_3 S_5 S_6}{a_{22}(y_e y_4 y_6)}$	$\overline{x}_1 \overline{x}_3$	$S_3 S_k^2 S_6 / a_{21}$
		$\overline{x}_1 x_3$	$S_1 S_k^2 S_6 / a_{23}$
		$x_1 \overline{x}_3$	$S_3 S_k^2 S_k^3 / a_{13}$
		$x_1 x_3$	$S_1 S_k^2 S_k^3 / a_{15}$
24	$\frac{S_1 S_k^2 S_6}{a_{23}(y_1 y_k^2 y_6)}$	$\overline{x}_1$	$S_k^1 S_k^2 S_6 / a_{12}$
		x_1	$S_k^1 S_k^2 S_k^3 / a_8$
25	$\frac{S_1 S_5 S_6}{a_{24}(y_1 y_4 y_6)}$	$\overline{x}_1$	$S_k^1 S_k^2 S_6 / a_{12}$
		x_1	$S_k^1 S_k^2 S_k^3 / a_8$
26	$\frac{S_8}{a_{25}(y_k)}$	1	S_0 / a_0

Как видно из ПТП (см. табл. 4.8), каждое состояние детерминированного автомата, кроме начального и заключительного, представлено совокупностью из трех событий, каждое из которых реализуется одной из параллельных ветвей алгоритма управления.

Искомая детерминированная СКУ управляющего автомата Мура, полученная на основании ПТП (см. табл. 4.8), представлена системой уравнений

$$a_0^{y_0}(t+1) = x_0 \vee a_{25} \vee a_0 \overline{x}_n,$$

$$a_1^{y_2 y_3 y_5}(t+1) = a_0 x_n x_1 x_2,$$

$$a_2^{y_2 y_3 y_6}(t+1) = a_0 x_n x_1 \overline{x}_2,$$

$$a_3^{y_1 y_3 y_5}(t+1) = a_0 x_n \overline{x}_1 x_2 x_3,$$

$$a_4^{y_1 y_3 y_6}(t+1) = a_0 x_n \overline{x}_1 \overline{x}_2 x_3,$$

$$a_5^{y_e y_3 y_5}(t+1) = a_0 x_n \overline{x}_1 \overline{x}_2 x_3,$$

$$a_6^{y_e y_3 y_6}(t+1) = a_0 x_n \overline{x}_1 \overline{x}_2 \overline{x}_3,$$

$$a_7^{y_k^1 y_4 y_k^3}(t+1) = (a_1 \vee a_3) x_4 x_5 \vee (a_2 \vee a_4) x_1 x_5,$$

$$a_8^{y_k^1 y_k^2 y_k^3}(t+1) = (a_1 \vee a_3) x_4 \overline{x}_5 \vee (a_2 \vee a_4) x_1 \overline{x}_5 \vee (a_9 \vee a_{10} \vee a_{19} \vee a_{20}) \overline{x}_4 \vee$$
$$\vee (a_{11} \vee a_{21} \vee a_{22} \vee a_{24}) x_1 \vee (a_7 \vee a_{15} \vee a_{16}),$$

$$a_9^{y_k^1 y_4 y_5}(t+1) = (a_1 \vee a_3) x_2 \overline{x}_4 x_5, \qquad (4.68)$$

$$a_{10}^{y_k^1 y_k^2 y_5}(t+1) = (a_1 \vee a_3) x_2 \overline{x}_4 \overline{x}_5 \vee (a_2 \vee a_9 \vee a_{10} \vee a_{19} \vee a_{20}) x_2 \overline{x}_4,$$

$$a_{11}^{y_k^1 y_4 y_6}(t+1) = (a_1 \vee a_3) \overline{x}_2 \overline{x}_4 x_5 \vee (a_2 \vee a_4) \overline{x}_1 x_5,$$

$$a_{12}^{y_k^1 y_k^2 y_6}(t+1) = (a_1 \vee a_3) \overline{x}_2 \overline{x}_4 \overline{x}_5 \vee (a_2 \vee a_4) \overline{x}_1 \overline{x}_5 \vee$$
$$\vee (a_2 \vee a_9 \vee a_{10} \vee a_{19} \vee a_{20}) x_2 \overline{x}_4 \vee (a_{11} \vee a_{12} \vee a_{23} \vee a_{24}) \overline{x}_1,$$

$$a_{13}^{y_e y_k^2 y_k^3}(t+1) = a_5 \overline{x}_3 x_4 \overline{x}_5 \vee a_6 x_1 \overline{x}_3 \overline{x}_5 \vee (a_{13} \vee a_{14}) \overline{x}_3 \vee$$
$$\vee (a_{17} \vee a_{18}) \overline{x}_3 x_4 \vee (a_{21} \vee a_{22}) x_1 \overline{x}_3,$$

$$a_{14}^{y_e y_4 y_k^3}(t+1) = a_5 \overline{x}_3 x_4 x_5 \vee a_6 x_1 \overline{x}_3 x_5,$$

$$a_{15}^{y_1 y_k^2 y_k^3}(t+1) = a_5 x_3 x_4 \overline{x}_5 \vee a_6 x_1 x_3 \overline{x}_5 \vee (a_{13} \vee a_{14}) x_3 \vee$$
$$\vee (a_{17} \vee a_{18}) x_3 x_4 \vee (a_{21} \vee a_{22}) x_1 x_3,$$

$$a_{16}^{y_1 y_4 y_k^3}(t+1) = (a_5 x_4 \vee a_6 x_1) x_3 x_5,$$

$$a_{17}^{y_e y_k^2 y_5}(t+1) = a_5 x_2 \overline{x}_3 \overline{x}_4 \overline{x}_5 \vee (a_{17} \vee a_{18}) x_2 \overline{x}_3 \overline{x}_4,$$

$$a_{18}^{y_e y_4 y_5}(t+1) = a_5 x_2 \overline{x}_3 \overline{x}_4 x_5,$$

$$a_{19}^{y_1 y_k^2 y_5}(t+1) = a_5 x_2 x_3 \overline{x}_4 \overline{x}_5 \vee (a_{17} \vee a_{18}) x_2 x_3 \overline{x}_4,$$

$$a_{20}^{y_1 y_4 y_5}(t+1) = a_5 x_2 x_3 \overline{x}_4 x_5,$$

$$a_{21}^{y_e y_k^2 y_6}(t+1) = a_5 \overline{x}_2 \overline{x}_3 \overline{x}_4 \overline{x}_5 \vee a_6 \overline{x}_1 \overline{x}_3 \overline{x}_5 \vee (a_{17} \vee a_{18}) \overline{x}_2 x_3 \overline{x}_4 \vee (a_{21} \vee a_{22}) \overline{x}_1 x_3,$$

$$a_{22}^{y_e y_4 y_6}(t+1) = a_5 \overline{x}_2 \overline{x}_3 \overline{x}_4 x_5 \vee a_6 \overline{x}_1 \overline{x}_3 x_5,$$

$$a_{23}^{y_1 y_k^1 y_6}(t+1) = a_5 \overline{x}_2 x_3 \overline{x}_4 \overline{x}_5 \vee a_6 \overline{x}_1 x_3 \overline{x}_5 \vee (a_{17} \vee a_{18}) \overline{x}_2 x_3 \overline{x}_4 \vee (a_{21} \vee a_{22}) \overline{x}_1 x_3,$$

$$a_{24}^{y_1 y_4 y_6}(t+1) = a_5 \overline{x}_2 x_3 \overline{x}_4 x_5 \vee a_6 \overline{x}_1 x_3 x_5,$$

$$a_{25}^{y_k}(t+1) = a_8.$$

П р и м е р 4.7. Для управляющего автомата, заданного на языке ГСАП (см. рис. 4.6), построить недетерминированную ПТП и соответствующую ей недетерминированную СКУ автомата Мура при условии, что в качестве соединителя параллельных ветвей используется комбинированный соединитель, для которого $S_{\text{в}} = S_k^1 \vee S_k^2 S_k^3$.

Для данного типа комбинированного соединителя НД ПТП с учетом требований однозначности управляющего алгоритма будет представлена табл. 4.9, которая отличается от НД ПТП предыдущего примера только в следующем: в строках со 2-й по 8-ю все обозначения частных событий в колонке 2 умножаются на $\overline{S}_{\text{в}}$; условия сохранения для событий типа S_k^i предусматриваются только для событий S_k^2 и S_k^3; событие S_k^1 не требуется сохранять после его появления, так как для истинного S_k^1 одновременно будет истинным и $S_{\text{в}}$.

Таблица 4.9

Шаг алго-ритма	Совокупность исходных событий $R_i(t)$	Частный входной сигнал $x_{i,j}(t)$	Событие перехода $S_j(t+1)(y_j)$	Примечание
1	S_0	$\overline{x}_n$	$S_0(y_0)$	Для 1-й ветви
		$x_n x_1$	$S_2(y_2)$	
		$x_n \overline{x}_1 x_3$	$S_1(y_1)$	
		$x_n \overline{x}_1 \overline{x}_3$	$S_3(y_e)$	
		x_n	$S_4(y_3)$	Для 2-й ветви
		$x_n x_2$	$S_7(y_5)$	Для 3-й ветви
		$x_n \overline{x}_2$	$S_6(y_6)$	
2	$S_1(\overline{S}_\text{в})$	1	$S_k^1(y_k^1)$	Для 1-й ветви с учетом коррекции
3	$S_2(\overline{S}_\text{в})$	1	$S_k^1(y_k^1)$	
4	$S_3(\overline{S}_\text{в})$	x_3	$S_1(y_1)$	
		$\overline{x}_3$	$S_3(y_e)$	
5	$S_4(\overline{S}_\text{в})$	x_5	$S_5(y_4)$	Для 2-й ветви с учетом коррекции
		$\overline{x}_5$	$S_k^2(y_k^2)$	
6	$S_5(\overline{S}_\text{в})$	1	$S_k^2(y_k^2)$	
7	$S_7(\overline{S}_\text{в})$	x_4	$S_k^3(y_k^3)$	Для 3-й ветви с учетом коррекции
		$\overline{x}_4 x_2$	$S_7(y_5)$	
		$\overline{x}_4 \overline{x}_2$	$S_6(y_6)$	
8	$S_6(\overline{S}_\text{в})$	$\overline{x}_1$	$S_6(y_6)$	
		x_1	$S_k^3(y_k^3)$	
9	$S_\text{в} = S_k^1 \vee S_k^2 S_k^3$	1	$S_8(y_k)$	Выход за соединитель
10	$S_k^2 \overline{S}_\text{в}$	1	$S_k^2(y_k^2)$	Сохранение событий $S_k^2 S_k^3$
11	$S_k^3 \overline{S}_\text{в}$	1	$S_k^3(y_k^3)$	
12	S_8	1	$S_0(y_0)$	Возврат в исходное событие

Искомая НД СКУ управляющего автомата Мура, построенная по НД ПТП (см. табл. 4.9), представлена системой уравнений

$$
\begin{gathered}
S_1^{y_1}(t+1) = S_0 x_n \overline{x}_1 x_3 \vee S_3(\overline{S}_{\text{в}}) x_3, \\
S_2^{y_2}(t+1) = S_0 x_n x_1, \\
S_3^{y_e}(t+1) = S_0 x_n \overline{x}_1 \overline{x}_3 \vee S_3(\overline{S}_{\text{в}}) \overline{x}_3, \\
S_k^1 y_k^1 (t+1) = (S_1 \vee S_2)(\overline{S}_{\text{в}}), \\
S_4^{y_3}(t+1) = S_0 x_n, \\
S_5^{y_4}(t+1) = S_4 x_5 \overline{S}_{\text{в}}, \qquad (4.69) \\
S_k^2 y_k^2 (t+1) = S_5(\overline{S}_{\text{в}}) \vee S_4(\overline{S}_{\text{в}}) \overline{x}_5 \vee S_k^2(\overline{S}_{\text{в}}), \\
S_6^{y_6}(t+1) = S_0 x_n \overline{x}_2 \vee S_7(\overline{S}_{\text{в}}) \overline{x}_4 \overline{x}_2 \vee S_6(\overline{S}_{\text{в}}) \overline{x}_1, \\
S_7^{y_5}(t+1) = S_0 x_n x_2 \vee S_7(\overline{S}_{\text{в}}) \overline{x}_4 x_2, \\
S_k^3 y_k^3 (t+1) = S_7(\overline{S}_{\text{в}}) x_4 \vee S_6(\overline{S}_{\text{в}}) x_1 \vee S_k^3(\overline{S}_{\text{в}}), \\
S_8^{y_k}(t+1) = S_{\text{в}} = S_k^1 \vee S_k^2 S_k^3, \\
S_0^{y_0}(t+1) = x_0 \vee S_8 \vee S_0 \overline{x}_n .
\end{gathered}
$$

Глава 5
СТРУКТУРНЫЙ СИНТЕЗ СИСТЕМ МИКРОПРОГРАММНОГО УПРАВЛЕНИЯ (МПУ), ЗАДАННЫХ МОДЕЛЬЮ НДА

Если исходный алгоритм работы управляющего автомата (УА) представлен моделью НДА, то после его детерминизации структуру УА можно построить на основе использования известных классических методов структурного синтеза детерминированных конечных автоматов. В соответствии с этим методом выполняют два основных этапа структурного синтеза УА: кодирование внутренних состояний УА, представленных в процессе детерминизации в виде сочетаний частных *S*-событий НДА, и построение функций возбуждения и функций выходов для принятой системы элементов памяти УА. Использование такого классического метода структурного синтеза УА приводит к тому, что исходная информация об алгоритме управления, представленная в компактной форме в виде НД СКУ для *S*-событий, дважды преобразуется: первый раз при выполнении детерминизации и второй раз при кодировании внутренних состояний УА, полученных в результате детерминизации. Такое двойное преобразование исходной НД СКУ приводит к следующим негативным явлениям:

а) «теряется» исходная информация о реализуемом алгоритме управления, представленном в результате преобразований в виде СКУ для *Q*-событий, что затрудняет впоследствии выполнение контрольных и диагностических процедур на различных этапах как при проектировании УА, так и при его изготовлении и эксплуатации;

б) функции возбуждения элементов памяти, представленные в виде СКУ для *Q*-событий, сильно усложняются по сравнению с исходной НД СКУ для *S*-событий. Это объясняется тем обстоятельством, что сложность СКУ для *Q*-событий, полученных после детерминизации исходного алгоритма управления, значительно увеличивается по сравнению со сложностью исходной СКУ для *S*-событий. Например, для УА, заданного ГСАП (см. рис. 4.6), сложность булевых функций, определяющих СКУ для *Q*-событий, примерно в 5 раз выше сложности булевых функций, представляющих исходную СКУ для *S*-событий, если выполнить оценку сложности булевых функций по Квайну.

Учитывая отмеченные выше недостатки использования классического метода синтеза для построения структуры УА, заданного моделью НДА, в данной главе будут рассмотрены другие подходы и методы построения структур УА, заданных моделью НДА, позволяющие в той или иной степени не только избежать отмеченных выше недостатков использования классического метода синтеза структур УА, заданных моделью НДА, но и развить новые более эффективные методы, которые вытекают (следуют) из достоинств представления управляющих алгоритмов моделью НДА. К числу таких методов структурного синтеза УА, заданных моделью НДА, относятся методы, базирующиеся на использовании унитарного способа кодирования частных *S*-событий, входящих в исходную НД СКУ, а также методы, основанные на разбиении всех частных *S*-событий, реализуемых в УА, на группы несовместимых событий, каждая из которых может быть реализована отдельным подавтоматом. Такой подход к методу синтеза структуры УА позволяет реализовать операционные устройства, выполняющие распределенную параллельную обработку информации.

В заключение отметим, что рассматриваемые в данной главе методы преобразования алгоритмов управления, заданных моделью НДА, могут быть использованы не только для построения структуры систем управления преобразованием информации на микропрограммном уровне, но и для построения высокопроизводительных систем управления преобразованием информации для различных распределенных и параллельных вычислительных систем.

5.1. Структурная реализация систем МПУ на основе разбиения частных событий на группы несовместимых событий

5.1.1. Преобразование структуры управляющего алгоритма, представленного моделью НДА, для построения распределенной системы МПУ параллельной обработки

В том случае, если исходный управляющий алгоритм, заданный моделью НДА, может быть разбит на группы несовместимых событий, его обобщенную структуру можно представить на осно-

ве языка ГСАП в виде графа, включающего последовательную и параллельную компоненты. При этом последовательная компонента будет состоять из двух частей: начальной и заключительной (рис. 5.1).

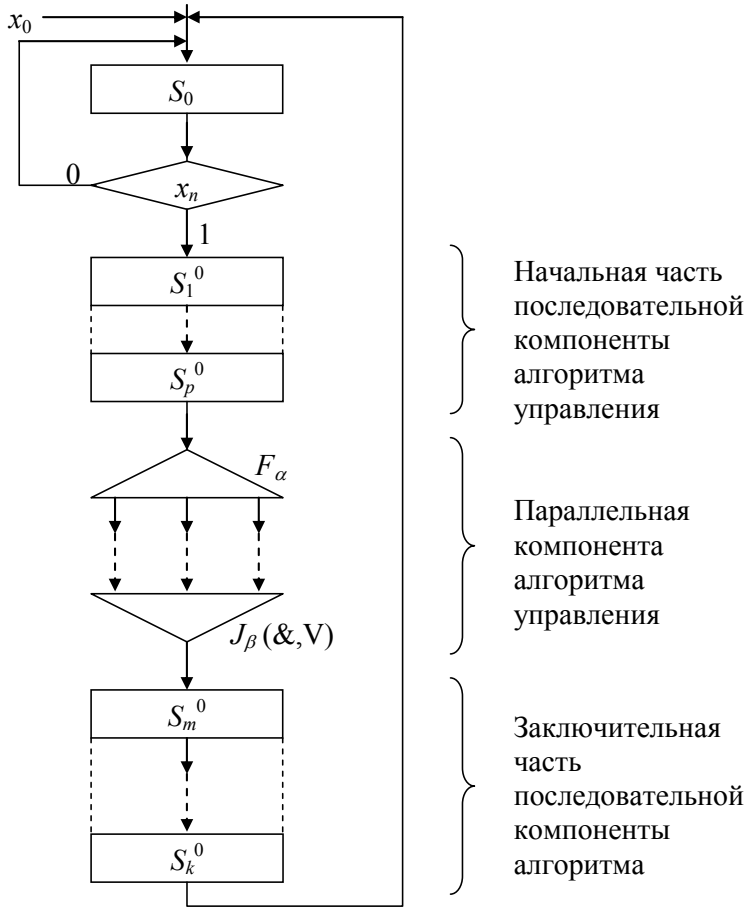


Рис. 5.1. Обобщенная структура ГСАП управляющего алгоритма с параллельными ветвями

На рис. 5.1 введены следующие обозначения:

S_0 – начальное событие;

x_0 – сигнал приведения УА в начальное состояние;

x_n – пусковой сигнал;

S_1^0 – первое событие начальной части последовательной компоненты алгоритма управления;

S_p^0 – событие, определяющее начало распараллеливания алгоритмического процесса;

S_m^0 – событие, свидетельствующее о выходе алгоритмического процесса за вершину объединения параллельных ветвей

(первое событие заключительной части последовательной компоненты алгоритма управления);

S_k^0 – заключительное событие алгоритма управления.

Из графического представления управляющего алгоритма (см. рис. 5.1) следует, что все частные *S*-события, представляющие управляющий алгоритм, в явном виде разбиты на группы несовместимых событий. Поэтому структуру УА можно представить в виде композиции из нескольких подавтоматов (п/А), каждый из которых реализует одну из групп несовместимых событий.

Практически можно отметить два варианта исходных управляющих алгоритмов, представленных моделью НДА, для которых все частные события можно разбить на группы несовместимых событий. Для первого варианта все частные события, представленные в исходном управляющем алгоритме, в явном виде разбиты на группы несовместимых частных событий, что соответствует представлению исходного управляющего алгоритма на языке ГСАП.

Для второго варианта, в отличие от первого, частные события, представленные в исходном управляющем алгоритме в явном виде, не разбиты на группы несовместимых событий, и для того, чтобы разбить их на такие группы, необходимо выполнить над моделью НДА управляющего алгоритма некоторые процедуры. К числу таких процедур относятся: детерминизация НДА, представляющего исходный управляющий алгоритм, и построение матрицы совместимости частных событий и матрицы включения, на основе которых могут быть получены группы несовместимых частных событий. Такие процедуры над моделью НДА управляющего алгоритма будут рассмотрены в последующем разделе.

В данном разделе рассмотрим методику преобразования исходного управляющего алгоритма, представленного в виде графа рис. 5.1, в соответствии с которой структуру УА можно представить в виде композиции из нескольких п/А, каждый из которых реализует одну из групп несовместимых частных событий.

В дальнейшем будем рассматривать структуру УА, состоящую из *n* параллельно работающих п/А, названных в монографии рабочими п/А, каждый из которых реализует одну из параллельных ветвей алгоритма управления, и одного п/А, названного главным п/А, который реализует последовательную компоненту алгоритма управления и осуществляет взаимодействие его параллельных ветвей.

В связи с тем, что параллельные ветви алгоритма управления реализуются отдельными п/А независимо друг от друга, то в каждую *i*-ю ветвь алгоритма управления вводится дополнительная операторная вершина (событие S_0^i), которая будет играть роль начальной вершины в *i*-й ветви. Тогда ГСА для любого *i*-го рабочего п/А будет иметь вид (рис. 5.2).

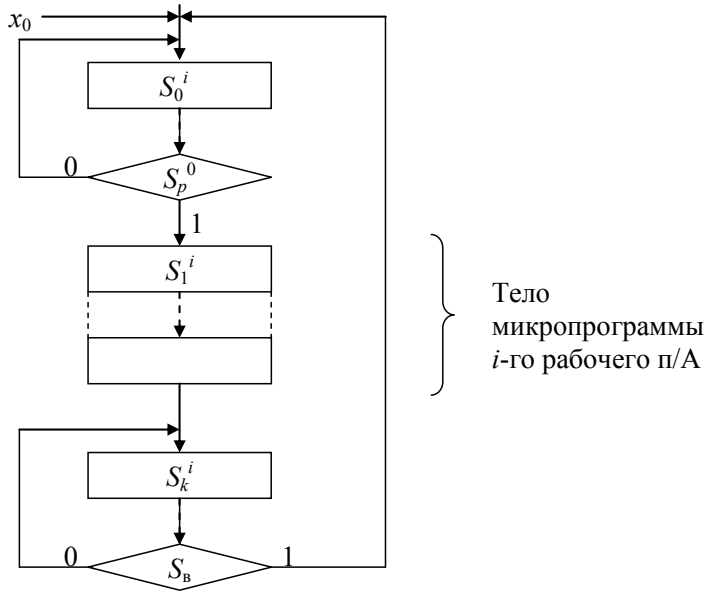


Рис. 5.2. ГСА *i*-го рабочего п/А, реализующего *i*-ю ветвь алгоритма управления

Основные события, реализуемые *i*-м рабочим п/А, определяющие начало и конец работы алгоритма управления для *i*-й ветви, определятся следующим образом:

$$\begin{aligned} S_0^i(t+1) &= x_0 \vee S_k^i S_в \vee S_0^i \overline{S}_p^0, \\ S_1^i(t+1) &= (S_0^i S_p^0) S_{1,з}^i \vee S_1^i S_{1,с}^i, \\ S_k^i(t+1) &= S_{k,з}^i \vee S_k^i \overline{S}_в, \end{aligned} \qquad (5.1)$$

где S_k^i – заключительное событие для *i*-го рабочего п/А; $S_в$ – комбинационное событие, определяющее условие выхода алгоритмического процесса за вершину объединения параллельных ветвей; S_1^i – первое событие микропрограммы работы *i*-го п/А; $S_{1,з}^i$ и $S_{1,с}^i$ – события, определяющие зарождение и сохранение события S_1^i, соответственно; $S_{k,з}^i$ – событие, определяющее зарождение события S_k^i.

ГСА для главного п/А, реализующего последовательную часть алгоритма управления, будет иметь следующий вид (рис. 5.3).

Рис. 5.3. ГСА главного п/А, реализующего начальную и заключительную части последовательной компоненты алгоритма управления

Основные события, реализуемые главным п/А, определяющие начало и конец работы алгоритма управления для его последовательной компоненты, будут иметь вид

$$\begin{aligned} S_0(t+1) &= x_0 \vee S_k^0 \vee \overline{x}_n S_0, \\ S_1^0(t+1) &= (S_0 x_n) S_{1,з}^0 \vee S_1^0 S_{1,с}^0, \\ S_r^0(t+1) &= (S_p^0 \vee S_r)\overline{S}_в, \\ S_m^0(t+1) &= (S_r S_в) S_{m,з}^0 \vee S_m^0 S_{m,с}^0, \end{aligned} \qquad (5.2)$$

где S_r^0 – событие, символизирующее ожидание условия выхода алгоритмического процесса за вершину объединения параллельных ветвей; $S_{m,з}^0$ и $S_{m,с}^0$ – события, определяющие зарождение и сохранение событий S_m^0, соответственно; y_e – пустой выходной сигнал.

В том случае, когда выполнение функций главного п/А возлагается на один из рабочих п/А, то ГСА для такого п/А примет следующий вид (рис. 5.4).

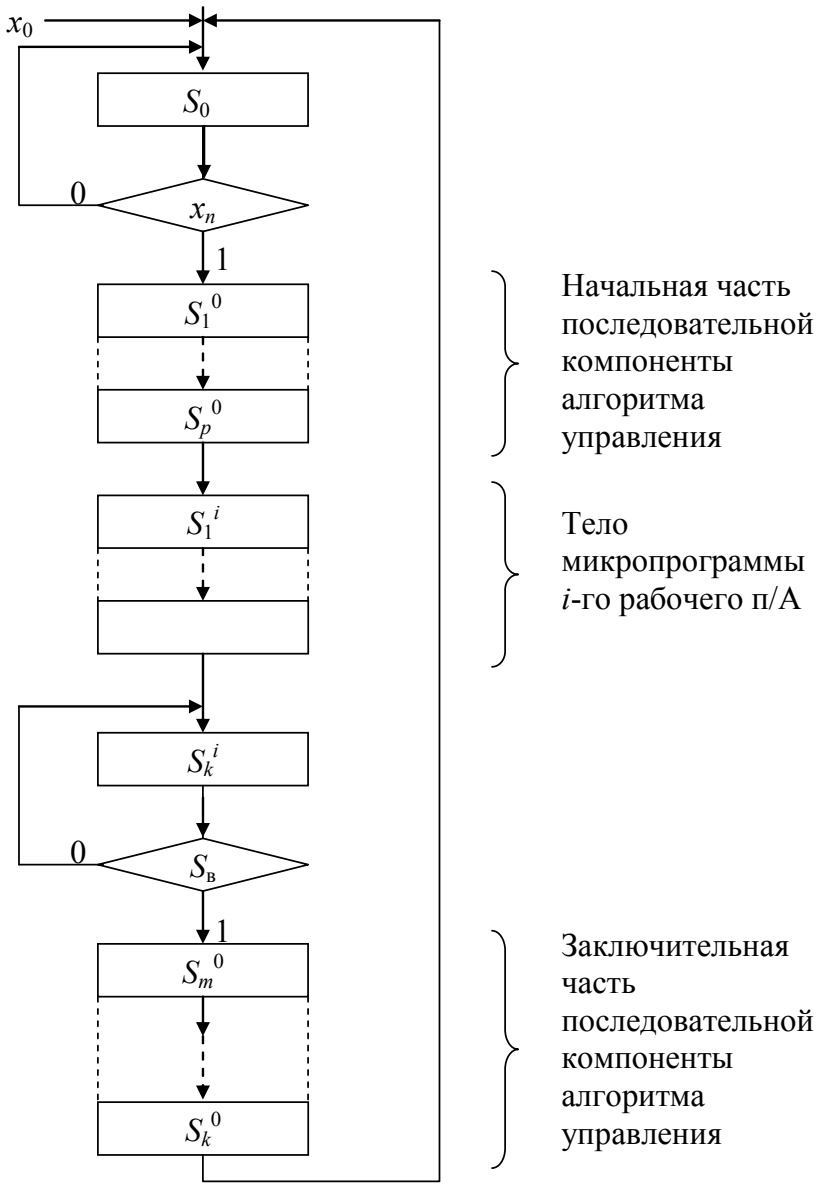


Рис. 5.4. ГСА рабочего *i*-го п/А,
выполняющего функции главного п/А

Построив по исходной НД СКУ и по ГСА (см. рис. 5.2, 5.3, 5.4) скорректированные СКУ для *S*-событий для всех п/А, каждый из которых реализует одну из групп несовместимых событий, представляющих исходный управляющий алгоритм, можно приступить к непосредственно структурному синтезу п/А. Пример реализации таких подавтоматов системы микропрограммного управления (МПУ) будет рассмотрен в последующем разделе 5.2.

5.1.2. Методика разбиения частных событий на группы несовместимых событий для управляющего алгоритма, заданного моделью НДА

В том случае, если из исходного управляющего алгоритма, заданного моделью НДА, не следует в явном виде разбиение частных событий, представляющих управляющий алгоритм, на группы несовместимых событий и требуется преобразовать такой алгоритм к виду, соответствующему обобщенной структуре с параллельными ветвями (см. рис. 5.1), то для решения этой задачи необходимо использовать специальные процедуры, позволяющие разбить частные события на группы несовместимых событий. К числу таких процедур относятся процедура детерминизации исходного управляющего алгоритма и процедура построения матрицы включения частных событий в группы несовместимых событий, которая строится на основе использования вспомогательной матрицы совместимости частных *S*-событий.

Как следует из гл. 2, в результате детерминизации исходного управляющего алгоритма, представленного в виде НД СКУ для частных *S*-событий, получают все полные *a*-события, определенные в виде сочетаний частных *S*-событий, одновременное существование которых для исходного управляющего алгоритма возможно.

По полученным *a*-событиям, учитывая вхождение в них частных *S*-событий, все множество последних разбивают на группы, в каждой из которых все частные события должны быть несовместимыми. Отметим, что под несовместимыми частными событиями понимаются события, которые не входят совместно ни в одно из полных *a*-событий. Число таких групп несовместимых частных событий должно быть не менее максимального числа частных событий, входящих в отдельные полные *a*-события.

При решении задачи распределения частных событий на группы несовместимых событий необходимо руководствоваться двумя условиями:

$$\begin{aligned} & m_i \bigcap m_j = \varnothing, \quad i \neq j, \quad i, j = \overline{1, \mathrm{H}}; \\ & m_1 \bigcup m_2 \bigcup \ldots \bigcup m_{\mathrm{H}} = n, \end{aligned} \tag{5.3}$$

где m_i – подмножество частных событий, входящих в i-ю группу несовместимых частных событий; H – число всех групп несовме-

стимых частных событий; n – число всех частных S-событий, представленных в исходном управляющем алгоритме.

Первое условие (5.3) обеспечивает требование несовместимости частных событий, входящих в любые пары из Н групп. Второе условие обеспечивает включение каждого из всех n частных событий в какую-либо из Н групп.

Такой способ размещения частных событий, входящих в исходный управляющий алгоритм, аналогичен способу размещения кодов микроопераций в полях операционной части микрокоманд, который основан на построении матрицы включения при использовании вспомогательной матрицы совместимости микроопераций [2, 21, 47].

В нашем случае вспомогательная матрица совместимости частных S-событий строится на основе учета вхождения частных событий в полные a-события и будет иметь вид (рис. 5.5), где $\tau_{\alpha,\beta}= 0$, если S_α и S_β не совместимы; $\tau_{\alpha,\beta} =1$, если S_α и S_β совместимы.

	S_1	S_2		S_β		S_n
S_1	0	$\tau_{1,2}$	…	$\tau_{1,\beta}$	…	$\tau_{1,n}$
S_2	$\tau_{2,1}$	0	…	$\tau_{2,\beta}$	…	$\tau_{2,n}$
S_α	$\tau_{\alpha,1}$	$\tau_{\alpha,2}$	…	$\tau_{\alpha,\beta}$	…	$\tau_{\alpha,n}$
S_n	$\tau_{n,1}$	$\tau_{n,2}$	…	$\tau_{n,\beta}$	…	0

Рис. 5.5. Матрица совместимости частных событий

В матрице включения (рис. 5.6) после завершения ее построения каждая i-я строка будет содержать подмножество m_i частных несовместимых событий, размещаемых в одной группе, а каждый столбец матрицы должен включать только одно частное событие, это следует из условий (5.3), где $P_{i,\beta} = 1$, если $S_\beta \subset m_i$; $P_{i,\beta} = 0$, если $S_\beta \not\subset m_i$.

	S_1	S_2		S_β		S_n
m_1	$P_{1,1}$	$P_{1,2}$	…	$P_{1,\beta}$	…	$P_{1,n}$
m_2	$P_{2,1}$	$P_{2,2}$	…	$P_{2,\beta}$	…	$P_{2,n}$
m_i	$P_{i,1}$	$P_{i,2}$	…	$P_{i,\beta}$	…	$P_{i,n}$
m_H	$P_{\text{H},1}$	$P_{\text{H},2}$	…	$P_{\text{H},\beta}$	…	$P_{\text{H},n}$

Рис. 5.6. Матрица включения частных событий
в группы несовместимых событий

Матрица включения строится постепенно по шагам, число которых равно числу всех частных событий n. При этом для облегчения решения задачи на каждом шаге используется вспомогательная матрица совместимости частных событий (см. рис. 5.5).

На каждом шаге построения матрицы включения для очередного частного события, например, инициируемого S_β, отыскивается такое подмножество частных событий, для которого рассматриваемое частное событие S_β ни с одним из частных событий этого подмножества не совместимо. Если такое подмножество частных событий найдено, то в него включается рассматриваемое частное событие S_β; если оно не найдено, то образуется новое подмножество частных событий, в которое включается в качестве первого элемента частное событие S_β.

Для первого шага построения матрицы включения для S_1 имеем только пустое подмножество частных событий, поэтому для S_1 назначают подмножество $m_1 = [S_\beta, 0, \ldots, 0]$.

Для второго шага для S_2 определяют результат операции пересечения m_1 со второй строкой матрицы совместимости, обозначенной буквой S_2. Если $m_1 \cap S_2 = \varnothing$, то S_2 войдет в m_1, в противном случае образуется новое подмножество $m_2 = [0, S_2, 0, ..., 0]$.

Для третьего шага для S_3 определяют результат операции пересечения m_1 и m_2 *с* третьей строкой матрицы совместимости $m_1 \cap S_3$ и $m_2 \cap S_3$. Если $m_1 \cap S_3 = \varnothing$ или $m_2 \cap S_3 = \varnothing$, то S_3 войдет в m_1 (или m_2, соответственно). Если результаты операции пересечения в обоих случаях не равны $\varnothing$, то образуется новое подмножество m_3; если же эти результаты в обоих случаях равны $\varnothing$, то возможно включение S_3 в одно из подмножеств m_1 или m_2. Последним шагом для S_n завершится построение матрицы включения, и может быть коррекция (перераспределение) состава подмножеств несовместимых частных событий m_i, если в процессе построения матрицы включения выявилась возможность отнести очередное частное событие S_β в одно из нескольких подмножеств, m_i.

Коррекция состава подмножеств m_i позволит в дальнейшем при структурном синтезе полнее использовать кодовые группы для частных событий, входящих в подмножество m_i с учетом того, что, с точки зрения минимизации оборудования, наиболее выгодным является разбиение всех частных событий на равные по числу событий в подмножествах m_i.

Проиллюстрируем методику разбиения частных событий на группы несовместимых событий на примере, когда исходный управляющий алгоритм представлен НД СКУ для *S*-событий.

П р и м е р 5.1. Выполнить разбиение частных событий на группы несовместимых событий для управляющего алгоритма, заданного моделью НДА на основе НД СКУ для *S*-событий.

В качестве примера такого управляющего алгоритма для иллюстрации методики разбиения частных событий на группы несовместимых событий будем использовать НД СКУ (4.67) из раздела 4.4.3, условно предполагая, что она получена не по ГСАП, а на основе какого-то другого начального языка, для которого управляющий алгоритм в явном виде не выявляет параллельные ветви алгоритма.

Используя результаты детерминизации исходной НД СКУ, представленные в табл. 4.8 (раздел 4.4.3), построим вспомогательную матрицу совместимости частных событий, которая будет иметь следующий вид (рис. 5.7).

На данном рисунке не представлены события S_0 и S_8, которые для данного исходного управляющего алгоритма не входят ни в одну из совокупностей из трех частных событий, определяющих *a*-событие.

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_k^1	S_k^2	S_k^3
S_1	0	0	0	1	1	1	1	0	1	1
S_2	0	0	0	1	0	1	1	0	0	0
S_3	0	0	0	1	1	1	1	0	1	1
S_4	1	1	1	0	0	1	1	0	0	0
S_5	1	0	1	0	0	1	1	1	0	1
S_6	1	1	1	1	1	0	0	1	1	0
S_7	1	1	1	1	1	0	0	1	1	0
S_k^1	0	0	0	0	1	1	1	0	1	1
S_k^2	1	0	1	0	0	1	1	1	0	1
S_k^3	1	0	1	0	1	0	0	1	1	0

Рис. 5.7. Матрица совместимости частных событий исходного управляющего алгоритма

Построение матрицы включения частных событий в группы несовместимых событий будет содержать следующие шаги:

1. Для частного события S_1 подмножество частных несовместимых событий пусто, поэтому для S_1 назначаем первое подмножество m_1, включающее пока один элемент

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_k^1	S_k^2	S_k^3
$m_1 =$	1	0	0	0	0	0	0	0	0	0

2. Поскольку $S_2 \cap m_1 = \varnothing$ (здесь и в дальнейшем под первой буквой операции пересечения понимается номер строки матрицы совместимости, отмеченный буквой S_2), то S_2 включается в m_1:

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_k^1	S_k^2	S_k^3
$m_1 =$	1	1	0	0	0	0	0	0	0	0

3. Поскольку $S_3 \cap m_1 = \varnothing$, то S_3 включается в m_1:

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_k^1	S_k^2	S_k^3
$m_1 =$	1	1	1	0	0	0	0	0	0	0

4. Поскольку $S_4 \cap m_1 \neq \varnothing$, то S_4 не включается в m_1 и образуется подмножество m_2, в которое войдет S_4 в качестве первого элемента:

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_k^1	S_k^2	S_k^3
$m_1 =$	1	1	1	0	0	0	0	0	0	0
$m_2 =$	0	0	0	1	0	0	0	0	0	0

5. Поскольку $S_5 \cap m_1 \neq \varnothing$, а $S_5 \cap m_2 = \varnothing$, то S_5 не войдет в m_1, а войдет в m_2:

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_k^1	S_k^2	S_k^3
$m_1 =$	1	1	1	0	0	0	0	0	0	0
$m_2 =$	0	0	0	1	1	0	0	0	0	0

6. Поскольку $S_6 \cap m_1 \neq \varnothing$ и $S_6 \cap m_2 \neq \varnothing$, то S_6 не войдет ни в m_1, ни в m_2, поэтому образуется новое подмножество m_3, в которое войдет S_6 в качестве первого элемента:

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_k^1	S_k^2	S_k^3
$m_1 =$	1	1	1	0	0	0	0	0	0	0
$m_2 =$	0	0	0	1	1	0	0	0	0	0
$m_3 =$	0	0	0	0	0	1	0	0	0	0

7. Поскольку $S_7 \cap m_1 \neq \varnothing$ и $S_7 \cap m_2 \neq \varnothing$, $S_7 \cap m_3 = \varnothing$, то S_7 не войдет ни в m_1, ни в m_2, а войдет в m_3:

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_k^1	S_k^2	S_k^3
$m_1 =$	1	1	1	0	0	0	0	0	0	0
$m_2 =$	0	0	0	1	1	0	0	0	0	0
$m_3 =$	0	0	0	0	0	1	1	0	0	0

8. Поскольку $S_k^1 \cap m_1 = \varnothing$ и $S_k^1 \cap m_2 \neq \varnothing$, $S_k^1 \cap m_3 \neq \varnothing$, то S_k^1 войдет в m_1:

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_k^1	S_k^2	S_k^3
$m_1 =$	1	1	1	0	0	0	0	1	0	0
$m_2 =$	0	0	0	1	1	0	0	0	0	0
$m_3 =$	0	0	0	0	0	1	1	0	0	0

9. Поскольку $S_k^2 \cap m_1 \neq \varnothing$, $S_k^2 \cap m_3 \neq \varnothing$, а $S_k^2 \cap m_2 = \varnothing$, то S_k^2 не войдет ни в m_1, ни в m_3, а войдет в m_2:

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_k^1	S_k^2	S_k^3
$m_1 =$	1	1	1	0	0	0	0	1	0	0
$m_2 =$	0	0	0	1	1	0	0	0	1	0
$m_3 =$	0	0	0	0	0	1	1	0	0	0

10. Поскольку $S_k^3 \cap m_1 \neq \varnothing$, $S_k^3 \cap m_2 \neq \varnothing$, а $S_k^3 \cap m_3 = \varnothing$, то S_k^3 не войдет ни в m_1, ни в m_2, а войдет в m_3:

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_k^1	S_k^2	S_k^3
$m_1 =$	1	1	1	0	0	0	0	1	0	0
$m_2 =$	0	0	0	1	1	0	0	0	1	0
$m_3 =$	0	0	0	0	0	1	1	0	0	1

11. Поскольку события S_0 и S_8 не совместимы ни с одним из частных событий исходного управляющего алгоритма (в полные события a_0 и a_{25} включается только по одному событию S_0 и S_8, соответственно), то они могут войти в любое из подмножеств m_1, m_2, m_3 либо образовать новое подмножество m_4.

Таким образом, в результате построения матрицы включения получили следующие группы несовместимых частных событий:

$$\begin{aligned} m_1 &= [S_1, S_2, S_3, S_k^1]; \\ m_2 &= [S_4, S_5, S_k^2]; \\ m_3 &= [S_6, S_7, S_k^3]; \\ m_4 &= [S_0, S_8]. \end{aligned} \qquad (5.4)$$

Полученное в рассмотренном примере разбиение частных событий на группы несовместимых частных событий полностью соответствует размещению этих событий на ГСАП, представленной на рис. 4.6.

5.1.3. Пример построения структуры распределенной системы МПУ по управляющему алгоритму с взаимодействующими параллельными ветвями

Как было показано в разделе 5.1.2, если управляющий алгоритм содержит параллельные взаимодействующие ветви, то для структурной реализации такого алгоритма может быть предложена структура распределенной системы управления, состоящая из нескольких параллельно работающих подавтоматов, каждый из которых реализует одну из групп несовместимых частных событий, и главного подавтомата, реализующего последовательную часть алгоритма управления и взаимодействие рабочих подавтоматов.

В качестве исходного управляющего алгоритма для примера построения структуры распределенной системы МПУ примем управляющий алгоритм, заданный на языке ГСАП (см. рис. 4.6), с тремя параллельными ветвями и с конъюнктивным соединителем, для которого выход алгоритмического процесса за соединитель возможен только в том случае, когда закончится алгоритмический процесс во всех ветвях, но необязательно одновременно. Особенностью данного примера управляющего алгоритма является то, что начальная и заключительная компоненты последовательной части алгоритма управления содержат только по одному событию: S_0 и S_8 (см. ГСАП на рис. 4.6).

Для нашего примера для исходного управляющего алгоритма имеют место 4 группы несовместимых частных S-событий (5.4).

Будем рассматривать структуру МПУ, состоящую из главного п/А и 3 рабочих п/А. Тогда в соответствии с изложенным исходная ГСАП (см. рис. 4.6) будет представлена следующими частными ГСА и соответствующими им СКУ для частных S-событий.

Для главного п/А (рис. 5.8):

$$\begin{aligned} S_0(t+1) &= x_0 \vee S_8 \vee S_0\overline{x_n}, \\ S_9(t+1) &= (S_0 x_n \vee S_9)\overline{S_{\text{в}}}, \\ S_8(t+1) &= S_9 S_{\text{в}}, \\ S_{\text{в}} &= S_k^1 S_k^2 S_k^3. \end{aligned} \qquad (5.5)$$

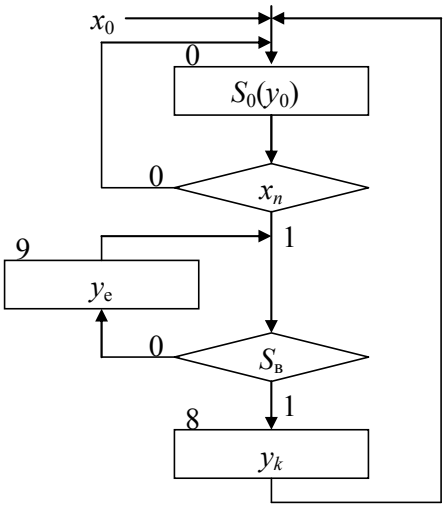


Рис. 5.8. Главный п/А

Для первого п/А (рис. 5.9):

$$
\begin{aligned}
S_0^1(t+1) &= x_0 \vee S_k^{'} S_{\text{в}} \vee S_0^{'}(\overline{x_n S_0}),\\
S_1(t+1) &= S_0^{'}(x_n S_0)\bar{x}_1 x_3 \vee S_3 x_3,\\
S_2(t+1) &= S_0^{'}(x_n S_0)x_1,\\
S_3(t+1) &= S_0^{'}(x_n S_0)\bar{x}_1\ \bar{x}_3 \vee S_3 \bar{x}_3,\\
S_k^{'}(t+1) &= S_1 \vee S_2 \vee S_k^{'}\bar{S}_{\text{в}}.
\end{aligned}
\qquad (5.6)
$$

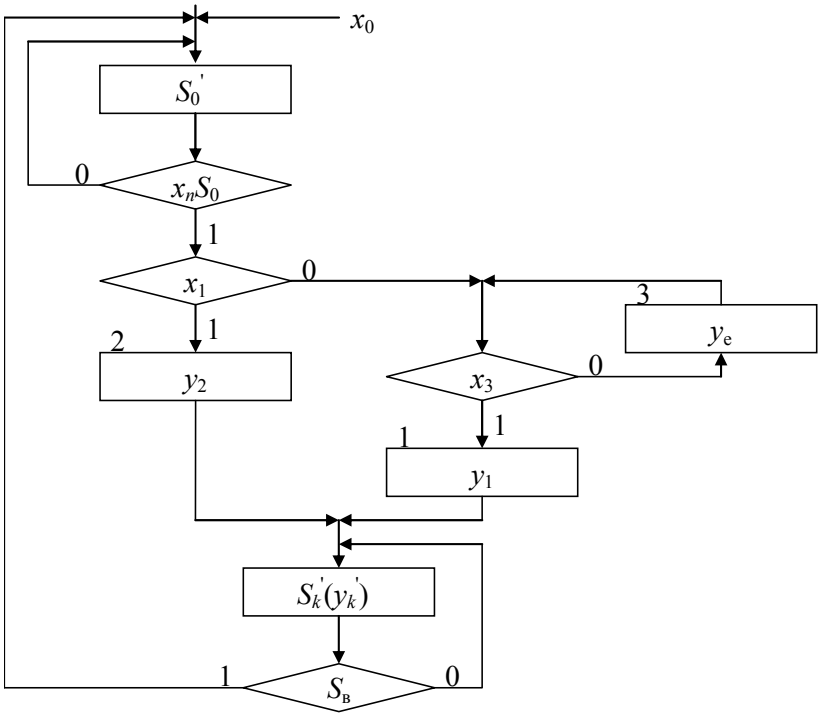


Рис. 5.9. Первый п/А

Для второго п/А (рис. 5.10):

$$
\begin{aligned}
&S_0^2(t+1) = x_0 \vee S_k^2 S_{\text{в}} \vee S_0^2(\overline{x_n S_0}),\\
&S_4(t+1) = S_0^2(x_n S_0),\\
&S_5(t+1) = S_4 x_5,\\
&S_k^2(t+1) = S_4 \overline{x}_5 \vee S_5 \vee S_k^2 \overline{S}_{\text{в}}.
\end{aligned}
\qquad (5.7)
$$

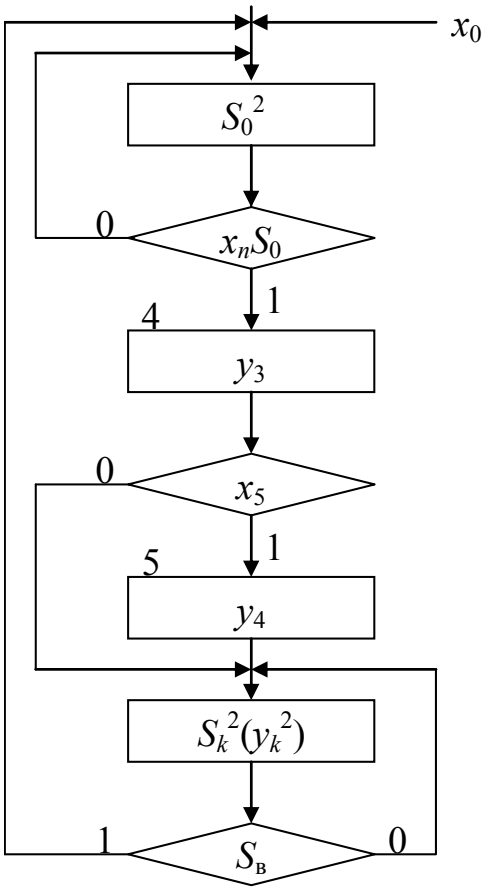


Рис. 5.10. Второй п/А

Для третьего п/А (рис. 5.11):

$$
\begin{aligned}
&S_0^3(t+1) = x_0 \vee S_k^3 S_{\text{в}} \vee S_0^3(\overline{x_n S_0}),\\
&S_6(t+1) = S_0^3(x_n S_0)\overline{x}_2 \vee S_6 \overline{x}_1 \vee S_7 \overline{x}_4 \overline{x}_2,\\
&S_7(t+1) = S_0^3(x_n S_0)x_2 \vee S_7 \overline{x}_4 x_2,\\
&S_k^3(t+1) = S_6 x_1 \vee S_7 x_4 \vee S_k^3 \overline{S}_{\text{в}}.
\end{aligned}
\qquad (5.8)
$$

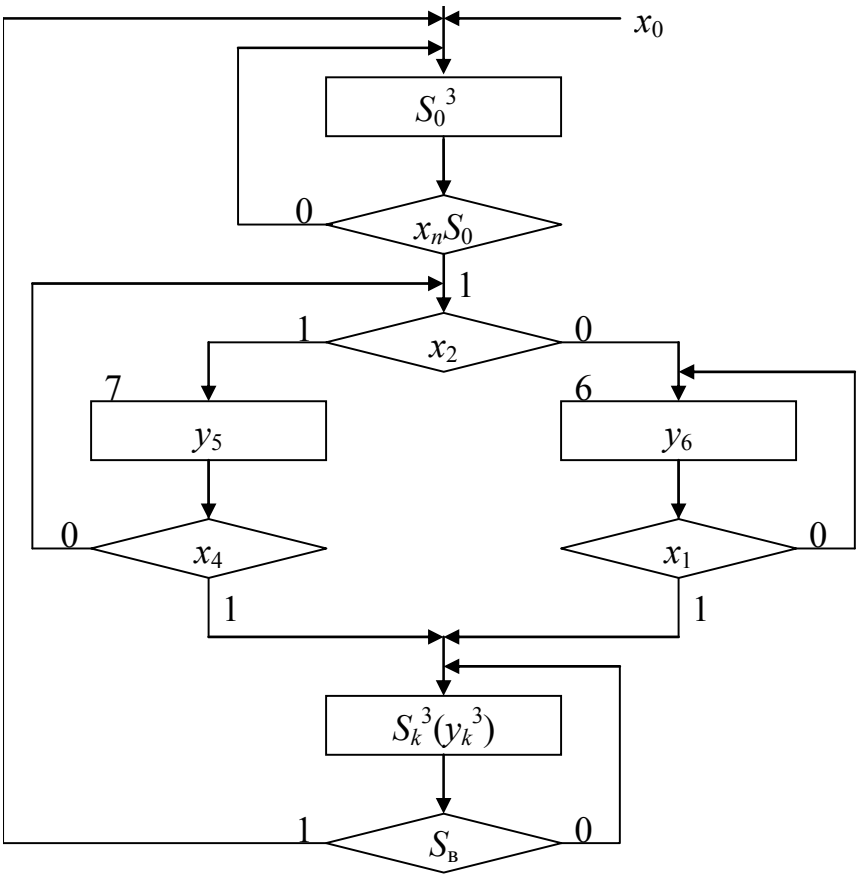


Рис. 5.11. Третий п/А

Для построения структуры всех п/А по их СКУ необходимо выполнить следующие процедуры:

а) кодирование событий (состояний) (рис. 5.12) в каждой из групп m_1, m_2, m_3 и m_4 несовместимых событий, реализуемых одним из подавтоматов.

m_1	$Q_1Q_2Q_3$
S_0^1	0 0 0
S_1	0 0 1
S_2	0 1 0
S_3	0 1 1
S_k^1	1 0 0

m_2	Q_4Q_5
S_0^2	0 0
S_4	0 1
S_5	1 0
S_k^2	1 1

m_3	Q_6Q_7
S_0^3	0 0
S_6	0 1
S_7	1 0
S_k^3	1 1

m_4	Q_8Q_9
S_0^4	0 0
S_8	1 0
S_9	0 1

Рис. 5.12. Кодирование событий

Внутри каждой группы несовместимых событий примем произвольное максимальное кодирование;

б) построение СКУ для Q-событий для каждого п/А.

Будем предполагать, что в качестве элементов памяти приняты Д-триггеры. Тогда по таблице кодирования и по СКУ для каждого из п/А получим СКУ для Q-событий.

$$
\begin{aligned}
&Q_1(t+1) = S_k^1(t+1) = S_1 \vee S_2 \vee S_k^1 \overline{S}_{\text{в}},\\
&Q_2(t+1) = (S_2 \vee S_3)(t+1) = S_0^1(x_n S_0)x_1 \vee S_0^1(x_n S_0)\overline{x}_1\,\overline{x}_3 \vee S_3\overline{x}_3,\\
&Q_3(t+1) = (S_1 \vee S_3)(t+1) = S_0^1(x_n S_0)\overline{x}_1 x_3 \vee S_3 x_3 \vee S_0^1(x_n S_0)\overline{x}_1\ \overline{x}_3 \vee S_3\overline{x}_3,\\
&Q_4(t+1) = (S_5 \vee S_k^2)(t+1) = S_4 x_5 \vee S_4\overline{x}_5 \vee S_5 \vee S_k^2\overline{S}_{\text{в}},\\
&Q_5(t+1) = (S_4 \vee S_k^2)(t+1) = S_0^2(x_n S_0) \vee S_4\overline{x}_5 \vee S_5 \vee S_k^2\overline{S}_{\text{в}},\\
&Q_6(t+1) = (S_6 \vee S_k^3)(t+1) = S_0^3(x_n S_0)\overline{x}_2 \vee S_6\overline{x}_1 \vee S_6 x_1 \vee S_7 x_4 \vee S_k^3\overline{S}_{\text{в}},\\
&Q_7(t+1) = (S_7 \vee S_k^3)(t+1) = S_0^3(x_n S_0)x_2 \vee S_7\overline{x}_4 x_2 \vee S_6 x_1 \vee S_7 x_4 \vee S_k^3\overline{S}_{\text{в}},\\
&Q_8(t+1) = S_8(t+1) = S_9 S_{\text{в}},\\
&Q_9(t+1) = S_9(t+1) = (x_n S_0 \vee S_9)\overline{S}_{\text{в}}.
\end{aligned}
\qquad (5.9)
$$

В полученных уравнениях (5.9) СКУ для Q-событий выполним подстановку в правую часть кодов для всех событий в соответствии с таблицами кодирования. Тогда получим СКУ для Q-событий, в которых правая часть будет зависеть от Q-событий. Далее для иллюстрации процедур будем рассматривать все преобразования только для 1-го п/А. Для всех других п/А все преобразования будут выполняться аналогично. Учитывая отмеченное, получим для 1-го п/А СКУ для Q-событий:

$$
\begin{aligned}
&Q_1(t+1) = \overline{Q}_1\ \overline{Q}_2\ \ Q_3 \vee \overline{Q}_1 Q_2\overline{Q}_3 \vee Q_1\overline{Q}_2\ \overline{Q}_3\,\overline{S}_{\text{в}},\\
&Q_2(t+1) = \overline{Q}_1\ \ \overline{Q}_2\ \overline{Q}_3(x_n S_0)(x_1 \vee \overline{x}_3) \vee \overline{Q}_1\ Q_2\,Q_3\overline{x}_3,\\
&Q_3(t+1) = \overline{Q}_1\ \ \overline{Q}_2\ \overline{Q}_3(x_n S_0)x_1 \vee \overline{Q}_1\ Q_2 Q_3;
\end{aligned}
\qquad (5.10)
$$

в) минимизацию систем уравнений типа (5.10) СКУ для Q-событий для всех п/А с учетом неиспользованных кодовых групп.

Для 1-го п/А в результате минимизации получим следующую систему СКУ для Q-событий:

$$
\begin{aligned}
&Q_1(t+1) = Q_2\overline{Q}_3 \vee Q_1\overline{S}_{\text{в}} \vee \overline{Q}_2 Q_3,\\
&Q_2(t+1) = Q_2 Q_3\overline{x}_3 \vee \overline{Q}_1\,\overline{Q}_2\overline{Q}_3(x_n S_0)(x_1 \vee \overline{x}_3),\\
&Q_3(t+1) = \overline{Q}_1\,\overline{Q}_2(x_n S_0)\overline{x}_1 \vee Q_2 Q_3.
\end{aligned}
\qquad (5.11)
$$

Так как в качестве элементарных автоматов приняты Д-триггеры, то система уравнений (5.11) является функциями возбуждения элементов памяти, по которой нетрудно построить структурную схему для рабочего п/А, реализующую его функции перехода.

В том случае, когда при построении структуры распределенной системы МПУ функции главного п/А возлагают на один из рабочих п/А, тогда строится ГСА и соответствующая ему СКУ для такого совмещенного п/А (рис. 5.13). Если в качестве такого рабочего п/А принять 3-й п/А, то ГСА и СКУ совмещенного п/А будут иметь вид

$$S_0(t+1) = x_0 \vee S_8 \vee S_0\overline{x}_n,$$

$$S_6(t+1) = (S_0 x_n)\overline{x}_2 \vee S_0\,\overline{x_1} \vee S_7\overline{x}_1\overline{x}_2,$$

$$S_7(t+1) = S_7\overline{x}_4 x_2 \vee (S_0 x_n)x_2,$$

$$S_k^3(t+1) = S_6 x_1 \vee S_7 x_4 \vee S_k^3\overline{S}_{\text{в}},$$

$$S_8(t+1) = S_k^3 S_{\text{в}},$$

$$S_{\text{в}}(t+1) = S_k^1 S_k^2 S_k^3.$$

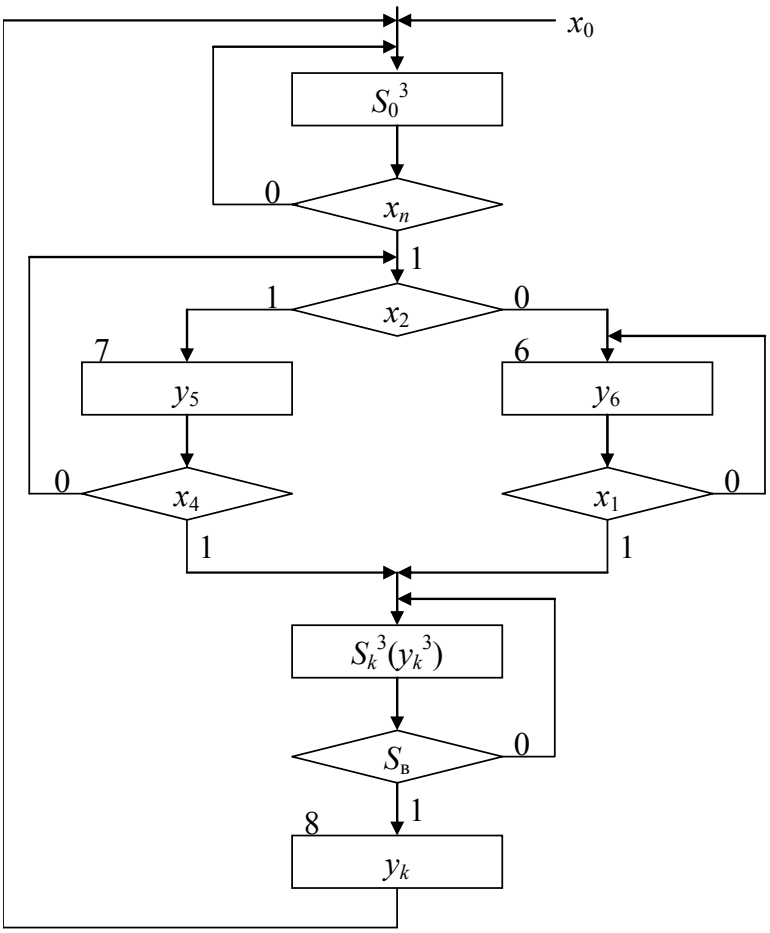


Рис. 5.13. ГСА рабочего п/А, выполняющего функции главного п/А

5.2. Структурная реализация систем МПУ, заданных моделью НДА, на основе использования унитарного кодирования частных событий

Для некоторых вариантов управляющих алгоритмов, представленных моделью НДА, частные события, входящие в состав НД СКУ, не могут быть разложены на группы несовместимых событий, поэтому такой управляющий алгоритм невозможно реализовать на основе структуры, состоящей из параллельно работающих п/А, т.е. не может быть построена управляющая система для распределенной параллельной обработки.

Отличительной особенностью управляющих алгоритмов такого типа является то, что каждое частное событие, представленное в алгоритме, совместимо со всеми другими частными событиями, входящими в той или иной комбинации в полные события, определяемыми при детерминизации управляющего алгоритма. Для таких управляющих алгоритмов, заданных моделью НДА, чтобы избежать недостатков классического метода синтеза структур МПУ, можно предложить методику построения структуры МПУ, базирующуюся на использовании унитарного кодирования частных событий. Особенно целесообразно использование такой методики для построения структуры систем МПУ, для которых в управляющем алгоритме должны параллельно выполняться различные микрооперации и имеется достаточно много повторяющихся комбинаций из них. Такие комбинации микроопераций, выполняемые в одно и то же время, принято называть ***нанокомандой.***

В дальнейшем будем рассматривать два возможных варианта организации систем МПУ на основе использования унитарного кодирования частных событий. Для первого варианта, соответствующего одноуровневой организации системы МПУ, функции переходов и функции выходов системы МПУ реализуются на основе использования комбинационных схем. Для второго варианта, соответствующего двухуровневой организации функции выходов системы МПУ, реализуются на основе использования нанопамяти, в которую записывается все многообразие нанокоманд, предусмотренных в управляющем алгоритме.

5.2.1. Одноуровневая организация структуры системы МПУ для унитарного кодирования частных событий

Если исходный управляющий алгоритм задан недетерминированной СКУ, то один из вариантов одноуровневой организации структур системы МПУ можно представить следующей схемой (рис. 5.14).

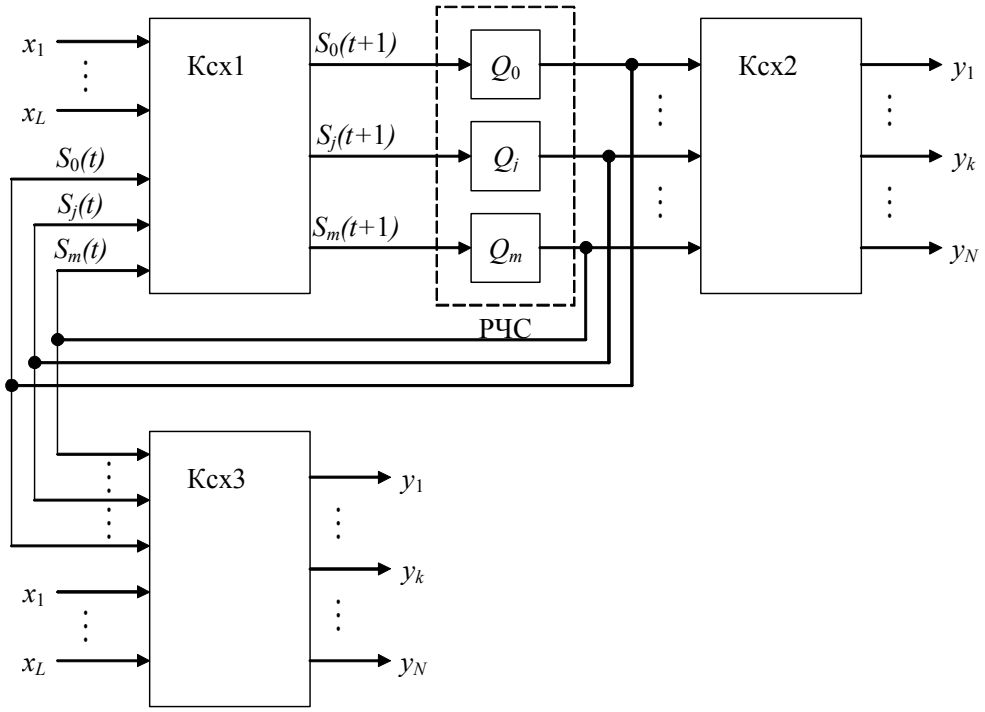


Рис. 5.14. Структурная схема одноуровневой системы МПУ, представленная на основе использования НД СКУ в виде совмещенного *C*-автомата

На рис. 5.14 представлены следующие блоки системы МПУ:

Ксх1 – многовыходная комбинационная схема реализует функции переходов в системе МПУ (последовательность выполнения микрокоманд):

$$S_j^{Y_j}(t+1) = f_j(x_1, x_2, ..., x_L, S_0, S_1, ..., S_m), \quad j = \overline{0, m}; \qquad (5.12)$$

Ксх2 – многовыходная комбинационная схема реализует функции выходов модели автомата Мура:

$$\begin{gathered} y_k(t) = \bigvee S_j^{Y_j}(t), \quad k = \overline{1, N}; \\ (\forall S_j)(Y_j \subset y_k); \end{gathered} \qquad (5.13)$$

Ксх3 – многовыходная комбинационная схема реализует функции выходов модели автомата Мили:

$$y_k(t) = \bigvee S_j^{Y_j}(t+1), \quad k = \overline{1,N}, \tag{5.14}$$

где $[x_1, x_2,\ldots, x_L]$ – элементарные входные сигналы (осведомительные сигналы от операционного автомата (ОА)); $[y_1,\ldots, y_k,\ldots, y_N]$ – элементарные выходные сигналы, инициирующие выполнение микроопераций в ОА; $[S_0, S_1,\ldots, S_m]$ – частные события, реализуемые в алгоритме управления; Y_j – совокупность управляющих сигналов, отмечающих событие S_j; РЧС – регистр частных событий, организованный на двухступенчатых Д-триггерах; для унитарного кодирования событий каждому событию S_j соответствует свой триггер, для которого функции возбуждения имеют вид: $q_j(t) = Q_j(t+1) = S_j(t+1)$.

Отличительной особенностью структуры системы МПУ, представленной на рис. 5.14, по сравнению с вариантом структуры систем МПУ, построенной на основе использования детерминированной СКУ, является ее значительная простота.

В тех случаях, когда исходный управляющий алгоритм будет иметь также такие сложные управляющие конструкции, как обращение к микроподпрограммам, сложные циклы, и простые линейные части, представленная на рис. 5.14 структура системы МПУ может быть использована в качестве составной части общей системы МПУ в качестве блока реализации многоальтернативных переходов.

5.2.2. Двухуровневая организация структуры системы МПУ для унитарного кодирования частных событий

Двухуровневую организацию структуры системы МПУ для унитарного кодирования частных событий целесообразно использовать, когда совокупности параллельно выполняемых микроопераций (нанокоманд) часто повторяются в различных микрокомандах управляющего алгоритма. В этом случае количество адресов нанокоманд может существенно сократиться, что приведет к упрощению общей структуры системы МПУ. Рассмотрим один из возможных вариантов двухуровневой организации системы МПУ,

когда функции переходов, определяющие последовательность выполнения микрокоманд, реализованы с использованием многовыходной комбинационной схемы, которая определяет первый уровень структуры системы МПУ, а второй уровень представлен нанопамятью со схемой формирования адреса нанокоманд. Такая организация структуры системы МПУ представлена на рис. 5.15.

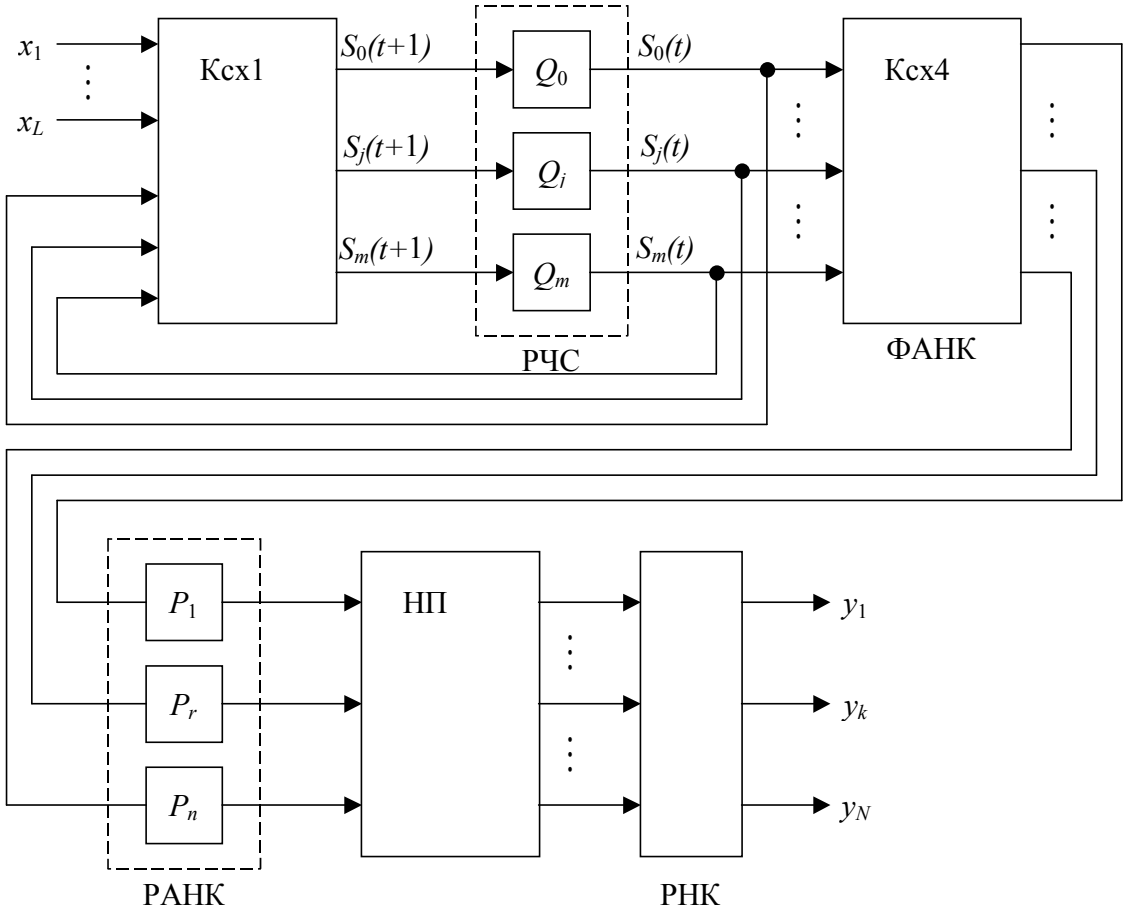


Рис. 5.15. Структурная схема двухуровневой системы МПУ для унитарного кодирования частных событий

Структура системы МПУ (см. рис. 5.15) включает следующие три основных блока: блок определения всех частных событий исходной НД СКУ, состоящий из многовыходной комбинационной схемы (Ксх1) и регистра частных событий (РЧС); блок формирования адреса нанокоманд (ФАНК) с регистром адреса нанокоманд (РАНК) и блок нанопамяти (НП) с выходным регистром нанокоманд (РНК).

Блок формирования всех частных событий строится на основе НД СКУ модели автомата Мура, формализующих последовательность выполнения микрокоманд (функции переходов) управ-

ляющего алгоритма. Регистр частных событий состоит из двухступенчатых Д-триггеров, каждому из которых соответствует свое частное событие.

Блок формирования нанокоманд (ФАНК) реализуется на основе комбинационной схемы (Ксх4), выполняющей функции шифратора. Для построения этого блока используется таблица соответствия, которая формируется по результатам детерминизации исходного управляющего алгоритма. В соответствии с алгоритмом детерминизации определяются все сочетания одновременно существующих частных событий, каждому из которых ставится в соответствие полное событие или вполне определенное состояние системы МПУ, отмеченное совокупностью выходных управляющих сигналов.

Поскольку для автомата Мура существует однозначное соответствие между состояниями автомата и сочетаниями отмечающих их управляющих сигналов, то каждому полному событию можно поставить в соответствие адрес нанопамяти. По нему в нанопамять можно записать код совокупности управляющих сигналов, которыми отмечаются частные события, входящие в полное событие. При этом каждому k-му управляющему сигналу y_k будет соответствовать свой разряд слова нанопамяти. Таким образом, таблица соответствия, используемая для построения блока ФАНК, будет иметь следующий вид (табл. 5.1).

Таблица 5.1

Полное событие	Сочетание частных событий, реализуемых управляющим алгоритмом одновременно	Код адреса нано-команды	Код нано-команды
a_i	$S_0\, S_1\, S_2 \ldots S_j \ldots\, S_{m-1}\, S_m$	$p_1 \ldots p_r \ldots p_n$	$y_1\, y_2 \ldots y_k \ldots y_N$

При построении таблицы соответствия необходимо иметь в виду, что, если некоторые полные события отмечаются одинаковыми совокупностями выходных сигналов, соответствующие им адреса нанопамяти должны иметь одинаковый код.

Блок нанопамяти строится на основе использования ПЗУ, число ячеек которого не должно быть меньше числа всех возможных сочетаний выходных сигналов, определяемых в процессе детерминизации управляющего алгоритма. Количество разрядов в слове нанопамяти для унитарного кодирования определяется числом всех управляющих сигналов исходного алгоритма управ-

ления, каждый из которых инициирует выполнение определенной микрооперации.

Двухуровневая реализация системы МПУ дает большой эффект при высокой частоте появления одинаковых адресов нанокоманд для различных микрокоманд. Такая организация управления позволяет:

– сохранить возможность параллельного выполнения микроопераций и эффективно использовать память ПЗУ, так как исключаются адреса с одинаковыми комбинациями управляющих сигналов;

– значительно упростить структуру блока, реализующего функции переходов управляющего алгоритма, так как основой его построения является НД СКУ, которая представляет самое компактное описание алгоритма управления.

П р и м е р 5.2. Построить блок формирования адреса нанокоманды для системы МПУ, управляющий алгоритм которой представлен недетерминированным направленным графом модели автомата Мура (рис. 5.16).

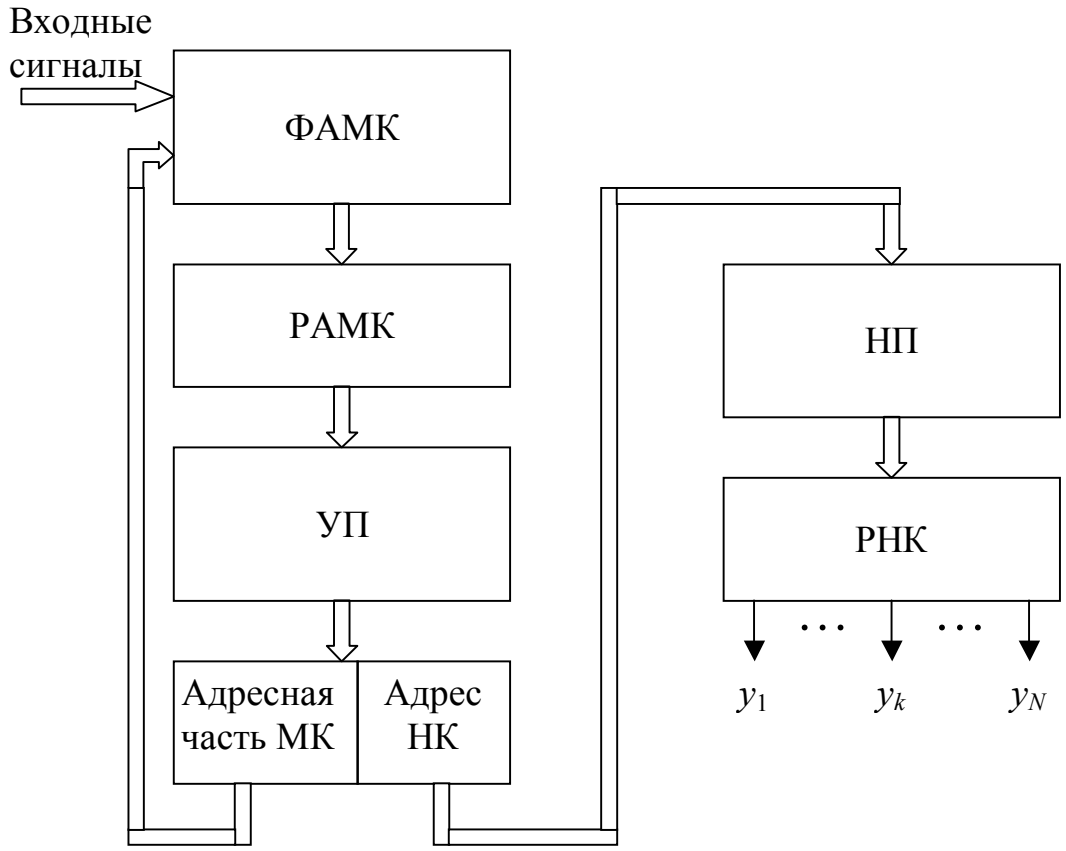


Рис. 5.16. Обобщенная схема двухуровневой системы МПУ

В результате детерминизации исходного управляющего алгоритма получены следующие полные события и соответствующие им совокупности управляющих сигналов:

$$a_0 = S_0(y_0),$$
$$a_1 = S_1S_2(y_1y_2),$$
$$a_2 = S_1S_4(y_1y_2y_4),$$
$$a_3 = S_2S_3S_4(y_2y_3y_4),$$
$$a_4 = S_1S_3S_5(y_1y_3y_4),$$
$$a_5 = S_3(y_3y_4),$$
$$a_6 = S_1S_3(y_1y_3y_4),$$
$$a_7 = S_3S_4(y_2y_3y_4),$$
$$a_8 = S_2S_3(y_2y_3y_4),$$
$$a_9 = S_4(y_2y_4),$$
$$a_{10} = S_3S_4S_5(y_1y_2y_3y_4),$$
$$a_{11} = S_3S_5(y_1y_3y_4),$$
$$a_{12} = S_1S_3S_4(y_1y_2y_3y_4).$$

Таблица соответствия, построенная по результатам детерминизации исходного управляющего алгоритма, будет иметь следующий вид (табл. 5.2).

Таблица 5.2

Полное событие	Сочетание частных событий, реализуемых управляющим алгоритмом одновременно						Код адреса нано-команды			Код нанокоманды				
a_i	S_0	S_1	S_2	S_3	S_4	S_5	p_1	p_2	p_3	y_0	y_1	y_2	y_3	y_4
a_0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
a_1	0	1	1	0	0	0	0	0	1	0	1	1	0	0
a_2	0	1	0	0	1	0	0	1	0	0	1	1	0	1
a_3	0	0	1	1	1	0	0	1	1	0	0	1	1	1
a_4	0	1	0	1	0	1	1	0	0	0	1	0	1	1
a_5	0	0	0	1	0	0	1	0	1	0	0	0	1	1
a_6	0	1	0	1	0	0	1	0	0	0	1	0	1	1
a_7	0	0	0	1	1	0	0	1	1	0	0	1	1	1
a_8	0	0	1	1	0	0	0	1	1	0	0	1	1	1
a_9	0	0	0	0	1	0	1	1	0	0	0	1	0	1
a_{10}	0	0	0	1	1	1	1	1	1	0	1	1	1	1
a_{11}	0	0	0	1	0	1	1	0	0	0	1	0	1	1
a_{12}	0	1	0	1	1	0	1	1	1	0	1	1	1	1

При кодировании адресов НК, как было отмечено выше, необходимо учитывать наличие в таблице истинности полных событий, которые отмечены одинаковыми выходными сигналами. Для нашего примера это три группы полных событий:

$$(a_3, a_7, a_8), (a_4, a_6, a_{11}), (a_{10}, a_{12}).$$

Для каждой группы таких полных событий предусматривается свой индивидуальный код адреса НК. Для произвольного кодирования адресов НК из таблицы соответствия следует, что адрес НК можно сформировать комбинационной схемой, реализующей три не полностью определенные булевые функции: p_1, p_2, p_3.

Для нашего примера функции p_1, p_2 и p_3 не определены на 51-м наборе значений переменных S_0, S_1, …, S_5. Используя диаграммы Вейча для минимизации указанных не полностью определенных функций, получим следующие представления их в ДНФ:

$$\begin{aligned} p_1 &= S_5 \vee S_1 S_3 \vee \overline{S}_1\,\overline{S}_3 S_4 \vee \overline{S}_2 S_3 \overline{S}_4, \\ p_2 &= S_4 \vee S_2 S_3 \vee \overline{S}_5, \\ p_3 &= S_2 \vee \overline{S}_1 S_3 \overline{S}_5 \vee S_3 S_4. \end{aligned} \qquad (5.15)$$

Блок формирования адреса НК может быть построен на ЛЭ, например, ПЛМ, на основе использования многовыходных булевых функций типа (5.15).

В том случае, если исходный управляющий алгоритм содержит, как было отмечено в п. 5.3.2, и другие сложные управляющие конструкции, структуру системы МПУ для таких управляющих алгоритмов целесообразно строить на основе использования типовых блоков микропрограммного управления (БМУ) с двухуровневой реализацией системы МПУ. Для таких блоков БМУ программа работы управляющего алгоритма будет представлена в управляющей памяти (УП), а блок формирования адреса следующей микрокоманды (ФАМК) расширяется за счет организации в нем многоальтернативных переходов. В основе построения блока, обеспечивающего многоальтернативные переходы с высокой скоростью, может быть использован принцип построения блока формирования частных событий, (см. рис. 5.15). Если адреса нанокоманд будут представлены в УП в каждой микрокоманде [2], то обобщенная схема структуры системы МПУ будет иметь следующий вид, (см. рис. 5.16). Подобная структурная организация двухуровневой системы МПУ используется, например, в микропроцессоре МС 68010 фирмы «Моторола» [48].

В том случае, если в пределах неизменяемых микропрограмм имеется однозначное соответствие между адресом микрокоманды и адресом НК, адрес нанокоманды можно сформировать путем преобразования адреса микрокоманды в соответствую-щий адрес НК с помощью комбинационной схемы ФАНК. Такой вари-

ант формирования адреса НК позволит еще более уменьшить длину слов управляющей памяти. Структурная организация системы МПУ в соответствии со схемой рис. 5.16 позволяет при сохранении возможности параллельного выполнения микроопераций вынести основную управляющую память (УП) за пределы кристалла микропроцессора, что, в свою очередь, позволяет сэкономить площадь кристалла до 40 %, на которой могут быть размещены дополнительные схемы. Кроме того, повышается универсальность системы МПУ путем возможной модификации УП. Такая реализация основной УП нашла применение, например, в ЭВМ типа LSI 11 фирмы DEC, Т-88000 фирмы «Тосиба» и др. [48].

5.3. Некоторые операции композиции систем канонических уравнений и их структурная интерпретация

В данном разделе рассматриваются операции над СКУ, которые позволяют выполнить ее преобразование с целью удовлетворения определенным структурным требованиям, предъявляемым к проектируемой системе управления. Среди таких операций рассматриваются операции суммирования, конкатенации и объединения СКУ [11].

Суммирование СКУ. Эта операция сводится к получению из двух или более СКУ общей СКУ с целью построения нового автомата, который будет эквивалентен двум или более параллельно работающим автоматам, построенным по исходным СКУ. Таким образом, операция суммирования в структурной интерпретации эквивалентна замене параллельного соединения нескольких автоматов одним автоматом. При этом, если каждый из исходных автоматов представлен детерминированной СКУ, то в результате операции суммирования будет получена недетерминированная СКУ, так как в новой СКУ будет объединено несколько начальных событий в одно событие.

Для выполнения операции суммирования с последующей детерминизацией объединенной СКУ необходимо:

– переобозначить события (состояния) исходных СКУ, например, введением для событий дополнительного верхнего индекса, указывающего на принадлежность события к СКУ с номером i, где $i = \overline{1,l}$, l – число исходных автоматов;

– образовать новую СКУ путем объединения уравнений всех исходных СКУ в одну группу;

– образовать начальное событие для результирующей объединенной СКУ путем объединения множеств начальных событий исходных СКУ в группу (совокупность), представляющую собой конъюнкцию начальных событий исходных СКУ;

– выполнить детерминизацию объединенной СКУ, если это необходимо для дальнейшего синтеза объединенного автомата.

