Лабораторные работы по дисциплине

«ПЛИС- технологии и методы создания эффективных прикладных программ для ПЛИС»

Оглавление

[Введение 3](#_Toc438816991)

[Лабораторная работа №1. Реализация АЛУ с расширением 27](#_Toc438816995)

[Рекомендуемая литература 32](#_Toc438816996)

# 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. Реализация АЛУ с расширением с использованием DSP блоков

 Цель работы: Ознакомиться с возможностью реализации арифметическо- логических операций с использованием DSP блоков. реализуемыми простейшим АЛУ. Получить практические навыки в реализации и тестировании АЛУ с использованием DSP блоков 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;

1.2.5.     Побитовое НЕ;

1.2.6.     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.

Возможности 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 |

В ходе этой работы необходимо:

1. Изучить и по схеме и описанию DSP блок ПЛИС. Предложить систему команд.
2. Разместить в проекте IP ядро, разработать управление IP ядром, добавить для выходного состояния тактируемый регистр.
3. Добавить к АЛУ операцию из индивидуального задания.
4. 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.

Рекомендуется в установках проекта выбрать ПЛИС *XC7V585T-FFG1761-1-С* для того, чтобы в дальнейшем можно было использовать проект для работы с «Плеядой». Пользователи web лицензий Vivado могут использовать доступные им ПЛИС.

# Лабораторная работа №2. Подготовка проекта для платы «Плеяда»

Цель работы- подготовка и запись проекта для функционирования в системе «Плеяда».

Последовательность работы:

- адаптация проекта к работающей системе синхронизации на дифференциальных парах и сброса

- проведение моделирования с использованием

- полный цикл проектирования, включающий получение bitmap и запись в реальную ПЛИС

-

Общая информация о платах «Плеяда».

1.1. Количество ПЛИС установленных на плате – 7.

1.2. Позиционные обозначения ПЛИС на плате: DD1, DD2, DD3, DD4, DD5, DD6, DD7.

1.3. Функциональное назначение ПЛИС:

DD7 – контроллер базового модуля (ПЛИС КБМ);

DD1- DD6 – ПЛИС вычислительного поля (ПЛИС ВП).

1.4. Тип установленных ПЛИС:

DD7 – XC6VLX130T-FF1156-1-C.

DD1- DD6 – XC7V585T-FFG1761-1-С;

Размещение, стандарт и временные ограничения для выводов ПЛИС ВП, на которые подаются сигналы синхронизации.

К каждой ПЛИС ВП подведено по два дифференциальных сигнала синхронизации 250 МГц и 200 МГц.

Для подключения к сигналам синхронизации 250 МГц или 200 МГц следует использовать дифференциальный буфер IBUFGDS. Буфер необходимо вставить в описание проекта и добавить в XDC-файл строки из столбца «Описание подключения в XDC-файле» из Таблицы 1

Таблица 1

|  |  |  |
| --- | --- | --- |
| № ПЛИС на плате | Описание подключения | Частота синхро-сигнала |
| DD1-DD6 | **set\_property PACKAGE\_PIN AL34 [get\_ports CLK1\_DD1\_N]**  **set\_property IOSTANDARD LVDS [get\_ports CLK1\_DD1\_N]**  **set\_property DIFF\_TERM\_ADV TERM\_100 [get\_ports CLK1\_DD1\_N]**  **set\_property IBUF\_LOW\_PWR FALSE [get\_ports CLK1\_DD1\_N]**  **set\_property PACKAGE\_PIN AK34 [get\_ports CLK1\_DD1\_P]**  **set\_property IOSTANDARD LVDS [get\_ports CLK1\_DD1\_P]**  **set\_property DIFF\_TERM\_ADV TERM\_100 [get\_ports CLK1\_DD1\_P]**  **set\_property IBUF\_LOW\_PWR FALSE [get\_ports CLK1\_DD1\_P]**  **create\_clock -period 4.000 -name clk\_board1 [get\_ports CLK1\_DD1\_P]** | 250 мГц |
| DD1-DD6 | **set\_property PACKAGE\_PIN AK32 [get\_ports CLK2\_DD1\_N]**  **set\_property IOSTANDARD LVDS [get\_ports CLK2\_DD1\_N]**  **set\_property DIFF\_TERM\_ADV TERM\_100 [get\_ports CLK2\_DD1\_N]**  **set\_property IBUF\_LOW\_PWR FALSE [get\_ports CLK2\_DD1\_N]**  **set\_property PACKAGE\_PIN AJ32 [get\_ports CLK2\_DD1\_P]**  **set\_property IOSTANDARD LVDS [get\_ports CLK2\_DD1\_P]**  **set\_property DIFF\_TERM\_ADV TERM\_100 [get\_ports CLK2\_DD1\_P]**  **set\_property IBUF\_LOW\_PWR FALSE [get\_ports CLK2\_DD1\_P]**  **create\_clock -period 5.000 -name clk\_board2 [get\_ports CLK2\_DD1\_P]** | 200 мГц |

Размещение и стандарт выводов ПЛИС ВП, к которым подключен сигнал общего сброса «RESET» формируемый ПЛИС КБМ.

К каждой ПЛИС ВП подведён сигнал общего сброса «RESET» предназначеннный для возвращения логической схемы в первоначальное состояние. Сигнал «RESET» не следует путать с сигналом «PROGRAMM», который является сигналом сброса конфигурации ПЛИС. Сигнал «RESET» формируется на частоте 250 МГц.

