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](#_Toc166506407)

[2. Проект 1: 4](#_Toc166506408)

[2.1. Структура проекта: 4](#_Toc166506409)

[2.2. Решение: 4](#_Toc166506410)

[3. Проект 2: 16](#_Toc166506411)

[3.1. Структура проекта: 16](#_Toc166506412)

[3.2. Решение: 16](#_Toc166506413)

[4. Проект 3: 20](#_Toc166506414)

[4.1. Структура проекта: 20](#_Toc166506415)

[4.2. Решение: 20](#_Toc166506416)

[5. Проект 3: 25](#_Toc166506417)

[5.1. Структура проекта: 25](#_Toc166506418)

[5.2. Решение: 26](#_Toc166506419)

[6. Вывод: 33](#_Toc166506420)

# Список иллюстраций:

[Рис. 2.1. Структура разрабатываемого проекта. 4](#_Toc166506346)

[Рис. 2.2. Настройки для QP. 4](#_Toc166506347)

[Рис. 2.3. Настройки модуля счетчика. 5](#_Toc166506348)

[Рис. 2.4. Настройка модуля памяти. 5](#_Toc166506349)

[Рис. 2.5. Модуль ввода для светодиодов. 6](#_Toc166506350)

[Рис. 2.6. Модуль ввода для кнопки. 6](#_Toc166506351)

[Рис. 2.7. Модуль JTAG UART. 7](#_Toc166506352)

[Рис. 2.8. Настройки вектора для Nios II. 7](#_Toc166506353)

[Рис. 2.9. Platform Designer. 7](#_Toc166506354)

[Рис. 2.10. Schematic в Platform Designer. 8](#_Toc166506355)

[Рис. 2.11. System Interconnections. 8](#_Toc166506356)

[Рис. 2.12. RTL Viewer. 9](#_Toc166506357)

[Рис. 2.13. Pin Planner. 9](#_Toc166506358)

[Рис. 2.14. Результат компиляции. 10](#_Toc166506359)

[Рис. 2.15. Создание проекта для Nios II. 11](#_Toc166506360)

[Рис. 2.16. Создание файла с исходным кодом. 11](#_Toc166506361)

[Рис. 2.17. Настройки BSP. 12](#_Toc166506362)

[Рис. 2.18. Компиляция программы. 12](#_Toc166506363)

[Рис. 2.19. Приветственные сообщения после запуска. 12](#_Toc166506364)

[Рис. 2.20. Консоль после нескольких нажатий на pba. 12](#_Toc166506365)

[Рис. 2.21. Новые настройки для BSP. 13](#_Toc166506366)

[Рис. 2.22. Компиляция проекта с новыми настройками BSP. 13](#_Toc166506367)

[Рис. 2.23. Результат компиляции. 14](#_Toc166506368)

[Рис. 2.24. Optimization level: size. У lab3\_sw\_bsp. 14](#_Toc166506369)

[Рис. 2.25. Optimization level: size. У lab3\_sw. 15](#_Toc166506370)

[Рис. 2.26. Компиляция после выставления настроек оптимизации. 15](#_Toc166506371)

[Рис. 2.27. Результат запуска программы и нескольких переключений pba. 15](#_Toc166506372)

[Рис. 3.1. Структура разрабатываемого проекта. 16](#_Toc166506373)

[Рис. 3.2. Настройки для QP. 16](#_Toc166506374)

[Рис. 3.3. Новые настройки модуля PIO. 17](#_Toc166506375)

[Рис. 3.4. Подключение прерываний к процессору. 17](#_Toc166506376)

[Рис. 3.5. Назначения в Pin Planner. 17](#_Toc166506377)

[Рис. 3.6. Результат компиляции. 18](#_Toc166506378)

[Рис. 3.7. Результат компиляции. 19](#_Toc166506379)

[Рис. 3.8. Консоль проекта после записи программы. 19](#_Toc166506380)

[Рис. 3.9. Консоль после нескольких запусков программы. 20](#_Toc166506381)

[Рис. 4.1. Структура разрабатываемого проекта. 20](#_Toc166506382)

[Рис. 4.2. Настройки для QP. 21](#_Toc166506383)

[Рис. 4.3. Настройки компонента timer. 21](#_Toc166506384)

[Рис. 4.4. Подключение таймера к процессору. 21](#_Toc166506385)

[Рис. 4.5. Назначения в Pin Planner. 22](#_Toc166506386)

[Рис. 4.6. Результат компиляции. 23](#_Toc166506387)

[Рис. 4.7. Результат компиляции. 24](#_Toc166506388)

[Рис. 4.8. Результат запуска программы на плате. 24](#_Toc166506389)

[Рис. 4.9. Результат запуска программы на плате. 24](#_Toc166506390)

[Рис. 4.10. Результат запуска программы на плате. 25](#_Toc166506391)

[Рис. 5.1. Структура разрабатываемого проекта. 26](#_Toc166506392)

[Рис. 5.2. Добавление модуля Custom Instruction. 27](#_Toc166506393)

[Рис. 5.3. Блок модуля Custom Instruction. 27](#_Toc166506394)

[Рис. 5.4. Настройки модуля ввода 1. 27](#_Toc166506395)

[Рис. 5.5. Настройки модуля ввода 2. 28](#_Toc166506396)

[Рис. 5.6. Подключения в PD. 28](#_Toc166506397)

[Рис. 5.7. Схема разработанного устройства. 29](#_Toc166506398)

[Рис. 5.8. Назначения в Pin Planner. 29](#_Toc166506399)

[Рис. 5.9. Результат компиляции. 30](#_Toc166506400)

[Рис. 5.10. Результат компиляции. 31](#_Toc166506401)

[Рис. 5.11. Результат запуска программы на процессоре. 31](#_Toc166506402)

[Рис. 5.12. Результат компиляции. 32](#_Toc166506403)

[Рис. 5.13. Результат запуска программы на процессоре. 32](#_Toc166506404)

[Рис. 5.14. Результат запуска на процессоре. 32](#_Toc166506405)

[Рис. 5.15. Результат запуска на процессоре. 33](#_Toc166506406)

# Проект 1:

## Структура проекта:

Структура разрабатываемого проекта приведена ниже:

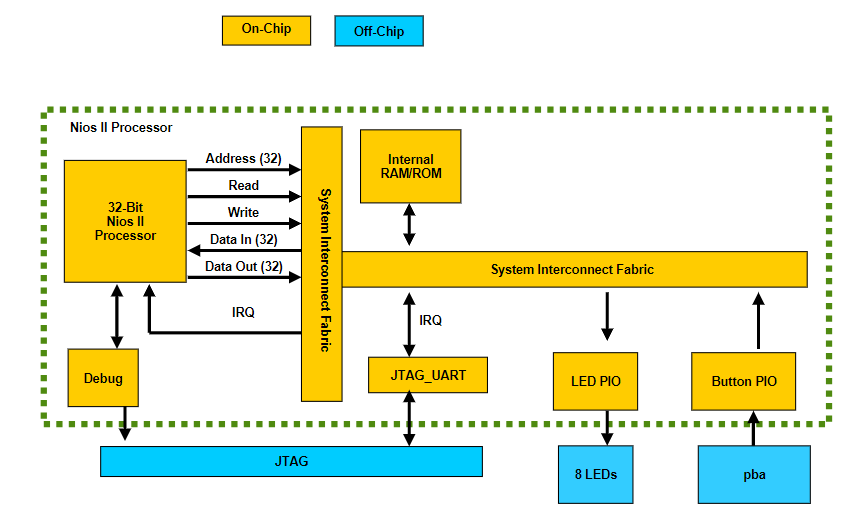


Рис. 2.1. Структура разрабатываемого проекта.

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).

## Решение:

Выполним создание проекта в QP со следующими настройками:

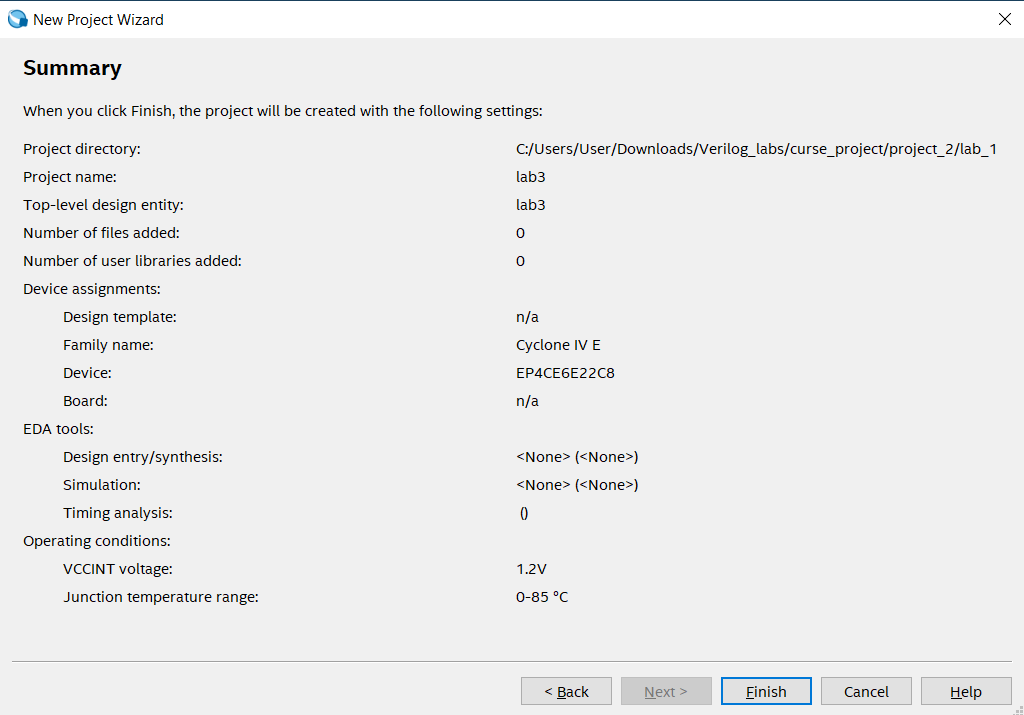


Рис. 2.2. Настройки для QP.

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

Начнем с модуля clk:

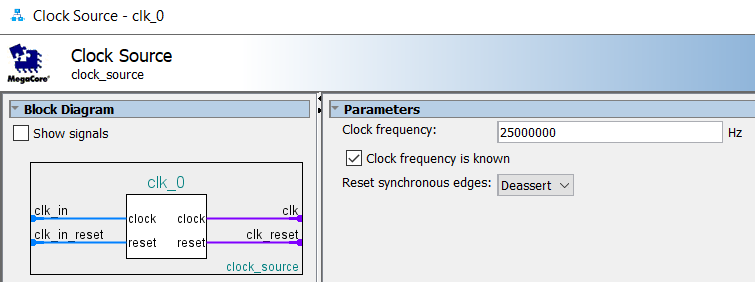


Рис. 2.3. Настройки модуля счетчика.

Добавим модуль памяти для процессора Nios II со следующими настройками:

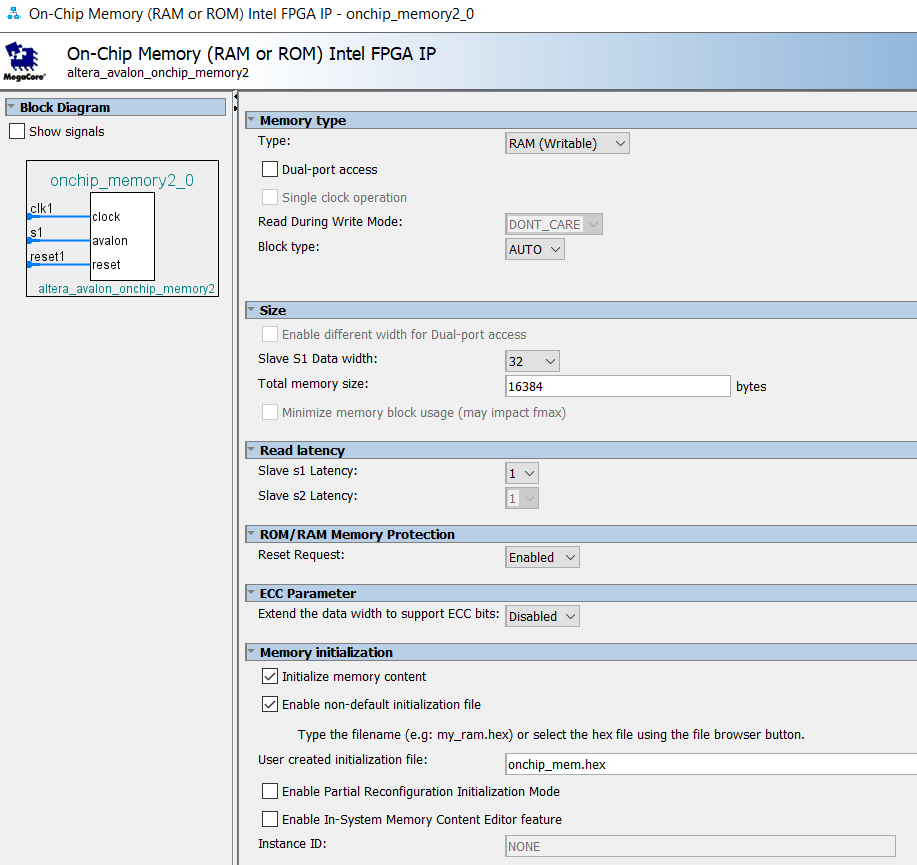


Рис. 2.4. Настройка модуля памяти.

