

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
СНЕЖИНСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ

В. В. Крушный

СИНТЕЗ ЦИФРОВЫХ УПРАВЛЯЮЩИХ АВТОМАТОВ

*Рекомендовано УМО «Ядерные физика и технологии»
в качестве учебного пособия
для студентов высших учебных заведений*

Москва 2011

УДК 681.3.001:681.5

ББК 32.972

К 84

Крушиный В.В. Синтез цифровых управляющих автоматов: Учебное пособие. – М.: НИЯУ МИФИ, 2011. – 164 с.

В учебном пособии рассматриваются вопросы формализации синтеза дискретных управляющих устройств от описания их работы до получения функционально-логических схем. Упор делается на изложении существующих методов структурного синтеза цифровых автоматов, удобных в инженерной практике и позволяющих строить микропрограммные автоматы реальной степени сложности. Изложение сопровождается достаточным количеством примеров.

Пособие предназначено для студентов специальностей: управление и информатика в технических системах, электроника и автоматика физических установок, а также для студентов других специальностей ядерно-инжинирингового профиля, изучающих методы синтеза дискретных устройств управления ЯЭК и ЯОК.

Пособие подготовлено в рамках Программы создания и развития НИЯУ МИФИ.

Рецензент А.Н. Кармазинский, проф., доктор техн. наук (НИЯУ МИФИ)

ISBN-978-5-7262-1521-1

© Национальный исследовательский
ядерный университет «МИФИ», 2011

Содержание

ПРЕДИСЛОВИЕ	5
ГЛАВА 1. МАТЕМАТИЧЕСКИЕ МОДЕЛИ СИСТЕМ ДИСКРЕТНОЙ АВТОМАТИКИ. ЦИФРОВЫЕ АВТОМАТЫ	6
1.1. Математическая модель комбинационной схемы	6
1.2. Математическая модель абстрактного автомата.....	9
1.3. Основные способы задания автоматов	12
1.3. Связь между цифровыми автоматами Мили и Мура.....	17
1.4. Минимизация числа состояний полностью определённых автоматов.....	22
Контрольные вопросы и упражнения	25
ГЛАВА 2. СТРУКТУРНЫЙ СИНТЕЗ ЦИФРОВЫХ АВТОМАТОВ. КАНОНИЧЕСКИЙ МЕТОД СТРУКТУРНОГО СИНТЕЗА.....	26
2.1. Задача структурного синтеза.....	26
2.2. Канонический метод структурного синтеза.....	27
2.3. Элементарные цифровые автоматы. Элементы памяти	31
2.4. Пример канонического метода структурного синтеза автомата	36
2.5. Особенности синтеза автоматов на базе T-, RS-, JK-триггеров	41
2.6. Кодирование внутренних состояний. Гонки в автомате.....	44
2.7. Кодирование состояний и сложность комбинационной логической схемы автомата	48
Контрольные вопросы и упражнения	58
ГЛАВА 3. МИКРОПРОГРАММНОЕ УПРАВЛЕНИЕ ДИСКРЕТНЫМИ УСТРОЙСТВАМИ	59
3.1. Принцип микропрограммного управления	59
3.2. Структуризация дискретного устройства	60
3.3. Граф-схемы алгоритмов	62
3.4. Типовые операционные элементы	64
Контрольные вопросы и упражнения	68
ГЛАВА 4. СИНТЕЗ МИКРОПРОГРАММНЫХ АВТОМАТОВ ПО ГРАФ-СХЕМЕ АЛГОРИТМА	69
4.1 Этапы синтеза и построение таблиц переходов.....	69
4.2 Структурная таблица микропрограммного автомата	74

4.3 Построение схемы по структурной таблице	76
4.4 Задача факторизации	79
4.5 Декомпозиция схемы из однотипных элементов	83
Контрольные вопросы и упражнения	86
ГЛАВА 5. ЦИФРОВЫЕ АВТОМАТЫ НА	
ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ МАТРИЦАХ	87
5.1 Программируемые логические структуры	87
5.2 Тривиальная реализация микропрограммного автомата	88
5.3 Замена входных переменных	92
5.4 Кодирование микрокоманд	96
5.5 Декомпозиция автомата	102
Контрольные вопросы и упражнения	109
ГЛАВА 6. СИНТЕЗ ЦИФРОВЫХ АВТОМАТОВ	
НА РАСПРЕДЕЛИТЕЛЯХ СИГНАЛОВ	110
6.1 Распределители сигналов	110
6.2 Интерпретация линейных микропрограмм	112
6.3 Интерпретация микропрограмм, содержащих разветвления и циклы	115
Контрольные вопросы и упражнения	118
ГЛАВА 7. ЦИФРОВЫЕ АВТОМАТЫ С ХРАНИМОЙ	
В ПАМЯТИ МИКРОПРОГРАММОЙ	119
7.1 Структура микрокоманд. Общие положения	119
7.2 Постоянные запоминающие устройства	120
7.3 Структура и функционирование управляющего автомата ...	122
7.4 Кодирование микроопераций	124
7.5 Адресация микрокоманд	136
7.6 Сегментация постоянной памяти	149
7.7 Сегментация микропрограмм	153
Контрольные вопросы и упражнения	161
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ	163

Предисловие

Пособие охватывает основной круг вопросов, связанных с проектированием цифровых конечных автоматов – начиная с краткого и доступного для студентов изложения основ теории и заканчивая рассмотрением основных инженерных методов структурного синтеза цифровых автоматов. Изложение сопровождается большим количеством примеров, ориентированных на практическое применение.

В первой главе рассмотрены математические модели систем дискретной автоматики; основные способы задания автоматов; связь между автоматами Мили и Мура; минимизация числа состояний полностью определённых автоматов.

Во второй главе рассмотрен канонический метод структурного синтеза цифровых автоматов; особенности синтеза автоматов на триггерных устройствах различных типов; кодирование внутренних состояний и гонки в автомате; кодирование состояний и сложность комбинационной логической схемы.

В третьей главе рассмотрено микропрограммное управление дискретными устройствами: принцип микропрограммного управления; структуризация дискретных устройств; графический способ описания алгоритмов и программ.

В четвёртой главе рассмотрен синтез микропрограммных автоматов по граф-схеме алгоритма функционирования: этапы синтеза и построение таблиц переходов; построение схемы по структурной таблице; задача факторизации; декомпозиция схемы.

В пятой главе рассмотрен синтез цифровых автоматов на программируемых логических матрицах: тривиальная реализация; замена входных переменных; кодирование микрокоманд; декомпозиция автомата.

В шестой главе рассмотрен синтез автоматов на распределителях сигналов: интерпретация линейных микропрограмм; интерпретация микропрограмм, содержащих ветвления и циклы.

В седьмой главе рассмотрены цифровые автоматы с хранимой в памяти микропрограммой: структура и функционирование управляющего автомата; кодирование микроопераций; адресация микрокоманд; сегментация постоянной памяти; сегментация микропрограмм.

Глава 1. МАТЕМАТИЧЕСКИЕ МОДЕЛИ СИСТЕМ ДИСКРЕТНОЙ АВТОМАТИКИ. ЦИФРОВЫЕ АВТОМАТЫ

1.1. Математическая модель комбинационной схемы

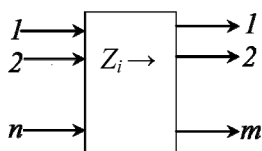


Рис. 1.1. Дискретное устройство

Рассмотрим некоторое дискретное устройство (рис. 1.1) с n входами и m выходами, на каждый вход которого может быть подан произвольный символ из конечного алфавита сигналов $X = \{x_1, x_2, \dots, x_k\}$, называемого *входным алфавитом*. Совокупность символов, поданных на вход устройства, образуют входное слово (*входной сигнал*) Z_i алфавита X нашего устройства. На выходе устройства появляются выходные слова (*выходные сигналы*), составленные из символов конечного *выходного алфавита* $Y = \{y_1, y_2, \dots, y_l\}$. В силу конечности как алфавитов X и Y , так и входных и выходных сигналов (длина входного сигнала всегда равна n , а выходного m) общее число различных входных и выходных сигналов конечно.

Элементарный такт работы устройства состоит в том, что при появлении на входе устройства входного сигнала Z_i устройство выдаёт на выходах комбинацию выходных символов, образующих выходной сигнал W_j .

Пусть работа устройства полностью определяется лишь входным сигналом. Тогда работа такого устройства будет определена, если мы зададим следующую таблицу соответствия для всех входных сигналов:

$$\left. \begin{array}{l} Z_1 \rightarrow W_{j1}, \\ Z_2 \rightarrow W_{j2}, \\ \dots\dots\dots \\ Z_{k^n} \rightarrow W_{jk^n}. \end{array} \right\} \quad (1.1)$$

В таблице имеется k^n строк по числу различных входных сигналов длины n в алфавите X , состоящем из k различных символов. Устройство, условия работы которого описываются при помощи

и, следовательно, может быть реализовано комбинационной схемой. Обычно комбинационная схема синтезируется на основе булевых функций, которые используются в качестве аналитической формы описания преобразования входных сигналов при построении синтезируемой структуры. Процедура синтеза состоит в следующем. Выбирается функционально полная система логических элементов. Функции (1.2) приводятся к функциональному базису выбранной системы элементов с использованием, например, методов функциональной декомпозиции и минимизируются с целью сокращения затрат оборудования. В схеме выделяются входы $x_1, x_2, \dots, x_n$, на которые поступают сигналы входных переменных. Организация соединений логических элементов производится в порядке, принятом для вычисления значений булевых функций. Этот порядок определяется старшинством логических операций и скобками, в аналитической записи функций.

Например, совместно минимизируя функции (1.3), можно получить следующие скобочные формы:

$$\left. \begin{aligned} y_1 &= x_1 x_2 \vee (x_1 \vee x_2) x_3, \\ y_2 &= x_1 x_2 x_3 \vee y_1 (x_1 \vee x_2 \vee x_3), \end{aligned} \right\}$$

по которым построена комбинационная логическая схема (КЛС) (рис. 1.2).

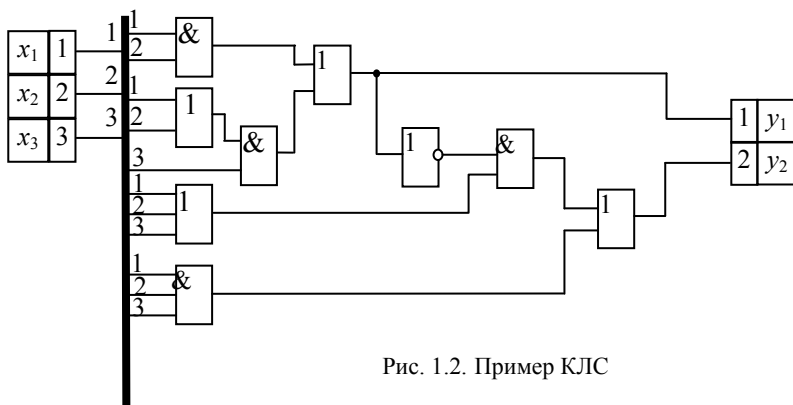


Рис. 1.2. Пример КЛС

Логическая схема называется комбинационной, если её функцию можно представить в виде (1.2). Логическая схема допускает

такое представление, если в ней отсутствуют петли (цепи обратной связи). Под петлёй понимается замкнутая цепь в схеме, по которой сигнал с выхода i -го элемента непосредственно или проходя через некоторую последовательность элементов схемы поступает на вход того же i -го элемента. Схема, содержащая петли, не является комбинационной.

Сложность схемы характеризуется суммарной стоимостью используемых в ней элементов:

$$C = \sum_{i=1}^n k_i c_i, \quad (1.4)$$

где k_i – количество элементов i -го типа, используемых в схеме, c_i – цена элемента. Эта оценка сложности схемы используется при разработке конкретных схем, когда известна стоимость элементов.

В теоретических исследованиях сложность схемы принято характеризовать ценой по Квайну. *Цена схемы* определяется суммарным числом входов элементов, составляющих схему. Например, цена схемы, изображённой на рис. 1.2, равна 19. Эта оценка является объективной характеристикой структуры.

Из-за того, что входы элементов разных типов уравниваются в цене, сопоставление вариантов схемы на основе оценок (1.4) и по Квайну может приводить к противоречивым результатам. Но при использовании транзисторной логики и интегральной технологии данное противоречие сглаживается, что увеличивает правдоподобность оценок сложности схем по Квайну.

1.2. Математическая модель абстрактного автомата

Существует большой класс дискретных устройств, работу которых невозможно адекватно определить лишь прямым соответствием входных и выходных сигналов $W(t) = \Phi\{Z(t)\}$, $t = 0, 1, 2, 3, \dots$. Подразумевается, что значение выходного сигнала зависит не только от значения входного сигнала в данный момент времени, но и от предыстории развития входных сигналов, начиная с нулевого момента времени:

$$W(t) = \Phi \{ Z(t), Z(t-1), \underbrace{Z(t-2), \dots, Z(2), Z(1), Z(0)}_{A(t-2)}, \underbrace{A(t-1)}_{A(t)} \}. \quad (1.5)$$

Для сохранения динамики развития входных сигналов во времени введём некоторую функцию памяти $A(t)$, отображающую реакцию дискретного устройства на воздействие входного сигнала $Z(t)$ и переход этого устройства в иное состояние в зависимости от значения функции памяти в предыдущий момент времени $A(t-1)$: $A(t) = \{Z(t), A(t-1)\}$. Тогда с учётом сказанного выражение (1.5) можно представить в виде системы уравнений

$$\begin{cases} A(t) = F^\delta \{Z(t), A(t-1)\}, \\ W(t) = F^\lambda \{Z(t), A(t-1)\} \end{cases} \quad (1.6)$$

или в виде

$$\begin{cases} A(t) = F^\delta \{Z(t), A(t-1)\}, \\ W(t) = F^\lambda \{A(t)\}. \end{cases} \quad (1.7)$$

Исходя из математической трактовки выражения (1.5) обе системы уравнений вполне корректны, а вот семантическая трактовка существенно отличается.

Математическая модель дискретного устройства в виде системы уравнений (1.6) называется *абстрактным автоматом 1-го рода, или автоматом Мили (Mealy)*. Здесь выходной сигнал (выходная реакция устройства) двумерный и в явном виде зависит от двух параметров: значения входного сигнала (входного воздействия в момент времени t) и состояния функции памяти в предыдущий момент времени. В зависимости от состояния устройства при одном и том же входном воздействии можно получать различные выходные сигналы.

Математическая модель дискретного устройства в виде системы уравнений (1.7) называется *абстрактным автоматом 2-го рода, или автоматом Мура (Moore)*. Из сравнения законов функционирования видно, что в отличие от автомата Мили выходной сиг-

нал в автомате Мура зависит только от текущего состояния автомата и в явном виде не зависит от входного сигнала.

Наибольшее распространение при описании законов функционирования абстрактных автоматов получил множественный подход, когда автомат задаётся шестёркой объектов:

$$S = \{A, Z, W, \delta, \lambda, a_1\}, \quad (1.8)$$

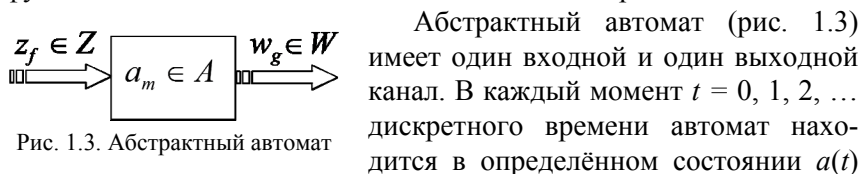
где $A = \{a_1, \dots, a_m, \dots, a_M\}$ – перечисленное множество состояний (алфавит состояний); $Z = \{z_1, \dots, z_f, \dots, z_F\}$ – перечисленное множество входных сигналов (входной алфавит); $W = \{w_1, \dots, w_g, \dots, w_G\}$ – перечисленное множество выходных сигналов (выходной алфавит); δ – функция переходов, реализующая отображение множества $D_\delta \subseteq A \times Z$ на множество A ($a_s = \delta(a_m, z_f)$, $a_s \in A$); λ – функция выходов, реализующая отображение множества $D_\lambda \subseteq A \times Z$ на множество W ($w_g = \lambda(a_m, z_f)$) для автомата Мили или множества A на множество W ($w_g = \lambda(a_m)$) для автомата Мура; $a_1 \in A$ – начальное состояние автомата.

Абстрактный автомат, заданный подобным образом, будем называть *цифровым автоматом* (цифровым абстрактным автоматом). Автомат называется *конечным*, если конечны множества A , Z и W . В дальнейшем будут рассматриваться только конечные цифровые автоматы и термин «конечный», также как и «цифровой», почти всегда будут опускаться.

Автомат называется *полностью определённым*, если $D_\delta = D_\lambda = A \times Z$. Иными словами, у полностью определённого автомата области определения функций δ и λ совпадают со множеством $A \times Z$ – множеством всевозможных пар вида (a_m, z_f) . У частично-го автомата функции δ или λ определены не для всех пар $(a_m, z_f) \in A \times Z$.

Понятие состояния в определении автомата как раз и введено в связи с необходимостью описания поведения систем, выходные сигналы которых зависят не только от состояния входных сигналов в данный момент времени, но и от некоторой предыстории, т.е. от сигналов, которые поступали на входы системы ранее. Состояния

соответствуют некоторой памяти о прошлом, позволяя устранить время как явную переменную и выразить выходные сигналы как функцию состояний и входов в данный момент времени.



из множества A состояний автомата, причём в начальный момент $t = 0$ он всегда находится в начальном состоянии $a(0) = a_1$. В момент t , будучи в состоянии $a(t)$, автомат способен воспринять на входе сигнал $z(t) \in Z$ как входное воздействие и выдать на выходе некоторую реакцию в виде выходного сигнала $w(t) = \lambda(a(t), z(t))$, переходя в состояние $a(t+1) = \delta(a(t), z(t))$; $a(t) \in A$. Сущность понятия цифрового абстрактного автомата состоит в том, что он реализует некоторое отображение множества слов входного алфавита Z во множество слов выходного алфавита W . Другими словами, если на вход автомата, установленного в начальное состояние a_1 подавать буква за буквой некоторую последовательность букв входного алфавита $z(0), z(1), z(2), \dots$ – входное слово, то на выходе автомата будут последовательно появляться буквы выходного алфавита $w(0), w(1), w(2), \dots$ – выходное слово. Относя к каждому входному слову соответствующее ему выходное слово, мы получим отображение ϕ , индуцированное абстрактным автоматом. Исходя из сказанного закон функционирования автомата Мили задаётся уравнениями

$$a(t+1) = \delta(a(t), z(t)); w(t) = \lambda(a(t), z(t)), t = 0, 1, 2, \dots, \quad (1.9)$$

а закон функционирования автомата Мура – уравнениями

$$a(t+1) = \delta(a(t), z(t)); w(t) = \lambda(a(t)), t = 0, 1, 2, \dots \quad (1.10)$$

1.3. Основные способы задания автоматов

Чтобы задать конечный автомат S , необходимо описать все элементы множества $S = \{A, Z, W, \delta, \lambda, a_1\}$, т. е. входной и выходной алфавиты и алфавит состояний, а также функции переходов и

выходов. Среди множества состояний необходимо выделить состояние a_1 , в котором автомат находится в момент $t = 0$. Существует несколько способов задания работы автомата, но наиболее часто используются *табличный* и *графический*.

При табличном способе задания автомат Мили описывается с помощью двух таблиц. Одна из них (таблица переходов) задаёт функцию δ , т.е. $a(t+1) = \delta(a(t), z(t))$, вторая (таблица выходов) – функцию λ , т.е. $w(t) = \lambda(a(t), z(t))$.

Описание работы автомата Мили таблицами переходов и выходов иллюстрируется табл. 1.1 и 1.2.

Таблица 1.1

Общий вид таблицы переходов автомата Мили

	a_1		a_M
z_1	$\delta(a_1, z_1)$	...	$\delta(a_M, z_1)$
...	...	...	...
z_f	$\delta(a_M, z_f)$	...	$\delta(a_M, z_f)$

Таблица 1.2

Общий вид таблицы выходов автомата Мили

	a_1		a_M
z_1	$\lambda(a_M, z_f)$	...	$\lambda(a_M, z_1)$
...	...	...	...
z_f	$\lambda(a_M, z_f)$	...	$\lambda(a_M, z_f)$

Строки этих таблиц соответствуют входным сигналам, а столбцы – состояниям, причём крайний левый столбец состояний обозначен начальным состоянием a_1 . На пересечении столбца a_m и строки z_f в таблице переходов ставится состояние $a_s = \delta(a_m, z_f)$, в которое автомат переходит из состояния a_m под действием сигнала z_f , а в таблице выходов – соответствующий этому переходу выходной сигнал $w_g = \lambda(a_m, z_f)$. Пример табличного способа задания полностью определённого автомата Мили S_1 с тремя состояниями, двумя входными и двумя выходными сигналами приведён в таблице 1.3 и 1.4.

Таблица 1.3

Таблица переходов автомата Мили S_1

	a_1	a_2	a_3
z_1	a_3	a_1	a_1
z_2	a_1	a_3	a_2

Таблица 1.4

Таблица выходов автомата Мили S_1

	a_1	a_2	a_3
z_1	w_1	w_1	w_2
z_2	w_1	w_2	w_1

Для частичных автоматов, у которых функции δ или λ определены не для всех пар $(a_m, z_f) \in A \times Z$, на месте неопределённых состояний и выходных сигналов ставится прочерк (частичный автомат S_2 задан табл. 1.5 и 1.6).

Таблица 1.5

Таблица переходов частичного автомата Мили S_2

	a_1	a_2	a_3	a_4
z_1	a_2	a_3	a_4	—
z_2	a_3	—	a_2	a_2

Таблица 1.6

Таблица выходов частичного автомата Мили S_2

	a_1	a_2	a_3	a_4
z_1	w_1	w_3	w_3	—
z_2	w_2	—	w_1	w_2

Так как в автомате Мура выходной сигнал зависит только от состояния, то автомат Мура задаётся одной отмеченной таблицей переходов (табл. 1.7), в которой каждому её столбцу приписан, кроме состояния a_m , ещё и выходной сигнал $w_g = \lambda(a_m)$, соответст-

вующий этому состоянию. Пример табличного описания автомата Мура S_3 иллюстрируется табл. 1.8.

Таблица 1.7

Общий вид отмеченной таблицы переходов автомата Мура

	$\lambda(a_1)$	...	$\lambda(a_M)$
	a_1	...	a_M
z_1	$\delta(a_1, z_1)$	...	$\delta(a_M, z_1)$
...	...	...	...
z_f	$\delta(a_M, z_f)$	...	$\delta(a_M, z_f)$

Таблица 1.8

Отмеченная таблица переходов автомата Мура S_3

	w_1	w_1	w_3	w_2	w_3
	a_1	a_2	a_3	a_4	a_5
z_1	a_2	a_5	a_5	a_3	a_3
z_2	a_4	a_2	a_2	a_1	a_1

При графическом способе автомат задаётся в виде ориентированного связного графа, вершины которого соответствуют состояниям, а дуги – переходам между ними. Две вершины графа автомата a_m и a_s (исходное состояние и состояние перехода) соединяются дугой, направленной от a_m к a_s , если в автомате имеется переход из a_m в a_s , т.е. если $a_s = \delta(a_m, z_f)$ при некотором $z_f \in Z$. Дуге (a_m, a_s) графа автомата приписывается входной сигнал z_f и выходной сигнал $w_g = \lambda(a_m, z_f)$. При описании автомата Мура в виде графа выходной сигнал $w_g = \lambda(a_m)$ записывается внутри вершины a_m . На рис. 1.4–1.6 приведены заданные ранее таблицами графы автоматов S_1, S_2, S_3 .

Состояние a_s автомата S называется устойчивым состоянием, если для любого входа $z_f \in Z$, такого, что $\delta(a_m, z_f) = a_s$, имеет место $\delta(a_s, z_f) = a_s$. Если же существует переход $\delta(a_m, z_f) = a_s$ и под

действием того же самого входного сигнала есть переход в другое состояние $a_k = \delta(a_s, z_f)$, то состояние $a_s \in A$ является не устойчивым.

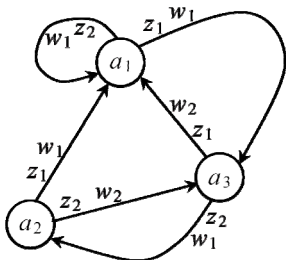


Рис. 1.4. Граф автомата Мили S_1

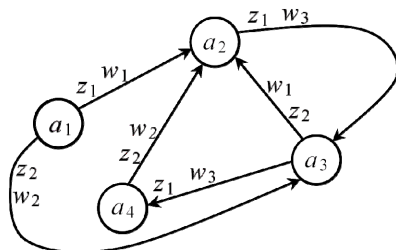


Рис. 1.5. Граф автомата Мили S_2

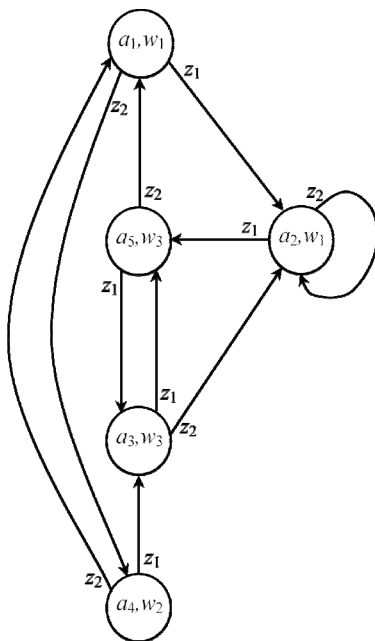


Рис. 1.6. Граф автомата Мура S_3

Автомат S называется *асинхронным*, если каждое его состояние $a_s \in A$ устойчиво. Автомат S называется *синхронным*, если у автомата существует хотя бы одно не устойчивое состояние. Необходимо заметить, что построенные на практике автоматы — всегда асинхронные и устойчивость их состояний всегда обеспечивается тем или иным способом, например, введением двух серий синхронизирующих сигналов (c_1 и c_2) при условии, что $c_1 \& c_2 = 0$. Однако на уровне абстрактной теории, когда автомат есть лишь математическая модель, которая не отражает многих конкретных особенностей его возможной реализации, часто оказывается более удобным оперировать с синхронными

автоматами. Анализ графов автоматов на рис. 1.4–1.6 показывает, что автоматы S_1 , S_2 и S_3 являются синхронными.

1.4. Связь между цифровыми автоматами Мили и Мура

Цифровой абстрактный автомат фактически является универсальным преобразователем слов входного алфавита в слова выходного алфавита. Рассмотрим это более подробно, взяв в качестве примера автомат Мили, заданный таблицами переходов и выходов (табл. 1.9 и 1.10).

Таблица 1.9

$$\delta : A \times Z \rightarrow A$$

	a_1	a_2	a_3
z_1	a_2	a_3	a_3
z_2	a_3	a_1	a_1

Подадим на вход автомата, установленного в состояние a_1 , входное слово $\xi = z_1 z_2 z_2 z_1 z_2 z_2$. Так как $\delta(a_1, z_1) = a_2$ и $\lambda(a_1, z_1) = w_2$, то под воздействием входного сигнала z_1 автомат перейдёт в состояние a_2 и выдаст на переходе выходной сигнал w_2 . Затем, находясь в состоянии a_2 под воздействием сигнала z_2 перейдёт в состояние $a_1 = \delta(a_2, z_2)$ и выдаст сигнал $w_1 = \lambda(a_2, z_2)$ и так далее.

Таблица 1.10

$$\lambda : A \times Z \rightarrow W$$

	a_1	a_2	a_3
z_1	w_2	w_1	w_2
z_2	w_2	w_1	w_1

В табл. 1.11 приведена последовательность состояний, которые автомат проходит, воспринимая входное слово ξ , и выходные сигналы, вырабатываемые на этих переходах.

Таблица 1.11

Реакция автомата Мили

Последовательность состояний	a_1	a_2	a_1	a_3	a_3	a_1	a_3
Входное слово ξ	z_1	z_2	z_2	z_1	z_2	z_2	
Выходное слово ω	w_2	w_1	w_2	w_2	w_1	w_2	

Назовём выходное слово $\omega = \lambda(a_1, \xi)$ *реакцией* автомата Мили в состоянии a_1 на входное слово ξ . В приведённом примере реакцией является $\omega = w_2 w_1 w_2 w_2 w_1 w_2$. Как видно из примера, в ответ на

входное слово длины k автомат Мили выдаст последовательность состояний длины $k + 1$ и выходное слово длины k .

В общем виде поведение автомата Мили, установленного в состояние a_m , описывается в табл. 1.12.

Таблица 1.12

Общий вид реакции автомата Мили

Входное слово	z_{i1}	z_{i2}	z_{i3}
Последовательность состояний	a_m	$a_{i2} = \delta(a_m, Z_{i1})$	$a_{i3} = \delta(a_{i2}, Z_{i2})$
Выходное слово	$w_{i1} = \lambda(a_m, Z_{i1})$	$w_{i2} = \lambda(a_{i2}, Z_{i2})$	$w_{i3} = \lambda(a_{i3}, Z_{i3})$

Аналогично можно описать поведение автомата Мура, находящегося в состоянии a_m , при приходе входного слова $\xi = z_{i1} z_{i2} \dots z_{ik}$, учитывая, что $w(t) = \lambda(a(t))$ (табл. 1.13).

Таблица 1.13

Общий вид реакции автомата Мура

Входное слово	z_{i1}	z_{i2}	z_{i3}	z
Последовательность состояний	a_m	$a_{i2} = \delta(a_m, z_{i1})$	$a_{i3} = \delta(a_{i2}, z_{i2})$	$a_{i4} = \delta(a_{i3}, z_{i3})$
Выходное слово	$w_{i1} = \lambda(a_m)$	$w_{i2} = \lambda(a_{i2})$	$w_{i3} = \lambda(a_{i3})$	$w_{i4} = \lambda(a_{i4})$

Исходя из определения автомата Мура выходной сигнал $w_{i1} = \lambda(a_m)$ в момент времени $i1$ не зависит от входного сигнала z_{i1} и определяется только состоянием a_m . Следовательно, сигнал w_{i1} никак не связан с входным словом ξ .

В связи с этим под реакцией автомата Мура, установленного в состояние a_m , на входное слово $\xi = z_{i1} z_{i2} \dots z_{ik}$ будем понимать выходное слово той же длины $\omega = \lambda(a_m, \xi) = w_{i2} w_{i3} \dots w_{ik+1}$, сдвинутое по отношению к ξ на один такт автоматного времени.

Рассмотрим пример. Пусть задан автомат Мура (табл. 1.14). Подадим на вход этого автомата последовательность, как и для автомата Мили: $\xi = z_1 z_2 z_2 z_1 z_2 z_2$.

Последовательность смены состояний и вырабатываемых выходных сигналов представлена в табл. 1.15.

Таблица 1.14

Отмеченная таблица переходов автомата Мура

	w_1	w_2	w_1	w_2
	a_1	a_2	a_3	a_4
z_1	a_2	a_3	a_4	a_4
z_2	a_4	a_1	a_1	a_1

Таблица 1.15

Реакция автомата Мура

Последовательность состояний	
Входное слово ξ	
Выходное слово ω	

$\omega = \lambda(a_m, \xi)$

Сравнивая реакции автомата Мили и автомата Мура, отмечаем, что эти реакции на одно и то же слово ξ совпадают. Следовательно, автоматы Мили и Мура реализуют одно и то же преобразование слов входного алфавита. Такие автоматы называются *эквивалентными*. Определение эквивалентности следующее: *два автомата с одинаковыми входными и выходными алфавитами называются эквивалентными, если после установки их в начальное состояние их реакции на любое входное слово совпадают*.

Для каждого автомата Мили может быть построен эквивалентный ему автомат Мура и наоборот. Переход от автомата Мура к эквивалентному ему автомату Мили тривиален и легко осуществляется при графическом способе задания автомата. Для получения графа автомата Мили необходимо выходной сигнал w_g ,

записанный в вершине a_s исходного автомата Мура, перенести на все дуги, входящие в эту вершину. На рис. 1.7 приведён граф автомата Мили, эквивалентного автомату Мура S_3 (см. рис. 1.6).

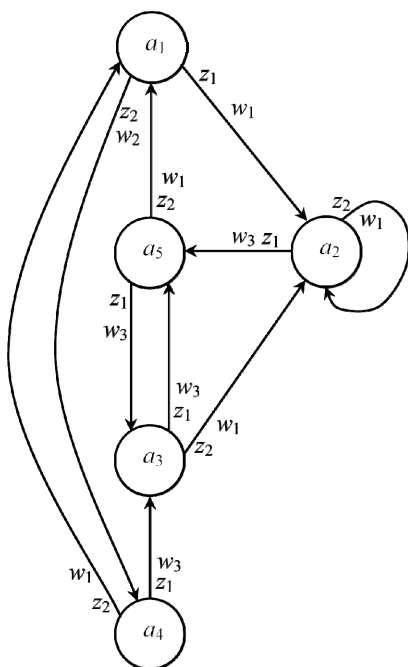


Рис. 1.7. Граф автомата Мили, эквивалентного автомату Мура S_3

Легко убедиться, что полученный автомат Мили действительно эквивалентный исходному автомату Мура. Для этого достаточно рассмотреть реакцию обоих автоматов на произвольную входную последовательность. Необходимо также отметить, что в эквивалентном автомате Мили количество состояний такое же, как и в исходном автомате Мура.

Переход от автомата Мили к эквивалентному ему автомату Мура несколько сложнее. Это связано с тем, что в автомате Мура в каждом состоянии вырабатывается только один выходной сигнал. Как и в предыдущем случае, переход наиболее наглядно делать при графическом способе задания

автомата. В этом случае каждое состояние a_i исходного автомата Мили порождает столько состояний автомата Мура, сколько различных выходных сигналов вырабатывается в исходном автомате при попадании в состояние a_i . Рассмотрим переход от автомата Мили S_a к автомату Мура S_b на примере автомата S_2 (см. рис. 1.5).

Как следует из рис. 1.5, для автомата S_2 перехода в состояние a_1 не существует (состояние преходящее, выходной сигнал не определён и обозначим его как w_0), а при переходе в состояние a_2 вырабатываются выходные сигналы w_1, w_2 , при переходе в $a_3 \rightarrow w_2, w_3$; $a_4 \rightarrow w_3$. Каждой паре (a_i, w_j) , где a_i – состояние, а w_j –

выходной сигнал, который вырабатывается при переходе в это состояние, поставим в соответствие состояние b_k эквивалентного автомата Мура Sb с тем же выходным сигналом w_j : $b_1 = (a_1, w_0)$, $b_2 = (a_2, w_1)$, $b_3 = (a_2, w_2)$, $b_4 = (a_3, w_2)$, $b_5 = (a_3, w_3)$, $b_6 = (a_4, w_3)$. Таким образом, каждое состояние a_i автомата Мили порождает некоторое множество A_i состояний эквивалентного автомата Мура: $A_1 = \{b_1\}$, $A_2 = \{b_2, b_3\}$, $A_3 = \{b_4, b_5\}$, $A_4 = \{b_6\}$. Как видно, в эквивалентном автомате Мура количество состояний 6. Для построения графа Sb поступаем следующим образом. Так как в автомате Мили Sa есть переход из состояния a_2 в состояние a_3 под действием сигнала z_1 с выдачей w_3 , то из множества состояний $A_2 = \{b_2, b_3\}$, порождаемых состоянием a_2 автомата Sa , в автомате Sb должен быть переход в состояние $(a_3, w_3) = b_5$ под действием сигнала z_1 и т.д.

Граф эквивалентного автомата Мура представлен на рис. 1.8.

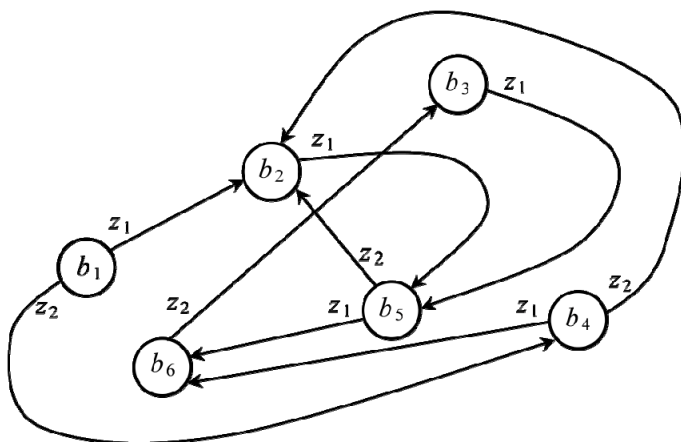


Рис.1.8. Граф автомата Мура, эквивалентного автомату Мили S_2 на рис. 1.5

Эквивалентность автоматов Sb и Sa при преобразовании автомата Мили в автомат Мура на множестве входных слов конечной длины легко доказать сравнением реакций обоих автоматов.

Изложенные процедуры взаимной транспозиции моделей Мили и Мура показывают, что при переходе от автомата Мура к автомату Мили число состояний автоматов не меняется, тогда как при

обратном переходе число состояний в автомате Мура, как правило, возрастает. Если, например, от автомата Мура (см. рис. 1.8), эквивалентного автомату Мили S_2 (см. рис. 1.5), перейти вновь к автомату Мили, то получим другой автомат Мили с шестью состояниями. Вследствие транзитивности отношения эквивалентности два автомата Мили также будут эквивалентны, но у последнего на два состояния будет больше. Таким образом, эквивалентные между собой автоматы могут иметь различное число состояний, в связи с чем возникает задача нахождения минимального (т.е. с минимальным числом состояний) автомата в классе эквивалентных между собой автоматов. Существование для любого абстрактного автомата S эквивалентного ему абстрактного автомата $S_{\min}$ с минимальным числом внутренних состояний впервые было доказано Муром.

1.5. Минимизация числа состояний полностью определённых автоматов

Рассмотрим метод минимизации полностью определённых абстрактных автоматов Мили, предложенный Д.Д. Ауфенкампом и Ф.Е. Хоном в 1957 г. [1,2]. Основная идея этого метода заключается в разбиении всех состояний исходного абстрактного автомата на попарно непересекающиеся классы эквивалентных состояний и замене каждого класса эквивалентности одним состоянием. Таким образом, получающийся в результате минимальный автомат имеет столько состояний, на сколько классов эквивалентности разбиваются состояния исходного автомата.

Для пользования методом введём несколько определений.

Два состояния абстрактного автомата называются 1-эквивалентными в том случае, если реакции автомата в этих состояниях на всевозможные входные слова совпадают.

Объединение всех 1-эквивалентных состояний абстрактного автомата образует *1-й класс эквивалентности*.

Состояния автомата 1-эквивалентные называются 2-эквивалентными, если они переводятся любым входным сигналом также в 1-эквивалентные состояния.

Объединение всех 2-эквивалентных состояний образует 2-й класс эквивалентности.

По индукции можно распространить определение до k -эквивалентных состояний и k -го класса эквивалентности.

Если для некоторого k -го разбиения состояний автомата на $(k + 1)$ классы совпадает с разбиением на k классы, то оно является разбиением и на ∞ -й класс эквивалентности.

Разбиение множества внутренних состояний автомата на ∞ -й класс эквивалентности и является требуемым разбиением на классы эквивалентности. При этом такое разбиение может быть получено за конечное число шагов.

Все вышеизложенное непосредственно применимо к минимизации автомата Мили. При минимизации полностью определённых автоматов Мура дополнительно вводится понятие 0-эквивалентности состояний и разбиение множества состояний на 0-эквивалентные классы: к такому классу относятся одинаково отмеченные состояния автомата Мура.

Если два 0-эквивалентных состояния любым входным сигналом переводится в два 0-эквивалентных состояния, то они называются 1-эквивалентными. Все дальнейшие классы эквивалентности состояний для автомата Мура определяются аналогично приведённому для автоматов Мили.

Рассмотрим пример минимизации автомата Мили S , заданного таблицами переходов и выходов (табл. 1.13).

Из таблицы выходов получаем разбиение на 1-е классы эквивалентности π_1 , объединяя в эквивалентные классы B_i состояния с одинаковыми столбцами:

$$\pi_1 = \{B_1, B_2\}; \quad B_1 = \{a_1, a_2, a_5, a_6\}; \quad B_2 = \{a_3, a_4\}.$$

Для получения 2-эквивалентных состояний строим таблицу π_1 разбиения (табл. 1.14), заменяя в таблице переходов состояния a_i соответствующими классами эквивалентности B_1 или B_2 .

Таблица 1.13

Автомат Мили S

$$\lambda : A \times Z \rightarrow W$$

	a_1	a_2	a_3	a_4	a_5	a_6
z_1	w_1	w_1	w_1	w_1	w_1	w_1
z_2	w_1	w_1	w_2	w_2	w_1	w_1

$$\delta : A \times Z \rightarrow A$$

	a_1	a_2	a_3	a_4	a_5	a_6
z_1	a_3	a_4	a_3	a_4	a_5	a_6
z_2	a_5	a_6	a_5	a_6	a_1	a_2

Таблица 1.14

Таблица π_1 разбиения

	B ₁				B ₂	
	a ₁	a ₂	a ₅	a ₆	a ₃	a ₄
z ₁	B ₂	B ₂	B ₁	B ₁	B ₂	B ₂
z ₂	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁

Из полученной таблицы π_1 разбиения получаем 2-е классы эквивалентности C_i и разбиение $\pi_2 = \{C_1, C_2, C_3\}$, где $C_1 = \{a_1, a_2\}$, $C_2 = \{a_5, a_6\}$, $C_3 = \{a_3, a_4\}$. Сравнивая π_2 и π_1 , отмечаем, что эти разбиения отличаются друг от друга. Поэтому аналогично строим таблицу π_2 разбиения (таблица 1.15), опять заменяя в таблице переходов состояния a_i соответствующими классами эквивалентности C_i .

