Лабораторная работа №4
Конечные автоматы

Когда мы проектировали секундомер, у нас возникла необходимость выделить признак того, что в данный момент секундомер работает. Для этого мы создали специальный регистр для хранения этого признака и описали схему управления этим регистром. В других блоках секундомера мы могли проверить признак работы (содержание регистра) и, в зависимости от него, разрешить, запретить или изменить работу.

Похожий подход применяется и в других случаях, когда необходимо выделить разные режимы работы цифрового устройства или обеспечить последовательное выполнение некоторых операций.

Действительно, ведь цифровое устройство существует целиком в один момент времени. Тогда как возможно добиться от него последовательного выполнения каких-то действий?

Вариант только один – описать еще один блок, который будет управлять работой всей остальной частью цифрового устройства.

Блок, который используется для этого, называется «конечный автомат».

Существует целый раздел математики, посвященный автоматам (и в частности конечным автоматам), который, вполне ожидаемо, называется «теория автоматов».

Но, так как нас, прежде всего, интересует прикладное применение конечных автоматов, мы рассмотрим их с точки зрения цифровой схемотехники.

В цифровой схемотехнике, конечным автоматом называется цифровая схема, которая отслеживает текущее состояние и обеспечивает переход от одного состояния к другому в зависимости от входных воз-

действий.

Например, в случае секундомера, который мы проектировали в прошлой лабораторной работе, конечный автомат, который мог бы управлять им, переключался бы между двумя состояниями: «остановлен» и «работает».

Для удобства проектирования конечный автомат представляют в виде графа. Вершины графа - это состояния, а рёбра графа - это воздействия, которые приводят к изменению состояния. Этот граф называют графом переходов конечного автомата.

Обычно граф переходов выглядит подобным образом:

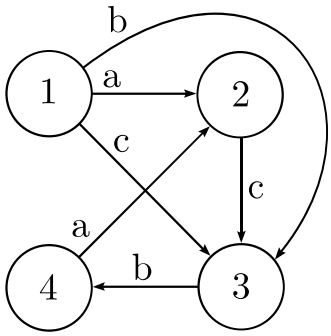

Рис. 4.1: Граф переходов конечного автомата

Граф переходов – это основной источник информации о конечном автомате. Он дает наглядное представление о том, как управляется и работает конечный автомат.

Обратите внимание – конечный автомат всегда прибывает в одном из своих состояний. Конечный автомат не может прибывать в двух состояниях одновременно. Чем-то граф переходов может напомнить вам игровую доску, где фишка находится на одном из полей и, в зависимости от выполненных условий, может перейти в одно из полей, соединённых с текущим.

Классическим примером, наглядно демонстрирующим работу конечных автоматов, может служить конечный автомат управления светофором. Попробуем разработать такой автомат.

Можно легко выделить состояния светофора: «горит зеле-

ный» - «мигает зеленый» - «горит желтый» - «горит красный» - «горит желтый и красный» - «мигает желтый».

Обратите внимание: автомат управления не может находиться в двух состояниях одновременно и, поэтому, чтобы описать ситуацию когда горят одновременно желтый и красный сигналы, нам потребовалось ввести новое состояние - «горит желтый и красный».

Обозначим все вершины графа – состояния конечного автомата:

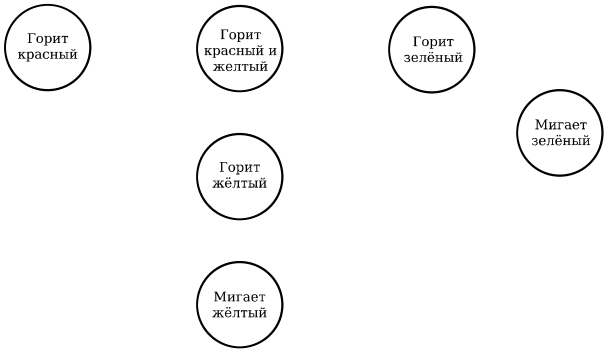

Рис. 4.2: Вершины графа переходов конечного автомата

Теперь стрелками отметим на графе все возможные переходы из каждого состояния.

Например, из состояния «мигает зеленый» светофор может переключиться в состояние «горит желтый» или, если регулирование закончилось, в состояние «мигает желтый».

