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

Отчёт курсовой работе. Часть 2

Дисциплина: Автоматизация проектирования дискретных устройств (на английском языке).

Выполнил студент гр. 5130901/10101 _____ Д.Л. Симоновский
(подпись)

Руководитель _____ А.А. Федотов
(подпись)

“13” мая 2024 г.

Санкт-Петербург
2024

Оглавление

1. Список иллюстраций: 2

2. Проект 1: 4

2.1. Структура проекта: 4

2.2. Решение: 4

3. Проект 2: 16

3.1. Структура проекта: 16

3.2. Решение: 16

4. Проект 3: 20

4.1. Структура проекта: 20

4.2. Решение: 20

5. Проект 3: 25

5.1. Структура проекта: 25

5.2. Решение: 26

6. Вывод: 33

1. Список иллюстраций:

Рис. 2.1. Структура разрабатываемого проекта. 4
Рис. 2.2. Настройки для QP. 4
Рис. 2.3. Настройки модуля счетчика. 5
Рис. 2.4. Настройка модуля памяти. 5
Рис. 2.5. Модуль ввода для светодиодов. 6
Рис. 2.6. Модуль ввода для кнопки. 6
Рис. 2.7. Модуль JTAG UART. 7
Рис. 2.8. Настройки вектора для Nios II. 7
Рис. 2.9. Platform Designer. 7
Рис. 2.10. Schematic в Platform Designer. 8
Рис. 2.11. System Interconnections. 8
Рис. 2.12. RTL Viewer. 9
Рис. 2.13. Pin Planner. 9
Рис. 2.14. Результат компиляции. 10
Рис. 2.15. Создание проекта для Nios II. 11
Рис. 2.16. Создание файла с исходным кодом. 11
Рис. 2.17. Настройки BSP. 12
Рис. 2.18. Компиляция программы. 12
Рис. 2.19. Приветственные сообщения после запуска. 12
Рис. 2.20. Консоль после нескольких нажатий на pba. 12
Рис. 2.21. Новые настройки для BSP. 13
Рис. 2.22. Компиляция проекта с новыми настройками BSP. 13
Рис. 2.23. Результат компиляции. 14
Рис. 2.24. Optimization level: size. У lab3_sw_bsp. 14
Рис. 2.25. Optimization level: size. У lab3_sw. 15
Рис. 2.26. Компиляция после выставления настроек оптимизации. 15
Рис. 2.27. Результат запуска программы и нескольких переключений pba. 15
Рис. 3.1. Структура разрабатываемого проекта. 16
Рис. 3.2. Настройки для QP. 16
Рис. 3.3. Новые настройки модуля PIO. 17
Рис. 3.4. Подключение прерываний к процессору. 17
Рис. 3.5. Назначения в Pin Planner. 17
Рис. 3.6. Результат компиляции. 18
Рис. 3.7. Результат компиляции. 19
Рис. 3.8. Консоль проекта после записи программы. 19
Рис. 3.9. Консоль после нескольких запусков программы. 20
Рис. 4.1. Структура разрабатываемого проекта. 20
Рис. 4.2. Настройки для QP. 21
Рис. 4.3. Настройки компонента timer. 21
Рис. 4.4. Подключение таймера к процессору. 21
Рис. 4.5. Назначения в Pin Planner. 22
Рис. 4.6. Результат компиляции. 23
Рис. 4.7. Результат компиляции. 24
Рис. 4.8. Результат запуска программы на плате. 24
Рис. 4.9. Результат запуска программы на плате. 24
Рис. 4.10. Результат запуска программы на плате. 25
Рис. 5.1. Структура разрабатываемого проекта. 26
Рис. 5.2. Добавление модуля Custom Instruction. 27
Рис. 5.3. Блок модуля Custom Instruction. 27
Рис. 5.4. Настройки модуля ввода 1. 27
Рис. 5.5. Настройки модуля ввода 2. 28

Рис. 5.6. Подключения в PD. 28
Рис. 5.7. Схема разработанного устройства. 29
Рис. 5.8. Назначения в Pin Planner. 29
Рис. 5.9. Результат компиляции. 30
Рис. 5.10. Результат компиляции. 31
Рис. 5.11. Результат запуска программы на процессоре. 31
Рис. 5.12. Результат компиляции. 32
Рис. 5.13. Результат запуска программы на процессоре. 32
Рис. 5.14. Результат запуска на процессоре. 32
Рис. 5.15. Результат запуска на процессоре. 33

2. Проект 1:

2.1. Структура проекта:

Структура разрабатываемого проекта приведена ниже:

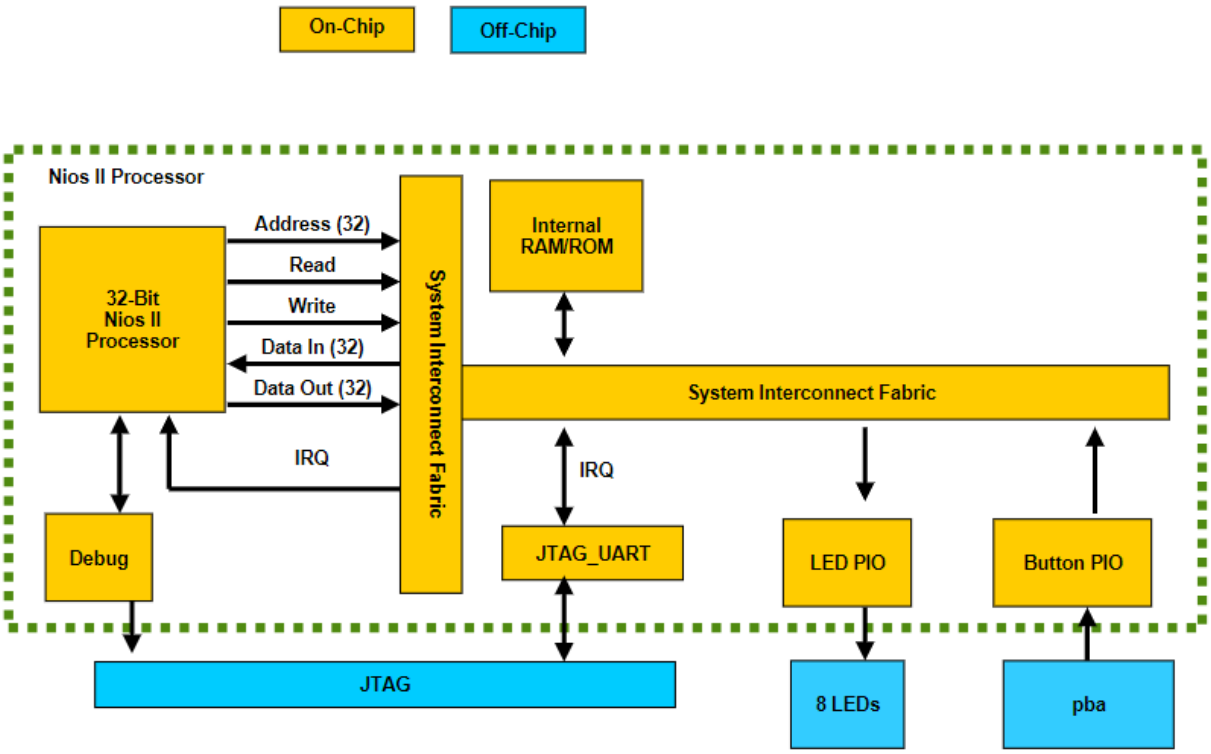

Рис. 2.1. Структура разрабатываемого проекта.

Под управлением процессора NIOSII обеспечивается:

- Опрос состояния кнопки pba.
- Формирование на консоли сообщений о нажатой кнопке.
- При каждом нажатии кнопки pba происходит изменение номера включенного светодиода от led1 к led8 на одну позицию (с циклическим переходом от led8 к led1).

2.2. Решение:

Выполним создание проекта в QP со следующими настройками:

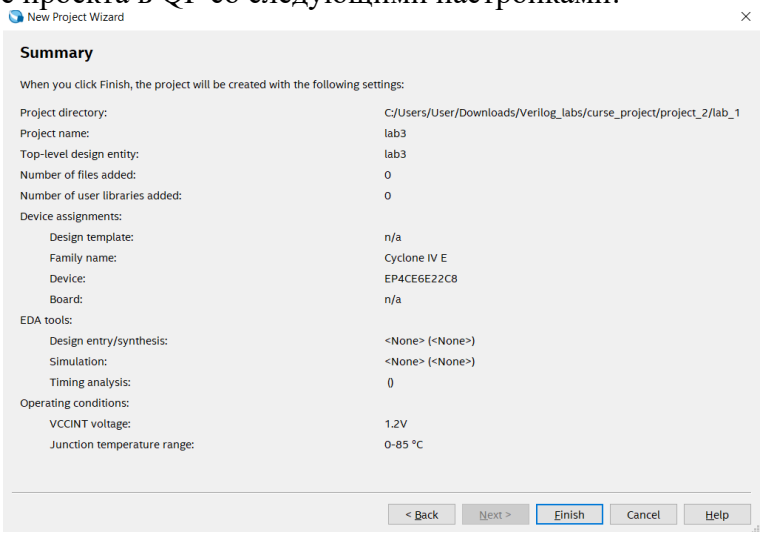

Рис. 2.2. Настройки для QP.

Далее переходим в Platform Designer и начинаем реализовывать схему с *Рис. 2.1*.

Начнем с модуля clk:

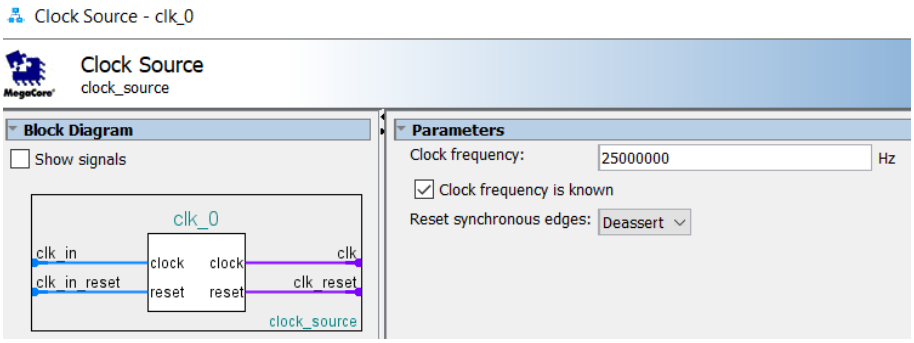

Рис. 2.3. Настройки модуля счетчика.

Добавим модуль памяти для процессора Nios II со следующими настройками:

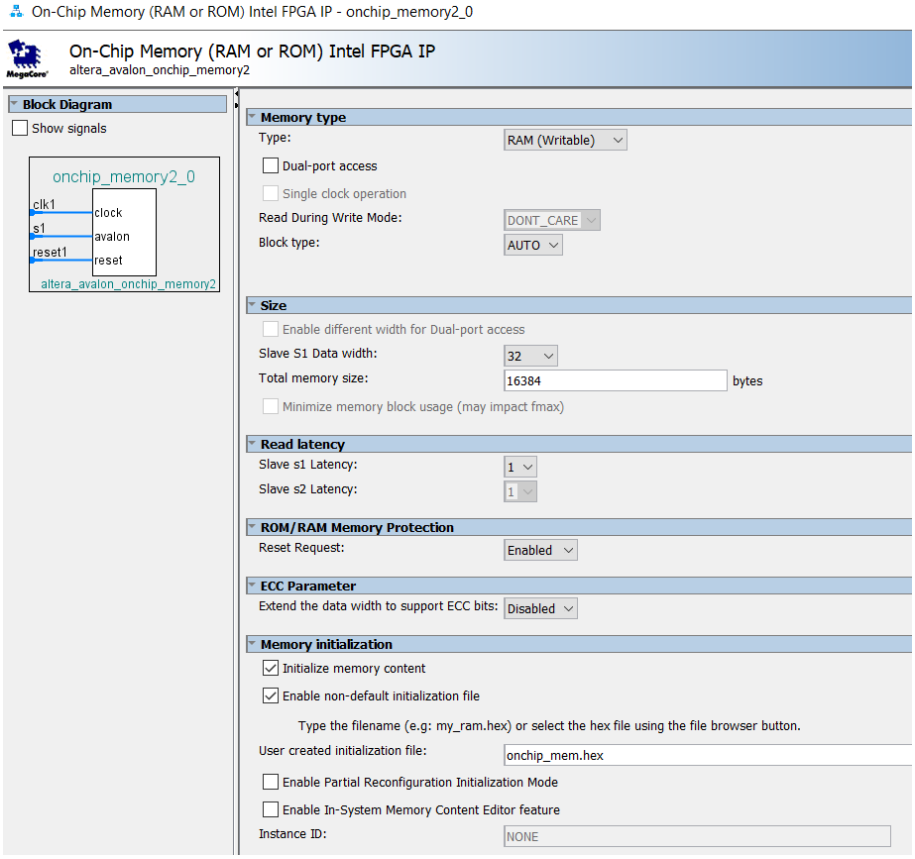

Рис. 2.4. Настройка модуля памяти.