Таблица 2.

|  |  |  |
| --- | --- | --- |
| Номер ПЛИС | Описание подключения в XDC-файле | Активный уровень |
| DD1-DD6 | **set\_property PACKAGE\_PIN AP37 [get\_ports RESET]**  **set\_property IOSTANDARD LVCMOS18 [get\_ports RESET]**  **set\_property PULLUP true [get\_ports RESET]**  **set\_property IOB TRUE [get\_ports RESET]** | 0 |

Шины ПЛИС, задействованные для информационного обмена

Таблица 3.

|  |  |  |
| --- | --- | --- |
| Номер ПЛИС | Описание подключения в XDC-файле | Приме-чание |
| DD1 | **set\_property PACKAGE\_PIN AY28 [get\_ports {DATA\_IO[0]}]**  **set\_property PACKAGE\_PIN AV29 [get\_ports {DATA\_IO[1]}]**  **set\_property PACKAGE\_PIN BA27 [get\_ports {DATA\_IO[2]}]**  **set\_property PACKAGE\_PIN BB29 [get\_ports {DATA\_IO[3]}]**  **set\_property PACKAGE\_PIN BB27 [get\_ports {DATA\_IO[4]}]**  **set\_property PACKAGE\_PIN BA29[get\_ports {DATA\_IO[5]}]**  **set\_property PACKAGE\_PIN AW26 [get\_ports {DATA\_IO[6]}]**  **set\_property PACKAGE\_PIN AT29 [get\_ports {DATA\_IO[7]}]**  **set\_property PACKAGE\_PIN AV26 [get\_ports {DATA\_IO[8]}]**  **set\_property PACKAGE\_PIN AW28 [get\_ports {DATA\_IO[9]}]**  **set\_property PACKAGE\_PIN AV28 [get\_ports {DATA\_IO[10]}]**  **set\_property PACKAGE\_PIN AU27 [get\_ports {DATA\_IO[11]}]**  **set\_property PACKAGE\_PIN AT27 [get\_ports {DATA\_IO[12]}]**  **set\_property PACKAGE\_PIN AR28 [get\_ports {DATA\_IO[13]}]**  **set\_property PACKAGE\_PIN AP28 [get\_ports {DATA\_IO[14]}]**  **set\_property PACKAGE\_PIN AT26 [get\_ports {DATA\_IO[15]}]**  **set\_property PACKAGE\_PIN AR25 [get\_ports {DATA\_IO[16]}]**  **set\_property PACKAGE\_PIN AN26 [get\_ports {DATA\_IO[17]}]**  **set\_property PACKAGE\_PIN AM29[get\_ports {DATA\_IO[18]}]**  **set\_property PACKAGE\_PIN AL27 [get\_ports {DATA\_IO[19]}]**  **set\_property PACKAGE\_PIN BA35 [get\_ports {DATA\_IO[20]}]**  **set\_property PACKAGE\_PIN AW36 [get\_ports {DATA\_IO[21]}]**  **set\_property PACKAGE\_PIN BB34 [get\_ports {DATA\_IO[22]}]**  **set\_property PACKAGE\_PIN BB36 [get\_ports {DATA\_IO[23]}]**  **set\_property PACKAGE\_PIN BB33 [get\_ports {DATA\_IO[24]}]**  **set\_property PACKAGE\_PIN AY35 [get\_ports {DATA\_IO[25]}]**  **set\_property PACKAGE\_PIN AU34 [get\_ports {DATA\_IO[26]}]**  **set\_property PACKAGE\_PIN AU36 [get\_ports {DATA\_IO[27]}]**  **set\_property PACKAGE\_PIN AU33 [get\_ports {DATA\_IO[28]}]**  **set\_property PACKAGE\_PIN AT35 [get\_ports {DATA\_IO[29]}]**  **set\_property PACKAGE\_PIN AV33[get\_ports {DATA\_IO[30]}]**  **set\_property PACKAGE\_PIN AW33 [get\_ports {DATA\_IO[31]}]**  **set\_property IOSTANDARD LVCMOS25 [get\_ports {DATA\_IO[\*]}]**  **set\_property IOB TRUE [get\_ports {DATA\_IO[\*]}]** | i/o |

