Минобрнауки России

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Оренбургский государственный университет»**

Кафедра промышленной электроники и информационно-измерительной техники

**Методические указания**

для обучающихся по освоению дисциплины

*«Б1.Д.В.1 Проектирование цифровых устройств на программируемой логике»*

Уровень высшего образования

МАГИСТРАТУРА

Направление подготовки

*11.04.04 Электроника и наноэлектроника*

(код и наименование направления подготовки)

*Электронные приборы и устройства*

(наименование направленности (профиля) образовательной программы)

Квалификация

*Магистр*

Форма обучения

*Очная*

Оренбург, 2024

Составитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Хлуденев

Методические указания рассмотрены и одобрены на заседании кафедры промышленной электроники и информационно-измерительной техники

Заведующий кафедрой ПЭиИИТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.С. Лелюхин

Методические указания являются приложением к рабочей программе дисциплины « Проектирование цифровых устройств на программируемой логике», зарегистрированной в ЦИТ под учетным номером \_\_\_\_\_\_\_\_\_\_.

**1 Общие положения**

Дисциплина относится к базовой части блока Д «Дисциплины (модули)». Изучается в 2 семестре.

В результате изучения дисциплины студент должен:

**Уметь:** подготавливать технические задания на выполнение проектных работ на платформе ПЛИС, разрабатывать устройства электронной техники на платформе ПЛИС.

**Владеть:** навыками разработки цифровых устройств на платформе ПЛИС, навыками разработки и отладки проектов на платформе ПЛИС.

Достижение планируемых результатов обучения обеспечивается добросовестным отношением обучающегося к изучению теоретического материала дисциплины с использованием материала лекций и рекомендованной литературы, ознакомлением с рекомендованными периодическими изданиями, своевременным выполнением заданий на практических занятиях, а также выполнением и защитой курсового проекта.

**2 Общие методические рекомендации**

Контроль достижения планируемых результатов обучения обеспечивается использованием оценочных средств, представленных в «Фонде оценочных средств …» (приложение к рабочей программе дисциплины), в учебном процессе во время занятий по расписанию, а также во время промежуточной аттестации.

**2.1 Теоретический материал, изучаемый при освоении дисциплины**

**Раздел 1 Архитектура и схемотехника ПЛИС**

Классификация ПЛИС. Структуры ПЛМ и ПЛУ, CPLD. FPGA. Системы на программируемом кристалле.

**Раздел 2 Языки описания аппаратуры (HDL)**

Основы языков Verilog HDL и System Verilog. Структурные и функциональные описания.

**Раздел 3 Проектирование на базе ПЛИС**

Специализированные САПР. Основные этапы проектирования на базе ПЛИС. Проблемы и методы проектирования систем на кристалле. Сложные функциональные блоки (СФБ). Ядро процессора NIOS II. Встроенные средства контролепригодности.

**Вопросы, изучаемые самостоятельно с использованием системы электронного обучения Moodle и рекомендуемой литературы:**

**Раздел 1 Архитектура и схемотехника ПЛИС**

Архитектура FPGA Cyclone. Архитектура системы на кристалле Cyclone V. Отладочные стенды SDK 6/1 и DE1-SoC.

**Раздел 2 Языки описания аппаратуры (HDL)**

HDL описания типовых функциональных узлов (на языке Verilog)..

**Раздел 3 Проектирование на базе ПЛИС**

Инструменты Quartus II Web Edition и Quartus Lite Edition.

**Контроль усвоения** изученного теоретического материала осуществляется методом опроса на практических занятиях.

**Рекомендуемая литература**:

1. Грушвицкий, Р. И.  Проектирование систем на микросхемах программируемой логики [Текст] : учеб. пособие для вузов / Р. И. Грушвицкий, А. Х. Мурсаев, Е. П. Угрюмов. - CПб. : БХВ-Петербург, 2002. - 608 с. : ил. - (Учебное пособие). - ISBN 5-94157-002-3.
2. Стешенко, В. Б. ПЛИС фирмы ALTERA: проектирование устройств обработки сигналов [Текст]  / В. Б. Стешенко. - М. : ДОДЭКА, 2000. - 128 с. - ISBN 5-94020-001-Х*.*

**2.2 Практикум**

С целью выработки умений использовать средства разработки цифровых устройств на ПЛИС предусмотрено выполнение студентами заданий на практических занятиях. Тематика практических занятий приведена в рабочей программе дисциплины.

Отчет о выполненном задании каждый студент защищает индивидуально. Во время защиты студент должен быть готовым пояснить методику выполнения, показать умение анализировать результаты, полученные в ходе выполнения задания, быть готовым ответить на вопросы преподавателя по теме задания.

Практические задания студенты выполняют под руководством преподавателя с использованием методических рекомендаций, которые приведены в приложении А.

**2.3 Курсовое проектирование**

Целью выполнения курсового проекта является закрепление практических навыков самостоятельного решения инженерных задач в области реализации алгоритмов управления и обработки информации, развитие творческих способностей и овладение инструментальными средствами разработки цифровых устройств на платформе ПЛИС.

Темой проекта является разработка цифрового устройства, реализующего функции обработки информации и управления (по вариантам). Предполагается, что цифровое устройство выполнено на ПЛИС семейства Cyclone или Cyclone V, и в качестве его прототипа используется учебно-отладочный стенд SDK 6.1 или DE1-SoC. Требуется разработать и отладить цифровое устройство на кристалле ПЛИС. Задание должно быть выполнено с использованием инструментальных средств разработки Quartus II Web Edition или Quartus Lite Edition. Правильность выполнения задания контролируется выполнением тестирования на функционально-логической модели и отладочном стенде. В ходе тестирования устанавливается соответствие реализуемых функций требованиям задания, локализуются и исправляются допущенные ошибки и неточности.

Над курсовым проектом студенты работают самостоятельно – в свободное от занятий время. Возникающие в процессе работы вопросы студенты могут решать в часы консультаций с руководителем проекта. При выполнении курсового проекта руководствоваться методическими рекомендациями (приложение Б).

Срок сдачи преподавателю законченного курсового проекта на проверку – за 1 – 3 дня до защиты. Защита курсового проекта – 18 учебная неделя (согласно расписанию).

**Приложение А**

**Методические рекомендации к выполнению практических заданий**

**1 Мегафункции типовых функциональных узлов**

* 1. **Описание работы с кратким изложением теории**

Для построения типовых узлов удобно использовать функциональные или поведенческие описания, используя средства языка Verilog. Рассмотрим формирование функционального описания с помощью операторов always и case на примере мультиплексора.

Мультиплексор «4 в 1» передает данные с одного из входов C, D, E, F на выход OUT. Выбор определяется значением двухразрядного кода S [1:0]. Функциональное описание модуля mux4\_1:

module mux4\_1 (

input C,

input D,

input E,

input F,

input [1:0] S,

output OUT

);

reg OUT;

always @(\*) begin

case (S)

2'b00 : OUT = C;

2'b01 : OUT = D;

2'b10 : OUT = E;

default : OUT = F;

endcase

end

endmodule

При описании функциональных узлов последовательностного типа целесообразно использовать RTL описания (на уровне регистровых перечач). Работу регистра можно описать в форме таблицы режимов (таблица 1.1).

Таблица 1.1 – Режимы работы регистра

|  |  |  |  |
| --- | --- | --- | --- |
| Режим работы | clk | ld | Q+[ 7..0] |
| Загрузка | ↑ | 1 | D[7..0] |
| Хранение | x | 0 | Q[7..0] |

Функциональное описание регистра Rg:

module rg (

input wire clk,

input wire [7:0]di,

input wire ld,

output reg [7:0]q

);

always @ (posedge clk)

if (ld)

q <= d;

Режимы работы универсального счетчика с возможностью загрузки кода di[7:0] и асинхронным сбросом приведены в таблице 1.2.

Таблица 1.2 – Режимы работы счетчика

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Режим работы | clk | ld | ce | Q+[ 7..0] |
| Загрузка | ↑ | 1 | x | D[7..0] |
| Инкремент | ↑ | 0 | 1 | Q[7..1] + 1 |
| Хранение | х | 0 | 0 | Q[7..0] |

Представление счетчика на уровне регистровых передач приведено на рисунке 1.1.

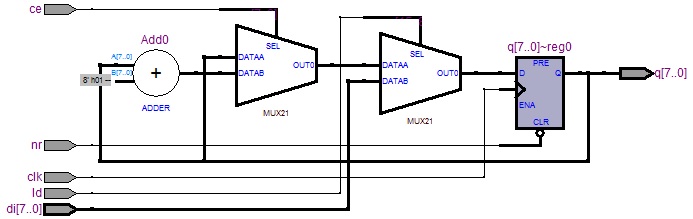


Рисунок 1.1

Функциональное описание счетчика:

module cntr (

input wire nr,

input wire clk,

input wire ce,

input wire [7:0]di,

input wire ld,

output reg [7:0]q

);

always @ (posedge clk or negedge nr)

if (~nr)

q <= 8'h00;

else

if (ld)

q <= di;

else

if (ce)

q <= q + 8'h01;

endmodule

**1.2 Задание к выполнению лабораторной работы**

1.2.1 Сформировать мегафункцию (Verilog, SystemVerilog) комбинационного узла в соответствии с вариантом задания (таблица 1.1).

1.2.2 Сформировать функциональный тест.

1.2.3 Выполнить функциональное моделирование мегафункции комбинационного узла.

1.2.4 Сформировать мегафункцию (Verilog, SystemVerilog) последовательностного узла в соответствии с вариантом задания (таблица 1.2).

1.2.5 Сформировать функциональный тест.

1.2.6 Выполнить функциональное моделирование мегафункции последовательностного узла.

1.2.7 Ответить на контрольные вопросы, оформить отчет.

Таблица 1.1

|  |  |  |
| --- | --- | --- |
| Серия SN74 | Серия К155 | Функция |
| 7483A | К155ИМ3 | Четырехразрядный сумматор |
| 7485 | К155СП1 | Четырехразрядный компаратор |
| 74145 | К155ИД10 | Двоично-десятичный дешифратор |
| 74147 | K155ИВ3 | Двоично-десятичный шифратор |
| 74148 | К155ЛН1 | Двоичный шифратор |
| 74150 | К155КП1 | Мультиплексор |
| 74151 | K155КП7 | Мультиплексор |
| 74154 | К155ИД3 | Двоичный дешифратор |
| 74182 | К155ИП4 | Формирователь переноса |
| 74184 | K155ПР6 | Преобразователь кода |

Таблица 4.2

|  |  |  |
| --- | --- | --- |
| Серия SN74 | Серия К155 | Функция |
| 7495 | К155ИР1 | Четырехразрядный регистр |
| 74160 | К155ИЕ9 | Четырехразрядный счетчик |
| 74164 | К155ИР8 | Регистр сдвига |
| 74165 | K155ИР9 | Регистр сдвига |
| 74191 | К155ИЕ13 | Четырехразрядный счетчик |
| 74192 | К155ИЕ6 | Четырехразрядный счетчик |
| 74193 | K155ИЕ7 | Четырехразрядный счетчик |
| 74194 | К155ИР11 | Регистр сдвига |
| 74197 | К155ИЕ15 | Четырехразрядный счетчик |
| 74198 | K155ИР13 | Регистр сдвига |

**1.3 Порядок выполнения задания**

1.3.1 Запустите на выполнение Quartus II, используя мастер *New Project Wiz­ard…*, создайте проект в директории C:\altera\qdesigns\Pz1.

1.3.2 Сформируйте и сохраните в файле <имя>.v исходный текст мегафункции.

1.3.3 Используя сигнальный редактор, создайте тест, включающий последовательность наборов входных сигналов. Задайте шаг сетки моделирования для периода 10 нс. Задайте требуемое время моделирования для обработки всех входных наборов. Сохраните тест в файле <имя>.vwf.

При формировании теста следует учесть, что проверка мегафункции на всех наборах исходных данных может потребовать слишком много тактов моделирования. Поэтому рекомендуется построить минимально короткий тест, обеспечивающий максимально полную проверку мегафункции.

1.3.4 Используя инструмент *Simulator Tool*, создайте модель мегафункции и выполните функциональное моделирование *Functional* (без учета задержек). Выполните анализ временных диа­грамм входных и выходного сигналов. По результатам моделирования оцените правильность работы функционального узла. Зарисуйте временные диаграммы.

**1.4 Содержание отчета**

Отчет должен содержать:

- исходные тексты мегафункций узлов;

- результаты моделирования, выводы.

**1.5 Контрольные вопросы**

Поясните реализацию функционального описания комбинационного узла.

Каким образом можно проверить корректность функционального описания комбинационного узла?

Поясните особенности формирования функциональных описаний последовательностныых узлов на уровне регистрвых передач.

Поясните правила использования блокирующих и неблокирующих присваиваний?

