518(07)

Л851

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение высшего образования**

**«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

И.И. Левин

Б.Е. Механцев

**ЛАБОРАТОРНЫЕ РАБОТЫ ПО ДИСЦИПЛИНЕ**

**«ПЛИС-ТЕХНОЛОГИИ И МЕТОДЫ СОЗДАНИЯ**

**ЭФФЕКТИВНЫХ ПРИКЛАДНЫХ ПРОГРАММ ДЛЯ ПЛИС»**

|  |
| --- |
| Кафедра интеллектуальных имногопроцессорных систем |

Учебное пособие



**Таганрог**

**Издательство Южного федерального университета**

**2017**

УДК 004.432

Рецензенты:

*д.ф.-м. н. А.В. Никитина, Южный федеральный университет,*

*к.т.н. Семерников Е.А., НИЦ СЭ и НК*

**Левин И.И., Механцев Б.Е.** Лабораторные работы по дисциплине «ПЛИС-технологии и методы создания эффективных прикладных программ для ПЛИС»: Учеб. пособие. - Таганрог: Издательство ЮФУ, 2017. - 35 с.

*В данном пособии описаны современные принципы программирования многопроцессорных вычислительных систем гибридного типа: представлена параллельно-конвейерная форма представления решения прикладной задачи, необходимая для обеспечения принципа ресурсонезависимого программирования, приводятся описания и примеры использования основных видов редукции параллельной программы, написанной на языке программирования высокого уровня COLAMO, что позволяет оптимально задействовать (использовать) аппаратные ресурсы вычислительной системы гибридного типа, а также описываются программные средства разработки прикладных программ для вычислительных систем гибридного типа. Программные средства представляют собой программный комплекс, состоящий из четырех основных программных модулей: модуля средств трансляции, модуля средств синтеза масштабируемых параллельно-конвейерных решений прикладных программ, модуля средств управления и синхронизации и модуля мониторинга. Приводится описание каждого программного модуля с описанием его функциональных возможностей, а также описания входных и выходных данных и схема взаимодействия модулей между собой. В заключение главы приведен пример практического использования программного комплекса на примере решения тестовой прикладной задачи на вычисли-тельной системе гибридного типа.*

*Учебное пособие предназначено для студентов, магистрантов и аспирантов по направлению 09.06.01 «Прикладная математика и информатика», магистрантов по направлению 01.04.02 «Информатика и вычислительная техника», а также других направлений при проведении практических и лекционных занятий*

Ил. 10. Библиогр. 7 назв.

© Южный федеральный университет, 2017

© Левин И.И., 2017

© Механцев Б.Е., 2017

Оглавление

[Введение 4](#_Toc438816991)

[Лабораторная работа №1. Разработка комбинационной схемы 5](#_Toc438816992)

[Лабораторная работа №2. Синхронные схемы. Разработка счетчика. 17](#_Toc438816993)

[Лабораторная работа №3. Реализация конечного автомата 26](#_Toc438816994)

[Лабораторная работа №4. Реализация АЛУ с расширением 32](#_Toc438816995)

[Рекомендуемая литература 36](#_Toc438816996)

Дополнительные задания………………………………………………………..34

# Введение

Лабораторные работы относятся к изучению дисциплины «ПЛИС-технологии и методы создания эффективных прикладных программ для ПЛИС».

Методические указания содержат сведения, необходимые для выполнения цикла лабораторных работ по изучению методов реализации цифровых устройств на программируемых логических интегральных схемах (ПЛИС). Первые лабораторные работы посвящены изучению разработки на ПЛИС и основам реализации логических функций. Третья лабораторная работа связана с изучением вопросов реализации конечных автоматов, четвертая - с реализацией выполнения арифметических операций.

Целью выполнения цикла лабораторных работ является формирование профессиональной компетентности разработчика цифровых систем на ПЛИС.

Задачи лабораторных работ:

- освоение полного цикла разработки простой комбинационной схемы с использованием САПР Vivado;

- освоение логического проектирования синхронных схем на примере счетчиков;

- освоение разработки конечных автоматов и реализации таковых в ПЛИС;

- освоение реализации в ПЛИС простого арифметико-логического устройства.

В результате выполнения цикла лабораторных работ магистрант получает практические знания об организации современных ПЛИС, о цикле разработки устройства на основе ПЛИС, об особенностях реализации логических функций на ПЛИС, о методике работы с современными САПР.

# Лабораторная работа №1. Разработка комбинационной схемы

**Цель работы**

Получение исходных навыков проведения разработки в Vivado с использованием VHDL.

**Этапы лабораторной работы**

1. Формирование логического описания схемы.
2. Кодировка описания в VHDL.
3. Ввод и проверка VHDL текста в систему Vivado как нового

проекта.

1. Проверка описания схемы.
2. Разработка тестовых сигналов в системе Vivado.
3. Логический анализ и принятие решения о завершении

формирования описания схемы.

**Исходные задания для работы**

Разработать схему с 64 входами, реагирующую на заданные входные воздействия.

На выходе селектора разместить триггер, тактируемый внешним сигналом clk.

Использовать структурный способ разработки.

**Последовательность выполнения работы**

1. Составление логической схемы.
2. Составление VHDL- описания необходимых компонент и сборки.
3. Составление описания схемы VHDL- в VIVADO, подключение

внешних контактов.

1. Проверка синтаксиса.
2. Синтез и имплементация.
3. Логическое моделирование.
4. Составление отчета.

Отчет должен включать в себя:

- задание;

- схему в виде эскиза;

- VHDL-описание;

- результаты моделирования.

|  |  |  |
| --- | --- | --- |
| № | Реакция схемы | Входные данные  Min Max |
| 1 | «0» если, иначе «1» | 11111111A00020 11111111А00040 |
| 2 | «0» если, иначе «1» | 11111111A00021 11111111А00041 |
| 3 | «0» если, иначе «1» | 11111111A00022 11111111А00042 |
| 4 | «0» если, иначе «1» | 11111111A00023 11111111А00043 |
| 5 | «0» если, иначе «1» | 11111111A00024 11111111А00044 |
| 6 | «0» если, иначе «1» | 11111111A00025 11111111А00045 |
| 7 | «0» если, иначе «1» | 11111111A00026 11111111А00046 |
| 8 | «0» если, иначе «1» | 11111111A00027 11111111А00047 |
| 9 | «0» если, иначе «1» | 11111111A00028 11111111А00048 |
| 10 | «0» если, иначе «1» | 11111111A00029 11111111А00049 |
| 11 | «0» если, иначе «1» | 11111111A0002A 11111111А0004A |
| 12 | «0» если, иначе «1» | 11111111A0002B 11111111А0004B |
| 13 | «0» если, иначе «1» | 11111111A0002C 11111111А0004C |
| 14 | «0» если, иначе «1» | 11111111A0002D 11111111А0004D |
| 15 | «0» если, иначе «1» | 11111111A0002E 11111111А0004E |
| 16 | «0» если, иначе «1» | 11111111A0002F 11111111А0004F |
| 17 | «0» если, иначе «1» | 11111111A00030 11111111А00050 |
| 18 | «0» если, иначе «1» | 11111111A00021 11111111А00051 |
| 19 | «0» если, иначе «1» | 11111111A00032 11111111А00052 |
| 20 | «0» если, иначе «1» | 11111111A00033 11111111А00053 |
| 21 | «0» если, иначе «1» | 11111111A00034 11111111А00054 |
| 22 | «0» если, иначе «1» | 11111111A00035 11111111А00055 |
| 23 | «0» если, иначе «1» | 11111111A00036 11111111А00056 |
| 24 | «0» если, иначе «1» | 11111111A00037 11111111А00057 |
| 25 | «0» если, иначе «1» | 11111111A00038 11111111А00058 |
| 26 | «0» если, иначе «1» | 11111111A00039 11111111А00059 |
| 27 | «0» если, иначе «1» | 11111111A0003A 11111111А00060 |
| 28 | «0» если, иначе «1» | 11111111A0003B 11111111А00061 |
| 29 | «0» если, иначе «1» | 11111111A0003C 11111111А00062 |
| 30 | «0» если, иначе «1» | 11111111A0003D 11111111А00063 |
| 31 | «0» если, иначе «1» | 11111111A0003E 11111111А00040 |
| 32 | «0» если, иначе «1» | 11111111A0003F 11111111А00040 |