Для варианта суммирования СКУ, когда исходные автоматы заданы в детерминированной форме, при выполнении операции детерминизации объединенной СКУ будут возникать сочетания из l событий (состояний). В каждое такое сочетание из l событий будет входить только одно из событий (состояний) каждой из исходных детерминированных СКУ, а общее число таких сочетаний, полных событий, объединенной детерминированной СКУ будет не более величины $M = M_1 ... M_i ... M_l$, где M_i – число событий (состояний) i-го автомата. В том случае, если в каждом из автоматов все состояния, в том числе и начальное, достижимы из любого состояния, общее число полных событий объединенной детерминированной СКУ будет равно M. Тогда для рассматриваемого варианта исходных автоматов все сочетания из обозначений событий (состояний), входящих в исходные СКУ, можно заранее перечислить, т.е. в этом случае для решения задачи определения всех возможных сочетаний событий, одновременное существование которых в объединенном автомате возможно, не потребуется использование алгоритма детерминизации, рассмотренного ранее. В то же время описание полных событий детерминированной СКУ объединенного автомата можно получить в этом случае путем выполнения операции конъюнкции правых частей тех уравнений исходных СКУ, формализующих события, сокращенные обозначения которых входят в рассматриваемое полное событие.

В том случае, если в исходных детерминированных СКУ не удовлетворяются условия достижимости состояний, указанные выше, или если исходные СКУ заданы в недетерминированной форме, детерминизацию объединенной СКУ можно выполнить только с использованием алгоритма детерминизации, рассмотренного в гл. 2.

Получение СВФ объединенного автомата выполняется поразному в зависимости от того, одинаковый или разный смысл

имеют выходные сигналы объединяемых автоматов. Если все автоматы работают на один и тот же операционный автомат, то все сигналы имеют одинаковый смысл и СВФ объединенного автомата получают следующим образом. Осуществляют дизъюнкцию правых частей уравнений СВФ объединяемых автоматов для одинаковых выходных сигналов. Далее, если необходимо выразить уравнения СВФ через полные события, процедура их построения будет та же, что была рассмотрена в гл. 2.

Если же действие одинаково обозначенных выходных сигналов в разных автоматах на операционный автомат разное, то необходимо выходные сигналы в исходных уравнениях СВФ переобозначить, например, путем введения верхних индексов.

Конкатенация СКУ. Конкатенация двух СКУ позволяет получить автомат, последовательно реализующий два алгоритма. Для выполнения такой операции необходимо отождествить конечное событие первой СКУ $S_{k,1}$ с начальным событием второй $S_{0,2}$. Под конечным событием здесь понимается событие первой СКУ, от которого происходит переход ко второму алгоритму. В качестве конечного события первой СКУ может быть принято как тупиковое, так и не тупиковое событие.

Если в качестве конечного события первой СКУ принимается тупиковое событие, т.е. такое, которое не встречается в качестве предшествующего события ни в одном из уравнений СКУ, то результирующую СКУ можно получить переобозначением конечного события первой СКУ – обозначением начального события второй СКУ. Кроме того, если вторая СКУ является недетерминированной и имеет несколько начальных событий, то предварительно должна быть выполнена операция стягивания их в одно начальное событие.

Если в качестве конечного события первой СКУ принимается не тупиковое событие, то переход от него к алгоритму, представленному второй СКУ, будет возможен только при определенных значениях входных сигналов. Поэтому в результирующей СКУ необходимо сформировать событие $S_0 = S_{k,1} \vee S_{0,2}$, представляющее собой дизъюнкцию конечного события первой СКУ и начального события второй СКУ. В описании этого события и событий, выводимых из него, необходимо выполнить операцию замены обозначений $S_{k,1}$ и $S_{0,2}$ на обозначение S_0.

Если СКУ первого уровня имеет несколько конечных событий, после каждого из которых реализуется свой алгоритм, описываемый соответствующей СКУ второго уровня, то результирующую СКУ получим последовательным выполнением операций конкатенации СКУ первого уровня с каждой из СКУ второго уровня. В полученной результирующей СКУ будет, как и в исходной СКУ первого уровня, несколько конечных (финальных) событий, которые при необходимости стягиваются в один полюс путем введения нового конечного события, образованного дизъюнкцией правых частей уравнений для конечных событий всех исходных СКУ второго уровня. При таком объединении нескольких СКУ невозможно объединить их общие фрагменты и одинаковые события. Перед объединением все одинаковые события из разных СКУ должны быть переобозначены (например, введением верхнего индекса).

Операция объединения нескольких СКУ с расширением входного алфавита. Эта операция позволяет объединить фрагменты нескольких СКУ, реализуемых одним автоматом, в общую объединенную СКУ с расширением алфавита входных сигналов сигналами, кодирующими выбор той или иной СКУ. Такая операция объединения сохраняет детерминированность СКУ и позволяет выполнить один из возможных алгоритмов, задаваемых исходными СКУ, в зависимости от кода, поступающего на вход автомата (код операции). В этом случае можно не переобозначать одинаковые события из различных СКУ.

Перед объединением все исходные СКУ должны быть приведены к двухполюснику путем стягивания начальных и конечных (финальных) событий. Затем каждое из уравнений исходных СКУ умножается (своей правой частью) на конъюнкцию входных сигналов кода операции (частный входной сигнал кода операции). Эта конъюнкция задает код, при котором будет выполняться соответствующая СКУ. Далее выполняется непосредственно операция объединения СКУ путем дизъюнкции правых частей уравнений СКУ для одинаковых событий и объединения всех уравнений исходных СКУ в одну группу.

Таким образом, в зависимости от структуры объединяемых СКУ рассматриваемая операция во многих случаях позволяет достичь упрощения объединенной СКУ по сравнению с общей структурой необъединенных СКУ.

5.4. Контроль правильности построения функций возбуждения элементов памяти систем МПУ

В связи с тем, что операции детерминизации и кодирования НДА являются обратными друг другу, этим обстоятельством можно воспользоваться для организации контроля правильности построения функций возбуждения элементов памяти систем МПУ. Такая процедура может быть организована на основе выполнения операции детерминизации НД СКУ для Q-событий, полученной в результате кодирования, с последующей минимизацией выражений для исходных событий, представляющих управляющий алгоритм.

Если после детерминизации системы уравнений для Q-событий получим СКУ, которая будет соответствовать исходной СКУ до ее кодирования, то это будет свидетельствовать о правильности системы уравнений для Q-событий, построенной по результатам кодирования исходной СКУ.

Для выполнения операции детерминизации СКУ для Q-событий с целью ее контроля необходимо алгоритм детерминизации, рассмотренный в гл. 2, дополнить пунктами, вытекающими из специфики формирования полных событий (состояний автомата) и возможности отсутствия полноты входных сигналов, вызывающих переходы в Q-события. Это связано с тем, что полные события при их кодировании представляются в виде конъюнкции переменных Q_τ, часть из которых может быть взята с отрицанием.

Таким образом, в алгоритм детерминизации необходимо ввести следующие дополнения и разъяснения:

1. Если исходная система уравнений для Q-событий состоит из R уравнений, то любое полное событие, которое получается в результате детерминизации такой системы, должно быть представлено конъюнкцией (сочетанием) из R букв: $\widetilde{Q}_1 \ldots \widetilde{Q}_\tau \ldots \widetilde{Q}_R$; некоторые из них могут быть как с отрицанием, так и без отрицания. Отсюда следует, что, если на очередном шаге алгоритма детерминизации будут выводимы только некоторые из R событий, сочетание этих событий необходимо дополнить отрицаниями тех Q-событий, которые не вошли в полученное сочетание.

В том случае, если на каком-то шаге алгоритма детерминизации не будет выводимо ни одно из Q-событий, получим полное событие, равное конъюнкции отрицаний из всех Q-событий. Такое событие ранее было названо пустым событием. В нашем случае это

полное событие может быть и не пустым, ему соответствует код $\overline{Q}_1 \ldots \overline{Q}_R$, принятый при кодировании некоторого полного события.

2. Начало работы алгоритма детерминизации определяется начальным полным событием, поэтому его код должен быть зафиксирован в первой строке (первый шаг алгоритма детерминизации) прямой таблицы переходов детерминизации СКУ для Q-событий.

3. Если при определении сочетаний частных входных сигналов на переходе $X(a_m, a_S)$ их совокупность на каком-либо шаге алгоритма детерминизации не удовлетворяет условиям полноты, то эта совокупность входных сигналов должна быть дополнена в соответствии с рекомендациями п. 3 алгоритма детерминизации (см. гл. 2). При этом необходимо иметь в виду, что для дополненных входных сигналов события переходов и соответствующие им состояния переходов (полные события) не будут неопределенными, так как выполняемая операция над СКУ для Q-событий относится к реальному, вполне определенному автомату.

П р и м е р 5.3. Выполнить операцию детерминизации системы уравнений для Q-событий (5.16), полученной в результате кодирования СКУ (3.11).

$$\begin{aligned} Q_1(t+1) &= \overline{Q}_1\ \overline{Q}_2\ \overline{Q}_3 \vee x_1\overline{x}_2(Q_2Q_3), \\ Q_2(t+1) &= \overline{x}_1\, Q_2\, Q_3 \vee x_2\overline{x}_3\, Q_2\, \overline{Q}_3, \\ Q_3(t+1) &= \overline{x}_2\ \overline{Q}_3 \vee \overline{x}_4 Q_1 \vee \overline{Q}_2\ \overline{Q}_3. \end{aligned} \qquad (5.16)$$

В соответствии с системой уравнений (5.16) построим вспомогательную прямую таблицу переходов для Q-событий (табл. 5.3).

Таблица 5.3

Шаг алгоритма	Сочетания непосредственно предшествующих Q-событий в момент времени (t) $R_i = \tilde{Q}_{i,1} \ldots \tilde{Q}_{i,R}(t)$	Частные входные сигналы на переходе $X_{i,j}(t)$	Q-событие перехода в момент времени $(t+1)$ $Q_j(t+1)$
1	$\overline{Q}_1\ \overline{Q}_2\ \overline{Q}_3$	1	Q_1
2	Q_2Q_3	$x_1\overline{x}_2$ $\overline{x}_1$	Q_1 Q_2
3	$Q_2\overline{Q}_3$	$x_2\overline{x}_3$	Q_2
4	$\overline{Q}_3$	$\overline{x}_2$	Q_3
5	Q_1	$\overline{x}_4$	Q_3
6	$\overline{Q}_2\,\overline{Q}_3$	1	Q_3

На основании алгоритма детерминизации, представленного в гл. 2, используя вспомогательную прямую таблицу переходов для Q-событий (см. табл. 5.3) и учитывая требования к формированию полных событий и входных сигналов, получим прямую таблицу переходов детерминированного автомата (табл. 5.4).

Таблица 5.4

Шаг алгоритма	Сочетание всех Q-событий в момент времени (t) / Полное событие $a_m(t)$	Подмножества частных входных сигналов на переходе / Сочетание (конъюнкция) частных входных сигналов на переходе $X(a_m,a_S)(t)$	Сочетание всех Q-событий в момент времени ($t+1$) / Полное событие $a_S(t+1)$
1	Сочетание всех Q-событий, соответствующее начальному полному событию $\overline{Q}_1Q_2Q_3$ a_0	$[x_1\overline{x}_2],[\overline{x}_1]$ $x_1\overline{x}_2$ $(x_1x_2)^*$ $\overline{x}_1$	$Q_1\overline{Q}_2\overline{Q}_3 / a_3$ $\overline{Q}_1\ \overline{Q}_2\ \overline{Q}_3 / a_5$ $\overline{Q}_1Q_2\overline{Q}_3 / a_1$
2	$Q_1\overline{Q}_2\overline{Q}_3$ a_3	$[\overline{x}_4],[\overline{x}_2]$ $\overline{x}_4\ \overline{x}_2$ $(\overline{x}_4x_2)^*$ $(x_4)^*$	$\overline{Q}_1\ \overline{Q}_2Q_3 / a_6$ $\overline{Q}_1\ \overline{Q}_2Q_3 / a_6$ $\overline{Q}_1\ \overline{Q}_2Q_3 / a_6$
3	$\overline{Q_1}\ \overline{Q_2}\ \overline{Q_3}$ a_5	1	$Q_1\ \overline{Q}_2Q_3 / a_2$
4	$\overline{Q_1}Q_2\overline{Q_3}$ a_1	$[\overline{x}_2],[x_2\overline{x}_3]$ $x_2\overline{x}_3$ $(x_2x_3)^*$ $\overline{x}_2$	$\overline{Q}_1Q_2\overline{Q}_3 / a_1$ $\overline{Q}_1\ \overline{Q}_2\ \overline{Q_3} / a_5$ $\overline{Q}_1\ \overline{Q}_2Q_3 / a_6$
5	$Q_1\overline{Q_2}Q_3$ a_2	$[\overline{x}_4]$ $\overline{x}_4$ $(x_4)^*$	$\overline{Q}_1\ \overline{Q}_2Q_3 / a_6$ $\overline{Q}_1\overline{Q}_2\overline{Q}_3 / a_5$

Дадим некоторые пояснения к табл. 5.4. Частные входные сигналы, отмеченные звездочкой, являются дополнительными сигналами, обеспечивающими полноту переходов на соответствующем шаге алгоритма работы автомата, а обозначение сочетаний всех Q-событий принято таким же, что и при кодировании СКУ (3.11). По табл. 5.4 нетрудно построить СКУ, которая будет полностью соответствовать (до обозначений) исходной СКУ (3.11).

Глава 6
ФОРМАЛИЗАЦИЯ АЛГОРИТМОВ УПРАВЛЕНИЯ ВЗАИМОДЕЙСТВУЮЩИМИ ПРОЦЕССАМИ ДЛЯ ОРГАНИЗАЦИИ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ИНФОРМАЦИИ

6.1. Общие сведения о процессах и их взаимодействиях

При рассмотрении вопросов формального описания алгоритмов управления взаимодействующими процессами будем использовать некоторые основные понятия и определения, связанные с механизмом управления процессами и средствами их взаимодействия, которые приняты в теории и практике как системного программного обеспечения при решении вопросов координации взаимодействующих процессов, так и прикладного программного обеспечения при решении задач организации параллельных вычислений.

К числу таких основных понятий и определений относятся понятия:

– процесса и основных типов их взаимодействия;

– разделяемых и критических ресурсов;

– критического интервала (участка) и конфликтных ситуаций, возникающих при использовании общих ресурсов;

– основных базовых функций управления взаимодействием параллельных процессов и механизмов организации взаимодействия процессов для решения задач их координации.

Понятие процесса

В литературе по аппаратному и программному обеспечению вычислительных систем (ВС) существует множество неформальных определений процесса. В данной главе мы используем термин «процесс», полагая, что интуитивно понимаем его значения, связывая понятие процесса с программой, выполняемой на компьютере. В [61] дано такое неформальное определение процесса: «Последовательный процесс (иногда называемый «ЗАДАЧА») есть

работа, производимая последовательным процессом при выполнении программы с ее данными, с логической точки зрения каждый процесс имеет свой собственный процессор и программу. В действительности два разных процесса могут разделять одну и ту же программу или один и тот же процессор. Таким образом, процесс не эквивалентен программе, а также это не то же самое, что процессор; это пара <процессор, программа > при выполнении».

В нашей работе под понятием процесса будем понимать его неформальное определение: как «программа во время выполнения» [63]. В то же время мы не будем вдаваться в такие подразделения видов работ в вычислительных системах, для которых, кроме крупных единиц работы, определяющих процессы (задачи), определяются и более мелкие единицы работ, для обозначения которых используются термины «поток», «нить» [64].

Последовательные процессы (в дальнейшем просто процессы), которые могут существовать одновременно, называются параллельными. В том случае, если работа таких процессов будет зависеть друг от друга, говорят, что они взаимодействуют друг с другом, т.е. должны периодически синхронизироваться и выполнять функции обмена данными.

Основные типы (способы) взаимодействия процессов

В большинстве случаев процессы, определяемые программой, взаимодействуют друг с другом в основном двумя способами:

– путем обмена данными между процессами (связь процессов);

– синхронизацией процессов (согласование выполнения процессов).

Как отмечено в [65], эти два способа взаимодействия процессов не являются взаимоисключающими. Действительно, для обмена данными между процессами необходимо выполнить их синхронизацию, а для синхронизации процессов они обязательно обмениваются данными (хотя бы элементарными сигналами).

Разделяемые и критические ресурсы вычислительных систем

Для современных средств вычислительной техники по сравнению с первыми ВС понятие ресурса стало более универсальным и общим [66]. Ресурсом стали называть всякий объект ВС, кото-

рый может разделяться внутри ВС. К ресурсам ВС относятся как аппаратные (процессоры, память, устройства ввода/вывода и так далее), так и информационные и программные средства.

Для повышения эффективности ВС необходимо, чтобы ресурсы ВС использовались несколькими процессами, т.е. они должны быть разделяемыми. Таким образом, разделяемые ресурсы – это ресурсы, которые могут использоваться несколькими процессами.

Под критическим ресурсом понимается ресурс, который допускает обслуживание только одного пользователя за один раз.

Следует отметить, что, кроме физических ресурсов (память, печать и другие), которые являются критическими, ими могут быть и разделяемые переменные, значения которых могут менять несколько процессов. Например, пусть два процесса А и В разделяют переменную СЧЕТЧИК. Если А и В пытаются увеличить счетчик на единицу одновременно, то окончательное значение переменной СЧЕТЧИК может оказаться неверным. Поэтому СЧЕТЧИК в данном случае следует рассматривать как критический ресурс.

Основные базовые функции управления взаимодействующими процессами

Если несколько процессов хотят пользоваться критическим ресурсом в режиме разделения, им следует синхронизировать свои действия таким образом, чтобы этот ресурс всегда находился в распоряжении только одного из них. Для такого режима работы процессов необходимо, чтобы система управления процессами обеспечивала функцию **«взаимоисключение»** для процессов, которые могут пользоваться критическим ресурсом.

Если один процесс пользуется в данный момент критическим ресурсом, то все остальные процессы, которым нужен этот ресурс, временно получают отказ и должны ждать, пока он не освободится. Для реализации такого режима работы необходимо, чтобы система управления процессами обеспечивала функцию **«ожидание»**.

Таким образом, к основным базовым функциям управления взаимодействующими процессами относятся: функция **«взаимоисключение»** процессов и функция **«ожидание»**.

Критический интервал (или критический участок)

Критический интервал – это группа операций (или событий) процесса, которые непосредственно организуют обращение к разделяемому критическому ресурсу. Критические интервалы должны обладать важнейшим свойством – они должны быть **взаимоисключающими**. Это означает, что в каждый момент времени не более чем один процесс может быть занят выполнением своего критического, относительно некоторого ресурса, участка. В том случае, если критические интервалы для взаимодействующих процессов формализуются с использование систем канонических уравнений, события, обеспечивающие вход процессов в свои критические интервалы, должны быть **несовместимыми**.

Конфликтные ситуации

Конфликтные ситуации могут возникнуть в ВС при монопольном захвате и использовании общих ресурсов для нескольких параллельных процессов. К числу конфликтных ситуаций относятся: взаимоблокировка и взаимоотталкивание.

Взаимоблокировка может возникнуть вследствие конкуренции процессов при захвате разделяемых ресурсов. Простейшим примером возникновения взаимоблокировки является ситуация, когда два взаимодействующих процесса разделяют два ресурса, причем каждому из них для нормального функционирования требуются сразу оба ресурса [67]. В этом случае, если каждый из процессов захватит по одному ресурсу и будет ожидать освобождения второго, захваченного другим процессом, ни один из них не сможет продолжить свою работу. Таким образом, взаимоблокировка проявляется в том, что ни один из взаимодействующих процессов не в состоянии продолжить свою работу.

Взаимоотталкивание может возникнуть, когда один процесс (или группа процессов) не может продолжить свою работу вследствие постоянной занятости требуемых ресурсов, которые попеременно захватываются другими процессами [67]. Ситуация, связанная с взаимоотталкиванием, похожа на ситуацию взаимоблокировки (в обоих случаях конфликтные ситуации возникают как результат конкуренции процессов из-за ресурсов), но в отли-

чие от случая взаимоблокировки блокируются лишь некоторые из взаимодействующих процессов, другие же процессы могут продолжать нормальную работу. Таким образом, взаимоотталкивание проявляется в том, что один или несколько из взаимодействующих процессов не в состоянии продолжать свою работу, тогда как другие процессы продолжают корректно функционировать.

Механизмы взаимодействия (или средства взаимодействия) процессов, используемые для решения задач их координации

Для организации взаимодействия процессов, основными типами которой являются синхронизация и обмен сообщениями, используются следующие механизмы [68]:

– механизм семафорной техники, который обеспечивает взаимоисключающий доступ нескольких процессов к разделяемым ресурсам;

– организация специальных областей памяти (разделяемая память), в которые несколько процессов могут записать и считать информацию, осуществляя, таким образом, взаимодействие;

– механизм обмена сообщениями между процессами на уровне специальных сигналов;

– механизм организации очередей процессов, ожидающих обслуживания;

– механизм, обеспечивающий задержку выполнения процессов до наступления некоторого события.

В заключение отметим, что проблему обмена информацией и синхронизации процессов решают в современных ВС в основном программным способом с использованием различных механизмов. Такое решение вопросов управления межпроцессным взаимодействием имеет следующие недостатки: недостаточная надежность ВС и их защита от возможного несанкционированного доступа [62, 69]; требуется значительный объем оперативной памяти и сравнительно большое время для реализации таких взаимодействий, что уменьшает производительность ВС. Эти недостатки особенно усугубляются, когда многопроцессорные ВС используются для управления технологическими процессами и объектами в реальном масштабе времени.

В связи с этим возникает задача реализации таких функций управления процессами на аппаратном или микропрограммном уровне в виде аппаратной поддержки операционных систем, что, в свою очередь, ставит задачу формального описания алгоритмов управления взаимодействующими процессами [62, 81–83]. Для решения этих задач в последующих разделах монографии предлагается один из возможных вариантов формализации алгоритмов управления взаимодействующими процессами, основанный на использовании методов теории недетерминированных автоматов.

6.2. Формализация простейших базовых структур управления (управляющих конструкций) взаимодействующими процессами

Для формализации алгоритмов управления взаимодействующими процессами будем использовать модели НДА, представленные в виде ГСАП и НД СКУ. В данном разделе рассмотрим формализацию некоторых простейших базовых структур управления, на основе которых можно формализовать более сложные управляющие конструкции, в том числе и базовые функции управления взаимодействующими процессами, к числу которых, как было отмечено ранее, относятся две функции: функция «ожидание» и функция «взаимоисключение».

Функция «ожидание» включает в себя ожидание некоторого события, которое будет причиной (условием) прекращения существования другого события или наоборот будет причиной (условием) появления другого события.

Функция «взаимоисключение» обеспечивает (гарантирует) однозначность параллельных алгоритмов. Формализация этой функции основывается на понятиях несовместимости событий, принадлежащих разным параллельным ветвям (процессам) алгоритма управления процессами и обеспечивающих вход в свои критические интервалы.

Формализация простейших базовых структур управления основывается на использовании представления любого события

S_j модели НДА в виде формулы, входящей в систему рекуррентных предикатных выражений вида (1.12).

Варьируя составом выражений для R_i и R_j, входящих в формулы, определяющие событие R_j, можно влиять на условие зарождения и сохранения события S_j. Учитывая эти обстоятельства, можно выполнять описание событий, которые будут использованы для формализации простейших управляющих конструкций. К числу простейших управляющих конструкций, которые будут использованы в дальнейшем, относятся: управление длительностью сохранения события и его зарождения (первоначального появления), представление несовместимости событий, принадлежащих различным параллельным ветвям, и условий их реализации.

Сохранение события S_j

Если событие S_j после его зарождения необходимо сохранять до наступления или исчезновения другого события, например S_p, то описание события S_j будет иметь вид

$$S_j(t+1) = S_{j,3} \vee S_j \overline{S}_p \text{ или } S_j(t+1) = S_{j,3} \vee S_j S_p, \qquad (6.1)$$

где $S_{j,3}$ – сокращенное обозначение описания условия зарождения события S_j.

Задержка появления (или зарождения) события S_j

Если появление события S_j необходимо задержать до появления или исчезновения некоторого события S_n, то описание события S_j можно представить в следующем виде:

$$S_j(t+1) = S_{j,3} S_n \vee S_j S_{j,\text{с}} \text{ или } S_j(t+1) = S_{j,3} \overline{S}_n \vee S_j S_{j,\text{с}}, \quad (6.2)$$

где $S_{j,\text{с}}$ – сокращенное обозначение описания условий сохранения события S_j.

Описание событий в соответствии с формулами (6.1) и (6.2) является основой для формализации функции «ожидание», являющейся одной из базовых функций управления взаимодействующими процессами.

Формализация несовместимости двух событий S_i и S_j при условии их неодновременного зарождения

Если события S_i и S_j, принадлежащие двум параллельным ветвям алгоритма управления, не совместимы друг с другом и если не допускается их одновременное зарождение, то описание таких событий будет иметь вид

$$\begin{aligned} S_i(t+1) &= S_{i,3}\overline{S}_j \vee S_i S_{i,c}, \\ S_j(t+1) &= S_{j,3}\overline{S}_i \vee S_j S_{j,c}. \end{aligned} \tag{6.3}$$

Из уравнений (6.3) следует, что одновременное появление событий $S_{i,3}$ и $S_{j,3}$, определяющих зарождение событий S_i и S_j, соответственно, приведет к тупиковому результату.

Формализация несовместимости двух событий S_i и S_j, когда события, определяющие их зарождение, могут появляться одновременно

Если для двух не совместимых событий S_i и S_j, принадлежащих двум параллельным ветвям алгоритма управления, условия их зарождения могут появиться в один и тот же момент времени, то описанию таких событий будут соответствовать уравнения (6.3), но с учетом относительного приоритета одного события над другим. Тогда описание таких событий, предотвращающее конфликтную ситуацию, при условии более высокого приоритета, например, события S_i над событием S_j, будет иметь вид:

$$\begin{aligned} S_i(t+1) &= S_{i,3}\overline{S}_j \vee S_i S_{i,c}, \\ S_j(t+1) &= S_{j,3}\overline{S}_i\ \overline{S}_{i,3} \vee S_j S_{j,c}, \end{aligned} \tag{6.4}$$

где введение в первую часть второго уравнения отрицания события $S_{i,3}$, учитывающего условие приоритетности события S_i над событием S_j, обеспечивает отсутствие конфликтной ситуации.

Реализация несовместимых событий, принадлежащих различным параллельным ветвям алгоритма управления

Из системы уравнений (6.4) следует, что из двух событий S_i и S_j в любой момент времени может существовать только одно из них, т.е. такие события могут быть реализованы только последовательно.

Поэтому, если по условиям алгоритма управления требуется, чтобы после реализации алгоритмического процесса (или его части), например, в первой ветви по инициативе события S_i, имеющего более высокий приоритет, был реализован также и алгоритмический процесс (или его часть) во второй ветви по инициативе события S_j, необходимо событие $S_{j,3}$ после его зарождения сохранить. Эти условия учитываются в описании события $S_{j,3}$ следующим образом:

$$S_{j,3}(t+1) = [S_{j,3}(0) \vee S_{j,3}]\overline{S}_j. \qquad (6.5,а)$$

То же самое выполняется и для описания события $S_{i,3}$, если событие S_i может зародиться после события S_j. Тогда имеем

$$S_{i,3}(t+1) = [S_{i,3}(0) \vee S_{i,3}]\overline{S}_i, \qquad (6.5,б)$$

где $S_{i,3}(0)$, $S_{j,3}(0)$ – сокращенное обозначение условий первоначального появления событий $S_{i,3}$ и $S_{j,3}$, соответственно.

Из системы уравнений (6.5,а) и (6.5,б) следует, что события $S_{j,3}$ и $S_{i,3}$ сохраняются до появления событий S_j и S_i, соответственно.

Описания событий в соответствии с формулами (6.4), (6.5,а) и (6.5,б) являются основой для формализации функции **«взаимоисключение»** процессов, являющейся второй базовой функцией управления взаимодействующими процессами.

6.3. Формализация функций взаимоисключения критических интервалов (участков), обеспечивающих доступ к общим разделяемым данным (общему ресурсу) для двух процессов

В том случае, если параллельным процессам в соответствии с алгоритмом их работы требуется обратиться к общим разделяемым данным (критическим ресурсам), программа их работы должна включать специальные участки, содержащие операторы, обеспечивающие доступ к разделяемым данным. Эти участки, называемые критическими, должны обладать важнейшим свойством – они должны быть взаимоисключаемыми. В технической литературе такую группу операторов, обеспечивающих функцию взаимоисключения критических участков, часто называют **примитивами взаимоисключения** [62].

Взаимоисключение критических участков должно обеспечивать выполнение такого алгоритма взаимодействия параллельных процессов, когда один из рассматриваемых процессов выполняет обращение к разделяемым данным, а все остальные процессы, которым требуется обращение к тем же разделяемым данным в то же самое время, должны ждать. Только после завершения рассматриваемым процессом обращения к разделяемым данным будет разрешен доступ к разделяемым данным одному из ожидающих процессов. Какому именно процессу будет разрешен доступ к разделяемым данным, определяется принятыми условиями приоритетности процессов.

Следует обратить внимание, что критический участок не является последовательностью операторов программы, он является только последовательностью **действий**, которые выполняются этими операторами. На это обстоятельство особое внимание было обращено в [70]. Исходя из этого, можно утверждать, что несколько процессов, которые могут иметь место при выполнении одной и той же программы, могут выполнять критические интервалы, базирующиеся на одной и той же последовательности операторов программы.

В данном разделе с методической точки зрения рассматривается взаимодействие параллельных процессов при обращении к разделяемым данным только для двух процессов, а в последую-

щем разделе, базируясь на полученных результатах, делается обобщение для $n > 2$ процессов. Действительно, как мы убедимся в дальнейшем, структура основных уравнений, определяющих входы процессов в свои критические участки, остается одной и той же для любого числа конкурирующих процессов. Отметим, что программная реализация механизма взаимоисключения для двух процессов была впервые предложена голландским математиком Деккером, а для $n > 2$ процессов программную реализацию механизма взаимоисключения впервые предложил Дейкстра [62]. Им же был предложен механизм семафоров для реализации примитивов взаимоисключения.

Для того, чтобы избежать конфликтных ситуаций при взаимодействии параллельных процессов во время организации доступа к разделяемым данным, на критические участки налагаются следующие основные требования [62, 70]:

1. В любой момент времени только один процесс может находиться внутри своего критического интервала.

2. Ни один процесс не может оставаться внутри критического интервала бесконечно долго.

3. Ни один процесс не должен ждать бесконечно долго входа в критический интервал.

Первое требование, предъявляемое к критическим участкам, должно обеспечиваться взаимоисключением (несовместимостью) событий, определяющих как входы процессов в свои критические участки при организации доступа к разделяемым данным, так и нахождение процессов в своих критических участках с учетом принятой дисциплины приоритетности процессов.

Второе требование, предъявляемое к критическим участкам, должно обеспечиваться таким описанием условий сохранения событий, определяющих входы процессов в свои критические интервалы, для которых эти события будут существовать, пока не закончится процедура одноразового обращения к разделяемым данным. Учитывая эти замечания относительно обеспечения первого и второго требований, предъявляемых к критическим участкам, а также результат формализации несовместимых событий (6.4), система уравнений для событий, определяющих входы процессов в свои критические участки при обращении к разделяемым данным, будет иметь следующий вид:

$$S_k^1(t+1) = S_1\overline{S}_k^2 \vee S_k^1\overline{S}_p^1,$$

$$S_k^2(t+1) = S_2\overline{S}_k^1\overline{S}_1 \vee S_k^2\overline{S}_p^2, \qquad (6.6)$$

где S_1 и S_2 – сокращенное обозначение событий, определяющих прием заявок первого и второго процессов на обслуживание для обращения к разделяемым данным (событие пассивного ожидания обращения к разделяемым данным); S_k^1 и S_k^2 – сокращенное обозначение событий, определяющих входы первого и второго процессов в свои критические участки; эти события при их истинности свидетельствуют также о том, что соответствующие процессы находятся внутри своих критических участков; S_p^1 и S_p^2 – сокращенное обозначение событий, обеспечивающих выход первого и второго процессов из критического участка после окончания процедуры обращения к разделяемым данным.

Третье требование, предъявляемое к критическим участкам, должно обеспечиваться такой формализацией событий, определяющих прием заявок на обслуживание при обращении к разделяемым данным, когда заявка какого-либо процесса на обслуживание воспринимается только в том случае, когда данный процесс не находится в своем критическом участке. Тем самым должны исключаться повторные обращения к разделяемым данным для процесса, имеющего наивысший приоритет, несмотря на наличие заявки на обслуживание другого процесса, имеющего более низкий приоритет. Учитывая эти замечания, а также результат формализации несовместимых событий (6.5,а) и (6.5,б), система уравнений для событий, определяющих прием заявок двух процессов на обслуживание при обращении к разделяемым данным, будет иметь следующий вид:

$$S_1(t+1) = (S_{1,3} \vee S_1)\overline{S}_k^1,$$

$$S_2(t+1) = (S_{2,3} \vee S_2)\overline{S}_k^2, \qquad (6.7)$$

где $S_{1,3}$ и $S_{2,3}$ – сокращенное обозначение событий, свидетельствующих о поступлении заявок первого и второго процессов на обслуживание для обращения к разделяемым данным.

В заключение обратим внимание, что полученные в результате формализации функции взаимоисключения критических

участков (интервалов), представленные системами уравнений (6.6) и (6.7), являются основой для формализации алгоритмов управления параллельными процессами при решении многих задач межпроцессного взаимодействия. Решение некоторых из таких задач будет рассмотрено в последующих разделах.

6.4. Формализация алгоритма управления взаимодействующими параллельными процессами при обращении к разделяемым данным (общему ресурсу)

Рассматриваемый алгоритм управления взаимодействующими параллельными процессами является важнейшей составной частью решения многих «классических» задач, посвященных проблемам межпроцессного взаимодействия при использовании разделяемых ресурсов. К числу таких «классических» задач, варианты решения которых будут рассматриваться в последующих разделах, относятся задачи: «производители-потребители», «читатели-писатели» и «обедающие философы».

Решение задачи формализации рассматриваемого алгоритма управления взаимодействующими параллельными процессами при обращении к разделяемым данным основывается на результатах формализации функции взаимоисключения критических участков (раздел 6.3).

Напомним, что основное требование к алгоритму взаимодействия параллельными процессами при организации доступа к разделяемым данным заключается в том, чтобы во время обращения одного процесса к разделяемым данным всем другим процессам это было бы запрещено. Для того, чтобы избежать конфликтных ситуаций при таком взаимодействии параллельных процессов, необходимо, чтобы алгоритм взаимодействия удовлетворял бы требованиям, предъявляемым к критическим участкам (см. раздел 6.3).

Так же, как в разделе 6.3, с методической точки зрения будем сначала рассматривать алгоритм управления только для двух процессов [2, 51, 52].

6.4.1. Формализация алгоритма управления взаимодействующими параллельными процессами при обращении к разделяемым данным для $n = 2$

Представим для наглядности общую структуру рассматриваемого алгоритма управления параллельными процессами в виде графа НДА (рис. 6.1), на котором для простоты опущена некоторая информация о логических условиях для отдельных переходов.

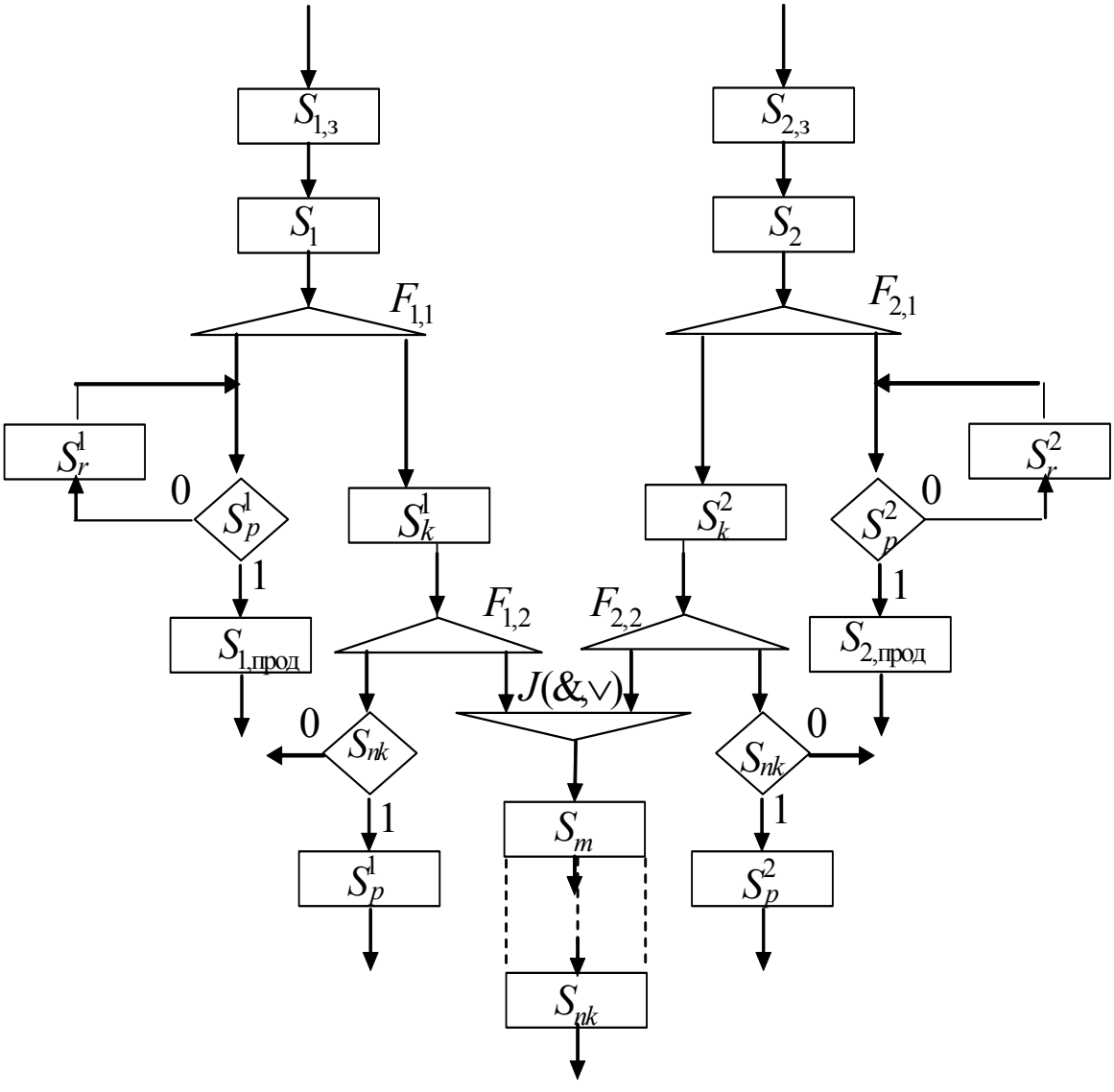


Рис. 6.1. Фрагмент графа НДА алгоритма управления взаимодействующими параллельными процессами при обращении к разделяемым данным (общему критическому ресурсу) для двух процессов

Учитывая рассматриваемые требования к алгоритму взаимодействия параллельными процессами при обращении к разделяемым данным и результаты формализации взаимоисключения критических участков, представленных в виде уравнений (6.6) и (6.7), систему канонических уравнений, формализующих рассматриваемый алгоритм управления для двух процессов, можно представить состоящей из следующих трех частей.

Для первой ветви:

$$S_1(t+1) = (S_{1,3} \vee S_1)\overline{S}_k^1,$$
$$S_k^1(t+1) = S_1\overline{S}_k^2 \vee S_k^1\overline{S}_p^1,$$
$$S_p^1(t+1) = S_{nk}S_k^1, \qquad (6.8,\text{а})$$
$$S_r^1(t+1) = (S_1 \vee S_r^1)\overline{S}_p^1,$$
$$S_{1,\,\text{прод}}(t+1) = S_r^1 S_p^1.$$

Для второй ветви:

$$S_2(t+1) = (S_{2,3} \vee S_2)\overline{S}_k^2,$$
$$S_k^2(t+1) = S_2\overline{S}_k^1\overline{S}_1 \vee S_k^2\overline{S}_p^2,$$
$$S_p^2(t+1) = S_{nk}S_k^2, \qquad (6.8,\text{б})$$
$$S_r^2(t+1) = (S_2 \vee S_r^2)\overline{S}_p^2,$$
$$S_{1,\,\text{прод}}(t+1) = S_r^2 S_p^2.$$

Для последовательной компоненты алгоритма управления:

$$S_m(t+1) = S_k^1 S_1 \vee S_k^2 S_2,$$
$$S_{nk}(t+1) = S_m, \qquad (6.8,\text{в})$$

где переменные ($S_{1,3}$ и $S_{2,3}$), (S_1 и S_2) и (S_p^1 и S_p^2) имеют тот же смысл, что и в разделе 6.3 при рассмотрении вопросов взаимоисключения критических участков; S_m – сокращенное обозначение события, обеспечивающего начало процедуры реализации обращения к разделяемым данным (критическому ресурсу); S_{nk} – сокращенное обозначение события, являющееся заключительным в процедуре реализации обращения к разделяемым данным; (S_r^1 и S_r^2) – сокращенное обозначение событий, символизирующих ожидание условия выхода первого и второго процессов из критического участка после окончания процедуры обращения к разделяемым данным; ($S_{1,\text{прод}}$ и $S_{2,\text{прод}}$) – сокращенное обозначение событий, инициирующих продолжение работы первого и второго процессов после окончания процедуры обращения к разделяемым данным.

Событие S_m будет истинным тогда, когда в непосредственно предшествующий момент времени будут истинными условия вы-

хода алгоритмического процесса за комбинированный соединитель $J(\&,\vee)$. Истинность этих условий определяется наличием принятой заявки на обслуживание процесса и нахождения этого процесса в своем критическом интервале ($S_k^i S_i = 1$). Учитывая это обстоятельство, формальное представление события S_m будет соответствовать выражению (6.8,в).

Для простоты рассмотрения вопросов реализации процедуры обращения к разделяемым данным будем считать, что для выполнения этой процедуры необходимо только два оператора (два события S_m и S_{nk}).

Проследим работу рассматриваемого алгоритма управления параллельными процессами по ГСАП (см. рис. 6.1) и системе уравнений (6.8,а), (6.8,б), (6.8,в). Для определенности был принят приоритет первого процесса выше приоритета второго процесса.

Будем рассматривать самый наихудший вариант межпроцессного взаимодействия, когда оба процесса одновременно достигли операторов, определяющих заявки на доступ к разделяемым данным (события $S_{1,3} = S_{2,3} = 1$). Тогда воспринимается для реализации заявка того процесса, который не находится в данный момент времени в критическом участке. В том случае, если оба процесса не находятся в критическом участке, воспринимаются обе заявки ($S_1 = S_2 = 1$), т.е. оба процесса будут находиться в режиме пассивного ожидания доступа к разделяемым данным. При этом будет открыт доступ к разделяемым данным для того процесса, который имеет наивысший приоритет.

Так как для нашего предположения наивысшим приоритетом обладает первый процесс, то будет истинным событие S_k^1 ($S_k^1 = 1$). Одновременно с зарождением события S_k^1 первый процесс приостанавливает свою работу и ждет окончания процедуры обращения к разделяемым данным. Это обеспечивается зарождением события S_r^1 и его сохранением ($S_r^1 = 1$) до тех пор, пока не закончится процедура обращения к разделяемым данным, и первый процесс выйдет из критического участка при $S_p^1 = 1$. При $S_r^1 S_p^1 = 1$ первый процесс, приостановленный на период обращения к разделяемым данным, продолжит свою работу ($S_{1,\text{прод}} = 1$).

В том случае, когда были восприняты заявки на обслуживание от обоих процессов, после выхода первого процесса из крити-

ческого участка ($S_k^1 = 0$) и в связи с тем, что событие S_1 к этому времени также будет равно нулю, зародится событие S_k^2, определяющее вход второго процесса в критический участок. После этого работа алгоритма управления доступом второго процесса к разделяемым данным будет такой же, как и работа алгоритма управления для первого процесса.

Для большей наглядности при изучении работы алгоритма управления взаимодействующими параллельными процессами при обращении к разделяемым данным можно воспользоваться временными диаграммами, отражающими организацию синхронизации процессов, которая реализуется событиями, представленными в управляющем алгоритме.

На рис. 6.2 представлены временные диаграммы для всех событий, реализуемых в алгоритме управления двумя параллельными процессами при обращении к разделяемым данным. Временные диаграммы построены по уравнениям (6.8,а), (6.8,б), (6.8,в), представляющим систему НД СКУ для наиболее худшего варианта межпроцессного взаимодействия, когда заявки на обслуживание восприняты от обоих процессов одновременно, причем после восприятия заявок они опять поступили на обслуживание.

S_1	S_1	S_1					S_1	S_1	S_1	S_1	S_1	S_1
S_k^1		S_k^1	S_k^1	S_k^1	S_k^1							S_k^1
S_m			S_m					S_m				
S_{nk}				S_{nk}					S_{nk}			
$S_p^1(S_p^2)$					S_p^1					S_p^2		
S_r^1		S_r^1	S_r^1	S_r^1	S_r^1						S_r^1	S_r^1
$S_{1,\text{прод}}$						$S_{1,n}$						
S_2	S_2	S_2	S_2	S_2	S_2	S_2	S_2					S_2
S_k^2							S_k^2	S_k^2	S_k^2	S_k^2		
S_r^2								S_r^2	S_r^2	S_r^2		
$S_{2,\text{прод}}$											$S_{2,n}$	

Рис. 6.2. Временные диаграммы работы алгоритма управления двумя параллельными процессами при обращении к разделяемым данным

Такие временные диаграммы для разных комбинаций заявок на обслуживание позволят осуществить проверку правильности работы алгоритма управления, представленного системой НД СКУ для различных комбинаций поступлений заявок на обслуживание. Действительно, так как для формализации алгоритма управления взаимодействующими параллельными процессами используется модель НДА, то на временных диаграммах можно наблюдать взаимное временное расположение отдельных событий, их возможное одновременное существование и моменты их зарождения и сохранения.

Для построения временных диаграмм работы алгоритма управления межпроцессного взаимодействия для большего числа процессов можно использовать систему моделирования СОМПА (77, 78).

6.4.2. Формализация алгоритма управления взаимодействующими параллельными процессами при обращении к разделяемым данным (общему ресурсу) для *n*-процессов

Рассматриваемая в данном разделе методика формализации алгоритма управления взаимодействующими параллельными процессами для организации обращения к разделяемым данным, когда число процессов $n > 2$, базируется на следующих предпосылках:

– все основные требования к алгоритму управления параллельными процессами не должны зависеть от количества процессов, участвующих в межпроцессных взаимодействиях;

– алгоритм управления взаимодействующими параллельными процессами должен строиться на основе использования результатов формализации функции взаимоисключения критических участков и на основе учета принятой приоритетности процессов по доступу к разделяемым данным;

– система уравнений, формализующих алгоритм управления в виде НД СКУ для всех событий, реализуемых в алгоритме управления, должна иметь одинаковую структуру для обслуживания любого *i*-го процесса.

Напомним основные требования к организации межпроцессного взаимодействия при обращении к разделяемым данным [63, 71].

Если несколько параллельных процессов, более двух, хотят пользоваться некоторым критическим ресурсом, под которым понимается ресурс, допускающий обслуживание только одного пользователя за один раз в режиме разделения, то им следует, во избежание конфликтных ситуаций, синхронизировать свои действия таким образом, чтобы этот ресурс всегда находился в распоряжении не более чем одного из них. Все другие процессы должны переводиться в режим ожидания обращения к общему ресурсу.

Учитывая основные требования к организации межпроцессного взаимодействия при обращении к разделяемым данным и основные подходы для решения задачи формализации алгоритма управления процессами (для $n > 2$), граф НДА, представляющий этот алгоритм (рис. 6.3), будет иметь ту же структуру, что и граф НДА для двух процессов (см. рис. 6.1).

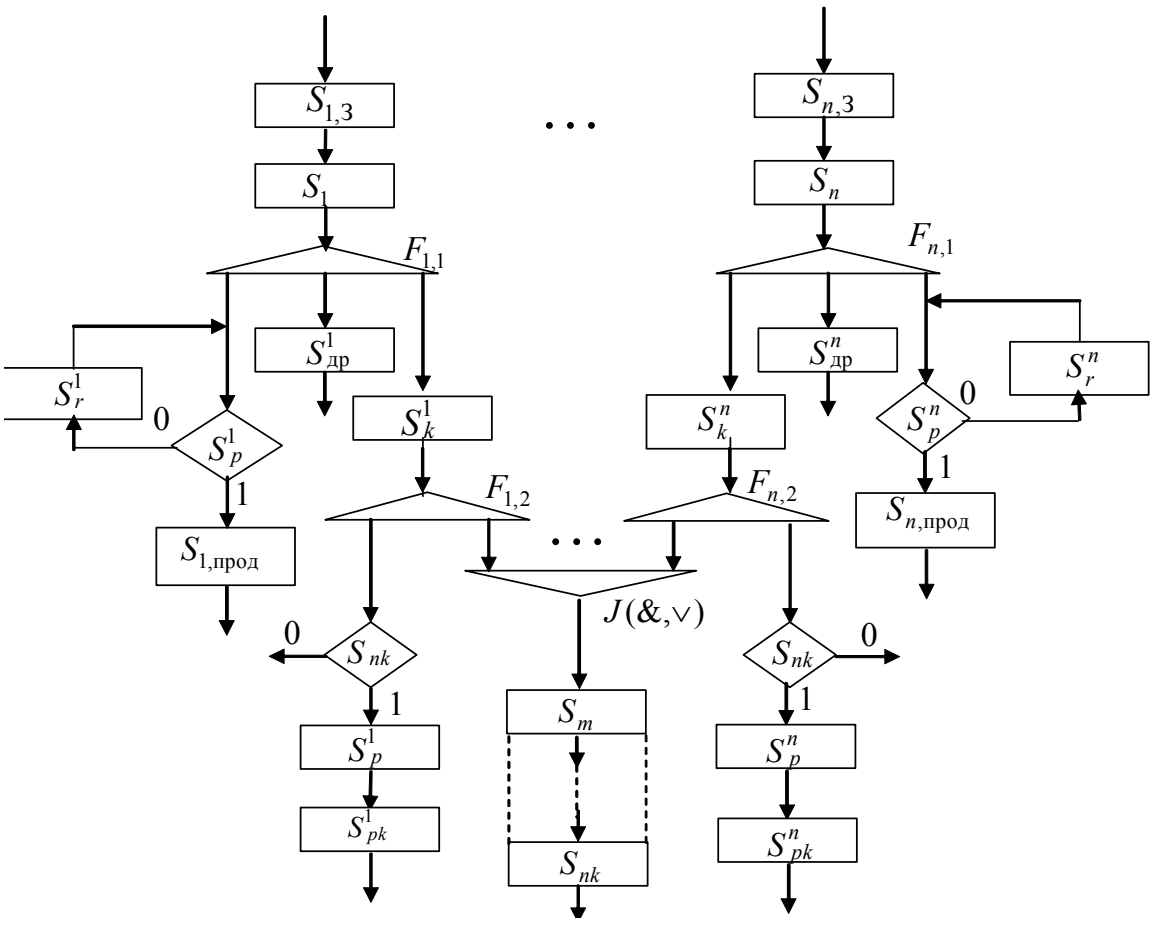


Рис. 6.3. Фрагмент графа НДА алгоритма управления взаимодействующими параллельными процессами при обращении к разделяемым данным для $n > 2$ процессов

В соответствии с графом НДА (см. рис.6.3) и учитывая системы канонических уравнений (6.8,а), (6.8,б) и (6.8,в), формализующих алгоритм управления межпроцессного взаимодействия для двух процессов, система канонических уравнений, формализующая алгоритм управления межпроцессного взаимодействия для $n > 2$ процессов для обслуживания любого i-го процесса и для последовательной части алгоритма управления, будет иметь следующий вид [71]:

$$
\begin{aligned}
&S_i(t+1) = (S_{i,3} \vee S_i)\overline{S}_k^i\ ,\\
&S_k^i(t+1) = S_i S_{\text{вз}}^i S_{\text{пр}}^i \vee S_k^i \overline{S}_p^i,\\
&S_p^i(t+1) = S_{nk} S_k^i,\\
&S_r^i(t+1) = (S_i \vee S_r^i)\overline{S}_p^i,\\
&S_{i\text{прод}}(t+1) = S_r^i S_p^i,\\
&S_m(t+1) = S_k^1 S_1 \vee S_k^2 S_2 \vee \cdots \vee S_k^n S_n,\\
&S_{nk}(t+1) = S_m,
\end{aligned}
\qquad (6.9)
$$

где переменные $S_{i,3}, S_{i,}, S_k^i, S_p^i, S_{nk}, S_r^i, S_m, S_{i,\text{прод}}$ имеют тот же смысл, что и аналогичные переменные в системах уравнений (6.8,а), (6.8,б), (6.8,в); $S_{\text{вз}}^i$ – комбинационное событие, обеспечивающее взаимоисключение критических интервалов на основе несовместимости события S_k^i с другими событиями из общего их числа n. Так как для всех событий типа S_k^i должно выполняться условие: $S_k^1 \overline{S}_k^2 ... \overline{S}_k^n \vee \overline{S}_k^1 S_k^2 \overline{S}_k^3 ... \overline{S}_k^n \vee ... \vee \overline{S}_k\, \overline{S}_k^2 ... \overline{S}_k^{n-1} S_k^n = 1$, то событие $S_{\text{вз}}^i$ определится следующей формулой:

$$
S_{\text{вз}}^i = \bigwedge_{(\forall\alpha)(\alpha \neq i)} \overline{S}_k^{\alpha},\ i = \overline{1,n}; \qquad (6.10)
$$

$S_{\text{пр}}^i$ – сокращенное обозначение события, обеспечивающего заданное приоритетное обслуживание i-го процесса при обращении к разделяемым данным.

В качестве примера в данном разделе принята циклическая дисциплина обслуживания (рис. 6.4).

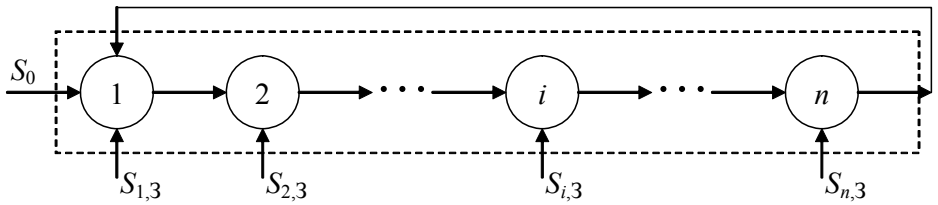


Рис. 6.4. Схема циклической дисциплины обслуживания

Для такой дисциплины $S^i_{\text{пр}}$ может быть представлена следующим выражением:

$$S^i_{\text{пр}}(t+1) = S^i_{\text{пр}}(0) \vee S^i_{\text{пр}} S_{i,3} [S^i_{pk} \underset{(\forall\alpha)(\alpha\neq i)}{\Lambda \overline{S}_{\alpha,3}} \vee S^{i\oplus 1}_{pk} \underset{(\forall\alpha)(i\oplus 1<\alpha\le i-1)}{\Lambda \overline{S}_{\alpha,3}} \vee$$

$$\vee S^{i\oplus 2}_{pk} \underset{(\forall\alpha)(i\oplus 2<\alpha\le i-1)}{\Lambda \overline{S}_{\alpha,3}} \vee \cdots \vee S^{i\oplus(n-1)}_{pk} \underset{(\forall\alpha)(i\oplus(n-1)<\alpha\le i-1)}{\Lambda \overline{S}_{\alpha,3}}], \qquad (6.11)$$

где $\oplus$ – знак сложения по модулю n; S^i_{pk} – сокращенное обозначение события, свидетельствующее о факте выполненного обслуживания i-го процесса при обращении к разделяемым данным, которое определится в данном варианте как задержка события S^i_p на один такт:

$$S^i_{pk}(t+1) = S^i_p. \qquad (6.12)$$

П р и м е ч а н и е. Отметим некоторые особенности в определении индексов событий S_{pk} и $S_{\alpha,3}$ в выражении (6.11). Если при определении верхнего индекса события S_{pk} в результате сложения по модулю n получили нуль, то в качестве индекса принимается не нуль, а величина n. Если при определении индекса α события $S_{\alpha,3}$ получилось противоречивое неравенство, например, $3 < \alpha \le 3$, то соответствующее событие $S_{\alpha,3}$ не существует, тогда $\overline{S}_{\alpha,3} = 1$;

$S^i_{\text{пр}}(0)$ – сокращенное обозначение комбинационного события, определяющего начальный приоритет обслуживания i-го процесса.

$$S^i_{\text{пр}}(0) = S_0 x_n S_{i,3} \underset{(\forall\alpha)(\alpha<i)}{\Lambda \overline{S}_{\alpha,3}}, \qquad (6.13)$$

где S_0 – начальное событие системы управления обращением к разделяемым данным;

$$S_0(t+1) = x_0 \vee S_{pk}^n \vee S_0 \overline{x}_n, \qquad (6.14)$$

где x_n – сигнал инициализации системы управления; x_0 – сигнал приведения системы управления в начальное состояние.

Введение события $S_{\text{пр}}^i(0)$ в условия зарождения события S_k^i, обеспечивающего вход *i*-го процесса в критический участок, объясняется необходимостью обеспечения функции взаимоисключения на начальном этапе работы алгоритма управления параллельными процессами, когда в один и тот же момент времени могли бы зародиться сразу несколько событий типа S_k^i.

В случае, если в многопроцессорной ВС необходимо использовать время ожидания обращения к разделяемым данным в каком-либо *i*-м процессе для решения других задач (реализуемых подпроцессом), не требующих обращения к разделяемым данным, событие, инициирующее выполнение такого подпроцесса, можно представить в следующем виде (см. рис. 6.3):

$$S_{\text{др}}^i(t+1) S_i\, S_{\text{д,З}}^i \overline{S}_k^i \vee S_{\text{др}}^i \overline{S}_p^i, \qquad (6.15)$$

где $S_{\text{д,З}}^i$ – сокращенное обозначение события, определяющего заявку на реализацию подпроцесса в *i*-м процессе.

П р и м е р 6.1. Построить систему уравнений для событий $S_{\text{пр}}^i$, определяющих приоритетность доступа к разделяемым критическим ресурсам для 3 параллельных процессов при циклической дисциплине обслуживания, и событий $S_{\text{вз}}^i$, обеспечивающих взаимоисключение критических интервалов.

На основании уравнений (6.10), (6.11) и (6.12) и учитывая при этом примечания относительно определения индексов событий, представленных в них, получим следующие представления искомых событий для *i* = 1, 2, 3:

$$S_{\text{пр}}^1(t+1) = S_0 x_n S_{1,3} \vee S_{1,3}[S_{pk}^1 \overline{S}_{2,3} \overline{S}_{3,3} \vee S_{pk}^2 \overline{S}_{3,3} \vee S_{pk}^3],$$

$$S_{\text{пр}}^2(t+1) = S_0 x_n S_{2,3} \overline{S}_{1,3} \vee S_{2,3}[S_{pk}^2 \overline{S}_{1,3} \overline{S}_{3,3} \vee S_{pk}^3 \overline{S}_{1,3} \vee S_{pk}^1],$$

$$S_{\text{пр}}^3(t+1) = S_0 x_n S_{3,3} \overline{S}_{1,3} \overline{S}_{2,3} \vee S_{3,3}[S_{pk}^3 \overline{S}_{1,3} \overline{S}_{2,3} \vee S_{pk}^1 \overline{S}_{2,3} \vee S_{pk}^2],$$

$$S_{\text{вз}}^1 = \overline{S}_k^2 \overline{S}_k^3, \qquad S_{\text{вз}}^2 = \overline{S}_k^1 \overline{S}_k^3, \qquad S_{\text{вз}}^3 = \overline{S}_k^1 \overline{S}_k^2.$$

6.5. Формализация алгоритма управления взаимодействующими параллельными процессами в задаче «производители-потребители»

Задача «производители-потребители» является одной из классических задач синхронизации двух асинхронных процессов: процесс – производитель (например, процессор) вырабатывает символы (слова), которые должны быть переданы другому процессу – потребителю (например, печать). Так как интенсивность воспроизведения и потребления символов (слов) различная, то требуется обеспечить передачу символов (слов) с согласованием их интенсивностей [67, 70, 72]. Процессы взаимодействуют через некоторую обобщенную область памяти – согласующий буфер сообщений. В эту область памяти процесс-производитель помещает очередное сообщение, а процесс-потребитель считывает очередное сообщение. В общем случае буфер способен хранить несколько сообщений. Необходимо согласовать работу двух процессов при одностороннем обмене сообщениями по мере развития процессов таким образом, чтобы удовлетворить следующие требования:

– выполнение требований задачи взаимного исключения по отношению к критическому ресурсу – буферу сообщений;

– учет состояния буфера сообщений, характеризующего возможность и невозможность посылки (принятия) очередного сообщения, а именно: процесс-производитель при попытке поместить очередное сообщение в полностью заполненный буфер должен быть полностью заблокирован; попытка процесса-потребителя чтения из пустого буфера также должна быть заблокирована.

Рассмотрим вначале простой вариант, когда процессы взаимодействуют через согласующий буфер в одно слово. Для этого варианта имеет место следующее:

– производитель производит по одному слову (за один раз), а потребитель использует их по одному (за один раз);

– согласующий буфер имеет размер в одно слово.

Для такого простейшего случая взаимодействия взаимное исключение процессов специально не предусматривается (не формализуется), так как в этом случае оно обеспечивается син-

хронизацией процессов. Фрагмент графа НДА алгоритма взаимодействия двух параллельных процессов представлен на рис. 6.5.

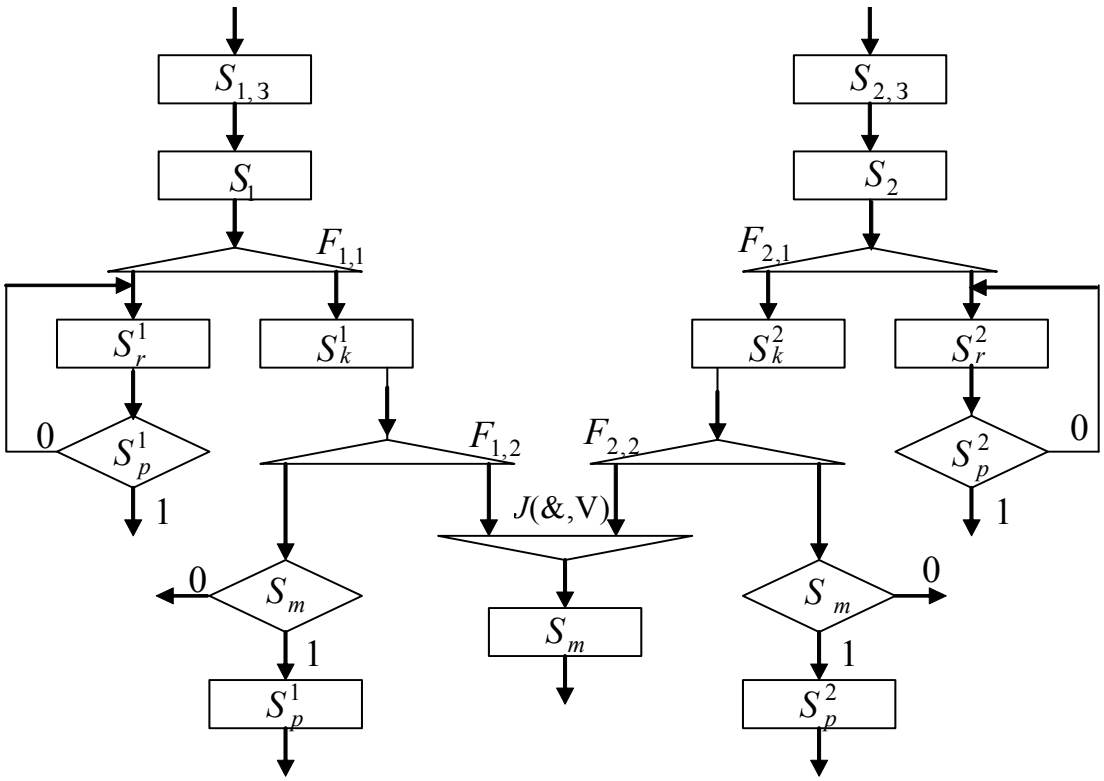


Рис. 6.5. Фрагмент графа НДА алгоритма управления параллельными процессами задачи «производители-потребители» с согласующим буфером в одно слово

Общая структура графа НДА (см. рис. 6.5) почти ничем не отличается от общей структуры графа НДА, представляющего алгоритм управления двумя параллельными процессами при обращении к разделяемому критическому ресурсу (см. рис. 6.1). Для рассматриваемого алгоритма управления процедура обращения к разделяемому критическому ресурсу (согласующему буферу в одно слово) на графе НДА (см. рис. 6.5) представлена не только общей частью (событие S_m), но и двумя непересекающимися операциями: запись в буфер (событие S_p^1) и чтение из буфера (событие S_p^2). Эти же события (S_p^1 и S_p^2) при их истинности осуществляют реализацию выхода из своих критических участков процесса-производителя и процесса-потребителя, соответственно.

Формальное описание рассматриваемого алгоритма управления процессами будет также в основном соответствовать систе-

мам уравнений (6.8,а), (6.8,б) и (6.8,в), но с некоторой корректировкой, учитывающей отмеченные выше конкретные требования к алгоритму управления межпроцессного взаимодействия в задаче «производители-потребители» с учетом наличия согласующего буфера в одно слово. Такое формальное описание, представленное системами канонических уравнений НД СКУ для событий, реализуемых алгоритмом управления, имеет следующий вид.

Для процесса-производителя:

$$\begin{aligned} &S_1(t+1) = (S_{1,3} \vee S_1)\overline{S}_k^1, \\ &S_k^1(t+1) = S_1 S_{\text{БО}} \vee S_k^1 \overline{S}_p^1, \\ &S_{\text{р}}^1(t+1) = S_m S_k^1, \\ &S_r(t+1) = (S_1 \vee S_r^1)\overline{S}_p^1, \end{aligned} \qquad (6.16,\text{а})$$

при $t = 0 / S_{\text{БО}} = 1$.

Для процесса-потребителя:

$$\begin{aligned} &S_2(t+1) = (S_{2,3} \vee S_2)\overline{S}_k^2, \\ &S_k^2(t+1) = S_2 \overline{S}_{\text{БО}} \vee S_k^2 \overline{S}_p^2, \\ &S_p^2(t+1) = S_m S_k^2, \\ &S_r^2(t+1) = (S_2 \vee S_r^2)\overline{S}_p^2. \end{aligned} \qquad (6.16,\text{б})$$

Для последовательной компоненты алгоритма управления:

$$S_m(t+1) = S_k^1 S_1 \vee S_k^2 S_2, \qquad (6.16,\text{в})$$

где S_m – сокращенное обозначение события, определяющего обращение к согласующему буферу; S_p^1 и S_p^2 – сокращенные обозначения событий, определяющих операции записи и чтения из буфера, а после выполнения которых обеспечивающих выход из критических участков процесса-производителя и процесса-потребителя, соответственно; $S_{\text{БО}}$ – сокращенное обозначение события, определяющего состояние буфера (при $S_{\text{БО}} = 1$ буфер пуст); $S_{1,3}$ и $S_{2,3}$ – сокращенные обозначения событий, определяющих заявку на запись и чтение из буфера, соответственно; S_1 и S_2 – сокращенные обозначения событий, определяющих прием заявок на запись и чтение, соответственно; S_r^1 и S_r^2 – сокращенные обозначения событий, символизирующих ожидание процессами производителя и потребителя окончания операции записи и чтения из буфера, соответственно.

Рассмотрим теперь вариант формализации алгоритма управления параллельными процессами в задаче «производители-потребители», когда процессы взаимодействуют через согласую-

щий буфер на число слов $N > 1$. Для этого алгоритма должно быть обеспечено:

– взаимоисключение процессов при обращении к критическому ресурсу – согласующему буферу;

– предотвращение чтения из пустого буфера и записи в полный буфер.

Для определения состояния буфера используется счетчик буфера (СчБ), в который перед операциями записи (чтения) в буфер заносится константа N, определяющая максимальное число слов, записываемых в буфер. При $t = 0$ / СчБ:=N.

Во время записи (чтения) в буфер в СчБ выполняются следующие операции:

– при каждой записи одного слова из СчБ вычитается единица (СчБ:=СчБ–1), поэтому при СчБ = 0 буфер будет полон, а его состояние будем характеризовать событием $S_{\text{БП}}$ (при $S_{\text{БП}} = 1$ буфер полон);

– при каждом чтении из буфера одного слова в СчБ посылается единица (СчБ:= СчБ + 1), поэтому при СчБ = N буфер будет пуст, а его состояние будем характеризовать событием $S_{\text{БО}}$ (при $S_{\text{БО}} = 1$ буфер пуст).

Введение двух событий $S_{\text{БП}}$ и $S_{\text{БО}}$ позволяет охарактеризовать промежуточное состояние буфера, когда он может быть и не пустым, и не полным ($S_{\text{БП}} = S_{\text{БО}} = 0$). Это может иметь место, когда при очередной записи буфер был заполнен не полностью, т.е. число слов в буфере после очередной записи будет меньше N.

Учитывая рассмотренные выше требования к алгоритму управления межпроцессного взаимодействия для двух параллельных процессов через согласующий буфер с числом слов $N > 1$, а также методику формализации функции взаимоисключения критических участков (раздел 6.3), граф НДА, представляющий этот алгоритм (рис. 6.6), будет иметь в основном такую же структуру, что и граф НДА (см. рис. 6.5), представляющий алгоритм управления процессами для варианта, когда согласующий буфер имеет объем в одно слово.

На графе НДА (см. рис. 6.6) по сравнению с графом НДА (см. рис. 6.5) дополнительно представлены необходимые переходы, связанные с организацией записи и чтения из буфера и выходом взаимодействующих процессов из своих критических участков. Следует также отметить, что на графе НДА (см. рис. 6.6),

с целью упрощения его схемы, опущены некоторые логические вершины, определяющие события (S_1 и S_2) и (S_k^1 и S_k^2). Несмотря на указанные упрощения, граф НДА (см. рис. 6.6) в достаточной степени дает представление об общей структуре алгоритма управления взаимодействующими процессами.

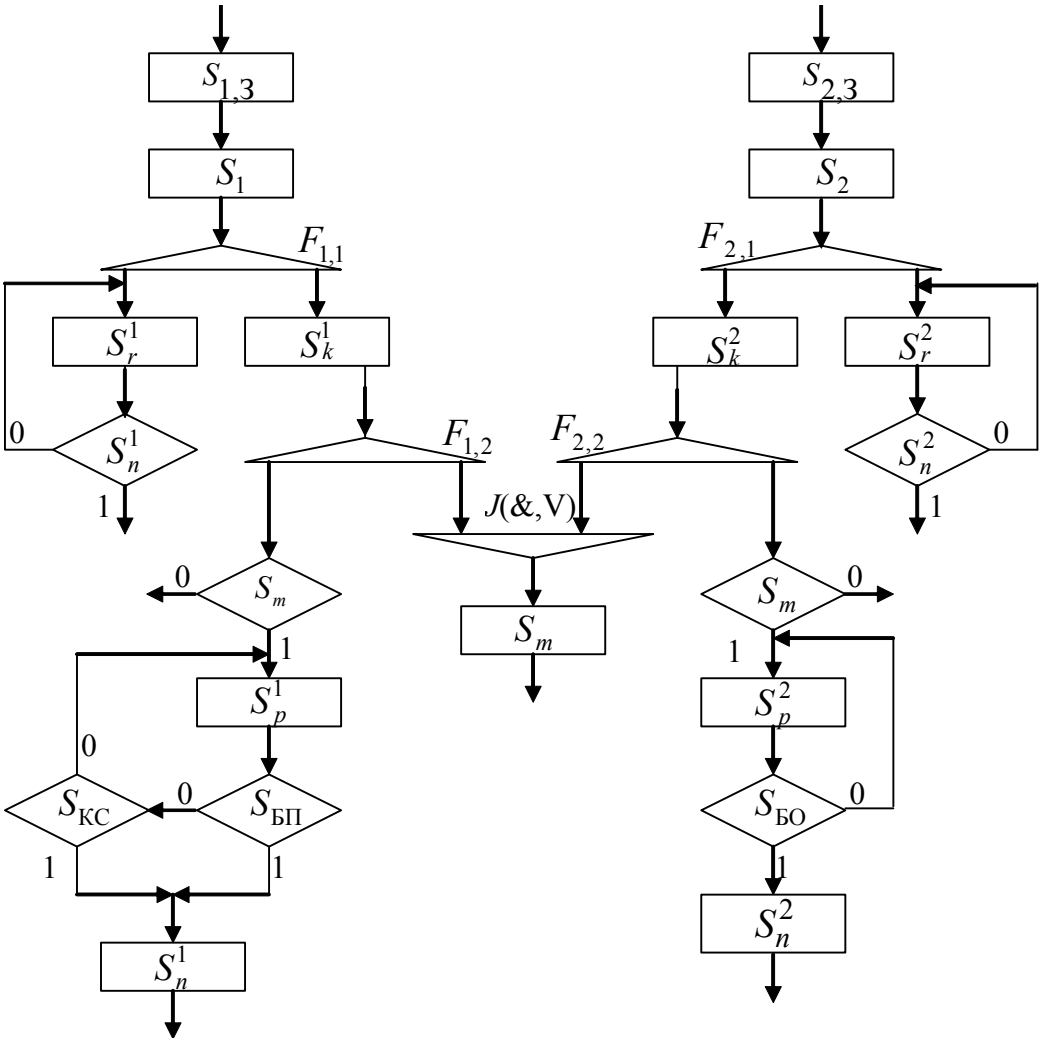


Рис. 6.6. Фрагмент графа НДА алгоритма управления параллельными процессами задачи «производители-потребители» с согласующим буфером в $N > 1$ слов

Такой граф НДА (см. рис. 6.6), как следует из дальнейшего изложения, совместно с системами канонических уравнений НД СКУ, описывающих все представимые в управляющем алгоритме события, дает полную картину межпроцессного взаимодействия двух параллельных процессов задачи «производители-потребители» для варианта использования согласующего буфера с числом слов $N > 1$.

Перейдем теперь к определению формального описания основных событий, реализуемых в рассматриваемом алгоритме управления процессами и представляемых в дальнейшем в виде системы канонических уравнений НД СКУ. В начале рассмотрим формализацию самых главных событий, определяющих входы процессов в свои критические участки и обозначенных в предыдущих разделах символом S_k^i (где i – номер процесса).

Для нашего варианта задачи «производители-потребители» используются только два взаимодействующих процесса. Для них указанные события для удобства дальнейшего анализа представим, на основании системы уравнений (6.9), в следующем общем виде:

$$\begin{aligned} S_k^1(t+1) &= S_{\text{вз}}^1 S_{\text{ко}}^1 \vee S_k^1 \overline{S}_n^1, \\ S_k^2(t+1) &= S_{\text{вз}}^2 S_{\text{ко}}^2 \vee S_k^2 \overline{S}_n^2, \end{aligned} \tag{6.17}$$

где S_n^1 и S_n^2 – сокращенные обозначения событий, свидетельствующих о факте окончания процедуры записи и чтения из буфера и обеспечивающих выход процессов производителя и потребителя из своих критических участков, соответственно.

Выражения для событий $S_{\text{вз}}^1$ и $S_{\text{вз}}^2$ определятся, исходя из соотношения (6.10), следующим образом:

$$S_{\text{вз}}^1 = \overline{S}_k^2, \;\; S_{\text{вз}}^2 = \overline{S}_k^1. \tag{6.18}$$

Представление условий зарождения событий S_k^1 и S_k^2 в (6.17), в отличие от представления их в системе уравнений (6.18), в виде объединенных событий $S_{\text{ко}}^1 = (S_1 S_{\text{пр}}^1)$ и $S_{\text{ко}}^2 = (S_2 S_{\text{пр}}^2)$ объясняется тем, что события $S_{\text{пр}}^1$ и $S_{\text{пр}}^2$, входящие в них, зависят не только от состояния согласующего буфера, но также и от событий S_1 и S_2.

Для получения описаний событий $S_{\text{ко}}^1$ и $S_{\text{ко}}^2$ воспользуемся прямой таблицей переходов (ПТП), определяющей эти события. ПТП (табл. 6.1) строится на основе требований к рассматриваемому алгоритму взаимодействия двух процессов.

Таблица 6.1

$S_{БО}(t)$	$S_{БП}(t)$	$S_1(t)$	$S_2(t)$	$S^1_{ко}(t+1)$	$S^2_{ко}(t+1)$	Комментарий
0	0	0	0	0	0	Буфер не пустой и не полный, возможна или запись, или чтение из буфера: $S^1_{ко}=1$ при $S_1=1$; $S^2_{ко}=1$ при $S_2=1$ и $S_1=0$
0	0	0	1	0	1	
0	0	1	0	1	0	
0	0	1	1	1	0	
0	1	0	0	0	0	Буфер полон (не пустой), возможно только чтение из буфера: $S^2_{ко}=1$ при $S_2=1$
0	1	0	1	0	1	
0	1	1	0	0	0	
0	1	1	1	0	1	
1	0	0	0	0	0	Буфер пустой («не полный»), возможна только запись в буфер: $S^1_{ко}=1$ при $S_1=1$
1	0	0	1	0	0	
1	0	1	0	1	0	
1	0	1	1	1	0	
1	1	0	0	–	–	Невозможные комбинации состояний буфера, $S^1_{ко}$ и $S^2_{ко}$ не определены
1	1	0	1	–	–	
1	1	1	0	–	–	
1	1	1	1	–	–	

Исходя из ПТП (см. табл. 6.1), минимальные выражения для событий $S^1_{ко}$ и $S^2_{ко}$, с учетом их неопределенных значений, примут следующий вид:

$$S^1_{ко}(t+1) = S_1\overline{S}_{БП},$$
$$S^2_{ко}(t+1) = S_2(S_{БП} \vee \overline{S}_1\overline{S}_{БО}). \qquad (6.19)$$

Назначение и формальное описание остальных основных событий, представленных на графе НДА (см. рис. 6.6), исходя из требований алгоритма управления процессами, определяются следующим образом:

– события S_1 и S_2 определяют прием заявок процессов на запись и чтение из согласующего буфера; они определяются так же, как в системах уравнений (6.16,а) и (6.16,б);

– событие S_m определяет начало процедуры обращения к критическому ресурсу (согласующему буферу); выражение для события S_m определяется в соответствии с уравнением (6.16,в);

– события S_p^1 и S_p^2 определяют непересекающиеся операции записи и чтения из буфера; выражения для них имеют вид

$$S_p^1(t+1) = S_m S_k^1 \vee S_p^1(\overline{S_{\text{БП}} \vee S}_{\text{КС}}),$$
$$S_p^2(t+1) = S_m S_k^2 \vee S_p^2 \overline{S}_{\text{БО}}. \qquad (6.20)$$

Из выражения (6.20) следует, что запись в буфер заканчивается только тогда, когда буфер будет полон ($S_{\text{БП}} = 1$) или когда закончится поступление записываемой информации в буфер, хотя буфер может быть еще не полон (при ($S_{\text{КС}} = 1$)); чтение из буфера осуществляется до тех пор, пока буфер полностью не очистится (при ($S_{\text{БО}} = 1$));

– события S_n^1 и S_n^2, обеспечивающие выход взаимодействующих процессов из соответствующих своих критических участков, определяются следующим образом:

$$S_n^1(t+1) = S_p^1(S_{\text{БП}} \vee S_{\text{КС}}),$$
$$S_n^2(t+1) = S_p^2 S_{\text{БО}}; \qquad (6.21)$$

– события S_r^1 и S_r^2 символизируют ожидание взаимодействующими процессами окончания процедуры записи и чтения из буфера, чтобы продолжить свою работу, прерванную обращением к согласующему буферу; выражения для них имеют вид

$$S_r^1(t+1) = (S_1 \vee S_r^1)\overline{S}_n^1,$$
$$S_r^2(t+1) = (S_2 \vee S_r^2)\overline{S}_n^2. \qquad (6.22)$$

Таким образом, общая сводная система канонических уравнений НД СКУ, представляющая алгоритм управления взаимодействующими процессами задачи «производители-потребители», когда процессы взаимодействуют через согласующий буфер с числом слов $N > 1$, будет представлена следующими выражениями.

Для процесса-производителя:

$$S_1(t+1) = (S_{1,3} \vee S_1)\overline{S}_k^1,$$
$$S_k^1(t+1) = S_1\overline{S}_k^2\overline{S}_{\text{БП}} \vee S_k^1\overline{S}_n^1,$$
$$S_p^1(t+1) = S_m S_k^1 \vee S_p^1(\overline{S_{\text{БП}} \vee S_{\text{КС}}}),$$
$$S_r^1(t+1) = (S_1 \vee S_r^1)\overline{S}_n^1, \quad (6.23,\text{а})$$
$$S_n^1(t+1) = S_p^1(S_{\text{БП}} \vee S_{\text{КС}}).$$

Для процесса-потребителя:

$$S_2(t+1) = (S_{2,3} \vee \mathrm{S}_2)\overline{S}_k^2,$$
$$S_k^2(t+1) = S_2\overline{S}_k^1(S_{\text{БП}} \vee \overline{S}_1\overline{S}_{\text{БО}}) \vee S_k^2\overline{S}_n^2,$$
$$S_p^2(t+1) = S_m S_k^2 \vee S_p^2\overline{S}_{\text{БО}},$$
$$S_r^2(t+1) = (S_2 \vee S_r^2)\overline{S}_n^2, \quad (6.23,\text{б})$$
$$S_n^2(t+1) = S_p^2 S_{\text{БО}}.$$

Для последовательной компоненты алгоритма управления:

$$S_m(t+1) = S_k^1 S_1 \vee S_k^2 S_2. \quad (6.23,\text{в})$$

6.6. Формализация алгоритмов управления взаимодействующими параллельными процессами в задаче «читатели-писатели»

Задача «читатели-писатели», наряду с задачей «производители-потребители», является еще одним примером классической задачи синхронизации двух классов процессов, которые имеют доступ к некоторому общему разделяемому ресурсу (РР), в качестве которого может быть, например, общая область памяти. Процессы первого класса («писатели») должны иметь монопольный доступ к ресурсу. Процессы второго класса («читатели») могут обращаться к данному ресурсу параллельно с любым числом процессов этого же класса, так как осуществляют чтение без изменения содержимого памяти. В качестве примера взаимодействия таких процессов может служить система продажи авиабилетов, позволяющая организовывать продажу билетов и изменять таким образом списки пассажиров и число свободных мест одновременно несколькими кассирами. Если в такой системе не будут предусмотрены правила, регламентирующие доступ к базам данных нескольких программ, то вполне возможна продажа билетов на одно и то же место нескольким пассажирам. Для исключения таких событий не должно допускаться параллельное исполнение двух процессов, которые читают и изменяют значение одного и того же объекта [74].

Словесная формулировка рассматриваемого варианта алгоритма взаимодействия параллельных процессов задачи «читатели-писатели» характеризуется следующими особенностями [61,70]. Алгоритм взаимодействия процессов должен гарантировать, чтобы в любой момент времени только один читатель или один писатель мог войти в критический интервал (участок). Для того, чтобы не монополизировать доступ к РР читателей, когда их будет слишком много и они могли бы заблокировать запись, должен быть предусмотрен счетчик читателей (СчЧ), в который заносится некоторая константа *E*. Занесение этой константы в СчЧ должно выполняться при пустом счетчике после окончания работы читателя.

«Читатель» получает доступ к данным только в том случае, если в настоящий момент нет как работающего, так и ожидающего писателя. Первое условие должно обеспечиваться взаимоисключением критических интервалов обращения читателей и писателей к РР. Второе условие предотвращает бесконечное откладывание реализации процессов писателей из-за наплыва читателей. После окончания работы читателя СчЧ уменьшается на единицу и, если он будет не пустой, работа читателей может быть продолжена до тех пор, пока СчЧ не станет пустым. Вычитание единицы из СчЧ осуществляется для не пустого счетчика после окончания работы «читателя».

«Писатель» получает монопольный доступ к РР, если в предшествующий момент времени не было ни чтения, ни записи в РР. В дальнейшем писатель может получить доступ к РР только в том случае, если в настоящий момент времени нет работающего читателя и СчЧ пуст или когда, независимо от состояния СчЧ, нет как ожидающего, так и работающего читателя. Так как доступ писателя к РР для обоих вариантов возможен только при условии отсутствия работающего читателя, то это условие должно обеспечиваться взаимоисключением критических интервалов обращения читателей и писателей к РР. Когда писатель заканчивает работу и СчЧ не пуст, то предпочтение отдается ожидающим читателям, а не ожидающим писателям, что предотвращает бесконечное откладывание процессов читателей из-за наплыва писателей. Если после окончания работы писателя СчЧ пуст, то в него заносится константа *E* и предпочтение также отдается ожидающим «читателям», а не ожидающим «писателям». Это обстоятельство обеспечивается, как это будет в дальнейшем показано, надлежащей формализацией события, характеризующего ожидающего «писателя».

Учитывая основные требования к алгоритму взаимодействия процессов в задаче «читатели-писатели», его формальное описание начнем с определения событий, характеризующих **ожидающих** и **работающих писателей и читателей**. При этом будем базироваться на методике формализации алгоритмов взаимодействия параллельных процессов при обращении к РР в [2, 71]. Представим также для наглядности алгоритм управления параллельными процессами в виде графа НДА (рис. 6.7). Представленный на рис. 6.7 граф НДА, несмотря на отсутствие в нем некоторых логических вершин, которые опущены для простоты схемы графа, в достаточной степени дает представление об общей структуре алгоритма управления процессами.