Таблица 1.15

Таблица π_2 разбиения

	C ₁		C ₂		C ₃	
	a ₁	a ₂	a ₅	a ₆	a ₃	a ₄
z ₁	C ₃	C ₃	C ₂	C ₂	C ₃	C ₃
z ₂	C ₂	C ₂	C ₁	C ₁	C ₂	C ₂

Из полученной таблицы π_2 разбиения получаем 3-и классы эквивалентности D_i и разбиение $\pi_3 = \{D_1, D_2, D_3\}$, где $D_1 = \{a_1, a_2\}$, $D_2 = \{a_5, a_6\}$, $D_3 = \{a_3, a_4\}$.

Сравнивая π_3 и π_2 , замечаем, что $D_1 = C_1$, $D_2 = C_2$, $D_3 = C_3$, $\pi_3 = \pi_2$. Следовательно, получили разбиение на ∞ -й класс эквивалентности. Так как всего три таких класса,

то минимальный автомат будет содержать всего три состояния. Выбираем из каждого класса D_i по одному состоянию и получаем множество состояний A' минимального автомата. Пусть, например, $A' = \{a_1, a_4, a_5\}$. Для получения минимального автомата из первоначальных таблиц переходов и выходов (см. табл. 1.13) вычёркиваем столбцы, соответствующие "лишним состояниям" a_2, a_3, a_6 . В результате получается минимальный автомат Мили, эквивалентный исходному автомату (табл. 1.16).

Таблица 1.16

Таблицы переходов и выходов минимального автомата

$$\delta : A \times Z \rightarrow A$$

	a ₁	a ₄	a ₅
z ₁	a ₄	a ₄	a ₅
z ₂	a ₅	a ₅	a ₁

$$\lambda : A \times Z \rightarrow W$$

	a ₁	a ₄	a ₅
z ₁	w ₁	w ₁	w ₁
z ₂	w ₁	w ₂	w ₁

Минимизацией числа внутренних состояний автомата и заканчивается этап абстрактного синтеза.

Контрольные вопросы и упражнения

1. В чём заключается принципиальное различие математических моделей автомата Мили и автомата Мура?
2. Что является объектами описания цифрового конечного автомата?
3. Перечислите базовые способы задания абстрактных автоматов.
4. Являются ли словами следующие комбинации символов: «ко-рабль»; «aaaa»; «2345», «1a25ABC».
5. Дайте определение эквивалентности автоматов.
6. Определите максимальное число состояний автомата Мили при эквивалентном преобразовании автомата Мура, имеющего три состояния и четыре входных сигнала.
7. Определите максимальное число состояний автомата Мура при эквивалентном преобразовании автомата Мили, имеющего три состояния и четыре входных сигнала.
8. Объясните основную идею метода минимизации числа состояний полностью определённых абстрактных автоматов, предложенного Ауфенкампом и Хоном.

Глава 2. СТРУКТУРНЫЙ СИНТЕЗ ЦИФРОВЫХ АВТОМАТОВ. КАНОНИЧЕСКИЙ МЕТОД СТРУКТУРНОГО СИНТЕЗА

2.1. Задача структурного синтеза

Вслед за этапом абстрактного синтеза автоматов следует этап структурного синтеза, целью которого является построение схемы, реализующей автомат на базе функционально полной системы логических элементов. Если абстрактный автомат был лишь математической моделью проектируемого устройства, то при структурном синтезе автомата учитывается форма представления входных и выходных сигналов автомата, а также его внутреннее устройство на уровне логических схем. Основной задачей структурной теории автоматов является разработка базовых методов построения структурных схем автоматов.

В отличие от абстрактного представления автомата, имеющего один вход и один выход, на которые поступают сигналы во *входном* $Z = \{z_1, \dots, z_f, \dots, z_F\}$ и выходят в *выходном* $W = \{w_1, \dots, w_g, \dots, w_G\}$ алфавитах, при структурном представлении автомата вводятся n входов $x_1, x_2, \dots, x_n$ и m выходов $y_1, y_2, \dots, y_m$, на каждом из которых присутствует сигнал в структурном алфавите, мощность которого значительно меньше мощностей входного и выходного алфавитов абстрактного автомата (рис. 2.1).

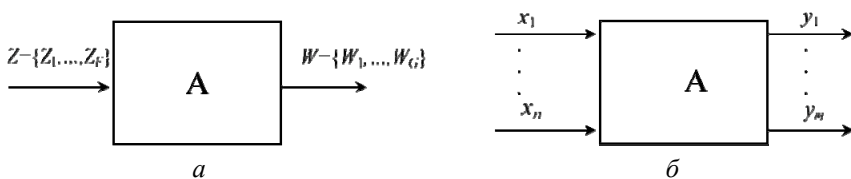


Рис. 2.1. Абстрактное (а) и структурное (б) представления автомата

Каждый входной z_f и выходной w_g сигналы абстрактного автомата могут быть представлены (закодированы) набором состояний входных и выходных сигналов при структурном представлении автомата.

Обычно для представления структурных сигналов используется двоичный алфавит: $x_i \in \{0,1\}$; $y_i \in \{0,1\}$. В этом случае каждому входному сигналу z_f абстрактного автомата соответствует некоторый двоичный вектор $(x_{f1}, x_{f2}, \dots, x_{fn})$, где $x_{fj} \in \{0,1\}$, а каждому выходному сигналу w_g автомата соответствует двоичный вектор $(y_{g1}, y_{g2}, \dots, y_{gm})$, где $y_{gj} \in \{0,1\}$.

Рассмотрим структуризацию входных и выходных сигналов. Очевидно, что для представления (кодирования) входных сигналов $z_1, \dots, z_F$ абстрактного автомата различными двоичными векторами должно быть выполнено условие $n \leq \lceil \log_2 F \rceil$, аналогично $m \geq \lceil \log_2 G \rceil$. Например, $Z = \{z_1, z_2, z_3, z_4\}$; $W = \{w_1, w_2, w_3\}$. Тогда $n \leq \lceil \log_2 4 \rceil = 2$, а $m \geq \lceil \log_2 3 \rceil = 2$.

Закодировать входные и выходные сигналы можно, например, таким образом:

$z_1 = 00$	$w_1 = 00$
$z_2 = 01$	$w_2 = 01$
$z_3 = 10$	$w_3 = 11$
$z_4 = 11$	

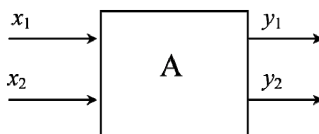


Рис. 2.2. Пример структуры автомата

Следовательно, структура автомата с двумя структурными входами x_1, x_2 и двумя структурными выходами y_1, y_2 может быть представлена в виде, показанном на рис. 2.2.

На этапе структурного синтеза предварительно выбираются элементарные автоматы, путём композиции которых строят логические схемы полученных на этапе абстрактного синтеза автоматов Мили или Мура. Для решения задачи структурного синтеза существует общий конструктивный приём, так называемый канонический метод структурного синтеза.

2.2. Канонический метод структурного синтеза

Рассмотрим канонический метод структурного синтеза, при котором используются элементарные автоматы, имеющие более одного состояния, и элементарные автоматы с одним состоянием (тривиальные автоматы). Первые автоматы называются *элементарными памяти*, вторые – *комбинационными логическими элементами*.

Теоретическим обоснованием канонического метода структурного синтеза автоматов служит теорема Квайна о структурной полноте системы синтеза.

Теорема о структурной полноте системы синтеза: всякая система элементарных автоматов, которая содержит нетривиальный автомат Мура, обладающий полнотой системы переходов и полнотой системы выходов, и какую-либо функционально полную систему логических элементов, является *структурно полной*.

В этом случае задача структурного синтеза конечных автоматов сводится к задаче структурного синтеза комбинационных логических схем.

Результатом канонического метода структурного синтеза является система логических уравнений, выражающая зависимость выходных сигналов автомата и сигналов, подаваемых на входы элементов памяти, от сигналов, приходящих на вход всего автомата в целом, и сигналов, снимаемых с выхода элементов памяти.

Автомат обладает *полнотой системы переходов*, если для любой упорядоченной пары состояний автомата (a_i, a_j) найдётся входной сигнал z_f , переводящий автомат из состояния a_i в состояние a_j . Это условие должно соблюдаться как при $i \neq j$, так и при $i = j$. В таком автомате в каждом столбце таблицы переходов должны встречаться все состояния автомата.

Автомат Мура обладает *полнотой системы выходов*, если каждому состоянию автомата поставлен в соответствие свой выходной сигнал, отличный от выходных сигналов других состояний. В таком автомате число выходных сигналов равно числу состояний автомата.

В связи с этим в автоматах памяти будем использовать одни и те же обозначения и для состояний, и для выходных сигналов.

Канонический метод структурного синтеза предполагает представление структурной схемы автомата в виде двух частей: памяти и комбинационной логической схемы (рис. 2.3).

Память состоит из элементарных автоматов Мура $\Pi_1, \Pi_2, \dots, \Pi_R$. Каждое состояние синтезируемого автомата $a_i \in A$ кодируется набором их состояний. Если все автоматы $\Pi_1, \Pi_2, \dots, \Pi_R$ одинаковы, что в общем случае необязательно, то их число $R \geq \log_b M$, где M

– число состояний синтезируемого автомата A , а b – число состояний элементарного автомата памяти. Обычно для элементарного автомата $b = 2$, тогда $R \geq \log_2 M$.

Например, переход автомата A , имеющего 5 элементов памяти, алфавит состояний которых – двоичный, из одного состояния $(a_m) = 01011$ в другое $(a_s) = 11000$, заключается в изменении состояний соответствующих элементов памяти: первый элемент памяти переходит из 0 в 1, второй – из 1 в 1, третий из 0 в 0, четвёртый – из 1 в 0, пятый – из 1 в 0.

Переходы элементов памяти, соответствующие переходам в автомате A , происходят под действием сигналов возбуждения памяти, поступающих с выхода комбинационной логической схемы (КЛС) на вход памяти автомата. Так, на рис. 2.3 $X = (x_1, x_2, \dots, x_n)$ и $Y = (y_1, y_2, \dots, y_m)$ – векторные структурные входной и выходной сигналы автомата, $U = (U_1, U_2, \dots, U_T)$ – векторная функция возбуждения элементов памяти и $Q = (Q_1, \dots, Q_T)$ – вектор состояний элементов памяти автомата.

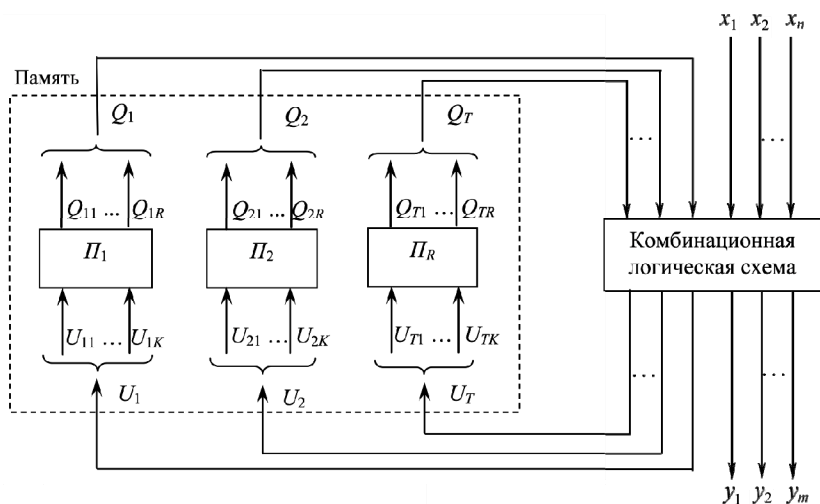


Рис. 2.3. Структурная схема цифрового автомата

Рассмотрим отдельно элемент памяти Π_i , таблица переходов которого для примера дана в табл. 2.1. Множество выходных

Таблица 2.1

**Пример таблицы переходов
элемента памяти**

	b_1	b_2	b_3
q_1	b_1	b_2	b_3
q_2	b_2	b_3	b_1
q_3	b_3	b_1	b_2

сигналов элементов памяти совпадает с множеством внутренних состояний. Полнота переходов очевидна из таблицы (в каждом столбце все состояния встречаются). При рассмотрении автомата на абстрактном уровне его можно представить в виде рис. 2.4, а.

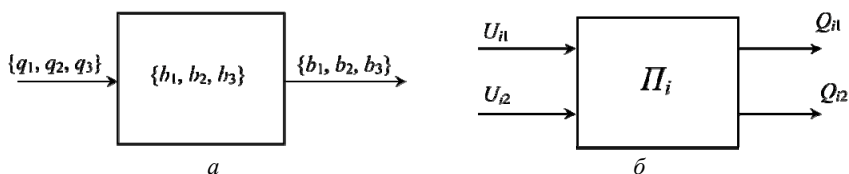


Рис. 2.4. Абстрактный (а) автомат и его структурное (б) представление

При переходе от абстрактного представления автомата к его структурному представлению, входные и выходные сигналы должны быть закодированы наборами сигналов структурного алфавита (входного или выходного соответственно). При двоичном структурном алфавите автомат Π_i будет иметь два входных ($2 \geq \log_2 3$) и два выходных ($2 \geq \log_2 3$) сигнала.

Итак, сами компоненты U_i и Q_i при $i=1, \dots, R$ векторов сигналов возбуждения памяти U и сигналов обратной связи от памяти Q также могут быть представлены в виде векторов:

$$U_i = (U_{i1}, U_{i2}, \dots, U_{iK}) \text{ и } Q_i = (Q_{i1}, Q_{i2}, \dots, Q_{iR}).$$

Если не оговорено особо, то используется двоичный структурный алфавит как для структурных входных и выходных сигналов синтезируемого автомата, так и для входных и выходных сигналов элементов памяти. Количество внутренних состояний элементов памяти также обычно принимается равным двум.

При построении функций возбуждения памяти синтезируемого автомата используют *функцию возбуждения* элемента памяти $\mu(b_i, b_j)$, ставящую в соответствие каждой паре состояний (b_i, b_j) сигнал, который должен быть подан на вход этого элемента памяти для перевода его из состояния b_i в состояние b_j . Для элемента

памяти (функция переходов которого приведена в табл. 2.1) функция возбуждения имеет вид, показанный на рис. 2.4.

μ	b_1	b_2	b_3
b_1	q_1	q_2	q_3
b_2	q_3	q_1	q_2
b_3	q_2	q_3	q_1

μ	b_1	b_2	b_3
b_1	00	01	10
b_2	10	00	01
b_3	01	10	00

а
б

Рис. 2.4. Функция возбуждения элемента памяти абстрактная (а) и кодированная (б)

Если входные сигналы элемента памяти $q_1, \dots, q_p$ закодированы наборами ($U_{i1}, \dots, U_{ik}$) сигналов на его входах, то элементами таблицы, задающей функцию возбуждения вместо q_i будут соответствующие наборы. Так, если $q_1 = 00$, $q_2 = 01$, $q_3 = 10$, то соответствующая функция возбуждения будет иметь кодированный вид (рис. 2.4, б).

2.3. Элементарные цифровые автоматы. Элементы памяти

В качестве элементов памяти структурного автомата обычно используются триггеры.

Триггер – это устройство, имеющее два устойчивых состояния, в которые он переходит под действием определённых входных сигналов. Обычно в триггерах выделяют два вида входных сигналов (и, соответственно, входов): информационные и синхросигналы.

Информационные сигналы определяют переход из одного состояния триггера в другое и присутствуют в любых триггерах. По типу информационных сигналов осуществляется классификация триггеров. Синхросигнал не является обязательным и вводится в триггерах для фиксации момента перехода триггера в новое со-

стояние, задаваемое информационными входами. Обычно при синтезе цифрового автомата используются триггеры с синхровходом, поэтому в дальнейшем нас будут интересовать только такие триггеры.

Рассмотрим основные типы триггеров, используемые для синтеза цифровых автоматов: D , T , RS , JK .

D -триггер – элемент задержки. Осуществляет задержку поступившего на его вход сигнала на один такт. Имеет один информационный вход D (*Delay*) и парафазный выход: Q – прямой выход, $\bar{Q}$ – инверсный выход. Условное графическое обозначение D -триггера представлено на рис. 2.5, таблица переходов – в табл. 2.2.

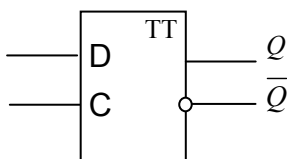


Рис. 2.5. Условное графическое изображение D -триггера

Таблица 2.2

Таблица переходов D -триггера

D	Q^t	Q^{t+1}
0	0	0
0	1	0
1	0	1
1	1	1

Таблица 2.3

Функция возбуждения
 D -триггера

Q^t	Q^{t+1}	D^t
0	0	0
0	1	1
1	0	0
1	1	1

Из приведенной таблицы переходов для данного триггера $Q^{t+1} = f(Q^t, D^t)$ можно получить таблицу его функции возбуждения $D^t = \phi(Q^t, Q^{t+1})$ (табл. 2.3). Как видно из таблицы, состояние, в которое переходит триггер (средний столбец), совпадает с поступившим на его вход сигналом $D(t)$ (правый столбец). В связи с этим таблица функций возбуждения памяти синтезируемого автомата с использованием D -триггеров будет полностью совпадать с кодированной таблицей переходов этого автомата.

T -триггер – триггер со счётным входом. Осуществляет суммирование по модулю два значений сигнала T и состояния Q в заданный момент времени. Имеет один информационный вход T (*Trigger*

– защёлка) и парафазный выход: Q – прямой выход, $\bar{Q}$ – инверсный выход. Условное графическое обозначение T -триггера представлено на рис. 2.6, таблица переходов – в табл. 2.4.

Таблица 2.4

Таблица переходов
 T -триггера

T	Q^t	Q^{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

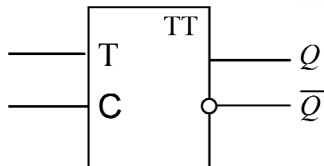


Рис. 2.6. Условное графическое изображение T -триггера

Таблица функции возбуждения T -триггера $T_t = f(Q^t, Q^{t+1})$ представлена в табл. 2.5.

На основании этой таблицы можно получать функцию возбуждения элементов памяти при синтезе автомата на базе T -триггера. Например, если автомат перешёл из состояния $a_i = 010$ в состояние $a_j = 110$, то для обеспечения этого перехода функции возбуждения должны быть:

для первого триггера при переходе из 0 в 1 $T_1 = 1$,

для второго триггера при переходе из 1 в 1 $T_2 = 0$,

для третьего триггера при переходе из 0 в 0 $T_3 = 0$ и т.д.

RS-триггер – триггер с отдельными установочными входами. Имеет два входных канала R и S и парафазный выход: Q – прямой выход, $\bar{Q}$ – инверсный выход. Вход S (*set*) называется входом установки в единичное состояние, вход R (*reset*) – входом установки в нулевое состояние. Условное графическое обозначение RS -триггера представлено на рис. 2.7, таблица переходов – в табл. 2.6 (а).

Таблица 2.5

Функция возбуждения
 T -триггера

Q^t	Q^{t+1}	T^t
0	0	0
0	1	1
1	0	1
1	1	0

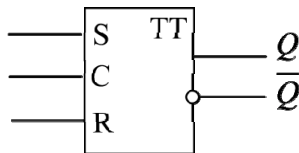


Рис. 2.7. Условное графическое изображение RS -триггера

Таблица 2.6

Таблица переходов *RS*-триггера

а)	R	S	Q^t	Q^{t+1}
	0	0	0	0
	0	0	1	1
	0	1	0	1
	0	1	1	1
	1	0	0	0
	1	0	1	0
	1	1	0	–
	1	1	1	–

б)	R	S	Q^{t+1}
	0	0	Q^t
	0	1	1
	1	0	0
	1	1	–

В таблице переходов при подаче комбинации $S = R = 1$ состояние перехода Q^{t+1} не определено, и эта комбинация сигналов является запрещённой для *RS*-триггера.

Эту таблицу переходов *RS*-триггера можно более компактно изобразить в другом виде (табл. 2.6 (б)). Анализируя таблицу переходов, отмечаем, что, например, переход триггера из 0 в 0 требует подачи комбинации $R = 0, S = 0$ или $R = 1, S = 0$, то есть этот переход произойдёт при $R = X$ (безразличное состояние), $S = 0$.

Аналогично рассуждая по отношению к другим переходам, получим таблицу функции возбуждения *RS*-триггера (табл. 2.7).

Таблица 2.7

Функция возбуждения *RS*

Q^t	Q^{t+1}	R^t	S^t
0	0	X	0
0	1	0	1
1	0	1	0
1	1	0	X

На основании табл. 2.7 можно получить функции возбуждения памяти автомата при синтезе на базе *RS*-триггеров. Например, если автомат переходит из состояния $a_i = 010$ в состояние $a_j = 110$, то для обеспечения такого перехода функции возбуждения должны быть:

для первого триггера при переходе из 0 в 1 $R_1 = 0, S_1 = 1$;

для второго триггера при переходе из 1 в 1 $R_2 = 0, S_2 = X$;

для третьего триггера при переходе из 0 в 0 $R_3 = X, S_3 = 0$.

Аналогично для любого другого перехода автомата.

JK-триггер – имеет два информационных входа J и K и парафазный выход: Q – прямой выход, $\bar{Q}$ – инверсный выход.

Вход J (*Jump*)– вход установки в единичное состояние, вход K (*Kill*)– вход установки в нулевое состояние, то есть эти входы аналогичны соответствующим входам RS -триггера: вход J соответствует S , K – соответствует R . Однако в отличие от RS -триггера, входная комбинация $J = 1$, $K = 1$ не является запрещённой. Условное графическое обозначение JK -триггера представлено на рис. 2.8, таблица переходов – в табл. 2.8.

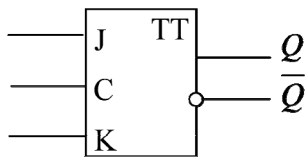


Рис. 2.8. Условное графическое изображение JK -триггера

Таблица 2.8

Таблица переходов JK -триггера

а)

J	K	Q^t	Q^{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

б)

J	K	Q^{t+1}
0	0	Q^t
0	1	0
1	0	1
1	1	$\overline{Q}^t$

Как следует из таблиц переходов, для комбинаций входных сигналов $JK = 00 \div 10$ триггер ведёт себя как RS -триггер, а при комбинации $JK = 11$ – как T -триггер.

Анализируя таблицу переходов, отмечаем, что переход триггера, например, из 0 в 1 требует подачи входных сигналов $J = 1$, $K = 0$ или $J = 1$, $K = 1$, то есть $J = 1$, $K = X$ (безразличное значение). Аналогично рассуждая по отношению к другим переходам, получим таблицу функции возбуждения JK -триггера (табл. 2.9).

Таблица 2.9

Функция возбуждения JK -триггера

Q^t	Q^{t+1}	J	K
0	0	X	0
0	1	1	X
1	0	X	1
1	1	0	X

На основании табл. 2.9 можно получить функцию возбуждения элементов памяти при синтезе автомата на JK -триггерах. Например, при переходе автомата из состояния $a_i = 010$ в состояние $a_j = 110$, функции возбуждения должны быть:

для первого триггера при переходе из 0 в 1 $J_1 = 1, K_1 = X$;

для второго триггера при переходе из 1 в 1 $J_2 = X, K_2 = 0$;

для третьего триггера при переходе из 0 в 0 $J_3 = 0, K_3 = X$.

2.4. Пример канонического метода структурного синтеза автомата

Выполним структурный синтез частичного автомата A , заданного своими таблицами переходов и выходов (табл. 2.10).

Таблица 2.10

Абстрактные таблицы переходов и выходов

Переходы					Выходы				
	a_1	a_2	a_3	a		a_1	a_2	a_3	a_4
z_1	a_1	a_4	a_4	-	z_1	w_2	w_1	w_3	-
z_2	-	a_3	a_1	-	z_2	-	w_3	w_1	-
z_3	a_2	-	a_2	a	z_3	w_1	-	w_4	w_2

Синтез будем выполнять в следующем порядке.

1. Выберем в качестве элементов памяти D -триггер, функция возбуждения которого представлена в табл. 2.2.

2. Закодируем входные, выходные сигналы и внутренние состояния автомата. Количество входных абстрактных сигналов $F = 3$, следовательно количество входных структурных сигналов $n = \lceil \log_2 F \rceil = \lceil \log_2 3 \rceil = 2$, то есть $X = \{x_1, x_2\}$.

Количество выходных абстрактных сигналов $G = 4$, следовательно, количество выходных структурных сигналов $m = \lceil \log_2 G \rceil = \lceil \log_2 4 \rceil = 2$, то есть $Y = \{y_1, y_2\}$. Количество внутренних состояний абстрактного автомата $M = 4$, следовательно, количество двоичных элементов памяти (триггеров) $R = \lceil \log_2 M \rceil = \lceil \log_2 4 \rceil = 2$.

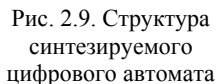


Таблица 2.11

Входные сигналы		
	x_1	x_2
z_1	0	0
z_2	0	1
z_3	1	1

Выходные сигналы		
	y_1	y_2
w_1	0	0
w_2	0	1
w_3	1	1
w_4	1	0

Состояния автомата		
	Q_1	Q_2
a_1	0	0
a_2	0	1
a_3	1	1
a_4	1	0

Получим кодированные таблицы переходов и выходов (табл. 2.12) структурного представления автомата. Для этого в таблицах переходов и выходов исходного абстрактного автомата вместо z_i , w_i , a_i поставим соответствующие коды.

Таблица 2.12

Кодированные таблицы переходов и выходов

		Переходы						Выходы			
		a_1	a_2	a_3	a_4			a_1	a_2	a_3	a_4
z_1		00	01	11	10	z_1		00	01	11	10
	00	00	10	10	—		00	01	00	11	—
	01	—	11	00	—		01	—	11	00	—
z_2	11	01	—	01	11	z_3	11	00	—	10	01

В кодированной таблице переходов заданы функции

$$Q_1^{t+1} = \psi_1(x_1, x_2, Q_1^t, Q_2^t), \quad Q_2^{t+1} = \psi_2(x_1, x_2, Q_1^t, Q_2^t).$$

В кодированной таблице выходов заданы функции:

$$y_1 = \varphi_1(x_1, x_2, Q_1^t, Q_2^t), \quad y_2 = \varphi_2(x_1, x_2, Q_1^t, Q_2^t).$$

При каноническом методе структурный синтез сводится к получению функций:

$$\begin{aligned} y_1 &= \varphi_1(x_1, x_2, Q_1, Q_2), & y_2 &= \varphi_2(x_1, x_2, Q_1, Q_2), \\ D_1 &= f_1(x_1, x_2, Q_1, Q_2), & D_2 &= f_2(x_1, x_2, Q_1, Q_2) \end{aligned}$$

и последующему построению комбинационных логических схем, реализующих данную систему булевых функций.

Функции y_1 и y_2 могут быть непосредственно получены из кодированной таблицы выходов, например, в виде совершенной дизъюнктивной нормальной формы:

$$\begin{aligned} y_1 &= \bar{x}_1 x_2 \bar{Q}_1 Q_2 \vee \bar{x}_1 \bar{x}_2 Q_1 Q_2 \vee x_1 x_2 Q_1 Q_2; \\ y_2 &= \bar{x}_1 \bar{x}_2 \bar{Q}_1 \bar{Q}_2 \vee \bar{x}_1 x_2 \bar{Q}_1 Q_2 \vee \bar{x}_1 \bar{x}_2 Q_1 Q_2 \vee x_1 x_2 Q_1 \bar{Q}_2. \end{aligned}$$

Однако выражения для y_1 и y_2 можно существенно упростить в результате минимизации, например, с помощью карт Карно (рис. 2.10).

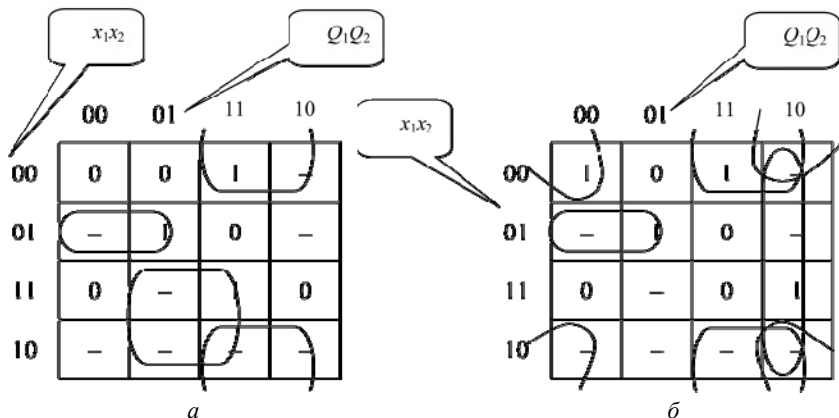


Рис. 2.10. Карты Карно для y_1 (а) и y_2 (б)

В результате минимизации имеем:

$$\left. \begin{aligned} y_1 &= \bar{x}_2 Q_1 \vee x_1 Q_2 \vee \bar{x}_1 x_2 \bar{Q}_1; \\ y_2 &= \bar{x}_2 Q_1 \vee Q_1 \bar{Q}_2 \vee \bar{x}_2 \bar{Q}_2 \vee \bar{x}_1 x_2 \bar{Q}_1. \end{aligned} \right\} \quad (2.1)$$

Для получения выражений D_1 и D_2 необходимо составить таблицы функций возбуждения автомата. Для чего в общем случае необходимо воспользоваться таблицей переходов и функциями возбуждения элементов памяти. Зная код исходного состояния автомата и код состояния перехода на основании таблицы возбуждения соответствующего триггера, получаем требуемое значение функции возбуждения, обеспечивающее заданный переход. Однако для D -триггеров, как отмечалось ранее, таблица переходов совпадает с таблицей функции возбуждения. Тогда либо непосредственно из кодированной таблицы переходов (см. табл. 2.12), либо в результате минимизации (рис. 2.11) получаем требуемые значения D_i :

$$\left. \begin{aligned} D_1 &= \bar{Q}_1 Q_2 \vee Q_1 \bar{Q}_2 \vee \bar{x}_2 Q_2; \\ D_2 &= x_1 \vee x_2 \bar{Q}_1. \end{aligned} \right\} \quad (2.2)$$

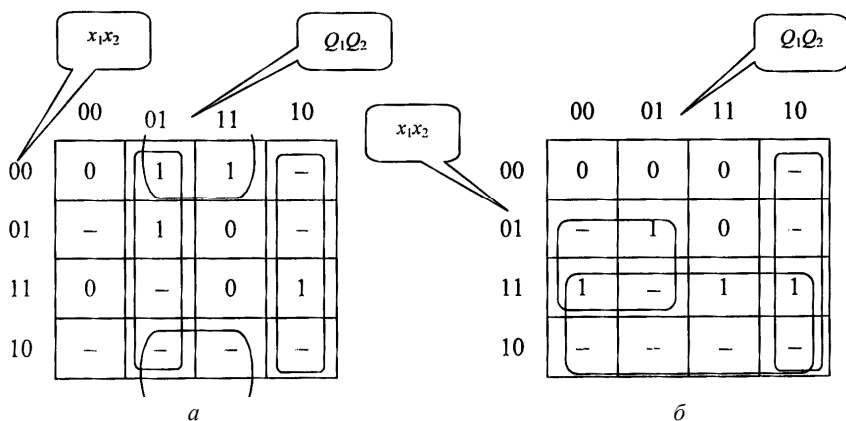


Рис. 2.11. Карта Карно для D_1 (а) и D_2 (б)

На основании полученных в результате синтеза булевых выражений (2.1), (2.2) строится функционально-логическая схема автомата (рис. 2.12).

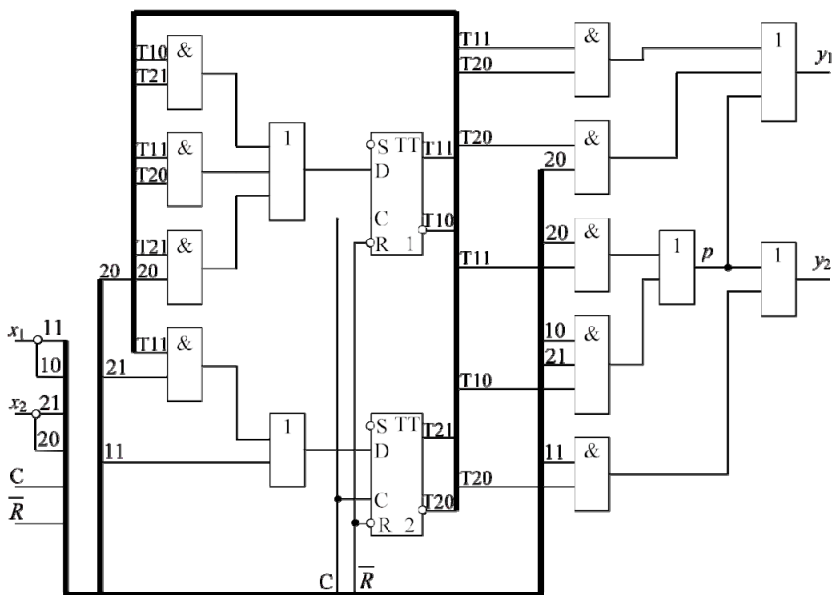


Рис. 2.12. Функционально-логическая схема автомата

Уравнения (2.1) и (2.2) представлены в виде:

$$\left. \begin{aligned} P &= \bar{x}_2 Q_1 \vee \bar{x}_1 x_2 \bar{Q}; \\ y_1 &= P \vee x_1 Q_2; \\ y_2 &= P \vee \bar{x}_2 \bar{Q}_2 \vee Q_1 \bar{Q}_2; \\ D_1 &= \bar{Q}_1 Q_2 \vee Q_1 \bar{Q}_2 \vee \bar{x}_2 Q_2; \\ D_2 &= x_1 \vee x_2 \bar{Q}_1. \end{aligned} \right\} \quad (2.3)$$

Дополнительно на функционально-логической схеме показан сигнал $\bar{R}$, устанавливающий автомат в начальное состояние (в данном случае 00).

2.5. Особенности синтеза автоматов на базе T -, RS - и JK -триггеров

Необходимо отметить, что синтез на базе T -, RS -, JK -триггеров осуществляется аналогично выполненному синтезу на базе D -триггеров. Кроме того, выходные сигналы не зависят от типа триггеров, поэтому выражение для y_i будут одинаковыми для любого типа триггеров. Однако функции возбуждения будут различны для разных типов триггеров и получаются на основании таблицы переходов исходного автомата и функции возбуждения выбранного триггера. Таблицы функций возбуждения триггеров, функций возбуждения синтезируемого автомата и карты Карно для минимизации функций возбуждения при использовании для синтеза автомата T -, RS -, JK -триггеров, рассмотренных в п. 2.4, приводим без особых пояснений.

T -триггер

Таблица 2.13

Функции возбуждения
 T -триггера

Q^t	Q^{t+1}	T^t
0	0	0
0	1	1
1	0	1
1	1	0

Таблица 2.14

Функции возбуждения автомата
 T_1 и T_2

		$Q_1 Q_2$			
		00	01	11	10
$x_1 x_2$	00	00	11	01	—
	01	—	10	11	—
	11	01	—	10	01

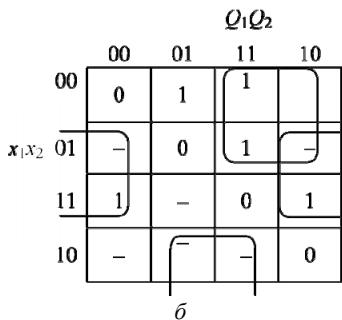
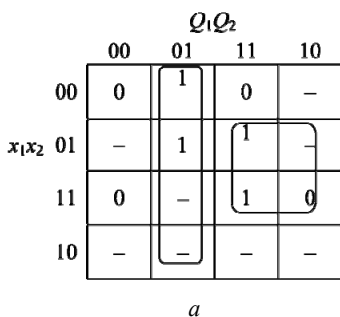


Рис. 2.13. Карты Карно:

a — для T_1 ($T_1 = \bar{Q}_1Q_2 \vee x_2Q_2$); b — для T_2 ($T_2 = x_2\bar{Q}_2 \vee \bar{x}_2Q_2 \vee \bar{x}_1Q_1$)

RS-триггер

Таблица 2.15

Функции возбуждения RS-триггера

Q^t	Q^{t+1}	R	S
0	0	X	0
0	1	0	1
1	0	1	0
1	1	0	X

Таблица 2.16

Функции возбуждения автомата R_1, S_1, R_2, S_2

		Q_1Q_2															
		00				01				11				10			
x_1x_2		R_1	S_1	R_2	S_2	R_1	S_1	R_2	S_2	R_1	S_1	R_2	S_2	R_1	S_1	R_2	S_2
00		X	0	X	0	0	1	1	0	0	X	1	0	—		—	
01			—		—	0	1	0	X	1	0	1	0	—		—	
11		X	0	0	1	—		—		1	0	0	X	0	X	0	1

		Q_1Q_2			
R_1		00	01	11	10
x_1x_2	00	X	0	0	—
	01	—	0	1	—
	11	X	—	1	0
	10	—	—	—	—

		Q_1Q_2			
S_1		00	01	11	10
x_1x_2	00	0	1	X	—
	01	—	1	0	—
	11	0	—	0	X
	10	—	—	—	—

		Q_1Q_2			
R_2		00	01	11	10
x_1x_2	00	X	1	1	—
	01	—	0	1	—
	11	0	—	0	0
	10	—	—	—	—

		Q_1Q_2			
S_2		00	01	11	10
x_1x_2	00	0	0	0	—
	01	—	X	0	—
	11	1	—	X	1
	10	—	—	—	—

Рис. 2.14. Карты Карно для $R_1 = x_2Q_1Q_2$; $S_1 = \bar{Q}_1Q_2$; $R_2 = \bar{x}_2 \vee \bar{x}_1Q_1$; $S_2 = x_1$

JK-триггер

Таблица 2.17

Функции возбуждения JK-триггера

Q^t	Q^{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Таблица 2.18

Функции возбуждения автомата J_1, K_1, J_2, K_2

		00				01				11				10			
		J_1	K_1	J_2	K_2	J_1	K_1	J_2	K_2	J_1	K_1	J_2	K_2	J_1	K_1	J_2	K_2
x_1x_2	00	0	X	0	X	1	X	X	1	X	0	X	1	—	—	—	—
	01	—	—	—	—	1	X	X	0	X	1	X	1	—	—	—	—
	11	0	X	1	X	—	—	—	—	X	1	X	0	X	0	1	X

		Q_1Q_2			
J_1		00	01	11	10
x_1x_2	00	0	1	X	—
	01	—	1	X	—
	11	0	—	X	X
	10	—	—	—	—

		Q_1Q_2			
J_2		00	01	11	10
x_1x_2	00	0	X	X	—
	01	—	X	X	—
	11	1	—	X	1
	10	—	—	—	—

		Q_1Q_2			
K_1		00	01	11	10
x_1x_2	00	X	X	0	—
	01	—	X	1	—
	11	X	—	1	0
	10	—	—	—	—

		Q_1Q_2			
K_2		00	01	11	10
x_1x_2	00	X	1	1	—
	01	—	0	1	—
	11	0	—	0	X
	10	—	—	—	—

Рис. 2.15. Карты Карно для $J_1 = Q_2; K_1 = X_2Q_2; J_2 = x_1; S_2 = \bar{x}_2 \vee \bar{x}_1Q_1$

Функционально-логические схемы автоматов с различными типами триггеров предлагается построить самостоятельно.

2.6. Кодирование внутренних состояний.

Гонки в автомате

Кодирование заключается в представлении каждого состояния автомата наборами состояний элементов памяти. При этом наборы для всех состояний должны иметь одинаковую длину, а разным состояниям автомата должны соответствовать разные наборы. Если элементы памяти двоичные, то их число $R \geq \lceil \log_2 M \rceil$.

Переход автомата из одного состояния в другое осуществляется за счёт изменения состояний элементов памяти. Если автомат переходит из состояния с кодом 010 в состояние с кодом 100, то это означает, что триггер V_1 переходит из состояния 0 в состояние 1, V_2 — из 1 в 0, V_3 — сохраняет своё состояние.

При функционировании автомата могут появиться так называемые состязания. Это явление возникает вследствие того, что

элементы памяти имеют различные, хотя и достаточно близкие, времена переключения. Различны также задержки сигналов возбуждения, поступающих на входы элементарных автоматов по логическим цепям неодинаковой длины.

Если при переходе автомата из одного состояния в другое должны изменить свои состояния сразу несколько элементов памяти, то между ними начинаются состязания. Тот элемент памяти, который выиграет эти состязания, то есть изменит своё состояние ранее, чем другие элементы памяти, может через цепь обратной связи изменить сигналы на входах некоторых элементов памяти до того, как другие, участвующие в состязаниях элементы памяти, изменят свои состояния. Это может привести к переходу автомата в состояние, не предусмотренное графом его функционирования. Поэтому в процессе перехода из состояния a_m в состояние a_s под действием входного сигнала z_f автомат может оказаться в состоянии a_k или a_l (рис. 2.16).