Служебные порты ПЛИС

|  |  |
| --- | --- |
| Порт | Функция |
| **set\_property PACKAGE\_PIN AY33 [get\_ports RDWR]**  **set\_property IOSTANDARD LVCMOS25 [get\_ports RDWR]**  **set\_property IOB TRUE [get\_ports RDWR]** | Сигнал информирует о том, что данные установленные на шине DATA\_IO, относятся к регистру команд (при этом STR\_COM = “0”) или регистру данных (при этом STR\_COM = “1”) и подтверждает валидность данных на шине DATA\_IO при записи, а также служит для запроса данных из вычислительных ПЛИС при чтении. |
| **set\_property PACKAGE\_PIN AV35 [get\_ports STR\_COM]**  **set\_property IOSTANDARD LVCMOS25 [get\_ports STR\_COM]**  **set\_property IOB TRUE [get\_ports STR\_COM]** | Сигнал, информирующий о том, что данные, установленные на шине DATA\_IO, относятся к регистру команд и подтверждает валидность команды на шине DATA\_IO. |
| **set\_property PACKAGE\_PIN AW31 [get\_ports ACK\_O]**  **set\_property IOSTANDARD LVCMOS25 [get\_ports ACK\_O]**  **set\_property IOB TRUE [get\_ports ACK\_O]** | Сигнал, подтверждающий валидность данных на шине DATA\_IO при чтении данных из ПЛИС вычислительного поля. |
| **set\_property PACKAGE\_PIN AY30 [get\_ports IRQ\_OUT]**  **set\_property IOSTANDARD LVCMOS25 [get\_ports IRQ\_OUT]**  **set\_property IOB TRUE [get\_ports IRQ\_OUT]** | Сигнал прерывания, непрерывно записывается в 5-й бит регистра команд КБМ. |
| **set\_property PACKAGE\_PIN AK25 [get\_ports VP\_READY]**  **set\_property IOSTANDARD LVCMOS25 [get\_ports VP\_READY]**  **set\_property IOB TRUE [get\_ports VP\_READY]** | Активный уровень этого сигнала указывает на необходимость остановки записи данных в режиме FAST в ПЛИС ВП, т.е. снять сигнал RDWR. |
| **set\_property PACKAGE\_PIN AT30 [get\_ports BURST]**  **set\_property IOSTANDARD LVCMOS25 [get\_ports BURST]**  **set\_property IOB TRUE [get\_ports BURST]** | Данный сигнал подтверждает валидность данных на шине DATA\_IO при потоковом чтении данных. |
| **set\_property PACKAGE\_PIN BB31 [get\_ports {MODE[0]}]**  **set\_property IOSTANDARD LVCMOS25 [get\_ports {MODE[0]}]**  **set\_property IOB TRUE [get\_ports {MODE[0]}]**  **set\_property PACKAGE\_PIN BA32 [get\_ports {MODE[1]}]**  **set\_property IOSTANDARD LVCMOS25 [get\_ports {MODE[1]}]**  **set\_property IOB TRUE [get\_ports {MODE[1]}]** | Режим работы. Используются следующие режимы:  00 – при работе с КРП - запись данных, поступающих из ПЛИС КБМ;  01 – при работе с КРП - чтение данных; в FAST режиме после установки данного сигнала необходимо активировать сигнал BURST;  10 – пуск КРП;  11 – сброс КРП. |
| **set\_property PACKAGE\_PIN AV31 [get\_ports PROG\_FULL]**  **set\_property IOSTANDARD LVCMOS25 [get\_ports PROG\_FULL]**  **set\_property IOB TRUE [get\_ports PROG\_FULL]** | При выполнении операции чтения из ПЛИС ВП в режиме FAST, при активном уровне этого сигнала необходимо остановить поток данных в КБМ, т.е. снять сигнал BURST. |

Последовательность выполнения лабораторной работы

Доработать проект АЛУ за счет организации работы с синхронизацией и управляемым использованием reset. Не использовать внешние контакты, задействованные под служебные функции.

Провести синтез и имплементацию проекта и сгенерировать bitmap

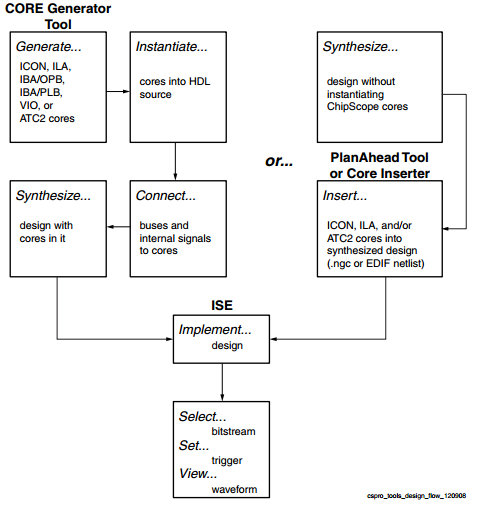
Используя режим удаленного экрана провести загрузку в ПЛИС «Плеяды» проекта (используя установленные на Плеяде варианты Vivado)

# Лабораторная работа №3. Знакомство с Chipscope

Цель работы. Размещение в схеме ILA ядра (тремя способами) и наблюдение статических состояний.

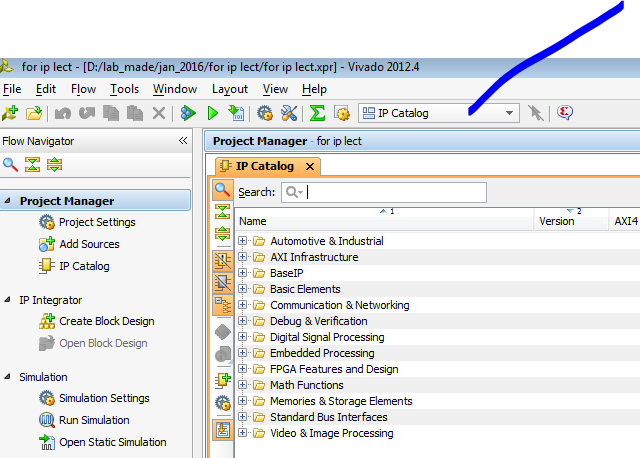
На первом этапе работы в подготовленный проект проводится имплементация ядра ILA.