Каким образом можно проверить корректность функционального описания последовательностного узла?

**2 Мегафункции модулей синхронной памяти**

**2.1 Описание работы с кратким изложением теории**

ПЛИС семейства Cyclone содержат в своей структуре встроенные блоки синхронной памяти M4K RAM, которые позволяют реализовать различные модули памяти. В работе исследуется ПЗУ и статическое ОЗУ. Модули памяти и другие функциональные блоки можно разработать в системе Quartus II на основе стандартных мегафункций. Для этого необходимо использовать инструмент системы Quartus II *MegaWizard Plug-In Manager*. Запуск этого инструмента выполняется из меню Tools. В процессе создания модуля последовательно открываются страницы окна этого инструмента. Далее излагается последовательность разработки синхронного статического ОЗУ:

1. выбрать пункт *Create a new custom megafunction variation*;

2) в окне *Select a megafunction* в папке *Memory Compiler* выбрать мегафункцию RAM: 1Port, в окне *Which device family* выбрать семейство Cyclone, в меню *Which type of output file* выбрать пункт Verilog, в окне *What name do you want for the output file* указать путь и имя файла для сохранения модуля;

3) задать разрядность слов данных (*How wide should the ‘q’ output bus be*), задать число ячеек памяти (*How many words of memory*), выбрать опцию синхронизации чтения и записи (*What clocking method would you like to use*) – Single;

4) в окне *Which ports should be registerd* выбрать пункт синхронизации выходного порта данных *‘q’ outport port*;

5) в окне *What should the ram block type be* выбрать пункт *Auto*, в окне *Do you want to specify the initial content of the memory?* выбрать *No*;

6) в указанной папке будут созданы файлы имя.v, имя.inc , имя.bsf.

Далее излагается последовательность разработки синхронного ПЗУ:

1) выбрать пункт *Create a new custom megafunction variation*;

2) в окне *Select a megafunction* в папке *Memory Compiler* выбрать мегафункцию ROM: 1Port, в окне *Which device family* выбрать семейство Cyclone, в меню *Which type of output file* выбрать пункт Verilog, в окне *What name do you want for the output file* указать путь и имя файла для сохранения модуля;

3) задать разрядность слов данных (*How wide should the ‘q’ output bus be*), задать число ячеек памяти (*How many words of memory*), выбрать опцию синхронизации чтения и записи (*What clocking method would you like to use*) – Single;

4) в окне *Which ports should be registerd* снять флажок с пункта синхронизации выходного порта данных *‘q’ outport port*;

5) в окне *What should the ram block type be* выбрать пункт *Auto*, в окне *Do you want to specify the initial content of the memory?* выбрать *Yes* и в окне *File Name* задать имя файла с информацией для записи в ПЗУ ROM.mif;

6) в указанной папке будут созданы файлы имя.v, имя.inc , имя.bsf.

В синхронных ОЗУ вход адреса стробируется сигналом clock, при этом работа ОЗУ осуществляется по конвейерному принципу. По фронту сигнала clock адрес фиксируется во внутреннем буферном регистре и декодируется, но в матрице памяти в этот момент выбрана ячейка, адрес которой был записан в буфер в предыдущем цикле обращения. Именно в эту ячейку в режиме записи заносятся данные, которые в этот момент находятся в буфере входных данных, а в режиме чтения данные из этой ячейки поступают на выход.

Одновременно с адресом сигнал clock стробирует входные данные, поэтому после каждого тактового импульса clock запись данных, зафиксированных в буфере по фронту предыдущего тактового импульса, выполняется по адресу, который также зафиксирован в буфере адреса в предыдущем такте работы.

При стробировании выхода сигналом clock считанные данные фиксируются в выходном регистре. Если выход не стробирован, эти данные непосредственно транслируются на выход ОЗУ.

Созданные файлы можно использовать для разработки различных проектов на языке AHDL или в графическом представлении. На рисунке 2.1 приведена схема для исследования работы сформированных модулей ROM и SRAM емкостью 256х8.

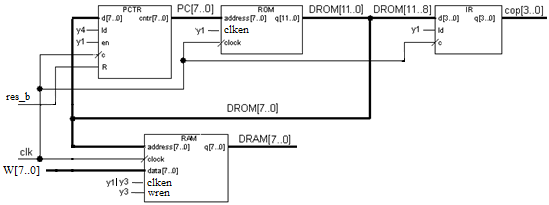


Рисунок 2.1

Данную схему необходимо кодировать на языке Verilog.

Исследовать работу схемы можно с помощью встроенного в систему Quartus II симулятора. Для выполнения моделирования необходимо создать тест, включающий последовательность входных сигналов для исследования работы модулей ROM и SRAM. План исследований может содержать последовательное чтение ячеек ПЗУ и запись данных с входа W[7..0] в ячейки ОЗУ. Данные для загрузки в ПЗУ на стадии компиляции будут находиться в файле Rom.mif.

Тест формируется с помощью сигнального редактора (Waveform Editor). Для выполнения моделирования используется инструмент Simulator Tool в режиме временного моделирования *Timing* (с учетом задержек). При выборе частоты синхронизации принять во внимание результаты анализа быстродействия *Timing Analyzer Tool*. Также необходимо определить значения параметров:

- время предустановки *tsu*;

- время удержания *th*;

- время задержки выхода от фронта синхроимпульса *tco*.

Содержимое ячеек памяти приводится в отчете моделирования *Processing> Simulation Report* в папке *Logical Memories*. По результатам моделирования необходимо сформулировать выводы.

**2.2 Задание к выполнению лабораторной работы**

1. Сформировать модули ROM и SRAM с помощью *MegaWizard Plug-In Manager* (вариант задания из курсового проекта).

2. Сформировать HDL описание схемы для исследования модулей ROM и SRAM.

3. Сформировать файл Rom.mif с данными для загрузки в ПЗУ на стадии компиляции.

4. Выполнить компиляцию проекта.

5. Сформировать тест для исследования модулей ROM и SRAM. Выполнить проверку работы схемы. Зарисовать временные диаграммы сигналов.

6. Ответить на контрольные вопросы, оформить отчет о выполненной работе.

**2.3 Содержание отчета**

1. Схема для исследования модулей ПЗУ и ОЗУ.

2. Временные диаграммы сигналов, значения параметров.

3. Таблица данных в ОЗУ.

**2.4 Контрольные вопросы**

Каким образом в системе Quartus II можно сформировать модули ПЗУ и ОЗУ на основе библиотек мегафункций?

Поясните назначение модулей ПЗУ и ОЗУ, их входов и выходов.

Используя полученные результаты моделирования, объясните работу модулей ПЗУ и ОЗУ.

**3 Мегафункция АЛУ**

**3.1 Описание работы с кратким изложением теории**

Комбинационные арифметико-логические устройства выполняют заданный набор арифметических и логических операций над словами данных. Логические операции над многоразрядными словами выполняются поразрядно. В качестве примера рассмотрим мегафункцию восьмиразрядного АЛУ, выполняющего набор операций из таблицы 3.1.

Таблица 3.1

|  |  |  |
| --- | --- | --- |
| Операция | Мнемоника | Код операции |
| Загрузка | MOV | 000 |
| Логическое И | AND | 001 |
| Логическое ИЛИ | OR | 010 |
| Логическое НЕ | NOT | 011 |
| Сложение с учетом переноса | ADC | 100 |
| Вычитание с учетом заёма | SBB | 101 |
| Циклический сдвиг влево | ROL | 110 |
| Циклический сдвиг вправо | ROR | 111 |

Ниже приводится исходный текст мегафункции на языке Verilog:

module ALU (

op,

a,

b,

res,

cin,

cout,

zout

);

input [1:0] op; // ALU Operation

input [7:0] a; // 8-bit Input a

input [7:0] b; // 8-bit Input b

input cin;

output [7:0] res; // 8-bit Output

output cout;

output zout;

// Reg declarations for outputs

reg cout;

reg zout;

reg [7:0] res;

// Internal declarations

reg addercout; // Carry out straight from the adder itself.

parameter ALUOP\_MOV = 3'b000;

parameter ALUOP\_AND = 3'b001;

parameter ALUOP\_OR = 3'b010;

parameter ALUOP\_NOT = 3'b011;

parameter ALUOP\_ADC = 3'b100;

parameter ALUOP\_SBB = 3'b101;

parameter ALUOP\_ROL = 3'b110;

parameter ALUOP\_ROR = 3'b111;

always @(a or b or cin or op) begin

case (op) // synopsys parallel\_case

ALUOP\_MOV: {addercout, res} = {1'b0, b};

ALUOP\_AND: {addercout, res} = {1'b0, a & b};

ALUOP\_OR : {addercout, res} = {1'b0, a | b};

ALUOP\_NOT: {addercout, res} = {1'b0, ~a};

ALUOP\_ADC: {addercout, res} = a + b + cin;

ALUOP\_SBB: {addercout, res} = a - b - cin;

ALUOP\_ROR: {addercout, res} = {a[0], cin, a[7:1]};

ALUOP\_ROR: {addercout, res} = {a[0], cin, a[7:1]};

default: {addercout, res} = {1'b0, 8'h00};

endcase

end

always @(res)

zout = (res == 8'h00);

always @(addercout or op)

if (op == ALUOP\_SUB)

cout = ~addercout; // Invert adder's carry to get borrow

else

cout = addercout;

endmodule

Исходный текст мегафункции на языке SystemVerilog:

module alu (input logic[2:0] op, input logic[7:0] a, b, input cin,

output logic[7:0] res, output cout, zout);

logic[8:0] alures;

parameter ALUOP\_MOV = 3'b000;

parameter ALUOP\_AND = 3'b001;

parameter ALUOP\_OR = 3'b010;

parameter ALUOP\_NOT = 3'b011;

parameter ALUOP\_ADC = 3'b100;

parameter ALUOP\_SBB = 3'b101;

parameter ALUOP\_ROL = 3'b110;

parameter ALUOP\_ROR = 3'b111;

always\_comb begin

case (op)

ALUOP\_MOV: alures = {1'b0, b};

ALUOP\_AND: alures = {1'b0, a & b};

ALUOP\_OR : alures = {1'b0, a | b};

ALUOP\_NOT: alures = {1'b0, ~a};

ALUOP\_ADC: alures = a + b + cin;

ALUOP\_SBB : alures = a - b - cin;

ALUOP\_ROL: alures = {a[7:0], cin};

ALUOP\_ROR: alures = {a[0], cin, a[7:1]};

default: alures = {1'b0, 8'h00};

endcase

res = alures[7:0];

zout = (alures[7:0] == 8'h00);

if (op == ALUOP\_SBB)

cout = ~alures[8]; // Invert adder's carry to get borrow

else

cout = alures[8];

end

endmodule

**3.2 Задание к выполнению задания**

3.2.1 Сформировать мегафункцию (Verilog, SystemVerilog) АЛУ в соответствии с вариантом задания курсового проекта.

3.2.2 Сформировать функциональный тест.

3.2.3 Выполнить функциональное моделирование мегафункции АЛУ.

3.2.4 Ответить на контрольные вопросы, оформить отчет.

**3.3 Порядок выполнения задания**

3.3.1 Запустите на выполнение Quartus II, используя мастер *New Project Wiz­ard…*, создайте проект ALU в директории C:\altera\qdesigns\lr3.

3.3.2 Сформируйте и сохраните в файле ALU.v исходный текст мегафункции.

3.3.3 Используя сигнальный редактор, создайте тест, включающий:

- последовательность двоичных кодов операций;

- последовательность наборов операндов;

- последовательность значений сигнала переноса C.

Задайте шаг сетки моделирования для периода 10 нс. Задайте требуемое время моделирования для обработки всех входных наборов. Сохраните тест в файле ALU.vwf.

При формировании теста следует учесть, что проверка мегафункции на всех наборах исходных данных потребует слишком много тактов моделирования. Поэтому рекомендуется построить минимально короткий тест, обеспечивающий максимально полную проверку мегафункции для всех операций и комбинаций признаков результатов.

3.3.4 Используя инструмент *Simulator Tool*, создайте модель мегафункции АЛУ и выполните функциональное моделирование *Functional* (без учета задержек). Выполните анализ временных диа­грамм входных и выходного сигналов АЛУ. По результатам моделирования оцените правильность работы АЛУ. Зарисуйте временные диаграммы.

**3.4 Содержание отчета**

Отчет должен содержать:

- исходный текст мегафункции АЛУ;

- результаты моделирования, выводы.

**3.5 Контрольные вопросы**

Какую функцию выполняют АЛУ в цифровых устройствах?

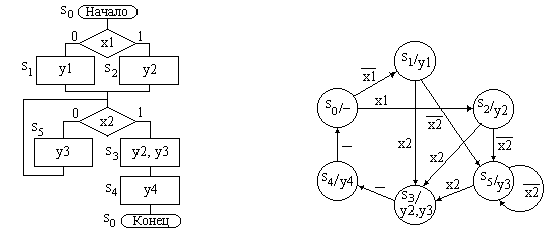
Приведите примеры выполнения типовых операций АЛУ.