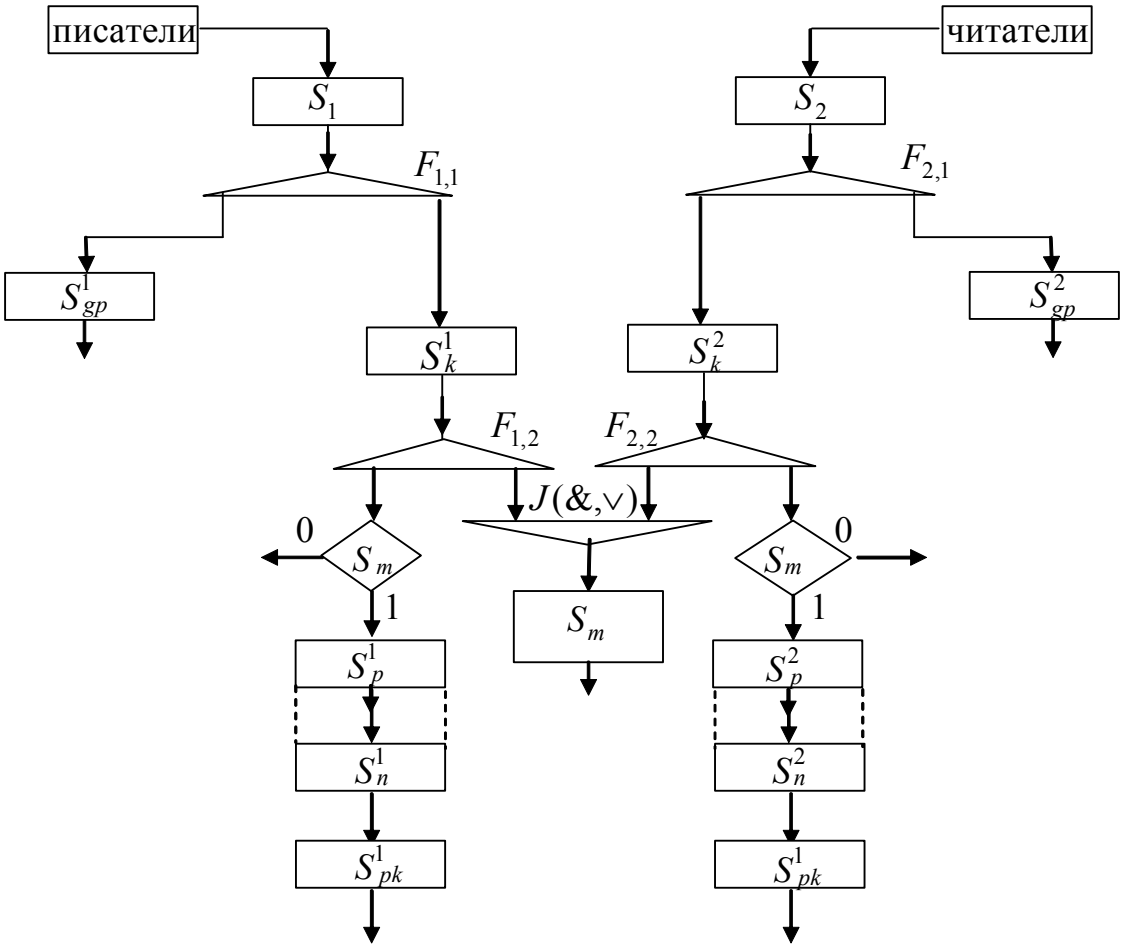


Рис. 6.7. Фрагмент графа НДА алгоритма управления процессами в задаче «читатели-писатели»

Работающего «писателя» и «читателя» будем характеризовать событиями, которые определяют не только вход в критический интервал писателей и читателей, но и нахождение их в кри-

тическом интервале. Обозначим эти события буквами S_k^1 и S_k^2, соответственно, т.е. если $S_k^1 = 1$, то ожидающий писатель переходит в разряд работающих писателей. То же самое относится и к событию S_k^2, т.е. при $S_k^2 = 1$ ожидающий читатель переходит в разряд работающих читателей.

События типа S_k^1 и S_k^2 в некоторых наших работах [74] называли событиями активного ожидания обращения к РР, так как фактически непосредственная запись или чтение в РР будет выполняться лишь после зарождения событий S_k^1 и S_k^2 (спустя несколько тактов после их зарождения).

Ожидающего «писателя» и «читателя» будем характеризовать событиями S_1 и S_2, соответственно, которые могут зародится только при наличии заявки на запись и чтение из РР и при условии, если свободен вход в критический интервал писателя и читателя, т.е. при условиях $S_k^1 = 0$ и $S_k^2 = 0$. События S_1 и S_2 будут сохраняться до тех пор, пока условия $S_k^1 = 0$ и $S_k^2 = 0$ не изменятся. Повторное зарождение событий S_1 и S_2 для очередного писателя и читателя возможно после окончания работы предыдущего писателя и читателя, соответственно. Исходя из рассмотренных условий события, S_1 и S_2, могут быть представлены следующими уравнениями:

$$\left.\begin{aligned} S_1(t+1) &= S_{1,3}(\overline{S}_k^1 \vee S_n^1) \vee S_1 \overline{S}_k^1 \\ S_2(t+1) &= S_{2,3}(\overline{S}_k^2 \vee S_n^2) \vee S_2 \overline{S}_k^2 \end{aligned}\right\}, \qquad (6.24)$$

где $S_{1,3}$ и $S_{2,3}$ – события, определяющие заявку на запись и чтение; S_n^1 и S_n^2 – события являются заключительными в процедуре записи или чтения при обращения к РР писателей и читателей и обеспечивающими выход писателей и читателей из критических участков, соответственно.

Условия зарождения событий S_k^1 и S_k^2, определяющих входы в критические интервалы «писателей» и «читателей», должны в соответствии с алгоритмом взаимодействия процессов удовлетворять требованиям как по взаимоисключению, так и по приоритетности взаимодействующих процессов при обращении к РР. В связи с этим уравнения, определяющие события S_k^1 и S_k^2, представим

в следующем общем виде, который соответствует формальному представлению их в системе (6.9):

$$\left.\begin{aligned} S_k^1(t+1) &= S_1 S_{\text{пр}}^1 S_{\text{вз}}^1 \vee S_k^1 \overline{S}_n^1 \\ S_k^2(t+1) &= S_2 S_{\text{пр}}^2 S_{\text{вз}}^2 \vee S_k^2 \overline{S}_n^2 \end{aligned}\right\}. \qquad (6.25)$$

Первая часть уравнений (6.25), определяющая зарождение событий S_k^1 и S_k^2, представлена конъюнкцией из трех событий:

S_1 и S_2 – события, определяющие ожидающего писателя и читателя;

$S_{\text{пр}}^1$ и $S_{\text{пр}}^2$ – события, обеспечивающие условия приоритетности процессов записи и чтения в соответствии с требованиями алгоритма взаимодействия процессов;

$S_{\text{вз}}^1$ и $S_{\text{вз}}^2$ – события, обеспечивающие взаимоисключение процессов записи и чтения на основе несовместимости событий S_k^1 и S_k^2, т.е. для них должно выполняться условие: $S_k^1 \overline{S}_k^2 \vee \overline{S}_k^1 S_k^2 = 1$, откуда $S_{\text{вз}}^1 = \overline{S}_k^2$, а $S_{\text{вз}}^2 = \overline{S}_k^1$.

Вторая часть уравнений (6.25), определяющая условие сохранения событий S_k^1 и S_k^2, обеспечивает занятие критических интервалов до тех пор, пока не закончится процесс записи (при $S_n^1 = 1$) или чтения (при $S_n^2 = 1$).

Исходя из требований алгоритма взаимодействия процессов, приоритетность процессов записи и чтения зависит от состояния СчЧ, от наличия ожидающих писателей и читателей (события S_1 и S_2) и от значений событий, свидетельствующих о завершении процессов записи и чтения, т.е. имеем

$$S_{\text{пр}}^1 = f_1(S_1, S_2, S_{\text{чо}}, S_{pk}^1, S_{pk}^2),$$

$$S_{\text{пр}}^2 = f_2(S_1, S_2, S_{\text{чо}}, S_{pk}^1, S_{pk}^2),$$

где S_{pk}^1 и S_{pk}^2 – события, свидетельствующие о факте окончания процедуры записи и чтения, соответственно, эти события фактически повторяют события S_n^1 и S_n^2, но со сдвигом на один такт.

В связи с тем, что события $S_{\text{пр}}^1$ и $S_{\text{пр}}^2$ зависят от событий S_1 и S_2, то целесообразно рассматривать формализацию условий для

зарождения событий S_k^1 и S_k^2, состоящих из двух событий ($S_1\, S_{\text{пр}}^1$) и, соответственно, ($S_2\, S_{\text{пр}}^2$), как индивидуальных событий, обозначив их символами $S_{k,0}^1$ и $S_{k,0}^2$. Тогда уравнения (6.25) примут вид:

$$\left.\begin{aligned} S_k^1(t+1) &= S_{k,0}^1 S_{\text{вз}}^1 \vee S_k^1 \overline{S}_n^1 \\ S_k^2(t+1) &= S_{k,0}^2 S_{\text{вз}}^2 \vee S_k^2 \overline{S}_n^2 \end{aligned}\right\} \qquad (6.26)$$

Для получения уравнений, формализующих события $S_{k,0}^1$ и $S_{k,0}^2$ с учетом пяти исходных событий ($S_1, S_2, S_{\text{чо}}, S_{pk}^1, S_{pk}^2$), воспользуемся построенной прямой таблицей переходов (ПТП) для событий $S_{k,0}^1$ и $S_{k,0}^2$ (табл. 6.2).

Таблица 6.2

$S_{\text{чо}}(t)$	$S_1(t)$	$S_2(t)$	$S_{pk}^1(t)$	$S_{pk}^2(t)$	$S_{k,0}^1(t+1)$	$S_{k,0}^2(t+1)$	Комментарий
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	СчЧ пустой, есть ожидающий читатель и нет ожидающего писателя, поэтому $S_{k,0}^2 = 1$ (чтение)
0	0	1	0	1	0	1	
0	0	1	1	0	0	1	
0	1	0	0	0	1	0	СчЧ пустой, есть ожидающий писатель и нет ожидающего читателя, поэтому $S_{k,0}^1 = 1$ (запись)
0	1	0	0	1	1	0	
0	1	0	1	0	1	0	
0	1	1	0	0	1	0	СчЧ пустой, есть ожидающий писатель и ожидающий читатель, поэтому $S_{k,0}^1 = 1$ (монопольный режим для писателей)
0	1	1	0	1	1	0	
0	1	1	1	0	1	0	

Окончание табл. 6.2

1	2	3	4	5	6	7	8
1	0	1	0	0	0	1	СчЧ не пустой, есть ожидающий читатель и нет ожидающего писателя, поэтому $S^2_{k,0}=1$ (чтение)
1	0	1	0	1	0	1	
1	0	1	1	0	0	1	
1	1	0	0	0	1	0	СчЧ не пустой, есть ожидающий писатель и нет ожидающего читателя, поэтому $S^1_{k,0}=1$ (запись)
1	1	0	0	1	1	0	
1	1	0	1	0	1	0	
1	1	1	0	0	1	0	СчЧ не пустой, есть ожидающий писатель и читатель: – для 1-й строки в предшествующий момент времени не было ни чтения, ни записи, поэтому $S^1_{k,0}=1$ (монопольный режим записи); – для 2-й строки в предшествующий момент времени было чтение, поэтому $S^2_{k,0}=1$ (чтение продолжается); – для 3-й строки в предшествующий момент времени была запись, поэтому она запрещается и $S^2_{k,0}=1$ (чтение), так как не допускается бесконечная запись
1	1	1	0	1	0	1	
1	1	1	1	0	0	1	

При построении такой ПТП учитывались следующие обстоятельства:

– так как события S^1_{pk} и S^2_{pk} не могут существовать в один и тот же момент времени, то для комбинации их значений $S^1_{pk} = S^2_{pk} = 1$ значения событий $S^1_{k,0}$ и $S^2_{k,0}$ будут неопределенными;

– так как события S^1_k и S^2_k по требованию алгоритма взаимодействия процессов должны быть несовместимыми, то в каждой строке ПТП значения для $S^1_{k,0}$ и $S^2_{k,0}$ имеют противоположные значения;

– с целью уменьшения размерности ПТП в ней опущены строки, для которых $S_1 = S_2 = 0$ и $S^1_{pk} = S^2_{pk} = 1$.

С учетом указанных замечаний и в соответствии со словесной формулировкой алгоритма взаимодействия процессов ПТП для событий $S^1_{k,0}$ и $S^2_{k,0}$ будет иметь следующий вид (см. табл. 6.2).

Для получения минимальных выражений для правых частей описания событий $S^1_{k,0}$ и $S^2_{k,0}$ воспользуемся представлением их диаграммами Вейча.

Из диаграмм Вейча (рис. 6.8) минимальные выражения для событий $S^1_{k,0}$ и $S^2_{k,0}$ будут иметь вид:

$$\left.\begin{aligned} S^1_{k,0}(t) &= S_1(\overline{S}_{\text{чо}} \vee \overline{S}_2 \vee \overline{S}^1_{pk}\overline{S}^2_{pk}), \\ S^2_{k,0}(t) &= S_2\left[S_{\text{чо}}(S^1_{pk} \vee S^2_{pk}) \vee \overline{S}_1\right]. \end{aligned}\right\} \qquad (6.27)$$

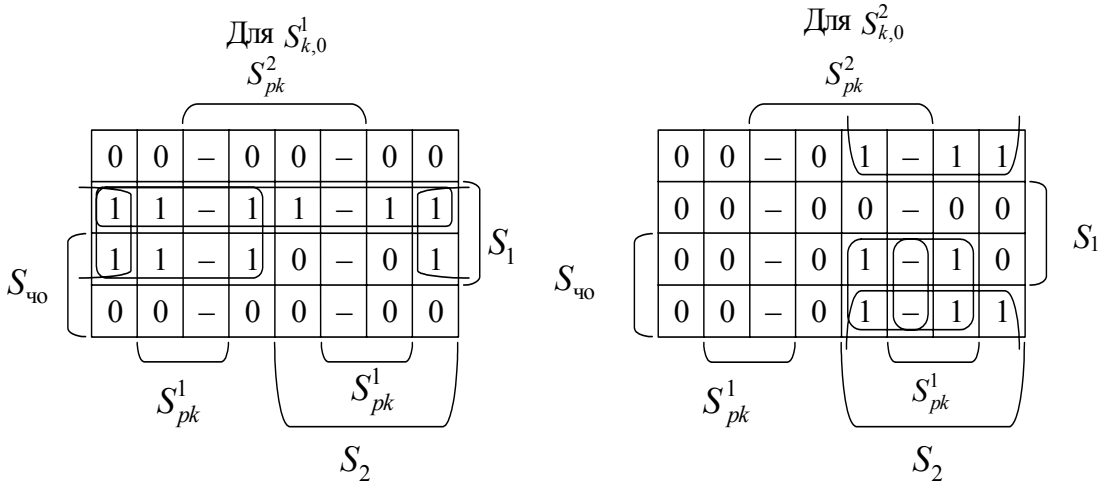


Рис. 6.8. Диаграммы Вейча, представляющие правые части описаний для событий $S^1_{k,0}$ и $S^2_{k,0}$

Учитывая полученные описания (6.24–6.27) для главных событий, формализующих алгоритм управления взаимодействием процессов в задаче «читатели-писатели», общая сводная система канонических уравнений НД СКУ, представляющая рассматриваемый алгоритм управления процессами, будет иметь вид

$$
\begin{aligned}
&S_1(t+1) = S_{1,3}(\overline{S}_k^1 \vee S_n^1) \vee S_1 \overline{S}_k^1,\\
&S_2(t+1) = S_{2,3}(\overline{S}_k^2 \vee S_n^2) \vee S_2 \overline{S}_k^2,\\
&S_k^1(t+1) = S_1 \overline{S}_k^2 (\overline{S}_2 \vee \overline{S}_{\text{чо}} \vee \overline{S}_{pk}^1 \overline{S}_{pk}^2) \vee S_k^1 \overline{\overline{S_n^1}},\\
&S_k^2(t+1) = S_2 \overline{S}_k^1 [\overline{S}_1 \vee S_{\text{чо}}(S_{pk}^1 \vee S_{pk}^2)] \vee S_k^2 \overline{S}_n^2,\\
&S_m(t+1) = S_k^1 S_1 \vee S_k^2 S_2, \qquad (6.28)\\
&S_p^1(t+1) = S_m S_k^1,\\
&S_p^2(t+1) = S_m S_k^2,\\
&S_n^1(t+1) = S_p^1, \qquad S_{pk}^1(t+1) = S_n^1,\\
&S_n^2(t+1) = S_p^2, \qquad S_{pk}^2(t+1) = S_n^2,
\end{aligned}
$$

где S_m – событие, определяющее обращение к РР, как для записи, так и для чтения; S_p^1 и S_p^2 – события, определяющие начало процедуры записи и чтения, соответственно.

П р и м е ч а н и е. Для рассматриваемого варианта алгоритма управления процессами условно принято, что для выполнения процедуры записи и чтения необходимы только по два оператора: для записи – события S_p^1 и S_n^1 и для чтения – события S_p^2 и S_n^2.

При t = 0 или при $\overline{S}_{\text{чо}} S_n^1 = 1$ выполняется операция СчЧ:=E, при $S_{\text{чо}} S_n^2 = 1$ – операция СчЧ:= СчЧ – 1. Представленные на графе (см. рис. 6.7) события S_{gp}^1 и S_{gp}^2 могут инициировать выполнение некоторых подпроцессов за время ожидания работы писателей и читателей [71];

$$
\begin{aligned}
&S_{gp}^1(t+1) = S_1 S_{g,3}^1 \overline{S}_k^1 \vee S_{gp}^1 \overline{S}_n^1,\\
&S_{gp}^2(t+1) = S_2 S_{g,3}^2 \overline{S}_k^2 \vee S_{gp}^2 \overline{S}_n^2,
\end{aligned}
$$

где S_{g3}^1 и S_{g3}^2 – события, определяющие заявки на реализацию подпрограммы для процессов писателей и читателей, соответственно.

Проверку правильности работы алгоритма управления процессами можно выполнить на основе моделирования системы канонических уравнений НД СКУ (6.28), описывающих все представленные в управляющем алгоритме события для различных комбинаций заявок на запись и чтение из разделяемого критического ресурса.

По результатам моделирования можно построить временные диаграммы, в каждом такте которых будут определены события, одновременное существование которых в данном такте возможно.

В качестве примера на рис. 6.9 представлены временные диаграммы для следующих начальных условий: при $t = 0$ / СчЧ:=2, $S_{\text{чо}} = 1,\ S_{1,\text{з}} = S_{2,\text{з}} = 1$.

S_1	S_1^1	S_1^1														
S_2	S_2^1	S_2^1	S_2^1	S_2^1	S_2^1	S_2^1	S_2^1				S_2^2	S_2^2				S_2^3
$S_{\text{чо}}$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
$S_k^1(S_k^2)$		S_k^1	S_k^1	S_k^1	S_k^1		S_k^2	S_k^2	S_k^2	S_k^2		S_k^2	S_k^2	S_k^2	S_k^2	
S_m			S_m					S_m					S_m			
$S_p^1(S_p^2)$				S_p^1					S_p^2					S_p^2		
$S_n^1(S_n^2)$					S_n^1					S_n^2					S_n^2	
$S_{pk}^1(S_{pk}^2)$						S_{pk}^1					S_{pk}^2					S_{pk}^2
Такты	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

а)

S_1	S_1^2	S_1^2	S_1^2	S_1^2	S_1^2	S_1^2				S_1^3	S_1^3				S_1^4	S_1^4
S_2	S_2^3														S_2^4	S_2^4
$S_{\text{чо}}$	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
$S_k^1(S_k^2)$	S_k^2	S_k^2	S_k^2	S_k^2		S_k^1	S_k^1	S_k^1	S_k^1		S_k^1	S_k^1	S_k^1	S_k^1		S_k^2
S_m		S_m					S_m					S_m				
$S_p^1(S_p^2)$			S_p^2					S_p^1					S_p^1			
$S_n^1(S_n^2)$				S_n^2					S_n^1					S_n^1		
$S_{pk}^1(S_{pk}^2)$					S_{pk}^2					S_{pk}^1					S_{pk}^1	
Такты	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

б)

Рис. 6.9. Временные диаграммы работы алгоритма управления взаимодействующими параллельными процессами в задаче «читатели-писатели»: *а* – такты 1–16; *б* – такты 17–32

Проследим работу управляющего алгоритма по временным диаграммам для некоторых частных случаев задания заявок на запись и чтение в процессе работы алгоритма по тактам:

– в 1-м такте фиксируются первые ожидающий писатель ($S_1^1 = 1$) и ожидающий читатель ($S_2^1 = 1$).

Примечание. Верхним индексом у событий S_1 и S_2 будем определять порядковые номера писателей и читателей, соответственно;

– во 2-м такте зарождается событие S_k^1, так как $\overline{S_{pk}^1 \vee S_{pk}^2} = 1$ и ожидающий писатель переходит в следующем такте в ранг работающих писателей (открытый вход в критический участок писателей – $S_k^1 = 1$);

– с 3-го такта начинается процедура обращения к РР ($S_m = 1$) и запрещается восприятие возможной повторной заявки на обращение к РР для записи ($S_1^2 = 0$);

– в 4-м и 5-м тактах выполняется запись в РР;

– в 6-м такте подтверждается факт окончания записи в РР ($S_{pk}^1 = 1$) и осуществляется выход первого писателя из своего критического участка ($S_k^1 = 0$);

– в 7-м такте зарождается событие S_k^2, так как $S_{\text{чо}} S_{pk}^1 = 1$ и первый ожидающий читатель (S_2^1) переходит в ранг работающих читателей, если даже был бы в наличии второй ожидающий писатель (т.е. при $S_1^2 = 1$);

– с 8-го по 10-й такты выполняется обращение к РР с последующим чтением из РР;

– в 11-м такте в СчЧ выполняется операция вычитания единицы (так как в предыдущем такте имело место соотношение $S_n^2 S_{\text{чо}} = 1$) и осуществляется выход первого читателя из своего критического участка ($S_k^2 = 0$);

– с 12-го по 15-й такты осуществляется вход второго ожидающего читателя (S_2^2) в свой критический участок ($S_k^2 = 1$) с последующим обращением ($S_m = 1$) к РР для выполнения чтения;

– в 16-м такте выполняются: операция вычитания единицы из СчЧ, в результате чего событие $S_{\text{чо}}$ становится равным единице ($S_{\text{чо}} = 1$); подтверждается факт окончания чтения из РР ($S^2_{pk} = 1$) и осуществляется выход второго читателя из своего критического участка;

– с 17-го по 20-й такты осуществляется вход уже третьего ожидающего читателя (S^3_2) в критический участок ($S^2_k = 1$) с последующим обращением ($S_m = 1$) к РР для выполнения чтения, так как при зарождении события S^2_k в предшествующий момент времени (16-й такт) отсутствовал ожидающий писатель ($S^2_1 = 0$);

– в 21-м такте подтверждается факт окончания чтения из РР ($S^2_{pk} = 1$) и осуществляется выход третьего читателя из своего критического участка $S^2_k = 0$;

– с 22-го по 25-й такты осуществляется вход второго писателя (S^2_1) в критический участок ($S^1_k = 1$) с последующим обращением ($S_m = 1$) к РР для выполнения записи, так как $C_{\text{чо}} = 0$;

– в 26-м такте в СчЧ заносится константа $E = 2$, подтверждается факт окончания записи в РР ($S^1_{pk} = 1$) и осуществляется выход второго писателя из своего критического участка ($S^1_k = 0$);

– с 27-го по 30-й такты осуществляется вход третьего писателя (S^3_1) в критический участок ($S^1_k = 1$) с последующим обращением ($S_m = 1$) к РР для выполнения записи, так как отсутствовал четвертый ожидающий читатель, и т.д.

6.7. Формализация алгоритма управления взаимодействующими параллельными процессами в задаче «обедающие философы»

Задача «обедающие философы» является примером классической задачи, иллюстрирующей работу параллельных процессов, совместно использующих пересекающие группы ресурсов. Эта задача рассматривается во многих источниках, например, в [75, 76], где условие задачи формулируется следующим образом.

Пять философов, решая свои философские проблемы, время от времени, почувствовав голод, заходят в общую столовую, чтобы подкрепиться. В столовой за круглым столом расставлены пять стульев, каждый из которых занимает определенный философ. В центре стола находится большое блюдо спагетти, а между двумя соседними стульями на столе лежит по одной вилке. Для того, чтобы покушать спагетти, философ должен обязательно взять две вилки, расположенные непосредственно слева и справа от него, и наполнить с их помощью свою тарелку спагетти из общего блюда. После этого философ может приступить к трапезе, используя те же обе вилки. Окончив трапезу, он кладет вилки на свои места и выходит из столовой.

Тупиковая ситуация может возникнуть в том случае, если все пять философов для утоления голода сядут на свои стулья одновременно и каждый из них возьмет одновременно по одной, например, правой вилке. В этом случае никто из философов не сможет приступить к трапезе, так как любому из них для этого необходимы две вилки. Такую конфликтную ситуацию принято называть взаимоблокировкой процессов.

Одно из возможных решений рассматриваемой задачи, которое может позволить избежать взаимоблокировки процессов, основывается на том, чтобы не допускать пребывания в столовой всех пяти философов. В частности, исходя из условий задачи, когда количество ресурсов (вилок) ограничено, трапезой в столовой могут одновременно заниматься лишь два философа (или один). Однако такое обслуживание не позволит избежать другой конфликтной ситуации, называемой взаимоотталкиванием процессов, если не предусмотреть специальных мер, позволяющих избежать такой ситуации. Для рассматриваемой задачи «обедающие философы» взаимоотталкивание процессов может возникнуть в том случае, если два философа по договоренности между собой поочередно занимают вилки слева и справа от третьего философа так, что он никогда не сможет воспользоваться обеими вилками. Это и есть ситуация взаимоотталкивания. Для иллюстрации действующих в задаче «обедающие философы» процессов и возможных связях между ними и ресурсами (вилками) воспользуемся диаграммой, представленной на рис. 6.10.

На данном рисунке введены следующие обозначения: $Ф_1$, …, $Ф_5$ – философы; $В_1$, …, $В_5$ – вилки (ресурсы); $Е_1$, …, $Е_5$ – тарелки философов; БС – общее блюдо спагетти.

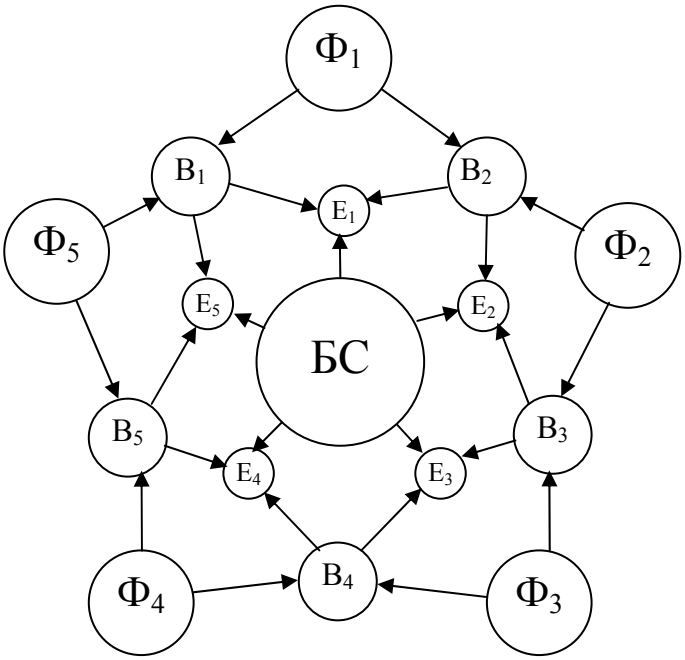


Рис. 6.10. Диаграмма возможных связей действующих лиц–философов и вилок в задаче «обедающие философы»

Для формализации алгоритма управления взаимодействующими процессами, позволяющего избежать тупиковых ситуаций, определим следующий состав событий, которые реализуются в алгоритме управления:

S_3^i – событие, свидетельствующее о том, что *i*-философ почувствовал потребность в утолении голода (заявка *i*-го философа на обслуживание);

S_i – событие, свидетельствующее о том, что заявка на обслуживание *i*-го философа принята (событие, характеризующее *i*-го философа как философа, ожидающего трапезу);

S_k^i – событие, свидетельствующее о том, что ожидающий *i*-й философ перешел в состав обедающих философов (событие, обеспечивающее вход *i*-го процесса в критический интервал);

S_{Ei} – событие, свидетельствующее об активном выполнении *i*-м философом процедуры трапезы;

S_{ik} – событие, свидетельствующее о том, что *i*-й философ закончил трапезу и положил вилки на стол (освободил ресурсы);

S_{Bi} – событие, свидетельствующее о том, что *i*-я вилка взята философом;

S_{pk}^i – событие, свидетельствующее о факте окончания *i*-го процесса.

В соответствии с введенными выше обозначениями событий алгоритм управления взаимодействующими процессами в задаче «обедающие философы» может быть представлен для наглядности следующим фрагментом графа НДА (рис. 6.11).

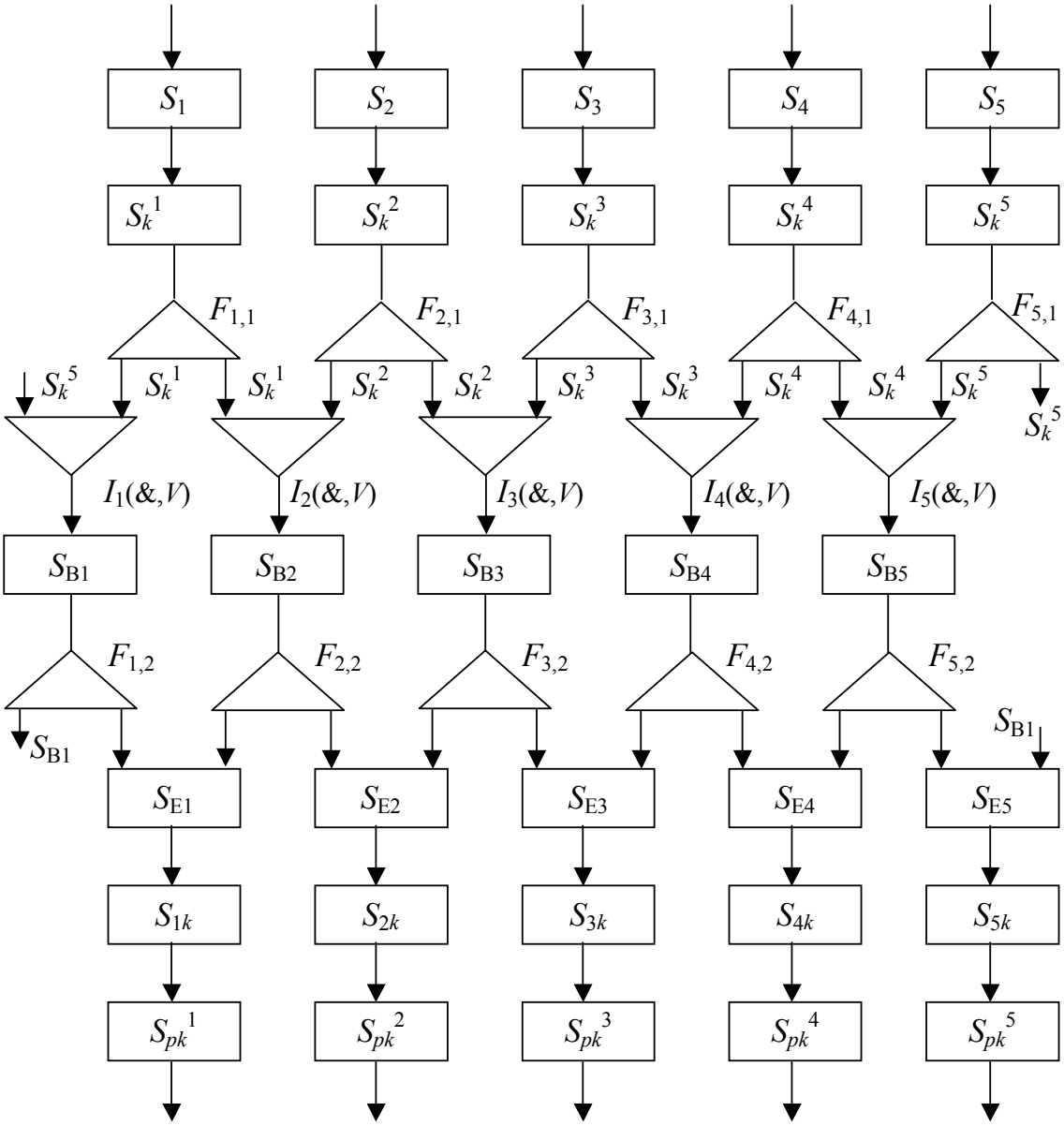


Рис. 6.11. Фрагмент графа НДА алгоритма управления процессами в задаче «обедающие философы»

Граф НДА, представленный на рис. 6.11, несмотря на то, что в нем для простоты опущены логические вершины, позволяет, наряду с системой уравнений для событий, реализуемых алгоритмом, более наглядно представить работу алгоритма управления параллельными процессами, совместно использующими пересекающие группы ресурсов.

Учитывая основные требования алгоритма взаимодействия процессами в задаче «обедающие философы», а также введенный выше состав событий, реализуемых в алгоритме управления, формальное описание такого алгоритма с учетом методики, рассмотренной в [2, 71], можно представить системой канонических уравнений вида (6.29).

В системе (6.29) правая часть всех уравнений рассматривается в момент времени t, но в данной записи системы время t для простоты опущено.

$$
\begin{aligned}
&S_i(t+1) = (S^i_{\text{З}} \vee S_i)\overline{S}^i_k,\\
&S^i_k(t+1) = S_i S^i_{\text{вз}} S^i_{\text{пр}} \vee S^i_k \overline{S}^i_{\text{E}i},\\
&S_{\text{E}i}(t+1) = S_{\text{B}i} S_{\text{B}(i\oplus 1)} S^i_k \vee S_{\text{E}i}\overline{S}_{ik},\\
&S_{\text{B1}}(t+1) = S_1 S^1_k \vee S_5 S^5_k \vee S_{\text{B1}}(S^1_k \overline{S}_{\text{E1}} \vee S^5_k \overline{S}_{\text{E5}}),\\
&S_{\text{B2}}(t+1) = S_2 S^2_k \vee S_1 S^1_k \vee S_{\text{B2}}(S^2_k \overline{S}_{\text{E2}} \vee S^1_k \overline{S}_{\text{E1}}),\\
&S_{\text{B3}}(t+1) = S_3 S^3_k \vee S_2 S^2_k \vee S_{\text{B3}}(S^3_k \overline{S}_{\text{E3}} \vee S^2_k \overline{S}_{\text{E2}}),\\
&S_{\text{B4}}(t+1) = S_4 S^4_k \vee S_3 S^3_k \vee S_{\text{B4}}(S^4_k \overline{S}_{\text{E4}} \vee S^3_k \overline{S}_{\text{E3}}),\\
&S_{\text{B5}}(t+1) = S_5 S^5_k \vee S_4 S^4_k \vee S_{\text{B5}}(S^5_k \overline{S}^5_{\text{E1}} \vee S^4_k \overline{S}_{\text{E4}}),\\
&S_{ik}(t+1) = S_{\text{E}i}\\
&S^i_{pk}(t+1) = S_{ik},
\end{aligned}
\qquad (6.29)
$$

где $\oplus$ – знак сложения по модулю 5.

Рассмотрим структуру отдельных уравнений системы (6.29).

Первое уравнение системы (6.29) обеспечивает восприятие заявки на обслуживание i-го философа, когда он будет представлен в ранге философа, ожидающего трапезу. Заявка будет воспринята и сохранена только в том случае, если будет свободен вход в критический интервал для i-го процесса ($S^i_k = 0$).

Второе уравнение системы (6.29) обеспечивает при $S^i_k = 1$ переход i-го философа, ожидающего трапезу, в ранг обедающих философов, когда будет обеспечен вход в критический интервал для i-го процесса. Условие зарождения события S^i_k представлено конъюнкцией из 3 событий: $S_i, S^i_{\text{вз}}, S^i_{\text{пр}}$.

Событие $S^i_{\text{вз}}$ обеспечивает функцию взаимоисключения процессов при обслуживании философов. Функция $S^i_{\text{вз}}$ определяется

несовместимостью события S_k^i с событиями, формализующими входы в критические интервалы, имеющими номера соседние с *i*-м номером. Для рассматриваемой задачи для любого *i*-го процесса функция $S_{\text{вз}}^i$ определится следующим образом:

$$S_{\text{вз}}^1 = \overline{S}_k^2 \overline{S}_k^5; \qquad S_{\text{вз}}^2 = \overline{S}_k^1 \overline{S}_k^3; \qquad S_{\text{вз}}^3 = \overline{S}_k^2 \overline{S}_k^4.$$
$$S_{\text{вз}}^4 = \overline{S}_k^3 \overline{S}_k^5; \qquad S_{\text{вз}}^5 = \overline{S}_k^1 \overline{S}_k^4;$$

Событие $S_{\text{пр}}^i$ обеспечивает условие приоритетности обслуживания процессов в соответствии с требованиями взаимодействия процессов, исключающих тупиковые ситуации.

В качестве примера в рассматриваемой задаче будем базироваться на циклической дисциплине обслуживания. Схематически такая последовательность обслуживания процессов представлена на рис. 6.12. Здесь в соответствии с условиями задачи одновременно могут активно выполнять трапезу только два философа, к числу которых относятся пары: (Φ_1 и Φ_3), (Φ_2 и Φ_4), (Φ_3 и Φ_5), (Φ_1 и Φ_4) и (Φ_2 и Φ_5).

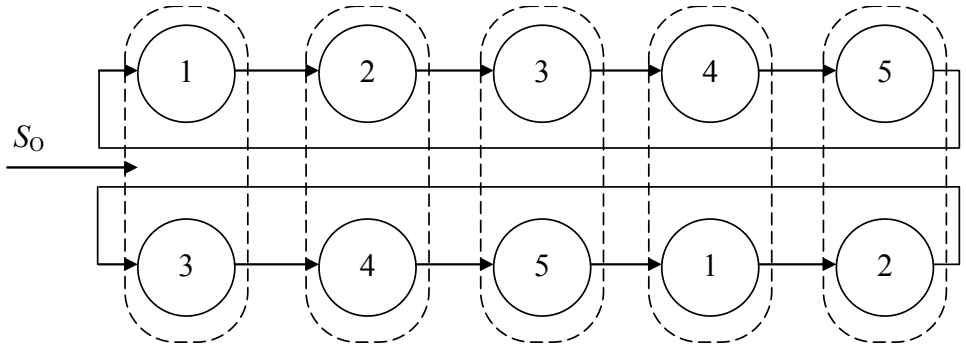


Рис. 6.12. Схема циклической дисциплины обслуживания процессов, принятая в задаче «обедающие философы»

Как видно из рис. 6.12, в одном цикле каждый философ может быть обслужен дважды, если будут соответствующие заявки на обслуживание. В то же время предусматривается обслуживание и по одному философу, если на начало цикла имеет место только одна заявка на обслуживание или только две заявки, но имеющие соседние номера. Условием для такого обслуживания является истинность следующего события:

$$S_y = \bigvee_{i=1}^{5} S_3^i \bigvee_{(\forall S_3^j)(j \neq i)}^{5} \overline{S}_3^j, i = \overline{1,5}. \qquad (6.30)$$

Функцию приоритетности процессов для рассматриваемой дисциплины обслуживания можно представить следующим уравнением:

$$S^i_{\text{пр}}(t+1) = S^i_{\text{пр}}(0) \vee f_i(S^1_3, ..., S^5_3, S^1_{pk}, ..., S^5_{pk}), \qquad (6.31)$$

где $S^i_{\text{пр}}(0)$ – начальный приоритет обслуживания *i*-го процесса.

Исходя из принятой циклической дисциплины обслуживания, выражения (6.31) для любого *i*-го процесса примут следующий вид:

$$\begin{aligned}
S^1_{\text{пр}}(t+1) &= S^1_{\text{пр}}(0) \vee S^1_3(S^5_{pk} \vee \overline{S}^5_3(S^4_{pk} \vee \overline{S}^4_3(S^3_{pk} \vee \overline{S}^3_3(S^2_{pk} \vee \overline{S}^2_3 S^1_{pk})))),\\
S^2_{\text{пр}}(t+1) &= S^2_{\text{пр}}(0) \vee S^2_3(S^1_{pk} \vee \overline{S}^1_3(S^5_{pk} \vee \overline{S}^5_3(S^4_{pk} \vee \overline{S}^4_3(S^3_{pk} \vee \overline{S}^3_3 S^2_{pk})))),\\
S^3_{\text{пр}}(t+1) &= S^3_{\text{пр}}(0) \vee S^3_3(S^2_{pk} \vee \overline{S}^2_3(S^1_{pk} \vee \overline{S}^1_3(S^5_{pk} \vee \overline{S}^5_3(S^4_{pk} \vee \overline{S}^4_3 S^3_{pk})))), \qquad (6.32)\\
S^4_{\text{пр}}(t+1) &= S^4_{\text{пр}}(0) \vee S^4_3(S^3_{pk} \vee \overline{S}^3_3(S^2_{pk} \vee \overline{S}^2_3(S^1_{pk} \vee \overline{S}^1_3(S^5_{pk} \vee \overline{S}^5_3 S^4_{pk})))),\\
S^5_{\text{пр}}(t+1) &= S^5_{\text{пр}}(0) \vee S^5_3(S^4_{pk} \vee \overline{S}^4_3(S^3_{pk} \vee \overline{S}^3_3(S^2_{pk} \vee \overline{S}^2_3(S^1_{pk} \vee \overline{S}^1_3 S^5_{pk})))).
\end{aligned}$$

Учитывая принятую систему обслуживания по одному или по два процесса, в зависимости от комбинации заявок, выражение для $S^i_{\text{пр}}(0)$ будет включать две составляющие:

$$S^i_{\text{пр}}(0) = S^i_{\text{пр},1}(0) \vee S^i_{\text{пр},2}(0), \qquad (6.33)$$

где $S^i_{\text{пр},1}(0)$ – начальный приоритет обслуживания *i*-го процесса при условии, что $S_y = 1$. Тогда система уравнений для $S^i_{\text{пр},1}(0)$ будет иметь вид

$$\begin{aligned}
S^1_{\text{пр},1}(0) &= S_0 x_n S^1_3 S_y,\\
S^2_{\text{пр},1}(0) &= S_0 x_n S^2_3 \overline{S}^1_3 S_y,\\
S^3_{\text{пр},1}(0) &= S_0 x_n S^3_3 \overline{S}^1_3 \overline{S}^2_3 S_y, \qquad (6.34)\\
S^4_{\text{пр},1}(0) &= S_0 x_n S^4_3 \overline{S}^1_3 \overline{S}^2_3 \overline{S}^3_3 S_y,\\
S^5_{\text{пр},1}(0) &= S_0 x_n S^5_3 \overline{S}^2_3 \overline{S}^3_3 \overline{S}^4_3 S_y,
\end{aligned}$$

где S_0 – начальное событие системы управления процессами; x_n – сигнал инициализации системы управления.

Так как *i*-й философ в цикле обслуживания может обслужиться дважды, то выражение для начального приоритета соот-

ветствующего *i*-го процесса при обслуживании по паре философов будет состоять из двух событий:

$$S^{i}_{\text{пр},2}(0) = S^{i,k}_{\text{пр},2}(0) \vee S^{i,m}_{\text{пр},2}(0), \qquad (6.35)$$

где для любого *i*-го процесса значения *k* и *m* определятся, исходя из табл. 6.3.

Таблица 6.3

i	*k*	*m*
1	3	4
2	4	5
3	5	1
4	1	2
5	2	3

События в правой части выражения (6.35) в соответствии с принятой дисциплиной обслуживания имеют вид

$$\begin{aligned}
S^{1,3}_{\text{пр},2}(0) &= S_0 x_n S^1_3 S^3_3, \\
S^{2,4}_{\text{пр},2}(0) &= S_0 x_n S^2_3 S^4_3 (\overline{S}^1_3 \vee \overline{S}^3_3), \\
S^{3,5}_{\text{пр},2}(0) &= S_0 x_n S^3_3 S^5_3 (\overline{S}^2_3 \vee \overline{S}^4_3) \overline{S}^1_3, \\
S^{4,1}_{\text{пр},2}(0) &= S_0 x_n S^4_3 S^1_3 (\overline{S}^2_3 \overline{S}^3_3), \\
S^{5,2}_{\text{пр},2}(0) &= S_0 x_n S^5_3 S^2_3 (\overline{S}^3_3 \overline{S}^4_3).
\end{aligned} \qquad (6.36)$$

Выполнив подстановку в уравнения (6.35) выражений из (6.36), получим систему уравнений для событий, определяющих приоритеты для всех *i*-х процессов на начало работы алгоритма управления процессами для циклической дисциплины обслуживания пар процессов:

$$\begin{aligned}
S^{1}_{\text{пр},2}(0) &= S_0 x_n S^1_3 (S^3_3 \vee S^4_3 \overline{S}^2_3), \\
S^{2}_{\text{пр},2}(0) &= S_0 x_n S^2_3 [S^4_3 (\overline{S}^1_3 \vee \overline{S}^3_3) \vee \overline{S}^3_3 (S^4_3 \vee S^5_3)], \\
S^{3}_{\text{пр},2}(0) &= S_0 x_n S^3_3 [S^1_3 \vee S^5_3 (\overline{S}^2_3 \vee \overline{S}^4_3)], \\
S^{4}_{\text{пр},2}(0) &= S_0 x_n S^4_3 [S^1_3 (\overline{S}^2_3 \vee \overline{S}^3_3) \vee S^2_3 (\overline{S}^1_3 \vee \overline{S}^3_3)], \\
S^{5}_{\text{пр},2}(0) &= S_0 x_n S^5_3 [S^2_3 \overline{S}^3_3 \overline{S}^4_3 \vee S^3_3 (\overline{S}^2_3 \vee \overline{S}^4_3) \overline{S}^1_3].
\end{aligned} \qquad (6.37)$$

Третье уравнение системы (6.29) представляет событие (S_{Ei}), определяющее активное выполнение *i*-м философом трапезы. Это событие зарождается при условии, что *i*-й философ владеет двумя

вилками ($S_{Bi} = S_{B(i\oplus 1)} = 1$) и вход в критический интервал занят i-м процессом ($S_k^i = 1$).

Уравнения системы (6.29) с четвертого по восьмое представляют события (S_{B1}, …, S_{B5}), свидетельствующие о том, что в любой момент времени i-й философ может занять две вилки (ресурсы) с соседними номерами (i-й и ($i \oplus 1$)-й). Этим номерам вилок (ресурсов) соответствуют события, условия зарождения которых определяют выражение $S_i\, S_k^i = 1$.

Отметим некоторые общие особенности систем уравнений для событий, представленных в данной главе.

Для конкретных практических условий время существования событий системы (6.29) можно регулировать за счет коррекции второй составляющей выражений, определяющих соответствующее событие.

В том случае, если после какого-либо такта цикла обслуживания философов в соответствии с заявками обслужился только один философ, то в соответствии с уравнениями (6.32), независимо от комбинаций заявок, обслуживание в последующих тактах будет выполняться только по одному философу. Для повышения эффективности системы управления обслуживанием, если после какого-нибудь такта обслуживания имеет место комбинация заявок, не удовлетворяющая условию $S_y = 1$, следует произвести повторную инициализацию системы управления, для которой будут работать уравнения системы (6.37).

Глава 7
ФОРМАЛЬНОЕ АНАЛИТИЧЕСКОЕ ОПИСАНИЕ АЛГОРИТМОВ УПРАВЛЕНИЯ ПАРАЛЛЕЛЬНЫМИ ПРОЦЕССАМИ В МНОГОПРОЦЕССОРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ ПРИ ИСПОЛЬЗОВАНИИ РАЗЛИЧНЫХ МЕХАНИЗМОВ СИНХРОНИЗАЦИИ

При разработке и создании систем логического управления (СЛУ) информационными процессами и ресурсами при реализации вычислительных задач или задач промышленной автоматики и управления объектами формальным методам описания управляющих алгоритмов (УА) придается большое значение. Это связано с тем, что такие формальные методы должны обеспечивать комплексное решение задач спецификации, разработки, верификации, реализации и анализа сложных систем управления, в том числе систем управления взаимодействующими параллельными процессами и ресурсами в вычислительных системах и сетях [85, 86]. Математический аппарат формальных методов описания УА должен обладать при этом высокой эффективностью, позволяющей решать в том числе задачи синхронизации процессов и задачи, связанные с повышением производительности и надежности систем управления.

Под высокой эффективностью математического аппарата формального языка описания УА понимается выполнение следующих основных требований [55]:

– простота, наглядность, компактность и удобство использования для заказчика и разработчика;

– высокая степень выразительности при описании сложных вариантов алгоритмов, в том числе с отражением временного фактора, а также включающих как синхронные, так и асинхронные процессы с отсутствием тупиковых ситуаций, когда имеется зависимость реализуемых событий по данным и ресурсам;

– наличие методов оптимизационных равносильных преобразований, связанных с минимизацией, композицией, декомпозицией и верификацией УА на моделях и возможностью простой трансформации описания с одного языка на другой [18, 85];

– простой переход от формального аналитического представления к структурной реализации УА, в том числе аппаратно

(схемно или микропрограммно), программно или в виде их комбинаций;

– математический аппарат должен быть ориентирован на описание параллельно-последовательных алгоритмов;

– при простоте математического аппарата языка должно быть обеспечено описание широкого круга применений по управлению процессами и объектами, начиная с торговых автоматов и кончая операционными системами с разделяемыми ресурсами [55].

В работах известных ученых, в том числе в [91, 92], отмечаются большие достоинства автоматного представления алгоритмов ЛУ, к числу которых относятся большое удобство, компактность и простота формального представления УА, по сравнению с представлением их на основе языка сетей Петри. Однако такое достоинство автоматного представления УА авторы относят только для представления систем последовательного действия, учитывая при этом использование моделей только детерминированных автоматов (ДА). Для систем параллельного действия авторы отдают предпочтение сетям Петри, не учитывая при этом основное достоинство событийных моделей недетерминированных автоматов (СНДА), заключающееся в скрытом в них параллелизме, который проявляется в том, что под действием одного и того же входного сигнала допускается переход от одного частного события к нескольким событиям, одновременное существование которых в соответствии с УА возможно, т.е. описание переходов в УА при использовании моделей НДА выполняется не в терминах состояний ДА, а в терминах частных событий НДА, что и обеспечивает такую же простоту, компактность и удобство описания УА в параллельных системах, которые характерны для описания алгоритмов систем последовательного действия при использовании моделей ДА.

Формальный язык представления УА на основе логики СНДА можно рассматривать и как вариант базового языка, который включает две составляющие: графическую форму для наглядного представления алгоритмов в виде граф-схем с параллельными ветвями (ГСАП) и аналитическую в виде системы рекуррентных канонических уравнений (СКУ), описывающих все реализуемые в алгоритме частные события. При этом под частными событиями, в зависимости от уровня детализации УА, понимается: для вычислительных алгоритмов – выполнение некоторых актов обработки информации, к которым могут относиться выполнение микроопераций и макроопераций, функциональных операторов и подпрограмм и т.д., а для управляющих алгоритмов промышленной

автоматики и объектов – выполнение отдельных технологических операций или групп операций. В дальнейшем аналитическую форму представления управляющего алгоритма в виде СКУ будем для краткости называть языком НД СКУ, который является аналитической интерпретацией языка ГСАП [16].

В дальнейшем будут рассмотрены вопросы представления основных свойств систем управления параллельными процессами в виде моделей событийного НДА, а также методы использования различных механизмов синхронизации для формального описания систем управления, обеспечивающих требуемую надежность и эффективность таких систем.

7.1. Модели событийных НДА для формального представления основных свойств систем управления параллельными процессами и ресурсами, обеспечивающих надежность и эффективность таких систем

Формальное аналитическое описание основных свойств систем управления необходимо, с одной стороны, для аналитического представления алгоритмов управления параллельными процессами и ресурсами, а с другой – для верификации таких алгоритмов. При этом основные свойства систем управления должны быть такими, чтобы обеспечивалось выполнение основных характеристик таких систем, к числу которых относятся надежность и эффективность систем. В связи с этим для реализации таких систем управления и их верификации необходимо использовать такой математический аппарат, который, с одной стороны, обеспечивал бы формальное аналитическое представление сложных алгоритмов управления, а с другой – отличался бы простотой и компактностью.

В качестве такого математического аппарата в данной работе предлагается использовать модели событийных недетерминированных автоматов (СНДА), которые обладают следующими достоинствами [85]. Во-первых, модели СНДА отличаются значительной простотой и компактностью, так как описание свойств систем управления выполняется не в терминах состояний детерминированных автоматов (ДА), а в терминах частных событий, реализуемых в системах управления, учитывая при этом, что чис-

ло таких частных событий (m) СНДА будет значительно меньше числа состояний (M) эквивалентного ему ДА, так как $M \le 2^m$. Во-вторых, использование моделей СНДА предоставляет широкие возможности для описания сложных алгоритмов управления параллельными процессами и ресурсами, так как основой таких моделей является практически сеть, вершины которой отождествляются с некоторыми событиями, а дуги – соотношениями между этими событиями.

Учитывая необходимость обеспечения высокой выразительности описания сложных алгоритмов управления, включающих как синхронные, так и асинхронные процессы с отсутствием тупиковых ситуаций при использовании общих ресурсов, в данной работе рассматриваются два варианта описания систем управления: для первого варианта описание реализуемых событий зависит от времени в явном виде, а для второго варианта описание реализуемых событий зависит от времени не в явном виде.

7.1.1. Основные свойства систем управления параллельными процессами и ресурсами и их формальное описание, когда события в явном виде зависят от времени

Для увеличения производительности решения сложных вычислительных задач или задач промышленной автоматики такие задачи обычно разбиваются на параллельные ветви. В связи с этим возникает необходимость в реализации синхронизации взаимодействия таких параллельных ветвей (процессов) при обращении к общим критическим ресурсам или при обмене сообщениями. Для этой цели используют различные механизмы синхронизации процессов, к числу которых относятся: критические интервалы, мониторы, кольцевые буферы, рандеву и др.

Для обеспечения надежности и эффективности таких систем управления параллельными процессами и ресурсами к используемым механизмам синхронизации предъявляют необходимые требования, которые являются основой определения и реализации основных свойств таких систем [86].

В последующих разделах подробно представлена методика формализации алгоритмов управления взаимодействующими процессами, базирующимися на использовании механизмов крити-

ческих интервалов при обращении к общему ресурсу. В данном разделе коротко представим формальное описание событий, определяющих вход процессов в свой критических интервал (и нахождение в нем), которые должны обладать следующими основными свойствами: справедливость, достижимость, взаимоисключение и приоритетность.

События, определяющие вход *i*-го процесса в свой критический интервал (S_k^i), можно представить таким образом:

$$S_k^i(t+1) = S_{\text{вп,к}}^i S_{\text{вз,к}}^i S_{\text{пр,к}}^i \vee S_k^i \overline{S}_{\text{пк}}^i. \qquad (7.1)$$

В уравнениях (7.1) первая составляющая, представленная конъюнкцией событий, определяет вход *i*-го процесса в свой критический интервал. Эти события и определяют отмеченные выше свойства, в том числе и свойства справедливости и достижимости, которые определяются событиями типа $S_{\text{вп}}^i$. Эти события для используемых механизмов синхронизации должны быть представлены в следующем виде:

$$S_{\text{вп,к}}^i(t+1) = (S_3^{\text{i}} \vee S_{\text{вп,к}}^i)\overline{S}_{\text{к}}^i, \qquad (7.2)$$

где $S_{\text{вп,к}}^i$ – события, свидетельствующие о восприятии заявки *i*-го процесса (события S_3^i) к общему критическому ресурсу и ее сохранении только в том случае, когда данный процесс не находится в своем критическом интервале. Это означает, что ни один процесс не будет бесконечно долго ждать входа в свой критический интервал.

Вторая составляющая уравнений (7.1), представленная конъюнкцией событий, определяет время нахождения *i*-го процесса в своем критическом интервале. В данном случае событие $S_{\text{пк}}^i$ свидетельствует об окончании одноразовой операции *i*-го процесса с общим критическим ресурсом.

Свойство достижимости событий $S_{\text{к}}^i$ обеспечивается наличием события типа $S_{\text{вп}}^i$, которое является обязательным непосредственно предшествующим для таких событий. Необходимо отметить, что проверить наличие достижимости всех частных событий, реализуемых в системах управления, можно путем построения прямой таблицы переходов (ПТП) для этих событий по определенному правилу. Это правило заключается в том, что очередная строка ПТП должна начинаться с представлением такого события, которое уже было отмечено в одной из предшествующих строк ПТП [85].

Свойство взаимоисключения событий типа $S^i_{\text{к}}$ обеспечивается входом в одно и то же время в свой критический интервал только одного *i*-го события из *n*-процессов. Это свойство формализуется следующими комбинационными событиями:

$$S^i_{\text{вз,к}}(t+1) = \underset{(\forall\alpha)[\alpha\neq i]}{\Lambda}\overline{S}^{\alpha}_{\text{к}} \quad . \tag{7.3}$$

Свойство – наличие приоритетности процессов – обеспечивает однозначность входа *i*-го процесса в свой критический интервал. Наличие приоритетности событий особенно необходимо в начальный период при обращении процессов к общему ресурсу, когда могут быть истинными все события типа $S^i_{\text{вз}}$ и $S^i_{\text{вп}}$.

В качестве примера рассмотрим приоритетность *i*-го процесса для циклической дисциплины обслуживания $S^i_{\text{пр}}$. Это событие можно представить следующим выражением:

$$S^i_{\text{пр}}(t+1) = S^i_{\text{пр}}(0) \vee S^i_{\text{вп}} S^i_{\text{т}}, \tag{7.4}$$

где $S^i_{\text{пр}}(0)$ представляет собой сокращенное обозначение комбинационного события, определяющего начальный приоритет обслуживания *i*-го процесса:

$$S^i_{\text{пр}}(0) = S_0 x_n S^i_{\text{вп}} S^i_{\text{т}} \underset{(\forall\alpha)[\alpha<i]}{\Lambda} S^{\alpha}_{\text{вп}} \quad , \tag{7.5}$$

$$S_0(t+1) = x_0 \vee S_0 \overline{x}_n,$$

где x_n – сигнал инициализации системы управления; x_0 – сигнал приведения системы управления в начальное состояние; $S^i_{\text{т}}$ – сокращенное обозначение комбинационного события, определяющего приоритет *i*-го процесса при наличии воспринятой заявки в повторных циклах циклической дисциплины обслуживания;

$$S^i_{\text{т}} = \underset{i=1}{\overset{n}{\text{V}}} (S^i_{pk} \Lambda \overline{S}^{\alpha}_{\text{вп}}), \tag{7.6}$$

где S^i_{pk} – сокращенное обозначение события, определяющего выход *i*-го процесса из критического интервала после окончания процедуры обращения к разделяемым данным.

7.1.2. Основные свойства систем управления параллельными процессами и ресурсами и их формальное описание, когда события не зависят в явном виде от времени

В работе известного ученого Э. М. Кларка [18] рассматривается язык временной темпоральной логики, когда порядок событий в системе описывается без привлечения времени в явном виде. В данном разделе рассматриваются вопросы описания основных свойств таких систем на основе использования моделей СНДА. В связи с этим будут показаны широкие возможности языка СНДА для описания сложных систем управления, когда события, представленные в таких системах, не зависят в явном виде от времени. К числу основных свойств таких событий относятся следующие: достижимости, достижимости в будущем «рано или поздно», инвариантности, условного ожидания и разблокировки. Эти свойства соответствуют пяти основным операциям темпоральной логики $CTL{\rightarrow}(X)$, (F), (G), (U), (R).

Свойство достижимости «в следующий момент» (X), означающее, что любое событие S_j будет достижимо тогда, когда имеет место событие, непосредственно предшествующее ему, т.е. такое свойство уже было рассмотрено при описании событий в гл. 1 и 2.

Свойство «рано или поздно» или «когда-то в будущем» (F). В данном случае таким свойством будет обладать некоторое событие S_j, которое будет истинным только в том случае, когда станет истинным некоторое событие S_p (определяющее условие зарождения события S_j), время наступления которого точно не определено. Учитывая это обстоятельство, вводят вспомогательное событие S_r, которое для события S_j будет играть роль непосредственно предшествующего события. Тогда описания таких событий будут иметь вид

$$\begin{aligned} S_j(t+1) &= S_r S_p \vee S_j \overline{S_{j,\mathrm{c}}}, \\ S_r(t+1) &= (S_{r,\mathrm{З}} \vee S_r)\overline{S_j}, \end{aligned} \tag{7.7}$$

где $S_{r,\mathrm{З}}$ – событие, определяющее зарождение события S_r; $S_{j,\mathrm{c}}$ – событие, определяющее условие сохранения события S_j.

Свойством инвариантности или «всегда, повсюду» (G) будет обладать некоторое событие S_j, которое будет истинным всегда (повсюду) для всех n непосредственно предшествующих событий (S_j) для события (S_j), т.е. (S_j) можно представить так:

$$S_j(t+1) = \overset{n}{\underset{i=1}{V}}(S_i)S_{j,3} \vee S_j\overline{S_{j,c}}, \tag{7.8}$$

где $S_{j,3}$ – событие, определяющее условие зарождения события S_j.

Свойство условного ожидания или «до тех пор, пока» (U). В данном случае таким свойством будет обладать некоторое событие S_j, которое не будет истинным долго – до тех пор, пока не будет истинным некоторое событие $S_{\text{р}}$. При этом каждое из n непосредственно предшествующих событий (S_r^i) (для события S_j) после зарождения события S_i станет равным нулю $(S_r^i = 0)$, т.е. каждое событие типа S_r^i тоже будет обладать отмеченным S_r свойством «до тех пор, пока». Учитывая это обстоятельство, события S_j и S_r^i будут иметь вид

$$\begin{gathered} S_r^i(t+1) = (S_{r,3}^i \vee S_r^i)\overline{S_{\text{р}}}, \\ S_j(t+1) = \overset{n}{\underset{i=1}{V}}(S_r^i)S_{\text{р}} \vee S_j\overline{S_{j,c}}. \end{gathered} \tag{7.9}$$

Свойство разблокировки «высвободить» (R). В данном случае свойство «высвободить» относится к некоторому событию S_j, которое после зарождения будет существовать (т.е. остается истинным) до тех пор, пока не зародится некоторое событие S_r, которое, в свою очередь, может никогда и не зародиться, если событие, реализующее его зарождение, $S_{\text{р}} = 0$. Учитывая это, описание событий S_j и S_r будет иметь вид:

$$\begin{gathered} S_j(t+1) = (S_iS_{j,3} \vee S_j)\overline{S_r}, \\ S_r(t+1) = S_jS_{\text{р}} \vee S_r\overline{S_{r,c}}, \end{gathered} \tag{7.10}$$

где S_i – событие, являющееся непосредственно предшествующим событию S_j.

Учитывая полученные результаты формального описания на языке НД СКУ основных свойств систем управления параллельными процессами и ресурсами, которые соответствуют пяти основным операторам логики CTL, можно отметить, что язык НД СКУ по существу является одной из разновидностей языка временной темпоральной логики CTL.

Это означает, что такое представление систем управления параллельными процессами и ресурсами соответствует представлению таких систем в виде обобщенной канонической формы, которая позволяет расширить выразительные возможности языка НД СКУ с учетом связи не только по управлению и информации, но и по данным и ресурсам с учетом временных факторов для асинхронного режима работы. Такая обобщенная каноническая форма представления алгоритмического управления в виде математической модели НДА Мура будет иметь следующий вид:

$$S_j^{Y_j}(t+1) = \bigvee_{i,j} S_{i,\text{З}} S_i \vee \bigvee_j S_{j,\text{с}} S_j, \qquad (7.11)$$

где $S_{i,\text{З}}$ и $S_{j,\text{с}}$ – комбинационные события, определяющие условия зарождения и сохранения события S_j, соответственно, которые можно представить так:

$$S_{j,\text{З}}(t+1) = X_{i,j} R_i,\ S_{j,\text{с}} = X_{i,j} R_j, \qquad (7.12)$$

где $X_{i,j}$ – совокупность некоторых частных входных сигналов входного алфавита $[X]$; R_i и R_j являются конъюнкциями переменных, представляющих реализуемые в параллельных ветвях некоторые частные события, влияющие на условия зарождения и сохранения события S_j, соответственно, т.е. такие события можно представить так:

$$R_i = \tilde{S}_{i,1}\tilde{S}_{i,2}...\tilde{S}_{i,k},\ R_j = \tilde{S}_{j,1}\tilde{S}_{j,2}...\tilde{S}_{j,r}, \qquad (7.13)$$

где ~ – знак, означающий, что переменные (частные события), входящие в R_i и R_j, могут быть взяты с отрицанием или без него.

События, входящие в R_i и R_j, могут иметь место как внутри рассматриваемого управляющего блока, так и вне его (в последнем случае события представляются как внешние входные сигналы). К их числу могут относиться также такие события, которые при параллельной обработке информации определяют взаимоис-

ключение событий типа S_j и их приоритетность, а также события, отражающие временной фактор, определяющий продолжительность, начало и конец обработки информации, характерной для асинхронного режима работы.

Необходимо также отметить, что один из возможных тривиальных вариантов отражения асинхронности режима работы системы логического (СЛУ) управления может быть реализован за счет расширения входного алфавита $[X]$ путем ввода двоичных входных сигналов, которые сигнализируют о готовности входной информации (исходных данных) для ее обработки и готовности блоков системы для приема результатов обработки, а также сигналов о завершении обработки исходных данных.

Для асинхронного режима работы характерно использование ждущих и временных логических условий [26]. В этом случае переход от события S_i к событию S_j в зависимости от значения ждущего или временного логического условия осуществляется путем ввода дополнительного пустого события S_r, которое имитирует ожидание появления истинности некоторого логического условия $S_{\text{р}}$, чтобы осуществить переход к событию S_j. В данном случае под логическим условием $S_{\text{р}}$ можно понимать или выполнение некоторого временного события, например, $t_{\text{н}} \le T \le t_{\text{в}}$, или окончание каких-либо действий, которые вызывают переход к событию S_j. Тогда описываемые события будут иметь вид

$$S_j(t+1) = S_r S_{\text{р}},\ S_r(t+1) = (S_i \vee S_r)\overline{S}_{\text{р}}. \qquad (7.14)$$

Анализируя выразительные возможности структуры модели НД СКУ для описания УА, определяющих поведение системы во времени, когда порядок выполнения событий описывается без привлечения времени в явном виде, можно утверждать, что язык НД СКУ является по существу одной из разновидностей языка временной темпоральной логики типа CTL [18]. Такое утверждение может быть доказано путем представления основных операторов языка CTL на языке НД СКУ, они имеют простой вид и стандартную форму в виде системы канонических уравнений [95].

В качестве примера рассмотрим представление оператора разблокировки *R* («высвободить») языка CTL на языке НД СКУ. Оператор *R* требует, чтобы некоторое второе свойство выполнялось на пути (т.е. управляющего алгоритма) вплоть до такого со-

стояния, в котором соблюдается некоторое первое свойство. Однако первое свойство необязательно должно когда-нибудь быть выполнено. Для этого оператора введем два события S_r и S_j, определяющие первое и второе свойство, соответственно, тогда эти события можно представить следующими уравнениями:

$$S_r(t+1) = S_j S_{\text{р}} \vee S_r \overline{S}_k\,,\ S_j(t+1) = S_i S_{i,3} \vee S_j \overline{S}_r\,, \qquad (7.15)$$

где S_i – событие, непосредственно предшествующее событию S_j; S_k – событие, определяющее условие сохранения события S_r; S_r – событие, отмеченное первым свойством; это событие может никогда не зародиться, если событие $S_{\text{р}} = 0$.

Необходимо отметить, что, учитывая отмеченные выразительные возможности языка НД СКУ, свойственные языку временной темпоральной логики, этот язык может быть успешно использован для верификации управляющих алгоритмов. Для этого исследуемые свойства управляющего алгоритма описываются в виде формальной модели на языке логики НД СКУ и выполняют ее проверку путем моделирования этой модели, преобразованной в программный код, например, на языке VHDL и SMV [94, 96]. В частности, предлагаемая обобщенная каноническая форма представления УА в виде НД СКУ позволяет естественно и просто переводить описание алгоритма на языки программирования, например, на язык VHDL, когда требуется выполнить моделирование алгоритма для его верификации или аппаратную реализацию, например, на ПЛИС [93]. Действительно, при представлении исходного УА в виде программы на языке VHDL правые части всех уравнений исходной СКУ НДА преобразуются в коды, которые имеют такой же вид, что и исходные уравнения, но только с другими обозначениями переменных и операций над ними, которые указываются в начале программы. Например, для события типа S_k^i входа процесса в критический интервал имеем:

$$S_k^i(t+1) = S_{\text{вп}}^i S_{\text{вз}}^i S_{\text{пр}}^i \vee S_k^i \overline{S_{\text{р}}^i}, \qquad (7.16)$$

$$sk \le (svp \wedge svz \wedge spr) or (sk \wedge \overline{sp}).$$

В то же время к предлагаемой стандартной форме представления УА в виде СКУ модели НДА может быть преобразовано описание управляющих алгоритмов, представленных на широко

известных начальных языках, к числу которых относятся, например, язык регулярных выражений алгебры событий (РВАС), язык исчисления предикатов первого порядка, язык операторных схем алгоритмов с параллельными ветвями (ГСАП) и др. [16]. В предыдущей главе подробно изложены методы и алгоритмы преобразования описания событий исходных систем, заданных на таких начальных языках, на описание в виде стандартной системы канонических уравнений (СКУ) модели НДА. Покажем это преобразование на примере описания события $S_j^{Y_j}$, включающего одну ветвь на языке НД СКУ. Представление этого описания на языке РВАС для свернутой и развернутой формы имеет вид

$$S_j^{Y_j} = S_i S_{i,3} \left\{ S_{i,\mathrm{c}} \right\} = S_i S_{i,3} \vee S_j S_{j,\mathrm{c}}. \qquad (7.17)$$

Представление события $S_j^{Y_j}$ на основе языка исчисления предикатов первого порядка с ограниченными кванторами и временной схемой формирования этого события (рис. 7.1) имеет вид

$$S_j^{Y_j}(t) = (\exists \tau)_{\tau \le t} S_{j,3}(\tau) \,\&\, \underset{\tau < \tau_1 \le t}{\forall} S_{j,\mathrm{c}}(\tau_1) \,\&\, S_i(\tau - 1). \qquad (7.18)$$

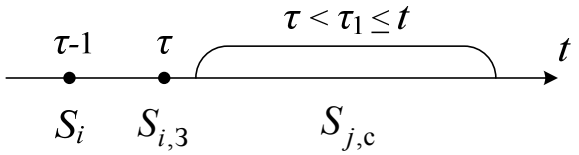


Рис. 7.1. Временная схема формирования события $S_j^{Y_j}$

Как видно из (7.13), описание события S_j представлено на основе использования трех основных операций: конкатенации, дизъюнкции и итерации, а описание (7.14) – с использованием кванторов существования и всеобщности и операции конъюнкции.

После введения дискретного времени и замены знака конкатенации на знак конъюнкции уравнение (7.14) примет вид, соответствующий языку НД СКУ:

$$S_j^{Y_j}(t+1) = S_i(t) \,\&\, S_{j,3}(t) \vee S_j(t) \,\&\, S_{j,\mathrm{c}}(t).$$

7.2. Формализация алгоритмов управления параллельными процессами с использованием механизма монитора и согласующего кольцевого буфера

В параллельных программах неизбежно возникают межпроцессные взаимодействия, проявляющиеся в виде обмена данными между параллельными ветвями алгоритма или параллельными процессами (потоками). Для такого обмена существует множество механизмов, таких как разделяемая память, именованные и неименованные каналы. Все эти механизмы реализуются программно в составе операционной системы, что приводит к снижению производительности операционной системы. Перенесение части функций межпроцессного обмена в аппаратную часть системы может существенно понизить латентность системы, повысить пропускную способность, а также освободить процессор для выполнения прикладной задачи.

Любой процесс обмена происходит с использованием буфера некоторого объема, причем буфером может быть либо разделяемая область в оперативной памяти, либо объект операционной системы. В каждом случае процесс-передатчик формирует данные и записывает их в буфер, т.е. является производителем, процесс-приемник читает их из него, т.е. является потребителем.

Будем считать, что используемая модель программирования системы обмена сообщениями является многопоточной. Отсюда следует, что в многопроцессорной системе возможно существование независимых потоков, в том числе и потоков процесса-передатчика. В частности, один поток может формировать данные, а другой поток записывать их в согласующий буфер. Такая постановка приводит к усложнению процедуры управления процессами и требует дополнительных средств ее реализации, однако является наиболее общей.

Для решения задач управления взаимодействующими параллельными процессами используют, как было отмечено ранее, различные механизмы синхронизации процессов, в том числе и механизм монитора [88], который был впервые предложен в 1974 г. Хоаром [55] и Бринчем Хансеном [98] для организации синхронизации процессов при обращении к общим критическим ресурсам.

Применение такого механизма синхронизации процессов по сравнению с механизмами критических интервалов с использованием семафоров позволяет существенно повысить надежность и компактность программного обеспечения реализации информационных систем.

В данном разделе рассматривается методика формального описания алгоритма управления взаимодействующими параллельными процессами при обмене сообщениями с использованием механизма мониторов и кольцевого буфера в качестве общего критического ресурса. В связи с этим рассмотрим вначале методику функционирования кольцевого согласующего буфера при обмене сообщениями между производителем и потребителем, позволяющую определить условия, для которых буфер будет полон, пуст или занимает промежуточное состояние.

7.2.1. Функционирование кольцевого согласующего буфера при обмене сообщениями между производителем и потребителем

Производитель помещает сообщения в разделяемый кольцевой буфер [87], потребитель извлекает их оттуда. Буфер содержит очередь уже помещенных, но еще не извлеченных сообщений. Эта очередь может быть представлена связным списком или массивом.

Для представления очереди сообщений используются две целочисленные переменные, которые указывают, соответственно, на первую заполненную и первую пустую ячейку буфера. Эти переменные организуются с помощью счетчиков буфера записи СчБз и счетчика буфера чтения СчБчт. Показания счетчика СчБз будет определять номер первой незаполненной (пустой) ячейки, а счетчика СчБчт – номер первой заполненной ячейки. Для этой цели при инициализации системы управления в указанные счетчики записываются единицы, т.е. при $t = 0$ / СчБз = 1; СчБчт = 1.

Если принять количество ячеек в кольцевом буфере, например, за $n = 8$, то счетчики буфера будут работать по модулю 8. Тогда при переполнении счетчиков может быть образован сигнал переполнения, который может быть использован для

определения состояний буфера, для которых буфер будет или полон, или пуст.

Для нашего варианта кольцевого буфера событие, когда буфер будет полон ($S_{бп}$ = 1), будет иметь место при условиях равенства значений счетчиков буфера (СчБз = СчБчт > 1) и был перенос из счетчика буфера записи (СчБз = 0).

Событие, когда буфер будет пуст ($S_{бо}$ = 1), будет иметь место также при условии равенства значений счетчиков буфера (СчБз = СчБчт) и не было сигнала переноса из СчБчт или при условии, когда в процессе работы счетчики буфера примут исходное состояние (СчБз = СчБчт = 1) и был перенос из СчБчт.

Рассмотрим некоторые варианты записи и чтения в кольцевой буфер.

Представим буфер массивом, в котором имеется n = 8 ячеек (рис. 7.2), а в каждую ячейку записывается одно сообщение.

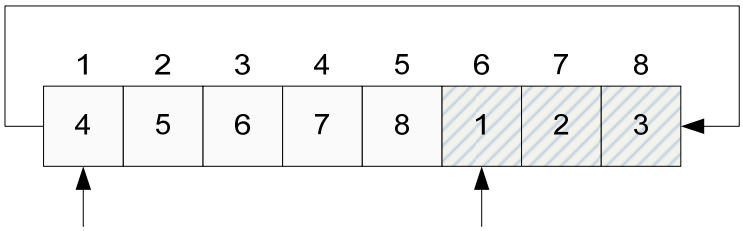


Рис. 7.2. Произвольное положение массива в буфере

Будем считать, что номер ячейки буфера совпадает с номером сообщения. Тогда верхний ряд цифр (см. рис. 7.2) указывает на исходное состояние буфера перед началом работы системы, когда все ячейки пусты.

Кольцевой буфер при записи информации работает следующим образом. Такт записи в одну ячейку буфера состоит из двух частей: в первой части осуществляется непосредственная запись в ячейку, номер которой указывает счетчик СчБз, во второй части производится кольцевой сдвиг содержимого буфера влево. На рис. 7.2 показаны результаты записи трех сообщений, когда положение счетчика СчБз стало равным 4, так как в исходном состоянии в СчБз была записана 1. Таким образом, ячейка буфера под номером 4 будет первой пустой ячейкой после сообщений в конце очереди.

Пусть после трех тактов записи выполняется чтение из буфера до полной его очистки. Тогда состояние счетчиков буфера

записи и чтения будет одинаковым: СчБз = СчБчт = 4. Это свидетельствует о том, что буфер пуст, так как не было сигнала переполнения из СчБчт.

Допустим, что после этого было выполнено 5 тактов записи в буфер. Тогда состояние буфера будет иметь вид (рис. 7.3), а состояние счетчика записи после 5 тактов определится сложением по модулю 8, т.е. СчБз = 4⊕5 = 1, а состояние счетчика чтения не изменится – СчБчт = 4.

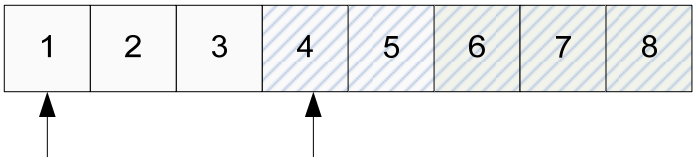


Рис. 7. 3. Состояние буфера

Пусть в следующем такте выполняется запись в буфер трех порций информации, тогда состояние буфера примет вид (рис. 7.4).

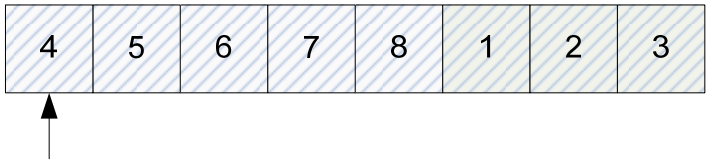


Рис. 7.4. Состояние буфера

Для этого состояния буфер будет полон, так как состояния счетчиков после операции записи имеют вид: СчБз = 1 + 3 = 4; СчБчт = 4 и был перенос из счетчика записи (в предыдущем такте записи).

В следующем такте возможна лишь операция чтения, так как буфер полон. Пусть считывается 5 порций информации. Тогда состояние буфера будет иметь вид (рис. 7.5).

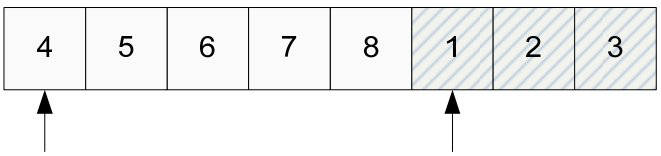


Рис. 7.5. Состояние буфера

Показания счетчиков буфера после операции чтения имеют следующие значения: СчБз = 4; СчБчт = 4 ⊕ 5 = 1.

7.2.2. Формализация основных событий, реализующих алгоритм управления параллельными процессами в решаемой задаче «производитель-потребитель» при использовании механизма монитора и языка СНДА

Для формального описания основных событий алгоритма управления процессами в решаемой задаче необходимо сформулировать основные свойства и требования к монитору, чтобы обеспечить бесконфликтную реализацию взаимодействия процессов (отсутствие тупиков) и свойства справедливости реализации операций над критическим ресурсом. Такие основные свойства монитора были сформулированы, например, в [97] таким образом:

а) чтобы изменить состояние общего ресурса, процесс должен войти в монитор. При этом только один процесс имеет исключительное право доступа в монитор, а вне монитора известны только имена процедур, которые могут изменить состояние общего ресурса, и неизвестно состояние общего ресурса;

б) внутри монитора процесс не может обращаться к переменным, объявленным вне монитора;

в) постоянные переменные, характеризующие состояние ресурса, инициируются до вызова его процедур.

Учитывая эти свойства, можно сформулировать основные требования к механизму монитора, которые в принципе идентичны требованиям к критическим интервалам, обеспечивающим отсутствие конфликтных ситуаций (свойство безопасности) и справедливость доступа процессов к общему ресурсу, которые были определены Деккером и Дейкстра [62, 70]. Таким образом, эти требования можно представить следующим образом:

1. В любой момент времени только один процесс может находиться внутри монитора.

2. Ни один процесс не может оставаться внутри монитора бесконечно долго.

3. Ни один процесс не должен бесконечно долго ждать входа в монитор.

4. Процесс, вышедший из монитора не обслуженный, имеет преимущественное право на повторный вход в монитор по сравнению с процессами, находящимися в начальной очереди и требующими выполнить такую же процедуру над общим ресурсом.

Учитывая указанные свойства и требования к монитору, можно сформулировать условия зарождения и сохранения основ-

ных событий, реализующих алгоритм управления процессами при обращении к общему ресурсу на основе использования языка СНДА следующим образом.

Первое и второе требование к механизму монитора обеспечивается реализацией событий, позволяющих войти в монитор и находиться в нем в любой момент времени только одному процессу, который не может бесконечно долго находиться в нем. Описание таких событий ($S^i_{\text{М}}$) будет состоять из двух частей: первая составляющая определяет зарождение события, определяющего начальный вход *i*-го процесса в монитор ($S^i_{\text{М,З}}$), а вторая – его сохранение в мониторе ($S^i_{\text{М,С}}$), т.е. имеем

$$S^i_{\text{М}}(t+1) = S^i_{\text{М,З}} \vee S^i_{\text{М,С}}, \tag{7.19}$$

где первая составляющая уравнения (7.18), в свою очередь, должна определяться конъюнкцией трех событий:

$$S^i_{\text{М,З}} = S^i_{\text{вп}} S^i_{\text{вз}} S^i_{\text{пр}}, \tag{7.20}$$

где $S^i_{\text{вп}}$ – событие, определяющее прием заявки *i*-го процесса на обслуживание; $S^i_{\text{вз}}$ – событие, обеспечивающее взаимоисключение событий $S^i_{\text{М}}$; $S^i_{\text{пр}}$ – событие, определяющее приоритет *i*-го процесса.

Вторая составляющая уравнения (7.19), в свою очередь, должна определяться конъюнкцией из трех событий:

$$S^{\text{i}}_{\text{М,С}} = S^i_{\text{М}} \overline{S^i_n} \overline{S^i_{\text{кр,з}}}, \tag{7.21}$$

где S^i_n – событие, свидетельствующее об окончании одноразовой операции *i*-го процесса с общим критическим ресурсом; $S^i_{\text{кр,з}}$ – событие, свидетельствующее о том, что критический ресурс для *i*-го процесса занят (буфер полон – $S_{\text{бп}} = 1$) или (буфер пуст – $S_{\text{бо}} = 1$).

Рассмотрим формальное описание отдельных составляющих в уравнениях (7.19) и (7.20).

Событие $S^i_{\text{вп}}$ обеспечивает реализацию ***третьего требования*** к механизму монитора, основанного на том, что заявка *i*-го процесса на обслуживание воспринимается и сохраняется только Эв том случае, когда данный процесс не находится в мониторе. Тогда это событие формализуется следующим образом:

$$S^i_{\text{вп}}(t+1) = (S^i_{\text{З}} \vee S^i_{\text{вп}})\overline{S^i_{\text{м}}}\,\overline{S^i_{\text{мп}}}, \qquad (7.22)$$

где $S^i_{\text{З}}$ – заявка i-го процесса; $S^i_{\text{мп}}$ – событие, определяющее повторный вход i-го процесса в монитор, вышедшего из монитора не обслуженным, т.е. это событие обеспечивает ***четвертое требование*** к механизму монитора и будет иметь следующий вид:

$$S^i_{\text{мп}}(t+1) = S^i_{\text{мн}} S^i_{\text{кр},0} \vee S^i_{\text{мп}} S^i_n, \qquad (7.23)$$

где $S^i_{\text{мн}}$ – событие, свидетельствующее о выходе i-го процесса из монитора не обслуженным; $S^i_{\text{кр},0}$ – событие, свидетельствующее об освобождении критического ресурса для операции с ним i-го процесса после выполнения предыдущей операции.

Событие $S^i_{\text{вз}}$ является основным событием, определяющим вход в монитор в одно и то же время только одного i-го процесса, и определяется взаимоисключением событий входа процессов в монитор на основе их несовместимости с i-м процессом. Это событие представляется следующей формулой:

$$S^i_{\text{вз}} = \Lambda_{(\forall\alpha)[\alpha\neq i]}(\overline{S^\alpha_{\text{М}} \vee S^\alpha_{\text{мп}}}). \qquad (7.24)$$

Событие $S^i_{\text{пр}}$ обеспечивает однозначность входа процессов в монитор, используя их приоритетность, которая определится следующей формулой:

$$S^i_{\text{пр}} = S^i_{\text{пр}}(0) \vee S^i_{\text{вп}} S^i_{\text{кр},0}\overline{S^i_{\text{мн}}}, \qquad (7.25)$$

где $S^i_{\text{пр}}(0)$ – событие, определяющее приоритет i-го процесса в начальный период. Введение этого события объясняется тем обстоятельством, что в начальный период обращения процессов к общему ресурсу могут быть истинны все события типа $S^i_{\text{вз}}$ и $S^i_{\text{вп}}$.