**Разработанная схема выдает однобитное сообщение на выход, если многобитный входной сигнал принимает значение из заданного диапазона.**

**Рассмотрим последовательность разработки логического описания простого комбинационного устройства.**

**Выход устройства должен принимать значение «1» только в случаях, когда входное слово A0, A1, A2 примет значения 010 или 011 (заметим, что зависимость от состояния бита A2 отсутствует). Можно записать соотношение Out <= not(A0) or A1, полностью описывающее его функционирование. Впрочем, возможности VHDL позволяют записать выражение, например так:**

**out <= '1' when (a0 = '0' and a1 = '1') else '0';**

**Оба эти выражения равнозначны для дальнейших действий. Второй вариант даже более соответствует использованию входов элемента LUT, представляющего собой схему статической памяти, синтезирующий заданную таблицу истинности.**

VHDL-описание схемы для этой лабораторной работы необходимо провести структурным способом, применяя в качестве библиотечных логических элементов логические блоки, реализующие необходимую функцию. Например, четырехвходовая схема НЕ-И с инверсией по трем входам будет выдавать «1» только при наличии на входах группы 0001:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity not3and is

Port ( a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : in STD\_LOGIC;

d : in STD\_LOGIC;

e : out STD\_LOGIC);

end not3and;

architecture Behavioral of not3and is

begin

E<= not(a) and not(b) and not(c) and d;

end Behavioral;

Аналогично проводится разработка следующих логических блоков (тех, которые нужны для реализации заданной схемы). Сборка использует сначала описание портов, а затем подключение к сигнальным линиям подготовленных элементов, например:

component not3and

port(a,b,c,d:in STD\_LOGIC; E:out STD\_LOGIC);

end component;

--Описание портов entity not3and

dd1a: entity work.not3and(not3and) port map(sou\_addr(63),sou\_addr(62),sou\_addr(61), sou\_addr(60), so0);

--ловим группу 0001

--Описание подключения компонента not3and

Привязка портов к внешним выводам (подключение контактов), как правило, согласуется с разработчиками печатной платы (рисунок 1). В файле конструкторских ограничений .XDS эта привязка осуществляется следующим образом:

**set\_property PACKAGE\_PIN u16 [get\_ports b];**

К внешнему контакту u16 должен быть подключен порт b.

Самостоятельную привязку внешних портов схемы Vivado осуществляет на этапе синтеза (synthesis). После синтеза можно посмотреть привязку портов и при необходимости скорректировать её. Для этого в верхнем меню выбирается I/O Planning, в нижней части экрана открывается блок I/O Pоrts. В этом блоке можно переопределить привязку портов к внешним выводам схемы (рисунок 2).



Рисунок 1. Выбор режима работы с внешними портами

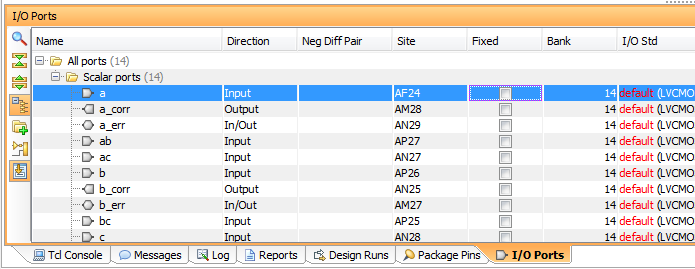


Рисунок 2. Блок I/O Pоrts для просмотра и переопределения соответствия портов внешним выводам

После того как выходной порт будет полностью определен по привязке, следует поставить флаг в поле «Fixed».

В файле необходимо указать описание электрических параметров портов, например:

**set\_property IOSTANDARD LVCMOS18 [all\_inputs]**

**set\_property IOSTANDARD LVCMOS18 [all\_outputs]**

**Можно указать желаемую тактовую частоту директивой**

**create\_clock -name CLK -period 1.6 [get\_ports {clk}]**

**После имплементации в окне Timing можно увидеть, возможна ли работа схемы на заданной частоте.**

**Моделирование схемы**

На всех этапах разработки моделирования необходимо использование файла тестирования, в котором разработанная схема «подключается» к тестирующим сигналам. Структура VHDL-файла для моделирования выглядит так:

Общие ссылки и декларации, например:

**-- VHDL Test Bench for functional and timing simulation**

**-- ссылки на используемые библиотеки и пакеты**

**LIBRARY IEEE;**

**USE IEEE.std\_logic\_1164.all;**

**-- описание интерфейса объекта**

**ENTITY testbench IS**

**END testbench;**

**-- описание архитектуры объекта**

**ARCHITECTURE testbench\_arch OF testbench IS**

**-- декларация компонента, представляющего модуль описания**

**--верхнего уровня иерархии проекта**

**--Порты разработанного устройства:**

**COMPONENT jc2\_top**

**PORT (**

**CLK : in STD\_LOGIC;**

**LEFT : in STD\_LOGIC;**

**RIGHT : in STD\_LOGIC;**

**STOP : in STD\_LOGIC;**

**Q: inout STD\_LOGIC\_VECTOR (3 DOWNTO 0)**

**);**

**END COMPONENT;**

**--Сигналы, используемые для тестирования:**

**SIGNAL CLK : STD\_LOGIC;**

**SIGNAL LEFT : STD\_LOGIC;**

**SIGNAL RIGHT : STD\_LOGIC;**

**SIGNAL STOP : STD\_LOGIC;**

**SIGNAL Q : STD\_LOGIC\_VECTOR (3 DOWNTO 0);**

**--Подключение сигналов к портам:**

**BEGIN**

**UUT : jc2\_top**

**PORT MAP (**

**LEFT => LEFT,**

**RIGHT => RIGHT,**

**STOP => STOP,**

**CLK => CLK,**

**Q => Q**

**);**

**--Описание тестовых сигналов:**

**PROCESS**

**BEGIN**

**-- --------------------**

**CLK <= transport '0';**

**LEFT <= transport '1';**

**RIGHT <= transport '1';**

**STOP <= transport '1';**

**-- --------------------**

**WAIT FOR 110 ns;**

**CLK <= transport '1';**

**-- --------------------**

**WAIT FOR 10 ns;**

**CLK <= transport '0';**

**-- --------------------**

**WAIT FOR 10 ns;**

**CLK <= transport '1';**

**-- --------------------**

**...**

**-- --------------------**

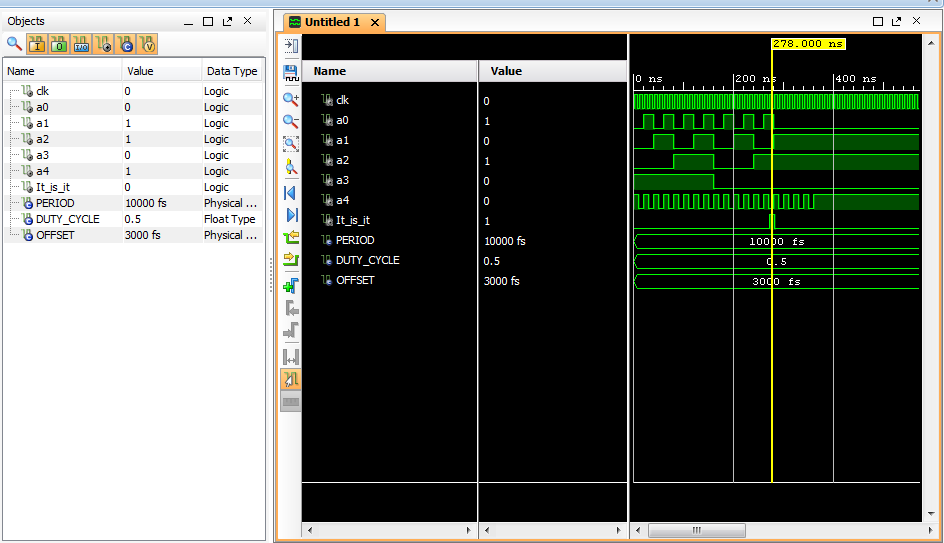
**WAIT;**

**END PROCESS;**

**--И наконец**

**END testbench\_arch;**

**После первичной отладки файлов описания устройства и данных для моделирования можно запустить simulation и получить временные диаграммы работы устройства, например в виде, отображенном на рисунке 3.**