Как формируются признаки результата операций АЛУ?

**4 Мегафункция микропрограммного автомата**

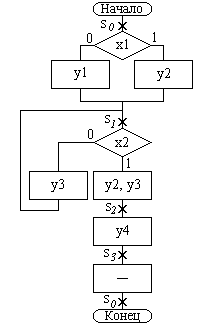
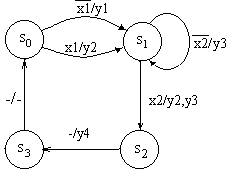
**4.1 Описание работы с кратким изложением теории**

Рассмотрим пример управляющего автомата, выполняющего микропрограмму, заданную граф-схемой алгоритма (ГСА) на рисунке 4.1. На ГСА выполнена отметка состояний для построения автомата Мура, рядом представлен граф автомата Мура. Отметка состояний на ГСА и граф для автомата Мили приведены на рисунке 4.2.



а) б)

Рисунок 4.1

а) б)

Рисунок 4.2

В языке Verilog не предусмотрены специальные типы данных для представления внутреннего состояния конечного автомата. Используется стандартный тип reg. Символы (или символические имена) множества внутренних состояний и их двоичные коды удобно объявлять как параметры. Функциональное описание конечного автомата должно включать описание процесса работы регистра состояния и описания комбинационной схемы, формирующей значения кода следующего состояния и значения выходов в соответствии с заданными функциями переходов и выходов.

Составим описания автоматов на языке Verilog, для задания функций переходов и выходов используем оператор CASE. Описание автомата Мура:

module Moore (

input clk,

input rst,

input en,

input [2:1] xr,

output [4:1] y

);

parameter s0 = 3'b000;

parameter s1 = 3'b001;

parameter s2 = 3'b010;

parameter s3 = 3'b011;

parameter s4 = 3'b100;

parameter s5 = 3'b101;

reg [2:0] state, next\_state;

reg [4:1] yc;

always @ (posedge clk)

if (rst) // reset strategy

state <= s0;

else

state <= next\_state;

always @ (xr, en, state) begin

next\_state = state;

yr = 4'b0000;

case (state)

s0: begin

yr = 4'b0000;

if (xr[1])

next\_state = s2;

else

next\_state = s1;

end

s1: begin

yr = 4'b0001;

if (xr[2])

next\_state = s3;

else

next\_state = s5;

end

s2: begin

yr = 4'b0010;

if (xr[2])

next\_state = s3;

else

next\_state = s5;

end

s3: begin

yr = 4'b0110;

next\_state = s4;

end

s4: begin

yr = 4'b1000;

next\_state = s0;

end

s5: begin

yr = 4'b0100;

if (xr[2])

next\_state = s3;

else

next\_state = s5;

end

endcase

end

assign y = yr;

endmodule

Переменные state и next\_state представляют текущее и следующее внутреннее состояние автомата. Переменная yc представляет выходы комбинационной схемы автомата, ее значения непрерывно передаются на выходы модуля y.

Особенностью автомата Мили является зависимость значений выходных сигналов не только от текущего состояния, но и от значений входных сигналов. Поэтому автомат Мили отличается способом задания значений функции выходов. Кроме этого, выходные сигналы автомата Мили с асинхронными выходами могут изменяться непосредственно вслед за изменениями входных сигналов, не дожидаясь прихода фронта синхросигнала Clk.Чтобы исключить такую возможность, достаточно на выходе автомата Мили включить регистр yr, управляемый синхросигналом Clk и сигналом сброса Rst. Учитывая, что выходные сигналы автомата Мили формируются на один такт раньше, чем у эквивалентного автомата Мура, а также, что включение синхронного регистра на выходе вызовет задержку на один такт, можно утверждать, что построенные с учетом изложенных выше рекомендаций автоматы Мура и Мили будут работать идентично. Описание автомата Мили:

module Mealy (

input clk,

input rst,

input en,

input [2:1] xr,

output [4:1] y

);

parameter s0 = 3'b000;

parameter s1 = 3'b001;

parameter s2 = 3'b010;

parameter s3 = 3'b011;

reg [2:0] state, next\_state;

reg [4:1] yc, yr;

always @ (posedge clk)

if (rst) begin // reset strategy

state <= s0;

yr <= 4'b0000;

end

else begin

state <= next\_state;

yr <= yc;

end

always @ (xr, en, state) begin

next\_state = state;

case (state)

s0: begin

next\_state = s1;

if (xr[1])

yc = 4'b0010;

else

yc = 4'b0001;

end

s1: begin

if (xr[2]) begin

next\_state = s2; yc = 4'b0110;

end

else begin

next\_state = s1; yc = 4'b0100;

end

end

s2: begin

yc = 4'b1000;

next\_state = s3;

end

s3: begin

yc = 4'b0000;

next\_state = s0;

end

endcase

end

assign y = yr;

endmodule

В языке SystemVerilog состояние конечного автомата представляется переменной перечислимого типа enum int unsigned. При объявлении переменных state и next\_state коды внутренних состояний можно закодировать. Используются различные формы оператора always для описания регистра состояний и комбинационной схемы автомата. Описание автомата Мура на языке SystemVerilog

module Moore\_s (input clk, rst, en, input logic[2:1] xr, output logic[4:1] y);

enum int unsigned { s0 = 0, s1 = 2, s2 = 4, s3 = 8, s4 = 16, s5 = 32 } state, next\_state;

always\_comb begin : next\_state\_logic

next\_state = s0;

case(state)

s0: if (xr[1])

next\_state = s2;

else

next\_state = s1;

s1: if (xr[2])

next\_state = s3;

else

next\_state = s5;

s2: if (xr[2])

next\_state = s3;

else

next\_state = s5;

s3: next\_state = s4;

s4: next\_state = s0;

s5: if (xr[2])

next\_state = s3;

else

next\_state = s5;

endcase

end

always\_comb begin

case (state)

s0: y = '{0,0,0,0};

s1: y = '{0,0,0,1};

s2: y = '{0,0,1,0};

s3: y = '{0,1,1,0};

s4: y = '{1,0,0,0};

s5: y = '{0,1,0,0};

endcase

end

always\_ff@ (posedge clk or posedge rst) begin

if(rst)

state <= s0;

else

state <= next\_state;

end

endmodule

Сформированные описания автоматов можно использовать для синтеза и анализа работы в среде системы Quartus II. Для автомата Мура алгоритмом предусмотрено выполнение переходов:

s0 – s1; s0 – s2;

s1 – s3; s1 – s5;

s2 – s3; s2 – s5;

s3 – s4;

s4 – s0;

s5 – s3; s5 – s5.

Необходимо сформировать такую цепочку из этих переходов, чтобы в ней были все переходы, а общее число переходов было минимально, например:

s0 – s1 – s5 – s5 – s3 – s4 – s0 – s2 – s3 – s4 – s0 – s2 – s5.

Для данной последовательности сформируем тест, задавая последовательность соответствующих значений входных сигналов xr1, xr2, необходимых для выполнения каждого перехода.

На рисунках 4.3 и 4.4 соответственно приведены диаграммы сигналов рассмотренных автоматов Мура и Мили. Анализ результатов позволяет сделать выводы:

- эти автоматы эквивалентны;

- автоматы работают в соответствии с заданным алгоритмом.

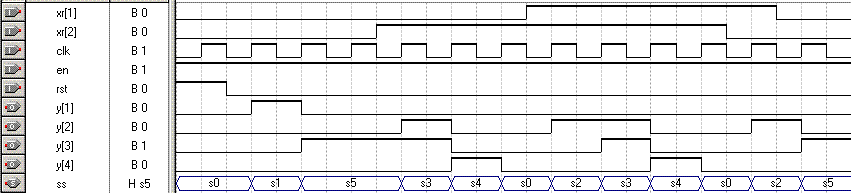


Рисунок 4.3

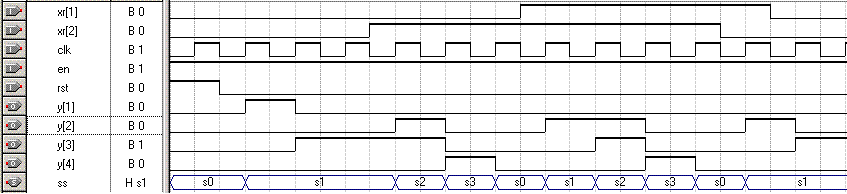


Рисунок 4.4

Исследование работы автомата на стенде SDK 6.1 удобно выполнять в потактовом режиме. Потактовый режим работы обеспечивается за счет использования формирователя одиночного импульса, выделяющего из импульсной последовательности высокой частоты импульс при нажатии кнопки SB. Схема подключения кнопки показана на рисунке 4.5.

В нажатом состоянии значение линии Ctrl\_bt будет иметь логический уровень нуля. В свободном состоянии – уровень единицы.

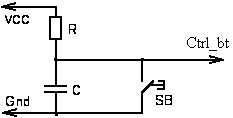


Рисунок 4.5 – Схема подключения кнопки

Следует обратить внимание, что при нажатиях на кнопку возможен “дребезг” сигнала, который может приводить к многократному переключению автомата вместо правильного однократного. Конденсатор C несколько улучшает ситуацию, однако не является абсолютным решением проблемы со стабильностью логического уровня на линии. Для устранения “дребезга” используем специальный модуль Debounce.tdf.

Полный текст описания устройства для исследования автомата в потактовом режиме приведен в файле Pz4.v.

Через меню Assignment/Device активируем окно назначения типа ПЛИС Settings. В окне Family выбираем семейство Cyclone и в окне Available devices тип ПЛИС EP1C3T144-8. Далее необходимо нажать кнопку Device & Pin Options, в открывшемся окне выбрать вкладку Programming Files и установить метку на пункте Raw Binary File (.rbf), затем выбрать вкладку Unused Pins и выбрать в группе переключателя Reserve all unused pins позицию As inputs, tri-stated. Закрываем окна Device & Pin Options и Settings.

Через меню Assignment/Pins активируем окно назначения используемых в проекте выводов ПЛИС. Необходимые для данной работы данные приведены в таблице 4.1.

Таблица 4.1

|  |  |  |
| --- | --- | --- |
| Вывод исследуемой схемы | Сигнальная линия стенда | Вывод ПЛИС |
| xr[1] | SW0 | Pin\_112 |
| xr[2] | SW1 | Pin\_120 |
| Clk | clk | Pin\_93 |
| Ctrl\_bt | Ctrl\_bt | Pin\_133 |
| y[1] | LED0 | Pin\_104 |
| y[2] | LED1 | Pin\_105 |
| y[3] | LED2 | Pin\_106 |
| y[4] | LED3 | Pin\_107 |

После полной компиляции проекта и конфигурирования ПЛИС необходимо проверить правильность работы автомата. Для этого, устанавливая комбинации положений переключателей «SW0»-«SW1», задаем значения входных сигналов, нажатием кнопки «CONTROL» переводим автомат в следующее состояние и по состоянию светодиодов «D0»-«D3» определяем значения выходных сигналов.

**4.2 Задание к выполнению задания**

4.2.1 Сформировать функциональное описание микропрограммного автомата для варианта курсового проекта.

Выполнить неполную компиляцию модуля автомата в среде Quartus II.

4.2.2 Создать файл с описанием тестовых сигналов (средствами Waveform Editor), выполнить моделирование модуля (команда Simulator) и убедиться в правильности его работы (в противном случае найти и устранить ошибки).

4.2.3 Создать прототип модуля (команда Create AHDL Include File for Current File).

4.2.4 Выполнить полную компиляцию проекта Lr5.tdf в среде системы Quartus II.

4.2.5 Выполнить программирование конфигурационного ПЗУ стенда.

4.2.6 Произвести проверку работы автомата в потактовом режиме на стенде SDK6.1. По результатам исследования в среде симулятора и на стенде сделать выводы о соответствии реализованного автомата заданному алгоритму функционирования.

4.2.7 Ответить на контрольные вопросы, оформить отчет.

**4.3 Содержание отчета**

Отчет должен содержать:

- описание проекта на языке Verilog или SystemVerilog;

- результаты исследования автомата в среде симулятора и на стенде, выводы.

**4.5 Контрольные вопросы**

Как в языках Verilog и SystemVerilog определен конечный цифровой автомат?

Как в языках Verilog и SystemVerilog можно описать функции переходов и выходов автомата?

Как отличается функционирование автомата Мура и эквивалентного ему автомата Мили с асинхронными выходами?

Как можно устранить эти отличия?

Как обеспечить функционально полный анализ работы конечного цифрового автомата?

Как можно устранить эффект «дребезга» контактов?