Над каждой стрелкой подпишем условие, которое должно выполниться, чтобы переход произошел. Т.е. над стрелкой от «мигает зеленый» к «горит желтый» напишем условие: «прошло 5 секунд», а над стрелкой от «мигает зеленый» к «мигает желтый» напишем условие «конец регулирования».

В итоге получим полный граф переходов конечного автомата:

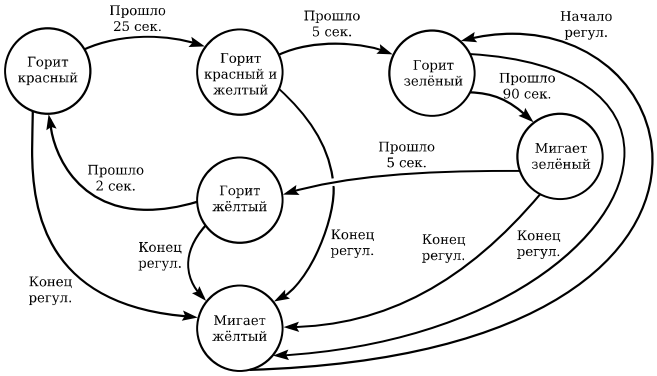

Рис. 4.3: Граф переходов конечного автомата для управления светофором

Изучите его работу. Проследите, как можно попасть в то или иное состояние.

Обратите внимание, что одни и те же входные воздействия могут приводить к разным переходам, если они возникают в разных состояниях конечного автомата.

Итак, для того чтобы реализовать в цифровой схемотехнике такое устройство, нам понадобятся регистр для хранения состояния (назовём его **«регистр состояния»**).

Также понадобится схема, которая будет принимать решение о том, какое состояние будет следующим. Посмотрите на граф: входом такой схемы должно быть текущее состояние и входные воздействия. Имея эту информацию можно определить, какое состояние будет следующим.

Регистр состояния, очевидно, синхронная схема. То есть, переход между состояниями может произойти только по положительному фронту тактового сигнала. Но как определить сам момент, когда необходимо переключиться? Ведь он зависит от входных воздействий – переход должен состояться, когда будут выполнены необходимые условия.

В такой ситуации поступают следующим образом: **регистр состояний меняет текущее состояние на следующее каждый такт вне зависимости ни от каких условий.** При этом следующее состояние, пока условия не выполнены, равно текущему. Таким образом, значение регистра не меняется,

пока не будут выполнены условия перехода.

Схема определения следующего состояния - асинхронная.

Входами этой схемы служат текущее состояние и все возможные входные сигналы (условия переходов). Выходом этой схемы будет следующее состояние.

Так как мы знаем все переходы, мы, фактически, определяем таблично заданную ФАЛ. Значит, эта схема является **дешифратором**.

Структура конечного автомата будет выглядеть следующим образом:

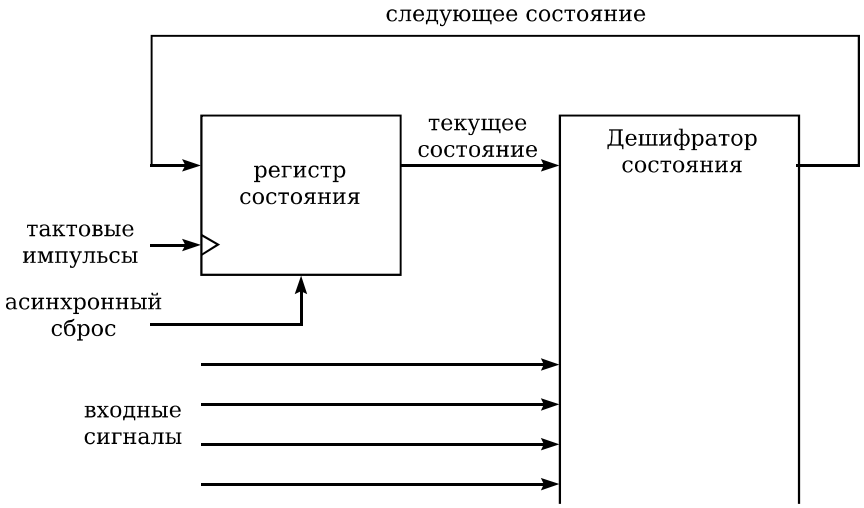

Рис. 4.4: Структура конечного автомата

Так как в составе конечного автомата присутствует триггер, **конечный автомат является синхронным цифровым устройством.** Состояние конечного автомата меняются по положительному фронту сигнала синхронизации.