Также необходим модуль вывода для светодиодов, добавим его со следующими настройками:

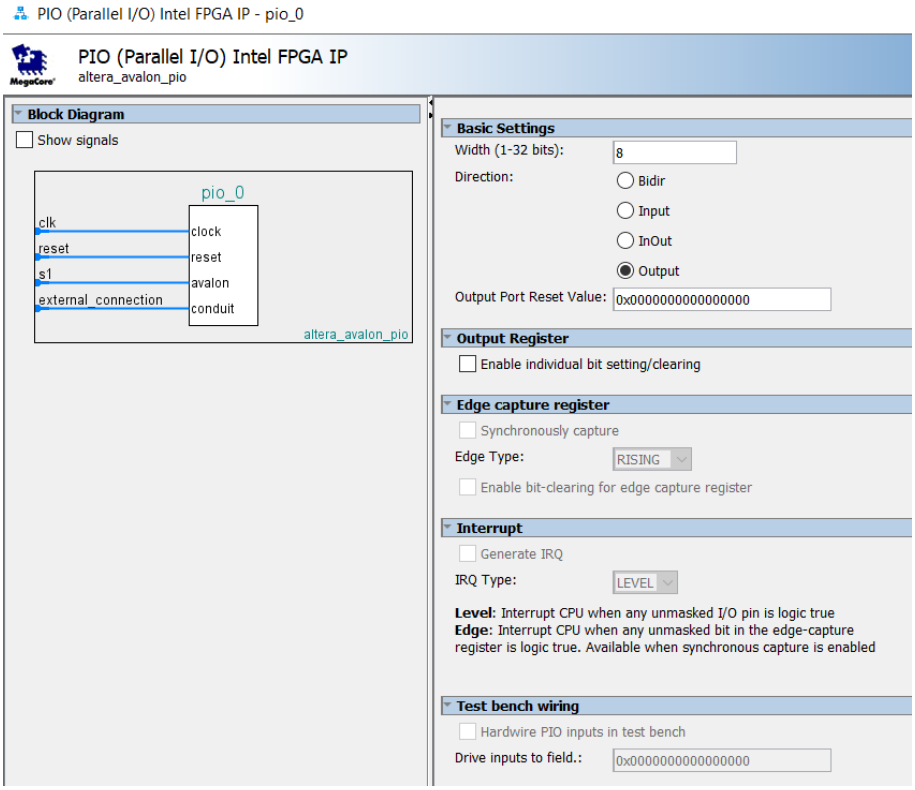

Рис. 2.5. Модуль ввода для светодиодов.

И модуль ввода для кнопки:

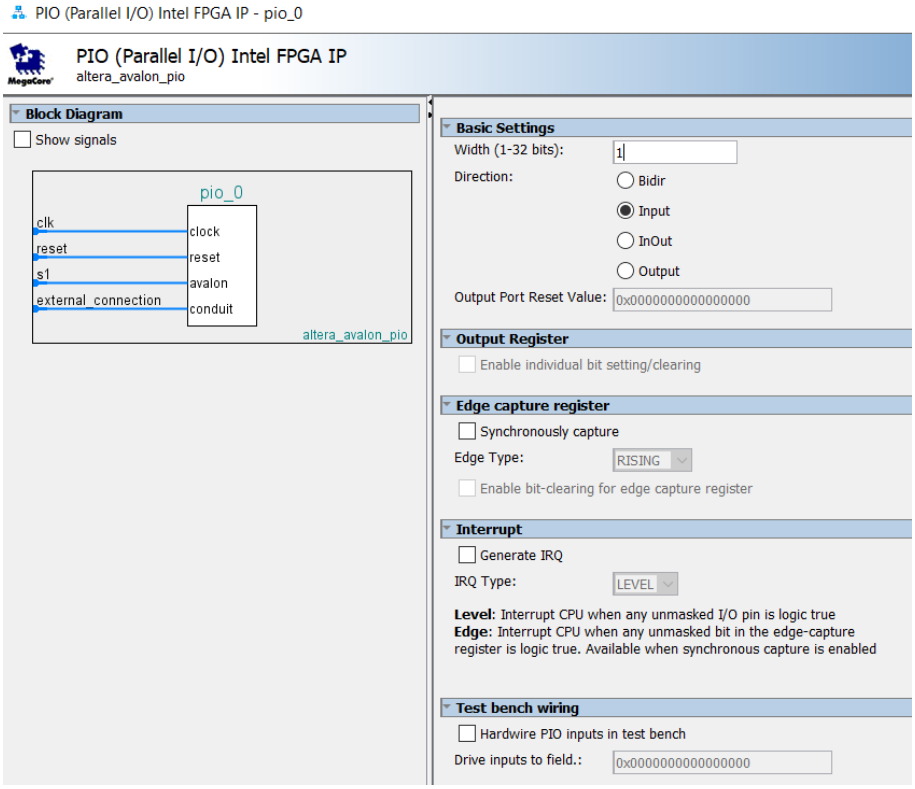

Рис. 2.6. Модуль ввода для кнопки.

Для того, чтоб процессор мог отправлять нам какие-то данные, добавим модуль UART:

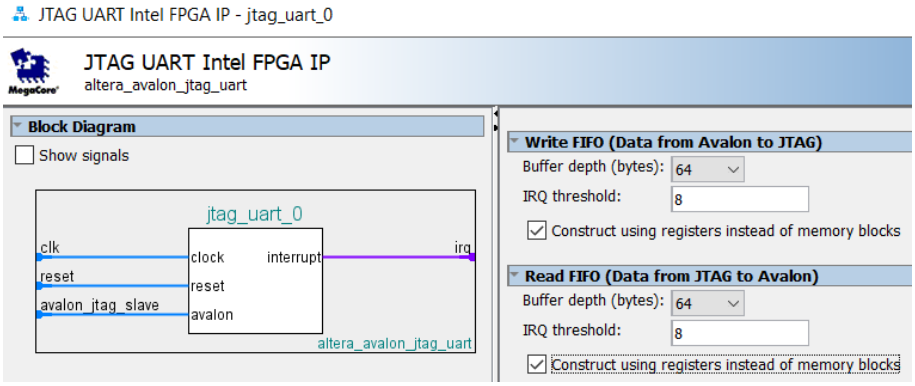

Рис. 2.7. Модуль JTAG UART.

Также необходимо добавить сам процессор с обычными настройками и следующими настройками вектора ошибок и сброса:

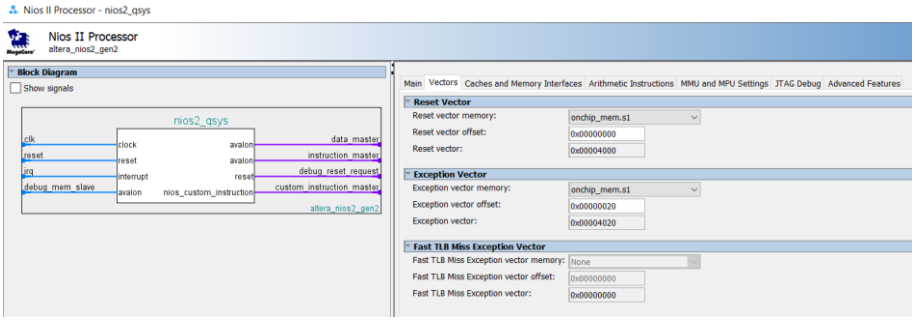

Рис. 2.8. Настройки вектора для Nios II.

Выполним соединения в Platform Designer созданных модулей следующим образом:

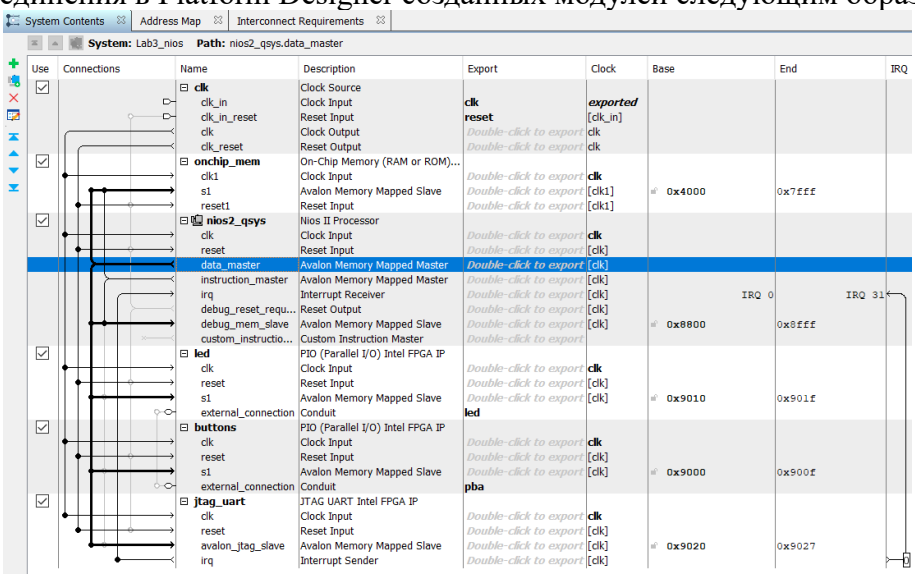

Рис. 2.9. Platform Designer.

Посмотрим, как выглядит Schematic:

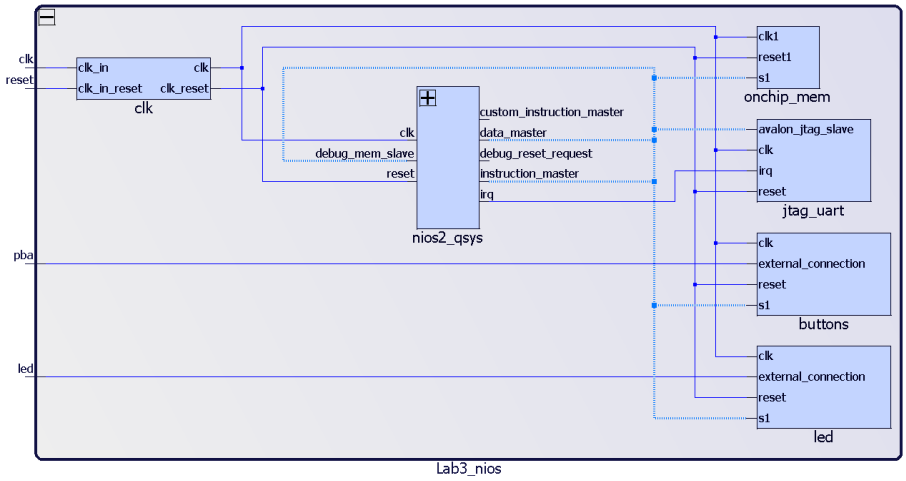

Рис. 2.10. Schematic в Platform Designer.

Как мы видим разработанная схема похожа на *Рис. 2.1.*, что свидетельствует о корректности разработанного устройства.

Посмотрим на Interconnections, которые были добавлены в разработанной схеме:

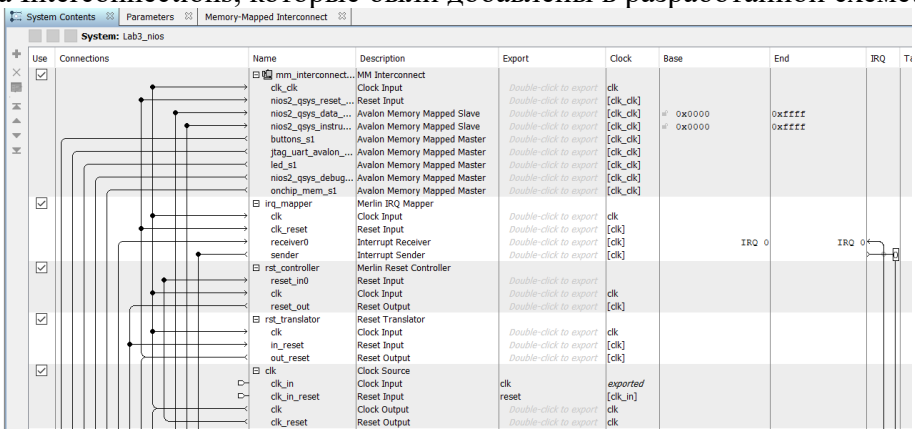

Рис. 2.11. System Interconnections.

Как мы видим, появилось 4 новых модуля.

rst_controller служит для сброса процессора и памяти, а также передает сигнал сброса в rst_translator, а он в свою очередь занимается сбросом периферии.

irq_mapper занимается, как понятно из названия, прерываниями.

mm_interconnect_0 отвечает за передачу данных между периферией и процессором.

Выполним генерацию разработанного устройства и создадим файл верхнего уровня:

Verilog_labs - lab3.sv

```
module lab3 (
  input  bit       clk,
  input  bit       pbb,
  input  bit       pba,
  output bit [7:0] led
);

  Lab3_nios u0 (
    .clk_clk      (clk),  //   clk.clk
    .reset_reset_n(pbb),  //   reset.reset_n
    .led_export   (led),  //   led.export
    .pba_export   (pba)   //   pba.export
  );

endmodule
```

Выполним компиляцию и посмотрим на результат в RTL Viewer:

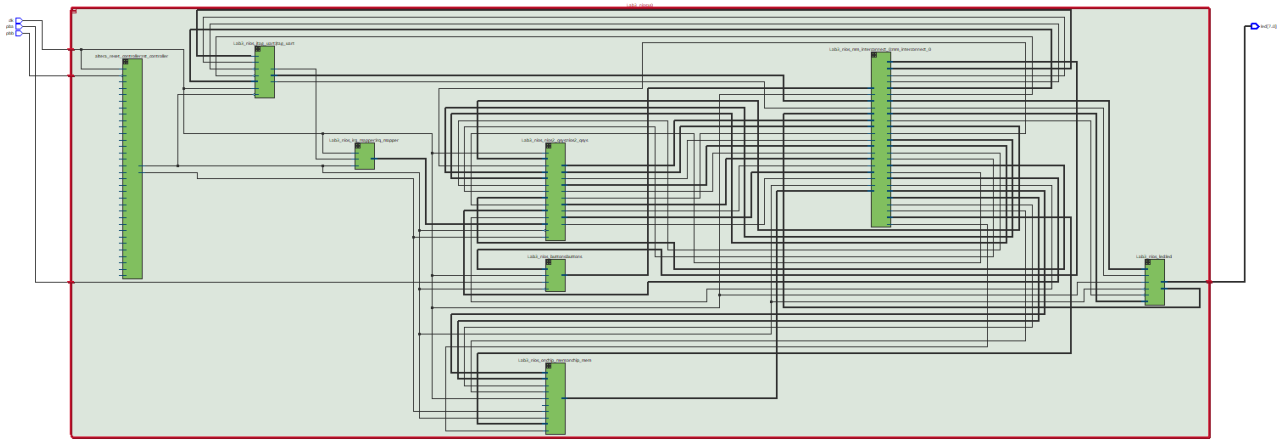

Рис. 2.12. RTL Viewer.