В дальнейшем при описании всех событий алгоритма управления при обращении к критическому ресурсу будем базироваться в качестве примера на вариант решения задачи взаимодействия для двух процессов: один производитель и один потребитель, заявки которых будут фиксироваться в двух очередях – одна начальная очередь, а другая очередь ждущих заявок процессов, вышедших из монитора не обслуженными. Для этого варианта полная система уравнений, определяющая доступ процессов

к критическому ресурсу и выполнение операций над ними, будет иметь следующий вид:

$$S^1_{\text{м}}(t+1) = S^1_{\text{вп}} S^1_{\text{вз}} S^1_{\text{пр}} \vee S^1_{\text{м}} (\overline{S^1_n \vee S^1_{\text{кр,з}}}),$$

$$S^2_{\text{м}}(t+1) = S^2_{\text{вп}} S^2_{\text{вз}} S^2_{\text{пр}} \vee S^2_{\text{м}} (\overline{S^2_n \vee S^2_{\text{кр,з}}}),$$

$$S^1_{\text{мп}}(t+1) = S^1_{\text{мн}} S^1_{\text{кр,0}} \vee S^1_{\text{мп}} \overline{S}^1_n,$$

$$S^2_{\text{мп}}(t+1) = S^2_{\text{мн}} S^2_{\text{кр,0}} \vee S^2_{\text{мп}} \overline{S}^2_n,$$

$$S^1_{\text{вп}}(t+1) = (S^1_{\text{З}} \vee S^1_{\text{вп}})(\overline{S^1_{\text{м}} \vee S^1_{\text{мп}}}),$$

$$S^2_{\text{вп}}(t+1) = (S^2_{\text{З}} \vee S^2_{\text{вп}})(\overline{S^2_{\text{м}} \vee S^2_{\text{мп}}}),$$

$$S^1_{\text{пр}} = S^1_{\text{пр}}(0) \vee S^1_{\text{вп}} S^1_{\text{кр,0}} \overline{S^1_{\text{мн}}},$$

$$S^2_{\text{пр}} = S^2_{\text{пр}}(0) \vee S^2_{\text{вп}} S^2_{\text{кр,0}} \overline{S^2_{\text{мн}}},$$

$$S^1_{\text{пр}}(0) = S^1_0 x_n S^1_{\text{вп}}, \qquad (7.26)$$

$$S^1_{\text{кр,з}} = S_{\text{бп}}, \; S^2_{\text{кр,з}} = S_{\text{бо}},$$

$$S^1_{\text{мн}}(t+1) = S^1_{\text{м}} S^1_{\text{кр,з}} \vee S^1_{\text{мн}} \overline{S^1_{\text{кр,0}}},$$

$$S^2_{\text{мн}}(t+1) = S^2_{\text{м}} S^2_{\text{кр,з}} \vee S^2_{\text{мн}} \overline{S^2_{\text{кр,0}}},$$

$$S^1_{\text{кр,0}}(t+1) = S^2_n \vee S^1_{\text{кр,0}} (\overline{S^1_{\text{мп}} \vee S^1_{\text{м}}}),$$

$$S^2_{\text{кр,0}}(t+1) = S^1_n \vee S^2_{\text{кр,0}} (\overline{S^2_{\text{мп}} \vee S^2_{\text{м}}}),$$

$$S^1_{\text{рз}}(t+1) = (S^1_{\text{м}} \vee S^1_{\text{мп}}) S^1_{\text{кр,с}} \vee S^1_{\text{рз}} (\overline{S_{\text{бп}} \vee S_{\text{кз}}}),$$

$$S^2_{\text{рч}}(t+1) = (S^2_{\text{м}} \vee S^2_{\text{мп}}) S^2_{\text{кр,с}} \vee S^2_{\text{рз}} (\overline{S_{\text{бп}} \vee S_{\text{кз}}}),$$

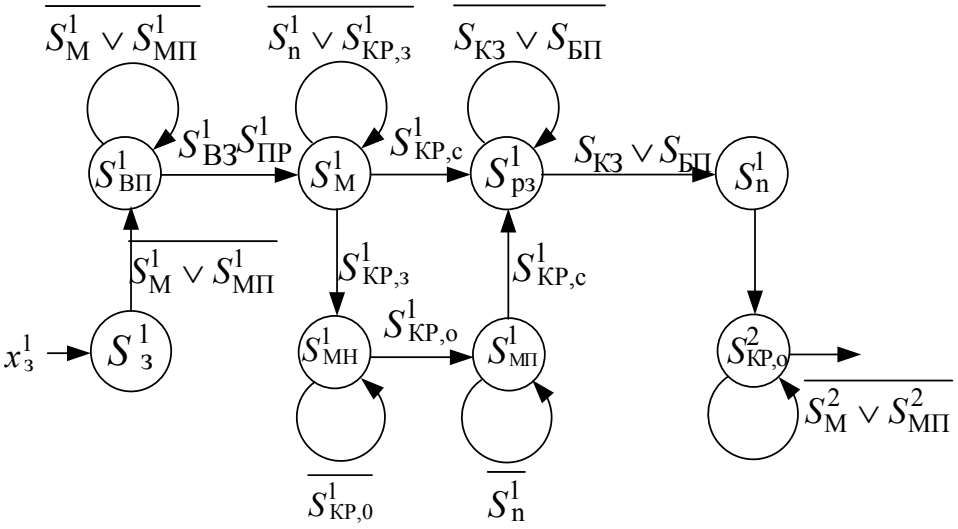
$$S^1_{\text{кр,с}} = (\overline{S_{\text{бп}}} \vee S_{\text{бо}}), \; S^2_{\text{кр,с}} = (S_{\text{бп}} \vee \overline{S_{\text{бо}}}),$$

$$S^1_n(t+1) = S^1_{\text{рз}} (S_{\text{кз}} \vee S_{\text{бп}}), \; S^2_n(t+1) = S^2_{\text{рз}} (S_{\text{кч}} \vee S_{\text{бо}}),$$

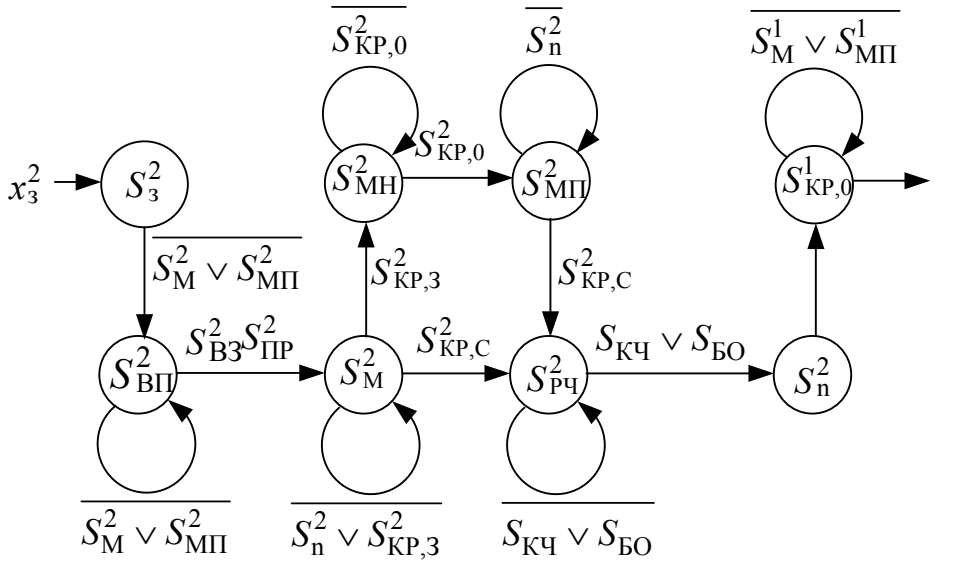
где $S^1_{\text{рз}}$ и $S^2_{\text{рч}}$ – события, определяющие выполнение операции записи и чтения, соответственно; $S^1_{\text{кр,с}}$ и $S^2_{\text{кр,с}}$ – комбинационные события, определяющие возможность выполнения операции записи и чтения, соответственно; $S_{\text{кз}}$ и $S_{\text{кч}}$ – события, свидетельствующие

об окончании операции записи и чтения порций информации, определяемых счетчиками $S_{ч*з}$ и $S_{ч*чт}$, соответственно.

Учитывая представленное аналитическое описание всех основных частных событий (уравнения (7.26)), реализующих управляющий алгоритм взаимодействия процессов в решаемой задаче, можно для наглядности представить рассматриваемый алгоритм в виде графа СНДА (рис. 7.6).



а)



б)

Рис. 7.6. Графы СНДА алгоритмов управления процессов: *а* – производителя; *б* – потребителя, при их обращении к общему ресурсу (буферу сообщений) с использованием механизма монитора

Необходимо отметить, что полученные уравнения описания алгоритма управления параллельными процессами в виде стандартной системы рекуррентных бескванторных предикатных формул, представляющих все основные частные события в решаемой задаче, отличаются следующими положительными характеристиками:

1) обеспечивается выполнение всех основных требований к монитору, а именно: отсутствие тупиковых ситуаций (свойство безопасности) и учет реализации свойства справедливости;

2) возможна простая трансформация описания алгоритма управления процессами на язык VHDL для моделирования и верификации алгоритма и аппаратной реализации алгоритма управления с использованием ПЛИС, что, в свою очередь, обеспечивает более высокую надежность и производительность систем управления;

3) если сравнить формальное описание основных частных событий управления процессами в решаемой задаче с использованием механизма монитора и механизма критических интервалов (участков), то можно отметить, что для второго варианта число уравнений, представляющих алгоритм управления процессами, будет значительно больше, так как каждый процесс должен иметь свой критический участок, который организует обращение процесса к критическому ресурсу. Следовательно, для первого варианта при использовании монитора структурная реализация алгоритмов управления процессами в решаемой задаче будет отличаться значительной простотой по сравнению с использованием механизмов критических участков для второго варианта.

7.3. Формализация алгоритмов управления параллельными процессами на основе использования механизма «рандеву»

В данном разделе рассматривается методика формальной спецификации алгоритма управления межпроцессорного взаимодействия в классической задаче «о спящем парикмахере», которая является примером классических задач синхронизации процессов. Эта задача иллюстрирует отношение типа «клиент–сервер», которое имеет место между процессами в многопроцессорных вычислительных системах, где для нее используется особый тип синхронизации, называемый «рандеву» [89].

7.3.1. Концептуальная модель задачи «о спящем парикмахере»

В общем виде задача «о спящем парикмахере» заключается в следующем. В одном из городков есть парикмахерская с одной входной и одной выходной дверями, в которой размещены несколько кресел для посетителей и одно кресло парикмахера, в котором он обычно сидит и спит, если в салоне отсутствуют клиенты. В это же кресло садится и клиент, когда его стрижет парикмахер. Салон парикмахерской по размерам мал, и ходить по нему может только парикмахер и один посетитель. Когда посетитель приходит и видит спящего парикмахера, то он будит его и, подождав, когда он освободит кресло, садится в него, после чего парикмахер может приступить к его стрижке. Если парикмахер занят стрижкой, когда приходит посетитель, то последний или садится в одно из свободных кресел и засыпает, или уходит, если нет свободных кресел. После стрижки парикмахер открывает посетителю выходную дверь и закрывает ее за ним. В том случае если в салоне есть ожидающие посетители, то парикмахер будит одного из них и ждет, пока тот сядет в его кресло. Если посетителей нет, то парикмахер садится в свое кресло и спит до прихода следующего посетителя.

7.3.2. Аналитическая модель алгоритма управления процессами в задаче «о спящем парикмахере» на основе использования СНДА

Рассматривая посетителей и парикмахера как взаимодействующие процессы, для которых посетитель – это клиент, запрашивающий сервис у парикмахера, а парикмахер – это сервер, обеспечивающий данный сервис, можно представить данный тип взаимодействия как пример отношений «клиент–сервер».

Действия парикмахера и посетителя необходимо синхронизировать таким образом, чтобы обеспечить парикмахеру и посетителю встречу – рандеву, когда парикмахер должен дождаться прихода посетителя, а посетитель – освобождения парикмахера. При этом посетителю необходимо ждать, пока парикмахер закончит его стричь, после чего он освобождает кресло парикмахера и ждет открытия двери парикмахером. Парикмахер, в свою очередь, перед тем как закрыть дверь, должен подождать, пока уйдет посе-

титель. Таким образом, парикмахер и посетитель проходят через последовательность синхронизированных этапов, начинающихся с рандеву.

Для формального описания алгоритма взаимодействия процессов в данной задаче будем использовать язык систем канонических уравнений. Для этой цели введем следующие основные частные события, реализуемые в данном алгоритме [89]:

$S_{\text{З}}^{k}$ – событие, определяющее приход нового клиента в парикмахерскую;

$S_{\text{оч}}^{k}$ – событие, определяющее наличие места в очереди к парикмахеру;

S_{0}^{k} и $S_{\text{зд}}^{n}$ – события, определяющие нахождение клиента и парикмахера в салоне парикмахерской, соответственно;

$S_{\text{бп}}^{k}$ и $S_{\text{бп}}^{n}$ – события, свидетельствующие о том, что клиент будит парикмахера, а парикмахер будит клиента, соответственно;

$S_{\text{кп}}^{k}$ и $S_{\text{ок}}^{n}$ – события, свидетельствующие о том, что клиент сел в кресло парикмахера, а парикмахер освободил свое кресло, соответственно;

$S_{\text{с}}^{k}$ и $S_{\text{с}}^{n}$ – события, свидетельствующие о том, что клиент и парикмахер спят, соответственно;

$S_{\text{гт}}^{k}$ и $S_{\text{гт}}^{n}$ – события, свидетельствующие о том, что клиент и парикмахер готовы к обслуживанию, соответственно;

S_{n}^{k} и S_{k}^{n} – события, свидетельствующие о начале и окончании стрижки, соответственно;

$S_{\text{ок}}^{k}$ и $S_{\text{од}}^{n}$ – события, свидетельствующие о том, что клиент освободил кресло парикмахера, а парикмахер открыл выходную дверь, соответственно;

$S_{\text{у}}^{k}$ и $S_{\text{ЗД}}^{n}$ – события, свидетельствующие о том, что клиент ушел из парикмахерской, а парикмахер закрыл выходную дверь, соответственно.

На основании словесно представленного алгоритма управления процессами в задаче о спящем парикмахере и введенных событий, реализуемых в этом алгоритме, система канонических уравнений, описывающих эти события, и соответствующий им граф недетерминированного автомата (НДА) будут иметь следующий вид:

– для процесса – клиент:

$$S_0^k(t+1) = S_{\text{З}}^k S_{\text{оч}}^k,$$
$$S_{\text{с}}^k(t+1) = S_0^k \overline{S_{\text{с}}^n} \vee S_{\text{с}}^k \overline{S_{\text{бк}}^n},$$
$$S_{\text{бп}}(t+1) = S_0^k S_{\text{с}}^n, \qquad (7.27)$$
$$S_{\text{кп}}^k(t+1) = S_{\text{с}}^k S_{\text{бк}}^n \vee S_{\text{бп}}^k S_{\text{ок}}^n,$$
$$S_{\text{гт}}^k(t+1) = S_{\text{кп}}^k S_{\text{ЗД}}^n \vee S_{\text{гт}}^k S_{\text{гт}}^n;$$

– для процесса – сервер (парикмахер):

$$S_{\text{с}}^n(t+1) = S_0^n \overline{S_0^k} \vee S_0^n \overline{S_{\text{бп}}^k},$$
$$S_{\text{бк}}^n(t+1) = S_0^n S_0^k S_{\text{с}}^k, \qquad (7.28)$$
$$S_{\text{ок}}^n(t+1) = (S_0^n S_0^k \overline{S_0^k} \vee S_{\text{с}}^{\text{n}}) S_{\text{бп}}^k,$$
$$S_{\text{гт}}^n(t+1) = (S_{\text{ок}}^n \vee S_{\text{бк}}^n) S_{\text{кп}}^k \vee S_{\text{гт}}^n \overline{S_{\text{гт}}^k}.$$

Система канонических уравнений, описывающая события после рандеву:

$$S_{\text{н}}(t+1) = S_{\text{гт}}^k S_{\text{гт}}^n, \quad S_{\text{од}}^n(t+1) = S_{\text{к}},$$
$$S_{\text{к}}(t+1) = S_{\text{н}}, \quad S_{\text{у}}^k(t+1) = S_{\text{ок}}^k S_{\text{од}}^k, \qquad (7.29)$$
$$S_{\text{ок}}^k(t+1) = S_{\text{к}}, \qquad S_{PL}^n(t+1) = S_{\text{од}}^m S_{\text{у}}^k.$$

В приведенных выше системах канонических уравнений и графе НДА (рис. 7.7) были опущены события, соответствующие пустым событиям обратной связи в циклах из логических условий. Эта связь символизирует ожидание некоторого события S_i истинности логического условия, при котором реализуется переход к событию S_j, являющемуся первым преемником события S_j. Например, рассмотрим переход от события $S_{\text{ок}}^k$ к событию $S_{\text{у}}^k$, для которого выход условной вершины $S_{\text{од}}^n$ соединен с ее входом.

Введя пустую операторную вершину в обратную связь и обозначив ее символом S_e, получим следующие уравнения:

$$S_e(t+1) = (S_{\text{ок}}^k \vee S_{\text{с}}) S_{\text{од}}^n;$$
$$S_{\text{у}}^k(t+1) = (S_{\text{ок}}^k \vee S_{\text{с}}) S_{\text{од}}^n. \qquad (7.30)$$

Аналогичные уравнения могут быть представлены и для других переходов, для которых выход условной вершины соединен с ее входом.

Граф недетерминированного автомата алгоритма управления взаимодействующими процессами в задаче «спящий парикмахер» имеет следующий вид (см. рис. 7.7).

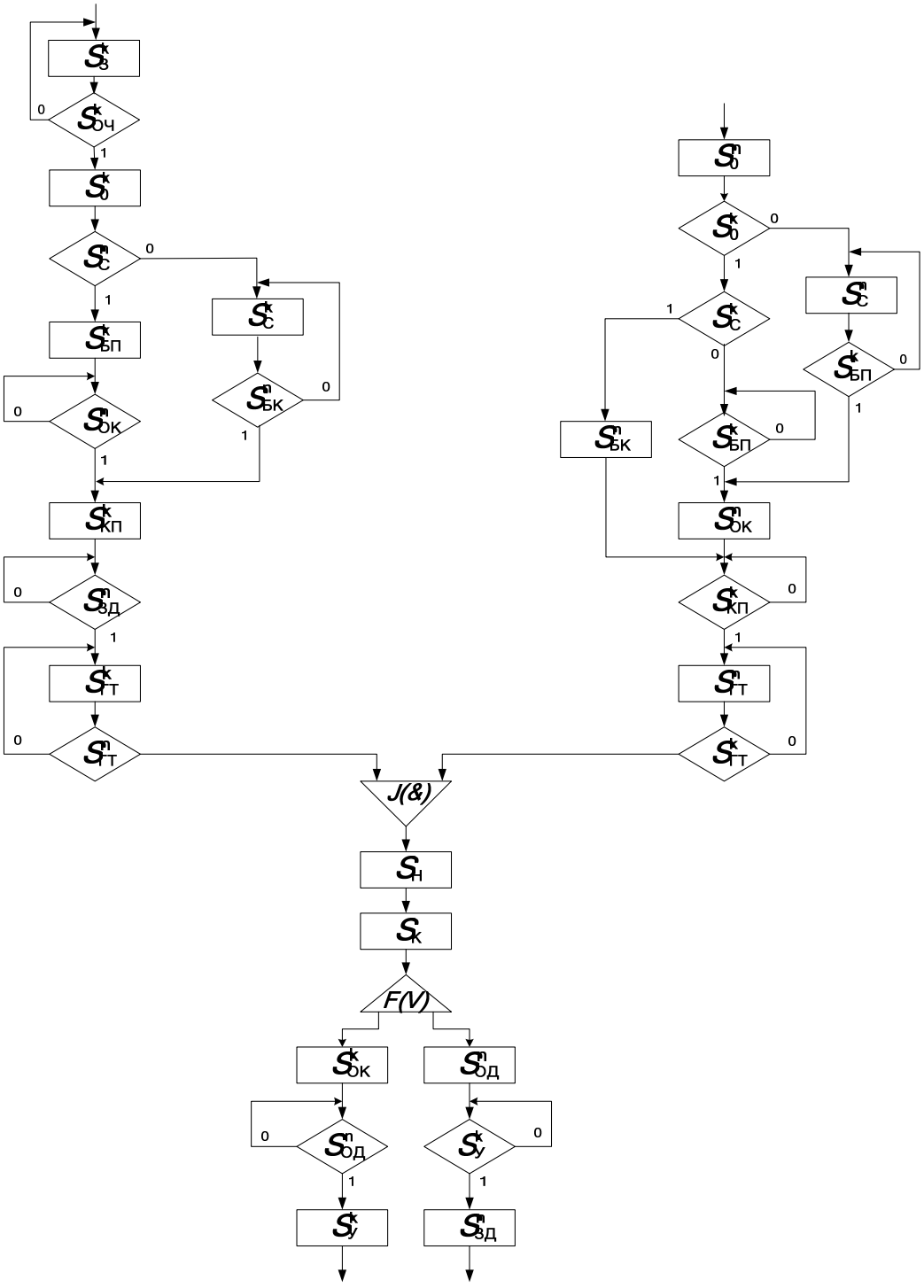


Рис. 7.7. Граф НДА алгоритма управления взаимодействующими процессами в задаче «спящий парикмахер»

Рассмотренная методика применения механизма «рандеву» для реализации алгоритма управления параллельными процессами может быть успешно использована для организации взаимодействия процессами в вычислительных системах, реализуемых по технологии «клиент–сервер», при разработке устройств организационной поддержки трудоемких функций операционных систем и, в частности, диспетчера задач многопроцессорной системы. Решение этих вопросов рассматривается в следующей главе.

Глава 8
АППАРАТНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМОВ УПРАВЛЕНИЯ ПРОЦЕССАМИ И РЕСУРСАМИ В МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ, ПРЕДСТАВЛЕННЫХ МОДЕЛЯМИ НДА

8.1. Проблемы управления процессами и ресурсами в многопроцессорных системах

Архитектура современных операционных систем (ОС) содержит типичные компоненты, которые наиболее часто используются выполняющимися задачами. К ним относятся механизмы межпроцессорного взаимодействия, управления памятью, планирования и диспетчеризации процессов, обработка прерываний. Значительная часть функций этих компонент реализуется программным путем. Высокая частота использования компонент ОС приводит к большим непроизводительным временным затратам, которые могут достигать нескольких десятков процентов от общего времени реализации процессов, порождаемых пользовательскими программами [100].

Уже достаточно давно наметилась тенденция к переходу от программной реализации алгоритмов ОС к аппаратной, которая вызвана стремлением к увеличению отношения производительность/стоимость ЭВМ и обусловлена высоким развитием микроэлектроники. Действительно, аппаратная реализация алгоритмов ОС позволяет уменьшить время его выполнения и, следовательно, повысить производительность ЭВМ. С другой стороны, стоимость дополнительного оборудования для аппаратной реализации, благодаря прогрессу элементной базы, с каждым годом падает. Кроме того, аппаратная реализация алгоритмов ОС позволяет в определенной степени уменьшить трудности создания программного обеспечения, а также улучшить совместимость различных версий ОС для компьютеров разных платформ. Кроме того, в значительной степени устраняется трудоемкая операция отладки нижних уровней ОС и повышается корректность всего программного обеспечения, благодаря лучшей формализации функций, реализованных аппаратным способом по сравнению с программным способом [100].

Наиболее чувствительно издержки ОС, основанные на программном обеспечении, сказываются на системах реального времени (СРВ). СРВ требуют высокой реакции на запрос, что, в свою очередь, вызывает необходимость повышения производительности операционной системы и особенно систем жесткого реального времени.

Традиционное управление процессами и ресурсами, осуществляемое ядром операционной системы (ОС) на основе системного программного обеспечения, приводит к значительным замедлениям выполнения приложений и, следовательно, к существенным потерям производительности вычислительной системы. Эта проблема вызвана наличием трудоемких компонент ядра ОС, к которым относятся: диспетчеры задач, связанные с переключением процессов; планировщики задач, связанные с выделением и освобождением ресурсов; средства взаимодействия процессов, такие как каналы передачи сообщений; примитивы синхронизации и другие API-функции.

В однопроцессорных операционных системах межпроцессные коммуникации (IPC) ориентированы на локальные взаимодействия, т.е. между адресными пространствами на той же самой машине. Средства IPC традиционно реализуются в ядре, однако у такого подхода существуют проблемы. Приложения, которые выполняются с использованием быстро выполняющихся потоков, сталкиваются с проблемой эффективности, поскольку при взаимодействии на уровне ядра эффективность ограничена высокой трудоемкостью вызова функций ядра и переключения процессора из одного адресного пространства в другое. На мультипроцессорах с разделяемой памятью эта проблема может быть решена путем перемещения средств коммуникации и механизмов синхронизации из ядра на пользовательский уровень в пределах адресного пространства. При этом производительность увеличивается, но возникают проблемы надежности и безопасности из-за ограниченных возможностей современных средств тестирования программного обеспечения [102].

Эффективность функций планирования и диспетчеризации определяется скоростью переключения процессов. Планировщик и диспетчер в многопроцессорной системе вызываются приложениями гораздо чаще, чем в однопроцессорной. Логично предположить, что в многопроцессорных системах планировщик и диспетчер должны работать быстрее, чтобы не создавать задержек.

Однако возникает обратная картина, когда накладные расходы растут вместе с ростом числа процессоров в системе. Так между запрашивающими процессорами возникают конфликты за доступ к диспетчеру задач, в результате чего создаются очереди запрашивающих процессоров. На устранение конфликтов требуется дополнительная синхронизация работы процессоров. Задача планирования и диспетчеризации в многопроцессорных системах, в отличие от однопроцессорных, является двумерной, поскольку, кроме функции назначения задачи, необходимо выполнять функцию выделения целевого процессора. Кроме того, в планировщиках с общей очередью существует явление перезагрузки кэш-памяти, связанное с переключением задач, когда прерванная задача с высокой вероятностью может быть направлена для продолжения обслуживания в другой процессорный узел. Названное явление увеличивает частоту кэш-промахов и неизбежно приводит к снижению производительности многопроцессорной системы.

Анализ зарубежных и отечественных публикаций за последние несколько лет показывает, что главные недостатки программного обеспечения, на котором базировались традиционные операционные системы, в том числе реального времени, могут быть устранены путем реализации функций ядра аппаратными средствами. Считается, что основной недостаток аппаратно-ориентированных операционных систем заключается в низкой гибкости, под которой понимается отсутствие возможности настройки функций ядра на изменяющиеся условия обслуживания разнохарактерных пользовательских запросов (высокоприоритетные, низкоприоритетные, бесприоритетные, высокоскоростные, низкоскоростные, фоновые и др.). При выполнении таких приложений применяют разные критерии, например, минимизация времени выполнения работ, максимизация реакции системы и др., что требует настройки (реконфигурации) ОС на оптимальный метод планирования (временное разделение, пространственное разделение, круговой выбор (алгоритм RR и его модификации), приоритетный выбор (алгоритмы RM, EDF, LSTF) и др.).

Существуют два подхода к аппаратно-ориентированной реализации алгоритмов управления процессами и ресурсами в многопроцессорных (многоядерных) системах: микропрограммная и схемная.

Микропрограммный способ (см. гл. 5) предполагает внесение обоснованно выбранных функций управления процессами и

ресурсами в систему команд целевого процессора. В настоящее время микропрограммно реализуют неизменяемые функции ОС, которые способны работать в пользовательском пространстве и могут вызываться выполняющимся приложением. Микропрограммно реализуют примитивы типа «тестирование и блокировка», «выборка и инкремент», «выборка и добавление» и др. Они позволяют выполнять синхронизацию процессов методом активного ожидания (спин-блокировки) в пространстве пользователя. В некоторых языках программирования микропрограммно реализуют такие высокоуровневые механизмы синхронизации как, например, мониторы (в языке Jawa). Функции же управления задачами (планировщики, диспетчеры) чаще вызываются автоматически по таймеру и реализуются как процедура, выполняющаяся в пространстве ядра. Обычно такие функции требуют перенастройки для оптимизации вычислительного процесса под конкретный класс прикладных задач, что затрудняет использование микропрограммного подхода.

Схемная реализация заключается в перенесении компонент ядра операционной системы на уровень устройства управления или управляющего спецпроцессора. Схемная реализация позволяет полностью освободить ядро ОС от функций управления переключением задач и прерываниями, синхронизации процессов, обмена сообщениями и т.д. В результате API-функции, выполняемые в спецпроцессоре, значительно ускоряются, а тщательная отладка аппаратуры современными средствами проектирования ПЛИС гарантирует надежность и безопасность операционной системы. Такой подход обеспечивает повышение производительности многопроцессорной системы за счет:

1) быстрого выполнения функций ОС по управлению процессами и ресурсами в аппаратных средствах;

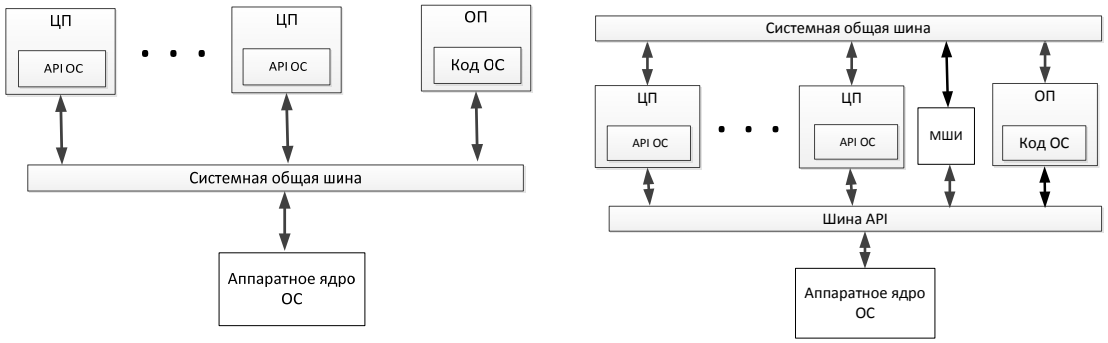
2) освобождения центральных процессоров от функций ОС по управлению процессами и ресурсами, что позволяет полностью занять их только выполнением приложений.

8.2. Архитектура аппаратного ядра многопроцессорной операционной системы

Возможны два способа включения аппаратного ядра ОС в многопроцессорную систему:

1) через системную общую шину (рис. 8.1,*а*);

2) через отдельную (специальную, периферийную) шину (рис. 8.1,*б*).



а) *б*)

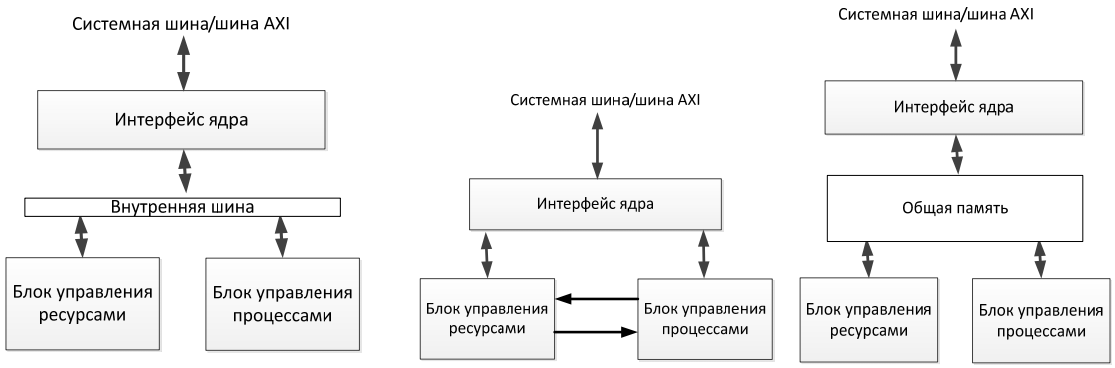
Рис. 8.1. Интерфейсы аппаратного ядра ОС:
а – с общей шиной; *б* – с отдельной шиной

Первый вариант структурной схемы использует единственную общую шину для взаимодействия между процессорами, памятью системы, а также между API (Application Programming Interface) операционной системы (API ОС) и аппаратным ядром ОС. API ОС является частью операционной системы, обеспечивающей программный интерфейс между выполняющимися приложениями через ядро ОС. API определяет функциональность, которую предоставляет программа операционной системы, при этом API позволяет абстрагироваться от того, как именно эта функциональность реализована. САМОЙ ВАЖНОЙ частью API операционных систем является множество системных вызовов. Положительной стороной такой архитектуры является то, что единый интерфейс дает возможность использования типовых средств (стандартного интерфейса) для включения аппаратного ядра в многопроцессорную систему. Однако она имеет недостаток, заключающийся в последовательном характере взаимодействия через общую шину, поскольку только два устройства могут связываться на любом промежутке времени. Это обстоятельство не обеспечивает параллелизм связей, что может создать высокий трафик, приводящий к перегрузке общей шины.

В архитектуре отдельной (специальной, периферийной) шины взаимодействие процессоров с оперативной памятью (ОП) по системной шине осуществляется параллельно с взаимодействием

API ОС с модулем аппаратного ядра – по специальной шине API. В такой архитектуре системная шина не нагружена дополнительным трафиком от операционной системы, что позволяет увеличить общую производительность многопроцессорной системы. Однако для обеспечения связи аппаратного ядра ОС с программным кодом ОС, хранящимся в оперативной памяти системы, требуется сопряжение шины API с оперативной памятью через некоторый межшинный интерфейс. При этом проблема облегчается тем обстоятельством, что межшинный интерфейс входит в состав аппаратуры некоторых процессоров. Так, например, в soft-процессор Microblaze [133] фирмы Xilinx встроена стандартная шина AXI4 для связи процессоров с общей памятью и потоковая шина AXI-Stream для связи периферии с той же памятью методом файлового обмена. Здесь потоковая шина работает в режиме межшинного интерфейса.

Устройство управления процессами и ресурсами (УУПР) составляет аппаратное ядро операционной системы и состоит из трех основных частей: интерфейса ядра, блока управления процессами (БУП) и блока управления ресурсами (БУР). На рис. 8.2 показаны возможные способы структурной реализации УУП. Интерфейс ядра состоит из логики, необходимой для декодирования инструкций API ОС и выполнения всех необходимых обменных взаимодействий.



а) *б*) *в*)

Рис. 8.2. Структура аппаратного ядра ОС

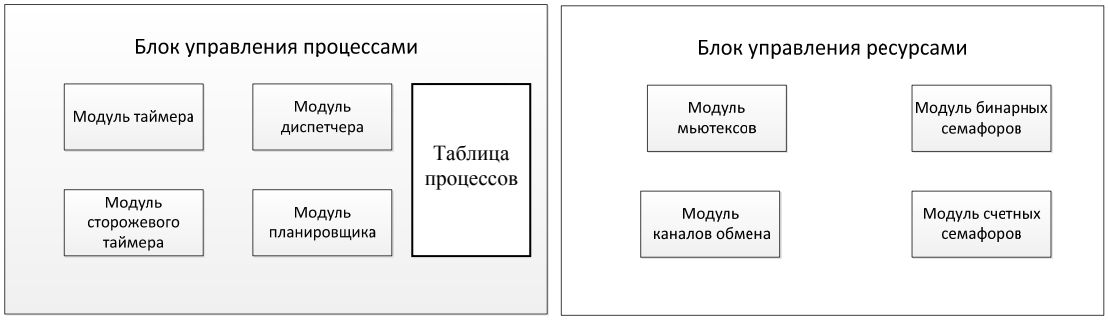
Первый вариант (см. рис. 8.2,*а*) использует единственную общую шину для взаимодействия между блоками УУПР, а также между API ОС и аппаратным ядром ОС. Положительной стороной является то, что единственный тип интерфейса позволяет использовать типовые средства для подключения аппаратного ядра в

многопроцессорную систему. Такая архитектура имеет недостаток, вызванный последовательным характером взаимодействия через общую шину, что не способствует высокой производительности аппаратного ядра. Второй вариант реализации заключается в том (см. рис. 8.2,*б*), что взаимодействие внутренних модулей выполняется путем создания непосредственных связей. Такая архитектура связей имеет значительно более высокую пропускную способность, однако сложность ее реализации велика.

Архитектура общей памяти показана на рис. 8.2,*в*. Она характеризуется совместным использованием модулей памяти и регистров, принадлежащих каждому блоку БУП и БУР. Идея состоит в том, что каждый блок владеет рядом регистров и, возможно, модулей памяти, в которые они пишут и читают данные, причем каждый блок также в состоянии считывать данные из регистров и памяти других блоков, поскольку они имеют единое адресное пространство. Таким образом, интерфейс каждого блока относительно простой, состоит только из регистров и логики доступа к памяти. Это обеспечивает единственную и довольно простую интерфейсную архитектуру, которую будет легче поддержать, чем несколько различных интерфейсов. API ОС может связаться с аппаратным ядром путем записи в соответствующие регистры. Все модули ядра могут связаться друг с другом, поскольку и регистры, и сегменты памяти поддерживают несколько читателей и писателей. Архитектура общей памяти обеспечивает параллелизм в связях как между блоками ядра, так и между аппаратурой ядра и API ОС. В то же время блоки ядра могут получать доступ только к частям регистров и памяти каждого из других модулей, а API-интерфейс в состоянии получать доступ ко всей карте распределения памяти.

Главные модули блока БУП (рис. 8.3,*а*): таймер, сторожевой таймер, диспетчер и планировщик задач. В его состав входит также таблица процессов, являющаяся RAM-памятью для хранения дескрипторов, готовых к выполнению процессов, которые содержат необходимые данные для управления сохранением и восстановлением контекстов.

БУР содержит в своем составе средства управления ресурсами (механизмы синхронизации процессов), которые обеспечивают взаимодействие параллельных процессов при доступе к общим (разделяемым) ресурсам, основанное на критических интервалах: мьютексы, бинарные и счетные семафоры, очереди сообщений. Этот блок может включать и другие механизмы синхронизации, например, флаги событий, почтовые ящики и т.д.



а) *б*)

Рис. 8.3. Состав блоков аппаратного ядра ОС:
а – управления процессами; *б* – управления ресурсами

В последующих разделах будут рассмотрены вопросы структурной реализации аппаратного ядра многопроцессорных операционных систем с использованием средств функционального моделирования языка. При проектировании устройств аппаратной поддержки важен выбор математического аппарата для формализации, обеспечивающего простоту описания, верификации и структурной реализации алгоритмов управления процессами и ресурсами. Наиболее подходящим аппаратом, на наш взгляд, является язык логики недетерминированных автоматов. Использование этого языка позволило разработать НДА-модели алгоритмов взаимодействия параллельных процессов, основанные на критических интервалах (модуль мьютексов), и модели очередей сообщений (модуль каналов). Из средств управления процессами разработаны модули диспетчера и частично планировщика задач, реализация которых показана в приведенных ниже разделах.

8.3. Структурная реализация алгоритмов управления ресурсами с использованием критических интервалов

Проблема аппаратной поддержки функций управления ресурсами возникла вместе с развитием многопроцессорных и распределенных вычислительных систем. Наиболее интересными решениями данной проблемы является аппаратная поддержка барьерной синхронизации, примененная в архитектуре компьютера Crey T3D/ T3E [112], и аппаратная синхронизации процессов, реализованная в многопроцессорной системе Sequent Balance

модели 21000. В последней реализации синхронизация производится программно, ускорение достигается применением специализированной быстродействующей памяти, которая названа атомарной памятью блокировки [113].

В вычислительных системах синхронизация взаимодействующих процессов заключается в том, что запрашивающий процесс анализирует глобальную блокирующую переменную (mutex, битовый семафор), закрепленную за общим ресурсом. Операция чтения блокирующей переменной, ее проверки и установки, а также записи в ячейку производится единой (неделимой) командой [72]. В процессорах время выполнения команды «читать – проверить – установить – записать» по сравнению со временем выполнения обычных команд увеличивается в два-три раза, что приводит к простоям конвейера команд и уменьшению реакции системы прерывания.

В многопроцессорной системе (МПС) блокирующая переменная хранится в ячейке общей оперативной памяти и доступна любым задачам (процессам, потокам), в том числе и тем, которые развиваются в разных процессорных узлах. Для доступа к ней (в случае кэш-промаха) процессор должен захватить межпроцессорную шину, выполнить операцию чтения ячейки памяти, в которой хранится блокирующая переменная, провести ее анализ, изменить состояние на противоположное (если общий ресурс не занят) и записать новое значение в ту же ячейку памяти. Применение неделимой операции «читать – проверить – установить – записать» в многопроцессорной системе вызывает также необходимость блокировки межпроцессорной шины, функционирующей в режиме расщепления транзакций чтения памяти. Этим исключается возможность одновременного доступа общего ресурса со стороны нескольких процессов, выполняющихся в разных процессорах, поскольку в промежутке недовыполненной транзакции шину может захватить другой процессор и обеспечить вход его процесса в свою критическую секцию. Для решения проблемы работы с блокирующими переменными в многопроцессорных конфигурациях большинству современных процессоров предоставляют аппаратные примитивы взаимоисключения: средства, позволяющие процессору монопольно захватить шину и выполнить несколько операций над памятью. Реализации этих примитивов различны у разных процессоров. Например, у x86 это специальный код опера-

ции, называемый префиксом захвата шины, который сам по себе не совершает никаких действий, но зато дает возможность исполнять следующую за ним операцию в монопольном режиме. Благодаря этому можно одновременно получить старое значение блокирующей переменной, установить и записать новое [114].

Кроме того, если процессоры содержат в своем составе кэш-память, то в случае участия в процедуре синхронизации нескольких процессорных узлов возможно необоснованное увеличение трафика на межпроцессорной шине из-за частого перемещения строки кэш, содержащей слово блокировки, которая необходима процессам, конкурирующим за доступ к общему ресурсу [72]. Последние два обстоятельства приводят к снижению реальной пропускной способности межпроцессорной шины и, в конечном счете, к потерям производительности всей вычислительной системы.

При формализации и описании основных событий на языке НДА при обращении n процессов к общему ресурсу использована методика, представленная в гл. 6, где для наглядности приведено как аналитическое, так и графическое представление реализуемых в алгоритме управления событий. Основой для формализации алгоритмов управления параллельными процессами при решении задачи «обращение к общему ресурсу» является взаимоисключение критических интервалов (секций), т.е. таких участков программы, которые обеспечивают доступ к разделяемым данным.

Результатом формализации является система канонических уравнений, которая в основном соответствует системе канонических уравнений, полученной в разделе 6.4. Обозначения событий изменены с целью единства представления их в модели на языке VHDL. Итоговая СКУ НДА имеет следующий вид (для i-го процесса):

$$S_k^i = S_{v\mathrm{p}}^i \,\&\, S_{vz}^i \,\&\, S_{pr}^i \vee S_k^i \overline{S}_{\mathrm{p}}^i,\ i = \overline{1,n};$$

$$S_{vz}^i = \bigwedge_{(\forall\alpha)[\alpha\neq i]} \overline{S}_k^{\alpha},\ i = \overline{1,n};$$

$$S_{\mathrm{p}r}^i(t+1) = S_{\mathrm{p}r}^i(0) \vee S_{vr}^i \Big(S_{\mathrm{p}k}^i \bigwedge_{(\forall\alpha)[\alpha\neq i]} \overline{S}_{v\mathrm{p}}^{\alpha} \vee$$

$$\vee S_{\mathrm{p}k}^{i\oplus 1} \bigwedge_{(\forall\alpha)[i\oplus 1<\alpha\le i-1]} \overline{S}_{v\mathrm{p}}^{\alpha} \vee \ldots \vee S_{\mathrm{p}k}^{i\oplus(n-1)} \bigwedge_{(\forall\alpha)[i\oplus(n-1)<\alpha\le i-1]} \overline{S}_{v\mathrm{p}}^{\alpha}\Big);$$

$$S^i_{\mathrm{p}r}(0) = S_0 x_n S^i_{v\mathrm{p}} \bigwedge_{(\forall\alpha)[\alpha<i]} \overline{S}^{\alpha}_{v\mathrm{p}};$$

$$S_0(t+1) = x_0 \vee S_0 \overline{x}_n \vee S_T;$$

$$S_T = \bigvee_{i=1}^{n} S^i_{\mathrm{p}k} \bigwedge_{(\forall\alpha)[\alpha<i]} \overline{S}^{\alpha}_{\mathrm{p}k};$$

$$S^i_{\mathrm{p}r}(\mathrm{t}+1) = \left(S^i_z \vee S^i_{v\mathrm{p}}\right)\overline{S}^i_k; \qquad (8.1)$$

$$S^i_z(t+1) = x^i_z \vee S^i_z \overline{S}^i_{v\mathrm{p}};$$

$$S_m(t+1) = S^1_k S^1_{v\mathrm{p}} \vee S^2_k S^2_{v\mathrm{p}} \vee \cdots \vee S^n_k S^n_{v\mathrm{p}};$$

$$S_{nk}(t+1) = S_m;$$

$$S^i_r(t+1) = \left(S^1_{v\mathrm{p}} \vee S^i_r\right)\overline{S}^i_{\mathrm{p}};$$

$$S^i_{\mathrm{p}}(t+1) = S_{nk} S^i_k;$$

$$S^i_{\mathrm{p}k}(t+1) = S^i_{\mathrm{p}} \vee S^i_{\mathrm{p}k} S^i_{\mathrm{p}r} S_T;$$

$$S^i_j(t+1) = S^i_r S^i_{\mathrm{p}},$$

где S^i_k – события, определяющие вход и нахождение *i*-го процесса в своем критическом интервале; $S^i_{v\mathrm{p}}$ – события, определяющие прием заявки *i*-го процесса на обслуживание для обращения к разделяемым данным; S^i_{p} – события, обеспечивающие выход *i*-го процесса из критического интервала после окончания процедуры обращения к разделяемым данным; S^i_{vz} – комбинационное событие, обеспечивающее взаимоисключение критических интервалов на основе несовместимости событий S^i_k с другими событиями из их общего числа, равного *n*; $S^i_{\mathrm{p}r}$ – события, обеспечивающие заданное приоритетное обслуживание *i*-го процесса; $S^i_{\mathrm{p}r}(0)$ – комбинационные события, определяющие начальный приоритет обслуживания *i*-го процесса; S_0 – начальное событие системы управления перед обращением к разделяемому ресурсу; x_n – сигнал инициализации системы управления; x_0 – сигнал приведения системы управления в начальное состояние; $\overline{S}^{\alpha}_{v\mathrm{p}}$ – события, определяющие восприятие заявки *i*-го процесса на обслуживание;

$S^i_{\text{p}k}$ – события, определяемые как событие S^i_{p}, задержанное на один такт; это событие используется, в принятом варианте, для организации циклической дисциплины обслуживания; S_T – комбинационные события, свидетельствующего об окончании цикла обслуживания процессов, от которых были восприняты заявки на обслуживание; S^i_z – сокращенное обозначение события, определяющего наличие заявки *i*-го процесса и их обслуживание; x^i_z – сигнал обращения к общему ресурсу *i*-го процесса; S_m – сокращенное обозначение события, обеспечивающего начало процедуры реализации обращения к разделяемым данным; S_{nk} – сокращенное обозначение события, являющегося заключительным в процедуре обращения к разделяемым данным; S^i_r – сокращенное обозначение события, символизирующего ожидание условий выхода *i*-го процесса из критического участка; S^i_j – сокращенное обозначение события, инициирующего продолжение работы *i*-го процесса после выхода из критического интервала.

Граф НДА, представляющий алгоритм взаимодействия при обращении к разделяемым данным, будет иметь вид (рис. 8.4).

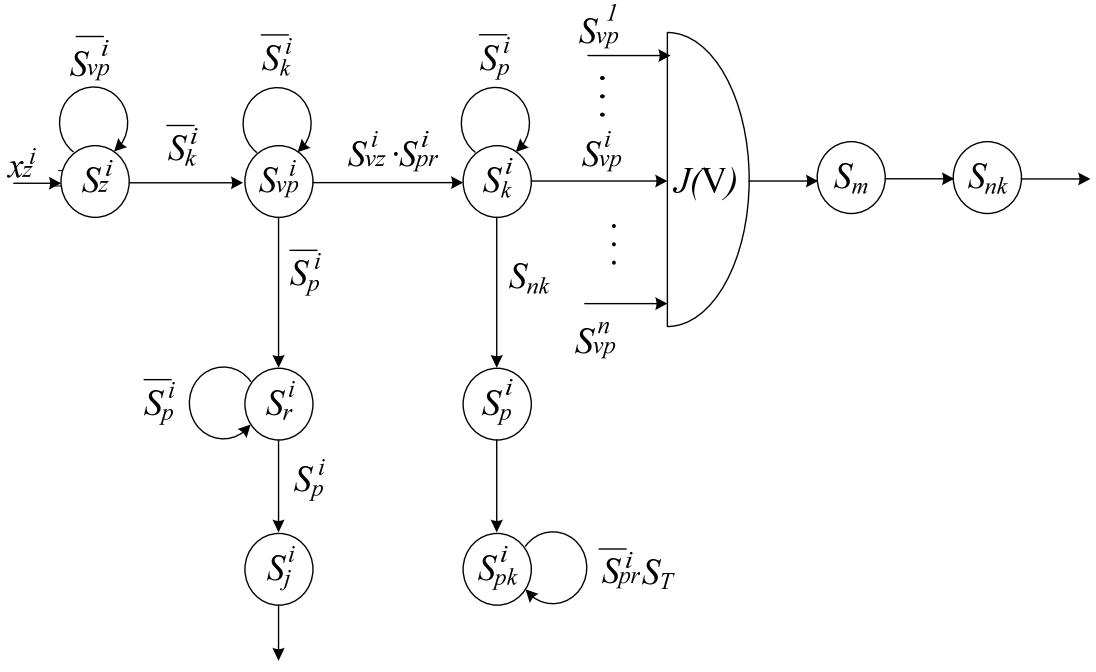


Рис. 8.4. Фрагмент графа НДА алгоритма управления взаимодействующими параллельными процессами при обращении к общему ресурсу для *n*-процессов (показана *i*-я ветвь)

Аппаратная реализация алгоритма управления межпроцессным взаимодействием выполнена на примере обращения к общему критическому ресурсу для 4 процессов. В связи с этим, на основании (8.1), системы уравнений, определяющие события S^i_{vz}, $S^i_{\mathrm{p}r}$ и S_T для 4 процессов, будут иметь следующий вид:

$$S^1_{vz} = \overline{S^2_l S^3_k S^4_k}, \qquad S^3_{vz} = \overline{S^1_k S^2_k S^4_k},$$

$$S^2_{vz} = \overline{S^1_k S^3_k S^4_k}, \qquad S^4_{vz} = \overline{S^1_k S^2_k S^3_k}.$$

$$S^1_{\mathrm{p}r}(t+1) = S^1_{\mathrm{p}r}(0) \vee S^1_{v\mathrm{p}}(S^4_{\mathrm{p}k} \vee S^3_{\mathrm{p}k}\overline{S^4_{v\mathrm{p}}} \vee S^2_{\mathrm{p}k}\overline{S^3_{v\mathrm{p}}S^4_{v\mathrm{p}}} \vee S^1_{\mathrm{p}k}\overline{S^2_{v\mathrm{p}}S^3_{v\mathrm{p}}S^4_{v\mathrm{p}}}),$$

$$S^2_{\mathrm{p}r}(t+1) = S^2_{\mathrm{p}r}(0) \vee S^2_{v\mathrm{p}}(S^1_{\mathrm{p}k} \vee S^2_{\mathrm{p}k}\overline{S^1_{v\mathrm{p}}S^3_{v\mathrm{p}}S^4_{v\mathrm{p}}} \vee S^3_{\mathrm{p}k}\overline{S^4_{v\mathrm{p}}S^1_{v\mathrm{p}}} \vee S^4_{\mathrm{p}k}\overline{S^1_{v\mathrm{p}}}),$$

$$S^3_{\mathrm{p}r}(t+1) = S^3_{\mathrm{p}r}(0) \vee S^3_{v\mathrm{p}}(S^2_{\mathrm{p}k} \vee S^3_{\mathrm{p}k}\overline{S^2_{v\mathrm{p}}S^4_{v\mathrm{p}}S^1_{v\mathrm{p}}} \vee S^4_{\mathrm{p}k}\overline{S^1_{v\mathrm{p}}S^2_{v\mathrm{p}}} \vee S^2_{\mathrm{p}k}),$$

$$S^4_{\mathrm{p}r}(t+1) = S^4_{\mathrm{p}r}(0) \vee S^4_{v\mathrm{p}}(S^3_{\mathrm{p}k} \vee S^4_{\mathrm{p}k}\overline{S^1_{v\mathrm{p}}S^2_{v\mathrm{p}}S^3_{v\mathrm{p}}} \vee S^1_{\mathrm{p}k}\overline{S^2_{v\mathrm{p}}S^3_{v\mathrm{p}}} \vee S^2_{\mathrm{p}k}\overline{S^3_{v\mathrm{p}}}),$$

$$S^1_{\mathrm{p}r}(0) = S_0 x_n S^1_{v\mathrm{p}}, S^3_{\mathrm{p}r}(0) = S_0 x_n S^3_{v\mathrm{p}}\overline{S^1_{v\mathrm{p}}S^2_{v\mathrm{p}}},$$

$$S^2_{\mathrm{p}r}(0) = S_0 x_n S^2_{v\mathrm{p}}\overline{S^1_{v\mathrm{p}}}, S^4_{\mathrm{p}r}(0) = S_0 x_n S^4_{v\mathrm{p}}\overline{S^1_{v\mathrm{p}}S^2_z S^3_{v\mathrm{p}}}.$$

$$S_T = S^1_{\mathrm{p}k}\overline{S^2_{v\mathrm{p}}S^3_{v\mathrm{p}}S^4_{v\mathrm{p}}} \vee S^2_{\mathrm{p}k}\overline{S^3_{v\mathrm{p}}S^4_{v\mathrm{p}}} \vee S^3_{\mathrm{p}k}\overline{S^4_{v\mathrm{p}}} \vee S^4_{\mathrm{p}k}. \qquad (8.2)$$

Аппаратно устройство синхронизации представлено в виде двух типов блоков (рис. 8.5). Блок 1 регистрирует запросы к общему ресурсу и фиксирует вход *i*-го процесса в критический интервал. Блок 2 реализует логические функции управления общим ресурсом (отмечает начало S_m и конец S_{nk} процедуры обращения к общему ресурсу), а также функцию взаимоисключения и выделения приоритетного запроса на 4 запроса, которые реализованы по логическим выражениям (8.2).

Модель устройства синхронизации процессов была выполнена на языке VHDL в виде двух типов программных модулей. Первый тип реализует логические функции 1-го блока, задаваемые выражениями (8.1), кроме формул для S_m и S_{nk}. Этот модуль используется в дальнейшем как компонент во втором программном модуле, представляющем собой модель четырехканального устройства синхронизации. Число программных модулей первого типа в устройстве синхронизации равно числу узлов регистрации запросов *n*.

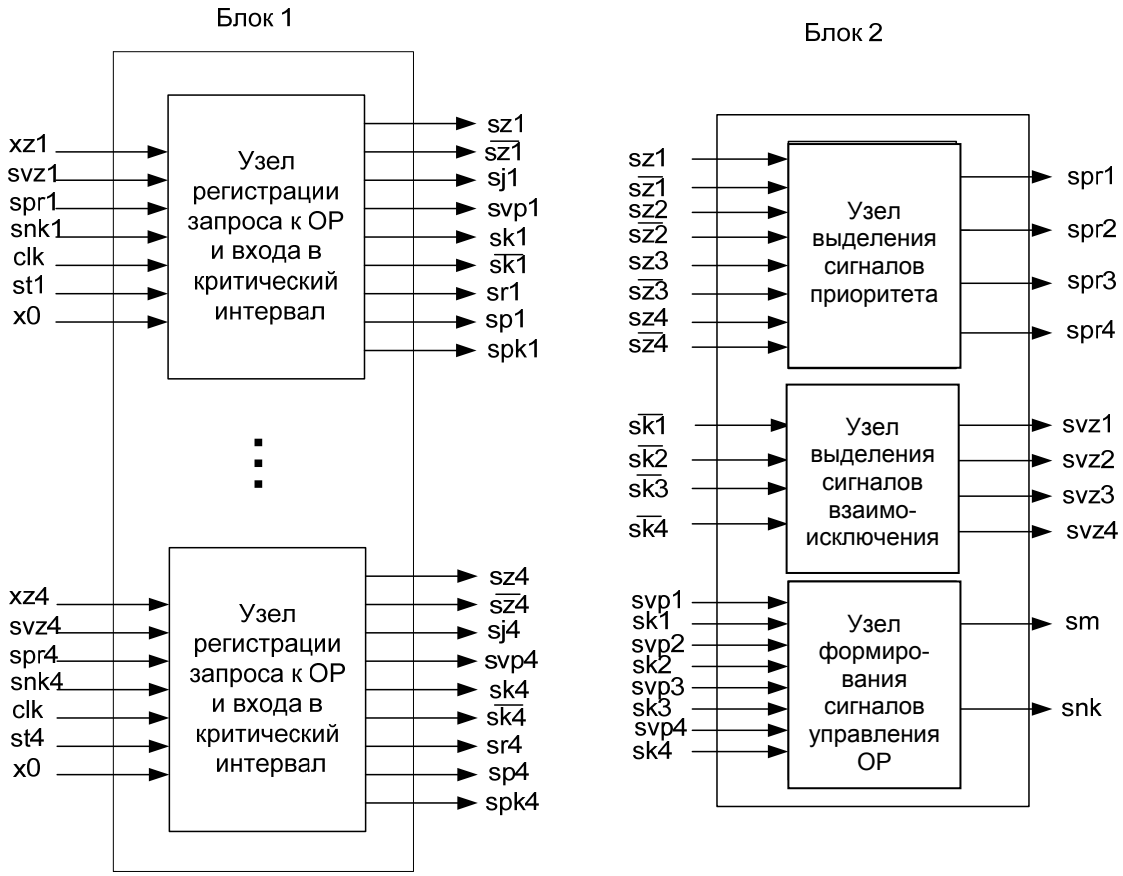


Рис. 8.5. Схема устройства синхронизации процессов на 4 входа

В качестве примера ниже приведен фрагмент программы для реализации узла регистрации запроса к общему ресурсу и входа в критический интервал, для чего составлено следующее описание:

1. *library IEEE;*
2. *use IEEE.STD_LOGIC_1164.ALL;*
3. *use IEEE.STD_LOGIC_ARITH.ALL;*
4. *use IEEE.STD_LOGIC_UNSIGNED.ALL;*

5. *entity np is*
6. *Port (x0, xz, spr, svz, st, snk, clk: in std_logic;*
7. *szo, sro, sjo, svpo, sksvpo, sko, spo, spko : out std_logic);*
8. *end np;*

9. *architecture Behavioral of np is*
10. *signal sz, sr, sj, svp, sksvp, sk, sp, spk: std_logic:='0';*
11. *begin*
12. *process (CLK)*
13. *begin*

```
14. if CLK='1' and CLK'event then
15. sz <= xz or (sz and (not svp));
16. sr <= (svp or sr) and (not sp);
17. svp <= (sz or svp) and (not sk);
18. sk <= (svp and svz and spr) or (sk and (not sp));
19. sp <= snk and sk;
20. spk <= sp or (spk and ((not spr) and st));
21. sj <= sr and sp;
22. end if;
23. end process;
24. szo<=sz; sro<=sr; sjo<=sj;
25. svpo<=svp;
26. sksvpo<=sksvp; sko<=sk; spo<=sp; spko<=spk;
27. end Behavioral.
```

В заголовке (строки 5–8) указываются входные и выходные переменные. Их имена составлены так, чтобы быть максимально похожими на наименования переменных в модели НДА. К именам выходных переменных на конце добавлена буква «о».

Непосредственная реализация формул выполнена в строках 15–21, которые входят в состав процесса (строки 12–23), задающего работу синхронных схем по переднему фронту синхросигнала *clk*. В строках 24–26 выполняется присваивание выходных сигналов, которые дублируют соответствующие внутренние сигналы. Этот модуль использовался как компонент в модели четырехканального устройства, результаты исследования которого приведены ниже.

Ввод схем и работа производилась в свободно распространяемой версии системы *WebPack ISE*, моделирование производилось в системе *ModelSim XE Stater* [117]. Результаты моделирования приведены на рис. 8.6.

Из временных диаграмм видно, латентность запроса (задержка запроса в условиях бесконфликтности) составляет два процессорных такта, что значительно меньше, чем при традиционной программной реализации рассмотренной *IPC*-функции в разделяемой памяти, которая по оценкам зарубежных исследователей составляет сотни тактов [101, 131]. При реализации этой же функции в пространстве ядра операционной системы, как показывают измерения, произведенные с помощью специализированной программы [126], основанной на программе *lmbench* [106], – десятки тысяч тактов.

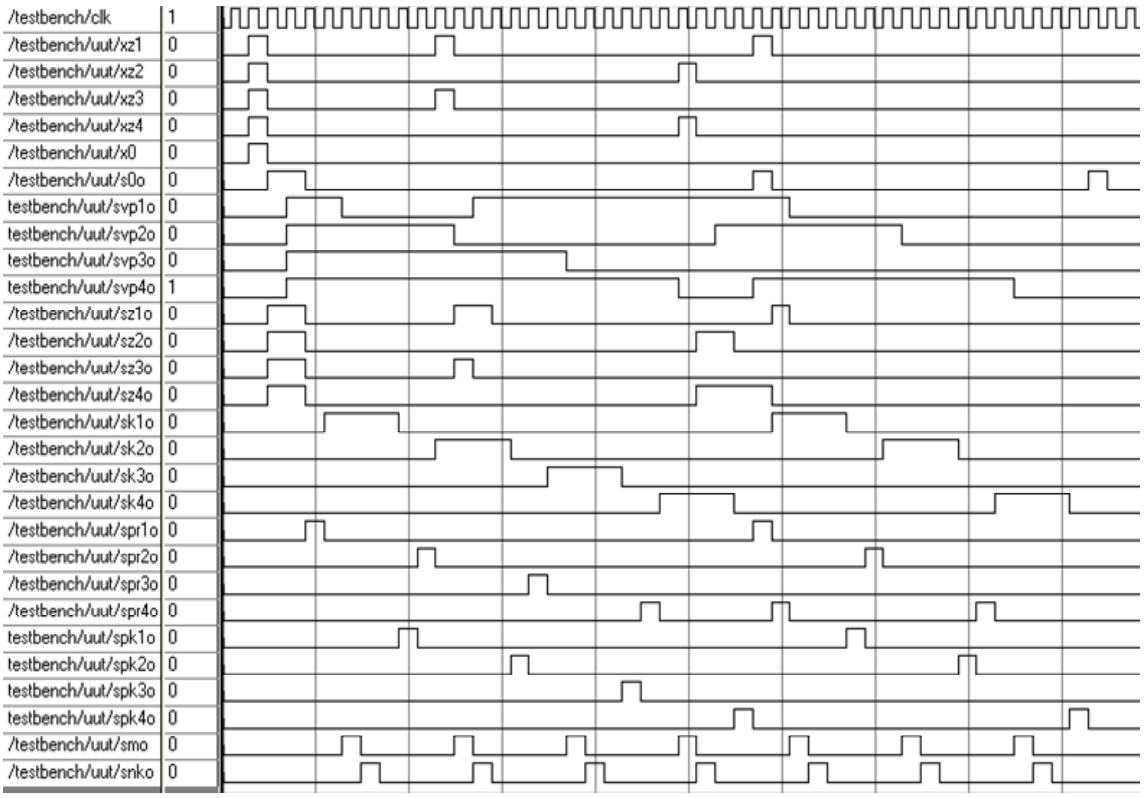


Рис. 8.6. Временная диаграмма работы устройства синхронизации

Полная версия программной модели устройства синхронизации, выполненная на языке *VHDL*, приведена в приложении к гл. 8.

8.4. Структурная реализация алгоритмов управления каналами обмена типа FIFO

При распараллеливании программ неизбежно возникают межпроцессные взаимодействия, которые проявляются в виде обмена между параллельными ветвями алгоритма (параллельными процессами, потоками). Для такого обмена существуют механизмы, реализуемые через именованные и неименованные каналы или разделяемую память [72]. Эти механизмы выполняются программно в составе операционной системы и входят, наряду с сигналами, мьютексами, семафорами, мониторами, в средства межпроцессных коммуникаций (*IPC*).

В случае неэффективной реализации механизмов обмена производительность системы может снизиться, а время отклика возрасти. Решением проблемы может служить аппаратная реализация средств межпроцессного обмена. Аппаратная реализация

различных алгоритмов часто встречается в вычислительной технике и позволяет сильно повысить производительность, а часто и снизить ее себестоимость [131].

Существуют различные исследования в данной области, например, в работе [123] возможность ускорения различных мультимедийных приложений, в частности декодирования *MJPEG* посредством использования аппаратных каналов *FIFO*. В другой работе [124] аппаратные каналы используются для ускорения адаптивных вычислений в системе с несколькими ядрами *PowerPC*. Это лишь пара примеров показывающих, что за рубежом уже давно идут разработки аппаратных ускорителей межпроцессорных коммуникаций.

В общем случае любой процесс обмена происходит с использованием буфера некоторого объема. Этим буфером может быть либо разделяемая область в оперативной памяти, либо объект операционной системы, или файл на жестком диске. В любом случае процесс-передатчик сначала записывает данные в буфер, а процесс-приемник читает их из него. Здесь возникает необходимость синхронизации этих процессов, которая обеспечивается механизмом, называемым каналом (*pipe*).

Для реализации обмена сообщениями использован алгоритм управления взаимодействующими асинхронными процессами в задаче «производители-потребители», подробно рассмотренный в гл. 6 и 7. В этом алгоритме синхронизация процессов решается за счет использования монитора, запрещающего одновременный доступ двух или более процессов к общему ресурсу, представленному в виде кольцевого буфера, чем обеспечивается бесконфликтный обмен данными между процессом-производителем и процессом-потребителем [33, 105] .

В связи с тем, что обмен может происходить между несколькими процессами одновременно, устройство аппаратного ускорителя должно быть многоканальным, т.е. в нем должно быть сразу несколько независимых буферов. Так как обычно используют односторонние (симплексные) каналы, то для обмена между двумя процессами в обе стороны необходимо использовать два канала, настроенные на передачу в разные стороны, причем каждый канал имеет свою очередь запросов, в которую помещаются номера запросивших обмен процессоров. Структурно устройство состоит из двух типов блоков: устройства управления и 16 блоков очереди

FIFO. Первый реализует алгоритм обмена, выстраивает очередь запросов и осуществляет коммутацию выбранного буфера сообщений с системной шиной многопроцессорной системы. Второй выполняет функцию кольцевого буфера сообщений и формирует сигналы переполнения буфера. Количество этих блоков равно количеству каналов в устройстве и может варьироваться. На рис. 8.7 приведена структурная схема устройства на 16 каналов, предназначенного для включения в восьмипроцессорную систему.

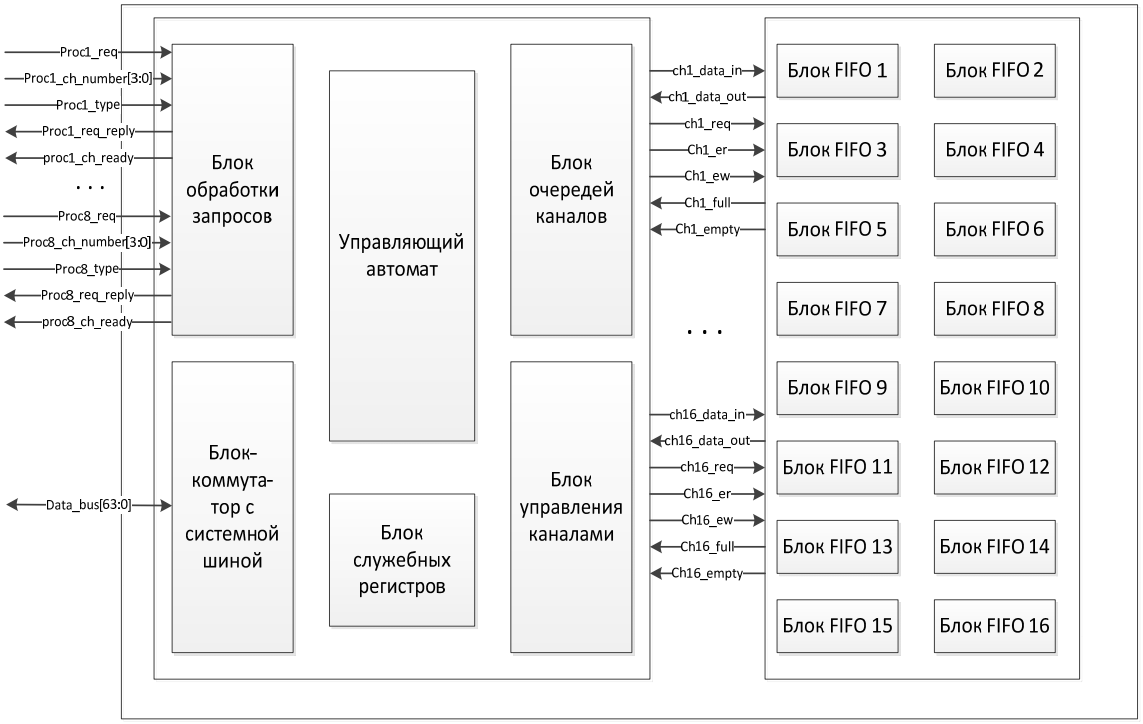


Рис. 8.7. Структурная схема аппаратного ускорителя, реализующего механизм очередей сообщений

В работе использован алгоритм управления взаимодействующими асинхронными процессами при обмене сообщениями в задаче «производители-потребители», представленный в виде недетерминированного автомата. В данном алгоритме процесс синхронизации процессов решается за счет использования монитора, запрещающего одновременный доступ двух или более процессов к общему ресурсу, представленному в виде кольцевого буфера, чем обеспечивается бесконфликтный обмен данными между процессом-производителем и процессом-потребителем.

Блоки FIFO.

Каждый из шестнадцати блоков *FIFO* представляет собой кольцевой буфер размером 1024 записи.

Запись в буфер производится по принципу «первый вошел, первый вышел». В каждом канале имеется 4 счетчика:

– счетчик, хранящий значение «головы» буфера. При записи его значение увеличивается на единицу;

– счетчик, хранящий значение «хвоста» буфера. При чтении его значение увеличивается на единицу;

– счетчик, подсчитывающий общее количество занятых ячеек в буфере. Его начальное значение равно нулю, при каждой записи оно увеличивается на единицу, а при каждом чтении из буфера уменьшается на единицу;

– счетчик, подсчитывающий общее количество пустых ячеек в буфере. Начальное значение счетчика равно 1024, при каждой записи оно уменьшается на единицу, а при каждом чтении увеличивается на единицу.

По значениям двух последних счетчиков определяется одно из трех состояний буфера (пуст, полон, не полон и не пуст). Состояние каждого буфера поступает на вход блока каналами.

Блоки *FIFO* соединены с блоком управления каналами внутренним интерфейсом, состоящим из следующих сигналов:

Ch_req – сигнал запроса;

Ch_er – сигнал разрешения чтения;

Ch_ew – сигнал разрешения записи;

Ch_empty – сигнал, оповещающий управляющее устройство о пустоте буфера канала;

Ch_full – сигнал, оповещающий управляющее устройство о заполненности буфера канала;

Ch_data_in – входная шина данных (запись данных в канал);

Ch_data_out – выходная шина данных (чтение из канала).

Блок управления каналами.

Данный блок хранит состояние каждого счетчика и на основе этих данных принимает решение, возможна ли запрошенная процессором операция или нет.

Блок очередей каналов.

В этом блоке содержатся буферы, в которых хранятся запросы для каждого канала устройства, а также номера процессоров, которые произвели запрос. Когда устройство управления проверяет очереди каналов, блок очередей каналов выдает не только сам запрос, но и номер процессора, который этот запрос подал.

Блок обработки запросов.

Блок соединяется с каждым процессором линией запроса; когда на нее поступает активный сигнал, происходит фиксация запроса и номера процессор, запросившего обмен. Эти данные передаются в блок очередей каналов, где хранятся до выборки устройством управления. В случае, когда процессор подал запрос, но при этом уже находится в одной из очередей каналов, запрос не обрабатывается, а передается напрямую в устройство управления для дальнейшей обработки.

Блок коммутации.

Данный блок реализует взаимодействие устройства с системной шиной и производит передачу всей необходимой информации в устройство.

Блок служебных регистров.

В блок входят различные буферные регистры и регистры, необходимые для работы устройства управления. Например, шестнадцать регистров *ch_cur_proc*, в которых хранятся номера процессоров, обрабатываемых каждым каналом устройства.

Устройство подключается к системной шине с помощью системного интерфейса и выступает в роли пассивного устройства. Инициатором процесса обмена сообщениями выступает один из процессоров системы. Процесс обмена между процессором и устройством состоит из следующих этапов:

1) процессор посылает запрос устройству. Он записывает *proc_req*, *proc_type* и *proc_ch_number* в сервисный регистр устройства;

2) если выбранный канал не занят, то устройство записывает в свой статусный регистр сигнал *proc_req_reply*, который считывает процессор;

3) устройство производит дополнительные подготовки и записывает в свой статусный регистр сигнал *ready*;

4) процессор производит захват системной шины, выставляет на нее данные (либо готовится к приему информации) и записывает *proc_req* в сервисный регистр устройства. Устройство считывает данные из системной шины и записывает их в выбранный канал.

Проектирование средств аппаратной поддержки является сложной технической задачей и требует разработки верифицированных алгоритмов управления передачей сообщений, поэтому при разработке блока использовался аппарат недетерминирован-

ных автоматов, обеспечивающий как формальное описание, так и структурную реализацию механизмов передачи сообщений. С помощью языка НДА описываются все реализуемые в алгоритме частные события, что позволяет представить алгоритм управления в простой и компактной форме, так как описание алгоритма выполняется не в терминах состояний детерминированных автоматов (ДА), а в терминах частных событий [104]. Такое представление алгоритма управления позволяет просто трансформировать его в языки моделирования аппаратуры, а также их верификации реализации на этих же моделях [105].

8.4.1. Формальное описание алгоритма на языке НДА

Как указывалось выше, модель основывается на алгоритме управления взаимодействующими параллельными процессами в задаче «производители-потребители» с использованием согласующего кольцевого буфера (см. гл.7, раздел 7.2). Ветви алгоритма «производителя» и «потребителя» были объединены, а также были оптимизированы некоторые части алгоритма для увеличения быстродействия. Система канонических уравнений НДА, описывающая модель, представлена ниже.

$$
\begin{aligned}
S_z(t+1) &= x_z \vee (S_{ready} S_{ret}) \vee (S_z S_{ret});\\
S_i(t+1) &= (S_z \overline{S_{ret}}) \vee (S_i \overline{S_m});\\
S_m(t+1) &= (S_i \overline{S_b} S_{ech}) \vee (S_m \overline{S_{ready}}\,\overline{S_{ech}});\\
S_b(t+1) &= (S_m S_{pt} S_{che}) \vee (S_m \overline{S_{pt}} S_{chf});\\
S_{wr}(t+1) &= (S_m \overline{S_{pt}}\ \overline{S_{chf}}) \vee (S_b \overline{S_{pt}}\ \overline{S_{chf}}) \vee (S_{wr} \overline{S_{prq}});\\
S_{rd}(t+1) &= (S_m S_{pt}\ \overline{S_{che}}) \vee (S_b S_{pt} \overline{S_{che}}) \vee (S_{rd} \overline{S_{prq}});\\
S_{rd2}(t+1) &= S_{rd} \vee (S_{rd2} \overline{S_{prq}});\\
S_{ready}(t+1) &= (S_{rd}\ S_{prq}) \vee (S_{wr} S_{prq}).
\end{aligned}
\qquad (8.3)
$$

Граф НДА, соответствующий модели, представлен на рис. 8.8. Из модели убрана часть алгоритма, отвечающая за доступ к шине запрашивающих процессоров, где заявки проходят процедуру приоритетного выбора и взаимного исключения.

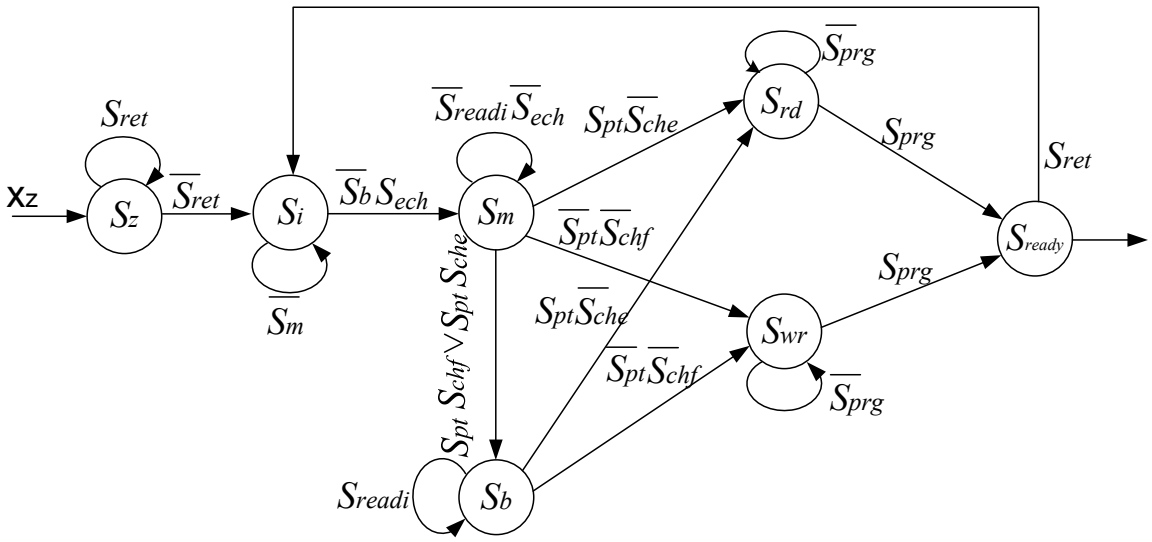


Рис. 8.8. Граф НДА модели канала обмена

В системе канонических уравнений (8.3) представлены следующие события:

x_z – входной сигнал запроса операции чтения/записи из/в буфера;

S_z – событие, свидетельствующее о поступлении заявки на обслуживание;

S_i – событие, свидетельствующее о принятии заявки на обслуживание;

S_m – событие, определяющее вход процесса производителя или потребителя в монитор;

S_b – событие, свидетельствующее о том, что процесс вышел из монитора необслуженным;

S_{wr} – событие, определяющее операцию записи;

S_{rd} – событие, определяющее операцию чтения;

S_{ready} – событие, определяющее выход из монитора после обслуживания;

S_{ech} – сигнал, свидетельствующий о пустоте очереди заявок;

S_{pt} – сигнал, свидетельствующий о том, что выбрана операция чтения;

S_{chf} – сигнал, свидетельствующий о том, что буфер заполнен;

S_{prq} – сигнал, определяющий наличие запроса от процессора;

S_{che} – сигнал, свидетельствующий о том, что буфер пуст;

S_{ret} – сигнал, свидетельствующий о необходимости повторного обслуживания текущей заявки.

8.4.2. Моделирование алгоритма на языке VHDL

Для моделирования устройства логика его работы была описана с помощью языка описания аппаратуры *VHDL*. Моделирование производилось в среде *ISim* [134], входящей в состав *Xilinx ISE* 14.4. Модель на языке *VHDL* состоит из нескольких модулей, описанных в отдельных файлах. Исходный код модели представлен в приложении к гл. 8. Ниже описаны основные функции этих модулей.

Модуль *control_unit.vhdl* реализует основную логику работы устройства, основанную на модели, описанной НДА. Также в этом модуле реализуются интерфейс взаимодействия с внешней шиной и обработка заявок от процессоров. Схема этого модуля представлена на рис. 8.9.

Рис. 8.9. Схема модуля *control_unit.vhdl*

Управляющий автомат, реализующий логику НДА, управляет обменом сразу по 4 независимым каналам с не более, чем двумя, процессорами. Каждый канал обладает своим независимым блоком памяти, описанным в модуле *channel1.vhdl*.

Модуль *channel1.vhdl* реализует блок памяти независимых каналов обмена. Схема этого модуля представлена на рис. 8.10.

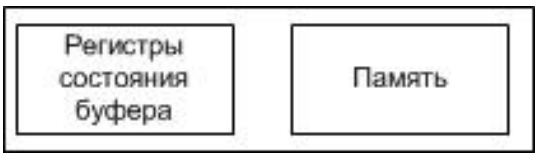


Рис. 8.10. Схема модуля *channel*1.*vhdl*

Память представляет собой кольцевой буфер, работающий по принципу FIFO. Состояние этого буфера хранится в регистрах состояния, в частности там содержится информация о смещении головы и хвоста буфера, общее количество занятых и пустых ячеек, а также биты пустоты и переполнения буфера, которые используются в управляющем автомате.

Модуль *emul_test.vhdl* генерирует синхросигнал *clk* с периодом 10 нс. Общая схема моделируемой системы представлена на рис. 8.11.

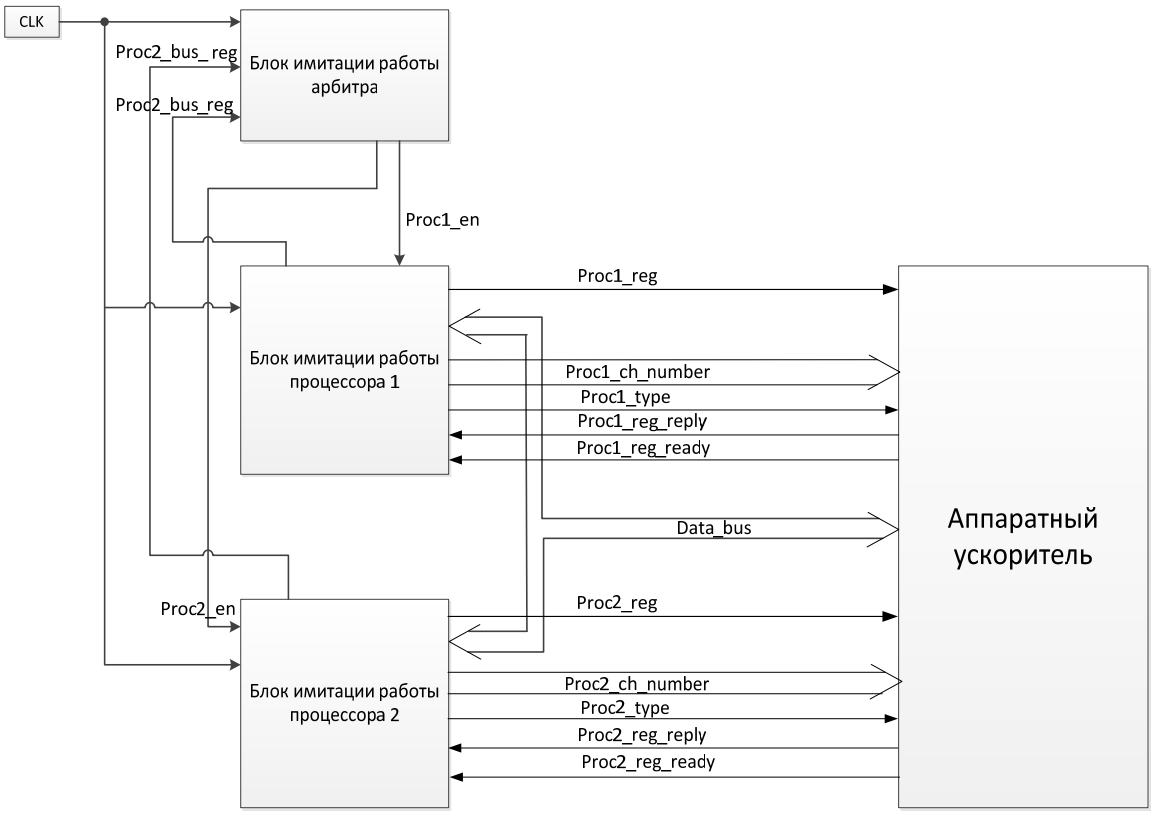


Рис. 8.11. Схема моделируемой системы
для экспериментальных исследований

Так как моделируемое устройство не входит в состав какой-либо многопроцессорной системы, то для проверки его работы был написан еще один модуль, имитирующий работу двухпроцессорной системы. Модуль *processor_emulation.vhdl* реализует ими-

тацию работы 2 процессоров в части обмена данными через моделируемое устройство. Также имитируется работа арбитра системной шины, распределяющего доступ к шине.

Хотя этот блок и является единым программным модулем, но он четко разделен на две части: часть, производящая имитацию работы процессоров, и часть, выполняющая функцию имитации работы системного арбитра. Рассмотрим работу каждой из частей подробнее.