Используя такую структуру в качестве основы, опишем на языке Verilog автомат, реализующий работу светофора.

```
1 ...
2
3 localparam yellow          = 3'b000;
```

```
 4  localparam yellow_blinking = 3'b001;
 5  localparam green           = 3'b010;
 6  localparam green_blinking  = 3'b011;
 7  localparam red             = 3'b100;
 8  localparam red_and_yellow  = 3'b101;
 9
10  reg [2:0] state, next_state;
11
12  always @(posedge clk or posedge rst) begin
13    if (rst) state <= yellow_blinking;
14    else     state <= next_state;
15    end if;
16  end;
17
18  always @(*) begin
19    if (end_work) next_state <= yellow_blinking;
20    else begin
21      case (state)
22        yellow_blinking: if (start_work)
23          next_state <= green;
24        green: if (passed_90_seconds)
25          next_state <= green_blinking;
26        green_blinking: if (passed_5_seconds)
27          next_state <= yellow;
28        yellow: if (passed_2_seconds)
29          next_state <= red;
30        red: if (passed_25_seconds)
31          next_state <= red_and_yellow;
32        red_and_yellow: if (passed_5_seconds)
33          next_state <= green;
34        others:
35          next_state <= yellow_blinking;
36      endcase;
37    end;
38  end;
39
40  ...
```

Листинг 4.1: Конечный автомат, реализующий работу светофора

Итак, мы реализовали конечный автомат. Но как им пользоваться? Как с помощью автомата управлять работой цифровой схемы?

Возможно, вы уже догадались, что для управления работой цифрового устройства используют анализ текущего состояния. Текущее состояние, в подавляющем большинстве случаев, и является выходом конечного автомата.

Анализируя состояние, можно переключать мультиплексоры, подавать (с помощью ФАЛ) необходимые входные воздействия или наборы данных, разрешать или запрещать работу блоков и модулей. Таким образом, можно существенно изменить поведение цифрового устройства.

Рассмотрим на конкретных примерах, как управлять работой светофора с помощью конечного автомата.

Управление светодиодами может осуществляться выходами компараторов.

Например, в состоянии «Горит зеленый», компаратор будет выдавать единицу, поступающую на зеленую лампочку.

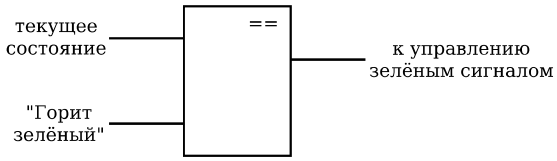

Рис. 4.5: Компаратор для управления светодиодами

В состоянии «Мигает зеленый» все немного сложнее.

Существуют разные варианты решения этой задачи.

Один из возможных – следующий: разработать схему, вырабатывающую периодические импульсы, и подключить через вентиль **И** к сигналу управления зеленой лампой светофора. При этом сам сигнал управления подавать уже не в одном, а в двух состояниях автомата.

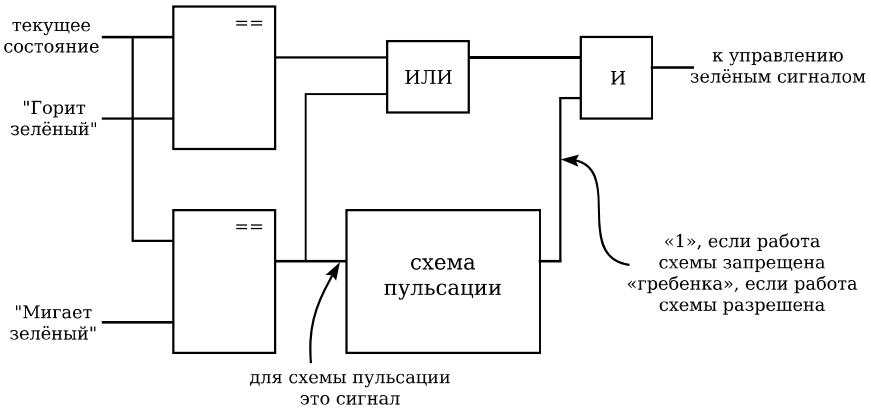

Рис. 4.6: Схема управления миганием зелёного светодиода

Схема работает следующим образом:

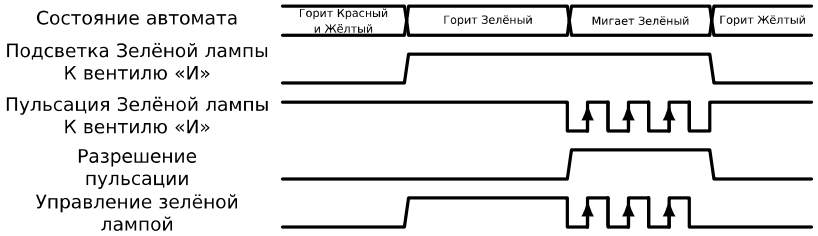

Рис. 4.7: Временная диаграмма работы схемы управления зелёным светодиодом

Схему пульсации можно использовать также для того чтобы мигать желтой лампой светофора.

Вспомним, что в цифровой схемотехнике нельзя закорачивать выходы цифровых блоков. Для того чтобы соединить выходы, необходимо использовать подходящие логические вентили.

Также вспомним, что автомат может прибывать только в одном своем состоянии.

Тогда общая схема будет выглядеть следующим образом:

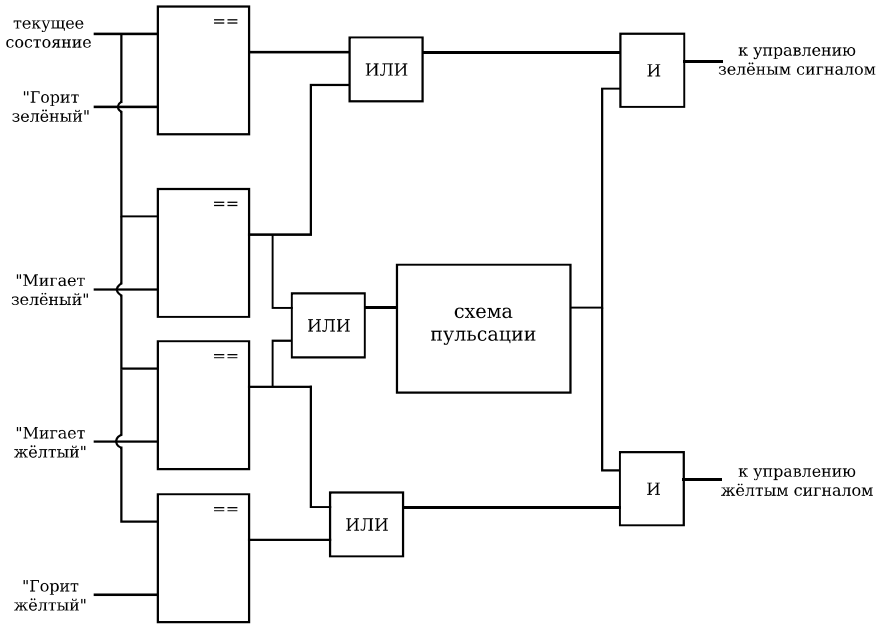

Рис. 4.8: Схема управления миганием светодиодов

Несмотря на внушительный вид это довольно простая схема:

Схема пульсации включается, когда автомат находится «Мигает Зеленый» **ИЛИ** «Мигает Желтый».

Зеленая лампа подсвечивается, когда автомат находится в состоянии «Горит Зеленый» **ИЛИ** «Мигает Зеленый» **И** при этом выход схемы пульсации равен единице.

Для желтой лампы ситуация аналогична ситуации для зеленой лампы.

В форме поведенческого кода данная схема выглядит еще проще для понимания:

```
1  ...
2
3  wire green_light;
4  wire yellow_light;
5  wire blink_en;
```

```
 6  wire blink;
 7
 8  assign green_light  = (state == green)  |
 9                        (state == green_blinking);
10  assign yellow_light = (state == yellow) |
11                        (state == yellow_blinking);
12  assign blinking_en  = (state == green_blinking) |
13                        (state == yellow_blinking);
14
15  //код, описывающий поведение схемы пульсации
16  //выход схемы - сигнал blink
17  always @(posedge clk) begin
18    if (blinking_en = '1') then
19    ... // опустим описание
20  end;
21
22  assign green  = green_light & blink;
23  assign yellow = yellow_light & blink;
24
25  ...
```

Листинг 4.2: Описание схемы пульсации лампы светофора

Хотелось бы сразу отметить некоторую особенность реализации конечного автомата в виде цифрового устройства. Взгляните на следующий граф переходов:

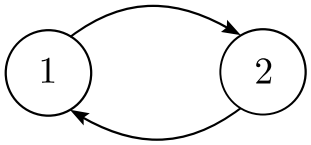

Рис. 4.9: Граф переходов конечного автомата с двумя состояниями

Несмотря на свою примитивность на его примере хорошо видно реакцию конечного автомата на внешнее воздействие.