Три способа подключения ядра к проекту

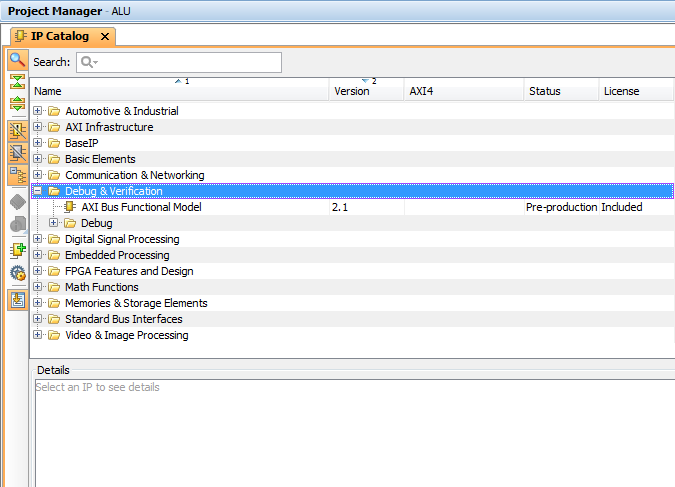


* Стандартная технология подключения IP ядра
* Прямое указание в XDC файле
* Работа с результатами синтеза

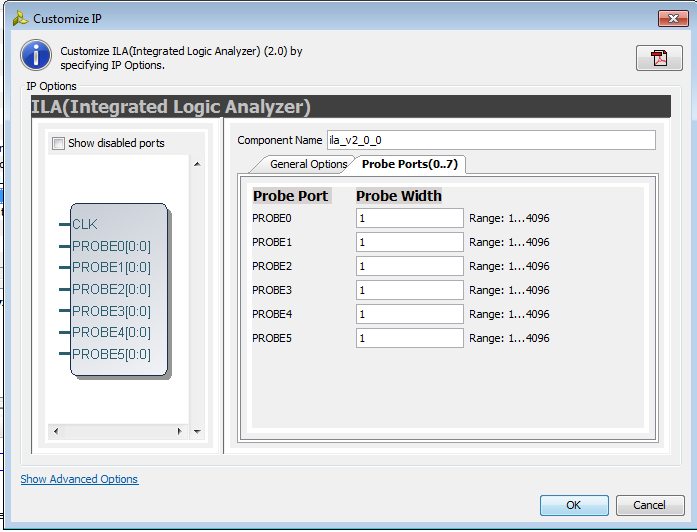
1. В меню layout выбирается «IP catalog»

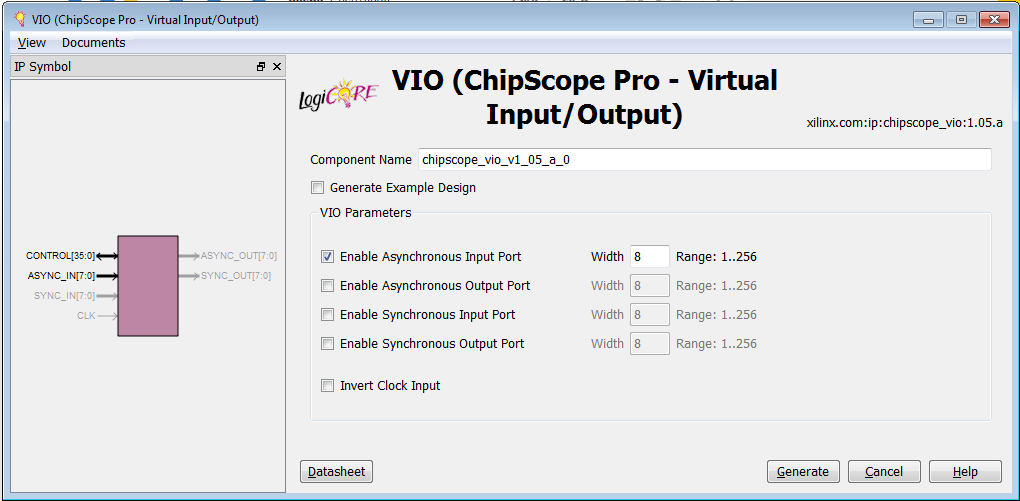


Выбирается нужное IP ядро

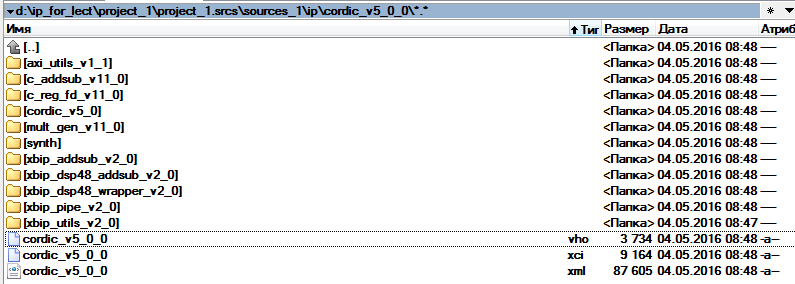


Проводится настройка параметров IP ядра





После этого в папке sources\_1 проекта сформируется папка ip с папкой описания ip ядра и файлом описания компонента



Рассмотрим методику работы на примере. В качестве исходных данных возьмем знакомый по лабораторной работе 4 проект АЛУ (в восьмибитном упрощенном варианте):

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 "100" => tmp <= a xor b;

when "101" => tmp <= not a;

when "110" => tmp <= a + 1;

-- LED\_temp <= LED\_temp(2 downto 0) & LED\_temp(3);

when "111" => tmp <= x"00"; -- место для Вашей операции

when others => tmp <= x"00"; -- need others when case

end case;

end if;

end process;