**Рисунок 3. Временные диаграммы работы устройства**

**Последовательность** разработки устройства на ПЛИС Xilinx представлена на рисунке 4.

**Рисунок 4. Последовательность** разработки устройства на ПЛИС Xilinx

Реализация комбинационных функций в ПЛИС Xilinx осуществляется на блоках LUT (Look- Up Table). В ПЛИС Xilinx 6-й и 7-й серий эти блоки состоят из элемента памяти с шестью входами и двумя выходами и D-триггеров (рисунок 5).

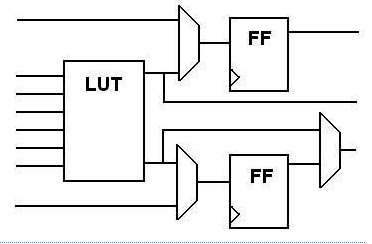


Рисунок 5. Упрощенная схема логической ячейки ПЛИС Xilinx

6-й и 7-й серий

Таким образом, в каждом элементе можно реализовать до двух логических функций с шестью переменными в каждой. Конкретная функциональность каждого LUT осуществляется при программировании микросхемы записью в память таблицы истинности.

Кроме того, современные ПЛИС содержат такие блоки как:

– блоки синхронной статической двухпортовой памяти BRAM;

– блоки цифровой обработки сигналов «умножение с накоплением» XtremeDSP;

– формирователи тактовых сигналов MMCM (Multi-Mode Clock Managers) и PLL (Phase-Locked Loop);

– скоростные последовательные приемопередатчики (MGT, Multi-GigabitTransceivers);

– контроллеры Ethernet MAC (Virtex-4, 5, 6);

– контроллеры PCI Express endpoint;

– процессорные ядра PowerPC (Virtex-II Pro, Virtex-4 FX, Virtex-5 FXT), ARM (Zynq-7000).

Коммутация между блоками осуществляется программируемыми трассировочными линиями, размещенными в горизонтальном и вертикальном направлениях с помощью цифровых ключей. Это позволяет соединить любые ячейки FPGA, замыкая соответствующие ключи. Состояния ключей (замкнуты или разомкнуты) записываются в соответствующие разделы конфигурационной памяти. В процессе проектирования разработчику не требуется задавать состояния ключей и таблицы истинности непосредственно, поскольку это выполняет САПР ПЛИС на основе анализа введенной схемы. Более того, формат конфигурационного потока, загружаемого в FPGA, является закрытым, а его формирование производится исключительно программными средствами компании-производителя.

**Приложение. Пример логического устройства на ПЛИС**

|  |  |
| --- | --- |
| Описание устройства | Описание входных сигналов |
| **library IEEE;**  **use IEEE.STD\_LOGIC\_1164.ALL;**  **entity addr\_sel is**  **Port ( a0 : in STD\_LOGIC;**  **a1 : in STD\_LOGIC;**  **a2 : in STD\_LOGIC;**  **a3 : in STD\_LOGIC;**  **a4 : in STD\_LOGIC;**  **clk : in STD\_LOGIC;**  **It\_is\_it : out STD\_LOGIC);**  **end addr\_sel;**  **architecture Behavioral of addr\_sel is**  **begin**  **It\_is\_it <= (a0 and (not a1)) and (a2 and (not a3)) and (not a4) and clk;**  **end Behavioral;** | **LIBRARY ieee;**  **USE ieee.std\_logic\_1164.ALL;**  **USE ieee.numeric\_std.ALL;**  **use IEEE.STD\_LOGIC\_ARITH.ALL;**  **use IEEE.STD\_LOGIC\_UNSIGNED.ALL;**  **LIBRARY UNISIM;**  **USE UNISIM.Vcomponents.ALL;**  **ENTITY testbench IS**  **END testbench;**  **ARCHITECTURE testbench\_arch OF**  **testbench IS**  **COMPONENT addr\_sel**  **Port ( a0 : in STD\_LOGIC;**  **a1 : in STD\_LOGIC;**  **a2 : in STD\_LOGIC;**  **a3 : in STD\_LOGIC;**  **a4 : in STD\_LOGIC;**  **clk : in STD\_LOGIC;**  **It\_is\_it : out STD\_LOGIC);**  **END COMPONENT;**  **signal clk :std\_logic;**  **signal a0 : STD\_LOGIC;**  **signal a1 : STD\_LOGIC;**  **signal a2 : STD\_LOGIC;**  **signal a3 : STD\_LOGIC;**  **signal a4 : STD\_LOGIC;**  **signal It\_is\_it : STD\_LOGIC;**  **-- \*\*\* Clock Process for CLK - User Defined Section 1 \*\*\***  **constant PERIOD : TIME := 10 ns;**  **constant DUTY\_CYCLE : real := 0.5;**  **constant OFFSET : time := 3 ns;**  **-- \*\*\* END Clock Process for CLK - User Defined Section 1 \*\*\***  **BEGIN**  **UUT: addr\_sel PORT MAP(**  **CLK => CLK,**  **a0 => a0,**  **a1 => a1,**  **a2 => a2,**  **a3 => a3,**  **a4 => a4,**  **It\_is\_it => It\_is\_it);**  **--- \*\*\* Clock Process for CLK - User Defined Section 2 \*\*\***  **clk\_gen: PROCESS**  **BEGIN**  **WAIT for OFFSET;**  **CLOCK\_LOOP : LOOP**  **CLK <= '1';**  **WAIT FOR (PERIOD - (PERIOD \* DUTY\_CYCLE));**  **CLK <= '0';**  **WAIT FOR (PERIOD \* DUTY\_CYCLE);**  **END LOOP CLOCK\_LOOP;**  **END PROCESS;**  **--- \*\*\* END Clock Process for CLK - User Defined Section 2 \*\*\***  **a0\_gen : PROCESS**  **BEGIN**  **a0 <= '0'; WAIT FOR PERIOD\*2;**  **a0 <= '1'; WAIT FOR PERIOD\*2;**  **a0 <= '0'; WAIT FOR PERIOD\*2;**  **a0 <= '1'; WAIT FOR PERIOD\*2;**  **a0 <= '0'; WAIT FOR PERIOD\*2;**  **a0 <= '1'; WAIT FOR PERIOD\*2;**  **a0 <= '0'; WAIT FOR PERIOD\*2;**  **a0 <= '1'; WAIT FOR PERIOD\*2;**  **a0 <= '0'; WAIT FOR PERIOD\*2;**  **a0 <= '1'; WAIT FOR PERIOD\*2;**  **a0 <= '0'; WAIT FOR PERIOD\*2;**  **a0 <= '1'; WAIT FOR PERIOD\*2;**  **a0 <= '0'; WAIT FOR PERIOD\*2;**  **a0 <= '1'; WAIT FOR PERIOD\*2;**  **a0 <= '0'; WAIT FOR PERIOD\*2;**  **WAIT; -- will wait forever**  **END PROCESS;**  **a1\_gen : PROCESS**  **BEGIN**  **a1 <= '0'; WAIT FOR PERIOD \*4;**  **a1 <= '1'; WAIT FOR PERIOD \*4;**  **a1 <= '0'; WAIT FOR PERIOD \*4;**  **a1 <= '1'; WAIT FOR PERIOD \*4;**  **a1 <= '0'; WAIT FOR PERIOD \*4;**  **a1 <= '1'; WAIT FOR PERIOD \*4;**  **a1 <= '0'; WAIT FOR PERIOD \*4;**  **a1 <= '1'; WAIT FOR PERIOD \*4;**  **WAIT; -- will wait forever**  **END PROCESS;**  **a2\_gen : PROCESS**  **BEGIN**  **a2 <= '0'; WAIT FOR PERIOD \*8;**  **a2 <= '1'; WAIT FOR PERIOD \*8;**  **a2 <= '0'; WAIT FOR PERIOD \*8;**  **a2 <= '1'; WAIT FOR PERIOD \*8;**  **WAIT; -- will wait forever**  **END PROCESS;**  **a3\_gen : PROCESS**  **BEGIN**  **a3 <= '1'; WAIT FOR PERIOD\*16;**  **a3 <= '0'; WAIT FOR PERIOD\*16;**  **WAIT; -- will wait forever**  **END PROCESS;**  **a4\_gen : PROCESS**  **BEGIN**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **a4 <= '0'; WAIT FOR PERIOD;**  **a4 <= '1'; WAIT FOR PERIOD;**  **WAIT; -- will wait forever**  **END PROCESS;**  **---DI\_gen : PROCESS**  **---BEGIN**  **---if (DI < X"FFFFFFFF") then**  **---DI <= DI + 1; WAIT FOR PERIOD;**  **---else**  **---WAIT; -- will wait forever**  **---end if;**  **---END PROCESS;**  **-- RND\_gen : PROCESS**  **-- BEGIN**  **-- if (RND < X"FFFFFFFF") then**  **-- RND <= RND + 7; WAIT FOR PERIOD;**  **-- else**  **-- RND <= X"00000000"; WAIT FOR PERIOD;**  **-- end if;**  **-- END PROCESS;**  **--M1 <= RND(1);**  **--MI <= RND(3);**  **--MI\_gen : PROCESS**  **--BEGIN**  **--MI <= '0'; WAIT FOR PERIOD\*3;**  **--MI <= '1'; WAIT FOR PERIOD\*30;**  **--MI <= '0'; WAIT FOR PERIOD\*12;**  **--MI <= '1'; WAIT FOR PERIOD\*10;**  **--MI <= '0'; WAIT FOR PERIOD\*50;**  **--MI <= '1'; WAIT FOR PERIOD\*30;**  **--MI <= '0';**  **--WAIT; -- will wait forever**  **--END PROCESS;**  **--M1\_gen : PROCESS**  **--BEGIN**  **--M1 <= '0'; WAIT FOR PERIOD\*3;**  **--M1 <= '0'; WAIT FOR PERIOD\*30;**  **--M1 <= '1'; WAIT FOR PERIOD\*12;**  **--M1 <= '1'; WAIT FOR PERIOD\*10;**  **--M1 <= '1'; WAIT FOR PERIOD\*50;**  **--M1 <= '1'; WAIT FOR PERIOD\*30;**  **--M1 <= '0';**  **--WAIT; -- will wait forever**  **--END PROCESS;**  **END;** |

