МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Национальный исследовательский ядерный университет «МИФИ»

ТЕОРИЯ АВТОМАТОВ

Лабораторный практикум

Под редакцией Б.Н. Ковригина

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

Москва 2012

УДК 004.3(076.5)
ББК 32.973я7
Т33

Теория автоматов. Лабораторный практикум: *учебное пособие* /Под ред. Б.Н. Ковригина. М.: НИЯУ МИФИ, 2012. 192 с.

Авторы: Н.А. Дмитриев, А.А. Дюмин, М.Н. Ёхин, Б.Н. Ковригин, В.Г. Тышкевич, Л.И. Шустова, И.М. Ядыкин.

В пособии содержится описание шести лабораторных работ по курсу «Теория автоматов». В каждой работе дано краткое изложение теоретических основ и особенностей выполнения работ.

Предназначено для приобретения и закрепления практических навыков при использовании теории автоматов для разработки цифровых устройств и программного обеспечения. Сопутствующая задача – это начальное изучение и освоение профессиональной САПР фирмы Xilinx, используемой на последующих трех семестрах.

Пособие предназначено для студентов, специализирующихся в области информатики и вычислительной техники.

Пособие подготовлено в рамках
Программы создания и развития НИЯУ МИФИ

Рецензент канд. техн. наук Воронков А.Ф.

ISBN 978-5-7262-1781-9

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Конечные автоматы. Основные понятия и определения

Конечный автомат представляет собой хотя и абстрактную, но с функциональной точки зрения довольно точную модель дискретного процесса либо (дискретного) вычислительного или управляющего устройства и является удобным средством описания многих систем, взаимодействующих с окружением и реагирующих на поток внешних событий, составляющих многие компоненты аппаратного и программного обеспечения. Такая модель обладает наглядностью и выразительностью, ясностью семантики (трактовки элементов модели) и в то же время достаточно строга и формальна. Ее используют при решении самых разнообразных задач, связанных с информационными процессами, таких как:

- построение систем ПО, включая лексические анализаторы компиляторов;
- проектирование систем технической диагностики;
- проектирование узлов и блоков ЭВМ и других вычислительных систем и устройств;
- проектирование устройств промышленной автоматики и др.

Общую теорию автоматов подразделяют на абстрактную и структурную. Различие между ними заключается в том, что абстрактная теория, отвлекаясь от структуры автомата (т.е. не интересуясь способом его построения), изучает лишь поведение автомата относительно внешней среды.

В противоположность абстрактной теории, структурная интересуется как структурой самого автомата, так и структурой входных воздействий и реакций автомата на них. В структурной теории изучаются способы построения автоматов, способы кодирования входных воздействий и реакций автомата. Таким образом, структурная теория автоматов является продолжением и дальнейшим развитием абстрактной теории. Опираясь на аппарат булевых функций и на абстрактную теорию автоматов, структурная теория дает эффективные рекомендации по разработке реальных устройств вычислительной техники.

Автоматом называют дискретный преобразователь информации, способный принимать различные состояния, переходить под воздействием входных сигналов из одного состояния в другое и выдавать выходные сигналы.

Если множество состояний автомата, а также множества входных и выходных сигналов конечны, то автомат называют *конечным автоматом*. Все реальные автоматы являются конечными.

В процессе работы конечного автомата происходят последовательные переходы между конечным числом его внутренних состояний, причем состояние автомата в определенный момент времени однозначно определяется входным и выходным сигналами. Такие автоматы представляют собой основу всей современной вычислительной техники и всевозможных дискретных систем автоматического контроля и управления.

Можно выделить несколько классов конечных автоматов, основными из которых являются *детерминированные* и *недетерминированные* конечные автоматы; конечные автоматы *без памяти* и конечные автоматы *с памятью* (магазинной и произвольного доступа). Детерминированные автоматы определяются: множеством состояний, одно из которых является стартовым и минимум одно является принимающим, входным алфавитом (множеством сигналов, которые могут подаваться на вход автомата), и функцией переходов, которая однозначно определяет следующее состояние автомата в зависимости от текущего состояния автомата и символа алфавита на входе автомата. Основным (но не единственным) отличием недетерминированных конечных автоматов, от детерминированных конечных автоматов является возможность нахождения в нескольких состояниях одновременно, что значительно усложняет их моделирование, но в ряде случаев значительно облегчает синтез. Следует, отметить, что в теории конечных автоматов доказывается эквивалентность детерминированных и недетерминированных конечных автоматов.

Конечные автоматы с памятью помимо того, что хранят свое текущее состояние, имеют возможность хранения некоторой дополнительной информации, которая также может определять характер его работы. Данные автоматы являются более сложными в синтезе и реализации по сравнению с автоматами без памяти, но при этом позволяют решать значительно более широкий круг задач.

Для задания конечного автомата фиксируют три конечных множества (алфавита):

множество входных сигналов $X = (x_1, x_2, \ldots, x_m)$,

множество выходных сигналов $Y = (y_1, y_2, \ldots, y_k)$,

множество внутренних состояний автомата $S=(s_0, s_1 \ldots, s_n)$.

На этих множествах задают две функции:

функцию переходов f, определяющую состояние автомата $s(t+1)$ в момент дискретного времени $t+1$ в зависимости от состояния автомата $s(t)$ и значения входного сигнала $x(t)$ в момент времени t;

функцию выходов φ, определяющую зависимость выходного сигнала автомата $y(t)$ от состояния автомата и входного сигнала $x(t)$ в момент времени t.

Таким образом, функция переходов f устанавливает зависимость внутреннего состояния автомата в следующий момент времени от состояния входа и внутреннего состояния в настоящий момент времени. Функция выходов φ устанавливает зависимость состояния выхода автомата от состояния входа и внутреннего состояния автомата.

По способу формирования функций выхода выделяют автоматы Мили и Мура.

Закон функционирования автомата Мили задается уравнениями:

$$s(t+1) = f[s(t), x(t)],$$

$$y(t) = \varphi[s(t), x(t)].$$

Обобщенная структура автомата Мили приведена на рис. В.1.

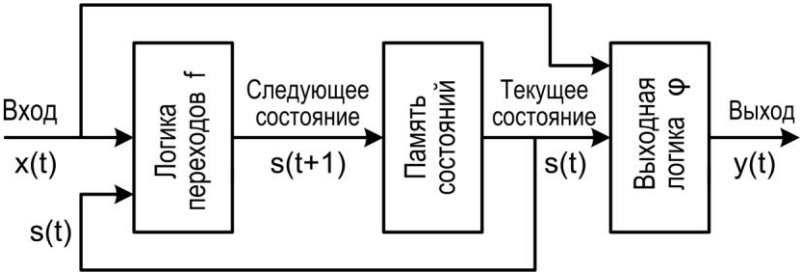


Рис. В.1. Структура автомата Мили

Автомат Мура описывается следующей функцией переходов и функцией выходов:

$$s(t+1) = f[s(t), x(t)],$$
$$y(t) = \varphi[s(t)].$$

Структура автомата Мура приведена на рис. В.2.

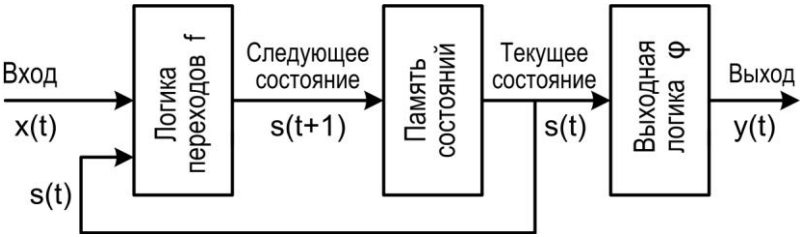


Рис. В.2. Структура автомата Мура

Из сравнения законов функционирования видно, что, в отличие от автомата Мили, выходной сигнал в автомате Мура зависит только от текущего состояния автомата и в явном виде не зависит от входного сигнала. Отличительная особенность автоматов Мили состоит в том, что их выходные сигналы зависят как от состояния автомата, так и от значения входного сигнала.

Конечный автомат состоит из трех основных частей.

Память состояний. Для того чтобы выходные сигналы автомата зависели от предыдущих входных воздействий на автомат, автомат должен сохранять информацию о предыдущих входных воздействиях. Так, вводится понятие «состояния автомата», где состояние соответствует некоторой памяти о прошлых входных воздействиях. Тогда зависимость выходных сигналов как от настоящих, так и от прошлых входных воздействий может быть выражена в виде функции от настоящих входных воздействий и *состояния* автомата.

Логика переходов. Конечный автомат может находиться в каждый конкретный момент времени только в одном состоянии. Правила перехода в очередное состояние определяются комбинационной схемой, называемой логикой переходов. Следующее состояние определяется как функция текущего состояния и входного воздействия.

Логика формирования выхода. Выход автомата обычно определяется как функция текущего состояния и исходного состояния

входа (в случае автомата Мили). Формирование выходного сигнала автомата определяется с помощью логики формирования выхода.

На практике разработчики вычислительных средств часто используют совмещенную модель Мили–Мура, называемую С-автоматом. Отличие С-автомата от моделей Мили и Мура состоит в том, что он одновременно реализует две различные функции выходов y_1 и y_2, каждая из которых характеризует одну из этих моделей.

Кроме того, на множестве состояний автомата обычно фиксируют одно из внутренних состояний s_0 в качестве начального состояния. Выделение на множестве S начального состояния s_o объясняется чисто практическими соображениями, связанными необходимостью фиксировать условия начала работы автомата. Если на множестве состояний S автомата выделяется специальное начальное состояние $s_o \in S$, то такой автомат называют *инициальным*.

В инженерной практике довольно часто алгоритмы, определяющие условия работы цифрового устройства, задаются в такой форме, что необходимость проведения этапа абстрактного синтеза отпадает. Поэтому проектирование цифрового устройства реализуется решением задачи структурного синтеза конечного автомата. Целью этапа структурного синтеза является построение логической схемы, реализующей автомат на элементах заданного типа.

Существует единый прием (канонический метод), позволяющий свести проблему структурного синтеза произвольных автоматов к проблеме синтеза комбинационных схем. В классической постановке канонический метод синтеза состоит из следующих макрошагов: построение таблиц возбуждений элементов памяти автомата; определение булевых выражений функций возбуждения и выходов; минимизация булевых выражений функций возбуждения и выходов.

Более подробно задачи структурного синтеза конечного автомата рассматриваются в лабораторных работах данного пособия.

Успешное выполнение данного лабораторного практикума должно создать основу, позволяющую студентам воспринимать и усваивать другие специальные дисциплины по информационным технологиям, вычислительным средствам и системам.

Лабораторная работа 1

ИЗУЧЕНИЕ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ПРОЕКТИРОВАНИЯ ЦИФРОВЫХ АВТОМАТОВ

Цель: изучить состав и возможности органов управления универсального лабораторного стенда; изучить маршрут проектирования цифровых автоматов с использованием САПР.

Введение

Лабораторный практикум по курсу «Теория автоматов» выполняется в учебной лаборатории, основу которой составляет *автоматизированное рабочее место студента-проектировщика (АРМ)*. Данная универсальная компьютеризированная лаборатория объединяет на единой инструментальной базе все практикумы по дисциплинам системотехнического цикла.

АРМ студента-проектировщика – это комплекс аппаратно-программных средств, предназначенный для обучения основам теории автоматов, схемотехники, проектированию цифровых систем на основе микроконтроллеров и программируемой логики. Каждое рабочее место включает:

- персональный компьютер, оснащенный платой расширения цифрового осциллографа серии BORDO и 16-канального логического анализатора;
- профессиональную САПР ПЛИС и инструментальные средства автоматизации программирования микроконтроллера;
- универсальный лабораторный стенд, содержащий ПЛИС FPGA XCS10-3PC84 фирмы XILINX, 8-разрядный микроконтроллер семейства MCS-51 PCF80C552 фирмы PHILIPS, память и органы управления и индикации.

В лабораторном практикуме по курсу «Теория автоматов» используется только часть оборудования стенда, а именно, ПЛИС FPGA XCS10-3PC84, клавишные регистры, генераторы и индикация.

Разработка цифровых автоматов на ПЛИС невозможна без применения систем автоматизированного проектирования (САПР). Особо значимыми становятся процедуры отладки и верификации

проектных решений. Понимание единых общепризнанных средств описаний, создаваемых автоматизированными средствами проектирования, необходимо для современного квалифицированного разработчика.

Тщательное изучение возможностей и особенностей работы вспомогательного оборудования стенда и САПР имеет принципиальное значение для успешного выполнения лабораторного практикума в целом. Недостаточное знание инструментальных средств проектирования, используемых в практикуме, может привести к значительным затратам времени при выполнении лабораторных работ и «в конечном итоге» к неудаче в работе.

Начальные сведения о ПЛИС

Микросхемы программируемой логики или ПЛИС (программируемые логические интегральные схемы) – одно из наиболее динамично развивающихся направлений современной цифровой электроники. Привлекательность данной технологии заключается в предоставляемой конечному пользователю возможности быстрого создания цифровых устройств с произвольной внутренней структурой. По сравнению со специализированными цифровыми микросхемами (Application Specific Integral Circuit, ASIC), цикл разработки устройств на ПЛИС занимает значительно меньше времени и неизмеримо дешевле (благодаря тому, что изменение логической схемы выполняется путем перепрограммирования одного и того же экземпляра ПЛИС). Таким образом, вместо металлических соединений, реализуемых в процессе производства ASIC, в ПЛИС используются соединения, коммутируемые программируемыми ключами. Для задания этих соединений в ПЛИС существует теневая (конфигурационная) память, хранящая таблицу соединений.

В настоящее время наиболее распространенные серии ПЛИС имеют следующую архитектуру:

- CPLD (Complex Programmable Logic Device) устройства, использующие для хранения конфигурации энергонезависимую память (Flash или EEPROM);
- FPGA (Field Programmable Gate Array) устройства, использующие для хранения конфигурации энергозависимую память, которая требует инициализации после включения питания.

Поскольку универсальный лабораторный стенд содержит ПЛИС FPGA, то ниже рассматривается только этот тип микросхем программируемой логики.

Интегральные схемы типа FPGA

В наиболее типичном варианте ПЛИС, выполненная по технологии FPGA, состоит из прямоугольной матрицы конфигурируемых логических блоков (Configurable Logic Blocks, CLB), окруженной блоками ввода-вывода (Input/Output Block, IOB). Между CLB располагаются программируемые трассировочные линии (рис. 1.1). Между матрицей CLB и блоками ввода-вывода имеются отдельные межсоединения, которые и обеспечивают подключение внешних сигналов.

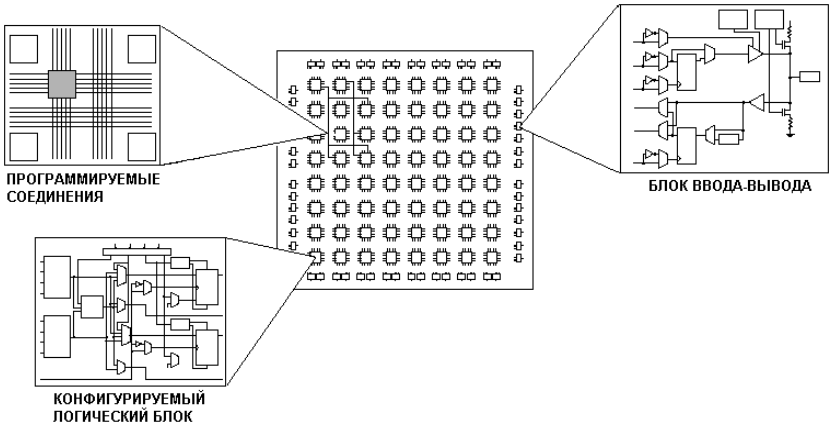


Рис. 1.1. Обобщенная структура ПЛИС FPGA

Все части FPGA (конфигурируемые логические блоки, система межсоединений и блоки ввода/вывода) являются конфигурируемыми или реконфигурируемыми, причем средствами самих пользователей.

При конфигурировании FPGA логические блоки настраиваются на выполнение необходимых операций преобразования данных, а система межсоединений – на требуемые связи между логическими блоками. В результате во внутренней области FPGA реализуется

схема нужной конфигурации. Расположенные по краям кристалла блоки ввода/вывода обеспечивают интерфейс FPGA с внешней средой. Блоки ввода/вывода современных FPGA можно программировать на выполнение требований множества стандартов передачи данных (число таких стандартов может доходить до 20).

Конфигурирование – это процесс загрузки битовой последовательности, полученной с помощью программного обеспечения проектирования, во внутреннюю энергозависимую конфигурационную память кристалла FPGA. При выключении питания конфигурация в ПЛИС FPGA разрушается. Поэтому при включении питания необходим процесс программирования (инициализации, конфигурирования) кристалла FPGA – загрузка данных конфигурации. Для отладки спроектированной схемы используют специальный кабель для загрузки конфигурационных данных. Этот кабель соединяет персональный компьютер с микросхемой FPGA. В конечном устройстве отлаженные данные конфигурации хранятся в ПЗУ, которое непосредственно подключается к ПЛИС FPGA, и после включения питания они автоматически загружаются в ПЛИС. Процесс конфигурирования может производиться неограниченное число раз.

Для более детальной иллюстрации архитектуры ПЛИС FPGA рассмотрим используемую в лабораторном стенде микросхему XCS10-3PC84 популярного семейства Spartan фирмы Xilinx. В табл. 1.1 приведены данные этой микросхемы.

Таблица 1.1

Состав микросхемы XCS10-3PC84

Размер матрицы логических блоков	Число логических блоков CLB	Макс. число доступных вводов/выводов	Число триггеров	Макс. число вентилей (без ОЗУ)	Типичное число вентилей (в логике и в ОЗУ)
14 × 14	196	61	616	10000	3000 - 10000

Порядок выполнения работы

1. Войти в систему и открыть новый проект под своим именем.

Примечание. Данный пункт и все последующие выполняются, руководствуясь приложением 1.

2. Выполнить ввод схемы, приведенной на рис. 1.2 (счетчик + комбинационная схема), в редакторе схем системы Xilinx Foundation. Комбинационная схема реализует минимальное выражение логической функции $F(X3,X2,X1,X0) = \Sigma(0, 3, 8, 11, 12, 15)$, которое равно $F(X3,X2,X1,X0) = X3 \cdot X1 \cdot X0 \vee \overline{X2} \cdot X1 \cdot X0 \vee X3 \cdot \overline{X1} \cdot \overline{X0} \vee \vee \overline{X2} \cdot \overline{X1} \cdot \overline{X0}$

3. Выполнить функциональное моделирование созданной схемы (рис. 1.3).

4. Проверить правильность функционирования комбинационной схемы.

5. Продемонстрировать преподавателю работу схемы на экране.

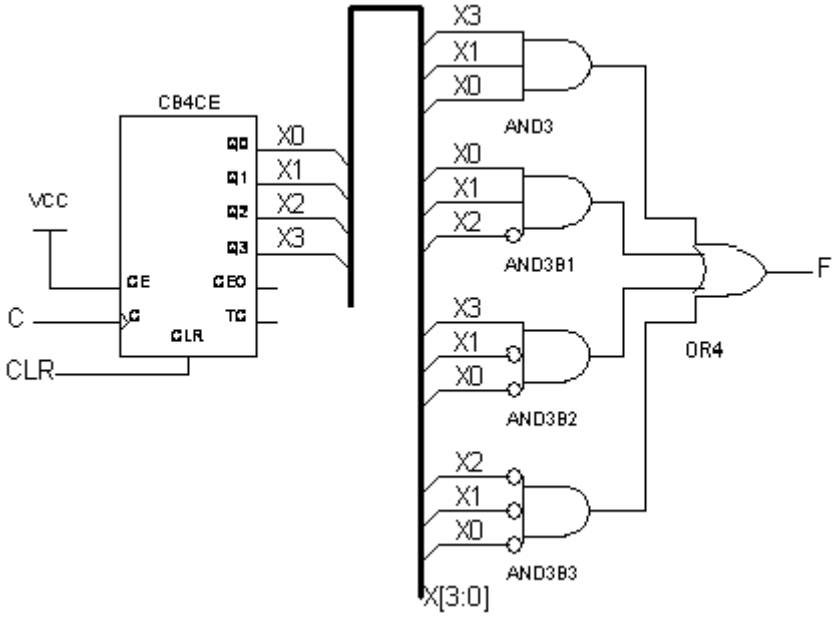


Рис. 1.2. Схема соединения счетчика и комбинационной схемы

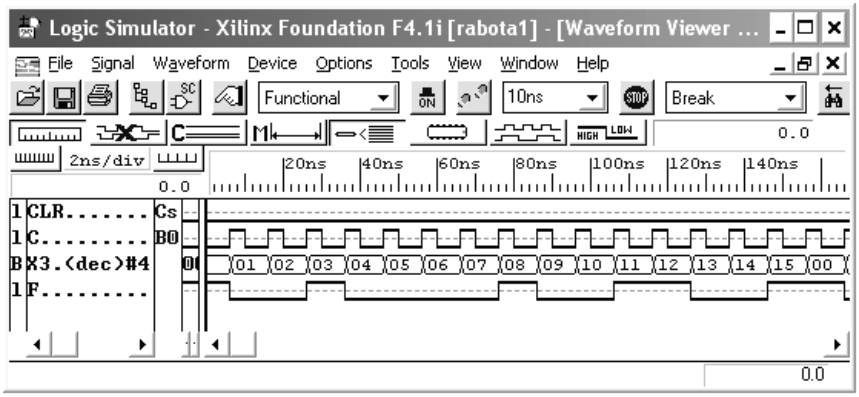


Рис. 1.3. Временная диаграмма работы счетчика и комбинационной схемы

СПИСОК ЛИТЕРАТУРЫ

1. Кузелин М.О., Кнышев Д.А., Зотов В.Ю. Современные семейства ПЛИС фирмы Xilinx. – М.: Горячая линия-Телеком, 2004. – 440 с.

2. Ковригин Б.Н. Введение в инструментальные средства проектирования и отладки цифровых устройств на ПЛИС. Учебное пособие. – М.: МИФИ, 2006. – 157 с.

Лабораторная работа 2

СТРУКТУРНЫЙ СИНТЕЗ СИНХРОННОГО АВТОМАТА МИЛИ

Цель: изучить структурный синтез синхронного конечного автомата Мили, закон функционирования которого представлен в виде словесного описания; овладеть практическими навыками отработки проектируемого автомата моделированием с использованием профессиональной САПР.

Введение

Синхронный конечный автомат – конечный автомат, в котором изменение состояния происходит в моменты времени, задаваемые тактовыми сигналами от независимого источника.

Таким независимым источником, как правило, является генератор синхронизирующих импульсов. Синхроимпульсы от генератора тактовых сигналов задают дискретные моменты времени t = 0,1,2, ... перехода автомата в очередное состояние. Моменты времени t = 0,1,2, ... будем называть тактами. При этом длительность такта имеет фиксированный промежуток времени, величина которого определяется временем протекания самого длительного переходного процесса.

Таким образом, в синхронных конечных автоматах моменты времени, в которые автомат считывает входные сигналы, определяются принудительно синхронизирующими сигналами. После очередного синхронизирующего сигнала с учётом «считанного» состояния и в соответствии с соотношениями для функционирования автомата происходит переход в новое состояние и выдача сигнала на выходе, после чего автомат может воспринимать следующее значение входного сигнала.

Синхронизирующие сигналы поступают на специальный вход запоминающих элементов (ЗЭ), обозначаемый символом *C* на рисунках и называемый синхровходом запоминающего элемента.

В структурной теории автоматов существует универсальный прием (канонический метод структурного синтеза), позволяющий свести задачу структурного синтеза произвольных автоматов к задаче синтеза комбинационных схем. Результатом канонического

метода структурного синтеза является система логических уравнений, выражающая зависимость выходных сигналов автомата и сигналов, подаваемых на входы запоминающих элементов, от сигналов, приходящих на вход автомата в целом, и сигналов, снимаемых с выхода запоминающих элементов. Данный метод предполагает представление структурной схемы автомата в виде трех частей: блока запоминающих элементов и двух комбинационных схем КС1 и КС2 (рис. 2.1). Поясним назначение каждой части схемы.

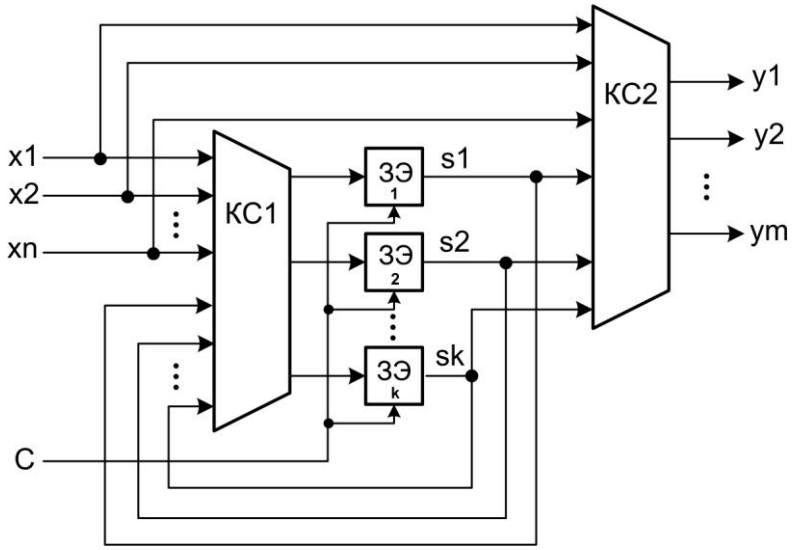


Рис. 2.1. Структура синхронного конечного автомата Мили

Память состояний автомата состоит из *k* запоминающих элементов (ЗЭ), каждый из которых под действием синхронизирующего сигнала *C* может устанавливаться в состояние 0 или 1. Состояние запоминающих элементов определяет текущее состояние автомата.

Комбинационная схема КС1 формирует очередное состояние автомата, которое определяется как функция текущего состояния и входного воздействия. Это состояние будет записано в запоминающие элементы автомата при поступлении синхросигнала.

Выходные сигналы *Y* автомата определяются комбинационной схемой КС2 как функция входных сигналов *X* и текущего состояния автомата *S*.

Структурный синтез конечных автоматов заключается в выборе типов запоминающих элементов, в составлении функций возбуждения каждого запоминающего элемента и функций кодированных выходов заданного автомата.

В общей теории автоматов для задания функций переходов и выходов автомата используются таблицы переходов и выходов, а для состояний используются мнемонические имена. Затем выбирается способ кодирования внутренних состояний автомата и комбинации переменных состояния ставятся в соответствие именам состояний. Выбирается тип триггера для памяти состояния (например, D- или JK-триггеры). После чего составляется кодированная таблица переходов и выход, на основе которых выполняется синтез комбинационных схем, формирующих сигналы возбуждения запоминающих элементов и выходные сигналы.

Для цифровых автоматов небольшой размерности, как показывает инженерная практика, часто наиболее удобен непосредственный переход от описательного задания способов функционирования автомата к заданию автомата кодированными таблицами переходов и выходов. Следует также отметить, что при проектировании цифровых устройств большой размерности, как правило, находят применение более простые и компактные средства описания закона функционирования, например, язык граф-схем алгоритмов, чем общие методы синтеза автомата. В связи с этим общие языки (таблицы переходов и выходов, матрицы переходов, диаграммы переходов и т.д.) на практике при проектировании цифровых устройств большой размерности обычно не применяются.

Для обеспечения стабильной и безотказной работы используется установка автомата в начальное состояние. Таким образом, всегда обеспечивается инициализация автомата в заранее предопределенном состоянии при первом тактовом импульсе. В случае если установка автомата в начальное состояние не предусмотрена, невозможно предсказать, с какого начального состояния начнется функционирование автомата, что может привести к сбоям в работе всей системы.

Задача структурного синтеза синхронного автомата, рассматриваемая в данной работе, заключается в кодировании состояний, задании автомата кодированными таблицами переходов и выходов и

синтезе комбинационных схем, формирующих сигналы возбуждения заданного типа запоминающих элементов и выходные сигналы.

Пример

Постановка задачи. Спроектировать синхронный конечный автомат Мили с одним входом X и одним выходом Y. При $X = 0$ автомат последовательно принимает состояния 0,1,2,4,5,0,1,…; при $X = 1$ – 0,1,5,6,4,7,0,1,… (последовательности циклические). Автомат на выходе Y формирует сигнал 1 при $X = 0$ в состоянии 5; при $X = 1$ – в состоянии 4. В остальных состояниях автомата $Y = 0$.

В качестве элемента памяти использовать D-триггер. Автомат должен иметь вход установки в начальное состояние.

Составление кодированной таблицы переходов и выходов

Вначале необходимо определить, сколько потребуется двоичных переменных для представления состояний в таблице переходов. Из анализа исходных данных (условия задачи) следует, что автомат может принимать семь различных состояний: 0,1,2,4,5,6,7. Таким образом, минимальное число двоичных разрядов, необходимое для кодирования этого количества состояний, равно трем.

Далее необходимо присвоить конкретному состоянию двоичную комбинацию, которую назовем *кодом состояния* автомата. В нашем случае естественно сопоставить каждому номеру состояния его двоичный номер, $0 = 000_2$, $1 = 001_2$, $4 = 100_2$ и т.д. Обозначим двоичные разряды кода состояния через $Q2$, $Q1$, $Q0$.

В кодированной таблице переходов для каждой комбинации кода состояния и входного воздействия указывается код следующего состояния.

Получить кодированную таблицу переходов автомата просто. Для этого необходимо в одном столбце записать значение входа X и соответствующие двоичные коды состояний автомата $Q2$, $Q1$, $Q0$. Эти состояния отнесем к моменту времени t и будем называть *текущим состоянием* автомата. Текущие состояния автомата записаны в колонках 1-4 табл. 2.1.

Затем в следующем столбце напротив каждого двоичного набора предыдущего столбца запишем новое состояние автомата, в ко-

торое он перейдет после поступления синхронизирующего сигнала. Например, если текущее состояние автомата при $X = 0$ равно $Q2,Q1,Q0 = 100$, то новое состояние автомата, в которое он должен перейти в соответствии с исходными условиями задачи, будет равно 101. Данные состояния отнесем к моменту времени $t+1$ и будем называть *следующим (будущим) состоянием* автомата. Следующие состояния автомата записаны в колонках 5-7 табл. 2.1. В восьмой колонке задано состояние выхода Y автомата в текущий момент времени t.

Таблица 2.1

Кодированная таблица переходов и выходов

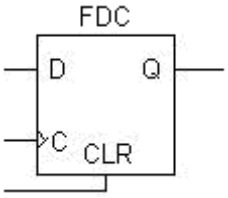
Время t				Время t+1			Время t
X	$Q2$	$Q1$	$Q0$	$Q2$	$Q1$	$Q0$	Y
1	2	3	4	5	6	7	8
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	1	0	0	0
0	1	0	0	1	0	1	0
0	1	0	1	0	0	0	1
1	0	0	0	0	0	1	0
1	0	0	1	1	0	1	0
1	1	0	1	1	1	0	0
1	1	1	0	1	0	0	0
1	1	0	0	1	1	1	1
1	1	1	1	0	0	0	0

Кодированная таблица переходов определяет зависимость состояний запоминающих элементов $Q2(t+1)$, $Q1(t+1)$ и $Q0(t+1)$ в момент времени $t+1$ от значения входного сигнала и внутренних состояний запоминающих элементов в предыдущий момент времени. В кодированной таблице выходов выходной сигнал Y определен в зависимости от значения входного сигнала в момент времени t и внутренних состояний запоминающих элементов в этот же момент времени.

Составление таблицы возбуждения автомата

Следующий шаг заключается в составлении *таблицы возбуждения,* т.е. определения закона функционирования комбинационной схемы КС1 (см. рис. 2.1). В этой таблице для каждой комбинации кода состояния и входного воздействия указываются значения сигналов, которые необходимо подать на входы триггеров, чтобы заставить автомат перейти в желаемое следующее состояние с соответствующим кодом. Структура и содержание этой таблицы зависят от типа используемых триггеров (D-, JK-, T-триггеры и т.д.).

В нашем случае тип триггера задан в условии задачи – это D-триггер. В библиотеке базовых элементов ПЛИС XC10PC84 синхронный D-триггер с асинхронной установкой в 0 имеет имя FDC. Условное графическое обозначения D-триггера FDC и его таблица переходов приведены на рис. 2.2.



Входы			Выход
CLR	D	C	Q
1	×	×	0
0	1	0/1	1
0	0	0/1	0

Рис. 2.2. Условное графическое обозначение D-триггера FDC и его таблица переходов

В этом обозначении: *C* – прямой синхронизирующий вход, *CLR* (*Clear*) – асинхронный вход установки триггера в 0, *D* – логический вход данных. В таблице переходов D-триггера: × – произвольное значение, 0/1 – изменение синхросигнала из 0 в 1.

С помощью кодированной таблицы переходов можно определить для всех трех D-триггеров способ формирования входных сигналов *D*2, *D*1 и *D*0. Для формирования каждого из входных сигналов *D*2, *D*1 и *D*0 (выходов комбинационной схемы КС1) используются входной сигнал *X* в момент времени *t* и выходные сигналы, определяемые состояниями триггеров в тот же момент времени *t* (см. рис. 2.1).

Зависимость входного сигнала $D_i(t)$ триггера от состояний всех D-триггеров в момент времени t и от значения входного сигнала автомата $x(t)$ называют *функцией возбуждения* D-триггера.

Таким образом, в таблице возбуждения следует указать значение сигнала, который необходимо подать на D-вход каждого триггера при всех возможных комбинациях вход/код состояния. В нашем примере кодированная таблица переходов, дополненная функциями возбуждения, имеет вид, указанный в табл. 2.2.

Поскольку для D-триггера $Q(t+1) = D(t)$ (времена t и $t+1$ означают время до и после поступления синхросигнала), таблица возбуждения тождественна кодированной таблице переходов автомата, за исключением того, как обозначается то, что вносится в эту таблицу (см. колонки 5-7 и 8-10 в табл. 2.2). Таким образом, при использовании D-триггеров фактически не нужно составлять отдельную таблицу возбуждения, достаточно назвать кодированную таблицу переходов таблицей «переход/возбуждение».

Таблица 2.2

Таблица переходов и функций возбуждения D-триггеров автомата

Номер набора	Время t				Время $t+1$			Время t		
	X	$Q2$	$Q1$	$Q0$	$Q2$	$Q1$	$Q0$	$D2$	$D1$	$D0$
	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	1	0	0	1
1	0	0	0	1	0	1	0	0	1	0
2	0	0	1	0	1	0	0	1	0	0
4	0	1	0	0	1	0	1	1	0	1
5	0	1	0	1	0	0	0	0	0	0
8	1	0	0	0	0	0	1	0	0	1
9	1	0	0	1	1	0	1	1	0	1
13	1	1	0	1	1	1	0	1	1	0
14	1	1	1	0	1	0	0	1	0	0
12	1	1	0	0	1	1	1	1	1	1
15	1	1	1	1	0	0	0	0	0	0

Табл. 2.2, составленная по изложенной методике, определяет функции возбуждения входов всех триггеров. Аргументы этих функций записаны в колонках 1–4. Поскольку значения всех переменных определены для одного и того же момента времени t, то

функции возбуждения D-триггеров являются логическими функциями.

Минимизация функций возбуждения

Минимизация – это процесс нахождения такого эквивалентного выражения функции алгебры логики, которое содержит минимальное число вхождений переменных.

Представим функции возбуждения в минимальной дизъюнктивной нормальной форме (МДНФ). Один из наиболее удобных методов минимизации основан на использовании диаграмм Вейча (карт Карно).

Диаграмма Вейча – это графическое представление таблицы истинности, которой задается логическая функция.

Диаграмма Вейча для логических функций четырех аргументов $X,Q2,Q1,Q0$ приведена на рис. 2.3.

Она содержит $2^4 = 16$ клеток, каждая из которых соответствует одному двоичному значению аргументов, на которых задается логическая функция. Эти наборы показаны на рис. 2.3 в каждой клетке. Разметка строк и столбцов диаграммы Вейча выполняется таким образом, что для любой клетки легко определить соответствующую ей комбинацию переменных по заголовкам строки и столбца, на пересечении которых находится эта клетка. Эта комбинация переменных (логическое произведение тех переменных, которые соответствуют записанным двоичным наборам), приведена в каждой клетке диаграммы Вейча (см. рис. 2.3).

При этом диаграмма строится таким образом, что склеивающиеся между собой наборы аргументов оказываются расположенными в соседних клетках, так как отличаются значением только одной переменной. Можно увидеть, что в первой и последней колонках диаграммы находятся склеивающиеся между собой наборы аргументов, то же самое можно сказать и в отношении верхней и нижней строк, т.е. эти колонки и строки следует считать соседними. Поэтому диаграмму Вейча для функций четырех аргументов следует представлять нанесенной на поверхность тора.

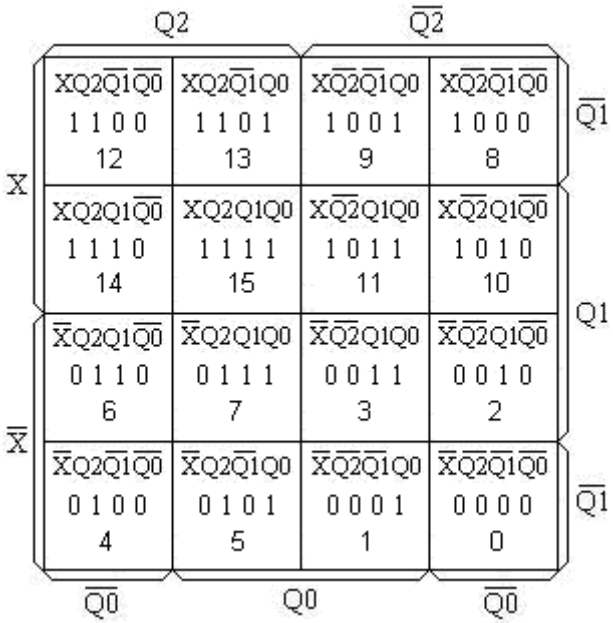


Рис. 2.3. Диаграмма Вейча для функций четырех аргументов

Имея такое изображение диаграммы Вейча с нанесенными значениями двоичных наборов аргументов, процесс составления диаграмм Вейча для функций возбуждения триггеров автомата предельно облегчается. Чтобы представить функцию возбуждения диаграммой Вейча, следует записать единицы и нули в клетки диаграммы, соответствующие наборам, на которых функция равна единице или нулю (рис. 2.4).

Из табл. 2.2 видно, что из 16 возможных состояний используются только 11. Остальные пять состояний являются запрещенными, они никогда не появляются при правильной работе автомата. Это состояния 0011, 0110, 0111, 1010, 1011 (им будут соответствовать незаполненные поля диаграмм Вейча). Отметим на диаграммах Вейча эти состояния символом «×» (см. рис. 2.4).

На этих наборах аргументов значения функций могут быть произвольными, т.е. равными 0 или 1. Поэтому при синтезе схем с не полностью определенной функцией можно произвольно задать значения выходных сигналов для запрещенных комбинаций вход-

ных сигналов; нормальная работа схемы при этом не нарушается. При минимизации не полностью определенных функций на запрещенных комбинациях придают такие значения, которые приводят к получению минимальной ДНФ.

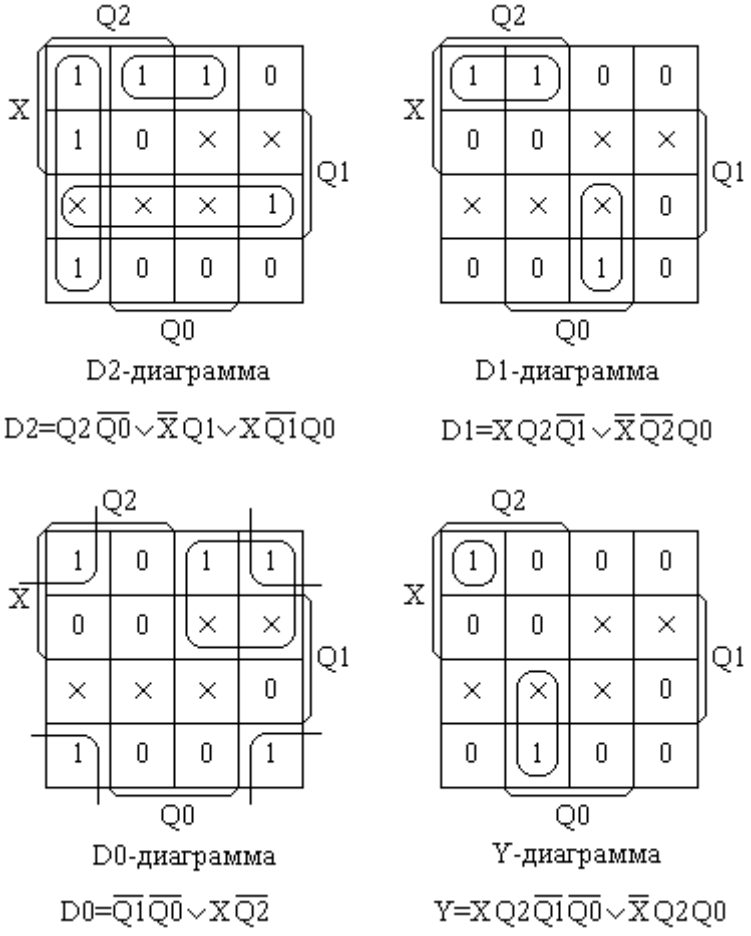


Рис. 2.4. Диаграммы Вейча функций возбуждения и функции *Y*

Для получения минимальной ДНФ с помощью диаграммы Вейча необходимо покрыть все единицы минимальным числом «пра-

вильных» площадей. «Правильная» площадь − это прямоугольник или квадрат, содержащий 2^n клеток диаграммы.

Для не полностью определенных функций отдельные значения, помеченные символом «×», можно включать в покрытие, если это приводит к уменьшению числа площадей и/или к увеличению их размера.

При этом следует помнить, что площади покрытия могут пересекаться, т.е. наборы могут входить в несколько покрытий. Также возможно несколько вариантов минимального покрытия.

Покрытия, составленные по указанным правилам, приведены на рис. 2.4 для функций возбуждения и функции *Y* выхода автомата.

При записи минимальной ДНФ, полученной с помощь покрытий на диаграмме Вейча, руководствуются следующими правилами:

число простых импликант (самых коротких элементарных произведений) равно числу покрывающих площадей;

переменные, которые входят в простую импликанту, определяются переменными, входящими в покрытие только одним своим обозначением (либо без отрицания, либо с отрицанием).

Минимальные ДНФ для нашего примера приведены на рис. 2.4.

Логическая схема автомата

Логическая схема проектируемого автомата, построенная на базовых элементах ПЛИС XC10PC84 (см. приложение 1) по полученным минимальным ДНФ функций возбуждения D-триггеров и функции выхода *Y*, показана на рис. 2.5.

Чтобы обеспечить предварительную установку автомата в начальное состояние «0», в приведенной схеме автомата введен вход *CLR*.

Временная диаграмма работы автомата приведена на рис. 2.6.

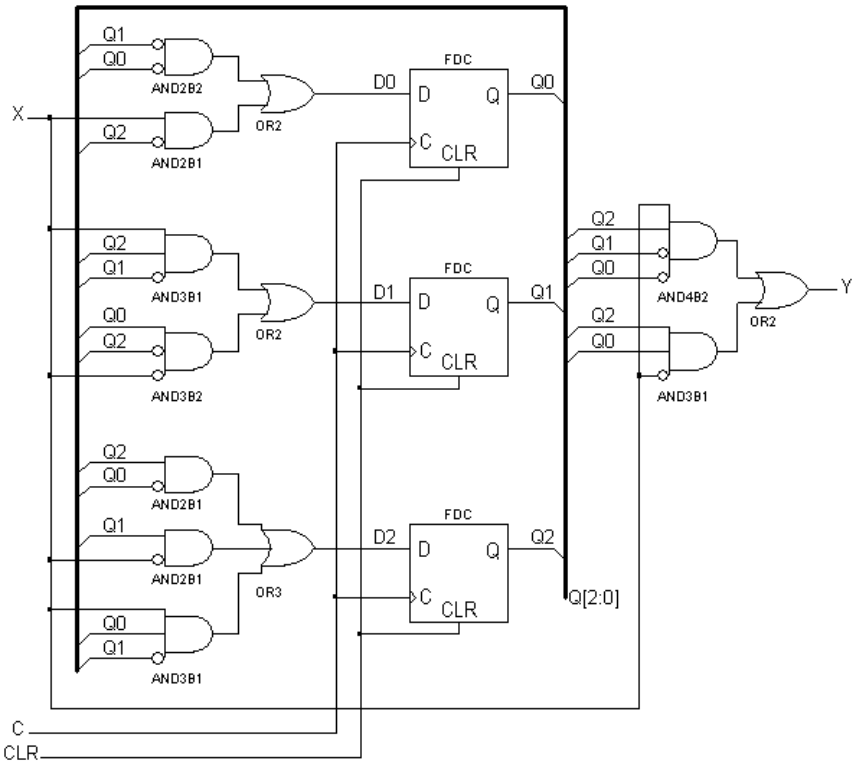


Рис. 2.5. Логическая схема автомата Мили

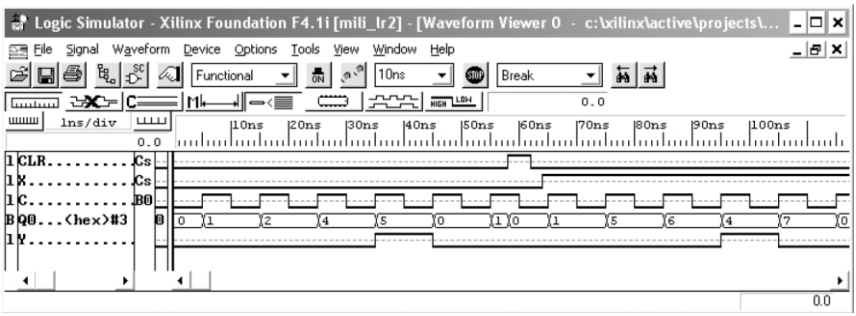


Рис. 2.6. Временная диаграмма работы автомата Мили

Подготовка к выполнению работы

1. Изучить описание лабораторной работы.
2. Получить индивидуальное задание у преподавателя.
3. Составить кодированную таблицу переходов и выхода автомата.
4. Составить таблицу возбуждения триггеров автомата.
5. Выполнить минимизацию функций возбуждения и выхода автомата
6. Построить логическую схему проектируемого цифрового автомата.