**5 Исследование преобразователя кода**

**5.1 Описание работы с кратким изложением теории**

5.1.1 Цель работы – экспериментально исследовать работу преобразователя кода символов на стенде SDK6.1.

5.1.2 Двоичное кодирование используется в цифровых устройствах для представления не только чисел, но и символьной информации. При использовании кода ASCII можно представлять цифры, прописные и строчные буквы латинского алфавита, другие символы. Расширенный восьмиразрядный код ASCII позволяет также представлять буквы национальных алфавитов. В стенде SDK6.1 установлен двухстрочный жидко-кристаллический индикатор (ЖКИ) типа PC1602. Каждая строка содержит 16 знакомест. Для отображаемых символов используются ASCII коды.

Для многих прикладных задач бывает достаточно отображать ограниченный набор символов. В этих случаях для кодирования символов обычно используют другие коды с меньшим количеством разрядов. Например, для представления символов шестнадцатеричного алфавита достаточно использовать четырехразрядный код. Тогда для отображения этих символов на ЖКИ типа PC1602 необходим преобразователь четырехразрядного кода в код ASCII.

5.1.3 Преобразователи кода относятся к цифровым узлам комбинационного типа. Алгоритм их функционирования можно представить таблицей истинности.

5.1.4 Задачу построения преобразователя кода на кристалле ПЛИС можно возложить на компилятор системы Quartus II, если описать его алгоритм функционирования на языке AHDL, используя средства описания таблиц истинности.

5.1.5 Примем, что в качестве задатчика входного двоичного кода будут использоваться движковые переключатели SW[3..0] стенда SDK6.1. Отображаемый символ будет располагаться на месте младшего разряда первой строки. Для загрузки ASCII кода в ЖКИ типа PC1602 используем контроллер, реализованный в модуле Lcd\_controller (файл Lcd\_controller.v).

Исследование преобразователя на стенде SDK6.1 будем проводить в составе проекта, описанного в файле Bin\_LCD.v. Данный проект описывает схему включения преобразователя кода, контроллера ЖКИ, а также функциональных узлов, обеспечивающих его правильное функционирование.

module Bin\_LCD (clk,sw,data\_ind,e,rs,rw);

//input res\_b; //system reset

input clk; //system clock

input [3:0] sw;

inout [7:4] data\_ind;

output e, rs, rw;

//------------------------------ REG/WIRE declarations -------------------------------

reg [3:0] cnt;

wire reset;

reg [18:0] divf;

wire f200Hz;

wire bt;

//-------------------------------- Structural coding ---------------------------------

always @(posedge clk)

if (cnt == 16'hFFFF)

cnt <= cnt;

else

cnt <= cnt + 1'b1;

assign reset = cnt == 16'hFFFF;

wire res\_b = ~reset;

always @(posedge clk)

divf <= divf + 1'b1;

assign f200Hz = (divf==524287);

wire [127:0] str1, str2;

bin\_ASCII (.binary (sw), .ASCII(str1[127:120]));

assign str1[119:0] = 120'h202020202020202020202020202020;

assign str2 = 128'h20202020202020202020202020202020;

lcd\_controller (.clk (clk), .reset\_b (res\_b), .data\_ind (data\_ind), .e (e), .rw (rw), .rs (rs), .string1 (str1), .string2 (str2), .refresh (f200Hz));

endmodule

**5.2 Задание к выполнению лабораторной работы**

5.2.1 Сформировать описание преобразователя кода Bin\_ASCII и выполнить его моделирование в среде системы Quartus II.

5.2.2 Изучить описание проекта Bin\_LCD.v. Выполнить компиляцию проекта Bin\_LCD и сформировать конфигурационный файл Bin\_LCD.rbf.

5.2.3 Выполнить конфигурирование ПЛИС.

5.2.4 Выполнить проверку работы преобразователя кода на стенде SDK6.1, по ее результатам сделать выводы.

**5.3 Порядок выполнения работы**

5.3.1 Запустите на выполнение Quartus II, выберите из меню *File* пункт *New Project Wiz­ard…* - мастер создания новых проектов. В открывшемся окне нажмите кнопку *Next* и в окне текущей директории проекта за­полните строки:

C:\altera\qdesigns\lr5

Bin \_ASCII

Bin \_ASCII

Нажмите кнопку *Finish* и подтвердите создание проекта. Текущий проект будет назван Bin \_ASCII.

5.3.2 При помощи меню *File/New…* на закладке *Device Design File* выберите *Verilog File* и нажмите ОК. В открывшемся окне текстового редактора введите описание преобразователя кода. После окончания ввода сохраните текст в файле Bin\_ASCII.v через меню *File/Save As…*

5.3.3 Для моделирования необходимо создать тест, включающий последовательность двоичных наборов входных сигналов из таблицы 5.1. Для этого откройте сигнальный редактор и создайте в нем файл с расшире­нием .vwf. При помощи меню *Edit/End Time…* задайте время моделирования 160 нс, а в меню *Edit/Grid Size…* - шаг сетки моделирования для периода 10 нс. Да­лее в окне сигнального редактора двойным щелчком правой кнопки мыши на поле *Name*  вызовите меню *Insert Node or Bus*, при помощи которого задайте имя входного вектора Bin в поле *Name* и число разрядов 4 в поле *Bus Width*. Для входных выводов схемы задайте значения сигналов на протяжении необходимого времени моделирования – 160 нс. В данном случае для формирования значений сигналов проще всего воспользоваться инструмен­том *Count Value*, который активируется после выбора сигнала в поле *Name.* Используя инструмент *Count Value,* задайте для Bin значения наборов 0000 … 1111 на интервале 160 нс (смена наборов через 10 нс задается значением параметра *Count Every*). В окне редактора входные сигналы представ­ляются в форме временных диаграмм. Убедитесь, что сформированные сигналы полностью соответствуют замыслу модельного эксперимента (сформи­рованы необходимые 16 двоичных наборов). В противном случае необходимо выполнить редактирование теста. После этого сохраните тест в файле Bin\_To\_ASCII.vwf (расширение присваивается автоматически) через меню *File/Save As…*

5.3.4 Для выполнения моделирования вызовите инструмент *Simulator Tool*. В окне инструмента в поле *Simulation Mode* выберите режим функционального моделирования *Functional* (без учета задержек). Для создания модели схемы нажмите кнопку *Generate Functional Simulation Netlist*. В поле *Simulation Input* выберите и задайте имя созданного файла тестовых сигналов, после чего наж­мите кнопку *Start*. После завершения расчетов нажатием кнопки *Open* откройте окно сигнального редактора, в котором будут отражены временные диа­граммы входных и выходного сигналов схемы. По результатам моделирования необходимо выполнить проверку правильности работы преобразователя кода путем сравнения с таблицей истинности.

5.3.5 Выберите из меню *File* пункт *New Project Wiz­ard…* - мастер создания новых проектов. В открывшемся окне нажмите кнопку *Next* и в окне для задания текущей директории проекта за­полните три строки:

C:\altera\qdesigns\lr5

Bin\_LCD

Bin\_LCD

нажмите кнопку *Finish* и подтвердите создание проекта. Текущий проект будет назван Bin\_LCD в директории по адресу С:\altera\qdesigns\Lr5.

5.3.6 Выполните анализ исходного описания и логический синтез проекта из меню *Processing/Start/Start Analysis & Synthesis*.

5.3.7 Через меню *Assignment/Device* активируйте окно назначения типа ПЛИС *Settings*. В окне *Family* выберите семейство Cyclone и в окне *Available devices* - тип ПЛИС EP1C3T144-8. Далее необходимо нажать кнопку *Device & Pin Options*, в открывшемся окне выбрать вкладку *Programming Files* и установить метку на пункте *Raw Binary File* (.rbf). Затем выберите вкладку *Unused Pins* и в группе переключателя *Reserve all unused pins* позицию *As inputs, tri-stated*. **Невыполнение или неправильное вы­полнение последнего требования может привести в выходу из строя ПЛИС и периферийных модулей стенда!** Закройте окна *Device & Pin Options и Settings*.

5.3.8 Через меню *Assignment/Pins* активируйте окно назначения используемых в проекте выводов ПЛИС. Необходимые для данной работы данные приведены в таблице 5.2.

Таблица 5.2

|  |  |
| --- | --- |
| Вывод исследуемой схемы (сигнальная линия стенда) | Вывод ПЛИС |
| clk | Pin\_93 |
| SW0 | Pin\_112 |
| SW1 | Pin\_120 |
| SW2 | Pin\_113 |
| SW3 | Pin\_121 |
| E | Pin\_131 |
| RW | Pin\_130 |
| RS | Pin\_132 |
| Data\_ind4 | Pin\_124 |
| Data\_ind5 | Pin\_125 |
| Data\_ind6 | Pin\_128 |
| Data\_ind7 | Pin\_129 |

5.3.9 Для запуска процесса компиляции выберите пункт *Start Compilation* из меню *Processing*. Если сообщения об ошибках (Error) отсутствуют, то файл конфигурации сформирован.

5.3.10 Подключите разъем J7 стенда SDK-6.1 к COM-порту ПК при помощи интерфейсного кабеля. По таблице 3.2 задайте страницу flash-памяти 5. ***Только затем*** подключите питание к стенду SDK-6.1.

5.3.11 Переключите стенд в режим загрузки конфигурационных файлов для EP1C3 по последовательному каналу. Для этого нажмите кнопку «RESET» при нажатой и удерживаемой кнопке «CONTROL». Запустите на ПК программатор sdk61fw со следующими параметрами:

sdk61fw write page5 Bin\_LCD.rbf com1

После того, как программатор завершит работу, путем нажатия кнопки «RESET» произведите рестарт системы. После конфигурирования ПЛИС должна выполнять функцию заданного цифрового устройства – в данном случае преобразователя кода.

5.3.12 Проверьте правильность работы преобразователя кода, последовательно устанавливая комбинации положений переключателей «SW3»-«SW0» и фиксируя символы, отображаемые на ЖКИ.

**5.4 Содержание отчета**

Отчет должен содержать:

- таблицу истинности преобразователя кода;

- HDL-описание преобразователя кода;

- схему включения преобразователя кода при исследовании на стенде (сформировать по описанию Bin\_LCD.v);

- выводы по результатам исследования преобразователя кода.

**5.5 Контрольные вопросы**

В каких случаях используются преобразователи кода? Приведите примеры.

Каким образом можно сформировать таблицу истинности для преобразователя кода?

Каким образом можно описать функционирование цифровых узлов комбинационного типа на HDL? Поясните структуру описания преобразователя кода.

Каким образом выполняется исследование работы преобразователя кода на стенде SDK6.1?

**6 Построение модуля процессора NIOS II**

**6.1 Описание работы с кратким изложением теории**

NiosII – программное процессорное ядро, описанное на языке программирования аппаратуры и предназначенное для реализации в ПЛИС фирмы ALTERA. Для создания «системы на кристалле» к процессорному ядру необходимо добавить дополнительные компоненты – память, порты ввода/вывода, таймер, необходимые интерфейсы. Для этой цели используется специальная утилита, входящая в состав САПР Quartus II – SOPC Builder, позволяющая формировать требуемую архитектуру процессорной системы. Пример одного из вариантов реализации архитектуры процессорного ядра показан на рисунке 6.1.

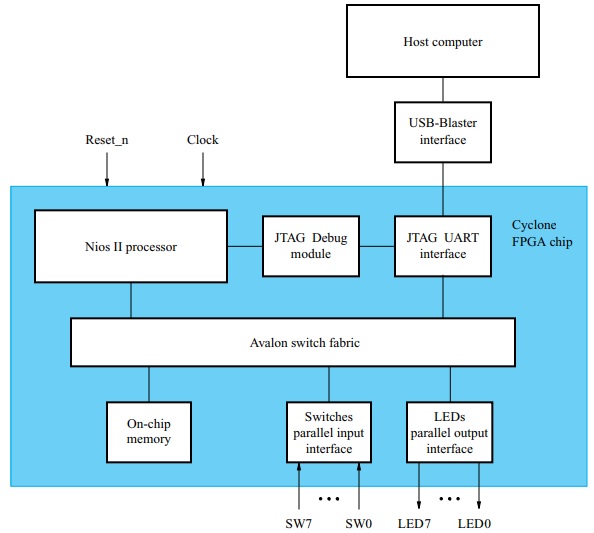


Рисунок 6.1

Для обмена данными с внешними микросхемами (например – SRAM, SDRAM, Flash и пр.) внутри ПЛИС формируются специальные компоненты – интерфейсные блоки. Они связаны с процессорным ядром с помощью специальной шины – Avalon switch fabric. В состав процессорного ядра могут входить и специализированные блоки:

- JTAG UART – для обмена информацией с компьютером с помощью загрузочного кабеля (USB-Blaster или ByteBlaster);

- модуль JTAG Debug – для выполнения отладки программного обеспечения с помощью компьютера;

- UART – модуль асинхронного приемопередатчика для обмена информацией с другими устройствами;

- порты ввода – вывода;

- таймеры.