Часть, имитирующая работу процессоров, содержит два процесса, каждый из которых реализуется в отдельном процессоре. Алгоритм работы каждого процесса моделируется простым автоматом с семью состояниями:

INIT – состояние инициализации, процессор не занимает шину, сигнал запроса системной шины *proc_bus_req* в пассивном состоянии, сигнал запроса на обмен с аппаратным ускорителем *proc_req* в активном состоянии, на шину *proc_ch_number* процессор выставляет номер канала, требуемого для обмена с устройством, а на линию *proc_type* – тип операции (0 – запись, 1 – чтение). Происходит переход в состояние *REQ*;

REQ – состояние ожидания ответа на запрос от аппаратного ускорителя. Если *proc_req_reply* находится в активном состоянии, то процессор переводит свой сигнал запроса *proc_req* в пассивное состояние и переходит в состояние *ENTER_MON*;

ENTER_MON – состояние ожидания готовности аппаратного ускорителя к обмену. В этом состоянии процессор дожидается, пока аппаратный ускоритель произведет подготовительные операции перед обменом или пока не наступят необходимые для обмена условия (например, не освободится место в канале, в который процессор запросил запись). Если сигнал *proc_ch_ready* находится в активном состоянии, то процессор производит запрос на захват системной шины, выставив на линию *proc_bus_req* единицу, и переходит либо в состояние *WR*, либо в состояние *RD* в зависимости от запрошенной операции;

WR – в этом состоянии процессор ожидает разрешения *proc_en*, от системного арбитра, на захват системной шины. В случае получения разрешения процессор снимает сигнал запроса шины *proc_bus_req*, выставляет на шину данных информацию, которая будет записана в запрошенный ранее канал, и выставляет сигнал *proc_req* в единицу, что сигнализирует аппаратному ускорителю, что данные готовы к передаче. Затем происходит переход в состояние *READY*;

RD – процесс чтения из аппаратного ускорителя разбит на два этапа, на первом этапе процессор ожидает разрешения на захват системной шины от арбитра. Дождавшись разрешения, процессор посылает сигнал *proc_req*, по которому аппаратный ускоритель начинает выборку информации из запрошенного канала. Дальше происходит переход на второй этап чтения; когда аппаратный ускоритель подготовил данные для передачи, он сигнализирует об этом пассивным сигналом *proc_ch_ready*. Процессор, восприняв этот сигнал, считывает данные с шины данных и переходит в состояние *READY*;

READY – в этом состоянии процессор обнуляет сигнал *proc_req* и посылает системному арбитру сигнал, сообщающий об освобождении шины *proc_bus* = 1. Затем происходит переход в состояние *INIT*, и процесс повторяется.

По данному алгоритму работает каждый из 2 процессоров, причем один из них производит операцию записи, а другой – операцию чтения.

Алгоритм работы системного арбитра реализован в виде недетерминированного автомата. Он, в свою очередь, так же, как и весь модуль, состоит из 2 частей: схемы приема запросов на захват шины от процессоров и схемы выдачи разрешений на захват шины. Первая схема на каждом такте принимает запросы от процессоров и помещает их в очередь запросов, которая представляет собой кольцевой буфер. Вторая схема извлекает запрос из очереди и передает запрашивающему процессору сигнал, разрешающий захват шины, но только в том случае, если другой процессор освободил к этому моменту шину. Таким образом, на данном этапе выделяется единственный запрос на доступ к буферу канала (на чтение или на запись).

Была промоделирована следующая ситуация: первый процессор записывает данные (произвольное восьмибитное число); второй процессор считывает это число и сохраняет у себя в регистре. Обмен производится через аппаратный ускоритель с использованием нулевого канала. Временные диаграммы работы системы представлены на рис. 8.12.

По ним видно, что время обмена (время от запроса первым процессором операции записи до записи данных в регистр второго процессора) составляет 16 тактов синхросигнала, что значительно меньше, чем при традиционной программной реализации очереди сообщений в разделяемой памяти, которая по оценкам зарубеж-

ных исследователей составляет сотни тактов [131]. При реализации очереди сообщений в пространстве ядра операционной системы и как показывают измерения, произведенные с помощью специализированной программы [126], составляет десятки тысяч тактов.

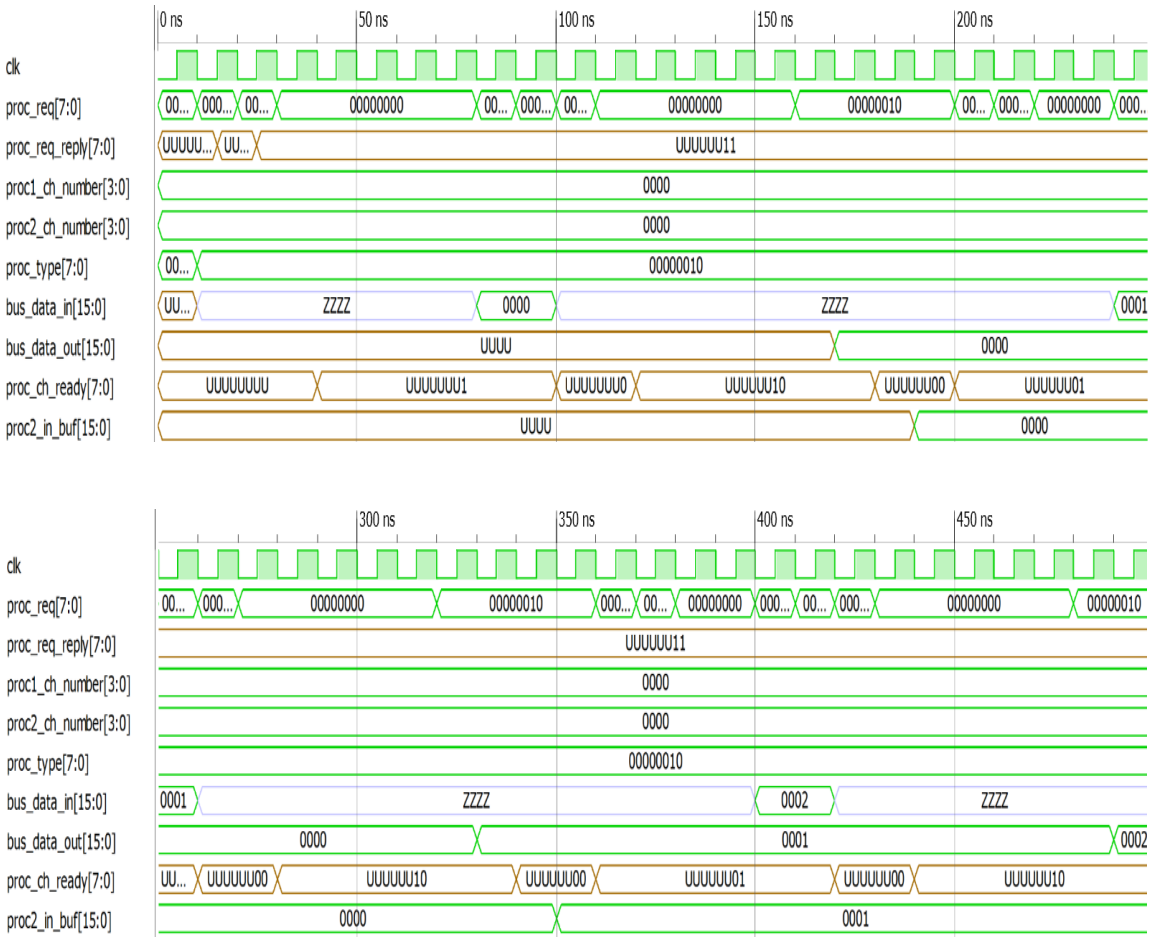


Рис. 8.12. Временные диаграммы моделирования системы

Измерения, произведенные программой *lmbench* [106], показывают, что пропускная способность механизма обмена сообщениями с использованием аппаратного ускорителя по сравнению со скоростью подобных IPC-функций в *ОС UNIX* увеличивается примерно на 25 %, а латентность уменьшается более чем в 500 раз. По предварительным оценкам наибольший выигрыш разрабатываемое устройство дает при использовании в многопроцессорных системах на кристалле (*SoC*), где прибавка быстродействия может быть еще более существенна за счет того, что оно будет интегрировано в кристалл и подключено к внутренней высокоскоростной системной шине, объединяющей все устройства *SoC*-системы.

Полная версия программной модели устройства канала обмена сообщениями, выполненная на языке *VHDL*, приведена в приложении к гл. 8.

8.5. Структурная реализация устройства управления процессами в многопроцессорных системах

При проектировании многопроцессорных и распределенных операционных систем реального времени, реализованных в системах на кристалле (СнК), особенно остро стоит проблема уменьшения накладных расходов, возникающих в системах управления процессами, в частности, при планировании и диспетчеризации задач (процессов, потоков) [118, 129]. Традиционно планирование и диспетчеризация процессов (задач) осуществляется программным путем в пространстве ядра операционной системы и на производительность вычислительной системы влияет слабо. В многопроцессорных системах относительные временные затраты на планирование/диспетчеризацию увеличиваются. С достаточно высокой точностью можно допустить, что для одной и той же пользовательской программы, выполняющейся в однопроцессорном и многопроцессорном режимах, временные затраты на планирование/диспетчеризацию задач одинаковы. Относительные же временные затраты резко отличаются вследствие уменьшения времени выполнения параллельных потоков. Это объясняется тем, что та часть программы, которая связана с синхронизацией процессов, является последовательной и, в соответствии с законом Амдала [112], является фактором, снижающим производительность многопроцессорной системы.

Пусть t – время выполнения пользовательской программы в однопроцессорном режиме, а время выполнения этой же программы на N-процессорной системе составит

$$t^* = t / N.$$

Пусть τ – время, затрачиваемое на планирование / диспетчеризацию процессов. Тогда время выполнения программы с учетом планирования / диспетчеризации процессов для однопроцессорного варианта составит $(t + \tau)$, а для многопроцессорного, соответственно, $(t^* + \tau)$.

Коэффициент использования производительности η, показывающий какова доля мощности одного процессора задействована при реализации одного из параллельных процессов, составит:

– для однопроцессорного варианта

$$\eta = t / (t + \tau); \tag{8.4}$$

– для N-процессорного варианта

$$\eta^* = t^* / (t^* + \tau) = t / (t + N\tau), \tag{8.5}$$

где τ и $N\tau$ – непроизводительное время, затрачиваемое на диспетчеризацию процессов в однопроцессорном и многопроцессорном режимах работы. Как следует из (8.4) и (8.5), временные затраты на диспетчеризацию в многопроцессорном режиме увеличиваются относительно однопроцессорного в N раз.

8.5.1. Структура многопроцессорной системы с аппаратным планировщиком/диспетчером задач на основе глобальной очереди

Рассмотрим многопроцессорную систему, в которой используется планировщик с разделением времени [72], который использует глобальную очередь задач типа FIFO, разделяемую всеми процессорами (рис. 8.13). Вновь поступившая задача (процесс) помещается планировщиком в конец очереди, а задачи, находящиеся в начале очереди, являются кандидатами на выполнение. Когда один из процессоров освобождается от текущей работы, он обращается к диспетчеру, который выбирает из «головы» очереди готовую к выполнению задачу и работает с ней до ее завершения или до момента блокирования, например, вследствие необходимости выполнения операции ввода-вывода.

Алгоритм планирования с разделением времени и глобальной очередью задач представляет наиболее простой и одновременно эффективный способ планирования, поскольку обладает рядом достоинств [62]:

– загрузка распределяется равномерно между процессорами, обеспечивая отсутствие простоев процессоров при наличии готовых к выполнению задач;

– простота функционирования, заключающаяся в том, что, когда процессор освобождается, он вызывает функцию назначения задач (диспетчер) из операционной системы.

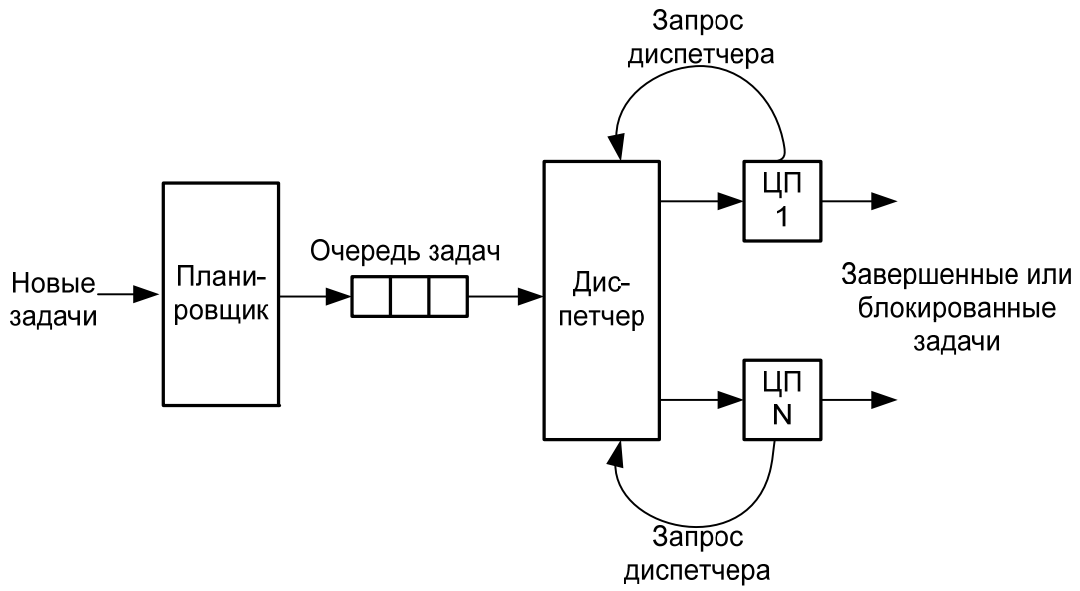


Рис. 8.13. Модель многопроцессорной системы, реализующей планировщика и диспетчера с глобальной очередью задач

Эти достоинства обеспечивают преимущественное применение такого способа планирования в многопроцессорных системах. Традиционно планировщик и диспетчер задач выполняются программно и реализуются в SMP-системах методом вызова этих функций из общей памяти, в которой резидентно хранится программа операционной системы. Практическая программная реализация предполагает два варианта:

1) в пространстве пользователя;

2) в пространстве ядра.

Программная реализация в пространстве пользователя хотя и является быстрой, однако вызывает затруднения при программировании, поскольку для выполнения сопряженной с диспетчеризацией процедуры синхронизации требуется 3 семафора: один – счетчик – для подсчета числа мест, занятых готовыми для выполнения процессами; другой – счетчик – для подсчета числа процессоров, занятых обслуживанием; и третий – мьютекс – для реализации взаимного исключения, предотвращающего одновременный доступ нескольких свободных процессоров к единственной очереди, которая в данном случае играет роль общего ресурса. Эти обстоятельства могут привести к трудно выявляемым ошибкам непредсказуемым последствиям при работе операционной системы, особенно реального времени.

Очевидным выходом из этого положения является реализация процедуры планирования/диспетчеризации в пространстве

ядра [72, 97], однако возрастающие при этом временные затраты на выполнение системных вызовов резко уменьшают производительность многопроцессорной системы.

Одним из путей повышения производительности многопроцессорной операционной системы является аппаратная реализация функций планирования/диспетчеризации [103, 118, 120], которая не только снимает проблему временных потерь, но и позволяет увеличить надежность операционной системы, что особенно важно для систем реального времени.

Предлагаемый подход заключается в том, что планировщик и диспетчер задач выполняются в виде независимого аппаратного устройства (специализированного процессора) в составе многопроцессорной системы (рис. 8.14).

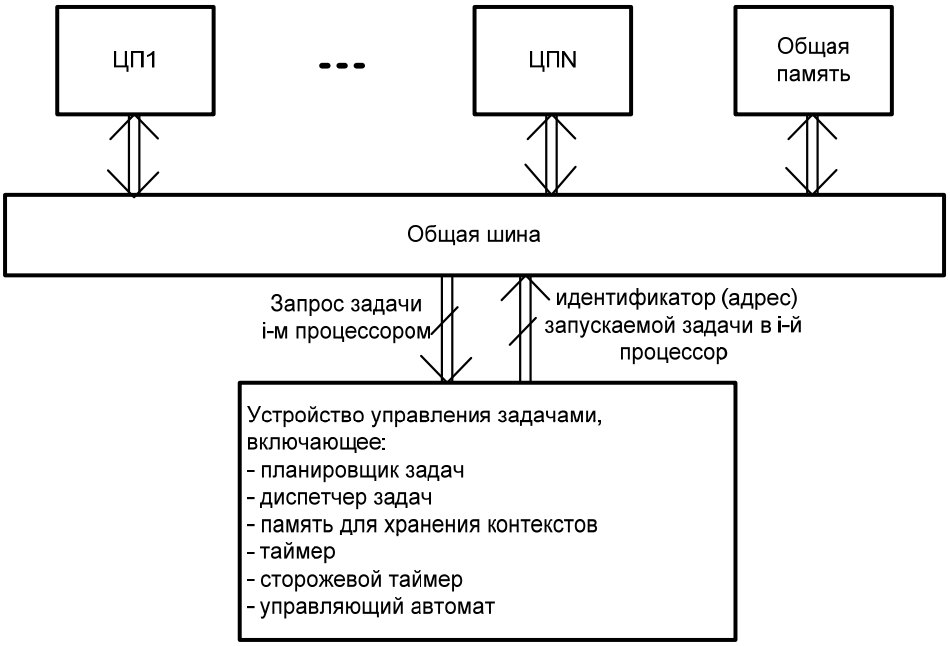


Рис. 8.14. Схема включения устройства управления задачами в многопроцессорную систему

Планировщик создает очередь, состоящую из идентификаторов задач, в соответствии с заданным правилом (FIFO, LIFO, приоритетная очередь) и осуществляет выборку задачи из очереди. Идентификатор задачи однозначно указывает на адрес задачи в памяти вычислительной системы. Традиционно очередь процессов хранится в виде блоков управления процессами (БУП) или *Process Control Block* (*PCB*) в общей памяти системы по виртуальному адресу либо в специализированной памяти устройства управления процессами (задачами).

Диспетчер осуществляет переключение задач (процессов) при прерываниях программ, например, от таймера. Результатом обработки прерывания является сохранение контекста текущей задачи и восстановление контекста задачи, выбираемой из очереди в соответствии с заданным ее типом (FIFO, LIFO, приоритетная). Традиционно контекст хранится в оперативной памяти в виде сегмента состояния процесса (ССП или *TSS*) и включается в состав БУП.

Таймер предназначен для формирования отметок времени, т.е. квантов процессорного времени, выделяемых каждой задаче из очереди готовых процессов, по которым производится переключение процессов.

Сторожевой таймер отслеживает задачи, которые занимают процессоры на неопределенное время, например, вследствие зацикливания (взаимной блокировки процессов или клинча).

В разработанном аппаратном устройстве управления процессами очередь, создаваемая планировщиком, представляет собой набор из 256 регистров, каждый из которых хранит идентификатор процесса (потока), приоритет процесса, счетчик времени и другую информацию (аналог БУП), необходимую для планирования и диспетчеризации. Каждая процесс имеет собственный идентификатор, который однозначно указывает на адрес задачи в памяти вычислительной системы. Такой подход позволяет значительно ускорить переключение процессов по окончании выделенного кванта времени.

8.5.2. Формализация алгоритма планирования/ диспетчеризации задач с использованием логики НДА

В самом общем виде алгоритм планирования/диспетчеризации задач выглядит следующим образом. В начальном состоянии каждый включающийся в работу процессор выполняет операцию «готов» и переходит в спящий режим, в котором находится до тех пор, пока на обслуживание не поступит задача из очереди. Если все процессоры заняты обслуживанием, то вновь поступившая задача помещается в конец очереди, число мест в которой ограничено. Если вновь поступившая задача обнаружит, что очередь заполнена, она на обслуживание не принимается и покидает

систему. Принятая на обслуживание задача находится в очереди до тех пор, пока не поступит на выполнение в процессор, при этом в очереди освобождается одно место. После выполнения очередной задачи планировщик/диспетчер просматривает очередь, и если в ней имеются ожидающие, то он назначает на выполнение в свободный процессор ту, которая стоит в «голове» списка (по принципу FIFO). Если очередь пуста, то свободные процессоры переходят в режим ожидания. Если в процессорном пуле имеется несколько свободных процессоров, то производится приоритетный выбор одного из них для обслуживания очередной задачи.

В общем случае алгоритм планирования/диспетчеризации в явном виде связан с взаимодействием процессов. С одной стороны задачи (процессы, потоки), требующие своего выполнения, с другой стороны – обслуживающие их процессорные узлы. Эти действия необходимо синхронизировать таким образом, чтобы обеспечить так называемое «рандеву» [97, 121], когда *j*-й процессорный узел должен дождаться поступления задачи, а *i*-й процесс – освобождения одного из процессоров, после чего она будет обслуживаться в течение некоторого времени.

Традиционный подход, связанный с таким обслуживанием, аналогичен задаче «спящего парикмахера» [72, 119], который широко применялся для вычислительных систем с одним процессором. Для многопроцессорных систем такой подход должен быть расширен до задачи, которую можно назвать «работа парикмахерской». Эта процесс иллюстрирует отношения «клиент–сервер», которые имеют место между процессами в многопроцессорных вычислительных системах, причем аналогом процессоров является коллектив парикмахеров, а аналогом задач выступают клиенты парикмахерской [120, 121].

Формализация алгоритмов управления параллельными процессами выполнена на основе использования механизма «рандеву», подробно описанного в разделе 7.3. Весь алгоритм работы диспетчера содержит три части: клиентскую (постановка задачи в очередь), серверную (обслуживание процессорами) и «рандеву» (наличие задачи и готовность одного из процессоров к обслуживанию этой задачи) [121]. Для описания алгоритма введены основные частные события, представленные в табл. 8.1. В последней колонке представлено их соответствие сигналам на схеме устройства диспетчеризации задач.

Таблица 8.1

Обозначение события	Описание частного события	Сигналы на схемах
S_I^t	Ожидание занесения задачи в очередь	*S_task_arrived*
S_{FQ}^t	В очереди имеются свободные места	$\overline{\text{S\_fifo\_full}}$
S_Q^t	Задача в очереди	*S_task_in_queue*
S_{ZPj}^t	Запрос *j*-го процессора диспетчером	*S_proc_zapr_j*
S_{GPj}^t	Задача готова к обслуживанию в *j*-м процессоре, *j*-й процессор помещается в пул занятых	*S_task_ready_j*
S_{PZ}^{pj}	*j*-й процессор выдал сигнал – подтверждение запроса	*S_proc_pzapr_j*
S_S^{pj}	*j*-й процессор помещается в пул свободных	*S_proc_free_j*
S_S^p	В процессорном пуле имеются свободные процессоры	*S_proc_free*
S_{PR}^{pj}	Приоритет *j*-го процессора	Внутренний сигнал, на схеме не обозначен
S_{PT}^{pj}	*j*-й процессор задачу принял	*S_proc_prinyal_j*
S_{SL}^{pj}	*j*-й процессор выбран для обслуживания задачи	*S_proc_sel_j*
S_A^{pj}	*j*-й процессор выполняет обслуживание задачи	Аппаратно не реализовано
S_{OF}^t	Удалить задачу из очереди	Внутренний сигнал, на схеме не обозначен
S_{TO}^{pj}	Снять задачу с исполнения	Внутренний сигнал, на схеме не обозначен
S_E^{pj}	Выполнение задачи закончено	Внутренний сигнал, на схеме не обозначен
S_{RT}^t	Выдача результата выполнения текущей задачи	Внутренний сигнал, на схеме не обозначен
S_O^{pj}	*j*-й процессор находится в режиме ожидания	Внутренний сигнал, на схеме не обозначен

На основании словесно представленного алгоритма управления процессами и введенных событий, реализуемых в этом алгоритме, система канонических уравнений, описывающих эти события, будет иметь следующий вид:

$$
\begin{gathered}
S_I^t(t+1) = x_z \vee S_I^t \overline{S_{FQ}^t}; \\
S_Q^t(t+1) = S_I^t S_{FQ}^t \vee S_Q^t \overline{S_{SL}^{pj}}; \\
S_{ZPj}^t(t+1) = S_Q^t S_{SL}^{pj} \vee S_{ZPj}^t \overline{S_{PZ}^{pj}}; \\
S_{GPj}^t(t+1) = S_{ZPj}^t S_{PZ}^{pj} \vee S_{GPj}^t \overline{S_{PT}^{pj}},
\end{gathered}
\tag{8.6}
$$

где x_z – сигнал запроса задачи на вхождение в очередь;

для процесса «сервер», реализуемого процессором до момента рандеву:

$$
\begin{gathered}
S_S^P(t+1) = (S_S^{p1} \vee S_S^{p2} \vee \ldots \vee S_S^{pN}) \vee S_S^P \overline{S_{ZP}^t}; \\
S_O^{pj}(t+1) = S_{ZP}^t \vee S_O^{pj} \overline{S_{SL}^{pj}}; \\
S_{SL}^{pj}(t+1) = \bigvee_{\forall j \in N} S_O^{pj} S_{PR}^{pj}; \\
S_{PZ}^{pj}(t+1) = S_{SL}^{pj} S_{ZPj}^t \vee S_{PZ}^{pj} \overline{S_{GPj}^t}; \\
S_{PT}^{pj}(t+1) = S_{PZ}^{pj} S_{GPj}^t.
\end{gathered}
\tag{8.7}
$$

Система канонических уравнений, описывающая события после рандеву:

$$
\begin{gathered}
S_A^{pj}(t+1) = S_{PT}^{pj} S_{GPj}^t \vee S_A^{pj} \overline{S_E^{pj}}; \\
S_{TO}^{pj}(t+1) = S_A^{pj} S_E^{pj} \vee S_{TO}^{pj} \overline{S_{RT}^t}; \\
S_{RT}^t(t+1) = S_A^{pj} S_E^{pj} \vee S_{RT}^t \overline{S_{TO}^{pj}}; \\
S_{OF}^t(t+1) = S_{TO}^{pj} S_{RT}^t; \\
S_S^{pj}(t+1) = S_{TO}^{pj} S_{RT}^t \vee S_S^{pj} \overline{S_{GPj}^t}.
\end{gathered}
\tag{8.8}
$$

Уравнениям соответствует граф недетерминированного автомата (НДА). На рис. 8.15 представлены графы выбора *j*-го процессора для обслуживания очередной задачи, клиентской и серверной частей алгоритма управления взаимодействующими

процессами до момента «рандеву» (до оператора объединения $J(\&)$) и часть графа после «рандеву».

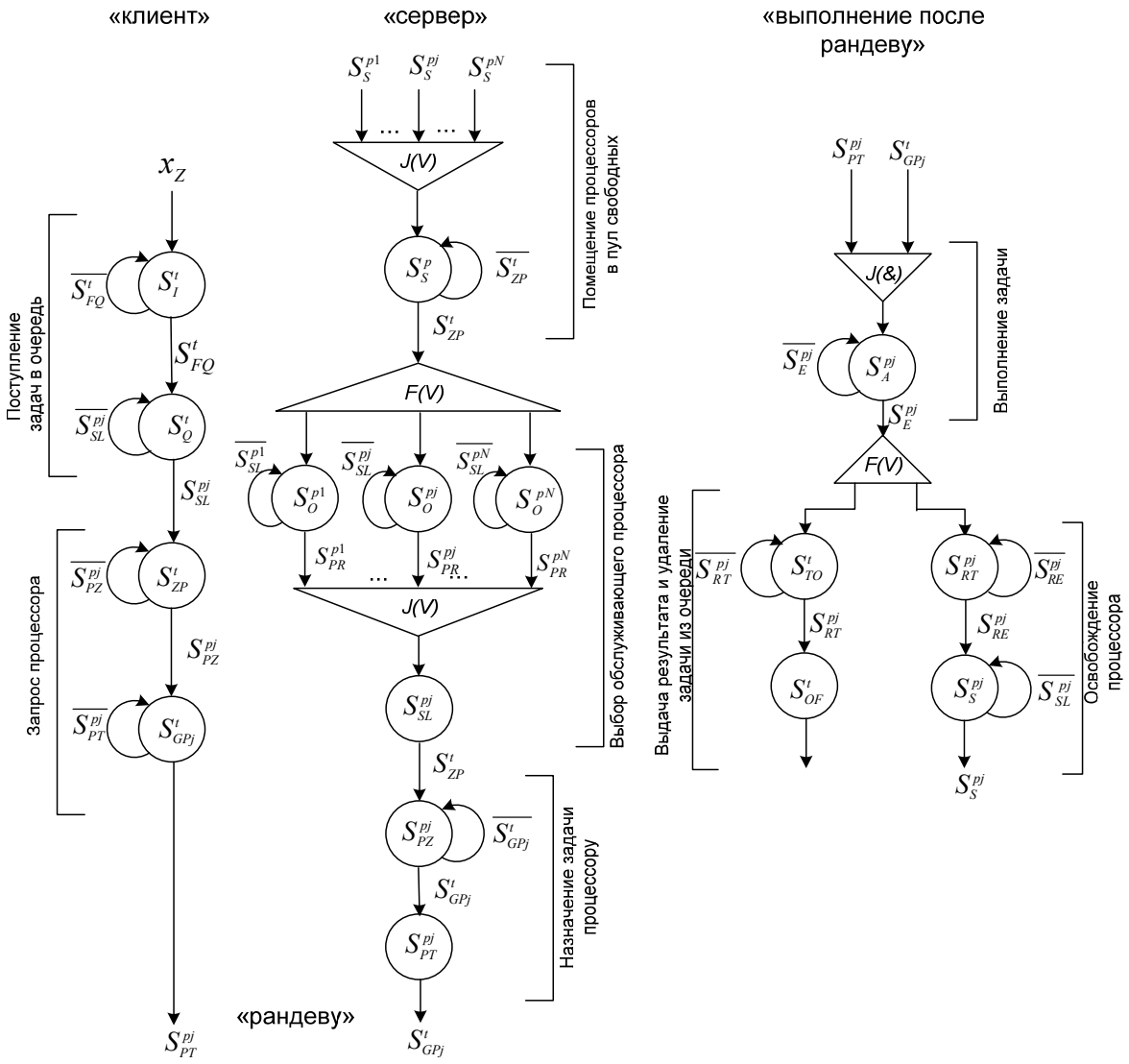


Рис. 8.15. Граф НДА алгоритма планирования/диспетчеризации задач в многопроцессорной системе

Следует иметь в виду, что в данном (аппаратно-ориентированном) алгоритме функция назначения задач выполняется диспетчером задач (клиентом). Он определяет наличие задачи в очереди, наличие свободных процессоров в пуле, формирует сигнал запроса процессора, дожидается ответного сигнала от процессорного блока (сервера), при получении которого возникает условие (рандеву), при котором задача направляется на выполнение в выбранный процессор.

Представленная система канонических уравнений и граф НДА алгоритма планирования/диспетчеризации с глобальной очередью задач отличаются тем, что здесь имеет место процедура выбора процессора из всех находящихся в режиме ожидания. Следует иметь в виду, что функция $F(V)$ является разветвительной вершиной, за которой следует выполнение параллельных процессов по обеспечению состояния ожидания процессоров, входящих в пул свободных.

8.5.3. Структура аппаратного планировщика/ диспетчера с глобальной очередью задач

Аппаратная поддержка алгоритма диспетчеризации задач реализована в упрощенном варианте, а именно: в части постановки задачи в очередь, выбора задачи для обслуживания в процессоре, синхронизации действий серверной части (процессоров) и клиентской (задачи) на основе метода «рандеву», выбор обслуживающего процессора. Другие функции, такие как сохранение и восстановление контекста, перемещение процессов из очереди готовых задач в очередь ожидающих и т.п., реализуются традиционным путем и в данной модели не рассматриваются.

Устройство аппаратной поддержки рассмотрено на примере многопроцессорной системы, состоящей из восьми процессоров, объединенных общей шиной [127]. Возможны два варианта подключения диспетчера к процессорному узлу. В первом варианте для передачи сигналов между процессором и диспетчером используется общая шина, что создает нежелательный дополнительный трафик на ней. Во втором варианте устройство диспетчеризации имеет собственный интерфейс взаимодействия с каждым процессором, т.е. в каждом процессорном узле реализован интерфейс для запроса и приема идентификаторов задач от диспетчера [109, 110]. Реализован второй подход, хотя он несколько усложняет процессорный блок, поскольку требует дополнительных выводов для вновь введенных сигналов. Общий вид системы представлен на рис. 8.16.

К преимуществам такой организации также можно отнести тот факт, что выход из строя одного из процессоров не влияет на работоспособность всей системы в целом. Если выйдет из строя один или несколько процессоров, то работоспособность многопроцессорной системы сохранится при некотором снижении общей производительности.

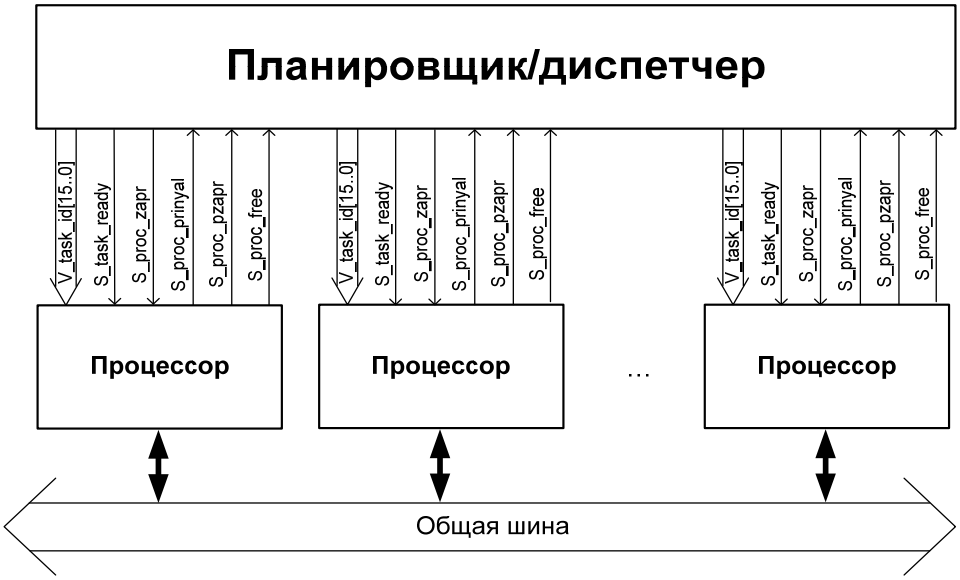


Рис. 8.16. Общий вид многопроцессорной системы
с аппаратной поддержкой планировщика/диспетчера задач

Интерфейс взаимодействия процессора с диспетчером показан на рис. 8.17.

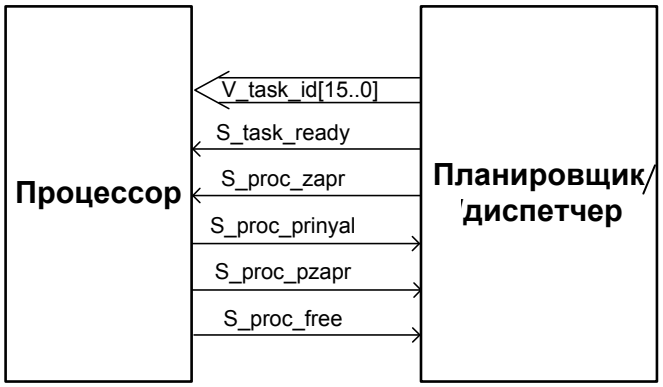


Рис. 8.17. Интерфейс взаимодействия одного процессора
с планировщиком/диспетчером

Клиентский процесс развивается следующим образом. Если имеется некоторый процесс, ожидающий обслуживания, то диспетчер при условии, что имеются свободные процессоры, устанавливает активный уровень сигнала «Запрос» (*S_proc_zapr*) на линии того процессора, который выбирается из числа свободных по заданному алгоритму. Процессор, получивший сигнал диспетчера о его выборе для обслуживания текущей задачи, выходит из спящего режима, и сообщает диспетчеру о том, что готов к приему и выполнению задачи. Для этого процессор устанавливает активный уровень сигнала «Подтверждение запроса» (*S_proc_pzapr*),

который воспринимает диспетчер. Кроме того, процессор снимает сигнал «Свободен» (*S_proc_free*), тем самым выходит из пула свободных. Процессор назначен, и далее диспетчер переходит к непосредственной передаче идентификатора задачи процессору. Для этого он извлекает идентификатор задачи из внутреннего регистра, выставляет его на шину (*V_task_id*) и передает в процессор сигнал «Процесс готов» (*S_task_ready*). Процессор воспринимает активный уровень сигнала «Процесс готов», тем самым ему гарантируется, что данные на шине достоверны (*V_task_id*), другими словами, на шине присутствует идентификатор текущей задачи. Процессор фиксирует эти данные в своем внутреннем регистре, и, когда его работа с шиной будет окончена, он выставляет сигнал «Процесс принят» (*S_proc_prinyal*).

Серверный процесс развивается так. Процессор, не занятый обслуживанием задачи (свободный), находится в спящем режиме. Сигнал «Свободен» (*S_proc_free*), поступающий от процессора к диспетчеру, принимает в этом случае активный уровень. Если в очереди имеются готовые задачи, выбирается обслуживающий процессор по приоритетному алгоритму, причем первый приоритет в данном случае присвоен процессору с номером 1. Выбранный процессор выходит из спящего режима, выбирает задачу из начала очереди (*S_fifo_read*), получает идентификатор задачи по шине (*V_task_id*), выставляет сигнал «Процесс принял» (*S_proc_prinyal*) и переходит в режим «Занято».

После этого планировщик/диспетчер может снять данные с шины и перейти к назначению следующей задачи. Процессор, получивший идентификатор задачи, переходит к ее обслуживанию. Все сигналы интерфейса сняты (на всех линиях устанавливается пассивный уровень). По завершению работы процессор известит диспетчер о том, что свободен, установлением активного уровня сигнала «Свободен» (*S_proc_free*), тем самым снова окажется в пуле свободных процессоров, и диспетчер вновь будет учитывать его при распределении задач.

Функционально устройство планирования/диспетчеризации задач представлено в виде 4 блоков (рис. 8.18):

– блок управления очередью задач;

– блок очереди задач FIFO;

– блок выбора процессора для обслуживания очередной задачи;

– блок синхронизации и назначения задач по процессорам.

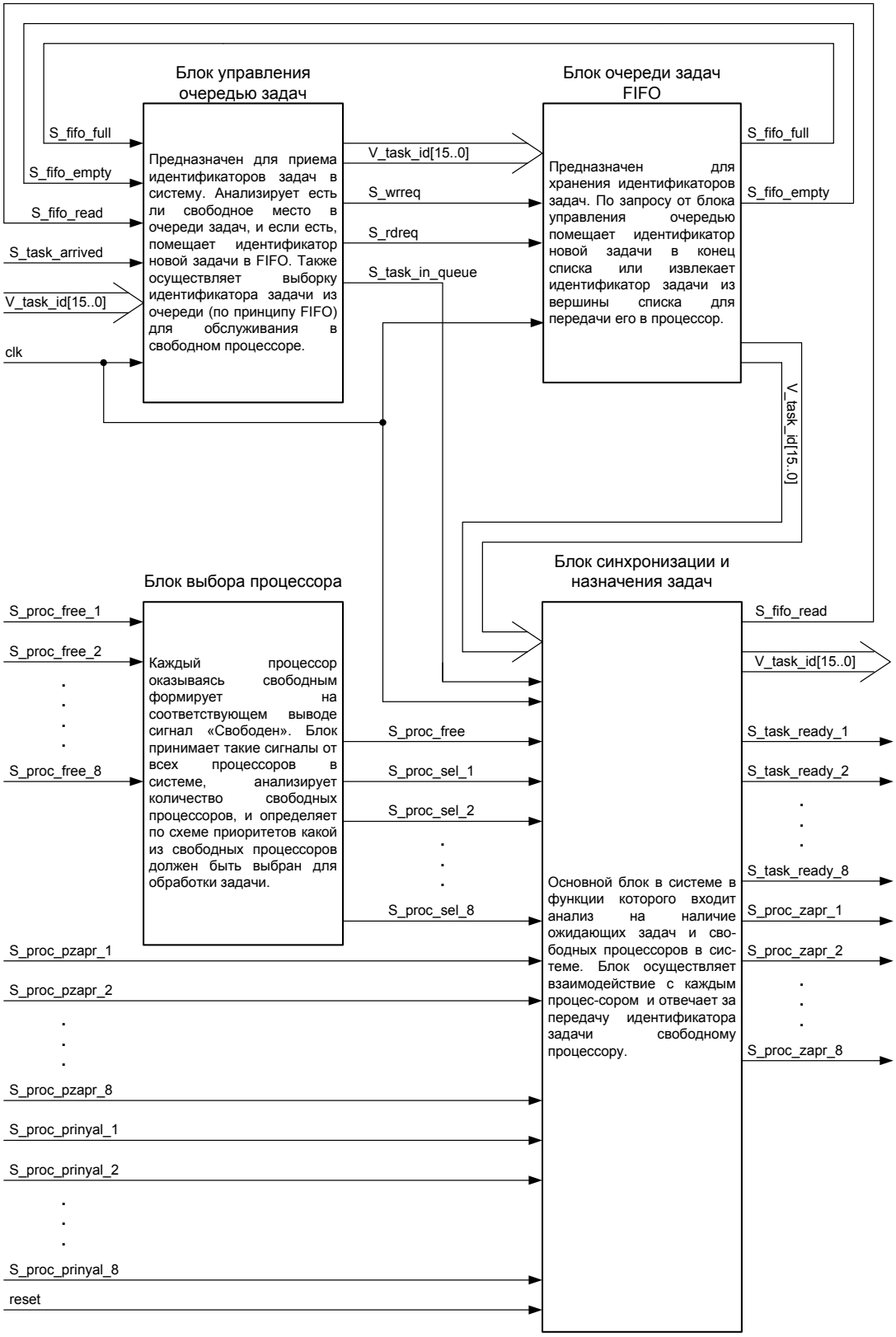


Рис. 8.18. Структура устройства диспетчеризации задач

Блок управления очередью задач предназначен для приема идентификаторов задач в систему. Он производит анализ на наличие свободных мест в очереди задач и, если они есть, помещает

идентификатор новой задачи в блок очереди FIFO. Также осуществляет выборку идентификатора задачи из очереди по принципу «первым пришел – первым обслужился» и помещает его в свободный процессор.

Таким образом, блок выполняет часть функций планировщика, т.е. формирует очередь задач и отслеживает ее состояние, используя для этого два сигнала. Активный уровень сигнала «FIFO заполнено» (*S_fifo_full*) сообщает планировщику о том, что принимать новые задачи запрещено, так как нет свободных регистров для записи и хранения новых идентификаторов. Так же обстоит дело и с поступающими процессами. Если в очереди нет места для записи новых идентификаторов задач, то они планировщиком не воспринимаются. Второй сигнал – «FIFO пустое» (*S_fifo_empty*) указывает на то, что ни один идентификатор в очередь не поставлен, т.е. очередь задач пуста. О наличии задачи в очереди свидетельствует сигнал *S_task_in_queue*. На его основе делается решение о том, выводить из спящего режима процессор или нет. Если очередь задач пуста, то блок синхронизации не обращается к процессорам. Сигналы «FIFO заполнено» (*S_fifo_full*) и «FIFO пустое» (*S_fifo_empty*) формируются и поступают в блок управления очередью задач из блока FIFO.

В блоке управления очередью задач имеется шина данных (*V_task_id*), по которой передаются идентификаторы задач. О том, что на шине присутствует новый процесс, планировщика извещает нарастающий фронт сигнала «Поступил процесс» (*S_task_arrived*). Если очередь не заполнена, то планировщик принимает данные с шины *V_task_id* и записывает их в очередь FIFO, в конец списка. Если очередь FIFO заполнена, данные на входной шине игнорируются. Запрос очередной задачи выполняет один из процессоров, выбранный в качестве обслуживающего. Для этих целей используется сигнал «Запрос задачи» (*S_fifo_read_j*). На основании этого сигнала блок диспетчеризации формирует сигнал «Чтение FIFO» (*S_fifo_read*) и передает его блоку управления очередью задач.

Планировщик управляет блоком очереди задач FIFO, пользуясь сигналами *S_wrreq* и *S_rdreq*. *S_wrreq* используется для записи в очередь FIFO, а *S_rdreq* – для чтения из нее.

Блок очереди задач FIFO хранит поступающие идентификаторы задач. Идентификатор является частью контекста задачи и может состоять из нескольких полей, одно из которых указывает на адрес задачи в памяти многопроцессорной системы. По запросу

от блока управления очередью заносит идентификатор новой задачи в конец списка или извлекает идентификатор запускаемой задачи из вершины списка для передачи его в процессор.

Блок выбора процессора принимает все сигналы «Свободен» (*S_proc_free*) от всех процессоров в системе. Он определяет, присутствуют ли в данный момент в системе свободные процессоры. Если в наличии имеется хотя бы один процессор, не занятый обслуживанием текущей задачи, то этот блок устанавливает активный уровень сигнала «Есть свободные процессоры» (*S_proc_free*) и таким образом извещает блок управления назначением задач о том, что имеются «спящие» процессоры.

Выбор процессоров осуществляется схемой взаимоисключения и схемой приоритетов, реализация которой показана в [104]. Для сообщения блоку диспетчеризации о том, какой из процессоров выбран для обслуживания, служат выводы (*S_proc_sel*_1,..., *S_proc_sel*_8).

Блок синхронизации и назначения задач анализирует информацию о наличии ожидающих задач в очереди и свободных процессоров, назначаемых для обработки этих задач. Данный блок, используя механизм синхронизации «рандеву», осуществляет взаимодействие очереди задач с процессорами по описанному ранее интерфейсу. Блок выбора процессора формирует информацию о том, какой из свободных процессоров выбран для обслуживания текущей задачи. Идентификатор принятой на обслуживание задачи передается блоком очереди задач FIFO по шине *V_task_id*. Кроме того, блок управления очередью задач сообщает о состоянии очереди и осуществляет выборку идентификатора задач из блока очереди FIFO по требованию блока синхронизации и назначения задач.

8.5.4. Моделирование на языке VHDL и анализ результатов

Модель устройства планировщика/диспетчера была реализована на языке VHDL в виде четырех программных модулей. Первый модуль реализует логические функции первого блока – управление очередью задач. Второй модуль – это блок FIFO. Третий модуль – блок управления свободными процессорами, четвертый модуль – блок синхронизации. В качестве примера ниже при-

веден фрагмент программы для реализации блока управления очередью, для чего составлено следующее описание:

```
28. LIBRARY ieee;
29. USE ieee.std_logic_1164.all;
30. USE ieee.std_logic_unsigned.all;
31. USE ieee.std_logic_arith.all;
32. ENTITY block_upr_ocher IS
33. PORT
34. (
35. CLK : IN STD_LOGIC;
36. V_task_id : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
37. S_task_arrived : IN STD_LOGIC;
38. S_fifo_full : IN STD_LOGIC;
39. S_fifo_empty : IN STD_LOGIC;
40. S_fifo_read : IN STD_LOGIC;
41. V_fifo_data_o : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
42. S_wrreq : OUT STD_LOGIC;
43. S_rdreq : OUT STD_LOGIC;
44. S_task_in_queue : OUT STD_LOGIC
45. );
46. END block_upr_ocher;
47. ARCHITECTURE block_upr_ocher_arch OF block_upr_ocher IS
48. SIGNAL reg_task_id : STD_LOGIC_VECTOR(15 DOWNTO 0);
49. SIGNAL S_task_arrived1 : STD_LOGIC;
50. SIGNAL S_task_arrived2 : STD_LOGIC;
51. BEGIN
52. PROCESS
53. BEGIN
54. WAIT UNTIL CLK='1';
55. S_task_in_queue<=not S_fifo_empty;
56. reg_task_id<=V_task_id;
57. S_task_arrived1<=S_task_arrived;
58. S_task_arrived2<=S_task_arrived1;
59. IF S_task_arrived1='0' AND S_task_arrived2='1' THEN
60. IF S_fifo_full='0' THEN
61. S_wrreq<='1';
62. V_fifo_data_o<=reg_task_id;
```

63. *END IF*;
64. *ELSE*
65. *S_wrreq<='0'*;
66. *END IF*;
67. *IF S_fifo_read='1' THEN*
68. *S_rdreq<='1'*;
69. *ELSE*
70. *S_rdreq<='0'*;
71. *END IF*;
72. *END PROCESS*;
73. *END block_upr_ocher_arch*;

В заголовке (строки 5–19) указываются входные и выходные переменные. Их имена соответствуют описанным выше сигналам для блока управления очередью задач. Непосредственная реализация логических выражений выполнена в строках 28–45, которые входят в состав процесса (строки 25–46), задающего работу синхронных схем по переднему фронту синхросигнала *CLK*. Этот модуль использовался как компонент в модели устройства диспетчера, результаты исследования которого будут приведены далее.

Ввод схем и работа производились в свободно распространяемой версии системы *Xilinx ISE WebPACK Design Suite* 12.1 [133], моделирование производилось в той же системе.

Прежде чем перейти к моделированию, необходимо описать еще два дополнительно разработанных блока не являющихся частью диспетчера, но входящих в состав макета и применявшихся на этапе моделирования.

Блок генерации задач. В процессе моделирования необходимо задавать диспетчеру идентификаторы задач, готовых к выполнению. Задачи поступают в систему постоянно, и их количество может быть очень большим. Поэтому на входе блока управления очередью задач устанавливается блок генерации задач, который через равные промежутки времени выставляет новый идентификатор задачи и самостоятельно извещает об этом блок управления очередью, тем самым имитируя поток задач. Чтобы запустить работу блока генерации задач, достаточно подать активный уровень на вход блока start.

Блок процессора. Кроме генератора задач, дополнительно был разработан блок имитации работы процессора, который полностью поддерживает интерфейс «процессор–диспетчер». Когда

идентификатор задачи получен блоком имитации процессора, запускается счетчик, отсчитывающий несколько десятков тактов (значение можно задать любое). В это время данный блок отключается от диспетчера точно так же, как если бы реальный процессор, получив задачу, перешел бы к ее обслуживанию. Счетчик отсчитывает фактическое время, затрачиваемое процессором на обработку задачи, после чего переходит в пул свободных и сообщает об этом диспетчеру.

Общий вид структурной схемы модели диспетчера задач в составе восьмипроцессорной системы представлен на рис. 8.19.

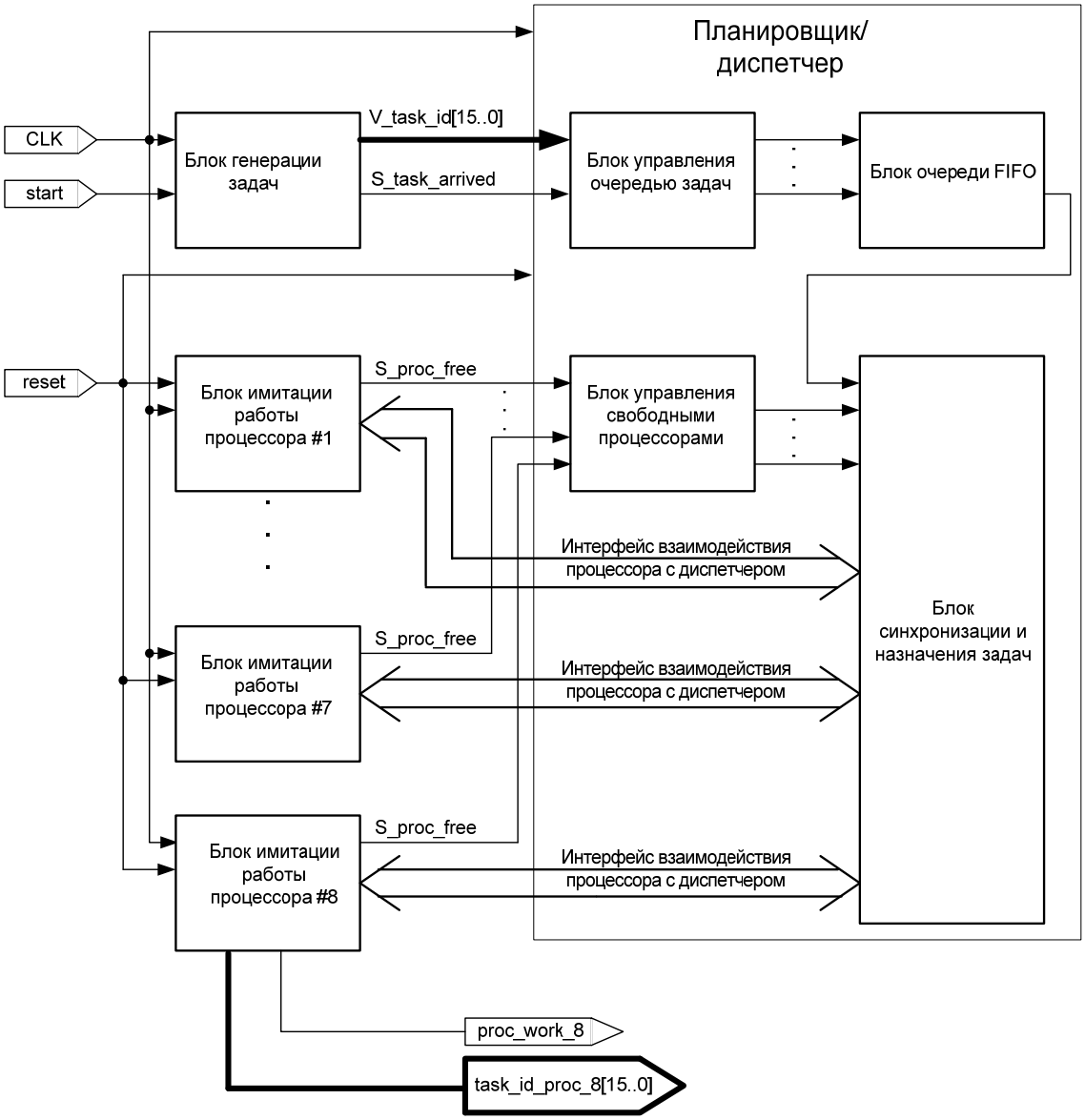


Рис. 8.19. Схема проведения экспериментальных исследований

Практически все описанные ранее сигналы для данной системы являются внутренними и генерируются самостоятельно. На входе системы подаются синхросигналы *CLK*, от которых тактируются все блоки, сигналы начального сброса системы (*reset*) и запуска генератора задач (*start*).

Для проверки результатов моделирования использовались выводы *proc_work_[j]* и *task_id_proc_[j][15..0]*. Первый вывод (*proc_work_[j]*) информирует о том, что *j*-й процессор получил идентификатор текущей задачи и приступил к ее обслуживанию. Шина *task_id_proc_[j][15..0]* передает информацию о том, какой идентификатор получил *j*-й процессор. Для контроля поступающих в систему идентификаторов задач фиксировались состояния шины данных блока очереди *FIFO*. Эти состояния на временной диаграмме обозначены как *Task_id_F*. Результаты моделирования представлены на рис. 8.20.

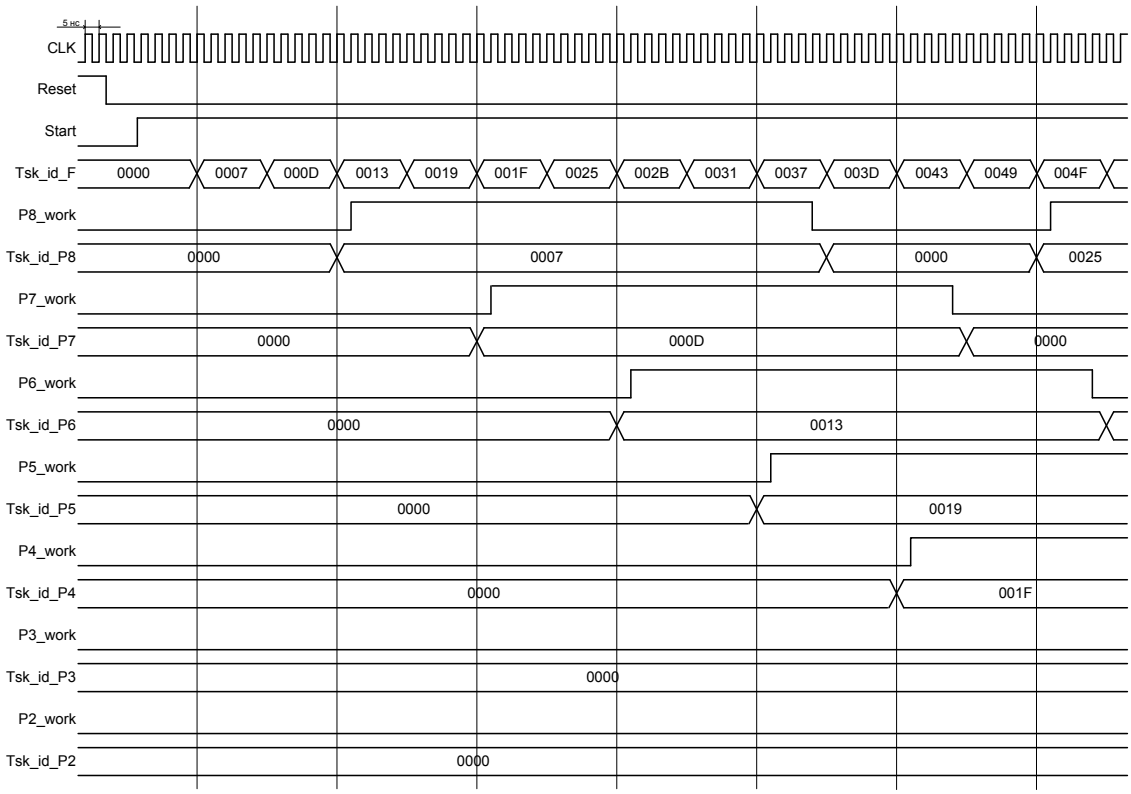


Рис. 8.20. Временные диаграммы работы аппаратного диспетчера с глобальной очередью

Из временной диаграммы видно, как идентификатор первой поступившей задачи (*«0007» hex*) ровно через 10 тактов поступил в процессор под номером 8, после чего процессор выставил сигнал о том, что занят обслуживанием. Так как восьмой процессор ока-

зался занятым, второй идентификатор задачи (*«000D» hex*) поступил через 10 тактов в процессор под номером 7. Этот процессор, аналогично предыдущему, выставил сигнал о том, что занят и приступил к обслуживанию второй задачи. Очевидно, что те 10 тактов, которые предшествуют назначению задачи процессору, уходят на работу самого диспетчера.

8.5.5. Структура многопроцессорной системы с аппаратным планировщиком/диспетчером задач на основе локальных очередей

Анализ временных диаграмм многопроцессорной системы, в которой используется планировщик/диспетчер с глобальной очередью задач, позволил выявить существенный недостаток такой организации, снижающий производительность всей системы в целом. Причина кроется в конфликтах, которые возникают, когда к диспетчеру может одновременно обратиться несколько свободных процессоров за назначением очередной задачи [72, 111].

Например, в системе освободилось сразу 10 процессоров, в очереди присутствует множество ждущих задач. Диспетчер начинает последовательно взаимодействовать с каждым процессором, назначая ему задачу, при этом он обращается к очереди. Если условно обозначить время, которое диспетчер затрачивает на взаимодействие с одним процессором, как 10 тактов, то первый процессор приступит к работе через 10 тактов, второй – через 20, десятый – через 100. К тому времени в системе могут освободиться еще процессоры, и им также придется ждать, пока диспетчер приступит к их обслуживанию. Таким образом, складывается ситуация, при которой в системе при наличии свободных процессоров происходит задержка обработки ожидающих задач из-за «нерасторопности» диспетчера.

Другой причиной снижения производительности является высокая вероятность перезагрузки кэш в системах с квантованием. Так, при переключениях контекста незавершенный процесс помещается в конец очереди (при обслуживании по принципу FIFO), после чего может быть назначен диспетчером не в тот процессор, в котором он обрабатывался ранее и где кэш полон блоками программы и данных, а в любой другой. Поэтому потребуется выгрузка содержимого кэш из предшествующего процессора и загрузка кэш текущего, на что значительно тратится процессорное время [72].

В планировщиках/диспетчерах с локальными очередями у каждого процессора имеется своя очередь задач. Функция диспетчера ограничивается выборкой очередной задачи и назначением ее освободившемуся процессору. При прерываниях по истечении кванта (переключениях контекста) процесс остается в той же очереди, в которой он находился ранее. Таким образом, в вычислительной системе действуют одновременно *N* диспетчеров, в результате чего задачи выбираются из очередей бесконфликтно, поступают в процессоры параллельно, что создает условия для повышения производительности [111].

Диспетчер задач является распределенным и выполняется в виде независимого аппаратного устройства в составе многопроцессорной системы в соответствии со схемой, представленной на рис. 8.21.

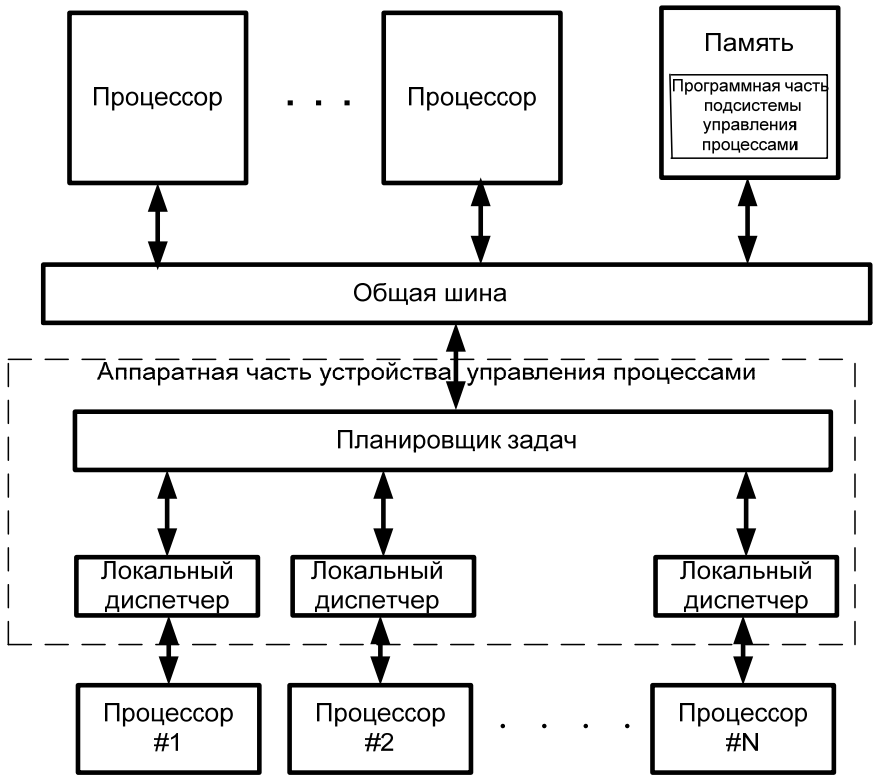


Рис. 8.21. Схема подключения устройства управления процессами в многопроцессорную систему

Аппаратная часть устройства управления процессами содержит два основных блока: планировщик задач и локальный диспетчер. Первый принимает извне новую задачу и передает ее локальному диспетчеру. В реальной системе эта процедура заключается в следующем: новой задаче выделяется раздел общей памяти

и присваивается соответствующий идентификатор, однозначно указывающий на адрес задачи в памяти. Идентификатор задачи помещается в очередь одного из локальных диспетчеров.

Модель многопроцессорной системы с использованием аппаратного устройства управления процессами представлена на рис. 8.22 [111]. Локальный диспетчер на основании идентификатора прерывает выполнение текущей задачи и организует запуск следующей по очереди задачи. Для упрощения программной модели принято, что часть функций планировщика и диспетчера задач, которые связаны с прерыванием по истечении кванта процессорного времени, сохранением и восстановлением контекстов задач и т.д., реализуется традиционно.

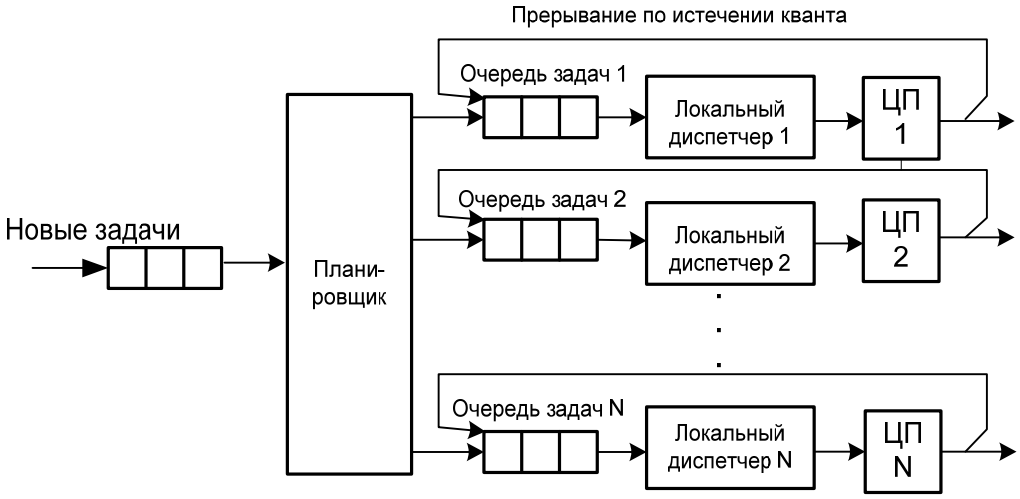


Рис. 8.22. Модель многопроцессорной системы с локальными диспетчерами

Интерфейс взаимодействия устройства локального диспетчера с каждым из процессоров и с планировщиком задач представлен на рис. 8.23.

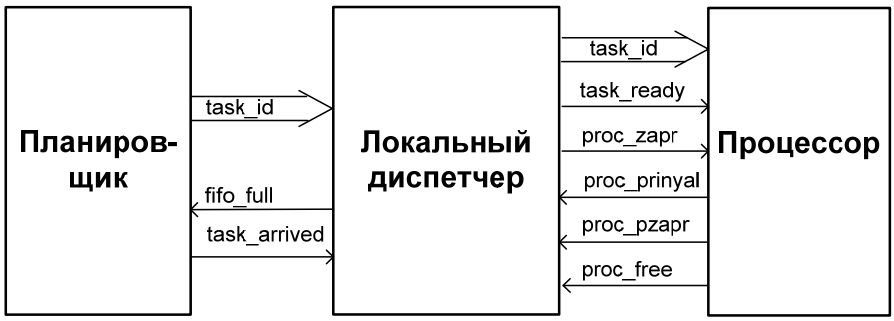


Рис. 8.23. Интерфейс взаимодействия локального диспетчера с планировщиком задач и процессором

Функционирование взаимодействующих устройств по предложенному интерфейсу происходит следующим образом. Если планировщик получил новую задачу, он пытается поместить ее в один из локальных диспетчеров. Если в очереди к локальному диспетчеру имеется процесс, ожидающий обслуживания, то он при условии, что процессор свободен, устанавливает активный уровень сигнала «Запрос» на линии (*proc_zapr*). Процессор выходит из спящего режима и сообщает локальному диспетчеру о том, что готов к приему и выполнению задачи. Для этого процессор устанавливает активный уровень сигнала «Подтверждение запроса» (*proc_pzapr*), который воспринимает локальный диспетчер. Кроме того, процессор снимает сигнал «Свободен» (*proc_free*). Далее локальный диспетчер переходит к непосредственной передаче идентификатора задачи процессору. Для этого он извлекает идентификатор задачи из очереди и выставляет его на шину данных процессора (*task_id*), после чего передает в процессор сигнал «Процесс готов» (*task_ready*). Процессор воспринимает активный уровень сигнала «Задача готова», тем самым ему гарантируется, что данные на шине достоверны (*task_id*). Процессор фиксирует эти данные в своем внутреннем регистре, и когда его работа с шиной будет окончена, он выставляет сигнал «Задача принята» (*proc_prinyal*).

Процессор, не занятый обслуживанием задачи, находится в спящем режиме. Сигнал «Свободен» (*proc_free*), поступающий от процессора к локальному диспетчеру, принимает активный уровень. Если в очереди имеются готовые задачи, процессор выходит из спящего режима, выбирает задачу из начала очереди (*task_read*), получает идентификатор задачи по шине (*task_id*), выставляет сигнал «Задача принята» (*proc_prinyal*) и переходит в режим «Занято». По завершению работы процессор извещает локальный диспетчер об освобождении установкой активного уровня сигнала «Свободен» (*proc_free*).

Алгоритм работы устройства управления процессами содержит две части: клиентскую и серверную. Клиентская часть осуществляет постановку задач в очередь, серверная часть выбирает задачу из «головы» очереди и запускает ее на обслуживание в процессор. Действия клиента и сервера синхронизируются по схемам «писатели-читатели» и «рандеву», причем последняя предполагает наличие выбранной задачи и готовность процессора к ее обслуживанию.

8.5.6. Формализация алгоритма планирования/ диспетчеризации с локальными очередями задач

Концептуально алгоритм управления процессами аналогичен алгоритму «спящего парикмахера» [72] и выглядит следующим образом. Первоначально поступивший процесс помещается в очередь планировщика и находится в ней до тех пор, пока не получит необходимые ему ресурсы, в число которых входит и место в очереди к одному из процессоров. Если имеется свободное место в одной из очередей, то выбранный планировщиком процесс занимает его, причем число мест в очереди ограничено. Принятая на обслуживание задача находится в очереди до тех пор, пока не поступит на выполнение в процессор. Если используется режим квантования, то незавершенная задача по окончании текущего кванта помещается в конец той же очереди, где она находилась ранее, иначе результат выдается пользователю, а в очереди освобождается одно место. При завершении задачи локальный диспетчер просматривает очередь, и если в ней имеются заявки на обслуживание, то назначается на выполнение процесс, стоящий в «голове» списка (очередь типа FIFO). Если очередь пуста, процессор переходит в режим ожидания.

Из приведенного описания следует, что алгоритм управления процессами использует две схемы синхронизации: «писатели-читатели» и «рандеву». Формальное описание алгоритма взаимодействия процессов в данной задаче базируется на использовании моделей недетерминированных автоматов [104]. Модели представляются в виде систем канонических уравнений, описывающих все реализуемые события управляющего алгоритма [109, 110].

Основные частные события алгоритма представлены в табл. 8.2. В последней колонке показано их соответствие сигналам на схеме устройства управления процессами.

Таблица 8.2

Обозначение события	Описание частного события	Сигналы на схеме
1	2	3
S_I^t	Ожидание поступления новой задачи	*S_task_arrived*
S_U^{Qj}	Функция состояния очереди (количество задач в очереди *j*-го процессора)	*S_use [5…0]*
S_{FQ}^t	Все очереди заполнены	*S_fifo_full*

Окончание табл. 8.2

1	2	3
$\overline{S_{FQj}^{t}}$	В *j*-й локальной очереди имеются свободные места	$\overline{S\_fifo\_fullj}$
S_{SL}^{Qj}	Выбор очереди с минимальным числом задач (занятых мест) и занесение новой задачи в выбранную очередь	*S_wrreg*
S_{Qj}^{t}	Задача в очереди *j*-го процессорного узла	*S_task_in_queue*
S_{ZPj}^{t}	Запрос *j*-го процессора локальным диспетчером	*S_proc_zapr_j*
S_{GPj}^{t}	Задача готова к обслуживанию в *j*-м процессоре	*S_task_ready_j*
S_{PZ}^{pj}	Подтверждение запроса *j*-м процессором	*S_proc_pzapr_j*
S_{S}^{pj}	Свободен *j*-й процессор	*S_proc_free*
S_{PT}^{pj}	Задачу *j*-й процессор принял	*S_proc_prinyal_j*
S_{A}^{pj}	Выполняет обслуживание задачи *j*-й процессор	Аппаратно не реализовано
S_{O}^{Qj}	Ожидание поступления задачи в очередь *j*-го процессора	Внутренний сигнал, на схеме не обозначен
S_{OF}^{t}	Удалить задачу из очереди	Внутренний сигнал, на схеме не обозначен
S_{TO}^{t}	Снять задачу с исполнения	Внутренний сигнал, на схеме не обозначен
S_{E}^{pj}	Выполнение задачи закончено	Внутренний сигнал, на схеме не обозначен
S_{RT}^{pj}	Выдача результата выполнения текущей задачи	Внутренний сигнал, на схеме не обозначен
S_{SZ}^{pj}	Процессор в режиме ожидания	Внутренний сигнал, на схеме не обозначен

На основании словесно представленного алгоритма управления процессами и введенных событий, реализуемых в этом алгоритме, система канонических уравнений, описывающих эти события, будет иметь следующий вид:

– для процесса «клиент», реализуемого локальным диспетчером до момента рандеву:

$$S_I^t(t+1) = x_Z \vee S_I^t \overline{S_{FQ}^t};$$

$$S_O^{Qj}(t+1) = S_{FQ}^t \vee S_O^{Qj} \overline{S_{SL}^{Qj}};$$

$$S_{SL}^{Qj}(t+1) = \bigvee_{\forall j \in N} S_O^{Qj} S_U^{Qj};$$

$$S_Q^t(t+1) = S_{SL}^{Qj} \vee S_Q^t \overline{S_{SZ}^{pj}};$$

$$S_{ZPj}^t(t+1) = S_Q^t S_{SZ}^{pj} \vee S_{ZPj}^t \overline{S_{PZ}^{pj}}; \qquad (8.9)$$

$$S_{GPj}^t(t+1) = S_{ZPj}^t S_{PZ}^{pj} \vee S_{GPj}^t \overline{S_{PT}^{pj}},$$

где x_Z означает наличие запроса задачи на постановку в очередь;

– для процесса «сервер», реализуемого процессором до момента рандеву:

$$S_{SZ}^{pj}(t+1) = S_S^{pj} \vee S_{SZ}^{pj} \overline{S_Q^t};$$

$$S_{PZ}^{pj}(t+1) = S_{SZ}^{pj} S_Q^t \vee S_{PZ}^{pj} \overline{S_{ZPj}^t}; \qquad (8.10)$$

$$S_{PT}^{pj}(t+1) = S_{PZ}^{pj} S_{ZPj}^t \vee S_{PT}^{pj} \overline{S_{ZPj}^t}.$$

Система канонических уравнений, описывающая события после рандеву:

$$S_A^{pj}(t+1) = S_{PT}^{pj} S_{GPj}^t \vee S_A^{pj} \overline{(S_E^{pj} \vee S_{PK}^{pj})};$$

$$S_{TO}^t(t+1) = S_A^{pj} S_E^{pj} \vee S_{TO}^t \overline{S_{RT}^{pj}};$$

$$S_{RT}^{pj} S_{TO}^t(t+1) = S_A^{pj} S_E^{pj} \vee S_{RT}^{pj} \overline{S_{TO}^t}; \qquad (8.11)$$

$$S_{OF}^t S_{TO}^t(t+1) = S_{RT}^t S_{TO}^{pj};$$

$$S_S^{pj} S_{TO}^t(t+1) = S_{RT}^t S_{OF}^{pj}.$$

Следует иметь в виду, что выбор очереди для записи в нее вновь поступившей задачи выполняется в соответствии с выражением

$$S_{SL}^{Qj} = \min_{(\forall j \in N)} \left\{ S_U^{Qj} \right\}. \qquad (8.12)$$

Выражение (8.12) означает взятие функции минимального количества занятых мест из всего массива локальных очередей, где N – число процессоров (очередей) в многопроцессорной системе.

Уравнениям соответствует граф недетерминированного автомата (рис. 8.24), содержащий клиентскую и серверную части

алгоритма управления взаимодействующими процессами до момента рандеву (до оператора объединения $J(\&)$) и после рандеву.

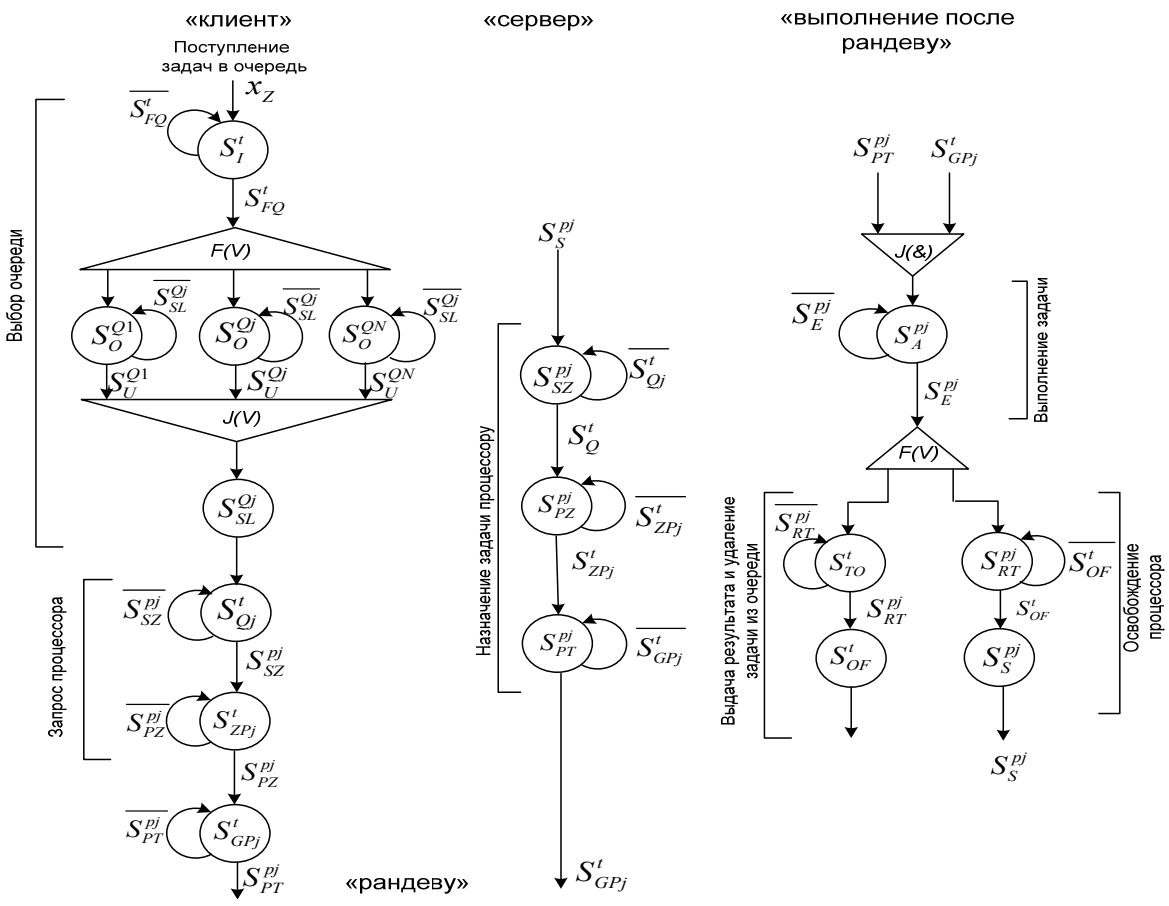


Рис. 8.24. Граф НДА алгоритма планирования/диспетчеризации с локальными очередями

Представленная система канонических уравнений и граф НДА алгоритма планирования/диспетчеризации с локальными очередями отличаются от соответствующей НДА-модели алгоритма планирования/диспетчеризации с глобальными очередями тем, что здесь присутствует алгоритм выбора локальной очереди, куда заносится вновь поступившая задача, и отсутствует процедура выбора процессора, поскольку локальную очередь обслуживает единственный процессор, за которым эта очередь закреплена. Функция $F(V)$ является разветвительной вершиной, за которой следует выполнение параллельных процессов, связанных с ожиданием поступления новой задачи в выбранную очередь.

8.5.7. Структура устройства планирования/ диспетчеризации с локальными очередями задач

Аппаратное устройство управления процессами представлено в виде трех типов блоков (рис. 8.25):

– блок планирования;

– *N*-блоков очередей со схемами управления, где *N* – число процессоров в многопроцессорной системе;

– блок выбора очереди.

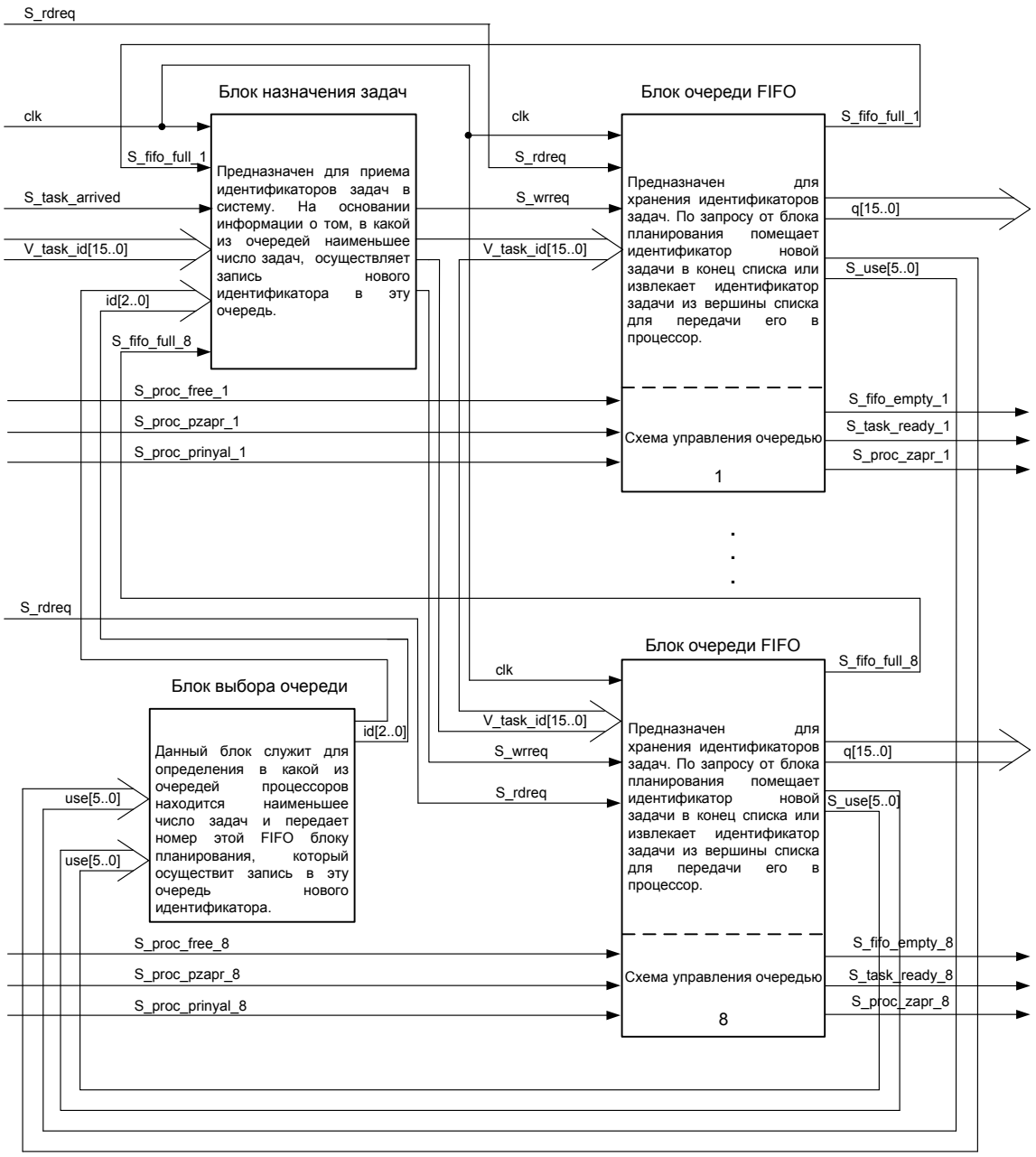


Рис. 8.25. Схема устройства управления процессами с локальными очередями

Блок очереди FIFO предназначен для хранения готовых к обслуживанию задач. Число этих блоков равно числу процессоров в системе. Функционально они не отличаются от тех же блоков в диспетчерах с глобальной очередью, но имеют меньшее количество ячеек для хранения задач. Несколько иначе они организованы, например, записывать в очередь может только диспетчер по сигналам записи *S_wrreq*, подаваемым из блока планирования. Считывание содержимого регистра очереди производится по сигналу *S_rdreq*, поступающему от каждого процессора по индивидуальным линиям.

Входная шина данных формируется блоком планирования (*V_task_id [15..0]*) и является общей для всех блоков очереди FIFO. Каждый процессор управляет только своей очередью, другими словами, он может читать содержимое только из одного блока FIFO. Также каждый процессор постоянно сканирует свою очередь (семафорный сигнал *S_fifo_empty*) на предмет наличия идентификаторов задач. Если в регистрах FIFO имеется идентификатор хотя бы одной задачи, процессор читает его, если нет, то переходит в режим ожидания.

Выходная шина данных из каждого блока FIFO заводится в индивидуальный процессор. Семафорный сигнал *S_fifo_full* используется блоком назначения задач для указания на заполнение очереди блока FIFO. По этому сигналу диспетчер прекращает прием нового идентификатора в данную очередь.

Блок выбора очереди предназначен для определения блока FIFO, содержащего наименьшее количество записей. Номер очереди FIFO с наименьшим числом записей передается в блок планирования по трехразрядной выходной шине *id [2..0]*.

Выбор наименее загруженной очереди осуществляется на основании специальных сигналов *use [5..0]*, вырабатываемых блоками очереди FIFO. Число занятых ячеек в каждом блоке очереди формируется как разность содержимого счетчиков записи и чтения. Блок выбора очереди осуществляет вычисление наименьшего значения из всех выставленных на шинах *use [5..0]* с помощью комбинационной схемы сравнения. На основании этого делается вывод о том, в какую очередь следует загружать идентификатор новой задачи.

Блок назначения задач принимает новый идентификатор задачи и под действием сигналов *S_wrreq* помещает его в наименее заполненную очередь FIFO. Кроме того, на данный блок может возлагаться функция контроля состояния всех процессоров и функция перераспределения задач по очередям для выравнивания нагрузки

на процессоры. Если один из процессоров выходит из строя, то запись новой задачи в его очередь не должна производиться, а его очередь необходимо перераспределить. Если очередь некоторых процессоров загружена трудоемкими процессами, то также должно осуществляться перераспределение. Однако с целью упрощения экспериментов функция контроля в данной модели не реализована.