Порядок выполнения работы

1. Выполнить ввод спроектированной схемы автомата в редакторе схем системы Xilinx Foundation.
2. Выполнить функциональное моделирование автомата.
3. Продемонстрировать преподавателю работу отлаженного автомата.
4. Сдать преподавателю оформленный отчет в конце занятия.

Отчет по работе

Отчет должен содержать:

1) исходные данные варианта задания;
2) все этапы проектирования автомата Мили;
3) принципиальную схему автомата.

СПИСОК ЛИТЕРАТУРЫ

1. Уэйкерли Дж. Ф. Проектирование цифровых устройств. Том 2. – М.: Постмаркет, 2002. – 528 с.

Лабораторная работа 3

СТРУКТУРНЫЙ СИНТЕЗ СИНХРОННОГО АВТОМАТА МУРА

Цель: изучить структурный синтез синхронного конечного автомата Мура, закон функционирования которого представлен в виде словесного описания; овладеть практическими навыками отработки проектируемого автомата моделированием с использованием профессиональной САПР.

Введение

Понятие «синхронный конечный автомат» было раскрыто в предыдущей работе. Напомним, автомат Мура описывается следующей функцией переходов и функцией выходов:

$$s(t+1) = f[s(t), x(t)],$$
$$y(t) = \varphi[s(t)].$$

Структура синхронного автомата Мура приведена на рис. 3.1.

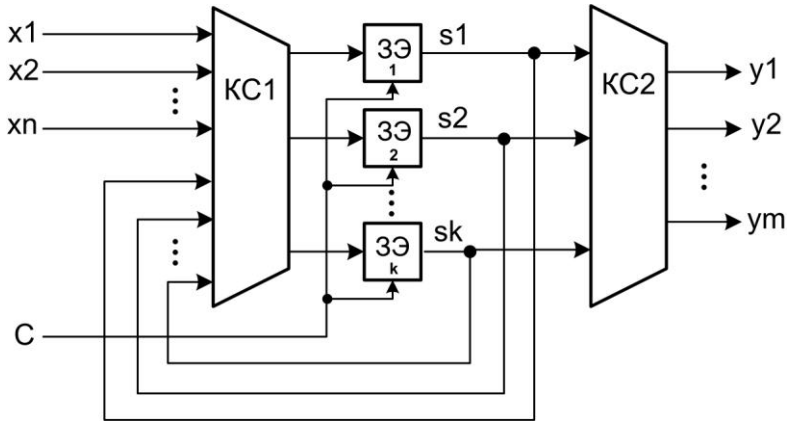


Рис. 3.1. Структура синхронного конечного автомата Мура

Память состояний автомата состоит из k запоминающих элементов (ЗЭ), каждый из которых под действием синхронизирующего

сигнала С может устанавливаться в состояние 0 или 1. Состояние запоминающих элементов определяет текущее состояние автомата.

Комбинационная схема КС1 формирует очередное состояние автомата в момент времени *t*. Это состояние определяется как функция текущего состояния и входного воздействия. Очередное состояние будет записано в запоминающие элементы автомата при поступлении синхросигнала.

Выходные сигналы *Y* автомата Мура определяются комбинационной схемой КС2 как функция текущего состояния автомата *S*.

В синхронных конечных автоматах моменты времени, в которые автомат считывает входные сигналы, определяются принудительно синхронизирующими сигналами. После очередного синхронизирующего сигнала с учётом «считанного» состояния и в соответствии с соотношениями для функционирования автомата происходит переход в новое состояние и выдача сигнала на выходе, после чего автомат может воспринимать следующее значение входного сигнала.

Канонический метод структурного синтеза автоматов, рассмотренный в работе 2, универсален. Он применим как к автоматам Мили, так и к автоматам Мура. Данный метод позволяет свести задачу структурного синтеза автоматов к задаче синтеза комбинационных схем.

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

Задача структурного синтеза синхронного автомата Мура, рассматриваемая в данной работе, заключается в кодировании состояний, задании автомата кодированными таблицами переходов и выходов и синтезе комбинационных схем, формирующих сигналы возбуждения заданного типа запоминающих элементов и выходные сигналы.

Пример

Постановка задачи. Спроектировать синхронный конечный автомат Мура с одним входом *X* и одним выходом *Y*. На вход автомата непрерывно поступает двоичная последовательность. Автомат должен распознать в ней последовательность 10110 и сформировать единичный сигнал на выходе *Y*. После обнаружения данной последовательности автомат прекращает работу. Возобновление работы автомата возможно только после приведения его в исходное состояние. Любые другие двоичные последовательности устанавливают *Y*=0. В качестве элемента памяти использовать JK-триггер.

Идея решения. Таблица переходов и выходов

Перевод словесного (возможно, неоднозначного) описания конечного автомата на обычном языке в формальное табличное описание является творческим процессом и самым важным шагом, так как именно здесь разработчик занимается собственно проектированием.

На этом шаге следует сгенерировать идею решения поставленной задачи. Один из возможных подходов состоит в следующем. Очередному правильному биту поставить новое состояние. Тогда число различных состояний автомата будет равно числу битов в распознаваемой последовательности плюс исходное состояние автомата, т.е. 5+1=6 состояний.

Присвоим каждому новому правильному состоянию автомата букву *S* с порядковым номером (табл. 3.1). Далее составим таблицу переходов и выходов следующим образом. Для каждого текущего состояния автомата будем рассматривать его переход в очередное состояние при входных сигналах *X*=0 и *X*=1.

В начальном состоянии *S*0 автомат ждет прихода первой единицы во входной последовательности. Он остается в начальном состоянии в течение всего времени, пока на вход поступают нули, и переходит в состояние *S*1, когда поступила 1.

Находясь в состоянии *S*1, автомат ждет прихода 0. Если поступает именно он, то автомат переходит в состояние *S*2; если нет, то автомат остается в состоянии *S*1.

Таблица 3.1

Таблица обозначений состояний автомата

Значение	Обозначение состояний
Начальное состояние	*S*0
Принята 1	*S*1
Принято 10	*S*2
Принято 101	*S*3
Принято 1011	*S*4
Принято 10110	*S*5

Пребывая в каждом следующем состоянии, автомат будет переходить дальше, если на вход поступает «правильный» сигнал, и в случае прихода «неправильного» сигнала возвращаться в то состояние, которое соответствует правильным фрагментам уже поступившей двоичной последовательности. Попав в состояние *S*5, когда принята требуемая последовательность, автомат выдает 1 на выходе *Y*. Любые следующие входные сигналы должны оставлять автомат в состоянии *S*5. Возобновление работы автомата возможно только после приведения его в исходное состояние.

Сказанное проиллюстрировано графом переходов автомата (рис. 3.2). Здесь состояния автомата представлены вершинами графа, а переходы между состояниями – дугами между соответствующими вершинами. Каждой дуге графа приписано значение входного сигнала, указывающее, что данный переход автомата осуществляется только при появлении этого входного сигнала. В каждой вершине графа задано значение выходного сигнала *Y*.

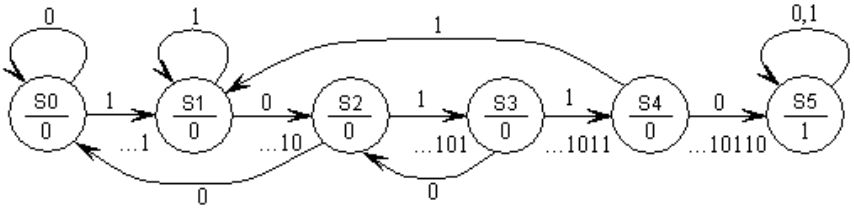


Рис. 3.2. Граф переходов автомата

Таблица переходов и выходов автомата, соответствующая приведенному графу переходов, приведена в табл. 3.2.

Таблица 3.2

Таблица переходов и выходов автомата

Время t		Время t+1	Время t
X	Текущее состояние	Следующее состояние	Y
0	$S0$	$S0$	0
1	$S0$	$S1$	0
0	$S1$	$S2$	0
1	$S1$	$S1$	0
0	$S2$	$S0$	0
1	$S2$	$S3$	0
0	$S3$	$S2$	0
1	$S3$	$S4$	0
0	$S4$	$S5$	0
1	$S4$	$S1$	0
0	$S5$	$S5$	1
1	$S5$	$S5$	1

Составление кодированной таблицы переходов и выходов

Перед составлением кодированной таблицы переходов необходимо определить, сколько потребуется двоичных переменных для представления состояний автомата. Число различных состояний автомата равно 6 (см. табл. 3.1). Шесть состояний автомата можно представить тремя двоичными переменными. Число способов установить соответствие между шестью состояниями нашего автомата и комбинациями состояний из трех двоичных переменных огромно.

При выборе разумного способа кодирования следует руководствоваться практическими принципами, изложенными в [1]. Простейший способ назначения кодов состояний из возможных двоичных комбинаций заключается в использовании первых целых двоичных чисел в порядке двоичного счета.

Код начального (исходного) состояния выберем таким образом, чтобы автомат можно было легко установить в это состояние при

запуске (в типичных схемах это 0). Тогда кодирование состояний автомата будет следующим: *S*0=000, *S*1=001, *S*2=010, *S*3=011, *S*4=100, *S*5=101.

Теперь составление кодированной таблицы переходов и выходов предельно облегчается. Обозначив двоичные разряды кода состояния через *Q*2, *Q*1, *Q*0 и заменив в табл. 3.2 буквенное обозначение состояний автомата на двоичные коды, получим кодированную таблицу переходов и выходов автомата (табл. 3.3).

Кодированная таблица переходов определяет зависимость состояний запоминающих элементов *Q*2(*t*+l), *Q*1(*t*+l) и *Q*0(*t*+l) в момент времени *t*+1 от значения входного сигнала *X* и состояний запоминающих элементов в предыдущий момент времени *t*. Выходной сигнал *Y* определен в зависимости от значения внутренних состояний запоминающих элементов в момент времени *t*.

Таблица 3.3

Кодированная таблица переходов и выходов

Время *t*				Время *t*+1			Время *t*
X	*Q*2	*Q*1	*Q*0	*Q*2	*Q*1	*Q*0	*Y*
0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
1	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0
1	0	1	0	0	1	1	0
0	0	1	1	0	1	0	0
1	0	1	1	1	0	0	0
0	1	0	0	1	0	1	0
1	1	0	0	0	0	1	0
0	1	0	1	1	0	1	1
1	1	0	1	1	0	1	1

Составление таблицы возбуждения автомата

Следующий шаг заключается в составлении *таблицы возбуждения*, т.е. определения закона функционирования комбинационной схемы КС1 (см. рис. 3.1). В этой таблице для каждой комбинации кода состояния и входного воздействия указываются значения сиг-

налов, которые необходимо подать на входы триггеров, чтобы заставить автомат перейти в желаемое следующее состояние с соответствующим кодом. Структура и содержание этой таблицы зависят от типа используемых триггеров (D-, JK-, T-триггеры).

В нашем случае тип триггера задан в условии задачи – это JK-триггер. В библиотеке базовых элементов ПЛИС XC10PC84 синхронный JK-триггер с асинхронной установкой в 0 имеет имя FJKC. Условное графическое обозначения JK-триггера FJKC приведено на рис. 3.3, а таблица переходов дана в табл 3.4.

В этом обозначении: *C* – прямой синхронизирующий вход, *CLR* **(***Clear***)** – асинхронный вход установки триггера в 0, *J* и *K* – логические входы триггера. В таблице переходов JK-триггера: × – произвольное значение, 0/1 – изменение синхросигнала из 0 в 1.

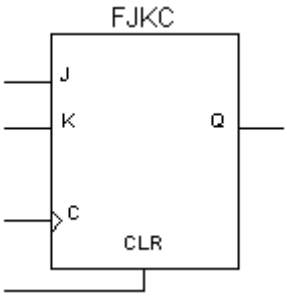


Рис. 3.3. Условное графическое обозначение JK-триггера FJKC

Таблица 3.4

Таблица переходов JK-триггера FJKC

Входы				Выход
CLR	*J*	*K*	*C*	$Q(t+1)$
1	×	×	×	0
0	0	0	0/1	$Q(t)$
0	0	1	0/1	0
0	1	0	0/1	1
0	1	1	0/1	$\bar{Q}(t)$

С помощью кодированной таблицы переходов можно определить для всех трех JK-триггеров способ формирования входных сигналов $J2$ и $K2$, $J1$ и $K1$, $J0$ и $K0$. Для формирования сигналов, поступающих на логические входы триггеров, используются выходные сигналы триггеров и входной сигнал X в момент времени t (см. рис. 3.1).

При синтезе конечного автомата удобно функции возбуждения триггеров автомата формировать с использованием кодированной таблицы переходов автомата и матрицы переходов триггера.

Число строк матрицы переходов для любого триггера равно четырем, что определяется числом возможных переходов триггера из одного состояния в другое, а количество столбцов – числу логических входов триггера. Матрица переходов JK-триггера (она составлена по таблице переходов JK-триггера) выглядит следующим образом:

$Q(t)$		$Q(t+1)$	J	K
0	-	0	0	a_1
0	-	1	1	a_2
1	-	0	a_3	1
1	-	1	a_4	0

Слева от матрицы записаны типы переходов, соответствующие значениям сигналов каждой строки матрицы. Элементы каждой строки матрицы представляют собой значения входных сигналов на логических входах триггера, под воздействием которых триггер переходит из состояния $Q(t)$ в состояние $Q(t+1)$. При этом каждый элемент матрицы может быть равен единице, нулю или являться неопределенным коэффициентом a_i, если значение сигнала на входе не влияет на данный переход триггера.

В таблице возбуждения триггеров автомата следует указать значение сигнала, который необходимо подать на J- и K-входы каждого триггера при всех возможных комбинациях вход/код состояния.

Чтобы облегчить последующий процесс занесения функции возбуждения на диаграмму Вейча, целесообразно:

а) образовать десятичный номер каждого двоичного набора текущего состояния автомата, считая двоичную запись состояния автомата как двоичное число с естественными весами разрядов;

б) затем записать в каждую строчку табл. 3.5 неопределенный коэффициент a с индексами, равными десятичному номеру соответствующего двоичного набора.

В нашем примере кодированная таблица переходов, дополненная функциями возбуждения, имеет вид, указанный в табл. 3.5.

Рассмотрим первую строку таблицы 3.5. Автомат из состояния $Q2=0$, $Q1=0$, $Q0=0$ при входном сигнале $X=0$ должен остаться в исходном состоянии, т.е. для всех триггеров реализуется переход «0-0». В соответствии с первой строкой матрицы переходов JK-триггера в столбцах 8, 10 и 12 табл. 3.5 необходимо записать 0, а в столбцах 9, 11 и 13 – a_0.

При входном сигнале $X=1$ (вторая строка табл. 3.5) автомат из состояния $Q2=0$, $Q1=0$, $Q0=0$ должен перейти в состояние $Q2=0$, $Q1=0$, $Q0=1$, т.е. для триггеров Т2, Т1 реализуется переход «0-0», а для триггера Т0 – переход «0-1». В соответствии с первой строкой матрицы переходов JK-триггера в столбцах 8 и 10 табл. 3.5 необходимо записать 0, а в столбцах 9 и 11 – a_8, а в соответствии со второй строкой матрицы (переход «0-1») в столбцах 12, 13 следует поставить 1 и a_8 соответственно.

Подобным образом заполняют остальные строки табл. 3.5.

Таблица 3.5

Таблица переходов и функций возбуждения JK-триггеров автомата

Время t				Время t+1			Время t						Время t
X	$Q2$	$Q1$	$Q0$	$Q2$	$Q1$	$Q0$	$J2$	$K2$	$J1$	$K1$	$J0$	$K0$	Y
1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	0	0	0	0	0	a_0	0	a_0	0	a_0	0
1	0	0	0	0	0	1	0	a_8	0	a_8	1	a_8	0
0	0	0	1	0	1	0	0	a_1	1	a_1	a_1	1	0
1	0	0	1	0	0	1	0	a_9	0	a_9	a_9	0	0
0	0	1	0	0	0	0	0	a_2	a_2	1	0	a_2	0
1	0	1	0	0	1	1	0	a_{10}	a_{10}	0	1	a_{10}	0
0	0	1	1	0	1	0	0	a_3	a_3	0	a_3	1	0
1	0	1	1	1	0	0	1	a_{11}	a_{11}	1	a_{11}	1	0
0	1	0	0	1	0	1	a_4	0	0	a_4	1	a_4	0
1	1	0	0	0	0	1	a_{12}	1	0	a_{12}	1	a_{12}	0
0	1	0	1	1	0	1	a_5	0	0	a_5	a_5	0	1
1	1	0	1	1	0	1	a_{13}	0	0	a_{13}	a_{13}	0	1

Табл. 3.5, составленная по изложенной методике, определяет функции возбуждения входов всех триггеров. Аргументы этих функций записаны в колонках 1–4. Поскольку значения всех переменных определены для одного и того же момента времени *t*, то функции возбуждения JK-триггеров являются логическими функциями.

Минимизация функций возбуждения и выходов

Минимизацию функций возбуждения и выходов выполним с использованием диаграмм Вейча. Процесс заполнения диаграмм Вейча после создания таблицы возбуждения триггеров по указанным выше правилам предельно облегчается. Используя эталонную диаграмму Вейча (см. рис. 2.3 в описании лабораторной работы 2), занесем неопределенные коэффициенты a_i, в поля, цифровые обозначения которых совпадают с индексами рассматриваемых коэффициентов. Из табл. 3.5 видно, что из 16 возможных состояний используются только 12. Остальные четыре являются запрещенными, они никогда не появляются при правильной работе автомата. Это состояния 0110, 0111, 1110, 1111. Отметим на диаграммах Вейча эти состояния символом «×». Затем выберем значения коэффициентов a_i и × в диаграммах так, чтобы получить минимальные выражения для функций возбуждения (рис. 3.4 и 3.5).

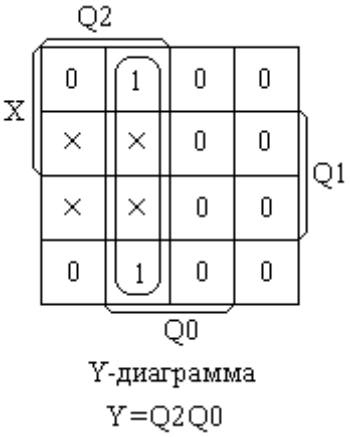


Рис. 3.4. Диаграммы Вейча функции *Y*

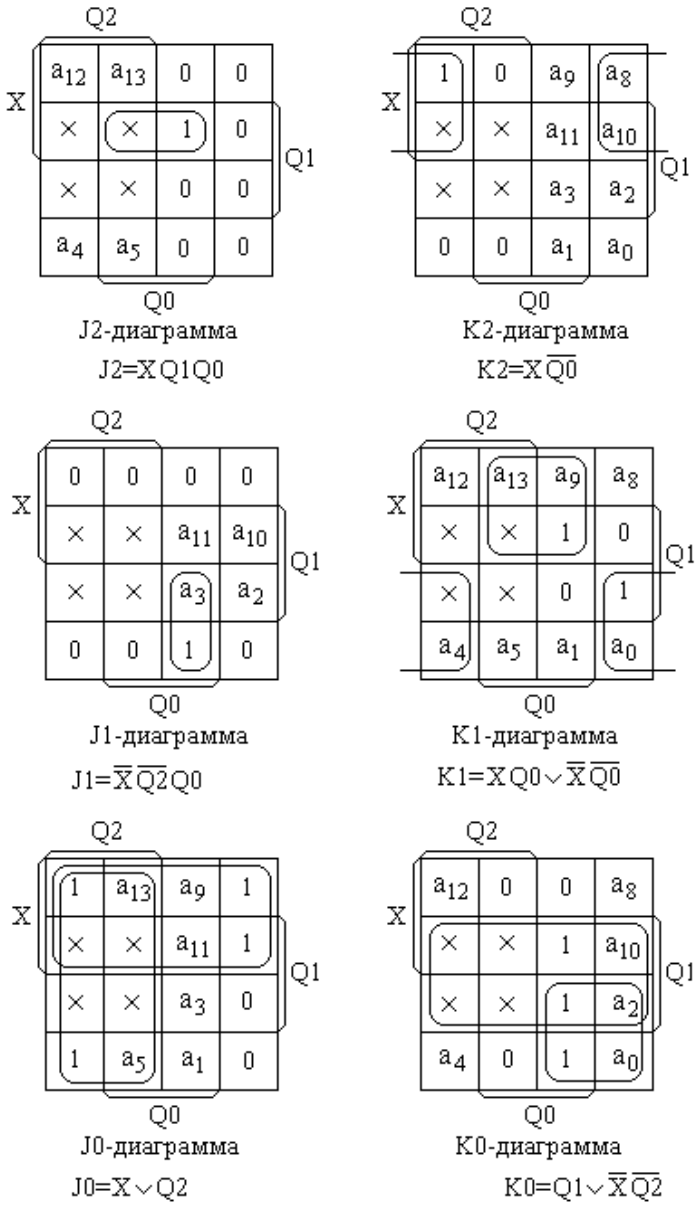


Рис. 3.5. Диаграммы Вейча функций возбуждения JK-триггеров

Логическая схема автомата

Логическая схема проектируемого автомата, построенная на базовых элементах ПЛИС XC10PC84 (см. приложение 1), приведена на рис. 3.6.

Чтобы обеспечить установку автомата в начальное состояние «0», в приведенной схеме автомата введен вход *CLR*.

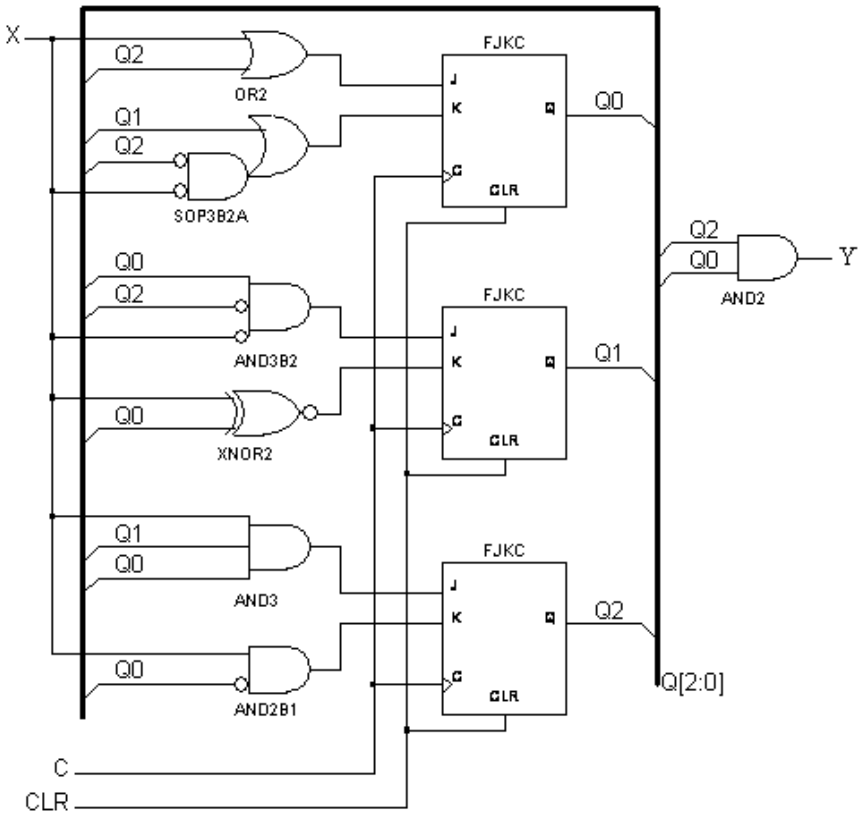


Рис. 3.6. Логическая схема автомата Мура

Временная диаграмма работы автомата Мура по распознаванию заданной входной последовательности приведена на рис. 3.7. На следующей временной диаграмме (рис. 3.8) показана реакция авто-

мата на входную последовательность, которая составлена с целью проверки полноты и правильности работы автомата.

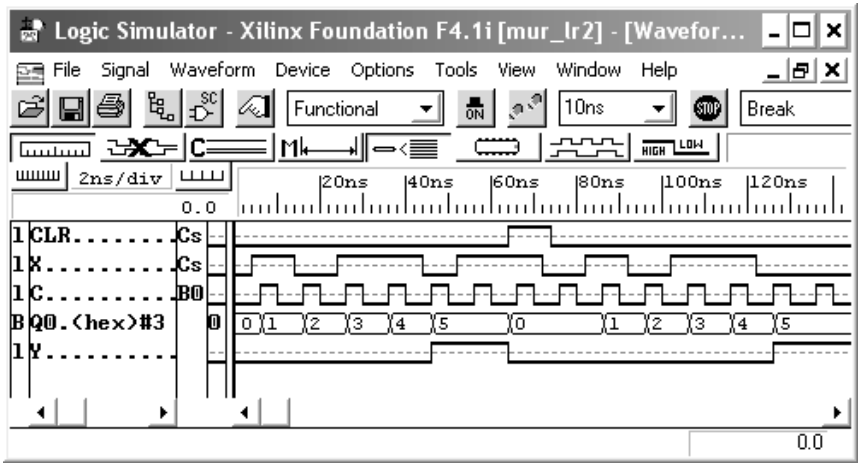


Рис. 3.7. Временная диаграмма работы автомата Мура при распознавании заданной входной последовательности

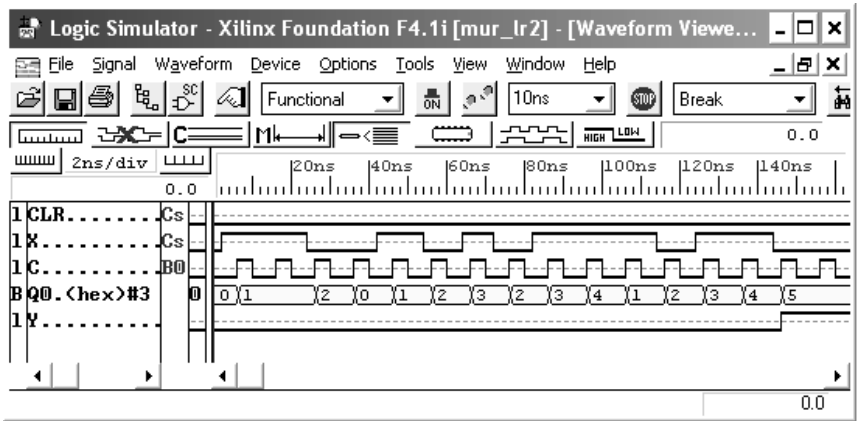


Рис. 3.8. Временная диаграмма реакции автомата Мура на входную последовательность, проверяющую правильность функционирования автомата

Подготовка к выполнению работы

1. Изучить описание лабораторной работы.
2. Получить индивидуальное задание у преподавателя.
3. Составить таблицу переходов и выходов автомата.
4. Составить кодированную таблицу переходов и выходов автомата.
5. Составить таблицу возбуждения триггеров автомата.
6. Выполнить минимизацию функций возбуждения и выходов автомата
7. Построить логическую схему проектируемого цифрового автомата.
8. Составить тестовые последовательности для входа *X* автомата.

Порядок выполнения работы

1. Выполнить ввод спроектированной схемы автомата в редакторе схем системы Xilinx Foundation.
2. Выполнить функциональное моделирование автомата.
3. Продемонстрировать преподавателю работу отлаженного автомата.
4. Сдать преподавателю оформленный отчет в конце занятия.

Отчет по работе

Отчет должен содержать:

1) исходные данные варианта задания;
2) все этапы проектирования автомата Мура;
3) принципиальную схему автомата.

СПИСОК ЛИТЕРАТУРЫ

1. Уэйкерли Дж. Ф. Проектирование цифровых устройств. Том 2. – М.: Постмаркет, 2002. – 528 с.

Лабораторная работа 4

СИНТЕЗ УПРАВЛЯЮЩЕГО АВТОМАТА

Цель: освоить практические приемы создания управляющих алгоритмов в виде графа переходов автомата; приобрести практические навыки работы с подсистемой Finite State Machine Editor; овладеть практическими навыками отработки проектируемых схем как моделированием с использованием САПР, так и макетированием на универсальном лабораторном стенде.

Введение

Управляющий автомат можно рассматривать как устройство, реализующее алгоритм функционирования систем управления, задающий последовательность выполнения тех или иных операций по управлению некоторым объектом. Управляющий автомат реализует разнообразные функции: контролирует значения входных/выходных сигналов, выполняет логические/арифметические операции, поддерживает необходимые значения параметров, обеспечивает связь с датчиками и исполнительными устройствами.

Создание управляющего автомата начинается с его словесного описания или задания его технических характеристик. При проектировании небольших конечных автоматов и устройств средних размеров часто пользуются графом переходов автомата (диаграммами состояний). В этой работе приводится пример разработки с использованием диаграмм состояний. Эта процедура лежит в основе метода, реализуемого средствами автоматизированного проектирования, которые могут создавать логическую структуру из графического представления.

Задание автомата с помощью ориентированного графа – более удобная и компактная форма описания автомата. Состояния автомата изображаются вершинами графа, а переходы между состояниями – дугами между соответствующими вершинами. Каждая дуга отмечается входным сигналом, вызывающим в автомате соответствующий данной ветви переход. Если графом изображается автомат Мили, то выходные сигналы автомата проставляются на дугах графа. Если графом изображается автомат

Мура, то выходные сигналы автомата проставляются около вершин графа.

Ориентированный граф конечного автомата называют *диаграммой состояний автомата* или *графом переходов автомата*.

Разработка графа переходов автомата во многом подобна составлению таблицы состояний. Однако имеется одно принципиальное различие между графом переходов и таблицей состояний автомата, которое делает составление графа переходов проще, хотя при этом риск допустить ошибку увеличивается. Таблица состояний представляет собой исчерпывающий список следующих состояний для каждой комбинации состояние/вход. Никакая неоднозначность не возможна. В графе переходов имеется множество стрелок, у которых надписаны выражения переходов. Даже при наличии многих входов с каждой стрелкой связано только одно выражение перехода. Однако при составлении графа переходов нет гарантии, что выражениями переходов у стрелок, выходящих из данного состояния, все комбинации входных сигналов покрываются точно по одному разу.

Таким образом, самым важным является шаг перехода от словесного описания автомата к его заданию в форме графа переходов, так как именно здесь разработчик занимается собственно проектированием в творческом процессе перевода словесного (возможно, неоднозначного) описания конечного автомата на обычном языке в формальное описание.

Современные системы автоматизированного проектирования цифровой аппаратуры содержат в своем составе средства, которые автоматизируют создание и реализацию конечных автоматов.

В лабораторном практикуме для выполнения задания используется подсистема Finite State Machine Editor (FSM), входящая в состав САПР XILINX FOUNDATION. Подсистема Finite State Machine Editor представляет собой инструмент графического ввода описания цифрового автомата в виде диаграммы (графа) состояний. После графического ввода диаграммы состояний цифрового автомата FSM Editor генерирует макромодель автомата с возможностью последующего воплощения её на программируемой логической интегральной схеме (ПЛИС) лабораторного стенда.

К преимуществам использования подсистемы FSM Editor можно отнести простой и легкий графический ввод описания функционирования автомата в виде диаграммы состояний, автоматическую реализацию всех этапов структурного синтеза автомата. Отсутствие необходимости углубляться в схемотехнику проектируемого автомата в некотором смысле также можно отнести к преимуществам данного подхода – для создания диаграммы состояний автомата достаточно определить алгоритм его работы, при этом от пользователя требуются только базовые знания в области теории автоматов и определенные навыки работы в графическом редакторе FSM Editor. Подход, реализованный в FSM Editor, значительно повышает производительность разработчика.

Пример

Задание. Спроектировать автомат, управляющий стиральной машиной. Стиральная машина работает в трех режимах: залив, стирка, слив.

Стиральная машина начинает работать, когда будет нажата кнопка *pusk*. После этого происходит залив воды до тех пор, пока датчик *d*1 уровня воды не подаст сигнал о заполнения бака стиральной машины. Затем происходит стирка. Эта операция ограничивается с помощью таймера *t***.** Если таймер исправен, то по истечении определенного времени таймер выдает сигнал *t* о завершении стирки и стиральная машина переходит в режим слива воды. Если таймер неисправен, то стиральная машина переходит в состояние «дефект», т.е. в ждущее состояние, которое будет прервано только после ремонта стиральной машины. Из состояния «дефект» стиральная машина не может возвратиться в исходное состояние, поэтому последующее нажатие кнопки *pusk* не приведет к запуску стиральной машины. Из неисправного состояния после ремонта стиральная машина возвращается в исходное состояние по сигналу *reset***.** Слив воды завершается при получении сигнала от датчика *d*2 уровня воды о том, что в баке воды нет. Стиральная машина возвращается в исходное состояние. Каждое состояние стиральной машины должно отображаться на отдельных индикаторах лицевой панели.

Решение поставленной задачи представим в виде последовательного выполнения нижеследующих шагов.

Определение входных и выходных сигналов

В реальных цифровых автоматах информация представляется электрическими сигналами, поступающими на входные контакты (входы) автомата, и сигналами, которые формируются на выходах. При этом на каждом отдельном входе или выходе присутствует два различных уровня электрического сигнала – высокий и низкий. При таком физическом представлении используется двоичное кодирование сигналов. Низкий уровень электрического сигнала ассоциируется с логическим нулём, а высокий – с логической единицей. Сопоставим каждому сигналу, определенному в задании, отдельный вход цифрового автомата. При этом появление входного сигнала будем кодировать единицей, а его отсутствие – нулём.

В соответствии с заданием, составим таблицу, в которой сопоставим каждому внешнему событию входной сигнал, и каждому такому сигналу присвоим имя (табл. 4.1).

В каждом задании предполагается, что проектируемый автомат является синхронным и инициальным. Поэтому управляющий автомат должен иметь два стандартных входа:

C – синхронизирующий вход,

reset – вход установки автомата в начальное состояние.

Таблица 4.1

Входные сигналы управляющего автомата

Имя входного сигнала	Пояснение
pusk	Сигнал начала работы стиральной машины
*d*1	Сигнал с датчика *d*1 «*бак заполнен водой*»
t	Сигнал с таймера о завершении стирки
t_err	Сигнал «*таймер неисправен*»
*d*2	Сигнал с датчика *d*2 «*в баке воды нет*»
C	Вход для синхронизирующих сигналов
reset	Сигнал установки автомата в начальное состояние

Определим выходные сигналы управляющего автомата. Как следует из задания, автомат должен выдавать управляющие сигналы на реализацию очередного режима работы стиральной машины. Кроме того, автомат должен сигнализировать пользователю через индикаторы, в каком режиме работы находится стиральная машина в текущий момент времени.

Исходя из текста задания, можно определить следующий набор индикаторов на лицевой панели стиральной машины: *G* – стиральная машина готова к работе, *Z* – залив воды, *ST* – стирка, *SL* – слив воды, *ERR* – неисправен таймер. Двоичные выходные сигналы автомата, которые формирует управляющий автомат, можно определить, как показано в табл. 4.2.

Таблица 4.2

Выходные сигналы управляющего автомата

Имя выходного сигнала	Пояснение
G	Включить индикатор «Стиральная машина готова к работе»
Z	Включить режим залива воды и индикатор «Залив воды»
ST	Включить режим стирки и индикатор «Стирка»
SL	Включить режим слива воды и индикатор «Слив воды»
ERR	Заблокировать работу стиральной машины и включить индикатор «Неисправен таймер»

Определение необходимого набора состояний автомата

Режимы работы стиральной машины, приведенные в задании, позволяют без особого труда определить состояния автомата. Возможные внутренние состояния управляющего автомата приведены в табл. 4.3.

Таблица 4.3

Множество состояний управляющего автомата

Состояние	Пояснение
*S*0	Начальное состояние автомата. Стиральная машина готова к работе
*S*1	Режим залива воды
*S*2	Режим стирки
*S*3	Режим слива воды
*S*4	Заблокирована работа стиральной машины. Неисправен таймер

Составление графа переходов

После определения входных, выходных сигналов и множества внутренних состояний можно приступить к созданию графа переходов автомата. Создание графа переходов творческая и неформализованная процедура и в каждом отдельном случае и каждым исполнителем может выполняться произвольно.

В данном случае вначале изобразим вершины графа, которые будут представлять состояния автомата (см. табл. 4.3). Поскольку каждое состояние автомата определяет и соответствующий выходной сигнал, то выходные сигналы автомата проставим около вершин графа. Таким образом, проектируемый автомат является автоматом Мура. (Заметим, что в заданиях также предполагается проектирование автомата Мура.)

Этот шаг построения графа переходов автомата представлен на рис. 4.1. Отметим, что при записи значений выходных сигналов использовался синтаксис, принятый в редакторе FSM Editor.

Синтаксис выражений должен быть следующим:

- оператор присваивания обозначается составным символом <=,
- константы заключаются в одинарные кавычки, например, ‘0’,
- выражения отделяются друг от друга символом точкой с запятой.

В выражениях можно использовать логические операторы, определенные в табл. 4.4.

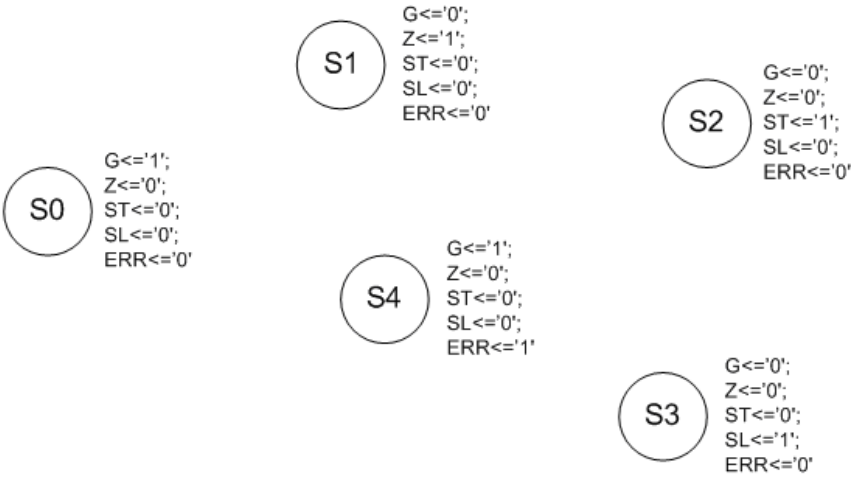


Рис. 4.1. Задание состояний и выходных сигналов автомата

Таблица 4.4

Логические операторы

not	Инверсия
and	И
or	ИЛИ
nand	И-НЕ
nor	ИЛИ-НЕ
xor	ИСКЛЮЧАЮЩЕЕ ИЛИ
xnor	ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ

У оператора *not* самый высокий приоритет, так что можно не заключать в скобки подвыражение типа *not A*1.

Логические операторы *and, or, nand, nor, xor, xnor* имеют одинаковое старшинство и выполняются слева направо в выражениях. В сложных логических выражениях порядок выполнения операторов регулируется скобками.

В соответствии с логикой работы стиральной машины и определенными выше состояниями зададим все переходы для автомата (рис. 4.2). Автомат будет находиться в одном из четырёх состояний *S*0,*S*1,*S*2,*S*3 до тех пор, пока соответствующий входной сигнал не переведет автомат в очередное состояние.

При отказе таймера возможны две ситуации:

а) отказ произошел во время работы стиральной машины до начала режима стирки или во время стирки (выявление отказа возможно в режиме стирки),

б) отказ произошел после окончания режима стирки.

В первом случае управляющий автомат перейдет в состояние *S*4 (неисправен таймер) из состояния *S2* (режима стирки). Во втором случае автомат перейдет в состояние *S*4 из начального состояния *S*0 только при повторном запуске стиральной машины (см. рис. 4.2).

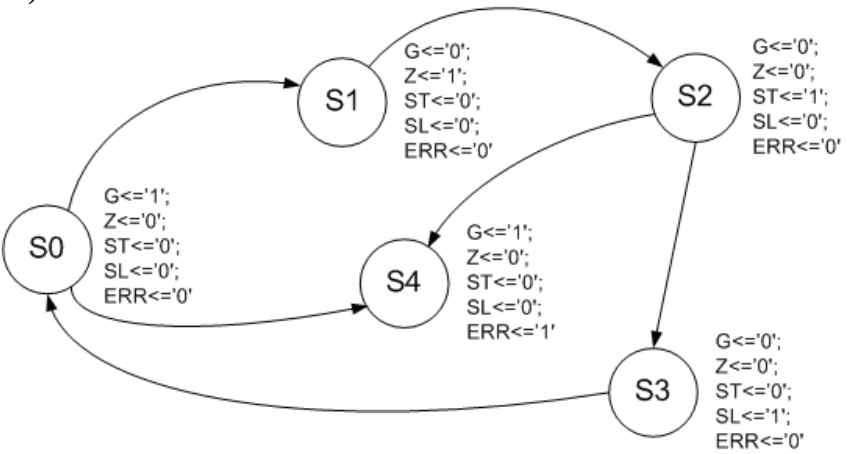


Рис. 4.2. Результат задания переходов

Далее необходимо каждую дугу графа пометить условием, которое вызывает в автомате соответствующий данной ветви переход. Условие перехода автомата из одного состояния в другое задаётся истинностью или ложностью (равенством 1 или 0) логического выражения или переменной, которые соответствуют появлению тех или иных входных сигналов. Выражение заключается в круглые скобки, ставится знак равенства, затем пишется 0 или 1, заключённые в одинарные кавычки. Например:

$$(not\ E1\ and\ E2) = \text{‘1’}.$$

В выражениях можно использовать логические операторы, указанные в табл. 4.4.

Если условие перехода определяется истинностью или ложностью переменной, то запись такого условия выглядит следующим образом:

$$E2 = \text{‘0’}.$$

На рис. 4.3 приведен граф переходов управляющего автомата с заданными условиями переходов. Следует отметить, переход автомата в то или иное состояние определяется, как ранее условились, единичным значением входного сигнала. Автомат будет находиться в том или ином состоянии до тех пор, пока на вход автомата не поступит определенный входной сигнал. Начало работы стиральной машины по сигналу *pusk* возможно только при исправном таймере – условие (*not t_err and pusk*)=’1’.

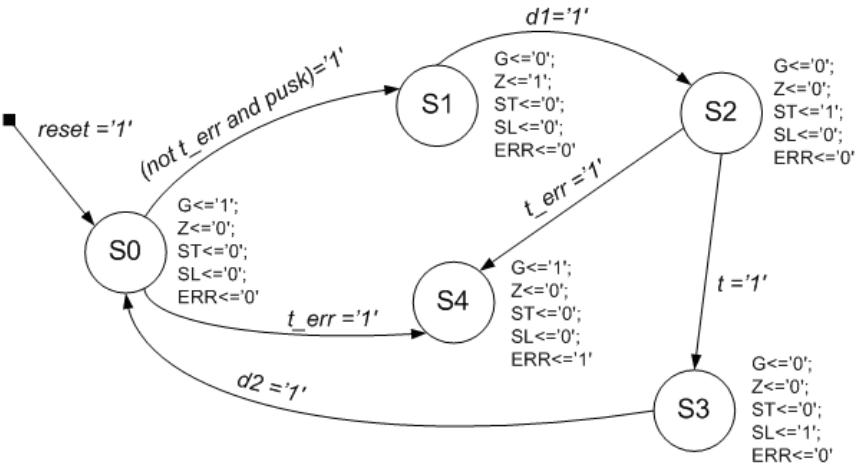


Рис. 4.3. Граф переходов управляющего автомата

Здесь же на графе показан и установочный вход *reset*, единичный сигнал которого переводит автомата в начальное состояние.

Реализация проекта управляющего автомата

Следующие шаги по выполнению задания выполняются в лаборатории. Они включают следующую последовательность действий:

- ввод графа переходов конечного автомата,

- создание макроэлемента автомата,
- моделирование макроэлемента автомата,
- реализацию проекта автомата на ПЛИС.

Реализация данных шагов подробно изложена в приложении 2. Ниже приведены результаты выполнения первых трех из перечисленных шагов по выполнению задания.

Граф переходов управляющего автомата, введенного в редакторе FSM Editor, приведен на рис. 4.4.

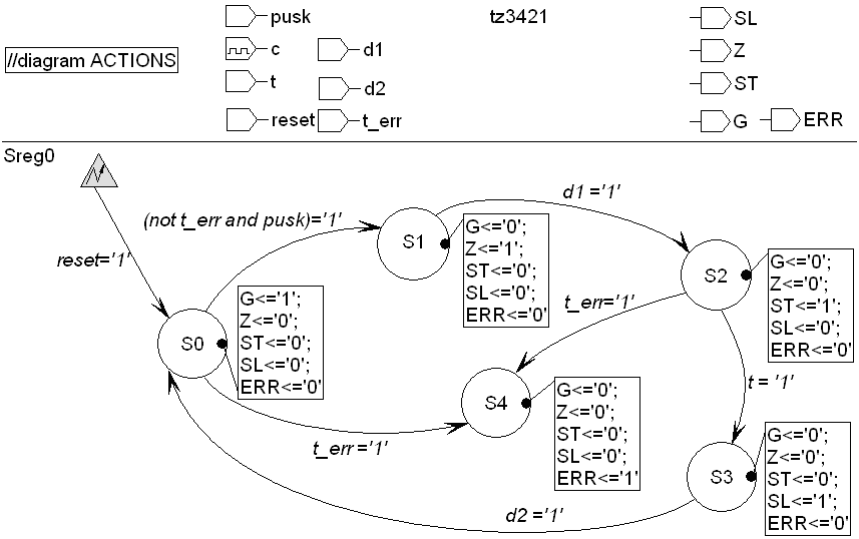


Рис. 4.4. Граф переходов управляющего автомата

Созданный системой макроэлемент управляющего автомата приведен на рис. 4.5.

Результаты моделирования созданного макроэлемента управляющего автомата показаны на рис. 4.6-4.7.

На рис. 4.6 приведена временная диаграмма рабочих режимов автомата, управляющего стиральной машиной.

Работа управляющего автомата при отказе таймера показана на временной диаграмме рис. 4.7. Возвращение стиральной машины после ремонта в рабочее состояние проиллюстрировано также на временной диаграмме рис. 4.7.