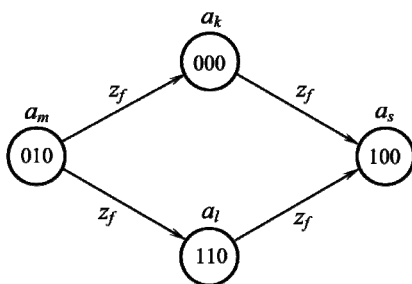


Рис. 2.16. Состязания в автомате

Если затем при том же входном сигнале z_f автомат из a_k и a_l перейдёт в a_s , то такие состязания являются допустимыми или *некритическими*.

Если же в этом автомате есть переход из a_k в $a_j \neq a_s$ под действием того же сигнала z_f , то автомат может перейти в a_j , а не в a_s и правильность его работы будет нарушена (рис. 2.17).

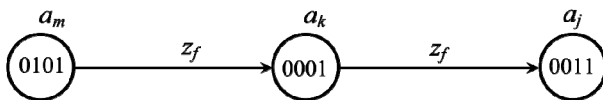


Рис. 2.17. Гонки в автомате

Такие состязания называются критическими состязаниями или *гонками* и необходимо принимать меры для их устранения.

Устранить гонки можно аппаратными средствами или с использованием специальных методов кодирования (методы противогоночного кодирования). Один из способов ликвидации гонок состо-

ит в тактировании входных сигналов автомата импульсами определённой длительности. Предполагается, что, кроме входных сигналов $x_1, \dots, x_n$, есть ещё сигнал C от генератора синхроимпульсов, по которому поступает сигнал $C = 1$ в момент прихода импульса и $C = 0$ при его отсутствии. В связи с этим входным сигналом на переходе (a_m, a_s) будет не z_f , а $C \& z_f$. Тогда, если длительность импульса t_c меньше самого короткого пути прохождения тактированного сигнала обратной связи по комбинационной схеме, то к моменту перехода в промежуточное состояние a_k сигнал $C = 0$, $C \& z_f = 0$, что исключает гонки. Сигнал C – это фактически синхровход триггера. Недостаток метода – в трудности подбора требуемой длительности импульса, так как она зависит от многих факторов, не поддающихся строгому учёту.

Другой способ ликвидации гонок заключается во введении двойной памяти. В этом случае каждый элемент памяти дублируется, причём перепись из первого элемента памяти во второй происходит в момент $C = 0$ (рис. 2.18).

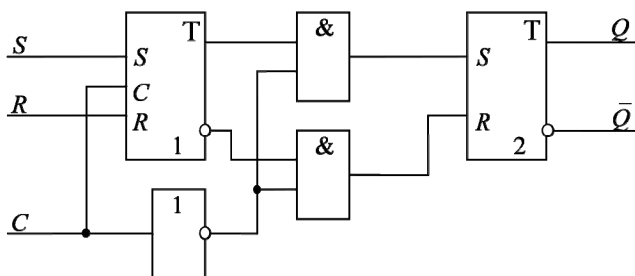


Рис. 2.18. Элемент с двойной памятью

Сигналы обратной связи для получения функций возбуждения и функций выходов автомата снимаются с выхода второго триггера. Таким образом, состязания могут возникнуть только между первыми триггерами, сигналы обратной связи (выходы вторых триггеров) не могут измениться до тех пор, пока C не станет равным 0. Но когда $C \& z_f = 0$, то первый триггер перестанет воспринимать входную информацию, и гонок не будет.

Для устранения гонок используются специальные методы противогоночного кодирования, среди которых чаще всего применяет-

ся так называемое соседнее кодирование состояний автомата, при котором условие отсутствия гонок всегда выполнено. При соседнем кодировании любые два, состояния связанные дугой на графе автомата, кодируются наборами, отличающимися состояниями лишь одного элемента памяти.

Соседнее кодирование не всегда возможно. Граф автомата, конфигурация которого допускает соседнее кодирование, должен удовлетворять ряду требований, а именно:

- в графе автомата не должно быть циклов с нечётным числом вершин;
- два соседних состояния второго порядка не должны иметь более двух состояний, лежащих между ними.

Под состояниями второго порядка понимаются такие два состояния, путь между которыми по графу автомата состоит из двух ребер (независимо от ориентации). Примеры графов автоматов допускающих и не допускающих соседнее кодирование представлены на рис. 2.19.

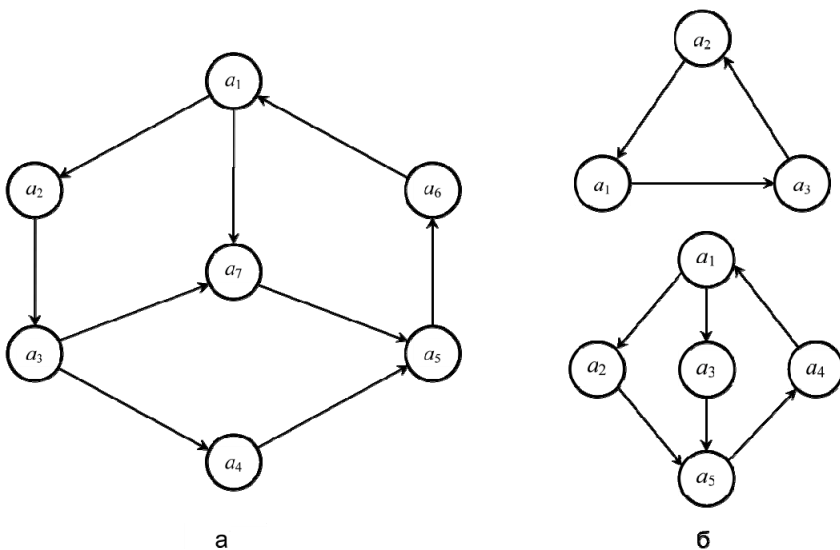


Рис. 2.19. Графы автоматов, допускающие (а) и не допускающие (б) соседнее кодирование

При соседнем кодировании обычно пользуются картой Карно. В этом случае состояния, связанные дугой, располагают на соседних клетках карты (рис. 2.20).

$Q_1 Q_2$									
Q_3		00	01	11	10	$a_1 \sim 000$	$a_2 \sim 010$	$a_3 \sim 110$	$a_4 \sim 111$
0		a_1	a_2	a_3	a_4				
1		a_6		a_4	a_5	$a_5 \sim 101$	$a_6 \sim 001$	$a_7 \sim 100$	

Рис. 2.19. Карта Карно для соседнего кодирования

Легко видеть, что при соседнем кодировании на каждом переходе переключается только один триггер, что принципиально упрощает гонки.

2.6. Кодирование состояний и сложность комбинационной логической схемы автомата

Анализ канонического метода структурного синтеза автомата показывает, что различные варианты кодирования состояний автомата приводят к различным выражениям функций возбуждения памяти и функций выходов, в результате чего сложность комбинационной логической схемы существенно зависит от выбранного кодирования. Среди множества существующих алгоритмов кодирования, минимизирующих выражения функций возбуждения памяти и функций выходов, рассмотрим лишь два наиболее часто встречаемых:

- алгоритм кодирования для D -триггеров;
- эвристический алгоритм кодирования.

Алгоритм кодирования для D -триггеров.

1. Каждому состоянию автомата a_m ($m = 1, 2, \dots, M$) ставится в соответствие целое число N_m , равное числу переходов в состояние a_m (N_m равно числу появлений состояния a_m в поле таблицы переходов или числу дуг, входящих в a_m при графическом способе задания автомата).

2. Числа $N_1, N_2, \dots, N_m$ упорядочиваются по убыванию.

3. Состояние a_s с наибольшим N_s кодируется кодом $(00\dots 0)$ длины R , где R – число элементов памяти.

4. Следующие R состояний согласно списка п. 2 кодируются кодами, содержащими только одну единицу: $(00\dots 01)$, $(00\dots 10)$, ..., $(01\dots 00)$, $(10\dots 00)$.

5. Для оставшихся $M - R - 1$ состояний опять в упорядоченном списке п. 2. используются коды с двумя единицами, затем с тремя и так далее пока не будут закодированы все состояния.

В результате получается такое кодирование, при котором, чем больше переходов в некоторое состояние, тем меньше единиц в его коде. Так как для D -триггеров функции возбуждения однозначно определяются кодом состояния перехода, то очевидно, что выражения для функций возбуждения будут проще. Этот метод особенно эффективен при отсутствии минимизации функций возбуждения, что имеет место в реальных автоматах с большим количеством внутренних состояний и входных переменных.

В частности, для автомата, заданного своими таблицами переходов и выходов (табл. 2.19) при кодировании на базе D -триггеров:

$a_1 \sim N_1 = 3$	N_3	$a_3 = 000$
$a_2 \sim N_2 = 4$	N_2	$a_2 = 001$
$a_3 \sim N_3 = 5$	N_1	$a_1 = 010$
$a_4 \sim N_4 = 5$	N_4	$a_4 = 100$
$a_5 \sim N_5 = 1$	N_5	$a_5 = 011$

Таблица 2.19

Таблицы переходов и выходов

Переходы						Выходы					
	a_1	a_2	a_3	a_4	a_5		a_1	a_2	a_3	a_4	a_5
z_1	a_1	a_1	a_5	a_3	a_1	z_1	w_1	w_2	w_1	w_1	w_1
z_2	a_2	a_3	a_2	a_3	a_3	z_2	w_1	w_3	w_4	w_2	w_2
z_3	a_3	a_4	a_2	a_4	a_2	z_3	w_2	w_2	w_2	w_1	w_3

Аналогично кодированию внутренних состояний для D -триггеров можно кодировать выходные сигналы для любого типа триггеров, то есть чем чаще вырабатывается данный выходной

сигнал w_i , тем меньше единиц в его коде. Так, для автомата (см. табл. 2.19) имеем:

$w_1 \sim N_1 = 6$	N_1	$w_1 = 00$
$w_2 \sim N_2 = 5$	N_2	$w_2 = 01$
$w_3 \sim N_3 = 2$	N_3	$w_3 = 10$
$w_4 \sim N_4 = 2$	N_4	$w_4 = 11$

Предлагается самостоятельно закончить синтез автомата при данном кодировании и при любом другом. Результаты сравнить.

Эвристический алгоритм кодирования. Данный алгоритм минимизирует суммарное число переключений элементов памяти на всех переходах автомата и используется для кодирования состояний автомата при синтезе на базе T -, RS -, JK -триггеров.

Для данных типов триггеров (в отличие от D -триггеров) на каждом переходе, где триггер меняет своё значение на противоположное, одна из функций возбуждения обязательно равна единице. Уменьшение числа переключений триггеров приводит к уменьшению количества единиц соответствующих функций возбуждения, что при отсутствии минимизации однозначно приводит к упрощению комбинационной логической схемы автомата.

Введём некоторые определения.

Пусть $G(S)$ – неориентированный граф переходов автомата S . Вершины графа отождествляются с состояниями автомата. Вершины i и j соединены ребром, если есть переход из a_i в a_j или наоборот. Обозначим $q(i, j)$ число всевозможных переходов автомата из a_i в a_j . Каждому ребру (i, j) графа $G(S)$ поставим в соответствие вес ребра $p(i, j) = q(i, j) + q(j, i)$.

Введём весовую функцию $w(i, j) = p(i, j) \cdot d(i, j)$, где $d(i, j)$ – число компонентов, которыми коды состояний a_i и a_j отличаются друг от друга (т.е. кодовое расстояние между кодами a_i и a_j).

Функция $w(i, j)$ имеет простой физический смысл. Переход автомата из a_i в a_j (или наоборот) сопровождается переключением столько же триггеров, сколько компонентами отличаются коды этих состояний, то есть их число равно $d(i, j)$. Следовательно, при переходе автомата по всем ребрам, соединяющим состояния a_i и a_j (их число $p(i, j)$), всего переключится количество триггеров, равное $p(i, j) \cdot d(i, j) = w(i, j)$.

Но тогда функция $w = \sum_{(i,j) \in \partial(S)} w(i,j) = \sum_{(i,j) \in \partial(S)} p(i,j) d(i,j) \frac{1}{2}$ пока-

зывает, сколько всего переключается триггеров при прохождении автомата по всем возможным переходам. Функция w показывает, сколько всего единиц в функции возбуждения, то есть позволяет оценивать сложность комбинационной схемы автомата. Функцию w можно рассматривать как некую целевую функцию, минимум которой определит такое кодирование, при котором сложность комбинационной логической схемы минимальна. Кстати, минимальное кодовое расстояние между различными состояниями равно 1, и если удаётся закодировать все состояния автомата соседним кодированием, то очевидно, что значение w будет минимально возможным и равным $w = \sum_{(i,j) \in \partial(S)} p(i,j)$, то есть суммарному числу

переходов для автомата.

Из выражения для w следует, что переход из a_i в a_i , для которого $d(i,i) = 0$, не влияет на значение функции w (что вполне очевидно, если учесть, что на этом переходе ни один триггер не переключается).

Рассмотрим применение эвристического алгоритма на конкретном примере автомата, заданного таблицами переходов и выходов (см. табл. 2.19). Для данного автомата можно построить неориентированный граф (без учёта петель), представленный на рис. 2.20. На каждом ребре указан его вес.

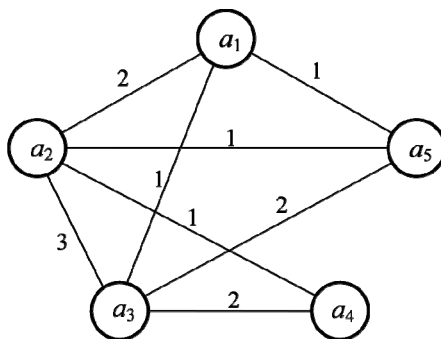


Рис. 2.20. Неориентированный граф

Эвристический алгоритм состоит из следующих шагов:

1. Строится матрица $\|T\|$, состоящая из всех пар номеров (i, j) , для которых $p(i, j) \neq 0$ (т.е. в автомате есть переход из a_i в a_j или наоборот) и $i < j$. Для каждой пары (i, j) в матрице $\|T\|$ указывается её вес $p(i, j)$, совпадающий с весом ребра, соединяющего состояния a_i и a_j .

$$\| T \| = \left\| \begin{array}{cc|c} i & j & p(i,j) \\ \hline 1 & 2 & 2 \\ 1 & 3 & 1 \\ 1 & 5 & 1 \\ 2 & 3 & 3 \\ 2 & 4 & 1 \\ 2 & 5 & 1 \\ 3 & 4 & 2 \\ 3 & 5 & 2 \end{array} \right\|$$

2. Сортируются строки матрицы $\|T\|$, для чего строится матрицу $\|M\|$ следующим образом. В первую строку матрицы $\|M\|$ помещается пара (i, j) с наибольшим весом ребра $p(i, j)$. В рассматриваемом примере $(i, j) = (2, 3)$, $p(2, 3) = 3$. Из всех пар, имеющих общий компонент с парой (i, j) , выбирается пара (γ, δ) с наибольшим весом и заносится во вторую строку матрицы $\|M\|$. Ясно, что $\{\alpha, \beta\} \cap \{\gamma, \delta\} \neq \emptyset$. Затем из всех пар, имеющих общий компонент хотя бы с одной из внесённых уже в матрицу $\|M\|$ пар, выбирается пара с наибольшим весом и заносится в матрицу $\|M\|$ и т.д. В случае равенства весов пар вычисляются суммы весов компонентов пар (весом $p(\alpha)$ компоненты α называется число появлений α в матрице $\|T\|$) и в матрицу $\|M\|$ заносится пара с наибольшей суммой весов. В рассматриваемом автомате на второе место вслед за парой $(2, 3)$ претендуют пары: $(1, 2)$ с весом $p(1, 2) = 2$; $(3, 4)$ с $p(3, 4) = 2$, $(3, 5)$ с $p(3, 5) = 2$.

Для определения того, какая пара займёт второе место в матрице $\|M\|$, находят веса компонентов пар:

$$\begin{array}{lll} p(1) = 3 & p(2) = 3 & p(1) + p(2) = 6 \\ p(3) = 4 & p(4) = 2 & p(3) + p(4) = 6 \\ p(3) = 4 & p(5) = 2 & p(3) + p(5) = 6 \end{array}$$

В данном случае для всех пар совпадают и их веса и сумма весов их компонентов. Поэтому на второе место матрицы $\|M\|$ может быть поставлена любая из пар $(1, 2)$, $(3, 4)$, $(3, 5)$. Но тогда на 3-м и 4-м будут остальные две.

Выполнив упорядочивание всех пар, получается матрица $\|M\|$ в виде:

$$\|M\| = \begin{array}{c|cc} & i & j & p(i,j) \\ \hline 2 & 3 & 3 \\ 1 & 2 & 2 \\ 3 & 4 & 2 \\ 3 & 5 & 2 \\ 1 & 3 & 1 \\ 1 & 5 & 1 \\ 2 & 4 & 1 \\ 2 & 5 & 1 \end{array}$$

3. Определяется разрядность кода для кодирования состояний автомата (количество элементов памяти – триггеров). Всего состояний $M = 5$. Тогда

$$R = \lceil \log_2 M \rceil = \lceil \log_2 5 \rceil = 3.$$

Состояния из первой строки матрицы $\|M\|$ кодируются следующим образом: $K_2 = K(a_2) = 000$; $K_3 = K(a_3) = 001$.

Для удобства кодирования можно проиллюстрировать этот процесс картой Карно (рис. 2.21).

$Q_1 \backslash Q_2 Q_3$	00	01	11	10
0	a_2	a_3		
1				

Рис. 2.21. Карта кодирования состояний

4. Из матрицы $\|M\|$ вычеркивается первая строка, соответствующая закодированным состояниям a_2 и a_3 . Получается матрица $\|M^*\|$:

$$\|M^*\| = \begin{array}{c|cc} & i & j & p(i,j) \\ \hline 1 & 2 & 2 \\ 3 & 4 & 2 \\ 3 & 5 & 2 \\ 1 & 3 & 1 \\ 1 & 5 & 1 \\ 2 & 4 & 1 \\ 2 & 5 & 1 \end{array}$$

5. В силу упорядочивания п.2 в первой строке матрицы $\|M^*\|$ закодирован ровно один элемент. Выбирается из первой строки неза-

кодированный элемент и обозначается, например, как γ (в нашем случае $\gamma = 1$).

6. Строится матрица $\|M_\gamma\|$ путём выбора из матрицы $\|M^*\|$ строк, содержащих γ . Для $\gamma = 1$:

$$\|M^*\| \rightarrow \|M_\gamma\| \rightarrow \left\| \begin{array}{cc|c} i & j & p(i,j) \\ 1 & 2 & 2 \\ 1 & 3 & 1 \\ 1 & 5 & 1 \end{array} \right\|$$

Пусть $B_\gamma = \{\gamma_1, \dots, \gamma_F\}$ – множество элементов из матрицы $\|M_\gamma\|$, которые уже закодированы. Их коды $K_{\gamma_1}, \dots, K_{\gamma_F}$ соответственно. В нашем случае:

$$B_\gamma = B_3 = \{2, 3\}; \quad K_2 = 000; \quad K_3 = 001.$$

7. Для каждого K_{γ_f} ($f = 1, \dots, F$) находится $C_{\gamma_f}^1$ – множество кодов, соседних с K_{γ_f} и ещё не занятых для кодирования состояний автомата (для соседних кодов кодовое расстояние $d = 1$):

$$K_2 = 000; \quad C_2^1 = \{100, 010\};$$

$$K_3 = 001; \quad C_3^1 = \{011, 101\}.$$

Строится множество $D_\gamma^1 = \bigcup_{f=1}^F C_{\gamma_f}^1 = C_{\gamma_1}^1 \cup C_{\gamma_2}^1 \cup \dots \cup C_{\gamma_F}^1$:

$$D_3^1 = C_2^1 \cup C_3^1 = \{100, 010, 011, 101\}.$$

Если оказывается, что $D_\gamma^1 = \emptyset$, то строится новое множество

$$D_\gamma^2 = \bigcup_{f=1}^F C_{\gamma_f}^2, \text{ где } C_{\gamma_f}^2 \text{ – множество кодов, у которых кодовое рас-}$$

стояние до кода K_{γ_f} равно 2 и так далее.

8. Для каждого кода из множества D_γ находится кодовое расстояние до кода K_{γ_f} : $K_2 = 000$; $K_3 = 001$;

$$d(100, 000) = 1; \quad d(010, 000) = 1; \quad d(011, 000) = 2; \quad d(101, 000) = 2;$$

$$d(100, 001) = 2; \quad d(010, 001) = 2; \quad d(011, 001) = 1; \quad d(101, 001) = 1.$$

9. Находится значение функции w для каждого кода из множества D_γ :

$$\begin{aligned}
w_{100} &= d(100, 000) \cdot p(1, 2) + d(100, 001) \cdot p(1, 3) = 1 \cdot 2 + 2 \cdot 1 = 4, \\
w_{010} &= d(010, 000) \cdot p(1, 2) + d(010, 001) \cdot p(1, 3) = 1 \cdot 2 + 2 \cdot 1 = 4, \\
w_{011} &= d(011, 000) \cdot p(1, 2) + d(011, 001) \cdot p(1, 3) = 2 \cdot 2 + 2 \cdot 1 = 5, \\
w_{101} &= d(101, 000) \cdot p(1, 2) + d(101, 001) \cdot p(1, 3) = 2 \cdot 2 + 2 \cdot 1 = 5.
\end{aligned}$$

10. Из множества D_γ выбирается код K_γ , у которого минимальное значение функции w . Для состояния a_1 , например, $K_1 = 100$ (рис. 2.22).

11. Из матрицы $\|M^*\|$ вычеркиваются строки, в которых оба элемента уже закодированы, в результате чего получается новая матрица $\|M^*\|$. Если в новой матрице $\|M^*\|$ не осталось ни одной строки, то кодирование закончено. В противном случае происходит возврат к п. 5. В нашем случае

		$Q_2 Q_3$			
Q_1		00	01	11	10
	0	a_2	a_3		
	1	a_1			

Рис. 2.22. Карта кодирования состояний

$$\|M^*\| = \left\| \begin{array}{cc|c} i & j & p(i,j) \\ \hline 3 & 4 & 2 \\ 3 & 5 & 2 \\ 1 & 5 & 1 \\ 2 & 4 & 1 \\ 2 & 5 & 1 \end{array} \right\|$$

$$\gamma = 4; \quad \|M_\gamma\| = \|M_4\| = \left\| \begin{array}{cc|c} 3 & 4 & 2 \\ \hline 2 & 4 & 1 \end{array} \right\|; \quad B_\gamma = B_4 = \{2, 3\}$$

$$K_2 = 000; \quad C_2^1 = \{010\};$$

$$K_3 = 001; \quad C_3^1 = \{011, 101\};$$

$$D_4^1 = C_2^1 \cup C_3^1 = \{010, 011, 101\}.$$

$$\begin{aligned}
K_2 &= 000; & K_3 &= 001; \\
d(010, 000) &= 1; & d(010, 001) &= 2; \\
d(011, 000) &= 2; & d(011, 001) &= 1; \\
d(101, 000) &= 2; & d(101, 001) &= 1.
\end{aligned}$$

$$w_{010} = d(010, 000) \cdot p(2, 4) + d(010, 001) \cdot p(3, 4) = 1 \cdot 1 + 2 \cdot 2 = 5,$$

$$w_{011} = d(011, 000) \cdot p(2, 4) + d(011, 001) \cdot p(3, 4) = 2 \cdot 1 + 1 \cdot 2 = 4,$$

$$w_{101} = d(101, 000) \cdot p(2, 4) + d(101, 001) \cdot p(3, 4) = 2 \cdot 1 + 1 \cdot 2 = 4.$$

По правилу п. 10 выбираем $K_4 = 101$ (рис. 2.23).

$Q_2 Q_3$	00	01	11	10
Q_1				
0	a_2	a_3		
1	a_1	a_4		

Рис. 2.23. Карта кодирования состояний

Далее исходя из п. 11 получаем:

$$\|M^*\| = \left\| \begin{array}{cc|c} 3 & 5 & 2 \\ 1 & 5 & 1 \\ 2 & 5 & 1 \end{array} \right\|; \quad \gamma=5;$$

$$\|M_\gamma\| = \|M_5\| = \left\| \begin{array}{cc|c} 3 & 5 & 2 \\ 1 & 5 & 1 \\ 2 & 5 & 1 \end{array} \right\|; \quad B\gamma = B_5 = \{1, 2, 3\}.$$

$$K_1 = 100; \quad C_1^1 = \{110\};$$

$$K_2 = 000; \quad C_2^1 = \{010\};$$

$$K_3 = 001; \quad C_3^1 = \{011\}.$$

$$D_5^1 = C_1^1 \cup C_2^1 \cup C_3^1 = \{110, 010, 011\}.$$

$$K_1 = 100; \quad K_2 = 000; \quad K_3 = 001;$$

$$d(110, 100) = 1; \quad d(110, 000) = 2; \quad d(110, 001) = 3;$$

$$d(010, 100) = 2; \quad d(010, 000) = 1; \quad d(010, 001) = 2;$$

$$d(011, 100) = 3; \quad d(011, 000) = 2; \quad d(011, 001) = 1.$$

$$w_{110} = d(110,100) \cdot p(1,5) + d(110,000) \cdot p(2,5) + d(110,001) \cdot p(3,5) = 1 \cdot 1 + 2 \cdot 1 + 3 \cdot 2 = 9,$$

$$w_{010} = d(010,100) \cdot p(1,5) + d(010,000) \cdot p(2,5) + d(010,001) \cdot p(3,5) = 2 \cdot 1 + 1 \cdot 1 + 2 \cdot 2 = 7,$$

$$w_{011} = d(011,100) \cdot p(1,5) + d(011,000) \cdot p(2,5) + d(011,001) \cdot p(3,5) = 3 \cdot 1 + 2 \cdot 1 + 1 \cdot 2 = 7.$$

Исходя из правила пункта 10 выбираем $K_5 = 011$ (рис. 2.24).

Рис. 2.24. Карта кодирования состояний

$Q_2 Q_3$	00	01	11	10
Q_1				
0	a_2	a_3	a_5	
1	a_1	a_4		

Так как все состояния автомата закодированы, то работа алгоритма заканчивается. Общее количество переключений триггеров:

$$\begin{aligned} w &= \sum p(i, j) d(i, j) = \\ &= p(2, 3) \cdot d(2, 3) + p(1, 2) \cdot d(1, 2) + p(3, 4) \cdot d(3, 4) + p(3, 5) \cdot d(3, 5) + \\ &+ p(1, 3) \cdot d(1, 3) + p(1, 5) \cdot d(1, 5) + p(2, 4) \cdot d(2, 4) + p(2, 5) \cdot d(2, 5) = \\ &= 3 \cdot 1 + 2 \cdot 1 + 2 \cdot 1 + 2 \cdot 1 + 1 \cdot 2 + 1 \cdot 3 + 1 \cdot 2 + 1 \cdot 2 = 18. \end{aligned}$$

Минимально возможное количество переключений (если бы состояния были закодированы соседним кодированием):

$$w_{\min} = \sum p(i, j) = 13.$$

Коэффициент эффективности кодирования при этом:

$$K_{\text{эф}} = \frac{w_{\min}}{w} = \frac{13}{18} \approx 0,72.$$

Рассмотренный алгоритм кодирования является машинно-ориентированным, и существуют различные версии программ, реализующих этот алгоритм.

В заключение необходимо отметить, что использование алгоритма кодирования для D -триггеров или эвристического алгоритма для других типов триггеров обеспечивает наиболее простую с точки зрения реализации схему, но при этом возможны гонки. Для радикального устранения гонок в автомате используют аппаратные

методы – триггеры с двойной памятью: триггеры, управляемые фронтом и др.

Контрольные вопросы и упражнения

1. Определите число структурных входов автомата при кодировании символами русского алфавита, если мощность входного алфавита абстрактного автомата равна 128.

2. Какое количество состояний абстрактного автомата можно представить с помощью пятиразрядных десятичных чисел?

3. Дайте определение полноты системы переходов автомата.

4. Дайте определение полноты системы выходов автомата Мура.

5. Поясните сущность функций возбуждения автомата.

6. Запишите в совершенных ДНФ и КНФ булеву функцию $y = f(x_1, x_2, x_3)$, принимающую значение 1 на наборах с номерами 3, 4, 7.

7. Найдите минимальную ДНФ функции $y = f(x_1, x_2, x_3, x_4)$, принимающей значение 1 на наборах 0, 1, 2, 5, 6, 7, 8, 12, 13.

8. Функция $y = f(x_1, x_2, x_3)$ равна 1 на наборах 1, 3, 4 и не определена на наборе с номером 5. Найдите её минимальную ДНФ.

9. По кодированной таблице переходов (см. табл. 2.12) определите функции возбуждения автомата с использованием T -триггеров.

10. Перечислите основные методы противогоночного кодирования.

11. Перечислите требования к графу автомата, допускающего соседнее кодирование.

Глава 3. МИКРОПРОГРАММНОЕ УПРАВЛЕНИЕ ДИСКРЕТНЫМИ УСТРОЙСТВАМИ

3.1. Принцип микропрограммного управления

Все устройства дискретных систем автоматики, вычислительной и информационной техники, а также систем управления предназначены для переработки (преобразования) информации, т.е. выполнения над ней соответствующих операций. Для выполнения операций над информацией, представленной в цифровой форме, используются операционные устройства – процессоры, блоки памяти, каналы ввода–вывода, устройства управления внешними устройствами и т.д. Функцией операционного устройства является выполнение заданного множества операций $F = \{f_1, \dots, f_G\}$ над входными словами (данными) $D = \{d_1, \dots, d_H\}$ с целью вычисления слов $R = \{r_1, \dots, r_Q\}$, которые представляют результаты операций над данными $R = f_g(D)$, где $g = 1, 2, \dots, G$.

Функциональная и структурная организация операционных устройств базируется на принципе микропрограммного управления, который состоит в следующем.

1. Любая операция f_g ($g = 1, \dots, G$), реализуемая устройством, рассматривается как сложное действие, которое разделяется на распределённую во времени последовательность элементарных действий над словами информации. Эти элементарные действия, выполняемые за один такт, называются *микрооперациями*.

2. Для управления порядком следования микроопераций используются *логические условия* $X = \{x_1, \dots, x_n, \dots, x_N\}$, которые в зависимости от значений слов, преобразуемых микрооперациями, принимают значения "ложь" (0) или "истина" (1).

3. Процесс выполнения операций в устройстве описывается в форме алгоритма, который представляется в терминах микроопераций и логических условий и называется *микропрограммой*.

Таким образом, из принципа микропрограммного управления следует, что структура и порядок функционирования дискретных устройств предопределяется алгоритмами выполнения операций $F = \{f_1, \dots, f_G\}$.

Для инициализации той или иной микрооперации дискретное устройство должно вырабатывать управляющие сигналы: $Y = \{y_1, \dots, y_m, \dots, y_M\}$. Если в устройстве одновременно реализуется несколько микроопераций, то это множество микроопераций называется *микрокомандой* и обозначается прописной буквой Y с индексами. Так, если $Y_t = \{y_{t1}, \dots, y_{tus}, \dots, y_{tU}\}$ – микрокоманда, то микрооперации $y_{t1}, \dots, y_{tus}, \dots, y_{tU}$ выполняются одновременно в один и тот же момент времени. Микрооперация может быть инициализирована одним или несколькими управляющими сигналами в зависимости от структурной организации устройства.

3.2 Структуризация дискретного устройства

Любое устройство обработки цифровой информации в структурном и функциональном отношении можно представить в виде двух основных блоков – операционного автомата (ОА) и управляющего автомата (УА) (рис. 3.1).

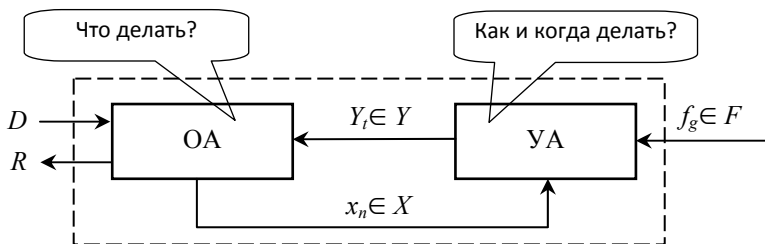


Рис. 3.1. Структурная схема дискретного устройства

ОА служит для хранения слов информации, выполнения набора микроопераций и вычисления значений логических условий, то есть является структурой, организованной для выполнения действий над информацией. Микрооперации, реализуемые ОА, инициализируются множеством управляющих сигналов, составляющих микрокоманду $Y_t \in Y = \{y_1, \dots, y_M\}$.

Значения логических условий, вычисляемые в ОА, отображаются множеством осведомительных сигналов $X = \{x_1, \dots, x_N\}$, каждый из которых отождествляется с определённым логическим условием.

УА генерирует распределённую во времени последовательность управляющих сигналов, предписанную микропрограммой и соответствующую значениям логических условий. Иначе говоря, УА задаёт порядок выполнения действий в ОА, вытекающий из алгоритма выполнения операций. Наименование операции, которую необходимо выполнить в устройстве, определяется кодом $f_g \in F$ операции, поступающим в УА извне. По отношению к УА сигналы $f_g \in F$, посредством которых кодируется наименование операции и осведомительные сигналы $x_n \in X$, формируемые в ОА, играют одинаковую роль. Поэтому сигналы $f_g \in F$ и $x_n \in X$ относятся к одному классу – классу осведомительных сигналов, поступающих на вход УА.

ОА, реализуя действия над словами информации, является исполнительной частью устройства цифровой обработки, работу которой инициализирует УА, генерирующий необходимые последовательности управляющих сигналов.

Операционный и управляющий автоматы могут быть определены своими функциями – перечнем выполняемых ими действий.

Функция ОА определяется следующей совокупностью сведений:

- множеством входных слов $D = \{d_1, ..., d_H\}$, вводимых в автомат в качестве данных;
- множеством выходных слов $R = \{r_1, ..., r_Q\}$, представляющих результаты операций;
- множеством внутренних слов $S = \{s_1, ..., s_P\}$, используемых для представления информации в процессе выполнения операций. Можно считать, что входные и выходные слова совпадают с определёнными внутренними $D \subseteq S, R \subseteq S$;
- множеством микроопераций $Y = \{y_m\}$, реализующих преобразование $S = \varphi_m(s)$ над словами информации, где φ_m – вычисляемая функция;
- множеством логических условий $X = \{x_n\}$, где $x_n = \psi_n(s_n)$ и ψ_n – булева функция.

Функция ОА задана, если заданы (определены) множества D, R, S, Y, X . Время не является аргументом функции ОА. Функция устанавливает список действий-микроопераций и логических условий,

которые может выполнять автомат, но никак не определяет порядок следования этих действий во времени. То есть функция ОА характеризует средства, которые могут быть использованы для вычислений, но не сам вычислительный процесс.

Задача синтеза операционных устройств вследствие их регулярности и большой повторяемости в настоящее время хорошо изучена и не представляет серьёзных трудностей. Иначе обстоит дело с устройством управления, которое является наименее регулярной частью дискретной системы и совершенно не повторяется при переходе от одной системы к другой. В связи с этим проектирование устройства управления представляет наибольшую сложность и, более того, часто проектирование системы после выбора основных частей операционного устройства сводится в основном к проектированию устройства управления.

Функция УА – это операторная схема алгоритма (микропрограмма), функциональными операторами которой являются управляющие сигналы, отождествляемые с микрооперациями, и в качестве логических условий используются булевы переменные $x_1, \dots, x_N$. Схема алгоритма в терминах микроопераций и логических условий наиболее наглядно описывается граф-схемой алгоритма, которая определяет вычислительный процесс последовательно во времени, устанавливая порядок проверки логических условий и порядок следования микроопераций.

3.3. Граф-схемы алгоритмов

При описании алгоритма в виде граф-схемы алгоритма (ГСА), представляющей собой ориентированный связный граф, обычно используют вершины четырёх типов: начальную, конечную, операторную и условную (рис. 3.2).

Граф микропрограммы состоит из совокупности перечисленных вершин и дуг, соединяющих выходы одних вершин с входами других. Соединение вершины и направление дуг графа определяют исходя из алгоритма операции, описываемого графом, и структуры ОА. Сама микропрограмма и её граф должны быть корректны, то есть отвечать определенным условиям:

- в графе должна быть только одна начальная и одна конечная вершина;
- в любую вершину графа должен вести, по крайней мере, один путь из начальной вершины;
- из каждого выхода любой вершины графа должен существовать, по крайней мере, один путь в конечную вершину;
- при всех возможных значениях логических условий и используемых слов должен существовать путь из начальной вершины в конечную.



Рис. 3.2. Типы вершин граф-схемы алгоритма

Примеры ГСА представлены на рис. 3.3.

ГСА на рис. 3.3, *а* называется содержательной, так как внутри вершин записаны в явном виде микрооперации и логические выражения. Если же каждой микрооперации поставить в соответствие множество управляющих сигналов $Y_i \in Y = \{y_1, \dots, y_M\}$, их инициализирующих, а логическим выражениям в условных вершинах – булевы переменные $x_j \in X$, то получится так называемая кодированная ГСА (рис. 3.3, *б*).

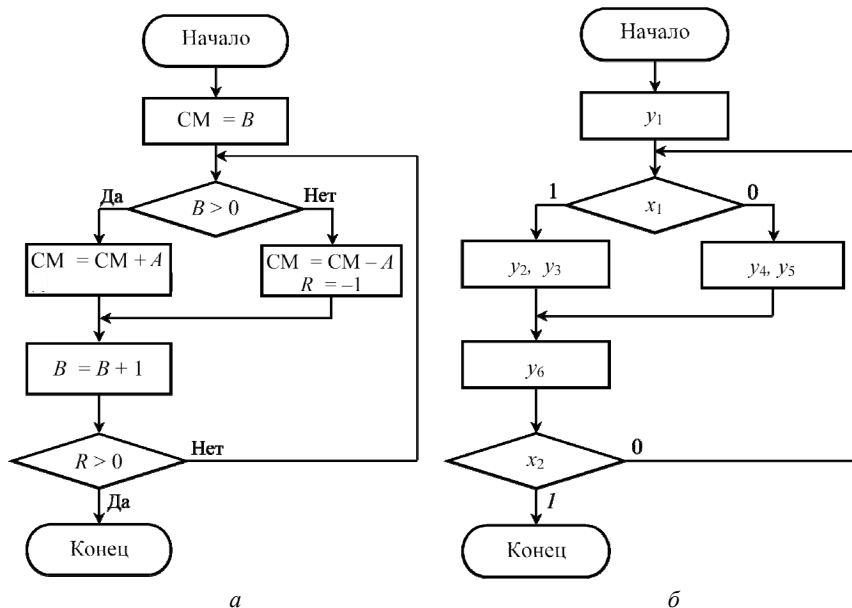


Рис. 3.3. Содержательная (а) и кодированная (б) ГСА

3.4. Типовые операционные элементы

Согласно принципу микропрограммного управления, любая сложная операция распадается на последовательность микрокоманд, состоящих из структурно и функционально совместимых микроопераций, выполняемых за один такт автоматного времени. Различные микрооперации выполняются *операционными элементами* (ОЭ), которые являются составными частями операционного автомата.

Под ОЭ понимают устройство, реализующее одну из следующих функций или их произвольную комбинацию:

- хранение слова информации (данных);
- выполнение некоторых микроопераций, в результате которых выполняется преобразование данных;
- вычисление логического условия, зависящего от состояния данных.

В зависимости от выполняемых микроопераций ОЭ как регулярные структуры делятся на типовые разновидности: шина, регистр, счётчик, сумматор, схема сравнения, дешифратор, шифратор и т.д. Рассмотрим основные, наиболее часто встречаемые, типы ОЭ.

Шина – это совокупность цепей, предназначенных для передачи слова данных. Шина определённой разрядности может быть неуправляемой или управляемой, когда передача осуществляется только при поступлении управляющего сигнала. Под действием управляющего сигнала (y_i) инициализируется выполнение микрооперации: $B[0:3] := A[0:3]$. Разрядность шины, указанная в квадратных скобках, может быть произвольной.

Регистр – это ОЭ, служащий для запоминания слов и обеспечивающий в общем случае выполнение ряда микроопераций:

- установка регистра в 0 (сброс);
- приём слова из другого регистра, шины и т.д.;
- передача слова в другой регистр, шину и т.д.;
- преобразование кодов хранимых слов в инверсные коды;
- сдвиг хранимого слова влево или вправо на требуемое число разрядов.

Регистр, выполняющий такие микрооперации, называется многофункциональным. Так как регистр предназначен для хранения информации, то он содержит элементы памяти, в качестве которых используются триггеры. Количество триггеров определяет разрядность регистра. На структурном уровне регистр обычно обозначается в виде прямоугольника с указанием разрядности (рис. 3.4, *а*). Регистр может состоять из отдельных подрегистров, имеющих самостоятельное значение. Например, на рис. 3.4, *б* представлен регистр, хранящий число в форме с плавающей запятой. В этом регистре три подрегистра: для хранения знака $Rg[0]$, характеристики $Rg[1:7]$ и мантиссы $Rg[8:31]$ числа.