8.5.8. Моделирование на языке VHDL и анализ результатов

Для моделирования работы планировщика/диспетчера с локальными (распределенными) очередями была применена схема, показанная на рис. 8.26.

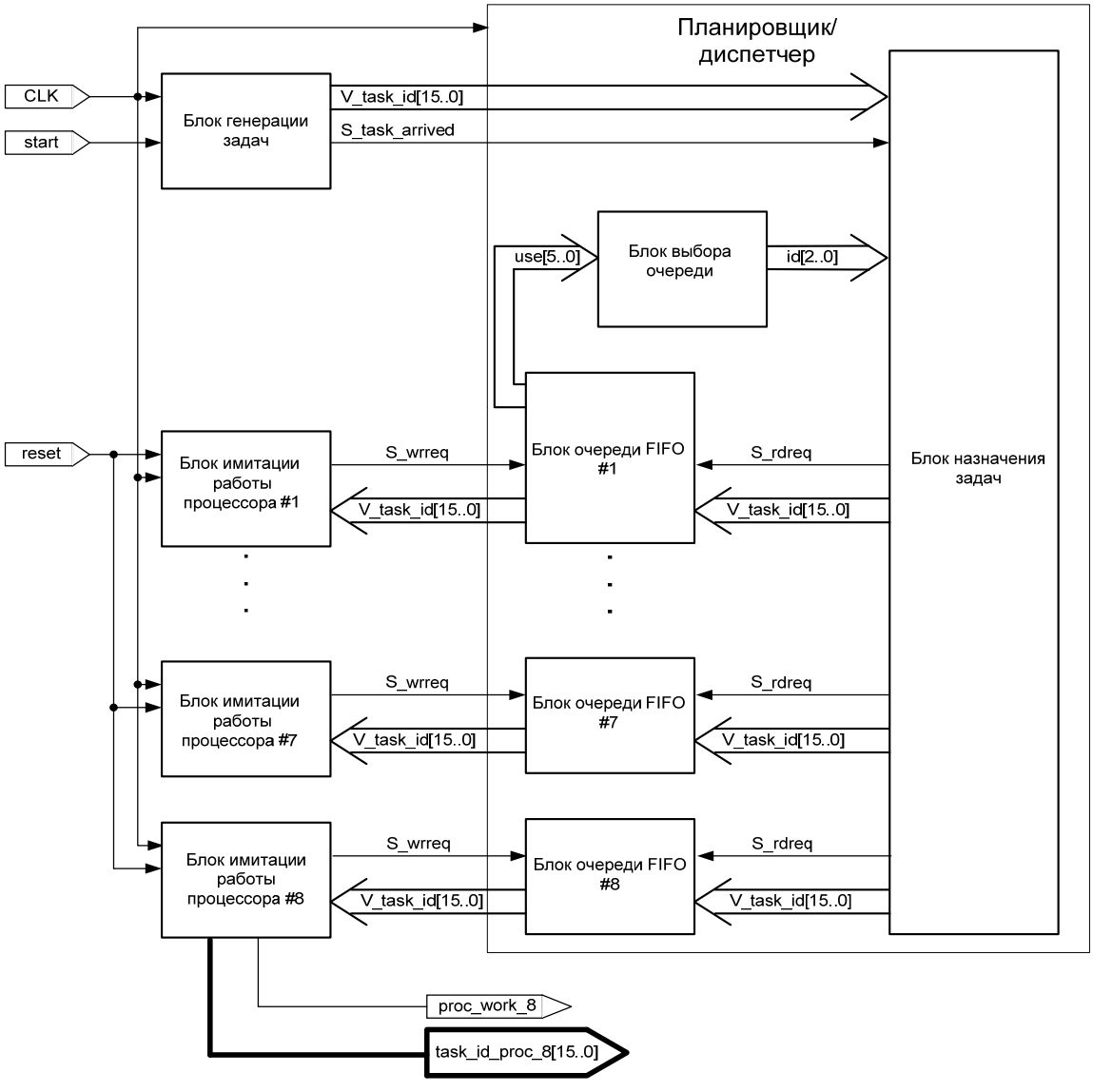


Рис. 8.26. Схема для проведения экспериментальных исследований

Чтобы иметь возможность сравнить результаты работы обоих вариантов реализации диспетчера в многопроцессорной системе, при моделировании использовались те же самые блоки имитации работы процессора и тот же блок генерации задач. Таким образом, условия моделирования системы остались прежними: задачи поступают в систему с периодичностью в 5 тактов, любой процессор обслуживает одну задачу на протяжении 32 тактов.

Временные диаграммы, полученные в результате проведенного моделирования, представлены на рис. 8.27.

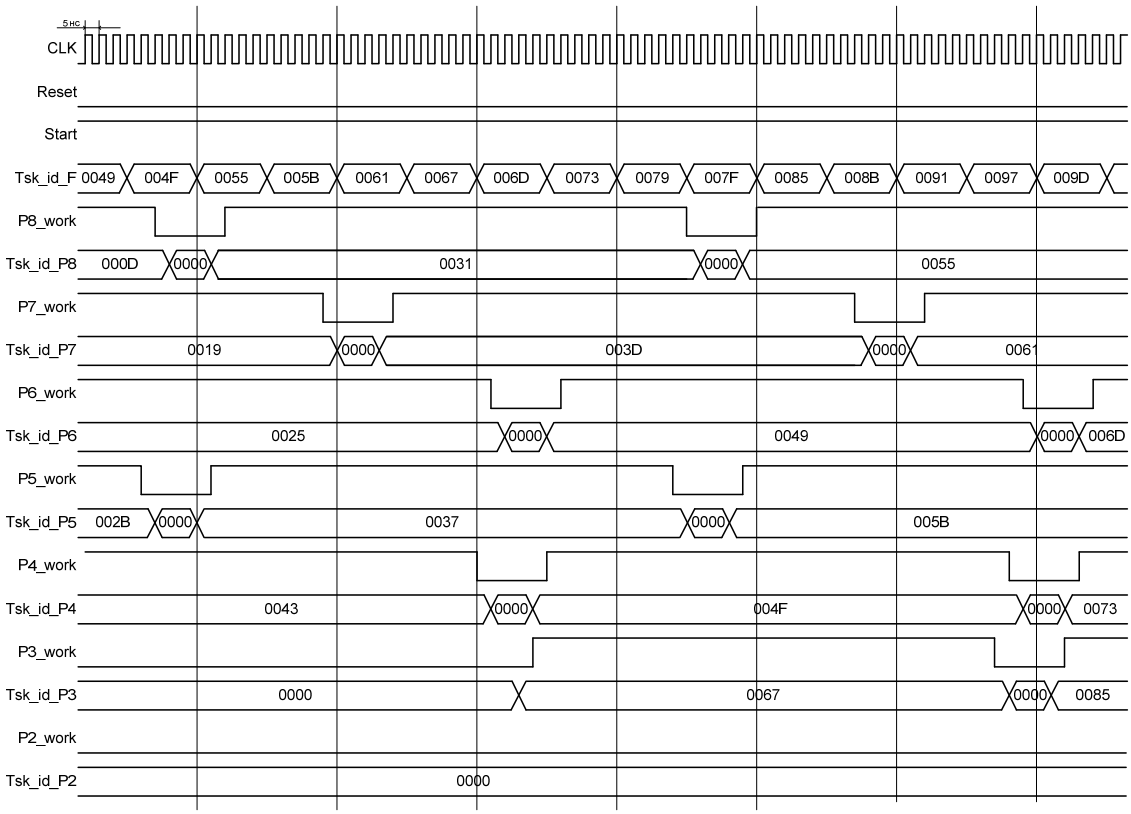


Рис. 8.27. Временные диаграммы работы аппаратного диспетчера с распределенными очередями

Главное отличие в результатах заключается в том, что при первом варианте организации диспетчера задачи накапливались в FIFO и постепенно заполняли ее. В результате наступал момент, когда новые идентификаторы не принимались в систему, она становилась перегружена. Время обработки вновь поступившей задачи в этом случае значительно возрастает. При этом в системе постоянно присутствовали свободные процессоры, но возникающий конфликт не позволял им получать задачи.

Планировщик/диспетчер с локальными очередями демонстрирует результат лучше, система успевает принимать и назначать все поступающие задачи. Она не перегружается на любом отрезке времени, кроме того, способна обрабатывать и более интенсивный поток задач. Причина увеличения быстродействия проста и очевидна – диспетчер во втором случае работает в десять раз быстрее, при этом выполняя все возложенные на него функции. Недостаток планировщика/диспетчера с локальными очередями заключается в отсутствии автоматической балансировки загрузки процессоров. Известно, что примерно 20 % всех задач загружают процессоры почти на 80 % от всего времени [72]. Из-за этого нельзя оценивать загруженность процессора, основываясь лишь на том, сколько задач присутствует в его собственной очереди. Ведь не исключено, что у одного процессора в очереди может быть десять коротких задач, в то время как у другого их будет всего пять, но они будут занимать намного больше времени, чем десять коротких. Поступает новый процесс, и его справедливо было бы отправить в очередь процессора с десятью процессами, но в данном случае он еще больше нагрузит второй процессор с пятью трудоемкими процессами. Поэтому при распределении задач по процессорам необходимо учитывать то обстоятельство, сколько по времени выполняется каждый процесс, но это потребует значительного объема дополнительного оборудования или поддержки соответствующими функциями медленно действующей традиционной операционной системы.

Приложение к главе 8

П8.1. Макет устройства доступа к общему ресурсу на ПЛИС

Экспериментальная установка для исследования выполнена на отладочной плате *Spartan-3E Starter Kit*, в состав которой входят ПЛИС *XC3S500E-4FG320C*, ПЗУ для хранения конфигурации *XCF04S-VO20C*, кварцевый генератор синхроимпульсов с частотой 50 МГц и другие блоки. Программирование ПЛИС осуществлялось с помощью *JTAG*-кабеля *Parallel Cable* III. На вход исследуемой системы (ИМ) подавались сигналы от имитатора сигналов, представленного в виде блока выработки сигналов (БВС).

Схема устройства приведена на рис. П8.1 и содержит исследуемый модуль ИМ, блок генерации входных сигналов БВС и логический анализатор ЛА.

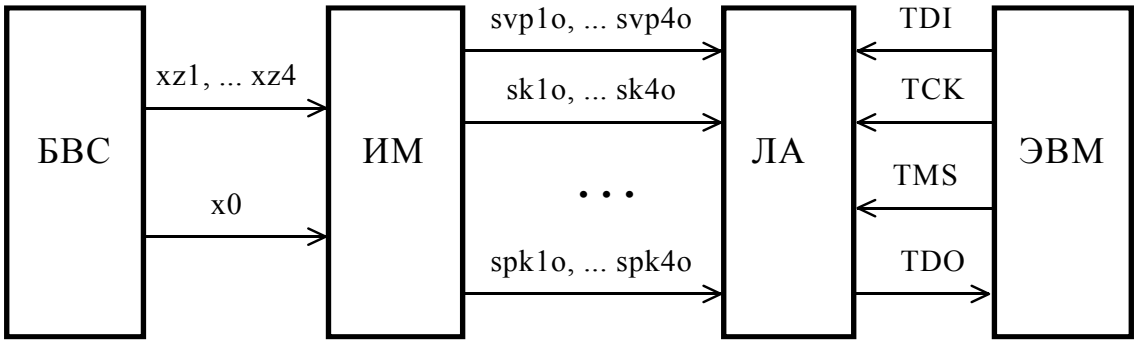


Рис. П8.1. Схема эксперимента

В качестве логического анализатора использовался модуль *ChipScope Pro* фирмы *Xilinx*, который входит в состав программного обеспечения с сокращенным сроком лицензии, поставляемого вместе с отладочной платой *Spartan-3E Starter Kit*. БВС, ИМ и ЛА реализованы внутри ПЛИС. Устройство подключается к ЭВМ через интерфейс *JTAG*, через который также производится программирование схемы.

После загрузки конфигурации в ПЛИС запускается логический анализатор, который захватывает и запоминает сигналы во внутреннем ЗУ. Затем информация из анализатора передается в ЭВМ и отображается на экране дисплея в виде временной диаграммы. Временная диаграмма, полученная во время экспериментального исследования, показана на рис. П8.2.

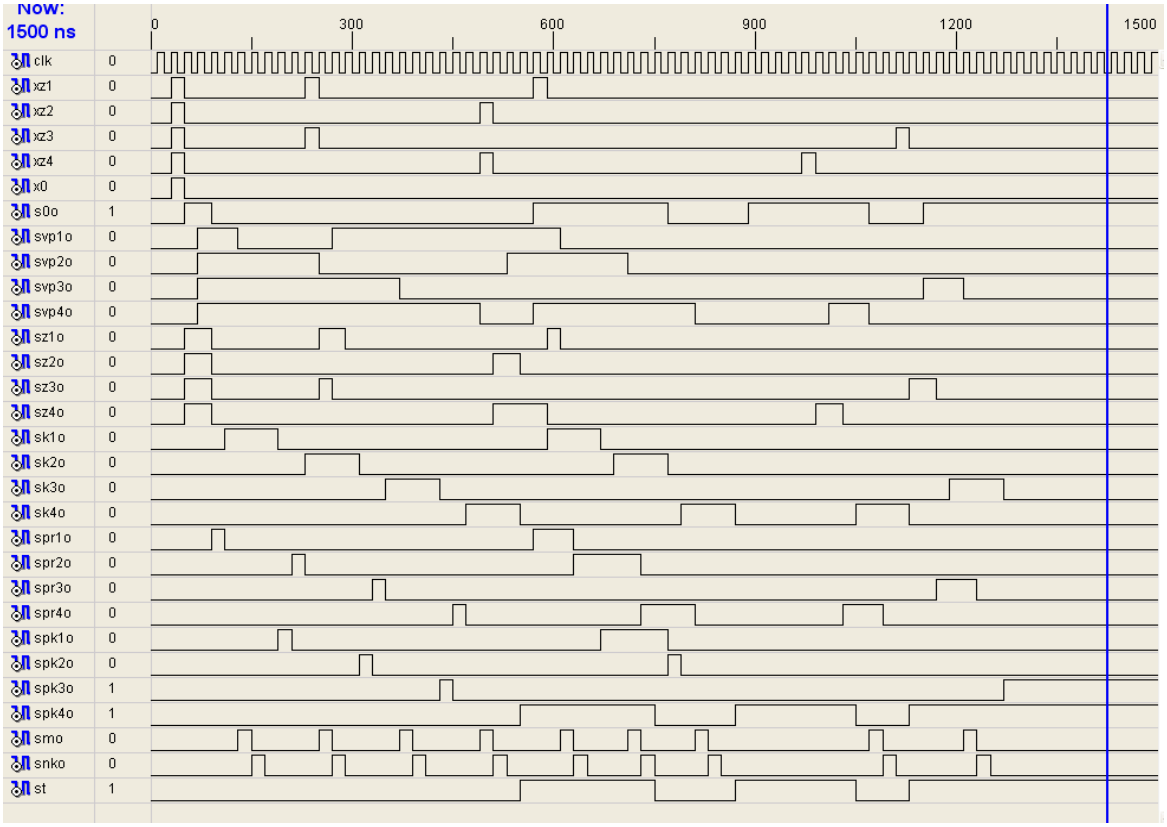


Рис. П8.2. Временные диаграммы функционирования блока синхронизации

Проведены экспериментальные исследования с получением временных диаграмм (см. рис. П 8.2) блока синхронизации, основанного на методе критической секции, для четырех параллельных процессов при их доступе к разделяемому ресурсу.

В результате проведенных исследований по аппаратной реализации алгоритмов синхронизации взаимодействующих параллельных процессов получены следующие результаты.

Проведенная экспериментальная аппаратная реализация устройства синхронизации на четыре входа (канала обслуживания) с использованием ПЛИС и полученные при этом временные диаграммы работы устройства полностью подтверждают правильность его функционирования. Определена латентность устройства синхронизации, которая составляет 2 цикла синхронизации тактового генератора, что соответствует трем процессорным тактам в реальной многопроцессорной системе. Полученная латентность значительно ниже, чем при программной реализации механизмов синхронизации процессов в традиционных операционных системах, где латентность битового семафора, как показывают измерения,

произведенные авторами с использованием специализированной программы измерения [128, 132], и зарубежными исследователями [130], составляет несколько тысяч процессорных тактов.

П8.2. Макет устройства доступа к общему ресурсу в составе четырехпроцессорной системы

Многопроцессорная система реализована на общей шине с использованием ПЛИС *Spartan-6*. Главным исполнительным элементом системы являются *soft*-процессоры *PicoBlaze* фирмы *Xilinx* (обозначен *kcpsm6*), вместе с ПЗУ (обозначено *prog*_1) они представляют блок *embedded_kcpsm6*_1 (рис. П8.3).

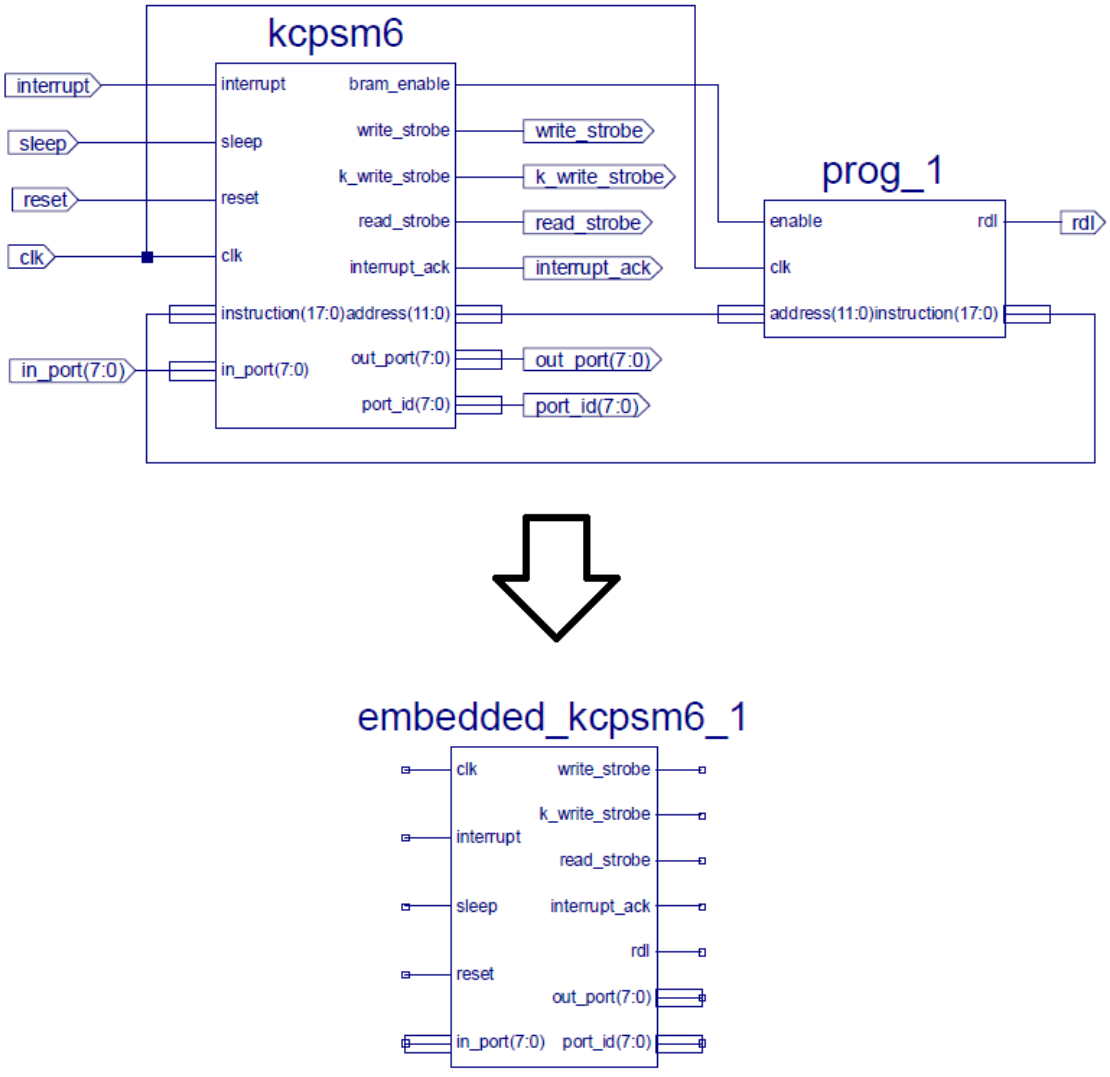


Рис. П8.3. Процессорный блок *PicoBlaze* и его функциональное обозначение

Для того, чтобы процессор мог взаимодействовать с шиной и блоком аппаратной синхронизации, потребовались дополнительные сигналы. Для их формирования был расширена схема процессорного узла с включением интерфейсного блока (*form_pico_ signal*), формирующего дополнительные сигналы: *asqure_bus* (захват шины), *free_bus* (освободить шину), *asqure_spin* (захват спин-блокировки), *free_spin* (освободить спин-блокировку). Сигналы формируются микропрограммами, хранящимися в блоке постоянной памяти *prog*_1. Сигналы *write_strobe* (запись в память), *read_strobe* (чтение памяти) предназначены для управления оперативной памятью. Шины *in_ port* (ввод), *out_port* (вывод), *port_id* осуществляют прием, выдачу данных и формируют адреса для памяти и периферии. Сигнал *k_write_strobe* управляет периферией, в частности, разрешает доступ к интерфейсному блоку *form_pico_signal*.

Два RS триггера (*sleep_trigger*) предназначены для перевода процессоров в режим сна подачей единичного уровня на вход *sleep*. Этот сигнал формируется при обращении к общему ресурсу. В случае занятости шины или блокирующей переменной, размещенной в аппаратном блоке синхронизации, он принимает единичное значение.

Все объединенные вместе вышеуказанные блоки представляют процессорный модуль *proc_module1* (рис. П8.4). Сигналы, адреса и данные, формируемые процессорным модулем или предназначенные процессорному модулю, обозначены *master*, а соответствующие сигналы, предназначенные для управления оперативной памятью, обозначены *slave.*

Сигналы, вырабатываемые процессорными модулями *reg* (3:0) предназначены для запроса общего ресурса (блокирующей переменной) и шины. Арбитр шины на основании сигнала от процессорного модуля *free_bus* (освободить шину) и *reg* (3:0) (запрос шины) формирует сигналы *gnt_num* (3:0) для управления мультиплексором (*mux*). Мультиплексор коммутирует восьмиразрядные шины адреса и данных на шину оперативной памяти.

Созданный процессорный модуль применен в четырехпроцессорной системе. Для взаимодействия устройств многопроцессорной системы была применена простая системная шина *easy_bus* собственной разработки, состоящая из арбитра (обозначен *easy_bus_arb*) и мультиплексора (обозначен *easy_bus_mux*) (рис. П8.5).

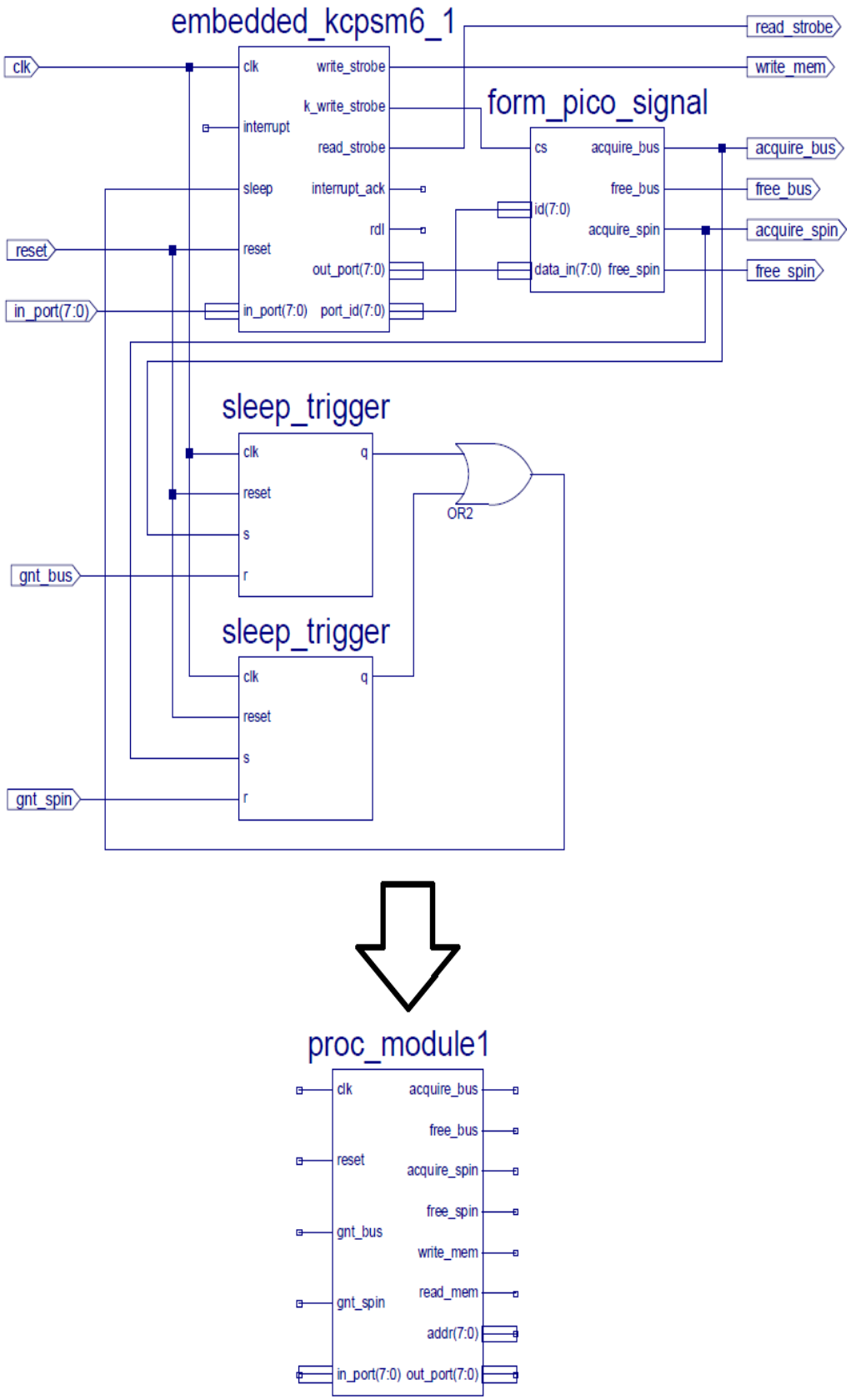


Рис. П8.4. Схема процессорного модуля и его функциональное обозначение

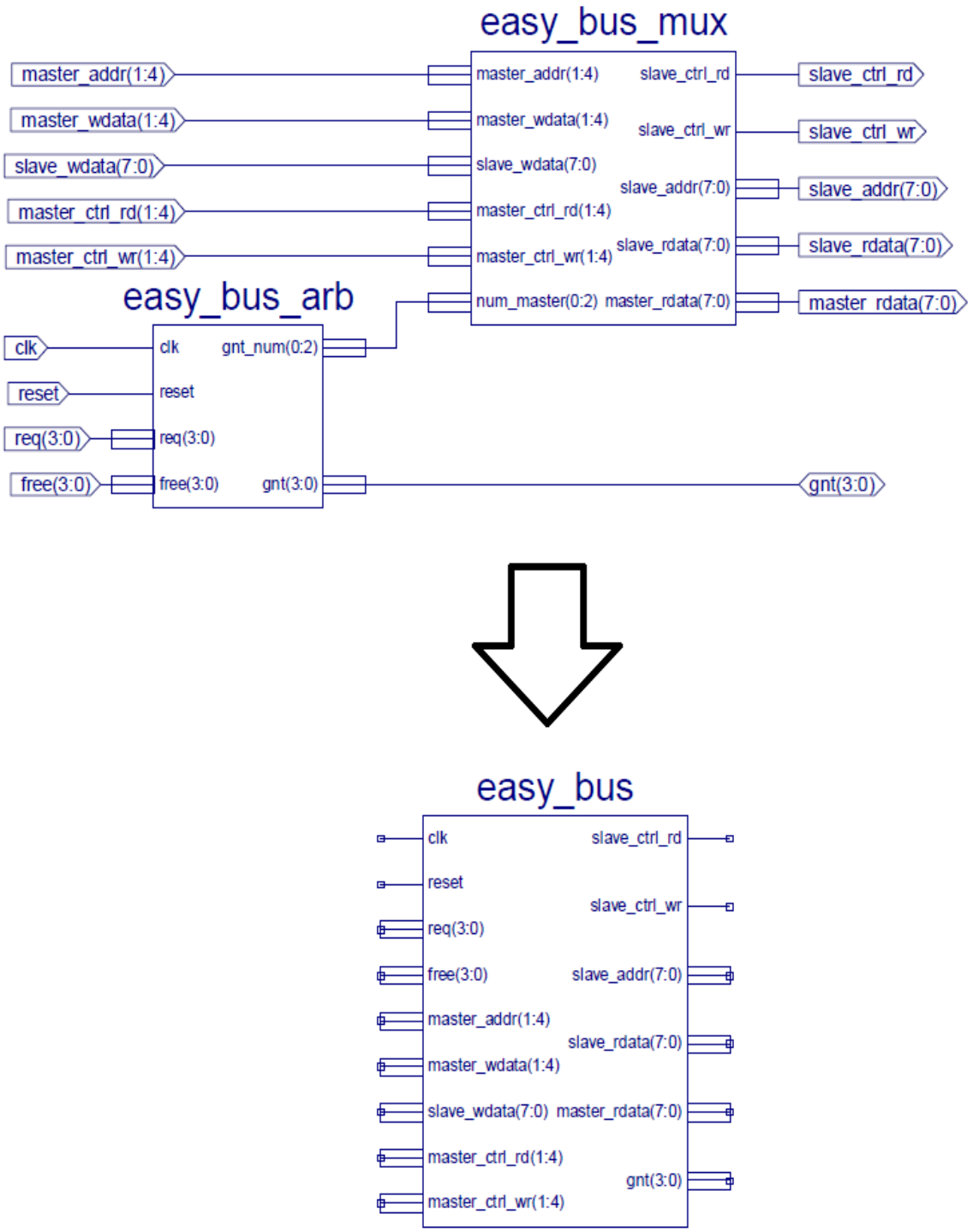


Рис. П8.5. Функциональная организация системной шины
и ее функциональное обозначение

Оперативная память типа ОЗУ подключена к процессорным модулям через шину. Функциональная схема блока ОЗУ размером 256 байт представлена на рис. П8.6.

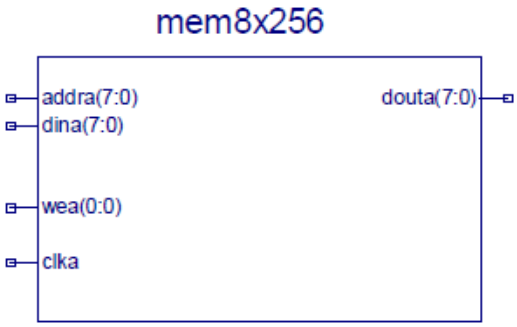


Рис. П8.6. Функциональное обозначение блока ОЗУ

В макете использован аппаратный блок синхронизации *easy_bus_arb*, аналогичный ранее разработанному блоку (рис. П8.7).

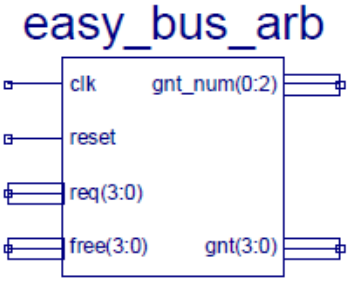


Рис. П8.7. Функциональное обозначение блока синхронизации

Схема макета моделируемой системы изображена на рис. П8.8, где представлен упрощенный вариант с одним процессорным узлом. Другие процессорные узлы подключаются аналогично. Также здесь показаны связи аппаратного блока синхронизации процессором.

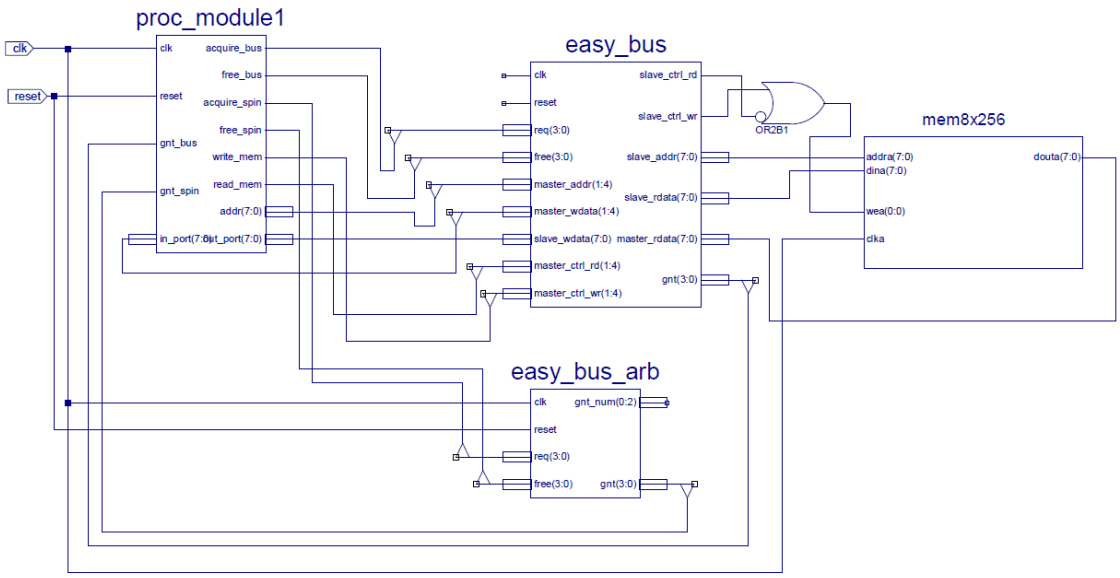


Рис. П8.8. Фрагмент многопроцессорной системы
с устройством синхронизации

В состав системы входят четыре процессорных модуля, системная шина, оперативная память и устройство синхронизации. Устройство реализует механизм синхронизации, основанный на spin-блокировке. Для захвата и освобождения блокировки в устройстве используются четыре сигнала *req* (3:0) и четыре сигнала *free* (3:0), по одному на каждый процессор. Для уведомления о предоставлении (захвате) ресурса используется четыре сигнала *gnt*, также отдельно для каждого процессора.

Алгоритм захвата блокировки выглядит следующим образом: процессор посылает сигнал *req* и считывает состояние сигнала *gnt*. Если *gnt* сигнализирует о предоставлении доступа, то процесс, выполняющийся в запрашивающем процессоре, вошел в критическую секцию и может осуществлять доступ к разделяемому ресурсу. Если доступ к ресурсу отсутствует (сигнал *gnt* имеет нулевой уровень), это означает занятость ресурса процессом, выполняющимся в другом процессоре. Запрашивающий процессор переходит в режим ожидания (спиннинга) и должен повторять попытку захвата до тех пор, пока блокировка не освободится. После завершения работы с разделяемым ресурсом процессор подает сигнал *free*, под действием которого освобождается устройство синхронизации и процесс выходит из критической секции. Макет предполагает другой вариант захвата и освобождения блокировки, когда запрашивающий процессор приостанавливается (засыпает) на занятой блокирующей переменной.

Для разрешения конфликтов, возникающих при одновременной подаче запросов на захват несколькими процессорами, используются приоритеты. Каждому входу *req* сопоставляется значение приоритета от 1 до *n*, где *n* – количество входов. При запросе устройства доступ предоставляется запросу с наибольшим приоритетом. Устройство синхронизации осуществляет циклическую смену приоритетов. После захвата устройства синхронизации приоритеты меняются следующим образом:

– значение приоритета входа, которому предоставляется доступ, становится равным 1 (обозначим этот приоритет *max_prior*);

– значения приоритетов входов, которые меньше *max_prior*, увеличиваются на единицу;

– значения приоритетов входов, которые равны *max_prior*, не изменяются.

Каждый процессор выполняет программу, которая в цикле производит следующие действия:

1) захват блока синхронизации;
2) запись данных;
3) освобождение блока синхронизации.

Для сравнения вариантов реализации разработанного устройства с другими вариантами было создано программное устройство синхронизации в разделяемой памяти, в одной из ячеек которой размещена блокирующая переменная. Такая синхронизация наиболее быстродействующая из всех возможных программных реализаций. В качестве процессоров использовались те же *soft*-процессоры *PicoBlaze* фирмы *Xilinx*. На рис. П8.9, П8.10 представлены временные диаграммы функционирования многопроцессорной системы при доступе к общему ресурсу с применением аппаратного блока синхронизации, а на рис. П8.11–П8.13 – те же действия с использованием разделяемой памяти.

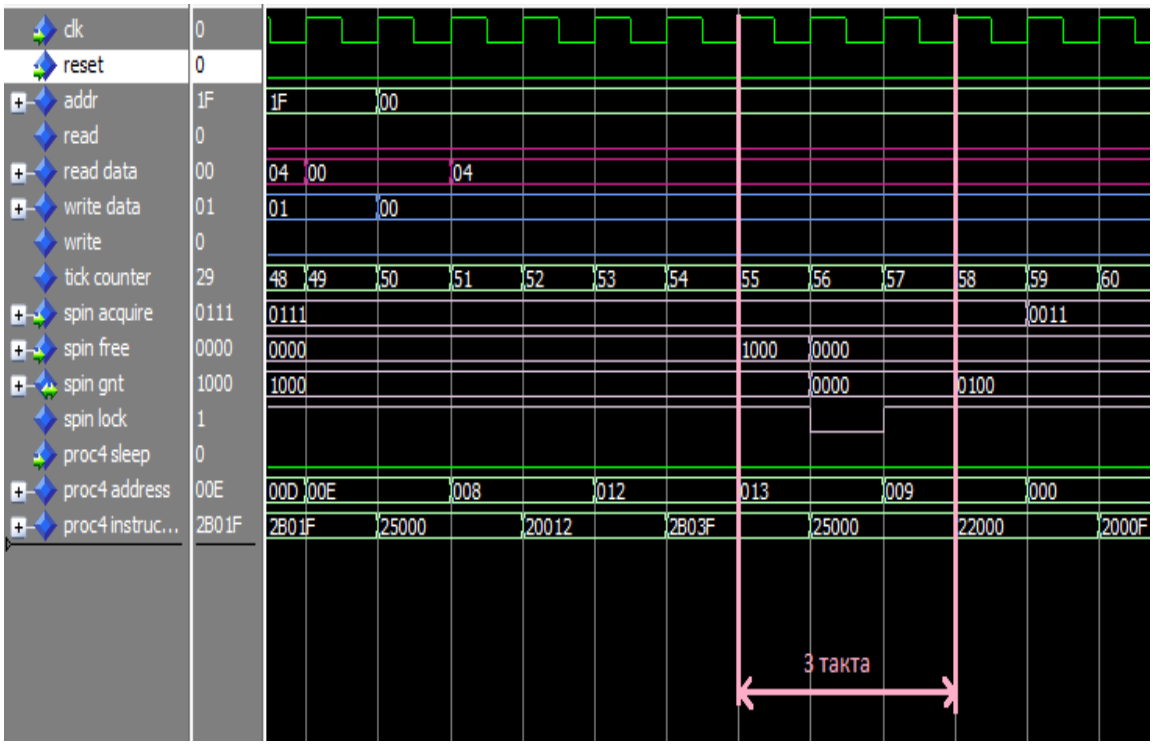


Рис. П8.9. Захват ожидающим процессором критической секции после освобождения текущим процессором

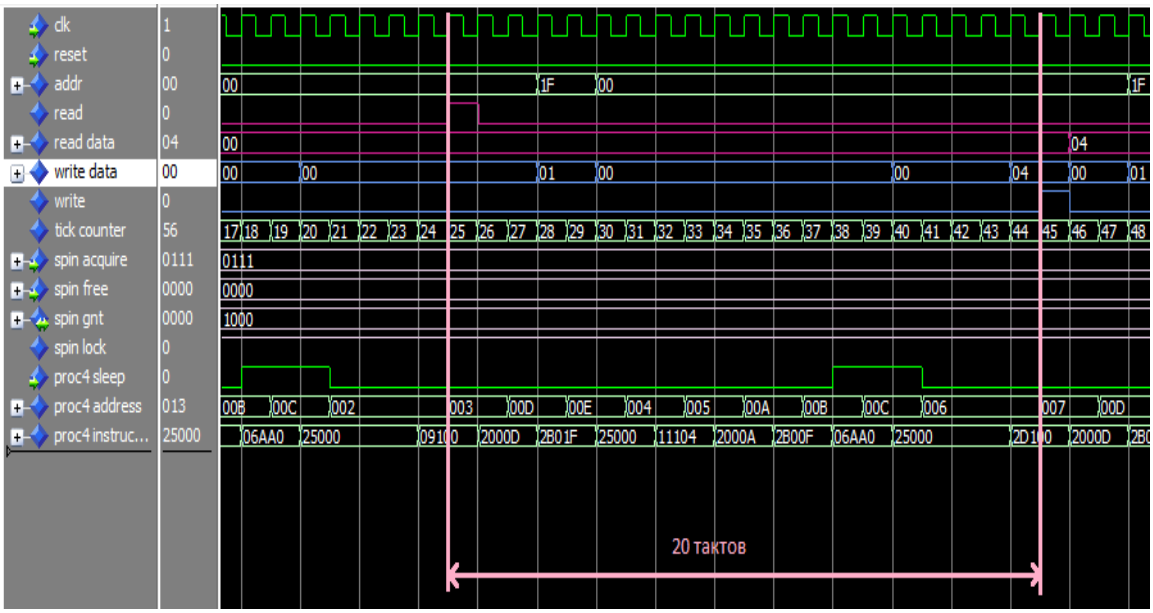


Рис. П8.10. Чтение, инкремент и запись ячейки памяти, аппаратная синхронизация

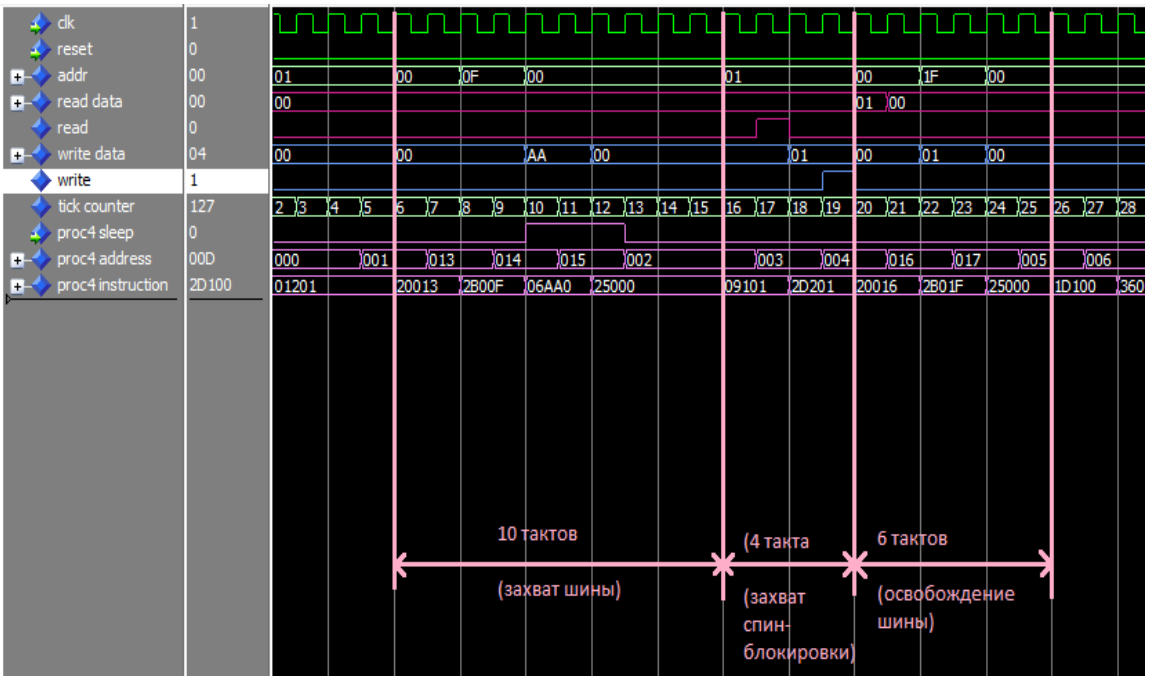


Рис. П8.11. Захват критической секции при программной синхронизации в разделяемой памяти

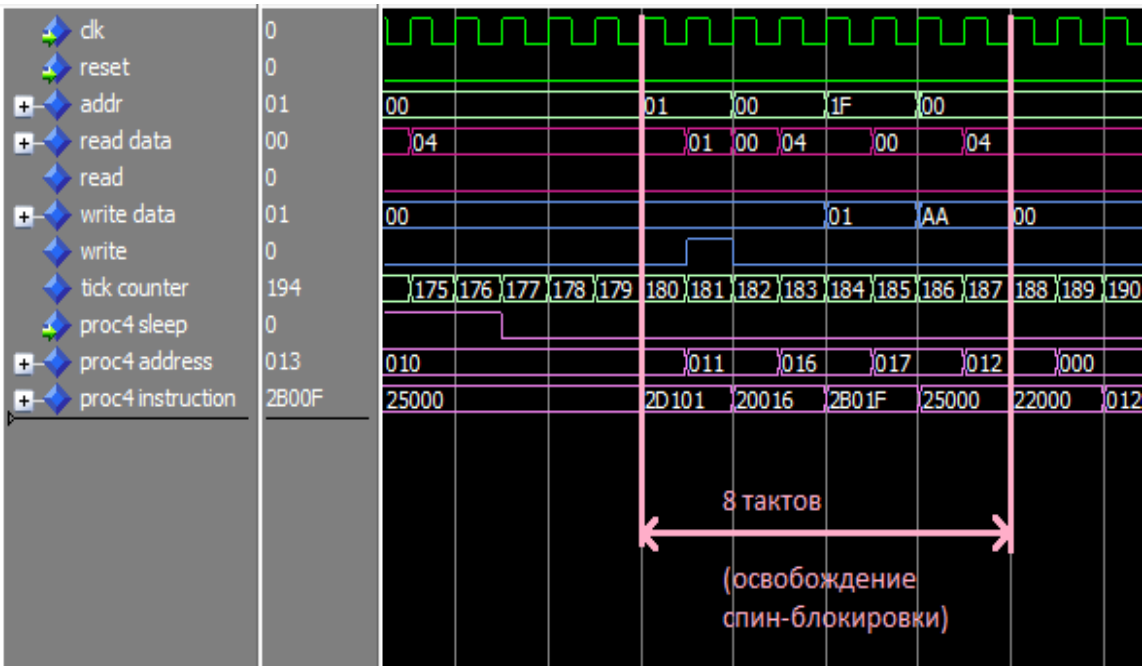


Рис. П8.12. Освобождение критической секции при программной синхронизации в разделяемой памяти

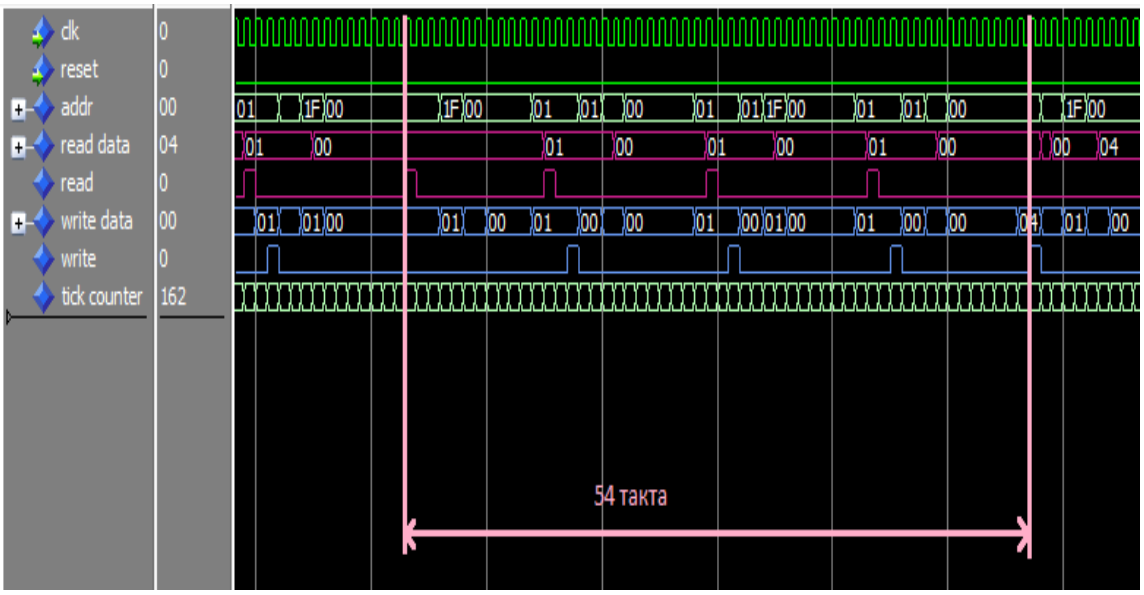


Рис. П8.13. Чтение, инкремент и запись ячейки памяти при программной синхронизации в разделяемой памяти

Устройство синхронизации реализовано на языке VHDL в составе многопроцессорной системы. В состав устройства синхронизации входит интерфейсный блок для приема сигналов запроса и сигналов состояния разделяемого ресурса, блок выдачи сигналов разрешения доступа в критическую секцию и управляющий автомат. Проведено моделирование устройства в режиме

спин-блокировки. Латентность устройства (бесконфликтная задержка доступа) составляет 2 процессорных такта, освобождение занимает 1 такт. Суммарная задержка аппаратного устройства доступа к общему ресурсу составляет 3 такта, что значительно меньше, чем задержка, создаваемая при программной реализации в разделяемой памяти, где она достигает 28 тактов. Таким образом, задержка в разделяемой памяти более чем в 9 раз больше, чем при использовании аппаратной синхронизации.

Скорость побайтного межпроцессного обмена (наиболее тяжелый режим обмена), реализуемого в многопроцессорной системе, через общую ячейку оперативной памяти в 2,5 раза ниже в разделяемой памяти, чем при использовании аппаратного блока синхронизации, о чем свидетельствуют результаты экспериментов, показанные на временных диаграммах (см. П8.10, П8.13).

Следует иметь в виду, что в данном макете память реализована внутри кристалла и работает на тактовой частоте процессора. В универсальном компьютере задержка доступа будет еще больше, так как общая (оперативная) память, в которой размещается блокирующая переменная, работает на частоте, которая значительно меньше частоты процессора, что, естественно, увеличивает время ее доступа. По оценкам зарубежных исследователей задержка доступа к общему ресурсу в разделяемой памяти ОС *Linux* составляет сотни тактов [131, 132].

При реализации доступа к общему ресурсу в пространстве ядра операционной системы, как показывают измерения задержек аналогичных механизмов доступа (мьютекса, битового семафора) в ОС *UNIX*, произведенные с помощью специализированной программы [126], она составляет около десяти тысяч тактов.

Общее увеличение производительности многопроцессорной системы, в которой применяется аппаратная синхронизация, по оценкам зарубежных исследователей может достигать 27 % [100].

Глава 9
ВЕРИФИКАЦИЯ АЛГОРИТМОВ УПРАВЛЕНИЯ ПРОЦЕССАМИ И РЕСУРСАМИ, ПРЕДСТАВЛЕННЫХ АВТОМАТНЫМИ МОДЕЛЯМИ

Верификация системы – проверка соответствия между требованиями к системе и свойствами работающей системы. Существуют три возможных метода верификации системы: тестирование, имитация (simulation) и математический метод. Первый, как показывает практика не обеспечивает контроля правильности выполнения алгоритма и выявляет только ошибки выполнения программ или аппаратуры. Однако метод не обеспечивает полного перебора всех возможных вариантов тестов, в результате чего с течением времени могут возникать сбои. Таким образом, не гарантируется надежность, а в некоторых случаях и безопасность программного или аппаратного обеспечения [69]. Второй метод труден для реализации, но хорошо подходит для отладки аппаратуры. Последний метод, называемый формальной верификацией, становится все более актуальным и в настоящее время бурно развивается.

Формальная верификация представляет собой доказательство с помощью формальных методов соответствия или несоответствия проектируемой системы ее формальному описанию. Предметом верификации выступают алгоритмы, программы, логические схемы и др. Математическими объектами, наиболее часто используемыми для моделирования и формальной верификации программ и систем, являются темпоральная (временная) логика; конечные автоматы и сети Петри.

Существует множество подходов к формальной верификации, однако наиболее часто используются следующие: проверка моделей (model checking) логический вывод (logical inference), символьное выполнение, систематический анализ алгоритмов и программ и др.

Формальную верификацию будем проводить на основе метода model checking, который подробно описан в [18]. Model checking – метод автоматической формальной верификации параллельных систем с конечным числом состояний. Он позволяет проверить, удовлетворяет ли заданная модель системы формальным спецификациям.

Обычно спецификации задаются на языке формальной логики. Для спецификации аппаратного и программного обеспечения, как правило, применяют **темпоральную логику** – специальный язык, позволяющий описывать поведение системы во времени.

Важным вопросом спецификации является полнота. Метод проверки на модели позволяет убедиться, что модель проектируемой системы соответствует заданной спецификации, однако определить, охватывает ли заданная спецификация все свойства, которым должна удовлетворять система, невозможно [18]. Основная трудность, которую приходится преодолевать в ходе проверки на модели, связана с эффектом комбинаторного взрыва в пространстве состояний. Эта проблема возникает в системах, состоящих из многих компонентов, взаимодействующих друг с другом, а также в системах, обладающих структурами данных, способных принимать большое число значений.

Другим математическим аппаратом являются **сети Петри,** которые используются для моделирования динамических дискретных систем с целью определения их свойств.

Основными свойствами сети Петри являются:

– ограниченность – число меток в любой позиции сети не может превысить некоторое значение K;

– безопасность – частный случай ограниченности, K = 1;

– сохраняемость – постоянство загрузки ресурсов;

– достижимость – возможность перехода сети из одного заданного состояния в другое;

– живость – возможность срабатывания любого перехода при функционировании моделируемого объекта.

В основе исследования перечисленных свойств лежит анализ достижимости. Методы анализа свойств сетей Петри основаны на использовании графов достижимых (покрывающих) маркировок, решении уравнения состояний сети и вычислении линейных инвариантов позиций и переходов. Основной недостаток сетей Петри заключается в сложности моделей, представляющих алгоритмы, и большом размере сети [142].

В данной монографии рассмотрим применение логики недетерминированных автоматов для верификации алгоритмов управления процессами и ресурсами. НДА являются расширением логики детерминированных конечных автоматов и обладают свойствами темпоральной (временной) логики [95].

При проектировании систем аппарат НДА может использоваться не только для формального описания (моделирования) алгоритмов управления и структурного синтеза, но и для формального контроля правильности функционирования разработанных моделей. В этом смысле применение НДА при проектировании систем управления является универсальным методом, который по сложности не превышает сложности хорошо отработанных методов теории конечных автоматов. Кроме того, логика НДА является компактным представлением автомата, что позволяет избежать комбинаторного взрыва в пространстве состояний, присущего детерминированным автоматам (см. раздел 1.1).

9.1. Верификация алгоритмов логического управления, представленных автоматной моделью, методом детерминизации НД СКУ

В процессе структурного синтеза систем логического управления на основе использования автоматных моделей исходный управляющий алгоритм, представленный на одном из начальных языков, подвергается на отдельных стадиях проектирования управляющих систем различным эквивалентным преобразованиям. Такие преобразования могут включать различные процедуры преобразования алгоритма на начальном языке как до его трансляции на стандартный язык, так и в процессе трансляции. Заключительной процедурой структурного синтеза будет процедура кодирования состояний (событий) исходного алгоритма, когда алгоритм будет формализован в виде некоторой стандартной системы канонических уравнений (СКУ), реализующих функции переходов алгоритма управления.

В процессе разнообразных эквивалентных преобразований управляющего алгоритма на отдельных стадиях структурного синтеза управляющего устройства могут быть допущены различные ошибки. В связи с этим является актуальной задача контроля правильности формального аналитического описания управляющего алгоритма, полученного на заключительном этапе структурного синтеза, когда управляющий алгоритм будет представлен в стандартной аналитической форме в виде СКУ для *O*-событий, где *O*-события представляют собой переходы в элементах памяти устройства управления.

Большое внимание при проектировании управляющих систем уделяется верификации управляющих алгоритмов, представленных также в виде программ, когда для их верификации используются автоматные модели [18]. Контроль правильности аппаратного программного обеспечения особенно важен, когда он используется в системах промышленной автоматики и в информационных системах управления, работающих в реальном масштабе времени [69].

Прежде чем приступить к рассмотрению методов верификации управляющих алгоритмов, представленных в аналитической форме, коротко рассмотрим основные этапы их формального описания, в процессе которых могут возникнуть ошибки в их аналитическом представлении на основе использования автоматных моделей.

В том случае, если на начальном этапе структурного синтеза в качестве одного из языков формального представления управляющего алгоритма используется язык логики недетерминированных автоматов (НДА), можно выделить три основных этапа эквивалентных преобразований алгоритма, когда каждому из них будет соответствовать свой вид формального представления алгоритма управления.

На первом этапе аналитическое описание управляющего алгоритма можно представить в виде системы рекуррентных канонических уравнений (НД СКУ), формализующих все реализуемые в алгоритме управления частные события [21, 104]. В этом случае функции переходов в алгоритме управления представляют в терминах частных событий, а система уравнений НД СКУ для модели автомата Мура будет иметь вид:

$$S_j^{\left[Y_j\right]}(t+1) = f_j(S_0, S_1, ..., S_\alpha, ..., S_\beta, ..., S_\gamma, ..., S_N, x_0, x_1, ..., x_L)(t),$$

$$j = \overline{0,N},\ S_0(0) = 1, \qquad (9.1)$$

где f_j – булева функция от входных сигналов и частных событий, реализуемых в алгоритме.

Под символом S_j в данном случае понимается сокращенное обозначение события, реализуемого в управляющем алгоритме, в соответствии с которым, в зависимости от уровня детализации алгоритма и его назначения, могут выполняться в управляемом устройстве различные действия. К числу таких действий, например, в информационной системе, могут относиться выходные

управляющие сигналы или группы сигналов, выполняемые микрооперации, операторы, подпрограммы и т.д., а также различные действия в системах промышленной автоматики, такие, например, как срабатывания датчиков, включения или выключения исполнительных механизмов и т.д. Инициируют такие действия частные выходные сигналы, совокупностью которых $[Y_j]$ отмечается событие S_j.

Система НД СКУ (9.1) может быть использована на втором этапе структурного синтеза, когда выполняется ее детерминизация, в соответствии с которой определяют всевозможные сочетания частных событий, одновременное существование которых в управляющем алгоритме возможно. Этому сочетанию частных событий ставится в соответствие внутреннее состояние детерминированного автомата (ДА), являющегося моделью устройства управления. Процедура детерминизации может быть использована также при решении вопросов распараллеливания управляющих алгоритмов, когда все частные события, реализуемые в алгоритме, разбивают на группы несовместимых частных событий, каждая из которых выполняется в одной из параллельных ветвей алгоритма [104].

В результате детерминизации системы НД СКУ (9.1) для частных событий получают детерминированную систему рекуррентных канонических уравнений, описывающих функции переходов алгоритма управления в терминах внутренних состояний управляющего устройства. Обозначим такую систему уравнений как СКУ ДА для *a*-событий, которая будет иметь вид

$$a_k^{[Y_k]}(t+1) = f_k(a_0, a_1, ..., a_k, ..., a_n, x_0, x_1, ..., x_L)(t),$$
$$k = \overline{0,n},\ a_0(0) = 1, \qquad (9.2)$$

где f_k – булева функция от входных сигналов и полных *a*-событий (состояний), реализуемых в алгоритме; $[Y_k]$ – подмножество выходных сигналов, отмечающих состояние a_k; a_k – внутреннее состояние управляющего устройства, называемое полным *a*-событием, так как оно представляется в виде кода из частных совместимых событий, т.е. событий, которые могут одновременно существовать в алгоритме. Например:

$$a_k^{[Y_k]} = S_\alpha^{Y_\alpha}\ S_\beta^{Y_\beta}\ S_\gamma^{Y_\gamma},\ [Y_k] = \left[Y_\alpha, Y_\beta, Y_\gamma\right].$$

На третьем этапе структурного синтеза системы управления ее формальное аналитическое описание будет представлено в виде системы рекуррентных канонических уравнений НД СКУ для Q-событий, которая будет недетерминированной относительно Q-событий. Такая система уравнений имеет следующий вид:

$$Q_m(t+1) = f_m\left(Q_0, Q_1, ..., Q_R,\ x_0, x_1, ..., x_L\right),\ m = \overline{1, R}, \qquad (9.3)$$

где f_m – булева функция от входных сигналов и Q-событий, определяющих функции возбуждения элементов памяти устройства управления.

Система уравнений (9.3) формируется в результате кодирования внутренних состояний устройства управления состояниями элементов памяти. Любое состояние a_k из (9.2) представляется кодом $a_k = Q_0\,\widetilde{Q}_1 \cdots \widetilde{Q}_m \cdots \widetilde{Q}_R$, где символ ~ означает, что переменная Q_m может быть взята как с отрицанием, так и без отрицания.

9.1.1. Верификация алгоритмов логического управления, представленных автоматной моделью, полученной без использования логики НДА

Для этого варианта верификации управляющий алгоритм представлен в стандартной аналитической форме в виде НД СКУ для Q-событий (9.3). Требуется выполнить контроль правильности представления системы (9.3) и ее соответствия исходной системе уравнений вида (9.2). Для решения этой задачи воспользуемся тем обстоятельством, что операции детерминизации и кодирования управляющего алгоритма являются обратными друг другу. В связи с этим процедура контроля может быть организована на основе выполнения операции детерминизации НД СКУ для Q-событий с последующей минимизацией системы уравнений для исходных событий (состояний), представляющих управляющий алгоритм [107]. Если после детерминизации НД СКУ для Q-событий получим СКУ, которая будет соответствовать исходной СКУ до ее кодирования, то это будет свидетельствовать о правильности проверяемой СКУ.

Для выполнения операции детерминизации НД СКУ для Q-событий с целью ее контроля необходимо алгоритм детерминизации, рассмотренный в [104], дополнить пунктами, вытекающи-

ми из специфики формирования внутренних состояний автомата, представляемых сочетаниями (конъюнкцией) Q-событий и возможным отсутствием полноты входных сигналов, вызывающих переходы в Q-событиях. Это связано с тем, что в кодах состояний автомата некоторые переменные (Q-события) могут быть взяты с отрицанием. Таким образом, в алгоритм детерминизации НДА, рассмотренный в гл. 2, вводят необходимые дополнения и разъяснения, которые представлены в гл. 4.

Будем рассматривать на простом примере методику детерминизации СКУ для Q-событий, используемую для верификации таких СКУ.

Пусть функции переходов исходного управляющего автомата представлены следующей системой рекуррентных канонических уравнений НДА Мура:

$$\begin{aligned}
a_0(t+1) &= x_0,\\
a_1(t+1) &= a_0\overline{x_1} \vee a_1 x_2 \overline{x_3},\\
a_2(t+1) &= a_4,\\
a_3(t+1) &= a_0 x_1 \overline{x_2},\\
a_4(t+1) &= a_0 x_1 x_2 \vee a_1 x_2 x_3 \vee a_2 x_4,\\
a_5(t+1) &= a_3 \vee a_1\overline{x_2} \vee a_2\overline{x_4},
\end{aligned} \tag{9.4}$$

где в правой части уравнений (9.4) и им подобным время t для простоты будет опущено.

Выполняя произвольное кодирование состояний исходного автомата, получим следующие кодовые группы:

$$\begin{aligned}
a_0 &= \overline{Q_1} Q_2 Q_3, \qquad a_3 = Q_1\overline{Q_2}\,\overline{Q_3},\\
a_1 &= \overline{Q_1} Q_2 \overline{Q_3}, \qquad a_4 = \overline{Q_1}\,\overline{Q_2}\,\overline{Q_3},\\
a_2 &= Q_1\overline{Q_2} Q_3, \qquad a_5 = \overline{Q_1}\,\overline{Q_2} Q_3.
\end{aligned}$$

Учитывая принятое кодирование, получим следующую СКУ для Q-событий, представляющую функции возбуждения элементов памяти (9.5):

$$\begin{aligned}
Q_1(t+1) &= (a_2 \vee a_3)(t+1) = a_4 \vee a_0 x_1 \overline{x_2},\\
Q_2(t+1) &= (a_0 \vee a_1)(t+1) = x_0 \vee a_0\overline{x_1} \vee a_1 x_2 \overline{x_3},\\
Q_3(t+1) &= (a_0 \vee a_2 \vee a_5)(t+1) = x_0 \vee a_3 \vee a_4 \vee a_1\overline{x_2} \vee a_2\overline{x_4}.
\end{aligned} \tag{9.5}$$

Выполняя в уравнениях (9.5) подстановку кодов состояний исходного автомата, получим систему уравнений (9.6), используемую для структурного синтеза:

$$
\begin{aligned}
Q_1(t+1) &= \overline{Q_1}\,\overline{Q_2}\,\overline{Q_3} \vee Q_2 Q_3 x_1 \overline{x_2}, \\
Q_2(t+1) &= x_0 \vee Q_2 Q_3 \overline{x_1} \vee Q_2 \overline{Q_3} x_2 \overline{x_3}, \\
Q_3(t+1) &= x_0 \vee \overline{Q_3}\,\overline{x_2} \vee Q_1 \overline{x_4} \vee \overline{Q_2}\,\overline{Q_3}.
\end{aligned}
\qquad (9.6)
$$

Для контроля правильности системы уравнений (9.6) выполняем ее детерминизацию. Для этого построим по уравнениям (9.6) вспомогательную прямую таблицу переходов для Q-событий (см. табл. 5.4, гл. 5).

На основании алгоритма детерминизации, представленного в гл. 2, используя вспомогательную табл. 5.4 и учитывая требования к формированию полных событий (состояний) автомата и входных сигналов на переходах, получим прямую таблицу переходов детерминированного автомата (см. табл. 5.4, гл. 5).

По табл. 5.4 строится СКУ для Q-событий, которая должна полностью соответствовать (до обозначений) исходной СКУ (9.4), представляющей функции переходов исходного автомата. Если такого соответствия нет, то это означает, что система СКУ для Q-событий, на основе которой должна строиться структура устройства управления, имеет ошибки и должна быть скорректирована.

9.1.2. Верификация алгоритмов логического управления, представленных автоматной моделью, полученной с использованием логики НДА

Рассмотрим методику верификации управляющего алгоритма, который задан системой СКУ для a-событий типа (9.2), и ее соответствия формальному представлению исходной управляющей системы на языке НД СКУ для всех частных событий, реализуемых в алгоритме. При этом такая методика верификации должна обеспечивать ответ на вопрос: обладает ли управляющая система некоторыми свойствами, заложенными в управляющий алгоритм путем формализации отдельных частных событий, определяющих эти свойства? Для иллюстрации рассматриваемой методики воспользуемся результатами построения управляющего

алгоритма по граф-схеме НДА, рассмотренного в гл. 1 (см. (1.8)), где исходный управляющий алгоритм представлен системой НД СКУ для частных событий.

Существует несколько способов построения структурной схемы управляющего устройства, для которого алгоритм управления представлен системой уравнений типа (1.8). Во всех этих способах обязательно осуществляется либо унитарное кодирование отдельных частных событий, либо максимальное кодирование совокупностей частных событий (*a*-событий), реализуемых в управляющем алгоритме совместно. В данном разделе, как было отмечено выше, будем рассматривать верификацию системы управления, построенную с использованием кодирования *a*-событий, т.е. когда функции переходов в алгоритме управления представлены системой СКУ ДА для *a*-событий, которая формируется в результате детерминизации исходной системы НД СКУ. Для нашего примера система СКУ ДА представлена следующими уравнениями ((2.2), гл. 2), где внутренние состояния *a*-события представлены следующими сочетаниями частных совместимых событий:

$$
\begin{aligned}
&a_0 = S_0, && a_7 = S_3 S_4,\\
&a_1 = S_1 S_2, && a_8 = S_2 S_3,\\
&a_2 = S_1 S_4, && a_9 = S_4,\\
&a_3 = S_2 S_3 S_4, && a_{10} = S_3 S_4 S_5,\\
&a_4 = S_1 S_3 S_5, && a_{11} = S_3 S_5,\\
&a_5 = S_3, && a_{12} = S_1 S_3 S_4.\\
&a_6 = S_1 S_3, &&
\end{aligned}
\qquad (9.7)
$$

Учитывая значение кодов внутренних состояний *a*-событий, выраженных через сочетания частных событий (9.7), и делая замену переменных с учетом соотношений (2.2) и (9.7), получим после минимизации СКУ, определяющую все частные события, реализуемые в управляющем алгоритме и являющиеся результатом верификации управляющего алгоритма, представленного СКУ (2.2):

$$
\begin{aligned}
&S_1(t+1) = (a_1 \vee a_2 \vee a_4 \vee a_6 \vee a_{12})(t+1) = S_0 x_1 \vee S_1 (S_2 \vee S_3 \vee S_4)\overline{x_2},\\
&S_2(t+1) = (a_1 \vee a_3 \vee a_8)(t+1) = S_0 x_1 \vee S_1 (S_2 \vee S_3 \vee S_4)\overline{x_1} x_2,\\
&S_3(t+1) = (a_3 \vee a_4 \vee a_5 \vee a_6 \vee a_7 \vee a_8 \vee a_{10} \vee a_{11})(t+1) =\\
&= S_1 (S_3 \vee S_4) x_2 \vee S_2 (S_1 \vee S_3) x_1 \vee S_3 \overline{x_3} \vee S_4 x_4 \vee S_5 S_3 x_4,
\end{aligned}
\qquad (9.8)
$$

$$S_4(t+1) = (a_2 \vee a_3 \vee a_7 \vee a_9 \vee a_{10} \vee a_{12})(t+1) =$$
$$= S_2(S_1 \vee S_3)\overline{x_1} \vee S_3 x_3 \vee S_4 \overline{x_4} \vee S_5 S_3 \overline{x_4},$$
$$S_5(t+1) = (a_4 \vee a_{10} \vee a_{11})(t+1) = S_2(S_1 \vee S_3) x_1 \overline{x_2}.$$

При анализе результатов верификации алгоритма управления необходимо учитывать некоторые особенности исходной системы уравнений НД СКУ для *S*-событий (1.8) и такой же системы уравнений, но полученной в результате верификации (9.8).

Исходная система НД СКУ для *S*-событий отличается тем, что для каждого частного события, представимого в системе, для любого входного сигнала имеет место только одно-единственное непосредственно предшествующее частное событие.

Система НД СКУ для *S*-событий, полученная в результате верификации, хотя может быть логически эквивалентна исходной системе (9.7), но может иметь отличие в следующем: при формировании отдельных частных событий, по результатам детерминизации, они могут зародиться (или сохраниться) при определенном сочетании частных входных сигналов только совместно с другими частными событиями (для нашего примера это события S_1, S_2, S_5); некоторые частные события могут зародиться (или сохраниться) или только индивидуально (для нашего примера это события S_3 и S_4), или совместно с другими частными событиями. В результате в системе уравнений НД СКУ для *S*-событий (9.8) для отдельных частных событий S_j их условия зарождения (или сохранения) могут быть представлены совокупностью (конъюнкцией) некоторых событий. Однако, если результат пересечения таких совокупностей непосредственно предшествующих событий определит одно частное событие, то оно и будет непосредственно предшествующим событием для события S_j, представленного в исходной системе НД СКУ для *S*-события.

Таким образом, если при анализе результатов верификации алгоритма логического управления, представленного автоматной моделью в виде СКУ для *a*-событий, полученная НД СКУ для *S*-событий логически эквивалентна исходной модели с учетом рассмотренных замечаний, то ошибки в автоматной модели СКУ для *a*-событий отсутствуют. Для нашего примера, исходя из изложенного, анализ систем уравнений (1.8) и (9.8) свидетельствует о том, что ошибки в исходной автоматной модели управляющего алгоритма отсутствуют.

Такую верификацию, когда анализируются все уравнения для частных событий, реализуемых в алгоритме управления, можно назвать полной. В том случае, если должна быть выполнена

частичная верификация, направленная на контроль: обладает ли система управления некоторым свойством, формируется только одно уравнение, соответствующее частному событию, которое определяет это свойство.

В заключение необходимо отметить, что если структурный синтез системы логического управления основывается на реализации управляющего алгоритма, представленного НД СКУ для *S*-событий, с использованием унитарного кодирования частных событий, то задача верификации управляющего алгоритма на наличие некоторого свойства, определяемого совокупностью совместно реализуемых в алгоритме частных событий, также должна решаться на основе результатов детерминизации исходного управляющего алгоритма.

9.2. Верификация алгоритмов управления, представленных на языке НДА, с использованием инструментальных средств

9.2.1. Программная система верификации алгоритмов управления, представленных на языке НДА

В связи со сложностью многих алгоритмов управления процессами и ресурсами процедура верификации полученного формального описания является крайне важной. Существует множество систем верификации, но для любой из них требуется сначала преобразовать формальное описание с описания на входном языке верификатора.

Как было указано ранее, после построения НД СКУ необходимо проверить ее на отсутствие в ней недостижимых событий и произвести верификацию управляющего автомата. Но процесс ручного составления прямой таблицы переходов (см. подразделы 4.3.2, 4.4.3) для достаточно большой СКУ является длительной задачей, также всегда есть вероятность допустить ошибки при занесении очередного события в таблицу. Поэтому требуется автоматизация процедуры ввода и преобразования алгоритмов управления для проведения моделирования и верификации.

Разработанная в Пензенском государственном университете программная система верификации алгоритмов управления (ПСВАУ) [144] позволяет выполнять:

– преобразование исходной системы канонических уравнений в модель в виде графа;

– пошаговое моделирование с возможностью ручного задания входных сигналов;

– автоматическое моделирование с возможностью задания входных сигналов в виде таблицы и генерацией отчета с пошаговым описанием состояния системы;

– генерацию описания управляющего автомата на языки SMV и VHDL для верификации в других инструментальных системах;

– генерацию описания управляющего автомата на язык C++ и C# для представления его прикладной программой.

Главное окно программы и вкладка ввода формального описания алгоритма в виде НД СКУ представлены на рис. 9.1.

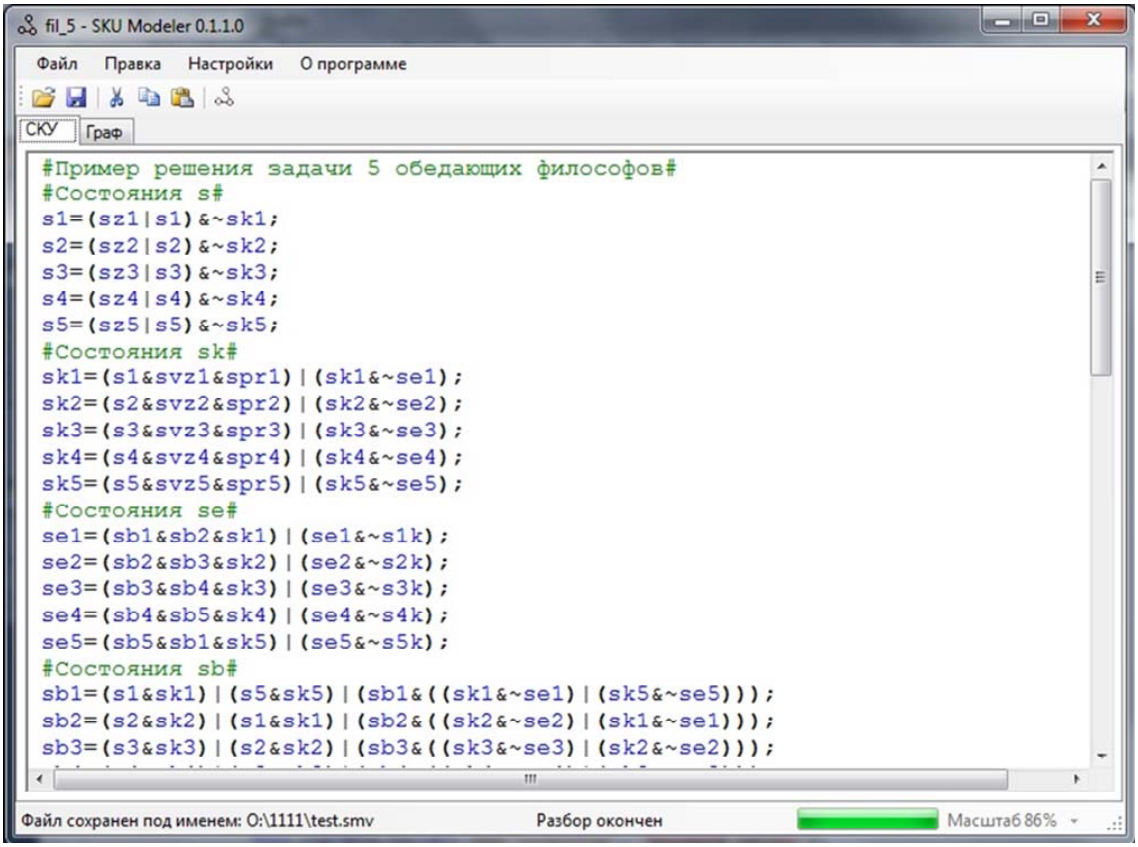


Рис. 9.1. Пример ввода исходных данных в систему ПСВАУ в виде НД СКУ

ПСВАУ позволяет преобразовать входную НД СКУ в модель, представленную в виде графа. На графе квадратными символами отображаются входные сигналы, круглыми символами – состояния автомата, а цветными линиями – связи между сигналами и состояниями. Также при моделировании сигналы и состояния могут закрашиваться цветом, что говорит либо об активности сигнала, либо о том, что автомат находится в данном состоянии.

После генерации модели возможно либо пошаговое моделирование, при котором пользователь сам подает команду на переход к следующему шагу и задает входные воздействия, либо автоматическое, при котором переход к следующему шагу происходит автоматически через заранее определенные пользователем промежутки времени, а входные воздействия берутся из таблицы сигналов, также заранее заполненной пользователем.

Кроме моделирования непосредственно в системе, возможна генерация описания на входном языке верификатора SMV, которое в дальнейшем может быть использовано для проверки достижимости всех состояний автомата. Также возможны генерация VHDL-представления автомата, которое может быть использовано непосредственно при разработке аппаратной реализации управляющего алгоритма, и описание автомата на языках C++ и C# для программной реализации управляющего алгоритма.

Рассмотрим пример моделирования управляющего автомата, реализующего функцию обеспечения приоритетного взаимоисключающего доступа к критическому ресурсу для четырех процессов. По представленным в гл. 6 (подраздел 6.4.2) НД СКУ был сгенерирован граф (рис. 9.2), на котором представлены все состояния управляющего автомата, входные и выходные сигналы, а так же связи между ними.

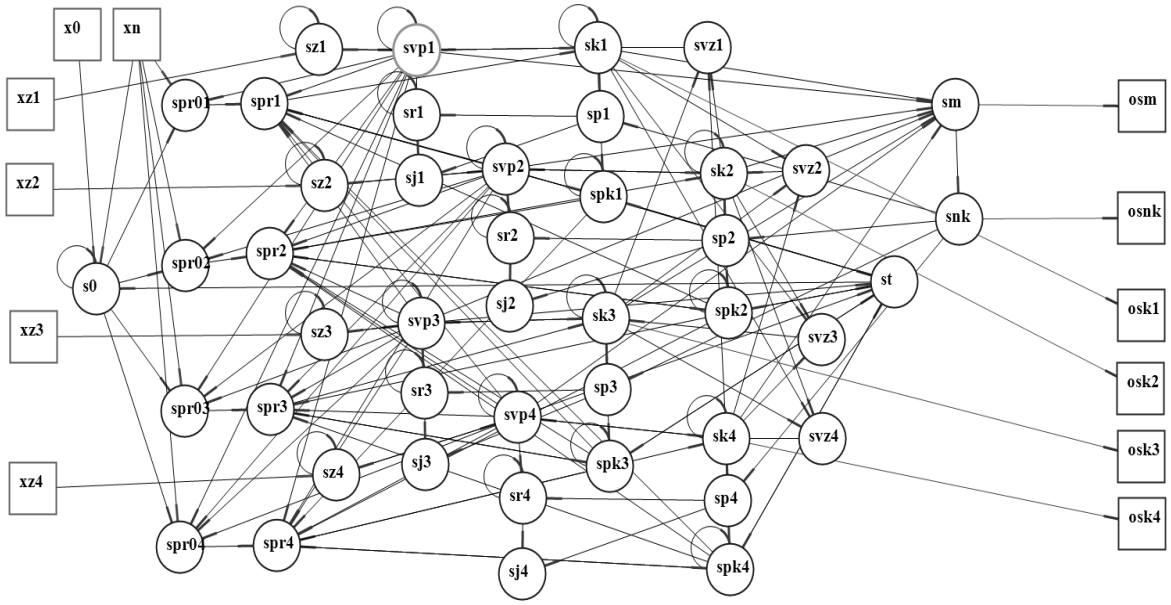


Рис. 9.2. Граф СНДА алгоритма управления ресурсом
в программе верификации

Далее была заполнена таблица входных сигналов (табл. 9.1), по которой проведено моделирование. При запуске моделирования значения из таблицы считываются и подставляются в модель, затем происходят пересчет модели и отображения текущего такта моделирования на графе. Когда моделирование доходит до конца таблицы, моделирование продолжается с игнорированием ее значений, это позволяет не заполнять таблицу полностью.

Таблица 9.1

Входной сигнал/ Шаг моделирования	1	2	3	4	5	6	7	8	9	10	11
x_n	0	0	1	0	0	0	0	0	0	0	0
x_{z1}	1	0	0	0	0	0	1	0	0	0	0
x_{z2}	1	0	0	0	0	0	0	0	1	0	0
x_{z3}	1	0	0	0	0	0	0	0	0	0	1
x_{z4}	1	0	0	0	0	0	0	0	0	0	0
x_0	1	0	0	0	0	0	0	0	0	0	0

В результате моделирования получается отчет, в котором в табличной форме представлена информация о состоянии модели на каждом шаге моделирования (табл. 9.2). По ней можно сделать вывод, что доступ к критическому ресурсу происходит в соответствии с требованиями, предъявленными к управляющему автомату (события $S^1_k - S^4_k$ происходят в соответствии с заданным приоритетом и последовательно, о чем свидетельствуют соответствующие выходные сигналы $O^1_{sk} - O^4_{sk}$). Таким образом доказывается правильность составленного описания управляющего автомата и выполнения предъявленных к нему требований.

Таблица 9.2

Шаг моделирования	*xn*	*xz*1	*xz*2	*xz*3	*xz*4	*s*0	*x*0	*osm*	*osnk*	*osk*1	*osk*2	*osk*3	*osk*4
1	2	3	4	5	6	7	8	9	10	11	12	13	14
Шаг 0	0	1	1	1	1	0	1	0	0	0	0	0	0
Шаг 1	0	0	0	0	0	1	0	0	0	0	0	0	0
Шаг 2	0	0	0	0	0	1	0	0	0	0	0	0	0
Шаг 3	1	0	0	0	0	1	0	0	0	0	0	0	0
Шаг 4	0	0	0	0	0	0	0	0	0	0	0	0	0
Шаг 5	0	0	0	0	0	0	0	0	0	0	0	0	0
Шаг 6	0	0	0	0	0	0	0	0	0	1	0	0	0
Шаг 7	0	0	0	0	0	0	0	1	0	1	0	0	0
Шаг 8	0	0	0	0	0	0	0	0	1	1	0	0	0

Окончание табл. 9.2

1	2	3	4	5	6	7	8	9	10	11	12	13	14
Шаг 9	0	0	0	0	0	0	0	0	0	1	0	0	0
Шаг 10	0	0	0	0	0	0	0	0	0	0	0	0	0
Шаг 11	0	0	0	0	0	0	0	0	0	0	0	0	0
Шаг 12	0	0	0	0	0	0	0	0	0	0	1	0	0
Шаг 13	0	0	0	0	0	0	0	1	0	0	1	0	0
Шаг 14	0	0	0	0	0	0	0	0	1	0	1	0	0
Шаг 15	0	0	0	0	0	0	0	0	0	0	1	0	0
Шаг 16	0	0	0	0	0	0	0	0	0	0	0	0	0
Шаг 17	0	0	0	0	0	0	0	0	0	0	0	0	0
Шаг 18	0	0	0	0	0	0	0	0	0	0	0	1	0
Шаг 19	0	0	0	0	0	0	0	1	0	0	0	1	0
Шаг 20	0	0	0	0	0	0	0	0	1	0	0	1	0
Шаг 21	0	0	0	0	0	0	0	0	0	0	0	1	0
Шаг 22	0	0	0	0	0	0	0	0	0	0	0	0	0
Шаг 23	0	0	0	0	0	0	0	0	0	0	0	0	0
Шаг 24	0	0	0	0	0	0	0	0	0	0	0	0	1
Шаг 25	0	0	0	0	0	0	0	1	0	0	0	0	1
Шаг 26	0	0	0	0	0	0	0	0	1	0	0	0	1
Шаг 27	0	0	0	0	0	0	0	0	0	0	0	0	1
Шаг 28	0	0	0	0	0	0	0	0	0	0	0	0	0
Шаг 29	0	0	0	0	0	1	0	0	0	0	0	0	0

9.2.2. Верификация алгоритмов управления, заданных недетерминированными автоматами, в системе SMV

Инструментальное средство SMV (Symbolic Model Verifier) [116] поддерживает верификацию параллельных процессов, которые взаимодействуют через общие переменные. Процессы в SMV могут исполняться синхронным или асинхронным способом. Спецификация на входном языке SMV состоит из описаний процессов, описаний (локальных и глобальных) переменных, описаний формул и спецификаций формул, которые должны быть верифицированы.