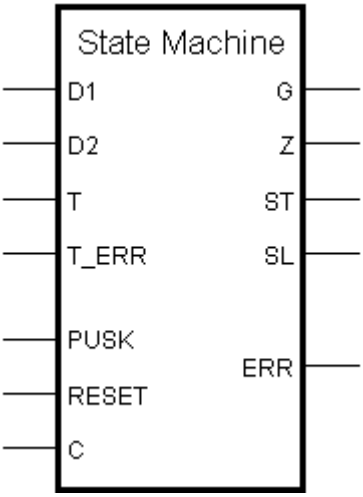


Рис. 4.5. Макроэлемент управляющего автомата

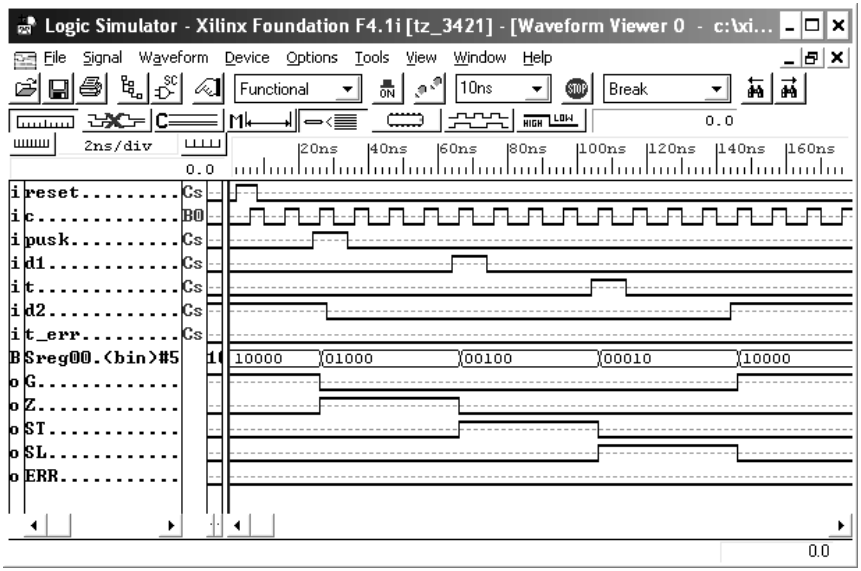


Рис. 4.6. Временная диаграмма работы автомата. Рабочие режимы автомата

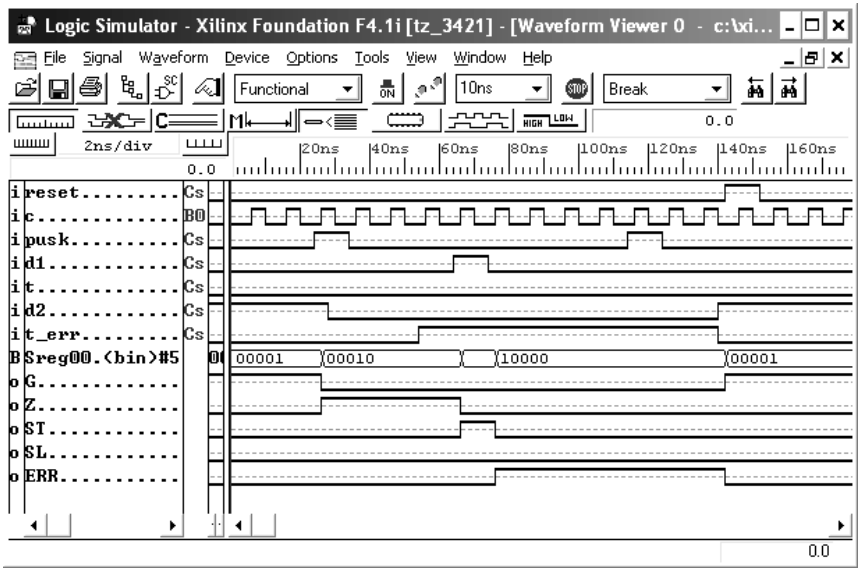


Рис. 4.7. Временная диаграмма работы автомата. Отказ таймера в рабочем режиме

Анализ результатов моделирования свидетельствует о правильности выполненного проектирования управляющего автомата.

Подготовка к выполнению задания

1. Изучить описание лабораторной работы.
2. Получить индивидуальное задание у преподавателя.
3. Составить в форме таблицы описания входных, выходных сигналов и необходимого набора состояний автомата.
4. Построить граф переходов проектируемого цифрового автомата.

Порядок выполнения работы

1. Ввести граф переходов автомата в подсистеме FSM Editor САПР Xilinx Foundation.
2. Создать макроэлемент автомата.
3. Выполнить функциональное моделирование автомата в соот-

ветствии с построенной лентой. Продемонстрировать преподавателю работу отлаженного автомата.

4. Выполнить загрузку проекта в ПЛИС стенда и провести проверку автомата на макете.

5. Сдать преподавателю оформленный отчет в конце занятия.

Отчет по работе

Отчет должен содержать:

1) оригинал текста задания;
2) описание входных, выходных сигналов и состояний автомата;
3) граф переходов автомата.

СПИСОК ЛИТЕРАТУРЫ

1. Уэйкерли Дж. Ф. Проектирование цифровых устройств. Том 2. – М.: Постмаркет, 2002. – 528 с.

Лабораторная работа 5

СИНТЕЗ АВТОМАТА РАСПОЗНАВАНИЯ ДЕЛИМОСТИ ДВОЧНЫХ КОДОВ БОЛЬШОЙ РАЗМЕРНОСТИ

Цель: освоить принципы синтеза конечных автоматов для решения различных прикладных задач на примере проверки делимости больших бинарных чисел; закрепить приемы синтеза простейших цифровых устройств; развить навыки рационализации, комплексного подхода при проектировании, формальной оценки сложности проектирования и затрат на аппаратную реализацию цифровых устройств; закрепить практические навыки отработки и верификации проектируемых цифровых устройств посредством моделирования с использованием профессиональной САПР.

Введение

Детерминированные конечные автоматы (ДКА) могут применяться для решения различных прикладных задач. Одной из сфер их применения является распознавание соответствия цифровых сигналов определенным критериям. Например, их можно использовать для проверки делимости больших чисел, представленных в определенной системе счисления, которые поступают на вход некоторого цифрового устройства, на заданное число, представленное в той же системе счисления. Стоит отметить, что существуют альтернативные способы решения подобных задач, которые в ряде случаев являются более эффективными, но, с академической точки зрения, подобная задача является хорошим примером для приобретения и закрепления навыков синтеза конечных автоматов и применения их для синтеза простейших цифровых устройств.

В данной лабораторной работе не рассматриваются методы формального синтеза детерминированных конечных автоматов. Синтез структуры детерминированного конечного автомата будет проводиться, исходя из прикладной задачи. Оценка сложности проектирования и затрат на аппаратную реализацию цифрового устройства будет проводиться, исходя из количества состояний ДКА и количества переходов между ними.

Теоретические основы, основные этапы и подходы к проектированию

Пусть дано большое число *N*, состоящее из ||*N*|| разрядов, и число *K*. Необходимо проверить, делится ли *N* на *K* без остатка.

Если размерность *N* заранее не известна, и код числа последовательно поступает на анализирующее устройство, начиная со старших разрядов, то, при сравнительно небольших *K*, данная задача может быть эффективно решена при помощи ДКА (за количество тактов *O(||N||)*, параллельно с поступлением кода на анализирующее устройство). Накладные временные расходы в этом случае составляют *O(1)*, т.е. не зависят от размера *N*. При этом число состояний данного автомата не превышает *K*, а число переходов из каждого состояния не превышает *M*, где *M* – основание системы счисления, в которой задаются числа *N* и *K*.

Очевидно, что каждое состояние автомата, проверяющего делимость, соответствует остатку от деления старшей (уже проанализированной) части числа *N* на *K*. Такой автомат фактически реализует классический алгоритм поразрядного деления, начиная со старших разрядов. Число *N* делится на *K*, если автомат при анализе последнего разряда числа *N* переключится в состояние, соответствующее остатку от деления, равному 0. Если автомат находится в любом другом состоянии, то данное число *N* нацело на *K* не делится.

Синтез подобного автомата сводится к определению переходов из текущего состояния $S_i(t)$ в следующее состояние $S_j(t+1)$ в зависимости от текущего анализируемого разряда числа N_k.

Условное графическое обозначение синтезируемого автомата приведено на рис. 5.1.

На вход *DATA* поступает число *N* со старших разрядов в заданном представлении. На вход *READ* поступает признак наличия корректных данных на входе *DATA* (0 – данные на входе *DATA* игнорируются, 1 – данные на входе *DATA* анализируются конечным автоматом). На вход *CLOCK* поступают синхроимпульсы. На вход *RESET* подается сигнал сброса конечного автомата в начальное состояние. На выходе *RESULT* формируется признак делимости числа *N* на *K* для текущего разряда *N* (1 – делится без остатка, 0 – не делится без остатка). При этом, так как разрядность числа заранее

не известна, признак делимости формируется для всех промежуточных результатов. При необходимости данное ограничение легко обходится за счет введения дополнительной управляющей логики в проектируемое устройство.

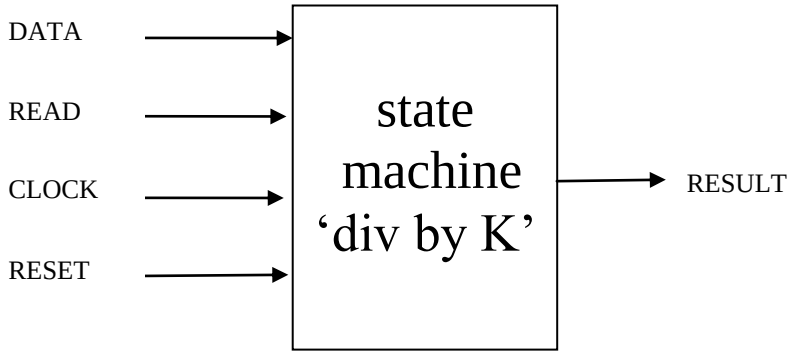


Рис. 5.1. Условное графическое обозначение автомата проверки делимости на *K*

В случае если число *K* представимо в виде $K = k_1 \times k_2 \times \ldots \times k_m$, где k_i – взаимно простые числа, то можно минимизировать количество состояний ДКА (*D*), разбив его на *m* отдельных ДКА ($D_1, \ldots D_m$), которые проверяют число *N* на делимость на k_i, и комбинационную схему с *m* входами и одним выходом, реализующую логическую функцию И, на вход которой подаются выходы с D_i. При этом суммарное количество состояний будет равно $k_1+k_2+\ldots+k_m$, а количество переходов между состояниями не превышает $M \times (k_1+k_2+\ldots+k_m)$, что, в общем случае, значительно меньше, чем количество состояний и переходов между ними в исходном конечном автомате ($k_1 \times k_2 \times \ldots \times k_m$ и $M \times (k_1 \times k_2 \times \ldots \times k_m)$ соответственно). Например, для числа *K*=42 схема проверки делимости будет выглядеть так, как представлена на рис. 5.2. При этом суммарное количество состояний автоматов будет равно 12 = 7 + 3 + 2 вместо 42.

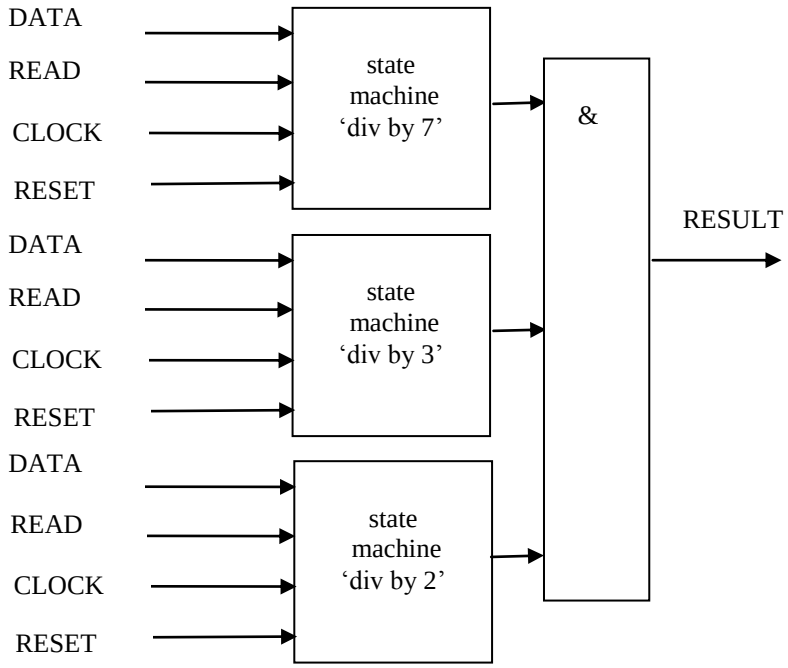


Рис. 5.2. Схема проверки делимости числа на 42

В случае если число K представимо как $K= k_1 \times k_2 \times \ldots \times k_m$, где $k_1 = k_2 = \ldots = k_m$, то для сокращения числа состояний конечного автомата можно воспользоваться следующим приемом: преобразовать конечный автомат в устройство, которое, помимо результата проверки делимости, возвращает и результат деления; соединить устройства последовательно, при этом сигнал корректности данных и результат делимости должны передаваться на очередной этап обработки с задержкой в один такт (задержка реализуется при помощи D-триггеров). На рис. 5.3 представлена схема проверки делимости числа на 25: результат целочисленного деления первой ступени в виде последовательного кода (выход DNEXT) подается на вход второго автомата.

Отметим, что количество состояний автомата для устройств, формирующих и не формирующих результат деления, одинаково;

результат деления формируется при выполнении переходов, т.е. зависит не только от текущего состояния, но и от предыдущего. Следовательно, при проектировании данного устройства, во-первых, надо описать дополнительно $M \times K$ условий формирования сигнала, т.е. возрастает сложность проектирования. Во-вторых, результат нужно сохранить в буферных регистрах (опция Registered, при указании типа выхода в FSM Editor), т.е. в данном автомате необходим один дополнительный элемент памяти.

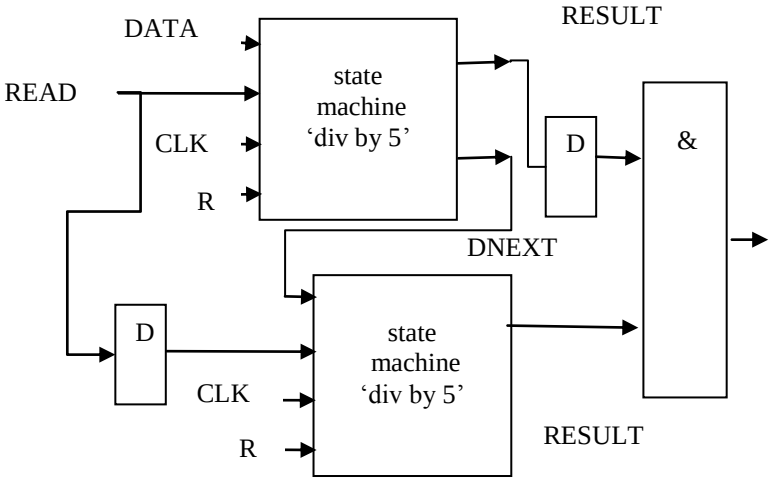


Рис. 5.3. Схема проверки делимости числа на 25

В случае если один из компонентов, на которые раскладывается K, является основанием системы счисления, в которой представлены числа (либо является степенью основания системы счисления), то целесообразно использовать внедрение проверки на делимость на этот компонент в принимающее состояние одного из ДКА для компонента разложения K на взаимно простые сомножители. В этом случае суммарное количество состояний автомата уменьшается на $k_M - \log_M(k_M)$, где k_M – член разложения K на простые сомножители, являющийся степенью M. В этом случае, например, схема проверки делимости на 42 будет содержать 11 = 7 + (3 + 1) состояний вместо 42. Данная схема представлена на рис. 5.4.

Следует отметить, что при подобном подходе к синтезу возникают сложности с описанием логики работы автоматов, формирующих и результат деления. В данном случае результат деления нельзя будет выразить исключительно как зависимость от предыдущего и текущего состояния, так как автомат фактически реализует деление на сомножитель, не кратный основанию системы счисления, а анализ на делимость на оставшуюся часть сомножителя проверятся только в расширенном принимающем состоянии. Поэтому автоматы с расширенным состоянием используются только на последней ступени, где не требуется формирования результата деления, а нужен лишь признак делимости.

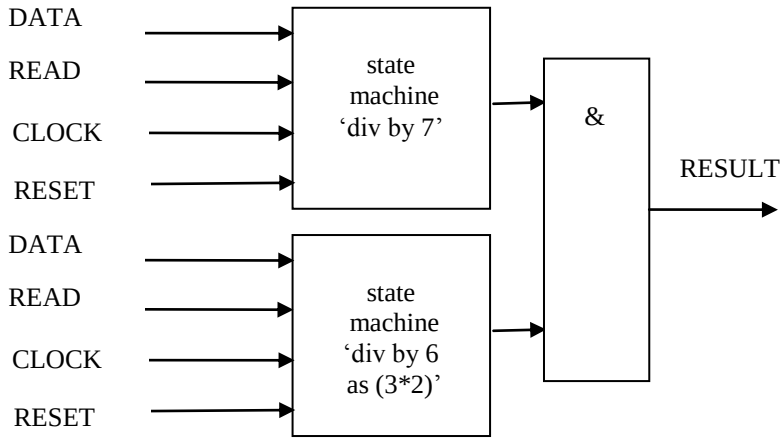


Рис. 5.4. Схема проверки делимости числа на 42 (оптимизированный вариант)

Пример

Постановка задачи. Спроектировать устройство, определяющее, делится ли нацело на 36 двоичный код большой размерности, поступающий последовательно со старших разрядов (число 0 также принимается устройством).

Разбиение на множители и выбор варианта реализации

Вначале необходимо определить элементарные ДКА на которые можно разбить данное устройство. Так как 36 = 2 × 2 × 3 × 3, то устройство, проверяющее делимость, можно спроектировать различными способами. Например, составить из четырех автоматов, объединенных попарно последовательно (2×2, 3×3), либо в виде двух автоматов, объединенных последовательно (6×6), без расширения принимающего состояния или с расширением ((6)×(3×2), либо 3×(3×2×2)). При этом при разных вариантах реализации затраты на синтез и характеристики полученного устройства будут различны.

Поясним принцип получения данных оценок на примере следующей схемы разбиения: (6)'×(3×2). Символ «'» означает, что первая ступень автомата, помимо результата оценки делимости, также должна сформировать и результат деления проанализированных разрядов на 6, т.е. автомат в данном случае является более сложным устройством, чем автомат, реализующий вторую ступень.

Отметим, что при оценке сложности проектирования учитываются затраты на проектирование каждого устройства только один раз, так как сгенерированные макроэлементы можно использовать повторно. Также, если было спроектировано устройство проверки делимости на K' с формированием результата деления, и необходимо спроектировать еще и устройство проверки делимости на K без формирования результата деления, то последнее не учитывается при оценке сложности проектирования.

Представим $K = 36$, как $K = k_1 \times k_2 \times k_3 = 6 \times 3 \times 2$.

Тогда количество проектируемых состояний первой ступени равно $||S_1|| = k_1 = 6$; количество состояний во второй ступени $||S_2|| = k_2 + \log_2 k_3 = 3 + 1 = 4$; суммарно $||S|| = ||S_1|| + ||S_2|| = 10$.

Количество проектируемых переходов в автомате в первой ступени равно $||L_1|| = ||S_1|| \times M = 6 \times 2 = 12$; количество переходов во второй ступени $||L_2|| = ||S_2|| \times M = 4 \times 2 = 8$; суммарно $||L|| = ||L_1|| + ||L_2|| = 20$. При этом при переходах в автомате первой ступени необходимо еще и формировать результат деления, т.е. для каждого перехода надо описать логику его формирования, что усложняет синтез автомата; количество таких переходов соответствует количеству переходов в первой ступени $||L'|| = ||L_1|| = 12$.

Объем памяти оценивается из следующих соображений: номер текущего состояния автомата хранится в двоичном регистре, количество разрядов регистров является минимальным целым числом, большим либо равным $\log_2(S)$, где S – количество состояний автомата. Кроме того, для сигналов, формируемых по переходам, выделяется дополнительная ячейка памяти для их фиксации (для автоматов, формирующих результат деления). Если используется устройство с несколькими ступенями, то для каждой i-й ступени необходимо формировать задержку в передаче сигнала *READ* в i–1 тактов, и результата для логики формирования признака делимости результата в $||P|| - i$, где $||P||$ – количество ступеней, т.е. количество дополнительных ячеек памяти составляет $2 \times (||P|| - 1)$.

При этом для количества ячеек памяти V в устройстве получим следующее соотношение: $||V|| = ||6'|| + ||3\times2|| + 2 \times (||P|| - 1) = = [\log_2 6] + 1 + [\log_2 4] + 2 \times (2 - 1) = = 3 + 1 + 2 + 2 = 8$.

Задержка в формировании результата соответствует

$Latency = ||P|| - 1 = 1$.

Оценки сложности синтеза, затрат на оборудование и задержек в получении результата как для рассмотренного выше, так и ряда других вариантов реализации устройства проверки делимости на 36 сведены в табл. 5.1.

Для оценки сложности синтеза можно ввести следующую функцию:

$Complex\ (Device) = a \times ||S|| + b \times ||L'|| + c \times ||P||$.

Коэффициенты трудоемкости a, b и c могут быть получены при помощи формальной оценки трудоемкости этапов разработки, при помощи экспертных оценок или, например, методов машинного обучения при анализе статистики реализованных проектов.

Аналогично, функцию для оценки качества разработки можно представить как:

$Quality\ (Device) = d \times ||V|| + e \times Latency$

Весовые коэффициенты (d и e) могут быть получены способами, аналогичными указанным выше.

Тогда для выбора решения для реализации можно воспользоваться следующей оценочной функцией:

$Cost(Device) = Complex\ (Device) \times Quality\ (Device)$

Стоит отметить, что функции *Complex(Device)*, *Quality (Device)* и *Cost (Device)* могут быть заданы и в любом другом виде (функциями более высоких порядков, экспоненциальными и т.п.), т.е. вид функции определяется, опять же, экспертом или при помощи специальных алгоритмов, рассмотрение которых находится вне рамок данной лабораторной работы.

Таблица 5.1

Оценка сложности синтеза и накладных расходов на реализацию

Вариант	Состояния	Переходы с логикой	Память	Задержка
36	36	0	$[\log_2 36] = 6$	0
(2×2), 9	(2 + 1) + 9 = 12	0	$[\log_2 3] + [\log_2 9] = 6$	0
2’×2’×3’×3	2 + 3 = 5	(2 + 3) × 2 = 10	$2 \times [\log_2 3] + 2 \times [\log_2 2] + 2 \times 3 + 3 = 15$	3
2’×2, 3’×3	2 + 3 = 5	(2 + 3) × 2 = 10	$2 \times [\log_2 2] + 1 + 2 \times [\log_2 3] + 1 + 2 \times 1 = 10$	1
(2×2), 3’×3	(2 + 1) + 3 = 6	3 × 2 = 6	$[\log_2(2 + 1)] + [\log_2 3] \times 2 + 1 + 2 \times 1 = 9$	1
6’×6	6	6 × 2 = 12	$2 \times [\log_2 6] + 2 \times 1 + 1 = 9$	1
6’× (3×2)	6 + (3 + 1) = 10	6 × 2 = 12	$[\log_2 6] + 1 + [\log_2(3 + 1)] + 2 \times 1 = 8$	1
3’, (3×4)	3 + (3 + 2) = 8	3 × 2 = 6	$[\log_2 3] + 1 + [\log_2(3 + 2)] + 2 \times 1 = 8$	1

Очевидно, что при задании функции *Cost (Device)* способом, определенном выше, наиболее перспективным для реализации вариантом будет вариант с наименьшей стоимостью.

Рассчитаем функцию стоимости для выше рассмотренных вариантов реализации. При этом пусть коэффициенты для оценки трудоемкости разработки выбраны некоторым экспертом следующим образом $a = 10$, $b = 1$, $c = 3$, а весовые коэффициенты для оценки качества устройства $d = e = 1$. Результаты расчетов приведены в табл. 5.2.

Таблица 5.2

Оценка стоимость вариантов реализации

Вариант	Сложность	Качество	Стоимость
36	363	6	2178
(2×2), 9	123	6	738
2’×2’×3’×3	72	18	1296
2’×2, 3’×3	66	11	726
(2×2), 3’×3	**72**	**10**	**720**
6’×6	78	10	780
6’× (3×2)	118	9	1062
3’, 3×4	92	9	828

Согласно полученным оценкам, наиболее перспективным для реализации по соотношению характеристик получаемого устройства и трудозатратам на его проектирование является вариант анализатора, состоящего из двух параллельно работающих устройств – проверки делимости на 4 (с расширением принимающего состояния) и проверки делимости на 9 (состоящий из двух ступеней, реализующих проверку делимости на 3). Реализуем выбранный вариант.

Синтез примитивных автоматов для проверки на делимость

Сначала синтезируем автомат для проверки делимости на 3, затем на основе его построим автомат, который будет возвращать также и результат деления.

В автомате проверки делимости на 3 будет три состояния, соответствующие остаткам от деления – 0, 1 и 2. Составление матрицы переходов между состояниями по управляющим воздействиям *DATA* и *READ* не составляет особой сложности, данная матрица приводится в табл. 5.3.

Начальным принимающим состоянием автомата является *S*1, в которое автомат переходит по асинхронному входу *RESET*. Значения на выходе *RESULT* можно задать согласно табл. 5.4.

Графическое представление логики работы спроектированного автомата в нотации Xilinx Foundation FSM Editor’а приведено на рис. 5.5. Следует отметить, что при описании конечных автоматов не обязательно указывать все переходы; необходимы только те пе-

реходы, по которым автомат меняет свое состояние, либо происходят изменения в сигналах на выходах автомата. Соответственно, переходы 1 и 6 для состояний *S*1 и *S*3 на графе переходов, отсутствуют.

Таблица 5.3

Матрица переходов для автомата проверки делимости на 3

Из \ В	S1 (00)	S2(01)	S3(10)
S1(00)	1: not DATA and READ	2: DATA and READ	–
S2(01)	3: DATA and READ	–	4: not DATA and READ
S3(10)	–	5: not DATA and READ	6: DATA and READ

Таблица 5.4

Значение выхода RESULT в зависимости от состояния автомата проверки делимости на 3

Состояние	S1 (00)	S2(01)	S3(10)
RESULT	RESULT<=’1’	RESULT<=’0’	RESULT<=’0’

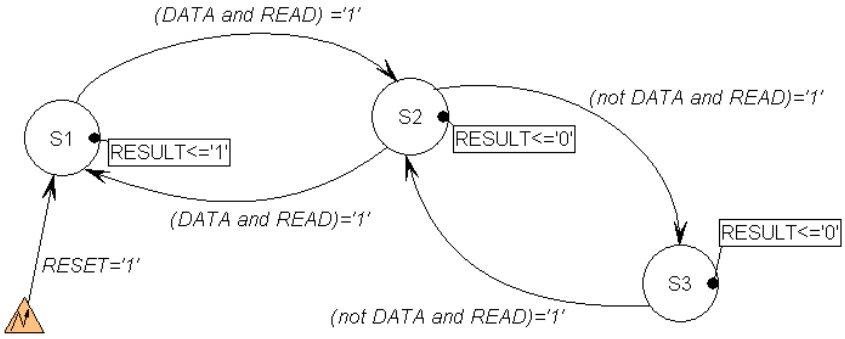


Рис. 5.5. Граф переходов автомата проверки делимости на 3

Макроэлемент спроектированного автомата приведен на рис. 5.6. Временная диаграмма работы автомата проверки делимости на 3 приведена на рис. 5.7. На вход схемы подаются три тесто-

вых воздействия – числа 9, 13 и 3. Перед каждым тестом схема сбрасывается сигналом RESET.

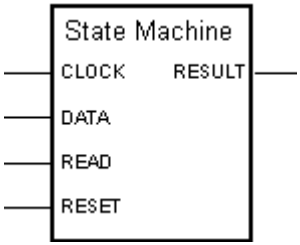


Рис. 5.6. Макроэлемент автомата для проверки делимости на 3

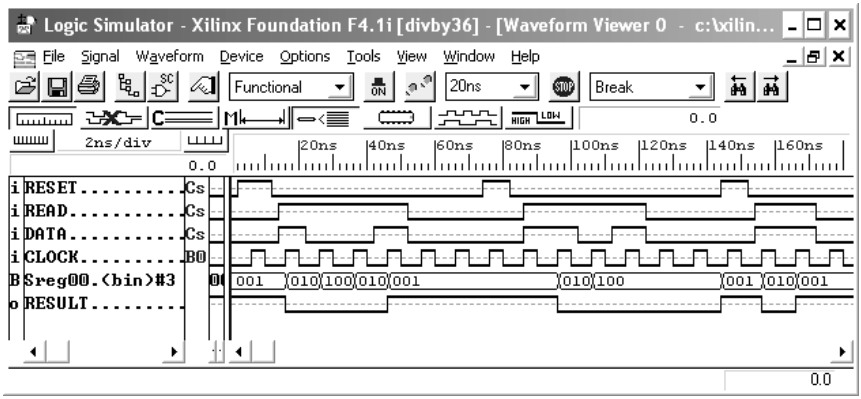


Рис. 5.7. Временная диаграмма работы автомата проверки делимости на 3

Расширим функционал данного автомата, добавив возможность формирования результата деления на 3. Для этого, во-первых, добавим буферизованный выход *DNEXT*; во-вторых, опишем правила формирования сигналов на данном выходе при выполнении переходов. Результат формируется, как результат деления суммы остатка, соответствующего состоянию, в котором находился автомат, умноженному на 2, и текущего разряда анализируемого числа. Действия по переходам для данного автомата приведены в табл. 5.5.

Таблица 5.5

Правила формирования выхода DNEXT автомата проверки делимости на 3

Номер перехода	Правило формирования DNEXT
1	DNEXT<=’0’
2	DNEXT<=’0’
3	DNEXT<=’1’
4	DNEXT<=’0’
5	DNEXT<=’1’
6	DNEXT<=’1’
RESET	DNEXT<=’0’

Графическое представление логики работы спроектированного автомата в нотации Xilinx Foundation FSM Editor’a приведено на рис. 5.8. Отметим, что в данном графе присутствуют все переходы, представленные в табл. 5.3, так как по ним осуществляется изменение сигнала на выходе *DNEXT*.

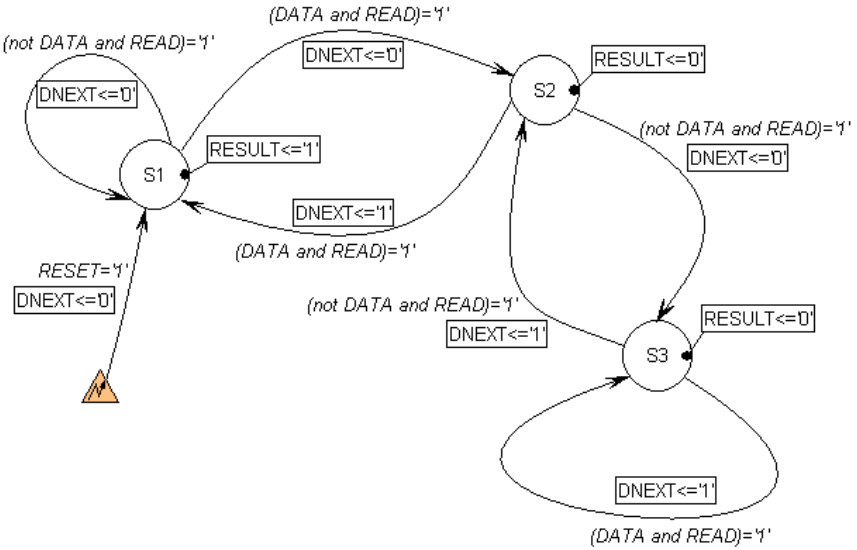


Рис. 5.8. Граф переходов автомата проверки делимости на 3 с формированием результата деления

Макроэлемент спроектированного автомата приведен на рис. 5.9. Временная диаграмма работы автомата проверки делимости на 3 с формированием результата деления показана на рис. 5.10. Тестовые воздействия – числа 9, 13 и 3.

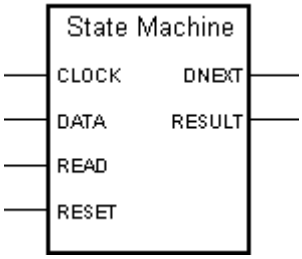


Рис. 5.9. Макроэлемент автомата для проверки делимости на 3 с формированием результата деления

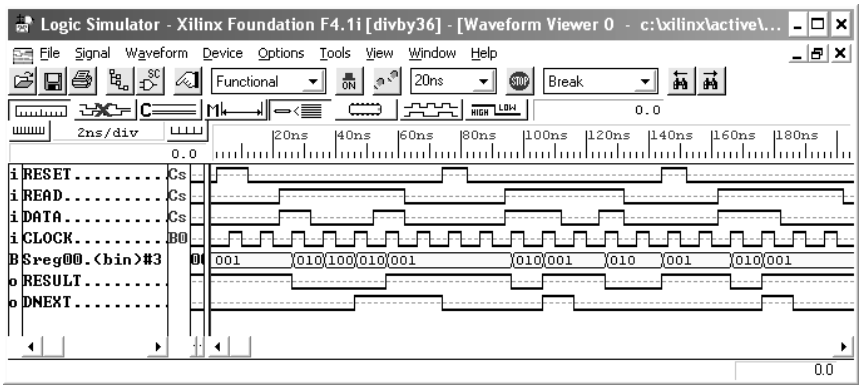


Рис. 5.10. Временная диаграмма работы автомата проверки делимости на 3 с формированием результата деления

Теперь синтезируем автомат для проверки делимости на 4, при этом воспользуемся свойством, что признаком делимости числа в двоичной системе счисления на 4 является наличие двух нулей в младших разрядах (фактически мы синтезируем автомат (2×2)). Синтез данного автомата не вызывает особых сложностей; состояния автомата и переходы между ними приведены в табл. 5.6. Начальным состоянием автомата является *S1*, в которое автомат переходит по асинхронному входу *RESET*.

Принимающим состоянием является только *S1*, тогда значение на выходе *RESULT* можно задать согласно табл. 5.7. Графическое представление логики работы автомата приведено на рис. 5.11.

Таблица 5.6

Матрица переходов для автомата проверки делимости на 4

Из \ В	S1 (00)	S2(X1)	S3(10)
S1(00)	1: not DATA and READ	2: DATA and READ	–
S2(X1)	–	3: DATA and READ	4: not DATA and READ
S3(10)	5: not DATA and READ	6: DATA and READ	–

Таблица 5.7

Значение выхода RESULT в зависимости от состояния автомата проверки делимости на 4

Состояние	S1 (00)	S2(X1)	S3(10)
RESULT	RESULT<=’1’	RESULT<=’0’	RESULT<=’0’

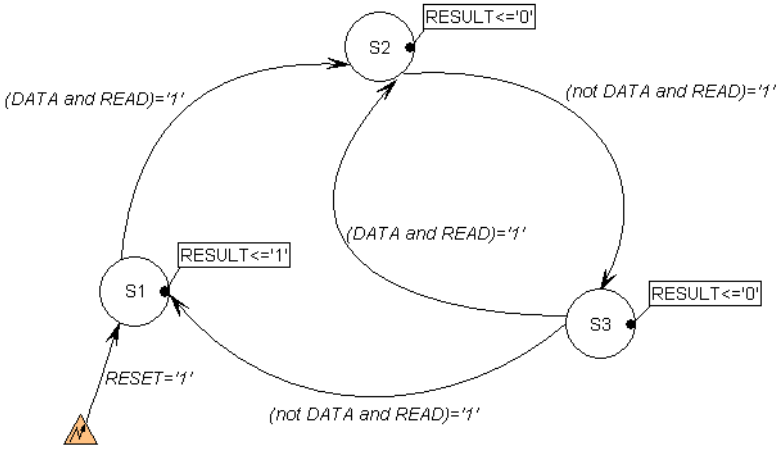


Рис. 5.11. Граф переходов автомата проверки делимости на 4

Макроэлемент спроектированного автомата приведен на рис. 5.10, а его временная диаграмма работы дана на рис. 5.13.

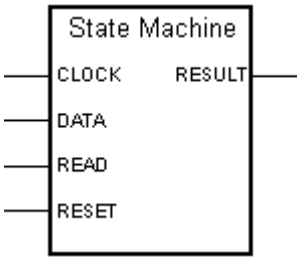


Рис. 5.12. Макроэлемент автомата для проверки делимости на 4

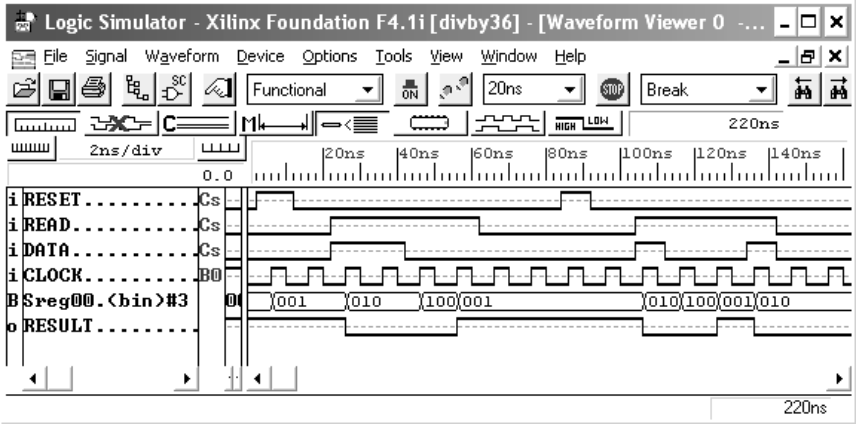


Рис. 5.13. Временная диаграмма работы автомата проверки делимости на 4

Спроектировав и синтезировав все базовые элементарные устройства, можно перейти непосредственно к синтезу логической схемы устройства проверки делимости.

Синтез логической схемы устройства проверки делимости на 36

Синтезируем схему устройства проверки делимости на 36 с использованием разработанных ранее элементарных устройств.

Согласно выбранному варианту реализации проекта, устройство состоит из двух параллельно работающих анализирующих блоков –

одноступенчатого устройства проверки делимости на 4 (конечный автомат DIV_BY_4) и двухступенчатого устройства проверки делимости на 9 (3×3) (последовательно соединенные конечные автоматы DIV_BY3_DN и DIV_BY_3 и триггер задержки передачи сигнала READ на вторую ступень). Устройство в целом соответствует схеме, приведенной на рис. 5.2, с дополнительной задержкой в передаче результата. Блок проверки делимости на 9 соответствует схеме приведенной на рис. 5.3. Логическая схема разработанного устройства делимости на 36 приведена на рис. 5.14. На триггере T12 запоминается результат делимости на 12, который задерживается на один такт для синхронизации с блоком проверки делимости на 9.

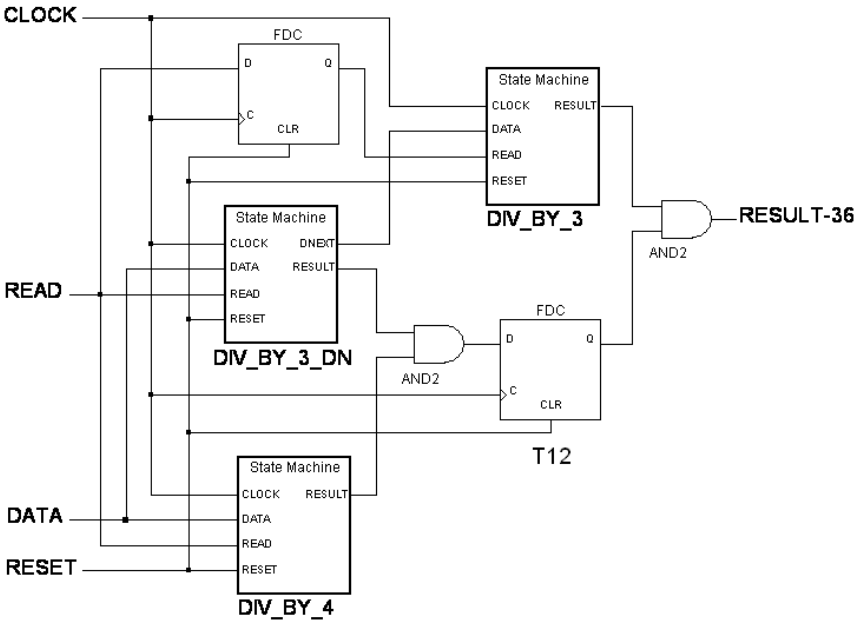


Рис. 5.14. Логическая схема устройства проверки делимости на 36

Временная диаграмма работы автомата, при проверке делимости числа 2340_{10} (100100100100_2) на 36_{10}, приведена на рис. 5.15.

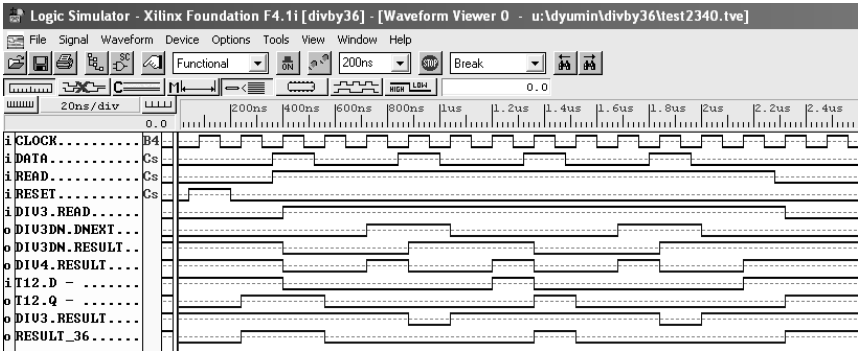


Рис. 5.15. Временная диаграмма работы устройства проверки делимости на 36

На первом такте работы схема устанавливается в начальное состояние сигналом RESET, в результате на втором такте формируется сигнал результата (остаток от деления 0 на 36_{10} равен 0). Анализируемое 12-ти разрядное двоичное число на вход DATA подается с третьего по четырнадцатый такт (при сигнале READ=1). Т.к. разработанная схема формирует сигнал результата с задержкой на один такт, то признак делимости формируется на седьмом такте (остаток от деления 36_{10} на 36_{10} равен 0), а на восьмом такте он снова устанавливается в ноль (73_{10} на 36_{10} не делится без остатка). На пятнадцатом такте формируется результирующий признак делимости *RESULT-36* = 1 (остаток от деления 2340_{10} на 36_{10} равен 0) и сохраняется, т.к. на входе схемы сигнал READ=0.

Синтез устройства проверки делимости двоичного сигнала на 36_{10} завершен. Согласно выбранному варианту реализации, получено устройство, формирующее признак делимости *RESULT-36* с задержкой в один такт относительно подаваемого на вход сигнала.

Подготовка к выполнению работы

1. Изучить описание лабораторной работы.
2. Получить индивидуальное задание у преподавателя.
3. Предложить варианты реализации устройства (разбиения на элементарные компоненты).

4. Выбрать оптимальный вариант с использованием формальных методов оценки сложности синтеза и качества получаемого устройства.

5. Составить матрицы переходов и таблицы генерации выходных сигналов для элементарных компонентов.

6. Составить описание элементарных компонентов устройства в виде графа.

7. Составить логическую схему устройства проверки делимости.

8. Составить тесты для проверки корректности функционирования спроектированного устройства.

Порядок выполнения работы

1. Выполнить ввод спроектированной схемы автомата в редакторе схем системы Xilinx Foundation.

2. Выполнить функциональное моделирование автомата.

3. Продемонстрировать преподавателю работу отлаженного автомата.

4. Сдать преподавателю оформленный отчет в конце занятия.

Отчет по работе

Отчет должен содержать:

1) исходные данные варианта задания;
2) все этапы проектирования устройства проверки делимости;
3) логическую схему устройства проверки делимости;
4) временные диаграммы, иллюстрирующие правильность как работы устройства проверки делимости в целом, так и его компонентов.

СПИСОК ЛИТЕРАТУРЫ

1. Хопкрофт Джон Э., Мотвани Раджив, Ульман Джеффри, Д.. Введение в теорию автоматов, языков и вычислений, 2-е изд.: /Пер. с англ. – М.: Издательский дом “Вильямс”, 2002. – 528 с.

Лабораторная работа 6

СИНТЕЗ АВТОМАТА РАСПОЗНАВАНИЯ СООТВЕТСТВИЯ БИНАРНОГО СИГНАЛА ЗАДАННОМУ ШАБЛОНУ

Цель: освоить принципы синтеза конечных автоматов для решения различных прикладных задач на примере проверки соответствия двоичных сигналов заданному шаблону; изучить применение регулярных выражений для задания шаблонов; изучить алгоритм формального синтеза недетерминированных конечных автоматов из регулярных выражений; изучить алгоритмы преобразования недетерминированных конечных автоматов в детерминированные; закрепить приемы синтеза простейших цифровых устройств; закрепить практические навыки отработки и верификации проектируемых цифровых устройств, посредством моделирования с использованием профессиональной САПР.

Введение

Как было показано в предыдущих лабораторных работах, *детерминированные конечные автоматы (ДКА)* могут применяться для проверки соответствия цифровых сигналов определенным критериям. В данной лабораторной работе будут рассматриваться методы синтеза ДКА для проверки соответствия цифровых сигналов определенному шаблону. Например, пусть правильно сформированный сигнал начинается и заканчивается синхропосылкой вида 101, а между ними должно находиться не менее одной произвольной триады (000, 001, 010, …). Данная задача может быть сравнительно просто решена с помощью конечного автомата.

Теоретические основы, основные этапы и подходы к проектированию

Для удобства описания шаблонов сигналов введем понятия *регулярного выражения*. Отметим, что далее приводится не строгое обобщенное определение регулярных выражений, а лишь проекция данного определения на предметный домен, связанный с

распознаванием соответствия цифровых сигналов определенному шаблону.