Данная схема аналогична приведенной в методических материалах, что свидетельствует о корректности разработанного устройства.

Далее выполним назначения входов и выходов в Pin Planner:

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard
altera_reserved_tck	Input				2.5 V ...fault)
altera_reserved_tdi	Input				2.5 V ...fault)
altera_reserved_tdo	Output				2.5 V ...fault)
altera_reserved_tms	Input				2.5 V ...fault)
clk	Input	PIN_23	1	B1_N0	2.5 V ...fault)
led[7]	Output	PIN_65	4	B4_N0	2.5 V ...fault)
led[6]	Output	PIN_66	4	B4_N0	2.5 V ...fault)
led[5]	Output	PIN_67	4	B4_N0	2.5 V ...fault)
led[4]	Output	PIN_68	4	B4_N0	2.5 V ...fault)
led[3]	Output	PIN_69	4	B4_N0	2.5 V ...fault)
led[2]	Output	PIN_70	4	B4_N0	2.5 V ...fault)
led[1]	Output	PIN_71	4	B4_N0	2.5 V ...fault)
led[0]	Output	PIN_72	4	B4_N0	2.5 V ...fault)
pba	Input	PIN_64	4	B4_N0	2.5 V ...fault)
pbb	Input	PIN_58	4	B4_N0	2.5 V ...fault)
<<new node>>					

Рис. 2.13. Pin Planner.

Также добавим .sdc файл с временными требованиями:

Verilog_labs - SDC1.sdc

```
#*******************************
# 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
#*******************************

derive_clock_uncertainty

#*******************************
# 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 {pba}]


set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {altera_reserved_tdi}]
set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {altera_reserved_tms}]


#*******************************
# Set Outnput 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]}]

set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {altera_reserved_tdo}]
```

Проведем полную компиляцию и проверим, что все они выполняются:

Table of Contents

- Flow Summary
- Flow Settings
- Flow Non-Default Glob
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
- Fitter
- Flow Messages
- Flow Suppressed Mess
- Assembler
- Timing Analyzer

Flow Summary

<<Filter>>

Flow Status	Successful - Fri May 03 09:56:55 2024
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	lab3
Top-level Entity Name	lab3
Family	Cyclone IV E
Device	EP4CE6E22C8
Timing Models	Final
Total logic elements	2,933 / 6,272 (47 %)
Total registers	1917
Total pins	11 / 92 (12 %)
Total virtual pins	0
Total memory bits	141,312 / 276,480 (51 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)

Рис. 2.14. Результат компиляции.

Как можно видеть, компиляция прошла успешно и временные требования выполняются. Перейдем к разработке программа для Nios II:

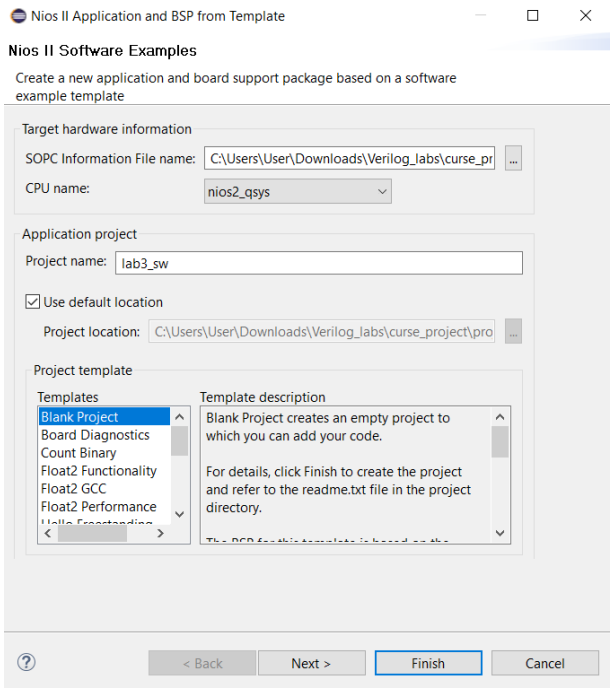

Рис. 2.15. Создание проекта для Nios II.

Создадим файл с исходным кодом:

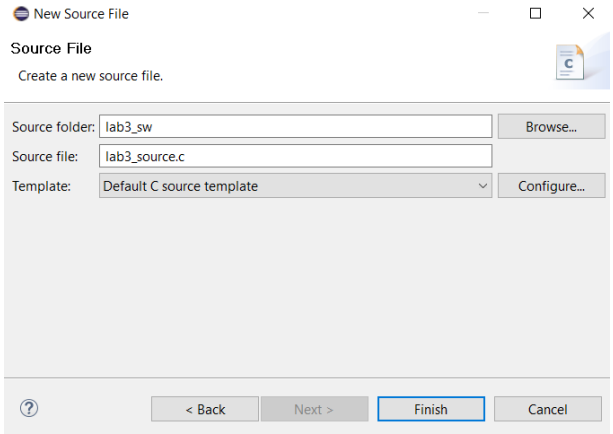

Рис. 2.16. Создание файла с исходным кодом.

Код приведен ниже:

Verilog_labs - lab3_source.c

```c
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include <unistd.h>
#include <stdio.h>
#define NONE_PRESSED 0x1
#define DEBOUNCE 50000

int main (void) {
    int buttons;
    int led = 0x00;

    printf("Привет\nПроцессор Nios II запущен!\n");
    printf("Нажмите кнопку pba на плате miniDiLab-CIV\n\n");

    IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, led);

    while (1) {
        buttons = IORD_ALTERA_AVALON_PIO_DATA(BUTTONS_BASE);

        if (buttons != NONE_PRESSED) {
            printf("Нажата кнопка pba\n");

            if (led >= 0x80 || led == 0x00) led = 0x01;
            else led = led << 1;

            IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, ~led);

            usleep(DEBOUNCE);
            while(buttons != NONE_PRESSED)
                buttons = IORD_ALTERA_AVALON_PIO_DATA(BUTTONS_BASE);
            usleep(DEBOUNCE);
        }
    }
}
```

При старте в консоль выводятся сообщения, а все светодиоды зажигаются (0 – активный).

Далее в цикле считывается значение с кнопки до тех пор, пока она не будет нажата. В случае нажатия происходит циклический сдвиг 1 в переменной, и она выводится на светодиоды в инвертированном виде, таким образом будет гореть только тот светодиод, которому соответствовала единица в переменной led.
После этого включается задержка, чтоб избежать дребезга.
Выполним следующие настройки для BSP, чтоб уменьшить объем проекта:

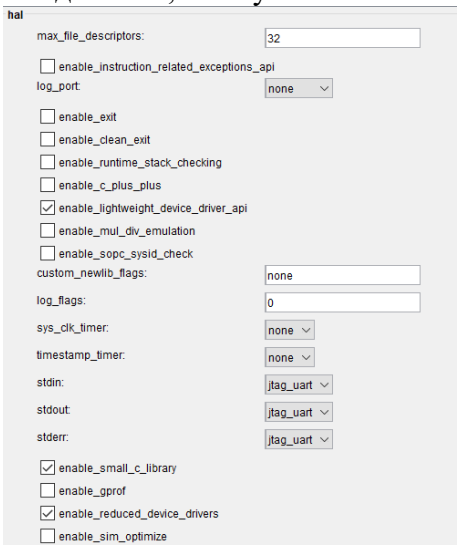

Рис. 2.17. Настройки BSP.

И выполним компиляцию:

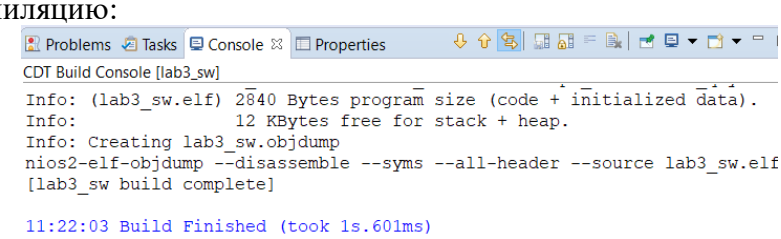

Рис. 2.18. Компиляция программы.

Компиляция прошла успешно.
Теперь запишем проект на плату и посмотрим на результат в консоли:

Рис. 2.19. Приветственные сообщения после запуска.

Как мы видим, после запуска действительно появились приветственные сообщения. Выполним несколько нажатий на pba:

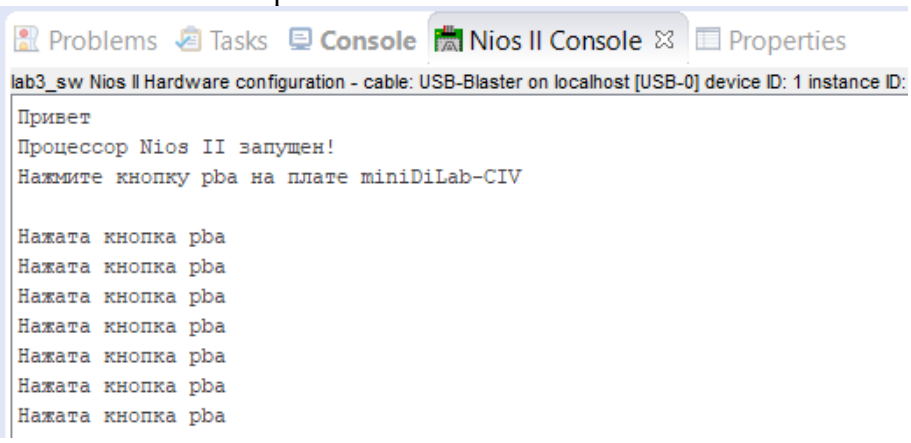

Рис. 2.20. Консоль после нескольких нажатий на pba.

После каждого нажатия появляется надпись, а также светодиод сдвигается левее. По достижению led8 он переходит к led1, как и было задумано.
Теперь выполним анализ размера файла после компиляции при различных настройках.
Поставим следующие настройки для BSP:

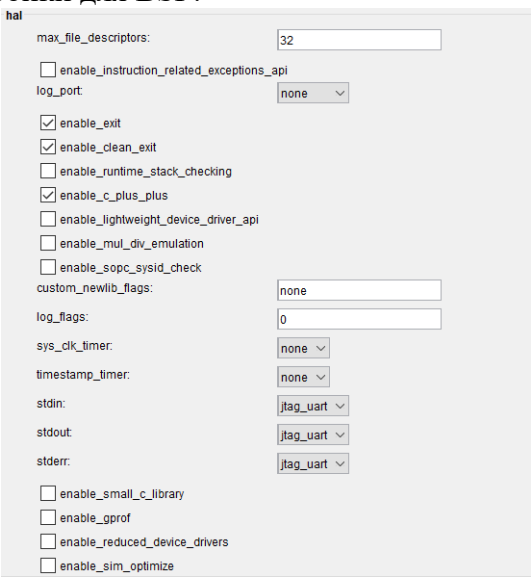

Рис. 2.21. Новые настройки для BSP.

Выполним генерацию этого BSP и компиляцию проекта:

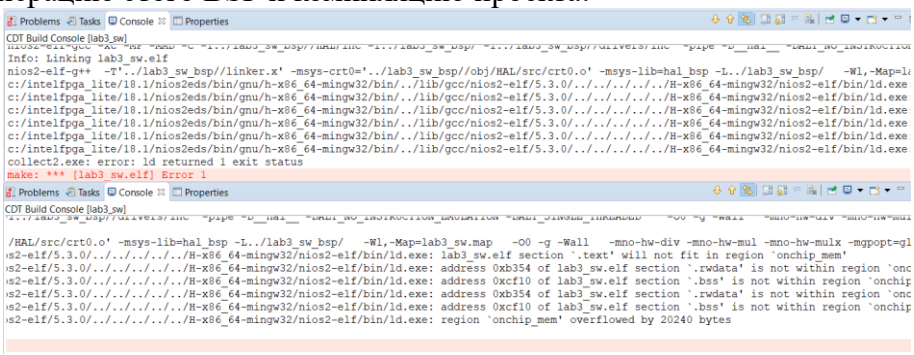

Рис. 2.22. Компиляция проекта с новыми настройками BSP.

