Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа компьютерных технологий и информационных систем

Отчёт по лабораторной NIOSII_1

Дисциплина: Автоматизация проектирования дискретных устройств (на английском языке)

Выполнил студент гр. 5130901/10101 _____ М.Т. Непомнящий
(подпись)

Руководитель _____ А.А. Федотов
(подпись)

Санкт-Петербург
2024

Оглавление

1. Задание 5
1.1. Цель работы 5
1.2. Структура проекта 5
2. Ход работы 6
2.1. Создание проекта 6
Начало работы в PD 6
2.2. Настройка сигналов 6
Настройка clk 6
2.3. Подключение сигналов 7
Настройка и подключение On-Chip Memory (RAM and ROM) 7
Настройка и подключение Nios II Processor 8
Настройка и подключение PIO (Parallel I/O) 9
Адресация системы 10
2.4. Анализ полученной системы 11
Анализ предустановок 11
Проверка блока 12
Проверка отсутствия проблемных подключений 12
Анализ с помощью Schematic 13
Генерация системы 13
2.5. Подключение файлов к проекту 15
2.6. Анализ проекта 16
RTL Viewer 16
Назначение выходов проекта 16
2.7. Создание программной части проекта 17
Создание файла Source 17
Компиляция проекта в Eclipse 18
Полная компиляция проекта в QP 19
2.8. Реализация на плате 20
3. Дополнительные задания 21
3.1. Дополнительное задание 1 21
Создание программного файла Lab1_task1_source.c 21
Реализация на плате 21
3.2. Дополнительное задание 2 22
Увеличение разрядности pio_SW 22
Создание программного файла Lab1_task2_source.c 22
Реализация на плате 23
4. Вывод 24

Список иллюстраций

Рис. 1 – Структура проекта 5
Рис. 2 – Создание проекта 6
Рис. 3 – Исходное окно PD 6
Рис. 4 – Настройка компонента clk 6
Рис. 5 – Настройка модуля onchip _mem 7
Рис. 6 – Подключение onchip_mem 7
Рис. 7 – Настройка модуля nios2_PD 8
Рис. 8 – Подключение модуля nios2_PD 8
Рис. 9 – Настройка параметров exception в модуле nios2_PD 9
Рис. 10 - Настройка модуля pio_LED 9
Рис. 11 – Подключение модуля pio_LED 9
Рис. 12 – Настройка модуля pio_SW 10
Рис. 13 – Подключение модуля pio_SW 10
Рис. 14 – Окно Address Map после выполнения автоматической адресации 10
Рис. 15 – Внешний вид созданной системы 11
Рис. 16 – Предустановки системы 11
Рис. 17 – Окно Messages с предупреждением 12
Рис. 18 – Символ системы 12
Рис. 19 – Анализ проблемных подключений 12
Рис. 20 – Show System with QSYS Interconnect 13
Рис. 21 – Schematic 13
Рис. 22 – Предустановки окна Genreration (по умолчанию) 14
Рис. 23 – Проверка генерации HDL 14
Рис. 24 – Подключение файла .qip к проекту 15
Рис. 25 – Синтаксис файла Lab1.sv 15
Рис. 26 – Создание символа файла верхнего уровня 15
Рис. 27 – Выбор файла верхнего уровня 16
Рис. 28 – Схема проекта в RTL Viewer 16
Рис. 29 – Назначение выводов платы средствами Pin Planner 16
Рис. 30 – Unused Pins 17
Рис. 31 – Создание проекта в Eclipse 17
Рис. 32 – Пред настройки Source File’a 17
Рис. 33 – Синтаксис файла Lab1_source.c 18
Рис. 34 – Build Project 18
Рис. 35 – Настройка BSP Editor 18
Рис. 36 – Build Project после изменения настроек BSP Editor 19
Рис. 37 – Окно Make Targets 19
Рис. 38 – Проверка размера и базового адреса памяти 19
Рис. 39 – Добавление meminit.qip в проект 19
Рис. 40 – SDC файл 20
Рис. 41 – Добавление SDC файла 20

Рис. 42 – Временные характеристики устройства 20
Рис. 43 – Изменённый исходный файл 21
Рис. 44 – Изменение настроек модуля pio_SW 22
Рис. 45 – Подключение SW[7:0] (Pin Planner) 22
Рис. 46 – Программный файл с реализацией сложения по модулю SW[7:0] 23

1. Задание

1.1. Цель работы

Познакомиться с процедурой реализации «системы на кристалле» – проекта на базе процессора NIOSII, включая следующие этапы:

✓ Создание проекта в пакете Quartus Prime (QP)

✓ Создание аппаратной части проекта помощью приложения Platform Designer (PD)

✓ Создание программной части проекта в рамках оболочки NIOSII IDE

✓ Проверка работы проекта на плате

1.2. Структура проекта

Процессор NIOSII на светодиодах LED8 … LED1 отображает двоичные коды чисел от 0 до 255, под управлением данных, получаемых с переключателей SW:

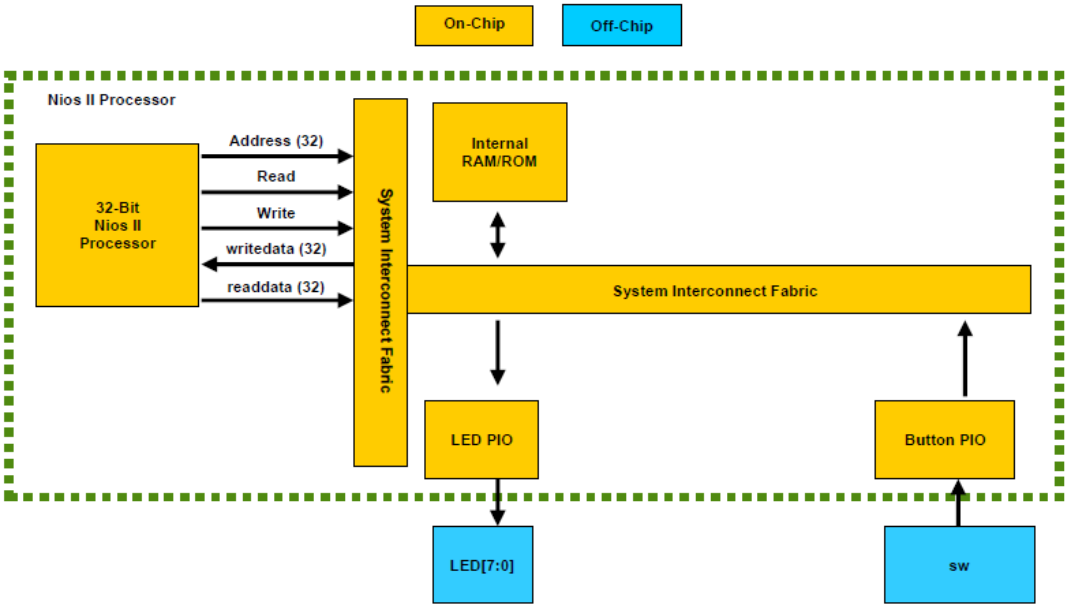


Рис. 1 – Структура проекта

Устройство, которое содержит master и 3 slave: 2 модуля my_slave и 1 модуль my_Dslave (default slave)

Master получает некоторые данные через Conduit, через 8-разрядный интерфейс мастер осуществляет адресный доступ к одному из slave'ов, настраивает соответственно slave's, либо что-то в них записывает, каждый из slave'ов имеет в себе Conduit, который помогает посмотреть на выводе slave'а то, что мы туда записали из мастера.

2. Ход работы

2.1. Создание проекта

Создадим проект, указав параметры, представленные на Рис. 2 ниже:

Summary

When you click Finish, the project will be created with the following settings:

Project directory:	D:\Users\Legion\Quartus and Verilog\Verilog\Third year\6th semester\labs\lab12 - lab_NIOSII_1\Lab1
Project name:	Lab1
Top-level design entity:	Lab1
Number of files added:	0
Number of user libraries added:	0
Device assignments:	
Design template:	n/a
Family name:	Cyclone IV E
Device:	EP4CE6E22C8
Board:	n/a
EDA tools:	
Design entry/synthesis:	<None> (<None>)
Simulation:	<None> (<None>)
Timing analysis:	()
Operating conditions:	
VCCINT voltage:	1.2V
Junction temperature range:	0-85 °C

Рис. 2 – Создание проекта

Начало работы в PD

Откроем PD и сохраним систему:

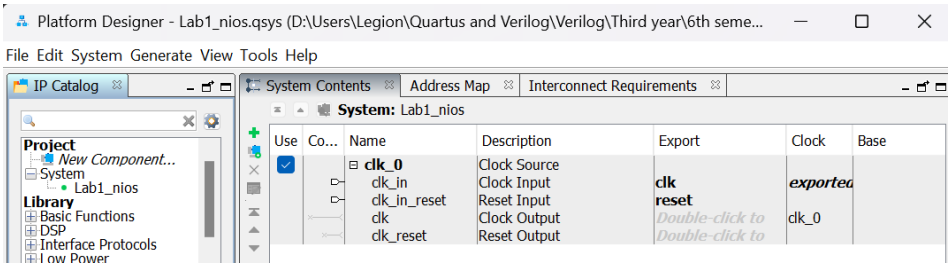


Рис. 3 – Исходное окно PD

2.2. Настройка сигналов

Настройка clk

Зададим значение Clock frequency = 25 MHz (частота кварцевого генератора на плате miniDiLaB-CIV, с которой предстоит работать), а также Reset synchronous edges = Deassert:

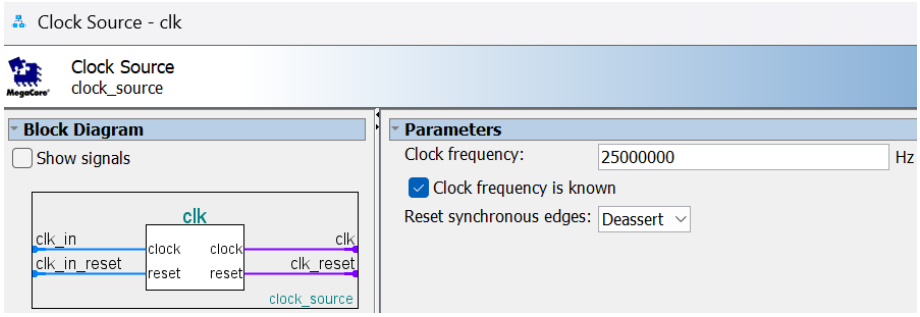


Рис. 4 – Настройка компонента clk

2.3. Подключение сигналов

Настройка и подключение On-Chip Memory (RAM and ROM)

Через окно Library добавим в проект модуль On-Chip Memory (RAM and ROM), переименуем его на onchip _mem и зададим следующие параметры:

- Тип памяти – RAM
- Размер памяти – 16384 байт (16 Кб)
- Остальные галочки и настройки так, как показано на ниже

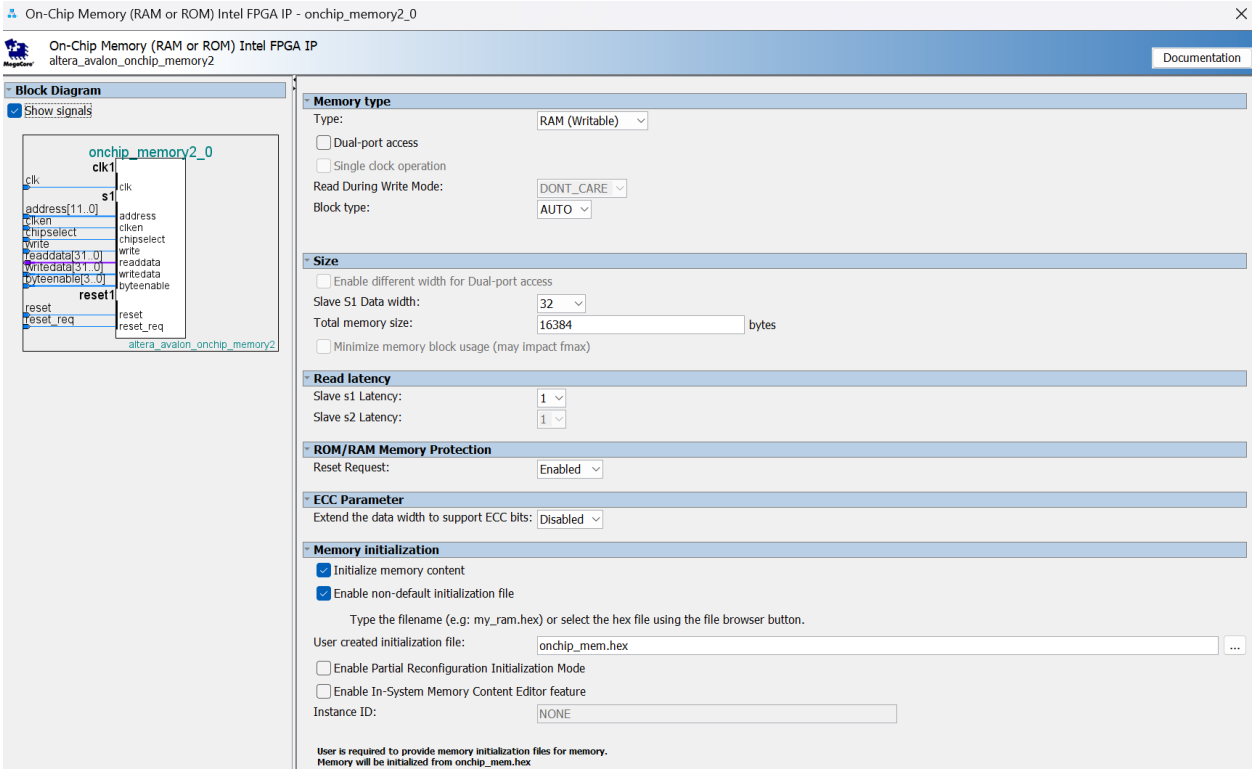


Рис. 5 – Настройка модуля onchip _mem

Заметим, что появившиеся ошибки связаны с тем, что для нормальной работы компонент должен быть подключен к тактовому сигналу, сигналу сброса внутренних регистров и Мастеру на шине Avalon-MM.

Подключим только что добавленный модуль. Для этого соединим выход clk.clk со входом clk1 компонента onchip_mem, а также выход clk.clk_reset со входом onchip_mem.reset1:

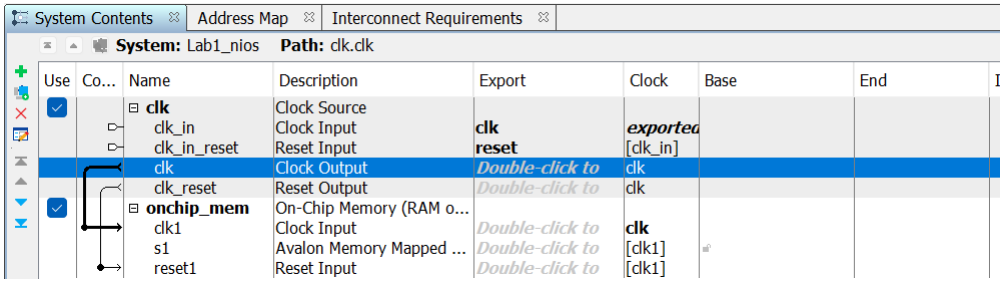


Рис. 6 – Подключение onchip_mem

Как и предполагалось, ошибки исчезли.

Настройка и подключение Nios II Processor

Добавим в систему ядро процессорного модуля, за него отвечает модуль Nios II Processor. Переименуем его на nios2_PD и зададим в его настройках следующие параметры:

- Тип процессора Main → Тип процессора – NIOSII/e (простейший вариант процессорного ядра)
- JTAG DEBUG → отключить режим Include JTAG Debug

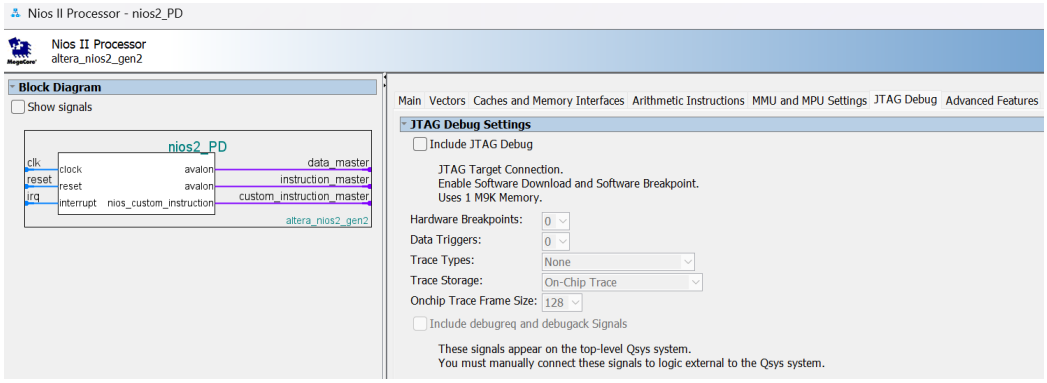


Рис. 7 – Настройка модуля nios2_PD

Подключим только что добавленный модуль. Для этого соединим вход nios2_PD.clk с выходом clk.clk, а также выход clk.clk_reset со входом nios2_PD.reset. Соединим вход onchip_mem.s1 с выходами data_master и instruction_master компонента nios2_PD. Получим следующую систему:

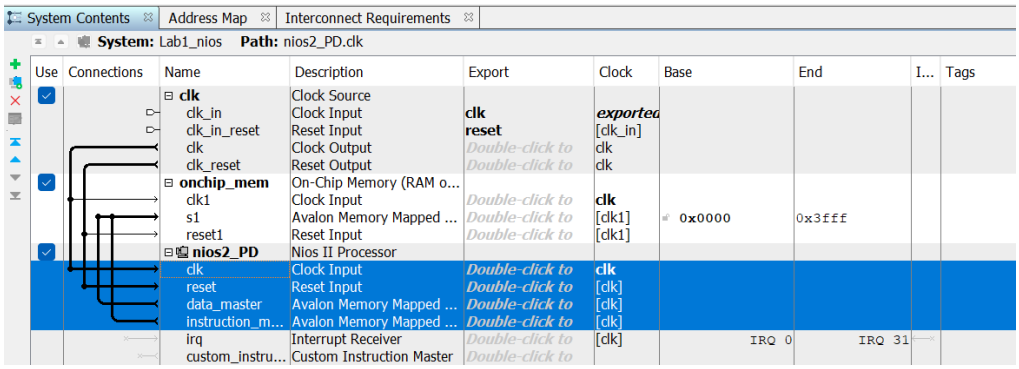


Рис. 8 – Подключение модуля nios2_PD

Дополнительно до настроим процессор NIOS II. Когда мы включаем или сбрасываем устройство, возникает некоторое событие Esception, которое говорит, что процессору нужно начать работать с определённого адреса. По этому адресу будет храниться первоначальная инициализация процессора. Поскольку, сброс, может быть нажат в любой момент, нужно указать следующие параметры:

- Память для вектора сброса
- Память для вектора exception
- JTAG Debug Module

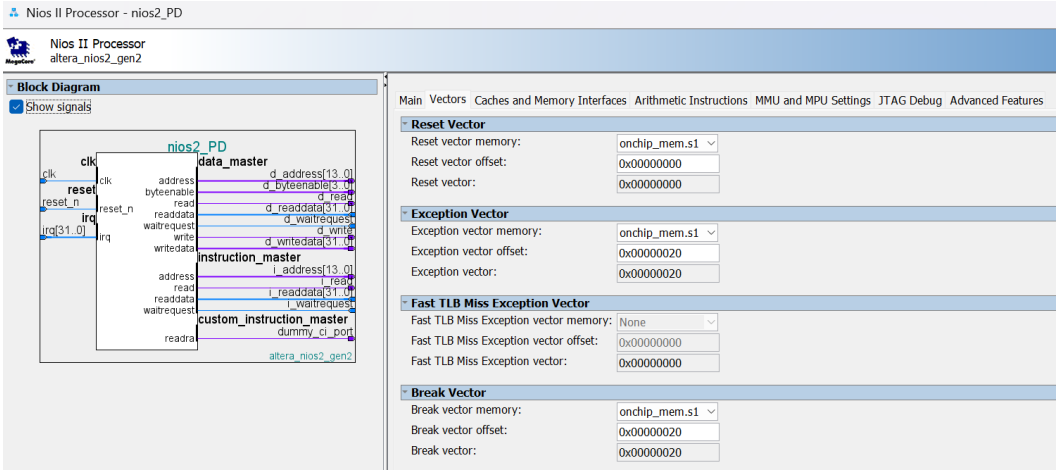


Рис. 9 – Настройка параметров exception в модуле nios2_PD

Настройка и подключение PIO (Parallel I/O)

pio_LED

Через окно Library добавим в проект модуль PIO (Parallel I/O), переименуем его на pio_LED и зададим следующие параметры:

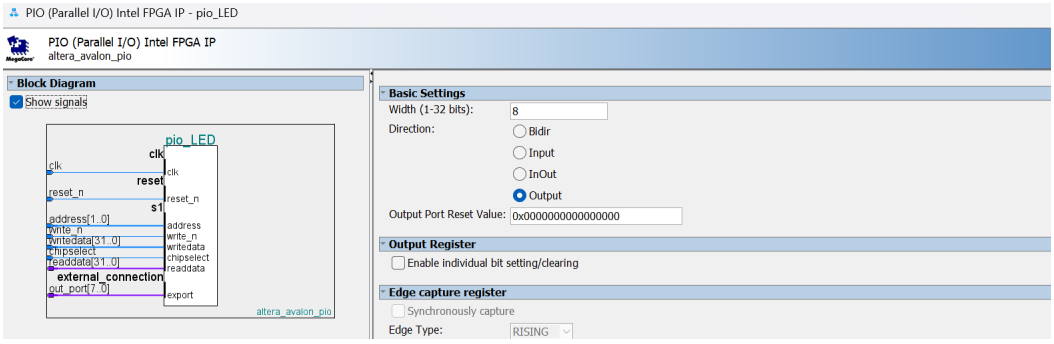


Рис. 10 - Настройка модуля pio_LED

Подключим только что добавленный модуль. Для этого выполним соединения так, как показано на рис. 12 ниже, а также в строке external_connections столбца Export введём имя внешнего вывода – led:

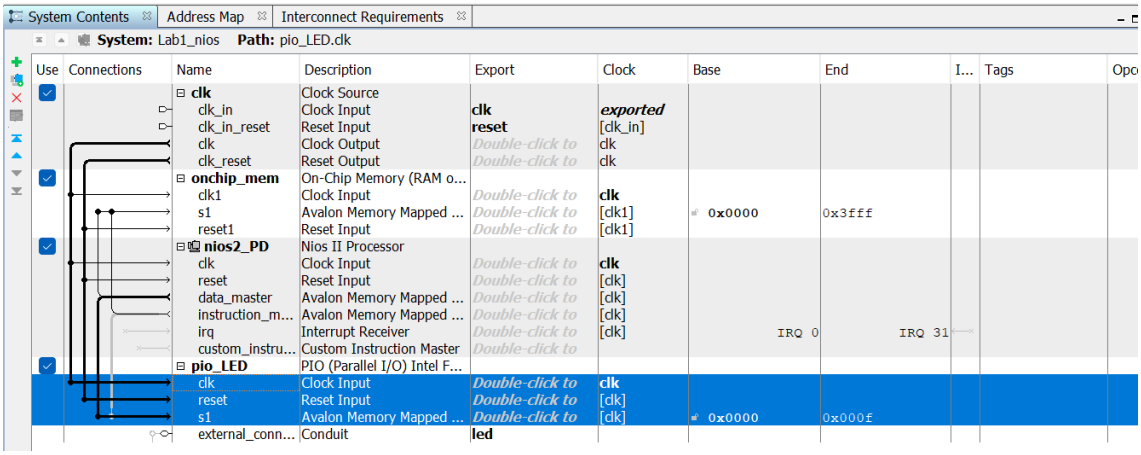


Рис. 11 – Подключение модуля pio_LED

pio_SW

Подключим второй модуль PIO (Parallel I/O), который будет отличаться от первого тем, что он будет иметь разрядность 1, а также направление передачи – Input. Переименуем его на pio_SW. Настройки этого компонента приведены ниже на рис. 13:

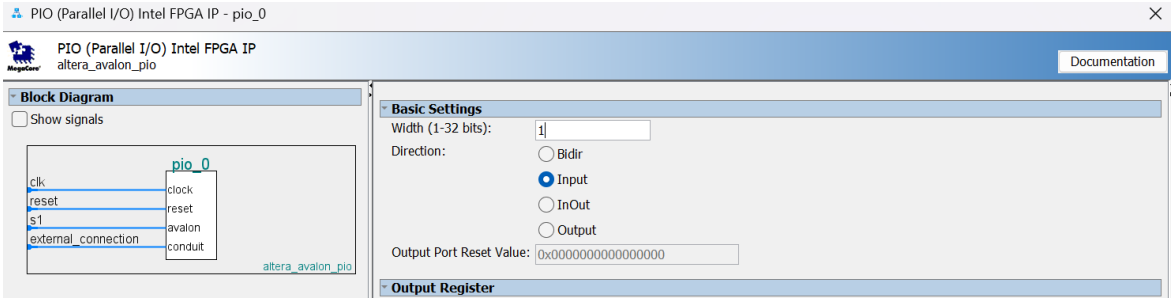


Рис. 12 – Настройка модуля pio_SW

Подключим только что добавленный модуль. Для этого выполним соединения так, как показано на рис. 14 ниже, а также в строке external_connections столбца Export введём имя внешнего вывода – sw:

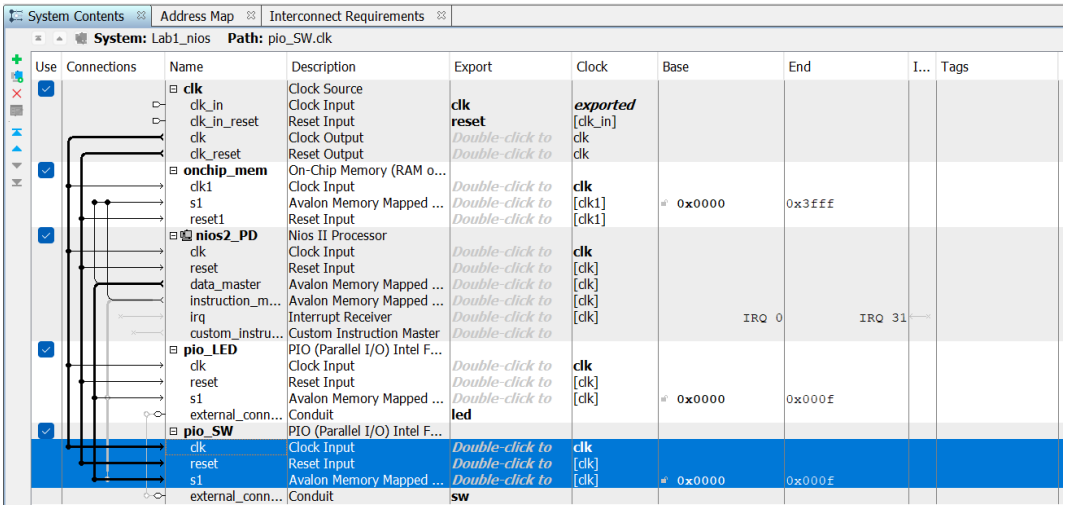


Рис. 13 – Подключение модуля pio_SW

Адресация системы

Выполним System → Assign base Addresses. После автоматической адресации модулей окно Address Map будет выглядеть следующим образом:

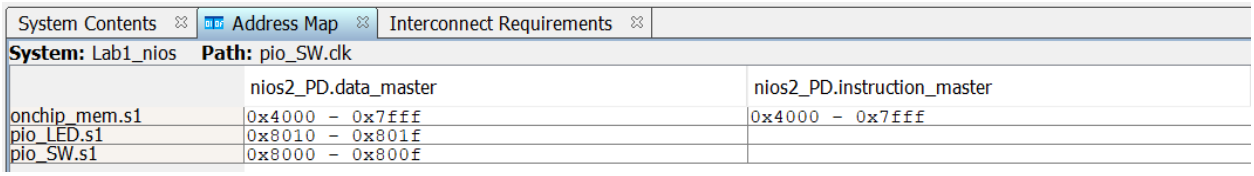


Рис. 14 – Окно Address Map после выполнения автоматической адресации

2.4. Анализ полученной системы

Полученная система будет выглядеть так, как показано на ниже:

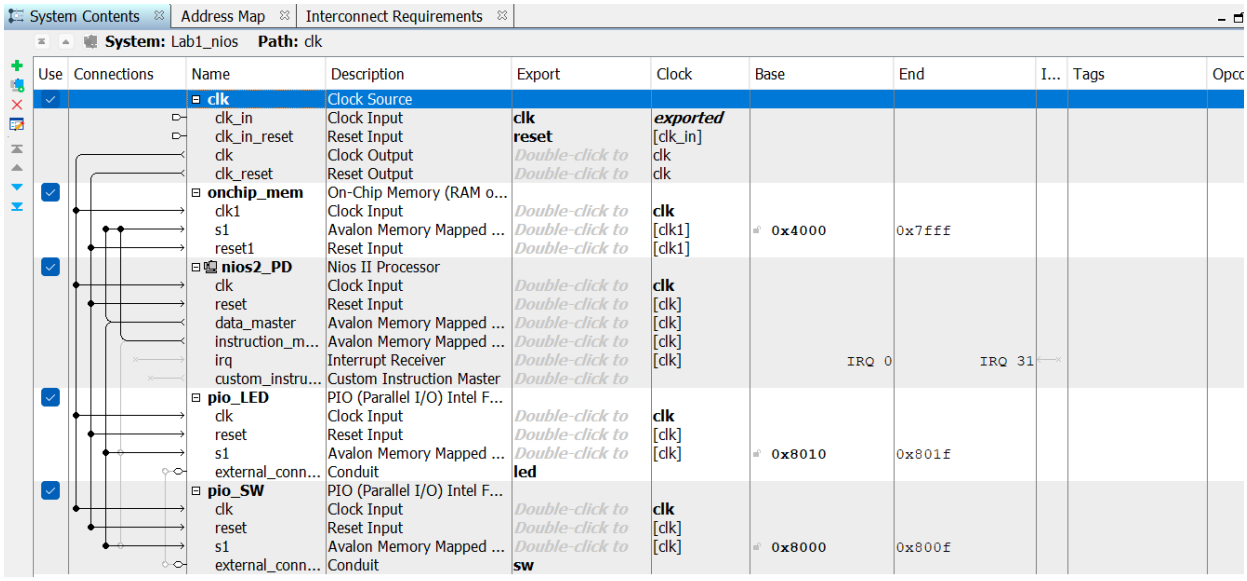


Рис. 15 – Внешний вид созданной системы

Анализ предустановок

Проверим, что предустановки для полученной системы указаны так, как показано на рис. 17 ниже:

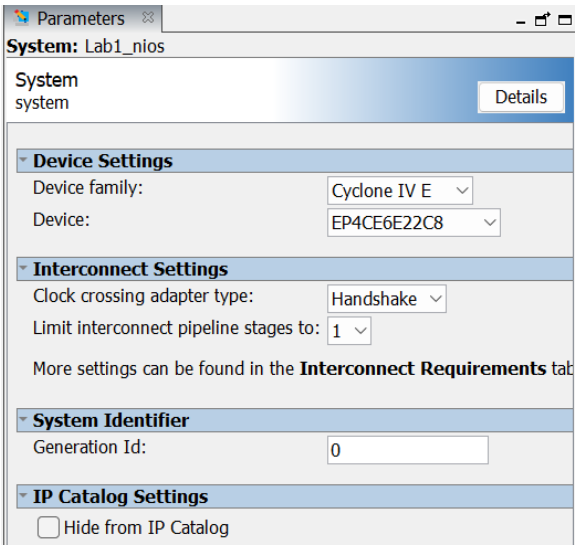


Рис. 16 – Предустановки системы

В окне Messages есть только 1 предупреждение, связанное с тем, что не подключён JTAG Debug модуль, однако это было сделано намеренно, поэтому на это предупреждение можем не обращать внимание:

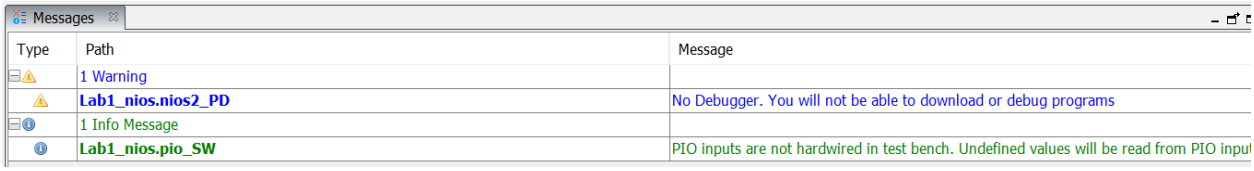


Рис. 17 – Окно Messages с предупреждением

Проверка блока

Выполним View → Block Symbol и убедимся в том, что символ системы построен правильно:

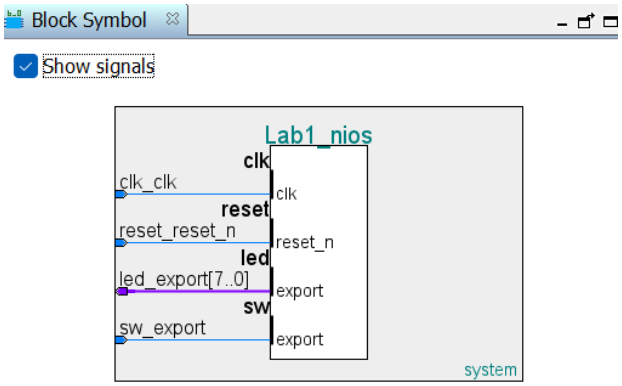


Рис. 18 – Символ системы

Проверка отсутствия проблемных подключений

Выполним View → Clock domains Beta, выберем режим отображения Reset, затем Clock. Заметим, что проблемных подключений ни в одном из случаев не выявлено:

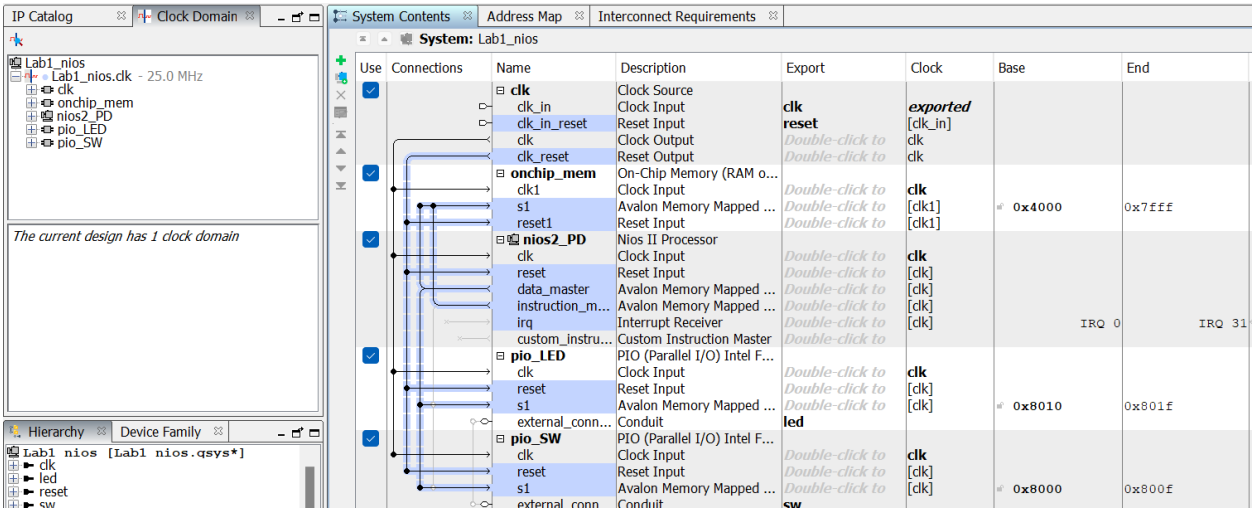


Рис. 19 – Анализ проблемных подключений

Выполним команду System → Show System with PD Interconnect (Show System with QSYS Interconnect). Проверим, был добавлен только модуль mm_interconnect_0.

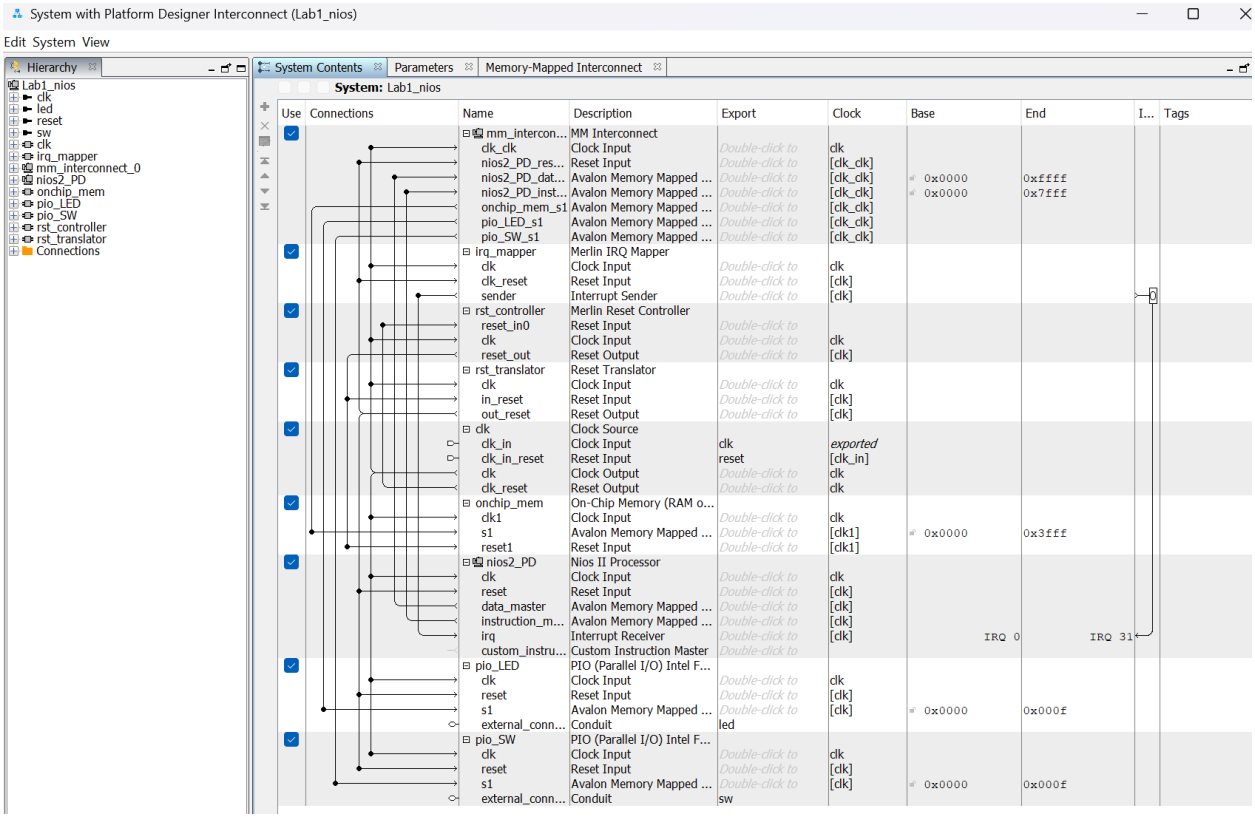


Рис. 20 – Show System with QSYS Interconnect

Анализ с помощью Schematic

Выполним View → Schematic, в качестве фильтра введём in и убедимся в том, что система синхронизации и каналы ST системы подключены верно:

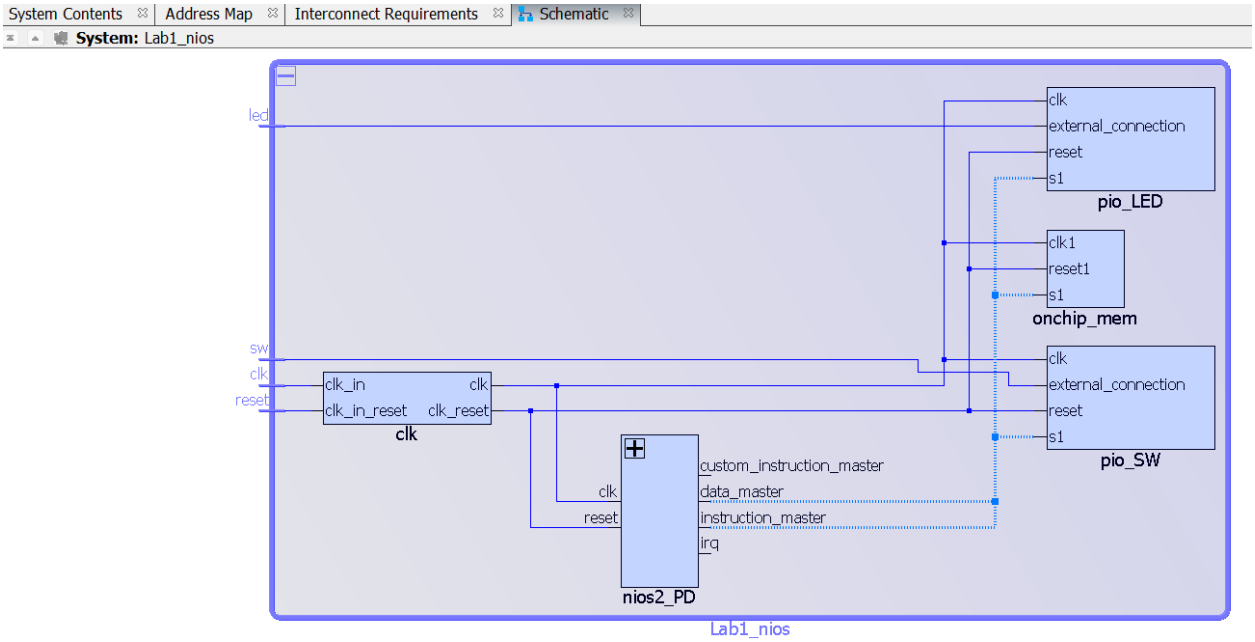


Рис. 21 – Schematic

Генерация системы

Выполним PD → Generate HDL и укажем следующие предустановки для генерации:

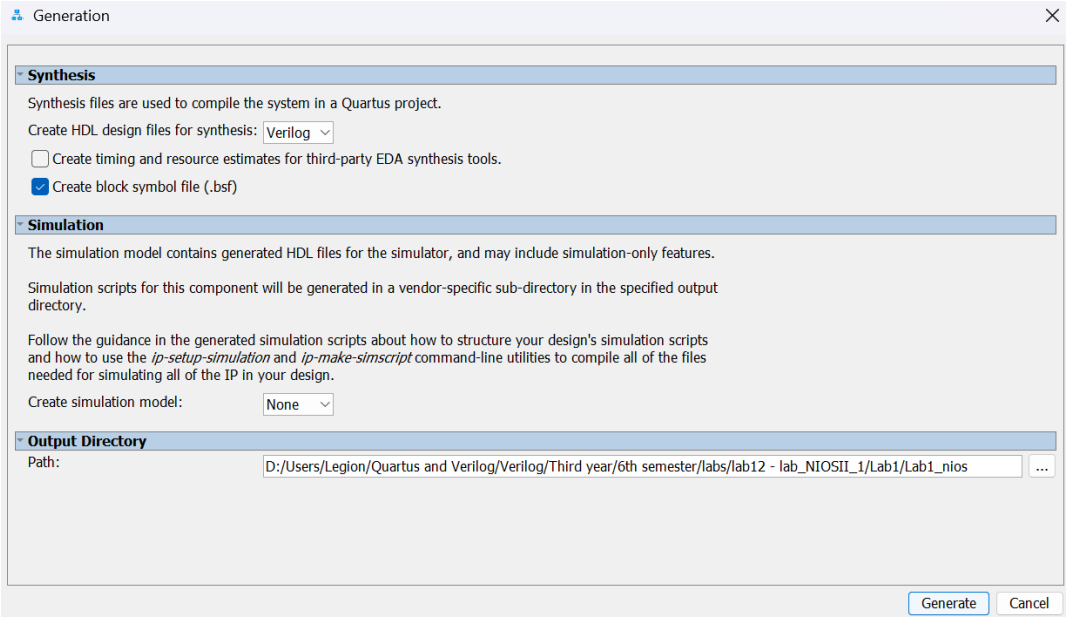


Рис. 22 – Предустановки окна Genreration (по умолчанию)

Удостоверимся в том, что среди предупреждений есть только 1 пункт, связанный с JTAG, про сто говорилось выше:

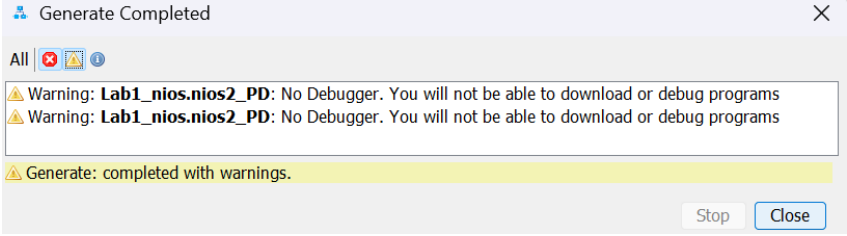


Рис. 23 – Проверка генерации HDL

2.5. Подключение файлов к проекту

Подключим файл .qip только что созданной системы к проекту в Quartus

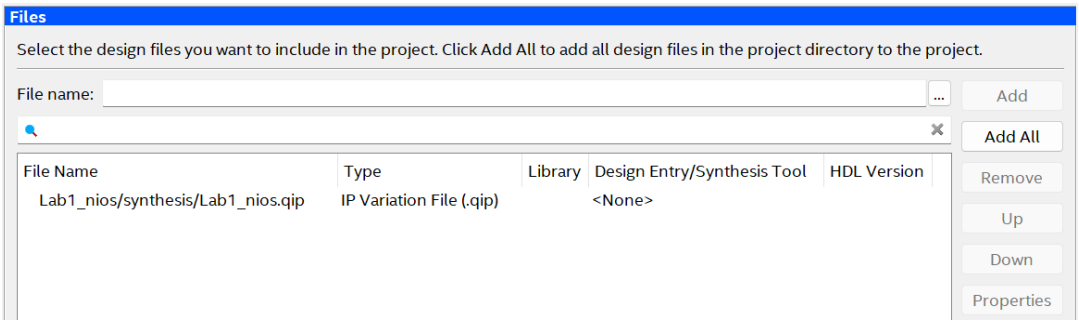


Рис. 24 – Подключение файла .qip к проекту

Создадим файл верхнего уровня Lab1.sv, синтаксис которого приведён ниже:

Lab1 - Lab1.sv

```
module Lab1 (
    input bit clk, // Clock
    input bit sw,  // data in
    input bit pbb, // Asynchronus reset active low
    output bit [7:0] led
);
    Lab1_nios u0 (
        .clk_clk       (<connected-to-clk_clk>),       //   clk.clk
        .reset_reset_n (<connected-to-reset_reset_n>), // reset.reset_n
        .led_export    (<connected-to-led_export>),    //   led.export
        .sw_export     (<connected-to-sw_export>)      //    sw.export
    );

endmodule

```

Рис. 25 – Синтаксис файла Lab1.sv

Этот файл был создан отдельно, но не добавлен в проект. Создадим этот же файл, но с помощью графического редактора:

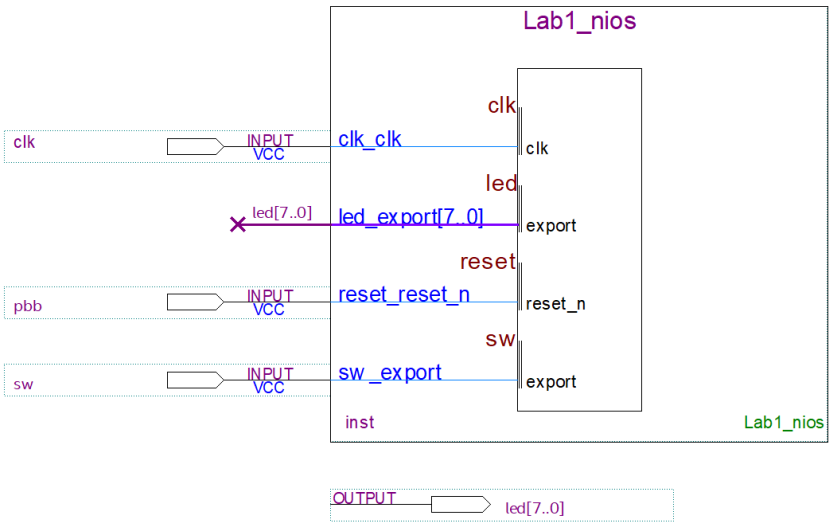


Рис. 26 – Создание символа файла верхнего уровня

Сохраним схему на Рис. 27 под именем lab1.bdf

2.6. Анализ проекта

В иерархии проекта выберем файл верхнего уровня:

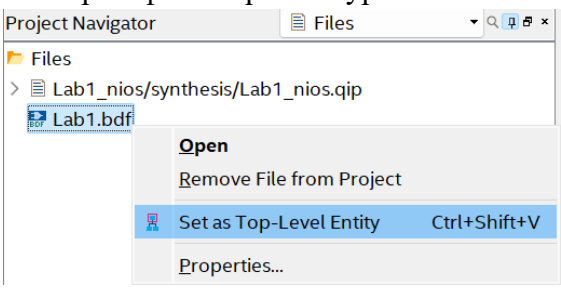


Рис. 27 – Выбор файла верхнего уровня

RTL Viewer

Выполним анализ и синтез проекта средствами QP и убедимся в правильности схемы средствами RTL Viewer:

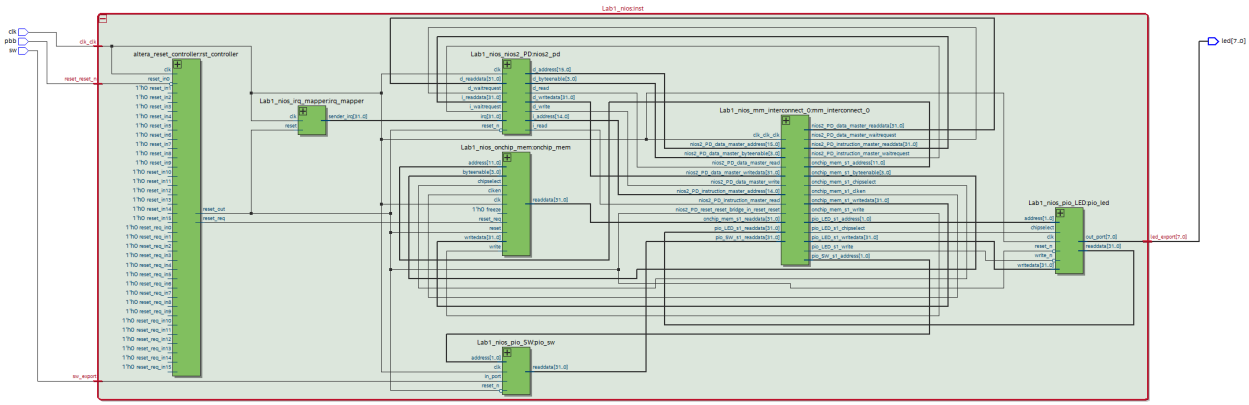


Рис. 28 – Схема проекта в RTL Viewer

Можем увидеть, что полученная в RTL Viewer схема совпадает с той, что была задана по условию (в зелёном блоке отображается тот фрагмент системы, который был создан средствами PD).

Назначение выходов проекта

Откроем редактор назначения выводов (Pin Planner): Assignment → Pin Planner. Назначим выводы на плату так, как показано на ниже:

Named: * Edit:

Node Name	Direction	Location	I/O Bank	/REF Group	I/O Standard	Reserved	rent Stren	Slew Rate	fferential P	ct Preserva
clk	Input	PIN_23	1	B1_N0	2.5 V (default)		8mA ...ult)			
led[7]	Output	PIN_65	4	B4_N0	2.5 V (default)		8ma	2 (default)		
led[6]	Output	PIN_66	4	B4_N0	2.5 V (default)		8ma	2 (default)		
led[5]	Output	PIN_67	4	B4_N0	2.5 V (default)		8ma	2 (default)		
led[4]	Output	PIN_68	4	B4_N0	2.5 V (default)		8ma	2 (default)		
led[3]	Output	PIN_69	4	B4_N0	2.5 V (default)		8ma	2 (default)		
led[2]	Output	PIN_70	4	B4_N0	2.5 V (default)		8ma	2 (default)		
led[1]	Output	PIN_71	4	B4_N0	2.5 V (default)		8ma	2 (default)		
led[0]	Output	PIN_72	4	B4_N0	2.5 V (default)		8ma	2 (default)		
pbb	Input	PIN_58	4	B4_N0	2.5 V (default)		8ma			
sw	Input	PIN_24	2	B2_N0	2.5 V (default)		8mA ...ult)			
<<new node>>										

Рис. 29 – Назначение выводов платы средствами Pin Planner

Выполним Assignment → Device → Device and Pin Options → Unused Pins, в появившемся окне установим опцию As input tri-started with weak pull-up:

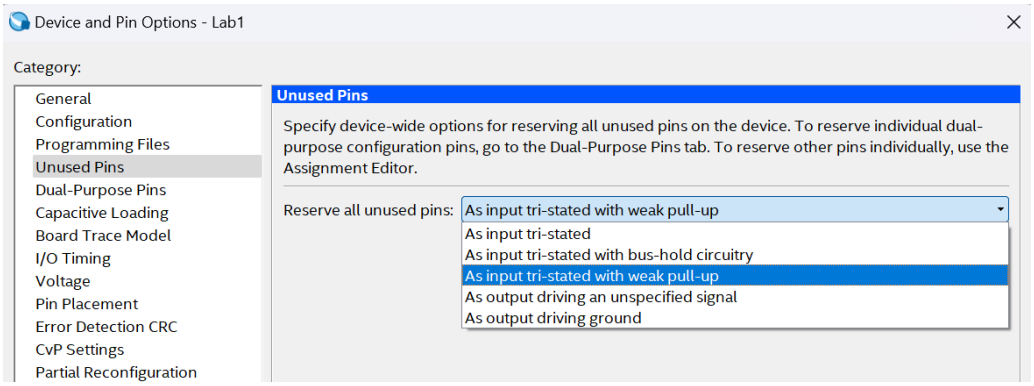


Рис. 30 – Unused Pins

2.7. Создание программной части проекта

Создадим оболочку Nios II SBT (Software Build Tools) средствами Eclipse.

Укажем файл с описанием программы lab1_nios.spocinfo. В качестве названия проекта укажем Lab1_sw.

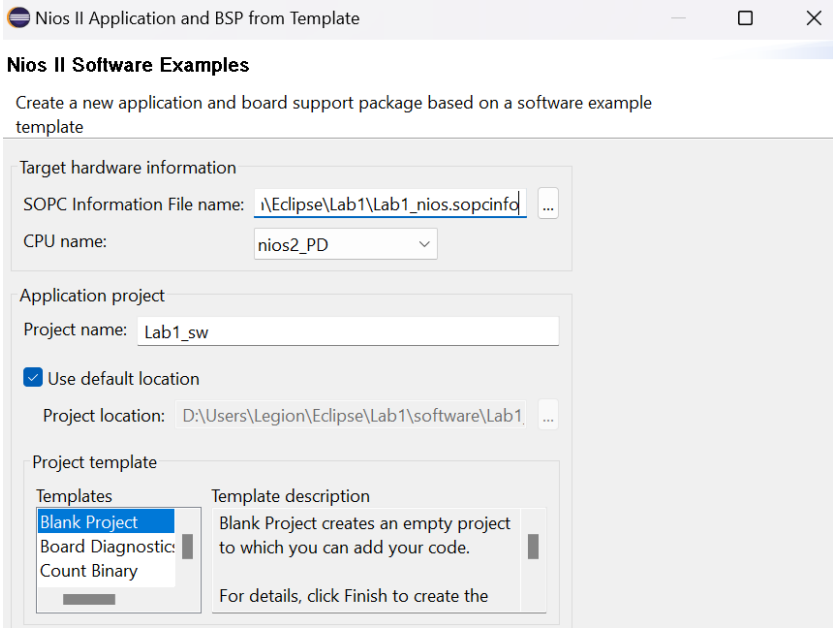


Рис. 31 – Создание проекта в Eclipse

Создание файла Source

Создадим Source File:

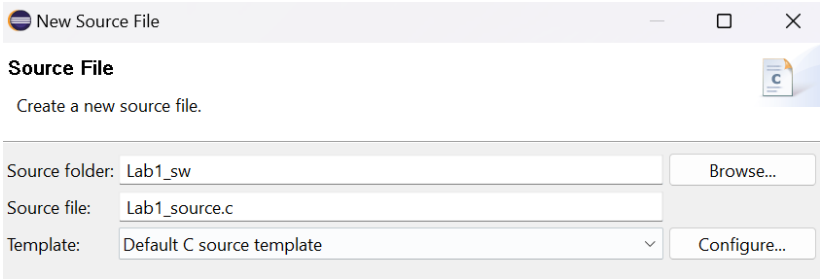


Рис. 32 – Пред настройки Source File’a

```c
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include <unistd.h>
int main(void)
{
int sw;
int count = 255;
while( 1 )
{
usleep (500000);
sw = IORD_ALTERA_AVALON_PIO_DATA(PIO_SW_BASE);/* read sw[0] value */
if (sw == 0x1) count++; /* Continue 0-ff counting loop. */
else count--; /* Continue ff-0 counting loop. */
IOWR_ALTERA_AVALON_PIO_DATA( PIO_LED_BASE, ~count );
}
return 0;
}
```

Рис. 33 – Синтаксис файла Lab1_source.c

Компиляция проекта в Eclipse

Выполним Lab1_sw → Build Project:

```
Info: Linking Lab1_sw.elf
nios2-elf-g++  -T'../Lab1_sw_bsp//linker.x' -msys-crt0='../Lab1_sw_bsp//obj/HAL/src/crt0.o' -msys-lib=hal_bsp -
nios2-elf-insert Lab1_sw.elf --thread_model hal --cpu_name nios2_PD --qsys true --simulation_enabled false --st
Info: (Lab1_sw.elf) 4572 Bytes program size (code + initialized data).
Info:               10 KBytes free for stack + heap.
Info: Creating Lab1_sw.objdump
nios2-elf-objdump --disassemble --syms --all-header --source Lab1_sw.elf >Lab1_sw.objdump
[Lab1_sw build complete]

00:21:09 Build Finished (took 11s.175ms)
```

Рис. 34 – Build Project

При создании платформы, программой и данными инициализации занято 4568 Байт, свободно 10 кБ.