Регулярные выражения строятся из меньших регулярных выражений по определенным правилам, задаваемым при помощи распознаваемых символов (в случае бинарных сигналов ‘0’ и ‘1’) и метасимволов, определяющих некоторые базовые операции. Пусть *r* – это регулярное выражение соответствующее некоторому семейству сигналов *S(r)*, которое рекурсивно определяется на основе сигналов, описываемых подвыражениями *r*:

1) ε – является регулярным выражением, задающим пустой сигнал (отсутствие сигнала);
2) 0 – является регулярным выражением, задающим сигнал из одного ‘0’;
3) 1 – является регулярным выражением, задающим сигнал из одной ‘1’;
4) Пусть *r1* и *r2* – регулярные выражения, определяющие некоторые семейства сигналов *S(r1)* и *S(r2)*; тогда:
    a) *r1* | *r2* – регулярное выражение, описывающее объединение семейств сигналов *S(r1)* и *S(r2)*;
    b) *r1r2* – регулярное выражение, описывающее конкатенацию сигналов, принадлежащих семействам *S(r1)* и *S(r2)*;
    c) *r1** – регулярное выражение, описывающее конкатенацию нуля либо большего количества сигналов, принадлежащих семейству *S(r1)*;
    d) (*r1*) – регулярное выражение, описывающее семейство сигналов *S(r1)*.

Для удобства введем дополнительные операции (и метасимволы) для составления регулярных выражений; отметим, что данные операции могут быть выражены через базовые операции.

Пусть *r* – это регулярное выражение, соответствующее некоторому семейству сигналов *S(r)*, тогда:

r+ – регулярное выражение, описывающее конкатенацию одного либо большего количества сигналов, принадлежащих семейству *S(r)*;

r? – регулярное выражение, описывающее пустой сигнал либо сигнал, принадлежащий семейству *S(r)*.

Все перечисленные выше операции левоассоциативны и имеют разные приоритеты: операции *, +, ? имеют наивысший приоритет, операция конкатенации имеет средний приоритет и операция | имеет наименьший приоритет.

Тогда сигнал, описанный в начале данного раздела, можно представить в виде следующего регулярного выражения:

$$101((0|1)(0|1)(0|1))+101.$$

Регулярные выражения, помимо того, что являются удобной для человека формой записи шаблонов для распознавания, обладают следующим полезным свойством – они могут быть преобразованы в детерминированные конечные автоматы при помощи сравнительно простых формальных алгоритмов.

Рассмотрим один из таких алгоритмов – алгоритм преобразования регулярного выражения в детерминированный конечный автомат через недетерминированный конечный автомат. Данный алгоритм состоит из трех основных этапов: построение синтаксического дерева для разбора регулярного выражения (алгоритм многопроходного сканирования), синтез недетерминированного конечного автомата (расширенный алгоритм МакНортона–Ямады–Томпсона), преобразование недетерминированного конечного автомата в детерминированный (алгоритм построения подмножеств). Опциональным четвертым этапом проектирования может быть применение алгоритма минимизации количества состояний конечного автомата.

Алгоритм преобразования регулярного выражения в детерминированный конечный автомат

Синтаксическое дерево – структура данных, представляющая порядок вычисления (разбора) некоторых выражений в виде древовидной структуры. В узлах дерева находятся операции, в листьях находятся операнды для данных операций.

Если рассматривать синтаксические деревья для расширенных регулярных выражений для бинарных сигналов (введенных выше),

то в синтаксическом дереве могут быть следующие типы узлов и листьев:

1-*node* – лист, соответствующий значению сигнала ‘1’;
0-*node* – лист, соответствующий значению сигнала ‘0’;
*-*node* – узел, соответствующий операции ‘*’ (имеет одного потомка);
+-*node* – узел, соответствующий операции ‘+’ (имеет одного потомка);
?-*node* – узел, соответствующий операции ‘?’ (имеет одного потомка);
·-*node* – узел, соответствующий операции конкатенации (имеет двух потомков);
|-*node* – узел, соответствующий операции ‘|’ (имеет двух потомков).

Отметим, что для метасимволов ‘( )’ нет соответствующих типов узлов, так как они только определяют порядок выполнения операций, что определяется структурой дерева.

Следует отметить, что часто, прежде чем проводить синтез синтаксического дерева, целесообразно упростить регулярное выражение, приведя его к более компактной форме, используя следующие базовые эквивалентности:

1) $r \mid s = s \mid r$
2) $r \mid (s \mid t) = (r \mid s) \mid t$
3) $r(st) = (rs)t$
4) $r\,(s \mid t) = rs \mid rt$
5) $(s \mid t)r = sr \mid tr$
6) $\varepsilon r = r\varepsilon = r$
7) $r^* = (r \mid \varepsilon)^*$
8) $r^{**} = r^*$
9) $r^* = r+ \mid \varepsilon$
10) $r+ = rr^* = r^*r$
11) $r? = r \mid \varepsilon$

Существует несколько алгоритмов для построения синтаксических деревьев для разбора регулярных выражений, в

том числе с использованием детерминированных конечных автоматов с памятью, с использованием контекстно-независимых грамматик и т.п., имеющие, в том числе, эффективную программную реализацию. Но наиболее простым, с точки зрения ручного синтеза для сравнительно простых регулярных выражений, является алгоритм многопроходного сканирования регулярного выражения с поиском высокоприоритетных примитивов. Алгоритм также может быть реализован и программно, но его эффективность несколько ниже, так как он требует многократного просмотра строки регулярного выражения, в отличие от алгоритмов, перечисленных ранее, которым требуется один проход по строке.

Алгоритм построения синтаксического дерева методом многопроходного сканирования регулярного выражения для бинарных сигналов приведен ниже.

Алгоритм 6.1

Построение синтаксического дерева для регулярного выражения

```
Пусть r – анализируемое регулярное выражение.
Заключим это выражение в скобки:
r:= (r)
start:=первый символ r (т.е. первая круглая скобка)
end:=последний символ r (последняя круглая скобка)
пока в r более одного элемента {
    найти ближайшую друг к другу пару скобок ‘(‘ и ‘)’ :
        first:=позиция ‘(‘, last:=позиция ‘)’
    Проход 1: создание узлов 1-node и 0-node:
        для каждого элемента в диапазоне от first до last {
            если элемент – символ ‘1’ {
                создать 1-node
                заменить в r ‘1’ на 1-node
            }
            иначе {
                создать 0-node
                заменить в r ‘0’ на 0-node
            }
        }
```

*Проход 2: создание узлов *-node, +-node и ?-node:*
для каждой пары соседних элементов в диапазоне от first до last {
если текущая пара элементов – node и '' {*
*создать узел *-node*
*установить node потомком для элемента *-node*
заменить в r пару node и '' на *-node*
*сделать *-node текущим элементом*
}
если текущая пара элементов – node и '+' {
создать узел +-node
установить node потомком для элемента +-node
заменить в r пару node и '+' на +-node
сделать +-node текущим элементом
}
если текущая пара элементов – node и '?' {
создать узел ?-node
установить node потомком для элемента ?-node
заменить в r пару node и '?' на ?-node
сделать ?-node текущим элементом
}
Проход 3: создание узлов ·-node:
для каждого элемента в диапазоне от first до last {
если текущий элемент – node и следующий – node {
создать узел ·-node
установить текущий node и следующий node потомками для элемента ·-node
заменить в r пару node и node на ·-node
сделать ·-node текущим элементом
}
}
Проход 4: создание узлов |-node:
для каждого элемента в диапазоне от first до last {
если текущая триада соответствует node, '|' и node {
создать узел |-node
установить элементы node и node потомками для элемента |-node
заменить в r триаду node, '|' и node на |-node

сделать |-node текущим элементом
}
}

Проход 5: между first (‘(‘) и last (‘)’) один элемент node:
заменить в r триаду ‘(‘, node и ‘)’ на node
}
root:=node

Регулярные выражения, представленные в виде синтаксических деревьев, можно преобразовать непосредственно в детерминированные конечные автоматы, но данные алгоритмы являются сравнительно сложными, по сравнению с использованием промежуточного представления регулярных выражений в виде недетерминированных автоматов с последующим их преобразованием в детерминированные.

Недетерминированный конечный автомат состоит из множества состояний S, одно из которых является начальным s_0, множества входных символов Σ, при этом $\varepsilon \notin \Sigma$, на которых определена функция перехода $f(s_i, a) = S`, a \in \Sigma \cup \{\varepsilon\}, s_i \in S, S' \subset S$ и определено множество принимающих (допускающих, или финальных) состояний $F \subset S$.

Отметим, что, исходя из определения, недетерминированный конечный автомат может находиться в нескольких состояниях одновременно. Для него возможны переходы из данного состояния по данному входному символу в несколько различных состояний, что усложняет непосредственное моделирование данного класса устройств. Но более мягкие ограничения на структуру данного автомата позволяют гораздо проще синтезировать подобные устройства (как вручную, так и программно) по сравнению с детерминированным конечными автоматами, решающими ту же задачу. В теории автоматов формально доказывается эквивалентность недетерминированных и детерминированных конечных автоматов.

Рассмотрим модифицированный алгоритм МакНортона–Ямады–Томпсона для преобразования регулярного выражения, описывающего бинарные сигналы, в недетерминированный

конечный автомат. Суть алгоритма заключается в том, что при обходе синтаксического дерева для каждого из поддеревьев строится структурное представление недетерминированного конечного автомата по следующим правилам.

Пусть r_i и r_j – некоторые подвыражения регулярного выражения r, а $N(r_i)$ и $N(r_j)$ – соответствующие им недетерминированные конечные автоматы, которые получаются при обходе синтаксического дерева, соответствующего r. Тогда:

1. Если текущий узел ε-node, то ему соответствует недетерминированный автомат, приведенный на рис. 6.1, где i – стартовое состояние, а f – принимающее состояние.

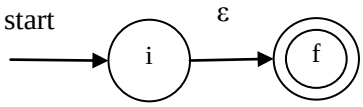


Рис. 6.1. Структура недетерминированного конечного автомата, соответствующего регулярному выражению $r = \varepsilon$

2. Если текущий узел *a-node* (для регулярных выражений, описывающих бинарные сигналы, 0-*node* или 1-*node*), то ему соответствует недетерминированный автомат, приведенный на рис. 6.2, где i – стартовое состояние, а f – принимающее состояние.

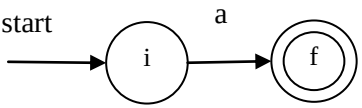


Рис. 6.2. Структура недетерминированного конечного автомата, соответствующего регулярному выражению $r = a$

3. Если текущий узел |-*node*, и его дочерним узлам соответствуют недетерминированные конечные автоматы *N(v)* со стартовым состоянием s_v и принимающим f_v и *N(w)* со стартовым состоянием s_w и принимающим f_w, тогда данному узлу соответствует недетерминированный автомат, приведенный на рис. 6.3, где *i* – стартовое состояние, а *f* – принимающее состояние.

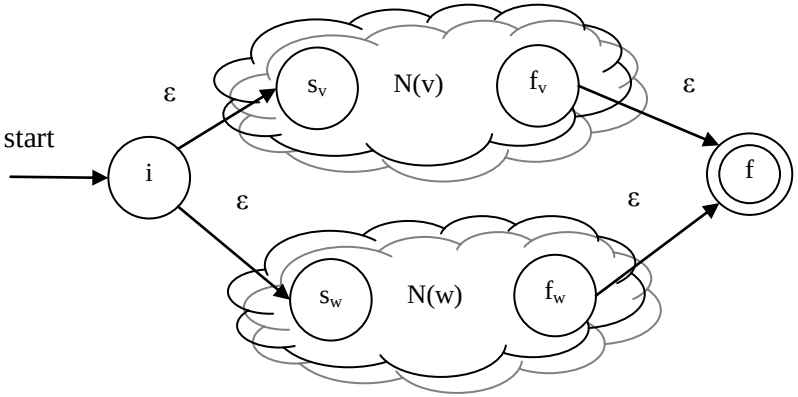


Рис. 6.3. Структура недетерминированного конечного автомата, соответствующего регулярному выражению $r = v|w$

4. Если текущий узел ·-*node*, и его дочерним узлам соответствуют недетерминированные конечные автоматы *N(v)* со стартовым состоянием s_v и принимающим f_v и *N(w)* со стартовым состоянием s_w и принимающим f_w, тогда данному узлу соответствует недетерминированный автомат, приведенный на рис. 6.4, где *i* – стартовое состояние, *f* – принимающее состояние, а состояния f_v и s_w сливаются в одно состояние, т.е. принимающее состояние первого операнда конкатенации становится стартовым состоянием второго операнда.

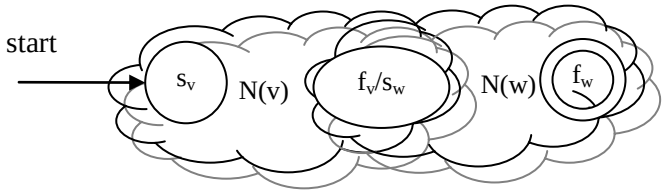


Рис. 6.4. Структура недетерминированного конечного автомата, соответствующего регулярному выражению $r = vw$

5. Если текущий узел **-node*, и его дочернему узлу соответствует недетерминированный конечный автомат *N(v)* со стартовым состоянием s_v и принимающим f_v, тогда данному узлу соответствует недетерминированный автомат, приведенный на рис. 6.5, где *i* – стартовое состояние, *f* – принимающее состояние.

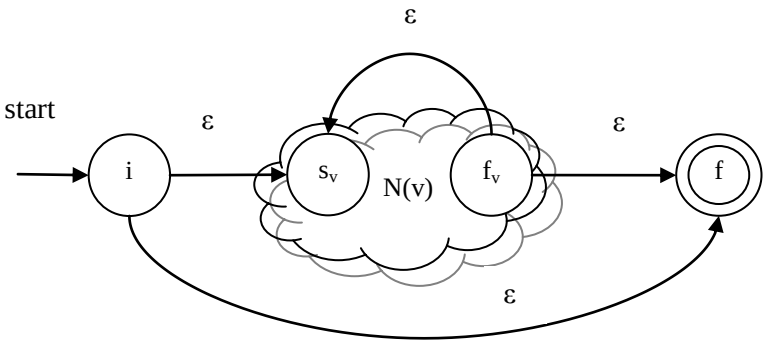


Рис. 6.5. Структура недетерминированного конечного автомата, соответствующего регулярному выражению $r = v*$

6. Если текущий узел *+-node*, и его дочернему узлу соответствует недетерминированный конечный автомат *N(v)* со стартовым состоянием s_v и принимающим f_v, тогда данному узлу соответствует недетерминированный автомат, приведенный на рис. 6.6, где *i* – стартовое состояние, *f* – принимающее состояние.

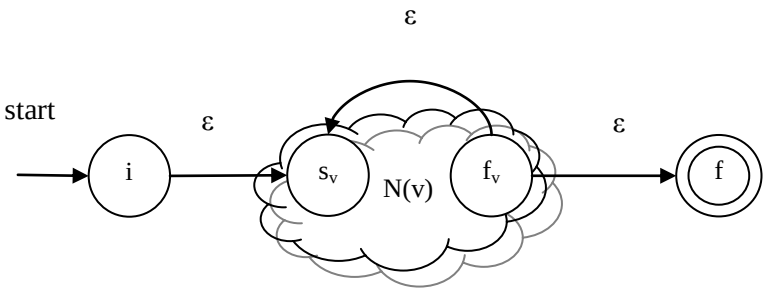


Рис. 6.6. Структура недетерминированного конечного автомата, соответствующего регулярному выражению $r = v*$

7. Если текущий узел ?-*node*, и его дочернему узлу соответствует недетерминированный конечный автомат *N(v)* со стартовым состоянием s_v и принимающим f_v, тогда данному узлу соответствует недетерминированный автомат, приведенный на рис. 6.7, где *i* – стартовое состояние, *f* – принимающее состояние.

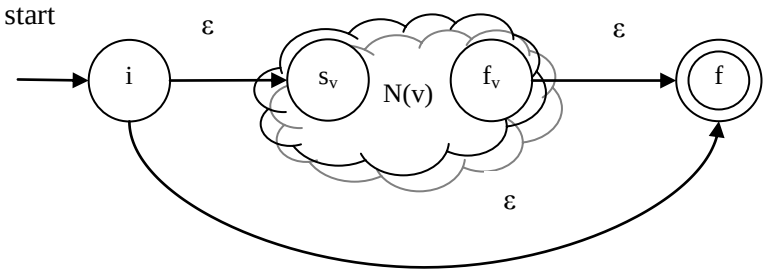


Рис. 6.7. Структура недетерминированного конечного автомата, соответствующего регулярному выражению $r = v*$.

Синтезированный с помощью данного алгоритма недетерминированный конечный автомат обладает следующими полезными свойствами, с точки зрения его анализа и дальнейшего использования:

1) количество состояний не более чем в два раза превышает количество символов в записи регулярного выражения;
2) автомат имеет одно начальное и одно принимающее состояние;
3) принимающее состояние не имеет исходящих переходов;
4) начальное состояние не имеет входящих переходов;
5) для любого не принимающего состояния имеется один переход по сигналу 0 или 1, или не более двух исходящих переходов по ε.

Недетерминированный конечный автомат может быть преобразован в детерминированный, при этом следует отметить, что количество состояний детерминированного конечного автомата в общем случае может экспоненциально зависеть от количества состояний недетерминированного автомата. Для начала, введем ряд понятий, которые будем использовать при описании алгоритма:

ε-замыкание $z_\varepsilon(S')$ от подмножества состояний недетерминированного конечного автомата $S' \subset S$ – это множество состояний, достижимых из данного подмножества состояний *S'* по произвольному количеству ε-переходов;

следующее a-подмножество $M_a(S')$ от подмножества состояний недетерминированного конечного автомата $S' \subset S$ – подмножество состояний, достижимых из подмножества состояний *S'* по одному *a*-переходу (для недетерминированных конечных автоматов для анализа регулярных выражений у нас будут только *a*-подмножества $M_0(S')$ и $M_1(S')$).

Идея данного алгоритма сводится к последовательной замене ε-замыканий от следующих *a*-подмножеств, построенных, начиная со стартового состояния, одним состоянием детерминированного конечного автомата.

Рассмотрим вспомогательный алгоритм построения ε-замыкания для подмножества состояний *S'* (алгоритм 6.2).

Алгоритм 6.2

Построение ε-замыкания

поместить все $s \in S'$ *в стек*
поместить все $s \in S'$ *в* $z_\varepsilon(S')$
пока стек не пуст {
  взять очередное s'
  для каждого ε-перехода из s' в s"{
    если $s'' \notin z_\varepsilon(S')${
      поместить s" в стек
      поместить s" в $z_\varepsilon(S')$
    }
  }
}

Рассмотрим один из алгоритмов преобразования недетерминированного конечного автомата в детерминированный – алгоритм построения подмножеств (алгоритм 6.3). В результате работы алгоритма получаем таблицу переходов *D* между состояниями детерминированного конечного автомата *S'* и их соответствие состояниям недетерминированного конечного автомата *S*, из которого получен данный детерминированный конечный автомат.

Алгоритм 6.3

Построение подмножеств для автоматов анализа бинарных сигналов

построить $z_\varepsilon(s_0)$ *от начального состояния*
назначить соответствие s'_0 *множеству* $z_\varepsilon(s_0)$
поместить s'_0 *в D, как непомеченные*
пока в D есть непомеченные состояния {
  взять непомеченное s'_i
  пометить s'_i
  $s'(0) := z_\varepsilon(M_0(s'_i))$
  если $s'(0) \notin D$
    поместить s'(0) D как непомеченное
  заполнить соответствующие переходы $f(s'_i,0)=s'(0)$

$s'(1) := z_{\varepsilon}(M_1(s'_i))$

если $s'(1) \notin D$

поместить s'(1) D как непомеченное

заполнить соответствующие переходы f(s'i,1)=s'(1)

}

Отметим, что если при построении таблицы переходов (см. алгоритм 6.3) $s'(0)$ или $s'(1)$ равняются $\emptyset$, то соответствующие переходы ведут в так называемое тупиковое состояние – состояние детерминированного конечного автомата, из которого нет переходов, кроме как в само себя. При попадании в данное состояние, которое не является принимающим, можно сразу сделать вывод о том, что сигнал не соответствует заданному шаблону.

Следует также отметить, что полученный детерминированный конечный автомат может содержать избыточное число состояний, которое можно минимизировать, используя соответствующие алгоритмы.

Условное графическое обозначение синтезируемого автомата приведено на рис. 6.8.

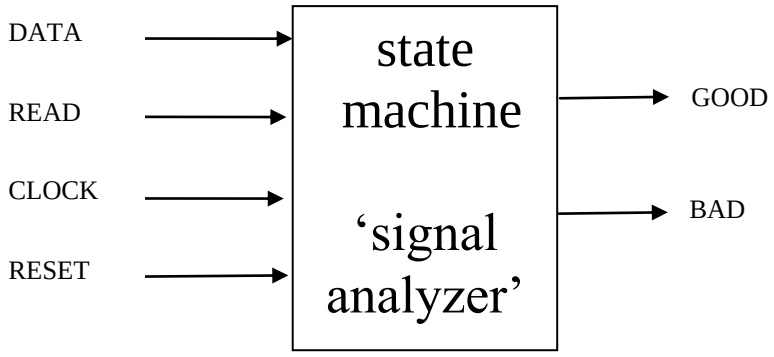


Рис. 6.8. Условное графическое обозначение автомата проверки соответствия сигнала некоторому шаблону

На вход *DATA* поступает анализируемый двоичный сигнал. На вход *READ* поступает признак наличия корректных данных на вхо-

де *DATA* (0 – данные на входе *DATA* игнорируются, 1 – данные на входе *DATA* анализируются конечным автоматом). На вход *CLOCK* поступают синхроимпульсы. На вход *RESET* подается сигнал сброса конечного автомата в начальное состояние. На выходе *GOOD* формируется признак соответствия сигнала заданному шаблону, на выходе *BAD* формируется сигнал о заведомо некорректном сигнале (при переходе в тупиковое состояние, которое не является принимающим).

Пример

Постановка задачи. Спроектировать устройство, определяющее, является ли сигнал правильно сформированным. Правильно сформированный сигнал начинается и заканчивается синхропосылкой вида 101, а между ними должно находиться не менее одной произвольной триады (000, 001, 010,…).

Составление регулярного выражения и построение синтаксического дерева

Правильный сигнал должен начинаться и заканчиваться последовательностями 101; между ними может находиться не менее одной произвольной триады (000, 001, 010, …, 111). Тогда формальное описание сигнала в виде регулярного выражения будет иметь следующий вид:

101(000|001|010|011|100|101|110|111)+101

Данное регулярное выражение можно сократить, используя стандартные преобразования:

101(000|001|010|011|100|101|110|111)+101 =
101(00(0|1)|01(0|1)|10(0|1)|11(0|1))+101 =
101((00|01|10|11)(0|1))+101=
101((0(0|1)|1(0|1))(0|1))+101=
101((0|1)(0|1)(0|1))+101

Отметим, что результат преобразований может быть получен и непосредственно из анализа задания.

Построим синтаксическое дерево для разбора сокращенной формы записи регулярного выражения, соответствующего шаблону сигнала, по алгоритму 6.1.

1. Получим исходное регулярное выражение (РВ), дополненное скобками в начале и конце:

РВ	(	1	0	1	(	(	0	|	1	)	(	0	|	1	)	(	0	|	1	)	)
№ поз.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

РВ	+	1	0	1	)
№ поз.	21	22	23	24	25

2. Находим первую пару ближайших скобок в позициях 5 и 9; заменяем позицию 6 на 0-node (*A*), позицию 8 на 1-node (*B*):

РВ	(	1	0	1	(	(	A	|	B	)	(	0	|	1	)	(	0	|	1	)	)
№ поз.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

РВ	+	1	0	1	)
№ поз.	21	22	23	24	25

3. Заменяем триаду *A* | *B* (позиции 6, 7, 8) на |-*node* (*C*); *A* и *B* – левое и правое поддеревья для *C* (все изменения выполняются в строке исходного регулярного выражения, поэтому номера позиций элементов изменяются):

РВ	(	1	0	1	(	(	C	)	(	0	|	1	)	(	0	|	1	)	)	+	1
№ поз.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

РВ	0	1	)
№ поз.	21	22	23

4. Заменяем триаду (*C*) (позиции 5, 6, 7) на *C*:

РВ	(	1	0	1	(	C	(	0	|	1	)	(	0	|	1	)	)	+	1	0	1
№ поз.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

РВ	)
№ поз.	21

5. Вновь находим первую пару ближайших скобок – теперь в позициях 6 и 10; заменяем позицию 7 на 0-*node* (*D*), позицию 9 на 1-*node* (*E*):

РВ	.	1	0	1	(	C	(	D	|	E	)	(	0	|	1	)	)	+	1	0	1
№ поз.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

РВ	)
№ поз.	21

6. Заменяем триаду *D* | *E* на |-node (*F*); *D* и *E* – левое и правое поддеревья для *F*:

РВ	(	1	0	1	(	C	(	F	)	(	0	|	1	)	)	+	1	0	1	)
№ поз.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

7. Заменяем триаду (*F*) на *F*:

РВ	(	1	0	1	(	C	F	(	0	|	1	)	)	+	1	0	1	)
№ поз.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

8. Находим очередную пару ближайших скобок в позициях 7 и 11; заменяем позицию 8 на 0-*node* (*G*), позицию 10 на 1-*node* (*H*):

РВ	(	1	0	1	(	C	F	(	G	|	H	)	)	+	1	0	1	)
№ поз.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

9. Заменяем триаду *G* | *H* на |-*node* (*J*); *G* и *H* – левое и правое поддеревья для *J*:

РВ	(	1	0	1	(	C	F	(	J	)	)	+	1	0	1	)
№ поз.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

10. Заменяем триаду (*J*) на *J*:

РВ	(	1	0	1	(	C	F	J	)	+	1	0	1	)
№ поз.	0	1	2	3	4	5	6	7	8	9	10	11	12	13

11. Находим первую пару ближайших скобок на позициях 4 и 8; заменяем пару CF на ·-*node* (*K*); *C* и *F* – левое и правое поддеревья для *K*:

РВ	(	1	0	1	(	K	J	)	+	1	0	1	)
№ поз.	0	1	2	3	4	5	6	7	8	9	10	11	12

12. Заменяем пару *KJ* на ·-*node* (*L*); *K* и *J* – левое и правое поддеревья для *L*:

РВ	(	1	0	1	(	L	)	+	1	0	1	)
№ поз.	0	1	2	3	4	5	6	7	8	9	10	11

13. Заменяем триаду (*L*) на *L*:

РВ	(	1	0	1	L	+	1	0	1	)
№ поз.	0	1	2	3	4	5	6	7	8	9

14. Находим очередную пару ближайших скобок в позициях 0 и 9; заменяем позиции 2 и 7 на 0-*node* (*M*, *N*), позиции 1, 3, 6 и 8 на 1-*node* (*P*, *Q*, *R*, *S*)

РВ	(	P	M	Q	L	+	R	N	S	)
№ поз.	0	1	2	3	4	5	6	7	8	9

15. Заменяем пару *L*+ на +-*node* (*T*); *L* – поддерево для *T*:

РВ	(	P	M	Q	T	R	N	S	)
№ поз.	0	1	2	3	4	5	6	7	8

16. Заменяем пару *PM* на ·-*node* (*U*); *P* и *M* – левое и правое поддеревья для *U*:

РВ	(	U	Q	T	R	N	S	)
№ поз.	0	1	2	3	4	5	6	7

17. Заменяем пару *UQ* на ·-*node* (*V*); *U* и *Q* – левое и правое поддеревья для *V*:

РВ	(	V	T	R	N	S	)
№ поз.	0	1	2	3	4	5	6

18. Заменяем пару *VT* на ·-*node* (*W*); *V* и *T* – левое и правое поддеревья для *W*:

РВ	(	W	R	N	S	)
№ поз.	0	1	2	3	4	5

19. Заменяем пару *WR* на ·-*node* (*X*); *W* и *R* – левое и правое поддеревья для *X*:

РВ	(	X	N	S	)
№ поз.	0	1	2	3	4

20. Заменяем пару *XN* на ·-*node* (*Y*); *X* и *N* – левое и правое поддеревья для *Y*:

РВ	(	Y	S	)
№ поз	0	1	2	3

21. Заменяем пару *YS* на ·-*node* (*Z*); *Y* и *S* – левое и правое поддеревья для *Z*:

РВ	(	Z	)
№ поз	0	1	2

22. Заменяем триаду (*Z*) на *Z*:

РВ	Z
№ поз.	0

23. Корнем дерева является *Z*.

Построенное дерево приведено на рис. 6.9.

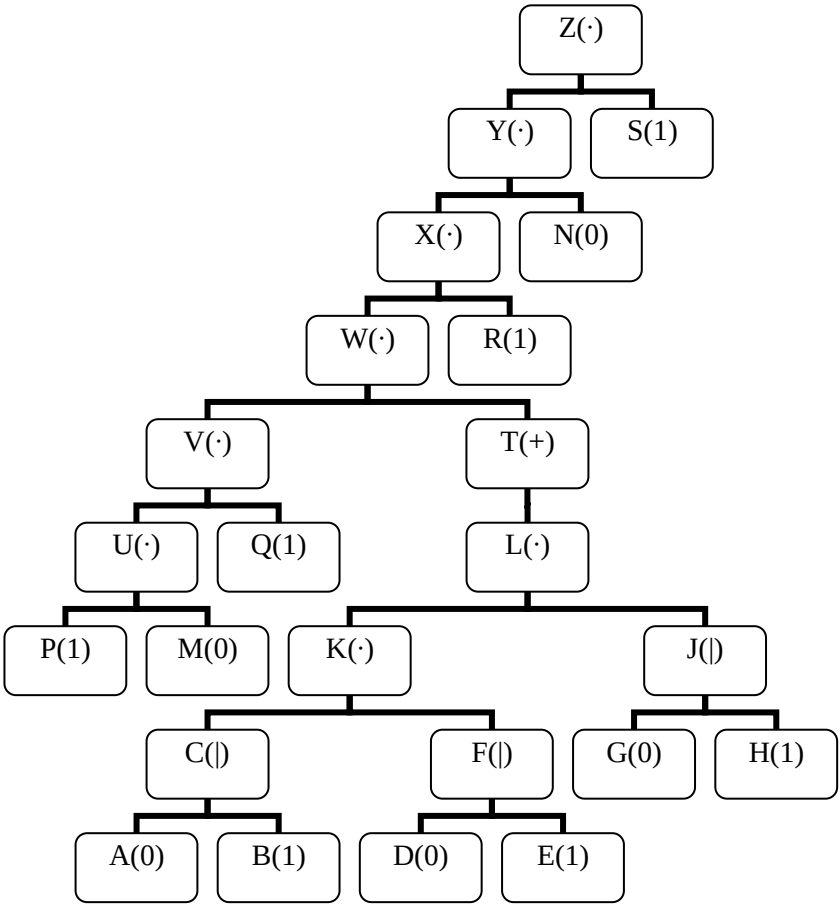


Рис. 6.9. Синтаксическое дерево для регулярного выражения 101((0|1)(0|1)(0|1))+101

Синтез недетерминированного конечного автомата

Для синтеза недетерминированного автомата воспользуемся рекурсивным обходом дерева в порядке левое поддерево – правое поддерево – корень (post-order), и синтезируем автоматы для посещаемых узлов в соответствии с модифицированным алгоритмом МакНортона–Ямады–Томпосона.

1. От корня спускаемся до узла *P*, синтезируем автомат *N1* по правилу 2.
2. Возвращаемся в узел *U*, посещаем узел *M*, синтезируем автомат *N2* по правилу 2.
3. Возвращаемся в узел *U*, синтезируем автомат *N3* по правилу 4, объединяя автоматы *N1* и *N2*.
4. Возвращаемся в узел *V*, посещаем узел *Q*, синтезируем автомат *N4* по правилу 2.
5. Возвращаемся в узел *V*, синтезируем автомат *N5* по правилу 4, объединяя автоматы *N3* и *N4*.
6. Возвращаемся в узел *W*, спускаемся до узла *A*, синтезируем автомат *N6* по правилу 2.
7. Возвращаемся в узел *C*, посещаем узел *B*, синтезируем автомат *N7* по правилу 2.
8. Возвращаемся в узел *C*, синтезируем автомат *N8* по правилу 3, объединяя автоматы *N6* и *N7*.
9. Возвращаемся в узел *K*, спускаемся в узел *D*, синтезируем автомат *N9* по правилу 2.
10. Возвращаемся в узел *F*, посещаем узел *E*, синтезируем автомат *N10* по правилу 2.
11. Возвращаемся в узел *F*, синтезируем автомат *N11* по правилу 3, объединяя автоматы *N9* и *N10*.
12. Возвращаемся в узел *K*, синтезируем автомат *N12* по правилу 4, объединяя автоматы *N7* и *N11*.
13. Возвращаемся в узел *L*, спускаемся в узел *G*, синтезируем автомат *N13* по правилу 2.
14. Возвращаемся в узел *J*, посещаем узел *H*, синтезируем автомат *N14* по правилу 2.
15. Возвращаемся в узел *J*, синтезируем автомат *N15* по правилу 3, объединяя автоматы *N13* и *N14*.

16. Возвращаемся в узел *L*, синтезируем автомат *N16* по правилу 4, объединяя автоматы *N12* и *N15*.

17. Возвращаемся в узел *T*, синтезируем автомат *N17* по правилу 6, преобразуя автомат *N17*.

18. Возвращаемся в узел *W*, синтезируем автомат *N18* по правилу 4, объединяя автоматы *N5* и *N17*.

19. Возвращаемся в узел *X*, посещаем узел *R*, синтезируем автомат *N19* по правилу 2.

20. Возвращаемся в узел *X*, синтезируем автомат *N20* по правилу 4, объединяя автоматы *N18* и *N19*.

21. Возвращаемся в узел *Y*, посещаем узел *N*, синтезируем автомат *N21* по правилу 2.

22. Возвращаемся в узел *Y*, синтезируем автомат *N22* по правилу 4, объединяя автоматы *N20* и *N21*.

23. Возвращаемся в узел *Z*, посещаем узел *S*, синтезируем автомат *N23* по правилу 2.

24. Возвращаемся в узел *Z*, синтезируем автомат *N24* по правилу 4, объединяя автоматы *N22* и *N23*.

25. Так как мы вернулись и обработали корень дерева, то синтез автомата завершен.

Граф переходов автомата приведен на рис. 6.10, таблица переходов приведена в табл. 6.1. Состояние *S0* является стартовым, принимающим является состояние *S23*.

Очевидно, что данный недетерминированный конечный автомат не является единственным, так как благодаря наличию ε-переходов можно построить бесконечное множество автоматов, соответствующих данному регулярному выражению. Данный автомат также не является и минимальным, так как количество состояний в автомате, представленном на рис. 6.10, может быть сокращено. Но в данном сокращении нет необходимости, так как данный автомат в дальнейшем будет преобразован в детерминированный конечный автомат. При этом алгоритм построения недетерминированного конечного автомата, которым мы воспользовались, гарантирует верхнюю границу анализируемых состояний автомата и количество переходов из его состояний, что делает дальнейшие этапы синтеза более предсказуемыми с точки зрения вычислительной сложности.

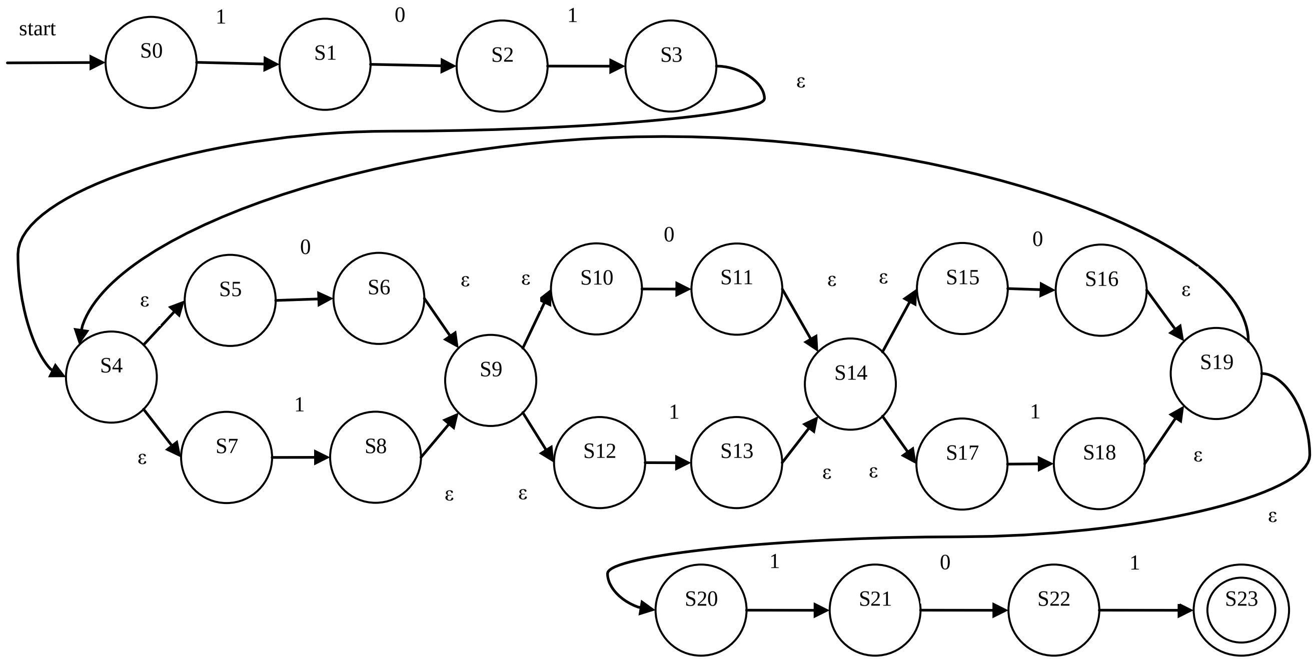


Рис. 6.10. Граф переходов недетерминированного конечного автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101

Таблица 6.1

Таблица переходов недетерминированного конечного автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101

Исходное состояние	Следующее состояние		
	0	1	ε
S0	-	S1	-
S1	S2	-	-
S2	-	S3	-
S3	-	-	S4
S4	-	-	S5, S7
S5	S6	-	-
S6	-	-	S9
S7	-	S8	-
S8	-	-	S9
S9	-	-	S10, S12
S10	S11	-	-
S11	-	-	S14
S12	-	S13	-
S13	-	-	S14
S14	-	-	S15,S17
S15	S16	-	-
S16	-	-	S19
S17	-	S18	-
S18	-	-	S19
S19	-	-	S20
S20	-	S21	-
S21	S22	-	-
S22	-	S23	-
S23	-	-	-

Синтез детерминированного конечного автомата

Рассмотрим синтез детерминированного конечного автомата по алгоритму 6.3.

1. Строим множество $z_\varepsilon(S0) = \{S0\}$, ставим ему в соответствие состояние ДКА *A*, которое будет стартовым (табл. 6.2).
2. Строим множество $z_\varepsilon(M_0(A)) = \varnothing$, т.е. из *A* по 0 будет переход

в тупиковое состояние. Строим множество $z_\varepsilon(M_1(A)) = \{S2\}$, ставим ему в соответствие состояние ДКА *B*. Заполняем соответствующим образом таблицу переходов. Помечаем проанализированное состояние (табл. 6.3).

Таблица 6.2

Синтез таблицы переходов ДКА (шаг 1)

ДКА	НКА	0	1
A	S0		

Таблица 6.3

Синтез таблицы переходов ДКА (шаг 2)

ДКА	НКА	0	1
A (*)	S0	Ø	B
B	S1		

3. Строим множество $z_\varepsilon(M_0(B)) = \{S2\}$, ставим ему в соответствие состояние ДКА *C*. Строим множество $z_\varepsilon(M_1(B)) = \varnothing$, т.е. из *B* по 0 будет переход в тупиковое состояние. Заполняем соответствующим образом таблицу переходов. Помечаем проанализированное состояние (табл. 6.4).

Таблица 6.4

Синтез таблицы переходов ДКА (шаг 3)

ДКА	НКА	0	1
A (*)	S0	Ø	B
B (*)	S1	C	Ø
C	S2		

4. Строим множество $z_\varepsilon(M_0(C)) = \varnothing$, т.е. из *C* по 0 будет переход в тупиковое состояние. Строим множество $z_\varepsilon(M_1(C)) = \{S3, S4, S5, S7\}$, ставим ему в соответствие состояние ДКА *D*. Заполняем соответствующим образом таблицу переходов. Помечаем проанализированное состояние (табл. 6.5).

5. Строим множество $z_\varepsilon(M_0(D)) = \{S6, S9, S10, S12\}$, ставим ему в соответствие состояние ДКА *E*. Строим множество $z_\varepsilon(M_1(D)) =$

= {*S7, S9, S10, S12*}, ставим ему в соответствие состояние ДКА *F*. Заполняем соответствующим образом таблицу переходов. Помечаем проанализированное состояние (табл. 6.6).

Таблица 6.5

Синтез таблицы переходов ДКА (шаг 4)

ДКА	НКА	0	1
A (*)	S0	Ø	B
B (*)	S1	C	Ø
C (*)	S2	Ø	D
D	S3, S4, S5, S7		

Таблица 6.6

Синтез таблицы переходов ДКА (шаг 5)

ДКА	НКА	0	1
A (*)	S0	Ø	B
B (*)	S1	C	Ø
C (*)	S2	Ø	D
D (*)	S3, S4, S5, S7	E	F
E	S6, S9, S10, S12		
F	S8, S9, S10, S12		

6. Строим множество $z_\varepsilon(M_0(E))$ = {*S11, S14, S15, S17*}, ставим ему в соответствие состояние ДКА *G*. Строим множество $z_\varepsilon(M_1(E))$ = {*S13, S14, S15, S17*}, ставим ему в соответствие состояние ДКА *H*. Заполняем соответствующим образом таблицу переходов. Помечаем проанализированное состояние (табл. 6.7).

7. Строим множество $z_\varepsilon(M_0(F))$ = {*S11, S14, S15, S17*}. Состояние, соответствующее такому множеству, уже есть в ДКА (*G*). Строим множество $z_\varepsilon(M_1(F))$ = {*S13, S14, S15, S17*}. Состояние, соответствующее такому множеству, также есть в ДКА (*H*). Заполняем соответствующим образом таблицу переходов. Помечаем проанализированное состояние (табл. 6.8).

8. Строим множество $z_\varepsilon(M_0(G))$ = {*S4, S5, S7, S16, S19, S20*}, ставим ему в соответствие состояние ДКА *I*. Строим множество $z_\varepsilon(M_1(G))$ = {*S4, S5, S7, S18, S19, S20*}, ставим ему в соответствие состояние ДКА *J*. Заполняем соответствующим образом таблицу переходов. Помечаем проанализированное состояние (табл. 6.9).

Таблица 6.7

Синтез таблицы переходов ДКА (шаг 6)

ДКА	НКА	0	1
A (*)	S0	Ø	B
B (*)	S1	C	Ø
C (*)	S2	Ø	D
D (*)	S3, S4, S5, S7	E	F
E (*)	S6, S9, S10, S12	G	H
F	S8, S9, S10, S12		
G	S11, S14, S15, S17		
H	S13, S14, S15, S17		

Таблица 6.8

Синтез таблицы переходов ДКА (шаг 7)

ДКА	НКА	0	1
A (*)	S0	Ø	B
B (*)	S1	C	Ø
C (*)	S2	Ø	D
D (*)	S3, S4, S5, S7	E	F
E (*)	S6, S9, S10, S12	G	H
F (*)	S8, S9, S10, S12	G	H
G	S11, S14, S15, S17		
H	S13, S14, S15, S17		

9. Строим множество $z_\varepsilon(M_0(H)) = \{S11, S14, S15, S17\}$. Состояние, соответствующее такому множеству, уже есть в ДКА (*I*). Строим множество $z_\varepsilon(M_1(H)) = \{S4, S5, S7, S18, S19, S20\}$. Состояние, соответствующее такому множеству также есть в ДКА (*J*). Заполняем соответствующим образом таблицу переходов. Помечаем проанализированное состояние (табл. 6.10).

10.Строим множество $z_\varepsilon(M_0(I)) = \{S6, S9, S10, S12\}$. Состояние, соответствующее такому множеству, уже есть в ДКА (*E*). Строим множество $z_\varepsilon(M_1(I)) = \{S8, S9, S10, S12, S21\}$, ставим ему в соответствие состояние ДКА *K*. Заполняем соответствующим образом таблицу переходов. Помечаем проанализированное состояние (табл. 6.11).

Таблица 6.9

Синтез таблицы переходов ДКА (шаг 8)

ДКА	НКА	0	1
A (*)	S0	Ø	B
B (*)	S1	C	Ø
C (*)	S2	Ø	D
D (*)	S3, S4, S5, S7	E	F
E (*)	S6, S9, S10, S12	G	H
F (*)	S8, S9, S10, S12	G	H
G (*)	S11, S14, S15, S17	I	J
H	S13, S14, S15, S17		
I	S4, S5, S7, S16, S19, S20		
J	S4, S5, S7, S18, S19, S20		

Таблица 6.10

Синтез таблицы переходов ДКА (шаг 9)

ДКА	НКА	0	1
A (*)	S0	Ø	B
B (*)	S1	C	Ø
C (*)	S2	Ø	D
D (*)	S3, S4, S5, S7	E	F
E (*)	S6, S9, S10, S12	G	H
F (*)	S8, S9, S10, S12	G	H
G (*)	S11, S14, S15, S17	I	J
H (*)	S13, S14, S15, S17	I	J
I	S4, S5, S7, S16, S19, S20		
J	S4, S5, S7, S18, S19, S20		

11. Строим множество $z_\varepsilon(M_0(J)) = \{S6, S9, S10, S12\}$. Состояние, соответствующее такому множеству, уже есть в ДКА (*E*). Строим множество $z_\varepsilon(M_1(J)) = \{S8, S9, S10, S12, S21\}$. Состояние, соответствующее такому множеству, также есть в ДКА (*K*). Заполняем соответствующим образом таблицу переходов. Помечаем проанализированное состояние (табл. 6.12).