Видим, что для компиляции не хватило памяти устройства, а именно 20240 байт.
Вернем настройки в прошлую форму, чтоб файл мог поместиться в выделенной памяти.
Проексперементируем теперь с кодом, заменим printf на более легкие alt_printf. Они более ограничены в функционали, однако для задачи вывода текста на экран вполне подходят:

```c
Verilog_labs - lab3_source.c
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include <unistd.h>
#include <stdio.h>
#include "sys\alt_stdio.h"
#define NONE_PRESSED 0x1
#define DEBOUNCE 50000

int main (void) {
    int buttons;
    int led = 0x00;

    alt_printf("Привет\nПроцессор Nios II запущен!\n");
    alt_printf("Нажмите кнопку pba на плате miniDiLab-CIV\n\n");

    IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, led);

    while (1) {
        buttons = IORD_ALTERA_AVALON_PIO_DATA(BUTTONS_BASE);

        if (buttons != NONE_PRESSED) {
            alt_printf("Нажата кнопка pba\n");

            if (led >= 0x80 || led == 0x00) led = 0x01;
            else led = led << 1;

            IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, ~led);

            usleep(DEBOUNCE);
            while(buttons != NONE_PRESSED)
                buttons = IORD_ALTERA_AVALON_PIO_DATA(BUTTONS_BASE);
            usleep(DEBOUNCE);
        }
    }
}
```

Выполним компиляцию и получим следующий результат:

```
Problems  Tasks  Console  Properties
CDT Build Console [lab3_sw]
nios2-elf-g++  -T'../lab3_sw_bsp//linker.x' -msys-crt0='../lab3_sw_bsp//obj/HAL/src/crt0.o' -msys-
nios2-elf-insert lab3_sw.elf --thread_model hal --cpu_name nios2_qsys --qsys true --simulation_ena
Info: (lab3_sw.elf) 2580 Bytes program size (code + initialized data).
Info:              13 KBytes free for stack + heap.
Info: Creating lab3_sw.objdump
nios2-elf-objdump --disassemble --syms --all-header --source lab3_sw.elf >lab3_sw.objdump
[lab3_sw build complete]

13:22:33 Build Finished (took 1s.676ms)
```

Рис. 2.23. Результат компиляции.

Как мы видим, размер файла стал меньше, как и ожидалось.

Попробуем еще уменьшить размер файла, выставим настройки оптимизации по размеру файла:

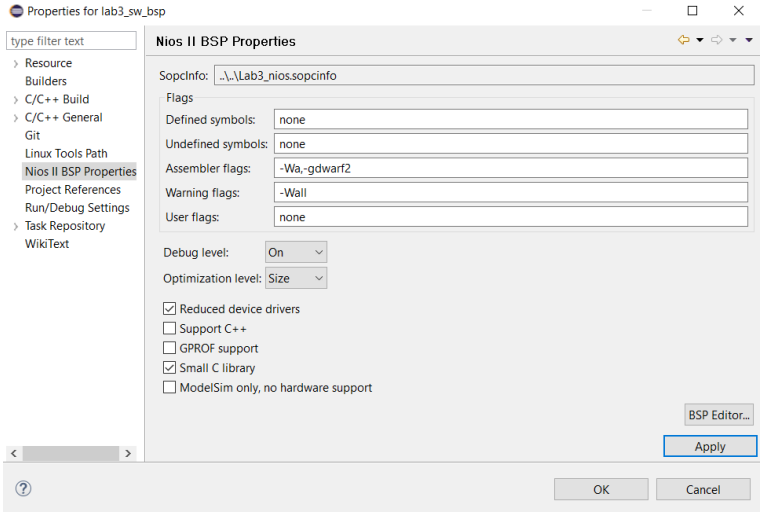

Рис. 2.24. Optimization level: size. У lab3_sw_bsp.

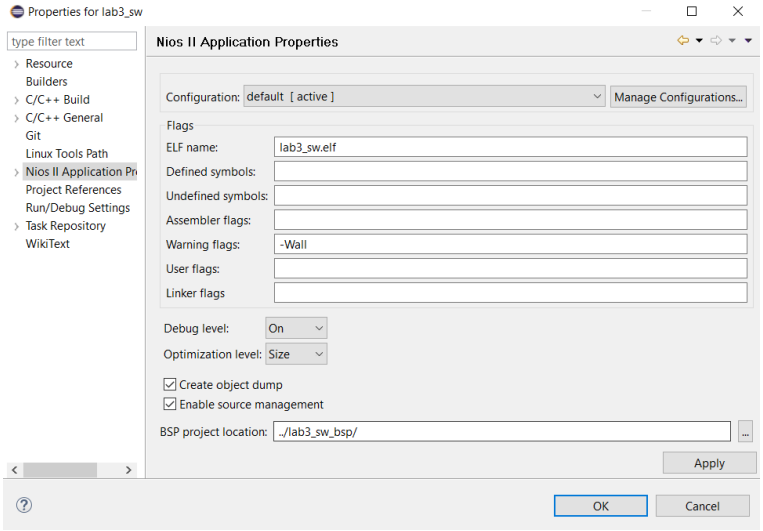

Рис. 2.25. Optimization level: size. У lab3_sw.

Повторим компиляцию:

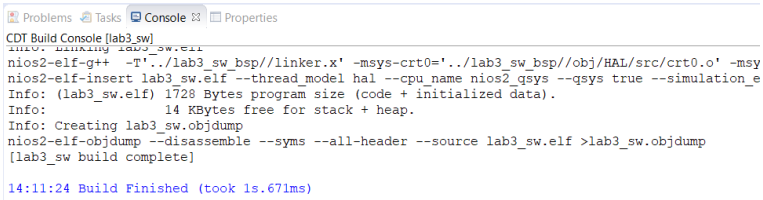

Рис. 2.26. Компиляция после выставления настроек оптимизации.

Видим, что размер файла уменьшился еще сильнее, до 1728 байт.

Повторно запрограммируем плату и убедимся, что результат выполнения не изменился:

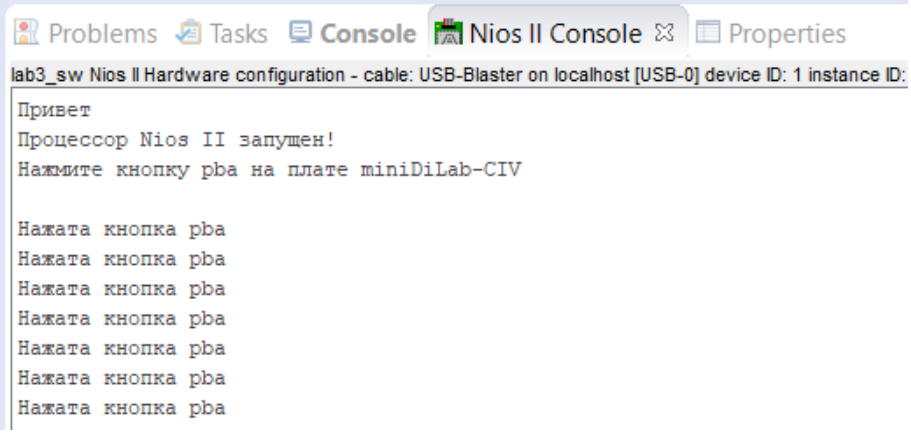

Рис. 2.27. Результат запуска программы и нескольких переключений pba.

Как мы видим, программа работает так же, как и до оптимизаций.

3. Проект 2:

3.1. Структура проекта:

Структура разрабатываемого проекта приведена ниже:

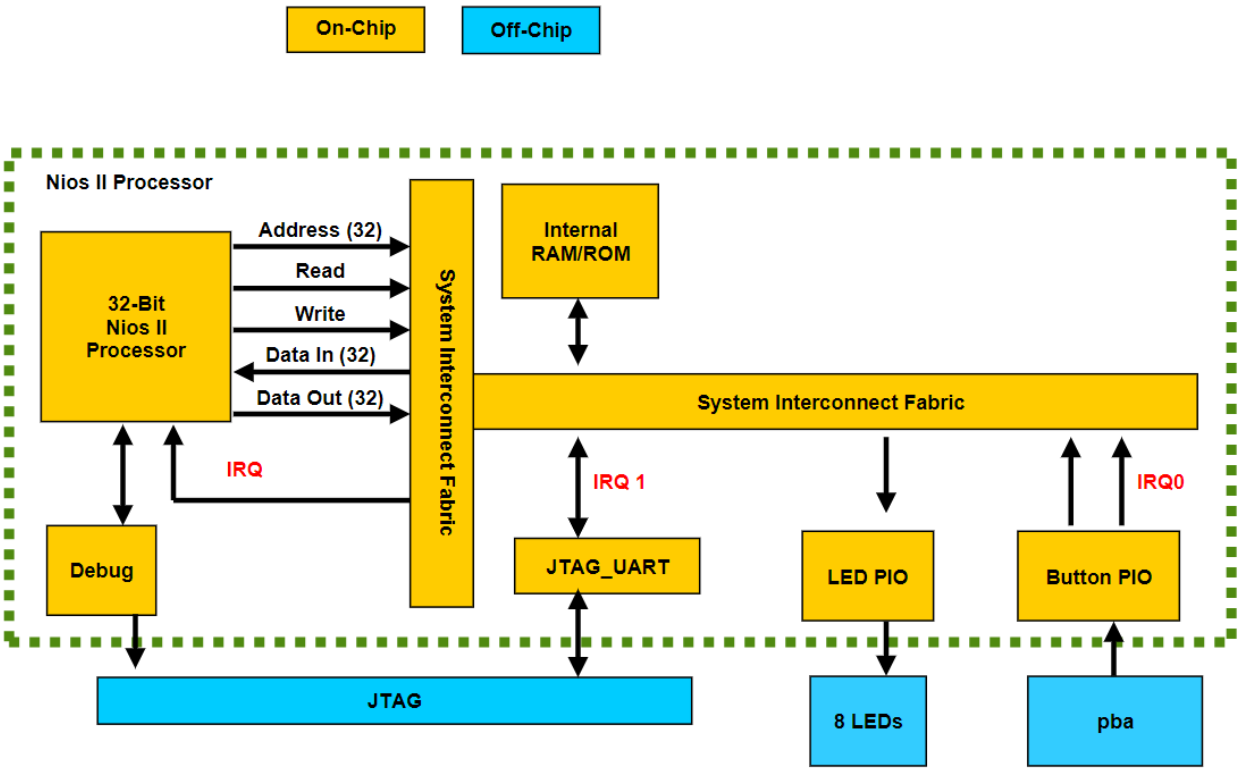

Рис. 3.1. Структура разрабатываемого проекта.

Под управлением процессора NIOSII обеспечивается:

- Работа по прерываниям от нажатия кнопки pba.
- Формирование на консоли сообщений о нажатой кнопке.
- При каждом нажатии кнопки pba происходит изменение номера включенного светодиода от led1 к led8 на одну позицию (с циклическим переходом от led8 к led1).

3.2. Решение:

Выполним создание проекта в QP со следующими настройками:

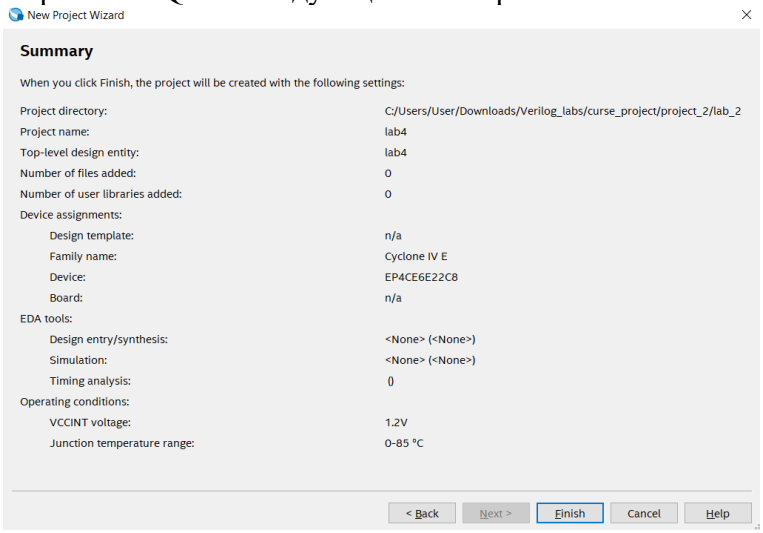

Рис. 3.2. Настройки для QP.

Далее переходим в Platform Designer. Скопируем PD из *Проект 1:*.

Для того, чтоб мы могли работать с кнопками по прерываниям необходимо выполнить следующие настройки с модулем PIO:

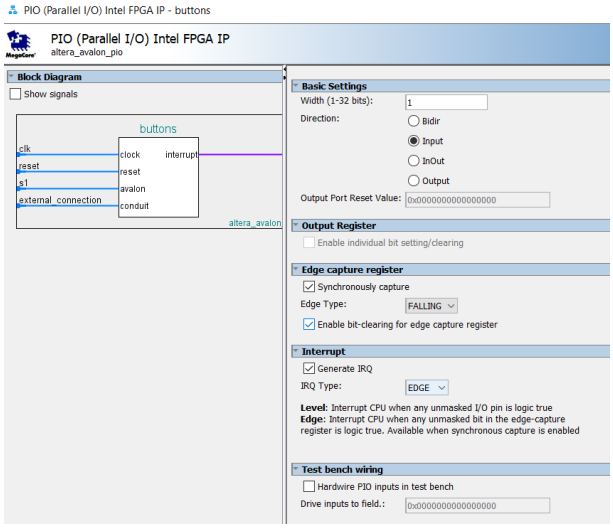

Рис. 3.3. Новые настройки модуля PIO.