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

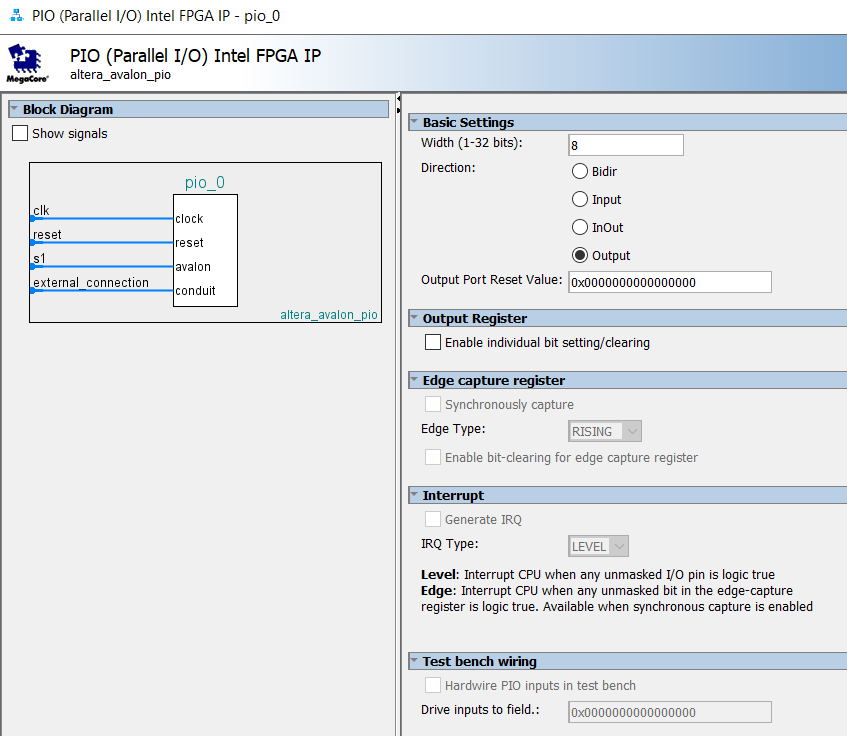


Рис. 2.5. Модуль ввода для светодиодов.

И модуль ввода для кнопки:

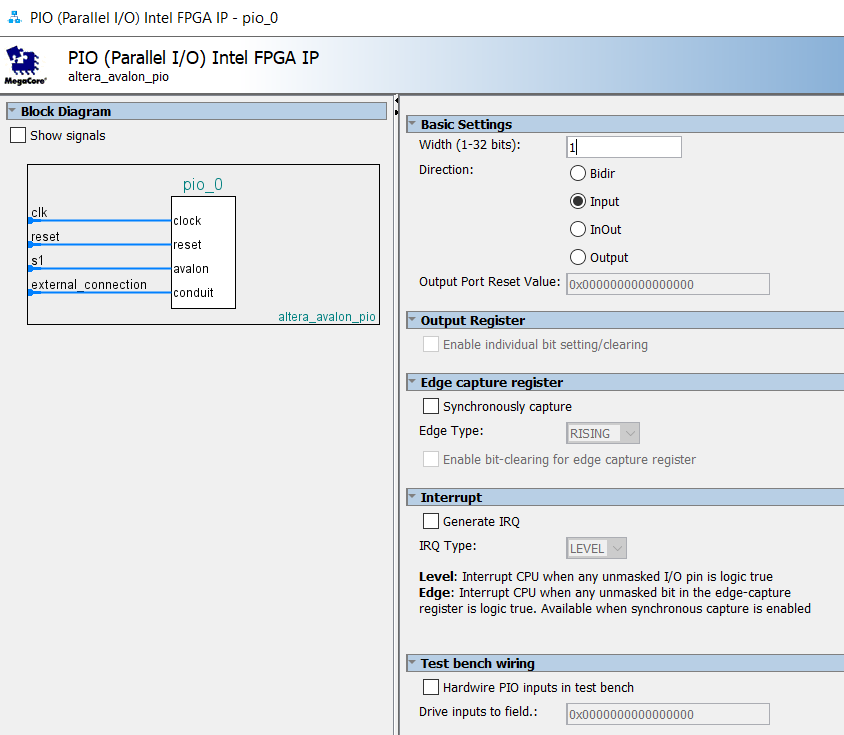


Рис. 2.6. Модуль ввода для кнопки.

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

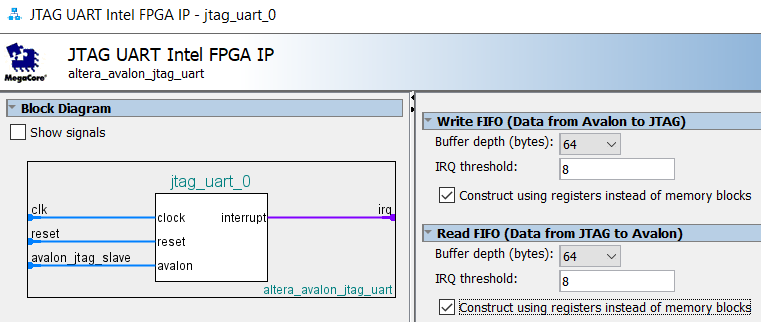


Рис. 2.7. Модуль JTAG UART.