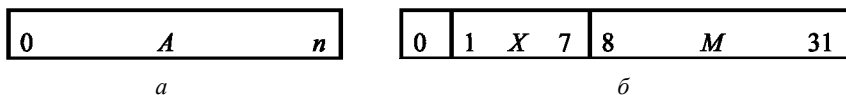


Рис. 3.4. Регистр $A[0:n]$ (*а*) и регистр для хранения числа в форме с плавающей запятой (*б*)

В табл. 3.1 приведены примеры микроопераций, выполняемые многофункциональным регистром.

Таблица 3.1

Микрооперации, выполняемые многофункциональным регистром

$\mu_1:$	$\text{Pr}[0]:= 0$	$\mu_6:$	$\text{Pr}[0:n]:= \text{Pr}[1:n].0$
$\mu_2:$	$\text{Pr}[0:n]:= A[0:n]$	$\mu_7:$	$\text{Pr}[0:n]:= \text{Pr}[1:n]. \text{Pr}[0]$
$\mu_3:$	$B[0:6]:= \text{Pr}[1:7]$	$\mu_8:$	$D[0:n]:= \text{Pr}[0:n]$
$\mu_4:$	$\text{Pr}[0:n]:= 0.\text{Pr}[0:n - 1]$	$\mu_9:$	$D[0:n]:= -\text{Pr}[0:n]$
$\mu_5:$	$\text{Pr}[0:n]:= \text{Pr}[n].\text{Pr}[0:n - 1]$	$\mu_{10}:$	$\text{Pr}[0:n]:= 0$

Регистр, который выполняет микрооперации μ_4 (сдвиг вправо), μ_6 (сдвиг влево) или μ_5 (циклический сдвиг вправо), μ_7 (циклический сдвиг влево) называется регистром сдвига.

Сумматор – операционный элемент, выполняющий суммирование кодов чисел. В зависимости от кодов чисел различают сумматоры прямого, обратного и дополнительного кодов. Кроме того, сумматоры бывают комбинационными и накапливающими.

Комбинационный сумматор вырабатывает выходные сигналы суммы и переноса, определяемые комбинацией цифр слагаемых, одновременно поданных на входы сумматора. Данный сумматор не обладает памятью и после снятия сигналов с входов выходные сигналы также исчезают.

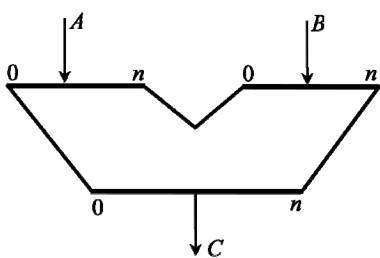


Рис. 3.5. Комбинационный сумматор: A, B – операнды, C – результат

Реализуемая модель арифметического суммирования: $C = A + B$. Условное графическое обозначение комбинационного сумматора на структурных схемах представлено на рис. 3.5.

Накапливающим называется сумматор, который осуществляет сложение слов при подаче их на сумматор одного за другим. В

накапливающем сумматоре имеется дополнительный регистр для хранения результата, позволяющий реализовать другую модель суммирования в виде $C = C + A$.

Структурно-накапливающий сумматор может быть реализован, как показано на рис. 3.6 или на регистровой структуре в режиме суммирования. Условное графическое обозначение накапливающего сумматора показано на рис. 3.7.

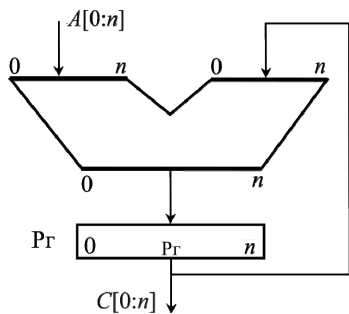


Рис. 3.6. Накапливающий сумматор

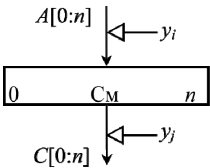


Рис. 3.7. Условное графическое обозначение накапливающего сумматора

Счётчик – операционный элемент, который реализует микрооперацию инкрементного и (или) декрементного счёта. Микрооперация счёта состоит в изменении состояния счётчика (значения хранимого слова) на единицу. Кроме того, счётчик может выполнять и такие микрооперации, как установка в ноль и приём (выдачу) произвольного числа. Полный набор возможных микроопераций для счетчика дан в табл. 3.2.

Таблица 3.2

Микрооперации, выполняемые счетчиком

$\mu_1:$	$Сч[0:n] := Сч[0:n] + 1$
$\mu_2:$	$Сч[0:n] := Сч[0:n] - 1$
$\mu_3:$	$Сч[0:n] := 0$
$\mu_4:$	$Сч[0:n] := A[0:n]$
$\mu_5:$	$A[0:n] := Сч[0:n]$

Счётчик, выполняющий как микрооперацию инкрементного счёта, так и декрементного, называется реверсивным.

Контрольные вопросы и упражнения

1. Дайте определение понятиям «микрооперация», «микрокоманда», «микропрограмма».
2. Поясните принцип микропрограммного управления.
3. Поясните сущность декомпозиции дискретного устройства на управляющий и операционный автоматы.
4. Перечислите условия корректности граф-схемы алгоритма.
5. Перечислите микрооперации, реализуемые регистром, как операционным элементом.

Глава 4. СИНТЕЗ МИКРОПРОГРАММНЫХ АВТОМАТОВ ПО ГРАФ-СХЕМЕ АЛГОРИТМА

4.1. Этапы синтеза и построение таблиц переходов

Конечный цифровой автомат, реализующий микропрограмму работы дискретного устройства, принято называть микропрограммным автоматом. Одну и ту же ГСА можно интерпретировать как автоматом Мили, так и автоматом Мура. Далее рассматривается только синтез микропрограммного автомата Мили, как наиболее приемлемого для реализации структур управляющих автоматов.

Синтез микропрограммного автомата по ГСА осуществляется в два этапа:

- получение отмеченной ГСА;
- построение графа автомата в виде списковых структур (таблиц переходов).

На этапе получения отмеченной ГСА *входы* вершин отмечают-ся символами $a_1, a_2, \dots, a_H$ по следующим правилам:

- символом a_1 отмечается вход вершины, следующей за начальной, а также вход конечной вершины;
- символами $a_2, \dots, a_H$ отмечаются входы всех вершин, следующих за операторными;
- если вход вершины отмечается, то только одним символом;
- входы различных вершин, за исключением конечной, отмечаются различными символами.

Ясно, что для проведения отметок потребуется конечное число символов. Предположим для определённости, что для отметки входов вершин граф-схемы использовались символы множества $A = \{a_1, \dots, a_h, \dots, a_H\}$. Результатом первого этапа является отмеченная ГСА, которая служит основой для второго этапа – перехода к графу автомата. Пример отмеченной ГСА показан на рис. 4.1. Множество отметок $A = \{a_1, \dots, a_h, \dots, a_H\}$ интерпретируются как множество состояний микропрограммного автомата.

Если идти от одной отметки $a_m \in A$ к другой отметке $a_s \in A$ в направлении ориентации дуг ГСА, выписывая содержимое лежа-

щих на этом пути вершин, то каждому такому пути можно поставить в соответствие цепь в терминах логических условий и микрокоманд.

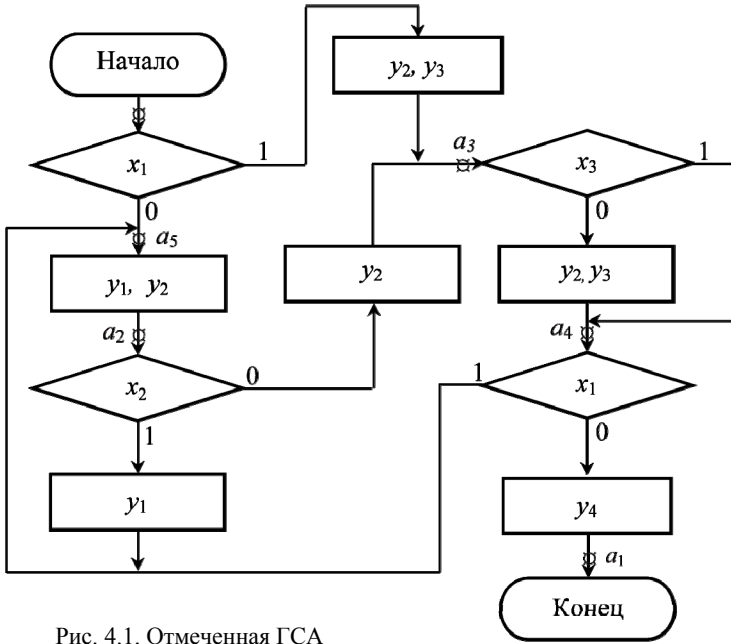


Рис. 4.1. Отмеченная ГСА

Чтобы подчеркнуть, что выписанная цепь соответствует пути из a_m в a_s , будем ограничивать эту цепь слева и справа символами a_m и a_s соответственно.

В дальнейшем рассматриваются только цепочки вида

$$a_m \tilde{x}_{m1} \dots \tilde{x}_{mr} \dots \tilde{x}_{mR} Y_t a_s \quad (4.1)$$

или

$$a_m \tilde{x}_{m1} \dots \tilde{x}_{mr} \dots \tilde{x}_{mR} Y_t a_1. \quad (4.2)$$

Здесь $\tilde{x}_{m1} \dots \tilde{x}_{mr} \dots \tilde{x}_{mR}$ — непрерывная последовательность логических переменных, записанных в условных вершинах по направлению ориентации дуг от одной отметки $a_m \in A$ к другой отметке $a_s \in A$, где

$$\tilde{x}_{mr} = \begin{cases} x_{mr}, & \text{если } x_{mr} = 1; \\ \bar{x}_{mr}, & \text{если } x_{mr} = 0, \end{cases}$$

Y_t – множество микроопераций, записанных в *одной* операторной вершине, после которой стоит отметка a_s .

Соответствующие этим цепочкам пути на СГА называются *путями перехода*. В пути перехода вида (4.1) не исключён случай пустой последовательности логических переменных ($R = 0$): $a_m Y_t a_s$.

Таким образом, путь по ГСА вида (4.1) – это путь перехода автомата из одного состояния a_m в другое состояние a_s (допустимо $a_m = a_s$). Путь по ГСА вида (4.2) – это путь перехода автомата из некоторого состояния a_m в начальное состояние a_1 (не допустимо $a_m = a_1$), при котором $Y_t = \emptyset$.

Каждому пути перехода вида (4.1) или (4.2) ставится в соответствие конъюнкция $X(a_m, a_s) = \bigwedge_{\forall r} \tilde{x}_{mr}$. При пустой последовательности

логических переменных путь перехода является безусловным и $X(a_m, a_s) = 1$. Для краткости эти пути перехода будем обозначать как $a_m X(a_m, a_s) Y(a_m, a_s) a_s$ и $a_m X(a_m, a_1) a_1$, где $Y(a_m, a_s) = Y_t$.

После получения отмеченной ГСА строится граф автомата Мили S , состояниями которого являются $\{a_1, \dots, a_h, \dots, a_H\}$, причём a_1 – начальное состояние. Переходы и выходные сигналы в этом автомате определяются следующим образом.

Находятся все пути перехода на отмеченной ГСА. Очевидно, что по законам булевой алгебры, если при некотором r ($r = 1, \dots, R$) имеется несколько вхождений символа x_r в путь перехода, то все символы x_r , кроме одного, из этого пути удаляются. Если при некотором r в путь перехода входит как x_r , так и $\bar{x}_r$, то такой путь перехода в дальнейшем не рассматривается.

Каждому пути перехода вида $a_m X(a_m, a_s) Y(a_m, a_s) a_s$, где $a_m, a_s \in \{a_1, \dots, a_h, \dots, a_H\}$, ставится в соответствие переход автомата S из состояния a_m в состояние a_s под действием входного сигнала $X(a_m, a_s)$ и выдачей выходного сигнала $Y(a_m, a_s)$. Каждому пути перехода вида $a_m Y(a_m, a_s) a_s$, где $a_m, a_s \in \{a_1, \dots, a_h, \dots, a_H\}$, ставится в соответствие переход автомата S из состояния a_m в состояние a_s , где a_m и a_s определяются так же, как и раньше. Входной сигнал на этом переходе также равен конъюнкции содержимого условных вершин в этом пути. Так как множество условных вершин в данном случае пусто, а конъюнкция пустого множества переменных равна едини-

це, то соответствующий входной сигнал полагается равным единице, а выходной сигнал, как и раньше, $Y(a_m, a_s)$.

Каждому пути перехода вида $a_m X(a_m, a_1) a_1$ ставится в соответствие переход из a_m в a_1 . При этом a_m и входной сигнал определяются, как и раньше, а выходной сигнал полагается равным Y_0 (пустой оператор).

В результате получается граф автомата Мили S , имеющий столько же состояний, сколько символов потребовалось для отметки входов вершин ГСА.

При использовании графов для задания автоматов с большим числом состояний и переходов наглядность теряется, поэтому оказывается предпочтительным задавать эти графы в виде списков. В связи с этим вводится понятие таблицы переходов микропрограммного автомата. В случае автомата Мили эта таблица имеет пять столбцов.

Если между состояниями a_m и a_s существует переход под действием входного сигнала $X(a_m, a_s)$ с выдачей выходного сигнала $Y(a_m, a_s)$, то в первом столбце помещается a_m – исходное состояние, во втором – состояние перехода a_s (состояние, в которое осуществляется переход), в третьем – входной сигнал $X(a_m, a_s)$, в четвертом – выходной сигнал $Y(a_m, a_s)$ и в пятом – номер пути перехода для удобства идентификации (P).

Иногда на переходе (a_m, a_s) выдаётся не один, а множество выходных сигналов – микрокоманд $Y(a_m, a_s) = \{Y_1(a_m, a_s), \dots, Y_j(a_m, a_s), \dots, Y_k(a_m, a_s)\}$, причём в общем случае один и тот же выходной сигнал $Y_j(a_m, a_s)$ может выдаваться также под действием множества входных сигналов $X_{jh}(a_m, a_s)$. Тогда в таблице последовательно перечисляются все пути перехода. Различают два вида таблицы переходов: *прямую* и *обратную*.

Прямой таблицей переходов микропрограммного автомата называется таблица, в которой последовательно перечисляются все переходы сначала из первого состояния, затем из второго и т.д. Табл. 4.1 является примером такой таблицы для автомата Мили, ГСА которого представлена на рис. 4.1. В ряде случаев оказывается удобным пользоваться обратной таблицей переходов, в которой сначала записываются все переходы в первое состояние, затем во второе и т.д. (табл. 4.2).

Таблица 4.1

Прямая таблица переходов

a_m	a_s	$X(a_m, a_s)$	$Y(a_m, a_s)$	P
1	2	3	4	5
a_1	a_2	$\bar{x}_1$	$y_1 y_2$	1
	a_3	x_1	$y_2 y_3$	2
a_2	a_3	$\bar{x}_2$	y_2	3
	a_5	x_2	y_1	4
a_3	a_4	$\bar{x}_3$	$y_2 y_3$	5
	a_2	$x_3 x_1$	$y_1 y_2$	6
	a_1	$x_3 \bar{x}_1$	y_4	7
a_4	a_1	$\bar{x}_1$	y_4	8
	a_2	x_1	$y_1 y_2$	9
a_5	a_2	1	$y_1 y_2$	10

Таблица 4.2

Обратная таблица переходов

a_m	a_s	$X(a_m, a_s)$	$Y(a_m, a_s)$	P
1	2	3	4	5
a_3	a_1	$x_3 \bar{x}_1$	y_4	1
a_4		$\bar{x}_1$	y_4	2
a_1	a_2	$\bar{x}_1$	$y_1 y_2$	3
a_3		$x_3 x_1$	$y_1 y_2$	4
a_4		x_1	$y_1 y_2$	5
a_5		1	$y_1 y_2$	6
a_1	a_3	x_1	$y_2 y_3$	7
a_2		$\bar{x}_2$	y_2	8
a_3	a_4	$\bar{x}_3$	$y_2 y_3$	9
a_2	a_5	x_2	y_1	10

Очевидно, что таблицу переходов микропрограммного автомата (прямую или обратную) целесообразно составлять непосредственно по отмеченной ГСА, записывая в неё все пути перехода, т.е. не нужно предварительно рисовать граф автомата, поскольку эта таблица и есть граф, заданный в виде списка.

4.2. Структурная таблица микропрограммного автомата

Структурные таблицы микропрограммного автомата являются расширением таблиц переходов, рассмотренных в п. 4.1. Структурная таблица автомата Мили (табл. 4.3) имеет восемь столбцов.

Состояния a_m из которых осуществляются переходы, указываются в первом столбце. Коды этих состояний $K(a_m)$ после кодирования заносятся во второй столбец. В третьем и четвёртом столбцах записываются состояния a_s , в которые происходят переходы, и их коды $K(a_s)$. Пятый и шестой столбцы содержат входные $X(a_m, a_s)$ и выходные сигналы $Y(a_m, a_s)$, входящие в пути перехода. В седьмом столбце таблицы перечисляются обязательные функции возбуждения $\Psi(a_m, a_s)$, вырабатываемые на соответствующих переходах в зависимости от типа выбранного триггерного устройства и в восьмом – номер пути перехода для удобства идентификации.

Таблица 4.3

Общий вид структурной таблицы

Исходное состояние	Код исходного состояния	Состояние перехода	Код состояния перехода	Входной сигнал	Выходной сигнал	Обязательные функции возбуждения	Номер перехода
1	2	3	4	5	6	7	8
a_m	$K(a_m)$	a_s	$K(a_s)$	$X(a_m, a_s)$	$Y(a_m, a_s)$	$\Psi(a_m, a_s)$	P

Если между двумя состояниями существует несколько путей перехода, даже содержащие один и тот же выходной сигнал, для каждого пути отводится в таблице отдельная строчка, в связи с чем в каждой строке столбца «Входной сигнал» содержится ровно одна конъюнкция.

Структурные таблицы, так же как и таблицы переходов микропрограммного автомата, могут быть прямые и обратные. В прямой таблице последовательно перечисляются все переходы сначала из первого состояния, затем из второго и т. д. Таблица 4.4 представляет собой прямую структурную таблицу автомата Мили, построенную по таблице переходов 4.1 при использовании в качестве элементов памяти D -триггеров.

Для простоты в этой таблице в качестве кода каждого состояния взят двоичный эквивалент его номера, например $K(a_3) = 011$, $K(a_4) = 100$ и т.д. В обратной структурной таблице столбцы обозначены так же, но сначала записываются все переходы в первое состояние, затем во второе и т. д. В таблице 4.5 приведена обратная структурная таблица того же автомата.

Таблица 4.4

Прямая структурная таблица автомата Мили

a_m	$K(a_m)$	a_s	$K(a_s)$	$X(a_m, a_s)$	$Y(a_m, a_s)$	$\Psi(a_m, a_s)$	P
1	2	3	4	5	6	7	8
a_1	001	a_2	010	$\bar{x}_1$	$y_1 y_2$	D_2	1
		a_3	011	x_1	$y_2 y_3$	$D_1 D_2$	2
a_2	010	a_3	011	$\bar{x}_2$	y_2	$D_1 D_2$	3
		a_5	101	x_2	y_1	$D_1 D_3$	4
a_3	011	a_4	100	$\bar{x}_3$	$y_2 y_3$	D_3	5
		a_2	010	$x_3 x_1$	$y_1 y_2$	D_2	6
		a_1	001	$x_3 \bar{x}_1$	y_4	D_1	7
a_4	100	a_1	001	$\bar{x}_1$	y_4	D_1	8
		a_2	010	x_1	$y_1 y_2$	D_2	9
a_5	101	a_2	010	1	$y_1 y_2$	D_2	10

Обратная структурная таблица автомата Мили

a_m	$K(a_m)$	a_s	$K(a_s)$	$X(a_m, a_s)$	$Y(a_m, a_s)$	$\Psi(a_m, a_s)$	P	
1	2	3	4	5	6	7	8	
a_3	001	a_1	001	$x_3 \bar{x}_1$	y_4	D_1	1	$\Phi(a_1)$
a_4	100			$\bar{x}_1$	y_4	D_1	2	
a_1	010	a_2	010	$\bar{x}_1$	$y_1 y_2$	D_2	3	$\Phi(a_2)$
a_3	011			$x_3 x_1$	$y_1 y_2$	D_2	4	
a_4	100			x_1	$y_1 y_2$	D_2	5	
a_5	101			1	$y_1 y_2$	D_2	6	
a_1	001	a_3	011	x_1	$y_2 y_3$	$D_1 D_2$	7	$\Phi(a_3)$
a_2	010			$\bar{x}_2$	y_2	$D_1 D_2$	8	
a_3	011	a_4	100	$\bar{x}_3$	$y_2 y_3$	D_3	9	$\Phi(a_4)$
a_2	010	a_5	101	x_2	y_1	$D_1 D_3$	10	$\Phi(a_5)$

4.3. Построение схемы по структурной таблице

Так как структурная таблица представляет собой граф микро-программного автомата, заданный в виде списка, из этой таблицы можно получать выражения функций возбуждения и функций выходов в виде дизъюнкции конъюнкций кода состояния, из которого происходит переход $K(a_m)$, и входного сигнала, под действием которого происходит переход $X(a_m, a_s)$. Дизъюнкция берётся по всем строчкам (переходам) структурной таблицы, которые помечены символом конкретной функции возбуждения и управляющего сигнала: $f_i = \bigvee_{\forall p} K(a_m) X(a_m, a_s)$.

Однако для встречающихся на практике микропрограммных автоматов подобные выражения очень сложны: они представляют собой систему из десятков и сотен функций десятков и сотен переменных. В этом случае известные классические методы минимизации булевых функций, работающие с выражениями в классе нормальных форм, оказываются неприемлемыми. Кроме того, практика показала, что основное сокращение объёма схемы происходит не за счёт минимизации в нормальных формах, а в результате выделения ряда функций и подфункций, допускающих совместную минимизацию, и представления систем функций в виде декомпозиции.

Первым шагом при минимизации логической схемы микропрограммного автомата является объединённое построение компонент функций возбуждения и функций выходов, записанных в одних и тех же строчках структурной таблицы. Не вдаваясь в подробности можно отметить, что построение функционально-логической схемы автомата, исходя из этой концепции, удобнее производить по обратной структурной таблице.

В обратной структурной табл. 4.5 $\Phi(a_i)$ – это все пути перехода в некоторое состояние, которые можно реализовать в виде $f_i = \bigvee_p K(a_m)X(a_m, a_s)$ (рис. 4.2).

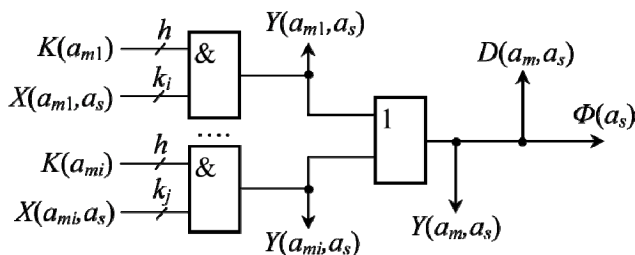


Рис. 4.2. Обобщённый путь перехода в произвольное состояние

Дальнейшая дизъюнктивная сборка функций возбуждения для всех D -триггеров осуществляется с обобщённых путей перехода $\Phi(a_s)$. Одноимённые управляющие сигналы также собираются с обобщённых путей перехода, но только в том случае, если на всех переходах в конкретное состояние a_s вырабатываются одни и те же управляющие сигналы: $Y(a_{m1}, a_s) = \dots = Y(a_{mi}, a_s)$. В противном случае

сборка одноимённых управляющих сигналов производится с конъюнкций, реализующих конкретный путь перехода. Используя ассоциативный закон булевой алгебры, коды исходных состояний $K(a_m)$ представляются переменными, которыми закодированы состояния (выходными сигналами триггерных устройств), или выходными сигналами преддешифраторов или полных дешифраторов состояний.

Функционально-логическая схема для автомата по структурной таблице переходов (см. табл. 4.5) с использованием полной дешифрации состояний представлена на рис. 4.3.

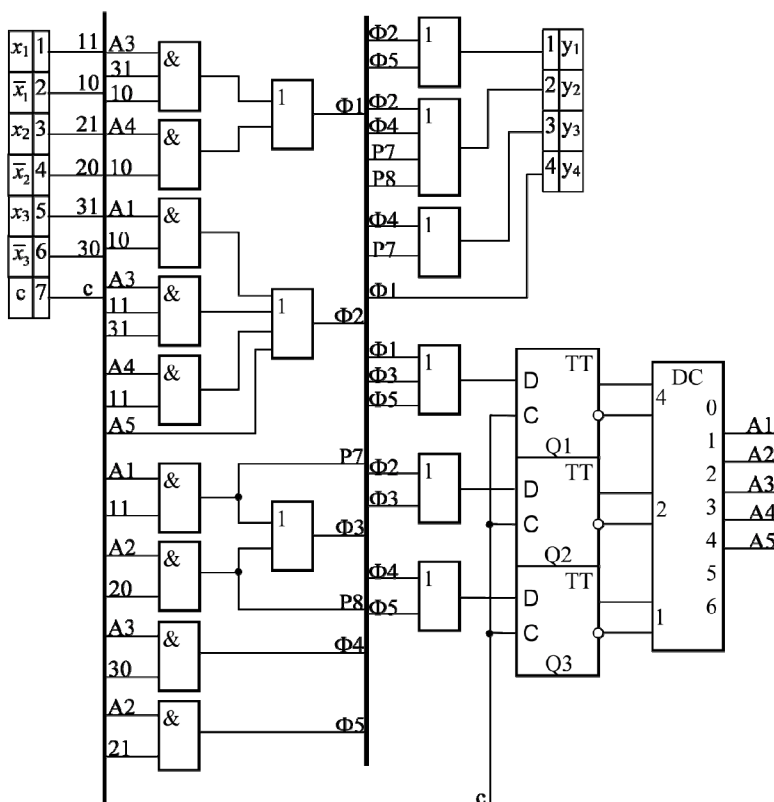


Рис. 4.3. Функционально-логическая схема автомата Мили, построенная по структурной таблице переходов (табл. 4.5) согласно ГСА, представленной на рис. 4.1

4.4. Задача факторизации

Задача факторизации основана на использовании дистрибутивного закона булевой алгебры. Для примера возьмём реализацию всех путей перехода в состояние a_2 (см. табл. 4.5). Цена реализации этих путей по Квайну (т.е. сумма всех входов конъюнктивных и дизъюнктивных элементов без использования частичной или полной дешифрации состояний) составляет 20 единиц ($W = 20$).

Функция $\Phi(a_2)$ выглядит следующим образом:

$$\Phi(a_2) = B_1 \vee B_2 \vee B_3 \vee B_4, \quad (4.3)$$

где $B_1 = q_1 \bar{q}_2 \bar{q}_3 \bar{x}_1$; $B_2 = q_1 q_2 \bar{q}_3 x_1 x_3$; $B_3 = \bar{q}_1 \bar{q}_2 q_3 x_1$; $B_4 = q_1 \bar{q}_2 q_3$.

Очевидно, что минимизация функции $\Phi(a_2)$ в классе нормальных форм невозможна.

Пусть задана функция

$$f = X_1 \vee X_2 \vee X_3 \vee X_4 \vee X_5, \quad (4.4)$$

где

$$\begin{aligned} X_1 &= x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_{11}; & X_2 &= x_1 x_2 x_3 x_8; & X_3 &= x_1 x_2 x_5 x_6 x_{10} x_{11} x_{12}; \\ X_4 &= x_5 x_6 x_9; & X_5 &= x_1 x_2 x_5 x_6 x_{10} x_{12} x_{13}. \end{aligned}$$

Возможна минимизация схемы, реализующей эту функцию, если предварительно вынести общий множитель из всех или нескольких конъюнкций. Так, формулу (4.3) можно преобразовать:

$$f = x_1 x_2 (X'_1 \vee X'_2 \vee X'_3 \vee X'_5) \vee X_4. \quad (4.5)$$

Схема для формулы (4.3) требует на 4 входа в логические элементы меньше, чем схема для формулы (4.2).

В общем случае, если имеется дизъюнкция n конъюнкций

$$f = X_1 \vee \dots \vee X_i \vee \dots \vee X_n, \quad (4.6)$$

где каждая конъюнкция $X_i = x_1 \dots x_m z_{i1} \dots z_{ik_i}$ ($i = 1, \dots, n$), т.е. у всех конъюнкций есть общий множитель $x_1, \dots, x_m$, он может быть вынесен за скобки, и тогда очевидно, что выигрыш в числе входов в логические элементы по сравнению с непосредственным построением по формуле (4.6) будет

$$\Delta W = mn - m - 1 = m(n - 1) - 1,$$

где m – число общих букв (с отрицанием или без него) во всех конъюнкциях, n – число конъюнкций.

Может случиться, что вынесение возможно не из всех конъюнкций. Так, если

$$f = X_1 \vee \dots \vee X_n \vee X_{n+1} \vee \dots \vee X_t, \quad (4.7)$$

причём только первые n конъюнкций содержат общий член $x_1, \dots, x_m$: $X_i = x_1 \dots x_m z_{i1} \dots z_{ik_i}$ ($i = 1, \dots, n$), а остальные ($t - n$) членов имеют вид $X_j = z_{j1} \dots z_{jk_j}$ ($j = n+1, \dots, t$), то при реализации функции f выигрыш в цене схемы составит:

$$\Delta W = mn - m - 1 - 1 = m(n-1) - 2.$$

Если к тому же у r из n конъюнкций после вынесения общего члена $x_1, \dots, x_m$ остаётся по одной букве: $X_p = x_1 \dots x_m z_p$, то имеет место наиболее общий случай, при котором выигрыш при таком построении схемы по сравнению с непосредственной реализацией составит

$$\Delta W = mn - m - 2 + r = m(n-1) - 2 + r.$$

Рассмотрим один из возможных алгоритмов решения задачи вынесения общего множителя за скобки на примере реализации функции (4.2). Пересечения множеств букв, входящих в конъюнкции $X_1, \dots, X_5$, дают всевозможные наибольшие общие части, полученные попарным сравнением. Эти общие части могут быть вынесены за скобки:

$X_1 = x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_{11}$	X_1			
$X_2 = x_1, x_2, x_3, x_8$	x_1, x_2, x_3	X_2		
$X_3 = x_1, x_2, x_5, x_6, x_{10}, x_{11}, x_{12}$	$x_1, x_2, x_5, x_6, x_{11}$	x_1, x_2	X_3	
$X_4 = x_5, x_6, x_9$	x_5, x_6	—	x_5, x_6	X_4
$X_5 = x_1, x_2, x_5, x_6, x_{10}, x_{12}, x_{13}$	x_1, x_2, x_5, x_6	x_1, x_2	$x_1, x_2, x_5, x_6, x_{10}, x_{12}$	x_5, x_6

В нашем примере это будут общие части:

$$\begin{aligned} Z_1 &= x_1, x_2, x_3 (X_1, X_2); & Z_2 &= x_1, x_2, x_5, x_6, x_{11} (X_1, X_3); \\ Z_3 &= x_5, x_6 (X_1, X_3, X_4, X_5); & Z_4 &= x_1, x_2, x_5, x_6 (X_1, X_3, X_5); \\ Z_5 &= x_1, x_2 (X_1, X_2, X_3, X_5); & Z_6 &= x_1, x_2, x_5, x_6, x_{10}, x_{12} (X_3, X_5). \end{aligned}$$

В скобках после общей части указаны конъюнкции, в которые она входит. По приведённым выше формулам легко подсчитать

цену каждой из общих частей $Z_1, \dots, Z_6$ как выигрыш, который даёт вынесение этой общей части:

$$\begin{aligned} W(Z_1) &= 2; & W(Z_2) &= 3; & W(Z_3) &= 5; \\ W(Z_4) &= 6; & W(Z_5) &= 4; & W(Z_6) &= 6. \end{aligned}$$

На первом шаге для вынесения выбирается общая часть, имеющая наибольшую цену: $W(Z_4) = \max \{W(Z_1), \dots, W(Z_6)\} = 6$. В результате вынесения общей части Z_4 из исходного множества конъюнкций $A_1 = \{X_1, \dots, X_5\}$ образуются два множества: $A_2 = \{X_6, X_7, X_8\}$ – конъюнкции, из которых удалены буквы, входящие в Z_4 ($X_6 = X_1 \setminus Z_4 = \{x_3, x_4, x_7, x_{11}\}$; $X_7 = X_3 \setminus Z_4 = \{x_{10}, x_{11}, x_{12}\}$; $X_8 = X_5 \setminus Z_4 = \{x_{10}, x_{12}, x_{13}\}$), и $B_2 = \{X_2, X_4, Z_4\}$ – конъюнкции, из которых не делалось вынесение, плюс общая часть Z_4 .

Исходя из сущности дистрибутивного закона очевидно, что дальнейшее вынесение возможно только раздельно для множеств A_2 и B_2 . Продолжая аналогично отдельно для A_2 и B_2 до тех пор, пока не найдётся ни одной общей части, имеющей цену больше нуля, получим процедуру вынесения за скобки, в которой на каждом шаге выбирается общая часть, определяемая по наибольшей цене. Такой алгоритм не обязательно приведёт к оптимальной схеме, но в большинстве случаев даёт решение, близкое к оптимальному.

Ищем пересечения слов в множестве A_2 :

$$\begin{array}{l|l|l} X_6 = x_3, x_4, x_7, x_{11} & X_6 & \\ X_7 = x_{10}, x_{11}, x_{12} & x_{11} & X_7 \\ X_8 = x_{10}, x_{12}, x_{13} & - & x_{10}, x_{12} \end{array}$$

$$\begin{aligned} Z_7 &= x_{11} (X_6, X_7); & W(Z_7) &= -1; \\ Z_8 &= x_{10}, x_{12} (X_7, X_8); & W(Z_8) &= 2. \end{aligned}$$

Вынесение Z_8 порождает два множества A_3, B_3 :

$$A_3 = \{X_9, X_{10}\}; \quad X_9 = X_7 \setminus Z_8 = \{x_{11}\}; \quad X_{10} = X_8 \setminus Z_8 = \{x_{13}\}; \quad B_3 = \{X_6, Z_8\}.$$

В каждом из множеств A_3, B_3 оставшиеся члены не пересекаются. Дальнейшие вынесения невозможны.

Переходим к множеству B_2 :

$$\begin{array}{l|l|l}
X_2 = x_1, x_2, x_3, x_8 & X_2 & \\
X_4 = x_5, x_6, x_9 & - & X_4 \\
Z_4 = x_1, x_2, x_5, x_6 & x_1, x_2 & x_5, x_6
\end{array}$$

$$Z_9 = x_1, x_2 (X_2, Z_4); \quad W(Z_9) = 0;$$

$$Z_{10} = x_5, x_6 (X_4, Z_4); \quad W(Z_{10}) = 1.$$

Вынесение Z_{10} порождает два множества, A_4, B_4 :

$$A_4 = \{X_{11}, X_{12}\}; \quad X_{11} = X_4 \setminus Z_{10} = \{x_9\}; \quad X_{12} = Z_4 \setminus Z_{10} = \{x_1, x_2\}; \quad B_4 = \{X_2, Z_{10}\}.$$

В каждом из множеств A_4, B_4 элементы не пересекаются, дальнейшие вынесения невозможны.

Функционально-логическая схема, соответствующая описанной процедуре, приведена на рис. 4.4. Выигрыш в результате вынесений общих частей:

$$W = W(Z_4) + W(Z_8) + W(Z_{10}) = 9.$$

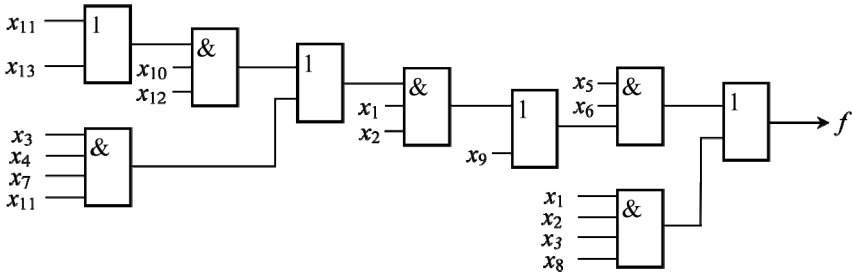


Рис. 4.4. Пример вынесения за скобки

К сожалению, решение задачи факторизации при построении функции $\Phi(a_2)$ (4.3) не приводит к снижению цены схемы уже на первом этапе:

$$\begin{array}{l|l|l|l}
B_1 = q_1 \bar{q}_2 \bar{q}_3 \bar{x}_1 & B_1 & & \\
B_2 = q_1 q_2 \bar{q}_3 x_1 x_3 & q_1 \bar{q}_3 & B_2 & \\
B_3 = \bar{q}_1 \bar{q}_2 q_3 x_1 & \bar{q}_2 & x_1 & B_3 \\
B_4 = q_1 \bar{q}_2 q_3 & q_1 \bar{q}_2 & q_1 & \bar{q}_2 q_3
\end{array}$$

$$\begin{array}{lll}
Z_1 = q_1 \bar{q}_3 (B_1, B_2); & Z_2 = \bar{q}_2 (B_1, B_3, B_4); & Z_3 = q_1 \bar{q}_2 (B_1, B_4); \\
Z_4 = x_1 (B_2, B_3); & Z_5 = q_1 (B_1, B_2, B_4); & Z_6 = \bar{q}_2 q_3 (B_3, B_4).
\end{array}$$

$$\begin{aligned} W(Z_1) &= 0; & W(Z_2) &= 0; & W(Z_3) &= 0; \\ W(Z_4) &= -1; & W(Z_5) &= 0; & W(Z_6) &= 0. \end{aligned}$$

4.5 Декомпозиция схемы из однотипных элементов

Задача декомпозиции схемы из однотипных элементов основана на использовании ассоциативного закона булевой алгебры. Пусть задана система функций:

$$\left. \begin{aligned} f_1 &= x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_{11}; \\ f_2 &= x_1 x_2 x_3 x_8; \\ f_3 &= x_1 x_2 x_5 x_6 x_{10} x_{11} x_{12}; \\ f_4 &= x_5 x_6 x_9; \\ f_5 &= x_1 x_2 x_5 x_6 x_{10} x_{12}. \end{aligned} \right\} \quad (4.8)$$

Очевидно, что возможна минимизация схемы, реализующей эту систему функций, если предварительно построить какую-либо общую для некоторых из этих функций конъюнкцию, например $x_1 x_2$, в виде отдельного логического элемента, выход которого подать на входы схем, реализующих f_1, f_2, f_3, f_5 ; при этом число входов в логические элементы уменьшится на два.

В общем случае пусть задано множество функций $f_1, \dots, f_b, \dots, f_T$ от множества двоичных переменных $X = \{x_1, \dots, x_b, \dots, x_L\}$:

$$f_t = x_{t1} \Theta \dots \Theta x_{tr} \dots \Theta x_{tR_t}; \quad t = 1, \dots, T; \quad x_{ir} \in X.$$

Операция Θ есть некоторая логическая функция, подчиняющаяся ассоциативному закону, например дизъюнкция или конъюнкция, но обязательно только одна из них. Эти функции можно реализовать с помощью T логических элементов, каждый из которых имеет R_t входов. Обозначим через X_t ($t = 1, \dots, T$) множество переменных, поступающих на входы элемента t . Может случиться, что пересечение некоторых множеств из системы множеств $X_1, \dots, X_T$ не пусто. Пусть, например, это будут первые n множеств $X_1, \dots, X_n$:

$\bigcap_{t=1}^n X_t = Z \neq \emptyset$. Тогда множество функций можно представить в виде декомпозиции, такой, что $f_t = f(X'_t, f_z)$, где $X'_t = X_t \setminus Z$; $t = 1, \dots, n$.

Ясно, что если $\sum_{i=1}^n (P(X'_i) + 1) + P(Z) < \sum_{i=1}^T P(X_i)$, то полученная в

результате декомпозиции схема имеет меньшую цену. В последней формуле $P(X'_i)$, $P(Z)$ и $P(X_i)$ – число элементов в множествах X'_i , Z и X_i соответственно. Ставится задача нахождения среди множества эквивалентных схем, реализующих функции $f_1, \dots, f_T$, схемы с минимальной ценой, выполненной в виде декомпозиции однотипных элементов (т.е. реализующих одинаковые логические функции). Выигрыш от вынесения общих входов из n функций общей части Z , состоящей из m переменных, равен:

$$W(Z) = m(n-1) - n + r, \quad (4.9)$$

где r – число множеств из $X_1, \dots, X_T$, полностью совпадающих с Z , поскольку соответствующие им функции можно снять прямо с выхода схемы для общей части Z .