Также необходимо подключить прерывания к процессору:

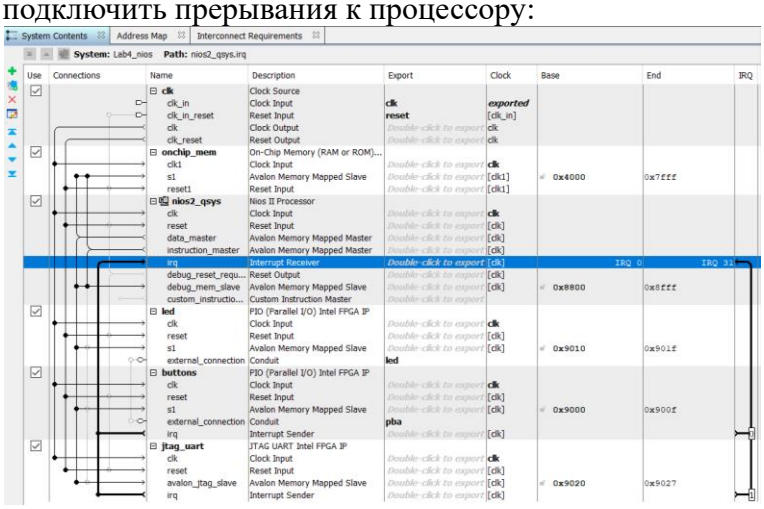

Рис. 3.4. Подключение прерываний к процессору.

Выполним генерацию модуля и создадим файл верхнего уровня:

Verilog_labs - lab4.sv

```
module lab4 (
  input  bit       clk,
  input  bit       pbb,
  input  bit       pba,
  output bit [7:0] led
);

  Lab4_nios u0 (
    .clk_clk       (clk),  //   clk.clk
    .reset_reset_n(pbb),  //   reset.reset_n
    .led_export    (led),  //   led.export
    .pba_export    (pba)   //   pba.export
  );

endmodule
```

Выполним компиляцию и выполним назначения для входов и выходов:

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	urrent Streng	Slew Rate	ifferential Pai	ict Preservati
altera_...ved_tck	Input				2.5 V ...fault)		8mA (default)			
altera_r...rved_tdi	Input				2.5 V ...fault)		8mA (default)			
altera_...ved_tdo	Output				2.5 V ...fault)		8mA (default)	2 (default)		
altera_...ved_tms	Input				2.5 V ...fault)		8mA (default)			
clk	Input	PIN_23	1	B1_N0	3.3-V LVTTL		8mA (default)			
led[7]	Output	PIN_65	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
led[6]	Output	PIN_66	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
led[5]	Output	PIN_67	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
led[4]	Output	PIN_68	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
led[3]	Output	PIN_69	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
led[2]	Output	PIN_70	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
led[1]	Output	PIN_71	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
led[0]	Output	PIN_72	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
pba	Input	PIN_64	4	B4_N0	2.5 V ...fault)		8mA (default)			
pbb	Input	PIN_58	4	B4_N0	2.5 V ...fault)		8mA (default)			
<<new node>>										

Рис. 3.5. Назначения в Pin Planner.

Также добавим файл для временных характеристик:

Verilog_labs - SDC1.sdc

```
#*******************************
# 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
#*******************************

derive_clock_uncertainty

#*******************************
# 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 {pba}]


set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {altera_reserved_tdi}]
set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {altera_reserved_tms}]


#*******************************
# Set Outnput 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]}]

set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {altera_reserved_tdo}]
```

Выполним полную компиляцию и проверим, выполняются ли временные характеристики:

lab4.sv | Compilation Report - lab4

Table of Contents
- Flow Summary
- Flow Settings
- Flow Non-Default Glob
- Flow Elapsed Time
- Flow OS Summary
- Flow Log
- Analysis & Synthesis
- Fitter
- Flow Messages
- Flow Suppressed Mess
- Assembler
- Timing Analyzer

Flow Summary

<<Filter>>

Flow Status	Successful - Fri May 03 14:35:34 2024
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	lab4
Top-level Entity Name	lab4
Family	Cyclone IV E
Device	EP4CE6E22C8
Timing Models	Final
Total logic elements	2,946 / 6,272 (47 %)
Total registers	1925
Total pins	11 / 92 (12 %)
Total virtual pins	0
Total memory bits	141,312 / 276,480 (51 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)

Рис. 3.6. Результат компиляции.

Как видим все временные характеристики выполняются.

Теперь перейдем к созданию проекта Nios II. Сам проект создается с настройками, рассмотренными в *Проект 1:*. Текст программы приведен ниже:

```c
Verilog_labs - lab4_source.c
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include <unistd.h>
#include "sys\alt_irq.h"
#include "sys\alt_stdio.h"
#define DEBOUNCE 100000

static void buttons_isr(void* context) {
    volatile int* button_val_ptr = (volatile int *) context;
    *button_val_ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE);
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE, 0x3);
    usleep(DEBOUNCE);
}

int main() {
    volatile int buttons;
    int led = 0x00;
    buttons = 0;

    IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, led);
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTONS_BASE, 0x1);
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTONS_BASE, 0x1);
    alt_ic_isr_register(BUTTONS_IRQ_INTERRUPT_CONTROLLER_ID, BUTTONS_IRQ, buttons_isr, (void*)&buttons, 0x0);

    alt_printf("Процессор Nios II запущен!\n");
    alt_printf("Нажмите кнопку pba\n\n");

    while(1) {
        if (buttons != 0) {
            alt_printf("Нажата кнопка pba\n");
            if (led == 0x80 || led == 0x00) led = 0x01;
            else led = led << 1;
            buttons = 0;
            IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, ~led);
        }
    }
    return 0;
}
```

Функция buttons_isr должна вызываться при нажатии кнопки. Она записывает 1 при нажатии в переданную переменную, после этого она очищает все позиции захвата и запускает паузу в этой функции.

Основная функция зажигает све светодиоды, включает прерывания для кнопки, сбрасывает позицию захвата и регестрирует функцию, которую вызывать при прерывании.

Дадее уходим в бесконечный цикл, когда срабатывает прерывание значение переменной buttons устанавливается в 1 и мы реагируем на это, сдвигая заженный светодиод.

Выполним компиляцию:

```
Problems  Tasks  Console  Properties
CDT Build Console [lab4_sw]
nios2-elf-gcc -xc -MP -MMD -c -I../lab4_sw_bsp//HAL/inc -I../lab4_sw_bsp/ -I../lab4_sw_bsp//c
Info: Linking lab4_sw.elf
nios2-elf-g++  -T'../lab4_sw_bsp//linker.x' -msys-crt0='../lab4_sw_bsp//obj/HAL/src/crt0.o' -
nios2-elf-insert lab4_sw.elf --thread_model hal --cpu_name nios2_qsys --qsys true --simulatic
Info: (lab4_sw.elf) 2560 Bytes program size (code + initialized data).
Info:               13 KBytes free for stack + heap.
Info: Creating lab4_sw.objdump
nios2-elf-objdump --disassemble --syms --all-header --source lab4_sw.elf >lab4_sw.objdump
[lab4_sw build complete]

16:01:42 Build Finished (took 1s.818ms)
```

Рис. 3.7. Результат компиляции.

Как видим размер проекта равен 2560 байт. Запишем его на плату, тогда в консоли увидим:

```
Problems  Tasks  Console  Nios II Console  Properties
lab4_sw Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance
Процессор Nios II запущен!
Нажмите кнопку pba
```

Рис. 3.8. Консоль проекта после записи программы.

Нажмем кнопку несколько раз, будет следующий результат:

```
Problems  Tasks  Console  Nios II Console  Properties
lab4_sw Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0
Процессор Nios II запущен!
Нажмите кнопку pba

Нажата кнопка pba
Нажата кнопка pba
Нажата кнопка pba
Нажата кнопка pba
Нажата кнопка pba
Нажата кнопка pba
```

Рис. 3.9. Консоль после нескольких запусков программы.

Как мы видим программа работает корректно в соответствии с ожиданиями.

4. Проект 3:

4.1. Структура проекта:

Структура разрабатываемого проекта приведена ниже:

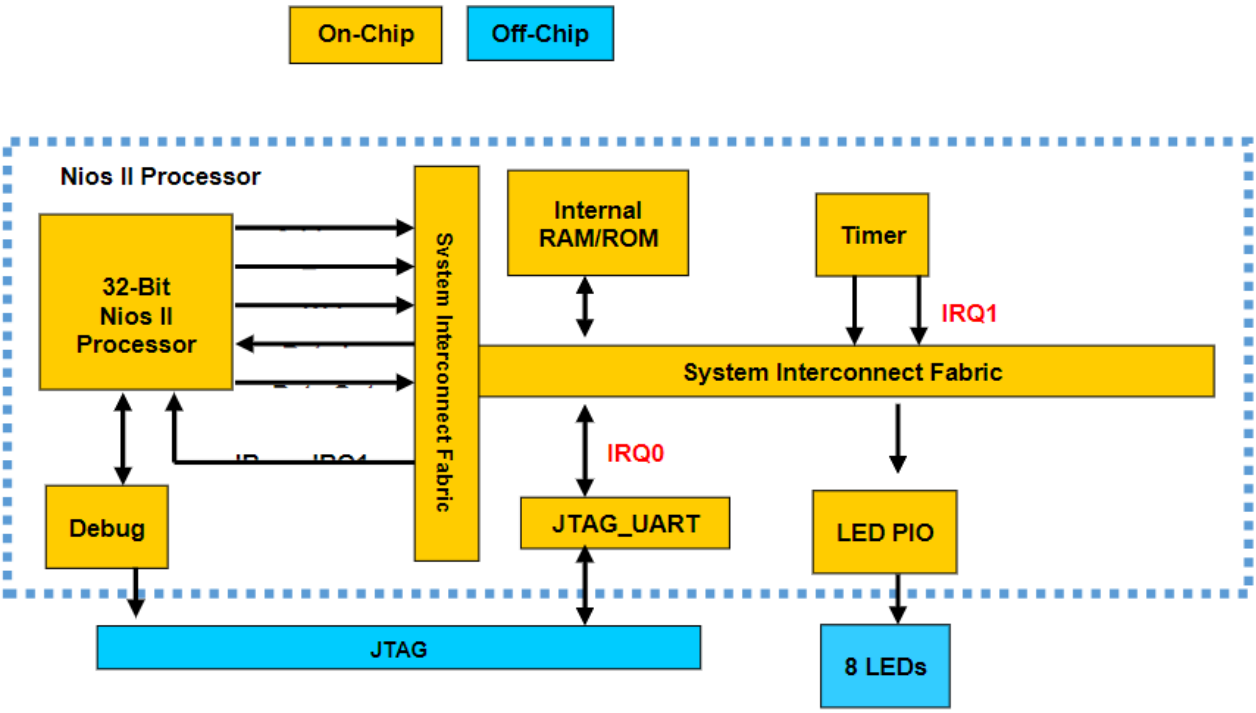

Рис. 4.1. Структура разрабатываемого проекта.

Под управлением процессора NIOSII обеспечивается:

- Циклический вывод на светодиоды платы miniDiLab значений 0x03; 0x0c; 0x30; 0xc0 с формированием на консоли соответствующего сообщения.
- Измерение числа ticks, необходимых для выполнения 4 итераций циклического выводы значений на светодиоды.
- Отображение на консоли:
  - числа ticks в секунду (частоты)
  - числа ticks, требуемых на выполнение 4 итераций циклического выводы значений на светодиоды.

4.2. Решение:

Выполним создание проекта в QP со следующими настройками:

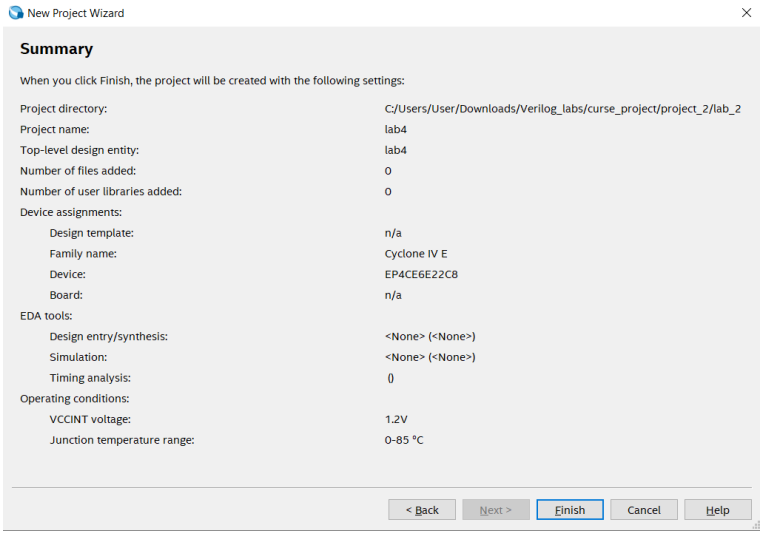

Рис. 4.2. Настройки для QP.

Далее переходим в Platform Designer. Скопируем PD из *Проект 2:*.

Удалим из системы компонент buttons, т.к. теперь лампочки будут переключаться по таймеру, а не переключателям.

Создадим сам компонент timer:

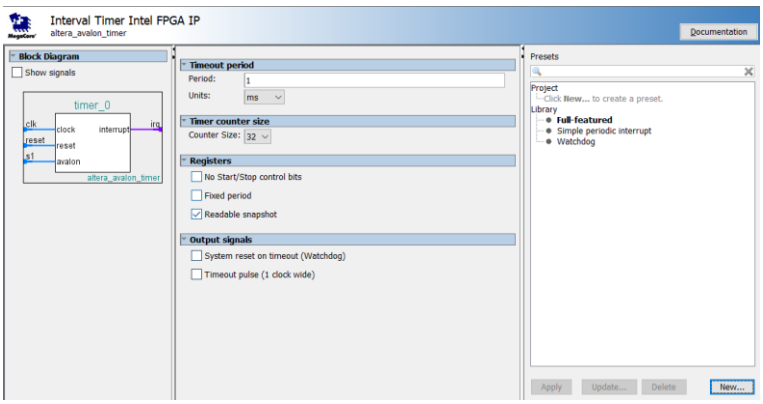

Рис. 4.3. Настройки компонента timer.