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

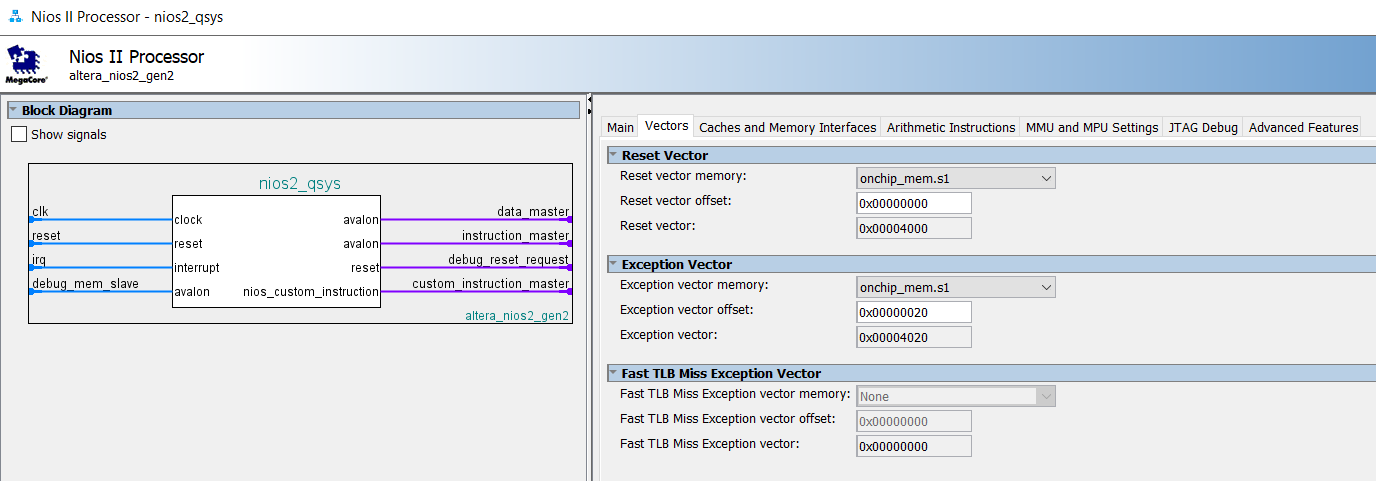


Рис. 2.8. Настройки вектора для Nios II.

Выполним соединения в Platform Designer созданных модулей следующим образом:

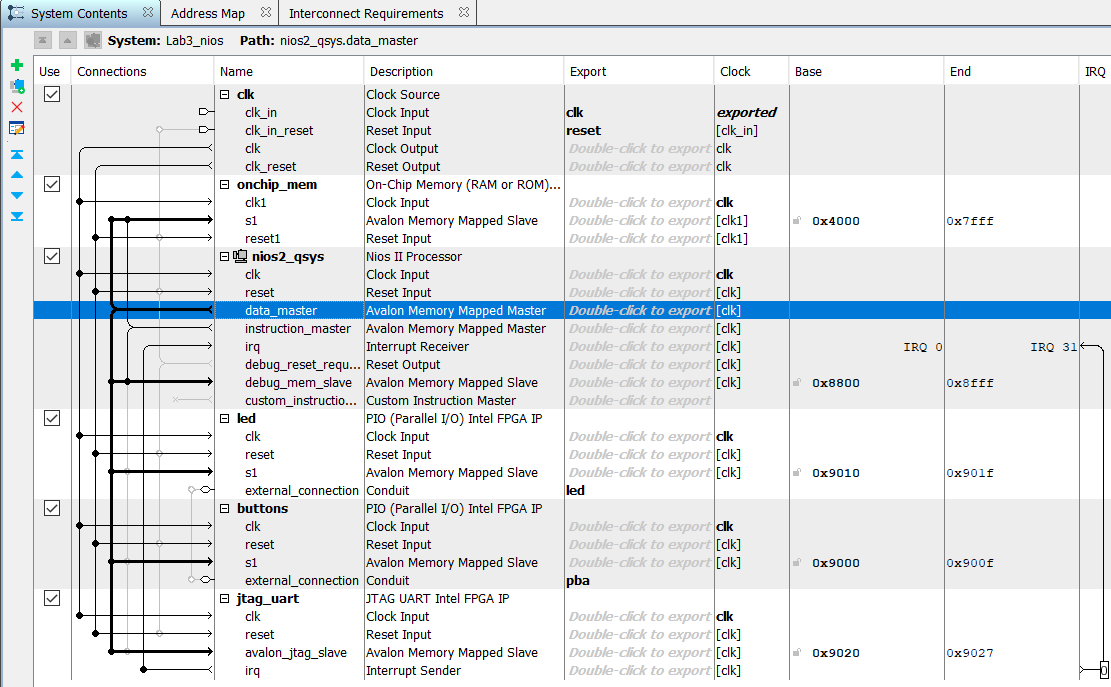


Рис. 2.9. Platform Designer.

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:



Выполним компиляцию и посмотрим на результат в RTL Viewer:

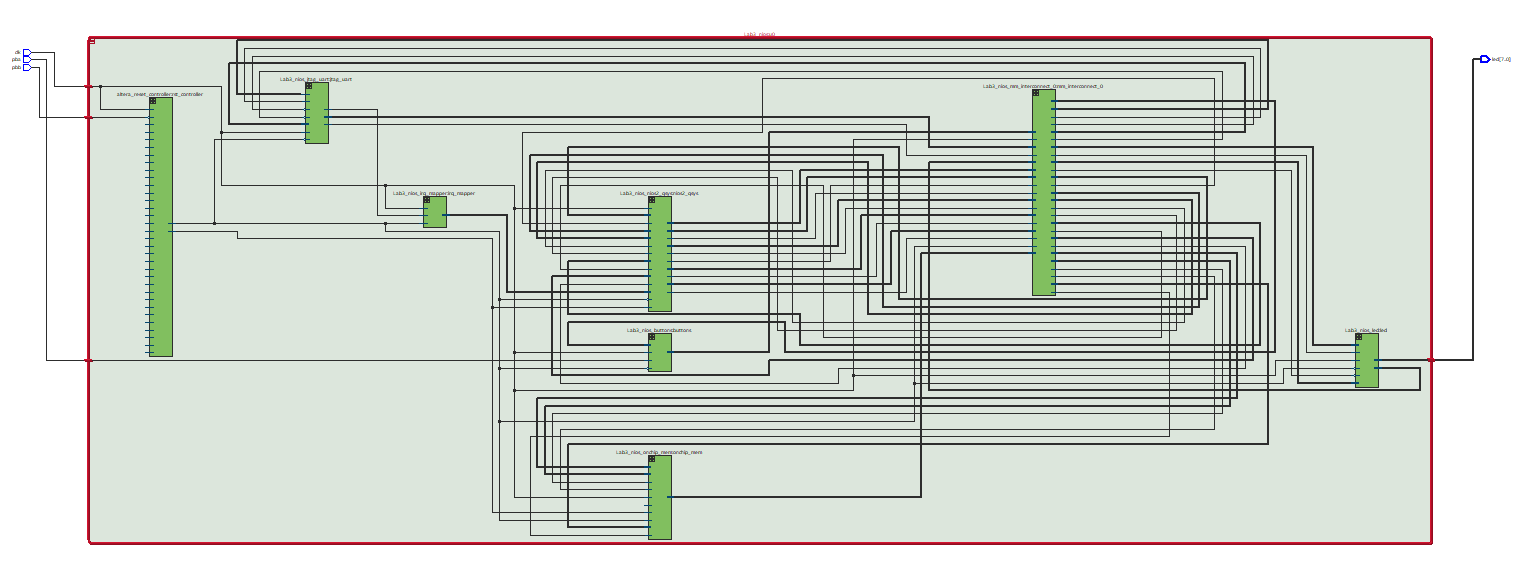


Рис. 2.12. RTL Viewer.

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

Далее выполним назначения входов и выходов в Pin Planner:

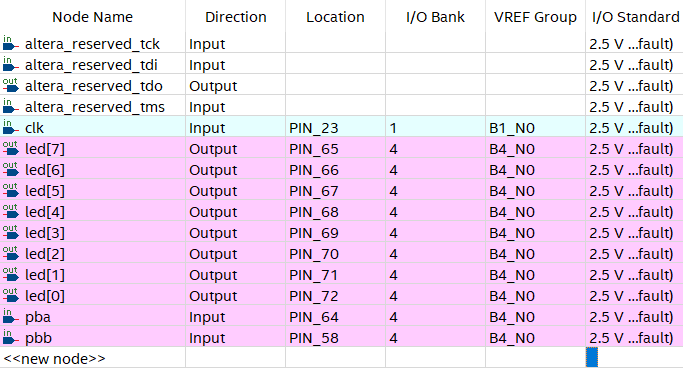


Рис. 2.13. Pin Planner.

Также добавим .sdc файл с временными требованиями:



Проведем полную компиляцию и проверим, что все они выполняются:

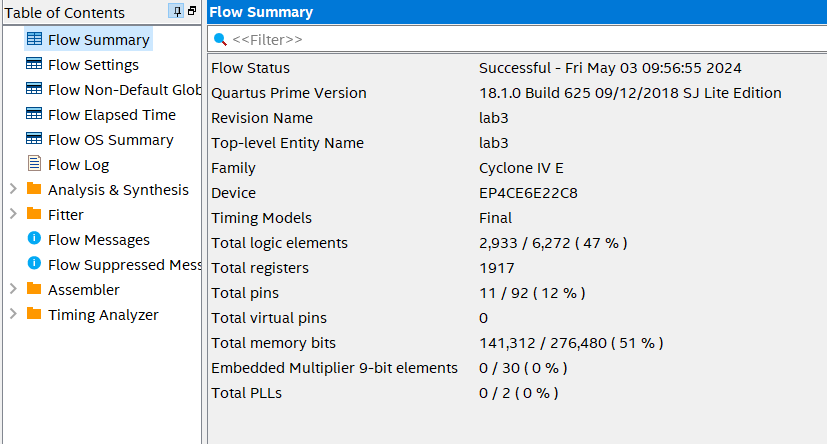


Рис. 2.14. Результат компиляции.

Как можно видеть, компиляция прошла успешно и временные требования выполняются.

Перейдем к разработке программа для Nios II:

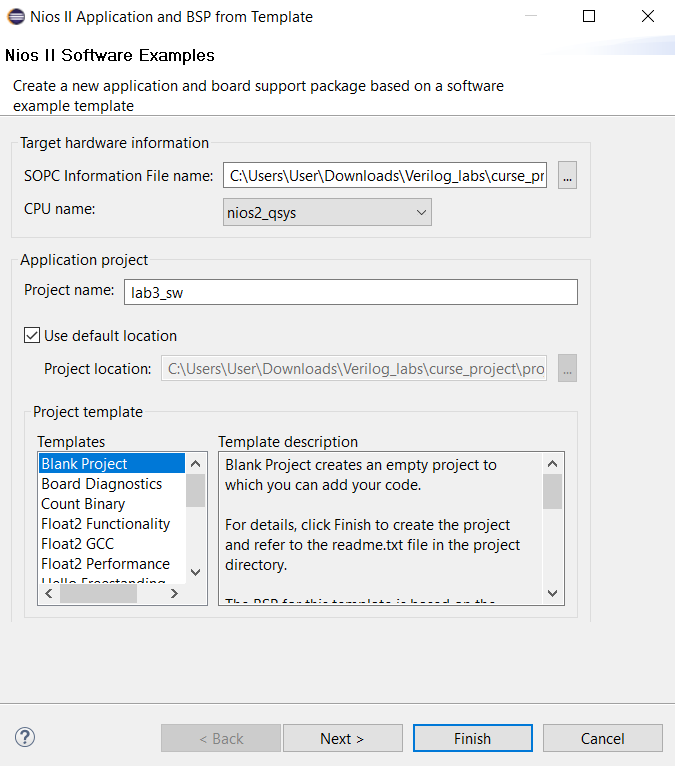


Рис. 2.15. Создание проекта для Nios II.

Создадим файл с исходным кодом:



Рис. 2.16. Создание файла с исходным кодом.

Код приведен ниже:



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:

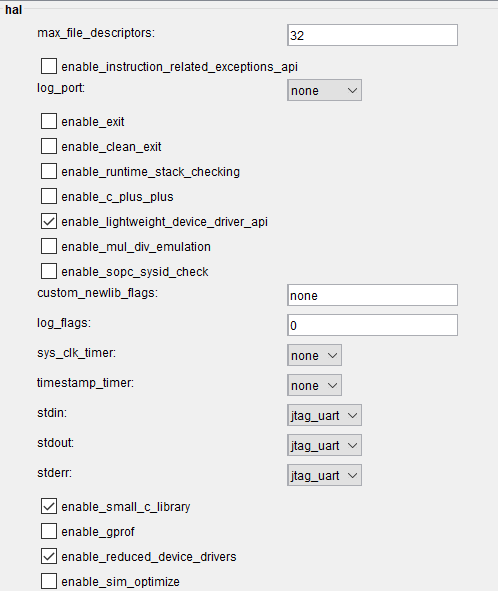


Рис. 2.17. Настройки BSP.