Вернёмся к функциям (4.8). Пересечения множеств букв, входящих в $X_1, \dots, X_5$, дают всевозможные наибольшие общие части, полученные попарным сравнением:

$X_1 = x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_{11}$	X_1			
$X_2 = x_1, x_2, x_3, x_8$	x_1, x_2, x_3	X_2		
$X_3 = x_1, x_2, x_5, x_6, x_{10}, x_{11}, x_{12}$	$x_1, x_2, x_5, x_6, x_{11}$	x_1, x_2	X_3	
$X_4 = x_5, x_6, x_9$	x_5, x_6	—	x_5, x_6	X_4
$X_5 = x_1, x_2, x_5, x_6, x_{10}, x_{12}$	x_1, x_2, x_5, x_6	x_1, x_2	$x_1, x_2, x_5, x_6, x_{10}, x_{12}$	x_5, x_6

В нашем примере это будут общие части:

$$\begin{aligned} Z_1 &= x_1, x_2, x_3 (X_1, X_2); & Z_2 &= x_1, x_2, x_5, x_6, x_{11} (X_1, X_3); \\ Z_3 &= x_5, x_6 (X_1, X_3, X_4, X_5); & Z_4 &= x_1, x_2, x_5, x_6 (X_1, X_3, X_5); \\ Z_5 &= x_1, x_2 (X_1, X_2, X_3, X_5); & Z_6 &= x_1, x_2, x_5, x_6, x_{10}, x_{12} (X_3, X_5). \end{aligned}$$

В скобках после общей части указаны функции, в которые она входит. По формуле (4.7) легко подсчитать цену каждой из полученных общих частей $Z_1, \dots, Z_6$ как выигрыш, который даёт вынесение этой общей части:

$$\begin{aligned} W(Z_1) &= 1; & W(Z_2) &= 3; & W(Z_3) &= 2; \\ W(Z_4) &= 5; & W(Z_5) &= 2; & W(Z_6) &= 5. \end{aligned}$$

В результате вынесения из множества $X_1, \dots, X_T$ некоторой общей части Z , входящей в первые n функций $X_1, \dots, X_n$, образуется новое множество по следующим правилам:

1. Если некоторое множество из $X_1, \dots, X_T$ совпадает с Z , то оно исключается из множества рассматриваемых функций:

2. В $X_1, \dots, X_n$ множество букв, входящих в Z , заменяется новой буквой x_p , не принадлежащей алфавиту $X = \{x_1, \dots, x_L\}$.

3. В множествах $X_1, \dots, X_T$ функции X_i ($i = 1, \dots, n$) преобразуются путём замены переменных: $X_{T+i} = (X_i \setminus Z) \cup \{x_p\}$.

4. К полученным в пунктах (1–3) функциям добавляется новая функция в виде вынесенной общей части Z .

Построение нового множества реализуемых функций иллюстрируется рис. 4.5.

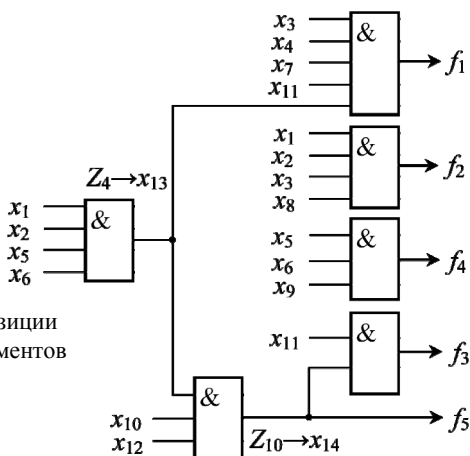


Рис. 4.5. Пример декомпозиции
схемы из однотипных элементов

После вынесения общей части с выбранной максимальной ценой выигрыша $Z_4 = x_1x_2x_5x_6$ из функций X_1 , X_3 и X_5 они преобразуются в X_6 , X_7 и X_8 соответственно, в которых вместо переменных x_1 , x_2 , x_5 , x_6 появляется новая переменная x_{13} . Из полученного преобразованного множества возможны дальнейшие вынесения. Найдём всевозможные общие части. Так как вынесение одной переменной смысла не имеет, ищем пересечения, имеющие не менее двух общих переменных:

$X_2 = x_1, x_2, x_3, x_8$	X_2				
$X_4 = x_5, x_6, x_9$	—	X_4			
$X_6 = x_3, x_4, x_7, x_{11}, x_{13}$	—	—	X_6		
$X_7 = x_{10}, x_{11}, x_{12}, x_{13}$	—	—	x_{11}, x_{13}	X_7	
$X_8 = x_{10}, x_{12}, x_{13}$	—	—	—	x_{10}, x_{12}, x_{13}	X_8
$Z_4 = x_1, x_2, x_5, x_6$	x_1, x_2	x_5, x_6	—	—	—

$$Z_7 = x_1, x_2 (X_2, Z_4); \quad W(Z_7) = 0;$$

$$Z_8 = x_5, x_6 (X_4, Z_4); \quad W(Z_8) = 0;$$

$$Z_9 = x_{11}, x_{13} (X_6, X_7); \quad W(Z_9) = 0;$$

$$Z_{10} = x_{10}, x_{12}, x_{13} (X_7, X_8); \quad W(Z_{10}) = 2.$$

После вынесения общей части $Z_{10} = x_{10}x_{12}x_{13}$ получаем новое множество функций, у которых нет пересечений, содержащих более одной переменной.

Кроме декомпозиции схемы для элементов И, эта операция может быть использована при построении схем ИЛИ, например, для выходных сигналов. После построения всех переходов в каждое состояние одни и те же микрооперации могут выдаваться на различных переходах с различных схем. Одинаковые микрооперации должны быть собраны на схему «ИЛИ», с которой и снимается соответствующий выходной сигнал. Это же относится и к сборке сигналов функций возбуждения триггерных устройств.

Контрольные вопросы и упражнения

1. Перечислите правила отметки состояний автомата Мили на граф-схеме алгоритма.
2. Дайте определение пути перехода автомата с использованием ГСА.
3. Опишите типы возможных путей перехода.
4. Поясните понятия прямой таблицы переходов и обратной таблицы переходов.
5. Составьте граф-схему алгоритма умножения двоичных чисел с фиксированной запятой после знакового разряда и отметьте на ней состояния автомата Мили.
6. Для полученной ГСА умножения составьте обратную структурную таблицу переходов.

Глава 5. ЦИФРОВЫЕ АВТОМАТЫ НА ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ МАТРИЦАХ

5.1. Программируемые логические структуры

Основой программируемых логических матриц (ПЛМ) (PLA, *Programmable Logic Array*), программируемой матричной логики (PAL, *Programmable Array Logic*) служат последовательно включённые программируемые матрицы элементов И и ИЛИ. Программируемые логические структуры типа PLA и PAL в английской терминологии объединяются термином SPLD (*Simple Programmable Logic Devices* – простые программируемые логические устройства) или короче PLD (*Programmable Logic Devices*). Разработка интегральных схем с программируемой и репрограммируемой структурой оказалась чрезвычайно перспективным направлением и привела к созданию новых эффективных средств разработки цифровых устройств и систем, таких как CPLD (*Complex PLD*), FPGA (*Field Programmable Gate Array*).

Трактовка программируемых логических структур в виде сочетания матриц И и ИЛИ обязана своим происхождением раннему этапу их развития. Позднее стали встречаться матрицы других логических элементов, причём возникло несколько вариантов. Однако каноническое представление через матрицы И и ИЛИ остаётся самым удобным, поскольку схемы, реализованные в булевом базисе, наиболее наглядны и легко понимаются. Общность изложения при этом не страдает, так как, в конечном счёте, на логическом этапе синтеза функциональные возможности разных схемных вариантов оказываются идентичными.

Основными параметрами ПЛМ являются: число входов n , число термов h и число выходов m . Для пояснения упрощённая логическая структура ПЛМ (n, h, m) на вентильном уровне показана на рис. 5.1. Крестики в пересечениях горизонтальных и вертикальных линий обозначают программируемые точки связей. В зависимости от характера применяемых элементов связи возможны различные варианты программирования. Обычно незапрограммированная ПЛМ имеет соединения во всех пересечениях, а при её программи-

ровании часть соединений удаляется, например, пережиганием перемычек.

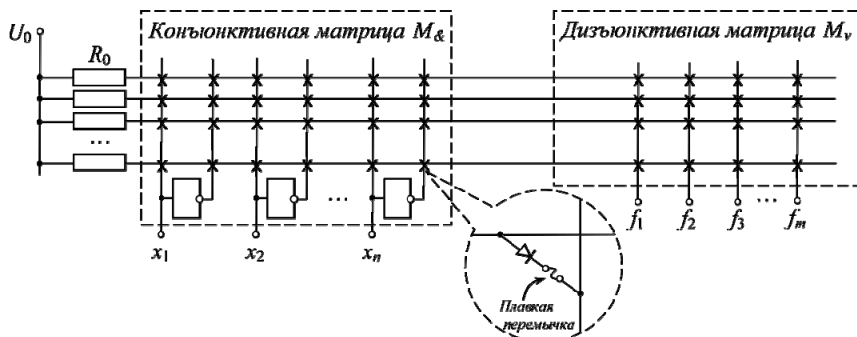


Рис. 5.1. Упрощённая схема ПЛМ

Переменные $x_1, \dots, x_n$ подаются на входы элементов И (конъюнкторов) и в матрице конъюнкций $M_\&$ образуются h термов. Под термом понимается конъюнкция входных переменных, взятых с отрицанием или без него. Число формируемых термов равно числу конъюнкторов (числу выходов матрицы $M_\&$). Термы подаются далее на входы дизъюнктивной матрицы M_v (ИЛИ), т.е. на входы дизъюнкторов, формирующих выходные функции. Число дизъюнкторов равно числу вырабатываемых функций m . Таким образом, ПЛМ реализует дизъюнктивную нормальную форму воспроизводимой функции.

5.2. Тривиальная реализация микропрограммного автомата

Микропрограммный автомат (МПА) может быть тривиально реализован на одной ПЛМ (n, h, m), если выполняются условия: $(P+R) \leq n$, $(K+R) \leq m$, $B \leq h$, где P , K и R – число соответственно входных сигналов, выходных управляющих сигналов и элементов памяти МПА; B – число строк в структурной таблице переходов МПА. При этом в матрице $M_\&$ ПЛМ (n, h, m) формируется множество элементарных конъюнкций переменных $Q_1, \dots, Q_R, x_1, \dots, x_P$, соответствующих строкам структурной таблицы переходов МПА, а

матрицей M_v реализуется множество функций выходных управляющих сигналов и функций возбуждения памяти автомата. Так, если в некоторой строке t структурной таблицы записан переход из состояния a_m в состояние a_s под действием входного сигнала $X(a_m, a_s)$, то этому переходу соответствует промежуточная шина ПЛИМ (n, h, m) , на которой реализована конъюнкция $e_t = \lambda_m X(a_m, a_s)$, где λ_m – конъюнкция, соответствующая коду $K(a_m)$ состояния a_m . Пусть в структурной таблице переходов микрооперация y_i записана в строках с номерами $b_1, \dots, b_U$, а функция возбуждения D_r – в строках с номерами $r_1, \dots, r_Q$, тогда $y_i = e_{n1} \vee \dots \vee e_{nU}$; $D_r = e_{r1} \vee \dots \vee e_{rQ}$.

Схема МПА при его тривиальной реализации на ПЛИМ (n, h, m) показана на рис. 5.2. Память автомата построена на R -разрядном регистре RG из D -триггеров, комбинационная логическая часть – на ПЛИМ (n, h, m) . Все дальнейшие рассуждения будут иллюстрироваться на примере автомата Мили, заданного прямой структурной таблицей переходов (табл. 5.1).

Вопросы минимизации цены схемной реализации автомата на матричной логике и подбора параметров ПЛИМ (n, h, m) исходя из соотношений, указанных в самом начале подраздела 5.2, определяются, в основном, площадью матриц $M_\&$ и M_v :

$$\begin{cases} S(M_\&) = 2(P + R)B; \\ S(M_v) = (K + R)B. \end{cases}$$

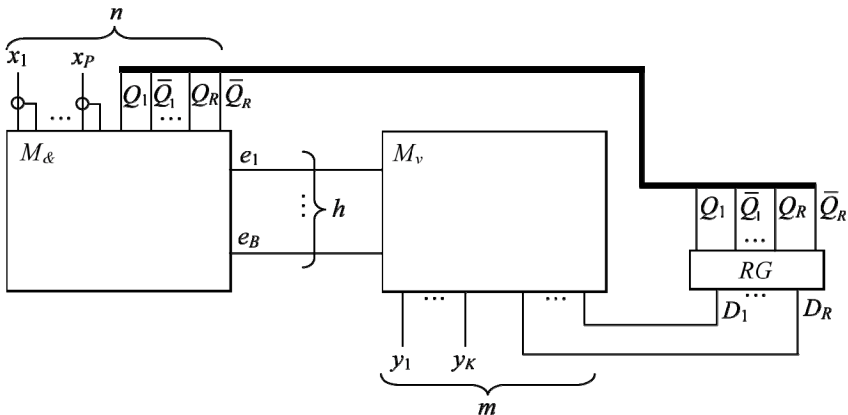


Рис. 5.2. Тривиальная реализация автомата на ПЛИМ

Таблица 5.1

Прямая структурная таблица автомата Мили

a_m	$K(a_m)$	a_s	$K(a_s)$	$X(a_m, a_s)$	$Y(a_m, a_s)$ Y_t	$\Psi(a_m, a_s)$	B
1	2	3	4	5	6	7	8
a_1	0000	a_1	0000	$x_1 \bar{x}_2$	$y_1 y_2$ Y_1	-	1
		a_5	0001	$\bar{x}_1$	$y_1 y_2 Y_{12}$ Y_2	D_4	2
		a_6	1010	$x_1 x_2 \bar{x}_3$	$y_2 Y_{12}$ Y_7	$D_1 D_3$	3
		a_3	1110	$x_1 x_2 x_3$	$y_1 y_2$ Y_1	$D_1 D_2 D_3$	4
a_2	0111	a_2	0111	x_6	- Y_0	$D_2 D_3 D_4$	5
		a_3	1110	$\bar{x}_6$	$y_3 Y_5$ Y_4	$D_1 D_2 D_3$	6
a_3	011	a_3	1110	x_{10}	$y_3 Y_5$ Y_4	$D_1 D_2 D_3$	7
		a_8	0101	$\bar{x}_4 \bar{x}_{10}$	$y_5 y_8 Y_9$ Y_6	$D_2 D_4$	8
		a_9	0110	$x_4 \bar{x}_{10}$	$y_{10} Y_{15}$ Y_{10}	$D_2 D_3$	9
a_4	1101	a_4	1101	$\bar{x}_7 x_9$	$y_{13} Y_{18}$ Y_5	$D_1 D_2 D_4$	10
		a_6	1010	x_7	y_{13} Y_3	$D_1 D_3$	11
		a_8	0101	$\bar{x}_7 \bar{x}_9$	$y_{13} Y_{14}$ Y_{13}	$D_2 D_4$	12
a_5	0001	a_5	0001	$\bar{x}_1$	$y_7 Y_{11}$ Y_8	D_4	13
		a_6	1010	x_1	$y_{16} Y_{17}$ Y_{12}	$D_1 D_3$	14
a_6	1010	a_1	0000	x_5	$y_1 y_2$ Y_1	-	15
		a_1	0000	$\bar{x}_5$	$y_{16} Y_{17} Y_{12}$	-	16

a_m	$K(a_m)$	a_s	$K(a_s)$	$X(a_m, a_s)$	$Y(a_m, a_s)$	$\Psi(a_m, a_s)$	B
a_7	0010	a_2	0111	x_8	$y_{14} Y_{18}$ Y_{15}	$D_2 D_3 D_4$	17
		a_4	1101	$\bar{x}_8$	$y_{13} Y_{18}$ Y_5	$D_1 D_2 D_4$	18
a_8	0101	a_4	1101	$\bar{x}_9$	y_6 Y_9	$D_1 D_2 D_4$	19
		a_7	0010	x_9	$y_4 Y_6$ Y_{11}	D_3	20
a_9	0110	a_2	0111	$\bar{x}_6 x_{11}$	$y_5 y_8 Y_9$ Y_6	$D_2 D_3 D_4$	21
		a_3	1110	$\bar{x}_{11}$	$y_3 y_8 Y_9$ Y_{14}	$D_1 D_2 D_3$	22
		a_9	0110	$x_6 x_{11}$	$y_{10} Y_{15}$ Y_{10}	$D_2 D_3$	23

Как видно из рис. 5.2, параметры ПЛМ накладывают ограничения на объекты описания цифрового автомата при его тривиальной реализации. Например, для автомата, структурная таблица переходов которого показана в табл. 5.1:

$$S(M_{\&}) = 2(11+4)23 = 690;$$

$$S(M_v) = (18+4)23 = 506.$$

Уменьшить площадь матриц за счёт количества триггерных устройств (R) регистра памяти состояний в явном виде не удаётся, так как длина кода состояний уже выбирается исходя из условия минимальности, а количество переходов (число строк структурной таблицы) определяется только алгоритмом функционирования автомата.

Минимизировать суммарную площадь матричной реализации автомата позволяют три практические задачи:

- задача замены входных переменных ($\min P$);
- задача кодирования микрокоманд ($\min K$);
- задача декомпозиции автомата в виде сети связанных между собой компонентных автоматов.

5.3. Замена входных переменных

Замена структурных входных сигналов автомата $\{x_1, \dots, x_P\}$ в конъюнктивной матрице $M_\&$ может привести к желаемому положительному результату, если мощность множества новых фиктивных переменных $\{\lambda_1, \dots, \lambda_E\}$ намного меньше мощности реальных структурных входных сигналов. Мощность множества новых входных переменных E определяется максимальным количеством входных сигналов, от которых *существенно* зависит переход из какого-либо состояния a_m . Например, по табл. 5.1 скомпоновав табл. 5.2, можно увидеть, что мощность множества новых входных переменных $E = 3 \rightarrow \{\lambda_1, \lambda_2, \lambda_3\}$.

Таблица 5.2

**Таблица зависимости
переменных от состояния**

$a_\tau \rightarrow E_\tau$	$X(a_\tau)$
$a_1 \rightarrow E_1$	x_1, x_2, x_3
$a_2 \rightarrow E_2$	x_6
$a_3 \rightarrow E_3$	x_4, x_{10}
$a_4 \rightarrow E_4$	x_7, x_9
$a_5 \rightarrow E_5$	x_1
$a_6 \rightarrow E_6$	x_5
$a_7 \rightarrow E_7$	x_8
$a_8 \rightarrow E_8$	x_9
$a_9 \rightarrow E_9$	x_6, x_{11}

Таблица 5.3

Таблица замены переменных

$a_\tau \rightarrow \lambda_e$	λ_1	λ_2	λ_3
a_1	x_1	x_2	x_3
a_2	x_6	-	-
a_3	x_4	x_{10}	-
a_4	-	x_7	x_9
a_5	x_1	-	-
a_6	-	x_5	-
a_7	-	-	x_8
a_8	-	-	x_9
a_9	x_6	-	x_{11}

Замена структурных входных сигналов новыми переменными производится дизъюнктивной сборкой конъюнкции кода состояния, из которого происходит переход, и реального входного сигнала по составленной таблице замены (табл. 5.3):

$$\lambda_e = \bigvee_{\forall i} K(a_m) x_i = \bigvee_{\forall i} Z_i.$$

Составление таблицы замены можно делать произвольным образом, опираясь на интуицию и некоторые эвристические правила, например:

- один и тот же входной сигнал на переходах из разных состояний необходимо заменить на единственную новую переменную;

• плотность замены переменных по столбцам желательно иметь равномерной.

Для удобства дальнейшего изложения код состояния $K(a_m)$, из которого происходит переход заменим символом этого состояния. Тогда для нашего примера выражения для замены переменных будут выглядеть следующим образом:

$$\begin{aligned}\lambda_1 &= x_1(a_1 \vee a_5 \vee [a_4 \vee a_6 \vee a_7 \vee a_8]) \vee x_6(a_2 \vee a_9 \vee [\dots]) \vee x_4(a_3 \vee [\dots]); \\ \lambda_2 &= x_2(a_1 \vee [a_2 \vee a_5 \vee a_7 \vee a_8 \vee a_9]) \vee x_{10}(a_3 \vee [\dots]) \vee x_7(a_4 \vee [\dots]) \vee x_5(a_6 \vee [\dots]); \\ \lambda_3 &= x_3(a_1 \vee [a_2 \vee a_3 \vee a_5 \vee a_6]) \vee x_9(a_4 \vee a_8 \vee [\dots]) \vee x_8(a_7 \vee [\dots]) \vee x_{11}(a_9 \vee [\dots]).\end{aligned}$$

Здесь в квадратных скобках при замене переменных указаны состояния, при переходе из которых замена не определена для переменной x_i и может быть использована для минимизации термов выражения $\lambda_e = \bigvee_{\forall i} K(a_m) x_i = \bigvee_{\forall i} Z_i$ по кодам состояний. Коды состояний автомата из табл. 5.1 сведём в карту Карно для минимизации λ_e (табл. 5.4). При проведении процедуры кодирования, естественно, необходимо учитывать, что состояния, переходы из которых существенно зависят от одной и той же переменной, должны кодироваться, по возможности, соседними кодами. Так, для нашего примера соседними кодами закодированы состояния $(a_1 \text{ и } a_5)$, $(a_2 \text{ и } a_9)$, $(a_4 \text{ и } a_8)$.

Процедура получения выражений для замены каждой структурной переменной показана на рис. 5.3-5.15). В таблицах состояния, переход из которых существенно зависит от данной переменной, обозначены буквой с индексом, а состояния, переход из которых не зависит от данной переменной (не определён), обозначены просто цифровым индексом.

Результирующие выражения для новых переменных тогда будут выглядеть следующим образом:

$$\begin{aligned}\lambda_1 &= x_1 \bar{Q}_3 \vee x_4 Q_1 \vee x_6 \bar{Q}_1 Q_2; \\ \lambda_2 &= x_2 \bar{Q}_1 \vee x_5 \bar{Q}_2 Q_3 \vee x_7 Q_4 \vee x_{10} Q_2 Q_3; \\ \lambda_3 &= x_3 \bar{Q}_2 \bar{Q}_3 \vee x_8 \bar{Q}_2 Q_3 \vee x_9 Q_4 \vee x_{11} Q_2 Q_3.\end{aligned}$$

Таблица 5.4

Коды состояний автомата

$Q_3 Q_4$					
$Q_1 Q_2$	00	01	11	10	
00	a_1	a_5		a_7	
01		a_8	a_2	a_9	
11		a_4		a_3	
10				a_6	

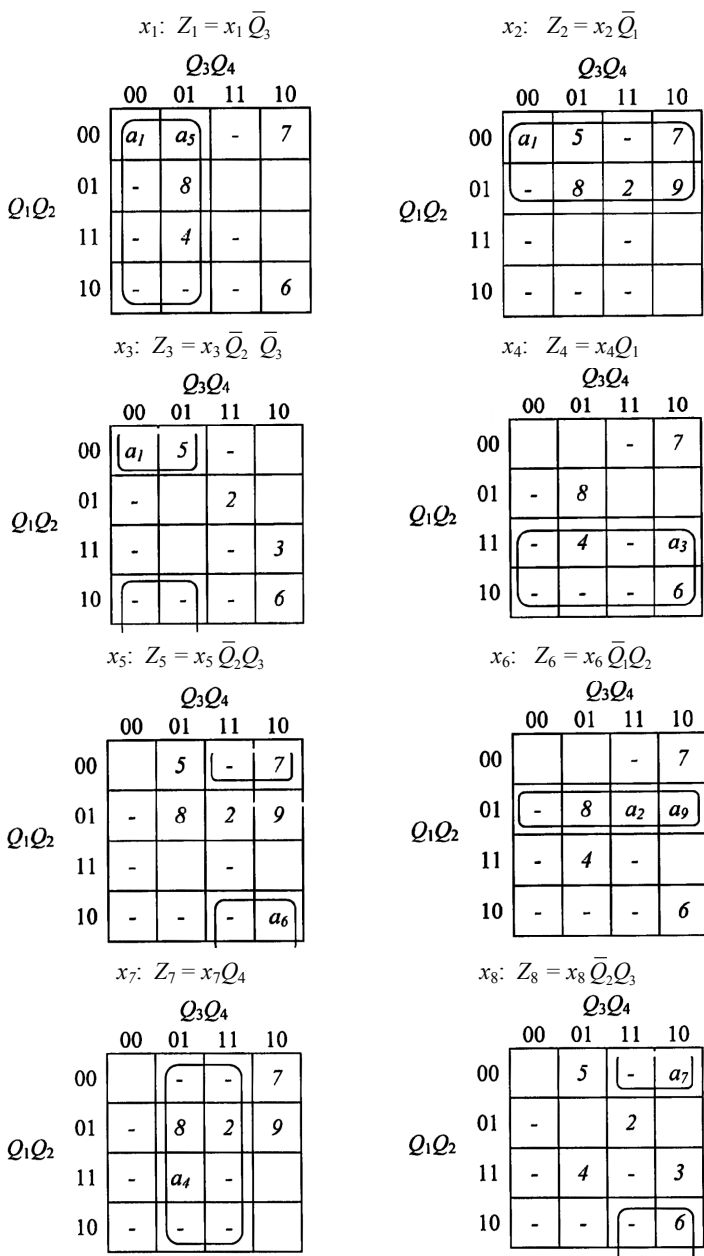


Рис. 5.3. Карты Карно для переменных $x_1 - x_8$

		$x_9: Z_9 = x_9 Q_4$						$x_{10}: Z_{10} = x_{10} Q_2 Q_3$						$x_{11}: Z_{11} = x_{11} Q_2 Q_3$			
		$Q_3 Q_4$						$Q_3 Q_4$						$Q_3 Q_4$			
		00	01	11	10			00	01	11	10			00	01	11	10
$Q_1 Q_2$	00		5	-		$Q_1 Q_2$	00		5	-	7	$Q_1 Q_2$	00		5	-	
	01	-	a_8	2			01	-	8	2	9		01	-		2	a_9
	11	-	a_4	-			11	-		-	a_3		11	-		-	3
	10	-	-	-	6		10	-	-	-			10	-	-	-	6

Рис. 5.4. Карты Карно для переменных $x_9 - x_{11}$

Схема реализации автомата с заменой входных переменных для рассматриваемого примера приведена на рис. 5.5.

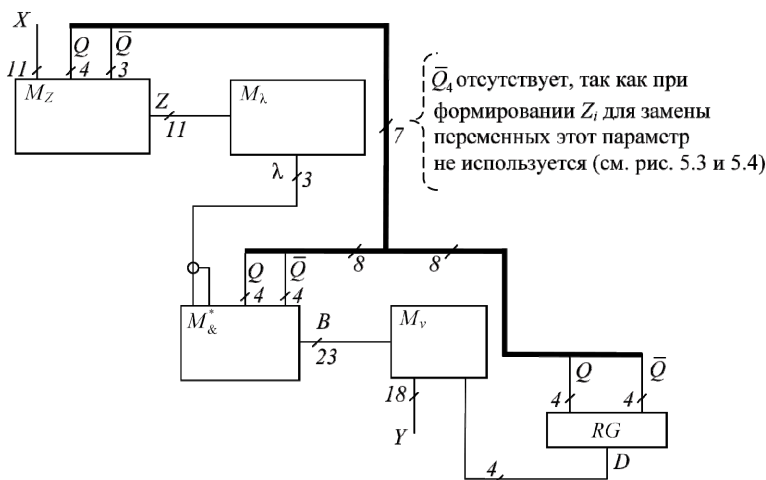


Рис. 5.5. Реализация автомата при замене входных переменных

Конъюнктивная матрица M_* тривиальной реализации автомата на схеме трансформируется и реализуется в виде трёх компонентных структур: M_Z – конъюнктивной матрицы для получения термов Z_i формирования новых входных переменных; $M_λ$ – дизъюнктивной матрицы формирования новых входных переменных $λ_e$; M_* – конъюнктивной матрицы для получения термов B формирования строк структурной таблицы переходов: $M_* \rightarrow \{M_Z, M_λ, M_*^*\}$.

До замены структурных входных переменных $S(M_{\&}) = 690$.

После замены: $S(M_Z) = 198$; $S(M_{\lambda}) = 33$; $S(M_{\&}^*) = 322$.

Суммарная площадь матриц: $S\{M_Z, M_{\lambda}, M_{\&}^*\} = 553$.

Положительная разница площадей матриц после замены структурных входных сигналов автомата на новые переменные составила $\delta(S) = 137$ единиц $= 19,85\%$.

5.4. Кодирование микрокоманд

Минимизация площади матрицы выходных управляющих сигналов и функций возбуждения возможна путём кодирования не отдельных управляющих сигналов, инициализирующих выполнение микроопераций, а микрокоманд $Y_t = \{y_{t1}, \dots, y_{tu}, \dots, y_{tU}\}$ в виде множества выходных сигналов, записанных в каждой строке структурной таблицы переходов: $Y_t (t = 1, \dots, T) \rightarrow K(Y_t)$.

Разрядность кода определяется количеством микрокоманд T , подлежащих кодированию (включая пустую микрокоманду с нулевой кодовой комбинацией): $c = \lceil \log_2 T \rceil \rightarrow (g_1, g_2, \dots, g_c)$. Но количество микрокоманд, подлежащих кодированию, нуждается в некотором анализе.

Во-первых, находятся выходные управляющие сигналы, которые встречаются только в одной строке структурной таблицы переходов и, естественно, не нуждаются в дальнейшей дизъюнктивной сборке матрицей M_v . Такие выходные сигналы можно непосредственно снимать с конъюнктивной матрицы реализации переходов $M_{\&}$ (или $M_{\&}^*$ при замене входных переменных). Например, для рассматриваемого примера (см. табл. 5.1) $Y^* = \{y_4, y_7, y_{11}\}$.

Во-вторых, если некоторая микрокоманда $Y_t \in Y^*$, то она в кодировании микрокоманд не участвует. Из табл. 5.1 видно, что в 13-й строке $Y_8 \in Y^* = \{y_4, y_7, y_{11}\}$ и в кодировании микрокоманд принимать участие не будет. Разрядность кода для рассматриваемого примера: $c = \lceil \log_2 (T^* = 15) \rceil = 4$.

При проведении процедуры кодирования, естественно, необходимо учитывать, что микрокоманды, содержащие одинаковые выходные управляющие сигналы, должны кодироваться, по возможности, соседними кодами. Например, для нашего примера соседними кодами должны быть закодированы следующие группы микрокоманд: (Y_1, Y_2) ; (Y_1, Y_2, Y_7) ; (Y_4, Y_{14}) ; (Y_4, Y_6) ; (Y_9, Y_{11}) ; (Y_6, Y_{14}) ; (Y_2, Y_7) ; (Y_3, Y_5, Y_{13}) ; (Y_{13}, Y_{15}) ; (Y_5, Y_{15}) . Вариант кодировки показан в табл. 5.5, а выражения для реализации выходных управляющих сигналов сведены в табл. 5.6, где символом E_i обозначены термы для их получения.

Таблица 5.5

Коды микрокоманд

$g_1 g_2$	$g_3 g_4$			
	00	01	11	10
00	Y_0	Y_2	Y_1	Y_{13}
01	Y_6	Y_7	-	Y_{14}
11	Y_4	Y_{10}	Y_3	Y_5
10	Y_{12}	Y_9	Y_{11}	Y_{15}

Таблица 5.6

Выходные управляющие сигналы

$y_1 = Y_1 \vee Y_2 = \bar{g}_1 \bar{g}_2 g_4 = E_1$	$y_{10} = Y_{10} = g_1 g_2 \bar{g}_3 g_4 = E_8$
$y_2 = Y_1 \vee Y_2 \vee Y_7 = \bar{g}_1 g_4 = E_2$	$y_{11} \rightarrow \text{с матрицы } M_{\&} \text{ (13-я линия)}$
$y_3 = Y_4 \vee Y_{14} = g_1 g_2 \bar{g}_3 \bar{g}_4 \vee \bar{g}_1 g_2 g_3 \bar{g}_4 = E_3 \vee E_4$	$y_{12} = Y_2 \vee Y_7 = \bar{g}_1 \bar{g}_3 g_4 = E_9$
$y_4 \rightarrow \text{с матрицы } M_{\&} \text{ (20-я линия)}$	$y_{13} = Y_3 \vee Y_5 \vee Y_{13} = g_1 g_2 g_3 \vee \bar{g}_1 \bar{g}_2 g_3 \bar{g}_4 = E_{10} \vee E_{11}$
$y_5 = Y_4 \vee Y_6 = g_2 \bar{g}_3 \bar{g}_4 = E_5$	$y_{14} = Y_{13} \vee Y_{15} = \bar{g}_2 g_3 \bar{g}_4 = E_{12}$
$y_6 = Y_9 \vee Y_{11} = g_1 \bar{g}_2 g_4 = E_6$	$y_{15} = y_{10} = E_8$
$y_7 \rightarrow \text{с матрицы } M_{\&} \text{ (13-я линия)}$	$y_{16} = Y_{12} = g_1 \bar{g}_2 \bar{g}_3 \bar{g}_4 = E_{13}$
$y_8 = Y_6 \vee Y_{14} = \bar{g}_1 g_2 \bar{g}_4 = E_7$	$y_{17} = y_{16} = E_{13}$
$y_9 = y_8 = E_7$	$y_{18} = Y_5 \vee Y_{15} = g_1 g_3 \bar{g}_4 = E_{14}$

Из табл. 5.6 видно, что благодаря успешному кодированию микрокоманд дизъюнктивной сборке подлежат только два выходных управляющих сигнала: y_3 и y_{13} . Реализация автомата на матричных структурах с учётом кодирования микрокоманд

показана на рис. 5.6 (здесь: M_{GD} – матрица дизъюнктивной сборки функций возбуждения и единичных значений переменных g_i кода микрокоманды; M_E – конъюнктивная матрица термов E_i для формирования выходных управляющих сигналов; M_Y – матрица сборки выходных сигналов с двумя и более термами).

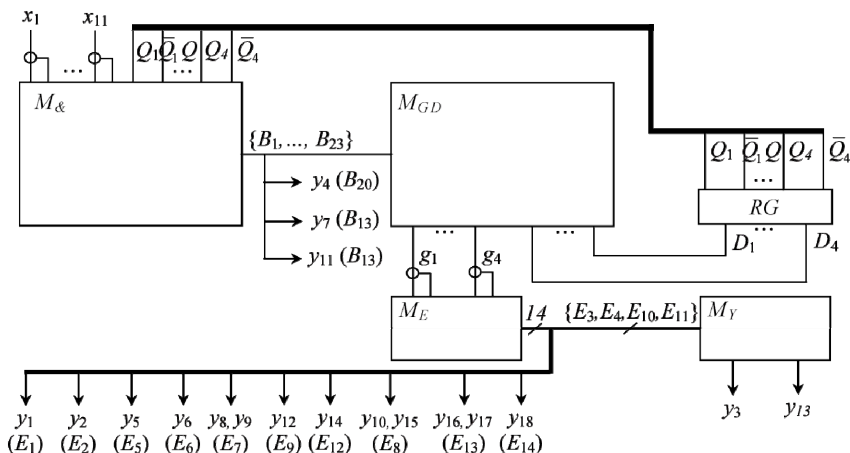


Рис. 5.6. Реализация автомата с кодированием микрокоманд

Дизъюнктивная матрица M_v тривиальной реализации автомата, как показано на схеме, трансформируется и реализуется в виде трёх компонентных структур: M_{GD} , M_E , M_Y .

До кодирования микрокоманд $S(M_v) = 506$.

После кодирования микрокоманд: $S(M_{GD}) = 184$; $S(M_E) = 112$; $S(M_Y) = 8$.

Суммарная площадь матриц: $S\{M_{GD}, M_E, M_Y\} = 304$.

Положительная разница площадей матриц после проведения процедуры кодирования микрокоманд автомата составила $\delta(S) = 202$ единицы = 39,92 %.

С учётом замены входных сигналов на новые переменные и проведением процедуры кодирования микрокоманд структурная таблица переходов автомата для рассматриваемого примера примет вид, показанный в табл. 5.7, а её реализация на матричных структурах – на рис. 5.7.

Таблица 5.7

Структурная таблица переходов автомата с заменой переменных
и кодированием микрокоманд

a_m	$K(a_m)$	a_s	$K(a_s)$	$\lambda(a_m, a_s)$	$Y(a_m, a_s)$	Y_t	$K(Y_t)$	$\Psi(a_m, a_s)$	B
a_1	0000	a_1	0000	$\lambda_1 \bar{\lambda}_2$	$y_1 y_2$	Y_1	$g_3 g_4$	-	1
		a_5	0001	$\bar{\lambda}_1$	$y_1 y_2 y_{12}$	Y_2	g_4	D_4	2
		a_6	1010	$\lambda_1 \lambda_2 \bar{\lambda}_3$	$y_2 y_{12}$	Y_7	$g_2 g_4$	$D_1 D_3$	3
		a_3	1110	$\lambda_1 \lambda_2 \lambda_3$	$y_1 y_2$	Y_1		$D_1 D_2 D_3$	4
a_2	0111	a_2	0111	λ_1	-	Y_0	-	$D_2 D_3 D_4$	5
		a_3	1110	$\bar{\lambda}_1$	$y_3 y_5$	Y_4	$g_1 g_2$	$D_1 D_2 D_3$	6
a_3	011	a_3	1110	λ_2	$y_3 y_5$	Y_4	$g_1 g_2$	$D_1 D_2 D_3$	7
		a_8	0101	$\bar{\lambda}_1 \bar{\lambda}_2$	$y_5 y_8 y_9$	Y_6	g_2	$D_2 D_4$	8
		a_9	0110	$\lambda_1 \bar{\lambda}_2$	$y_{10} y_{15}$	Y_{10}	$g_1 g_2 g_4$	$D_2 D_3$	9
a_4	1101	a_4	1101	$\bar{\lambda}_2 \lambda_3$	$y_{13} y_{18}$	Y_5	$g_1 g_2 g_3$	$D_1 D_2 D_4$	10
		a_6	1010	λ_2	y_{13}	Y_3	$g_1 g_2 g_3 g_4$	$D_1 D_3$	11
		a_8	0101	$\bar{\lambda}_2 \bar{\lambda}_3$	$y_{13} y_{14}$	Y_{13}	g_3	$D_2 D_4$	12

Окончание табл. 5.7

a_m	$K(a_m)$	a_s	$K(a_s)$	$\lambda(a_m, a_s)$	$Y(a_m, a_s)$	Y_t	$K(Y_t)$	$\Psi(a_m, a_s)$	B
a_5	0001	a_5	0001	$\bar{\lambda}_1$	$y_7 y_{11}$	Y_8	-	D_4	13
		a_6	1010	λ_1	$y_{16} y_{17}$	Y_{12}	g_1	$D_1 D_3$	14
a_6	1010	a_1	0000	λ_2	$y_1 y_2$	Y_1	$g_3 g_4$	-	15
		a_1	0000	$\bar{\lambda}_2$	$y_{16} y_{17}$	Y_{12}	g_1	-	16
a_7	0010	a_2	0111	λ_3	$y_{14} y_{18}$	Y_{15}	$g_1 g_3$	$D_2 D_3 D_4$	17
		a_4	1101	$\bar{\lambda}_3$	$y_{13} y_{18}$	Y_5	$g_1 g_2 g_3$	$D_1 D_2 D_4$	18
a_8	0101	a_4	1101	$\bar{\lambda}_3$	y_6	Y_9	$g_1 g_4$	$D_1 D_2 D_4$	19
		a_7	0010	λ_3	$y_4 y_6$	Y_{11}	$g_1 g_3 g_4$	D_3	20
a_9	0110	a_2	0111	$\bar{\lambda}_1 \lambda_3$	$y_5 y_8 y_9$	Y_6	g_2	$D_2 D_3 D_4$	21
		a_3	1110	$\bar{\lambda}_3$	$y_3 y_8 y_9$	Y_{14}	$g_2 g_3$	$D_1 D_2 D_3$	22
		a_9	0110	$\lambda_1 \lambda_3$	$y_{10} y_{15}$	Y_{10}	$g_1 g_2 g_4$	$D_2 D_3$	23

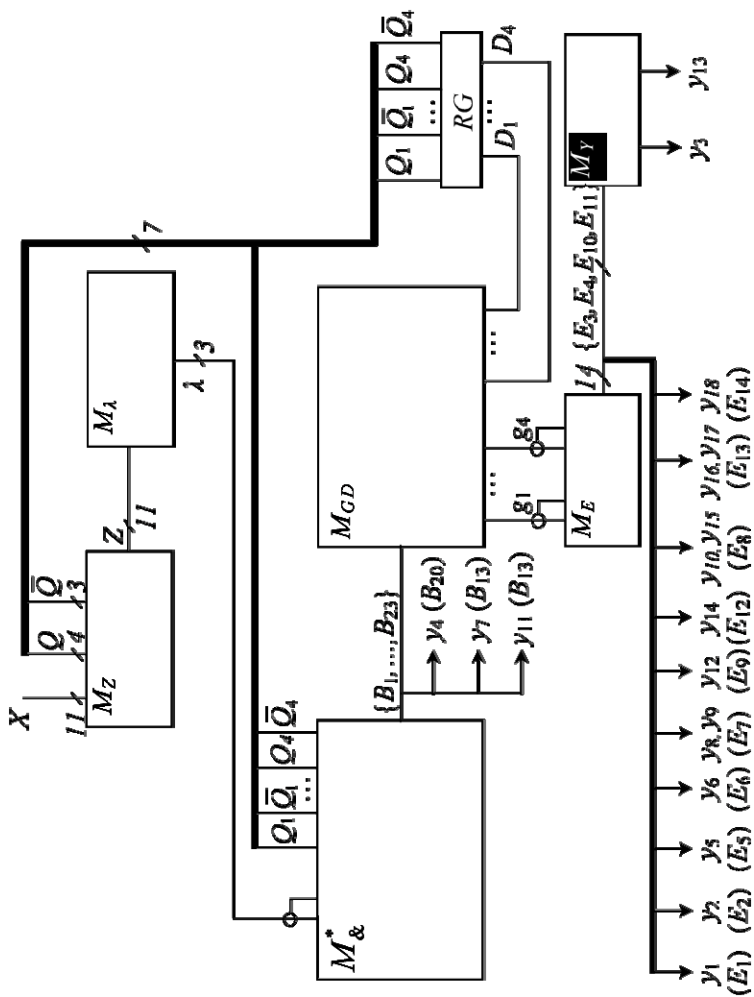


Рис. 5.7. Реализация автомата на матричных структурах по табл. 5.7

5.5. Декомпозиция автомата

Под декомпозицией автомата принято понимать построение для заданного автомата S сети C , состоящей из связанных между собой компонентных автоматов. Каждой сети C можно сопоставить её результирующий автомат S_C . Сеть C будет результатом декомпозиции автомата S , если у S_C есть подавтомат, изоморфный S . Ясно, что вследствие неоднозначности решения задачи декомпозиции любой автомат S может быть реализован различными сетями, и к этим сетям часто предъявляют определённые требования, например:

- число состояний в каждом компонентном автомате не должно превышать число состояний в исходном автомате ($|A^i| \leq |A|$);
- $|A^i||X^i| \leq |A||X|$, здесь X^i и X – множества входных сигналов в компонентном S^i и исходном S автоматах;
- в сети должно быть минимальное число связей между компонентными автоматами;
- сеть должна быть построена из компонентных автоматов определённого типа;
- в результате декомпозиции должна получиться сеть из последовательно работающих компонентных автоматов и другие требования исходя из условий применения.