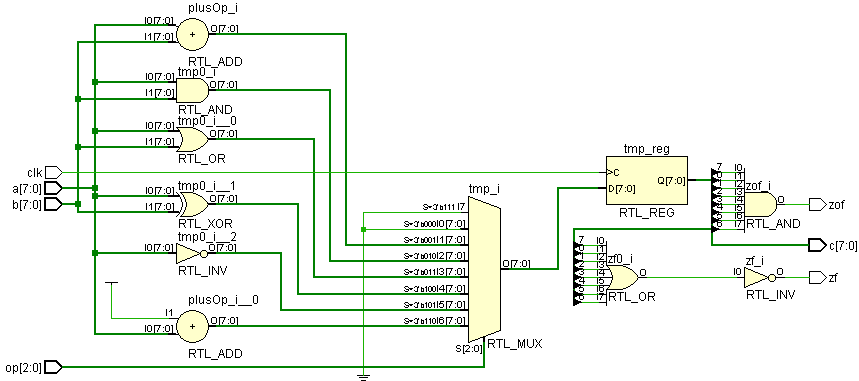
c<=tmp;

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));

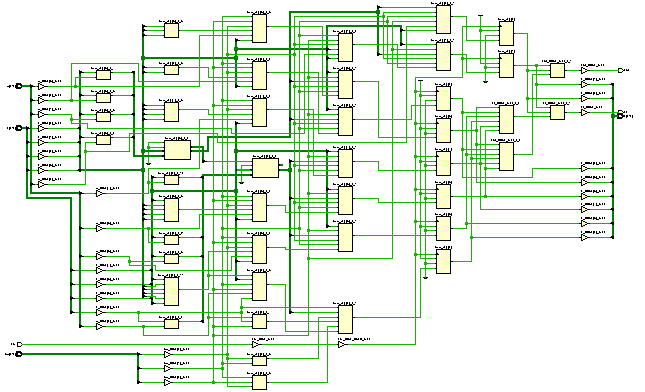
zof <= tmp(7) and tmp(6) and tmp(5) and tmp(4) and tmp(3) and tmp(2) and tmp(1) and tmp(0);

end Behavioral;

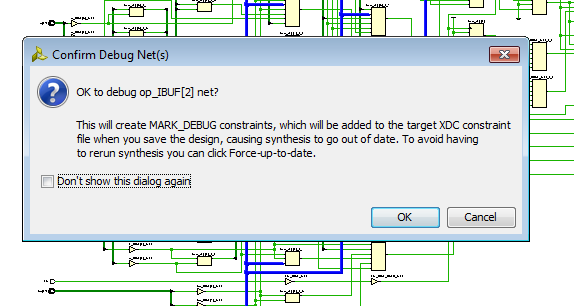
Схема в RTL представлении



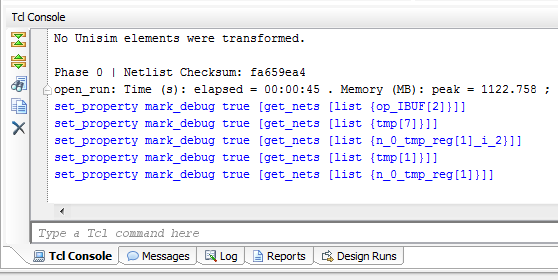
Синтезированная схема. Из sinthesized design



В этом режиме можно перейти в layout Debug, выбрать и отметить как «подлежащие отладке» нужные сигналы.



Сообщения о пометке сигналов как отладочных отражаются в TCL консоли и xdc файле (constraints)



set\_property mark\_debug true [get\_nets [list {op\_IBUF[2]}]]

set\_property mark\_debug true [get\_nets [list {tmp[7]}]]

set\_property mark\_debug true [get\_nets [list {n\_0\_tmp\_reg[1]\_i\_2}]]

set\_property mark\_debug true [get\_nets [list {tmp[1]}]]

set\_property mark\_debug true [get\_nets [list {n\_0\_tmp\_reg[1]}]]

set\_property mark\_debug true [get\_nets [list {op\_IBUF[2]}]]

set\_property mark\_debug true [get\_nets [list {tmp[7]}]]

set\_property mark\_debug true [get\_nets [list {tmp[1]}]]

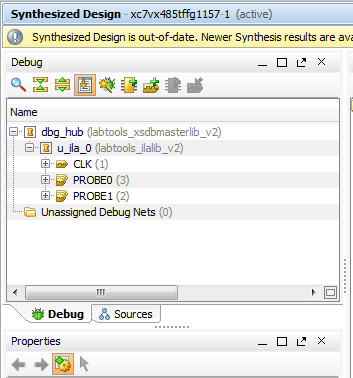
set\_property mark\_debug true [get\_nets [list {n\_0\_tmp\_reg[1]}]]

Указать шины под debug можно и непосредственно в xdc файле.

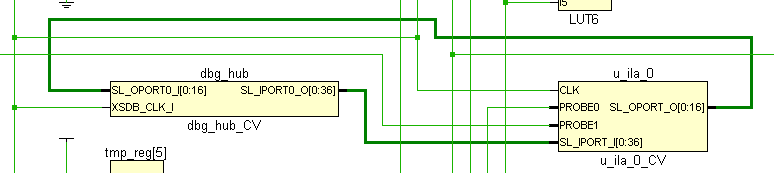
После этого нужно активировать из окна синтеза режим debug:

Open syntez design, Layout- debug

Запускается Set up debug, Implement Debug Core.



Отмеченные как отладочные связи окажутся (после запуска визарда set up debug и имплементации) подключенными к отладочному ядру.