Пользователь может подключать к процессорному ядру как библиотечные компоненты (входящие в состав САПР Quartus II), так и собственные модули.

Для выполнения данной лабораторной работы используется отладочный стенд SDK6.1.

**6.2 Порядок выполнения работы**

Исходные данные для работы размещены в папке «Pz5».

1. Создайте новый проект в рабочей директории. Укажите название проекта.

2. В качестве микросхемы, для которой создается проект, укажите микросхему **EP1C3F144C8** семейства Cyclone (установлена в отладочном стенде SDK6.1).

3. Создаваемый процессорный модуль должен содержать следующие компоненты:

– процессорное ядро (**Nios II CPU);**

**–**внутренний блок памяти для хранения программы и данных (**onchip memory);**

– контроллер последовательного канала связи **JTAG UART;**

– параллельные порты ввода/вывода для связи с внешними компонентами отладочного модуля – светодиодами, переключателями (**PIO peripherals).**

4. Вызовите утилиту формирования архитектуры процессорного ядра (меню **Tools > SOPC Builder…**). Укажите имя процессорной системы – **nios\_system** и язык описания – **Verilog**.

5. В открывшемся диалоговом окне проверьте системные настройки – семейство ПЛИС **Cyclone**, тактовая частота (**clk\_0**) – внешняя, 40 МГц.

6. Добавьте процессорное ядро в проект. В окне библиотеки системных компонентов (**Component Library**) в папке **Processors**выделите **Nios II Processor**и нажмите кнопку **Add**. В открывшемся диалоговом окне выберите архитектуру **Nios II/e**.

7. Нажмите кнопку **Finish**. В окне сообщений утилиты SOPC Builder могут появиться предупреждения или сообщения об ошибке. Они связаны с тем, что для процессора Nios II не был выбран ряд настроек. Они будут сделаны позже.

8.  Подключите к процессорному ядру внутренний блок памяти. Для этого в разделе **Memories and Memory Controllers**раскройте папку **On-Chip**и выделите компонент **On-Chip Memory (RAM or ROM).**Нажмите кнопку **Add…**В открывшемся диалоговом окне установите следующие параметры: тип памяти RAM (Writable), объем памяти – 4096 байт, разрядность шины данных – 32. Остальные настройки оставьте по умолчанию. Нажмите кнопку **Finish.**

9. Подключите к процессорному ядру контроллер порта ввода для переключателей. Для этого в перечне устройств в разделе **Peripherals**раскройте папку **Microcontroller Peripherals**и выделите компонент **PIO (Parallel I/O).**Нажмите кнопку **Add…**В открывшемся диалоговом окне установите разрядность порта равную 8 бит и направление передачи данных – **Input ports only**. Нажмите кнопку **Finish.**

10. Аналогичным способом подключите к процессорному ядру контроллер порта вывода для светодиодов. Для этого в перечне устройств в разделе **Peripherals**раскройте папку **Microcontroller Peripherals**и выделите компонент **PIO (Parallel I/O).**Нажмите кнопку **Add…**В открывшемся диалоговом окне установите разрядность порта равную 8 бит и направление передачи данных – **Output ports only**. Нажмите кнопку **Finish.**

11. Подключите к процессорному ядру контроллер последовательного канала JTAG UART. Для этого в перечне устройств в разделе **Interface Protocols**раскройте папку **Serial**и выделите компонент **JTAG UART.**Нажмите кнопку **Add…**В открывшемся диалоговом окне измените параметры буферов Write FIFO и Read FIFO: измените глубину буферов (Buffer depth) на 32 байта, установите флаги Construct using registers instead memory blocks. Нажмите кнопку **Finish.**

12. Процессорный модуль собран. Для изменения имени компонента, входящего в состав процессорного модуля, нужно выделить компонент, нажать правую кнопку мышки и выбрать команду **Rename**из контекстного меню. По заданным именам происходит обращение к компонентам из программы.

13. Переименуйте порт ввода **pio\_0**на **Switches,**а порт вывода **pio\_1**на **LEDs**.

14. Для присвоения базовых адресов устройствам, входящим в состав процессорной системы, в меню **System**утилиты SOPC Builder выберите команду **Auto-Assign Base Addresses**. Обратите внимание на изменение базовых адресов компонентов, входящих в состав процессорного модуля.

15. Укажите размещение стартового адреса программы и таблицы векторов прерывания. Для этого вызовите диалоговое окно настройки процессорного модуля **cpu\_0** (двойным нажатием левой кнопки мыши на его имени). Из выпадающего меню для **Reset Vector**выберите значение **onchip\_memory2\_0**. И для **Exception Vector**– значение **onchip\_memory2\_0** (рисунок 13). Нажмите кнопку **Finish**.

16. Перейдите на страницу **System Generation**утилиты SOPC Builder. Проверьте, что опция **Simulation. Create project simulator files**осталась не выбранной. Сохраните все настройки процессорной системы (меню **File > Save**). Нажмите кнопку **Generate**.

17. После получения сообщения об успешном завершении создания процессорной системы, закройте утилиту SOPC Builder.

18. Создайте в САПР Quartus II файл верхнего уровня иерархии. Это может быть графический файл или файл на языке HDL. В примере показано создание графического файла В нем должны быть указаны внешние сигналы, которые подаются на процессорный модуль (тактовый сигнал, сигнал сброс, входные сигналы с переключателей и выходные сигналы на светодиоды).

19. Добавьте в файл созданный процессорный модуль (рисунок 5.3). Он находится в рабочей директории проекта..

20. Выполните назначение выводов ПЛИС для всех внешних входов и выходов. Для этого в меню **Assignments**выберите команду **Pin Assignments**.

21. Сохраните файл (меню **File > Save**). Сохраните проект (меню **File > Save Project**).

22. Выполните полную компиляцию проекта.

**6.3 Содержание отчета**

Отчет должен содержать:

- описание проекта на языке Verilog или SystemVerilog;

- результаты исследования автомата в среде симулятора и на стенде, выводы.

**6.4 Контрольные вопросы**

Как в языках Verilog и SystemVerilog определен конечный цифровой автомат?

Как в языках Verilog и SystemVerilog можно описать функции переходов и выходов автомата?

Как отличается функционирование автомата Мура и эквивалентного ему автомата Мили с асинхронными выходами?

Как можно устранить эти отличия?

Как обеспечить функционально полный анализ работы конечного цифрового автомата?

**7 Исследование модуля процессора NIOS II**

1. После успешной компиляции проекта, загрузите его в ПЛИС, установленную в отладочном модуле (конфигурационный файл – \***.sof**).

2. Откройте среду разработки программного обеспечения для встраиваемого процессорного ядра Nios II IDE.

3. Создайте новый проект для процессорного ядра. Для этого в меню **File**выберите команду **New > Project**. В открывшемся диалоговом окне выберите **Nios II C/C++ Application**и нажмите кнопку **Next>**.

4. На следующей странице диалогового окна укажите имя проекта**.**В графе **SOPC Builder System PTF File**укажите расположение файла описания процессорного модуля **nios\_system.ptf**(используйте кнопку **Browse…**Файл располагается в рабочей директории проекта). В окне выбора шаблона для проекта (**Select Project Template**) укажите **Blank Project** (рисунок 17). Нажмите кнопку **Next>**.

5. На следующей странице выберите настройку **Create a new system library named:.**Нажмите кнопку **Finish**. В окне доступных проектов приложения Nios II IDE появятся две новые директории: директория исходных данных проекта (**training**) и директория системной библиотеки проекта (**training\_syslib[nios\_system]**).

6. Создайте для проекта новый исходный файл **training.c**. Для этого в меню **File**выберите команду **New > Source File**. В открывшемся диалоговом окне укажите имя файла (рисунок 18). Нажмите **Finish.**

Напишите простую программу, которая отображает состояние переключателей с помощью светодиодов. Пример такой программы представлен ниже:

#define Switches (volatile char\*) 0x0003000

#define LEDs (char\*) 0x0003010

long i;

char j;

void main()

{

while(1)

{

j = \*Switches;

\*LEDs = ~j;

for (i = 0; i < 400000; i++);

\*LEDs = j;

for (i = 0; i < 400000; i++);

}

}

**Обратите внимание –**имена переменных Switches и LEDs совпадают с именами портов ввода/вывода процессорного модуля, а значения адресов этих переменных – с адресами, назначенными утилитой SOPC Builder.

7.Сохраните файл.

8. Выделите директорию системной библиотеки проекта, нажмите правую кнопку мыши и выберите команду **Properties** из открывшегося контекстного меню. В открывшемся диалоговом окне выберите закладку **System Library**(рисунок 19).

9. Проверьте, что в настройках системной библиотеки проекта функции **stdout, stderr, stdin**связаны с устройством **jtag\_uart\_0**. Все секции программы располагаются в **onchip\_memory2\_0**. Сделаны установки для использования минимальных библиотек языка С и системных драйверов. Нажмите **ОК**.

10. Для компиляции проекта выделите рабочую директорию проекта **training**, нажмите правую кнопку мыши и выберите команду **Build Project** из открывшегося контекстного меню.

11. Если компиляция проекта завершилась без ошибок, проверьте работу программы с помощью отладочного модуля. Для этого выделите рабочую директорию проекта, нажмите правую кнопку мыши и выберите команду **Run As > Nios II Hardware**из открывшегося контекстного меню.

12. После загрузки программы, проверьте ее работоспособность с помощью отладочного модуля.

13. Перейдите в режим отладки программного обеспечения. Для этого выделите рабочую директорию проекта, нажмите правую кнопку мыши и выберите команду **Debug As > Nios II Hardware**из открывшегося контекстного меню. Подтвердите разрешение перехода в режим отладки (кнопка **Yes**).

14. Установите точку останова в строке 13 исходного текста программы (двойное нажатие левой кнопки мыши слева от номера строки). Если номер строки не отображается в текстовом редакторе, в меню **Window**выберите команду **Preferences..** В открывшемся окне раскройте папку **General,**затем папку **Editors**и выделите строку **Text Editors**. В правой части диалогового окна установите опцию **Show line numbers**. Нажмите **ОК**.

15. Нажмите кнопку запуска программы **resume** (или клавишу **F8**). Обратите внимание на точку останова программы.

16. Нажмите один из переключателей на отладочном модуле и повторно запустите программу. Что изменилось?

17. Для выхода из режима отладки в окне **Debug**выделите элемент **Thread[0]**, нажмите правую кнопку мыши и выберите команду **Terminate and Remove**из открывшегося контекстного меню.

**Задания для самостоятельной работы**

1. Разработайте свой вариант программы для процессорного ядра **nios\_system**. Проверьте ее работоспособность на отладочном модуле.

**8 Построение и исследование модуля DSP**

**8.1 Описание работы с кратким изложением теории**

Одна из распространенных операций цифровой обработки сигналов - фильтрация. Для ее реализации могут быть использованы аппаратные ресурсы ПЛИС. В лабораторной работе исследуется процесс реализации фильтра с конечной импульсной характеристикой (КИХ) с помощью инструмента Mega Core FIR Compiler системы Quartus II. Для расчета КИХ-фильтра требуются следующие исходные данные:

- частота дискретизации FS;

- порядок фильтра n;

- граница полосы пропускания fp;

- граница полосы задерживания (подавления) fs;

- неравномерность АЧХ в полосе пропускания δ1; минимальное затухание в полосе задерживания δ.

Варианты заданий приведены в таблице 8.1.

Таблица 8.1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вар | Тип  фильтра | fc1,  кГц | fc2,  кГц | fc3,  кГц | fc4,  кГц | δ,  дБ | δ1,  дБ | FS  МГц |
| 1 | ФНЧ | 800 | 1600 |  |  | 53 | 0.2 | 8 |
| 2 | ФНЧ | 2000 | 4000 |  |  | 50 | 0.15 | 16 |
| 3 | ФНЧ | 1600 | 3200 |  |  | 65 | 0.1 | 16 |
| 4 | ФВЧ | 800 | 1600 |  |  | 53 | 0.2 | 8 |
| 5 | ФВЧ | 2000 | 4000 |  |  | 45 | 0.5 | 16 |
| 6 | ФВЧ | 1000 | 1500 |  |  | 42 | 1 | 8 |
| 7 | ПФ | 500 | 1000 | 2500 | 3000 | 42 | 1 | 8 |
| 8 | ПФ | 500 | 1500 | 2500 | 3500 | 65 | 0.1 | 8 |
| 9 | РФ | 500 | 1000 | 2500 | 3000 | 65 | 0.2 | 8 |
| 10 | РФ | 1000 | 2000 | 5000 | 6000 | 55 | 0.15 | 16 |

Обратное преобразование Фурье передаточной характеристики дает бесконечную последовательность отсчетов импульсной характеристики. В методе окон для получения КИХ-фильтра заданного порядка эта последовательность усекается путем выбора центрального фрагмента нужной длины. Для ослабления паразитных эффектов в этом методе синтеза усеченная импульсная характеристика умножается на весовую функцию (окно), плавно спадающую к краям.