Таблица 6.11

Синтез таблицы переходов ДКА (шаг 10)

ДКА	НКА	0	1
A (*)	S0	Ø	B
B (*)	S1	C	Ø
C (*)	S2	Ø	D
D (*)	S3, S4, S5, S7	E	F
E (*)	S6, S9, S10, S12	G	H
F (*)	S8, S9, S10, S12	G	H
G (*)	S11, S14, S15, S17	I	J
H (*)	S13, S14, S15, S17	I	J
I (*)	S4, S5, S7, S16, S19, S20	E	K
J	S4, S5, S7, S18, S19, S20		
K	S8, S9, S10, S12, S21		

Таблица 6.12

Синтез таблицы переходов ДКА (шаг 11)

ДКА	НКА	0	1
A (*)	S0	Ø	B
B (*)	S1	C	Ø
C (*)	S2	Ø	D
D (*)	S3, S4, S5, S7	E	F
E (*)	S6, S9, S10, S12	G	H
F (*)	S8, S9, S10, S12	G	H
G (*)	S11, S14, S15, S17	I	J
H (*)	S13, S14, S15, S17	I	J
I (*)	S4, S5, S7, S16, S19, S20	E	K
J	S4, S5, S7, S18, S19, S20	E	K
K	S8, S9, S10, S12, S21		

12. Строим множество $z_\varepsilon(M_0(K)) = \{S11, S14, S15, S17, S22\}$, ставим ему в соответствие состояние ДКА *L*. Строим множество $z_\varepsilon(M_1(K)) = \{S13, S14, S15, S17\}$. Состояние, соответствующее такому множеству, уже есть в ДКА (*H*). Заполняем соответствующим образом таблицу переходов. Помечаем проанализированное состояние (табл. 6.13).

Таблица 6.13

Синтез таблицы переходов ДКА (шаг 12)

ДКА	НКА	0	1
A (*)	S0	Ø	B
B (*)	S1	C	Ø
C (*)	S2	Ø	D
D (*)	S3, S4, S5, S7	E	F
E (*)	S6, S9, S10, S12	G	H
F (*)	S8, S9, S10, S12	G	H
G (*)	S11, S14, S15, S17	I	J
H (*)	S13, S14, S15, S17	I	J
I (*)	S4, S5, S7, S16, S19, S20	E	K
J (*)	S4, S5, S7, S18, S19, S20	E	K
K (*)	S8, S9, S10, S12, S21	L	H
L	S11, S14, S15, S17, S22		

13. Строим множество $z_\varepsilon(M_0(L)) = \{S4, S5, S7, S16, S19, S20\}$. Состояние, соответствующее такому множеству, уже есть в ДКА (*I*). Строим множество $z_\varepsilon(M_1(L)) = \{S4, S5, S7, S18, S19, S20, S23\}$, ставим ему в соответствие состояние ДКА *M*. Данное состояние будет принимающим для ДКА, так как содержит принимающее состояние НКА. Заполняем соответствующим образом таблицу переходов. Помечаем проанализированное состояние (табл. 6.14).

14. Строим множество $z_\varepsilon(M_0(M)) = \{S6, S9, S10, S12\}$. Состояние, соответствующее такому множеству, уже есть в ДКА (*E*). Строим множество $z_\varepsilon(M_1(M)) = \{S8, S9, S10, S12, S21\}$. Состояние, соответствующее такому множеству, также есть в ДКА (*K*). Заполняем соответствующим образом таблицу переходов. Помечаем проанализированное состояние.

15. В таблице переходов не осталось не помеченных состояний – синтез детерминированного конечного автомата завершен (табл. 6.15).

Таблица 6.14

Синтез таблицы переходов ДКА (шаг 13)

ДКА	НКА	0	1
A (*)	S0	Ø	B
B (*)	S1	C	Ø
C (*)	S2	Ø	D
D (*)	S3, S4, S5, S7	E	F
E (*)	S6, S9, S10, S12	G	H
F (*)	S8, S9, S10, S12	G	H
G (*)	S11, S14, S15, S17	I	J
H (*)	S13, S14, S15, S17	I	J
I (*)	S4, S5, S7, S16, S19, S20	E	K
J (*)	S4, S5, S7, S18, S19, S20	E	K
K (*)	S8, S9, S10, S12, S21	L	H
L (*)	S11, S14, S15, S17, S22	I	M
M	S4, S5, S7, S18, S19, S20, S23		

Таблица 6.15

Синтез таблицы переходов ДКА (шаг 14)

ДКА	НКА	0	1
A (*)	S0	Ø	B
B (*)	S1	C	Ø
C (*)	S2	Ø	D
D (*)	S3, S4, S5, S7	E	F
E (*)	S6, S9, S10, S12	G	H
F (*)	S8, S9, S10, S12	G	H
G (*)	S11, S14, S15, S17	I	J
H (*)	S13, S14, S15, S17	I	J
I (*)	S4, S5, S7, S16, S19, S20	E	K
J (*)	S4, S5, S7, S18, S19, S20	E	K
K (*)	S8, S9, S10, S12, S21	L	H
L (*)	S11, S14, S15, S17, S22	I	M
M (*)	S4, S5, S7, S18, S19, S20, S23	E	K

Граф переходов синтезированного детерминированного конечного автомата приведен на рис. 6.11.

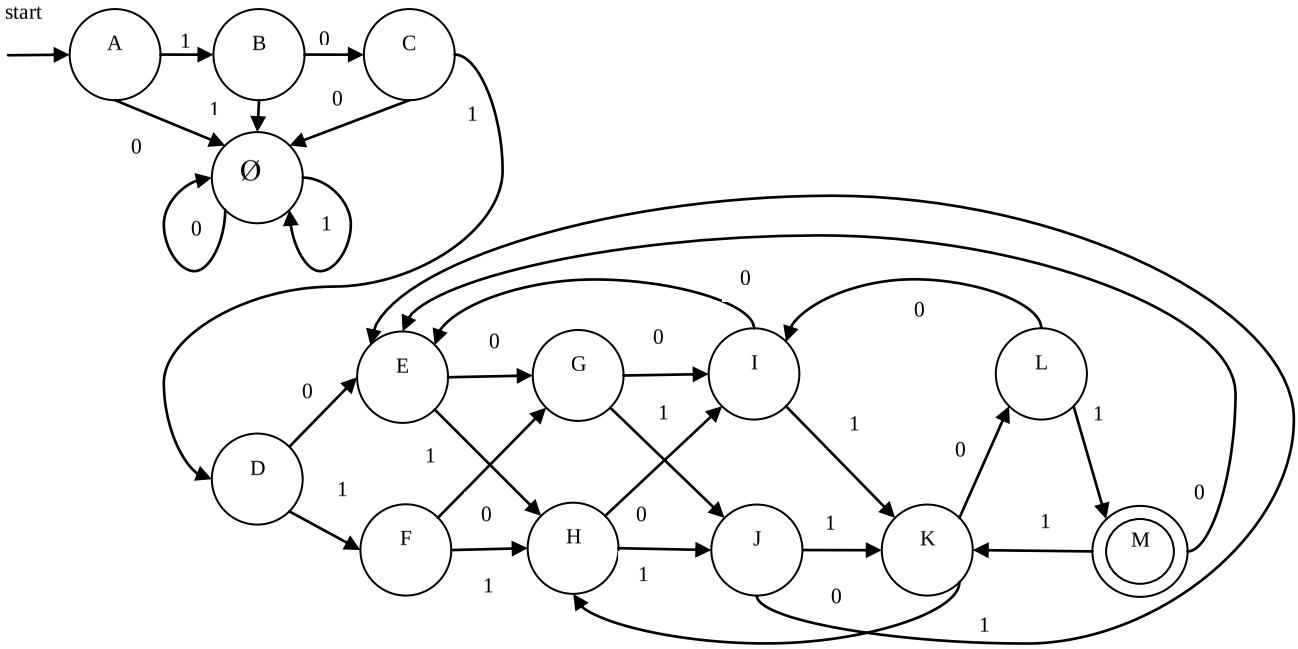


Рис. 6.11. Граф переходов детерминированного конечного автомата,
соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101

Полученный детерминированный конечный автомат не является минимальным. Для его минимизации можно воспользоваться рядом формальных алгоритмов (см. приложение 4), либо упростить его, воспользовавшись идеей, что можно объединить состояния автомата, из которых переходы ведут в одни и те же состояния; при этом все объединяемые состояния должны относиться либо к категории принимающих, либо к категории не принимающих. В данном случае мы можем объединить состояния *E* и *F*, *G* и *H*, *I* и *J*, а состояние *M* с состояниями *I* и *J* объединено быть не может.

Табл. 6.16 описывает переходы для минимизированного автомата, где *S0* – начальное, *S10* – принимающее, а *S2* – тупиковое.

Таблица 6.16

Таблица переходов минимизированного детерминированного конечного автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101

Состояние минимизированного ДКА	Состояния исходного ДКА	Следующее состояние	
		0	1
S0	A	S2	S1
S1	B	S3	S2
S2	Ø	S2	S2
S3	C	S2	S4
S4	D	S5	S5
S5	E, F	S6	S6
S6	G, H	S7	S7
S7	I, J	S5	S8
S8	K	S9	S6
S9	L	S7	S10
S10	M	S5	S8

Синтез устройства проверки сигнала на совпадение с шаблоном

Синтезируем схему устройства проверки совпадения сигнала с шаблоном, определяемым регулярным выражением 101((0|1)(0|1)(0|1))+101, в соответствии с таблицей переходов, приведенной в табл. 6.2. В данном случае 1 соответствует условию (*DATA and READ* = ‘1’), 0 соответствует условию (*not DATA and*

READ). Из состояний *S4*, *S5* и *S6*, в которых оба перехода ведут в одно и то же состояние, будем переходить по условию наличия на входе корректных данных *READ*=’1’. По *RESET*=’1’ будем переходить в начальное состояние *S1*.

Для сокращения количества вводимых в редакторе данных воспользуемся следующим приемом: будем указывать изменение сигналов для выходов *GOOD* и *BAD* только в тех состояниях, где они имеют значение, отличное от стартового состояния *S1*. Во всех других состояниях автоматически для этих сигналов будут использоваться значения, заданные в стартовом состоянии. Правила переключения сигналов *GOOD* и *BAD* приведены в табл. 6.17.

Таблица 6.17

Сокращенная таблица переключений сигналов автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101

Состояние	Переключение выходных сигналов
S0	GOOD<=’0’; BAD<=’0’;
S2	BAD<=’1’
S10	GOOD<=’1’

Макроэлемент спроектированного устройства приведен на рис. 6.12.

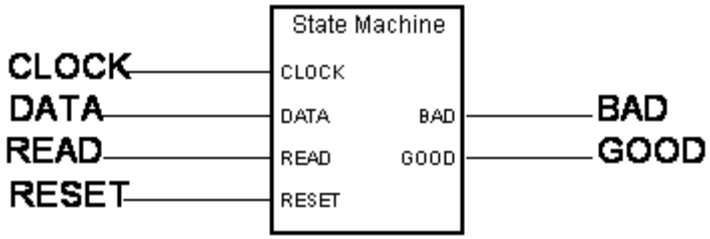


Рис. 6.12. Макроэлемент автомата для проверки сигнала на соответствие заданному шаблону и схема эксперимента

Граф переходов автомата проверки сигнала на соответствие заданному шаблону в нотации Xilinx Foundation FSM Editor’a приведен на рис. 6.13. Временная диаграмма работы автомата, при проверке сигналов 101011101 и 100, приведена на рис. 6.14.

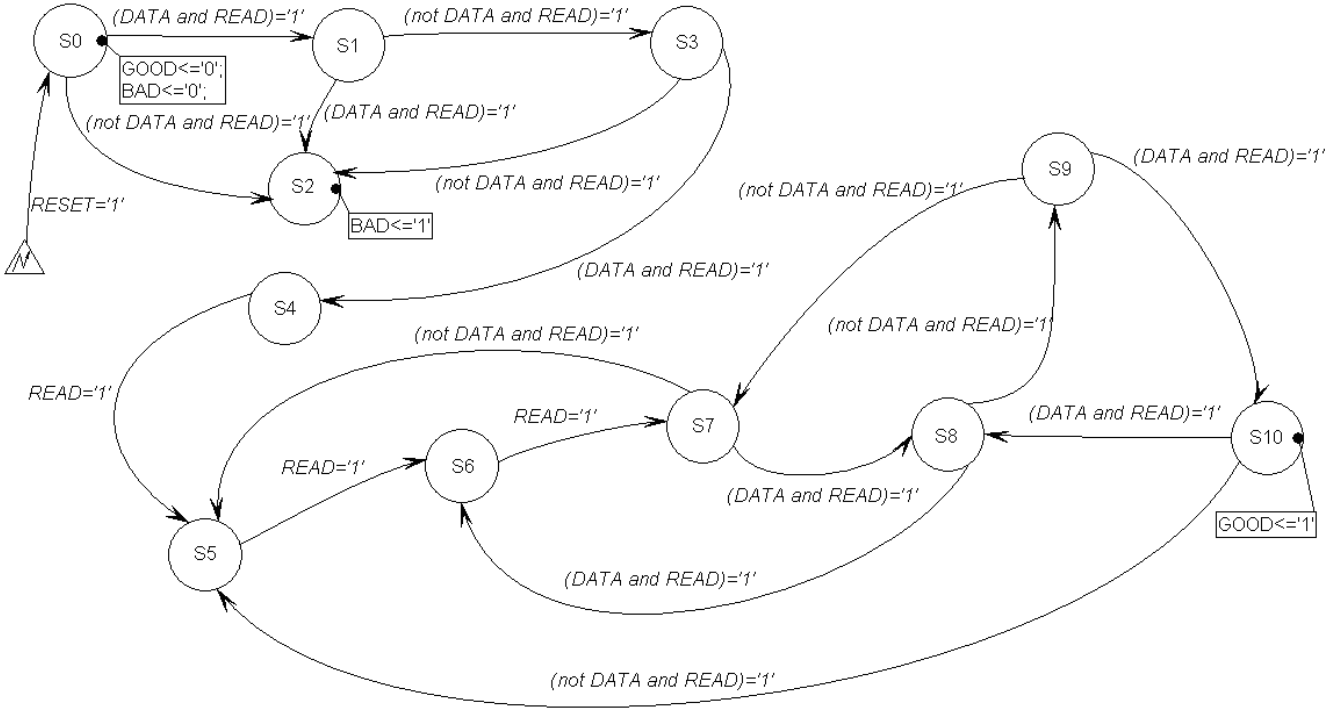


Рис. 6.13. Граф переходов автомата проверки сигнала на соответствие шаблону, заданному в виде регулярного выражения 101((0|1)(0|1)(0|1))+101 в нотации Xilinx Foundation FSM Editor'a

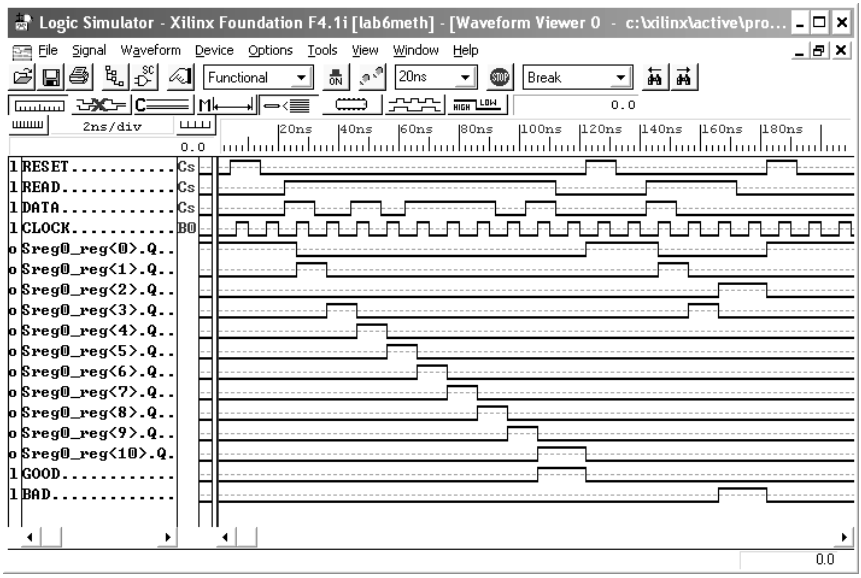


Рис. 6.14. Временная диаграмма работы устройства при проверке соответствия сигнала шаблону, заданному в виде регулярного выражения 101((0|1)(0|1)(0|1))+101

Подготовка к выполнению работы

1. Изучить описание лабораторной работы.
2. Получить индивидуальное задание у преподавателя.
3. Написать и минимизировать регулярное выражение, соответствующее словесному описанию шаблона сигнала.
4. Построить синтаксическое дерево для регулярного выражения, описывающего шаблон распознаваемых сигналов.
5. Синтезировать недетерминированный конечный автомат по построенному синтаксическому дереву.
6. Преобразовать недетерминированный конечный автомат в детерминированный.
7. Опционально: минимизировать детерминированный конечный автомат.
8. Составить таблицы переключений выходных сигналов автомата, в зависимости от состояния, составить описание устройства в виде графа переходов.

9. Составить тесты для проверки корректности функционирования спроектированного устройства.

Порядок выполнения работы

1. Выполнить ввод спроектированной схемы автомата в редакторе схем системы Xilinx Foundation.
2. Выполнить функциональное моделирование автомата.
3. Продемонстрировать преподавателю работу отлаженного автомата.
4. Сдать преподавателю оформленный отчет в конце занятия.

Отчет по работе

Отчет должен содержать:

1) исходные данные варианта задания;
2) все этапы проектирования устройства проверки соответствия сигнала, заданному шаблону;
3) временные диаграммы, иллюстрирующие правильность работы устройства проверки соответствия сигнала, заданному шаблону.

СПИСОК ЛИТЕРАТУРЫ

1. Хопкрофт Джон Э., Мотвани Раджив, Ульман Джеффри, Д.. Введение в теорию автоматов, языков и вычислений, 2-е изд..: Пер. с англ. – М.: Издательский дом “Вильямс”, 2002. – 528 с.

2. Ахо Альфред В, Лам Моника С., Сети Рави, Ульман Джеффри Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд.: Пер. с англ. – М.: ООО “И.Д. Вильямс”, 2008. – 1184 с.

Приложение 1

СХЕМНЫЙ РЕДАКТОР XILINX FOUNDATION

Система сквозного проектирования цифровых устройств на базе ПЛИС Xilinx Foundation Series включает в себя средства схемотехнического ввода, графический редактор, языки описания аппаратуры (HDL) - VHDL, Verilog и Abel, средства моделирования, синтеза структуры кристалла и программирования.

ВХОД В СИСТЕМУ

Для входа в систему необходимо на рабочем столе Windows дважды щелкнуть мышью пиктограмму *Project Manager*. *Project Manager* – графическое средство управления файлами проекта и основными модулями системы автоматизированного проектирования ПЛИС.

Откроется окно **Getting Started** на фоне главного окна системы - **Project Manager** (рис. П1.1).

Работа в системе начинается либо с открытия нового проекта, либо с выбора уже существующего проекта для продолжения выполнения проектных операций и процедур.

Открытие нового проекта

1. В окне **Getting Started** активизируйте опцию *Greate a New Project* и нажмите кнопку **OK** (см. рис. П1.1). Откроется диалоговое окно **New Project** (рис. П1.2).
2. В открывшемся окне в поле **Name** введите имя проекта. В качестве имени проекта должно быть указана фамилия и номер работы. Имя должно содержать не более 8 символов.
3. В поле **Directory** выберите место расположения Вашего проекта – это должен быть диск U:

U:\<фамилия № работы>.

Например, U:\sidorov1.

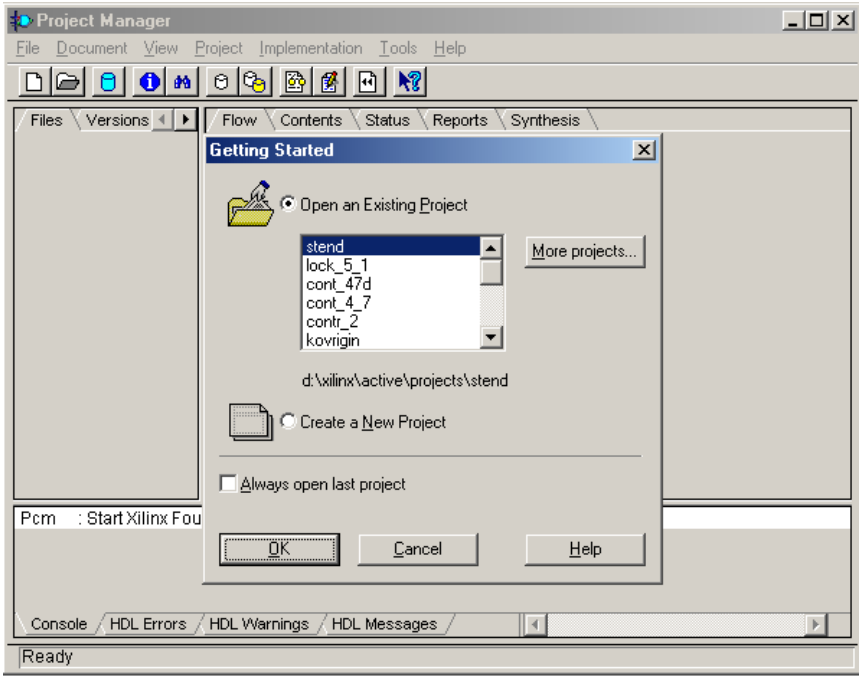


Рис. П1.1. Окно Getting Started

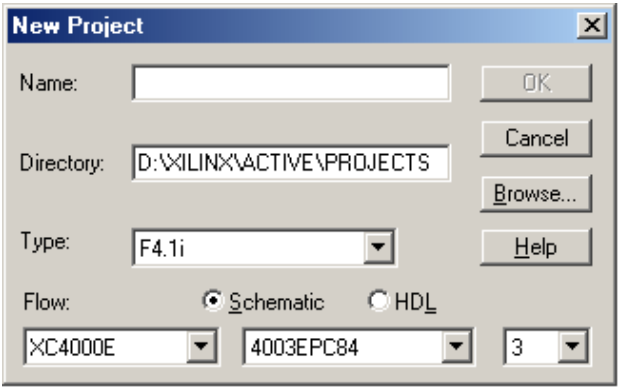


Рис. П1.2. Окно открытия нового проекта

4. Не изменяйте предложенное Вам значение в поле **Type** (см. рис. П1.2).

5. Выберите в окнах **Flow**:
   серию ПЛИС - **Spartan**,
   марку ПЛИС – **S10PC84**,
   и указатель быстродействия – **3**.

Заполненное окно **New Project** приведено на рис. П1.3.

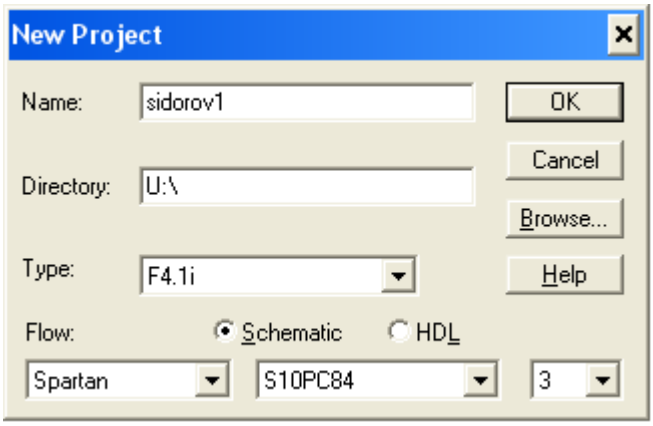


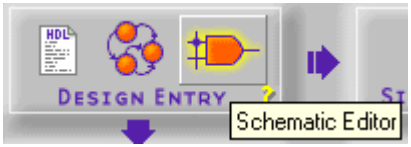
Рис. П1.3. Заполненное окно New Project

6. Нажмите кнопку **OK**.

Имя проекта и марка выбранной ПЛИС с заданным значением быстродействия появятся в поле заголовка главного окна системы (рис. П1.4).

ВВОД И РЕДАКТИРОВАНИЕ СХЕМЫ

Рисование схемы осуществляется с помощью редактора схем. Для входа в редактор следует щелкнуть мышью пиктограмму редактора схем в окне **Project Manager** системы.



Окно редактора схем, открытого впервые, показано на рис. П1.5.

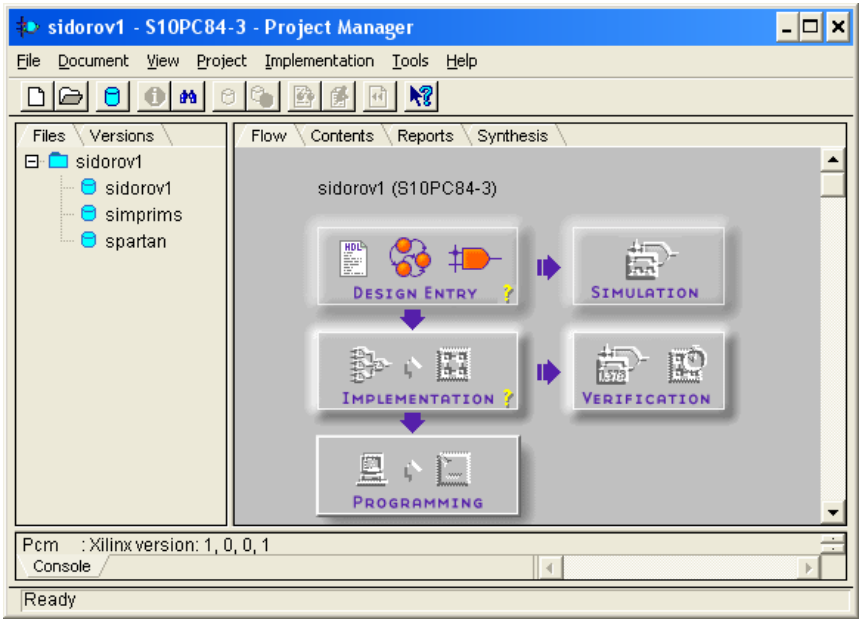


Рис. П1.4. Главное окно системы – Project Manager

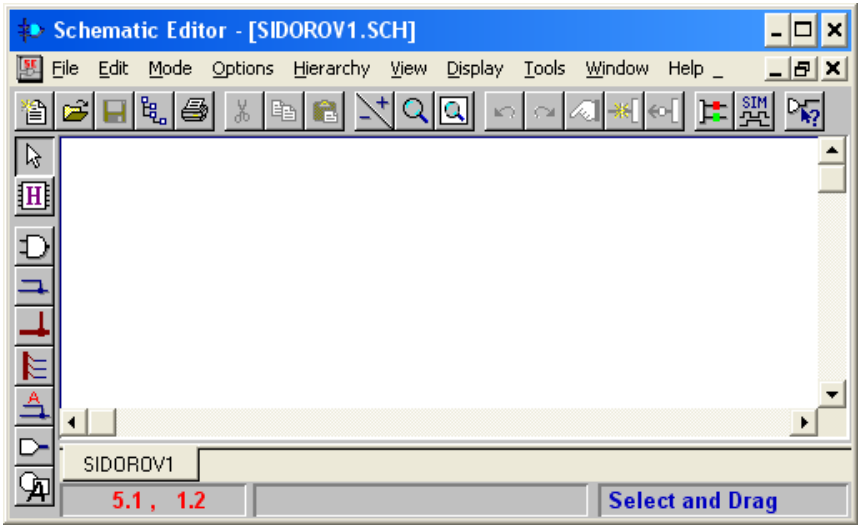


Рис. П1.5. Окно редактора схем

Основные инструменты рисования схемы

Вертикальная панель инструментов


выбор/перемещение объекта


выбор элемента


рисование цепи


рисование шины


рисование проводников, входящих в шину


присвоение имени цепи

Горизонтальная панель инструментов


увеличить/уменьшить изображение


увеличение выбранной зоны изображения


показать весь лист с изображением

вызов окна моделирования схемы


контроль соединений

Основные операции

Выбор элемента

Для выбора требуемого элемента нажмите кнопку «Выбор элемента» на вертикальной панели инструментов 

Появится диалоговое окно SC Symbols выбора модели элемента (рис. П1.6).

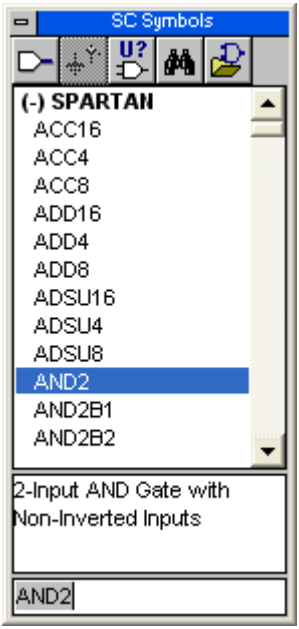


Рис. П1.6. Окно выбора элемента

В этом окне найдите требуемый элемент, затем:

выделите мышью имя найденного элемента (см. рис. П1.6);

переместите курсор в поле рисования схемы. Появится фантом (упрощенный рисунок УГО) выбранного элемента;

передвигая курсор (одновременно будет передвигаться и фантом элемента), выберите требуемую позицию для размещения элемента и нажмите левую клавишу мыши. Появится рисунок УГО выбранного элемента.

Поиск требуемого элемента можно облегчить, сократив обширный общий список элементов до списка элементов, принадлежащих заданному типу. Например, для выбора триггера целесообразно вывести на экран только список всех триггеров.

Для усечения общего списка элементов следует нажать в окне **SC Symbols** кнопку «выбор типа элемента».

Появится окно выбора типа элемента – **Library Filter** (рис. П1.7). В этом окне:

нажать кнопку Clear для сброса флажков у каждого типа элементов (см. рис. П1.7);

выбрать требуемый тип элемента, поставив флажок слева от его имени. Например, на рис. П1.7 выбран тип Flip Flop/Latch;

нажать кнопку OK для окончания операции.

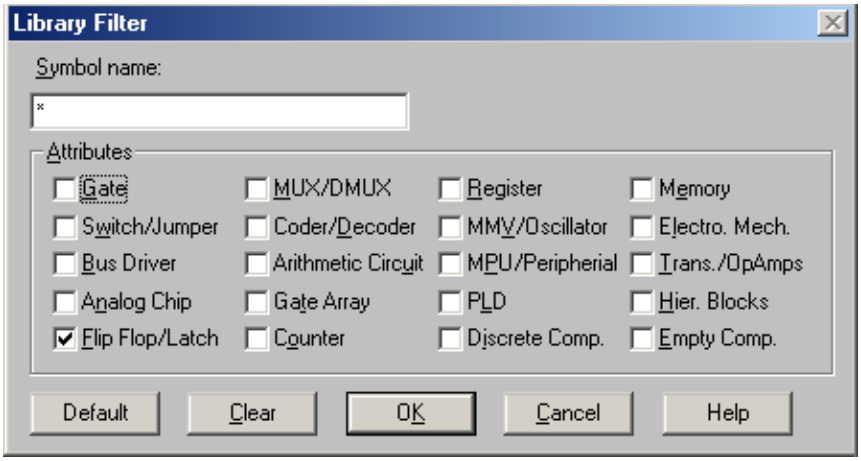


Рис. П1.7. Окно выбора типа элемента

После этого в окне выбора элемента появится список элементов, принадлежащих указанному типу.

Второй вариант нахождения требуемого элемента – это набор его имени в нижней строке окна SC Symbols (см. рис. П1.6).

Примечание. Полный перечень логических элементов приведен в разделе ЛОГИЧЕСКИЕ ЭЛЕМЕНТЫ СХЕМНОГО РЕДАКТОРА в конце этого приложения.

Перемещение элемента или фрагмента схемы

1. Выберите на вертикальной панели инструментов кнопку **Select and Drag**.

2. Для перемещения элемента щелкните мышью его изображение. Элемент будет выделен красной рамкой. Для перемещения фрагмента схемы укажите курсором положение одного из углов прямоугольника, охватывающего фрагмент, и нажмите левую клавишу мыши. Не отпуская клавишу, определите положение противоположного угла прямоугольника и отпустите левую клавишу мыши.

3. Нажмите левой клавишей мыши на выделенный объект и, не отпуская клавишу, переместите его на требуемую позицию и отпустите клавишу. Вместе с объектом будут перемещаться и прикрепленные к нему связи.

Присвоение имени элементу

1. Сделайте двойной щелчок мышью на требуемом элементе. Появится диалоговое окно **Symbol Properties** (рис. П1.8).

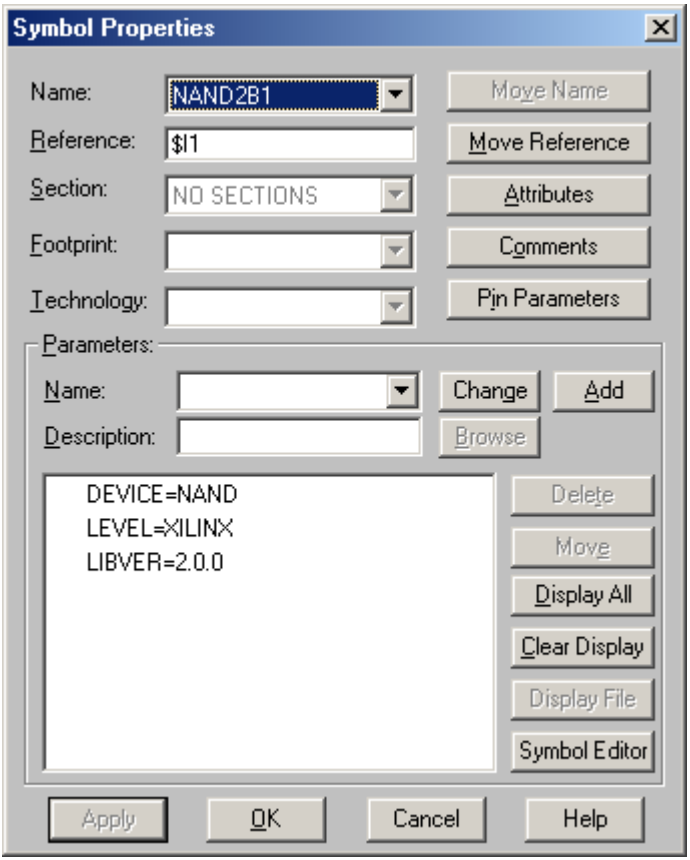


Рис. П1.8. Окно Symbol Properties

2. Наберите имя элемента в поле **Reference** и нажмите кнопку **OK** для окончания операции.

Удаление элемента

Выделите щелчком мыши удаляемый элемент и нажмите на клавиатуре клавишу **Delete.**

Рисование цепи

Создание цепи осуществляется путем рисования ее фрагментов. Фрагмент цепи – это отрезок, соединяющий два объекта. Объектом является либо контакт (входной или выходной) элемента, либо цепь. Таким образом, фрагмент цепи соединяет контакт с контактом, контакт с цепью, цепь с контактом.

Рисование фрагмента цепи выполняется не прорисовкой конфигурации соединения, а только указанием соединяемых объектов, фрагмент цепи прорисовывается автоматически.

Можно начать рисование фрагмента цепи и на свободном поле, но закончить его вы должны на объекте.

1. Щелкните на вертикальной панели инструментов кнопку «рисование цепи» (**Draw Wires**). Курсор примет вид карандаша.

2. Укажите щелчком левой клавиши мыши первый объект или желаемую точку на свободном поле (начало фрагмента цепи), затем вторым щечком – второй объект (конец фрагмента цепи). После этого на экране появится автоматически прорисованный фрагмент цепи.
3. Повторите действия пункта 2 для рисования всей цепи.

Перемещение цепи

1. Переместите курсор на цепь и нажмите левую клавишу мыши.
2. Не отпуская клавишу, переместите цепь на требуемую позицию и отпустите клавишу мыши.

Удаление цепи

Щелкните мышью цепь, которую вы хотите удалить. Она будет выделена цветом. Нажмите на клавиатуре клавишу **Delete**.

Присвоение имени цепи

1. Щелкните дважды цепь, которой вы хотите присвоить имя. Откроется окно **Net Name** (рис. П1.9).
2. Введите имя и нажмите кнопку **OK**.

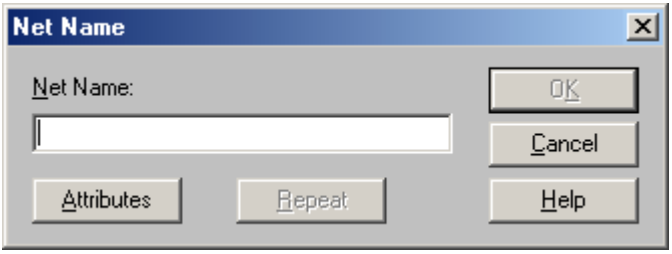


Рис. П1.9. Окно присвоения имени цепи

Удаление имени цепи

Выделите мышью имя, которое вы хотите удалить, и нажмите клавишу **Delete**.

Переименование цепи

1. Щелкните правой клавишей мыши цепь, которую вы собираетесь переименовать. Появится меню операций (рис. П1.10).
2. Если необходимо изменить имя только выделенной цепи, то выберите в меню *Net Properties* (см. рис. П1.10). В появившемся окне **Net Name** (см. рис. П1.9) измените имя и нажмите кнопку **OK**.
3. Если надо в схеме изменить имя всех цепей с данным именем, то выберите в меню *Rename Net* (см. рис. П1.10).

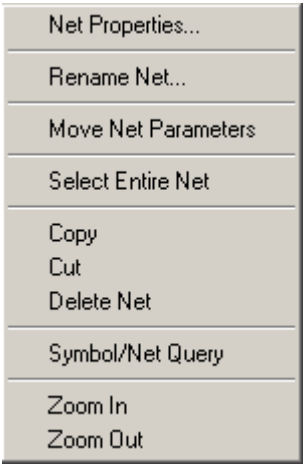


Рис. П1.10. Меню операций над именем цепи

Задание констант 0 и 1

Для подачи на вход элемента постоянного сигнала 0 необходимо подсоединить к нему символ «земля» – ⏚ GND. Для этого из библиотеки элементов необходимо извлечь элемент с именем **GND**.

Для подачи на вход элемента постоянного сигнала 1 необходимо подсоединить к нему символ «питания» – ⊤ VCC. Для этого из библиотеки элементов необходимо извлечь элемент с именем **VCC**.

Шинные соединения

Для более наглядного представления схемы и создания удобств ее рисования в системе предусмотрено использование шин.

Шина – это совокупность проводников, изображаемая на рисунке одной утолщенной линией.

Имя шины состоит из идентификатора и индексных пределов, заключенных в квадратные скобки: *BUS_NAME [X:Y],*

где X и Y – целые числа больше или равные 0. Возможно как X>Y, так и X<Y.

Каждый проводник (сигнал), входящий в шину, получает имя шины и индекс: *BUS_NAMEN,*

где *N* – индексное значение, заключенное между X и Y.

Внимание! *Следует иметь в виду, что проводники шины упорядочиваются (приобретают индексы), начиная с левого индекса X. При выполнении операций с шиной ее крайний левый бит (индекс) всегда считается старшим.*

Например, шина DATA[3:0] представляет набор дискретных сигналов:

DATA3, DATA2, DATA1, DATA0,

здесь DATA3 – старший бит шины.

Шина DATA[0:3] представляет тот же набор сигналов, но в обратном порядке:

DATA0, DATA1, DATA2, DATA3,

здесь старший бит шины – DATA0.

Это позволяет изменять соединения в схеме без удаления и перерисовки шинных соединений, для этого достаточно в имени шины поменять местами индексы X и Y.

Внимание! *Имя шины не должно заканчиваться цифрой.*

Иначе могут быть непредвиденные подключения. Например, шины DATA1[0:7] и DATA[0:10] совместно используют сигнал DATA10, который является членом обеих шин.

Пример простого шинного соединения

Рассмотрим на примере простейший вариант использования шин при рисовании схемы. Допустим, требуется соединить четыре выхода регистра с входами пяти элементов AND2 (рис. П1.11). Выход Q0 должен быть соединен с входами двух верхних элементов AND2.

1. Выберите на вертикальной панели инструментов кнопку «рисование шины» (Draw Buses). 

2. Укажите щелчком левой клавиши мыши желаемую точку на свободном поле (начало шины). Затем, фиксируя расположение и повороты проводимой шины щелчком левой клавиши мыши, создайте требуемый рисунок шины. В завершении рисунка шины нажмите правую клавишу мыши, появится выпадающее меню (рис. П1.11).

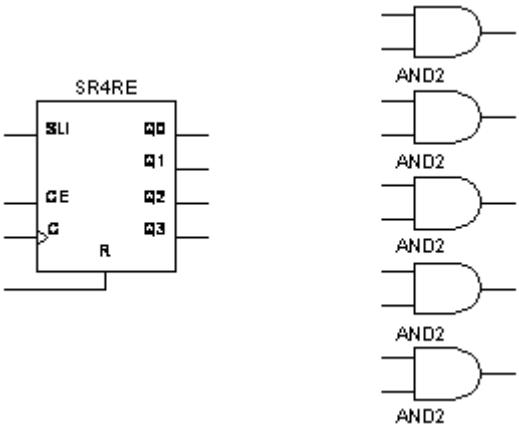


Рис. П1.11. Пример логической схемы

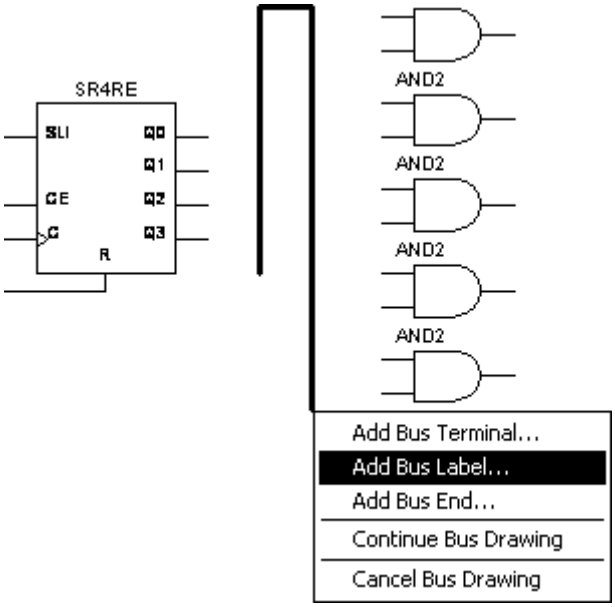


Рис. П1.12. Рисование шины

3. Выберите в меню **Add Bus Label** (рис. П1.12). Откроется окно **Add Bus Terminal/label**, в котором присвойте имя шине и ука-

жите индексные пределы (рис. П1.13). Нажмите кнопку **OK**. На рисунке шины появится введенное имя с индексными пределами (рис. П1.14).

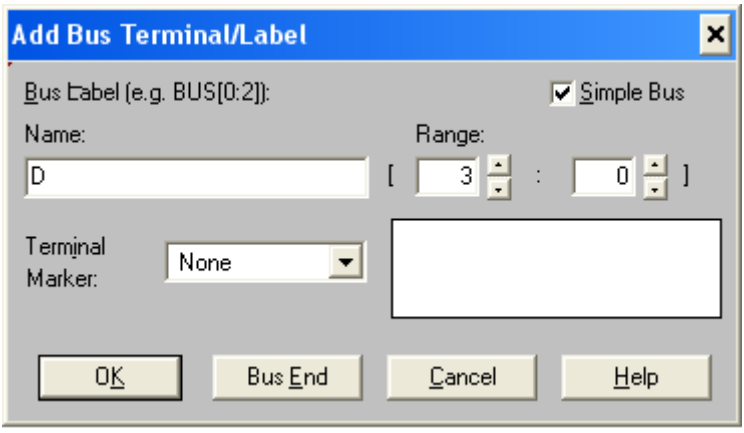


Рис. П1.13. Именование шины

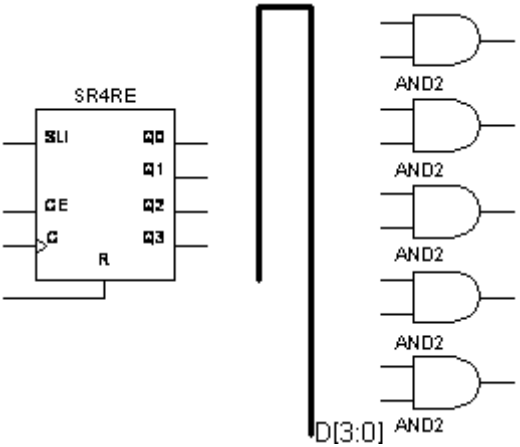


Рис. П1.14. Шина с присвоенным именем

4. Выберите на вертикальной панели инструментов кнопку Draw Bus Tap. Курсор примет новую форму.

5. Щелкните шину, к которой вы хотите подключить контакты элементов. В строке состояния появится номер текущего проводника шины.

6. Щелкните требуемый контакт элемента. Между шиной и контактом элемента автоматически прорисуется соединение, которому будет присвоено текущее имя сигнала шины, указанное в строке состояния (рис. П1.15). После этой операции индекс текущего сигнала шины будет увеличен на единицу. Индекс текущего сигнала шины можно менять нажатием клавиш перемещения курсора вверх или вниз.

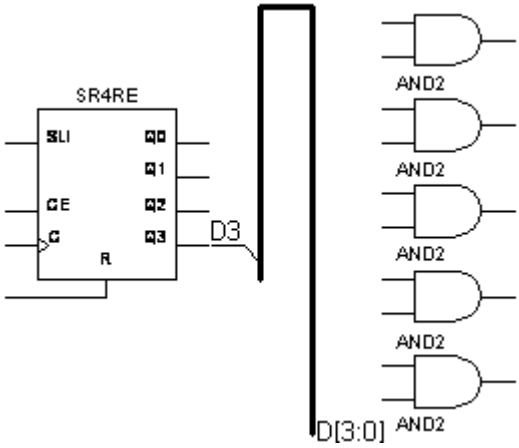


Рис. П1.15. Подключение контакта элемента к шине

7. Щелкните другие контакты элементов в требуемой последовательности, меняя при необходимости текущий индекс в строке состояния (рис. П1.16).

8. После окончания нажмите правую клавишу мыши и в появившемся меню выберите режим **Select and Drag**.

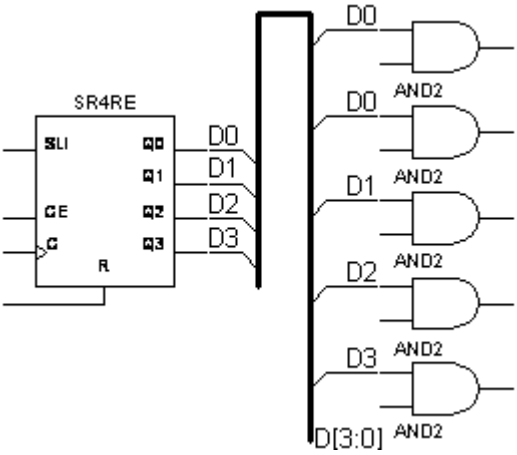


Рис. П1.16. Шинное соединение

Сохранение схемы

Нажмите кнопку сохранения на горизонтальной панели инструментов редактора схем.

ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ СХЕМЫ

Окно моделирования схемы

Открыть окно моделирования схемы можно одним из следующих способов:

щелкнуть мышью пиктограмму «Simulator» на горизонтальной панели инструментов редактора схем;

щелкнуть мышью пиктограмму «Simulation» в окне *Project Manager*.

На рис П1.17 приведено окно моделирования схемы **Logic Simulator**.

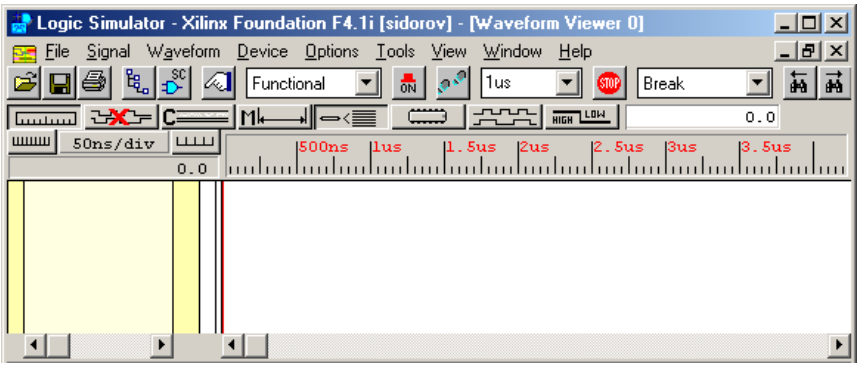


Рис. П1.17. Окно моделирования схемы

Панель инструментов окна моделирования

Верхний ряд панели инструментов:

– загрузка временной диаграммы,

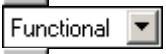
– сохранение временной диаграммы,

– печать временной диаграммы,

– открытие окна менеджера проектов,

– открытие окна редактора схем,

– задание частоты синхросигналов,



– выбор режима моделирования,

– возврат моделирования в начало,

– выполнение моделирования по шагам,

– выбор шага моделирования,

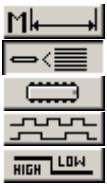
– останов моделирования при генерации схемы.

Второй ряд панели инструментов:

– выключение шкалы времени,

– удаление всех временных диаграмм,

– убрать вставленный комментарий,



– убрать вставленные измерения,

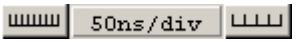
– развернуть свернутую группу сигналов,

– выбор контрольных точек схемы,

– задание периодических входных сигналов,

– задание состояния сигнала.

Третий ряд панели инструментов:



– изменение масштаба шкалы времени.

Выбор контрольных точек схемы

Выбор контрольных точек предполагает выбор как внешних входов схемы, так и требуемых точек схемы для просмотра временных диаграмм.

1. Щелкните кнопку выбора контрольных точек на панели инструментов окна моделирования.

Появится окно выбора контролируемых точек (рис. П1.18). Окно имеет три колонки: **Signals selection** – колонка имен цепей и шин, **Chip selection** – колонка имен элементов, **Scan Hierarchy** – колонка имен элементов, входящих в макроэлементы.

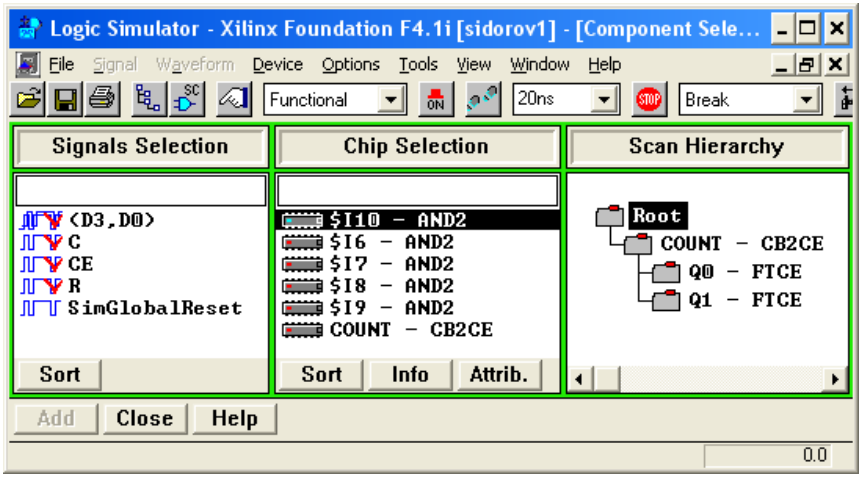


Рис. П1.18. Окно выбора контрольных точек

2. Выберите имя в левой колонке **Signals selection**, затем нажмите кнопку **Add**. Альтернативный способ – двойной щелчок на имени цепи или шины. Выбранное имя будет помечено галочкой красного цвета (см. рис. П1.18).

3. Для просмотра временных диаграмм на контактах элемента дважды щелкните имя элемента в средней колонке **Chip selection**. После этого в правой колонке окна появится список контактов выбранного элемента и колонка сменит заголовок на **Pins for:** с указанием имени выбранного элемента (рис. П1.19).

4. Выберите двойным щелчком мыши контакт элемента в правом столбце окна. Выбранный контакт будет помечен галочкой (см. рис. П1.19).

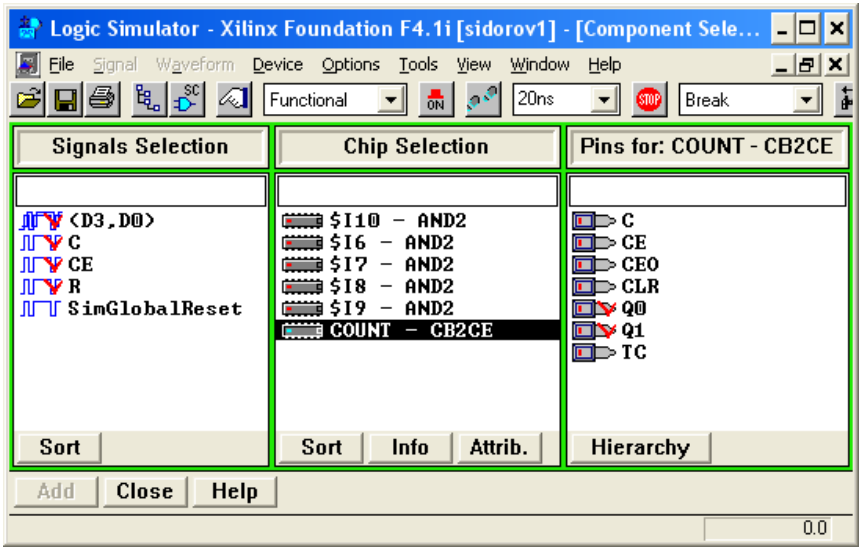


Рис. П1.19. Окно выбора контрольных точек элемента

5. Повторите пп. 3-4 для других элементов, временные диаграммы которых необходимо вывести на просмотр.

6. Нажмите кнопку **Close**, чтобы завершить операцию выбора контрольных точек схемы.

Удаление контрольных точек из окна моделирования

Выделите в поле имен окна моделирования имя сигнала, который вы хотите удалить и нажмите правую клавишу мыши. В появившемся меню выберите **Delete Signal** и затем щелкните **Selected** (рис. П1.20). Имя сигнала будет удалено.

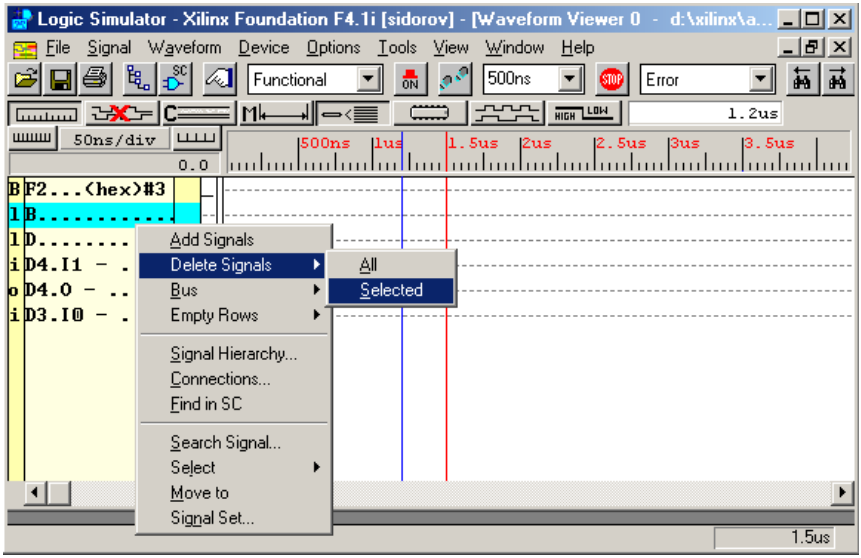


Рис. П1.20. Удаление сигнала

Изменение порядка имен в окне моделирования

Если вас не устраивает порядок расположения имен контрольных точек в поле имен окна моделирования (см. рис. П1.20), то вы можете его изменить следующим образом:

1. Поместите указатель мыши на то имя, которое вы хотите переместить. Нажмите левую клавишу мыши и начинайте движение. Вслед за указателем будет перемещаться контур имени.

2. Перемещайте мышь, пока контур не займет желаемого положения. После этого отпустите кнопку мыши.

Задание синхросигналов

1. Для задания синхросигналов на требуемом внешнем входе сначала выделите щелчком его имя в поле имен окна моделирования схемы.

2. Затем нажмите кнопку задания периодических входных воздействий на панели инструментов окна моделирования (см. рис. П1.17). Появится окно **Simulator Selection** (рис. П1.21).

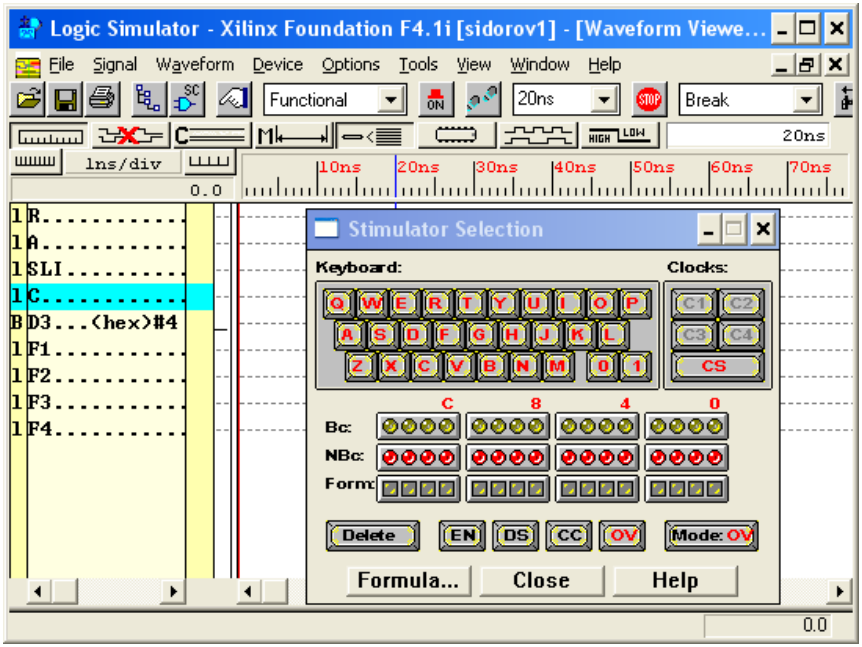


Рис. П1.21. Окно задания периодических входных воздействий

Встроенные генераторы периодических сигналов в этом окне представлены изображением светодиодов в двух строчках **Bc:** и **NBc:** (прямое и инверсное значения сигналов). Каждая строка светодиодов представляет собой выходы 16-разрядного двоичного счетчика. Выходы счетчика могут быть поданы на любое число внешних входов. Частота переключения младшего разряда счетчи-

ка определяется, как базовая частота синхросигналов и устанавливается в окне **Preferences**. На остальных выходах счетчика будет появляться пересчитанная базовая частота.

3. Щелкните требуемый светодиод генератора сигналов в окне задания входных воздействий (см. рис. П1.21). В окне моделирования имя входа будет помечено символом *B* с указанием номера выбранного разряда счетчика, например, *B0* - младший разряд счетчика, *B3* - четвертый разряд.

4. Закройте окно Stimulator Selection**,** нажав кнопку *Close* (см. рис. П1.21).

Как отсоединить генератор от внешнего входа схемы

1. Выделите имя внешнего входа в окне моделирования (см. рис. П1.21).

2. Нажмите кнопку *Delete* в окне задания периодических входных воздействий (см. рис. П1.21).

Задание входных воздействий в окне моделирования

1. Щелкните правую клавишу мыши в поле временных диаграмм окна моделирования схемы. Появится меню, в котором щелкните опцию **Edit** (рис. П1.22)**.** Откроется окно **Test Vector State Selection** (рис. П1.23).

2. Расположите указатель мыши в поле задания сигнала окна моделирования (рис. П1.24) напротив имени внешнего входа и нажмите левую клавишу мыши. Не отпуская клавиши, переместите вправо курсор на отрезок времени с неизменным значением сигнала. Данный отрезок будет выделен (см. рис. П1.24). После чего щелкните мышью кнопку **Low** или **High** в окне **Test Vector State Selection** для задания требуемого значения входного сигнала 0 или 1 на данном отрезке.

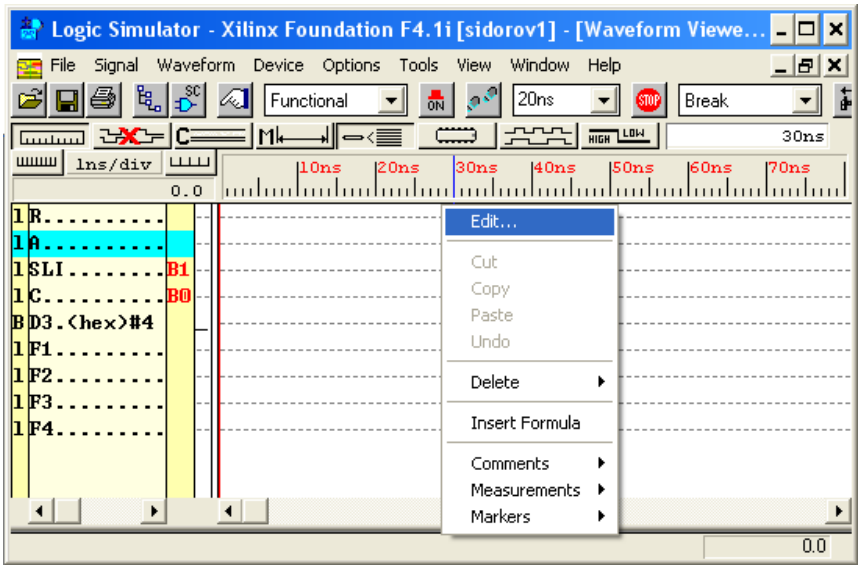


Рис. П1.22. Меню выбора редактора входных временных диаграмм

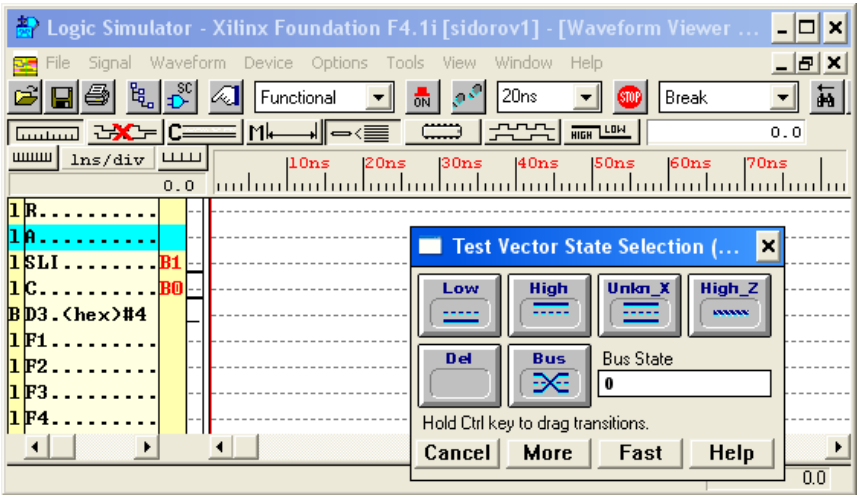


Рис. П1.23. Окно Test Vector State Selection

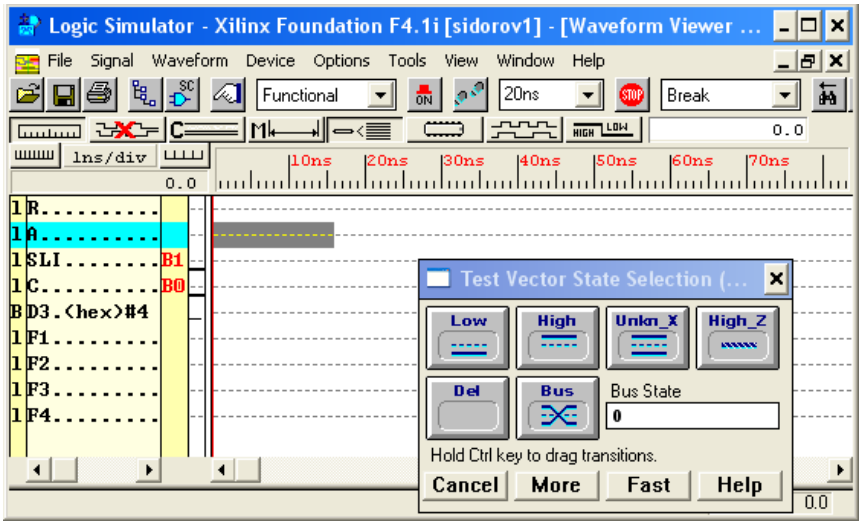


Рис. П1.24. Выделение отрезка времени на временной диаграмме

3. Далее продолжайте выделение новых отрезков времени и задание требуемых значений сигнала (рис. П1.25).

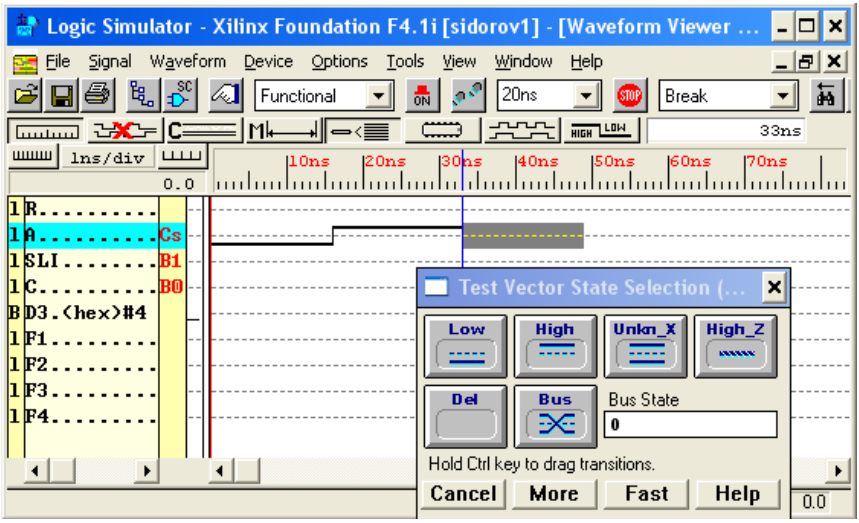


Рис. П1.25. Построение входного сигнала

Выполнение моделирования схемы

Для моделирования схемы на заданную последовательность входных сигналов выполните следующие действия.

1. Установите шаг моделирования 10 нс или 20 нс, открыв в окне моделирования «Выбор шага моделирования» (рис. П1.26).

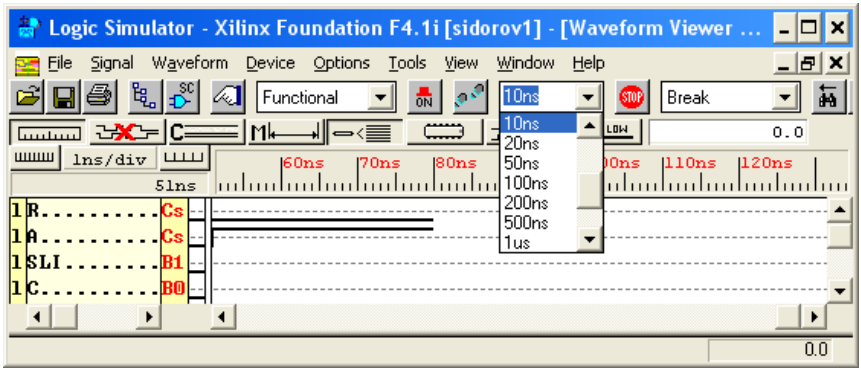


Рис. П1.26. Выбор шага моделирования

2. Затем последовательно нажимайте кнопку выполнения моделирования по шагам в окне **Logic Simulator**  (рис. П1.27).

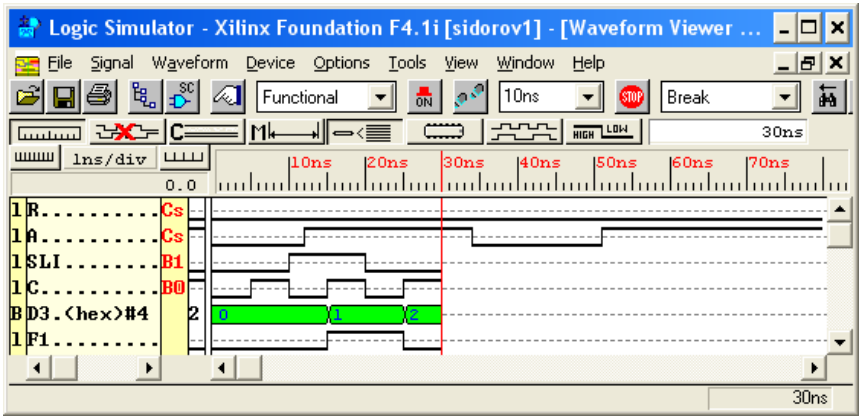
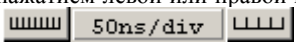


Рис. П1.27. Моделирование схемы по шагам

Примечание. Изменение масштаба отображения временной диаграммы осуществляется нажатием левой или правой кнопок панели шкалы времени окна моделирования 

(см. рис. П1.27).

Сохранение результатов моделирования

1. Щелкните кнопку *«Сохранение временной диаграммы»* на панели инструментов окна моделирования. Появится окно **Save Waveform** (рис. П1.28).

2. Введите имя файла с расширением .tve и нажмите кнопку **OK**.

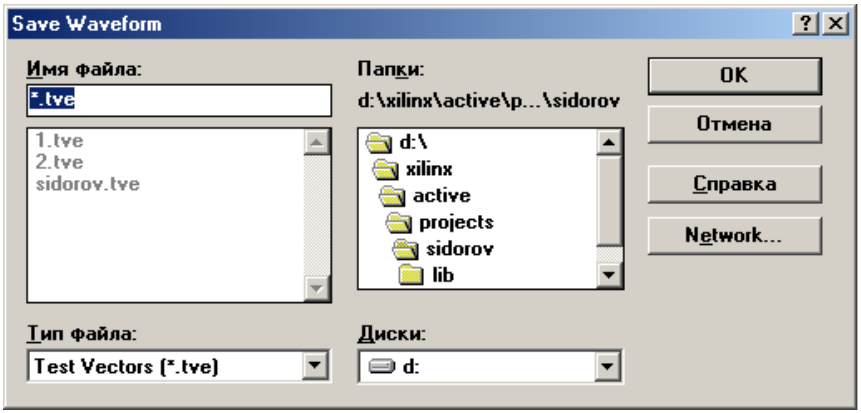


Рис. П1.28. Окно сохранения временных диаграмм

ЛОГИЧЕСКИЕ ЭЛЕМЕНТЫ СХЕМНОГО РЕДАКТОРА

Группа *Gate* окна выбора типа элемента (см. рис. П1.7) содержит следующий набор логических элементов:

- инвертор НЕ (**INV**);
- элементы И (конъюнктор, **AND**) с инверсными и не инверсными входами от 2 до 9;
- элементы ИЛИ (дизъюнктор, **OR**) с инверсными и не инверсными входами от 2 до 9;
- элементы ИНЕ (**NAND**) с инверсными и не инверсными входами от 2 до 9;

- элементы ИЛИНЕ (**NOR**) с инверсными и не инверсными входами от 2 до 9;
- элементы “Исключающее ИЛИ” (сложение по модулю 2, **XOR**) с числом входов от 2 до 9;
- элементы, выполняющие функцию логическая равнозначность (**XNOR**) с числом входов от 2 до 9;
- элементы 2-2И-2ИЛИ (**SOP4**) с инверсными и не инверсными входами;
- элементы 1-2И-2ИЛИ (**SOP3**) с инверсными и не инверсными входами.

Расшифровка обозначений элементов этой группы достаточно проста, поскольку тип элемента явно указан в его названии, а число обозначает количество входов. Кроме того, у элементов некоторые входы могут быть проинвертированы, что отмечается символом *B* в обозначении элемента. Пример соглашения, принятого для обозначения логических элементов, приведен на рис. П1.29.

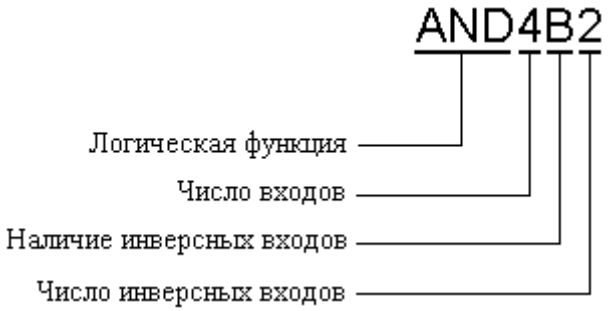


Рис. П1.29. Обозначение логических элементов

Инвертор НЕ (INV)

Условное графическое обозначение инвертора, приведено на рис. П1.30. Инвертор является базовым элементом.

Рис. П1.30. Условное графическое обозначение инвертора

Элементы И (AND)

Условные графические обозначения элементов AND, приведены на рис. П1.31. Элементы AND до пяти входов включительно являются базовыми элементами и могут иметь инверсные входы в любой комбинации (см. рис. П1.31).

Элементы AND с числом входов от 6 до 9 не имеют инверсных входов и являются макроэлементами.

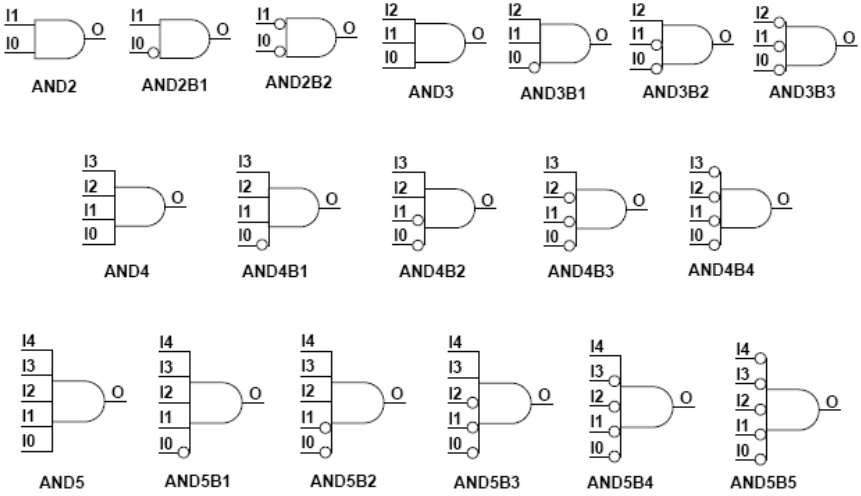


Рис. П1.31. Условное графическое обозначение элементов AND

Элементы ИЛИ (OR)

Условные графические обозначения элементов OR приведены на рис. П1.32. Элементы OR до пяти входов включительно являются базовыми элементами и могут иметь инверсные входы в любой комбинации (см. рис. П1.32).

Элементы OR с числом входов от 6 до 9 не имеют инверсных входов и являются макроэлементами.

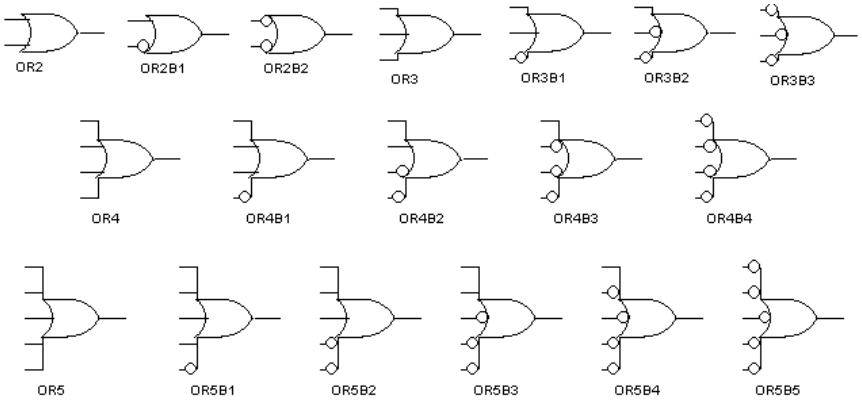


Рис. П1.32. Условное графическое обозначение элементов OR

Элементы ИНЕ (NAND)

Условные графические обозначения элементов NAND, приведены на рис. П1.33. Элементы NAND до пяти входов включительно являются базовыми элементами и могут иметь инверсные входы в любой комбинации.

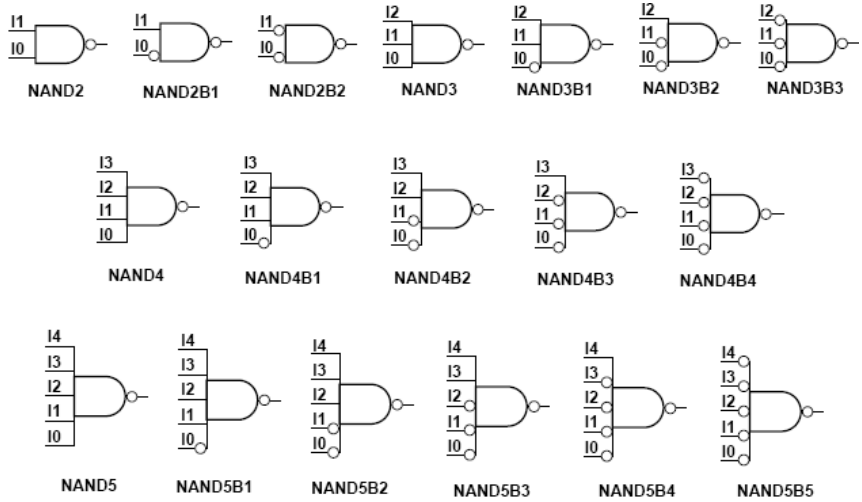


Рис. П1.33. Условное графическое обозначение элементов NAND

Элементы ИЛИНЕ (NOR)

Условные графические обозначения элементов NOR приведены на рис. П1.34. Элементы NOR до пяти входов включительно являются базовыми элементами и могут иметь инверсные входы в любой комбинации (см. рис. П1.34). Элементы NOR с числом входов от 6 до 9 не имеют инверсных входов и являются макроэлементами.

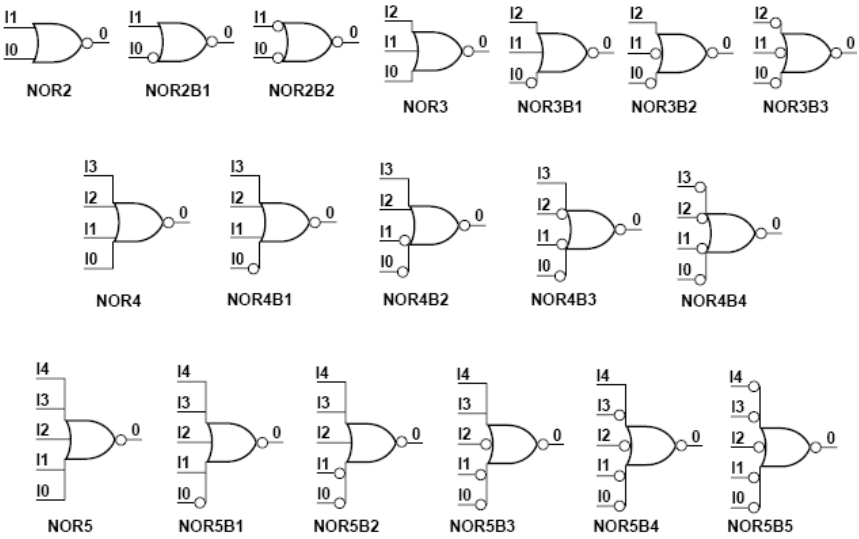


Рис. П1.34. Условное графическое обозначение элементов NOR

Элементы 2-2И-2ИЛИ (SOP4)

Условные графические обозначения элементов SOP4 приведены на рис. П1.35. Элементы SOP4 являются макроэлементами и могут иметь инверсные входы в любой комбинации (см. рис. П1.35).

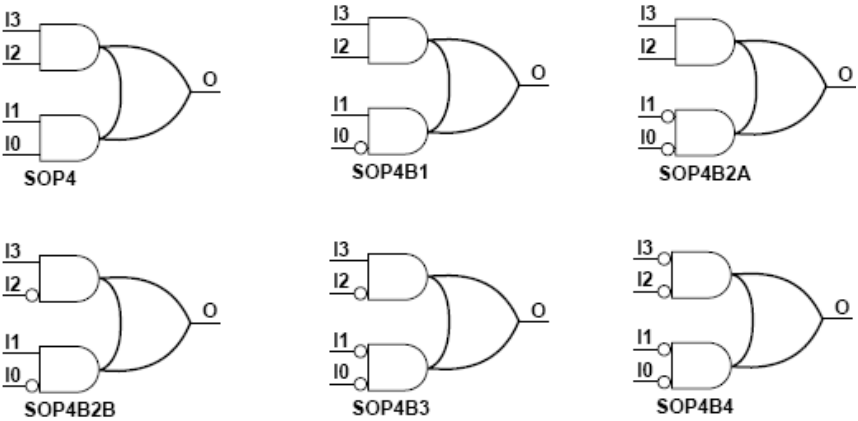


Рис. П1.35. Условное графическое обозначение элементов SOP4

Элементы 1-2И-2ИЛИ (SOP3)

Условные графические обозначения элементов SOP3 приведены на рис. П1.36. Элементы SOP3 являются макроэлементами и могут иметь инверсные входы в любой комбинации (см. рис. П1.36).

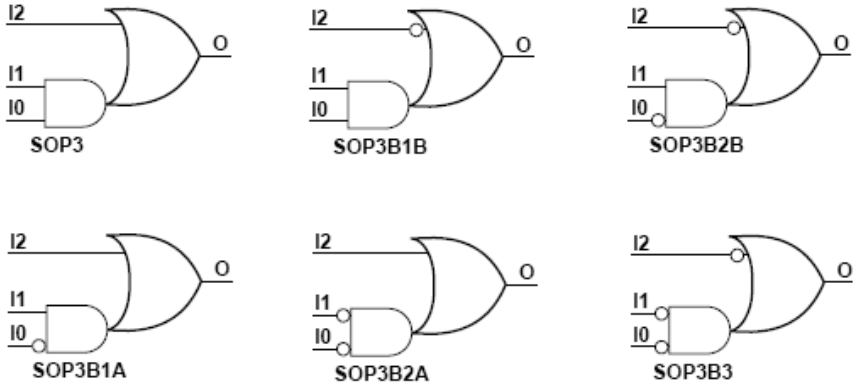


Рис. П1.36. Условное графическое обозначение элементов SOP3

Элементы «ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR)» и «ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ (XNOR)»

В библиотеке элементы “Исключающее ИЛИ” (сложение по модулю 2, **XOR**) и элементы “Исключающее ИЛИ-НЕ” (логическая равнозначность, **XNOR**) имеют число входов от 2 до 9.

Условное графическое обозначение элементов XOR2, XNOR2 с двумя входами приведено на рис. П1.37, а их таблица истинности представлена табл. П1.1.

Рис. П1.37. Условное графическое обозначение элементов XOR и XNOR

Таблица П1.1

Таблица истинности элементов XOR2, XNOR2

Входы		Выход элемента XOR	Выход элемента XNOR
I0	I1	Q	Q
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Приложение 2

СРЕДСТВА ВИЗУАЛЬНОЙ РАЗРАБОТКИ ЦИФРОВЫХ АВТОМАТОВ

Система сквозного проектирования цифровых устройств на базе ПЛИС Xilinx Foundation Series включает подсистему FINITE STATE MACHINE EDITOR (FSM Editor), которая представляет собой графический редактор, позволяющий создавать описание конечного автомата в форме графа переходов (диаграммы).

Открытие редактора FSM Editor

1. После входа в систему и появления главного окна системы – Project Manager (рис. П2.1) следует щелкнуть мышью пиктограмму редактора FSM Editor.

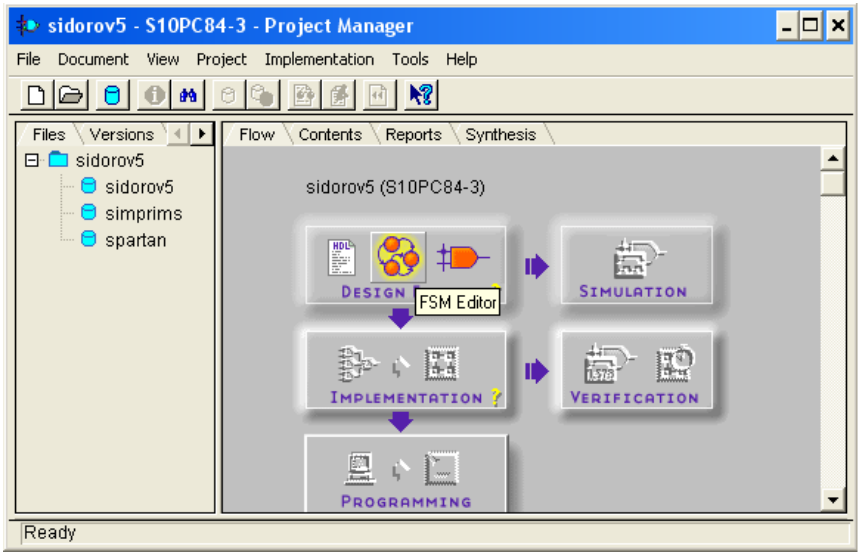


Рис. П2.1. Главное окно системы Project Manager

Откроется окно выбора документа *State Editor* (рис. П2.2). В этом окне пользователю предлагается либо создать новый доку-

мент – раздел **Create new document**, либо открыть существующий документ – раздел **Open.**

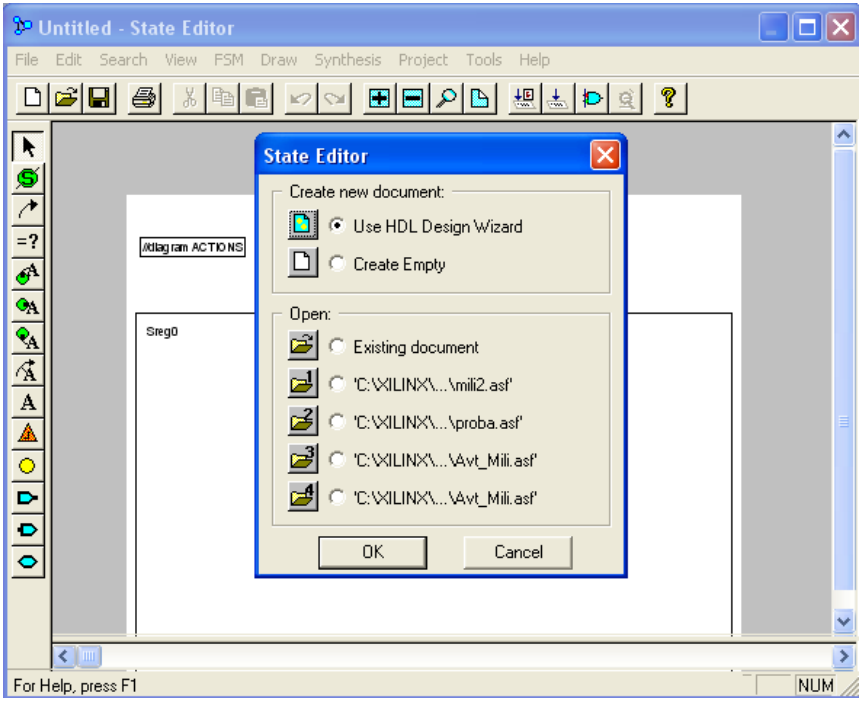


Рис. П2.2. Окно выбора документа State Editor

2. При первом открытии для удобства работы целесообразно воспользоваться мастером конструирования (Design Wizard). Design Wizard автоматизирует процесс создания начального каркаса нового документа. Выберите пункт **Use HDL Design Wizard** (см. рис. П2.2).

3. Нажмите кнопку **OK**.

После запуска откроется вводное окно с общими указаниями (рис. П2.3).

Рис. П2.3. Окно с общими указаниями

4. Нажмите в этом окне кнопку **Далее**. Откроется окно выбора языка HDL (рис. П2.4).

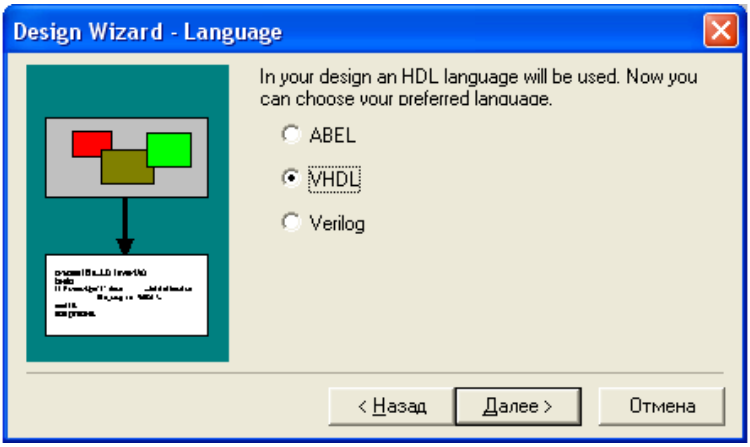


Рис. П2.4. Окно выбора языка HDL

5. Выберите язык VHDL и нажмите кнопку **Далее**. Появится окно выбора имени создаваемого автомата (рис. П2.5).

6. Задайте имя автомата. В качестве имени следует набрать номер своего задания. Например, **var34.**

ВНИМАНИЕ! Имя автомата и имя проекта не должны совпадать.

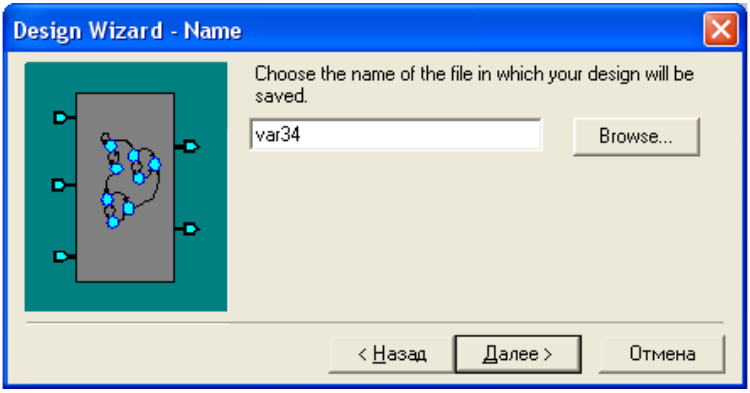


Рис. П2.5. Задание имени автомата

7. Нажмите кнопку **Далее.** Появится окно задания внешних контактов (портов) автомата (рис. П2.6).

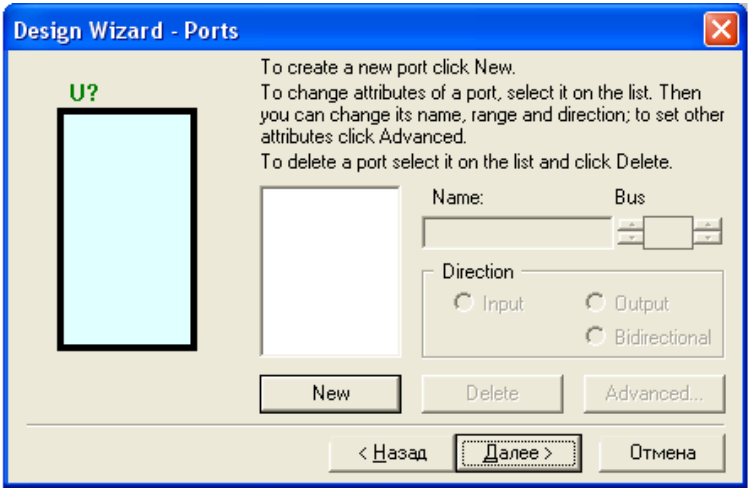


Рис. П2.6. Окно Design Wizard – Ports

Пример

Дальнейшее рассмотрение работы с редактором FSM Editor проведем на примере создания графа переходов триггера (табл. П2.1). Любой триггер является элементарным автоматом Мура.

Таблица П2.1
Таблица переходов триггера

$E_1(t)$	$E_2(t)$	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	Q(t)

Все переходы реализуются по фронту 0/1 синхроимпульсов. Состояния, помеченные временем t в таблице, – это состояния до момента изменения 0/1 синхроимпульса, а состояния со временем t+1 – это состояния, которые примет триггер после изменения 0/1 синхроимпульса. Триггер также должен иметь асинхронный вход установки в состояние «0».

Таким образом, триггер имеет следующие входы и выходы: C – синхронизирующий вход, CLR – асинхронный вход установки триггера в состояние «0», E1, E2 – логические входы, Q – прямой выход, NQ – инверсный выход.

Определение внешних контактов автомата

Для задания очередного нового контакта (порта) проделайте следующую последовательность действий.

1. Нажмите кнопку **New** (см. рис. П2.6) для задания очередного нового контакта (порта). Это вызовет активизацию всех полей окна **Port**.
2. Введите в поле **Name** имя контакта.
3. Выберите на панели **Direction** направление контакта: входной (In), выходной (Out), двунаправленный (Bidirectional).

На рис. П2.7 показано задание входа C.