Также необходимо подключить таймер к процессору:

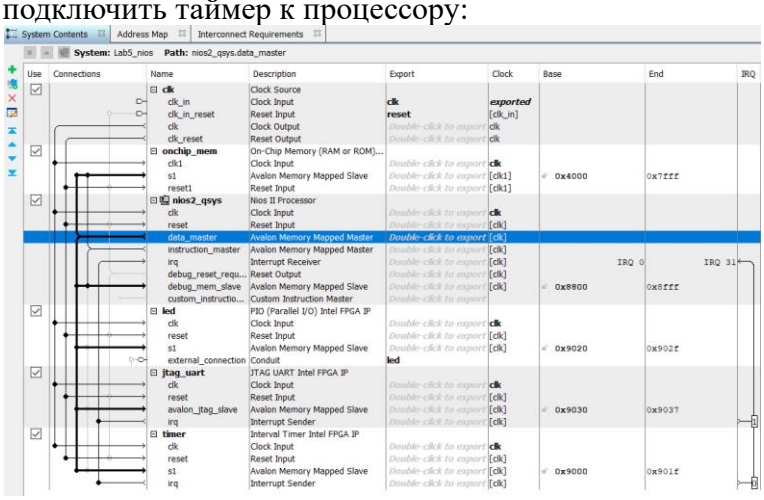

Рис. 4.4. Подключение таймера к процессору.

Выполним генерацию модуля и создадим файл верхнего уровня:

Verilog_labs - lab_5.sv

```
module lab_5 (
  input  bit       clk,
  input  bit       pbb,
  output bit [7:0] led
);

  Lab5_nios u0 (
    .clk_clk      (clk),  //   clk.clk
    .led_export   (led),  //   led.export
    .reset_reset_n(pbb)   // reset.reset_n
  );

endmodule
```

Выполним компиляцию и выполним назначения для входов и выходов:

Named: * Edit: 4

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	urrent Streng	Slew Rate	ifferential Pai	ict Preservati
altera_...ved_tck	Input				2.5 V ...fault)		8mA (default)			
altera_r...rved_tdi	Input				2.5 V ...fault)		8mA (default)			
altera_...ved_tdo	Output				2.5 V ...fault)		8mA (default)	2 (default)		
altera_...ved_tms	Input				2.5 V ...fault)		8mA (default)			
clk	Input	PIN_23	1	B1_N0	3.3-V LVTTL		8mA (default)			
led[7]	Output	PIN_65	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
led[6]	Output	PIN_66	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
led[5]	Output	PIN_67	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
led[4]	Output	PIN_68	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
led[3]	Output	PIN_69	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
led[2]	Output	PIN_70	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
led[1]	Output	PIN_71	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
led[0]	Output	PIN_72	4	B4_N0	2.5 V ...fault)		8mA (default)	2 (default)		
pbb	Input	PIN_58	4	B4_N0	2.5 V ...fault)		8mA (default)			
<<new node>>										

Рис. 4.5. Назначения в Pin Planner.

Также добавим файл для временных характеристик:

Verilog_labs - SDC1.sdc

```
#*******************************
# 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
#*******************************

derive_clock_uncertainty

#*******************************
# 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 {altera_reserved_tdi}]
set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {altera_reserved_tms}]


#*******************************
# Set Outnput 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]}]

set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {altera_reserved_tdo}]
```

Выполним полную компиляцию и проверим, выполняются ли временные характеристики:

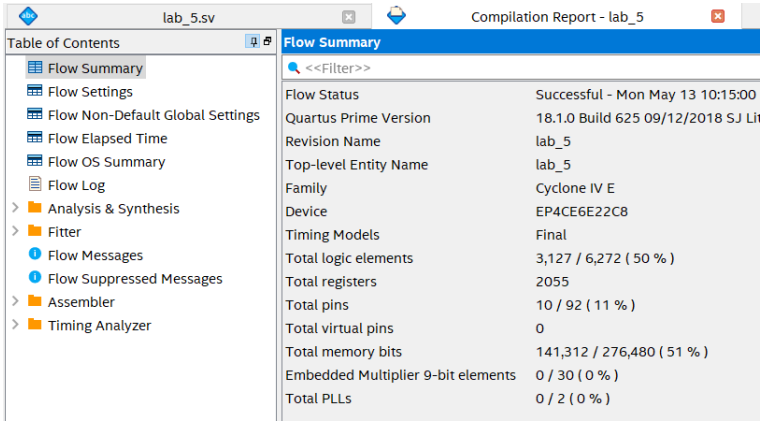

Рис. 4.6. Результат компиляции.

Как видим все временные характеристики выполняются.

Теперь перейдем к созданию проекта Nios II. Сам проект создается с настройками, рассмотренными в *Проект 1:*. Помимо этого, необходимо задать наш таймер на режим sys_clk_timer.

Текст программы приведен ниже:

Verilog_labs - lab5_source_a.c

```c
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include <time.h>
#include <unistd.h>
#include <sys\alt_alarm.h>
#include "stdio.h"

int main() {
    int i;
    alt_u32 num_ticks = 0;
    alt_u32 time1, time2, timer_overhead;
    printf("Процессор NIOS II запущен!\n");

    IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, 0x00);

    while(1) {
        time1 = alt_nticks();
        time2 = alt_nticks();
        timer_overhead = time2 - time1;

        time1 = alt_nticks();
        for(i = 1; i < 5; i++) {
            switch(i) {
            case 4:
                IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, 0xC0);
                printf("LED 11000000\n");
                break;
            case 3:
                IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, 0x30);
                printf("LED 00110000\n");
                break;
            case 2:
                IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, 0x0C);
                printf("LED 00001100\n");
                break;
            case 1:
                IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, 0x03);
                printf("LED 00000011\n");
                break;
            }
            usleep(500000);
        }
        time2 = alt_nticks();
        num_ticks = time2 - time1 - timer_overhead;
        printf("HAL system clock (Гц)\t: %u\n", (unsigned int)alt_ticks_per_second());
        printf("Число ticks\t\t: %u\n", (unsigned int)num_ticks);
        // Процессорное время
        printf("CPU time (ms)\t\t: %u\n", ((unsigned int)num_ticks * (unsigned int)1000 / ((unsigned int)alt_ticks_per_second()) ));
    }
    return 0;
}
```

Данный код в бесконечном цикле замеряет погрешность таймера, после чего выводит (используя цикл) заданную заданием последовательность с 500 ms интервалом, далее выводит затраченное число тиктов и процессорное время в ms затраченное на данную операцию.

Выполним компиляцию:

```
Problems  Tasks  Console  Properties
CDT Build Console [lab5_sw]
Info: (lab5_sw.elf) 9864 Bytes program size (code + initialized data).
Info:               6268 Bytes free for stack + heap.
Info: Creating lab5_sw.objdump
nios2-elf-objdump --disassemble --syms --all-header --source lab5_sw.elf >lab5_sw.objdump
[lab5_sw build complete]

09:41:48 Build Finished (took 1s.412ms)
```

Рис. 4.7. Результат компиляции.

Компиляция прошла успешно, запишем код на плату и увидим следующий результат в консоли:

```
Problems  Tasks  Console  Nios II Console  Properties
ab5_sw Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instanc
LED 00000011
LED 00001100
LED 00110000
LED 11000000
HAL system clock (Гц)          : 1000
Число ticks                    : 2357
CPU time (ms)                  : 2357
LED 00000011
LED 00001100
LED 00110000
```

Рис. 4.8. Результат запуска программы на плате.

Также видим, что на светодиодах также бегает по 2 потухших значения. Результат был продемонстрирован преподавателю практики.

Далее поменяем программу, сделаем чтоб при срабатывании таймера вызывалась функция, которая бы двигала светодиод:

```
Verilog_labs - lab5_source_b.c
1  #include "sys\alt_stdio.h"
2  #include "system.h"
3  #include "altera_avalon_pio_regs.h"
4  #include <time.h>
5  #include <unistd.h>
6  #include <sys\alt_alarm.h>
7
8  alt_u32 my_alarm_callback (void* context) {
9      alt_printf("Подпрограмма активизирована.\n");
10     volatile int* leds_val_ptr = (volatile int*) context;
11     if ((*leds_val_ptr) == 0x80) (*leds_val_ptr) = 0x01;
12     else (*leds_val_ptr) = (*leds_val_ptr) << 1;
13     IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, (*leds_val_ptr));
14     // IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, ~(*leds_val_ptr));
15     return alt_ticks_per_second();
16 }
17
18 int main() {
19     volatile int leds = 0x01;
20     static alt_alarm alarm;
21     IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, 0x00);
22
23     if (alt_alarm_start(&alarm, alt_ticks_per_second(), my_alarm_callback, (void*) &leds) < 0)
24         alt_printf("No system clock available!\n");
25     else
26         alt_printf("Процессор Nios II запущен!\n");
27     while (1) {}
28     return 0;
29 }
```

В main происходит настройка, чтоб при срабатывании таймера вызывалась функция my_alarm_callback.

В my_alarm_callback происходит сдвиг светодиода, после чего вывод на сами выводы. В случае, если нужно, чтоб бегал не потухших светодиод, а зажжённый, необходимо инвертировать выходные данные (т.е. как в строка 14).

Запишем данный код на плату и посмотрим на консоль:

```
Problems  Tasks  Console  Nios II Console  Properties
lab5_sw Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name:
Процессор Nios II запущен!
Подпрограмма активизирована.
Подпрограмма активизирована.
Подпрограмма активизирована.
```

Рис. 4.9. Результат запуска программы на плате.

Как можно заметить подпрограмма успешно запускается сразу после запуска процессора.

Также на светодиодах наблюдается бегающий светодиод (выключенный, если закомментирована строка 14, включенный, если 13).

Программа была успешно продемонстрирована преподавателю практики.
И последняя программа будет повторять первую, но будет использовать timestamp таймер:

Verilog_labs - lab5_source_c.c

```c
#include <stdio.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include <time.h>
#include <unistd.h>
#include <sys/alt_timestamp.h>

int main() {
    int i;
    alt_u32 num_ticks = 0;
    alt_u32 time1, time2, timer_overhead;
    IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, 0x00);

    if (alt_timestamp_start() < 0) printf("Проблема инициализации таймера!\n");
    else printf("Процессор Nios II запущен!\n");

    while (1) {
        time1 = alt_timestamp();
        time2 = alt_timestamp();
        timer_overhead = time2 - time1;
        time1 = alt_timestamp();
        for(i = 1; i < 5; i++) {
            switch(i) {
            case 4:
                IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, ~0xC0);
                printf("LED 11000000\n");
                break;
            case 3:
                IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, ~0x30);
                printf("LED 00110000\n");
                break;
            case 2:
                IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, ~0x0C);
                printf("LED 00001100\n");
                break;
            case 1:
                IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, ~0x03);
                printf("LED 00000011\n");
                break;
            }
            usleep(500000);
        }
        time2 = alt_timestamp();
        num_ticks = time2 - time1 - timer_overhead;
        if (alt_timestamp_start() < 0)
            printf("Проблема инициализации таймера!\n");
        printf("CPU clock (Гц)\t: %u\n", (unsigned int)alt_timestamp_freq());
        printf("Число ticks\t: %u\n", (unsigned int)num_ticks);
        // Процессорное время
        printf("CPU time (ms)\t: %u\n", ((unsigned int)num_ticks * (unsigned int)1000 / ((unsigned int)alt_timestamp_freq()) ));
    }
    return 0;
}
```

Поскольку она практически не отличается по логике работы, перейдем сразу к результату запуска на плате:

Problems | Tasks | Console | Nios II Console | Properties

lab5_sw Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instanc

```
LED 00001100
LED 00110000
LED 11000000
CPU clock (Гц)      : 25000000
Число ticks         : 56034965
CPU time (ms)       : 8
LED 00000011
LED 00001100
LED 00110000
LED 11000000
```

Рис. 4.10. Результат запуска программы на плате.

После запуска программы также начали переключаться и светодиоды.
Программа была успешно продемонстрирована преподавателю практики.