Наибольшей производительностью обладают реализации КИХ-фильтров с параллельной архитектурой. Mega Core FIR Compiler представляет возможность проанализировать влияние преобразования из формата с плавающей запятой в формат

с фиксированной запятой с заданной точностью и влияние величины масштабирования коэффициентов КИХ-фильтра на АЧХ.

Следующий шаг — графический анализ влияния ошибок квантования коэффициентов фильтра на его АЧХ. Ошибки квантования — представление коэффициентов фиксированным набором битов, например 8 бит.

Операции сложения и вычитания в формате с фиксированной запятой не приводят к необходимости округления результатов — они могут лишь вызвать переполнение. В отличие от сложения умножение чисел с фиксированной запятой приводит к увеличению числа значащих цифр результата и, следовательно, к необходимости округления.

Mega Core FIR Compiler позволяет формировать мегафункции FIR-фильтров с архитектурой:

- полностью параллельной;

- полностью последовательной;

- многоразрядной последовательной.

Подробные сведения по реализации фильтров с помощью Mega Core FIR Compiler приведены в руководстве FIR Compiler User Guide

**8.2 Порядок выполнения работы**

8.2.1 Создайте новый проект в среде Quartus II.

8.2.1 Откройте инструмент *Tools >MegaWizard Plug-In Manager*. Выберите в папке DSP инструмент FIR Compiler. На панели Documentation откройте руководство FIR Compiler User Guide.

8.2.3 На панели Parameterize на закладке Edit Coefficient Set задайте исходные данные в соответствии с вариантом задания.

8.2.4 На панели Parameterize задайте выбранные опции реализации фильтра. Подтвердите выбор нажатием кнопки Выбор.

8.2.5 На панели Setup Simulation задайте формирование модели на Verilog HDL.

8.2.6 Запустите процесс генерации мегафункции FIR-фильтра нажатием кнопки Generate.

8.2.7 Включите в проект корневой файл проекта <имя>.qip. Выполните неполную (Analysis and Synthesis) компиляцию проекта.

8.2.8 Сформируйте тесты для проверки мегафункции. Тесты должны обеспечивать проверку реакции фильтра на гармонические сигналы в полосе пропускания и подавления и определение импульсной характеристики. Выполните функциональное моделирование.

**8.3 Содержание отчета**

Отчет должен содержать:

- исходные данные;

- АЧХ и импульсную характеристики фильтра.

- мегафункцию на языке Verilog;

- результаты исследования фильтра в среде симулятора, выводы.

**8.4 Контрольные вопросы**

Приведите примеры мегафункций DSP, которые могут быть построены с помощью инструмента MegaWizard Plug-In Manager.

Какие исходные данные необходимы для построения КИХ-фильтра с помощью инструмент FIR Compiler.

Перечислите доступные в FIR Compiler архитектуры КИХ-фильтров и поясните, чем они отличаются.

Поясните функциональное назначение сигнальных линий внешнего интерфейса сформированной мегафункции.

Поясните результаты тестирования мегафункции.

**9 Работа со встроенным логическим анализатором**

**9.1 Описание работы с кратким изложением теории**

Завершающим этапом процесса разработки проектов на ПЛИС является верификация проекта на физической модели, или системная отладка, то есть отладка на плате в окружении реальных устройств, с которыми взаимодействует ПЛИС. Системная отладка позволяет проверить работоспособность устройства при подаче на него реальных сигналов, устранить ошибки проекта. Часто исследования на физической модели в реальном времени являются единственным средством верификации системы, так как имеется ряд причин, ограничивающих возможности верификации средствами симулятора.

Эффективным средством системной отладки САПР Quartus II является встраиваемый логический анализатор SignalTap II. Функциональная схема SignalTap II приведена на рис. 14. Встраиваемый в проект логический анализатор SignalTap II представляет собой параметризируемую мафункцию, имеющую специализированный пользовательский интерфейс. Доступ к мегафункции SignalTap II возможен с помощью редактора параметризируемых модулей MegaWizard, или через специализированный пользовательский интерфейс — открытием stp-файла в окне Tools.

SignalTap II позволяет:

- осуществлять запись логических состояний сигналов проекта, используя выбранный внутренний или внешний сигнал для тактирования;

- выбирать сигналы проекта для наблюдения;

- подключаться к САПР Quartus II через JTAG-интерфейс СБИС ПЛ;

- захватывать наблюдаемые сигналы в реальном времени.

Конфигуратор SignalTap II доступен в Quartus II Web Edition при включенной опции TalkBack.

Основные шаги работы с SignalTap II:

- установка и настройка;

- определение момента начала захвата данных (Data Triggering);

- захват данных (Data Capture);

- анализ записанных данных (Data Analysis).

Для открытия нового файла STP существуют два способа:

- в меню Tools выбрать пункт SignalTap II Logic Analyzer;

- создать новый файл через меню File => New => Other Files => SignalTap II File.

В обоих вариантах откроется файл с именем по умолчанию STP1.stp.

Окно файла STP содержит несколько областей, обеспечивающих настройку тестов, их запуск, отображение и хранение результатов:

- область менеджера тестов (Instance Manager);

- область редактирования параметров (Signal Configuration);

- область выбора наблюдаемых сигналов и просмотра временных диаграмм (Waveform Viewer);

- область настройки JTAG (JTAG Chain Configuration);

- область иерархии;

- область сохранения данных.

Для создания нового теста следует щелкнуть правой кнопкой в поле области менеджера тестов и выбрать Create Instance. Создаваемым тестам по умолчанию присваивается имя auto\_signaltap\_x, которое в дальнейшем может быть заменено разработчиком на содержательное имя. Менеджер тестов позволяет выбрать нужный тест для установок, запуска, отображения, показывает статус каждого теста и занимаемый аппаратный ресурс, указывает на готовность SignalTap II к запуску или необходимость перекомпиляции проекта при изменениях настроек тестов.

В области редактирования параметров (Signal Configuration) можно задать следующие параметры:

- тактовый сигнал для записи отсчетов в память логического анализатора (Sample

Clock);

- количество отсчетов в записываемой выборке (Sample Depth);

- режим формирования захвата (Trigger flow control);

- позицию захвата (Trigger);

- число условий для формирования сигнала захвата (Trigger Level).

Выбор сигнала тактирования осуществляется через Node Finder. При выборе тактового сигнала предпочтение должно быть отдано глобальному сигналу. Запись данных в память логического анализатора осуществляется по каждому положительному перепаду тактового сигнала.

SignalTap II имеет два режима формирования сигнала захвата:

• последовательностный режим (Sequental)— позволяет использовать и комбинировать стандартные условия захвата;

• режим формирования условий на основе конечного автомата (State-based) — позволяет создавать сложные пользовательские условия захвата.

Выбор положения позиции захвата определяет соотношение выводимых отсчетов до позиции Trigger и после нее и осуществляется посредством использования следующих установок:

- Pre — отображаются данные, записанные до момента захвата;

- Center — отображаются данные, записанные как до, так и после момента захвата;

- Post — отображаются данные, записанные после момента захвата.

Средство просмотра временных диаграмм Waveform Viewer предоставляет возможности для выбора записываемых в кольцевой буфер сигналов и формирования условий захвата для заданного количества условий Trigger Level. Также в этой области отображаются записанные в память SignalTap II значения наблюдаемых сигналов.

В колонке Trigger Enable отмечаются сигналы, которые будут участвовать в логике формирования сигнала захвата (Trigger). Количество условий для формирования сигнала захвата (Trigger Level) задается в области редактирования параметров. Простейший способ формирования сигнала захвата (Trigger) в последовательностном режиме — использование базовых условий (Basic), варианты которых предлагаются при щелчке правой кнопкой в колонке Trigger Level соответствующей строки. Каждый Trigger Level выполняется только тогда, когда одновременно выполняются условия, заданные для всех сигналов, участвующих в его формировании.

Сигнал захвата Trigger формируется в случае выполнения любого из активных условий Trigger Level. После осуществления захвата (в соответствии с установленной позицией захвата — Pre, Center или Post) запись данных в буфер SignalTap II прекращается и содержимое буфера становится доступным для анализа.

Для просмотра временных диаграмм после захвата данных нужно перейти на закладку Data. Формат отображаемых данных может быть задан во всплывающем меню, вызываемом щелчком правой кнопки на выбранном сигнале.

Область иерархии (Hierarchy Display) отображает иерархию сигналов, выбранных для просмотра временных диаграмм, и позволяет скрывать любые из них для удобства анализа остальных. Область сохранения данных (Data Log) позволяет записывать тестовые последовательности для дальнейшего их просмотра средствами SignalTap II или преобразования в другие форматы.

После окончания отладки логический анализатор SignalTap II может быть исключен из проекта (пункт меню Assignments => Settings => SignalTap II Logic Analyzer). После удаления SignalTap II из проекта следует сохранить существующую разводку проекта (пункт меню Assignments => Back-Annotate Assignments => Pin, sell, routing and device assignments) и перекомпилировать проект.

Установки всех средств системной отладки, используемых в проекте, отображаются в окне отчета Compilation Report => Analysis & Synthesis => Debug Settings Summary.

**9.2 Порядок выполнения задания**

9.2.1 С среде Quartus II открыть проект процессорного устройства (вариант задания курсового проекта).

9.2.2 Сформировать функциональный тест для проверки выполнения выбранного участка тестовой программы.

9.2.3 Выполнить конфигурацию мегафункции SignalTap II и добавить ее в проект.

9.2.4 Выполнить тестирование процессорного устройства в режимах Pre, Center, Post.

9.2.5 Сделать выводы о корректности работы процессорного устройства.

**9.3 Содержание отчета**

Отчет должен содержать:

- план тестирования;

- пояснения по настройке мегафункции SignalTap II для задачи тестирования;

- результаты тестирования (временные диаграммы);

- выводы.

**9.4 Контрольные вопросы**

Сформулируйте условия контролепригодности устройства.

Поясните особенности обеспечения условий контролепригодности проектов на платформе ПЛИС.

С какой целью используется логический анализатор при отладке цифровых устройств на платформе ПЛИС?

Каким образом выполняется подготовка к работе мегафункции SignalTap II?

Каким образом выполняется регистрация сигналов в режимах режимах Pre, Center, Post?

Что понимают под условиями запуска (захвата) логического анализатора?

**Приложение Б**

**Методические рекомендации к выполнению курсового проекта**

Методику выполнения курсового проекта рассмотрим на примере построения процессорного устройства с исходными данными:

Архитектура – гарвардская;

Операции АЛУ– AND, ADD, ROR;

Операции управления – JMP;

Разрядность данных – 8;

Число ячеек ПЗУ программ – 256;

Емкость ОЗУ данных – 256x8;

Адресация – прямая, регистровая;

Элементная база – ЕР1С3.

Необходимо разработать:

1) Схему электрическую функциональную;

2) HDL – описание процессорного устройства.

**1 Структура процессорного устройства**

Необходимо выполнить развернутый анализ задания и принять решения, определяющие структуру процессорного устройства.

Формат данных 8 разрядов. Оперативная память (ОЗУ) имеет 256 8-разрядных ячеек. Каждое слово данных занимает одну ячейку ОЗУ. Все команды в ЦВУ будут одноадресные, за исключением безадресных команд IN и OUT.

Примем, что будет использована архитектура аккумуляторного типа, то есть один из операндов в двуместных операциях будет находиться в аккумуляторе W. Второй из операндов может находиться в ОЗУ (в операциях с прямой адресацией) или в банке регистров общего назначения (в операциях с регистровой адресацией). Результаты операций сохраняются в аккумуляторе.

Дополним заданный набор операций следующими:

mov W, RAM(Adr) – загрузка в аккумулятор операнда, находящегося в ОЗУ по адресу Adr[7..0] (прямая адресация);

mov W, GPR(Adr) – загрузка в аккумулятор операнда из банка регистров общего назначения по адресу Adr[1..0] (регистровая адресация);

mov RAM(Adr), W – запись содержимого аккумулятора в ОЗУ по адресу Adr (прямая адресация);

mov GPR(Adr), W – запись содержимого аккумулятора в банк регистров общего назначения по адресу Adr[1..0] (регистровая адресация).

Выполним синтез команд. Команда будет иметь два поля:

- поле кода операции - будет занимать старшие разряды кода команды;

- поле параметра команды - будет занимать младшие разряды кода команды.

Поле параметра:

- в командах с прямой адресацией – 8-разрядный адрес ячейки ОЗУ;

- в командах с регистровой адресацией – 2-разрядный адрес регистра;

- в команде перехода – 8-разрядный адрес точки перехода.