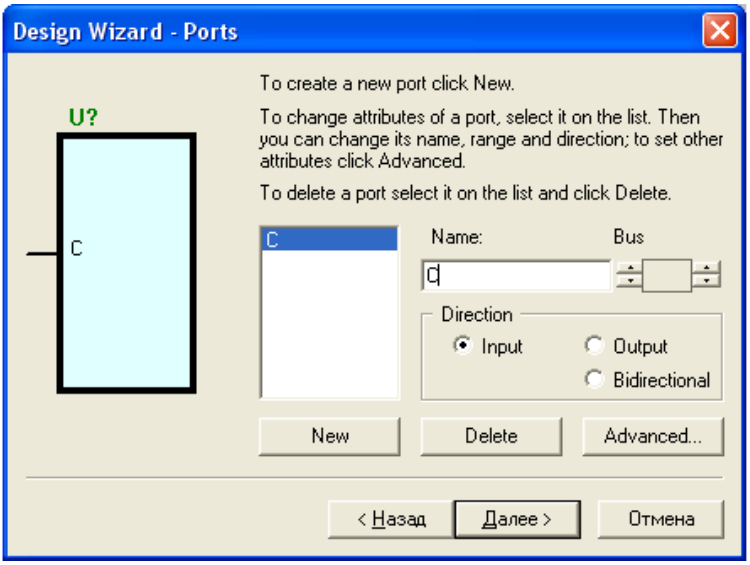


Рис. П2.7. Задание входа C

ВНИМАНИЕ! Синхронизирующий вход в окне **Ports** необходимо определить, задав его тип.

4. Для этого нажмите кнопку **Advanced** (см. рис. П2.7). Появится окно дополнительных установок порта (рис. П2.8). Поставьте галочку в поле **Clock.** Нажмите кнопку OK.

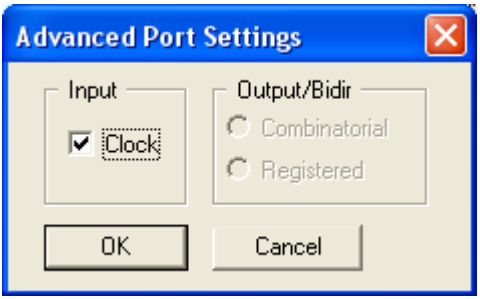


Рис. П2.8. Задание входа C как синхронизирующего

Чтобы добавить новые порты, следует выполнить шаги 1-3. При создании очередного порта мастер добавляет его изображение в условное графическое обозначение создаваемого устройства. Это графическое обозначение является средством визуального контроля выполняемых действий.

На рис П2.9 показан вид окна Ports после определения всех контактов.

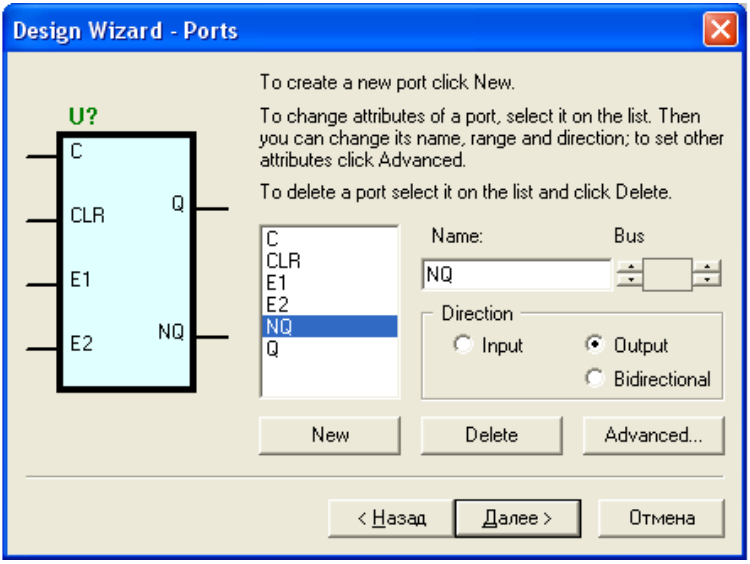


Рис. П2.9. Вид окна Ports после определения всех контактов

ВНИМАНИЕ! Выходы автомата также следует определить, задав их тип.

5. Для этого нажмите кнопку **Advanced** (см. рис. П2.9). Появится окно дополнительных установок порта (рис. П2.10). Если состояние выхода в данном внутреннем состоянии автомата должно сохраниться при переходе в другое внутреннее состояние, то выход следует определить как **Registered.** Если состояние выхода при переходе автомата в другое внутреннее состояние изменяется, то выход определяется как **Combinatorial.** В нашем случае выходы следует определить как **Combinatorial,** поставив галочку в соответствующем поле (см. рис. П2.10). Нажмите кнопку **OK.**

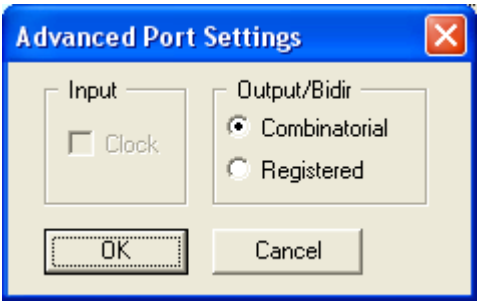


Рис. П2.10. Задание типа выхода

Примечание. Ошибочно созданные порты можно удалить, выделив их имя в списке портов и нажав кнопку **Delete.**

6. После задания последнего контакта, нажмите кнопку **Далее** (см. рис. П2.9). Появится окно редактора выбора количества автоматов на диаграмме (рис. П2.11).

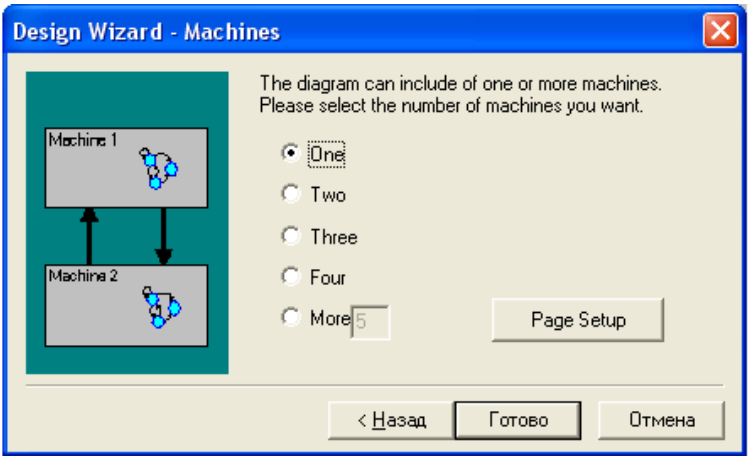


Рис. П2.11. Окно выбора количества автоматов

7. Поставьте галочку в поле **One** и нажмите кнопку **Готово.**

Появится окно редактора **State Editor** с описанием портов автомата (рис. П2.12).

Design Wizard сделал за нас всю работу по описанию портов создаваемого автомата. Декларации всех портов точно соответствуют данным, введенным в окне **Ports.**

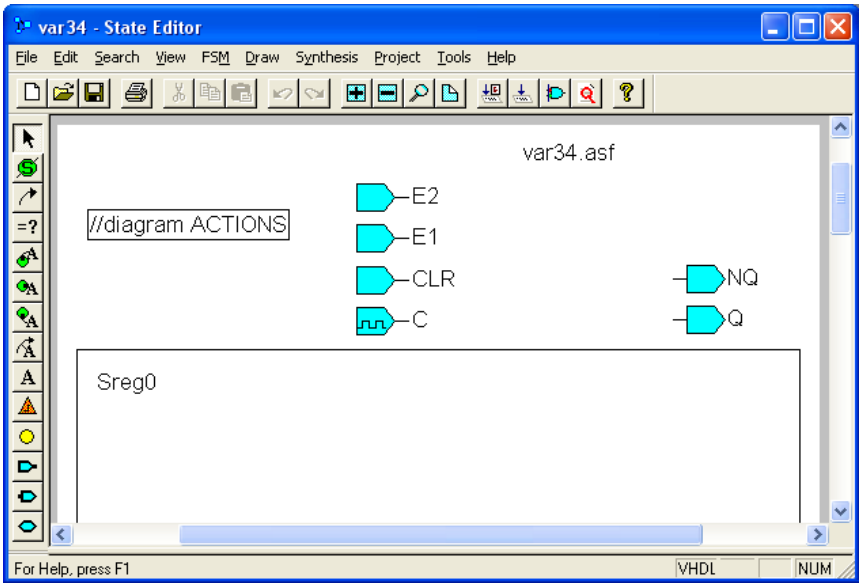


Рис. П2.12. Окно State Editor с описанием портов автомата

Окно редактора FSM Editor содержит вертикальную и горизонтальную панели инструментов, строку меню, поле редактирования диаграммы и строку сообщений.

Редактирование портов автомата

В редакторе **State Editor** возможно откорректировать все просчеты и упущения, сделанные на предыдущем этапе. Проиллюстрируем эту возможность на примере порта CLK. Допустим, на предыдущем этапе вы забыли определить его как синхронизирующий вход. Для исправления этого упущения необходимо выполнить следующие шаги.

1. Выбрать изображение CLK в редакторе **State Editor** и нажать правую кнопку мыши.

2. Выбрать опцию Properties из сокращенного меню (рис. П2.13). Появится окно реквизитов порта (Port Properties).

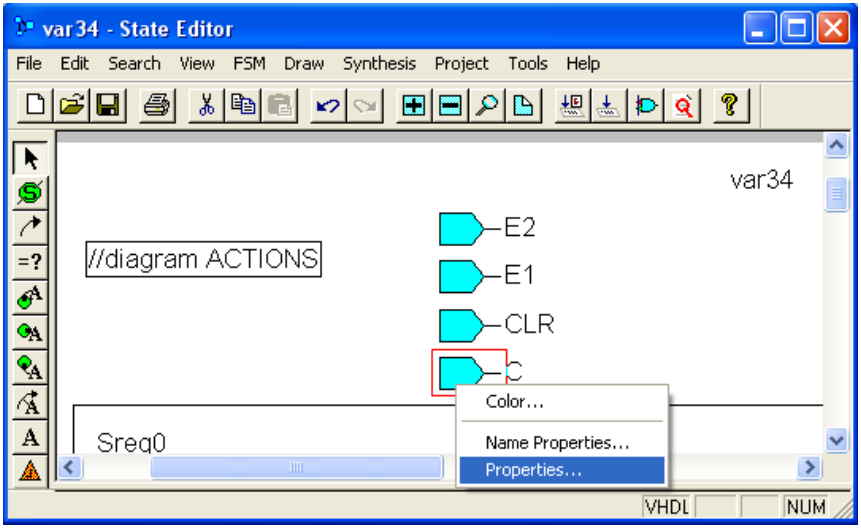


Рис. П2.13. Выбор опции Properties

3. Выбрать Clock, как показано на рис. П2.14, а затем нажать OK.

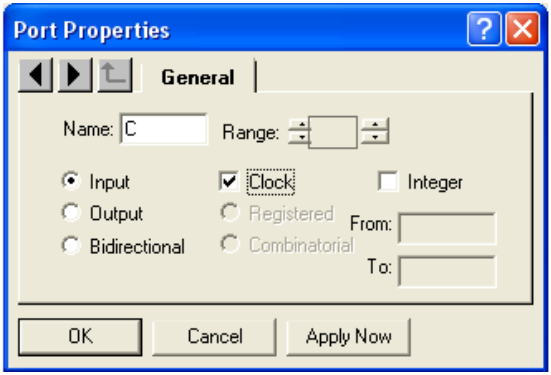


Рис. П2.14. Окно свойств порта

На рис. П2.15 показано как будет выглядеть порт CLK после этих действий.

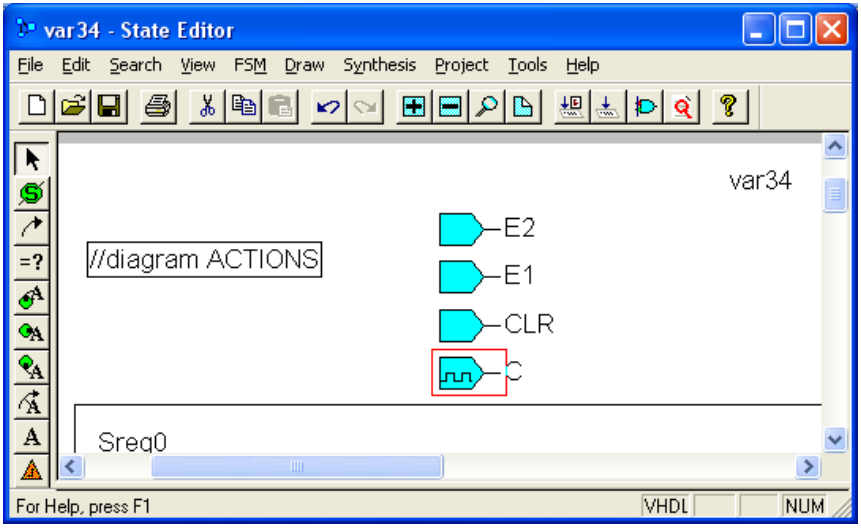


Рис. П2.15. Отредактированное значение порта CLK

Изменение свойств портов выполняется аналогично (рис. П2.16).

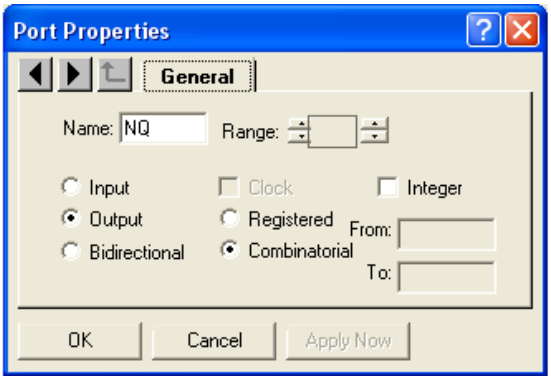


Рис. П2.16. Изменение свойств порта

Дополнительные порты ввода и вывода могут быть добавлены в редакторе FSM Editor в любое время, нажимая кнопки портов ввода

и вывода, размещенные внизу вертикальной инструментальной панели (см. рис. П2.12).

Ввод графа переходов конечного автомата

Задание состояний автомата

На графе переходов состояния автомата представляются вершинами.

1. Для задания очередного состояния автомата щёлкните на вертикальной панели инструментов по кнопке **State.**  Курсор примет форму круга с квадратом в центре.

2. Переместите курсор в поле прямоугольной рамки и, выбрав подходящее расположение, щёлкните мышью. Появится новое состояние – вершина графа (рис. П2.17).

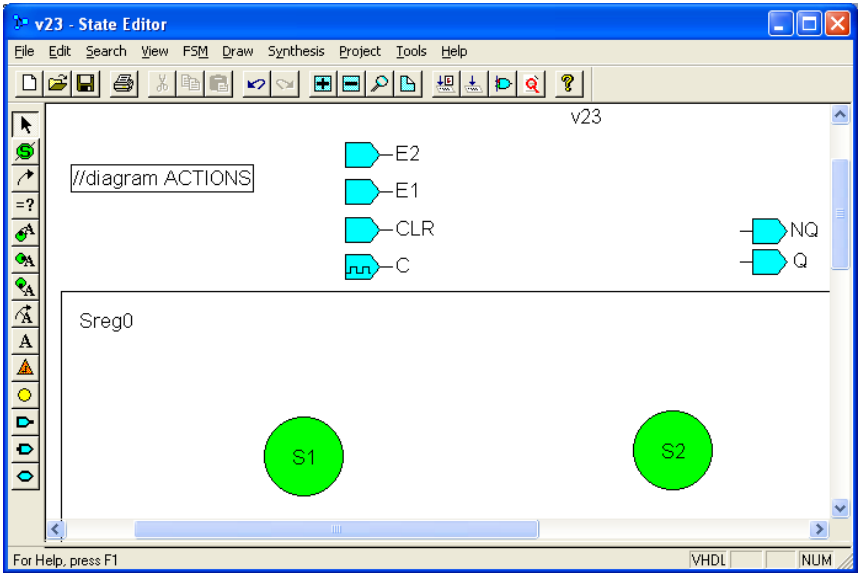


Рис. П2.17. Окно State Editor с заданными состояниями автомата

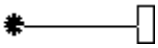
Нажимая левую кнопку мыши на изображении кружка и двигаясь в желательном направлении, можно перемещать кружок.

Задание операций для выходов автомата

Под операцией в FSM Editor подразумевается набор исполняемых выражений, которые определяют состояние выходов автомата. Операция определения значения выходов автомата Мура для рассматриваемого состояния выполняется, как только автомат перешёл в это состояния.

1. Для записи выражений, определяющих состояния выходов автомата, щелкните на вертикальной панели инструментов по кнопке **State Action.**

Курсор примет форму линии с небольшим кружком на конце.



2. Поместите этот кружок курсора на правую сторону кружка состояния (для автомата Мура) и щёлкните мышью. Появится поле для ввода текста (рис. П2.18).

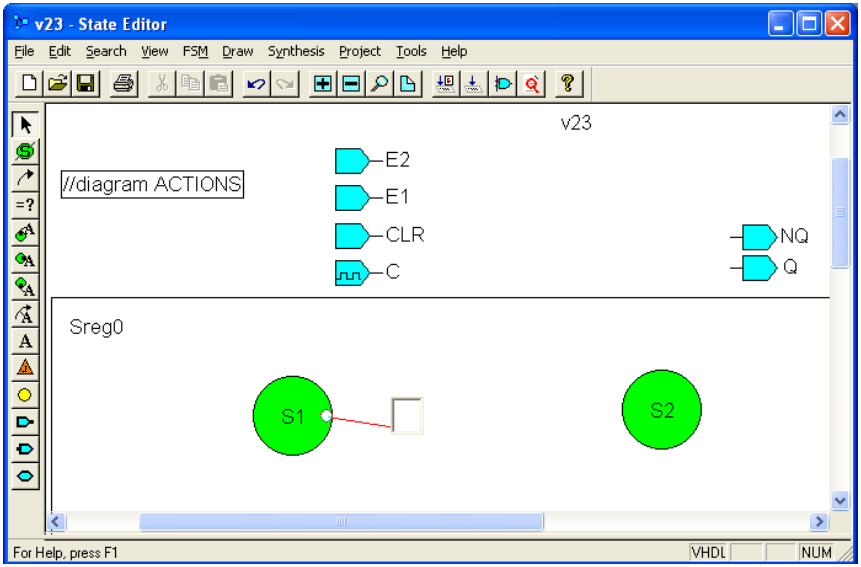


Рис. П2.18. Окно State Editor с полем для ввода текста

3. Далее вводят необходимые выражения для выходных портов автомата. В нашем случае присвоим состоянию S1 значения выходов Q = 0, NQ = 1, а состоянию S2 – Q = 1, NQ = 0 (рис. П2.19).

4. Для завершения ввода щелкните левой клавишей мыши на свободном поле.

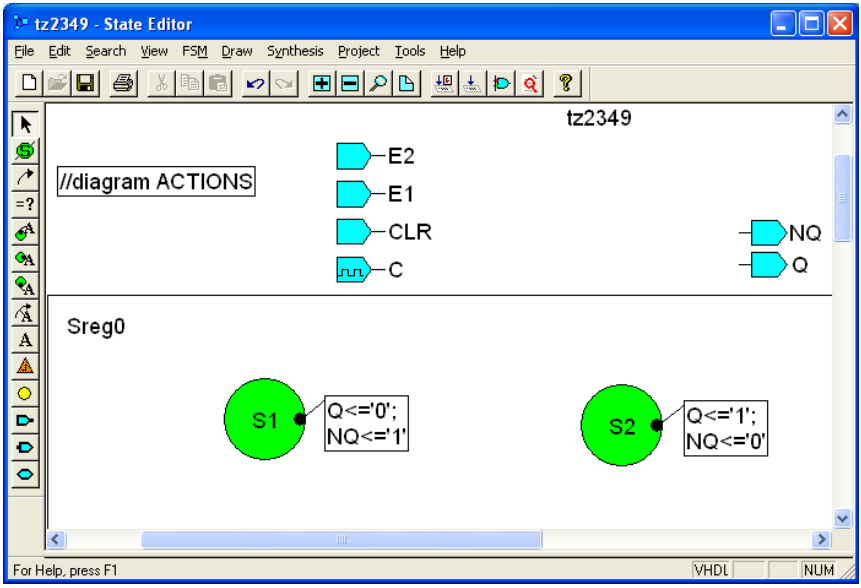


Рис. П2.19. Окно State Editor с присвоенными значениями выходов автомата

Правила записи выражений

Синтаксис выражений следующий:

- оператор присваивания обозначается составным символом **<=,**
- константы **заключаются в одинарные кавычки**, например, ‘0’,
- выражения для каждого выхода отделяются друг от друга символом **точка с запятой** «;»,
- в конце последнего выражения никакой символ не ставится (см. рис. П2.19).

В выражениях можно использовать логические операторы, определенные в табл. П2.2.

Таблица П2.2

Логические операторы

not	Инверсия
and	И
or	ИЛИ
nand	И-НЕ
nor	ИЛИ-НЕ
xor	ИСКЛЮЧАЮЩЕЕ ИЛИ
xnor	ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ

У оператора **not** самый высокий приоритет, так что можно не заключать в скобки подвыражение типа not A1.

Логические операторы **and, or, nand, nor, xor, xnor** имеют одинаковое старшинство и выполняются слева направо в выражениях. В сложных логических выражениях порядок выполнения операторов регулируется скобками.

Пробелы для удобства чтения выражений можно вставлять произвольно.

Задание переходов

Переход – это изменение состояния автомата под воздействием входного сигнала. Переход, как правило, осуществляется при выполнении некоторого условия, однако возможны и безусловные переходы.

Переходы конечного автомата из одного состояния в другое задаются ориентированными рёбрами (дугами) графа переходов. Иногда требуется, чтобы автомат остался в данном состоянии при соответствующем условии. В этом случае переход представляется ориентированным ребром, замкнутым на ту же вершину.

Для задания перехода выполните следующие действия.

1. Щёлкните на вертикальной панели инструментов по кнопке **Transition.** Курсор примет форму перекрестия.

2. Наведите перекрестие курсора на вершину, из которой будет осуществляться переход, и щёлкните мышью. Затем повторите то же самое с вершиной, в которую будет осуществляться переход. Появится кривая со стрелкой, привязанная к двум выбранным вершинам – ребро графа (рис. П2.20).

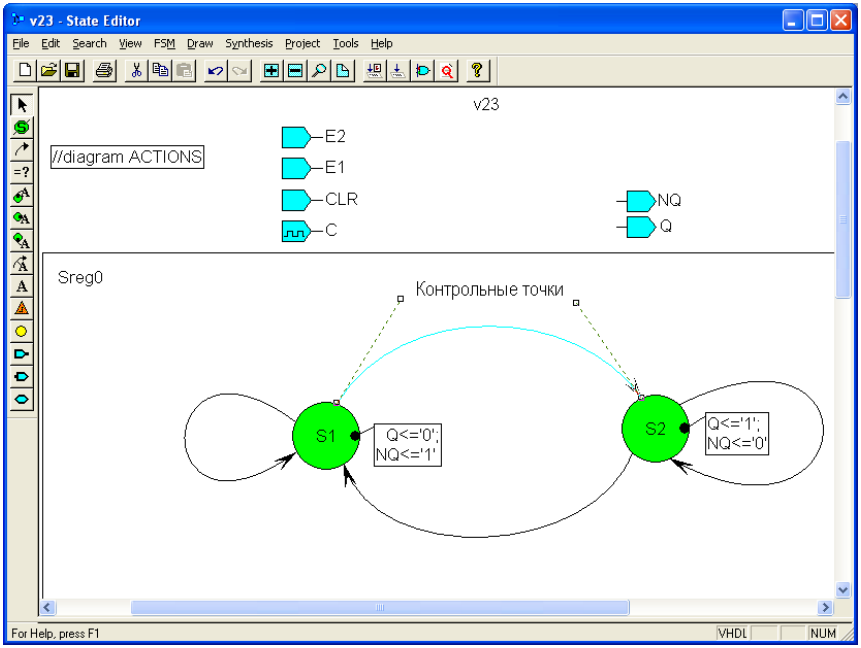


Рис. П2.20. Окно State Editor с заданными переходами автомата

Используйте контрольные точки, чтобы изменить форму кривой (см. рис. П2.20).

Задание условий переходов

Условие перехода автомата из одного состояния в другое задаётся равенством 1 или 0 логического выражения или переменной, которые соответствуют появлению тех или иных входных сигналов. Выражение заключается в круглые скобки, ставится знак равенства, затем пишется 0 или 1, **заключённые в одинарные кавычки.** Например:

(not E1 and E2) = ‘1’

В выражениях можно использовать логические операторы, определенные в табл. П2.2.

Если условие перехода определяется истинностью или ложностью переменной, то запись такого условия выглядит следующим образом:

E2 = ‘0’

Для задания условия перехода:

1. Щёлкните на вертикальной панели инструментов по кнопке **Condition.**

2. Наведите перекрестие на ребро перехода, для которого необходимо задать условие, и щёлкните мышью. Появится текстовое поле.

3. Введите условие перехода и щелкните левой клавишей мыши на свободном поле. Над ребром появится условие перехода (рис. П2.21).

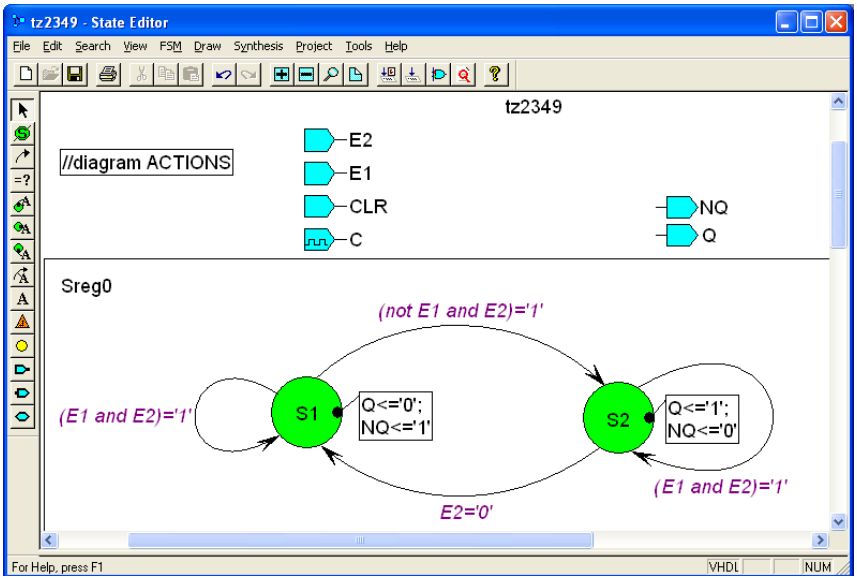


Рис. П2.21. Окно State Editor с заданными условиями переходов автомата

Чтобы изменить введённый код, выделите условие перехода, и дважды щёлкните на нём мышью.

Задание сброса автомата в начальное состояние

Как правило, на множестве состояний S автомата выделяется специальное начальное состояние $s_o \in S$. Переход автомата в начальное состояние реализуется по специальному входу. Этот вход имеет стандартное имя – clear (CLR) или reset. Перевод автомата в начальное состояние называют сбросом автомата. Подача определенного сигнала (0 или 1) на этот установочный вход называют условием сброса.

Переход в начальное состояние может осуществляться *синхронно* или *асинхронно*. При синхронной установке переход автомата в начальное состояние происходит по активному фронту синхросигнала при наличии разрешающего сигнала на установочном входе. При асинхронном сбросе автомат переходит в начальное состояние сразу при поступлении сигнала на установочный вход.

Для задания сброса автомата в начальное состояние выполните следующие действия.

1. Щёлкните на вертикальной панели инструментов по кнопке **Reset.** Курсор примет форму чёрного треугольника.

2. Наведите курсор на поле около состояния, которое необходимо объявить начальным, и щёлкните мышью. Появится символ сброса и курсор перейдёт в режим рисования перехода.

3. Подведите курсор к состоянию, которое необходимо объявить начальным, и щёлкните мышью. Появится ориентированное ребро от символа сброса к указанному состоянию (рис. П2.22). Это состояния и будет считаться начальным.

Теперь необходимо задать синхронность или асинхронность сброса.

4. Щёлкните правой кнопкой мыши на символе сброса и в появившемся меню установите галочку напротив Synchronous или Asynchronous (рис. П2.23).

Условное графическое обозначение синхронного и асинхронного сброса:

 – синхронный сброс,  – асинхронный сброс.

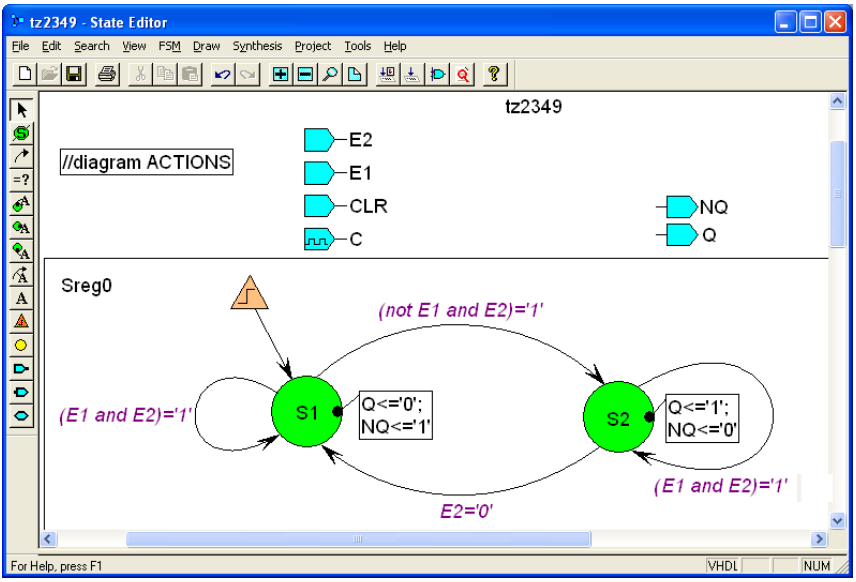


Рис. П2.22. Окно State Editor с изображением синхронного сброса автомата

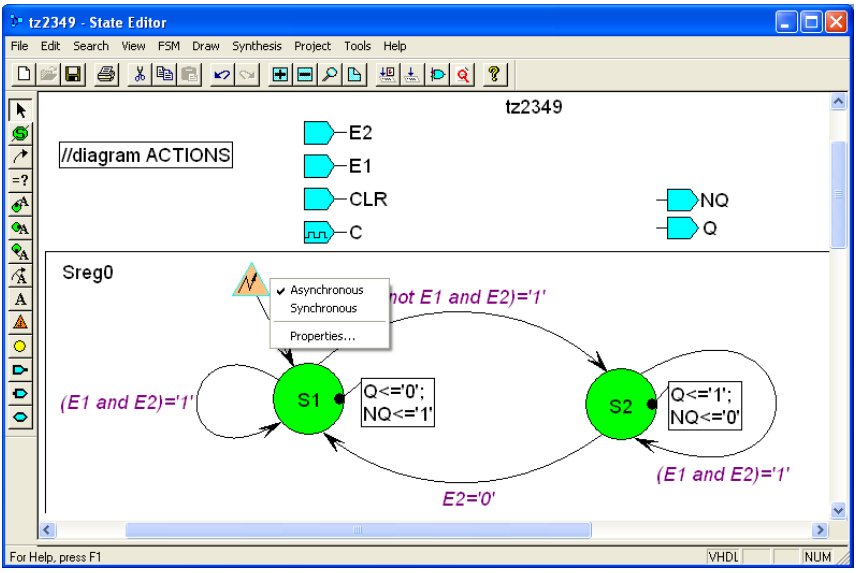


Рис. П2.23. Окно State Editor с выбором режима сброса автомата

Условие сброса автомата в начальное состояние задаётся точно так же, как и любое другое условие перехода.

Для задания условия сброса автомата в начальное состояние:

5. Щёлкните на вертикальной панели инструментов по кнопке **Condition.**

6. Наведите перекрестие на ребро перехода, для которого вы хотите задать условие, и щёлкните мышью. Появится текстовое поле.

7. Введите условие перехода и щелкните левой клавишей мыши на свободном поле. Над ребром появится условие сброса автомата (рис. П2.24).

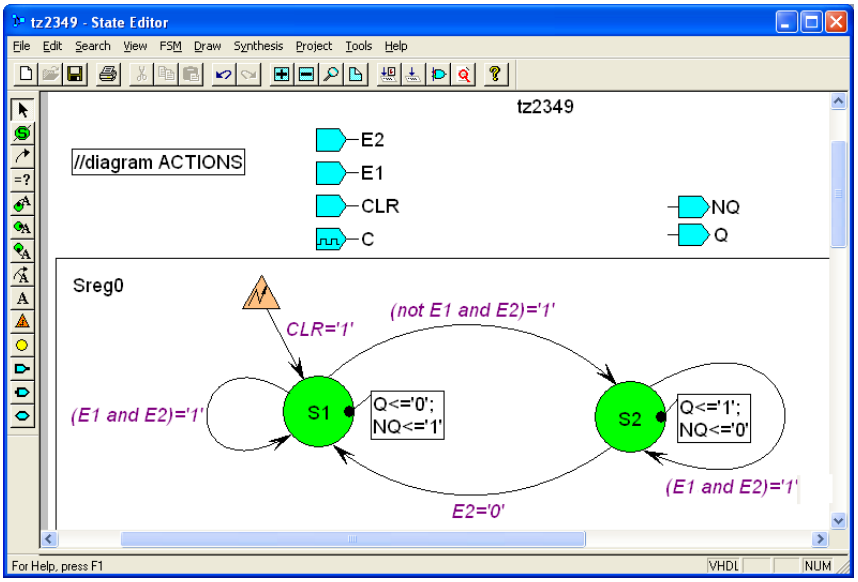


Рис. П2.24. Окно State Editor с заданным условием сброса автомата

Синтез автомата

Для того чтобы начать процесс синтеза, выберите пункт *Synthesize* из меню **Synthesis** в окне редактора State Editor (рис. П2.25).

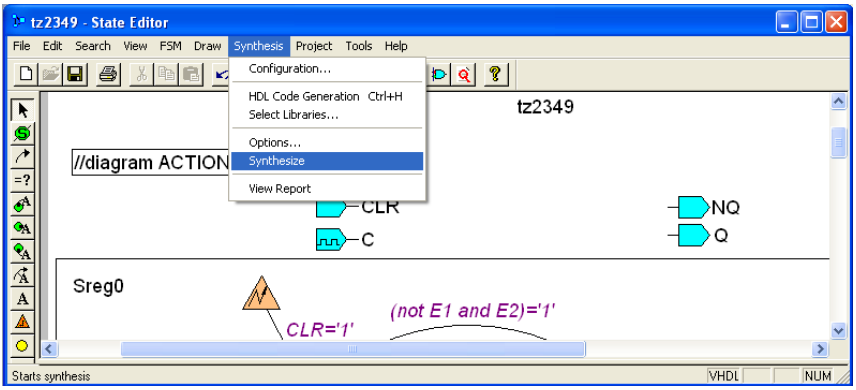


Рис. П2.25. Меню **Synthesis**

При отсутствии ошибок будет выдано сообщение Synthesis successful (рис. П2.26).

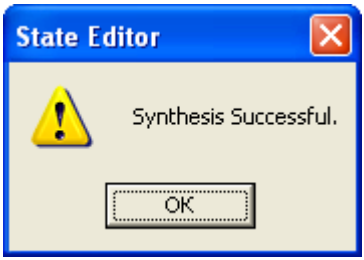


Рис. П2.26. Сообщение об успешном завершении процедуры синтеза

Если по какой-либо причине компиляция невозможна, то соответствующее сообщение об ошибках будет помещено в нижней части окна редактора State Editor, а место ошибки будет выделено на графе.

Создание макроэлемента

Создание макроэлемента необходимо для выполнения моделирования с целью отладки введенного графа переходов, а также для дальнейшего использования как элемента в составе другого устройства.

Для создания макроэлемента выберите *Create Macro* из меню **Project** в окне редактора State Editor (рис. П2.27). Успешное завершение процедуры сопровождается выводом соответствующего сообщения. Макроэлемент с тем же именем, что и созданный граф переходов, будет помещен в рабочую библиотеку проекта.

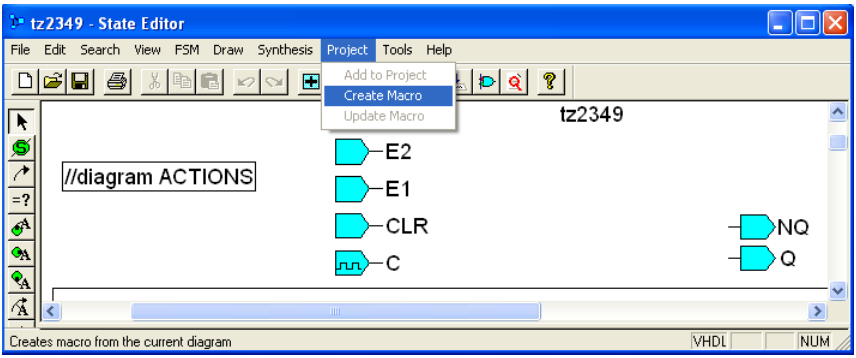


Рис. П2.27. Выбор команды Create Macro

После создания макроэлемента команда *Create Macro* автоматически заменяется командой *Update Macro.*

Моделирование макроэлемента автомата

Функциональное моделирование схемы изложено в приложении 1. Здесь рассмотрим только особенности, которые связаны с моделированием макроэлемента автомата.

Выбор макроэлемента в схемном редакторе

Откройте схемный редактор и для выбора макроэлемента нажмите кнопку «Выбор элемента» на вертикальной панели инструментов (см. приложение 1). Появится диалоговое окно SC Symbols выбора модели элемента. В этом окне выбрать созданный макроэлемент автомата (рис. П2.28).

Условное графическое обозначение извлеченного макроэлемента автомата показано на рис. П2.29.

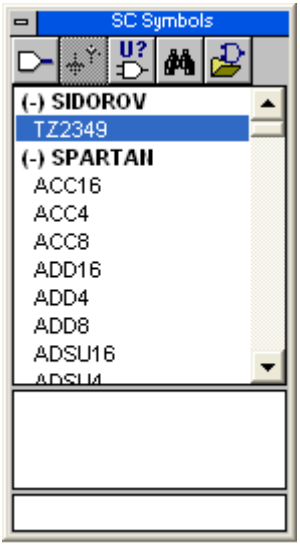


Рис. П2.28. Выбор макроэлемента

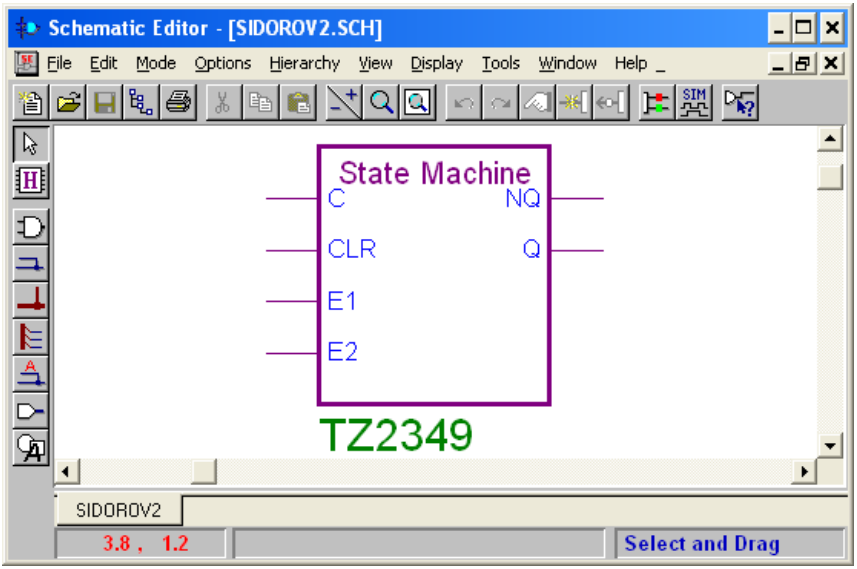


Рис. П2.29. Условное графическое обозначение макроэлемента автомата

Окно моделирования

Для проверки и отладки созданного макроэлемента в окне моделирования необходимо выполнить следующую последовательность действий.

1. Открыть окно моделирования (см. Приложение 1).
2. В окне **Logic Simulator** выполнить команду **Simulate Single Component** из меню **File** (рис. П2.30).

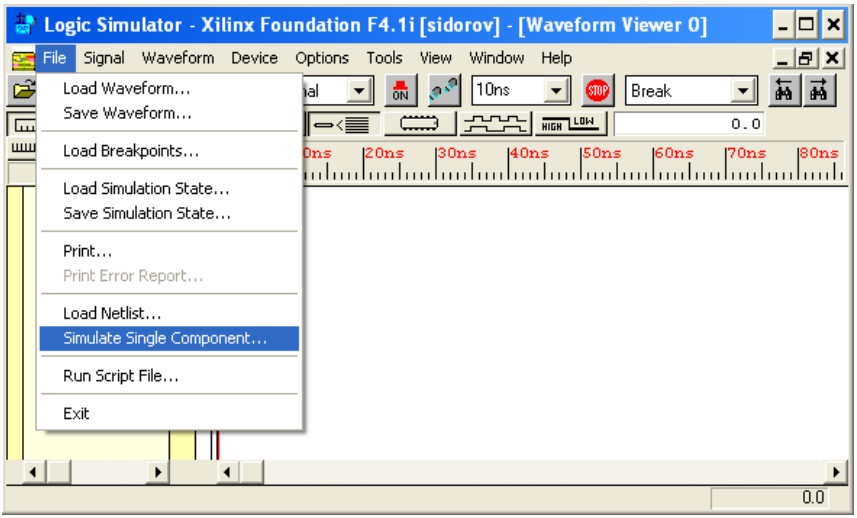


Рис. П2.30. Выбора команды **Simulate Single Component**

Появится окно **Simulate Single Component** выбора макроэлемента (рис. П2.31).

3. Выделите макроэлемент и нажмите кнопку **ОК**.

После выполнения указанных действий появляется возможность отслеживать изменения не только входов и выходов макроэлемента, но и внутренних сигналов.

4. Выполните команду **Add Signals** из меню **Signal** в окне **Logic Simulator** или нажмите пиктограмму **Select Component** на панели инструментов (см. приложение 1).

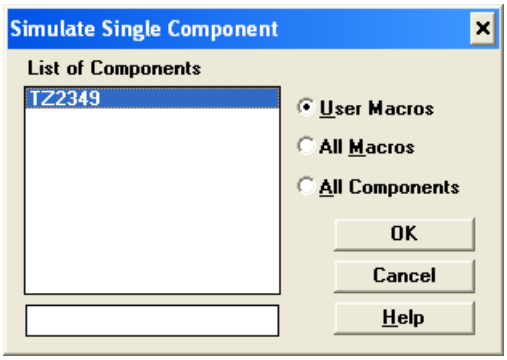


Рис. П2.31. Выбор макроэлемента

5. В появившемся списке сигналов **Signals selection** (рис. П2.32) выберите, щелкнув дважды по имени, входные и выходные сигналы макроэлемента для просмотра на временной диаграмме.

Сигналы, представляющие внутренние состояния S0 и S1 автомата, имеют стандартное имя Sreg00 и Sreg01 П2.32 (см. рис. П2.32).

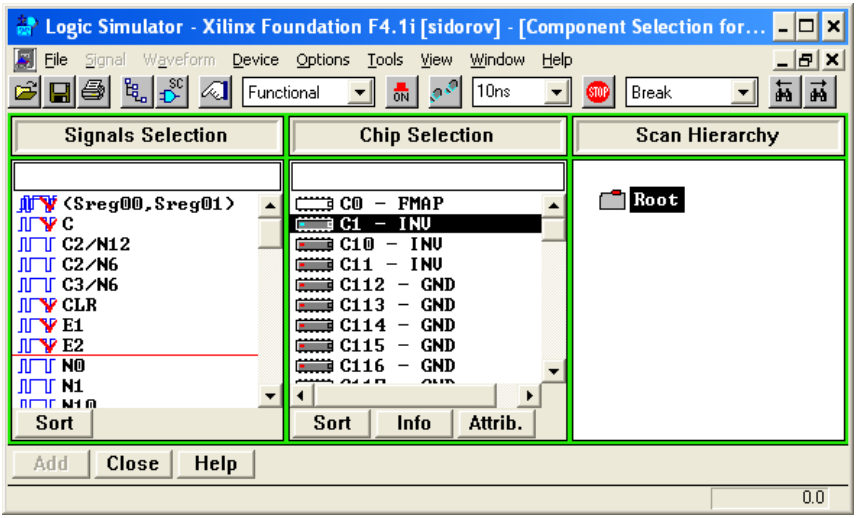


Рис. П2.32. Список доступных сигналов

Коррекция графа переходов макроэлемента

Коррекцию графа переходов созданного макроэлемента можно выполнить, не выходя из редактора схем **Schematic Editor.**

1. Нажмите пиктограмму **Hierarchy Push/Pop** на вертикальной панели инструментов редактора схем и затем дважды, щелкните мышью изображении макроэлемента.

Появится окно редактора **State Editor** с графом переходов автомата.

2. Выполните необходимые исправления. Затем выберите **Update Macro** из меню **Project** в окне редактора **State Editor** (рис. П2.33). Успешное завершение процедуры сопровождается выводом соответствующего сообщения.