# Лабораторная работа № 2. Синхронные схемы. Разработка счетчика

Проблемы, требующие синхронизации. Всякая логическая операция, выполняемая реальными полупроводниковыми приборами, формирует выходное состояние с задержкой, определяемой как устройством, так и параметрами входного сигнала и тем, куда устройство подключено, а также линиями связи, имеющими свои распределенные сопротивления и ёмкости. При достаточно большом количестве последовательно включенных устройств неверный учет задержек может изменить логическое функционирование. Более того, за счет разного по времени прихода сигналов на входы логической схемы возможно появление на выходах некорректного сигнала различной длительности. Эти случаи практически всегда означают отказ, то есть действия устройства, не соответствующие заданным.

На рисунке 1 приведена ситуация с состоянием устройства, близким к отказу.

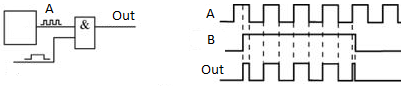


Рисунок 1. Состояние устройства, близкое к отказу (к необходимости использования синхронизации)

Действительно, часть выходных сигналов имеет очень малую временную протяженность. Если учесть, что логическая модель идеализирована (хотя есть еще и post- и p&r-модели), а реально сигнал имеет фронты ненулевой длительности, и энергетика сигнала должна обеспечить переключение нагрузки, то предсказать, как будет реагировать устройство на сигнал out, вряд ли возможно без точного схемотехнического моделирования на уровне транзисторов и описания параметров RC-цепочек. Более того, у схемотехнического моделирования на таком уровне должны быть модели, адекватные как реальным компонентам устройства, так и реальным входным сигналам. Всё это усложняет процесс моделирования, увеличивая ресурсоемкость процесса разработки.

Введение в схему устройств выравнивания потоков данных позволяет обойти эти проблемы. Собственно, даже уровень описания синхронной системы в потоках сигналов называется уровнем описания регистровых передач (register transfer lоgic), RTL.

Последовательная синхронная схема должна отвечать следующим критериям.

* Каждый элемент схемы является либо регистром, либо

комбинационной схемой.

* Как минимум один элемент схемы является регистром.
* Все регистры тактируются единственным тактовым сигналом.
* В каждом циклическом пути присутствует как минимум один

регистр.

На рисунке 2 приведена схема арифметико-логического устройства, разработанная в 70-е годы ХХ века. Разработчики использовали принцип «все триггеры будут размещены снаружи», проблемы, включающие синхронизацию, были переданы на следующий уровень. В ситуации с тогдашними технологическими возможностями это было, видимо, вполне разумным. Разработчики устройств, использующих это АЛУ, были вынуждены учитывать различные задержки, формируемые АЛУ для обеспечения нормальной работы всего комплекса.