Ниже приводится методика верификации систем, построенных на основе НДА, с использованием метода *Model Checking* в системе SMV. Основные шаги методики представлены на рис. 9.3.

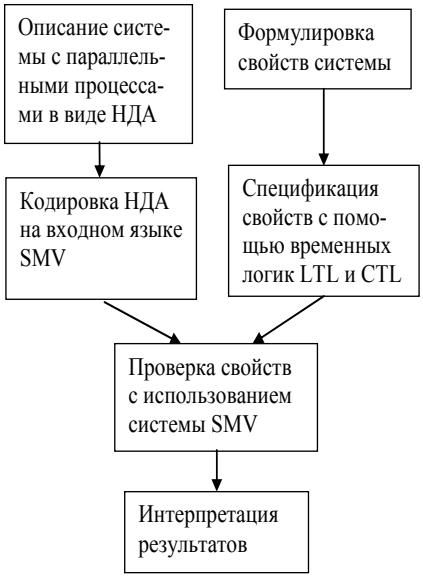


Рис. 9.3. Методика верификации систем на основе НДА

Правила описания СКУ НДА на языке SMV просты. НДА оформляется в виде одного модуля SMV. Входные переменные, а также события НДА, зависящие от внешнего окружения, представляются как параметры модуля. Считается, что они могут принимать произвольные значения. Каждому событию в НДА ставится в соответствие булева переменная. Начальные единичные значения устанавливаются для тех переменных, которые соответствуют начальным состояниям. Начальные значения остальных булевых переменных устанавливаются в ноль.

В основном режиме (левая ветвь методики) уравнения СКУ один к одному транслируются в операторы присваивания SMV, причем для представления события, стоящего в левой части уравнения, используется конструкция *next* в левой части оператора присваивания. Правая часть уравнения СКУ определяется как логическое выражение языка SMV. Операторы присваивания, соответствующие СКУ, объединяются в составной оператор, который предваряется конструкцией *default*. Операторы присваивания, стоящие в этой конструкции, сбрасывают все переменные, соответствующие событиям НДА, в ноль. Смысл *default*-оператора в данном контексте означает, что все события, которые не устанавливаются в следующем такте работы системы, удаляются из совокупного состояния. Для представления НДА, функционирующего в режиме B, для каждого события s_i НДА вводятся два условия: условие установки (set_s_i) и условие сброса ($reset\_s_i$) события. Условия установки события – эквиваленты правой части соответствующего уравнения СКУ. Условия сброса определяются поиском сбрасыва-

емого события в правых частях в системе уравнений СКУ и определения условий установки событий, стоящих в левых частях найденных уравнений. Условие сброса формируется путем дизъюнкции этих условий установки. Для каждого события s_i формируется оператор присваивания SMV следующего вида: $next(s_i) := set\_s_i \mid (\sim set\_s_i \ \& \ \sim reset\_s_i \ \& \ s_i)$, вычисляющий статус события s_i в следующий момент времени.

Свойства системы первоначально формулируются на естественном языке (правая ветвь методики). Все свойства можно разделить на общие и частные. В отношении систем с параллельными процессами обычно выделяют следующие общие свойства: безопасности, живости и справедливости (*fairness*). В системе SMV свойства формулируются с помощью линейной временной логики *LTL* или логики дерева вычислений *CTL* [136]. В логике *LTL* определены линейные временные операторы *X* (в следующий момент времени), *F* (возможно в будущем), *G* (всегда) и *U* (пока не). В логике *CTL* в дополнение к этому используется квантификация путей. При этом кванторы *E* и *A* предваряют линейные временные операторы (например, *AG*, *EF*, *AX* и т.п.). Атомарные высказывания формулируются на основе событий НДА, причем существуют два элементарных вида высказываний: событие НДА входит в совокупное состояние и событие не входит в совокупное состояние.

Ниже в качестве примера рассматривается простая система двух взаимодействующих параллельных процессов, обращающихся к общему ресурсу (задача взаимного исключения). Данная задача хорошо исследована и освещена в литературе. Например, в работе [125] рассмотрены и верифицированы пять алгоритмов взаимного исключения для программной реализации. Задача взаимного исключения для двух конкурирующих процессов с использованием НДА представлена в подразделе 6.4.1. НДА состоит из трех частей: двух ветвей, представляющих процессы, и общей части кода, выполняемой в режиме критической секции. Ниже представлена система СКУ НДА, описывающая упомянутую систему параллельных процессов:

для 1-й ветви	для 2-й ветви	для последовательной компоненты
$S_1(t+1)=(S_{1,z} \vee S_1) \& \sim S^1{}_k$	$S_2(t+1)=(S_{2,z} \vee S_2) \& \sim S^2{}_k$	$S_m(t+1)=S_k{}^1 \& S_1 \vee$
$S_k{}^1(t+1)=S_1 \& \sim S^2{}_k \vee S_k{}^1 \& \sim S_{nk}$	$S_k{}^2(t+1)=S_2 \& \sim S^1{}_k \vee S_k{}^2 \& \sim S_{nk}$	$\vee \ S_k{}^2 \& S_2$
$S_p{}^1(t+1)=S_{nk} \& S_k{}^1$	$S_p{}^2(t+1)=S_{nk} \& S_k{}^2$	$S_{nk}(t+1)=S_m$
$S_r{}^1(t+1)=(S_1 \vee S_r{}^1) \& \sim S_p{}^1$	$S_r{}^2(t+1)=(S_2 \vee S_r{}^2) \& \sim S_p{}^2$	
$S_{1,n}(t+1)=S_r{}^1 \& S_p{}^1$	$S_{2,n}(t+1)=S_r{}^2 \& S_p{}^2$	

Смысл используемых в СКУ событий следующий: $S_{1,z}$ и $S_{2,z}$ – поступление заявок первого и второго процессов на обслуживание; S_1 и S_2 – прием заявок первого и второго процессов на обслуживание; S_k^1 и S_k^2 – входы первого и второго процессов в свои критические участки; S_p^1 и S_p^2 – выход первого и второго процессов из критического участка после окончания процедуры обращения к разделяемым данным; S_m и S_{nk} – начало и окончание процедуры обращения к разделяемым данным, соответственно.

Ниже представлена кодировка НДА на входном языке SMV.

module main(*s1z*,*s2z*)
{INPUT s1z,*s2z* : *boolean*;
VAR s1,*sk1*,*sp1*,*sr1*,*s1n*, *s2*,*sk2*,*sp2*,*sr2*,*s2n*, *sm*,*snk* : *boolean*;
ASSIGN
init(*s1z*):=*0*; *init*(*s2z*):=*0*; *init*(*s1*):=*0*; *init*(*sk1*):=*0*; *init*(*sp1*):=*0*; *init*(*sr1*):=*0*; *init*(*s1n*):=*0*; *init*(*s2*):=*0*; *init*(*sk2*):=*0*; *init*(*sp2*):=*0*; *init*(*sr2*):=*0*; *init*(*s2n*):=*0*; *init*(*sm*):=*0*; *init*(*snk*):=*0*;
default {next(*s1*):=*0*; *next*(*sk1*):=*0*; *next*(*sp1*):=*0*; *next*(*sr1*):=*0*; *next*(*s1n*):=*0*; *next*(*s2*):=*0*;*next*(*sk2*):=*0*; *next*(*sp2*):=*0*; *next*(*sr2*):=*0*; *next*(*s2n*):=*0*; *next*(*sm*):=*0*; *next*(*snk*):=*0*; *}*
in { next(*s1*):= (*s1z* | *s1*)*&~sk1*; *next*(*sk1*):= *s1&~sk2* | *sk1&~snk*; *next*(*sp1*):= *snk&sk1*; *next*(*sr1*):= (*s1* | *sr1*)*&~sp1*; *next*(*s1n*):= *sr1&sp1*; *next*(*s2*):= (*s2z* | *s2*)*&~sk2*; *next*(*sk2*):= *s2&~sk1&~s1* | *sk2&~snk*; *next*(*sp2*):= *snk&sk2*; *next*(*sr2*):= (*s2* | *sr2*)*&~sp2*; *next*(*s2n*):= *sr2&sp2*; *next*(*sm*):= *sk1&s1* | *sk2&s2*; *next*(*snk*):= *sm*;*}*

Определим свойства системы параллельных процессов, описанной выше. Свойство *безопасности* выражается следующим образом: «Никогда два процесса не могут быть в своих критических областях». Данное свойство выражается с помощью следующих формул временной логики *CTL*: *AG* ~(*sk1&sk2*) и *EF* (*sk1&sk2*). При удовлетворении данного свойства первая формула должна быть истинной, а вторая – ложной. Преимущество первой формулы заключается в том, что при нарушении этого свойства будет выведен контрпример, показывающий, при каких событиях происходит нарушение этого условия.

Свойство *живости* выражается в том, что всегда при приеме заявки на обслуживание процесс неизбежно войдет в свою крити-

ческую секцию. Для первого и второго процессов это свойство представляется с помощью следующих формул *CTL*: *AG* (*s1->AF sk1*) и *AG* (*s2->AF sk2*).

Свойство *отсутствия строгого чередования* занятия критических секций процессами выражается с помощью формулы [139]: *EF*(*sk1 & E*(*sk1 U* (*sk1& E* (*sk2 U sk1*)))).

Свойство *наличия приоритетности* вхождения в критические секции можно представить следующими формулами: *AG* (*s1&s2&~sk1&~sk2 ->AX sk1*) и *AG* (*s1&s2&~sk1&~sk2 ->AX sk2*). Первая из формул выражает утверждение, определяющее приоритетность первого процесса над вторым: «Всегда, если приняты заявки от двух процессов и оба процесса не находятся в критической секции, сразу после этого в критическую секцию входит первый процесс».

Свойство *справедливости* в общем случае не может быть выражено в *CTL* [140]. На основе результатов машинных экспериментов можно констатировать, что три из отмеченных выше свойств удовлетворяются, а четвертое – нет, что объясняется входным описанием, определяющим приоритетную дисциплину вхождения в критический участок.

9.2.3. Верификация НДА-модели алгоритма управления планированием/диспетчеризацией задач в системе SMV

Система канонических уравнений алгоритма планирования/диспетчеризации задач в многопроцессорной системе с использованием механизма «рандеву» была представлена в разделе 8.5 (гл. 8). Она включает в себя три части: «клиент», «сервер» и «выполнение после рандеву». Ниже представлен фрагмент кодировки НДА на входном языке SMV, полученный в соответствии с методикой, описанной ранее в этом разделе. Количество процессоров в системе равно четырем.

Кодировка НДА на языке SMV

module main (*Sk_OD*, *Sk_K*, *Sp_S*, *Sp1_P*, *Sp2_P*, *Sp3_P*, *Sp4_P*, *Sp1_0*, *Sp2_0*, *Sp3_0*, *Sp4_0*, *Sp1_ZS*, *Sp2_ZS*, *Sp3_ZS*, *Sp4_ZS*, *Sk_U*, *Sp1_ZD*, *Sp2_ZD*, *Sp3_ZD*, *Sp4_ZD*)

{

```
        input Sk_OD, Sk_K, Sp_S : boolean;
        input Sp1_P, Sp2_P, Sp3_P, Sp4_P : boolean;
        input Sp1_0, Sp2_0, Sp3_0, Sp4_0 : boolean;
        input Sp1_ZS, Sp2_ZS, Sp3_ZS, Sp4_ZS: boolean;

        output Sk_U :boolean;
        output Sp1_ZD, Sp2_ZD, Sp3_ZD, Sp4_ZD :boolean;

        VAR Sk_0 : boolean;
        VAR Sk_BP1, Sk_BP2, Sk_BP3, Sk_BP4 : boolean;
        VAR Sk_KP1, Sk_KP2, Sk_KP3, Sk_KP4 : boolean;
        VAR Sk_g, Sk_S : boolean;

        VAR Sp_0 : boolean;
        VAR Sp1_BK, Sp2_BK, Sp3_BK, Sp4_BK : boolean;
        VAR Sp1_S, Sp2_S, Sp3_S, Sp4_S : boolean;
        VAR Sp1_OK, Sp2_OK, Sp3_OK, Sp4_OK : boolean;
        VAR Sp1_g, Sp2_g, Sp3_g, Sp4_g : boolean;

        VAR Sp1_ST, Sp2_ST, Sp3_ST, Sp4_ST : boolean;
        VAR Sk_OK : boolean;
        VAR Sp1_OD, Sp2_OD, Sp3_OD, Sp4_OD : boolean;

       ASSIGN
       -- Начальное состояние
       init(Sk_OD):=0;
       init(Sk_K):=0;
       init(Sp_S):=0;

       init(Sp1_P):=0; init(Sp2_P):=0; init(Sp3_P):=0; init(Sp4_P):=0;
       init(Sp1_0):=0; init(Sp2_0):=0; init(Sp3_0):=0; init(Sp4_0):=0;
       init(Sp1_ZS):=0; init(Sp2_ZS):=0; init(Sp3_ZS):=0; in-
it(Sp4_ZS):=0;

       init(Sk_U):=0;
       init(Sp1_ZD):=0; init(Sp2_ZD):=0; init(Sp3_ZD):=0;
init(Sp4_ZD):=0;
       ---------------------------------------------------------------
       init(Sk_0):=0;
       init(Sk_g):=0;
```

```
        init(Sk_S):=0;
        init(Sk_BP1):=0; init(Sk_BP2):=0; init(Sk_BP3):=0;
init(Sk_BP4):=0;
        init(Sk_KP1):=0; init(Sk_KP2):=0; init(Sk_KP3):=0;
init(Sk_KP4):=0;

        init(Sp_0):=0;
        init(Sp1_BK):=0; init(Sp2_BK):=0; init(Sp3_BK):=0;
init(Sp4_BK):=0;
        init(Sp1_S):=0; init(Sp2_S):=0; init(Sp3_S):=0; init(Sp4_S):=0;
        init(Sp1_OK):=0; init(Sp2_OK):=0; init(Sp3_OK):=0;
init(Sp4_OK):=0;
        init(Sp1_g):=0; init(Sp2_g):=0; init(Sp3_g):=0; init(Sp4_g):=0;

        init(Sk_OK):=0;
        init(Sp1_ST):=0; init(Sp2_ST):=0; init(Sp3_ST):=0;
init(Sp4_ST):=0;
        init(Sp1_OD):=0; init(Sp2_OD):=0; init(Sp3_OD):=0;
init(Sp4_OD):=0;

        -- Функция переходов
        default {
        next(Sk_0):=0;
        next(Sk_g):=0;
        next(Sk_S):=0;
        next(Sk_BP1):=0; next(Sk_BP2):=0; next(Sk_BP3):=0;
next(Sk_BP4):=0;
        next(Sk_KP1):=0; next(Sk_KP2):=0; next(Sk_KP3):=0;
next(Sk_KP4):=0;

        next(Sp_0):=0;
        next(Sp1_BK):=0; next(Sp2_BK):=0; next(Sp3_BK):=0;
next(Sp4_BK):=0;
        next(Sp1_S):=0; next(Sp2_S):=0; next(Sp3_S):=0;
next(Sp4_S):=0;
        next(Sp1_OK):=0; next(Sp2_OK):=0; next(Sp3_OK):=0;
next(Sp4_OK):=0;
        next(Sp1_g):=0; next(Sp2_g):=0; next(Sp3_g):=0;
next(Sp4_g):=0;
```

```
        next(Sk_OK):=0;
        next(Sp1_ST):=0; next(Sp2_ST):=0; next(Sp3_ST):=0;
next(Sp4_ST):=0;
        next(Sp1_OD):=0; next(Sp2_OD):=0; next(Sp3_OD):=0;
next(Sp4_OD):=0;
        }
        in {
        -- Client --
                next(Sk_0):=Sk_OD & Sk_K;

                next(Sk_BP1):=Sk_0 & Sp_S & Sp1_P;
                next(Sk_BP2):=Sk_0 & Sp_S & Sp2_P;
                next(Sk_BP3):=Sk_0 & Sp_S & Sp3_P;
                next(Sk_BP4):=Sk_0 & Sp_S & Sp4_P;

                next(Sk_KP1):=(Sk_BP1 & Sp1_OK) | (Sk_S & Sp1_P &
Sp1_BK & Sk_OK);
                next(Sk_KP2):=(Sk_BP2 & Sp2_OK) | (Sk_S & Sp2_P &
Sp2_BK & Sk_OK);
                next(Sk_KP3):=(Sk_BP3 & Sp3_OK) | (Sk_S & Sp3_P &
Sp3_BK & Sk_OK);
                next(Sk_KP4):=(Sk_BP4 & Sp4_OK) | (Sk_S & Sp4_P &
Sp4_BK & Sk_OK);

        --      next(Sk_g):=(Sk_KP1 & Sp1_ZD) | (Sk_KP2 & Sp2_ZD) |
(Sk_KP3 & Sp3_ZD) | (Sk_KP4 & Sp4_ZD) | (Sk_g & (~Sp1_g)) |
(Sk_g & (~Sp2_g)) | (Sk_g & (~Sp3_g)) | (Sk_g & (~Sp4_g));
                next(Sk_g):=(Sk_KP1) | (Sk_KP2) | (Sk_KP3) | (Sk_KP4) |
(Sk_g & (~Sp1_g)) | (Sk_g & (~Sp2_g)) | (Sk_g & (~Sp3_g)) | (Sk_g &
(~Sp4_g));

                next(Sk_S):=(Sk_0 & (~Sp_S)) | (Sk_S & Sp1_P &
(~Sp1_BK)) | (Sk_S & Sp2_P & (~Sp2_BK)) | (Sk_S & Sp3_P &
(~Sp3_BK)) | (Sk_S & Sp4_P & (~Sp4_BK));
        --------
        -- Server --
                next(Sp_0):= Sp1_0 | Sp2_0 | Sp3_0 | Sp4_0;
```

next(*Sp1_BK*):=*Sp1_0* & *Sk_0* & *Sp1_P*;
next(*Sp2_BK*):=*Sp2_0* & *Sk_0* & *Sp2_P*;
next(*Sp3_BK*):=*Sp3_0* & *Sk_0* & *Sp3_P*;
next(*Sp4_BK*):=*Sp4_0* & *Sk_0* & *Sp4_P*;

next(*Sp1_S*):=(*Sp1_0* & (~*Sk_0*)) | (*Sp1_S* & *Sp1_P* & (~*Sk_BP1*));
next(*Sp2_S*):=(*Sp2_0* & (~*Sk_0*)) | (*Sp2_S* & *Sp2_P* & (~*Sk_BP2*));
next(*Sp3_S*):=(*Sp3_0* & (~*Sk_0*)) | (*Sp3_S* & *Sp3_P* & (~*Sk_BP3*));
next(*Sp4_S*):=(*Sp4_0* & (~*Sk_0*)) | (*Sp4_S* & *Sp4_P* & (~*Sk_BP4*));

next(*Sp1_OK*):=(*Sp1_S*) & *Sp1_P* & *Sk_BP1*;
next(*Sp2_OK*):=(*Sp2_S*) & *Sp2_P* & *Sk_BP2*;
next(*Sp3_OK*):=(*Sp3_S*) & *Sp3_P* & *Sk_BP3*;
next(*Sp4_OK*):=(*Sp4_S*) & *Sp4_P* & *Sk_BP4*;

next(*Sp1_g*):=((*Sp1_BK* | *Sp1_OK*) & *Sk_KP1*) | (*Sp1_g* & (~*Sk_g*));
next(*Sp2_g*):=((*Sp2_BK* | *Sp2_OK*) & *Sk_KP2*) | (*Sp2_g* & (~*Sk_g*));
next(*Sp3_g*):=((*Sp3_BK* | *Sp3_OK*) & *Sk_KP3*) | (*Sp3_g* & (~*Sk_g*));
next(*Sp4_g*):=((*Sp4_BK* | *Sp4_OK*) & *Sk_KP4*) | (*Sp4_g* & (~*Sk_g*));

-- Randevu --

next(*Sp1_ST*):=(*Sk_g* & *Sp1_g*) | ((~*Sp1_ZS*) & *Sp1_ST*);
next(*Sp2_ST*):=(*Sk_g* & *Sp2_g*) | ((~*Sp2_ZS*) & *Sp2_ST*);
next(*Sp3_ST*):=(*Sk_g* & *Sp3_g*) | ((~*Sp3_ZS*) & *Sp3_ST*);
next(*Sp4_ST*):=(*Sk_g* & *Sp4_g*) | ((~*Sp4_ZS*) & *Sp4_ST*);

next(*Sk_OK*):=(*Sp1_ST* & *Sp1_ZS*) | (*Sp2_ST* & *Sp2_ZS*) | (*Sp3_ST* & *Sp3_ZS*) | (*Sp4_ST* & *Sp4_ZS*);

next(*Sp1_OD*):=*Sp1_ST* & *Sp1_ZS*;
next(*Sp2_OD*):=*Sp2_ST* & *Sp2_ZS*;
next(*Sp3_OD*):=*Sp3_ST* & *Sp3_ZS*;
next(*Sp4_OD*):=*Sp4_ST* & *Sp4_ZS*;

next(*Sp1_ZD*):=*Sp1_OD & Sk_U*;
next(*Sp2_ZD*):=*Sp2_OD & Sk_U*;
next(*Sp3_ZD*):=*Sp3_OD & Sk_U*;
next(*Sp4_ZD*):=*Sp4_OD & Sk_U*;

next(*Sk_U*):=*Sk_OK &* (*Sp1_OD* | *Sp2_OD* | *Sp3_OD* | *Sp4_OD*);

}

}

Данные свойства достижимости состояний в ветвях алгоритма «клиент», «сервер» (для одного процессора) и «выполнения после рандеву» выражаются с помощью следующих выражений временной логики:

SPEC EF Sk_0;
SPEC EF Sk_BP1;
SPEC EF Sk_KP1;
SPEC EF Sk_g;
SPEC EF Sk_S;
SPEC EF Sp_0;
SPEC EF Sp1_BK;
SPEC EF Sp1_S;
SPEC EF Sp1_OK;
SPEC EF Sp1_g;
SPEC EF Sp1_ST;
SPEC EF Sk_OK;
SPEC EF Sp1_OD;
SPEC EF Sp1_ZD;
SPEC EF Sk_U;

Результатом прогона программы SMV является протокол, приведенный на рис. 9.4, который показывает, что все указанные состояния достижимы при заданных начальных условиях. Таким образом, в системе нет состояний, которые никогда бы не были достижимы из начальных состояний.

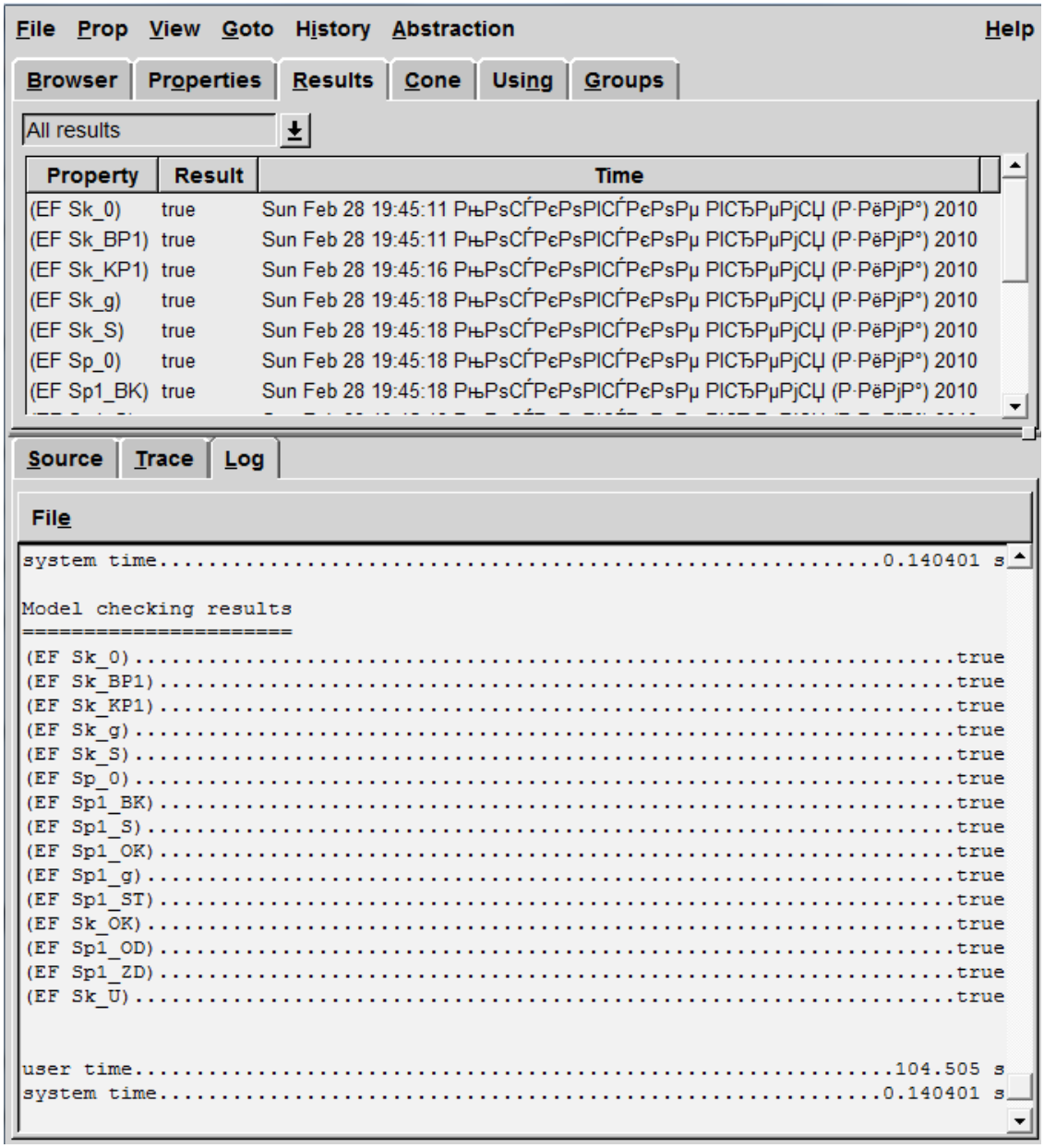


Рис. 9.4. Результаты верификации НДА алгоритма функционирования диспетчера задач в системе SMV

9.2.4. Верификация НДА-модели алгоритма планирования/ диспетчеризации задач с использованием пакета Stateflow Matlab

Stateflow – пакет моделирования событийно-управляемых систем, основанный на теории конечных автоматов. Этот пакет предназначен для использования вместе с пакетом моделирования динамических систем *Simulink.* В любую *Simulink*-модель можно

вставить Stateflow-диаграмму (или SF-диаграмму), которая будет отражать поведение компонентов объекта (или системы) моделирования. SF-диаграмма является анимационной. По ее выделяющимся цветом блокам и связям можно проследить все стадии работы моделируемой системы или устройства и поставить ее работу в зависимость от тех или иных событий [143].

Stateflow позволяет комбинировать графические и табличные представления, включая граф-схемы переходов состояний, таблицы переходов состояний и таблицы истинности, для того, чтобы смоделировать реакцию системы управления на возникающие события, условия во времени и внешние входные сигналы. При помощи конечных автоматов моделируются различные компоненты систем управления как состояния, которые выполняются последовательно или параллельно. Моделирование в Stateflow позволяет убедиться, что проект является согласованным и целостным (верифицированным) до реализации его в аппаратных или программных средствах. Stateflow позволяет управлять сложностью проекта путем иерархической организации графа автомата, моделирующего функции и компоненты системы уп-равления [145].

Для целей верификации проведено моделирование алгоритмов управления планированием/диспетчеризацией задач с временным разделением в многопроцессорных системах, описанных с использованием логики НДА (см. гл. 8, раздел 8.5). На рис. 9.5 представлена модель вышеописанного автомата, реализованная в Stateflow.

Автомат разделяется на две части: до и после рандеву (до того, как задача поступила в процессор и он готов к ее исполнению, и после). В части до рандеву параллельно выполняются две ветви: клиент и сервер. Со стороны клиента происходит выборка задачи и запрос свободного процессора. Со стороны сервера – выбор процессора и принятие задачи на исполнение.

Клиентская часть в Stateflow представлена на рис. 9.6.

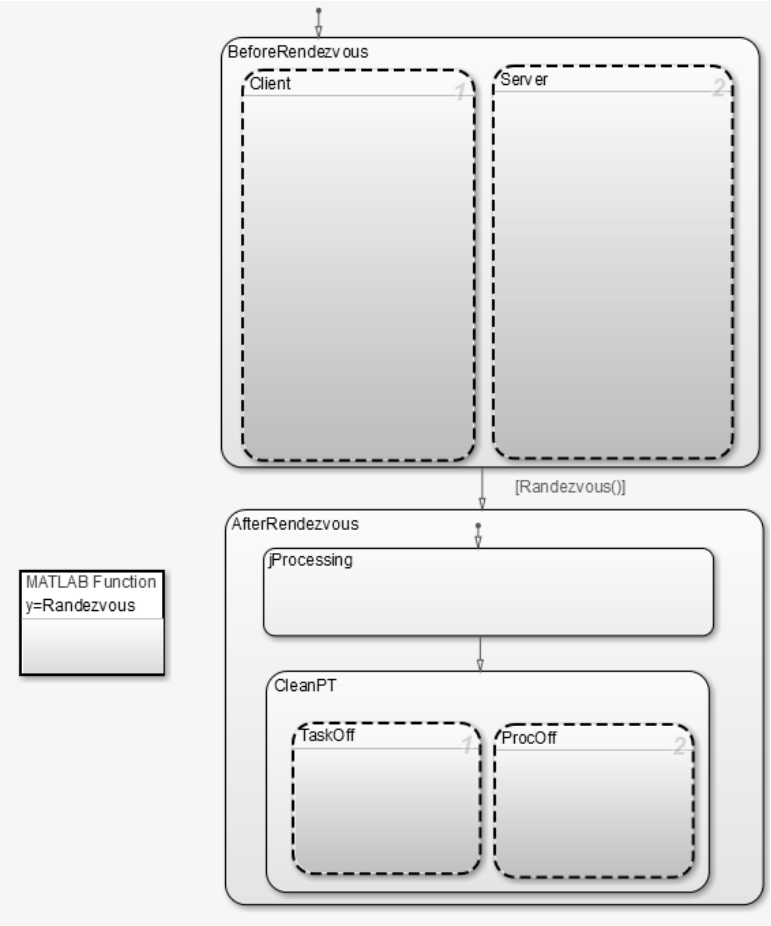


Рис. 9.5. Модель автомата в Stateflow

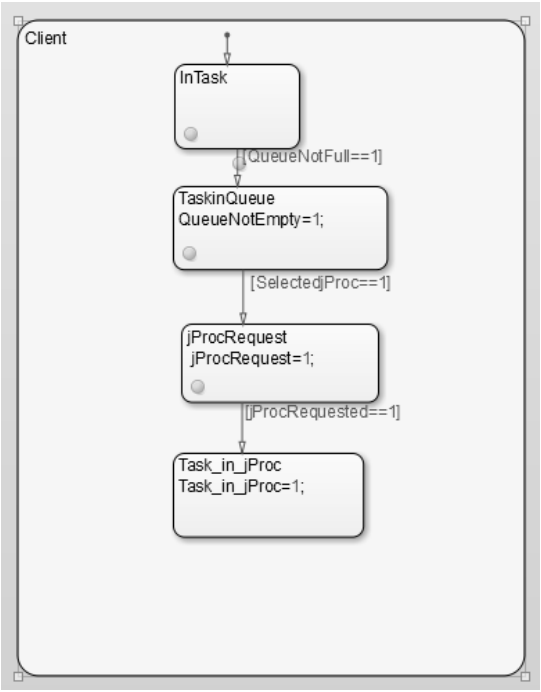


Рис. 9.6. Модель клиентской части алгоритма диспетчеризации в Stateflow

Блок *InTask* означает поступление задачи в диспетчер извне (моделирует событие S_I^t). Блок *TaskInQueue* означает, что задача помещена в очередь (S_Q^t). Событие *QueueNotEmpty* (S_{FQ}^t) означает, что в очереди есть задачи, ожидающие обслуживания. Блок *jProcRequest* (S_{ZPj}^t) означает запрос процессора диспетчером. Событие *jProcRequest* означает то же самое и используется для осуществления перехода в параллельной ветви. Блок и событие *Task_in_Proc* (S_{PT}^{pj}) означают, что задача поступила на обслуживание в свободный процессор.

Серверная часть в Stateflow представлена на рис. 9.7. Блок *FreeProcChoise* (S_S^p) означает выбор свободного процессора из их множества. Событие *SelectedjProc* (S_{SL}^{pj}) означает, что процессор выбран. Блок *TaskRequest* (S_{GPj}^t) означает запрос наличия задачи в очереди. Сигнал *[proc_rqstd()]* является функцией, которая представляет собой конъюнкцию двух событий: очередь не пуста и диспетчер запрашивает свободный процессор (*QueueNotEmpty* и *jProcRequest*). Блок и событие *jProcRequested* (S_{PZ}^{pj}) означают, что свободный процессор ответил на запрос и готов к чтению задачи из очереди. Блок и событие *ProcTaskReaded* (S_{GPj}^t) означает, что задача прочитана процессором и находится на обслуживании.

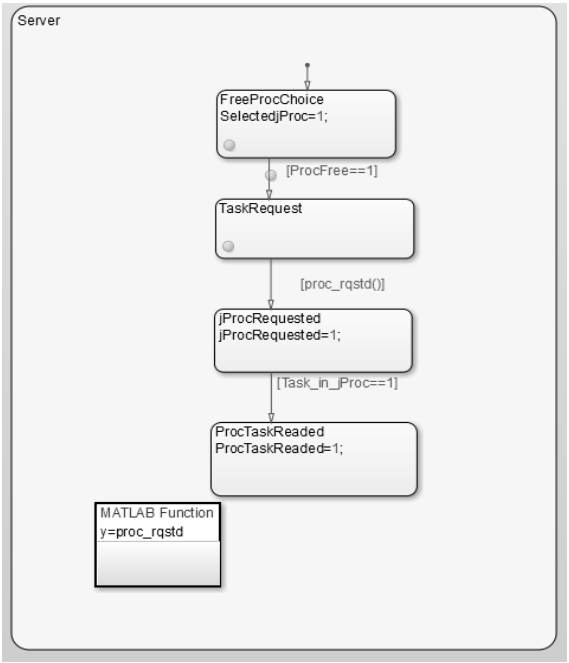


Рис. 9.7. Модель серверной части алгоритма диспетчеризации в Stateflow

После рандеву задача выполняется процессором, а потом выполняется освобождение процессора от задачи и запись результатов. Это происходит параллельно.

Момент рандеву реализуется функцией *Randezvous*(), которая представляет собой конъюнкцию двух событий: задача из очереди поступила на обслуживание в процессор и процессор прочитал задачу из очереди (*Task_in_Proc* и *ProcTaskReaded*). Блок *jProcessing* (S_A^{pj}) означает выполнение задачи в процессоре. Освобождение процессора от задачи реализуется в модели блоками, показанными на рис. 9.8.

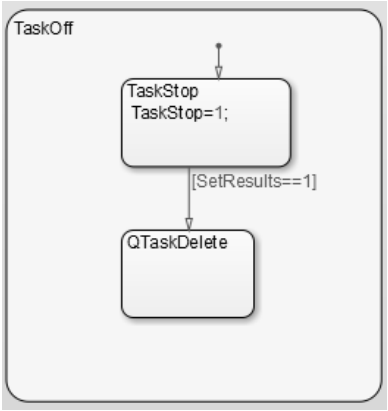


Рис. 9.8. Модель алгоритма в части «освобождение процессора»

Блок и событие *TaskStop* (S_E^{pj}) означают, что процессор завершил выполнение задачи. Блок *QTaskDelete* (S_{OF}^{t}) означает, что задача удалена из очереди. Запись результатов производится блоками, показанными на рис. 9.9.

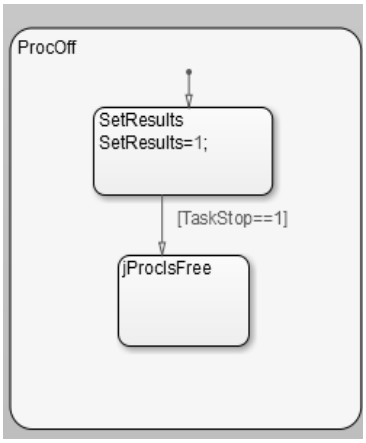


Рис. 9.9. Модель алгоритма в части «запись результатов»

Блок и событие *SetResults* (S_{RT}^{t}) означают, что результаты выполнения задачи записаны в память. Блок *jProcIsFree* (S_{S}^{pj}) означает, что процессор помещается в пул свободных.

В Stateflow можно запустить отладку и наблюдать за работой автомата в целом. Для этого необходимо объявить все сигналы переходов и задать между ними соответствия. После реализации модель была отлажена.

Stateflow позволяет создавать различные отчеты по результатам компиляции и отладки. Наибольший интерес вызывают верификация и отчет об ошибках в модели, представленные на рис. 9.10 и 9.11.

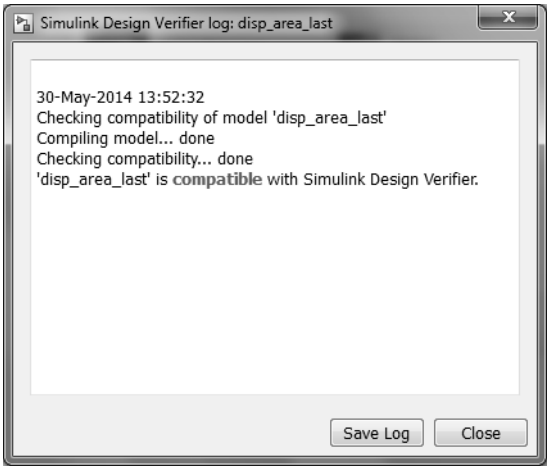


Рис. 9.10. Результаты верификации автоматной модели

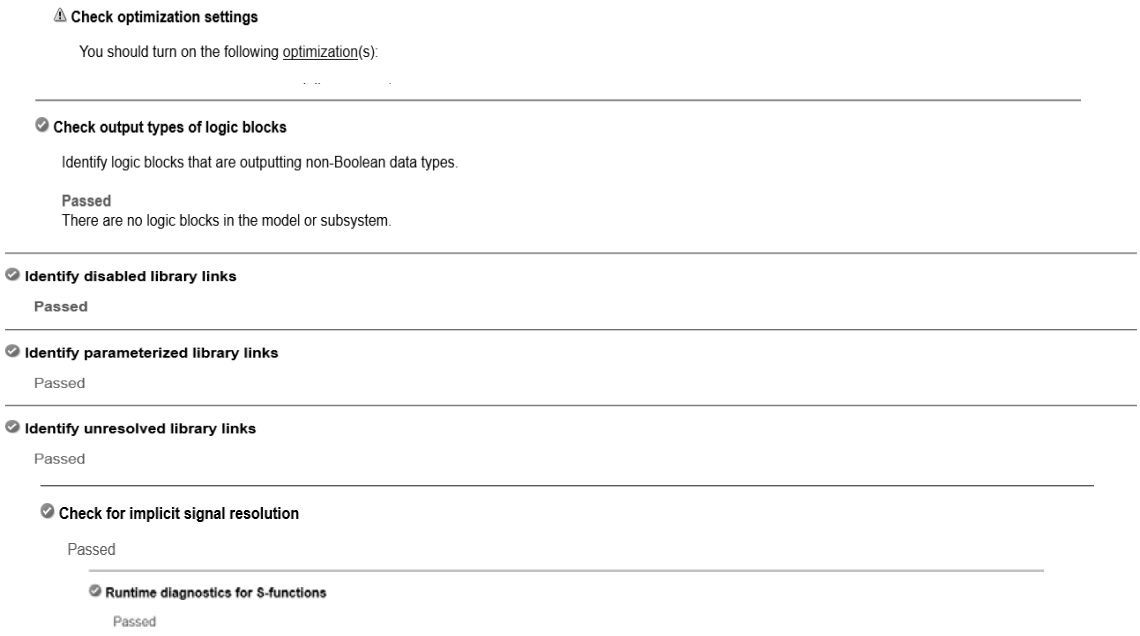


Рис. 9.11. Отчет об ошибках

Из рис. 9.10 можно сделать вывод о том, что все состояния автомата достижимы и все связи и сигналы запрограммированы верно и полностью функционируют.

Первая запись на рис. 9.11 (отчет) говорит о том, что нужно проверить настройки оптимизации кода, который генерируется при компиляции модели. Вторая запись означает, что проверены типы выходных данных используемых блоков. Следующие три проверки также относятся к части компилятора и говорят о корректных ссылках на программные библиотеки. Две последние записи означают, что в модели отсутствуют неявно заданные (неопределенные) сигналы и отмечено корректное выполнение функций.

9.2.5. Верификация НДА-модели алгоритма планирования/диспетчеризации задач с использованием пакета CPN TOOLS

При моделировании работы цифровых устройств одной из проблем является описание алгоритма взаимодействия всех блоков устройства, которые по назначению делятся на два класса: операционные и управляющие. Операционные блоки выполняют передачу и обработку данных, а управляющие – формируют для этого необходимый набор управляющих сигналов. При описании алгоритма работы управляющих блоков наиболее эффективно использовать теорию цифровых автоматов, так как полученный автомат легко преобразуется формальным путем в цифровое устройство. При описании операционных блоков применяют языки регистровых передач или микроопераций, а вся процедура их проектирования носит эвристический характер. В силу этого возможно внесение ошибок и неверных решений, поэтому имеется насущная необходимость в верификации (отладке) проекта, содержащего в комплексе как операционные, так и управляющие блоки.

Одним из путей решения данной задачи является применение имитационного моделирования, позволяющего описывать цифровые устройства без привязки к конкретным аппаратным решениям. Существуют языковые и инструментальные средства имитационного моделирования. Первые основаны на любых уни-

версальных либо на специализированных языках (например, *GPSS*), вторые – на применении готовых шаблонов моделей из базы данных (например, Matlab Simulink, CPN Tools и др.).

Одним их критериев, предъявляемых к системе имитационного моделирования, является возможность визуализации, что значительно упрощает процесс отладки. Инструментальная система моделирования CPN Tools удовлетворяет этому критерию и позволяет строить системы любой сложности. Она использует аппарат функциональных цветных иерархических временных сетей Петри. С помощью CPN Tools выполнена верификация алгоритмов планирования/диспетчеризации задач и отладка средств аппаратной поддержки алгоритмов планирования/диспетчеризации задач в составе многопроцессорной системы.

На основании представленной в разделе 8.5 НДА-модели алгоритма управления планированием и диспетчеризацией задач была построена модель цветной сети Петри [125]. При построении использованы следующие цвета: *BOOL*, *DATA*, *lDATA*, *PAK*. Для описания позиций и моделирующих сигналов (частные события цифрового автомата) использовался цвет *BOOL*. Фишки данного цвета могут принимать значения *true* и *false*. Как будет показано в дальнейшем, при использовании данного цвета описание функции передачи фишки в позицию сети Петри полностью аналогично уравнению перехода в новое состояние для цифрового автомата. С помощью цвета *DATA* моделируются задачи, поступающие в диспетчер. Данный цвет позволяет изменять назначение и количество полей, описывающих задачу [135]. Использование данного цвета позволяет при необходимости выполнять в процессе моделирования сбор статистики. Цвет *lDATA* используется при описании очереди задач в буфере типа FIFO (*first in first out*). Так как используемая сеть Петри не поддерживает ингибиторные дуги, мы используем цвет *PAK*. Позиции данного цвета могут содержать фишки двух цветов *DATA* или *AVAIL*. Если в позиции находится фишка цвета *AVAIL*, то это означает, что данная позиция свободна.

Модель устройства диспетчеризации задач, включающая операционную часть и управляющий автомат, представлена на рис. 9.12.

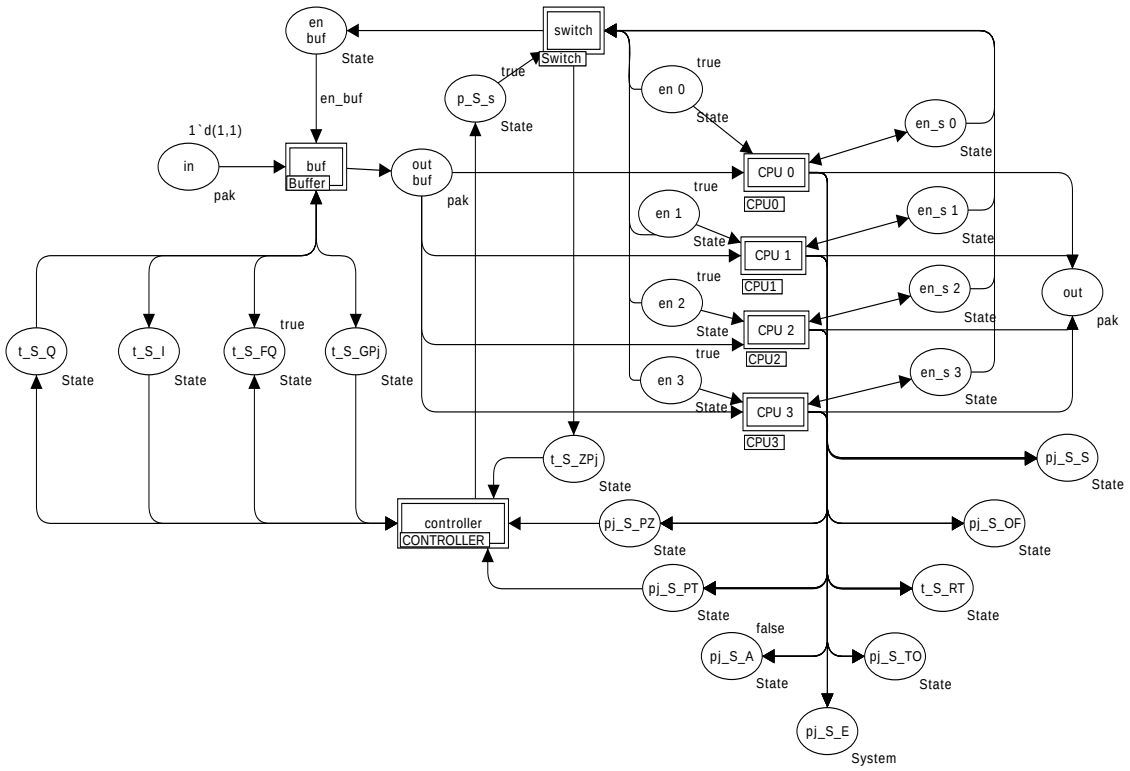


Рис. 9.12. Модель устройства диспетчеризации

Она содержит: входной порт (*in*), через который поступают задачи; выходной порт (*out*), через который процессоры выдают результаты выполнения задачи; очередь задач (*buf*), которая строится по принципу буфера FIFO; узел выбора (*switch*), определяющий процессор для обработки очередной задачи в соответствии с приоритетной дисциплиной обслуживания и циклическим сдвигом приоритетов; четыре процессора (*CPU0*, *CPU1*, *CPU2*, *CPU3*), которые моделируют алгоритм выполнения задачи; сигнальные позиции, через которые узлы обмениваются информацией; узел управления (*controller*), представляющий собой управляющий автомат.

На рис. 9.13 представлена модель буфера задач. При поступлении новой задачи переходом *in* формируется сигнал, по которому при наличии свободного места задача помещается в буфер. Если в очереди отсутствуют свободные места, то задача помещается в позицию *Garbage collector* и далее не обслуживается. Данная позиция необходима для сбора статистики о необслуженных задачах. При постановке задачи в очередь проверяется, остаются ли при этом свободные места. Если свободных мест нет, то формируется сигнал о заполнении очереди. Если при заполненной очереди была выполнена операция изъятия задачи, то формируется сигнал о

наличии свободного места. При изъятии задачи из очереди формируется сигнал о готовности ее к исполнению.

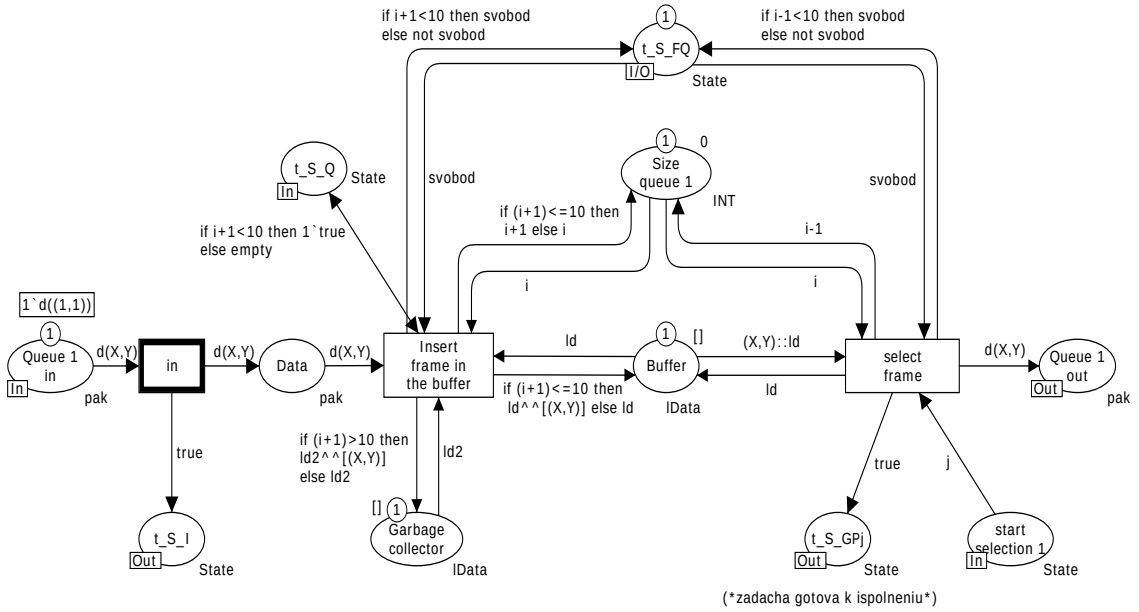


Рис. 9.13. Модель буфера задач

Узел выбора процессора представлен на рис. 9.14. Если хотя бы один процессор не занят, формируется сигнал наличия свободного процессора. Узел выбора обслуживающего процессора запускается по сигналу запроса от диспетчера. Если процессор выбран, то формируются сигналы *en_buf*, разрешающий изъять из буфера задачу, и *en_s*, разрешающий выбранному процессору начать выполнение задачи.

Подсети, моделирующие работу процессорных узлов, идентичны. Модель работы одного процессора представлена на рис. 9.15. При получении сигнала запроса процессор формирует сигнал подтверждения запроса на выполнение задачи. Изъятая из буфера задача поступает в процессор, где формируется сигнал о ее принятии на выполнение. Одновременно с этим процессор изменяет флаг своего состояния, который указывает на занятость процессора, и начинает выполнять задачу. По окончании обслуживания процессор формирует сигнал завершения, после чего он последовательно формирует сигналы снятия задачи с выполнения, выдачи результата, включения свободного процессора в пул и удаления задачи, после чего флаг состояния процессора изменяется, указывая на освобождение процессора.

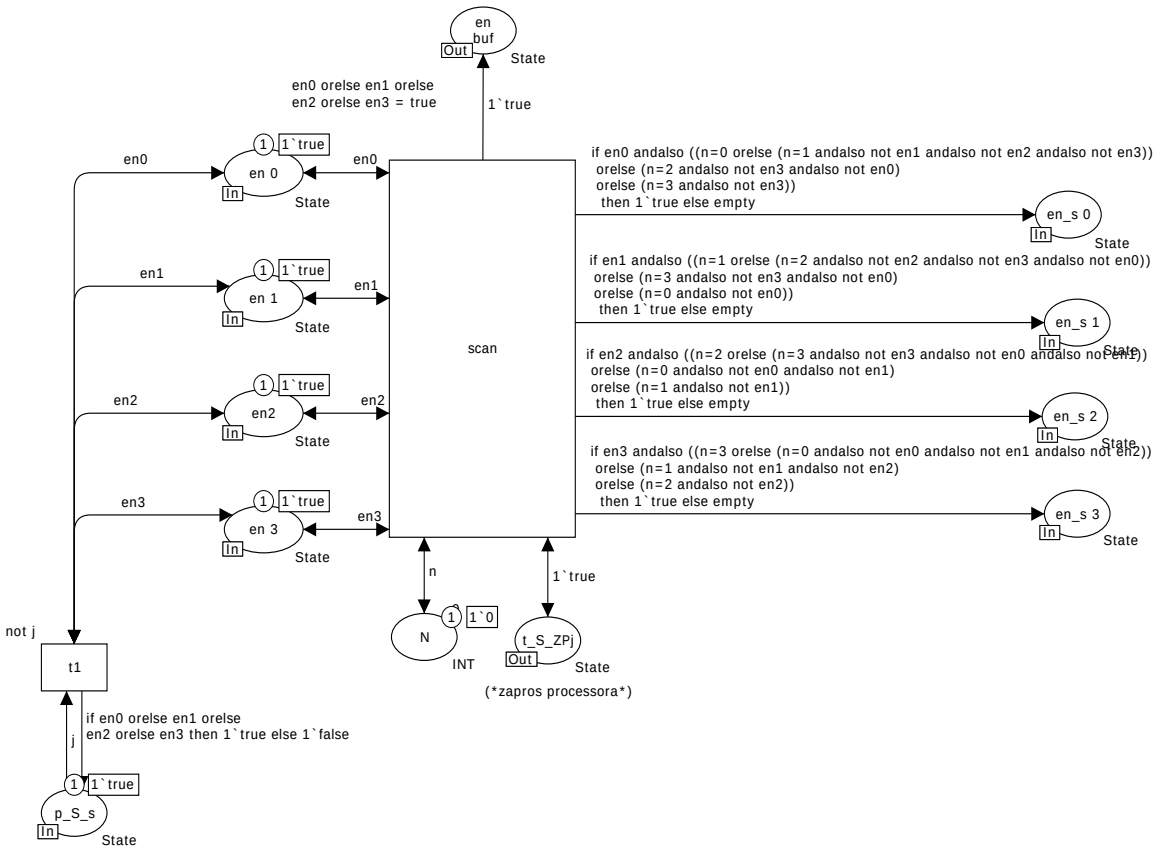


Рис. 9.14. Модель узла выбора процессора

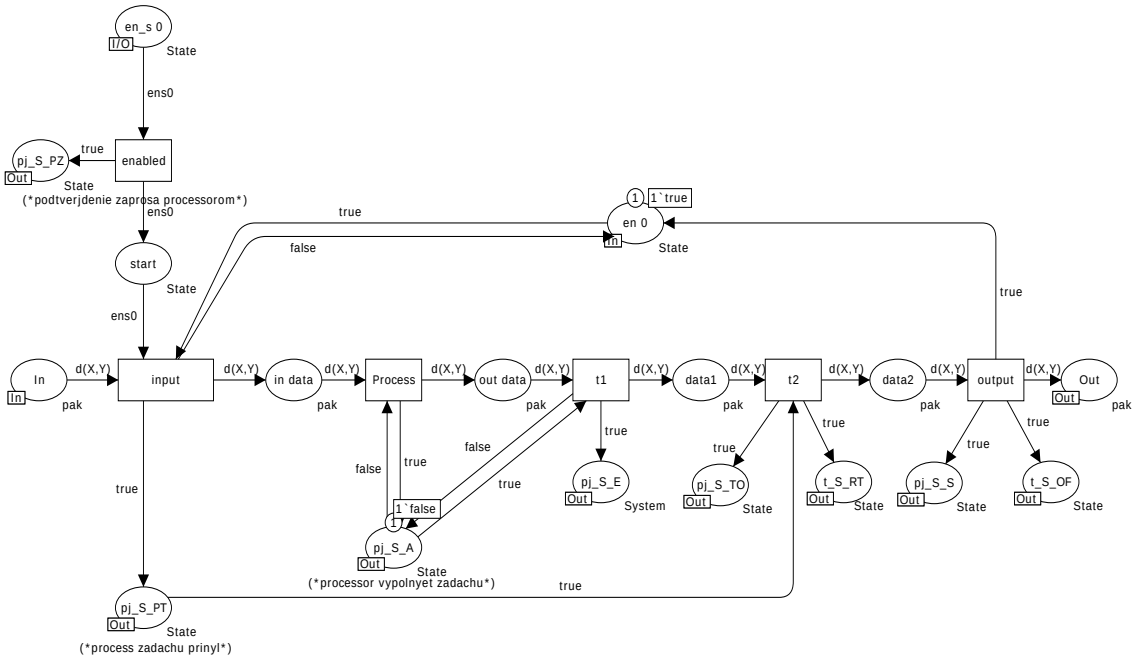


Рис. 9.15. Модель процессорного узла

Узел управления *controller*, представленный на рис. 9.16, описывает последовательность формирования сигналов конечным автоматом, используя метод синхронизации «рандеву» для параллельных процессов, реализуемый клиентом и сервером.

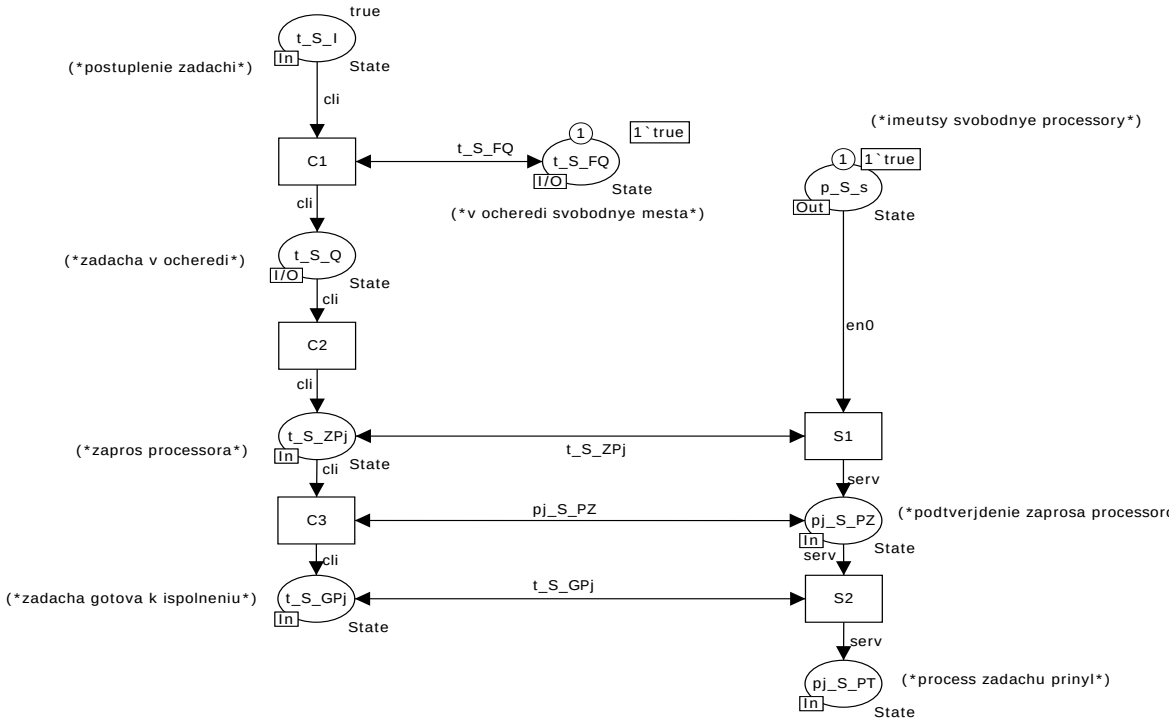


Рис. 9.16. Модель узла управления *controller* методом синхронизации «рандеву» для параллельных процессов, реализуемым клиентом и сервером

Условием перехода в позицию *t_S_Q* фишки является одновременное наличие фишек в позициях *t_S_I* и *t_S_FQ*. Это можно записать уравнением

$$t\_S\_Q = t\_S\_I \vee t\_S\_FQ.$$

Заменив обозначения состояний в сети Петри на обозначение частных состояний цифрового автомата, получаем следующее уравнение:

$$S_Q^t(t{+}1) = S_I^t \vee S_{FQ}^t.$$

Рассмотрев все переходы в подсети *controller*, приходим к выводу, что узел управления сети Петри соответствует цифровому автомату, описанному в разделе 8.5 (гл. 8).

Использование разработанной имитационной модели позволяет производить оптимизацию алгоритмов управления. Например, в процессе проработки модели проверена полнота набора управляющих сигналов, их необходимость и достаточность. Кроме того, данный метод может служить основой для оценки показателей производительности, например, латентности, времени ответа устройства и др.

ЗАКЛЮЧЕНИЕ

Отметим наиболее важные достоинства способа представления алгоритмов управления на основе использования модели НДА и ее аналитического представления на языке НД СКУ и методов его использования для синтеза систем логического управления на разных уровнях преобразования информации.

Математическая модель алгоритмов логического управления, представленная на языке НД СКУ, совместно с графическим представлением ее на языке ГСАП, является достаточно простой и универсальной, позволяющей ее использовать для широкого круга применений по формализации алгоритмов управления объектами и процессами, начиная от простейших устройств управления выполнением арифметических операций в процессоре и кончая управлением процессами и ресурсами в вычислительных системах и сетях. При этом обеспечивается сравнительная легкость представления информации об алгоритме управления в ЭВМ с помощью списочных структур для целей автоматизации проектирования и моделирования.

Способ обладает компактностью представления алгоритмов управления, в том числе алгоритмов с взаимодействующими параллельными ветвями с любой степенью параллелизма, что в значительной степени позволяет снизить ограничения на размерность проектируемых и моделируемых систем управления, понимая под размерностью произведение числа событий и числа входных сигналов.

Наиболее естественно (по шагам) отражается алгоритм работы управляющего устройства, что позволяет осуществить непосредственный переход к различным вариантам структурных схем проектируемого устройства. В частности, если каждому событию поставить в соответствие *D*-триггер, то правые части уравнений СКУ будут функциями возбуждения этих триггеров.

Способ допускает относительно легкий переход к СКУ для систем управления, заданных на различных начальных языках, что позволяет с единых позиций подойти к синтезу таких систем независимо от первоначального формального описания их на одном из начальных языков.

Математический аппарат эквивалентных преобразований НДА, представленный в монографии, может успешно сочетаться с хорошо разработанным аппаратом математической логики и мно-

гими важными положениями теории цифровых автоматов для выполнения различных эквивалентных преобразований СКУ и СВФ на различных этапах синтеза систем логического управления.

В связи с тем, что операции детерминизации и кодирования НДА являются операциями, обратными друг другу, представляется возможность организации некоторых эффективных процедур контроля и верификации, получаемых в процессе синтеза различных систем канонических уравнений.

Представление состояний (полных событий) в виде совокупности частных событий, одновременное существование которых в системе управления возможно, позволяет использовать для структурного синтеза систем управления другой подход к кодированию состояний системы управления, когда код каждого состояния представляется в виде композиции кодов групп несовмес-тимых частных событий. Такое кодирование приводит к значительному упрощению как структуры системы управления, так и процедур их диагностики.

Язык НД СКУ может быть успешно использован для формализации алгоритмов управления взаимодействующими синхронными и асинхронными параллельными процессами с решением задач обмена данными между процессами и их синхронизации с отсутствием тупиковых ситуаций. Это перспективное направление использования языка НД СКУ может быть значительно расширено и дополнено с учетом решения задач структурной реализации и моделирования автоматных недетерминированных и детерминированных моделей систем параллельной и распределенной обработки информации.

В монографии были рассмотрены также вопросы построения структур микропрограммного управления на основе использования языка НД СКУ для систем параллельной обработки. Такие структуры отличаются высокой производительностью и простотой. Используя их формальное представление на языке НД СКУ, можно успешно решать задачи автоматизации отдельных этапов их структурного синтеза.

Рассмотрены вопросы структурной реализации наиболее трудоемких функций ядра многопроцессорных операционных систем, основанной на языке НДА. Выявлена принципиальная возможность аппаратной реализации указанных функций в ПЛИС. Полученные на функциональных моделях, выполненных на языке

VHDL, характеристики показывают высокую производительность механизмов управления процессами (планировщиков и диспетчеров задач) и механизмов управления ресурсами. Возможности верификации и тщательной отладки аппаратуры с использованием логики НДА и языка VHDL дают основание полагать о получении высоконадежных и безопасных функций ядра проектируемых многопроцессорных операционных систем.

Из анализа работ, посвященных теории синтаксического анализа, перевода и компиляции [3, 6, 7, 78], в которых авторы используют модель НДА, следует, что представленные в данном издании результаты в области теории НДА и их эквивалентных преобразований могут быть успешно использованы также и при проектировании различных синтаксических анализаторов и компиляторов.

Учитывая отмеченные выше достоинства представления алгоритмов управления на языке НД СКУ, можно отметить, что этот язык позволяет в некоторой степени приблизить решение задачи, поставленной Ч. Хоаром в [55], по поиску как можно более простой математической теории, позволяющей:

а) описывать широкий круг применений по управлению процессами и дискретному моделированию событий, начиная с торговых автоматов и кончая операционными системами с разделяемыми ресурсами;

б) обеспечивать эффективную реализацию систем управления от простейших автоматов до мультипроцессорных устройств и сетей взаимодействующих микропроцессоров;

в) обеспечивать комплексное решение вопросов в деле спецификации, разработки, реализации, верификации и анализа сложных систем управления объектами промышленной автоматики, а также систем управления параллельными процессами и ресурсами в вычислительных системах и сетях.

СПИСОК ЛИТЕРАТУРЫ

1. Глушков, В. М. Синтез цифровых автоматов / В. М. Глушков. – М. : Физматгиз, 1962. – 476 с.

2. Вашкевич, Н. П. Синтез микропрограммных управляющих автоматов : учеб. пособие / Н. П. Вашкевич. – Пенза : Пенз. политехн. ин-т, 1990. – 115 с.

3. Льюис, Ф. Теоретические основы проектирования компиляторов / Ф. Льюис, Д. Розенкранц, Р. Стирнз. – М. : Мир, 1979. – 654 с.

4. Рабин, М. О. Конечные автоматы и задачи их разрешения / М. О. Рабин, Д. Скотт // Кибернетический сборник. – М. : ИЛ, 1962. – № 4. – С. 56–71.

5. Трахтенброт, Б. А. Конечные автоматы (поведение и синтез) / Б. А. Трахтенброт, Я. И. Барздинь. – М. : Наука, 1970. – 400 с.

6. Ахо, А. Теория синтаксического анализа, перевода и компиляции / А. Ахо, Дж. Ульман. – М. : Мир, 1978. – Т. 1. – 612 с.

7. Рейуорд-Смит, В. Дж. Теория формальных языков. Вводный курс / В. Дж. Рейуорд-Смит. – М. : Радио и связь, 1988. – 129 с.

8. Кузнецов, О. П. Дискретная математика для инженеров / О. П. Кузнецов, Г. М. Андельсон-Вельский. – М. : Энергия, 1980. – 341 с.

9. Зубков, В. А. Синтез абстрактных цифровых автоматов с использованием языка рекурсивных предикатов / В. А. Зубков, Н. П. Вашкевич // Вычислительная техника : учеб. записки. – Пенза : Пенз. политехн. ин-т, 1969. – Вып. 3. – С. 3–12.

10. Вашкевич, Н. П. Проектирование алгоритмов и структур цифровых устройств : учеб. пособие / Н. П. Вашкевич, В. Г. Пучков. – Пенза : Пенз. политехн. ин-т, 1978. – 110 с.

11. Вашкевич, Н. П. Синтез цифровых управляющих автоматов на основе языка систем канонических уравнений : учеб. пособие / Н. П. Вашкевич, В. Г. Пучков. – Пенза : Пенз. политехн. ин-т, 1980. – 100 с.

12. Зубков, В. А. Алгоритм синтеза цифровых автоматов Мура с использованием языка исчисления предикатов / В. А. Зубков, Н. П. Вашкевич // Вычислительная техника : учен. записки. – Пенза : Пенз. политехн. ин-т, 1969. – Вып. 3. – С. 25–36.

13. Логика. Автоматы. Алгоритмы / М. А. Айзерман [и др.]. – М. : Гос. изд-во физ.-мат. литературы, 1963. – 556 с.

14. Кузнецов, О. П. О сравнительной теории алгоритмических языков логического управления / О. П. Кузнецов // Теория дискретных управляющих устройств. – М. : Наука, 1982. – С. 113–127.

15. Клини, С. К. Представление событий в нейронных сетях и конечных автоматах / С. К. Клини // Автоматы : сб. – М. : ИЛ, 1956. – С. 15–67.

16. Вашкевич, Н. П. Об одном способе синтеза цифровых автоматов по граф-схеме алгоритма с параллельными ветвями / Н. П. Вашкевич // Вычислительная техника : межвуз. сб. науч. тр. – Пенза : Пенз. политехн. ин-т, 1973. – Вып. 1, 2. – С. 49–57.

17. Вашкевич, Н. П. Способ синтеза цифровых управляющих автоматов, заданных на языке логических схем алгоритмов (ЛСА) с параллельными ветвями / Н. П. Вашкевич // Специализированные и комбинированные вычислительные устройства : межвуз. сб. науч. тр. – Рязань : Рязан. радиотехн. ин-т, 1975. – Вып. 2. – С. 19–24.

18. Кларк, Э. М. Верификация моделей программ: Model Checking / Э. М. Кларк, О. Граммберг, Д. Пелед. – М. : МЦНМО, 2002. – 416 с.

19. Вашкевич, Н. П. Построение системы канонических уравнений для управляющих автоматов, заданных на языках ЛСА и ГСА с параллельными ветвями / Н. П. Вашкевич, В. Г. Пучков // Вопросы радиоэлектроники. Сер. ЭВТ. – 1977. – Вып. 13. – С. 57–67.

20. Вашкевич, Н. П. Особенности одного способа представления алгоритмов функционирования цифровых управляющих автоматов / Н. П. Вашкевич // Специализированные и комбинированные вычислительные устройства : межвуз. сб. науч. тр. – Рязань : Рязан. радиотехн. ин-т, 1975. – Вып. 2. – С. 24–28.

21. Вашкевич, Н. П. Недетерминированные автоматы и их использование для синтеза систем уравнений. Эквивалентные преобразования недетерминированных автоматов : учеб. пособие / Н. П. Вашкевич, С. Н. Вашкевич. – Пенза : Изд-во Пенз. гос. техн. ун-та, 1996. – Ч. 1. – 88 с.

22. Зубков, В. А. Описание регулярных выражений алгебры событий рекурсивными предикатами / В. А. Зубков, Н. П. Вашкевич // Вычислительная техника : сб. – Саратов ; Пенза : Приволж. кн. изд-во, 1968. – С. 130–144.

23. Баранов, С. И. Синтез микропрограммных автоматов / С. И. Баранов. – Л. : Энергия, 1979. – 231 с.

24. Дьяченко, В. Ф. Управление на сетях связи / В. Ф. Дьяченко, В. Г. Лазарев, Г. Г. Саввин. – М. : Наука, 1967. – 223 с.

25. Лазарев, В. Г. Построение программируемых управляющих устройств / В. Г. Лазарев, Е. И. Пийль, Е. Н. Турута. – М. : Энергоатомиздат, 1984. – 192 с.

26. Лазарев, В. Г. Синтез управляющих автоматов / В. Г. Лазарев, Е. И. Пийль. – М. : Энергоатомиздат, 1989. – 328 с.

27. Гилл, А. Введение в теорию конечных автоматов / А. Гилл. – М. : Наука, 1966. – 272 с.

28. Вашкевич, Н. П. Минимизация систем канонических уравнений, представляющих цифровой автомат, с учетом распределения сдвигов / Н. П. Вашкевич // Вопросы радиоэлектроники, ЭВТ : сб. – 1980. – Вып. 10. – С. 80–86.

29. Клини, С. К. Введение в метаматематику / С. К. Клини. – М. : ИЛ, 1957. – 526 с.

30. Кутепов, В. П. О тождественных преобразованиях регулярных выражений / В. П. Кутепов // Цифровая вычислительная техника и программирование : сб. – М. : Сов. радио, 1967. – Вып. 2. – С. 67–73.

31. Вавилов, Е. Н. Синтез схем электронных цифровых машин / Е. Н. Вавилов, Г. П. Портной. – М. : Сов. радио, 1963. – 440 с.

32. Система преобразования и моделирования недетерминированных управляющих автоматов, заданных автоматными языками / Н. П. Вашкевич, А. М. Краснов, О. С. Виноградов, Е. В. Горбунков // Вычислительная техника в автоматизированных системах контроля и управления : межвуз. сб. науч. тр. – Пенза : Изд-во Пенз. гос. техн. ун-та, 1995. – Вып. 23. – С. 41–48.

33. Котов, В. Е. Языки параллельного программирования / В. Е. Котов // Алгоритмы, математическое обеспечение и архитектура микропроцессорных вычислительных систем. – М. : Наука, 1982. – С. 139–167.

34. Кобринский, Н. Е. Введение в теорию конечных автоматов / Н. Е. Кобринский, Б. А. Трахтенброт. – М. : Физматгиз, 1962. – 405 с.

35. Черч, А. Введение в математическую логику / А. Черч. – 2-е изд. – М. : ИЛ, 2009. – Т. 1. – 482 с.

36. Новиков, П. С. Элементы математической логики / П. С. Новиков. – М. : Физматгиз, 1973. – 400 с.

37. Вашкевич, С. Н. Структурный синтез управляющих автоматов, заданных НДА / С. Н. Вашкевич, А. Ю. Филатов // Новые

информационные технологии и системы : материалы Междунар. науч.-техн. конф. – Пенза, 1994. – С. 123.

38. Котов, В. Е. Формальные модели параллельных вычислений / В. Е. Котов // Алгоритмы, математическое обеспечение и архитектура микропроцессорных вычислительных систем. – М. : Наука, 1982. – С. 104–138.

39. Барашенков, В. В. Временное совмещение алгоритмов / В. В. Барашенков // Вычислительная техника : сб. – Л. : Энергия, 1972. – Вып. 2. – С. 8–11.

40. Моисеева, Г. Н. О синтезе автоматов по графам алгоритмов с параллельными ветвями / Г. Н. Моисеева // Вычислительная техника и вопросы кибернетики : сб. – М. : Изд-во МГУ, 1979. – Вып. 9. – С. 115–125.

41. Апериодические автоматы / под ред. В. Н. Варшавского. – М. : Наука, 1976. – 232 с.

42. Гаврилов, С. А. Логическое проектирование дискретных автоматов / С. А. Гаврилов, В. В. Девятов, Е. И. Пупырев. – М. : Наука, 1977. – 352 с.

43. Исьянов, В. М. О логических схемах алгоритмов второго рода / В. М. Исьянов // Синтез дискретных автоматов и управляющих устройств : сб. – М. : Наука, 1968. – С. 26–32.

44. Миллер, Р. Теория переключательных схем / Р. Миллер. – М. : Наука, 1971. – Т. II. – 304 с.

45. Янов, Ю. И. О логических схемах алгоритмов / Ю. И. Янов // Проблемы кибернетики : сб. – М. : Наука, 1958. – С. 17–26.

46. Миклошко И. Синтез параллельных алгоритмов / И. Миклошко // Алгоритмы, математическое обеспечение и архитектура многопроцессорных вычислительных систем. – М. : Наука, 1982. – С. 220–240.

47. Майоров, С. А. Структура электронных вычислительных машин / С. А. Майоров, Г. И. Новиков. – Л. : Машиностроение, 1979. – 383 с.

48. Компьютеры на СБИС : в 2 кн. / Т. Мотоока, С. Томата, Х. Таната [и др.]. – М. : Мир, 1988. – Кн. 1. – 387 с.

49. Вашкевич, Н. П. О кодировании недетерминированных автоматов / Н. П. Вашкевич // Повышение эффективности средств обработки информации на базе математического и машинного моделирования : материалы Межреспубликанской конф. – Тамбов, 1993. – С. 8–9.

50. Вашкевич, Н. П. Формализация алгоритмов управления с взаимодействующими параллельными ветвями / Н. П. Вашкевич, С. Н. Вашкевич // Вычислительная техника : межвуз. сб. науч. тр. – Пенза : Изд-во Пенз. гос. техн. ун-та, 1993. – Вып. 18. – С. 17–22.

51. Вашкевич, Н. П. Синтез автоматных моделей управления параллельными взаимодействующими процессами / Н. П. Вашкевич // Новые информационные технологии и системы : материалы Междунар. науч.-техн. конф. – Пенза, 1994. – С. 49–54.

52. Вашкевич, Н. П. Формализация взаимодействия параллельных ветвей алгоритмов управления / Н. П. Вашкевич, С. Н. Вашкевич // Повышение эффективности средств обработки информации на базе математического и машинного моделирования : материалы Межреспубликанской конф. – Тамбов, 1993. – С. 259–260.

53. Вашкевич, Н. П. Аппаратно-программная реализация микропрограммных устройств управления с параллельными ветвями / Н. П. Вашкевич // Микропроцессорные системы автоматики : сб. тез. II Всесоюз. науч.-техн. конф. – Новосибирск, 1990. – С. 106–111.

54. Вашкевич, Н. П. Микропрограммные системы управления для параллельных алгоритмов / Н. П. Вашкевич // Вопросы радиоэлектроники, сер. ЭВТ. – 1990. – Вып. 13. – С. 83–90.

55. Хоар, Ч. Взаимодействующие последовательные процессы / Ч. Хоар. – М. : МИР, 1989. – 264 с.

56. Ульман, Дж. Д. Вычислительные аспекты СБИС / Дж. Д. Ульман. – М. : Радио и связь, 1990. – 480 с.

57. Грис, Д. Конструирование компиляторов для цифровых вычислительных машин / Д. Грис. – М. : МИР, 1975. – 456 с.

58. Ахо, А. Построение и анализ вычислительных алгоритмов / А. Ахо, Дж. Хопкрофт, Дж. Ульман. – М. : МИР, 1979. – 535 с.

59. Foster, I. Designing and Building Parallel Programs : конспект лекций / I. Foster. – URL: www.mcs.anl.gov.

60. Вашкевич, Н. П. Формализация алгоритма взаимодействия параллельных процессов в задаче «читатели-писатели» / Н. П. Вашкевич, Р. А. Бикташев // Новые информационные технологии и системы : тр. V Междунар. науч.-техн. конф. – Пенза : ПГУ, 2002. – С. 204–208.

61. Шоу, А. Логическое проектирование операционных систем : пер. с англ. / А. Шоу. – М. : МИР, 1981. – 360 с.

62. Дейтел, Х. М. Операционные системы. Основы и принципы / Х. М. Дейтел, П. Дж. Дейтел, Д. Р. Чофнес. – М. : Бином-Пресс, 2011. – Ч. 1. – 1024 с.

63. Мехатроника / Т. Исии, И. Симояма, Х. Икоуэ [и др.]. – М. : МИР, 1988. – 314 с.

64. Олифер, В. Г. Сетевые операционные системы / В. Г. Олифер, Н. А. Олифер. – СПб. : Питер, 2001. – 544 с.

65. Калинин, А. Г. Универсальные языки программирования / А. Г. Калинин, И. В. Мацкевич. – М. : Радио и связь. 1991. – 400 с.

66. Гордеев, А. В. Системное программное обеспечение / А. В. Гордеев, А. Ю. Молчанов. – СПб. : Питер, 2001. – 736 с.

67. Артемьев, М. Ю. Программное обеспечение управляющих систем электросвязи / М. Ю. Артемьев. – М. : Радио и связь, 1990. – 272 с.

68. Робачевский, А. Операционная система UNIX / А. Робачевский. – СПб. : БХВ–Петербург, 2002. – 528 с.

69. Аджиев, В. Мифы безопасности ПО: Уроки знаменитых катастроф / В. Аджиев // Открытые системы. – 1998. – № 6. – С. 34–50.

70. Кейлингерт, П. Элементы операционных систем : пер. с англ. / П. Кейлингерт. – М. : МИР, 1985. – 295 с.

71. Вашкевич, Н. П. Автоматный подход к формализации алгоритмов управления взаимодействием параллельными процессами при обращении к общим переменным (общему ресурсу) для *n*-процессов / Н. П. Вашкевич // Вычислительные системы и технологии : межвуз. сб. науч. тр. – Пенза : Изд-во Пенз. гос. ун-та, 2002. – Вып. 1 (27). – С. 3–10.

72. Таненбаум, Э. Современные операционные системы / Э. Таненбаум. – 3-е изд. – СПб. : Питер, 2010. – 1120 с.

73. Ульман, Д. Основы систем баз данных / Д. Ульман. – М. : Финансы и статистика, 1983. – 334 с.

74. Вашкевич, Н. П. Автоматное представление алгоритмов управления параллельными процессами в задаче «читатели-писатели» на основе конценции недетерминизма и механизма монитора / Н. П. Вашкевич, Р. А. Бикташев // Известия вузов. Поволжский регион. Технические науки. – 2015. – № 3. – С. 65–75.

75. Соловьев, Г. Н. Операционные системы ЭВМ : учеб. пособие / Г. Н. Соловьев, В. Д. Никитин. – М. : Высш. шк., 1989. – 255 с.

76. Элементы параллельного программирования / В. А. Вальковский, В. Е. Котов, А. Г. Марчук, Н. Н. Миренков ; под ред. В. Е. Котова. – М. : Радио и связь, 1983. – 240 с.

77. Вальковский, В. А. Автоматическое построение параллельных программ / В. А. Вальковский, В. Е. Котов // Алгоритмы, математическое обеспечение и архитектура многопроцессорных вычислительных систем. – М. : Наука, 1982. – С. 170–219.

78. Вашкевич, Н. П. Проектирование параллельных алгоритмов в задачах идентификации : учеб. пособие / Н. П. Вашкевич, Е. И. Калиниченко. – Пенза : Изд-во Пенз. гос. ун-та, 2000. – 72 с.

79. Вашкевич, Н. П. Автоматная модель распределенных вычислительных систем / Н. П. Вашкевич, Е. И. Калиниченко // Новые информационные технологии и системы : тр. VI Междунар. науч.-техн. конф. – Пенза : Изд-во ПГУ, 2004. – С. 11–19.

80. Оллонгрен, А. Определение языков программирования интерпретирующими автоматами / А. Оллонгрен. – М. : Мир, 1977. – 228 с.

81. Ганькин, А. Л. Исследования и разработки в области аппаратной реализации алгоритмов операционных систем / А. Л. Ганькин, Ю. В. Сидоров, Г. Н. Соловьев // Зарубежная электроника. – 1984. – № 2. – С. 34–50.

82. Назаров, С. В. Операционные системы специализированных вычислительных комплексов: Теория построения и системного проектирования / С. В. Назаров. – М. : Машиностроение, 1989. – 400 с.

83. Майерс, Г. Архитектура современных ЭВМ : в 2 кн. : пер. с англ. / Г. Майерс. – М. : МИР, 1985. – Кн. 1. – 364 с. ; кн. 2. – 312 с.

84. Вашкевич, Н. П. Преобразование структуры управляющих алгоритмов, заданных моделью НДА для построения распределенных управляющих систем параллельной обработки / Н. П. Вашкевич // Актуальные проблемы науки и образования : тр. Междунар. юбилейного симп. : в 2 т. – Пенза : ИИЦ ПГУ, 2003. – Т. 2. – С. 375–377.

85. Вашкевич, Н. П. Достоинства формального языка, основанного на концепции недетерминизма, для функционального описания и преобразования алгоритма логического управления кризисами и процессами в параллельных системах обработки ин-

формации / Н. П. Вашкевич, Р. А. Бикташев // Телекоммуникации. – М., 2011. – № 1. – С. 18–25.

86. Вашкевич, Н. П. Модели событийных недетерминированных автоматов для формального представления основных свойств систем управления параллельными процессами и ресурсами / Н. П. Вашкевич, Р. А. Бикташев // Инфокоммуникационные технологии. – М., 2013. – № 3. – С. 40–44.

87. Вашкевич, Н. П. Формализованное описание алгоритма управления взаимодействующими параллельными процессами в задаче «производители-потребители» с использованием согласующего кольцевого буфера // Н. П. Вашкевич, Р. А. Бикташев, А. А. Тараканов // Известия вузов. Поволжский регион. Технические науки. – Пенза, 2008. – № 4. – С. 98–107.

88. Вашкевич, Н. П. Модели событийных недетерминированных автоматов представления алгоритмов управления взаимодействующими процессами в многопроцессорных вычислительных системах на основе использования механизма монитора // Н. П. Вашкевич, В. Н. Волчихин, Р. А. Бикташев // Известия высших учебных заведений. Поволжский регион. Технические науки. – Пенза, 2013 . – № 2. – С. 5–14.

89. Вашкевич, Н. П. Формализация алгоритмов управления параллельными процессами на основе использования механизма «рандеву» / Н. П. Вашкевич, Р. А. Бикташев // XXI век: Итоги прошлого и проблемы настоящего. Технические науки. Информатика. IT технология. – 2011. – № 3. – С. 111–115.

90. Вашкевич, Н. П. Основы вычислительной техники : учеб. пособие для электротехнических специальностей вузов / Н. П Вашкевич, Н. П. Сергеев. – 2-е изд., перераб. и доп. – М. : Высш. шк., 1998. – 311 с.