Декомпозицию микропрограммного автомата Мили проиллюстрируем примером автомата, заданного структурной таблицей переходов (см. табл. 5.1).

Пусть на множестве A состояний декомпозируемого автомата $S = (A, X, Y, \delta, \lambda, a_1)$ определено разбиение $\pi = \{A^1, \dots, A^n\}$. На выборе такого разбиения остановимся подробнее после рассмотрения процедуры декомпозиции. В данном примере положим $\pi = \{A^1, A^2, A^3\}$: $A^1 = \{a_1, a_5, a_6\}$, $A^2 = \{a_2, a_3, a_9\}$, $A^3 = \{a_4, a_7, a_8\}$. Как принято, запишем это разбиение следующим образом:

$$\pi = \overline{1.5.6}; \overline{2.3.9}; \overline{4.7.8}.$$

Паре (S, π) поставим в соответствие сеть C из n компонентных автоматов Мили $S^m = (B^m, X^m, Y^m, \delta^m, \lambda^m, a_1^m)$; $m = 1, \dots, n$; $a_1^m \in B^m$.

Объекты описания компонентного автомата S^m :

1. $B^m = A^m \cup \{b_m\}$, где A^m – блок π -разбиения, а b_m – дополнительное состояние в автомате S^m (такое состояние, играющее роль состояния ожидания при взаимодействии процессов функционирования сети C , есть в каждом компонентном автомате). Очевидно, что

$$\sum_{m=1}^n |b_m| - n = |A|. \text{ В рассматриваемом примере:}$$

$$B^1 = A^1 \cup \{b_1\} = \{b_1, a_1, a_5, a_6\};$$

$$B^2 = A^2 \cup \{b_2\} = \{b_2, a_2, a_3, a_9\};$$

$$B^3 = A^3 \cup \{b_3\} = \{b_3, a_4, a_7, a_8\}.$$

$$2. X^m = \bigcup_{a_i \in A^m} X(a_i) \cup Z_x^m = \tilde{X}^m \cup Z_x^m, \text{ где } X(a_i) - \text{множество входных}$$

переменных, встречающихся в столбце $X(a_m, a_s)$ структурной таблицы автомата S на всех переходах из состояния $a_i \in A^m$; Z_x^m – множество дополнительных входных переменных автомата S^m для связи с другими компонентными автоматами. Таким образом, $\tilde{X}^m$ равно множеству входных переменных, встречающихся на всех переходах из подмножества A^m состояний исходного автомата S . В рассматриваемом примере:

$$\tilde{X}^1 = \{x_1, x_2, x_3, x_5\};$$

$$\tilde{X}^2 = \{x_4, x_6, x_{10}, x_{11}\};$$

$$\tilde{X}^3 = \{x_7, x_8, x_9\}.$$

$$3. Y^m = \bigcup_{a_i \in A^m} Y(a_i) \cup Z_y^m = \tilde{Y}^m \cup Z_y^m, \text{ где } Y(a_i) - \text{множество выходных}$$

сигналов, встречающихся в столбце $Y(a_m, a_s)$ структурной таблицы автомата S на всех переходах из состояния $a_i \in A^m$; Z_y^m – множество дополнительных выходных сигналов автомата S^m для связи с другими компонентными автоматами. В рассматриваемом примере:

$$\tilde{Y}^1 = \{y_1, y_2, y_7, y_{11}, y_{12}, y_{16}, y_{17}\};$$

$$\tilde{Y}^2 = \{y_3, y_5, y_8, y_9, y_{10}, y_{15}\};$$

$$\tilde{Y}^3 = \{y_4, y_6, y_{13}, y_{14}, y_{18}\}.$$

4. Определим функции переходов δ^m и функции выходов λ^m . Пусть в исходном автомате S есть переход из a_i в a_j под действием входного сигнала X_h с выдачей выходного сигнала Y_l : $\delta(a_i, x_h) = a_j$;

$\lambda(a_i, x_h) = Y_i$; и пусть $a_i \in A^m$, то есть a_i – состояние компонентного автомата S^m . При функционировании сети C возможны две ситуации.

Ситуация 1. $a_j \in A^m$, то есть a_j – состояние того же самого компонентного автомата S^m . Тогда $\delta^m(a_i, x_h) = \delta(a_i, x_h) = a_j$; $\lambda^m(a_i, x_h) = \lambda(a_i, x_h) = Y_i$, и соответствующий этому переходу подграф в автомате S^m совпадает с подграфом исходного автомата S (рис. 5.8, а).

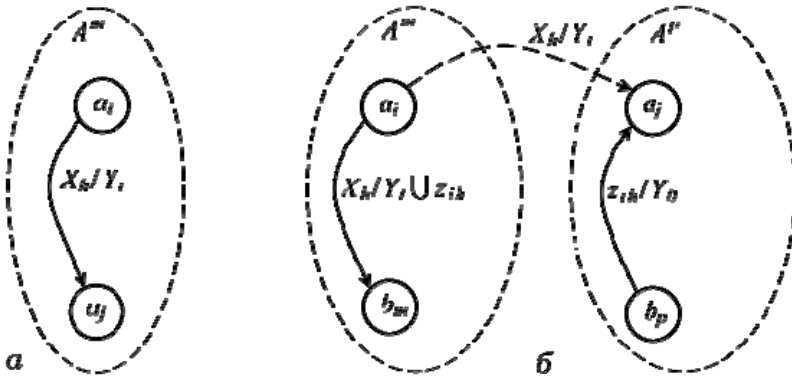


Рис. 5.8. Функции переходов и выходов компонентных автоматов

Ситуация 2. $a_j \in A^p$, то есть a_j – состояние компонентного автомата S^p ($p \neq m$). Таким образом, состояния a_i и a_j принадлежат разным компонентным автоматам. Будем говорить, что в этом случае компонентный автомат S^m возбуждает компонентный автомат S^p , а сам переходит в режим ожидания b_m . Тогда в автомате S^m функция переходов и выходов определяется как

$$\delta^m(a_i, x_h) = b_m; \quad \lambda^m(a_i, x_h) = Y_i \cup z_{ih},$$

где z_{ih} – дополнительная микрооперация, вырабатываемая в компонентном автомате S^m и являющаяся входным сигналом для компонентного автомата S^p . В этой ситуации функция переходов и выходов автомата S^p будет определена, как показано на рис. 5.8, б: $\delta^p(b_p, z_{ih}) = \delta(a_i, x_h) = a_j$; $\lambda^p(b_p, z_{ih}) = Y_0$, где Y_0 – пустая микрокоманда, которой соответствует векторный выходной сигнал со всеми нулевыми компонентами.

Таблицы переходов без структуризации кодов состояний компонентных автоматов S^1, S^2, S^3 для рассматриваемого примера приведены в табл. 5.8–5.10 соответственно.

Таблица 5.8

Таблица переходов компонентного автомата S^1

a_m	a_s	$X(a_m, a_s)$	$Y(a_m, a_s)$	Y_t	B
a_1	a_1	$x_1 \bar{x}_2$	$y_1 y_2$	Y_1	1
	a_5	$\bar{x}_1$	$y_1 y_2 y_{12}$	Y_2	2
	a_6	$x_1 x_2 \bar{x}_3$	$y_2 y_{12}$	Y_7	3
	b_1	$x_1 x_2 x_3$	$y_1 y_2 z_1$	$Y_1 \rightarrow Y_{16}$	4
a_5	a_5	$\bar{x}_1$	$y_7 y_{11}$	Y_8	5
	a_6	x_1	$y_{16} y_{17}$	Y_{12}	6
a_6	a_1	x_5	$y_1 y_2$	Y_1	7
	a_1	$\bar{x}_5$	$y_{16} y_{17}$	Y_{12}	8
b_1	a_6	z_3	-	Y_0	9

Таблица 5.9

Таблица переходов компонентного автомата S^2

a_m	a_s	$X(a_m, a_s)$	$Y(a_m, a_s)$	Y_t	B
a_2	a_2	x_6	-	Y_0	1
	a_3	$\bar{x}_6$	$y_3 y_5$	Y_4	2
a_3	a_3	x_{10}	$y_3 y_5$	Y_4	3
	b_2	$\bar{x}_4 \bar{x}_{10}$	$y_5 y_8 y_9 z_2$	$Y_6 \rightarrow Y_{17}$	4
	a_9	$x_4 \bar{x}_{10}$	$y_{10} y_{15}$	Y_{10}	5
a_9	a_2	$\bar{x}_6 x_{11}$	$y_5 y_8 y_9$	Y_6	6
	a_3	$\bar{x}_{11}$	$y_3 y_8 y_9$	Y_{14}	7
	a_9	$x_6 x_{11}$	$y_{10} y_{15}$	Y_{10}	8
b_2	a_3	z_1	-	Y_0	9
	a_2	z_4	-	Y_0	10

Таблица переходов компонентного автомата S^3

a_m	a_s	$X(a_m, a_s)$	$Y(a_m, a_s)$	Y_t	B
a_4	a_4	$\bar{x}_7 x_9$	$y_{13} y_{18}$	Y_5	1
	b_3	x_7	$y_{13} z_3$	Y_3^*	2
	a_8	$\bar{x}_7 \bar{x}_9$	$y_{13} y_{14}$	Y_{13}	3
a_7	b_3	x_8	$y_{14} y_{18} z_4$	Y_{15}^*	4
	a_4	$\bar{x}_8$	$y_{13} y_{18}$	Y_5	5
a_8	a_4	$\bar{x}_9$	y_6	Y_9	6
	a_7	x_9	$y_4 y_6$	Y_{11}	7
b_3	a_8	z_2	-	Y_0	8

*-индексация микрокоманд не поменялась, так как с появлением дополнительных входных сигналов прежние микрокоманды просто потеряли своё существование.

5. Начальное состояние a_1^m компонентного автомата S^m определяется следующим образом:

$$a_1^m = \begin{cases} a_1, & \text{если } a_1 \in A^m; \\ b_m & \text{в противном случае.} \end{cases}$$

Таким образом, в начальный момент времени все компонентные автоматы находятся в состояниях b_m , за исключением того автомата, у которого начальное состояние совпадает с начальным состоянием декомпозируемого автомата S .

Обсудим теперь критерии и методику выбора исходного разбиения, которое однозначно определяет сеть C . Очевидно, что было бы желательно, чтобы при декомпозиции у любых двух компонентных автоматов сети S^m и S^p множества входных (X^m и X^p) или выходных (Y^m и Y^p) переменных (а может быть и тех и других) не пересекались, или это пересечение было минимальным. Для этого на множестве A состояний исходного автомата S можно ввести ряд отношений, используя его таблицу переходов.

Введём на множестве A внутренних состояний исходного декомпозируемого автомата S отношение ω . Два состояния a_i и a_j ($i \neq j$) автомата S находятся в отношении ω ($a_i \omega a_j$), если и только если в множествах $X(a_i)$ и $X(a_j)$ есть общие входные переменные: $X(a_i) \cap X(a_j) \neq \emptyset$. Ясно, что это отношение симметрично и не транзитивно. Построим граф G_ω этого отношения (рис. 5.9, а) для автомата S из табл. 5.1.

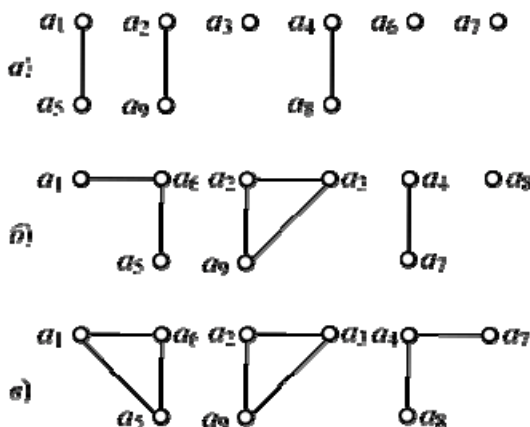


Рис. 5.9. Графы отношений исходного автомата

Графу G_ω можно поставить в соответствие некоторое разбиение π_ω на множестве A , такое, что в один блок π_ω попадут все состояния из одной компоненты связности G_ω . Число блоков π_ω , таким образом, равно числу компонент связности G_ω .

Пусть A^m и A^p — множества вершин в m -м и p -м компонентах связности G_ω . Если $X(A^m) = \bigcup_{a_i \in A^m} X(a_i)$ и $X(A^p) = \bigcup_{a_j \in A^p} X(a_j)$, то

$X(A^m) \cap X(A^p) = \emptyset$ при $m \neq p$ для любых $m, p = 1, \dots, n$, где n — число компонент связности графа G_ω . Тогда, если в качестве исходного разбиения выбрать π_ω , то пересечение множеств входных переменных для любой пары компонентных автоматов будет пусто.

Часто оказывается необходимым, чтобы сеть удовлетворяла ещё каким-либо свойствам, кроме пустого пересечения множеств

входных переменных для компонентных автоматов. Например, может быть задано число компонентных автоматов, и оно меньше числа блоков в разбиении π_ω , или число состояний в каждом компонентном автомате должно быть больше, чем число элементов в каждом блоке разбиения π_ω . Тогда по π_ω можно построить некоторое разбиение $\pi_\omega^* > \pi_\omega$ путём объединения некоторых блоков π_ω в один блок. Очевидно, что если π_ω используется в качестве исходного разбиения при декомпозиции, то и для π_ω^* справедливо условие пустого пересечения множеств входных переменных.

Точно так же на множестве A состояний автомата S можно задать отношение φ , такое, что $a_i \varphi a_j$ ($i \neq j$), если и только если пересечение множеств выходных управляющих сигналов (выходных переменных) на всех переходах из состояний a_i и a_j не пусто: $Y(a_i) \cap Y(a_j) \neq \emptyset$. Тогда, если в качестве исходного разбиения использовать π_φ или $\pi_\varphi^* > \pi_\varphi$, то для любой пары компонентных автоматов S^m и S^p множества выходных переменных также разделяются: $Y(A^m) \cap Y(A^p) = \emptyset$. Из графа G_φ (рис. 5.9, б) для рассматриваемого автомата S получаем:

$$\pi_\varphi = \overline{a_1, a_5, a_6}; \quad \overline{a_2, a_3, a_9}; \quad \overline{a_4, a_7}; \quad \overline{a_8}.$$

Если на множестве A определить отношение α , в котором будут находиться два состояния a_i и a_j ($i \neq j$), если и только если пересечение множеств входных переменных или выходных переменных на всех переходах из состояний a_i и a_j не пусто:

$$(X(a_i) \cap X(a_j) \neq \emptyset) \vee (Y(a_i) \cap Y(a_j) \neq \emptyset) = 1,$$

то использование разбиения π_α или $\pi_\alpha^* > \pi_\alpha$ при декомпозиции позволит разделить входные и выходные переменные у компонентных автоматов. Непосредственно из рис. 5.9, в для автомата S (см. табл. 5.1) получаем разбиение

$$\pi_\alpha = \overline{a_1, a_5, a_6}; \quad \overline{a_2, a_3, a_9}; \quad \overline{a_4, a_7, a_8}$$

для реализации сети из трёх компонентных автоматов, что и было использовано в рассматриваемом примере.

На практике существуют и другие, эвристические методы решения задачи декомпозиции, при которых разделяются входные

или выходные переменные (или и те, и другие) в компонентных автоматах и минимизируется число связей между ними. Эти методы сводятся к задаче нахождения разбиения $\pi_{\omega}^* \geq \pi_{\omega}$ ($\pi_{\phi}^* \geq \pi_{\phi}$ или $\pi_{\alpha}^* \geq \pi_{\alpha}$), в которой ищется объединение блоков π_{ω} в блоки π_{ω}^* (блоков π_{ϕ} в блоки π_{ϕ}^* или блоков π_{α} в блоки π_{α}^*) с минимизацией числа связей между состояниями, попавшими в различные блоки.

Контрольные вопросы и упражнения

1. Поясните упрощённую схему программируемой логической матрицы.
2. Перечислите практические задачи, позволяющие минимизировать суммарную площадь матричной реализации автомата.
3. Поясните процедуру анализа микрокоманд с целью выявления множества микрокоманд, подлежащих кодированию.
4. Сформулируйте основные требования, предъявляемые к сетям компонентных автоматов.
5. Определите объекты описания компонентных автоматов исходя из описания объектов исходного автомата.
6. Определите функцию переходов и функцию выходов на паре состояний, принадлежащих разным компонентным автоматам.
7. В чём заключается сущность π -разбиения?
8. Перечислите основные критерии декомпозиции, определяющие сеть компонентных автоматов.

Глава 6. СИНТЕЗ ЦИФРОВЫХ АВТОМАТОВ НА РАСПРЕДЕЛИТЕЛЯХ СИГНАЛОВ

Для многих микропрограмм типична линейная последовательность функциональных операторов (микрокоманд). При этом большая часть микропрограммы состоит из групп операторов $Y_{i+1}, Y_{i+2}, \dots, Y_{i+n}$, выполнение которых связано с последовательностью тактов $t+1, t+2, \dots, t+N$. Для такого рода микропрограмм типичен жёсткий порядок следования управляющих сигналов во времени, который порождает жёсткий порядок переключения состояний автомата $a_{i+1}, a_{i+2}, \dots, a_{i+N}$. Учёт этого свойства автомата позволяет упростить средства формирования сигналов возбуждения, в результате чего сокращается количество оборудования в управляющем автомате.

6.1. Распределители сигналов

Распределителем сигналов называется схема формирования последовательности сигналов $a_1, a_2, \dots, a_N$, принимающих значения единицы в последовательные моменты времени $t = 1, 2, \dots, N$.

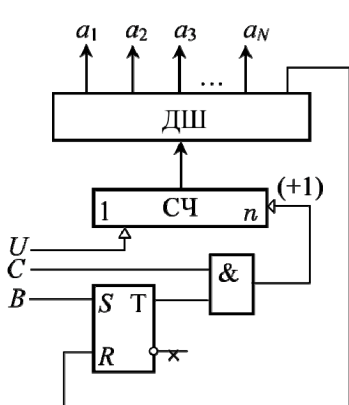


Рис. 6.1. Реализация
распределителя сигналов

Распределитель сигналов может строиться на основе счётчика и дешифратора по схеме рис. 6.1 (в простейшем функциональном варианте). Количество разрядов счётчика по отношению к длине распределителя сигналов равно $n = \lceil \log_2(N + 1) \rceil$. В исходное состояние СЧ = 0 счётчик приводится сигналом начальной установки U . Запуск распределителя производится сигналом B , который устанавливает управляющий триггер T в состояние 1, в результате чего снимается блокировка с цепи C и в

схеме по каждому синхронизирующему сигналу C выполняется действие $СЧ := СЧ + 1$. Счётчик последовательно переключается в

состояния $1, 2, \dots, 2^n - 1$, выделяемые дешифратором ДШ, на выходе которого формируется последовательность сигналов $a_1, a_2, \dots, a_N$. При переключении счётчика в состояние $(N+1)$ сигнал a_{N+1} переключает триггер T в состояние 0, и цикл формирования сигналов заканчивается.

Схемы с эквивалентными функциями могут быть построены на основе элементов задержки и сдвигающих регистров. В последних сдвигается унитарный код $100\dots 00$ и единица последовательно проходит через разряды $1, 2, \dots, N$.

В дальнейшем распределители сигналов РС независимо от их внутренней структуры обозначаются, как показано на рис. 6.2, а, или в виде отдельного операционного элемента (рис. 6.2, б).

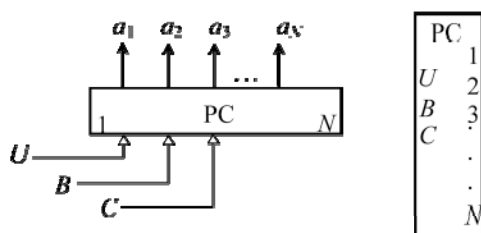


Рис. 6.2. Условное графическое обозначение распределителя сигналов

Сигнал U обеспечивает начальную установку распределителя сигналов, сигнал B – запуск распределителя сигналов и C – синхронизацию переходов. В ответ на сигнал B распределитель формирует последовательность сигналов $a_1, a_2, \dots, a_N$ и переключается в исходное состояние, в котором он пребывает до поступления очередного запускающего сигнала B . Длительность выходных сигналов определяется периодом следования синхронизирующих сигналов. Граф функционирования распределителя сигналов изображён на рис. 6.3. Символы $a_1, a_2, \dots, a_N$ обозначают как состояния, так и выходные сигналы, формируемые в каждом состоянии.

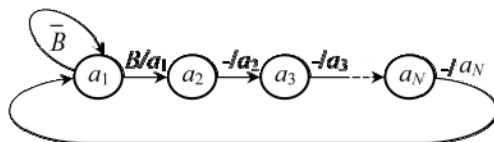


Рис. 6.3. Граф функционирования распределителя сигналов

6.2. Интерпретация линейных микропрограмм

Рассмотрим микропрограмму на рис. 6.4. Граф микропрограммы отметим символами $a_1, a_2, \dots, a_N$, обозначающими состояния автомата. Если все пути на отмеченном графе относятся к типу $a_i X_i Y_i a_{i+1}$ ($i = 1, 2, \dots, N-1$) или $a_N X_N Y_N a_1$, то микропрограмма называется *линейной*. Здесь $X_j = \tilde{x}_\alpha \tilde{x}_\beta \dots \tilde{x}_\omega$ – терм, определяющий путь через условные вершины $x_\alpha x_\beta \dots x_\omega$; $Y_j = \{y_a, y_b, \dots, y_w\}$ – множество выходных сигналов, которое может быть и пустым.

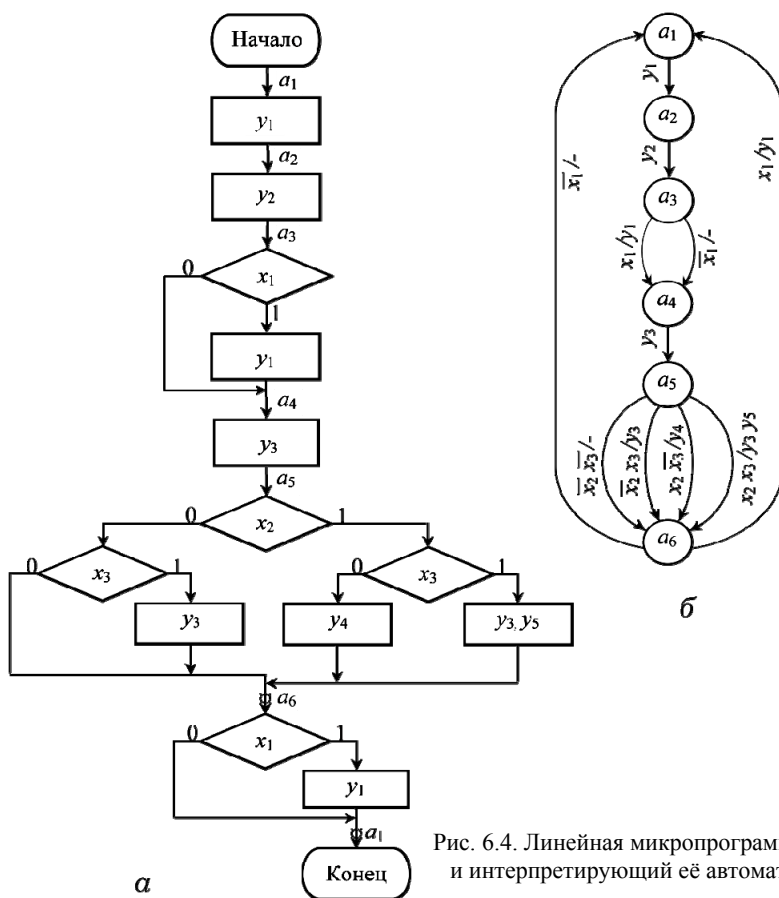


Рис. 6.4. Линейная микропрограмма (а) и интерпретирующий её автомат (б)

Линейная микропрограмма интерпретируется распределителем сигналов с N состояниями следующим образом. На основе графа микропрограммы строится таблица переходов, отмечающая состояния $a_1, a_2, \dots, a_N$, входные $\tilde{x}_\alpha \tilde{x}_\beta \dots \tilde{x}_\omega$ и выходные $\{y_a, y_b, \dots, y_w\}$ наборы, связанные с каждым переходом.

Графу на рис. 6.4 соответствует табл. 6.1. Так как порядок переключения состояний $a_1, a_2, \dots, a_N$ однозначно задан PC , функция автомата определяется функцией выхода, которая описывается системой булевых функций следующего вида:

$$y_k = \bigvee_{y_k \in Y_i} a_i (\tilde{x}_\alpha \tilde{x}_\beta \dots \tilde{x}_\omega)_i \quad (k = 1, 2, \dots, M).$$

Таблица 6.1

Переходы в графе микропрограммы рис. 6.4

Состояние	Входной набор	Выходной набор	Состояние	Входной набор	Выходной набор
a_1	1	y_1	a_5	$\bar{x}_2\bar{x}_3$	—
a_2	1	y_2		$\bar{x}_2x_3$	y_3
a_3	x_1	y_1		$x_2\bar{x}_3$	y_4
	$\bar{x}_1$	—		x_2x_3	y_3y_5
a_4	1	y_3	a_6	x_1 $\bar{x}_1$	y_1 —

Функции составляются из термов, которые соответствуют переходам $i = 1, 2, \dots, P$. Терм вводится в функцию y_k , если на данном переходе выходной набор содержит переменную y_k . В рассматриваемом примере имеем:

$$\left. \begin{aligned} y_1 &= a_1 \vee a_3 x_1 \vee a_6 x_1 = a_1 \vee (a_3 \vee a_6) x_1; \\ y_2 &= a_2; \\ y_3 &= a_4 \vee a_5 \bar{x}_2 x_3 \vee a_5 x_2 x_3 = a_4 \vee a_5 x_3; \\ y_4 &= a_5 x_2 \bar{x}_3; \\ y_5 &= a_5 x_2 x_3. \end{aligned} \right\}$$

На основе данных выражений построена функционально-логическая схема (рис. 6.5). Жёсткий порядок переключения состояний в автоматах с распределителями сигналов канонизирует порядок формирования сигналов возбуждения и функция переходов автомата реализуется распределителем сигналов при минимальных затратах оборудования, благодаря чему снижается стоимость автомата.

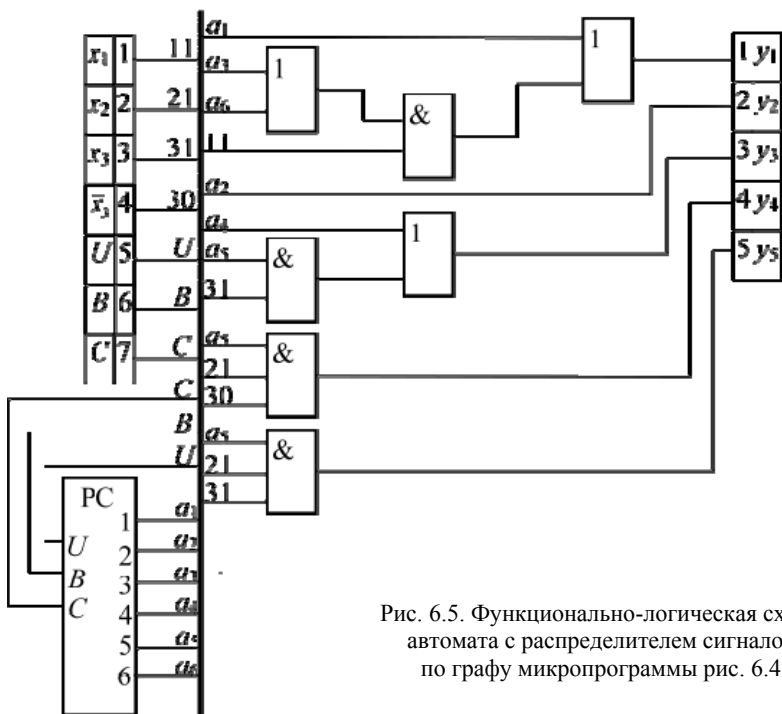


Рис. 6.5. Функционально-логическая схема автомата с распределителем сигналов по графу микропрограммы рис. 6.4

Из рис. 6.4 и табл. 6.1 видно, что некоторые переходы являются пустыми, то есть в некоторые такты автомат не формирует ни одного выходного сигнала. Из-за жёсткого порядка переключения состояний время выполнения микропрограммы всегда будет постоянным и равным максимальному значению, в результате чего снижается быстродействие операционного устройства, управляемого такого рода автоматом.

6.3. Интерпретация микропрограмм, содержащих разветвления и циклы

Рассмотрим микропрограмму (рис. 6.6), для изображения структуры которой используются следующие обозначения: H – начало микропрограммы; Y_i – функциональный оператор, содержащий некоторый набор микроопераций $y_a, y_b, \dots, y_w$; x_j – логическое условие; K – конец микропрограммы.

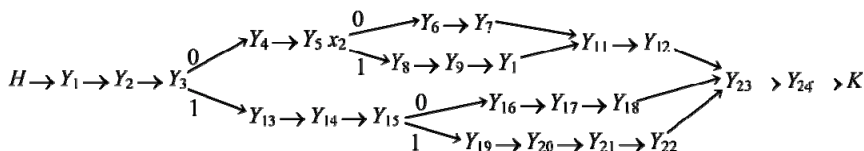


Рис. 6.6. Микропрограмма с разветвлениями

Для интерпретации такой микропрограммы автоматом с распределением сигналов необходимо каждый функциональный оператор Y_i связать с состоянием a_i автомата, т.е. с тактом t , в котором будет реализован оператор Y_i . Привязка операторов $Y_1, Y_2, \dots, Y_N$ к тактам $t = 1, 2, \dots, N$ производится следующим образом. Отметке начала микропрограммы H присваивается номер $t = 0$. Операторам Y_i присваиваются номера тактов:

$$t_i = \max(t_\alpha, t_\beta, \dots, t_\omega) + 1 \quad (i = 1, 2, \dots, K),$$

где $t_\alpha, t_\beta, \dots, t_\omega$ – номера тактов, которые присвоены операторам $Y_\alpha, Y_\beta, \dots, Y_\omega$, предшествующим и соседним оператору Y_i .

Операторы микропрограммы рис. 6.6 распределяются по тактам:

$$\begin{aligned}
 Y_1 &\rightarrow t_1 = 1; & Y_7 &\rightarrow t_7 = 7; & Y_{13} &\rightarrow t_{13} = 4; & Y_{19} &\rightarrow t_{19} = 7; \\
 Y_2 &\rightarrow t_2 = 2; & Y_8 &\rightarrow t_8 = 6; & Y_{14} &\rightarrow t_{14} = 5; & Y_{20} &\rightarrow t_{20} = 8; \\
 Y_3 &\rightarrow t_3 = 3; & Y_9 &\rightarrow t_9 = 7; & Y_{15} &\rightarrow t_{15} = 6; & Y_{21} &\rightarrow t_{21} = 9; \\
 Y_4 &\rightarrow t_4 = 4; & Y_{10} &\rightarrow t_{10} = 8; & Y_{16} &\rightarrow t_{16} = 7; & Y_{22} &\rightarrow t_{22} = 10; \\
 Y_5 &\rightarrow t_5 = 5; & Y_{11} &\rightarrow t_{11} = 9; & Y_{17} &\rightarrow t_{17} = 8; & Y_{23} &\rightarrow t_{23} = 11; \\
 Y_6 &\rightarrow t_6 = 6; & Y_{12} &\rightarrow t_{12} = 10; & Y_{18} &\rightarrow t_{18} = 9; & Y_{24} &\rightarrow t_{24} = 12.
 \end{aligned}$$

Следовательно, распределитель сигналов, интерпретирующий данную микропрограмму, должен иметь 12 состояний.

Для определения функций выходов $Y = F(a, x)$ каждому оператору Y_i ставится в соответствие терм $X_i = \tilde{x}_\alpha \tilde{x}_\beta \dots \tilde{x}_\omega$, определяющий условие выполнения этого оператора.

Терм X_i определяется в виде:

$$X_i = X_\alpha \tilde{x}_\alpha \vee X_\beta \tilde{x}_\beta \vee \dots \vee X_\omega \tilde{x}_\omega.$$

Здесь $X_\alpha, X_\beta, \dots, X_\omega$ – термы операторов $Y_\alpha, Y_\beta, \dots, Y_\omega$, предшествующих и соседних оператору Y_i и $\tilde{x}_\alpha, \tilde{x}_\beta, \dots, \tilde{x}_\omega$ – логические условия, указанные между операторами Y_α и Y_i , Y_β и Y_i и т.д.

Если между операторами Y_j и Y_i нет логических условий, то $\tilde{x}_k = 1$ и переменная $\tilde{x}_k$ не отмечается в терме.

Из микропрограммы рис. 6.6 следует:

$$\begin{array}{lll} X_1 = 1; & X_9 = \bar{x}_1 x_2; & X_{17} = x_1 \bar{x}_3; \\ X_2 = 1; & X_{10} = \bar{x}_1 x_2; & X_{18} = x_1 \bar{x}_3; \\ X_3 = 1; & X_{11} = X_7 \vee X_{10} = \bar{x}_1; & X_{19} = x_1 x_3; \\ X_4 = \bar{x}_1; & X_{12} = \bar{x}_1; & X_{20} = x_1 x_3; \\ X_5 = \bar{x}_1; & X_{13} = x_1; & X_{21} = x_1 x_3; \\ X_6 = \bar{x}_1 \bar{x}_2; & X_{14} = x_1; & X_{22} = x_1 x_3; \\ X_7 = \bar{x}_1 \bar{x}_1; & X_{15} = x_1; & X_{23} = X_{12} \vee X_{18} \vee X_{22} = 1; \\ X_8 = \bar{x}_1 x_2; & X_{16} = x_1 \bar{x}_3; & X_{24} = 1. \end{array}$$

Терм $X_i = 1$ означает, что оператор Y_i выполняется при каждой реализации микропрограммы.

Теперь функцию выходов $Y = F(a, x)$ можно описать системой булевых функций:

$$Y_i = a_i X_i \quad (i = 1, 2, \dots, K).$$

Функция выходов, соответствующая рассматриваемой микропрограмме, представляется следующими выражениями:

$$\begin{aligned}
Y_1 &= a_1; & Y_7 &= a_7 \bar{x}_1 \bar{x}_2; & Y_{19} &= a_7 x_1 x_3; \\
Y_2 &= a_2; & Y_8 &= a_6 \bar{x}_1 x_2; & Y_{20} &= a_8 x_1 x_3; \\
Y_3 &= a_3; & Y_9 &= a_7 \bar{x}_1 x_2; & Y_{21} &= a_9 x_1 x_3; \\
Y_4 &= a_4 \bar{x}_1; & & \dots\dots\dots & Y_{22} &= a_{10} x_1 x_3; \\
Y_5 &= a_5 \bar{x}_1; & Y_{17} &= a_8 x_1 \bar{x}_3; & Y_{23} &= a_{11}; \\
Y_6 &= a_6 \bar{x}_1 \bar{x}_2; & Y_{18} &= a_9 x_1 \bar{x}_3; & Y_{24} &= a_{12},
\end{aligned}$$

где $a_1, a_2, \dots, a_{12}$ – сигналы, формируемые распределителем сигналов.

Функционально-логическая схема автомата строится как реализация полученных выражений функций выходов.

Теперь рассмотрим интерпретацию микропрограмм, содержащих циклы. Пусть микропрограмма содержит цикл

$$H \rightarrow Y_1 \rightarrow Y_2 \rightarrow Y_3 \rightarrow Y_4 \rightarrow Y_5 \rightarrow Y_6 \rightarrow Y_7 x \begin{matrix} \xrightarrow{0} Y_8 \rightarrow Y_9 \rightarrow Y_{10} \rightarrow K \\ \xrightarrow{1} Y_3 \end{matrix}$$

Цикл состоит из операторов $Y_3, \dots, Y_7$, которые повторяются при $x = 1$. При $x = 0$ действие выходит из цикла. Для интерпретации этой микропрограммы в распределителе сигналов можно ввести цепи, посредством которых реализуется переход $a_7 \bar{x} a_3$ или переход $a_7 x a_8$. Другое решение представлено на рис. 6.7. Здесь для синхронизации используется двухфазная последовательность сигналов C_1, C_2 .

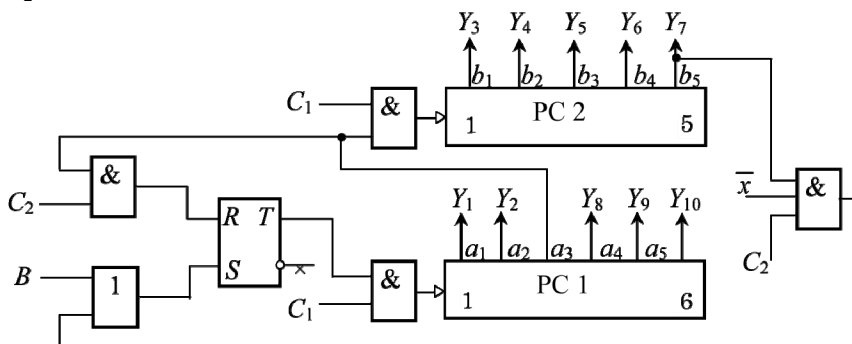


Рис. 6.7. Принцип интерпретации микропрограммы с циклом

Распределитель PC2 управляет выполнением операторов, составляющих тело цикла. Запускающий сигнал B устанавливает

триггер T в состояние 1, и распределитель PC1, последовательно переключаясь в состояния a_1 , a_2 , формирует сигналы Y_1 , Y_2 . При переходе в состояние a_3 триггер T переключается в состояние 0, и сигнал a_3 разблокирует цепь синхронизации распределителя PC2, который начинает генерировать последовательность сигналов $Y_3Y_4...Y_7Y_3Y_4...$ до появления сигнала $x = 0$. При $x = 0$ формируется сигнал $b_5\bar{x}$ и триггер T устанавливается в состояние 1. В следующем такте распределитель PC1 переходит в состояние a_4 и выполнение цикла заканчивается.

Контрольные вопросы и упражнения

1. Каковы основные способы реализации распределителей сигналов?
2. Дайте определение линейной микропрограммы.
3. Приведите пример линейной микропрограммы.
4. Как определяются такты выдачи микрокоманд при реализации разветвлённой микропрограммы?
5. Запишите условие определения входного сигнала, в зависимости от которого формируются соответствующая микрокоманда.
6. Проведите анализ эффективности реализации автомата с циклической микропрограммой на распределителях сигналов. Если эта реализация не эффективна по каким-либо причинам, то объясните почему.

Глава 7. ЦИФРОВЫЕ АВТОМАТЫ С ХРАНИМОЙ В ПАМЯТИ МИКРОПРОГРАММОЙ

7.1 Структура микрокоманд. Общие положения

Управляющий автомат может быть построен на основе *принципа программного управления*, использующего операционно-адресную структуру управляющих слов. При этом алгоритм управления представляется упорядоченной совокупностью управляющих слов. Управляющее слово определяет функционирование устройства в течение одного такта и задаётся микрокомандой (*МК*). Совокупность микрокоманд образует массив *МК* $[0:P]$, называемый *микропрограммой*. Отдельные микрокоманды в массиве *МК* $[0:P]$ выделяются посредством адреса хранения. Микрокоманда содержит информацию о микрооперациях, которые должны выполняться в данном такте работы устройства, и (или) информацию об адресе следующей микрокоманды.

Определим простейшую структуру управляющих слов, достаточную для представления микрокоманд.

Пусть множество Y содержит M управляющих сигналов, инициализирующих выполнение соответствующих микроопераций, причём в каждом такте совместно выдаётся не более K управляющих сигналов. Управляющие сигналы $y_1, y_2, \dots, y_M$ кодируются m -разрядными двоичными числами $Y(m)$, где $m = \lceil \log_2 M \rceil$. Операционная часть (ОЧ) микрокоманды, управляющая микрооперациями, должна содержать K полей (рис. 7.1).