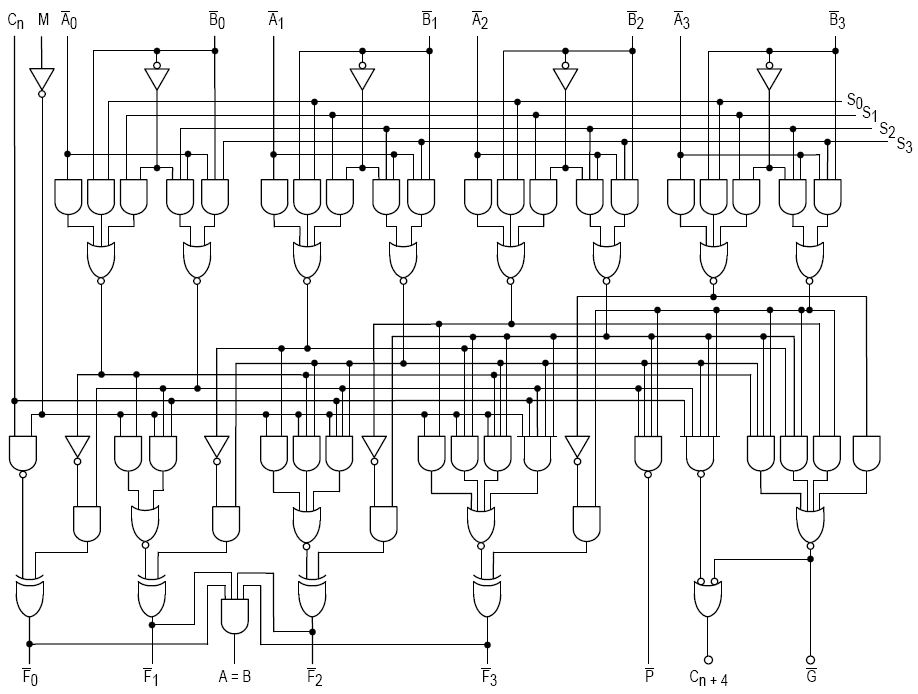


Рисунок 2. АЛУ TI 74181, не содержащее триггеры

**Последовательность разработки синхронной схемы**

По логике работы схемы проводится поиск критических путей с наибольшими задержками. Установленные критические пути и временные задержки на них дают основание для выбора тактовой частоты: задержка через критический путь в схеме должна быть меньше периода сигнала синхронизации. На этом этапе для увеличения допустимой тактовой частоты возможно разделение длинных цепочек логических элементов регистрами.

Организуются входные и выходные регистры с задержками сигнала между ними, гарантированно вписывающимися в тактовые частоты.

Анализируются обратные связи в схеме, организуется их синхронизация.

По результатам функционального моделирования принимаются решения о функциональной пригодности разработки.

Альтернативный способ - самосинхронизация, разрабатывавшаяся Маллером Д.Е. и Варшавским В.И., основана на принципе «завершившийся процесс формирует флаг, разрешающий выполнение следующего этапа». Этот способ не доведён до состояния конкурентных, четких, ясных и однозначных решений, вполне понятных разработчику. Тем не менее, исследовательские работы в этом направлении продолжаются.

*Счетчик*. К «классическим» синхронным устройствам относятся регистры и счетчики. В лабораторной работе будет рассмотрено создание таких схем.

*Сдвиговый регистр* представляет собой последовательность D- триггеров, использующих общую синхронизацию и при необходимости общую линию сброса. Сигнал с входа Inp последовательно с каждым новым фронтом синхросигнала Сlk передается от триггера к триггеру и оказывается на выходе после того, как число тактов синхронизации станет равным числу триггеров.

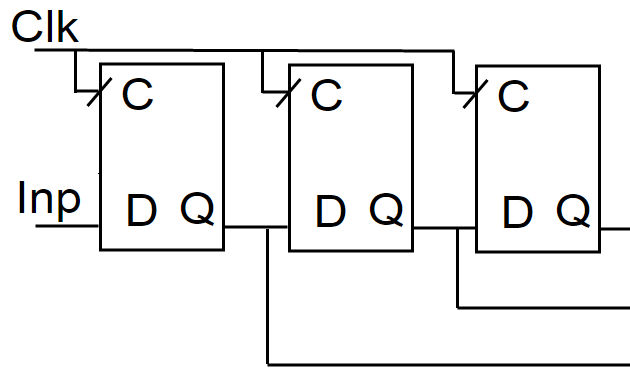


Рисунок 3. Реализация сдвигового регистра с использованием D- триггеров

*Счетчики и делители частоты*. При разработке устройств на базовых матричных кристаллах или на дискретных элементах счетчики реализуются по схеме с последовательным соединением триггеров через логические элементы, пример которой представлен на рисунке 4.

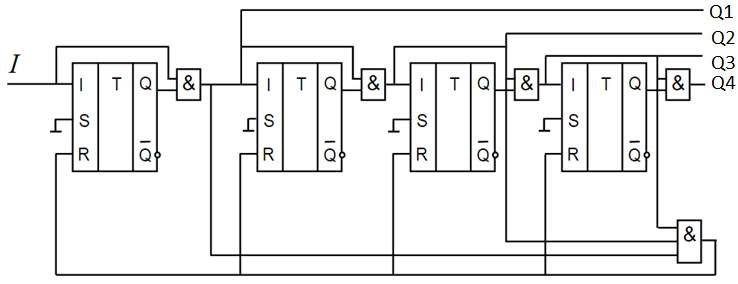


Рисунок 4. Счетчик с промежуточным сбросом, реализация на элементах базовых матричных кристаллах или дискретных элементах

При использовании счетчиков в проектах, реализованных на ПЛИС, счетчик представляет собой цепочку из последовательно включенных D-триггеров и логических элементах.

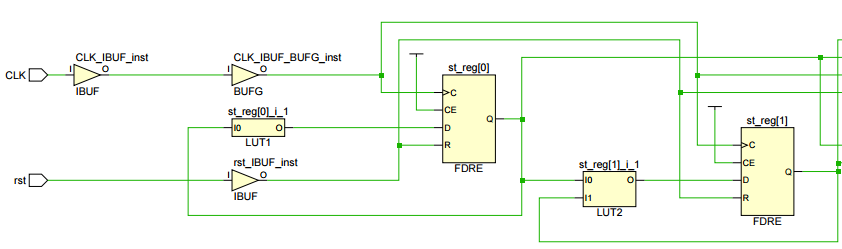


Рисунок 5. Реализация счетчика на элементах ПЛИС (фрагмент)

Частота на каждом выходе каждого триггера в два раза меньше, чем на входе. Если рассматривать входные и выходные состояния такой схемы, то каждый выход будет выдавать сигналы с частотой, вдвое меньшей входной.

Биты на выходах формируют число, представляющее собой номер импульса (разумеется, в пределах разрядности полученного устройства, то есть количества триггеров). В зависимости от того, будут использованы для выхода прямые или инверсные выходы триггеров, счет будет идти «с увеличением» или «с уменьшением» относительно исходного значения.

При описании счетчиков на ПЛИС используется другой подход, называемый поведенческим, например счетчик на 16 разрядов (диапазон 0…6553510) может быть описан так:

architecture count of counter is begin

process (c\_in, res\_c)

variable cnt:integer range 0 to 65535;

begin

if (res\_c='1') then --если высокий уровень Reset - сбрасываем счетчик

cnt:=0;

elsif (c\_in'event and c\_in='1') then

-- по переднему фронту увеличиваем значение счетчика на 1

if (enab\_c='1') then cnt:=cnt+1;

end if;

end if;

c\_out<=cnt;

end process;

end count;

«Буквальное» описание такой схемы выглядит в Vivado следующим образом:

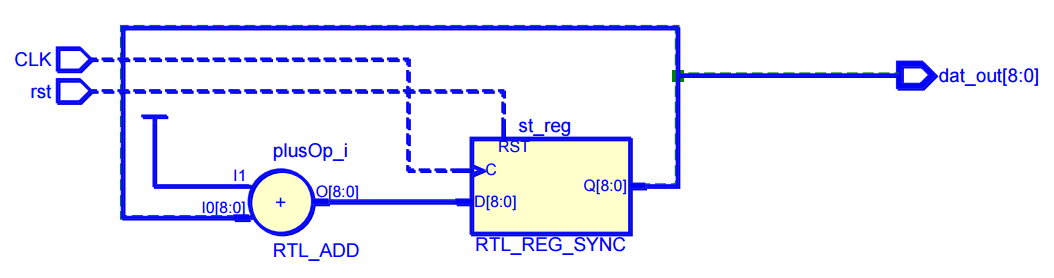


Рисунок 6. Делитель частоты, elaborate design

**Задания к лабораторной работе**

На вход схемы приходит однобитный сигнал. Восьмиразрядный выход отображает номер такта. Дополнительный выход сообщает о прохождения m тактов с начала работы счетчика. Сигнал с выхода третьего разряда счетчика (тактовая частота, поделенная на 8) отображается дополнительно с задержкой на n тактов. Схема устанавливается в начальное состояние (начало счета) сигналом rst.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № варианта | Начало счета a10 | Конец счета b10 | Прямой/  реверс | Установка по set | m10 | n10 |
| 1 | 3 | 32 | П | 4 | 5 | 2 |
| 2 | 4 | 33 | П | 5 | 6 | 3 |
| 3 | 5 | 34 | П | 6 | 7 | 2 |
| 4 | 6 | 35 | П | 7 | 8 | 3 |
| 5 | 7 | 36 | П | 8 | 9 | 2 |
| 6 | 8 | 37 | П | 9 | 10 | 3 |
| 7 | 9 | 38 | П | 10 | 11 | 2 |
| 8 | 10 | 39 | П | 11 | 12 | 3 |
| 9 | 11 | 40 | П | 12 | 13 | 2 |
| 10 | 12 | 41 | П | 13 | 14 | 3 |
| 11 | 13 | 42 | П | 14 | 15 | 2 |
| 12 | 14 | 43 | П | 15 | 16 | 3 |
| 13 | 15 | 44 | П | 16 | 17 | 2 |
| 14 | 16 | 45 | П | 17 | 18 | 3 |
| 15 | 17 | 46 | П | 18 | 19 | 2 |
| 16 | 18 | 47 | П | 19 | 20 | 3 |
| 17 | 48 | 19 | Р | 51 | 21 | 2 |
| 18 | 49 | 20 | Р | 50 | 22 | 3 |
| 19 | 50 | 21 | Р | 49 | 23 | 2 |
| 20 | 51 | 22 | Р | 48 | 24 | 3 |
| 21 | 52 | 23 | Р | 47 | 25 | 2 |
| 22 | 53 | 24 | Р | 46 | 26 | 3 |
| 23 | 54 | 25 | Р | 45 | 27 | 2 |
| 24 | 55 | 26 | Р | 44 | 28 | 3 |
| 25 | 56 | 27 | Р | 43 | 29 | 2 |

В отчете по лабораторной работе привести оценку задержек и установление критических путей в схеме АЛУ, приведенной на рисунке 2. Принять, что задержка простого логического элемента прямо пропорциональна количеству входов и количеству выходов, к которым подключен элемент.

**Приложения**

Пример описания делителя частоты структурным способом

Триггер:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

entity trig\_divider2 is

Port ( c : in STD\_LOGIC;

reset : in STD\_LOGIC;

q : out STD\_LOGIC);

end trig\_divider2;

architecture Behavioral of trig\_divider2 is

Signal d: std\_logic :='0';

begin

Process (c,reset)

Begin

if reset='0' then q<='0'; -- сброс по 0

elsif (c='0' and c'event) then -- передний фронт

d<= not d; q<=d;

End if;

End process;

end Behavioral;

Последовательное соединение триггеров:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity divider is

Port (clk : in STD\_LOGIC;

rst: in STD\_LOGIC;

div64: out STD\_LOGIC;

div2: out STD\_LOGIC;

div4: out STD\_LOGIC;

div8: out STD\_LOGIC;

div16: out STD\_LOGIC;

div32: out STD\_LOGIC);

end divider;

architecture structure of divider is

component trig\_divider2

port(c:in STD\_LOGIC; reset: in STD\_LOGIC; q:out STD\_LOGIC);

end component;

signal res: STD\_LOGIC;

signal res\_enable: STD\_LOGIC;

signal int\_div2: STD\_LOGIC;

signal int\_div4: STD\_LOGIC;

signal int\_div8: STD\_LOGIC;

signal int\_div16: STD\_LOGIC;

signal int\_div32: STD\_LOGIC;

signal int\_div64: STD\_LOGIC;

begin

res <= rst; ---(or not (div32 and div8));

dd11: trig\_divider2 port map(clk, res, int\_div2);

dd12: trig\_divider2 port map(int\_div2, res, int\_div4);

dd13: trig\_divider2 port map(int\_div4, res, int\_div8);

dd14: trig\_divider2 port map(int\_div8, res, int\_div16);

dd15: trig\_divider2 port map(int\_div16, res, int\_div32);

dd16: trig\_divider2 port map(int\_div32, res, int\_div64);

div2 <= int\_div2;

div4 <= int\_div4;

div8 <= int\_div8;

div16 <= int\_div16;

div32 <= int\_div32;

div64 <= int\_div64;

end structure;

Пример схемы с использованием триггера для работы выхода

**entity addr\_sel is**

**Port ( a0 : in STD\_LOGIC;**

**a1 : in STD\_LOGIC;**

**a2 : in STD\_LOGIC;**

**a3 : in STD\_LOGIC;**

**a4 : in STD\_LOGIC;**

**clk : in STD\_LOGIC;**

**It\_is\_it : out STD\_LOGIC);**

**end addr\_sel;**

**architecture Behavioral of addr\_sel is**

**begin**

**--- Триггер, тактируемый передним фронтом (Rising Edge Flip-Flop)**

**process (clk)**

**begin**

**if (clk'event and clk = '1') then**

**It\_is\_it <= (a0 and (not a1)) and (a2 and (not a3)) and (not a4) ;**

**end if;**

**end process;**

**-- Так было It\_is\_it <= (a0 and (not a1)) and (a2 and (not a3)) and (not a4);**

**end Behavioral;**

**Результат моделирования:**

# Лабораторная работа №3. Реализация конечного автомата

Конечные автоматы (КА) получили свое название из-за того, что схема с k-регистрами может находиться в одном из 2k, то есть в конечном числе состояний. У КА М входов, N выходов и k бит состояний. На вход КА также подается тактовый сигнал и, возможно, сигнал сброса. КА состоит из двух блоков комбинационной логики: логики перехода в следующее состояние и выходной логики, а также из регистра, в котором хранится текущее состояние. По фронту каждого тактового импульса автомат переходит в следующее состояние, которое определяется текущим состоянием и значениями на входах. Существует два основных класса конечных автоматов, которые отличаются своими функциональными описаниями. В автомате Мура выходные значения зависят лишь от текущего состояния, в то время как в автомате Мили выход зависит как от текущего состояния, так и от входных данных. Конечные автоматы предоставляют систематический способ проектирования синхронных последовательностных схем по заданному функциональному описанию. При выполнении лабораторной работы необходимо:

1. описать работу конечного автомата в виде диаграммы переходов (графа);
2. сформировать таблицу переходов;
3. записать таблицу состояний, таблицу выходов и логические выражения для описания схемы. Пример описания рассмотрен на с. 307-326 [1];
4. составить описание схемы в VHDL, провести моделирование в Vivado, подключить внешние контакты, определить частоты, на которых схема работоспособна.

Цифровые автоматы на VHDL описываются в форме двух процессов: комбинаторного и регистрового. В комбинаторном процессе формируется следующее состояние автомата (fsm\_next), в регистровом процессе текущее состояние (fsm\_current) заменяется на следующее (fsm\_next). Описание автомата начинается с создания типа и объявления переменных этого типа.