91. Горбатов, В. А. Логическое управление распределенными системами / В. А. Горбатов, М. И. Смирнов, И. С. Хлыбчиев. – М. : Энергоатомиздат, 1991. – 288 с.

92. Советов, Б. Я. Моделирование систем / Б. Я. Советов, С. А. Яковлев. – М. : Высш. шк., 2005. – 343 с.

93. Вашкевич, Н. П. Аппаратная реализация функций синхронизации параллельных процессов при обращении к раздельному ресурсу на основе ПЛИС / Н. П. Вашкевич, Р. А. Бикташев, Е. И. Гурин // Известия вузов. Поволжский регион. Технические науки. – Пенза, 2007. – № 2. – С. 3–12.

94. Вашкевич, Н. П. Синхронно-асинхронные модели и верификация замкнутых систем «управление – оборудование» / Н. П. Вашкевич, В. И. Волчихин, В. Н. Дубинин // Надежность и качество : тр. Междунар. науч.-техн. конф. – Пенза, 2014. – С. 21–26.

95. Вашкевич, Н. П. Преобразование временной темпоральной логики в стандартную форму в виде систем канонических уравнений / Н. П. Вашкевич, А. А. Тараканов // Вопросы радиоэлектроники, серия ЭВТ. – 2007. – Вып. 1 . – С. 69–76.

96. Вашкевич, Н. П. Формализованное описание и верификация дискретных событийных систем с параллельными процессами / Н. П. Вашкевич, В. Н. Дубинин // Вопросы радиоэлектроники, серия ЭВТ. – 2008. – Вып. 5. – С. 51–56.

97. Эндрюс, Г. Р. Основы многопоточного и распределенного программирования : пер. с англ. / Г. Р. Эндрюс. – М. : Вильямс, 2003. – 512 с.

98. Brinch, Hansen. The Programming Language Concurrent Pascal / Hansen Brinch // On Software Engineering. – 1975. – June. – Vol. SE-1. – P. 199–207.

99. Foster, Ian. Designing and Building Parallel Programs / Ian Foster. – 1995. – URL: http://www.mcs.anl.gov/~itf/dbpp/text/book.html

100. Saglam, Bilge E. System-on-a-chip processor synchronization support in hardware / Bilge E. Saglam, Vincent J. Mooney III // Proceedings of the conference on Design, automation and test in Europe, IEEE Press Piscataway. – NJ, USA, 2001. – P. 633–641.

101. Anderson, T. E. The performance of spin lock alternatives for shared-memory multiprocessors / T. E. Anderson // IEEE Trans. Parallel Distrb. Syst. 1. – 1990. – № 1, Jan. – P. 6–16.

102. Inter-Process Communication using Pipes in FPGA-based Adaptive Computing / Ming Liu, Zhonghai Lu, Wolfgang Kuehn, Axel Jantsch // ISVLSI'10. – Kefalonia, Greece, 2010. – P. 80–85.

103. Kuacharoen, P. A configurable hardware scheduler for Real-Time systems / P. Kuacharoen, M. A. Shalan, Vincent. J. Mooney III // In Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms. – 2003. – P. 96–101.

104. Вашкевич, Н. П. Недетерминированные автоматы в проектировании систем параллельной обработки : учеб. пособие / Н. П. Вашкевич. – Пенза : Изд-во Пенз. гос. ун-та, 2004. – 280 с.

105. Вашкевич, Н. П. Достоинства формального языка, основанного на концепции недетерминизма, для структурной реализации параллельных систем логического управления процессами

и ресурсами / Н. П. Вашкевич, Р. А. Бикташев // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2011. – № 1. – С. 4–14.

106. McVoy, L. Lmbench: Portable Tools for Performance Analysis / L. McVoy, C. Staelin // Proceedings of the 1996 Winter Technical Conference. – San Diego, Calif., 1996. – P. 279–294.

107. Вашкевич, Н. П. Верификация алгоритмов логического управления, представленных автоматными моделями / Н. П. Вашкевич // Вопросы радиоэлектроники. – 2009. – Т. 4, № 4. – С. 42–53.

108. Бикташев, Р. А. Модели оценки производительности средств синхронизации параллельных процессов / Р. А. Бикташев // Вопросы радиоэлектроники, серия ЭВТ. – 2010. – Вып. 5. – С. 21–29.

109. Вашкевич, Н. П. Планировщик задач с аппаратной поддержкой для многопроцессорных систем / Н. П. Вашкевич, В. И. Волчихин, Р. А. Бикташев // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2012. – № 1. – С. 3–11.

110. Вашкевич, Н. П. Аппаратная поддержка диспетчера задач с глобальной очередью в многопроцессорных системах / Н. П. Вашкевич, Р. А. Бикташев, А. И. Меркурьев // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2011. – № 3. – С. 3–12.

111. Бикташев, Р. А. Математическое моделирование диспетчеров задач со стратегией разделения пространства для параллельных вычислительных систем на основе разомкнутых сетей массового обслуживания / Р. А. Бикташев, А. И. Мартышкин // Технические науки – от теории к практике : сб. ст. по материалам XXVI Междунар. науч.-практ. конф. – Новосибирск : СибАК, 2013. – № 9 (22). – С. 36–42.

112. Воеводин, В. В. Параллельные вычисления / В. В. Воеводин, Вл. В. Воеводин. – СПб. : БХВ-Петербург, 2002. – 599 с.

113. Программирование на параллельных вычислительных системах ; пер. с англ. / под ред. Р. Боба II. – М. : Мир, 1991. – 372 с.

114. Иртегов, Д. В. Введение в операционные системы / Д. В. Иртегов. – СПб. : БХВ-Петербург, 2002. – 614 с.

115. Stotts, D. Model checking cobweb protocols for verification of HTML frames behavior / D. Stotts, J. Navon // Proc. 11th Int. Conf. on World Wide Web (WWW'02). – Honolulu, Hawaii, 2002. – P. 182–190.

116. SMV. Symbolic Model Verifier. – URL: http://www.cs.cmu.edu/~modelcheck/smv.html

117. Зотов, В. Ю. Проектирование цифровых устройств на основе ПЛИС фирмы Xilinx в САПР WebPACK ISE / В. Ю. Зотов. – М. : Горячая линия – Телеком, 2003. – 624 с.

118. Kohout, P. Hardware support for realtime operating systems / P. Kohout, B. Ganesh, B. Jacob // In Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis. – Newport Beach, CA, USA: ACM, 2003. – P. 45–51.

119. Шалыто, А. А. SWITCH-технология. Алгоритмизация и программирование задач логического управления / А. А. Шалыто. – СПб. : Наука, 1998. – 628 с.

120. Бикташев, Р. А. Аппаратная поддержка диспетчеризации задач в многопроцессорных системах / Р. А. Бикташев, А. И. Меркурьев // Современные методы и средства обработки пространственно-временных сигналов : сб. ст. VIII Всерос. науч.-техн. конф. – Пенза : Приволжский Дом знаний, 2010. – С. 112.

121. Вашкевич, Н. П. Формализация алгоритма синхронизации процессов при диспетчеризации задач в многопроцессорных системах с использованием механизма рандеву / Н. П. Вашкевич, Р. А. Бикташев // Информационные технологии. – 2009. – № 12 (160). – С. 12–17.

122. Кузьмин, Е. В. Моделирование, спецификация и верификация «автоматных» программ / Е. В. Кузьмин, В. А. Соколов // Программирование. – 2008. – № 1. – С. 38–60.

123. Moonmo, Koo. Hardware implementation of interprocessor communication in MPSoCs for multimedia applications / Koo Moonmo, Chae Soo-Ik // School of Electrical Engineering and Computer Sciences Seoul National University. – Seoul, Korea, 2007.

124. Minimizing the runtime partial reconfiguration overheads in reconfigurable systems / M. Liu, R. N. Pittman, A. Forin, J.-L. Gaudiot // Spinger Science+Business Media, 22 July 2011. – P. 894–911.

125. Jensen, K. Coloured Petri Nets: modelling and validation of concurrent systems / K. Jensen, L. Kristensen. – Springer-Verlag, 2009.

126. Бикташев, Р. А. Комплекс программ для измерения производительности функций операционных систем / Р. А. Бикташев, А. И. Мартышкин // XXI век: Итоги прошлого и проблемы настоящего плюс. Технические науки. Информатика, IT технологии. – 2013. – № 3. – С. 47–54.

127. Бикташев, Р. А. Реализация устройства аппаратной поддержки диспетчеризации задач для многопроцессорной системы на ПЛИС / Р. А. Бикташев, С. К. Шестаков // Оптико-электронные приборы и устройства в системах распознавания образов, обработки изображений и символьной информации. Распознавание. – 2013 : сб. ст. XI Междунар. науч.-техн. конф. – Курск : ЮЗГТУ, 2013. – С. 220–223.

128. Стивенс, У. UNIX: взаимодействие процессов / У. Стивенс. – СПб. : Питер, 2003. – 576 с.

129. Baker, T. P. A comparison of global and partitioned EDF schedulability tests for multiprocessors / T. P. Baker. – Florida State University, Tech. Rep. TR-051101, 2005.

130. The Performance and Energy Consumption of Embedded Real-Time Operating Systems / K. Baynes, C. Collins, E. Fiterman, B. Ganesh, P. Kohout, C. Smit, L. Zong, B. Jacob // IEEE Transactions on Computers. – 2003. – November. – Vol. 52. – № 11. – P. 1454–1469.

131. User-Level Interprocess Communication for Shared Memory Multiprocessors / B. N. Bershad, T. E. Anderson, E. D. Lazowska, H. M. Levy // ACM Trans. Comput. Syst. – 1991. – № 9 (2). – P. 175–198.

132. Wang, L. Measurement and Control of Linux Operating System Noise on the Godson-3A Shared-Memory Multicore Platform / L. Wang, Y. Zhao, Z. Wang // IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC). – Zhangjiajie, Hunan Province, P. R. China. – 2013. – № 13–15 Nov. – P. 474–481.

133. Xilinx, Microblaze processor reference guide – ug081 (v11. 2). – 2011. – May. – URL: http://www.xilinx.com/support/documentation/sw_ manuals/xilinx12_3/mb_ref_guide.pdf

134. ISE Simulator (ISim). – URL: www.xilinx.com/tools/isim.htm

135. Механов, В. Б. Построение сети Петри, моделирующей поведение недетерминированного конечного автомата / В. Б. Механов, Н. Н. Коннов, Е. А. Кизилов // Телематика'2012 : тр. XIX Всерос. науч.-метод. конф. – СПб. : ИТМО, 2012. – С. 330–332.

136. Вельдер, С. Э. О верификации автоматных программ на основе метода Model checking / С. Э. Вельдер, А. А. Шалыто // Информационно-управляющие системы. – 2007. – № 3. – С. 27–38.

137. Котов, В. Е. Сети Петри / В. Е. Котов. – М. : Наука, 1984. – 160 с.

138. Minas, M. Visual PLC-Programming using Signal Interpreted Petri Nets / M. Minas, G. Frey // Proc. American Control Conference 2002 (ACC 2002). – Anchorage, Alaska. – P. 5019–5024.

139. Bogunovic, N. Verification of mutual exclusion algorithms with SMV System / N. Bogunovic, E. Pek // EUROCON 2003. Computer as a Tool. The IEEE Region 8. – 2003. – Vol. 2. – P. 21–25.

140. Верификация автоматных программ / С. Э. Вельдер, М. А. Лукин, А. А. Шалыто, Б. Р. Яминов. – СПб. : СПбГУ: ИТМО, 2011. – 242 с.

141. Шалыто, А. А. Синхронное программирование / А. А. Шалыто, Д. Г. Шопырин // Информационно-управляющие системы. – 2004. – № 3. – С. 35–42.

142. Зайцев, Д. А. Композиционный анализ сетей Петри / Д. А. Зайцев // Кибернетика и системный анализ. – 2006. – № 1. – С. 143–154.

143. Дьяконов, В. MATLAB 6 : учеб. курс / В. Дьяконов. – СПб. : Питер, 2001. – 592 с.

144. Свидетельство о государственной регистрации программ для ЭВМ № 2015610759, 2015. Система верификации алгоритмов управления, представленных на языке НДА / В. В. Кутузов, Н. П. Вашкевич, Р. А. Бикташев, Д. В. Пащенко.

145. Stateflow. Model and simulate decision logic using state machines and flow charts. – URL: http://www.mathworks.com/products/stateflow/

ПРИЛОЖЕНИЕ

Основные понятия и определения из теории конечных цифровых автоматов

Все многообразие элементов, узлов, блоков и устройств, из которых состоит любая ЭВМ, является примером различных типов той или иной степени сложности преобразователей цифровой информации – цифровых автоматов. Методы теории цифровых автоматов, являющихся математической моделью цифровых (дискретных) устройств, используются в качестве теоретической базы для анализа и синтеза различных цифровых узлов и устройств ЭВМ. Так как при изучении конечных цифровых автоматов имеют дело с математическими моделями, то применение основных положений теории конечных цифровых автоматов не ограничивается конкретной областью, например ЭВМ, но может быть использовано для анализа и синтеза различных автоматических устройств во многих областях науки и техники. Рассмотрим лишь некоторые сведения из теории конечных цифровых автоматов, которые будут полезны как для понимания принципов действия, так и построения отдельных узлов, блоков и устройств ЦВМ и систем промышленной автоматики.

П1. Определение цифрового автомата

Под цифровым автоматом понимается устройство, предназначенное для преобразования цифровой (дискретной) информации, способное переходить под воздействием входных сигналов из одного состояния в другое и выдавать выходные сигналы.

Отличительной особенностью цифровых автоматов является то, что они имеют дискретное множество внутренних состояний и переход из одного состояния в другое осуществляется скачкообразно.

Дискретность информации, преобразуемой в автомате, проявляется в том, что она представляется посредством набора слов конечной длины в некотором алфавите. В частности, в двоичном алфавите, как принято в ЭВМ, слова представляются в виде цепочки, состоящей из нулей и единиц. Элементарные сигналы, составляющие двоичный алфавит, физически чаще всего представляются импульсами напряжения (тока) или уровнями напря-

жения (тока). При этом единица может быть представлена наличием импульса или высоким уровнем напряжения, а нуль – отсутствием импульса или низким уровнем напряжения.

Реальные цифровые автоматы называют конечными, если множество входных и выходных сигналов, а также множество внутренних состояний автомата конечны. Примерами таких автоматов может служить все многообразие различной степени сложности средств цифровой вычислительной техники, предназначенных для преобразования цифровой информации, включая элементы, узлы, блоки и устройства ЭВМ и систем.

Так как цифровые автоматы используются в качестве математической модели цифровых устройств, то методы теории цифровых автоматов могут быть использованы в качестве теоретической базы не только для анализа и синтеза цифровых узлов и устройств ЭВМ и систем, но и для анализа и синтеза различных автоматических устройств во многих областях науки и техники. Кроме того, так как значительная область теории цифровых автоматов является частью теории алгоритмов, то основные ее положения полезно использовать также при создании программных средств [6].

Цифровые автоматы функционируют в дискретные моменты времени, временной интервал T между которыми называют тактом. В зависимости от того, чем определяется время T, различают автоматы синхронного и асинхронного действия.

Для цифрового автомата ***синхронного действия*** входные сигналы действуют в строго определенные моменты времени при $T = \text{const}$, определяемые генератором синхронизирующих импульсов, в которые возможен переход автомата из одного состояния в другое.

Для цифрового автомата ***асинхронного действия*** $T \neq \text{const}$ и определяется моментами поступления входных сигналов, а переход автомата из одного состояния в другое осуществляется при неизменном состоянии входа.

Для идеализированных цифровых автоматов, когда не учитываются переходные процессы в элементах схем автомата, разница в фактических величинах T для правильного функционирования автомата не имеет значения, поэтому для описания законов функционирования цифровых автоматов вводят абстрактное автоматное время, принимающее целые неотрицательные значения $t = 0, 1, 2,....$

По степени детализации описания произвольных цифровых автоматов различают абстрактные и структурные автоматы. В соответствии с этой классификацией различают абстрактную и структурную теорию цифровых автоматов [90].

Абстрактные цифровые автоматы рассматриваются как «черный ящик», имеющий один вход и один выход, т.е. при рассмотрении таких автоматов отвлекаются от структуры как самого цифрового автомата, так и его входных $Z(t)$ и выходных $W(t)$ сигналов.

Для задания абстрактного автомата необходимо знать три алфавита: входной $Z = [Z_0, Z_1, \ldots, Z_F]$, выходной $W = [W_0, W_1, \ldots, W_G]$, и внутренних состояний $A = [a_0, a_1, \ldots, a_M]$. Тогда закон функционирования абстрактного цифрового автомата, его «поведение» может быть задано рекуррентными соотношениями вида

$$\begin{aligned} a(t+1) &= \delta\left[a(t), z(t)\right]; \\ W(t) &= \lambda\left[a(t), z(t)\right];\ a(0) = a_0, \end{aligned} \qquad (\text{П1.1})$$

где $\delta(a, z)$ – функция переходов автомата; $\lambda(a, z)$ – функция выходов автомата; a_0 – начальное состояние автомата.

Отметим, что автомат, у которого начальное состояние остается неизменным при любых экспериментах с ним, называют ***инициальным***.

Автомат функционирует следующим образом. В каждый момент дискретного времени он находится в некотором состоянии $a(t)$. Воспринимая в некоторый момент времени t входной сигнал $z(t)$, автомат выдает некоторый выходной сигнал $w(t)$ и осуществляет переход в новое состояние $a(t + 1)$. В начальный момент времени при $t = 0$ автомат находится в начальном состоянии $a(0) = a_0$. Таким образом автомат в соответствии с законом функционирования преобразует последовательность входных сигналов (входное слово p) $p = z(0), z(1), ..., z(m)$ в однозначно определенную последовательность выходных сигналов $q = w(0), w(1), ..., w(m)$ той же длины (выходное слово q). Такое преобразование называют также отображением φ множества слов входного алфавита во множество слов выходного алфавита: $q = \varphi(p)$. В литературе, например в [1], это отображение принято называть оператором. Исходя из изложенного, можно отметить, что если два абстрактных автомата с общим входным и выходным алфавитом индуци-

руют одно и то же отображение множества слов во входном алфавите во множество слов в выходном алфавите, то такие автоматы эквивалентны.

Структурный цифровой автомат в отличие от абстрактного является продуктом его дальнейшей детализации, когда рассматривается как его внутренняя структура, так и структура входных и выходных сигналов. Это означает, что в теории таких автоматов изучаются методы построения автоматов из элементарных автоматов, способы кодирования внутренних состояний автомата, которые являются его памятью, а также кодирования входных и выходных сигналов. При этом каждому внутреннему состоянию абстрактного автомата ставится в соответствие своя комбинация состояний элементарных автоматов, имеющих два внутренних состояния, а каждому входному (выходному) сигналу абстрактного автомата из алфавита $Z(W)$ – своя комбинация значений элементарных двузначных сигналов из структурного входного (выходного) алфавита $X = [x_1, x_2, \ldots, x_L]$ ($Y = [y_1, y_2, \ldots, y_N]$), одновременно подаваемых по входным (выходным) реальным физическим каналам.

П2. Варианты цифровых автоматов

Введенное понятие цифрового автомата является достаточно общим. Рассмотрим некоторые частные случаи автоматов, получаемые путем наложения тех или иных ограничений на компоненты A, Z, W, δ, λ, которые задают автомат.

Автомат Мили. Произвольные цифровые автоматы, закон функционирования которых задается уравнениями типа (П1.1), называют автоматами Мили.

Автомат Мура. Произвольные цифровые автомата, у которых наложено ограничение на функцию выходов, заключающееся в том, что выходной сигнал зависит только от состояния автомата и не зависит от значения входных сигналов, называют автоматами Мура. Для таких автоматов уравнения (П1.1) преобразуются в форму

$$\begin{aligned} a(t+1) &= \delta\left[a(t), z(t)\right]; \\ W(t) &= \mu\left[a(t)\right];\ a(0) = a_0, \end{aligned} \qquad \text{(П2.1)}$$

где $\mu\left[a(t)\right]$ называют сдвинутой функцией.

В дальнейшем будет рассмотрена методика преобразования автомата Мили в эквивалентный ему автомат Мура и наоборот. Отметим важное различие в функционировании этих автоматов. Выходные сигналы автомата Мура отстают на один такт от выходных сигналов автомата Мили, эквивалентного ему. Это различие позволяет простым способом согласовать работу автомата и внешней среды или работу двух автоматов.

Совмещенная модель автомата (С-автомат). В некоторых практических случаях удобнее использовать совмещенную модель автомата (С-автомат). Под С-автоматом понимается математическая модель цифрового устройства, определяемая как совокупность восьми объектов: A, Z, W, U, δ, λ_1, λ_2, a_0, где $U = [u_1, u_2, \ldots, u_H]$ – выходной алфавит автомата Мура, λ_1 и λ_2 – функция выходов автомата Мили и автомата Мура, соответственно.

Поведение С-автомата описывается соотношениями:

$$a(t+1) = \delta[a(t), z(t)];$$

$$W(t) = \lambda_1[a(t), z(t)]; \qquad (П2.2)$$

$$U(t) = \lambda_2[a(t)].$$

Очевидно, что от С-автомата легко перейти к автоматам Мили и Мура с учетом возможных сдвигов выходных сигналов на один такт, аналогично тому, как возможен переход от автомата Мили к автомату Мура и наоборот. Практически много реальных автоматов работает именно по модели С-автомата.

Автомат без памяти (комбинационный автомат). Алфавит состояний такого автомата содержит единственную букву, поэтому понятие функции переходов вырождается и становится ненужным для описания работы автомата. Автомат задается тремя объектами: Z, W, λ. Функция выходов принимает вид

$$W(t) = \lambda[z(t)], \qquad (П2.3)$$

т.е. выходной сигнал в данном такте зависит только от входного сигнала того же такта и никак не зависит от ранее принятых сигналов.

Такое преобразование выполняется комбинационной логической схемой, в которой каждой сходной комбинации соответствует определенная выходная комбинация. Описание работы таких схем осуществляется с использованием таблиц истинности и булевых функций.

Автономный автомат [2]. В таком автомате входной алфавит состоит из одной буквы. Автомат задается четырьмя объектами: A, W, δ, λ, с возможным выделением начального состояния a_0. Функции переходов и выходов имеют вид

$$\begin{aligned} a(t+1) &= \delta[a(t)]; \\ W(t) &= \lambda[a(t)]. \end{aligned} \qquad (П2.4)$$

Эта пара выражений однозначно определяет единственную выходную последовательность, выдаваемую автоматом, и последовательность состояний

$$\begin{aligned} &w(0)w(1)w(2)..., \\ &a(0)a(1)a(2)... \end{aligned} \qquad (П2.5)$$

Если автономный автомат конечен и число его состояний равно k, то среди значений $a(0)$, $a(1)$, ..., $a(k)$ найдутся повторяющиеся состояния. Следовательно, каждая из последовательностей (П2.5) окажется периодической с длиной периода не больше числа состояний автомата и, возможно, с некоторым предпериодом. Автономные автоматы используются для построения генераторов периодических последовательностей, генераторов синхросерий и в других задающих устройствах, применяемых в цифровой технике.

Автомат без выхода. В таком автомате выходной алфавит содержит только одну букву [2]. Автомат задается тремя объектами: A, Z, δ. Из функций, задающих поведение автомата, сохраняется лишь функция переходов

$$a(t+1) = \delta[a(t), z(t)]. \qquad (П2.6)$$

Поведение автомата без выхода [2] не может быть охарактеризовано в терминах операторов (отображений), перерабатывающих входные слова в выходные. Вместо этого можно было бы рассматривать операторы, перерабатывающие входные последовательности в последовательности внутренних состояний. Однако удобнее рассматривать поведение такого автомата, задавая настройку его путем выделения начального состояния и множества финальных состояний. После такой настройки автомат может рассматриваться как устройство, воспринимающее вопросы (входные последовательности) и выдающее ответы «да» или «нет» в зависимости от того, принадлежит ли слово интересующему нас языку или нет. Если после подачи на вход слова автомат окажется в одном из финальных состояний, то ответ утвердителен, в про-

тивном случае – отрицателен. В лингвистической интерпретации вопросы, задаваемые автомату, приобретают следующий смысл: является ли данная цепочка символов грамматически правильным предложением в рассматриваемом языке или нет. В [2] приводится и другой вариант понятия поведения автомата без выхода, в котором осуществляется так называемая макронастройка, заключающаяся в выделении начального состояния и предельных множеств состояний.

Если поведение автомата без памяти и автономного автомата сравнительно просто, то поведение автомата без выхода оказывается столь же разнообразно, как и поведение автомата с выходом. Это следует из того, что из двух функций, задающих поведение автомата, функция переходов является наиболее информативной.

Управляющие и операционные автоматы. По функциональному назначению цифровые автоматы можно подразделить на два класса: управляющие и операционные. Такое подразделение отражает структуру и функции любого устройства ЭВМ, предназначенного для обработки цифровой информации и называемого операционным устройством. Каждое такое операционное устройство можно представить моделью В.М. Глушкова, состоящей из двух тесно взаимодействующих между собой блоков (рис. П2.1), один из которых выполняет функции операционного автомата (ОА), а другой – управляющего автомата (УА).

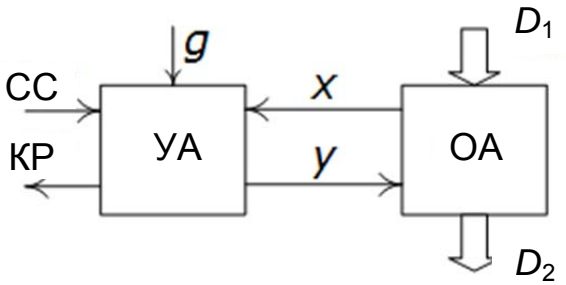


Рис. П2.1. Обобщенная структура операционного устройства

На рис. П2.1 представлена обобщенная структура произвольного операционного устройства, где D_1 и D_2 – шины, по которым поступает исходная информация и результаты ее обработки, соответственно: X – шины, по которым поступают сигналы, характеризующие состояние ОА (например, отрицательный результат, переполнение сумматора и т.д.), эти сигналы часто называют осведомительными; Y – шины, по которым поступают

управляющие сигналы из УА на ОА в соответствии с алгоритмом выполняемой в ОА операции; g – шины, по которым поступают сигналы, определяющие выполняемую операцию; СС – стартовый сигнал пуска операционного устройства; КР – сигнал, характеризующий конец работы алгоритма. Таким образом, можно отметить, что ОА реализует действия над исходной информацией (словами), т.е. является исполнительной частью операционного устройства, а управляющий автомат управляет работой ОА, т.е. вырабатывает необходимые последовательности управляющих сигналов в соответствии с алгоритмом выполняемой операции.

Управляющие автоматы используются не только в операционных устройствах вычислительной техники в системе УА-ОА, но и в разнообразных системах автоматики по управлению технологическими процессами и объектами, которые обобщенно можно назвать объектами управления (ОУ). В этом случае УА, в соответствии с алгоритмом функционирования ОУ и в зависимости от значений осведомительных сигналов о состоянии ОУ, поступающих от датчиков ОУ, и возможных внешних сигналов, вырабатывает и подает на исполнительные механизмы ОУ последовательность управляющих сигналов, обеспечивающих выполнение операций или процедур, предусмотренных целями управления. Для сложных систем управления, когда требуется выполнять большой объем работ по обработке информации о состоянии ОУ и выработке управляющих воздействий на ОУ в зависимости от целевой функции управления, в качестве УА могут выступать ЭВМ соответствующего класса.

Микропрограммные автоматы [2]. Управляющий автомат, реализующий микропрограмму выполнения операции обработки информации, часто называют микропрограммным автоматом (МПА). Это вытекает из следующей интерпретации взаимодействия ОА и УА в процессе выполнения каких-либо операций по обработке информации:

а) любая операция, реализуемая операционным устройством, рассматривается в виде последовательности элементарных неделимых актов обработки информации, выполняемых в течение одного такта автоматного времени (одного шага алгоритма), называемых ***микрооперациями***. Множество микроопераций и сигналов, инициирующих их выполнение, обозначают одними и теми же символами, которые составляют выходной структурный алфавит УА $Y = [y_1, y_2, \ldots, y_N]$. В случае, если несколько микроопераций

реализуются в ОА одновременно, это подмножество микроопераций называют ***микрокомандой***;

б) для управления порядком следования микрокоманд используются логические условия (ЛУ), которые в зависимости от результатов преобразования информации в ОА могут принимать значения 1 или 0. Множество ЛУ (осведомительных сигналов) будем обозначать символами, которые составляют входной структурный алфавит $X = [x_1, x_2, \ldots, x_L]$;

в) последовательность выполнения микрокоманд, определяемая функциями перехода УА, записывается в виде алгоритма, представленного в терминах микроопераций и ЛУ и называемого ***микропрограммой***. В качестве начального языка для описания микропрограмм выполнения операций чаще всего используется язык операторных схем алгоритмов ГСА и ЛСА, который рассматривается в [2, разделы 3.2–3.4].

Микропрограммные автоматы с жесткой и программируемой логикой [90]. Управляющие микропрограммные автоматы аппаратурно могут быть реализованы на основе так называемой жесткой логики и программируемой логики.

Автомат с жесткой логикой строится на базе использования логических элементов (ЛЭ) и элементов памяти (элементарных автоматов с двумя внутренними состояниями). Изменить алгоритм работы такого автомата нельзя, не изменяя соединений между элементами. Для таких автоматов характерны высокое быстродействие, определяемое только задержками используемых ЛЭ и элементов памяти, пропорциональный рост объема оборудования в зависимости от сложности реализуемого алгоритма и малые удельные затраты оборудования при реализации простых микропрограмм. Однако автоматы с жесткой логикой не обладают гибкостью при внесении изменений в алгоритм их функционирования, необходимость в которых особенно часто возникает в процессе проектирования цифровых устройств.

Для автомата с программируемой логикой алгоритм работы записывается в управляющую память в виде микропрограммы, состоящей из микрокоманд. Микрокоманда содержит информацию о микрооперациях, которые должны выполняться в данном такте работы устройства, и об адресе в управляющей памяти следующей микрокоманды, которая будет выполняться в следующем такте. Такие автоматы отличаются большой регулярностью структуры и

возможностью оперативного внесения изменений в алгоритм работы проектируемого устройства.

Примеры структурной реализации автоматов с жесткой и программируемой логикой представлены в [90].

П3. Способы задания цифровых автоматов

В зависимости от способа задания функций переходов и выходов (δ и λ) можно выделить два класса языков для описания функционирования цифровых автоматов: начальные языки и стандартные, или автоматные [2]. На начальных языках автомат описывается на поведенческом уровне, т.е. когда функция переходов и выходов, как правило, в явном виде не задана. Поведение автомата описывается в терминах входных и выходных последовательностей, реализуемых оператором (отображений), или управляющих последовательностей сигналов, воздействующих на операционный автомат.

Из начальных языков, предназначенных для описания функционирования абстрактных цифровых автоматов, можно отметить язык регулярных выражений алгебры событий, язык исчисления предикатов, язык логических схем алгоритмов (ЛСА) и наиболее часто используемый язык граф-схем алгоритмов (ГСА), а также языки ГСАП и ЛСАП, предназначенные для описания параллельных алгоритмических процессов. Структура начальных языков, их использование для описания поведения автоматов, преобразование описания функционирования автомата на начальном языке к описанию на одном из стандартных языков будут рассмотрены в следующих разделах. В этом разделе рассмотрим наиболее известные стандартные или автоматные языки.

При использовании автоматных языков поведение автомата задается путем задания функций переходов и выходов в явном виде. Наиболее распространенными автоматными языками являются таблицы, графы, матрицы переходов и выходов, их аналитические интерпретации.

Таблица переходов и выходов представляет собой таблицу с двойным входом, строки которой занумерованы входными буквами, а столбцы – состояниями. На пересечении указывается состояние, в которое переходит автомат (в таблице переходов) или выходной сигнал, выдаваемый им при переходе (в таблице выходов). Часто эти таблицы совмещают в одну. Этот способ описания

отображениями δ и λ иллюстрируется примером в табл. П 3.1, которая является совмещенной таблицей переходов и выходов автомата Мили.

Таблица П3.1

z \ a	a_0	a_1	a_2	a_3
z_1	a_1 / w_1	a_2 / w_2	a_3 / w_1	a_0 / w_1
z_2	a_2 / w_2	a_1 / w_3	a_0 / w_1	a_2 / w_1

Для автомата Мура таблица переходов и выходов преобразуется в так называемую отмеченную таблицу переходов, когда над каждым состоянием a_j автомата, обозначающим столбец таблицы, стоит соответствующий этому состоянию выходной сигнал.

Одним из преимуществ этого способа задания автомата является то, что любая таблица переходов и выходов задает конечный автомат. При этом должны удовлетворяться два условия автоматности отображения:

а) условие однозначности (детерминированности), которое означает, что для любой пары $a_j z_k$ задано единственное состояние перехода a_j и единственный выходной сигнал w_g, выдаваемый при переходе;

б) условие полной определенности, которое означает, что для всех возможных пар $a_i z_k$ всегда указано состояние и выходной сигнал.

Направленный граф, используемый для задания произвольного абстрактного автомата, представляет собой комбинацию вершин, изображаемых на рисунках кружочками, и соединяющих их стрелок – ребер графа. Вершины, графа отождествляют с состояниями автомата. При этом, если из состояния a_i под действием входного сигнала z_k автомат переходит в состояние a_j и выдает выходной сигнал w_g, то состояния a_i и a_j соединяются ребром со стрелкой, направленной от a_i к a_j, и отмечаются парой z_k / w_g. На рис. П3.1 приведен пример графа для автомата Мили, заданного табл. П3.1. Применительно к графу условия автоматности отображения будут заключаться в следующем:

а) не существует двух ребер с одинаковыми входными пометками, выходящих из одной и той же вершины (условие однозначности);

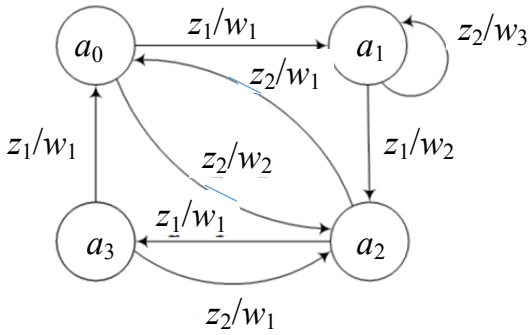


Рис. П3.1. Граф автомата

б) для всякой вершины a_i и для всякого входного сигнала z_k имеется такое ребро, помеченное символом z_k, которое выходит из a_i (условие полноты).

В дальнейшем будут рассматриваться такие обобщения абстрактного цифрового автомата, для которых не выполняются условия однозначности. Такие автоматы называют недетерминированными, они подробно рассматривались в гл. 2 монографии. В данном разделе рассмотрим такие обобщенные понятия абстрактного автомата, для которых не выполняется условие полной определенности. Такие автоматы называют ***частичными*** или ***не полностью определенными автоматами***. Для таких автоматов в местах таблиц переходов и выходов, в которых изображаемые ими функции не определены, ставятся черточки. При синтезе структурных схем автоматов, для которых исходное отображение задано частично, автомат всегда доопределяется до полного. При этом предполагается, что если не определен переход для какой-либо пары, например a_iz_k, и автомат находится в состоянии a_i, то на его вход никогда не поступает сигнал z_k и при доопределении этот переход может определяться произвольно. Если автомат полностью определен и детерминирован, то все клетки таблицы переходов и выходы заполнены и в каждой клетке записаны ровно одно состояние и один выходной сигнал.

Матрицы переходов и выходов, которые в некоторых случаях используются для задания автомата, представляют собой таблицу с двумя входами. Строки и столбцы таблицы отмечены состояниями. Если существует переход из a_i под действием сигнала z_k в a_j с выдачей w_g, то на пересечении строки a_i и столбца a_j записывается пара z_kw_g. Ясно, что не всякая матрица задает автомат. Как граф и таблица переходов и выходов, она должна удовлетворять условиям автоматности.

Система канонических уравнений (***СКУ***) ***и система выходных функций*** (СВФ) являются аналитической интерпретацией таблиц переходов и выходов или графов автоматов. Каждое состояние цифрового автомата интерпретируется как событие, соответствующее множеству переходов в это состояние.

Для автомата Мили, заданного табл. П3.1, получим СКУ для функций переходов состояний и СВФ для функций выходов, имеющих вид (П3.1) и (П3.2), соответственно:

$$\begin{aligned} a_0(t+1) &= z_1(t)\,\&\,a_3(t) \vee z_2(t)\,\&\,a_2(t); \\ a_1(t+1) &= z_1(t)\,\&\,a_0(t) \vee z_2(t)\,\&\,a_1(t); \\ a_2(t+1) &= z_2(t)\,\&\,a_0(t) \vee z_2(t)\,\&\,a_3(t) \vee z_1(t)\,\&\,a_1(t); \\ a_3(t+1) &= z_1(t)\,\&\,a_2(t). \end{aligned} \qquad \text{(П3.1)}$$

$$\begin{aligned} w_1(t) &= z_1(t)\,\&\,[a_0(t) \vee a_2(t) \vee a_3(t)] \vee z_2(t)\,\&\,[a_2(t) \vee a_3(t)]; \\ w_2(t) &= z_1(t)\,\&\,a_1(t) \vee z_2(t)\,\&\,a_0(t); \\ w_3(t) &= z_2(t)\,\&\,a_1(t). \end{aligned} \qquad \text{(П3.2)}$$

Примечание. В дальнейшем для сокращения записи, где это позволяет контекст, будем опускать знаки конъюнкции и времени t в правой части уравнений типа (П3.1) и (П3.2).

Прямая (***или обратная***) ***таблица переходов*** (***список переходов***) интерпретирует граф автомата, заданный в виде списка переходов. В прямой таблице переходов последовательно перечисляются все переходы сначала из нулевого состояния, затем из первого и т.д. [2]. В таблице предусматриваются четыре столбца: $a_i(t)$ – исходное состояние, $a_j(t+1)$ – состояние перехода, $x_{i,j}(t)$ – входной сигнал на переходе от a_i к a_j, $y_{i,j}$ – выходной сигнал на том же переходе для автомата Мили или выходной сигнал y_j, отмечающий состояние a_j для автомата Мура.

Для обратной таблицы переходов сначала записываются все переходы в нулевое состояние, затем в первое и т.д. Такой способ задания цифровых автоматов отличается, с одной стороны, наглядностью представления функционирования автомата, с другой – обеспечивает определенные удобства для ввода в ЭВМ информации при автоматизации этапов синтеза автомата. Это особенно важно при синтезе реальных практических автоматов с большим числом состояний и большим числом элементарных входных (выходных) сигналов, когда практически невозможно

использование классических таблиц переходов, выходов и графов для абстрактных автоматов.

Учитывая, что для таких реальных автоматов переходы из одного состояния в другое могут зависеть либо от всех, либо от части элементарных входных сигналов, а также то обстоятельство, что эта зависимость переходов от элементарных входных сигналов на разных шагах работы алгоритма функционирования автомата может быть различной, в качестве входного (выходного) алфавита целесообразно использовать множество элементарных входных (выходных) сигналов, которое называют также структурным входным (выходным) алфавитом [7]:

$$X = [x_1, x_2, \ldots, x_L],\ Y = [y_1, y_2, \ldots, y_N].$$

В связи с этим под входным сигналом $X_{i,j}$ на переходе от состояния a_i к a_j понимается комбинация (конъюнкция) элементарных входных сигналов из множества [X], действующих на этом переходе. При этом входной сигнал $X_{i,j}$ в дальнейшем будем называть частным входным сигналом, если в $X_{i,j}$ входят не все элементарные входные сигналы из множества [X]. В том случае, если в $X_{i,j}$ входят все элементарные входные сигналы из множества [X], такой сигнал в дальнейшем будем называть полным входным сигналом.

Под выходным сигналом $Y_{i,j}$ будем понимать комбинацию (сочетание) элементарных выходных сигналов из множества [Y], действующих в автомате Мили на переходе от состояния – a_i к a_j. Наконец, выходной сигнал Y_j – это комбинация элементарных сигналов из множества [Y], отмечающих состояние a_j автомата Мура.

Применительно к прямой таблице переходов условия автоматности отображения формулируются следующим образом:

а) попарное произведение всех частных входных сигналов $X_{i,j}$, действующих на всех переходах от состояния a_i, должно быть равно 0 (**условие однозначности**);

б) логическая сумма всех частных входных сигналов $X_{i,j}$, действующих на всех переходах от состояния a_i, должна быть равна единице (**условие полноты**).

Для фрагмента прямой таблицы переходов (табл. П3.2) автомата Мили проверку условий автоматности, например, при переходе из состояния a_3, можно определить, исходя из следующего:

$$x_5(\overline{x}_5x_2) \vee x_5(\overline{x}_5\overline{x}_2) \vee (\overline{x}_5x_2)(\overline{x}_5\overline{x}_2) = 0;$$

$$x_5 \vee \overline{x}_5x_2 \vee \overline{x}_5\overline{x}_2 = 1.$$

Таблица П3.2

Номер перехода	Исходное состояние $a_i(t)$	Частный входной сигнал $X_{i,j}(t)$	Состояние перехода $a_j(t+1)$	Выходной сигнал на переходе $Y_{i,j}(t)$
11	a_5	x_5	a_5	y_{11}
12		$\overline{x}_5x_2$	a_2	y_9y_2
13		$\overline{x}_5\overline{x}_2$	a_1	y_8y_2

Рассмотрим преобразование произвольного автомата Мили в эквивалентный ему автомат Мура и наоборот. Для решения этой задачи воспользуемся представлением автомата в виде СКУ и СВФ. Для нахождения всех состояний автомата Мура, эквивалентного ему автомата Мили необходимо выполнить расщепление состояний автомата Мили, исходя из следующего.

Если автомат Мили при переходе в некоторое состояние a_j может выдавать в разные моменты времени один из m различных выходных сигналов из алфавита [W], то такое состояние должно быть расщеплено на m состояний, которые будут принадлежать эквивалентному ему автомату Мура. Такое расщепление состояний автомата Мили необходимо, потому что все состояния эквивалентного ему автомата Мура должны быть отмечены только одним выходным сигналом из алфавита [$\boldsymbol{W}$].

Методика получения СКУ для всех m состояний автомата Мура, получающихся из расщепления состояния a_j, и для всех возможных M состояний автомата Мили заключается в следующем:

1) в каждом из уравнений СКУ исходного автомата Мили, пользуясь СВФ, находим все переходы z_ka_j, отмеченные одинаковым выходным сигналом;

2) объединяя эти переходы знаком дизъюнкции, получим описания для всех m состояний автомата Мура, полученных из расщепления состояния a_j, автомата Мили;

3) выполним рассмотренные процедуры для всех a_j состояний автомата Мили. При этом удобно первоначально обозначать состояния автомата Мура двойным индексом: $a_j,_1$, $a_j,_2$, …, $a_{j,m}$;

4) производя необходимые операции подстановки и замены для всех a_j, получим отмеченную СКУ для переходов состояний автомата Мура, эквивалентного исходному автомату Мили.

Для рассмотренного примера автомата Мили (см. табл. П3.1) имеем

$$
\begin{aligned}
&a_1 = a_{1,1} \vee a_{1,2};\ \ a_2 = a_{2,1} \vee a_{2,2};\\
&a_0^{w_1}(t+1) = z_1 a_3 \vee z_2 (a_{2,1} \vee a_{2,2});\\
&a_{1,1}^{w_1}(t+1) = z_1 a_0;\\
&a_{1,2}^{w_3}(t+1) = z_2 (a_{1,1} \vee a_{1,2});\\
&a_{2,1}^{w_2}(t+1) = z_1 (a_{1,1} \vee a_{1,2}) \vee (z_2 a_0);\\
&a_{2,2}^{w_1}(t+1) = z_2 a_3;\\
&a_{1,3}^{w_1}(t+1) = z_1 (a_{2,1} \vee a_{2,2}).
\end{aligned}
\qquad \text{(П3.3)}
$$

Если ввести новые обозначения для состояний автомата Мура:

$$a_0 = b_0;\ \ a_{1,1} = b_1;\ \ a_{1,2} = b_2;\ \ a_{2,1} = b_3;\ \ a_{2,2} = b_4;\ \ a_3 = b_5,$$

то на основании (П3.3) и замены переменных получим отмеченную таблицу переходов автомата Мура (табл. П3.3).

Таблица П3.3

w	w_1	w_1	w_3	w_2	w_1	w_1
b / z	b_0	b_1	b_2	b_3	b_4	b_5
z_1	b_1	b_3	b_3	b_5	b_5	b_0
z_2	b_3	b_2	b_2	b_0	b_0	b_4

Рассмотрим теперь переход от произвольного автомата Мура к эквивалентному ему автомату Мили. Эту задачу можно решить, например, одним из двух способов. Первый способ основан на использовании отмеченной таблицы переходов исходного автомата Мура. Применяя этот способ, осуществляют подстановку в отмеченную таблицу переходов вместе с состояниями и отмечающие их выходные сигналы. Затем выполняют минимизацию числа состояний автомата Мили известными способами, например, как в [2].

Второй способ основан на использовании отмеченной СКУ исходного автомата Мура. Для этого способа по отмеченной СКУ

строят СВФ, объединяя знаком дизъюнкции правые части тех уравнений СКУ, для которых состояния отмечены одинаковыми выходными сигналами. После этого выполняют минимизацию СКУ путем склеивания состояний и подстановки их в СВФ [2].

П4. Задачи анализа и синтеза цифровых автоматов

Решение задачи синтеза цифровых автоматов складывается из двух больших этапов: этапа синтеза абстрактных автоматов (или абстрактный синтез) и этапа синтеза структурных автоматов (или структурный синтез).

Этап ***синтеза абстрактных автоматов*** охватывает решение следующих основных вопросов синтеза автоматов на алгоритмическом уровне:

– описание закона функционирования (поведения) автомата на одном из начальных языков, т.е., следуя [2], получают описание оператора, преобразующего множество слов входного алфавита во множество слов исходного алфавита той же длины;

– выполнение необходимых эквивалентных преобразований описания оператора на начальном языке;

– выполнение перехода от описания оператора на начальном языке к описанию на одном из автоматных языков; эту последнюю операцию принято называть собственно процедурой синтеза абстрактных автоматов.

В качестве автоматного языка будем использовать СКУ и СВФ, а также прямые таблицы переходов (или списки переходов). При этом, как описано выше, для наиболее эффективного решения задачи синтеза целесообразно использовать в качестве входного (выходного) алфавита элементарные двузначные входные (выходные) сигналы. Отметим также, что в ряде случаев язык СКУ и СВФ наиболее эффективно может быть использован и в качестве начального языка. В общей задаче синтеза абстрактных цифровых автоматов должны решаться две проблемы: проблема существования и проблема единственности [2]. Первая проблема обычно формулируется в следующем виде: «реализуем ли (представим) описанный оператор в конечном автомате?». Для языков регулярных выражений алгебры событий, ЛСА, ГСА ответ на этот вопрос утвердительный. Для проблемы существования в такой форме, в какой она приведена выше, проблемы единственности, формулируемой в виде: «единствен ли оператор, удовлетворяющий заданным в задании условиям?», не существует. При утвердительном

ответе на вопрос о реализуемости основной становится задача, формулируемая следующим образом: «для какого-нибудь оператора, который удовлетворяет данным условиям, построить какой-нибудь автомат, реализующий (представляющий) его».

Решение этой задачи неоднозначно в связи с применением неопределенного термина «какой-нибудь». Так как проблемы единственности (при данной постановке проблемы существования) не существует, остается только неоднозначность, связанная с выбором автомата, обычно устраняемая рядом дополнительных требований: минимальности числа состояний, ограничений на способы реализации или способы взаимодействия автомата с внешней средой (синхронные, асинхронные и согласованные модели в системе «внешняя среда – автомат») и др.

Проблема анализа автоматов обратна проблеме синтеза, решение которой заключается в описании для заданного конечного автомата его поведения средствами исходного языка. Обычно задача анализа проще, чем задача синтеза.

Этап синтеза структурных цифровых автоматов охватывает решение вопросов построения схемы автомата по заданному описанию на одном из автоматных языков. Как показано в [90], решение задачи структурного синтеза цифровых автоматов, построенных на основе «жесткой» логики, в конечном итоге сводится в основном к синтезу комбинационных схем, построенных на логических элементах. Это обстоятельство и предопределило выбор СКУ и СВФ в качестве стандартного автоматного языка, а также использование прямых таблиц переходов (список переходов). В связи с этим на этапе структурного синтеза будут решаться следующие вопросы:

– выбор структуры автомата и системы логических элементов, а также элементов памяти;

– выполнение необходимых эквивалентных преобразований СКУ и СВФ (минимизация, композиция и декомпозиция, кодирование, детерминизация и др.);

– построение комбинационных схем на основе выбранной элементной базы и др.

Для реализации автоматов на основе «программируемой логики», и в частности, управляющих микропрограммных автоматов, на этапе структурного синтеза необходимо решать свои специфические задачи, из которых к наиболее важным можно отнести:

– выбор и реализацию способа последовательности обработки микрокоманд:

– кодирование и размещение микрокоманд в памяти;

– выбор структуры и построение блока управления последовательного выполнения микрокоманд и др.

Применение формальных методов синтеза наиболее эффективно при условии их автоматизации. Автоматизация предполагает использование определенных структур данных и преемственности этих структур при переходе от одного этапа (или подэтапа) к другому. В этом случае использование СКУ и СВФ или списка переходов обеспечивает простоту представления описания в ЭВМ и позволяет выполнить синтез автоматом без необходимости получения классических таблиц переходов и методов.

В заключение отметим, что, учитывая достоинства использования СКУ и СВФ для решения задач анализа и синтеза цифровых устройств ЭВУ и систем промышленной автоматики, можно свести реализацию этих задач к анализу и синтезу комбинационных схем. При этом в качестве основного математического аппарата используется аппарат алгебры логики, основные сведения о нем и его использовании представлены в следующем разделе.

П5. Основы алгебры логики, ее основные законы и методы использования для решения задач анализа и синтеза цифровых автоматов [90]

Наибольшее применение из-за простоты построения структурных схем ЭВМ нашел двоичный структурный алфавит. Элементарные сигналы, составляющие этот алфавит, чаще всего представляются импульсами электрического напряжения (тока), уровнями электрического напряжения (тока) или их комбинацией. При этом возможны различные способы отождествления реальных физических сигналов с нулем и единицей. Например, можно принять: наличие импульса – за единицу, отсутствие – за нуль; высокий уровень напряжения – за единицу, низкий – за нуль и др.

В связи с двоичным представлением структурного алфавита в качестве математического аппарата для целей анализа и синтеза цифровых схем ЭВМ и, в первую очередь, схем комбинационного действия оказалось удобным применять аппарат алгебры логики.

Введем понятия двоичной переменной и двоичной функции. Двоичными (булевыми) переменными $x_1, x_2, \ldots, x_n$ называются переменные, которые могут принимать только два значения: нуль и единицу. Совокупность значений таких переменных называют набором.

Двоичной (*булевой*) *функцией от двоичных переменных* называется функция, которая может принимать только два значения: нуль и единицу. Область определения булевой функции конечна, так как аргументы функции принимают только два значения. Общее число наборов двоичных аргументов, на которых определяется булева функция, равно 2^n.

Любая булева функция может быть задана с помощью таблицы, в левой части которой выписываются все наборы значений двоичных переменных, а в правой – соответствующие им значения функции (табл. П5.1). Такая таблица называется таблицей истинности. Действительно, если значения переменных, равных нулю, отождествлять с понятием ложного высказывания, а единицы – истинного высказывания, то функциональная таблица определит истинность или ложность сложного высказывания в зависимости от истинности или ложности составляющих высказываний.

Таблица П5.1

$x_1x_2 \ldots x_{n-1}x_n$	$f(x_1, x_2, \ldots x_{n-1}, x_n)$
00…00	$f_0(x_1, x_2, \ldots, x_{n-1}, x_n)$
00…01	$f_1(x_1, x_2, \ldots x_{n-1}, x_n)$
00…10	$f_2(x_1, x_2, \ldots, x_{n-1}, x_n)$
. . . . .	. . . . .
11 …10	$f_{n-2}(x_1, x_2, \ldots, x_{n-1}, x_n)$
11 … 11	$f_{n-1}(x_1, x_2, \ldots, x_{n-1}, x_n)$

Общее число различных булевых функций от n переменных равно 2^{2^n} и может быть определено индуктивно, исходя из следующего. Для одного набора аргументов могут быть определены две булевы функции, для двух – 2^2, для k наборов – $2^к$ и, наконец, для 2^n наборов – 2^{2^n}.

Для анализа и синтеза схем цифровых ЭВМ очень широко используются булевы функции одной и двух переменных. Для одной переменной имеются всего четыре различные булевы функции. Таблицы их соответствия приведены в табл. П5.2, откуда следует, что функции g_1 *и* g_4 являются константами «0» и «1», соответственно, а функция g_2 повторяет значение переменной x, т.е. $g_2 = x$.

Таблица П5.2

x	g_1	g_2	g_3	g_4
0	0	1	1	1
1	0	0	0	1

Функция g_3 называется *отрицанием переменной x* или *инверсией* и обозначается через $\overline{x}$, т.е. $g_3 = \overline{x}$. Число всех булевых функций двух переменных равно 16 (табл. П5.3).

В число шестнадцати функций входят и *вырожденные функции*: $f_0(x_1, x_2) = 0$ – константа нуль, $f_{15}(x_1, x_2) = 1$ – константа единица, $f_3(x_1, x_2) = x_1$ – переменная x_1, $f_5(x_1, x_2) = x_2$ – переменная x_2, $f_{12}(x_1, x_2) = \overline{x}_1$ – инверсия x_1, $f_{10}(x_1, x_2) = \overline{x}_2$ – инверсия x_2.

Таблица П5.3

x_1	x_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Остальные десять функций двух переменных с их обозначениями и названиями сведены в табл. П5.4. С помощью функций одной и двух двоичных переменных, называемых *элементарными логическими функциями*, можно, используя *принцип суперпозиции*, или иначе принцип подстановки булевых функций вместо аргументов в другую функцию, построить любую булеву функцию.

Таблица П5.4

Функция	Название	Выражение через конъюнкцию, дизъюнкцию и отрицание	Чтение
$f_1(x_1, x_2)$	Конъюнкция	x_1x_2	x_1 и x_2
$f_7(x_1, x_2)$	Дизъюнкция	$x_1 \vee x_2$	x_1 или x_2
$f_6(x_1, x_2)$	Сложение по модулю 2	$x_1\overline{x}_2 \vee \overline{x}_1x_2$	x_1 неравнозначно x_2
$f_8(x_1, x_2)$	Функция Пирса	$\overline{x}_1\,\overline{x}_2$	Ни x_1, ни x_2
$f_9(x_1, x_2)$	Эквивалентность	$\overline{x}_1\,\overline{x}_2 \vee x_1x_2$	x_1 равнозначно x_2
$f_{11}(x_1, x_2)$	Импликация	$x_1 \vee \overline{x}_2$	Если x_2, то x_1
$f_{14}(x_1, x_2)$	Функция Шеффера	$\overline{x}_1 \vee \overline{x}_2$	Неверно, что x_1 и x_2
$f_2(x_1, x_2)$	Запрет по x_2	$x_1\overline{x}_2$	Неверно, что если x_1, то x_2
$f_4(x_1, x_2)$	Запрет по x_1	$\overline{x}_1x_2$	Неверно, что если x_2, то x_1
$f_{13}(x_1, x_2)$	Импликация	$x_2 \vee \overline{x}_1$	Если x_1, то x_2

Рассматривая булевы функции одной и двух переменных как операции на множестве всех булевых функций, можно построить различные алгебры булевых функций.

В частности, нас в дальнейшем будет интересовать алгебра, называемая булевой, когда над булевым множеством определены три операции: отрицание, логическое умножение (конъюнкция) и логическое сложение (дизъюнкция). Необходимо отметить, что в качестве символа логического умножения используют также знак *&* или точку, а иногда знаки совсем опускают, что и будет использовано в дальнейшем.

Булева алгебра. Основные законы. В булевой алгебре для представления любой функции в виде формулы, кроме символов основных операций отрицания, конъюнкции и дизъюнкции, используются следующие символы: малые буквы типа *x*, *y*, *z*, ... для обозначения переменных (включая буквы с индексами), константы 0 и 1, пара символов (), которые называются скобками.

Формулы булевой алгебры будут представляться конечными последовательностями символов, указанных выше, записанных в виде строчек и удовлетворяющих требованиям определения формулы. Формулами булевой алгебры являются: булевы переменные *x*, *y*, *z*, ...; константы 0 и 1; выражения вида: (AB), (AVB), $\overline{A}$, где A и B являются формулами.

В булевой алгебре при отсутствии в выражении скобок вводится следующий порядок действий: первыми выполняются операции отрицания, далее – конъюнкции, а затем – дизъюнкции. Наличие в выражении скобок изменяет обычный порядок действий: в первую очередь должны выполняться операции внутри скобок.

Две формулы булевой алгебры будут равносильны (*равны*, *эквивалентны*), *если равны сопоставляемые им функции*, т.е. они принимают одинаковые значения на всех наборах значений аргументов.

Укажем **основные законы булевой алгебры**, позволяющие производить различные тождественные преобразования формул булевой алгебры.

1. Закон двойного отрицания $\overline{\overline{x}} = x$.

2. Закон коммутативности:

$$x_1 \vee x_2 = x_2 \vee x_1;$$

$$x_1 x_2 = x_2 x_1.$$

3. Закон ассоциативности:

$$x_1 \vee (x_2 \vee x_3) = (x_1 \vee x_2) \vee x_3;$$
$$x_1(x_2 x_3) = (x_1 x_2) x_3;$$

4.Законы дистрибутивности:

$$x_1 \vee (x_2 \vee x_3) = x_1 x_2 \vee x_1 x_3;$$
$$x_1 \vee x_2 x_3 = (x_1 \vee x_2)(x_1 \vee x_3).$$

5. Правила де-Моргана:

$$\overline{x_1 \vee x_2} = \overline{x}_1 \overline{x}_2;$$
$$\overline{x_1 x_2} = \overline{x}_1 \vee \overline{x}_2.$$

6. Правила операций с константами 0 и 1:

$$\overline{0} = 1; \ \overline{1} = 0;$$
$$x1 = x; \ x0 = 0;$$
$$x \vee 0 = x, \ x \vee 1 = 1.$$

7. Правила операций с переменной и ее инверсией:

$$x \vee \overline{x} = 1;$$
$$x\overline{x} = 0.$$

Справедливость основных законов (тождеств) булевой алгебры может быть доказана перебором всех значений переменных, входящих в проверяемые соотношения.

Из основных законов можно легко получить следующие важные соотношения:

1. Законы поглощения:

$$x_1 \vee x_1 x_2 = x_1;$$
$$x_1(x_1 \vee x_2) = x_1.$$

2. Закон идемпотентности дизъюнкции и конъюнкции:

$$x \vee x \vee \ldots \vee x = x;$$
$$xx \ldots x = x.$$

3. На основании правила де-Моргана, пользуясь методом математической индукции, отрицание любого выражения алгебры логики, построенного с использованием операций конъюнкции, дизъюнкции и отрицания, можно получить заменой в исходном выражении аргументов их отрицаниями и обменом местами символов конъюнкции и дизъюнкции.

4. Используя второй закон дистрибутивности и выполняя тождественные преобразования, можно получить:

$$x_1 \vee \overline{x}_1 x_2 = x_1 \vee x_2.$$

Формы представления булевых функций. В алгебре логики доказывается, что любую функцию алгебры логики, кроме функции $f = 0$, можно представить в виде формулы через конъюнкцию, дизъюнкцию и отрицание в виде

$$z = f(x_1, x_2, ..., x_k, ..., x_n) = \vee x_1^{\sigma_1} x_2^{\sigma_2} ... x_k^{\sigma_k} ... x_n^{\sigma_n}, \qquad (П5.1)$$

где $x_k^{\sigma_k}$ – общее обозначение для аргумента x_k и его отрицания $\overline{x}_k$,

причем $x_k^{\sigma_k} = \begin{cases} x_k \text{ при } \sigma_k = 1; \\ \overline{x}_k \text{ при } \sigma_k = 0. \end{cases}$

Логическое суммирование в (П5.1) ведется для тех наборов $\sigma_1, \sigma_2, \ldots, \sigma_k, \ldots \sigma_n$, для которых $f(x_1, x_2, \ldots, x_k, \ldots x_n) = 1$. Представление функции алгебры логики в форме (П5.1) называют *совершенной дизъюнктивной нормальной формой* (СДНФ). Члены, входящие в СДНФ, называют *дизъюнктивными членами* или *конституентами единицы.*

Для таблично заданной булевой функции СДНФ строится так: выписываем из таблицы те наборы, для которых функция равна единице, для каждого выписанного набора составляем конъюнкции $x_1^{\sigma_1}, x_2^{\sigma_2}, ..., x_n^{\sigma_n}$, соединяя полученные конъюнкции знаком дизъюнкции, получим СДНФ искомой функции.

• Составить СДНФ для таблично заданной функции (табл. П5.5). Воспользовавшись правилом построения СДНФ, получим:

$$z = f(x_1, x_2, x_3) = \overline{x}_1 \overline{x}_2 x_3 \vee \overline{x}_1 x_2 x_3 \vee x_1 \overline{x}_2 x_3 \vee x_1 x_2 \overline{x}_3.$$

Таблица П5.5

x_1	x_2	x_3	z		x_1	x_2	x_3	z
0	0	0	0		1	0	0	0
0	0	1	1		1	0	1	1
0	1	0	0		1	1	0	1
0	1	1	1		1	1	1	0

Возможно также иное представление функций алгебры логики, называемое *совершенной конъюнктивной нормальной формой* (СКНФ). В этом случае функция составляется из дизъюнкций,

называемых *конъюнктивными членами* СКНФ или *конституентами нуля*, объединенных знаком конъюнкции. Для рассмотренного примера СКНФ функции будет иметь вид

$$z = (x_1 \vee x_2 \vee x_3)(x_1 \vee \overline{x}_2 \vee x_3)(\overline{x}_1 \vee x_2 \vee x_3)(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3).$$

Переход от СДНФ к СКНФ представления функций алгебры логики осуществляется так: выписывается логическая сумма дизъюнктивных членов, не вошедших в СДНФ, т. е. отрицание функции; от полученной логической суммы дизъюнктивных членов берется отрицание.

• Например, построить СКНФ булевой функции по ее СДНФ:

$$z = f(x_1, x_2) = \overline{x}_1 x_2 \vee x_1 \overline{x}_2.$$

Для этой функции ее отрицание имеет вид $\overline{z} = x_1, x_2 \vee \overline{x}_1 \overline{x}_2$, откуда $z = \overline{\overline{z}} = (\overline{x}_1 \vee \overline{x}_2)(x_1 \vee x_2)$.

Аналогично осуществляется переход от СКНФ к СДНФ представления функции алгебры логики.

Понятие о полноте системы функций алгебры логики. Система элементарных булевых функций $\varphi_1, \varphi_2, \ldots, \varphi_m$ называется функционально полной, если любую функцию алгебры логики можно представить в виде суперпозиции функций $\varphi_1, \varphi_2, \ldots, \varphi_m$. Примером функционально полной системы элементарных булевых функций служит система трех функций: $\varphi_1 = \overline{x}$, $\varphi_2 = x_1 x_2$, $\varphi_3 = x_1 \vee x_2$. Это следует из того, что любую функцию алгебры логики можно представить в виде формулы с помощью конъюнкции, дизъюнкции и отрицания (П3.3). Примерами функционально полных систем элементарных функций также являются:

$\varphi_1 = \overline{x}$, $\varphi_2 = x_1 x_2$; $\varphi_1 = \overline{x}$, $\varphi_2 = x_1 \vee x_2$;

$\varphi = \overline{x}_1 \vee \overline{x}_2$; $\varphi = \overline{x}_1 \overline{x}_2$; $\varphi_1 = x_1 \oplus x_2$; $\varphi_2 = x_1 x_2$; $\varphi_2 = 1$;

$\varphi_1 = x_1 \overline{x}_2$, $\varphi_2 = 1$.

Доказательством полноты приведенных выше систем элементарных функций может служить возможность сведения любой из них к функционально полной системе из трех элементарных функций $\overline{x}_1$, $x_1 x_2$, $x_1 \vee x_2$. Например, для первой системы элементарных функций отсутствует дизъюнкция. Ее можно получить из выражения $x_1 \vee x_2 = \overline{\overline{x}_1 \vee \overline{x}_2}$. Для второй системы отсутствует конъюнкция, которая может быть получена из выражения

$x_1 x_2 = \overline{\overline{x}_1 \vee \overline{x}_2}$. Таким образом, первая и вторая системы элементарных функций сводятся к функционально полной системе, поэтому они являются функционально полными. Аналогично доказывается полнота и других приведенных выше систем.

Минимизация булевых функций. *Минимизация* – процесс приведения булевых функций к такому виду, который допускает наиболее простую, с наименьшим числом элементов, физическую реализацию функции. Частная задача минимизации булевой функции сводится к такому представлению заданной функции, которое содержит наименьшее возможное число букв и наименьшее возможное число операций над ними.

Оценить различные представления одной и той же булевой функции, например ДНФ, можно по количеству входов логических элементов, реализующих заданную функцию. Такую оценку реализации булевой функции называют *ценой реализации булевой функции по Квайну* (или ценой покрытия булевой функции системой логических элементов).

Ниже будут рассмотрены логические элементы, каждый из которых может реализовать некоторую булевую функцию, число переменных которых соответствует числу входов этого элемента.

Для изложения методов минимизации введем некоторые определения. Конъюнкция $p = x_1^{\sigma_1} x_2^{\sigma_2} \ldots x_k^{\sigma_k}$ называется элементарной, если число ее членов меньше некоторого множества переменных n, причем любая переменная $x_i^{\sigma_i}$ входит в конъюнкцию только один раз. Число членов элементарной конъюнкции определяет ее ранг.

Под *импликантой булевой функции* $f_1(x_1, x_2, ..., x_n)$ понимается такая булева функция $f_1(x_1, x_2, ..., x_n)$, если на любом наборе значений переменных $x_1, x_2, ..., x_n$, на котором значение функции f_1 равно единице, значение функции f также равно единице.

Под *простой импликантой булевой функции* $f(x_1, x_2, \ldots x_n)$ понимается всякое элементарное произведение $p = x_1^{\sigma_1} x_2^{\sigma_2} \ldots x_k^{\sigma_k}$, которое является импликантой функции f, и никакая собственная часть этого произведения в функцию f не входит, т.е. простые импликанты – элементарные конъюнкции наименьшего ранга, входящие в данную булеву функцию.

Сокращенной дизъюнктивной нормальной формой булевой функции называется такая функция, которая равна дизъюнкции всех своих простых импликант. Несмотря на то, что сокращенная ДНФ булевой функции содержит меньшее число букв, чем СДНФ этой же функции, она в большинстве случаев допускает дальнейшее упрощение за счет поглощения некоторых простых импликант дизъюнкцией других простых импликант. В случае, если в дизъюнкции простых импликант, представляющих заданную булеву функцию, ни одну из импликант исключить нельзя, такую дизъюнкцию называют *тупиковой дизъюнктивной нормальной формой заданной функции.*

Некоторые булевы функции имеют несколько тупиковых ДНФ. Тупиковая ДНФ булевой функции, содержащая наименьшее число букв, будет минимальной ДНФ.

Таким образом, общую задачу минимизации булевых функций можно решать в такой последовательности. Для заданной функции находят сокращенную ДНФ, т.е. все простые импликанты. Затем определяют тупиковые ДНФ заданной функции, среди которых выбирают минимальную ДНФ.

Рассмотрим один из наиболее распространенных методов нахождения сокращенных ДНФ – *метод Квайна.* Для этого метода исходная булева функция должна быть представлена в СДНФ. Если эта функция представлена в произвольной ДНФ, то ее с помощью операции развертывания, заключающейся в умножении некоторых членов на выражение вида $x \vee \overline{x} = 1$, приводят к СДНФ. Метод Квайна основан на последовательном применении к парам дизъюнктивных членов операций склеивания и элементарного поглощения. Операция склеивания основана на справедливости тождества $x_1x_2 \vee x_1\overline{x}_2 = x_1(x_2 \vee \overline{x}_2) = x_1$ (закон поглощения).

Вначале в СДНФ исходной булевой функции проводят все возможные операции склеивания дизъюнктивных членов − конъюнкций ранга n, где n − число аргументов функции. В результате склеивания получим конъюнкции $n - 1$ ранга. После выполнения операции поглощения с конъюнкциями $n - 1$ ранга осуществляют все возможные операции склеивания конъюнкций $n - 1$ ранга. Затем проводят операции поглощения с конъюнкциями $n - 2$ ранга и вновь выполняют операции склеивания и т.д.

• Найти сокращенную ДНФ булевой функции

$$z = f(x_1, x_2, x_3, x_4) = \overline{x}_1x_4 \vee x_1x_3 \vee \overline{x}_3\overline{x}_4 \vee \overline{x}_1x_2\overline{x}_3.$$

Используя операцию развертывания, представим исходную функцию в СДНФ (рис. П5.1).

1	2	3	4	5	6
$z = \bar{x}_1 x_2 \bar{x}_3 x_4$	$\vee \bar{x}_1\ \bar{x}_2\ \bar{x}_3$	$\vee \bar{x}_1\ \bar{x}_2\ \bar{x}_3$	$\vee \bar{x}_1\ x_2\ \bar{x}_3$	$\vee \bar{x}_1\ \bar{x}_2\ x_3$	$\vee \bar{x}_1\ x_2\ x_3$
7	8	9	10	11	12
$\vee x_1\ \bar{x}_2\ x_3$	$\vee x_1\ \bar{x}_2\ x_3\ x_4$	$\vee x_1\ x_2\ x_3$	$\vee x_1\ x_2\ x_3$	$\vee x_1\ x_2\ \bar{x}_3$	$\vee x_1\ \bar{x}_2\ \bar{x}_3$

Рис. П5.1. Сокращенная ДНФ функции

Пронумеруем все дизъюнктивные члены.

Выполним все возможные операции склеивания дизъюнктивных членов в такой последовательности: 1) первый член со всеми остальными, 2) второй член с остальными, кроме первого, 3) третий член с остальными, кроме первого и второго, и т.д.

В результате проведенных операций склеивания и поглощения получим ДНФ заданной функции в форме (рис. П5.2).

1	2	3	4	5	6
$z = \bar{x}_1\ \bar{x}_3 x_4$	$\vee \bar{x}_1\ \bar{x}_3\ \bar{x}_4$	$\vee \bar{x}_1\ x_3 x_4$	$\vee x_1\ x_3\ \bar{x}_4$	$\vee x_1\ x_3 x_4$	$\vee x_1\ \bar{x}_3\ \bar{x}_4$
7	8	9	10	11	12
$\vee \bar{x}_2\ x_3\ x_4$	$\vee x_2\ x_3\ x_4$	$\vee x_2\ \bar{x}_3\ \bar{x}_4$	$\vee \bar{x}_2\ \bar{x}_3\ \bar{x}_4$	$\vee \bar{x}_1\ x_2\ \bar{x}_3$	$\vee \bar{x}_1\ \bar{x}_2\ \bar{x}_3$
13	14	15	16	17	18
$\vee x_1\ \bar{x}_2\ x_3$	$\vee x_1\ x_2\ x_3$	$\vee \bar{x}_1\ x_2 x_4$	$\vee \bar{x}_1\ \bar{x}_2 x_4$	$\vee x_1\ \bar{x}_2\ \bar{x}_4$	$\vee x_1\ x_2\ \bar{x}_4$

Рис. П5.2. Результаты склеивания и поглощения ДНФ функции

Пронумеровав все члены выражения, полученного на предыдущем этапе, выполним все возможные операции склеивания конъюнкций третьего ранга в той же последовательности, что и ранее. В результате проведенных операций склеивания и поглощения исходная функция примет вид

$$z = \bar{x}_1\bar{x}_3 \vee \bar{x}_1 x_4 \vee x_1 x_3 \vee x_1\bar{x}_4 \vee x_3 x_4 \vee \bar{x}_3\bar{x}_4.$$

Метод импликантных таблиц является одним из методов нахождения тупиковых и минимальных ДНФ по сокращенным ДНФ. Импликантная таблица представляет собой прямоугольную таблицу, столбцы которой соответствуют конституентам единицы исходной булевой функции, а строки – простым импликантам. В случае если простую импликанту можно получить из конституенты единицы вычеркиванием некоторых букв, то говорят, что импликанта накрывает (покрывает) конституенту. В этом случае клетка импликантной

таблицы, соответствующая импликанте и конституенте, отмечается специальным знаком (например, звездочкой).

Заполним в соответствии с указанным выше правилом импликантную таблицу (табл. П5.6) для сокращенной ДНФ булевой функции, полученной в рассмотренном выше примере.

Таблица П5.6

Номер строки	Простые импликанты	Конституенты единицы											
		$\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4$	$\bar{x}_1\bar{x}_2\bar{x}_3x_4$	$\bar{x}_1\bar{x}_2x_3x_4$	$\bar{x}_1x_2\bar{x}_3\bar{x}_4$	$\bar{x}_1x_2\bar{x}_3x_4$	$\bar{x}_1x_2x_3x_4$	$x_1\bar{x}_2\bar{x}_3\bar{x}_4$	$x_1\bar{x}_2x_3\bar{x}_4$	$x_1\bar{x}_2x_3x_4$	$x_1x_2\bar{x}_3\bar{x}_4$	$x_1x_2x_3\bar{x}_4$	$x_1x_2x_3x_4$
1	$x_1\bar{x}_4$							×	×		×	×	
2	$\bar{x}_1x_4$		×	×		×	×						
3	x_1x_3								×	×		×	×
4	$\bar{x}_1\,\bar{x}_3$	×	×		×	×							
5	x_3x_4			×			×			×			×
6	$\bar{x}_3\bar{x}_4$	×			×			×			×		

Минимальной ДНФ заданной функции будет соответствовать такая система минимального числа строк таблицы, импликанты которых совместно накрывают крестиками все колонки таблицы.

Для получения тупиковых ДНФ можно пользоваться следующими рекомендациями:

1. Выделяются все существенные (или обязательные) импликанты. Существенной импликанте соответствуют столбцы таблицы, в которых имеется только одна метка (звездочка). Она обязательно входит в минимальное покрытие, так как, не используя ее, невозможно покрыть все конституенты единицы.

2. Вычеркиваются столбцы таблицы, которым соответствуют выделенные существенные импликанты.

3. Выбирается такая совокупность оставшихся простых импликант после выделения существенных импликант, которая обеспечивает покрытие всех оставшихся столбцов таблицы с минимальной ценой. При большом числе переменных для оставшихся простых импликант и конституент единицы строится новая импликантная таблица.

Для примера (см. табл. П5.6) возможны два варианта минимальных ДНФ:

$$z = \bar{x}_1\bar{x}_3 \vee x_1\bar{x}_4 \vee x_3x_4;$$

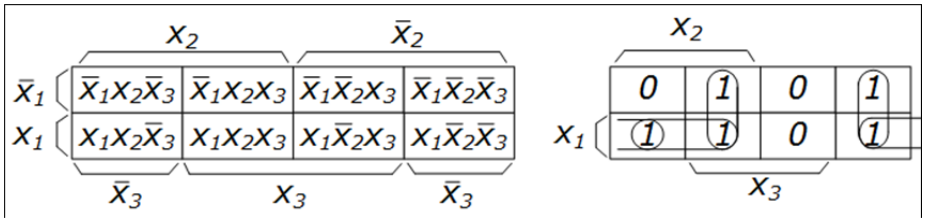
$$z = x_1x_3 \vee \bar{x}_1x_4 \vee \bar{x}_3\bar{x}_4.$$

Первый вариант соответствует системе строк, состоящей из 1-, 4- и 5-й строк; второй – из 2-, 3- и 6-й строк.

Диаграммы Вейча также являются методом, используемым для минимизации булевых функций. Диаграмма Вейча представляет собой своеобразный способ задания булевых функций с помощью специальной таблицы, число клеток которой равно числу всех возможных наборов аргументов булевой функции. Таким образом, каждой клетке диаграммы Вейча можно поставить в соответствие конституенту единицы.

Для записи конкретной булевой функции в те клетки, для которых конституента единицы входит в исходную булевую функцию, записываются единицы, в оставшиеся – нули.

Минимизация булевых функций с использованием диаграмм Вейча основывается на отыскании склеивающихся конституент единицы. Для диаграммы Вейча склеивающиеся конституенты единицы располагаются в соседних, вертикально или горизонтально расположенных клетках. Для диаграммы трех переменных (рис. П5.3,*а*) соседними клетками являются также клетки левого и правого столбцов для одноименных строк.

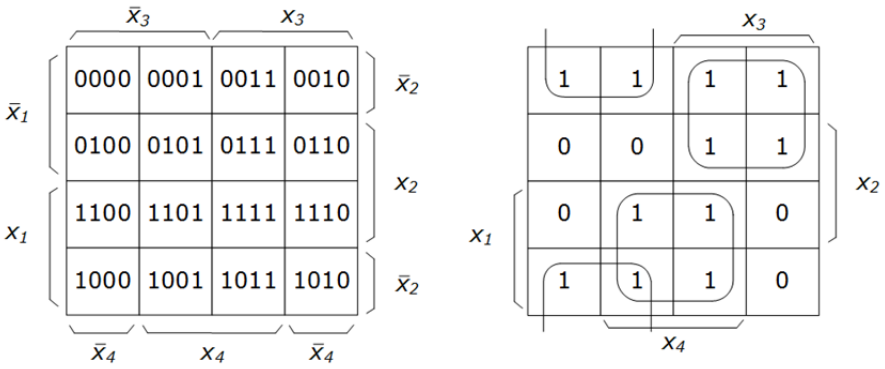


а) *б*)

Рис. П5.3. Диаграммы Вейча булевой функции трех переменных:
а – для всех возможных наборов аргументов;
б – для пяти наборов аргументов

Для диаграммы Вейча четырех переменных (рис. П5.4) соседними клетками будут также клетки, расположенные в верхних и нижних строках для одноименных столбцов.

Диаграммы Вейча для большего числа переменных могут быть составлены из диаграмм меньшего числа переменных. Например, диаграмму Вейча для пяти переменных можно составить, соединив две диаграммы для четырех переменных. При этом удобно подсоединять их друг к другу одноименными столбцами (рис. П5.5).



а) *б*)

Рис. П5.4. Диаграммы Вейча булевой функции четырех переменных:
а – для всех возможных наборов аргументов;
б – для одиннадцати наборов аргументов

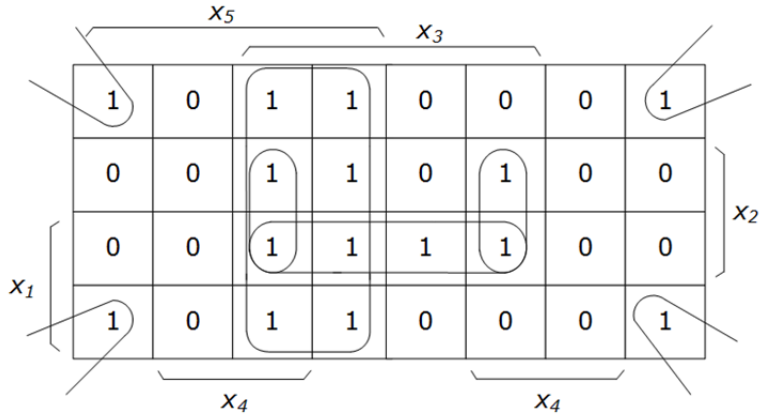


Рис. П5.5. Диаграмма Вейча булевой функции пяти переменных

Тогда соседними клетками будут также клетки столбцов, симметрично расположенных относительно линии присоединения диаграмм Вейча четырех переменных, для одноименных строк.

Процесс отыскания минимальной ДНФ заключается в том, чтобы всю совокупность единиц диаграммы Вейча накрыть наименьшим числом наиболее коротких произведений. Для этого соседние клетки диаграммы, содержащие единицы, объединяют в группы. Каждой такой группе будет соответствовать группа склеивающихся конституент единицы, причем количество клеток, входящих в одну группу, равно 2^k (где k = 1, 2, 3, …), а каждая клетка, входящая в группу, должна иметь k соседних клеток.

Приведем некоторые ***рекомендации по минимизации булевых функций с использованием диаграмм Вейча.***

1. Рассматриваются поочередно клетки, содержащие единицы, и анализируются всевозможные варианты склеивания. При

этом сначала склеивание выполняется только для тех клеток (единиц), для которых вариант склеивания единственный. В результате будут выделены обязательные (или существенные) простые импликанты.

2. Оставшиеся несклеенные клетки (единицы) необходимо склеивать таким образом, чтобы образовать минимальное число групп с максимальным числом клеток в каждой группе.

3. Каждой группе объединенных клеток в минимальной ДНФ будет соответствовать простая импликанта, определяемая как конъюнкция только тех переменных, значения которых постоянны для всех наборов, задающих клетки данной группы. Группе, состоящей из 2^k клеток, соответствует конъюнкция из $n-k$ букв, где n – число переменных в исходной булевой функции.

• Найти минимальную ДНФ для булевой функции трех переменных:

$$z = \overline{x}_1 x_2 x_3 \vee x_1 x_2 \overline{x}_3 \vee x_1 x_2 x_3 \vee \overline{x}_1 \overline{x}_2 \overline{x}_3 \vee x_1 \overline{x}_2 \overline{x}_3.$$

Диаграмма Вейча для этой функции представлена на рис. П5.3,*б*.

Для этой функции возможны два варианта объединения совокупности единиц в группы. Каждому варианту соответствует своя минимальная ДНФ:

$$z = x_1 \overline{x}_3 \vee x_2 x_3 \vee \overline{x}_2 \overline{x}_3;\ z = x_1 x_2 \vee x_2 x_3 \vee \overline{x}_2 \overline{x}_3.$$

• Найти минимальную ДНФ для булевой функции четырех переменных:

$$z = \overline{x}_1 \overline{x}_2 \overline{x}_3 \overline{x}_4 \vee \overline{x}_1 \overline{x}_2 \overline{x}_3 x_4 \vee \overline{x}_1 \overline{x}_2 x_3 x_4 \vee \overline{x}_1 \overline{x}_2 x_3 \overline{x}_4 \vee \overline{x}_1 x_2 x_3 x_4 \vee$$
$$\vee \overline{x}_1 x_2 x_3 \overline{x}_4 \vee x_1 x_2 \overline{x}_3 x_4 \vee x_1 x_2 x_3 x_4 \vee x_1 \overline{x}_2 \overline{x}_3 \overline{x}_4 \vee x_1 \overline{x}_2 \overline{x}_3 x_4 \vee x_1 \overline{x}_2 x_3 x_4.$$

Диаграмма Вейча для этой функции представлена на рис. П5.4,*б*, а минимальная ДНФ будет иметь вид

$$z = \overline{x}_1 x_3 \vee x_1 x_4 \vee \overline{x}_2 \overline{x}_3.$$

• Найти минимальную ДНФ для булевой функции пяти переменных, заданную диаграммой Вейча (см. рис. П5.5).

Выполняя необходимые операции по отысканию склеивающихся конституент единицы, получим следующую минимальную ДНФ:

$$z = x_3 x_5 \vee \overline{x}_2 \overline{x}_3 \overline{x}_4 \vee x_2 x_3 x_4 \vee x_1 x_2 x_3.$$

Рассмотрим упрощение неполностью определенных булевых функций, которые описывают работу логических схем с запрещенными комбинациями входных сигналов. Для неполностью определенных булевых функций можно произвольно задать значение функции на тех наборах, на которых она первоначально не была задана. При минимизации таких функций стремятся для запрещенных комбинаций входных сигналов присвоить такие значения исходной функции, которые позволяют ее упростить.

• Минимизировать булеву функцию четырех переменных, заданную диаграммой Вейча (рис. П5.6), только на 11 наборах двоичных аргументов. Неопределенные значения на диаграмме отмечены черточкой.

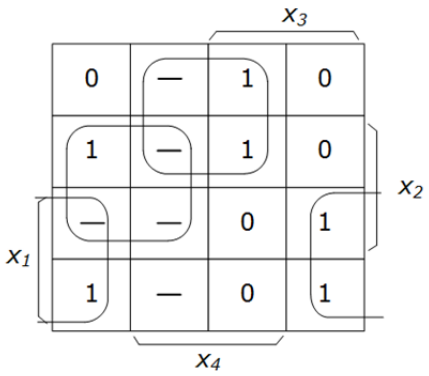


Рис. П5.6. Диаграмма Вейча неполностью определенной булевой функции четырех переменных

Принимая значения функции на некоторых наборах равными единице, а на других − нулю, там где функция не определена, получим ДНФ исходной функции в следующей минимальной форме:

$$z = x_1\overline{x}_4 \vee x_2\overline{x}_3 \vee \overline{x}_1 x_4.$$

ОГЛАВЛЕНИЕ

Научное издание

Вашкевич Николай Петрович,
Бикташев Равиль Айнулович

НЕДЕТЕРМИНИРОВАННЫЕ АВТОМАТЫ И ИХ ИСПОЛЬЗОВАНИЕ ДЛЯ РЕАЛИЗАЦИИ СИСТЕМ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ИНФОРМАЦИИ

Редактор *Т. Н. Судовчихина*
Компьютерная верстка *Р. Б. Бердниковой*
Дизайн обложки *А. А. Стаценко*

Подписано в печать 28.07.2016. Формат $60\times84^{1}/_{16}$.
Усл. печ. л. 22,9. Тираж 200.
Заказ № 404.

Издательство ПГУ.
440026, Пенза, Красная, 40.
Тел./факс: (8412) 56-47-33; e-mail: iic@pnzgu.ru