Регистр состояния защелкивает новое состояние каждый такт, поэтому для того, чтобы автомат переключился из состояния «1» в состояние «2», требуется, чтобы входное воз-

действие «a» имело единичную длительность.

В противном случае автомат переключится из состояния «1» в состояние «2», затем, на следующий такт автомат переключится из состояния «2» в состояние «1» и так далее, пока не закончится входное воздействие.

Сравните временные диаграммы ниже, чтобы лучше почувствовать разницу.

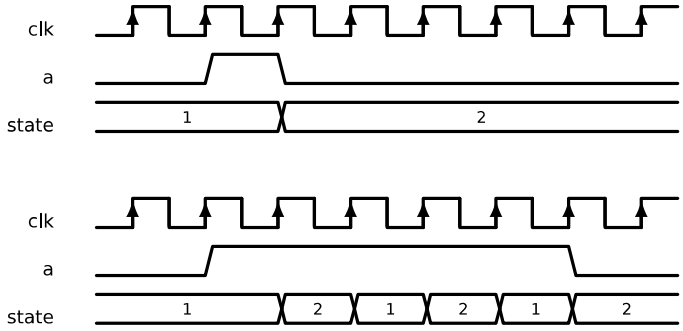

Рис. 4.10: Временные диаграммы конечного автомата с различной длительностью сигнала состояния

Кроме разделения режимов работы конечные автоматы позволяют цифровым устройствам «выполнять» некоторые алгоритмы действий. Ведь мы можем и не ставить условий для перехода из некоторых состояний в другие. Посмотрите на граф ниже - он очень напоминает алгоритм программы.

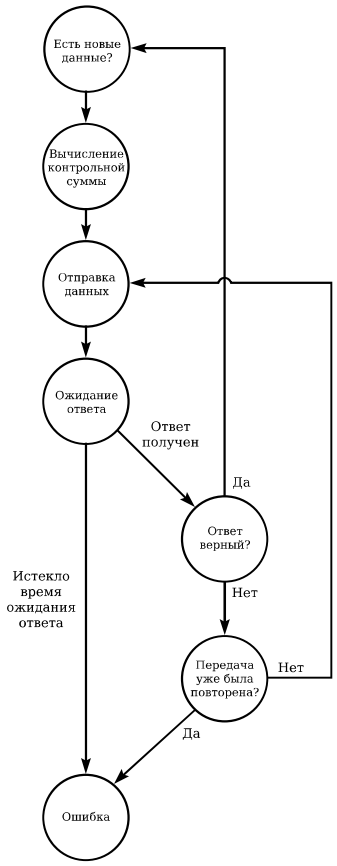

Рис. 4.11: Граф переходов, показывающий алгоритм выполнения программы

Мы вполне можем описать автомат, реализующий этот граф переходов:

```
1 ...
2
3 localparam idle = 3'b000;
4 localparam calc_checksum = 3'b001;
5 localparam send_data = 3'b010;
6 localparam wait_answer = 3'b011;
```

```
 7  localparam analyse_answer = 3'b100;
 8  localparam try_second_time = 3'b101;
 9  localparam error = 3'b110;
10
11  reg [2:0] state, next_state;
12
13  always @(posedge clk or posedge rst) begin
14    if (rst) state <= idle;
15    else state <= next_state;
16  end;
17
18  always @(*) begin
19      case (state)
20        idle:
21          if (new_data)
22            next_state <= calc_cheksum;
23
24        calc_cheksum:
25          if (checksum_calc_complete)
26            next_state <= send_data;
27
28        send_data:
29          next_state <= wait_answer;
30
31        wait_answer:
32          if (answer_recived)
33            next_state <= analyse_answer;
34          else if (wait_too_long)
35            next_state <= try_second_time;
36
37        analyse_answer:
38          if (answer_is_ok)
39            next_state <= idle;
40          else
41            next_state <= try_second_time;
42
43        try_second_time:
44          if (already_tried)
45            next_state <= error;
46          else
```

```
47            next_state <= send_data;
48
49        error:
50          if (reset_error)
51            next_state <= error;
52
53        others: next_state <= error;
54      endcase;
55  end;
56
57  ...
```