Пример описания автомата:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity fin\_avt is

Port (clk : in std\_logic;

cnt : in std\_logic;

rst : in std\_logic;

light : OUT std\_logic\_vector(2 downto 0));

end fin\_avt;

architecture Behavioral of fin\_avt is

TYPE state\_type IS (Init, R, RG, G, GR);

SIGNAL next\_st : state\_type;

SIGNAL present\_st : state\_type;

BEGIN

state\_proc: PROCESS (present\_st, cnt)

BEGIN

CASE present\_st IS

WHEN Init =>

IF cnt = '1'

THEN next\_st <= R;

ELSE

next\_st <= Init;

END IF;

WHEN R =>

if cnt = '1'

THEN next\_st <= RG;

ELSE

next\_st <= R;

END IF;

WHEN RG =>

IF cnt = '1'

THEN next\_st <= G;

ELSE

next\_st <= RG;

END IF;

WHEN G =>

IF cnt = '1'

THEN NEXT\_st <= GR;

ELSE

next\_st <= G;

END IF;

WHEN GR =>

IF CNT = '1'

THEN next\_st <= R;

ELSE

next\_st <= GR;

END IF;

WHEN OTHERS =>

next\_st <= Init;

END CASE;

END PROCESS;

next\_st\_proc:

PROCESS (clk, rst)

BEGIN

IF rst = '1'

THEN present\_st <= Init;

ELSIF (rising\_edge(clk))

THEN present\_st <= next\_st;

END IF;

END PROCESS;

out\_proc:

PROCESS (present\_st)

BEGIN

CASE present\_st IS

WHEN Init =>

light <= "000";

WHEN R =>

light <= "100";

WHEN RG =>

light <= "010";

WHEN G =>

light <= "001";

WHEN GR =>

light <= "010";

END CASE;

END PROCESS;

end Behavioral;

Задания на работу: разработать в виде конечного автомата схему, реагирующую на появление на входе заданной последовательности. Следует указать, используется автомат Мили или Мура.

Задания для выполнения лабораторной работы

На вход конечного автомата подается восьмибитное слово А. Переходы осуществляются по условию равенства А величине, приведенной в таблице в десятичном коде. Сигнал rst приводит к переходу схемы из любого состояния в состояние А. По предложенному графу и таблице условий перехода построить конечный автомат. При моделировании проверить все переходы между состояниями.

|  |  |
| --- | --- |
| Вариант 1 | Вариант 2 |
|  |  |

Вариант 1. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BE | CF | DE | EH | EF | FG | GI | FI | HI |
| 1 | 221 | 23 | 24 | 25 | 26 | 127 | 28 | 29 | 30 | 180 | 201 | 202 |

Вариант 2. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BE | CF | DH | EH | EI | FG | GI | FI | HI |
| 2 | 210 | 57 | 31 | 125 | 122 | 97 | 88 | 24 | 32 | 115 | 211 | 102 |

|  |  |
| --- | --- |
| Вариант 3 | Вариант 4 |
|  |  |

Вариант 3. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BE | CF | DE | FG | FE | EI | EH | HI | GI |
| 3 | 210 | 57 | 31 | 125 | 122 | 97 | 88 | 24 | 32 | 115 | 211 | 102 |

Вариант 4. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BF | CF | DE | FG | FE | EI | EH | HI | GI |
| 4 | 125 | 122 | 211 | 102 | 122 | 210 | 57 | 31 | 125 | 122 | 211 | 102 |

|  |  |
| --- | --- |
| Вариант 5 | Вариант 6 |
|  |  |

Вариант 5. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BF | CF | DE | FG | FE | EI | DH | HI | GI |
| 5 | 121 | 102 | 201 | 112 | 132 | 200 | 157 | 121 | 105 | 132 | 201 | 102 |

Вариант 6. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BF | CG | DE | | FG | FE | EI | DH | HI | GI |
| 6 | 121 | 112 | 211 | 212 | 130 | 250 | | 147 | 127 | 115 | 32 | 216 | 155 |
| Вариант 7 | | | | | | | Вариант 8 | | | | | | |
|  | | | | | | |  | | | | | | |

Вариант 7. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BF | CG | DI | FG | FE | EI | DH | HI | GI |
| 7 | 111 | 212 | 113 | 205 | 104 | 100 | 174 | 172 | 145 | 67 | 111 | 55 |

Вариант 8. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BF | CF | DI | FG | FE | EI | DH | HI | GI |
| 8 | 117 | 112 | 103 | 215 | 4 | 106 | 166 | 187 | 132 | 167 | 11 | 255 |

|  |  |
| --- | --- |
| Вариант 9 | Вариант 10 |
|  |  |

Вариант 9. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BF | CF | DI | FG | FE | EH | DH | HI | GI |
| 9 | 107 | 122 | 3 | 117 | 24 | 121 | 157 | 172 | 144 | 67 | 11 | 000 |

Вариант 10. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BF | CF | DI | FG | FE | EH | DH | HG | GI |
| 10 | 7 | 105 | 13 | 119 | 224 | 21 | 159 | 186 | 75 | 167 | 102 | 103 |

|  |  |
| --- | --- |
| Вариант 11 | Вариант 12 |
|  |  |

Вариант 11. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BE | CF | DE | FG | FE | EI | EH | HI | GI |
| 11 | 210 | 57 | 31 | 125 | 122 | 97 | 88 | 24 | 32 | 115 | 211 | 102 |

Вариант 12. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BF | CF | DE | FG | FE | EI | EH | HI | GI |
| 12 | 121 | 112 | 211 | 212 | 130 | 250 | 147 | 127 | 115 | 32 | 216 | 155 |

|  |  |
| --- | --- |
| Вариант 13 | Вариант 14 |
|  |  |

Вариант 13. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BF | CF | DE | FG | FE | EI | DH | HI | GI |
| 13 | 221 | 12 | 101 | 122 | 135 | 100 | 157 | 121 | 105 | 132 | 201 | 202 |

Вариант 14. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BF | CG | DE | FG | FE | EI | DH | HI | GI |
| 14 | 121 | 112 | 211 | 212 | 38 | 250 | 147 | 127 | 115 | 32 | 186 | 105 |

|  |  |
| --- | --- |
| Вариант 15 | Вариант 16 |
|  |  |

Вариант 15. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BF | CF | DE | FG | FE | EI | DH | HI | GI |
| 15 | 121 | 102 | 201 | 112 | 132 | 200 | 157 | 121 | 105 | 132 | 201 | 102 |

Вариант 16. Условия переходов

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | AB | AD | BC | BF | CG | DE | FG | FE | EI | DH | HI | GI |
| 16 | 121 | 112 | 211 | 212 | 130 | 250 | 147 | 127 | 115 | 32 | 216 | 155 |

Рекомендуемые дополнительные источники к работе:

Описание цифровых автоматов на VHDL <http://distant.msu.ru/pluginfile.php/39887/mod_resource/content/1/%D0%A2%D0%B5%D0%BC%D0%B0%208.pdf>

Описание цифровых автоматов на VHDL <http://geektimes.ru/post/254818/>

Лабораторная работа №4. Реализация АЛУ с расширением

Цель работы: Ознакомиться с функциями, реализуемыми простейшим АЛУ. Получить практические навыки в реализации и тестировании АЛУ на языке VHDL в среде Vivado.

1.      Общие сведения о проектируемом АЛУ

1.1.   Простейшее АЛУ содержит:

1.1.1.     два входа для операндов;

1.1.2.     вход для указания выполняемой операции;

1.1.3.     выход для результата;

1.1.4.     выходы для флагов;

1.1.5.     блоки, выполняющие операции;

1.1.6.     блоки, устанавливающие значения флагов.

1.2.   Простейшее АЛУ реализует:

1.2.1.     арифметическое суммирование;

1.2.2.     побитовое И;

1.2.3.     побитовое ИЛИ;

1.2.4.     побитовое исключающее ИЛИ;

1.2.5.     инкрементацию А;

1.2.6.     побитовое НЕ;

1.2.7.     нет операции, на выходе «0»;

1.3.   Простейшее АЛУ устанавливает значения флагов:

1.3.1.     Z – флаг нулевого результата;

1.3.2.     OV – флаг переполнения (используется при необходимости);

1.3.3.     P – флаг четности результата.

Структура АЛУ

Поскольку по заданию используется семь операций (плюс одна дополнительная), команды можно описать тремя битами:

Биты Операция

000 Нет операции

001 Суммирование;

010 Побитовое И;

011 Побитовое ИЛИ;

100 Побитовое исключающее ИЛИ;

101 Побитовое НЕ;

110 Инкремент х;

111 Резервная команда из списка.