5. Проект 3:

5.1. Структура проекта:

Структура разрабатываемого проекта приведена ниже:

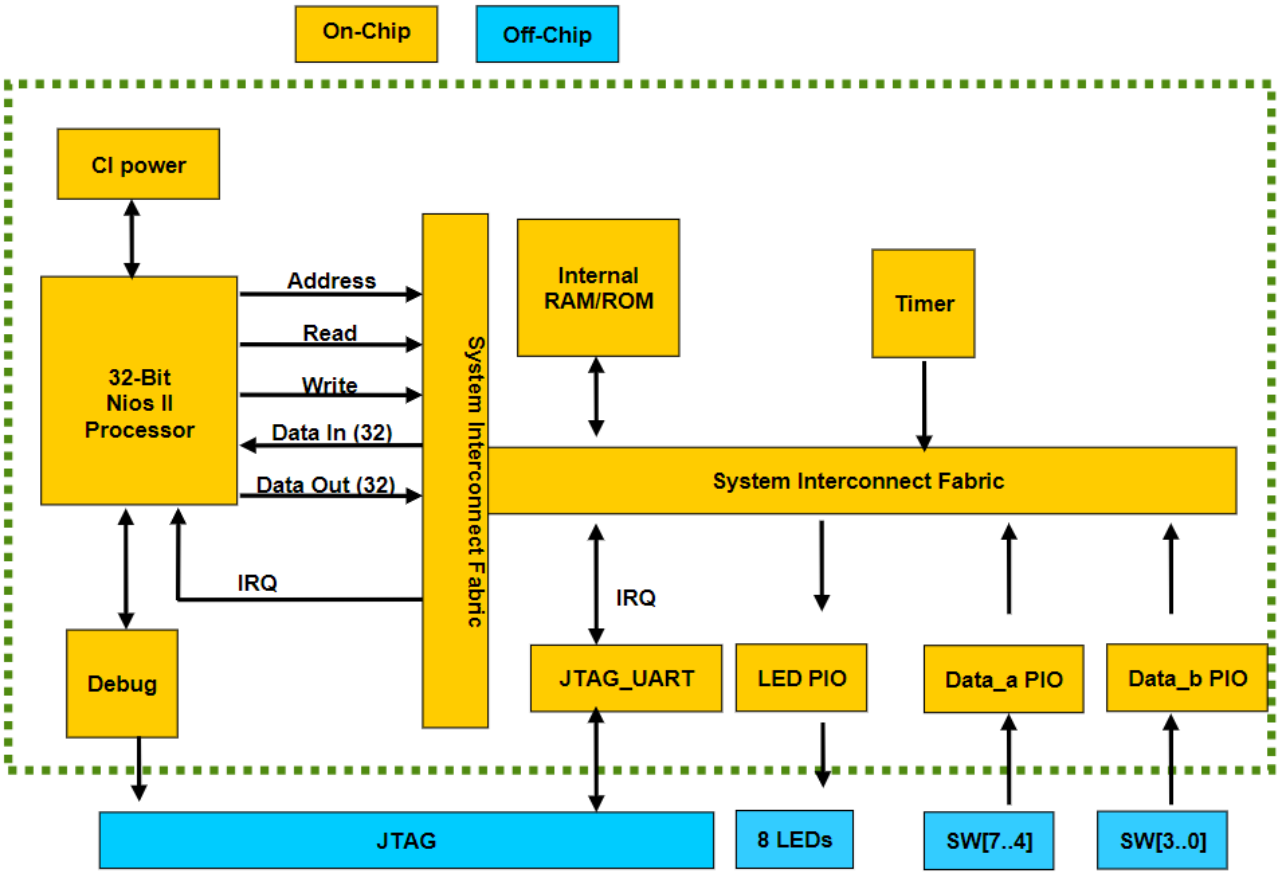

Рис. 5.1. Структура разрабатываемого проекта.

Этап 1.

- Под управлением процессора NIOSII обеспечивается:
  - Программное выполнение умножения данных, поступающих со входов Data_a и Data_b;
  - Измерение числа ticks, требуемых для реализации операции умножения и вывод соответствующей информации в окно консоли.

Этап 2.

- Под управлением процессора NIOSII обеспечивается:
  - Аппаратное (пользовательская инструкция CI_Power) выполнение умножения данных, поступающих со входов Data_a и Data_b;
  - Измерение числа ticks, требуемых для реализации операции умножения и вывод соответствующей информации в окно консоли.

5.2. Решение:

Выполним создание проекта в QP с стандартными настройками.

Разработаем модуль, который будет использоваться в проекте для ускорения (в качестве custom instruction):

Verilog_labs - CI_Power.sv

```
module CI_Power (
    input bit [31:0] dataa,
    input bit [31:0] datab,
    output bit [31:0] result
);

assign result = dataa * datab;

endmodule
```

Далее переходим в Platform Designer. Скопируем PD из *Проект 3:*.

Создадим новый компонент, добавим CI_Power в PD:

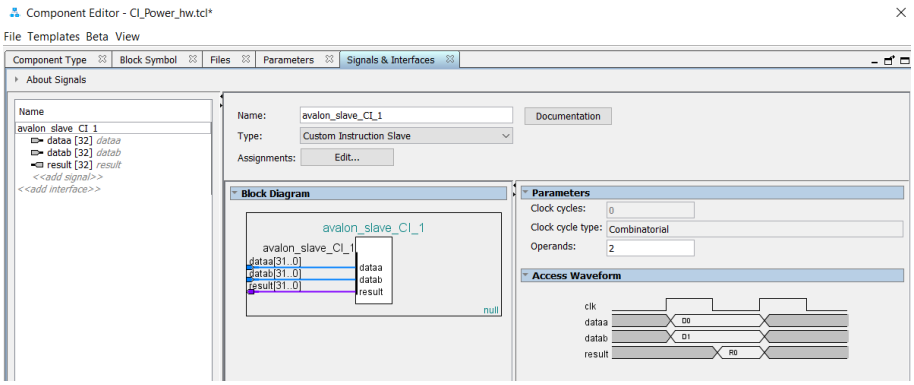

Рис. 5.2. Добавление модуля Custom Instruction.

Его блок будет выглядеть следующим образом:

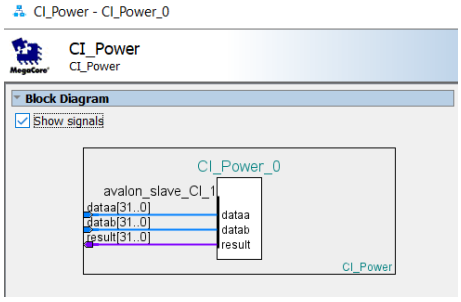

Рис. 5.3. Блок модуля Custom Instruction.

В качестве операндов у нас будут выступать переключатели (по 4 бита), для них нужно добавить 2 модуля ввода:

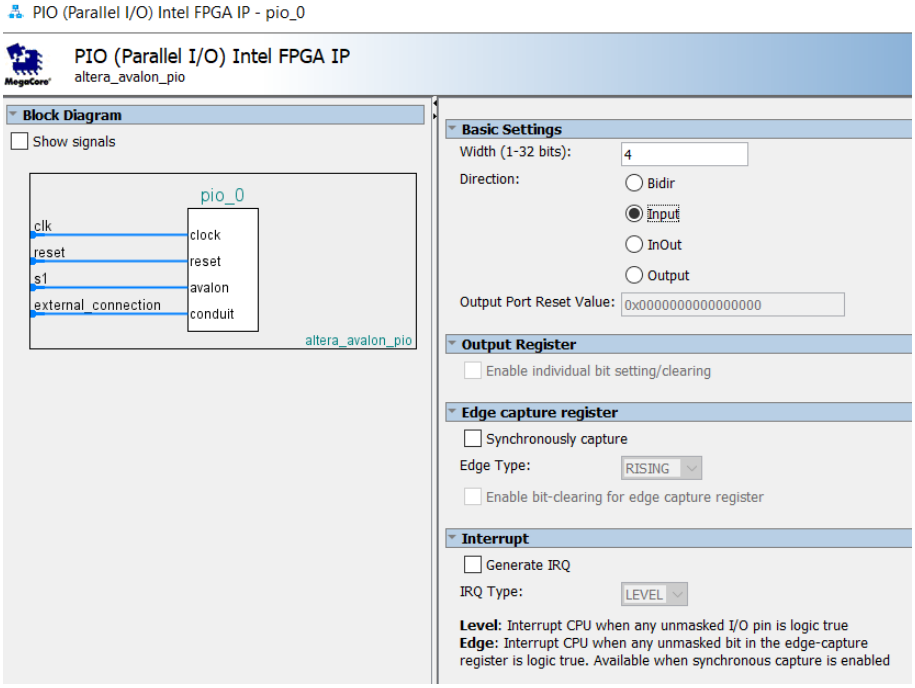

Рис. 5.4. Настройки модуля ввода 1.

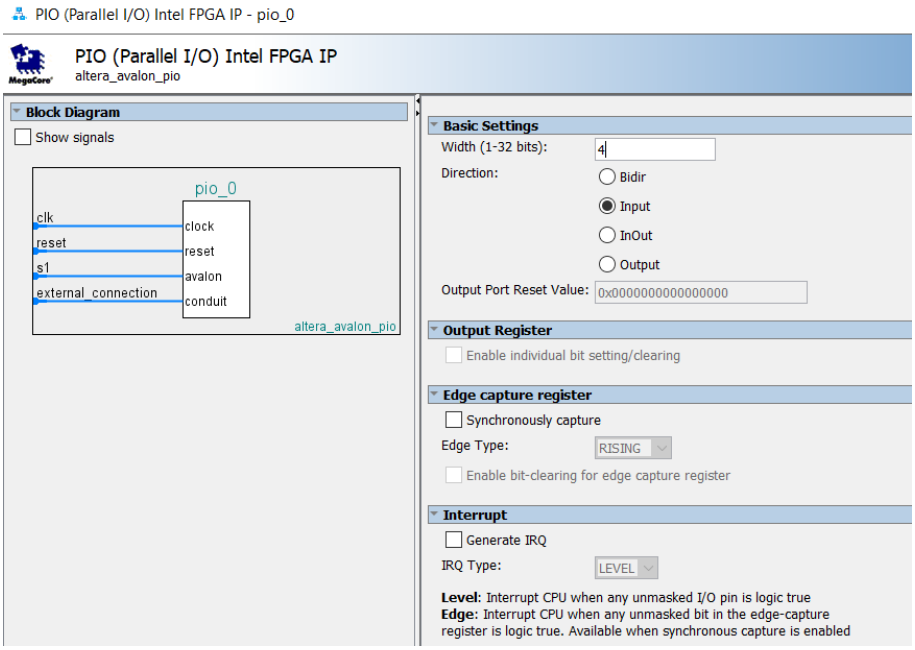

Рис. 5.5. Настройки модуля ввода 2.

Выполним все подключения и получим следующую схему в PD:

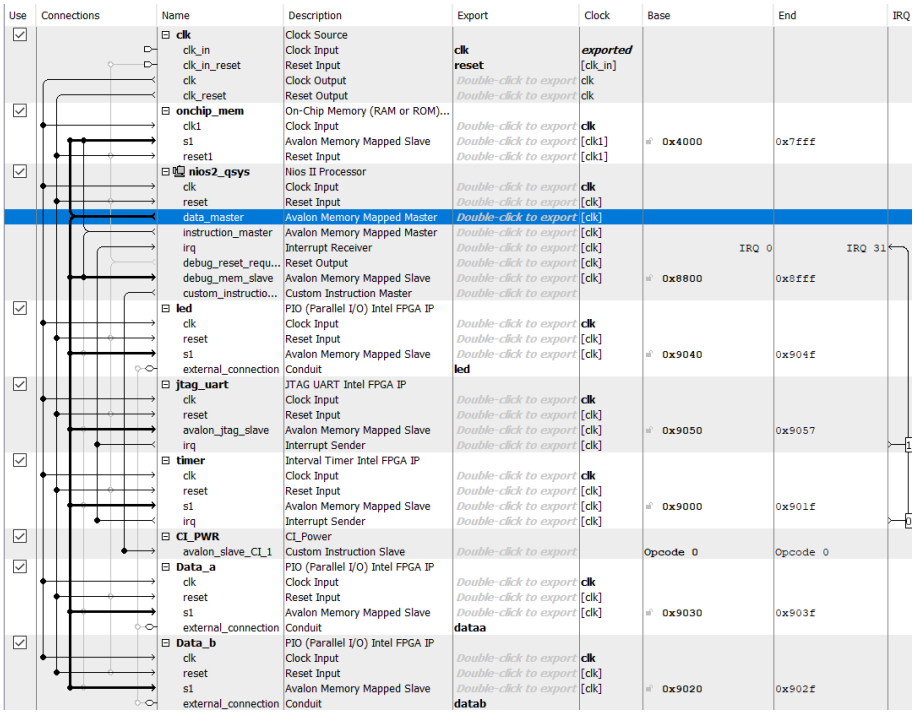

Рис. 5.6. Подключения в PD.

Schem разработанного устройства приведена ниже:

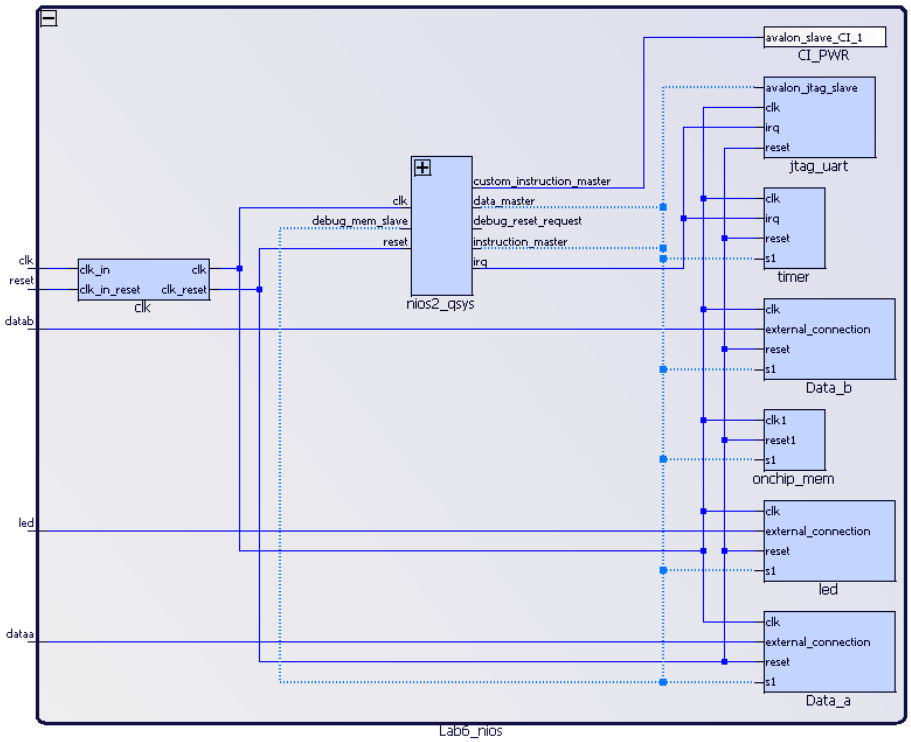

Рис. 5.7. Схема разработанного устройства.