Листинг 4.3: Описание конечного автомата для приведенного графа переходов

Отметим некоторые особенности таких автоматов. Обратите внимание на условия переходов. Условия присутствуют практически во всех состояниях, даже если переход между ними линеен. Например, переход из состояния «calc_checksum» в состояние «send_data». Если для вычисления контрольной суммы требуется более одного такта, то без проверки выполнена ли операция нельзя покидать состояние вычисления контрольной суммы.

Разберем структуру цифрового устройства, которым будет управлять приведенный выше конечный автомат.

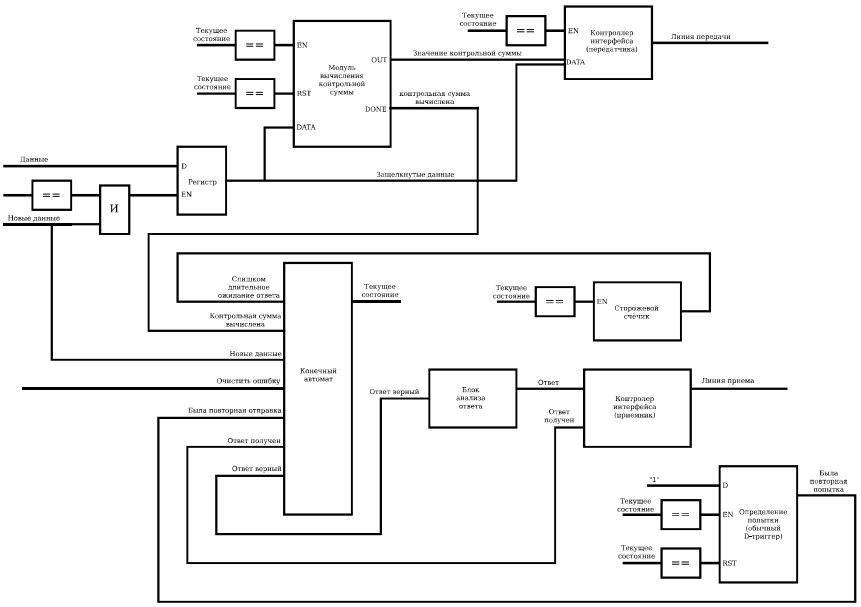

Рис. 4.12: Структура устройства с конечным автоматом

Как разработать подобную структуру?

Первое, что необходимо понимать, что блоки цифровых устройств существуют вне зависимости от того, используются ли они в данный момент или нет.

Теперь мы постараемся разбить устройство на блоки, определить функционал каждого из этих блоков и соединить их между собой.

Начинается разработка схемы с конечного автомата.

Наш конечный автомат уже разработан. Его мы помещаем в центр будущей структуры. Выходом конечного автомата служит текущее состояние. Именно его анализируют все остальные блоки устройства.

Теперь, когда мы поместили конечный автомат, приступим к проектированию периферии. Сначала регистр для того, чтобы защелкнуть и сохранить данные, которые в дальнейшем будем обрабатывать и передавать.

Когда необходимо защелкнуть данные? В самом начале ра-

боты. Т.е. в состоянии «idle», в момент, когда на вход пришел сигнал «new_data».

Идем дальше по алгоритму работы автомата. Теперь автомат переключился в состояние «calc_checksum». В структуру устройства необходимо добавить модуль, который будет считать контрольную сумму. Этот модуль будет производить вычисления, только когда автомат находится в состоянии «calc_checksum». После окончания вычисления контрольная сумма должна быть на выходе этого модуля и больше не должна меняться. Также этот модуль после завершения вычисления контрольной суммы должен выработать признак «checksum_calc_complete» для конечного автомата. Контрольную сумму можно сбросить в состоянии «idle».

Теперь автомат переходит в состояние «send_data». Для отправки уже должен быть сформирован пакет из данных и контрольной суммы (например, в виде объединенных шин).

Чаще всего контроллеры работают следующим образом. По сигналу разрешения записи данные с входа помещаются во внутренний регистр и, затем, начинается их отправка.