И выполним компиляцию:

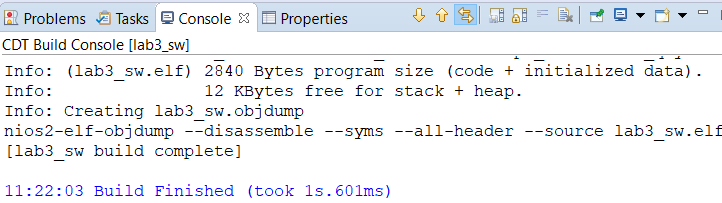


Рис. 2.18. Компиляция программы.

Компиляция прошла успешно.

Теперь запишем проект на плату и посмотрим на результат в консоли:

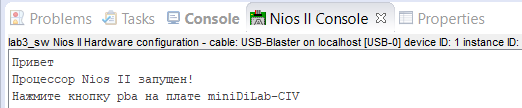


Рис. 2.19. Приветственные сообщения после запуска.

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

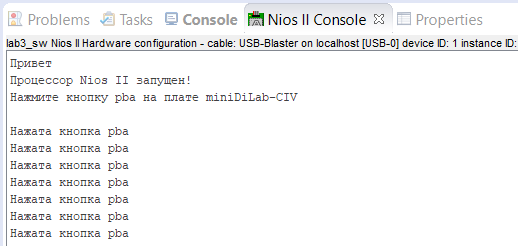


Рис. 2.20. Консоль после нескольких нажатий на pba.

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:

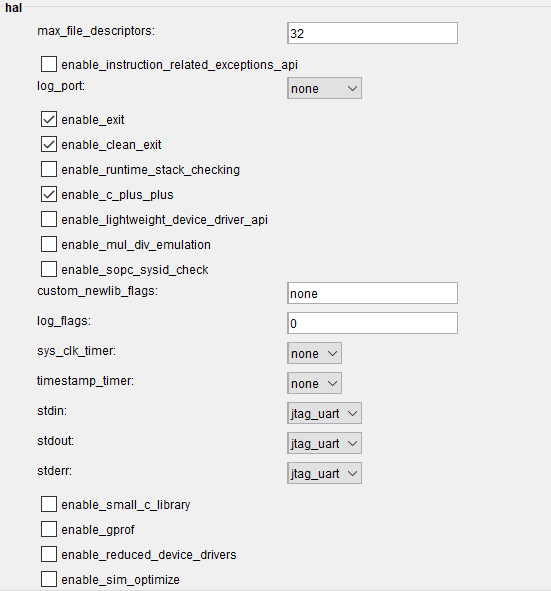
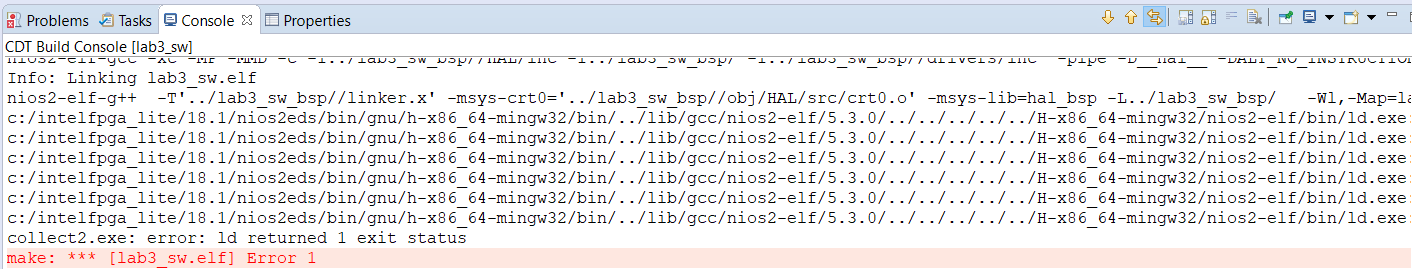


Рис. 2.21. Новые настройки для BSP.

Выполним генерацию этого BSP и компиляцию проекта:



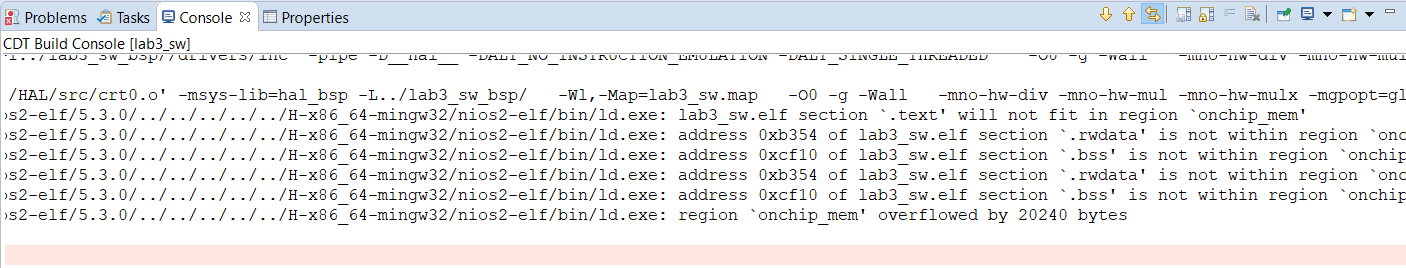
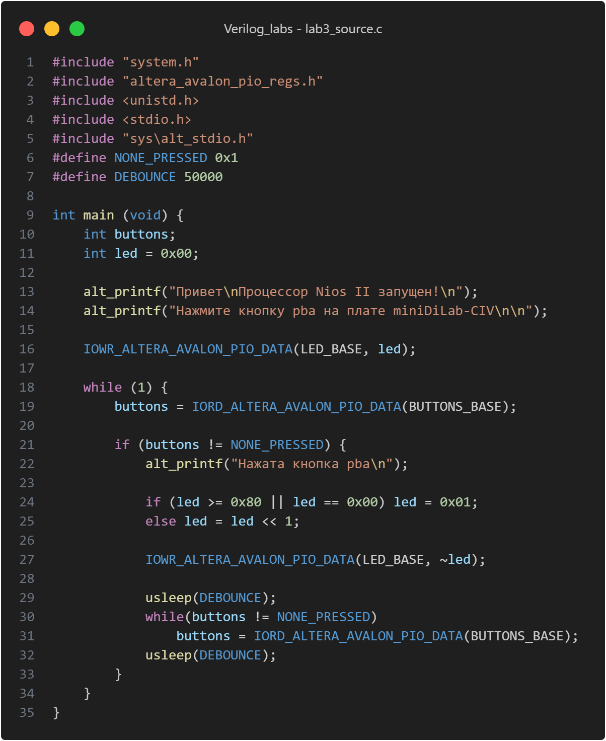


Рис. 2.22. Компиляция проекта с новыми настройками BSP.