Выполним Lab1_sw_bsp → Nios II Lab1 → BSP Editor и в окне настроек зададим параметры следующим образом:

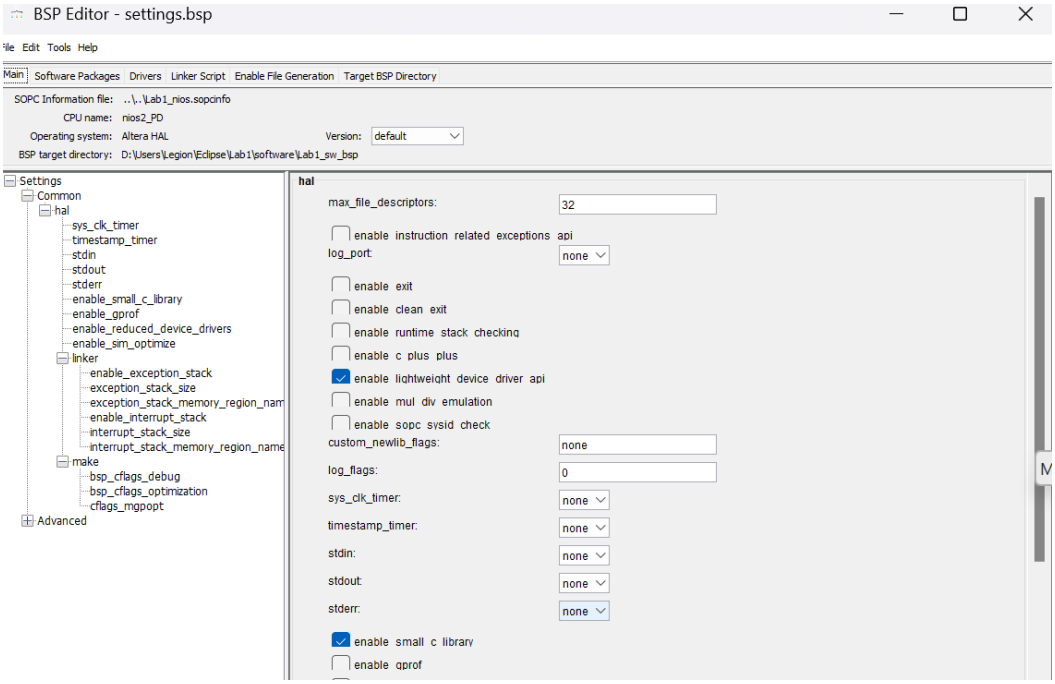


Рис. 35 – Настройка BSP Editor

Скомпилируем проект. Для этого выполним Lab1_sw → Build Project, проверим, что останется больше свободной памяти:

```
Problems  Tasks  Console  Properties  Progress
CDT Build Console [Lab1_sw]
Info: Linking Lab1_sw.elf
nios2-elf-g++  -T'../Lab1_sw_bsp//linker.x' -msys-crt0='../Lab1_sw_bsp//obj/HAL/src/crt0.o' -msys-lib=hal_bsp -
nios2-elf-insert Lab1_sw.elf --thread_model hal --cpu_name nios2_PD --qsys true --simulation_enabled false --st
Info: (Lab1_sw.elf) 1588 Bytes program size (code + initialized data).
Info:               14 KBytes free for stack + heap.
Info: Creating Lab1_sw.objdump
nios2-elf-objdump --disassemble --syms --all-header --source Lab1_sw.elf >Lab1_sw.objdump
[Lab1_sw build complete]

00:56:36 Build Finished (took 11s.754ms)
```

Рис. 36 – Build Project после изменения настроек BSP Editor

Заметим, что памяти, оставшейся свободной (из ОЗУ 16 Кбайт, указанных при создании платформы, программой и данными инициализации занято 1548 Байт (было - 4568 Байт), свободно 14кБайт (было - 10кБайт).

Выполним Lab1_sw → Make Targets → Build:

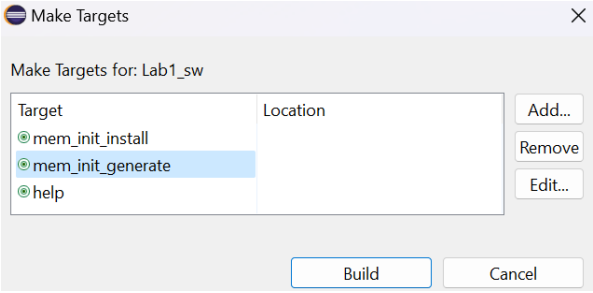


Рис. 37 – Окно Make Targets

Компиляция прошла успешно, проверим, что размер и базовый адрес памяти совпадают с теми, что были указаны ранее при создании системы в пакете PD:

```
[BSP build complete]
Post-processing to create mem_init/onchip_mem.hex...
elf2hex Lab1_sw.elf 0x00004000 0x00007fff --width=32 --little-endian-mem --create-lanes=0 mem_init/onchip_mem.hex
Post-processing to create mem_init/hdl_sim/onchip_mem.dat...
elf2dat --infile=Lab1_sw.elf --outfile=mem_init/hdl_sim/onchip_mem.dat \
        --base=0x00004000 --end=0x00007fff --width=32 \
        --little-endian-mem --create-lanes=0
Post-processing to create mem_init/hdl_sim/onchip_mem.sym...
nios2-elf-nm -n Lab1_sw.elf > mem_init/hdl_sim/onchip_mem.sym
Post-processing to create mem_init/meminit.spd...
Post-processing to create mem_init/meminit.qip...

09:58:24 Build Finished (took 1s.742ms)
```

Рис. 38 – Проверка размера и базового адреса памяти

Как видим, все значения совпали с теми, что были указаны ранее.

Полная компиляция проекта в QP

Добавим только что созданный файл meminit.qip в проект:

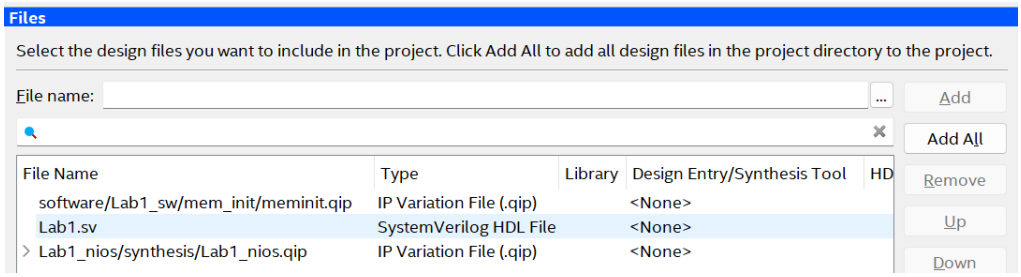


Рис. 39 – Добавление meminit.qip в проект

Создадим Lab1.sdc файл с временными требованиями так, как показано на Рис. 40 ниже:

```
##
## DEVICE "EP4CE6E22C8"
##


#**************************************************************
# Time Information
#**************************************************************

set_time_format -unit ns -decimal_places 3

#**************************************************************
# Create Clock
#**************************************************************

create_clock -name {clock} -period 40.000 -waveform { 0.000 20.000} [get_ports {clk}]

#**************************************************************
# Set Clock Uncertainty
#**************************************************************

set_clock_uncertainty -rise_from [get_clocks {clock}] -rise_to [get_clocks {clock}] 0.020
set_clock_uncertainty -rise_from [get_clocks {clock}] -fall_to [get_clocks {clock}] 0.020
set_clock_uncertainty -fall_from [get_clocks {clock}] -rise_to [get_clocks {clock}] 0.020
set_clock_uncertainty -fall_from [get_clocks {clock}] -fall_to [get_clocks {clock}] 0.020


#**************************************************************
# Set Input Delay
#**************************************************************

set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {pbb}]
set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {sw}]


#**************************************************************
# Set Output Delay
#**************************************************************

set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[0]}]
set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[1]}]
set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[2]}]
set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[3]}]
set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[4]}]
set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[5]}]
set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[6]}]
set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[7]}]
```

Рис. 40 – SDC файл

Добавим файл к проекту:

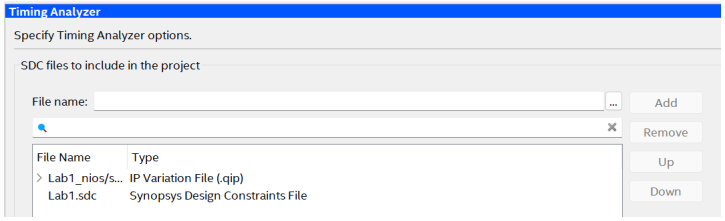


Рис. 41 – Добавление SDC файла

Выполним полную компиляцию. В отчете о компиляции видно, что устройство удовлетворяет временным параметрам.

Slow 1200mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	50.94 MHz	50.94 MHz	clock	

Рис. 42 – Временные характеристики устройства

2.8. Реализация на плате

Проект был загружен на плату, светодиоды LED8-LED1 отображали последовательное увеличение счётчика от 0 до 255 при sw[0] в положении 1, а при переключении sw[0] в положение 0 счёт происходил в обратном порядке с тем же шагом от 255 до 0. Работа на стенде была продемонстрирована преподавателю.

3. Дополнительные задания

3.1. Дополнительное задание 1

Создание программного файла Lab1_task1_source.c

Создадим отдельный проект аналогичный предыдущему, но изменим в нём файл с исходным кодом Lab1_task1_source.c. Будем использовать указатели в качестве адресов, а также увеличим частоту переключения светодиодов засчёт уменьшения значения функции usleep, изменим начальное значение счётчика

lab12 - lab_NIOSII_1 - Lab1_task1_source.c

```c
#include <unistd.h>

int main(void)
{
    int *psw = (int*) 0x8000;
    int *pled = (int*) 0x8010;
    int count = 64;

    while(1)
    {
        usleep(100000);
        if (*psw == 0x1)
            count++; /* Continue 0-ff counting loop. */
        else
            count--; /* Continue ff-0 counting loop. */

        *pled = ~count;
    }

    return 0;
}

```

Рис. 43 – Изменённый исходный файл

Реализация на плате

Как и ожидалось, визуально можем наблюдать корректную работу счётчика и правильное отображение на лабораторном стенде, аналогично тому, что было описано в основном задании проекта. Визуально заметно, что переключение светодиодов, засчёт увеличения частоты, происходит быстрее.