Справа сверху отображает созданная нами инструкция в белом прямоугольнике.

Выполним генерацию и создадим файл верхнего уровня:

Verilog_labs - lab6.sv

```
module lab6 (
  input  bit       clk,
  input  bit       pbb,
  input  bit [7:0] sw,
  output bit [7:0] led
);

  Lab6_nios u0 (
    .clk_clk      (clk),      //   clk.clk
    .led_export   (led),      //   led.export
    .reset_reset_n(pbb),      // reset.reset_n
    .dataa_export (sw[7:4]),  // dataa.export
    .datab_export (sw[3:0])   // datab.export
  );

endmodule
```

Выполним компиляцию и выполним назначения для входов и выходов:

Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	urrent Streng	Slew Rate	ifferential Pai	ict Preservati
clk	Input	PIN_23	1	B1_N0	3.3-V LVTTL		8mA (default)			
led[7]	Output	PIN_65	4	B4_N0	2.5 V		8mA (default)	2 (default)		
led[6]	Output	PIN_66	4	B4_N0	2.5 V		8mA (default)	2 (default)		
led[5]	Output	PIN_67	4	B4_N0	2.5 V		8mA (default)	2 (default)		
led[4]	Output	PIN_68	4	B4_N0	2.5 V		8mA (default)	2 (default)		
led[3]	Output	PIN_69	4	B4_N0	2.5 V		8mA (default)	2 (default)		
led[2]	Output	PIN_70	4	B4_N0	2.5 V		8mA (default)	2 (default)		
led[1]	Output	PIN_71	4	B4_N0	2.5 V		8mA (default)	2 (default)		
led[0]	Output	PIN_72	4	B4_N0	2.5 V		8mA (default)	2 (default)		
pbb	Input	PIN_58	4	B4_N0	2.5 V		8mA (default)			
sw[7]	Input	PIN_88	5	B5_N0	3.3-V LVTTL		8mA (default)			
sw[6]	Input	PIN_89	5	B5_N0	3.3-V LVTTL		8mA (default)			
sw[5]	Input	PIN_90	6	B6_N0	3.3-V LVTTL		8mA (default)			
sw[4]	Input	PIN_91	6	B6_N0	3.3-V LVTTL		8mA (default)			
sw[3]	Input	PIN_49	3	B3_N0	3.3-V LVTTL		8mA (default)			
sw[2]	Input	PIN_46	3	B3_N0	3.3-V LVTTL		8mA (default)			
sw[1]	Input	PIN_25	2	B2_N0	3.3-V LVTTL		8mA (default)			
sw[0]	Input	PIN_24	2	B2_N0	3.3-V LVTTL		8mA (default)			
<<new node>>										

Рис. 5.8. Назначения в Pin Planner.

Также добавим файл для временных характеристик:

```
Verilog_labs - SDC1.sdc
1  #**************************************
2  # Time Information
3  #**************************************
4
5  set_time_format -unit ns -decimal_places 3
6
7  #**************************************
8  # Create Clock
9  #**************************************
10
11 create_clock -name {clock} -period 40.000 -waveform {0.000 20.000} [get_ports {clk}]
12
13 #**************************************
14 # Set Clock Uncertainty
15 #**************************************
16
17 derive_clock_uncertainty
18
19 #**************************************
20 # Set Input Delay
21 #**************************************
22
23 set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {pbb}]
24 set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {sw[0]}]
25 set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {sw[1]}]
26 set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {sw[2]}]
27 set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {sw[3]}]
28 set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {sw[4]}]
29 set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {sw[5]}]
30 set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {sw[6]}]
31 set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {sw[7]}]
32
33
34 set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {altera_reserved_tdi}]
35 set_input_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {altera_reserved_tms}]
36
37
38 #**************************************
39 # Set Outnput Delay
40 #**************************************
41
42 set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[0]}]
43 set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[1]}]
44 set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[2]}]
45 set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[3]}]
46 set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[4]}]
47 set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[5]}]
48 set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[6]}]
49 set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {led[7]}]
50
51 set_output_delay -add_delay -clock [get_clocks {clock}] 10.000 [get_ports {altera_reserved_tdo}]
```

Выполним полную компиляцию и проверим, выполняются ли временные характеристики:

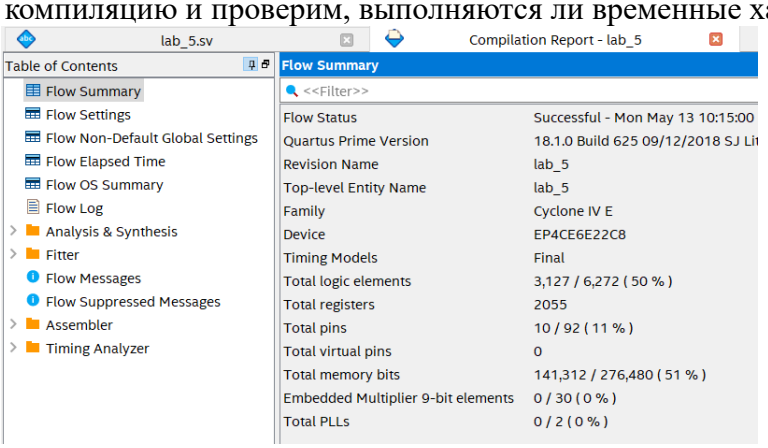

Рис. 5.9. Результат компиляции.

Как видим все временные характеристики выполняются.

Теперь перейдем к созданию проекта Nios II. Сам проект создается с настройками, рассмотренными в *Проект 3:*.

```c
Verilog_labs - lab6_source_sw.c
1  #include <stdio.h>
2  #include "system.h"
3  #include "altera_avalon_pio_regs.h"
4  #include <time.h>
5  #include <unistd.h>
6  #include <sys/alt_timestamp.h>
7
8  int main() {
9      int led;
10     int in_a;
11     int in_b;
12     alt_u32 num_ticks = 0;
13     alt_u32 time1, time2, timer_overhead;
14
15     if (alt_timestamp_start() < 0) printf("Проблема инициализации таймера!\n");
16     else printf("Процессор Nios II запущен!\n");
17     printf("Частота процессора - CPU Clock (Гц): %u\n", (unsigned int)alt_timestamp_freq());
18
19     IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, 0x00);
20     while(1) {
21         if (alt_timestamp_start() < 0)
22             printf("Проблема инициализации таймера!\n");
23         time1 = alt_timestamp();
24         time2 = alt_timestamp();
25         timer_overhead = time2 - time1;
26
27         in_a = IORD_ALTERA_AVALON_PIO_DATA(DATA_A_BASE);
28         in_b = IORD_ALTERA_AVALON_PIO_DATA(DATA_B_BASE);
29         time1 = alt_timestamp();
30         //led = (in_a * in_b);
31         led = ALT_CI_CI_PWR(in_a, in_b);
32         time2 = alt_timestamp();
33         num_ticks = time2 - time1 - timer_overhead;
34
35         IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, ~led);
36         printf("Результат: %u\t\tЧисло ticks: %u\n", (unsigned int) led, (unsigned int) num_ticks);
37         usleep(500000);
38     }
39     return 0;
40 }
41
```

Данный проект замеряет время работы функции, сначала при написании в коде (закомментирована строка 31 и раскомментирована строка 30) и при использовании Custom Instruction (раскомментирована строка 31 и закомментирована строка 30).

Выполним его компиляцию, сначала при варианте реализации умножения на плате:

```
Problems  Tasks  Console  Properties
CDT Build Console [lab6_sw]
nios2-elf-insert lab6_sw.elf --thread_model hal --cpu_name nios2_qsys --qsys true --simulation_
Info: (lab6_sw.elf) 4856 Bytes program size (code + initialized data).
Info:               11 KBytes free for stack + heap.
Info: Creating lab6_sw.objdump
nios2-elf-objdump --disassemble --syms --all-header --source lab6_sw.elf >lab6_sw.objdump
[lab6_sw build complete]
```

Рис. 5.10. Результат компиляции.

Как видим, компиляция прошла успешно, запустим код на плате, переключим переключатели несколько раз, результат в консоли:

```
Problems  Tasks  Console  Nios II Console  Properties
lab6_sw Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instal
Результат: 9                    Число ticks: 145
Результат: 6                    Число ticks: 145
Результат: 6                    Число ticks: 145
Результат: 2                    Число ticks: 101
Результат: 2                    Число ticks: 101
Результат: 10                   Число ticks: 182
Результат: 26                   Число ticks: 226
```

Рис. 5.11. Результат запуска программы на процессоре.

Как видим, инструкция выполняется за разное количество тиков, в зависимости от данных, а также имеет значение тиков около 100-200.

Также в ходе эксперимента видим дублирование вывода в консоли и на светодиодах.

Теперь выполним компиляцию с вариантом умножения в качестве Custom Instruction:

```
Problems  Tasks  Console  Properties
CDT Build Console [lab6_sw]
nios2-elf-insert lab6_sw.elf --thread_model hal --cpu_name nios2_qsys --qsys true --simulation_
Info: (lab6_sw.elf) 4856 Bytes program size (code + initialized data).
Info:               11 KBytes free for stack + heap.
Info: Creating lab6_sw.objdump
nios2-elf-objdump --disassemble --syms --all-header --source lab6_sw.elf >lab6_sw.objdump
[lab6_sw build complete]
```

Рис. 5.12. Результат компиляции.

Как видим, размер файла не изменился, однако запустим теперь этот код на плате, переключим переключатели несколько раз, результат в консоли:

```
Problems  Tasks  Console  Nios II Console  Properties
lab6_sw Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instar
Результат: 1                    Число ticks: 36
Результат: 3                    Число ticks: 36
Результат: 9                    Число ticks: 36
Результат: 21                   Число ticks: 36
Результат: 21                   Число ticks: 36
Результат: 7                    Число ticks: 36
Результат: 35                   Число ticks: 36
Результат: 35                   Число ticks: 36
```

Рис. 5.13. Результат запуска программы на процессоре.

Как видим время выполнения не изменяется и является очень маленьким.

Теперь вместо умножения необходимо реализовать функцию в соответствии с вариантом, а именно: $(A + B) * A - B$. Для того, чтоб создать Custom Instruction создадим следующий файл:

Verilog_labs - CI_018.sv

```
1  module CI_018 (
2      input bit [31:0] dataa,
3      input bit [31:0] datab,
4      output bit [31:0] result
5  );
6
7  assign result = (dataa + datab) * dataa - dataa;
8
9  endmodule
```

Далее все действия аналогичны и в отчете будут опущены.

Программа для NIOS II практически идентична, кроме вызова Custom Instruction:

Verilog_labs - lab6_source_sw.c

```
1  //led = (in_a + in_b) * in_a - in_a;
2  led = ALT_CI_CI_018_0(in_a, in_b);
```

Выполним компиляцию и запишем на плату вариант с реализацией логики на процессоре:

```
Problems  Tasks  Console  Nios II Console  Properties
lab_6_sw Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 insta
Результат: 7                    Число ticks: 220
Результат: 7                    Число ticks: 220
Результат: 0                    Число ticks: 213
Результат: 16                   Число ticks: 243
Результат: 16                   Число ticks: 243
Результат: 72                   Число ticks: 250
Результат: 48                   Число ticks: 243
Результат: 48                   Число ticks: 243
```

Рис. 5.14. Результат запуска на процессоре.

Однако если повторно скомпилировать, но уже Custom Instruction:

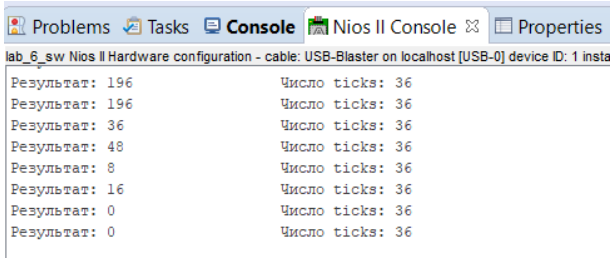

Рис. 5.15. Результат запуска на процессоре.

Как видим, тенденция сохраняется и вызов Custom Instruction занимает всего 36 тика, в то время как реализация этой же инструкции на процессоре аж 200 (и колеблется).

Все результаты были показаны преподавателю практики.

6. Вывод:

В ходе курсовой работы были изучены различные возможности процессора Nios II.

В первой программе были получены навыки по оптимизации программного кода и уменьшению его объема, занимаемого на кристалле, что позволит делать комплексные проекты тратя минимальные ресурсы на память.

Во втором проекте мы поработали с прерываниями, они позволяют эффективно обрабатывать запросы пользователей к программе.

В третьем проекте был рассмотрен таймер и принцип работы с ним на прерываниях. Таймер рассматривался в двух режимах, а именно sys_clk_timer и timestamp. Сравнивая результаты на Рис. 4.8 и Рис. 4.10, видно, что эти таймеры отличаются базовой тактовой частотой, что подтверждает лекционный материал, отсюда и такая большая разница в цифрах. Эти таймеры могут помочь делать какие-то действия с заданной периодичностью (как это было во второй программе), либо измерять время работы программы.

В четвертом проекте были получены навыки по работе с Custom Instruction, было показано, что они работают куда быстрее, чем аналогичные инструкции, реализованные на процессоре. Также были отточены навыки по использованию таймера для замера времени работы программы.