С учетом дополнительных операций процессорное устройство должно выполнять 13 команд. Для их кодирования потребуется 4-разрядный код операции Cop[3..0]. Старший разряд будем использовать, чтобы различать команды с загрузкой результата в аккумулятор от прочих команд. Разряды Cop[2..1] будем использовать, чтобы различать команды каждой группы. Младший разряд Cop[0] будем использовать, чтобы различать операции с прямой и регистровой адресацией.

Результаты синтеза команд приведены в таблице 1. В таблице 1 приняты следующие обозначения:

- a – бит, отводящийся под адрес;

- x – бит, значение которого безразлично для данной команды.

Все команды 12-разрядные и занимает одну ячейку ПЗУ программ. ПЗУ состоит из 256 ячеек по 12 разрядов каждая.

Таблица 1

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Мнемоника | Адресация | Код операции |
| W <- RAM(Adr) | MOV W,M | Прямая | 0000 aaaaaaaa |
| W<- GPR(Adr) | MOV W,R | Регистровая | 0001 xxxxxxaa |
| W <- W and RAM(Adr) | AND M | Прямая | 0010 aaaaaaaa |
| W <- W and GPR(Adr) | AND R | Регистровая | 0011 xxxxxxaa |
| W <- ror RAM(Adr) | ROR | Прямая | 0100 aaaaaaaa |
| W <- ror GPR(Adr) | ROR | Регистровая | 0101 xxxxxxaa |
| W <- W add RAM(Adr) | ADD M | Прямая | 0110 aaaaaaaa |
| W <- W add GPR(Adr) | ADD R | Регистровая | 0111 xxxxxxaa |
| W <- IPORT | IN | – | 1000 xxxxxxxx |
| OPORT <- W | OUT | - | 1100 xxxxxxxx |
| RAM(Adr) <- W | MOV M,W | Прямая | 1010 aaaaaaaa |
| GPR(Adr) <- W | MOV R,W | Регистровая | 1011 xxxxxxaa |
| Перейти по Adr | JMP | - | 111x aaaaaaaa |

Для выборки команд из ПЗУ программ емкостью 256 х 12 необходимы

- программный счетчик PCtr (8 разрядов);

- регистр команд IR (4 разряда).

В соответствии с вариантом задания для хранения данных необходимо использовать ОЗУ емкостью 256 х 8 и банк регистров общего назначения GPR (четыре 8- разрядных регистра). Будем использовать ОЗУ с раздельными входами и выходами данных.

Операции над словами информации будет выполнять операционный автомат, состоящий из АЛУ, аккумулятора и триггеров для хранения признаков операций (флагов):

- ZF – флаг нулевого результата;

- CF – флаг переноса.

В процессорном устройстве будет использоваться гарвардская архитектура с раздельными адресными пространствами памяти программ и данных. Для обмена данными между ОЗУ, банком регистров и операционным автоматом, а также передачи кода операции из ПЗУ программ в регистр команд будут использоваться раздельные шины данных. Для передачи адресов ячеек памяти программ и данных, а также адресов регистров будут использоваться раздельные шины адресов. Устройство управления реализуем в виде автомата Мура. Схема процессорного устройства, построенная в соответствии с принятыми решениями, приведена на рисунке 1.

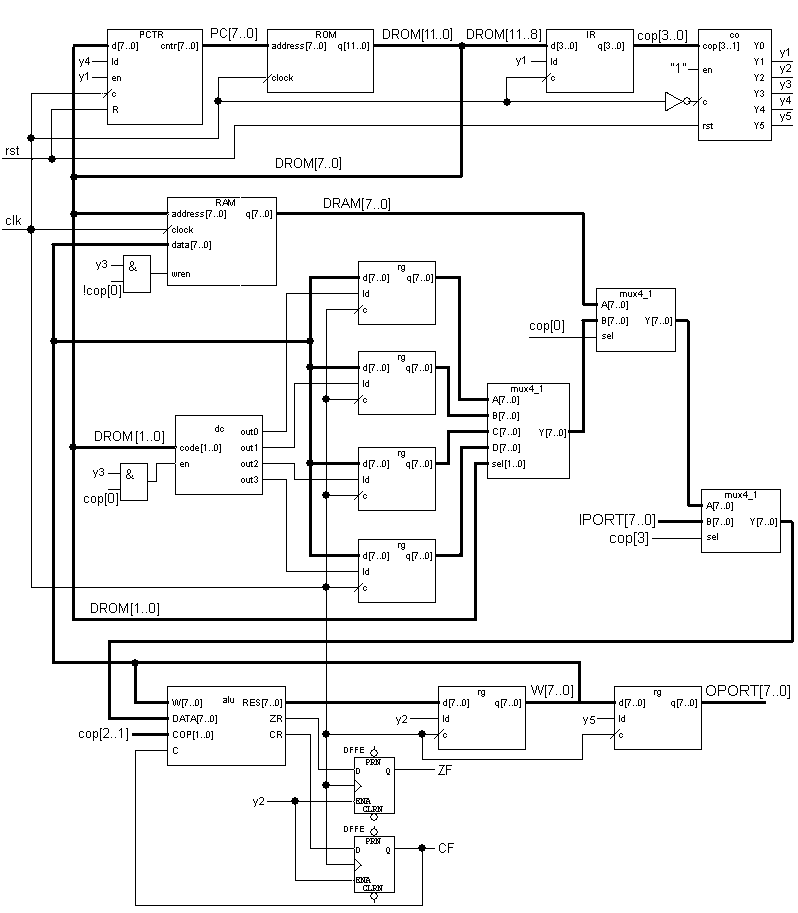


Рисунок 1

Адрес PC[7..0] выбираемой из ПЗУ команды определяется содержимым программного счетчика PCtr. Четыре старших разряда считанного слова DROM[11..8] загружаются в регистр команд IR и представляют код текущей операции Cop[3..0]. Разряды Cop[3..1] кода операции анализируется управляющим автоматом Co. Управляющий автомат формирует последовательность наборов микроприказов y1..y5, необходимых для выполнения текущей операции и подготовки к выборке следующей операции.

Восемь младших разрядов слова DROM[7..0] в командах с прямой адресацией определяют адрес ячейки памяти данных. В командах с регистровой адресацией адрес регистра определяется разрядами DROM[1..0].

Восемь младших разрядов слова DROM[7..0] в команде перехода JMP определяют адрес точки перехода.

Для устранения состязаний сигналов в процессорном устройстве использована двухфазная синхронизация:

- синхроимпульсы Clk от внешнего генератора поступают на синхровходы всех функциональных узлов, кроме управляющего автомата Co;

- синхронизация управляющего автомата Co выполняется инвертированной импульсной последовательностью !Clk.

Для инициализации процессорного устройства используется сигнал Rst, который сбрасывает программный счетчик и переводит в состояние инициализации управляющий автомат.

**2 Микропрограммный автомат**

Функция микропрограммного автомата состоит в анализе разрядов кода операции Cop[3..1] и формировании последовательности наборов микроприказов y1..y5, необходимых для выполнения текущей операции и подготовки к выборке следующей операции. Действие микроприказов отображено на функциональной схеме процессорного устройства.

При включении и рестарте процессорного устройства микропрограммный автомат будет находиться в состоянии idle. Для обеспечения возможности пошагового (покомандного) исполнения программы автомат сможет перейти в последующие состояния только при наличии сигнала разрешения en. При аппаратной отладке процессорного устройства такой сигнал необходимо формировать при нажатии кнопки «Шаг». При автоматическом выполнении программы на входе en постоянно должен действовать активный уровень.

Исполнение команды состоит из двух фаз:

- выборка команды;

- выполнение операции.

Выборка команды выполняется за один такт, при этом:

- текущее содержимое PCtr уже предварительно загружено в регистр адреса ПЗУ;

- код операции, считанный из ПЗУ, загружается в регистр команд IR, а параметр команд с прямой адресацией по шине адреса загружается в регистр адреса ОЗУ;

- выполняется инкремент PCtr для выборки следующей команды.

Для работы процессорного устройства необходимо формировать следующие микрокоманды:

- y1 – выборка команды и инкремент адреса;

- y2 – загрузка аккумулятора;

- y3 – сохранение результата в памяти или регистрах;

- y4 – загрузка адреса перехода в PCtr;

- y5 – вывод.

Поставим в соответствие каждой микрокоманде состояние автомата Мура:

- fetch – выборка команды и инкремент адреса;

- load – загрузка аккумулятора;

- store – сохранение результата в памяти или регистрах;

- jmp – загрузка адреса перехода в PCtr

- out – вывод в порт OPORT.

Микропрограмма автомата с отметкой внутренних состояний приведена на рисунке 2. Все команды выполняются за три такта.

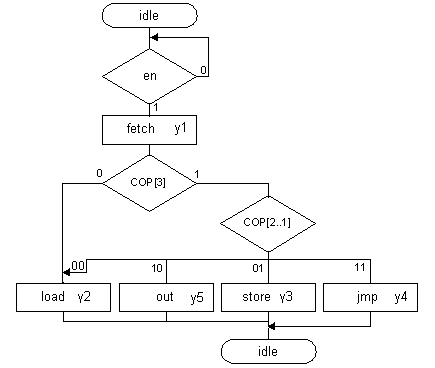


Рисунок 2 – Микропрограмма автомата

Введем переменную текущего состояния автомата code. Тогда с учетом принятых решений управляющий автомат можно описать на языке Verilog:

module FSM (

input clock,

input reset,

input en,

input [3:1] cop,

output [5:1] outY

);

parameter pIDLE = 3'b000;

parameter pFETCH1 = 3'b001;

parameter pLOAD = 3'b011;

parameter pSTORE = 3'b100;

parameter pOUT = 3'b101;

parameter pJMP = 3'b110;

reg [2:0] state, next\_state;

reg [5:1] Y;

// обслуживание регистра состояния

always @ (posedge clock or posedge reset)

if (reset) // reset strategy

state <= pIDLE;

else

state <= next\_state;

// обслуживание комбинаторной логики

always @ (cop, en, state) begin

next\_state = state;

case (state)

pIDLE: begin

Y = 5'b00000;

if (en)

next\_state = pFETCH;

end

pFETCH: begin

Y = 5'b00001;

if (~cop[3])

next\_state = pLOAD;

else begin

case (cop[2:1])

2'b00: next\_state=pHALT;

2'b01: next\_state=pSTORE;

2'b10: next\_state=pJMP;

2'b11: next\_state=pJMP;

endcase

end

end

pLOAD: begin

Y = 5'b00010;

next\_state = pIDLE;

end

pOUT: begin

Y = 5'b10000;

next\_state = pIDLE;

end

pSTORE: begin

Y = 5'b00100;

next\_state = pIDLE;

end

pJMP: begin

Y = 6'b01000;

next\_state = pIDLE;

end

endcase

end

assign outY = Y;

endmodule

**3** **Разработка функциональных блоков и узлов**

**3.1 АЛУ**

АЛУ – блок, выполняющий арифметические операции и логические преобразования над операндами.

Введем обозначения:

- W[], DATA[] – входы операндов;

- C – вход переноса;

- СOP[1..0] – код операции АЛУ;

- CR – выход переноса;

- ZR – выход признака нулевого результата;

- R[] – выход результата.

Выполняемую операцию определяет код операции АЛУ COP[1..0] (таблица 2).

Таблица 2

|  |  |  |
| --- | --- | --- |
| СОР[1..0] | Мнемоника | Операция |
| 00 | MOV | R[] = DATA[]; Z = (R[]= =0); |
| 01 | AND | R[] = W[] & DATA[]; Z = (R[]= =0); |
| 10 | ROR | (CR,R[])=(W[0],C,W[7..1]); Z = (R[]= =0); |
| 11 | ADD | (CR,R[])=(b"0",W[])+(b"0",DATA[]); Z=(R[]==0); |

Функциональное описание модуля АЛУ:

module ALU (

op,

a,

b,

res,

cin,

cout,

zout

);

input [1:0] op; // ALU Operation

input [7:0] a; // 8-bit Input a

input [7:0] b; // 8-bit Input b

input cin;

output [7:0] res; // 8-bit Output

output cout;

output zout;

// Reg declarations for outputs

reg cout;

reg zout;

reg [7:0] res;

// Internal declarations

reg addercout; // Carry out straight from the adder itself.

parameter ALUOP\_MOV = 2'b00;

parameter ALUOP\_AND = 2'b01;

parameter ALUOP\_RCR = 2'b10;

parameter ALUOP\_ADD = 2'b11;

always @(a or b or cin or op) begin

case (op)

ALUOP\_MOV: {addercout, res} = {1'b0, b};

ALUOP\_AND: {addercout, res} = {1'b0, a & b};

ALUOP\_RCR: {addercout, res} = {a[0], cin, a[7:1]};

ALUOP\_ADD: {addercout, res} = a + b;

default: {addercout, res} = {1'b0, 8'h00};

endcase

end

always @(res)

zout = (res == 8'h00);

always @(addercout or op)

// if (op == ALUOP\_SUB)

// cout = ~addercout; // Invert adder's carry to get borrow

// else

cout = addercout;

endmodule

**3.2 Регистры**

В состав ЦВУ входят 8-разрядные регистры общего назначения Rg и 4-разрядный регистр команд IR. Работу регистров можно описать в форме таблицы режимов (таблица 3).