3.2. Дополнительное задание 2

Создадим второй проект, где в системе изменим разрядность SW с одного разряда до 8 разрядов, подключим к переключателям SW[7:0].

Увеличение разрядности pio_SW

Изменим разрядность модуля pio_SW с 1 на 8:

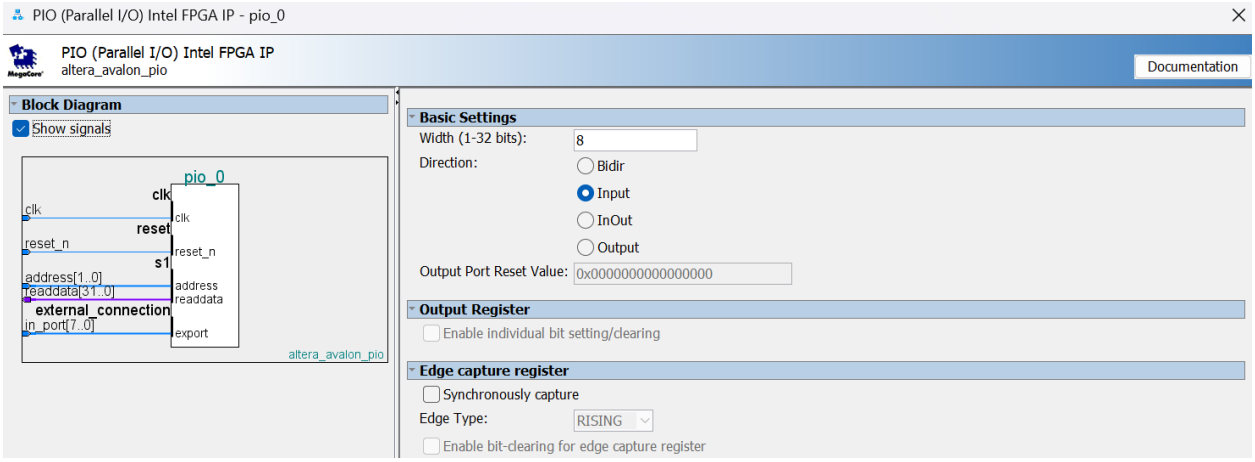
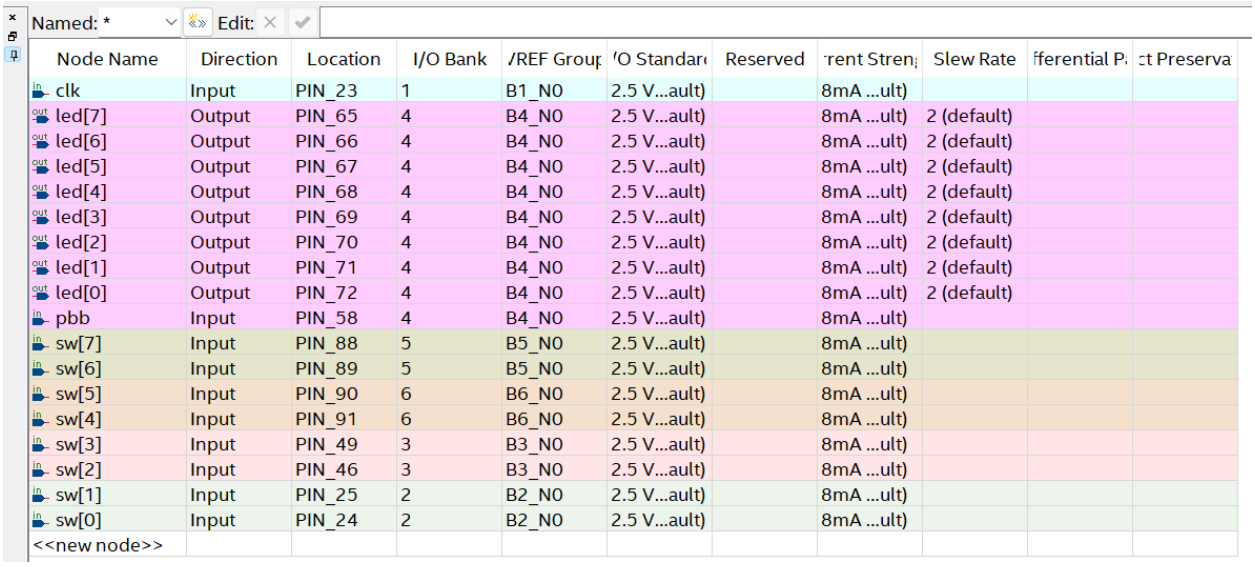


Рис. 44 – Изменение настроек модуля pio_SW

Подключим новые sw к переключателям SW[7:0] средствами Pin Planner:



Node Name	Direction	Location	I/O Bank	/REF Grou	'O Standar	Reserved	rent Stren	Slew Rate	fferential P	ct Preserva
clk	Input	PIN_23	1	B1_N0	2.5 V...ault)		8mA ...ult)			
led[7]	Output	PIN_65	4	B4_N0	2.5 V...ault)		8mA ...ult)	2 (default)		
led[6]	Output	PIN_66	4	B4_N0	2.5 V...ault)		8mA ...ult)	2 (default)		
led[5]	Output	PIN_67	4	B4_N0	2.5 V...ault)		8mA ...ult)	2 (default)		
led[4]	Output	PIN_68	4	B4_N0	2.5 V...ault)		8mA ...ult)	2 (default)		
led[3]	Output	PIN_69	4	B4_N0	2.5 V...ault)		8mA ...ult)	2 (default)		
led[2]	Output	PIN_70	4	B4_N0	2.5 V...ault)		8mA ...ult)	2 (default)		
led[1]	Output	PIN_71	4	B4_N0	2.5 V...ault)		8mA ...ult)	2 (default)		
led[0]	Output	PIN_72	4	B4_N0	2.5 V...ault)		8mA ...ult)	2 (default)		
pbb	Input	PIN_58	4	B4_N0	2.5 V...ault)		8mA ...ult)			
sw[7]	Input	PIN_88	5	B5_N0	2.5 V...ault)		8mA ...ult)			
sw[6]	Input	PIN_89	5	B5_N0	2.5 V...ault)		8mA ...ult)			
sw[5]	Input	PIN_90	6	B6_N0	2.5 V...ault)		8mA ...ult)			
sw[4]	Input	PIN_91	6	B6_N0	2.5 V...ault)		8mA ...ult)			
sw[3]	Input	PIN_49	3	B3_N0	2.5 V...ault)		8mA ...ult)			
sw[2]	Input	PIN_46	3	B3_N0	2.5 V...ault)		8mA ...ult)			
sw[1]	Input	PIN_25	2	B2_N0	2.5 V...ault)		8mA ...ult)			
sw[0]	Input	PIN_24	2	B2_N0	2.5 V...ault)		8mA ...ult)			
<<new node>>										

Рис. 45 – Подключение SW[7:0] (Pin Planner)

Создание программного файла Lab1_task2_source.c

Создадим текст программы в файле с исходным кодом Lab1_task2_source.c, где будет обеспечиваться сложение по модулю, задаваемому переключателями SW[7:0]. Синтаксис файла представлен на Рис. 47 ниже:

lab12 - lab_NIOSII_1 - LAb2_task2_source.c

```c
1  #include <unistd.h>
2
3  int main(void)
4  {
5      char *psw = (char*) 0x8000;
6      char *pled = (char*) 0x8100;
7      char count = 64;
8
9      while(1)
10     {
11         usleep(300000);
12
13         if(((*psw) != 0x00) && (((*psw)-1) > count))
14             count++; /* Continue 0-SW[7:0] counting loop. */
15         else
16             count = 0; /* start counting loop from 0 */
17
18         *pled = ~count;
19     }
20
21     return 0;
22 }
23
```

Рис. 46 – Программный файл с реализацией сложения по модулю SW[7:0]

Реализация на плате

Проект был загружен на плату. Увеличение счётчика по модулю работает корректно.

Так, например, при выставлении значения на SW[7..0] = 12 осуществляется увеличение счётчика от 0 до 11. Это связано с тем, что $psw - 1 = 11 < count = 64 \rightarrow$ выполняется условие на 16 строке (значение счётчика по умолчанию меняется с 64 на 0).

Если же изначально установить значение на SW[7:0] = 70, то счёт будет производиться от 64 до 69.

Работа с устройством на плате была продемонстрирована преподавателю.

4. Вывод

В ходе лабораторной работы был осуществлен процесс создания и настройки системы (Рис. 1) на базе процессора NIOS II с использованием пакета Quartus Prime и Eclipse IDE. Реализация проекта включала в себя создание аппаратной части с помощью Platform Designer (PD), создание программной части с помощью среды разработки Eclipse, где был создан файл для инициализации модуля памяти программ процессора, а также реализация проекта на плате.

В ходе тестирования проекта на плате светодиоды LED8-LED1 отображали последовательное увеличение счётчика от 0 до 255 при sw[0] в положении 1, а при переключении sw[0] в положение 0 счёт происходил в обратном порядке с тем же шагом от 255 до 0. Это говорит о корректности работы созданного устройства.

В дополнительных заданиях были реализованы различные функциональности, такие как изменение частоты переключения светодиодов, использование указателей для адресации данных, а также увеличение разрядности ввода-вывода для взаимодействия с переключателями.

Анализ результатов показал успешную работу созданной системы на плате, а также корректное отображение данных на светодиодах в зависимости от значений на переключателях.

Полученные навыки работы с NIOS II могут быть полезны при создании проектов в области встраиваемых систем, таких как системы управления, обработки сигналов, автоматизации и других.