# Лабораторная работа №4. Наблюдение процессов, проходящих в схеме, с помощью Chipscope

**Цель работы:** Отработка навыков использования Chipscope для наблюдения за процессами, происходящими в кристалле.

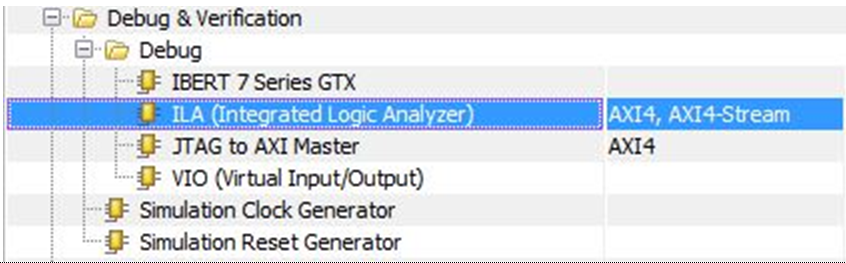
Последовательность выполнения работы.

1. Подключить к проекту отладочное ядро ILA.
2. Подключить ядро виртуального ввода-вывода VIO.
3. Сгенерировать bit-файл.
4. Записать полученный bit-файл в ПЛИС.
5. Провести наблюдение за потоками данных, проходящих в схеме с помощью Vivado или Vivado Lab.

Примечание. Vivado Lab- автономный комплекс программ, предназначенный для прошивки ПЛИС и отладки.

Последовательность выполнения работы.

* + 1. Для подключения IP-ядра ILA, в окне IP-catalog, заходим в директорию Debug & Verification и выбираем ILA:



После синтеза ядра копируем объявление компонента из файла ila\_0.vho и вставляем его в архитектуру проекта:

**COMPONENT ila\_0**

**PORT (**

**clk : IN STD\_LOGIC;**

**probe0 : IN STD\_LOGIC\_VECTOR(8 DOWNTO 0);**

**probe1 : IN STD\_LOGIC\_VECTOR(0 DOWNTO 0)**

**);**

**END COMPONENT;**

Необходимо подключить соответствующие порты (st и count\_BUF):

**ILA : ila\_0**

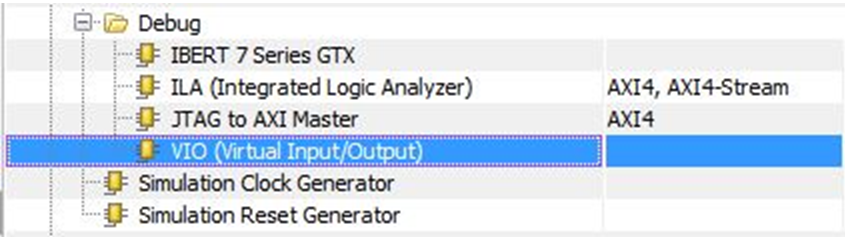
**PORT MAP (**

**clk => clk,**

**probe0 => st,**

**probe1 => count\_BUF);**

2. Для подключения ядра VIO, в окне IP-catalog, заходим в директорию Debug & Verification и выбираем ILA:



После синтеза ядра необходимо скопировать объявление компонента из файла ila\_0.vho и вставить его в основной файл проекта:

**COMPONENT vio\_0**

**PORT (**

**clk : IN STD\_LOGIC;**

**probe\_in0 : IN STD\_LOGIC\_VECTOR(0 DOWNTO 0);**

**probe\_out0 : OUT STD\_LOGIC\_VECTOR(0 DOWNTO 0) );**

**END COMPONENT;**

Подключение соответствующих управляющие сигналов:

**VIO : vio\_0**

**PORT MAP (**

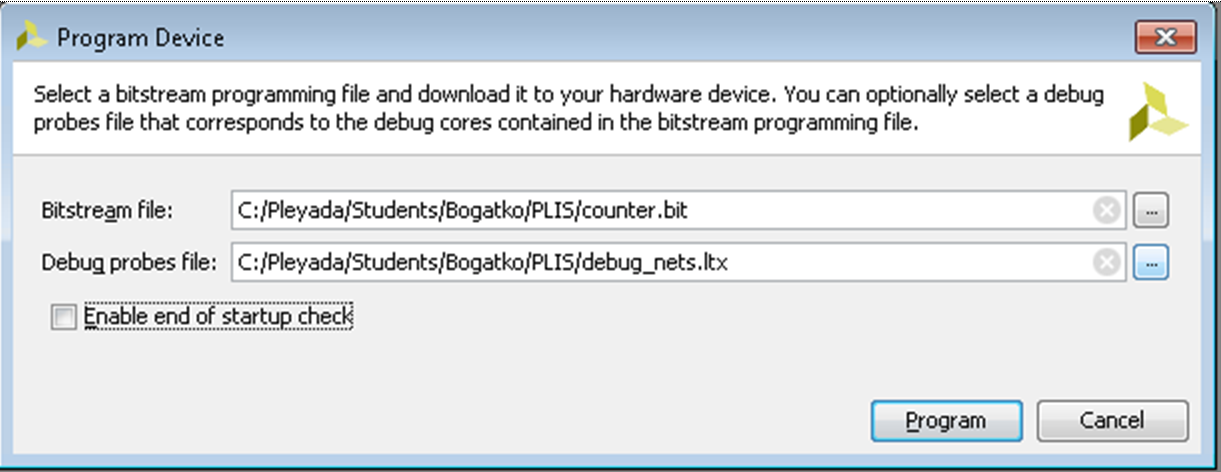
**clk => clk,**

**probe\_in0 => comp,**

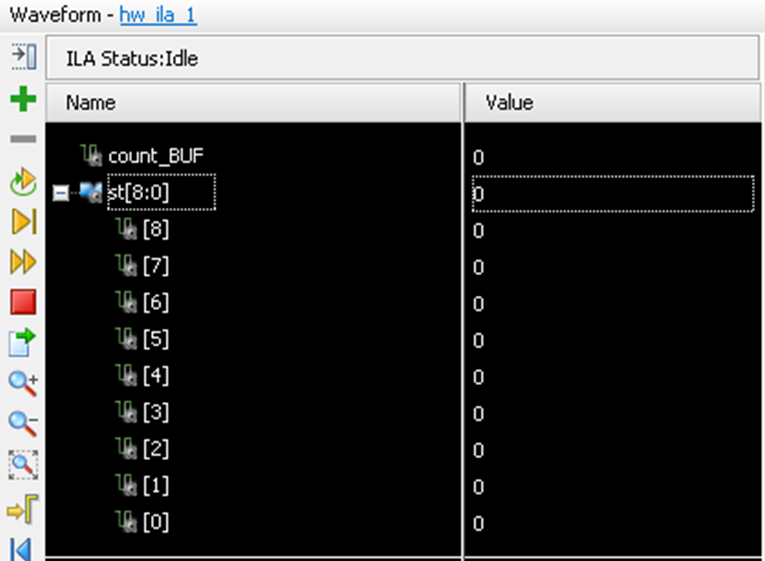
**probe\_out0 => reseto**

);

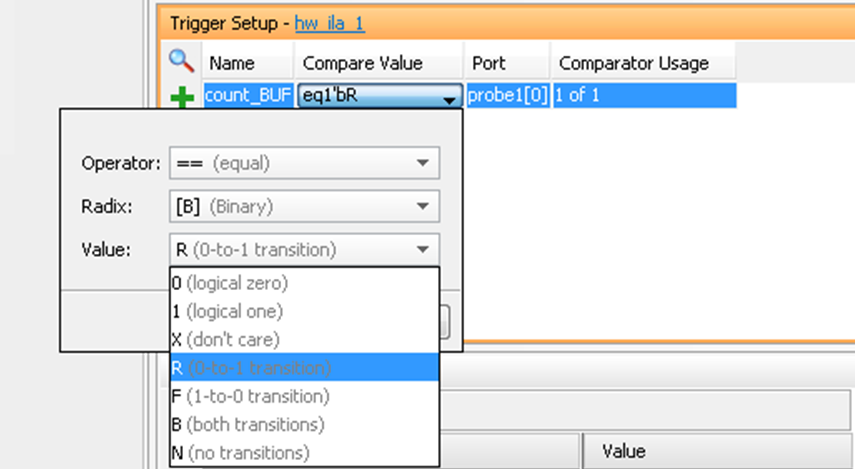
После этого необходимо сгенерировать bitstream и произвести через удаленный экран запись в ПЛИС:



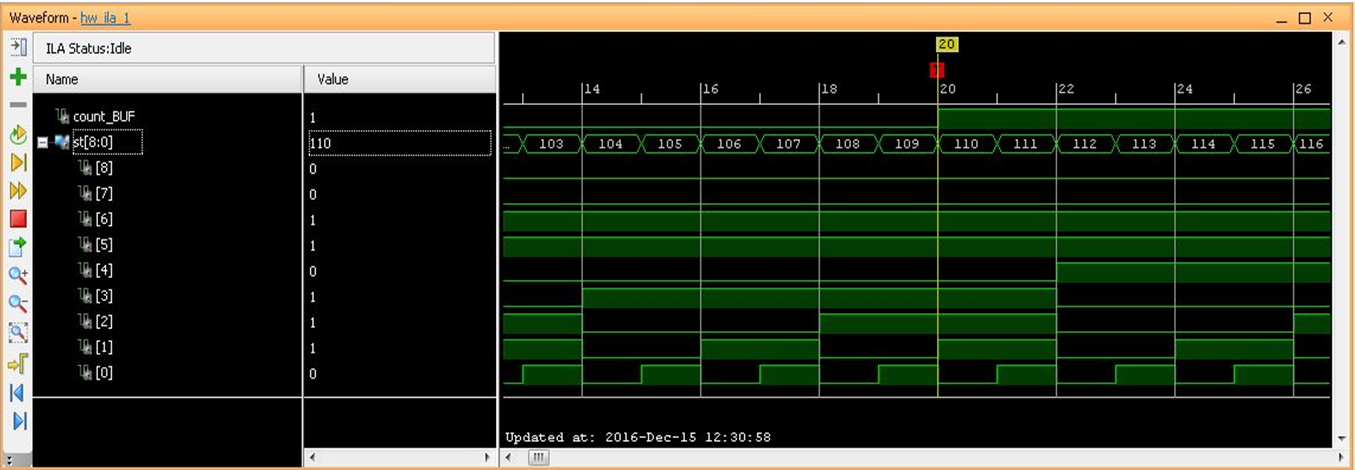
Для того, чтобы пронаблюдать процессы в схеме, для начала с помощью ядра VIO необходимо сбросить схему например подачей единицы на сигнал reseto:



Проводится настройка ядра ILA так, чтобы отследить нужный сигнал (в рассмотренном случае count\_BUF, который сигнализирует о прохождении 109 тактов с момента работы счётчика):



Далее в случае счетчика проводится запуск триггера, подача на reseto сигнала «ноль», чтобы счётчик начал работу и можно наблюдать результат:



Приложение. VHDL и XDC счетчика со встроенными отладочными ядрами

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

library UNISIM;

use UNISIM.VComponents.all;

entity counter is

Port ( CLK1\_DD1\_P : in STD\_LOGIC;

CLK1\_DD1\_N : in STD\_LOGIC;

dat\_out : out std\_logic\_vector(8 downto 0);

mycount : out std\_logic;

out\_zad : out std\_logic;

Q : out std\_logic);

end counter;

architecture Behavioral of counter is

COMPONENT ila\_0

PORT (

clk : IN STD\_LOGIC;

probe0 : IN STD\_LOGIC\_VECTOR(8 DOWNTO 0);

probe1 : IN STD\_LOGIC\_VECTOR(0 DOWNTO 0)

);

END COMPONENT ;

COMPONENT vio\_0

PORT (

clk : IN STD\_LOGIC;

probe\_in0 : IN STD\_LOGIC\_VECTOR(0 DOWNTO 0);

probe\_out0 : OUT STD\_LOGIC\_VECTOR(0 DOWNTO 0)

);

END COMPONENT;

signal CLK : STD\_LOGIC;

signal reseto: std\_logic\_vector(0 downto 0);

signal st: std\_logic\_vector(8 downto 0);

signal comp: std\_logic\_vector(0 downto 0);

signal count\_BUF: std\_logic\_vector(0 downto 0) := (others => '0');

signal a: std\_logic;

signal b: std\_logic;

signal c: std\_logic;

signal d: std\_logic;

signal e: std\_logic;

signal f: std\_logic;

signal g: std\_logic;

begin

CLK\_IBUFDS: IBUFDS

generic map(IOSTANDARD =>"DEFAULT")

port map(

I => CLK1\_DD1\_N,

IB => CLK1\_DD1\_P,

O => CLK

);

ILA : ila\_0

PORT MAP (

clk => clk,

probe0 => st,

probe1 => count\_BUF

);

VIO : vio\_0

PORT MAP (

clk => clk,

probe\_in0 => comp,

probe\_out0 => reseto

);

process(CLK)

begin

if CLK 'event and CLK = '1' then

if (reseto) = "1" then

st <= "000000000";

count\_BUF <= "0";

mycount <= '0';

Q <= '0';

else

if(st) = 230 then

Q <= '1';

else st <= st + 1;

Q <= '0';

end if;

if(st) = 109 then

mycount <= '1';

count\_BUF <= "1";

else

mycount <= '0';

end if;

end if;

end if;

end process;

dat\_out <= st;

process(CLK)

begin

if CLK 'event and CLK = '1' then

a <= st(2);

end if;

end process;

process(CLK)

begin

if CLK 'event and CLK = '1' then

b <= a;

end if;

end process;

process(CLK)

begin

if CLK 'event and CLK = '1' then

c <= b;

end if;

end process;

process(CLK)

begin

if CLK 'event and CLK = '1' then

d <= c;

end if;

end process;

process(CLK)

begin

if CLK 'event and CLK = '1' then

e <= d;

end if;

end process;

process(CLK)

begin

if CLK 'event and CLK = '1' then

f <= e;

end if;

end process;

process(CLK)

begin

if CLK 'event and CLK = '1' then

g <= f;

end if;

end process;

process(CLK)

begin

if CLK 'event and CLK = '1' then

out\_zad <= g;

end if;

end process;

end Behavioral;

set\_property IOSTANDARD LVCMOS18 [all\_outputs]

set\_property IOSTANDARD LVDS [get\_ports CLK1\_DD1\_N]

set\_property DIFF\_TERM TRUE [get\_ports CLK1\_DD1\_N]

set\_property IBUF\_LOW\_PWR FALSE [get\_ports CLK1\_DD1\_N]

set\_property PACKAGE\_PIN AK34 [get\_ports CLK1\_DD1\_N]

set\_property PACKAGE\_PIN AL34 [get\_ports CLK1\_DD1\_P]

set\_property IOSTANDARD LVDS [get\_ports CLK1\_DD1\_P]

set\_property DIFF\_TERM TRUE [get\_ports CLK1\_DD1\_P]

set\_property IBUF\_LOW\_PWR FALSE [get\_ports CLK1\_DD1\_P]

create\_clock -period 4.000 -name clk\_board1 [get\_ports CLK1\_DD1\_P]

set\_property PACKAGE\_PIN BB28 [get\_ports mycount]

set\_property PACKAGE\_PIN AW32 [get\_ports out\_zad]

set\_property PACKAGE\_PIN BB33 [get\_ports Q]

set\_property PACKAGE\_PIN AY34 [get\_ports {dat\_out[0]}]

set\_property PACKAGE\_PIN BA35 [get\_ports {dat\_out[1]}]

set\_property PACKAGE\_PIN AV36 [get\_ports {dat\_out[2]}]

set\_property PACKAGE\_PIN AW36 [get\_ports {dat\_out[3]}]

set\_property PACKAGE\_PIN BA34 [get\_ports {dat\_out[4]}]

set\_property PACKAGE\_PIN BB34 [get\_ports {dat\_out[5]}]

set\_property PACKAGE\_PIN BA36 [get\_ports {dat\_out[6]}]

set\_property PACKAGE\_PIN BB36 [get\_ports {dat\_out[7]}]

set\_property PACKAGE\_PIN BB32 [get\_ports {dat\_out[8]}]

# Рекомендуемая литература

1. 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