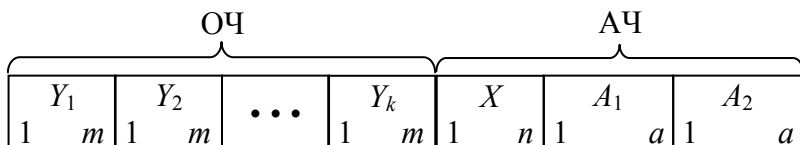


Рис. 7.1. Структура микрокоманды

Для формирования адреса следующей микрокоманды используется принудительная или естественная адресация. *Принудительная адресация* микрокоманд состоит в том, что в каждой микрокоманде

указывается адрес следующей микрокоманды (два принудительных адреса или один принудительный адрес). Адрес следующей микрокоманды может задаваться безусловно, т.е. независимо от значений логических условий X , или выбираться по условию, определяемому текущими значениями логических условий.

В каждой микрокоманде можно задавать проверку только одного логического условия из множества X . Тогда адресная часть (АЧ) микрокоманды в самом общем виде будет иметь структуру, изображённую на рис. 7.1. Поле $X(n)$ определяет номер (код) логического условия $x_1, x_2, \dots, x_n$, значение которого анализируется микрокомандой. Если $X = i \neq 0$, то адрес следующей микрокоманды определяется в зависимости от значения x_i : при $x_i = 0$ следующей выполняется микрокоманда с адресом A_1 , при $x_i = 1$ – микрокоманда с адресом A_2 . Если $X = 0$, то адрес следующей микрокоманды равен A_1 (отсутствие проверки значения логического условия – безусловный переход). Длина адресов A_2 и A_1 зависит от количества микрокоманд, составляющих микропрограмму. Если микропрограмма содержит не более P микрокоманд, то $a = \lceil \log_2 P \rceil$. В более общем случае длина адресов определяется ёмкостью постоянного запоминающего устройства и структурной организацией хранения микропрограммы.

Если адресная часть содержит только один принудительный адрес, то адрес следующей выполняемой микрокоманды функционально зависит от адреса перехода A_p . При отсутствии проверки значения логического условия ($X = 0$), или когда $x_i = 0$, следующей выполняется микрокоманда по адресу A_p . При $x_i = 1$ возможны два варианта формирования адреса следующей выполняемой микрокоманды. Первый вариант, когда при $x_i = 1$ адрес следующей микрокоманды формируется как инкремент адреса перехода A_p ($A_p + 1$), второй вариант – как инкремент адреса текущей микрокоманды ($A_i + 1$).

7.2. Постоянные запоминающие устройства

Для хранения микропрограмм используются постоянные запоминающие устройства (постоянная память – ПП), принцип структурной организации которых изображён на рис. 7.2. Постоянная

память разделяется на две части: запоминающую и адресную. Запоминающая ячейка ПП обеспечивает хранение k -разрядного слова информации, представляющего обычно одну микрокоманду. Информация в ячейку заносится на стадии изготовления либо, например, пережиганием перемычек (по аналогии с ПЛМ, см. гл. 5). В цифровой схемотехнике существуют и перепрограммируемые постоянные запоминающие устройства. Адресная часть состоит из дешифратора адреса, посредством которого a -разрядный двоичный адрес $A(a)$ преобразуется в сигнал, опрашивающий ячейку A запоминающей части.

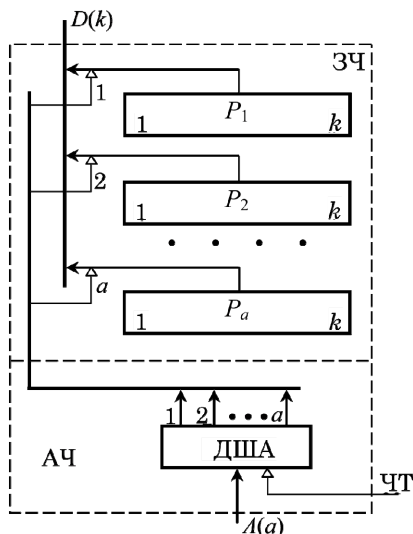


Рис. 7.2. Структура постоянной памяти

Обращение к постоянной памяти выполняется путём передачи адреса A на адресную шину $A(a)$ и посылки управляющего сигнала $ЧТ$, инициирующего процесс чтения слова информации. В результате этого на выходной шине $D(k)$ формируется слово $D := ПП[A]$, равное константе, хранимой в ячейке A постоянной памяти.

Принцип построения запоминающей части постоянной памяти на логическом уровне иллюстрируется рис. 7.3. Запоминающая часть состоит из адресных линий $a_1, a_2, a_3, \dots$, по которым поступают сигналы с дешифратора адреса ДША, и информационных линий $d_1, d_2, d_3, \dots, d_k$, с которых снимается код слова $D(k)$. Адресные и информационные линии образуют сетку, в узлах которой размещаются элементы, передающие сигналы с

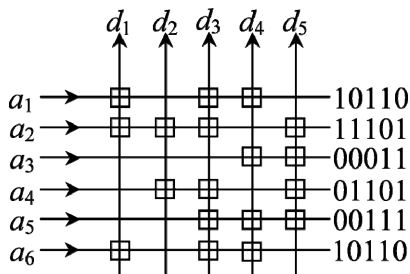


Рис. 7.3. Хранение информации в постоянной памяти

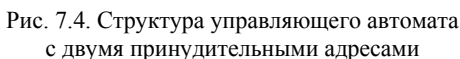
линии a_i в линию d_j . Если j -й разряд слова с адресом i должен содержать единицу, то линия a_i соединяется с линией d_j . Код нуля формируется при отсутствии соединения между указанными линиями. На рисунке 7.3 коммутация между цепями условно обозначается прямоугольником, «направляющей» сигнал a_i в линию d_j . В запоминающей части постоянной памяти, изображённой на рис. 7.3, сигналы $a_1, a_2, a_3, \dots$, опрашивающие ячейки с адресами 1, 2, 3, ..., формируют коды $D = 10110, 11101, 00011, \dots$

7.3 Структура и функционирование управляющего автомата

Пусть микрокоманды имеют структуру с двумя принудительными адресами, показанную на рис. 7.1, причём $K = 2$, т.е. операционная часть ограничивается двумя полями Y_1 и Y_2 , используемыми для возбуждения совместимых микроопераций. В таком случае управляющий автомат строится по схеме рис. 7.4. Для хранения микрокоманд используется постоянная память k -разрядных слов соответствующей ёмкости. Управляющий сигнал ЧТ инициирует процедуру чтения слова РМК:=ПП[РАПП], в результате выполнения которой из ячейки A согласно содержимому регистра адреса постоянной памяти РАПП на регистр микрокоманд РМК считывается микрокоманда.

Микрокоманда, выбранная на регистр РМК, обрабатывается следующим образом. Поля Y_1 и Y_2 дешифрируются, и выходные сигналы с дешифраторов ДШУ₁ и ДШУ₂ поступают в качестве управляющих сигналов $y_1, y_2, \dots, y_M$ в операционный автомат, возбуждая в нём заданные микрооперации. При назначенной структуре микрокоманды в каждом такте может выполняться не более двух микроопераций.

После выполнения микроопераций осуществляется переход к следующей микрокоманде, адрес которой определяется полем A_1 или A_2 в зависимости от значения поля кода логического условия КЛУ. Если значение проверяемого логического условия $x_i = 0$ или при отсутствии проверки (x_0 – безусловный переход), то РАПП := A_1 , если $x_i = 1$, то РАПП := A_2 .



Перед запуском управляющего автомата производится установка начального адреса микропрограммы A_0 . Запуск производится сигналом B , который переключает триггер T в состояние 1, в результате чего синхронизирующий сигнал C поступает на управляющий вход ЧТ, возбуждая в каждом такте процесс чтения мик-

рокоманды из постоянной памяти. В последнем такте реализации микропрограммы триггер T должен быть переключён в состояние 0. Для этого в системе микроопераций вводится дополнительная микрооперация y_F , которая интерпретируется в виде сигнала, останавливающего работу автомата (F).

Такт работы автомата разделяется на последовательность микротактов: T_1 – чтение микрокоманды РМК := ПП[РАПП]; T_2 – дешифрирование полей Y и КЛУ и формирование сигналов $\{y_j\}$; T_3 – выполнение микроопераций $\{y_j\}$ и проверка значений логических условий $\{x_i\}$; T_4 – формирование адреса следующей микрокоманды по управляющим сигналам ε_0 и ε_1 . Длительность такта равна сумме длительностей микротактов и определяется, в первую очередь, длительностью T_1 (цикла чтения микрокоманды) и максимальной длительностью микроопераций $T_3 = \max(\tau_1, \tau_2, \dots, \tau_M)$. При использовании в схеме рис. 7.4 двухступенчатых дешифраторов микротакты T_2 и T_4 имеют длительность $T_2 = 3\tau$ и $T_4 = 6\tau$.

Управляющие автоматы с программируемой логикой, реализующие алгоритм управления на основе хранимой в постоянной памяти микропрограммы, называются П-автоматами.

7.4. Кодирование микроопераций

Микрооперации из множества $Y = \{y_1, y_2, \dots, y_M\}$ возбуждаются микрокомандой, в операционной части которой указываются наименования (коды) микроопераций, выполняемых устройством совместно в течение такта.

Для кодирования совместно выполняемых микроопераций в микрокоманде выделяются *операционные поля* $Y_1, Y_2, \dots, Y_K$, количество которых определяет предельное число совместно выполняемых микроопераций. В общем случае поле Y_k может возбуждать некоторое подмножество микроопераций

$$Y_k = \{y_\alpha, y_\beta, \dots, y_\omega\} \subset Y,$$

для кодирования которых используется $m_k = \log_2(M_i + 1)$ двоичных разрядов, причём $Y_k = 0$ является признаком пустоты поля.

Сумма $M_0 = m_1 + m_2 + \dots + m_K$ определяет длину операционной части микрокоманды. Длина операционной части M_0 микрокоманды влияет на количество оборудования в управляющем автомате и быстродействие операционного устройства. В частности, уменьшение длины M_0 приводит к сокращению разрядности постоянной памяти, вследствие чего уменьшается её стоимость. Если следовать критерию экономии оборудования, то целесообразно по возможности уменьшать длину операционной части микрокоманды.

7.4.1. Распределение микроопераций по полям микрокоманды

Пусть автомат управляет микрооперациями $Y = \{y_1, y_2, \dots, y_M\}$, отношение между которыми задано матрицей совместимости:

$$S = [s_{ij}] = \begin{bmatrix} 0 & s_{12} & \dots & s_{1M} \\ s_{21} & 0 & \dots & s_{2M} \\ \dots & \dots & \dots & \dots \\ s_{M1} & s_{M2} & \dots & 0 \end{bmatrix}.$$

Элемент s_{ij} , характеризует совместимость микроопераций y_i и y_j , $s_{ij} = 0$, если микрооперации несовместимы, и $s_{ij} = 1$, если они совместимы. Матрица S является булевой и симметричной ($s_{ij} = s_{ji}$).

Положим, что для кодирования микрооперации используется K операционных полей $Y_1, Y_2, \dots, Y_K$. Чтобы инициализировать микрооперацию y_j достаточно приписать её одному операционному полю Y_i . Стремясь к минимизации длины операционной части микрокоманды можно потребовать, чтобы число операционных полей $Y_1, Y_2, \dots, Y_K$ было минимальным и каждая микрооперация y_i была приписана единственному полю, т.е.

$$Y_i \cap Y_j = \emptyset \quad (i, j = 1, 2, \dots, K; i \neq j). \quad (7.1)$$

Очевидно, что в одном поле могут размещаться только несовместимые микрооперации, т.е. если $Y_j = \{y_\alpha, y_\beta, \dots, y_\omega\}$, то $s_{kl} = 0$ для всех $k, l = \alpha, \beta, \dots, \omega$. Подмножества $Y_1, Y_2, \dots, Y_K$ должны отвечать требованию полноты

$$\bigcup_{j=1}^K Y_j = Y, \quad (7.2)$$

из которого следует, что каждая микрооперация y_i должна быть включена хотя бы в одно поле Y_j .

Таким образом, задача распределения микроопераций Y , для которых определены отношения совместимости, сводится к определению минимального числа подмножеств $Y_1, Y_2, \dots, Y_K \subset Y$, каждое из которых содержит только несовместимые микрооперации и удовлетворяет условиям (7.1) и (7.2).

Рассмотрим один из способов решения данной задачи, называемый *методом прямого включения*.

Процесс распределения микроопераций $Y = \{y_1, y_2, \dots, y_M\}$ по подмножествам $Y_1, Y_2, \dots$ будем разделять на M шагов. На каждом шаге для очередной микрооперации отыскивается подмножество Y_j , в которое можно включить микрооперацию y_i . Микрооперация y_i включается в подмножество Y_j , если она несовместима ни с одной микрооперацией этого подмножества. Если среди подмножеств $Y_1, Y_2, \dots, Y_k$ такое подмножество отсутствует, то для микрооперации y_i назначается новое подмножество Y_{k+1} , в которое включается y_i . Состояние процесса на каждом шаге будем характеризовать матрицей включения

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{1M} \\ r_{21} & r_{22} & r_{2M} \\ \dots & \dots & \dots \\ r_{k1} & r_{k2} & r_{kM} \end{bmatrix},$$

элемент r_{ji} которой равен 1, если $y_i \in Y_j$, и равен 0 в противном случае. Теперь условие включения микрооперации y_i в подмножество Y_j можно сформулировать следующим образом: микрооперация y_i включается в подмножество Y_j , если i -я строка S_i матрицы S не пересекается с j -й строкой R_j матрицы R , т.е. если $S_i \cap R_j = \emptyset$.

Для иллюстрации метода прямого включения рассмотрим пример распределения микроопераций $Y = \{y_1, y_2, \dots, y_{10}\}$, отношение совместимости которых задано матрицей

$$\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0
\end{bmatrix} &
\begin{matrix}
1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10
\end{matrix}
\end{array}$$

Шаг 1. Множество $\{Y_j\}$ пусто. Для микрооперации y_1 назначаем подмножество Y_1 :

$$\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
R = [1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0] 1 ;
\end{array}$$

Шаг 2. Поскольку $S_2 \cap R_1 = y_1$, для микрооперации y_2 назначается новое подмножество Y_2 :

$$\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
R = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} 1 \\ 2 \end{matrix} ;
\end{array}$$

Шаг 3. Поскольку $S_3 \cap R_1 = \emptyset$, микрооперация y_3 включается в подмножество Y_1 :

$$\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
R = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} 1 \\ 2 \end{matrix} ;
\end{array}$$

Шаг 4. Поскольку $S_4 \cap R_1 = \emptyset$, микрооперация y_4 включается в подмножество Y_1 :

$$\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
R = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} 1 \\ 2 \end{matrix} ;
\end{array}$$

Шаг 5. Поскольку $S_5 \cap R_1 = y_4$ и $S_5 \cap R_2 = \emptyset$, микрооперация y_5 включается в подмножество Y_2 :

$$R = \begin{matrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} & \begin{matrix} 1 \\ 2 \end{matrix} \end{matrix};$$

Шаг 6. Поскольку $S_6 \cap R_1 = y_4$ и $S_6 \cap R_2 = y_5$, для микрооперации y_6 назначается новое подмножество Y_3 :

$$R = \begin{matrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} & \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \end{matrix}.$$

Продолжая процесс, после шага 10 получаем

$$R = \begin{matrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} & \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \end{matrix},$$

чему соответствует следующее распределение микроопераций:

$$Y_1 = \{y_1, y_3, y_4, y_7, y_{10}\}; \quad Y_2 = \{y_2, y_5, y_8\}; \quad Y_3 = \{y_6, y_9\}.$$

Для кодирования операционной части микрокоманды требуется $L = 3+2+2 = 7$ двоичных разрядов, причём код $Y_j = 0$ используется в качестве признака пустоты поля.

Распределение микроопераций Y по полям $Y_1, Y_2, \dots, Y_K$ следует считать оптимальным, если оно минимизирует суммарную длину полей

$$L = \sum_{j=1}^K \log_2 m_j,$$

где m_j – число микроопераций, включённых в поле Y_j . Метод прямого включения даёт в большинстве случаев неоптимальное решение, которое только в редких случаях можно признать удовлетворительным.

Оптимальное распределение микроопераций по операционным полям строится на основе *метода нахождения максимальных со-*

вместимых подмножеств, разработанного А. Д. Закревским. Этот метод состоит в следующем.

Для решения задачи распределения микроопераций Y , совместимых в соответствии с матрицей S , строится множество максимальных совместимых подмножеств. Подмножество $P \subset Y$ называется совместимым, если содержит микрооперации, для любой пары которых $s_{ij} \neq 1$. Подмножество совместимых микроопераций P называется максимальным, если его нельзя расширить включением любой микрооперации $y_i \notin P$, не нарушая условия совместимости. Множество $\{P\}$ можно построить, перебирая микрооперации из Y , находя среди них совместимые подмножества и выделяя среди последних максимальные подмножества. Этот процесс реализуется в следующем порядке, иллюстрируемом на ранее рассмотренном примере.

Введём исходный вектор

$$\begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ [0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1] \end{array},$$

определяющий, что подмножество совместных микроопераций есть $\{y_{10}\}$. Расширим подмножество, вводя в него микрооперацию y_9 :

$$[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1] .$$

Новое подмножество $\{y_9, y_{10}\}$ несовместимо, поскольку $s_{9,10} = 1$. Поэтому добавленную единицу следует выбросить и попытаться расширить подмножество включением в него микрооперации y_8 :

$$[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1] .$$

Подмножество $\{y_8, y_{10}\}$ совместимо ($s_{8,10} = 0$), поэтому добавленная в вектор единица сохраняется и производится следующее расширение:

$$[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1] .$$

Оно оказывается неприемлемым, поскольку $s_{7,8} = 1$, поэтому добавленная в вектор единица удаляется и выполняется новое расширение. Продолжая процесс перебора, получим вектор

$$[1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1] ,$$

задающий подмножество $P_1 = \{y_1, y_3, y_5, y_8, y_{10}\}$ совместимых микроопераций. Это подмножество является максимальным и включает-ся в множество максимальных совместимых подмножеств.

Для нахождения других совместимых подмножеств перебор микроопераций проводится в следующем порядке. Из последнего вектора удаляются все единицы, левее которых нет нулей, и самая левая оставшаяся единица смещается на одну позицию влево. Полученный таким образом вектор

$$[0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1]$$

содержит микрооперацию y_2 , несовместимую с y_{10} . Удаляя элемент y_2 и расширяя подмножество новым элементом y_1 , получим вектор

$$[1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1],$$

задающий подмножество совместимых микроопераций $P = \{y_1, y_5, y_8, y_{10}\}$. Поскольку $P \subset P_1$, подмножество P не является максимальным и не включается в $\{P\}$. Удалением единиц слева формируется новый вектор

$$[0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1]$$

и процесс перебора продолжается.

Процесс перебора, приводящий к формированию максимальных совместимых подмножеств, показан на примере, где символом «+» отмечены совместимые подмножества, символом «-» – несовместимые подмножества, символом P_k – подмножества, включае-мые в $\{P\}$ и символом $\subset P_k$ – не максимальные подмножества:

1	2	3	4	5	6	7	8	9	10	
0	0	0	0	0	0	0	0	0	1	+
0	0	0	0	0	0	0	0	0	1	-
0	0	0	0	0	0	0	0	1	0	+
0	0	0	0	0	0	1	1	0	1	-
0	0	0	0	0	1	0	1	0	1	-
0	0	0	0	1	0	0	1	0	1	+
0	0	0	1	1	0	0	1	0	1	-
0	0	1	0	1	0	0	1	0	1	+
0	1	1	0	1	0	0	1	0	1	-

1	0	1	0	1	0	0	1	0	1	$+P_1$
0	1	0	0	1	0	0	1	0	1	$-$
1	0	0	0	1	0	0	1	0	1	$+ \subset P_1$
0	0	0	1	0	0	0	1	0	1	$+$
0	0	1	1	0	0	0	1	0	1	$+$
0	1	1	1	0	0	0	1	0	1	$-$
1	0	1	1	0	0	0	1	0	1	$+P_2$
0	1	0	1	0	0	0	1	0	1	$-$
1	0	0	1	0	0	0	1	0	1	$+ \subset P_2$
0	0	1	0	0	0	0	1	0	1	$+$
0	1	1	0	0	0	0	1	0	1	$-$
1	0	1	0	0	0	0	1	0	1	$+ \subset P_1$
0	1	0	0	0	0	0	1	0	1	$-$
1	0	0	0	0	0	0	1	0	1	$+ \subset P_1$
0	0	0	0	0	0	1	0	0	1	$+$
0	0	0	0	0	1	1	0	0	1	$-$
0	0	0	0	1	0	1	0	0	1	$+$
0	0	0	1	1	0	1	0	0	1	$-$
0	0	1	0	1	0	1	0	0	1	$+$
0	1	1	0	1	0	1	0	0	1	$-$
1	0	1	0	1	0	1	0	0	1	$+P_3$
0	1	0	0	1	0	1	0	0	1	$-$
1	0	0	0	1	0	1	0	0	1	$+ \subset P_3$

.....

Перебор заканчивается при получении вектора, у которого плотность расположения всех единиц максимальна слева. К этому моменту будут найдены все максимальные совместимые подмножества микроопераций. Для рассматриваемого примера множество максимальных совместимых подмножеств представляется следующей совокупностью векторов:

$$P = \{P_i\} = \begin{array}{c} \begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{l} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array}$$

Теперь нужно получить оптимальное разбиение множества Y на совместимые подмножества $Y_1, Y_2, \dots$, удовлетворяющие условиям (7.1) и (7.2), что сводится к поиску минимальной совокупности максимальных совместимых подмножеств, заданных множеством $\{P_i\}$.

Приближённо эту задачу можно решить следующим образом. В матрице $\{P_i\}$ выделяется вектор P_b , содержащий максимальное число единиц. В рассматриваемом примере таким можно считать вектор P_1 , который определяет подмножество Y_1 . В данном случае $Y_1 = \{y_1, y_3, y_5, y_8, y_{10}\}$. Подмножество Y_1 не включает в себя микрооперации $Y^2 = Y^1 \setminus Y_1$, где $Y^1 = Y$, которые должны быть покрыты за счёт других векторов. Вычёркивая из матрицы P строку 1 и столбцы, которым соответствуют единицы в строке 1, получаем новую мат-

$$\begin{array}{c} \begin{array}{cccccc} 2 & 4 & 6 & 7 & 9 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{l} 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array}$$

рицу P . В данном случае $P^2 =$

Выделяя в множестве P^2 вектор P_i^2 с максимальным числом единиц, определяем подмножество Y_2 . В данном случае максимальным является вектор P_8^2 , который порождает подмножество $Y_2 = \{y_2, y_6, y_7\}$. Применяя указанную процедуру к множествам $Y^{i+1} = Y^i \setminus Y_i$ ($i = 1, 2, \dots, K$) и соответствующим им матрицам P^{i+1} до получения пустого множества Y^{K+1} , определим подмножества $Y_1, Y_2, \dots, Y_K$. Для рассматриваемого примера имеем

$$P^3 = \begin{matrix} & \begin{matrix} 4 & 9 \end{matrix} \\ \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} & \begin{matrix} 2 \\ 5 \end{matrix} \end{matrix}$$

откуда $Y_3 = \{y_4\}$ и $Y_4 = \{y_9\}$.

В итоге рассмотренный метод нахождения максимальных совместимых подмножеств позволил распределить микрооперации следующим образом:

$$Y_1 = \{y_1, y_3, y_5, y_8, y_{10}\}; Y_2 = \{y_2, y_6, y_7\}; Y_3 = \{y_4\}; Y_4 = \{y_9\}.$$

При таком распределении операционная часть микрокоманды также имеет длину $L = 3+2+1+1 = 7$ двоичных разрядов. Экономия оказывается существенной при большом числе микроопераций (порядка 10^2), но одновременно с этим резко возрастает трудоёмкость алгоритма нахождения максимальных совместимых подмножеств микроопераций.

Для задачи распределения микроопераций исходной является матрица совместимости S , которая может формироваться на основе различных предпосылок. Матрица S может составляться исходя из функциональных свойств микроопераций с учётом структурных ограничений на совместимость, налагаемых операционным автоматом. В этом случае признаки совместимости устанавливаются между каждой парой микроопераций y_i и $y_j \in Y$. При этом предполагается, что может возникнуть необходимость в совместном использовании любой пары микроопераций y_i и y_j .

Другой подход к формированию матрицы S основывается на анализе микропрограмм, в микрокомандах которых перечисляются микрооперации, используемые совместно. В матрице совместимости элементу s_{ij} присваивается значение 1 только в том случае, ко-

гда микрооперации y_i и y_j используются в одной микрокоманде. В результате этого совместимые микрооперации y_i и y_j , но не используемые совместно в микропрограмме, отмечаются признаком несовместимости $s_{ij} = 0$. Такой подход приводит к уменьшению количества единиц в матрице S , в результате чего большее число микроопераций рассматривается в качестве несовместимых и объединяется в одно операционное поле. Следовательно, формирование матрицы S на основе списка микрокоманд позволяет сократить длину операционной части микрокоманды, но при этом теряется «универсальность» микрокоманды, поскольку изменения, вносимые в микропрограмму, могут породить новые функциональные операторы, реализуемые за несколько тактов. Если матрица S формируется на основе функциональных свойств микроопераций, то любой новый функциональный оператор микропрограммы реализуется за один такт.

7.4.2. Кодирование наборов микроопераций

Управляющие сигналы $Y = \{y_1, y_2, \dots, y_M\}$ можно формировать в П-автомате следующим образом.

Пусть автомат генерирует N различных микрокоманд $K = \{K_1, K_2, \dots, K_N\}$, состоящих из наборов $K_i = \{y_\alpha, y_\beta, \dots, y_\omega\}$ совместимых микроопераций. Микрокоманды K_i и K_j различаются наборами входящих в них микроопераций. Состав наборов можно описать булевой матрицей

$$K = \begin{bmatrix} k_{11} & k_{12} & k_{1M} \\ k_{21} & k_{22} & k_{2M} \\ \dots & \dots & \dots \\ k_{N1} & k_{N2} & k_{NM} \end{bmatrix}, \quad (7.3)$$

i -я строка которой описывает набор K_i , и элементы k_{ij} принимают следующие значения: $k_{ij} = 1$, если $y_i \in K_j$, и $k_{ij} = 0$ в противном случае.

Номера наборов $K_1, \dots, K_N$ закодируем n -разрядными двоичными позиционными кодами K , которые будут зафиксированы в опе-

рационной части управляющего слова П-автомата. В таком случае управляющие сигналы Y можно формировать по схеме, оказанной на рис. 7.5. Микрокоманда, выбранная из постоянной памяти П-автомата, поступает на регистр микрокоманды РМК. Для преобразования множества кодов наборов K в множество управляющих сигналов Y используется дополнительная постоянная память ёмкостью N и разрядности M . Слова $K[1]$, $K[2]$, ..., $K[N]$ представляют значения строк 1, 2, ..., N матрицы (7.3). Код набора K рассматривается в качестве адреса набора микрокоманды дополнительной постоянной памяти (РАН – регистр адреса набора).

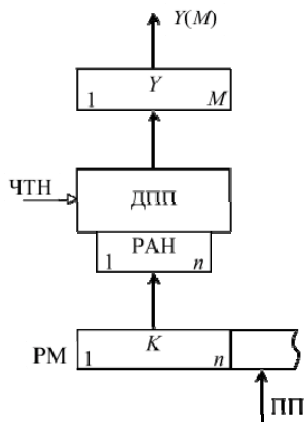


Рис. 7.5. Преобразование кодов микрокоманд с помощью дополнительной постоянной памяти наборов ДПП

При поступлении управляющего сигнала чтения набора ЧТН в ДПП инициируется операция чтения $PY := ДПП[РАН]$, в результате чего на регистр микроопераций выбирается слово. Набор единиц в разрядах $\alpha, \beta, \dots, \omega$, со слова PY порождает набор управляющих сигналов $y_\alpha, y_\beta, \dots, y_\omega$, возбуждающих соответствующие микрооперации.

Целесообразность использования указанного способа формирования управляющих сигналов определяется затратами оборудования и времени в схеме на рис. 7.5.

Затраты оборудования, необходимого для формирования сигналов Y , зависят от класса микропрограмм, реализуемых П-автоматом. Пусть микропрограмма состоит из P микрокоманд. Для кодирования в микрокоманде наборов микроопераций требуется не менее $n = \log_2 N$ разрядов, где N — число различных наборов.

Если для кодирования наборов микроопераций используется способ распределения микроопераций Y по полям $Y_1, Y_2, \dots, Y_K$, то, используя предположение о равномерном распределении микроопераций по полям, можно считать, что длина операционной части микрокоманды будет не менее $m = K \log_2(M/K)$.

Длина операционной части микрокоманды уменьшается, если $n < m$, т. е. если $(M/K)^K > N$.

Пусть k – длина адресной части микрокоманды. Тогда в результате уменьшения длины операционной части микрокоманды стоимость $C(P, k + m)$ основной постоянной памяти ёмкостью $P(k + m)$ -разрядных слов уменьшается до $C(P, k + n)$.

Использование способа кодирования наборов микроопераций требует введения в состав П-автомата дополнительной постоянной памяти, ёмкостью N M -разрядных слов с ценой $C(N, M)$. Если исходить из критерия экономии оборудования, то способ кодирования наборов микроопераций целесообразен, если

$$C(P, k + m) - C(P, k + n) > C(N, M). \quad (7.4)$$

Когда разрядность постоянной памяти имеет порядок от 10^1 до 10^2 , стоимость памяти примерно пропорциональна разрядности. С учётом этого формулу (7.4) можно представить в виде:

$$\frac{m - n}{k + m} C(P, k + m) > C(N, M)$$

Способ кодирования наборов микроопераций приводит к увеличению длительности такта на время, необходимое для чтения слова из дополнительной памяти. Затраты времени на формирование управляющих сигналов определяются в основном временем обращения к постоянной памяти, поэтому можно считать, что схема на рис. 7.5 увеличивает время формирования управляющих сигналов примерно в 2 раза.

7.5. Адресация микрокоманд

В П-автоматах микрокоманды выделяются своими адресами, определяющими номера ячеек постоянной памяти, в которых размещаются адресуемые микрокоманды.

Способ адресации микрокоманд задаёт правило определения адреса следующей микрокоманды. Используются два основных способа адресации: принудительная (с двумя принудительными адресами и с одним принудительным адресом) и естественная.

7.5.1. Адресация микрокоманд с двумя принудительными адресами

Адресация микрокоманд с двумя принудительными адресациями сводится к указанию в каждой микрокоманде обоих адресов следующей микрокоманды в явном виде. Этот способ реализован в управляющем автомате на рис. 7.4 по структуре микрокомандами на рис. 7.1. Адрес следующей микрокоманды определяется в зависимости от кода логического условия КЛУ и значения соответствующего логического условия x_i либо полем A_1 , либо полем A_2 , как определено в п. 7.3. Такая адресация строго детерминирована, легко поддается верификации и менее всего подвержена привнесению ошибки при формировании микропрограммы реализации алгоритма работы управляющего автомата.

Пример формирования микропрограммы с двумя принудительными адресами показан на рис. 7.6. Здесь и далее для наглядности и удобства понимания кодировка микроопераций сделана без учёта распределения по полям микроопераций с кодированием микропрограммы в восьмеричной системе счисления: поля кодов микроопераций – по два символа, поле кода логического условия – два символа, поля адресов следующей выполняемой микрокоманды – по три символа.

Для приведённого примера длина микрокоманды составила $k = 36$ бит (12 символов $\times 3$ бита) при начальном адресе загрузки микропрограммы $A_0 = 100$. Объём занимаемой ёмкости постоянной памяти при данном способе адресации микрокоманд для приведённой на рисунке конфигурации граф-схемы алгоритма составил $E = 324$ бита (9 микрокоманд $\times 36$ бит).

7.5.2 Адресация микрокоманд с одним принудительным адресом

С целью сокращения длины адресной части микрокоманды для формирования адреса следующей микрокоманды отводится единственное поле адреса перехода A_p . Если поле кода логического условия КЛУ = 0, то значение A_p , *безусловно*, определяет адрес следующей микрокоманды.

Кодировка микроопераций и логических условий

y_1	01	x_1	01
y_2	02	x_2	02
y_3	03	x_3	03
y_4	04	y_F	77

Кодировка микропрограммы

A_i	Y_i	Y_2	K/UY	A_i	A_2
100	00	00	01	101	102
101	01	02	02	103	104
102	03	02	03	105	106
103	00	02	03	105	106
104	01	00	00	101	000
105	03	02	01	107	101
106	00	00	01	107	101
107	04	00	00	110	000
110	77	00	00	000	000

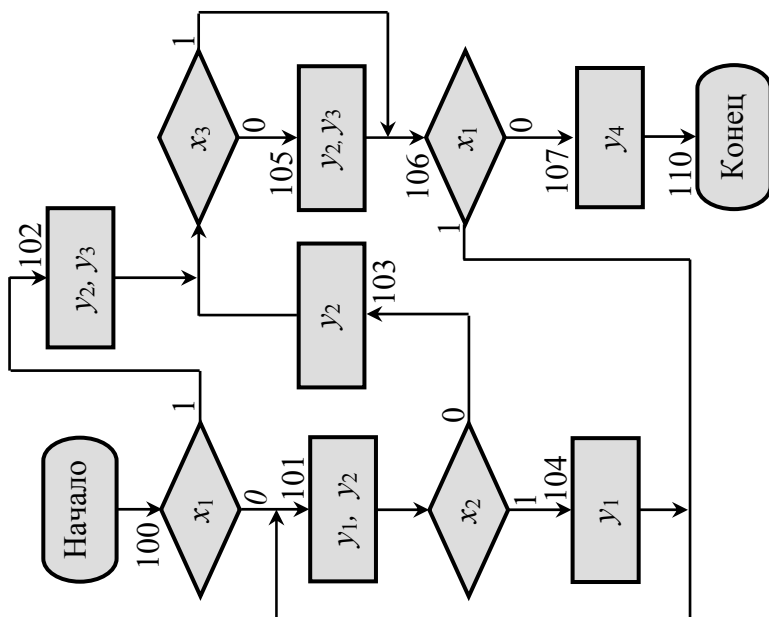
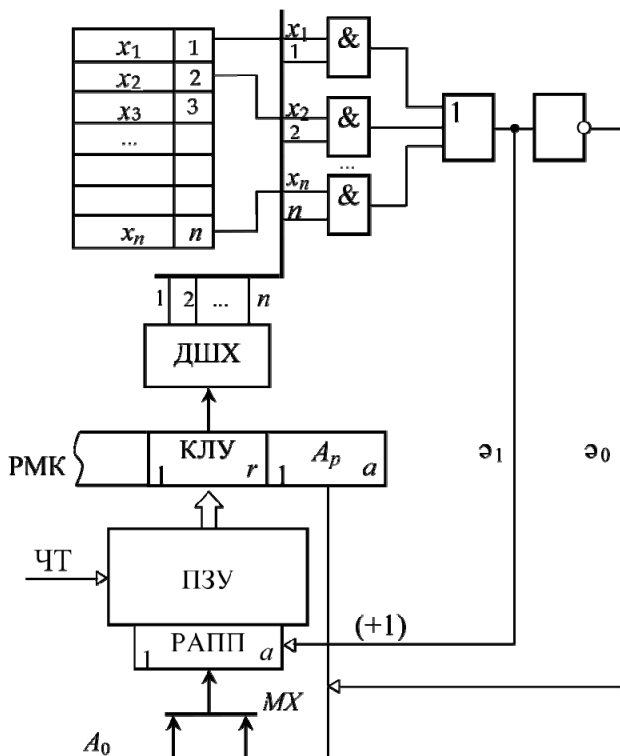


Рис. 7.6. Пример формирования микропрограммы с двумя принудительными адресами

Если поле кода логического условия не нулевое, то адрес следующей микрокоманды реализуется *условным переходом*: если значение проверяемого логического условия $x_i = 0$, то РАПП := A_p , если $x_i = 1$, то второй адрес перехода определяется функциональной зависимостью от адресов, явно зафиксированных в структуре автомата при выполнении микрокоманды. Здесь возможны два равноправных варианта: либо адрес следующей микрокоманды определяется инкрементом адреса текущей микрокоманды, хранящегося в регистре адреса постоянной памяти с функцией инкрементного счётчика $(A_T + 1)$, либо инкрементом адреса перехода $A_p + 1$, записанного в регистре микрокоманды.



Введение режима счёта в схему формирования адреса микрокоманды уменьшает быстродействие П-автомата, поскольку длительность такта должна быть увеличена на время выполнения микрооперации счёта в a -разрядном счётчике РАПП.

Порядок формирования адреса следующей микрокоманды с инкрементом адреса перехода может быть реализован схемой на рис. 7.8 без использования режима счёта.

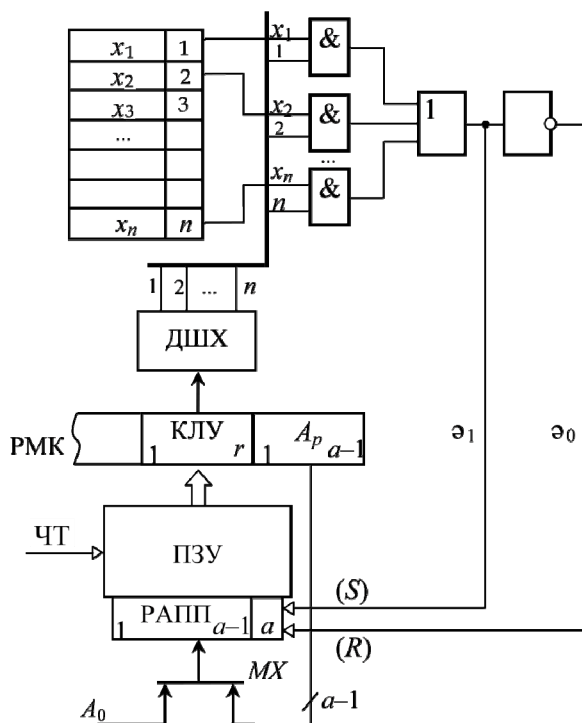


Рис. 7.8. Формирование адреса микрокоманды с инкрементом адреса перехода

При таком подходе регистр адреса постоянной памяти функционирует только в режиме хранения с укороченной на один бит длиной адреса, что является позитивным моментом наряду с отсутствием стробирования шины передачи адреса перехода. Поэтому адрес перехода может быть только чётным с установкой младшего бита РАПП в ноль сигналом ε_0 (R) или в единицу сигналом ε_1 (S).

Существует и негативная сторона рассмотренного способа формирования адреса следующей микрокоманды с инкрементом адреса перехода, заключающаяся в том, что конкретная конфигурация ГСА микропрограммы может привести к фрагментации постоянной памяти, особенно при непрерывной последовательности операторных вершин, когда невозможно использовать адреса с нечётными номерами.

На рис. 7.9 показан пример формирования адреса микрокоманды с инкрементом текущего адреса. Для приведённого примера длина микрокоманды составила $k = 27$ бит (9 символов $\times 3$ бита) при начальном адресе загрузки микропрограммы $A_0 = 200$. Объём занимаемой ёмкости постоянной памяти при данном способе адресации микрокоманд для той же конфигурации ГСА составил $E = 297$ бит (11 микрокоманд $\times 27$ бит).

На рис. 7.10 показан пример формирования адреса микрокоманды с инкрементом адреса перехода по рассмотренному способу. Для приведённого примера длина микрокоманды составила $k = 26$ бит (9 символов $\times 3$ бита – 1 младший бит адреса) при начальном адресе загрузки микропрограммы $A_0 = 300$. Объём занимаемой ёмкости постоянной памяти при данном способе адресации микрокоманд для той же конфигурации ГСА составил $E = 286$ бит (11 микрокоманд $\times 26$ бит). Следует отметить, что невозможность использования адреса $A = 301$ привела к фрагментации постоянной памяти.

7.5.3. Естественная адресация микрокоманд

Предпосылки дальнейшего сокращения длины микрокоманды заключаются в следующем. Пусть в алгоритме функционирования управляющего автомата присутствуют цепочки последовательно выполняемых микрокоманд без проверки логических условий, что встречается не так уж и редко. Это приводит к нерациональному, неэффективному использованию поля кода логических условий и адресного поля, так как при такой ситуации заполнять свободное пространство линейной структуры постоянной памяти автомата наиболее естественно инкрементом текущего адреса микрокоманды без его явного указания в адресном поле (рис. 7.11, а).