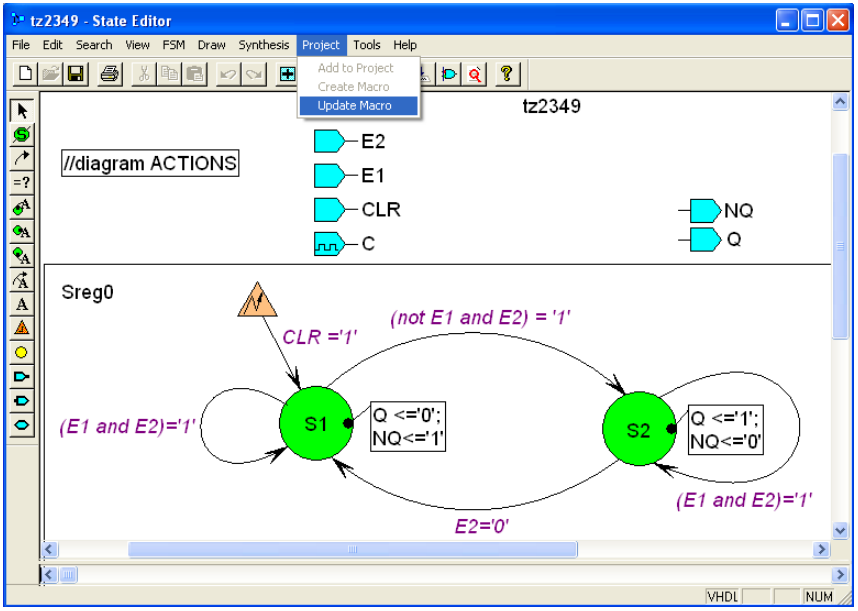


Рис. П2.33. Запрос на обновление макроэлемента

Приложение 3

РЕАЛИЗАЦИЯ ПРОЕКТА НА ПЛИС

После отработки макроэлемента автомата с использованием функционального моделирования выполняют загрузку макроэлемента автомата в ПЛИС и выполняют отработку её на стенде.

Подготовительные операции

Перед размещением макроэлемента на ПЛИС необходимо выполнить подготовительную работу. Для этого следует определить, какие точки схемы будут выводиться на внешние контакты корпуса ПЛИС. Контакты корпуса ПЛИС уже распаяны на печатной плате универсального лабораторного стенда и выведены на его органы управления (генераторы, клавишные регистры и индикаторы). Поэтому подготовка сводится к подключению органов управления стенда, которые в системе проектирования оформлены в виде макроэлементов. Таким образом, подготовка сводится к подключению данных макроэлементов к внешним контактам спроектированного макроэлемента автомата с целью дальнейшего размещения на ПЛИС.

Универсальный лабораторный стенд

В данном практикуме используется только часть оборудования универсального стенда, а именно: ПЛИС FPGA XCS10-3PC84, клавишные регистры, генераторы и индикация. Часть лицевой панели стенда с используемыми органами управления приведена на рис. П3.1.

На стенде для задания воздействий на схему имеются:

- два генератора одиночных импульсов – ГОИ1 и ГОИ2. Первый из них может работать в одном из двух режимов: либо в режиме одиночных импульсов, либо в режиме непрерывных импульсов. Выбор режима осуществляется крайним левым переключателем (см. рис. П3.1);
- три клавишных регистра, используемых для задания постоянных значений.

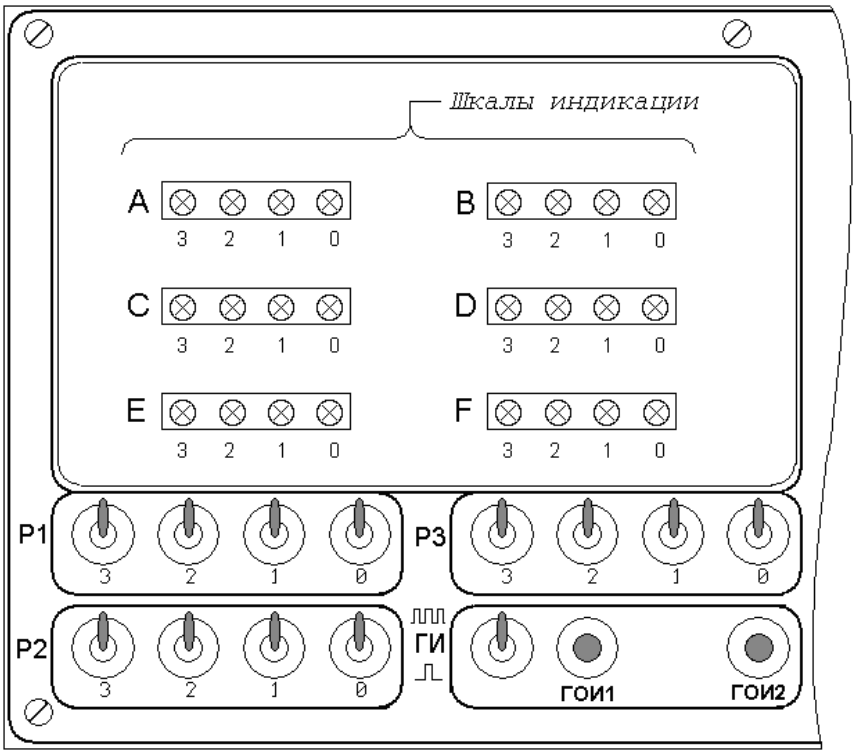


Рис. П3.1. Органы управления и индикации стенда

Для целей контроля состояния схемы в процессе её отладки на стенде имеются шесть четырехразрядных светодиодных шкал, которые можно подключать к произвольным точкам схемы (см. рис. П3.1). Таким образом, можно одновременно наблюдать состояние схемы в 24 точках.

Макроэлементы органов управления стенда

Перечисленные выше органы управления и индикации стенда в системе проектирования представлены в виде макроэлементов (рис. П3.2).

Два генератора импульсов ГОИ1 и ГОИ2 представлены одним символом (см. рис. П3.2,*а*), которому присвоено имя GI_GOI.

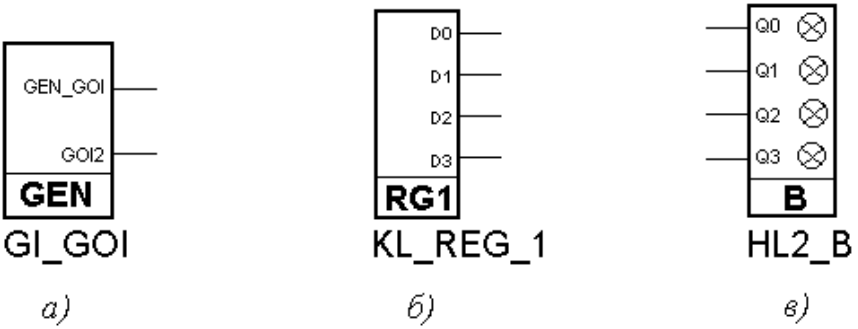


Рис. П3.2. Условные графические обозначения макроэлементов управления и индикации стенда

Клавишные регистры (см. рис. П3.2,*б*) имеют обозначение KL_REG_1, KL_REG_2 и KL_REG_3.

Пример условного графического обозначения четырехразрядных светодиодных шкал приведен на рис. П3.2,*в*). Макроэлементы светодиодных шкал имеют следующие имена: HL1_A, HL2_B, HL3_C, HL4_D, HL5_E, HL6_F.

Подключение макроэлементов стенда к проекту

В системе проектирования органы управления, оформленные в виде макроэлементов, содержатся в библиотеке проекта с именем «**maket**». Для использования макроэлементов необходимо библиотеку проекта «**maket**» подключить к своему проекту.

1. В окне **Project Manager** выполните команду меню *File/Project Libraries* (рис. П3.3).

2. В открывшемся окне **Project Libraries** (рис. П3.4) в его левой части выберите библиотеку того проекта, которую следует включить в свой проект. В нашем случае это библиотека проекта «**maket**» (см. рис. П3.4). После чего в окне **Project Libraries** нажмите кнопку **Add >>** (см. рис. П3.4).

3. В окне **Schematic Editor** нажмите кнопку «Выбор элемента».

В появившемся окне **SC Symbols** извлеките требуемые макроэлементы органов управления стенда (рис. П3.5).

На рис. П3.6 приведен пример подключения макроэлементов стенда к макроэлементу спроектированного автомата.

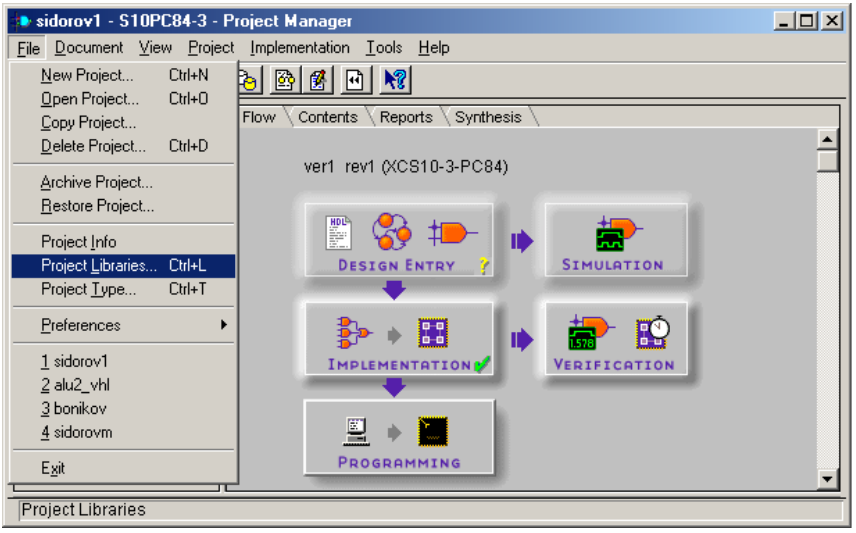


Рис. П3.3. Выбор окна Project Libraries

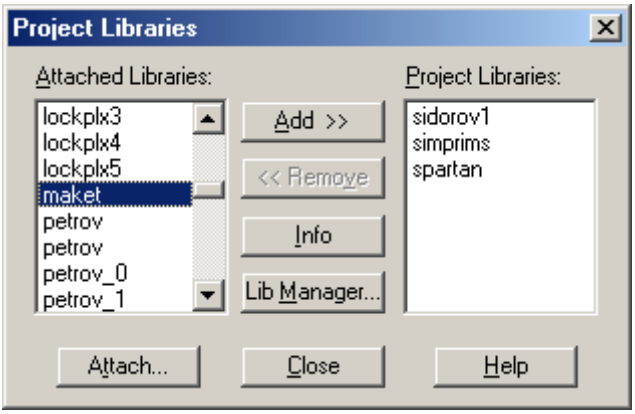


Рис. П3.4. Выбор библиотеки проекта «maket»

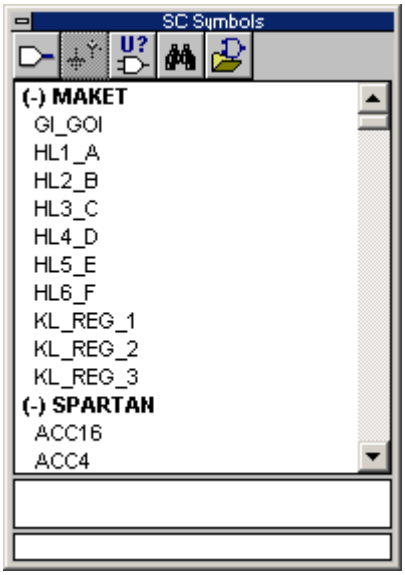


Рис. П3.5. Выбор макроэлементов

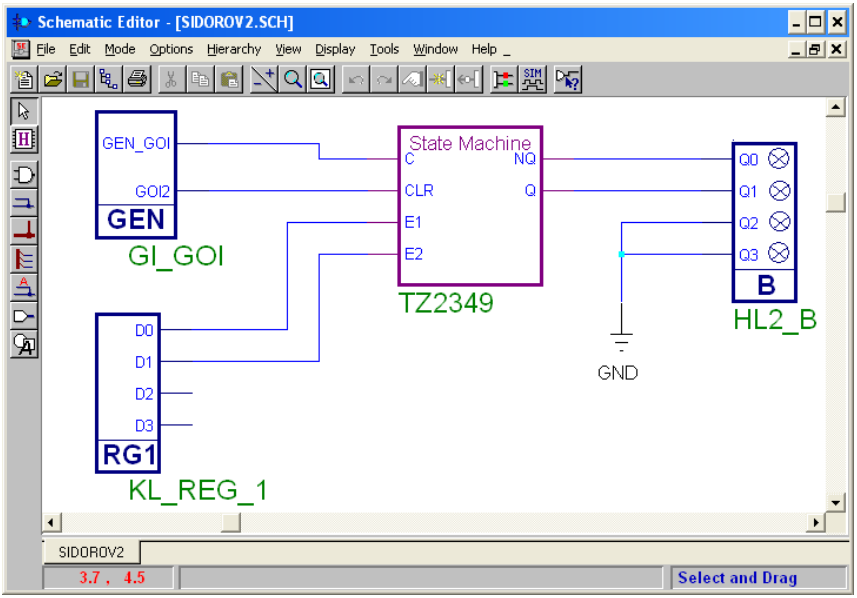
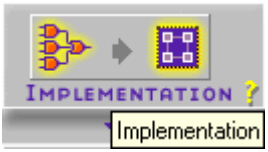


Рис. П3.6. Подключение органов управления стенда

Размещение схемы на кристалле

После проведенной подготовки можно приступать непосредственно к этапу размещения схемы на кристалле.

1. В окне **Project Manager** нажмите кнопку **Implementation.**
Система предложит скорректировать EDIF Netlist проекта (рис. П3.7).



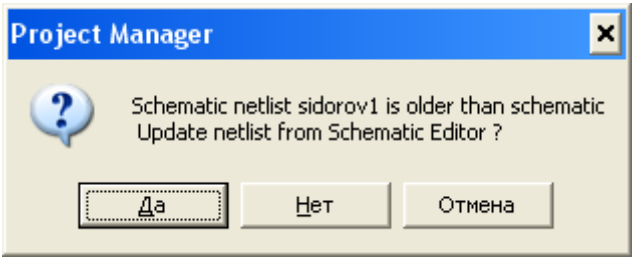


Рис. П3.7. Запрос системы

2. Нажмите кнопку «Да». После работы соответствующей программы появится окно **Implement Design** (рис. П3.8).

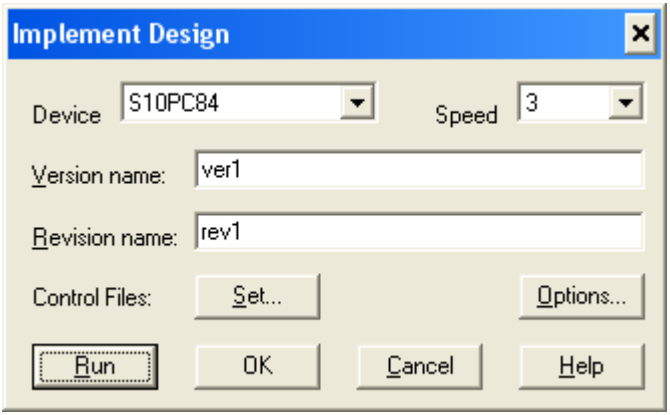


Рис. П3.8. Окно Implement Design

3. В окне Implement Design нажмите кнопку **Run**.

Система перейдет к автоматическому размещению схемы на кристалле. Этот процесс будет отображаться в окне **Flow Engine** (рис. П3.9).

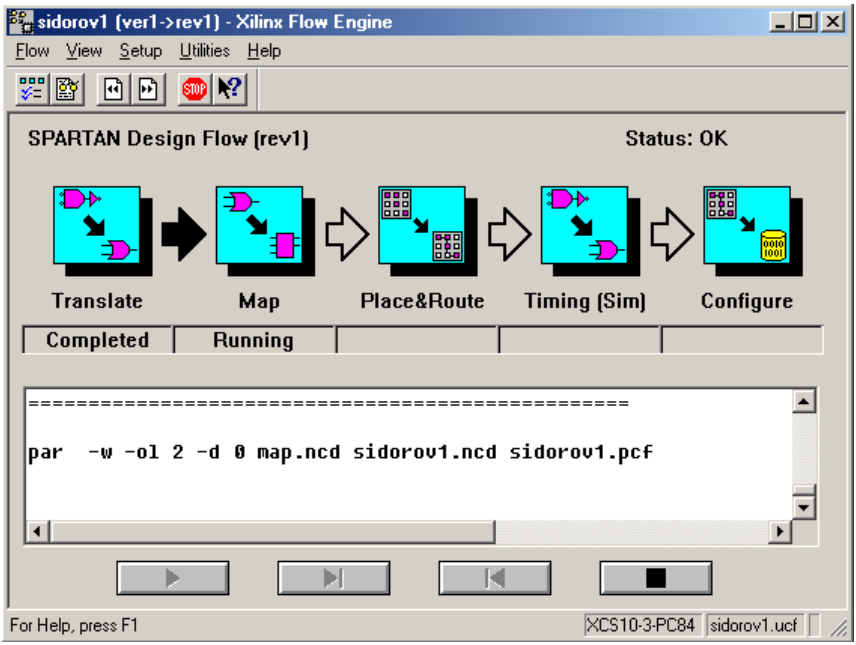


Рис. П3.9. Окно Flow Engine

При успешном завершении окно Flow Engine исчезнет и появится окно (рис. П3.10), в котором будет сообщение об успешном завершении размещения (в случае ошибки – сообщение об ошибке).

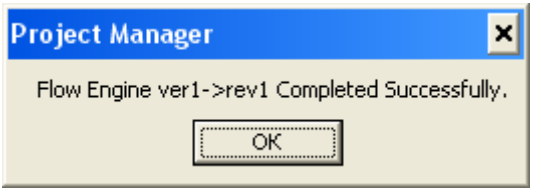


Рис. П3.10. Сообщение об успешном завершении размещения

Загрузка проекта в ПЛИС

1. В окне **Project Manager** нажмите кнопку **Programming.** Появится окно *Select Program* (рис. П3.11).

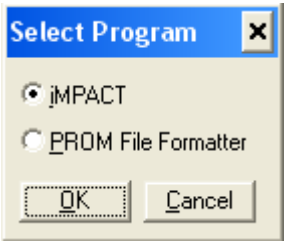


Рис. П3.11. Окно Select Program

2. Выберите **iMPACT** и нажмите кнопку **OK**. Появится окно *iMPACT* (рис. П3.12).

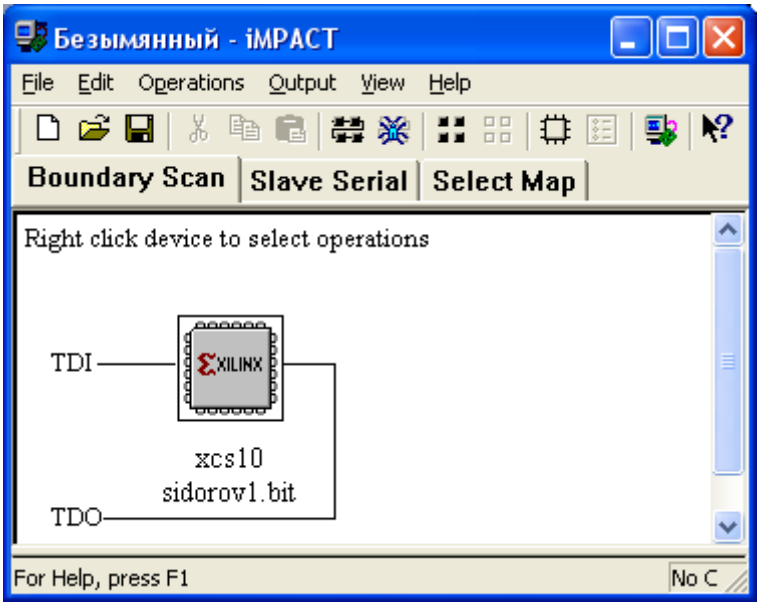


Рис. П3.12. Окно iMPACT

3. В окне *iMPACT* нажмите кнопку *Slave Serial* (см. рис. П3.12). После чего в данном окне появится сообщение «Right click to Add Device» (рис. П3.13).

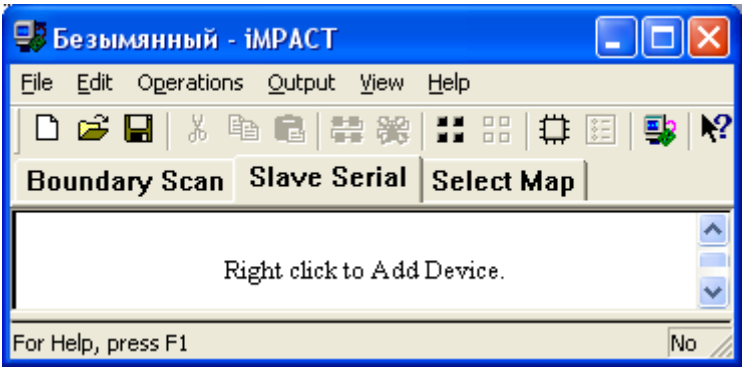


Рис. П3.13. Окно с сообщением

4. Сделайте щелчок правой клавишей мыши в данном окне и в появившемся меню выберите Add Xilinx Device (рис. П3.14).

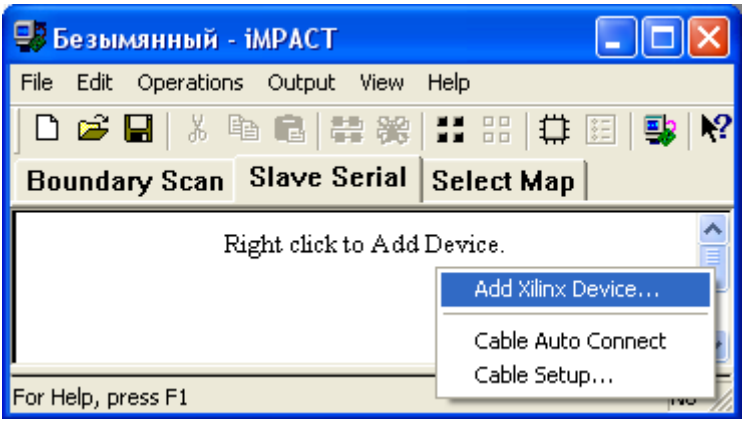


Рис. П3.14. Выбор Add Xilinx Device

5. В появившемся окне *Add Device* (рис. П3.15) выделите имя своего проекта с расширением **.bit** и щелкните кнопку Открыть.

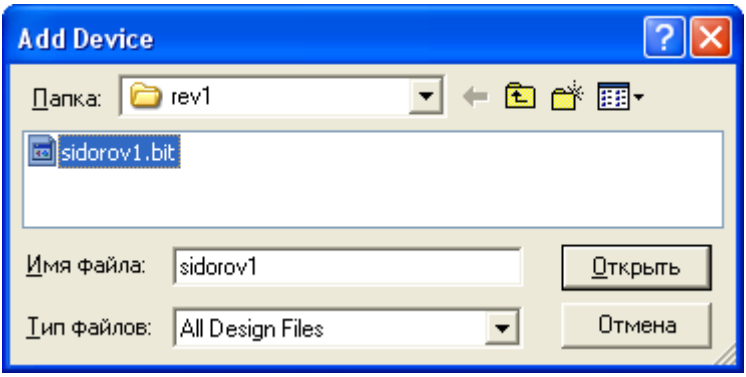


Рис. П3.15. Выбор Add Device

Вновь появится окно *iMPACT* (рис. П3.16) с подсказкой вверху «Right click device to select operations».

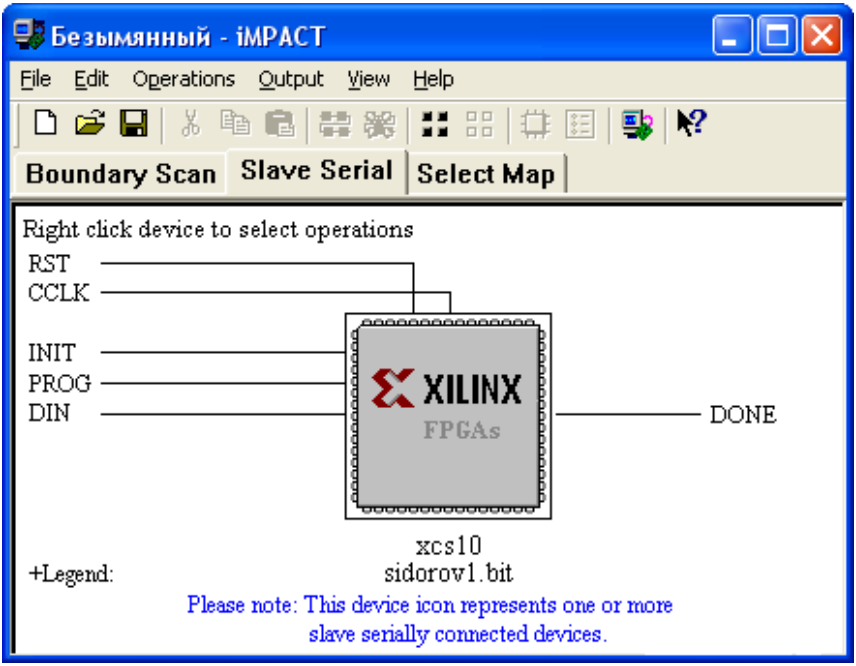


Рис. П3.16. Окно iMPACT с подсказкой

6. Включите питание стенда.

7. Поставьте курсор мыши на изображение ПЛИС и щелкните правую клавишу мыши. В появившемся меню выберите *Program* (рис. П3.17) и щелкните левую клавишу мыши. После чего последует загрузка ПЛИС.

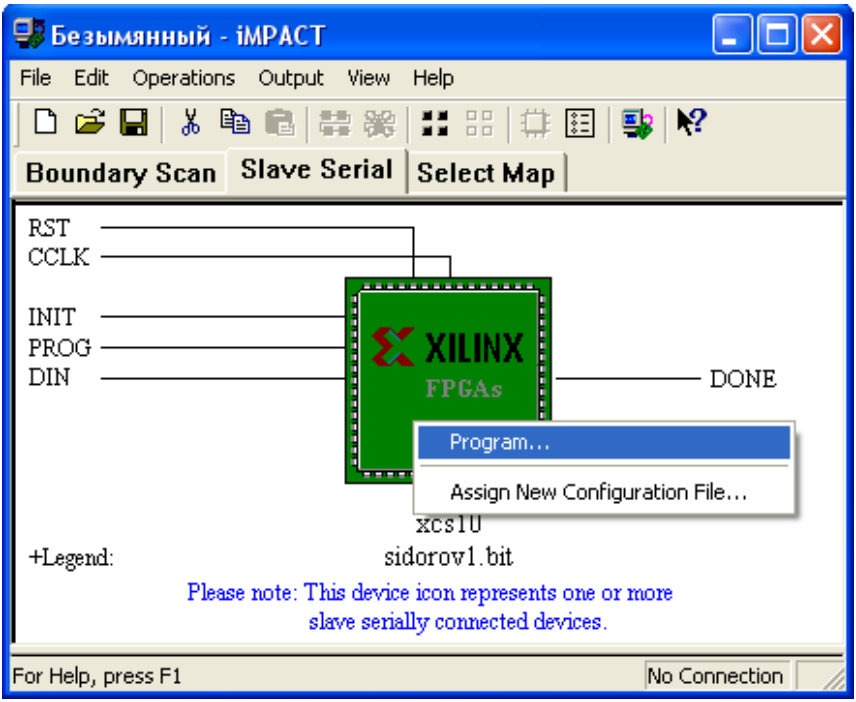


Рис. П3.17. Окно iMPACT с меню

После загрузки проекта в ПЛИС можно приступать к отладке на стенде.

Приложение 4

МИНИМИЗАЦИЯ СОСТОЯНИЙ ДЕТЕРМИНИРОВАННОГО КОНЕЧНОГО АВТОМАТА

Часто при использовании алгоритмов формального синтеза детерминированных конечных автоматов получаются автоматы с избыточным количеством состояний. С подобной ситуацией мы столкнулись в лабораторной работе №6 (см. рис. 6.10 и табл. 6.15). В теории автоматов доказывается, что для любого автомата, соответствующего регулярному выражению, существует единственный детерминированный конечный автомат с минимальным количеством состояний с точностью до имен состояний (два автомата одинаковы с точностью до имен состояний, если один из них может быть получен из другого простым переименованием состояний).

Существует несколько формальных алгоритмов, которые позволяют получить из произвольного детерминированного конечного автомата детерминированный конечный автомат с минимальным числом состояний. Рассмотрим один из них, применительно к автоматам, генерирующим некоторые управляющие воздействия (см. лабораторную работу №4) или признаки соответствия состояния системы определенным критериям (см. лабораторные работы №5 и №6).

Алгоритм состоит из трех основных шагов: построение начального разбиения состояний на группы, итерационного разбиения подгрупп на группы, замены групп состояний одним представителем и составление таблицы переходов для минимального автомата. Рассмотрим данные шаги более подробно.

Идея шага построения начального разбиения на группы состоит в следующем: необходимо построить группы состояний автомата, генерирующих одни и те же выходные сигналы. При этом если управляющие воздействия генерируются не только от состояния автомата, но и при переходах из одного состояния в другое (см. лабораторную работу №5), то состояния с такими переходами должны быть включены в разные группы, если по соответствующим входным воздействиям для состояний по переходам генерируются отличные сигналы. Результатом данного этапа будет начальное разбиение $\Pi_0 = \{G_0, G_1, .., G_n\}$.

Идея следующего шага состоит в итерационном анализе всех групп на соответствие переходов из состояний, входящих в данную группу, т.е. для состояний группы должно выполняться условие: $\forall s,t \in G_i \Leftrightarrow \forall a_j \in \Sigma : f(s,a_j) = u_{s,j}, f(t,a_j) = u_{t,j} : u_{s,j}, u_{t,j} \in G_k$ (в случае анализа бинарных сигналов Σ = {0, 1}). Если для каких-то состояний *s* и *t* данное условие не выполняется, то данная группа разбивается на подгруппы, в которые помещаются только состояния с одинаковыми переходами, т.е. если переходы из состояний различаются, то они помещаются в разные подгруппы. Фактически, мы получаем новое разбиение Π'. Данный шаг повторяется до тех пор, пока после анализа всех групп не будет получено разбиение, совпадающее с предыдущим ($\Pi_k = \Pi_{k-1}$, где *k* – текущая итерация шага 2). Более формально шаг 2 представлен в алгоритме П4.1

Алгоритм П4.1

Построение минимально разбиения групп состояний ДКА

$\Pi' := \Pi_0$

выполнить {

$\Pi := \Pi'$

$\Pi' := \varnothing$

для каждого G из Π *выполнить{*

для каждого a из Σ *выполнить*

{

разбить G на $\Pi_{G,a} = \{G^a_1, G^a_2, \ldots, G^a_n\}$, *где* $G^a_1, G^a_2, \ldots, G^a_n$ *имеют отличные переходы по a относительно* Π

}

по $\Pi_{G,a}$ *построить* $\Pi_G = \bigcup \bigcap_{\Sigma} \Pi_{G,a}$

$\Pi' += \Pi_G$

}

} до тех пор пока $\Pi \neq \Pi'$

П – искомое разбиение

Отметим, что для шага 2 существует множество алгоритмов кроме приведенного выше, в том числе, и имеющих более эффективную машинную реализацию, например, алгоритм Хопкрофта.

На последнем шаге производится замена каждой группы из результирующего разбиения на ее представителя, синтезируются соответствующие таблицы переходов и генерации сигналов. Стартовым состоянием назначается представитель группы, в которой содержится стартовое состояние исходного автомата.

Пример

Постановка задачи. Дан детерминированный конечный автомат (соответствующий РВ 101((0|1)(0|1)(0|1))+101), представленный на рис. 6.10). По состоянию автомата *M* вырабатываются сигналы *GOOD* = ‘1’ и *BAD* = ‘0’, по состоянию Ø – сигналы *GOOD* = ‘0’ и *BAD* = ‘1’, во всех остальных состояниях *GOOD* = ‘0’ и *BAD* = ‘0’.

Синтез минимального автомата. Шаг 1

По условию у нас существует три различных комбинации выходных сигналов, которые формируются в зависимости от состояния, в котором находится автомат. Тогда мы получаем начальное разбиение состояний, представленное в табл. П4.1.

Таблица П4.1

Начальное разбиение на группы состояний автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101

Группа	Состояния	Комбинация сигналов
G1	M	GOOD = ‘1’, BAD = ‘0’
G2	Ø	GOOD = ‘0’, BAD = ‘1’
G3	A, B, C, D, E, F, G, H, I, J, K, L	GOOD = ‘0’, BAD = ‘0’

Синтез минимального автомата. Шаг 2

Отметим, что группы, содержащие по одному состоянию, не нуждаются в анализе на втором шаге, для них лишь необходимо своевременно изменять таблицу переходов в модифицируемые на данном шаге группы.

1. Построим соответствие переходов для текущего разбиения на группы. Результат построения приведен в табл. П4.2.

Таблица П4.2

Итерация 1. Соответствия переходов для разбиения на группы состояний автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101

Группа	Сигнал	Состояния	Переход
G1	0	M	G3
	1	M	G3
G2	0	Ø	G2
	1	Ø	G2
G3	0	A, C	G2
		B, D, E, F, G, H, I, J, K, L	G3
	1	L	G1
		B	G2
		A, C, D, E, F, G, H, I, J, K	G3

2. Очевидно, что группу G3 необходимо разбить на несколько подгрупп. Так как входные сигналы ‘0’ и ‘1’ дают разные разбиения Π_0 и Π_1, то результирующее разбиение будет содержать объединение пересечений групп из данных разбиений $\bigcup_{\Pi_0,\Pi_1} G_i^0 \cap G_j^1$. Результирующее разбиение после первой итерации приведено в табл. П4.3.

Таблица П4.3

Разбиение на группы состояний автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101, после первой итерации

Группа	Состояния	Комбинация сигналов
G1	M	GOOD = ‘1’, BAD = ‘0’
G2	Ø	GOOD = ‘0’, BAD = ‘1’
G3	A, C	GOOD = ‘0’, BAD = ‘0’
G4	B	GOOD = ‘0’, BAD = ‘0’
G5	L	GOOD = ‘0’, BAD = ‘0’
G6	D, E, F, G, H, I, J, K	GOOD = ‘0’, BAD = ‘0’

3. Построим соответствие переходов для текущего разбиения на группы. Результат построения приведен в табл. П4.4.

Таблица П4.4

Итерация 2. Соответствия переходов для разбиения на группы состояний автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101

Группа	Сигнал	Состояния	Переход
G1	0	M	G6
	1	M	G6
G2	0	Ø	G2
	1	Ø	G2
G3	0	A, C	G2
	1	A	G4
		C	G6
G4	0	B	G3
	1	B	G2
G5	0	L	G6
	1	L	G1
G6	0	D, E, F, G, H, I, J	G6
		K	G5
	1	D, E, F, G, H, I, J, K	G6

4. Очевидно, что группы G3 и G6 необходимо разбить на несколько подгрупп. Построим объединение пересечений групп из соответствующих разбиений для этих групп. Результирующее разбиение после второй итерации приведено в табл. П4.5.

Таблица П4.5

Разбиение на группы состояний автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101, после второй итерации

Группа	Состояния	Комбинация сигналов
G1	M	GOOD = ‘1’, BAD = ‘0’
G2	Ø	GOOD = ‘0’, BAD = ‘1’
G3	A	GOOD = ‘0’, BAD = ‘0’
G4	B	GOOD = ‘0’, BAD = ‘0’
G5	L	GOOD = ‘0’, BAD = ‘0’
G6	K	GOOD = ‘0’, BAD = ‘0’
G7	C	GOOD = ‘0’, BAD = ‘0’
G8	D, E, F, G, H, I, J	GOOD = ‘0’, BAD = ‘0’

5. Построим соответствие переходов для текущего разбиения на группы. Результат построения приведен в табл. П4.6.

Таблица П4.6

Итерация 3. Соответствия переходов для разбиения на группы состояний автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101

Группа	Сигнал	Состояния	Переход
G1	0	M	G8
	1	M	G6
G2	0	Ø	G2
	1	Ø	G2
G3	0	A	G2
	1	A	G4
G4	0	B	G7
	1	B	G2
G5	0	L	G8
	1	L	G1
G6	0	K	G5
	1	K	G8
G7	0	C	G2
	1	C	G8
G8	0	D, E, F, G, H, I, J	G8
	1	I, J	G6
		D, E, F, G, H	G8

Таблица П4.7

Разбиение на группы состояний автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101, после третьей итерации

Группа	Состояния	Комбинация сигналов
G1	M	GOOD = ‘1’, BAD = ‘0’
G2	Ø	GOOD = ‘0’, BAD = ‘1’
G3	A	GOOD = ‘0’, BAD = ‘0’
G4	B	GOOD = ‘0’, BAD = ‘0’
G5	L	GOOD = ‘0’, BAD = ‘0’
G6	K	GOOD = ‘0’, BAD = ‘0’
G7	C	GOOD = ‘0’, BAD = ‘0’
G8	I, J	GOOD = ‘0’, BAD = ‘0’
G9	D, E, F, G, H	GOOD = ‘0’, BAD = ‘0’

6. Из табл. П3.6 видно, что группу G8 необходимо разбить на подгруппы. Результирующее разбиение после третьей итерации приведено в табл. П4.7.

7. Построим очередное соответствие переходов для текущего разбиения на группы. Результат построения приведен в табл. П4.8.

Таблица П4.8

Итерация 4. Соответствия переходов для разбиения на группы состояний автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101

Группа	Сигнал	Состояния	Переход
G1	0	M	G9
	1	M	G6
G2	0	Ø	G2
	1	Ø	G2
G3	0	A	G2
	1	A	G4
G4	0	B	G7
	1	B	G2
G5	0	L	G8
	1	L	G1
G6	0	K	G5
	1	K	G9
G7	0	C	G2
	1	C	G9
G8	0	I, J	G9
	1	I, J	G6
G9	0	G, H	G8
		D, E, F	G9
	1	G, H	G8
		D, E, F	G9

8. Из табл. П4.8 видно, что группу G9 необходимо разбить на несколько подгрупп. При этом разбиения для всех входных сигналов совпадают, поэтому можно не строить объедение пересечений подгрупп разбиений, а сформировать новое разбиение сразу. Результирующее разбиение после четвертой итерации приведено в табл. П4.9.

Таблица П4.9

Разбиение на группы состояний автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101, после четвертой итерации

Группа	Состояния	Комбинация сигналов
G1	M	GOOD = ‘1’, BAD = ‘0’
G2	Ø	GOOD = ‘0’, BAD = ‘1’
G3	A	GOOD = ‘0’, BAD = ‘0’
G4	B	GOOD = ‘0’, BAD = ‘0’
G5	L	GOOD = ‘0’, BAD = ‘0’
G6	K	GOOD = ‘0’, BAD = ‘0’
G7	C	GOOD = ‘0’, BAD = ‘0’
G8	I, J	GOOD = ‘0’, BAD = ‘0’
G9	G, H	GOOD = ‘0’, BAD = ‘0’
G10	D, E, F	GOOD = ‘0’, BAD = ‘0’

9. Продолжим построение соответствий переходов для текущего разбиения на группы. Результат построения на данной итерации приведен в табл. П4.11.

10.Очевидно, что на данном шаге группу G10 необходимо разбить на несколько подгрупп. Результирующее разбиение после четвертой итерации приведено в табл. П4.10.

Таблица П4.10

Разбиение на группы состояний автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101, после пятой итерации

Группа	Состояния	Комбинация сигналов
G1	M	GOOD = ‘1’, BAD = ‘0’
G2	Ø	GOOD = ‘0’, BAD = ‘1’
G3	A	GOOD = ‘0’, BAD = ‘0’
G4	B	GOOD = ‘0’, BAD = ‘0’
G5	L	GOOD = ‘0’, BAD = ‘0’
G6	K	GOOD = ‘0’, BAD = ‘0’
G7	C	GOOD = ‘0’, BAD = ‘0’
G8	I, J	GOOD = ‘0’, BAD = ‘0’
G9	G, H	GOOD = ‘0’, BAD = ‘0’
G10	E, F	GOOD = ‘0’, BAD = ‘0’
G11	D	GOOD = ‘0’, BAD = ‘0’

Таблица П4.11

Итерация 5. Соответствия переходов для разбиения на группы состояний автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101

Группа	Сигнал	Состояния	Переход
G1	0	M	G10
	1	M	G6
G2	0	Ø	G2
	1	Ø	G2
G3	0	A	G2
	1	A	G4
G4	0	B	G7
	1	B	G2
G5	0	L	G8
	1	L	G1
G6	0	K	G5
	1	K	G9
G7	0	C	G2
	1	C	G10
G8	0	I, J	G10
	1	I, J	G6
G9	0	G, H	G8
	1	G, H	G8
G10	0	E, F	G9
		D	G10
	1	E, F	G9
		D	G10

11.Построим соответствие переходов для разбиения на группы после пятой итерации. Результат построения на данной итерации приведен в табл. П4.12. Отметим, что на данном этапе мы не получили несоответствия переходов ни для одной группу из текущего разбиения. Следовательно, на прошлой итерации было получено окончательное разбиение, соответствующее минимальному количеству состояний в детерминированном конечном автомате.

12.Получено минимальное разбиение автомата на группы состояний. Алгоритм шага 2 завершен.

Таблица П4.12

Итерация 6. Соответствия переходов для разбиения на группы состояний автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101

Группа	Сигнал	Состояния	Переход
G1	0	M	G10
	1	M	G6
G2	0	Ø	G2
	1	Ø	G2
G3	0	A	G2
	1	A	G4
G4	0	B	G7
	1	B	G2
G5	0	L	G8
	1	L	G1
G6	0	K	G5
	1	K	G9
G7	0	C	G2
	1	C	G11
G8	0	I, J	G10
	1	I, J	G6
G9	0	G, H	G8
	1	G, H	G8
G10	0	E, F	G9
	1	E, F	G9
G11	0	D	G10
	1	D	G10

Синтез минимального автомата. Шаг 3

Назначим представителей группам в соответствии с табл. П4.13. При этом принимающим состоянием (формирующим сигнал *GOOD* = ‘1’) будет представитель S10, а тупиковым (формирующим сигнал BAD = ‘1’) – представитель S2. Стартовым состоянием минимального автомата будет S0 – представитель, содержащий стартовое состояние исходного автомата. Граф переходов автомата приведен на рис. П4.1.

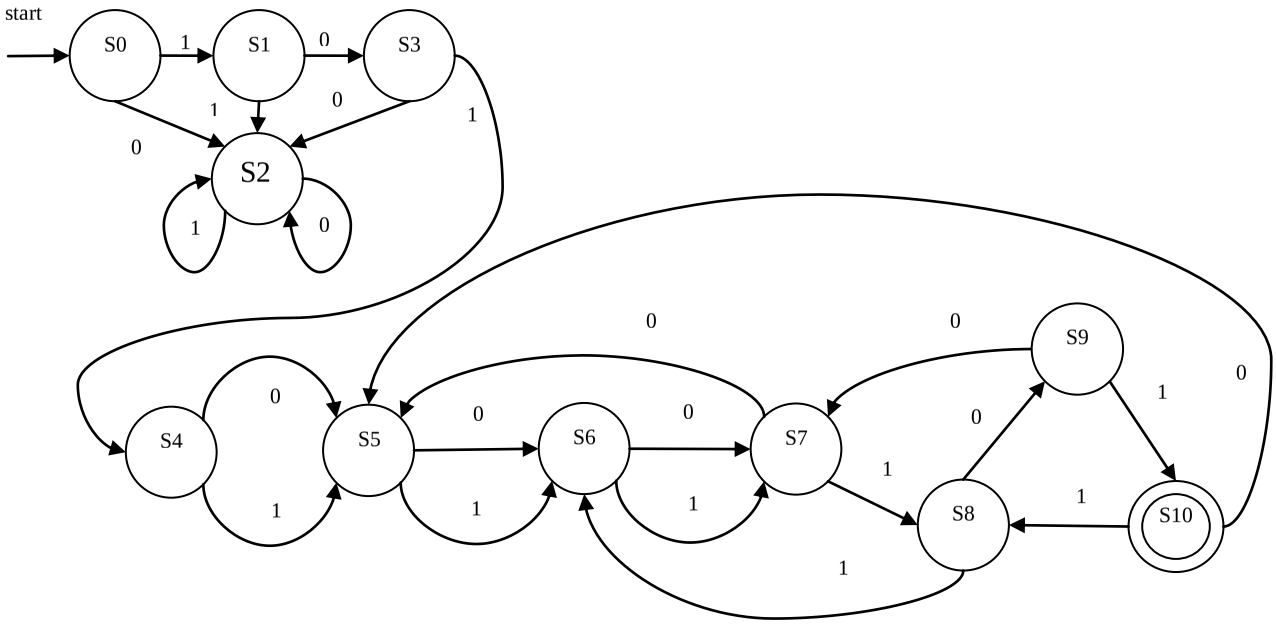


Рис. П4.1. Граф переходов минимального детерминированного конечного автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101

Таблица П4.13

Таблица переходов минимального детерминированного конечного автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101

Состояние	Группа	Переходы		Сигналы	
		0	1	GOOD	BAD
S0	G3	S1	S2	0	0
S1	G4	S3	S2	0	0
S2	G2	S2	S2	0	1
S3	G7	S2	S4	0	0
S4	G11	S5	S5	0	0
S5	G10	S6	S6	0	0
S6	G9	S7	S7	0	0
S7	G8	S5	S8	0	0
S8	G6	S9	S6	0	0
S9	G5	S7	S10	0	0
S10	G1	S5	S8	1	0

Синтез минимального детерминированного конечного автомата, соответствующего регулярному выражению 101((0|1)(0|1)(0|1))+101, завершен. Получен автомат, содержащий на три состояния (и шесть переходов) меньше, чем исходный.

СПИСОК ЛИТЕРАТУРЫ

1. Хопкрофт Джон Э., Мотвани Раджив, Ульман Джеффри Д. Введение в теорию автоматов, языков и вычислений, 2-е изд.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2002. – 528 с.

2. Ахо Альфред В, Лам Моника С., Сети Рави, Ульман Джеффри Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд.: Пер. с англ. – М.: ООО “И.Д. Вильямс”, 2008. – 1184 с.

Никита Алексеевич Дмитриев
Александр Александрович Дюмин
Михаил Николаевич Ёхин
Борис Николаевич Ковригин
Владимир Георгиевич Тышкевич
Лариса Ивановна Шустова
Игорь Михайлович Ядыкин

ТЕОРИЯ АВТОМАТОВ
Лабораторный практикум

Учебное пособие

Редактор Е.К. Коцарева

Подписано в печать 15.11.2012. Формат 60x84 1/16
Печ. л. 12,0. Уч.-изд. л. 12,0. Тираж 110 экз.
Изд. № 35/1. Заказ № 49.

Национальный исследовательский ядерный университет «МИФИ».
115409, Москва, Каширское ш., 31

ООО «Полиграфический комплекс «Курчатовский».
144000, Московская область, г. Электросталь, ул. Красная, д. 42