*Разрядность АЛУ-8*. Таким образом, операнды на входе будут иметь разрядность 8 бит, максимальная разрядность выходного слова (при выполнении команды 16\*A+B+1 составит 13 бит).

Варианты заданий

|  |  |
| --- | --- |
| Номер варианта | Дополнительная функция АЛУ |
| 1 | A+2\*B |
| 2 | A+4\*B |
| 3 | A+8\*B |
| 4 | A+16\*B |
| 5 | 2\*A+B |
| 6 | 4\*A+B |
| 7 | 8\*A+B |
| 8 | 16\*A+B |
| 9 | A+B+1 |
| 10 | A+2\*B+1 |
| 11 | A+4\*B+1 |
| 12 | A+8\*B+1 |
| 13 | A+16\*B+1 |
| 14 | 2\*A+B+1 |
| 15 | 4\*A+B+1 |
| 16 | 8\*A+B+1 |
| 17 | 16\*A+B+1 |
| 18 | 2\*A+4\*B) |
| 19 | 2\*A+8\*B |
| 20 | 4\*A+2\*B |
| 21 | 4\*A+4\*B |
| 22 | 8\*A+2\*B |
| 23 | 8\*A+8\*B |
| 24 | 8\*A+1 |
| 25 | 8\*A+2 |
| 26 | A-15 |
| 27 | B-15 |
| 28 | A+32 |
| 29 | B+32 |
| 30 | A-31 |

Для выполнения специфических задач целесообразно использовать дополнительные вычислительные блоки совместно с обычными арифметико-логическими устройствами. Применения ПЛИС даёт возможность реализовывать такие блоки быстро и достаточно эффективно. В ходе этой работы необходимо:

1. Изучить по схеме и описанию АЛУ TI 74181 или его аналога к155ИП3 (простейшие АЛУ) реализацию внешних команд (сложение, логические операции, вычитание 1 из операнда А).
2. Описать работу базового АЛУ в VHDL, добавив для выходного состояния тактируемый регистр.
3. Посмотреть по схеме критический по времени путь и, возможно, внести уточнения в описание проекта (с обязательным отражением в отчете «было-стало»).
4. Добавить к АЛУ операцию из индивидуального задания.
5. Разработать и провести синтез и имплементацию АЛУ с дополнительной функцией. Организовать привязку портов к внешним выводам в соответствии с рекомендациями из [7].

Отчет по лабораторной работе должен включать описание логики работы разработанного устройства (VHDL-текст), результаты моделирования после имплементации, установленную максимальную скорость работы устройства.

**Приложение**

Вариант описания базовой части АЛУ

|  |
| --- |
| ---------------------------------------------------------------------------------  -- Company:  -- Engineer:  --  -- Create Date: 17.12.2015 15:45:20  -- Design Name:  -- Module Name: simply\_alu - Behavioral  -- Project Name:  -- Target Devices:  -- Tool Versions:  -- Description:  --  -- Dependencies:  --  -- Revision:  -- Revision 0.01 - File Created  -- Additional Comments:  --  ---------------------------------------------------------------------------------  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_ARITH.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  -- Uncomment the following library declaration if instantiating  -- any Xilinx primitives in this code.  --library UNISIM;  --use UNISIM.VComponents.all;  entity simply\_alu is  port  (  clk : in std\_logic;  op : in std\_logic\_vector(2 downto 0);  a, b : in std\_logic\_vector(7 downto 0);  c : out std\_logic\_vector(7 downto 0);  zf : out std\_logic;  zof : out std\_logic);  end simply\_alu;  architecture Behavioral of simply\_alu is  signal tmp : std\_logic\_vector(7 downto 0) := (others => '0');  begin  process (clk,a,b,op)  begin  if rising\_edge(clk) then  case op is  when "000" => tmp <= x"00"; --нет операции  when "001" => tmp <= a + b;  when "010" => tmp <= a and b;  when "011" => tmp <= a or b;  when "100" => tmp <= a xor b;  when "101" => tmp <= not a;  when "110" => tmp <= a + 1;  -- temp <= temp(2 downto 0) & temp(3); так добавляются биты, первый сигнал поместится старшие разряды шины, второй сигнал - в младшие.  when "111" => tmp <= x"00"; -- место для Вашей операции  when others => tmp <= x"00"; -- need others when case  end case;  end if;  end process;  c<=tmp;  zf <= not(tmp(7) or tmp(6) or tmp(5) or tmp(4) or tmp(3) or tmp(2) or tmp(1) or tmp(0));  zof <= tmp(7) and tmp(6) and tmp(5) and tmp(4) and tmp(3) and tmp(2) and tmp(1) and tmp(0);  end Behavioral; |

# Рекомендуемая литература

1. Харрис Д., Сара Л. Харрис С. Цифровая схемотехника и архитектура компьютера. 2015. Распространяется свободно, доступно по адресу community.imgtec.com/downloads/digital-design-and-computer-architecture-russian-edition, зеркало https://cloud.mail.ru/public/LAqh/khU9qjNJJ
2. Поляков А. Языки VHDL и VERILOG в проектировании цифровой аппаратуры. - М.: Солон-Р, 2003. - 320 с.
3. Тарасов И.Е. Проектирование для ПЛИС Xilinx с применением языков высокого уровня в среде Vivado HLS. Компоненты и технологии, №12, 2013, доступно http://www.kit-e.ru/preview/pre\_40\_12\_13\_VHLS\_Xilinx.php.
4. Коноплев Б.Г., Рындин Е.А., Ивченко В.Г. Описание проектов СБИС с использованием языка VHDL. Таганрог, изд-во ТГРУ, 1998. Доступно http://window.edu.ru/resource/846/28846.
5. ПЛИС Xilinx. Общие сведения. https://alterozoom.com/ru/documents/20062.html.
6. Стешенко В.Б. ПЛИС фирмы Altera: проектирование устройств обработки сигналов. - М. ОЛЭКА, 2000.
7. Тарасов И.E. Методы и программные продукты для повышения производительности проектов на базе ПЛИС Xilinx. Компоненты и технологии, №1, 2008. Доступно http://kit-e.ru/articles/plis/2008\_01\_88.php.

**Дополнительные задания**

К лабораторной работе 1. Реализовать вариант устройства с использованием компараторов. Сравнить полученные устройства после имплементации по потребляемой мощности и временным возможностям.

**Илья Израилевич Левин**

**Борис Евгеньевич Механцев**

**ЛАБОРАТОРНЫЕ РАБОТЫ ПО ДИСЦИПЛИНЕ**

**«ПЛИС-ТЕХНОЛОГИИ И МЕТОДЫ СОЗДАНИЯ**

**ЭФФЕКТИВНЫХ ПРИКЛАДНЫХ ПРОГРАММ ДЛЯ ПЛИС»**

Редактор

Компьютерная верстка

|  |  |
| --- | --- |
| ЛР № 020565 от 23.06.97 г.  Формат 60×841/16  Офсетная печать. Усл. п. л. – 7,1  Заказ №\_\_\_ | Подписано к печати 12.12.2017  Бумага офсетная  Уч.-изд. л. – 6,4  Тир. 100 экз. |

«С»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Издательство Южного федерального университета

344091, г. Ростов-на Дону, пр. Стачки, 200/1, Тел. (863)2478051.

Отпечатано в Секторе обеспечения полиграфической продукцией кампуса в г. Таганроге отдела полиграфической, корпоративной и сувенирной продукции

ИПКА КИБИ МЕДИА ЦЕНТРА ЮФУ.

ГСП 17А, Таганрог, 28, Энгельса, 1, Тел. (8634)371717, 371655.