Кодировка микроопераций и логических условий

y_1	01	x_1	01
y_2	02	x_2	02
y_3	03	x_3	03
y_4	04	y_F	77

Кодировка микропрограммы

A_t	Y_1	Y_2	КЛУ	A_p
200	00	00	01	203
201	03	02	03	207
202	00	00	01	211
203	01	02	02	205
204	01	00	00	203
205	00	02	03	207
206	00	00	00	202
207	03	02	01	211
210	00	00	00	203
211	00	04	00	212
212	77	00	00	000

jump

jump

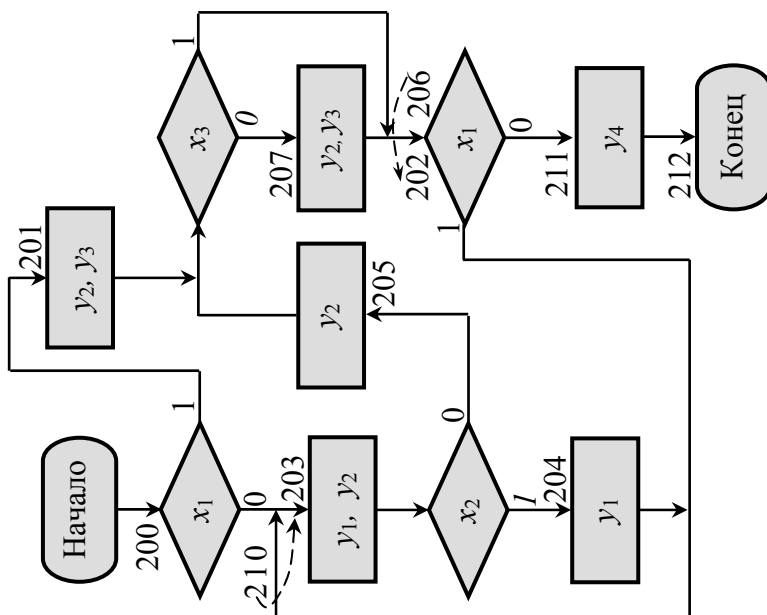


Рис. 7.9. Пример формирования микропрограммы с инкрементом текущего адреса

Кодировка микроопераций и логических условий

y_1	y_2	y_3	y_4	x_1	x_2	x_3	y_F
01	02	03	04	01	02	03	77

Кодировка микропрограммы

A_t	Y_1	Y_2	КЛЮ	A_p	$jump$
300	00	00	01	302	
301	$\times \times$	$\times \times$	$\times \times$	$\times \times \times$	
302	01	02	02	304	
303	03	02	03	306	
304	00	02	03	306	
305	01	00	00	302	
306	03	02	01	310	
307	00	00	01	310	
310	00	04	00	312	
311	00	00	00	302	
312	77	00	00	000	

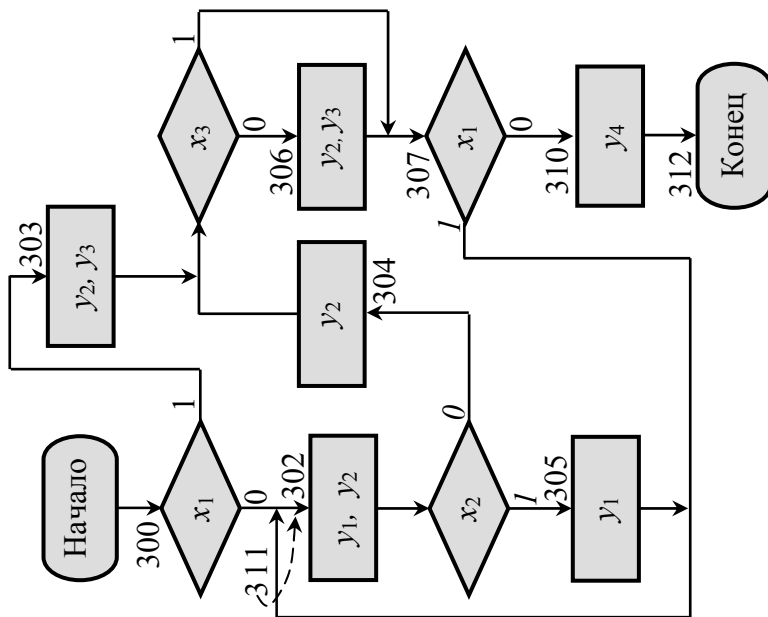


Рис. 7.10. Пример формирования микропрограммы с инкрементом адреса перехода

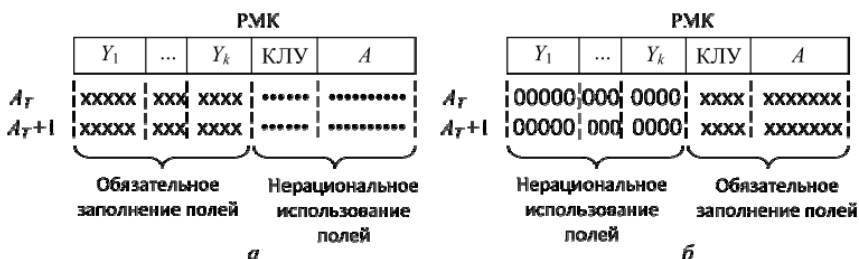


Рис. 7.11. Предпосылки естественной адресации микрокоманд

Вторая ситуация возникает тогда, когда алгоритм функционирования управляющего автомата содержит последовательную цепочку проверки логических условий (рис. 7.11, б). Эта ситуация приводит к не рациональному, не эффективному использованию операционных полей микрокоманды. Поэтому при естественной адресации адрес следующей микрокоманды принимается равным увеличенному на единицу адресу текущей микрокоманды $A_T + 1$ ($РАПП:=РАПП+1$).

При естественной адресации отпадает необходимость во введении адресного поля в каждую микрокоманду (и поля кода логического условия) или в необходимости операционной части микрокоманды. Если микрокоманды следуют в естественном порядке, то процесс адресации реализуется регистром адреса постоянной памяти в режиме счётчика, состояние которого увеличивается на единицу после чтения очередной микрокоманды. Следовательно, микрокоманды, которые задают функциональные преобразования, состоящие из набора микроопераций, могут содержать только операционную часть, представляемую полями $Y_1, Y_2, \dots, Y_k$.

После выполнения микрокоманды с адресом A_T может возникнуть необходимость в переходе к микрокоманде с адресом, отличным от инкремента текущего адреса. Переход может быть безусловным или зависеть от текущего значения проверяемого логического условия x_i . Реализация условных переходов такова: если отсутствует проверка логического условия (безусловный переход) или проверяемое логическое условие $x_i = 0$, то следующей выполняется микрокоманда, адрес которой указан в адресном поле текущей микрокоманды; если $x_i = 1$, то следующей выполняется микрокоманда с инкре-

ментом текущего адреса ($A_T + 1$). Для реализации условных переходов в микрокоманду вводится адресная часть, состоящая из полей кода логического условия КЛУ и адреса перехода A_p .

При естественной адресации обычно используются микрокоманды двух типов: операционные и управляющие.

Операционная микрокоманда задаёт набор микроопераций $Y_1, Y_2, \dots, Y_K$ и неявно полагает адрес следующей микрокоманды равным ($A_T + 1$).

Управляющие микрокоманды используются для изменения естественного порядка следования микрокоманд, что сводится к выполнению безусловных и условных переходов. Управляющая микрокоманда содержит поле КЛУ, определяющее код проверяемого логического условия, и поле адреса перехода A_p , определяющее адрес следующей микрокоманды. Если поле КЛУ = 0, то адрес следующей микрокоманды, безусловно, равен A_p . Для выделения операционных и управляющих микрокоманд в управляющем слове вводится одноразрядное поле признака d , определяющего тип микрокоманды: если $d = 0$, то микрокоманда является операционной; если $d = 1$, то управляющей. Возможная структура управляющих слов, построенных по указанному способу, изображена на рис. 7.12.

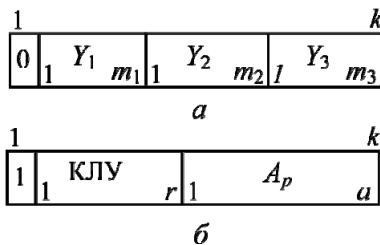
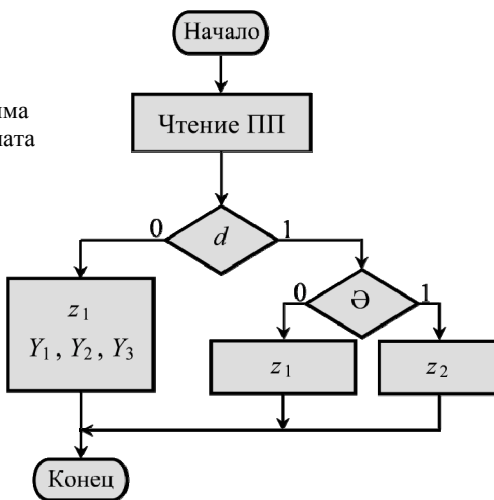


Рис. 7.12. Структура операционной (а) и управляющей (б) микрокоманд

П-автомат, работающий с микрокомандами, показанными на рис. 7.12, строится по схеме на рис. 7.13. Дешифраторы ДШ1, ДШ2, ДШ3, на выходе которых формируются управляющие сигналы, стробируются инверсным сигналом d при выполнении операционной микрокоманды.

Дешифратор ДШХ стробируется сигналом d при обработке управляющей микрокоманды. Адрес текущей микрокоманды хранится и преобразуется на регистре адреса постоянной памяти с режимом счётчика РАПП, с которым связаны микрооперации z_1 : РАПП:=РАПП+1 и z_2 : РАПП:= A_p .

Рис. 7.14. Микропрограмма функционирования автомата



При использовании двух типов микрокоманд увеличиваются затраты времени на реализацию микропрограмм, поскольку такты времени отводятся не только для выполнения функциональных операторов, но и для операторов перехода (управления). Для уменьшения затрат времени в одном управляющем слове совмещаются операционная и адресная микрокоманды. При этом последовательность из функционального оператора и оператора перехода выполняется за один такт, но, как отмечалось ранее, при выполнении последовательности из двух функциональных операторов не используется адресная часть, по крайней мере, в одной микрокоманде. В случае выполнения последовательности операторов перехода не используется операционная часть большинства микрокоманд. Таким образом, стремление повысить производительность управляющего автомата приводит к непроизводительному использованию ёмкости постоянной памяти, в результате чего увеличиваются затраты оборудования на реализацию микропрограммы.

На рис. 7.15 показан пример формирования адреса микрокоманды с естественной адресацией. Для приведённого примера длина микрокоманды составила $k = 16$ бит (5 символов $\times$ 3 бита + 1 бит признака d) при начальном адресе загрузки микропрограммы $A_0 = 400$. Объём занимаемой ёмкости постоянной памяти при данном способе адресации микрокоманд для той же конфигурации граф-схемы алгоритма составил $E = 224$ бита (14 микрокоманд $\times$ 16 бит).

Кодировка микроопераций и логических условий

y_1	001	x_1	01
y_2	02	x_2	02
y_3	003	x_3	03
y_4	04	y_F	77

Кодировка микропрограммы

A_r	d	Y_1	Y_2
A_p	d	K/IV	A_p
400	1	01	404
401	0	02	003
402	1	03	410
403	1	01	412
404	0	02	001
405	1	02	414
406	0	00	001
407	1	00	404
410	0	02	003
411	1	00	403
412	0	04	000
413	0	77	000
414	0	02	000
415	1	00	402

jump

jump

jump

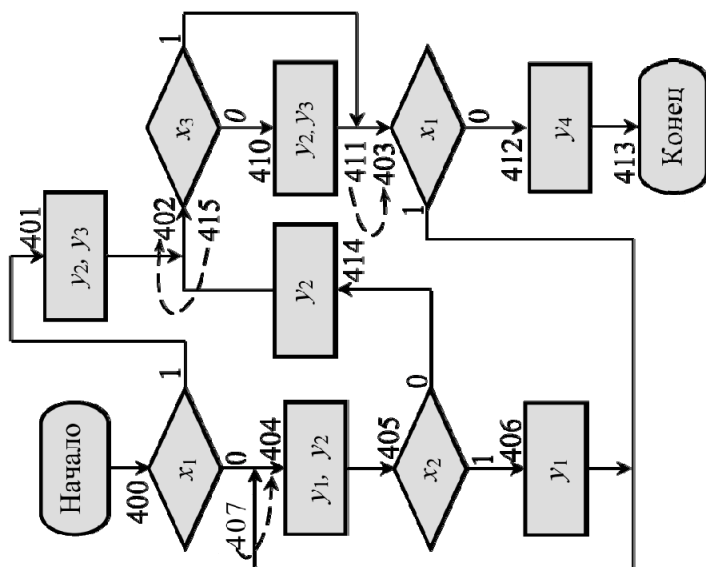


Рис. 7.15. Пример формирования микропрограммы с естественной адресацией микрокоманд

7.6. Сегментация постоянной памяти

Для уменьшения длины адресной части микрокоманды используется способ сегментации постоянной памяти. Постоянная память разделяется на сегменты, состоящие из 2^q соседних ячеек. При этом адрес ячейки постоянной памяти разделяется на два поля: $S[1:r]$ и $A[1:q]$. Поле $S(r)$ определяет адрес сегмента и поле $A(q)$ – адрес ячейки в сегменте S .

Для адресации микрокоманд сначала специальной микрокомандой устанавливается адрес сегмента S . В последующих микрокомандах указывается адрес ячейки A в сегменте. Поскольку адрес A является лишь частью полного a -разрядного адреса ячейки постоянной памяти, то адресная часть микрокоманды сокращается на $k(a - q)$ разрядов, где k – количество адресов в микрокоманде.

В случае принудительной адресации обработка сегментированных адресов реализуется схемой на рис. 7.16, *а*. По аналогии с микрокомандами на рис. 7.1 адресная часть микрокоманды разделена на поля X , A_1 и A_2 . Для хранения адреса сегмента в схеме на рис. 7.16, *а* используется регистр S .

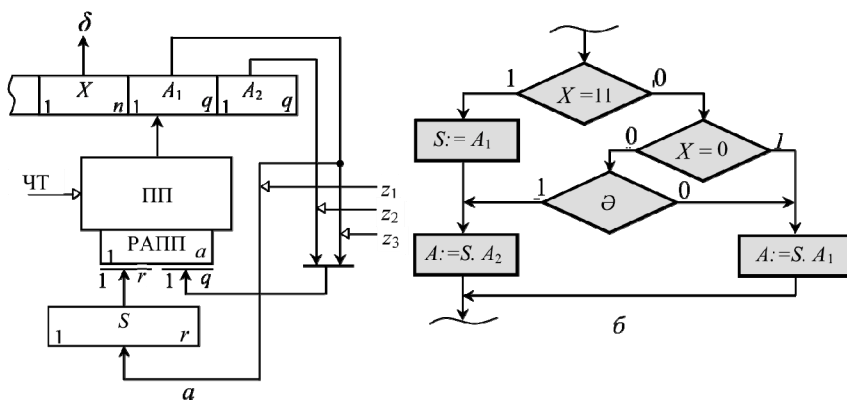


Рис. 7.16. Обработка сегментированных адресов при принудительной адресации

Допустим, что код $X = 11...1$ используется в качестве признака δ перехода к другому сегменту. Если $\delta = 1$, то микрокоманда изменяет старшие r разрядов адреса, определяющие адрес сегмента. Ес-

ли $\delta = 0$, то микрокоманда изменяет только младшие q разрядов адреса, определяющие адрес ячейки в сегменте S . Порядок вычисления адреса A следующей микрокоманды описан фрагментом микропрограммы на рис. 7.16, б. В случае естественной адресации обработка сегментированных адресов реализуется по схеме рисунка 7.17, а.

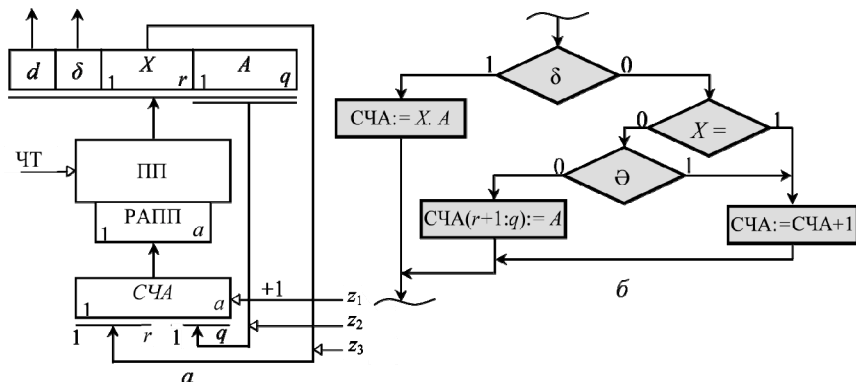


Рис. 7.17. Обработка сегментированных адресов при естественной адресации

В управляющей микрокоманде ($d = 1$) выделены три поля: δ – одноразрядный признак изменения сегмента; X – код проверяемого логического условия; A – адрес ячейки сегмента. В данном варианте принято, что адрес сегмента размещается в поле X . Микрокоманда обрабатывается в порядке, определяемом рисунком 7.17, б.

При сегментации постоянной памяти уменьшается длина основных микрокоманд, но одновременно с этим в микропрограмму включаются дополнительные микрокоманды, необходимые для передачи управления между сегментами. Величина экономии памяти, используемой для хранения основных микрокоманд, и ёмкость памяти, потребляемая дополнительными микрокомандами, зависят от размера сегмента.

Если постоянная память разделяется на сегменты размером S ячеек, то для указания адресов в основных микрокомандах используется $\log_2 S$ бит. В таком случае ёмкость постоянной памяти, необходимая для представления адресов в N основных микрокомандах, составит $E_1 = N \log_2 S$ бит.

Определим затраты памяти для хранения микрокоманд, используемых в целях передачи управления между сегментами микропрограммы. Пусть микропрограмма состоит из N основных микрокоманд, q -я часть которых предписывает переходы, нарушающие естественный порядок следования микрокоманд, т.е. остальные $(1 - q)N$ микрокоманды выполняются в естественном порядке. С вероятностью r переход выполняется вперёд, в сторону микрокоманд с большими адресами, и с вероятностью $(1 - r)$ – назад, в сторону микрокоманд с меньшими адресами. Разность $l = |\alpha - \beta|$, где α и β – адреса текущей и следующей микрокоманды, назовём длиной перехода и положим, что длины перехода распределены по геометрическому закону

$$p(l) = (1 - p)^{l-1} p \quad (l = 1, 2, \dots)$$

с математическим ожиданием $L = 1/p$, определяющим среднюю длину перехода. Заметим, что непрерывным аналогом геометрического распределения является экспоненциальное распределение. Вероятность того, что длина перехода l превысит k , будет

$$P(p \geq k) = \sum_{i=k}^{\infty} (1 - p)^{i-1} p = (1 - p)^{k-1} \quad (k = 1, 2, \dots). \quad (7.5)$$

Вероятность того, что при выполнении j -ой микрокоманды возникнет необходимость в переходе за пределы сегмента, которому принадлежит эта микрокоманда, будет

$$P_j = q[rP(l \geq S - j + 1) + (1 - r)P(l \geq j)] \quad (j = 1, 2, \dots, S). \quad (7.6)$$

Здесь q определяет вероятность перехода, нарушающего естественный порядок следования микрокоманд, первый член суммы – вероятность перехода в последующие сегменты и второй член – вероятность перехода в предшествующие сегменты микропрограммы. Допустим, что микрокоманды сегмента используются с одинаковой вероятностью $1/S$. Тогда из (7.5) и (7.6) следует, что вероятность перехода за пределы сегмента при выполнении любой микрокоманды, размещенной в этом сегменте, равна

$$Q = \sum_{j=1}^S \frac{1}{S} P_j = \frac{q}{S} \sum_{j=1}^S [r(1 - p)^{S-j} + (1 - r)(1 - p)^{j-1}] =$$

$$= \frac{q}{S} \sum_{j=1}^S (1-p)^{j-1} = \frac{q}{pS} \left[1 - (1-p)^S \right].$$

Подставляя $p = 1/L$, получаем

$$Q = \frac{qL}{S} \left[1 - \left(1 - \frac{1}{L} \right)^S \right]. \quad (7.7)$$

Заметим, что при геометрическом распределении длин переходов значение Q инвариантно в отношении вероятности r , определяющей возможные направления переходов. Значение Q , определяя вероятность перехода за пределы сегмента, задаёт одновременно вероятность того, что к основной микрокоманде должна быть приписана дополнительная микрокоманда, организующая переход в другой сегмент. Таким образом, Q -я часть микрокоманд предписывает переходы в другие сегменты, для программирования которых требуется QN дополнительных микрокоманд.

Если микрокоманда представляется K двоичными разрядами, то для хранения в постоянной памяти дополнительных микрокоманд требуется емкость $E_2 = KNQ$ бит, а всего для адресации используется

$$E = E_1 + E_2 = N \log_2 S + \frac{NqKL}{S} \left[1 - \left(1 - \frac{1}{L} \right)^S \right] \text{ бит.} \quad (7.8)$$

Оптимальное значение S , при котором для адресации используется минимальное количество информации, определятся корнем уравнения

$$\frac{dE}{dS} = \frac{N}{S} \left[\frac{1}{\ln 2} - \frac{qKL}{S} (1 - a^S) - qKL a^S \ln a \right] = 0, \quad (7.9)$$

где $a = 1 - 1/L$.

Значение S , определяемое выражением (7.9), можно реализовать лишь приближённо, поскольку всегда должно выполняться правило $S = 2^x$, где x – целое число. Учитывая это, можно отказаться от решения трансцендентного уравнения (7.9) и вычислять приближённое значение S . В большинстве случаев $L < 5$ и $S > 8$, что позволяет пренебречь значением $(1 - 1/L)^S$ по сравнению с единицей и привести формулу (7.8) к более простому виду:

$$E = N \left(\log_2 S + \frac{qKL}{S} \right).$$

Основываясь на этом выражении, оптимальное значение S можно вычислить как

$$\begin{aligned} \frac{dE}{dS} &= N \left(\frac{1}{S \ln 2} - \frac{qKL}{S^2} \right) = 0; \\ S &= qKL \ln 2 \approx 0,69qKL. \end{aligned} \quad (7.10)$$

Например, если длина микрокоманды $K = 40$ двоичных разрядов, вероятность перехода $q = 0,4$ и средняя длина перехода $L = 5$, то $0,69qKL = 55,2$, и длину сегмента можно принять равной $S = 2^6 = 64$.

Сегментация постоянной памяти позволяет экономить оборудование, но из-за введения в микропрограмму дополнительных микрокоманд увеличивается время выполнения микропрограмм. Если микропрограмма выполняется в среднем за t тактов, то при сегментации время выполнения микропрограмм увеличивается в среднем до $(1+Q)t$ тактов, где Q – вероятность появления дополнительной микрокоманды, определяемая (7.7). Основываясь на вышеизложенных допущениях, можно упростить формулу (7.7) и приближённо считать, что время выполнения сегментированной микропрограммы t_s определяется значением

$$t_s = \left(1 + \frac{qL}{S} \right) t.$$

Путём рационального размещения микропрограммы по сегментам можно уменьшить число переходов между сегментами по сравнению со значением, определяемым (7.7). В результате этого уменьшается ёмкость постоянной памяти и затраты времени на выполнение микропрограммы.

7.7. Сегментация микропрограмм

Сегментация адресов постоянной памяти является эффективным средством экономии оборудования в управляющем автомате, но при этом за счёт введения в микропрограмму дополнительных микрокоманд, организующих переходы между сегментами, увели-

чиваются затраты времени на выполнение микропрограмм, то есть снижается быстродействие операционного устройства. Время выполнения микропрограммы можно уменьшить, если распределять микрокоманды по сегментам постоянной памяти таким образом, чтобы число переходов между сегментами в процессе реализации микропрограммы было минимальным. Процесс распределения микрокоманд по сегментам постоянной памяти называется *сегментацией* микропрограммы.

Формально задачу сегментации микропрограмм можно определить следующим образом. Микропрограмма рассматривается как граф $G(V, D)$ с множеством вершин $V = \{1, 2, \dots, N\}$, каждая из которых соответствует микрокоманде с условным номером $1, 2, \dots, N$ и множеством дуг $D = \{(i, j)\}$ ($i, j = 1, 2, \dots, N$), определяющим совокупность переходов между каждой парой микрокоманд i и j . Дугам (i, j) ставится в соответствие цена c_{ij} , равная среднему числу переходов между микрокомандами i и j при реализации микропрограммы. Вершины V графа должны быть распределены по подмножествам $V_1, V_2, \dots, V_H$ каждое из которых, кроме последнего, должно включать в себя точно S элементов, таким образом, чтобы

$$\bigcup_{k=1}^H V_k = V, \quad (7.11)$$

$$V_k \cap V_l = \emptyset \quad (k, l = 1, 2, \dots, H; k \neq l) \quad (7.12)$$

и суммарная цена

$$C = \sum_{\substack{i \in V_k \\ j \in V_l}} c_{ij} \quad (k, l = 1, 2, \dots, H; k \neq l) \quad (7.13)$$

всех дуг, которые связывают вершины, распределённые в различные подмножества, была минимальной. Условие (7.11) определяет необходимость включения каждой вершины хотя бы в одно подмножество V_k , т. е. разбиение $V_1, V_2, \dots, V_H$ должно покрывать все N вершин графа. Условие (7.12) устанавливает, что каждая вершина может быть включена только в одно подмножество. Сумма C называется внешней ценой разбиения $V_1, V_2, \dots, V_H$ (или просто ценой разбиения) и определяет среднее число переходов между сегментами $V_1, V_2, \dots, V_H$ при реализации микропрограммы.

Наряду с данной задачей сегментации существует и ряд эквивалентных задач. Во-первых, в целях уменьшения цены разбиения можно допустить частичное использование ёмкости сегмента, то есть считать, что сегмент должен содержать не менее S_1 и не более S_2 элементов. Это допущение может привести к уменьшению числа переходов между сегментами, в результате чего сокращаются затраты времени на выполнение микропрограммы, но наряду с этим снижается степень использования ёмкости постоянной памяти. Во-вторых, вместо минимизации внешней цены (7.13) можно максимизировать внутреннюю цену разбиения $C_{\text{вн}} = \sum_{k=1}^H \sum_{i,j \in V_k} c_{ij}$, которая

равна суммарной цене дуг, связывающих вершины одного подмножества V_k ($k = 1, 2, \dots, H$). Поскольку полная цена всех дуг графа

$C_{\text{п}} = \sum_{i=1}^N \sum_{j=1}^N c_{ij}$ постоянна, то максимизация цены $C_{\text{вн}}$ эквивалентна

минимизации C , и наоборот. В дальнейшем рассматривается только основная задача сегментации микропрограммы, определяемая приведёнными условиями (7.11)–(7.13).

Точное решение задачи сегментации микропрограммы, состоящей из $N = S_H$ микрокоманд, может быть найдено перебором всех возможных вариантов разбиения, из которых выбирается разбиение

с минимальной ценой. Существует $\binom{N}{S}$ вариантов построения

подмножества V_1 , где $\binom{N}{S}$ – число сочетаний из N элементов по S ,

$\binom{N-S}{S}$ вариантов построения подмножества V_2 и так далее. Так

как порядок нумерации подмножеств несущественен, то данный способ решения задачи требует перебора следующего количества вариантов:

$$\frac{1}{H!} \binom{N}{S} \binom{N-S}{S} \binom{N-2S}{S} \dots \binom{S}{S}.$$

Даже для небольших значений N это выражение определяет очень большие числа, например, при $N = 40$ и $S = 4$ оно больше 10^{20} . Таким образом, процедура поиска точного решения задачи не является эффективной в вычислительном смысле, поскольку не может быть реализована практически.

Формально задача сегментации может быть сведена к задаче целочисленного линейного программирования с очень большим количеством ограничивающих выражений, многочисленность которых также исключает возможность получения решения.

Наиболее результативными являются эвристические методы, гарантирующие получение не оптимальных, но хороших решений за приемлемое время. Считается, что эвристическая процедура разбиения графа является приемлемой, если трудоёмкость вычислений возрастает не быстрее, чем квадрат числа вершин графа.

7.7.1 Алгоритм оптимизации парного разбиения

Простейшей является задача нахождения разбиения минимальной цены для графа, содержащего $N = 2n$ вершин, на два подграфа, каждый из которых содержит n вершин. Эта задача называется задачей парного разбиения и на её основе формулируются и решаются более сложные задачи, в том числе задача сегментации микропрограммы.

Метод решения задачи парного разбиения состоит в следующем. Множество вершин V произвольным образом делится на два непересекающихся подмножества A и B ($A \cup B = V$ и $A \cap B = \emptyset$) одинаковой мощности, определяющие начальное разбиение, которое имеет некоторую цену C .

Путем последовательных взаимных обменов элементами между множествами A и B можно уменьшить цену C . Когда никакими обменами невозможно снизить цену C , то последнее разбиение A и B считается локально оптимальным. Затем этот процесс можно повторить по отношению к другому начальному разбиению и получить любое желаемое количество локально оптимальных решений, из множества которых выбирается наилучшее.

Пусть A, B – любое произвольное парное разбиение множества V , и предположим, что A^* и B^* – парное разбиение минимальной цены. Ясно, что имеются подмножества $X \subset A$ и $Y \subset B$, содержащие одинаковое количество элементов, и такие, что взаимный обмен подмножествами X и Y между множествами A и B приводит к A^* и B^* :

$$A^* = (A \setminus X) \cup Y;$$

$$B^* = (B \setminus Y) \cup X.$$

Чтобы найти разбиения A^* и B^* минимальной цены, необходимо выявить подмножества X и Y , не прибегая к анализу всевозможных вариантов выбора X и Y . Приблизённо эти подмножества выявляются следующим образом.

Для каждой вершины $a \in A$ определяются внешняя цена E_a и внутренняя цена I_a :

$$E_a = \sum_{y \in B} c_{ay}; \quad I_a = \sum_{x \in A} c_{ax}.$$

Аналогично, для каждой вершины $b \in B$ определяются значения E_b и I_b . Разность между внешней и внутренней ценами вершины в данном разбиении равна

$$D_A = E_A - I_A;$$

$$D_B = E_B - I_B.$$

Рассмотрим два любых элемента $a \in A$ и $b \in B$. Если элементы a и b обмениваются между множествами A и B , то цена разбиения изменяется на

$$\delta_{ab} = D_a + D_b - 2c_{ab}.$$

Действительно, пусть Z – полная цена всех соединений между A и B , в которых не участвуют элементы a и b . Тогда цена C разбиения A и B равна

$$C = Z + E_a + E_b - c_{ab}.$$

Произведя обмен элементами a и b , получаем разбиение с новой ценой

$$C' = Z + I_a + I_b + c_{ab}.$$

Отсюда

$$\delta_{ab} = C - C' = D_a + D_b - 2c_{ab}.$$

Таким образом, для минимизации цены разбиения необходимо организовать взаимный обмен между теми элементами множеств A и B , для которых значение δ является положительным.

Вышеуказанные соображения позволяют построить следующий алгоритм минимизации цены парного разбиения графа.

1. Для всех элементов A и B вычисляются значения D .

2. Выбираются элементы $a_i \in A$ и $b_i \in B$, такие, что значение $\delta_{a_i b_i}$ положительно и максимально, то есть обмен элементами a_i и b_i приводит к максимальному эффекту при минимизации цены разбиения.

3. Для всех элементов a и b множеств $(A \setminus \{a_i\})$ и $(B \setminus \{b_i\})$ вычисляются новые значения D :

$$D'_a = D_a + 2c_{aa_i} - 2c_{ab_i}, \quad a \in (A \setminus \{a_i\});$$

$$D'_b = D_b + 2c_{bb_i} - 2c_{ba_i}, \quad b \in (B \setminus \{b_i\}).$$

4. Шаги 2 и 3 алгоритма повторяются ($i = 2, 3, \dots$) по отношению к множествам $(A \setminus \{a_i\})$ и $(B \setminus \{b_i\})$ с учётом новых значений D'_a и D'_b , в результате чего выбирается совокупность кандидатов на обмен

$$\{(a_1, b_1), (a_2, b_2), \dots, (a_k, b_k)\} \quad (k < n),$$

перестановка которых позволит уменьшить цену разбиения на величину

$$\Delta = \sum_{i=1}^k \delta_{a_i b_i}.$$

5. Подмножество элементов $X = \{a_1, a_2, \dots, a_k\}$ переносится в множество B и подмножество $Y = \{b_1, b_2, \dots, b_k\}$ – в множество A .

6. Новое разбиение A и B рассматривается как начальное и пункты (1–5) алгоритма повторяются до тех пор, пока $\Delta > 0$.

Трудоёмкость (количество выполняемых операций) алгоритма определения подмножеств X и Y пропорциональна $n^2 \log_2 n$ и разбиение с минимально возможной ценой находится за (2–4) обмена подмножествами X и Y .

Данный алгоритм позволяет найти лишь локально оптимальное решение, то есть решение, оптимальное для заданного начального

разбиения (A, B) . Другое начальное разбиение может привести к лучшему решению. В частности, можно использовать следующий способ порождения начальных разбиений. Первое начальное разбиение выбирается случайным образом, например, $A = \{1, 2, \dots, n\}$ и $B = \{n+1, n+2, \dots, 2n\}$. После применения алгоритма оптимизации следующее начальное разбиение строится на основе оптимизированного распределения. Для этого применением того же алгоритма множество A разбивается на минимально связанные подмножества A_1 и B_1 , а множество B – на подмножества B_1 и B_2 . В качестве новых начальных разбиений назначаются множества $A = A_1 \cup B_2$ и $B = B_1 \cup A_2$, и поиск глобально оптимального разбиения продолжается до тех пор, пока не будут исчерпаны ресурсы времени, отведённого на эту работу.

7.7.2. Алгоритм сегментации микропрограмм

При сегментации микропрограммы вершины V графа $G(V, D)$ распределяются между подмножествами $V_1, V_2, \dots, V_H$, каждое из которых, кроме последнего, содержит точно S элементов (микрокоманд).

Сегментация начинается с получения начального разбиения, в качестве которого можно использовать следующее произвольное разбиение:

$$\begin{aligned} V_1 &= \{1, 2, \dots, S\}; \\ V_2 &= \{S+1, S+2, \dots, 2S\}; \\ &\dots\dots\dots \\ V_H &= \{(H-1)S+1, (H-1)S+2, \dots, HS\}. \end{aligned}$$

Как показывает опыт, произвольные начальные разбиения приводят к значительному увеличению объёма работы, выполняемой в целях оптимизации разбиения. В связи с этим стремятся использовать не произвольные, а достаточно хорошие начальные разбиения. Обычно затраты труда на формирование хороших начальных разбиений окупаются за счёт снижения трудоёмкости процесса оптимизации.

Хорошие начальные разбиения N вершин V на H подмножеств можно получать следующими способами.

Применяя алгоритм оптимизации парного разбиения, множество V разделяют на два подмножества A и B с минимальной ценой C . Количество элементов в множестве A

$$n_A = \begin{cases} SH / 2 & \text{при четном } H; \\ S(H + 1) / 2 & \text{при нечетном } H \end{cases}$$

и остальные $n_B = N - n_A$ элементов размещаются в множестве B . В алгоритм оптимизации парного разбиения вводится дополнительное условие $k \leq \min(n_A, n_B)$, которое ограничивает число элементов, назначаемых для обмена, минимальным числом элементов в подмножестве, но процесс поиска пар для обмена продолжает охватывать всё множество элементов каждого подмножества. Затем алгоритм оптимизации парного разбиения применяется отдельно к подмножествам A и B , в результате чего каждое из них разделяется на два новых подмножества. Процесс продолжается до тех пор, пока не будет достигнут заданный размер подмножеств $V_1, V_2, \dots, V_H$, для чего потребуется не более $(H-1)$ применений алгоритма оптимизации.

Данному способу формирования начального распределения свойственно следующее противоречие. Первое применение алгоритма к множеству V порождает два максимально связанных подмножества A и B , имеющих наибольшую внутреннюю цену. Это свойство первого разбиения противоречит последующим этапам, на которых делается попытка разделить сильно связанные множества на слабо связанные подмножества. Поскольку это противоречие сохраняется и на каждой последующей паре шагов, данный способ может привести к относительно плохому начальному разбиению, но все же лучшему, чем совершенно произвольное.

Второй способ формирования начального разбиения сводится к разбиению множества V на подмножество V_1 , содержащее S элементов, и подмножество $(V \setminus V_1)$. Это разбиение улучшается алгоритмом оптимизации парного разбиения. Затем из оставшихся элементов $(V \setminus V_1)$ выделяется подмножество V_2 и так далее. Эта процедура продолжается до выделения всех H подмножеств, для чего алгоритм оптимизации парного разбиения применяется $(H-1)$ раз. В силу приближённости процедуры разбиения и оптимизации дан-

ный способ может привести к некачественному выбору первого подмножества V_1 , что повлияет на качество подмножества V_2 и так далее. Этот эффект особенно сильно сказывается при большом числе H подмножеств, что типично для случая сегментации микропрограмм.

Начальное разбиение $V_1, V_2, \dots, V_H$ улучшается путём применения алгоритма оптимизации к всевозможным парам V_i, V_j , число которых равно $\binom{N}{2}$. Попарная оптимальность разбиений является лишь необходимым условием глобальной оптимальности разбиения $V_1, V_2, \dots, V_H$.

При сегментации может возникнуть необходимость в многократном повторении цикла оптимизации, поскольку оптимизация разбиения в паре V_i, V_j может быть нарушена при оптимизации разбиения в паре V_k, V_l . Следовательно, качество разбиения зависит от объёма работы, выполненной в целях оптимизации: чем большее время затрачивается на оптимизацию, тем больше вероятность получения качественного разбиения. К тому же качество разбиения в большой мере зависит от качества начального разбиения, и поэтому представляется целесообразным оптимизировать несколько начальных разбиений, формируемых различными способами.

Контрольные вопросы и упражнения

1. Для управления операцией суммирования с использованием накапливающего сумматора, ГСА которого представлена на рис. 7.18, составьте микропрограммы с произвольной кодировкой:

- с формированием двух принудительных адресов выполнения следующей микрокоманды;
- с указанием одного принудительного адреса при использовании инкремента текущего адреса микрокоманды;
- с указанием одного принудительного адреса при использовании инкремента адреса перехода, указанного в микрокоманде;
- с использованием методики естественной адресации.

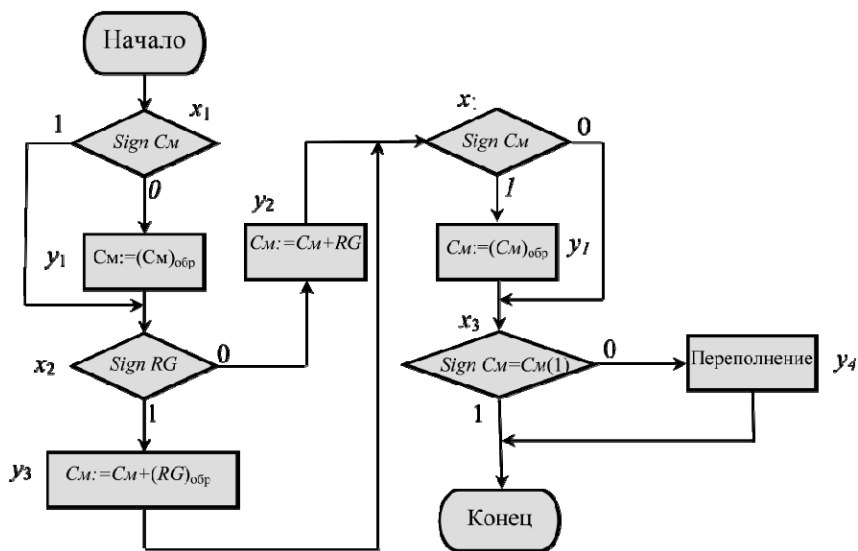


Рис. 7.18. ГСА накапливающего суммирования

Список использованной литературы

1. Ауфенкамп Д. и Хон Ф. Анализ последовательностных машин.// Математика. М.: Изд-во иностр. литер., 1959.
2. Баранов С.И. Синтез микропрограммных автоматов. Л.: Энергия, 1974.
3. Баранов С.И., Скляров В.А. Цифровые устройства на программируемых БИС с матричной структурой. М.: Радио и связь, 1986.
4. Брауэр В. Введение в теорию конечных автоматов. М.: Радио и связь, 1987.
5. Горбатов В.А., Горбатов А.В., Горбатова М.В. Теория автоматов. М.: АСТ: Астрель, 2008.
6. Горнец Н.Н. Организация ЭВМ и систем. М.: ИЦ «Академия», 2006.
7. Каган Б.М. Электронные вычислительные машины и системы. М.: Энергоатомиздат, 1991.
8. Корнеев В.В. Современные микропроцессоры. М.: НОЛИДЖ, 2000.
9. Лазарев В.Г., Пийль Е.И. Синтез управляющих автоматов. М.: Энергия, 1970.
10. Лысыков Б.Г. Цифровая и вычислительная техника. Мн.: УП Экоперспектива, 2002.
11. Майоров С.А., Новиков Г.И. Принципы организации цифровых машин. Л.: Машиностроение, 1974.
12. Миллер Р. Теория переключательных схем. Т.2. М.: Наука, 1971.
13. Поспелов А. Логические методы анализа и синтеза схем. М.: Энергия, 1974.
14. Проектирование цифровых устройств на программируемых логических интегральных схемах: Учебн. пособие / Бутаев М.М., Вашкевич Н.П., Гурин Е.И., Коннов Н.Н. Пенза: ПГТУ, 1996.
15. Столингс У. Структурная организация и архитектура компьютерных систем. М.: Вильямс, 2002.
16. Фридман А., Менон П.. Теория и проектирование переключательных схем. М.: Мир, 1978.
17. Цилькер Б.Я. Организация ЭВМ и систем. СПб.: Питер, 2004.

Валерий Васильевич Крушный

СИНТЕЗ ЦИФРОВЫХ
УПРАВЛЯЮЩИХ АВТОМАТОВ

Учебное пособие

Редактор *Е.Н. Кочубей*

Подписано в печать 15.11.2011. Формат 60х84 1/16.

Уч.-изд. л. 10,25. Печ. л. 10,25. Тираж 155 экз.

Изд. № 4/3. Заказ № 77.

Национальный исследовательский ядерный университет «МИФИ»
115409, Москва, Каширское ш., 31

ООО «Полиграфический комплекс «Курчатовский».
144000, Московская область, г. Электросталь, ул. Красная, д.42