Именно поэтому мы использовали состояние, в котором наш конечный автомат проводит только один такт. Именно в этом состоянии происходит загрузка и запуск передатчика, а именно выработка сигнала «разрешение записи» для него.

Далее автомат попадает в состояние ожидание ответа. Для получения ответа, нам понадобится модуль приемника. Модуль приемника будет выставлять на шину значение принятых данных, а также он должен будет вырабатывать сигнал «answer_received» для работы конечного автомата.

Для функционирования конечного автомата нам также понадобится сигнал «waiting_too_long», который сигнализирует об отсутствии ответа в течение слишком большего времени. Чтобы выработать такой сигнал, поместим в устройство счётчик, который будет работать все время, пока автомат находится в состоянии «waiting_for_answer». Счётчик, при достижении максимального значения (которое мы зададим самостоятельно) будет вырабатываться сигнал «waiting_too_long». Обычно его называют «сторожевой счётчик».

Итак, если ответ получен, то его надо обработать. Мы раз-

работали конечный автомат так, что обработка должна произойти за один такт, ведь в состоянии «analyse_answer» нет условий, говорящих о конце обработки. Внесем в устройство модуль, анализирующий ответ, полученный модулем приемника.

Осталось только одно состояние, в котором мы не определили входные данные для конечного автомата. Это состояние «try_second_time». Нам понадобится схема, вырабатывающая сигнал «already_tried».

Это довольно простая схема: регистр, на вход которого подается единица. Регистр сбрасывается в состоянии «idle», а запись в него разрешена только в состоянии «try_second_time». Выход этого регистра и есть «already_tried».

Когда мы попадаем в состояние «try_second_time», мы видим, что в регистре записан «0», и переходим в состояние «send_data» для повторной отправки данных. В этот же момент времени происходит запись «1» в регистр. Теперь, если мы снова окажемся в состоянии «try_second_time», мы увидим в регистре «1» и не будем осуществлять повторную отправку данных.

Таким образом, мы закончили описание структуры нашего цифрового устройства. Мы знаем, из каких модулей оно должно состоять и как должен функционировать каждый из этих модулей.

В рамках данной лабораторной работы нас не интересует конкретная реализация модулей. Но стоит заметить, что начав описание тех или иных из них, разработчик может столкнуться с необходимостью добавить новые управляющие сигналы, ввести новые состояния конечного автомата или добавить новые связи.

Эти случаи вовсе не редкость. При проектировании цифровых устройств, практически каждый разработчик неоднократно корректирует изначально разработанную им структуру. Как правило, по мере получения опыта уменьшается количество подобных правок.

Вы увидели, как разрабатывается структура цифровых устройств, включающих конечные автоматы.

Вы убедились, что структуры, необходимые для реализации

тех или иных действий существуют одновременно, поэтому исполнение некоторых алгоритмов может быть связано с очень большими аппаратными затратами.

В таких случаях используют другие подходы к построению цифровых устройств, например, использование вычислительных ядер с программным управлением, с устройством которых вы познакомитесь в курсе «Микропроцессорные системы и средства».

Тем не менее, выполнение простых алгоритмов, например, алгоритмов управления с малым количеством ветвлений, часто перекладывается на конечные автоматы с целью минимизации программного кода и ускорения работы.

4.1 Задание лабораторной работы:

Изучить разработку к лабораторной работе.

Реализовать конечный автомат, согласно индивидуальному заданию.

Ответьте на вопросы к защите лабораторной работы.

4.2 Вопросы к защите лабораторной работы

Что такое конечный автомат?

Для чего конечные автоматы используются в цифровых устройствах?

Из каких цифровых блоков состоит конечный автомат?

Конечный автомат это синхронное или асинхронное устройство?

Работу какого цифрового блока конечного автомата определяет граф переходов?

Почему для верного функционирования конечного автомата важна длительность управляющих сигналов?

Изучите рисунок структуры цифрового устройства на стр.10. Назовите значение состояния автомата, анализируе-

мое каждым из компараторов.

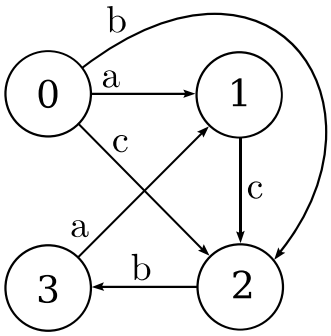

Рис. 4.13: Пример задания лабораторной работы