Таблица 3 – Режимы работы регистра

|  |  |  |  |
| --- | --- | --- | --- |
| Режим работы | С | LD | Q+[ 7..0] |
| Загрузка | ↑ | 1 | D[7..0] |
| Хранение | x | 0 | Q[7..0] |

Функциональное описание регистра Rg:

module rg (

input wire clk,

input wire [7:0]di,

input wire ld,

output reg [7:0]q

);

always @ (posedge clk)

if (ld)

q <= d;

**3.3 Программный счетчик**

Для ЦВУ необходим программный счетчик PCtr с возможностью загрузки адреса команды. Режимы работы программного счетчика с загрузкой приведены в таблице 4.

Таблица 4 – Режимы работы счетчика

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Режим работы | clk | Y[4] | Y[0] | Q+[ 7..0] |
| Загрузка | ↑ | 1 | x | D[7..0] |
| Инкремент | ↑ | 0 | 1 | Q[7..1] + 1 |
| Хранение | х | 0 | 0 | Q[7..0] |

Функциональное описание модуля PCtr:

reg [7:0] PC;

always @(posedge clk, negedge res\_b)

if(!res\_b)

PC <= 8'b0;

else if (Y[4])

PC <= DROM[7:0];

else if (Y[1])

PC <= PC + 1'b1;

**3.4 Дешифратор и мультиплексор**

В составе банка регистров общего назначения используются двоичный дешифратор и мультиплексор.

Дешифратор формирует сигналы выборки для регистров банка по двухразрядному адресу регистра при наличии сигнала разрешения. Введем внутренние обозначения сигналов:

- входной код адреса - code[1..0];

- сигнал разрешения - en;

- выходные сигналы - out0,out1,out2,out3.

Мультиплексор «4 в 1» передает данные с одного из регистров банка на выход. Введем внутренние обозначения сигналов:

- входной код адреса - sel[1..0];

- водные сигналы - A[7..0],B[7..0],C[7..0],D[7..0];

- выходной сигнал - Y[7..0].

AHDL-описание модуля mux4\_1:

always @ (posedge clk)

if (Y[1])

AdrRG <= DROM[1:0];

always @ (posedge clk)

if (Y[3]&COP[0])

case (AdrRG)

2'b00 : B <= W;

2'b01 : C <= W;

2'b10 : D <= W;

2'b11 : E <= W;

endcase

always @ (AdrRG or B or C or D or E)

case (AdrRG )

2'b00 : GPR = B;

2'b01 : GPR = C;

2'b10 : GPR = D;

2'b11 : GPR = E;

endcase

**3.5 Модули памяти**

Модули ОЗУ данных и ПЗУ программ реализуем на основе библиотечной параметризируемой мегафункции altsyncram. Настройка мегафункции выполнена средствами инструмента MegaWizard Plug-In Manager инструментальной системы Qartus II. Описание ОЗУ емкостью 256х8 на языке Verilog:

module RAM (

address,

clken,

clock,

data,

wren,

q);

input [7:0] address;

input clken;

input clock;

input [7:0] data;

input wren;

output [7:0] q;

`ifndef ALTERA\_RESERVED\_QIS

// synopsys translate\_off

`endif

tri1 clken;

tri1 clock;

`ifndef ALTERA\_RESERVED\_QIS

// synopsys translate\_on

`endif

wire [7:0] sub\_wire0;

wire [7:0] q = sub\_wire0[7:0];

altsyncram altsyncram\_component (

.clocken0 (clken),

.wren\_a (wren),

.clock0 (clock),

.address\_a (address),

.data\_a (data),

.q\_a (sub\_wire0),

.aclr0 (1'b0),

.aclr1 (1'b0),

.address\_b (1'b1),

.addressstall\_a (1'b0),

.addressstall\_b (1'b0),

.byteena\_a (1'b1),

.byteena\_b (1'b1),

.clock1 (1'b1),

.clocken1 (1'b1),

.clocken2 (1'b1),

.clocken3 (1'b1),

.data\_b (1'b1),

.eccstatus (),

.q\_b (),

.rden\_a (1'b1),

.rden\_b (1'b1),

.wren\_b (1'b0));

defparam

altsyncram\_component.address\_aclr\_a = "NONE",

altsyncram\_component.indata\_aclr\_a = "NONE",

altsyncram\_component.init\_file = "RAM.mif",

altsyncram\_component.intended\_device\_family = "Cyclone",

altsyncram\_component.lpm\_hint = "ENABLE\_RUNTIME\_MOD=NO",

altsyncram\_component.lpm\_type = "altsyncram",

altsyncram\_component.numwords\_a = 256,

altsyncram\_component.operation\_mode = "SINGLE\_PORT",

altsyncram\_component.outdata\_aclr\_a = "NONE",

altsyncram\_component.outdata\_reg\_a = "UNREGISTERED",

altsyncram\_component.power\_up\_uninitialized = "FALSE",

altsyncram\_component.widthad\_a = 8,

altsyncram\_component.width\_a = 8,

altsyncram\_component.width\_byteena\_a = 1,

altsyncram\_component.wrcontrol\_aclr\_a = "NONE";

endmodule

Полученное описание ПЗУ емкостью 256х12 на языке Verilog:

module ROM (

address,

clken,

clock,

q);

input [7:0] address;

input clken;

input clock;

output [11:0] q;

`ifndef ALTERA\_RESERVED\_QIS

// synopsys translate\_off

`endif

tri1 clken;

tri1 clock;

`ifndef ALTERA\_RESERVED\_QIS

// synopsys translate\_on

`endif

wire [11:0] sub\_wire0;

wire [11:0] q = sub\_wire0[11:0];

altsyncram altsyncram\_component (

.clocken0 (clken),

.clock0 (clock),

.address\_a (address),

.q\_a (sub\_wire0),

.aclr0 (1'b0),

.aclr1 (1'b0),

.address\_b (1'b1),

.addressstall\_a (1'b0),

.addressstall\_b (1'b0),

.byteena\_a (1'b1),

.byteena\_b (1'b1),

.clock1 (1'b1),

.clocken1 (1'b1),

.clocken2 (1'b1),

.clocken3 (1'b1),

.data\_a ({12{1'b1}}),

.data\_b (1'b1),

.eccstatus (),

.q\_b (),

.rden\_a (1'b1),

.rden\_b (1'b1),

.wren\_a (1'b0),

.wren\_b (1'b0));

defparam

altsyncram\_component.address\_aclr\_a = "NONE",

altsyncram\_component.init\_file = "ROM.mif",

altsyncram\_component.intended\_device\_family = "Cyclone",

altsyncram\_component.lpm\_hint = "ENABLE\_RUNTIME\_MOD=NO",

altsyncram\_component.lpm\_type = "altsyncram",

altsyncram\_component.numwords\_a = 256,

altsyncram\_component.operation\_mode = "ROM",

altsyncram\_component.outdata\_aclr\_a = "NONE",

altsyncram\_component.outdata\_reg\_a = "UNREGISTERED",

altsyncram\_component.widthad\_a = 8,

altsyncram\_component.width\_a = 12,

altsyncram\_component.width\_byteena\_a = 1;

endmodule

Значения параметра init\_file "Rom.mif" и "Ram.mif" указывают на файлы инициализации памяти (контент модулей памяти).

**4 Тестирование процессорного устройства**

**4.1 Компиляция и верификация проекта в среде Quartus II**

В приложении Б приведено AHDL-описание процессорного устройства верхнего иерархического уровня, сформированное в соответствие с функциональной схемой.

ЦВУ должно выполнять программу, хранящуюся в ПЗУ. Загрузочный код программы необходимо сформировать в виде файла ROM.mif. Вариант тестовой программы для ЦВУ включает циклическое выполнение семи команд:

DEPTH = 256; % Memory depth and width are required %

WIDTH = 12; % Enter a decimal number %

ADDRESS\_RADIX = HEX; % Address and value radixes are optional %

DATA\_RADIX = HEX;

CONTENT

BEGIN

00 : 000; % mov w,M(0) %

01 : B01; % mov Rg(1) %

02 : 601; % add M(1) %

03 : A02; % mov M(2),w %

04 : 400; % ror %

05 : 301; % and Rg(1) %

06 : A03; % mov M(3),w %

07 : C00; % jmp 0 %

[08..FF]: 000;

END ;

Чтобы на стадии компиляции выполнить загрузку данных в ячейки ОЗУ, необходимо сформировать файл загрузки RAM.mif:

DEPTH=256;

WIDTH=8;

ADDRESS\_RADIX=HEX;

DATA\_RADIX=HEX;

CONTENT

BEGIN

00 : FB; % %

01 : 0D; % %

02 : 02; % %

03 : 03; % %

04 : 04; % %

05 : 05; % %

[06..FF]: 00; % %

END ;

Для данного варианта тестирования исполнение команд над заданными операндами должно привести к получению следующих результатов:

- 00 W ← FB;

- 01 Rg1 ← FB;

- 02 W ← 08; CF;

- 03 RAM(2) ← 08; CF;

- 04 W ← 84;

- 05 W ← 80;

- 06 RAM(3) ← 80;

- 07 переход на 00.

Для проверки правильности работы процессорного устройства при выполнении тестовой программы выполняем анализ его работы методом имитационного моделирования в среде симулятора системы Quartus II с помощью сигнального редактора (Waveform Editor).

Анализ результатов моделирования позволяет оценить правильность работы проекта. На рисунках 3 и 4 приведены полученные временные диаграммы основных сигналов. Анализ этих результатов позволяет сделать вывод, что данную тестовую программу ЦВУ выполняет правильно.

Полученная оценка предельной частоты синхронизации для процессорного устройства составляет 49,11 МГц. Для построения ЦВУ на кристалле ПЛИС компилятор системы Quartus II использовал 111 программируемых логических элементов (4 % от имеющихся на кристалле) и 5120 бит модулей памяти (9 % от имеющихся на кристалле).

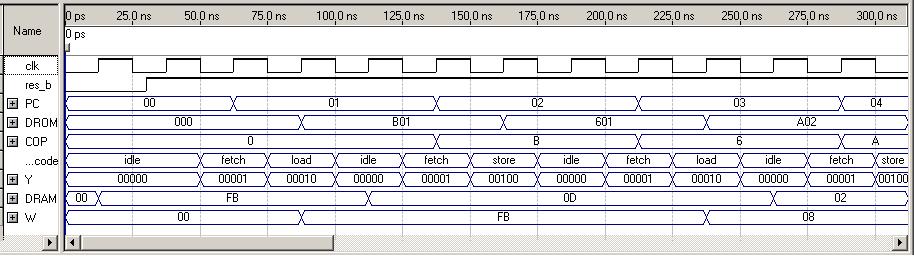


Рисунок 3

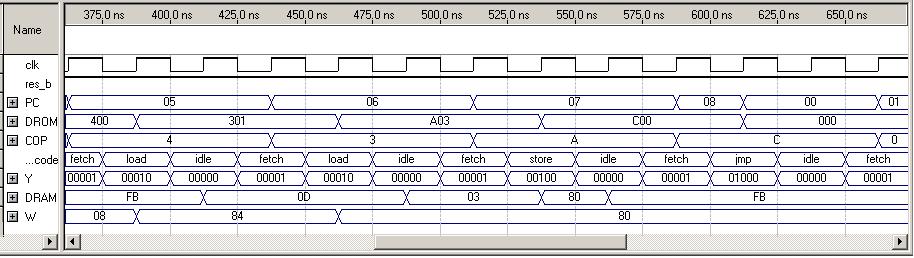


Рисунок 4

**4.2 Тестирование на стенде SDK6.1**

Для проверки процессорного устройства на стенде SDK 6.1 необходимо дополнить его схему функциональными узлами, необходимыми для обеспечения основных условий контролепригодности: наблюдаемости и управляемости.

Условие наблюдаемости обеспечивается путем вывода выходных сигналов важнейших узлов на индикаторы (символьный жидкокристаллический и светодиодные). Условие управляемости можно реализовать путем пошагового исполнения программы. Для этого необходимо формировать сигнал разрешения en для управляющего автомата при нажатии на кнопку «Шаг». Доработку схемы процессорного устройства предлагается выполнить самостоятельно.

В процессе компиляции проекта формируется файл с расширением «.rbf», который можно загрузить в ПЛИС стенда и проверить процессорное устройство при выполнении тестовой программы.

Сделать выводы о правильности работы устройства.