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:



Выполним компиляцию и получим следующий результат:

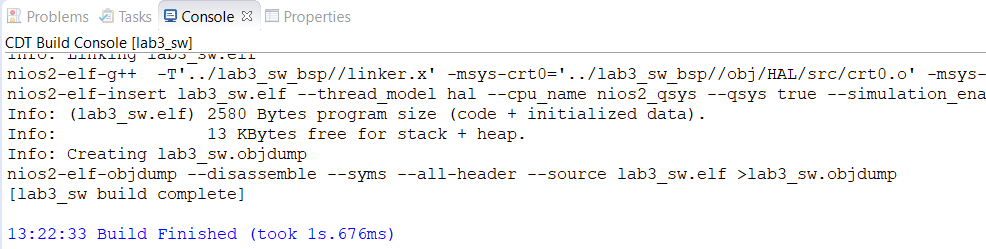


Рис. 2.23. Результат компиляции.

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.

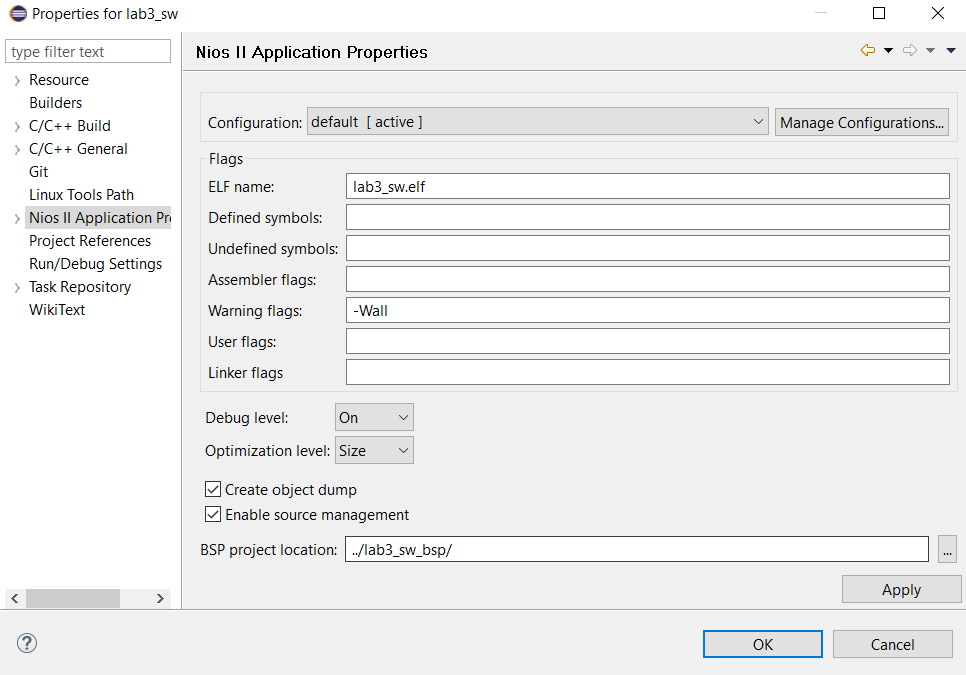


Рис. 2.25. Optimization level: size. У lab3\_sw.

Повторим компиляцию:

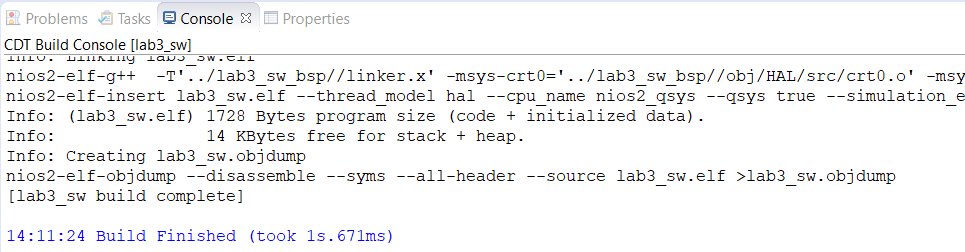


Рис. 2.26. Компиляция после выставления настроек оптимизации.

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

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

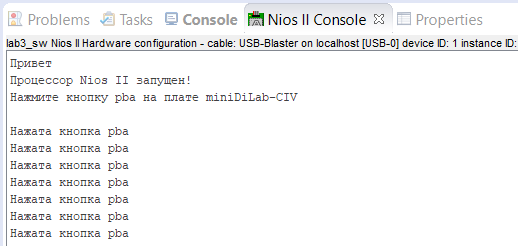


Рис. 2.27. Результат запуска программы и нескольких переключений pba.

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

# Проект 2:

## Структура проекта:

Структура разрабатываемого проекта приведена ниже:

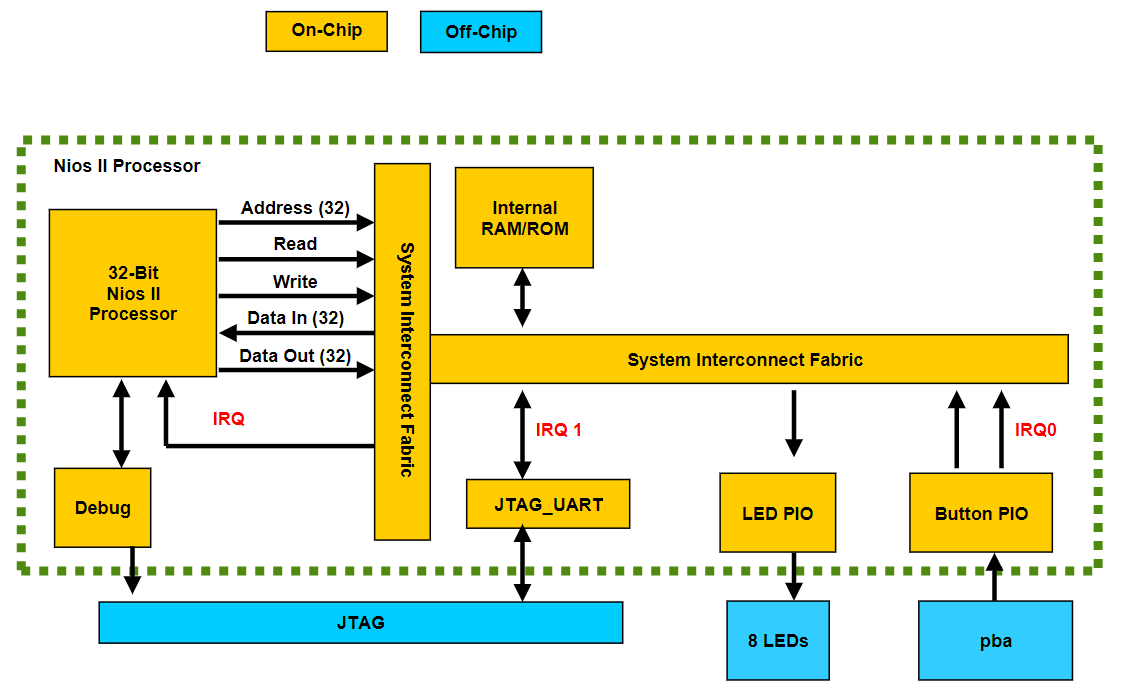


Рис. 3.1. Структура разрабатываемого проекта.

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).

## Решение:

Выполним создание проекта в QP со следующими настройками:

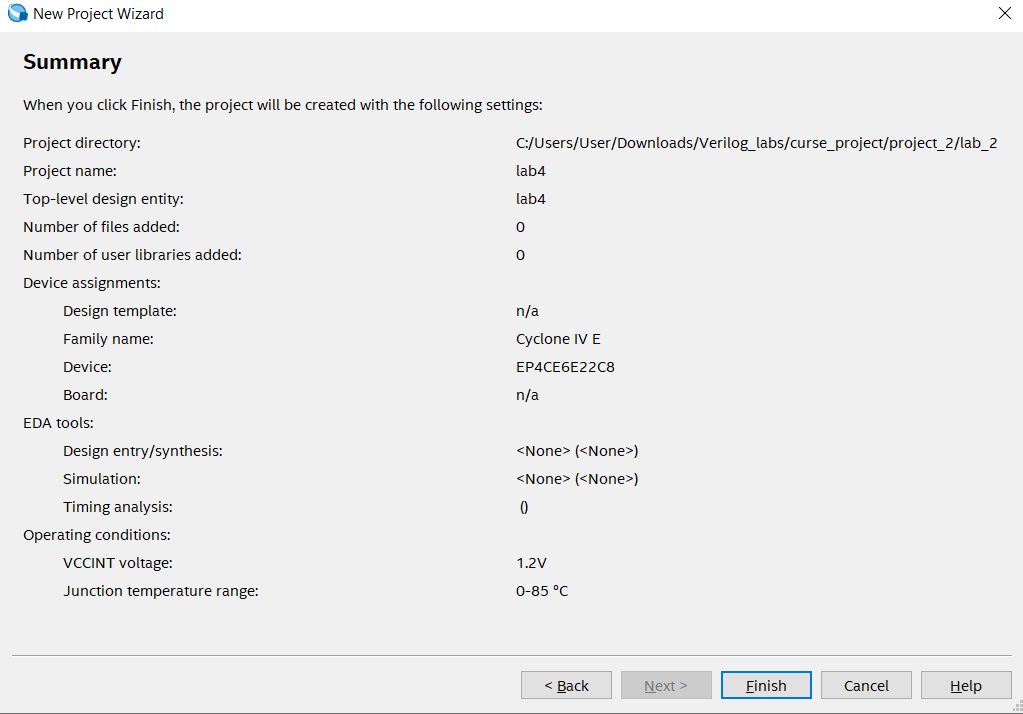


Рис. 3.2. Настройки для QP.

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

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

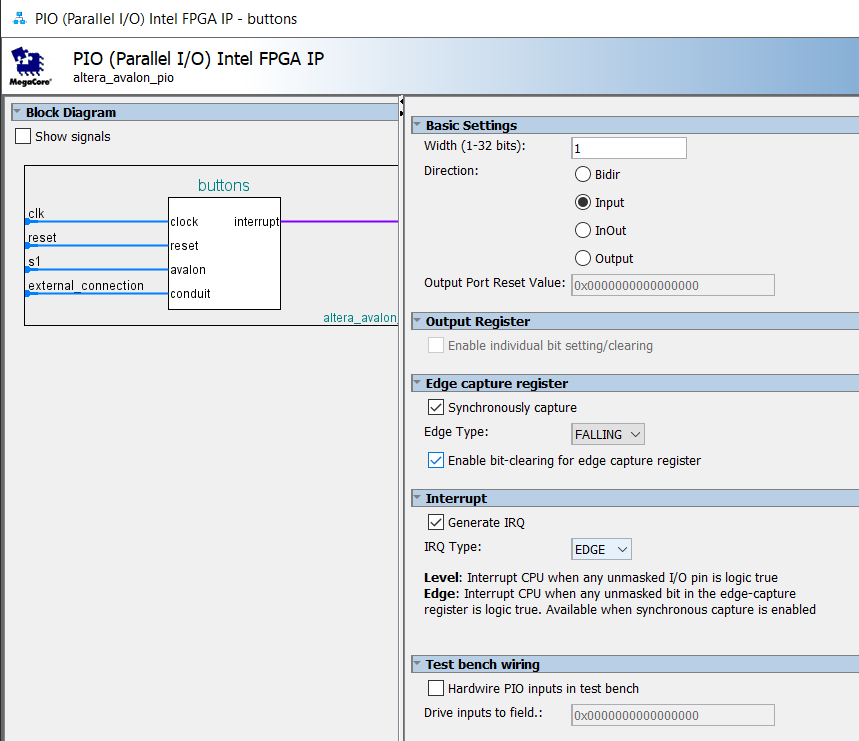


Рис. 3.3. Новые настройки модуля PIO.

Также необходимо подключить прерывания к процессору:

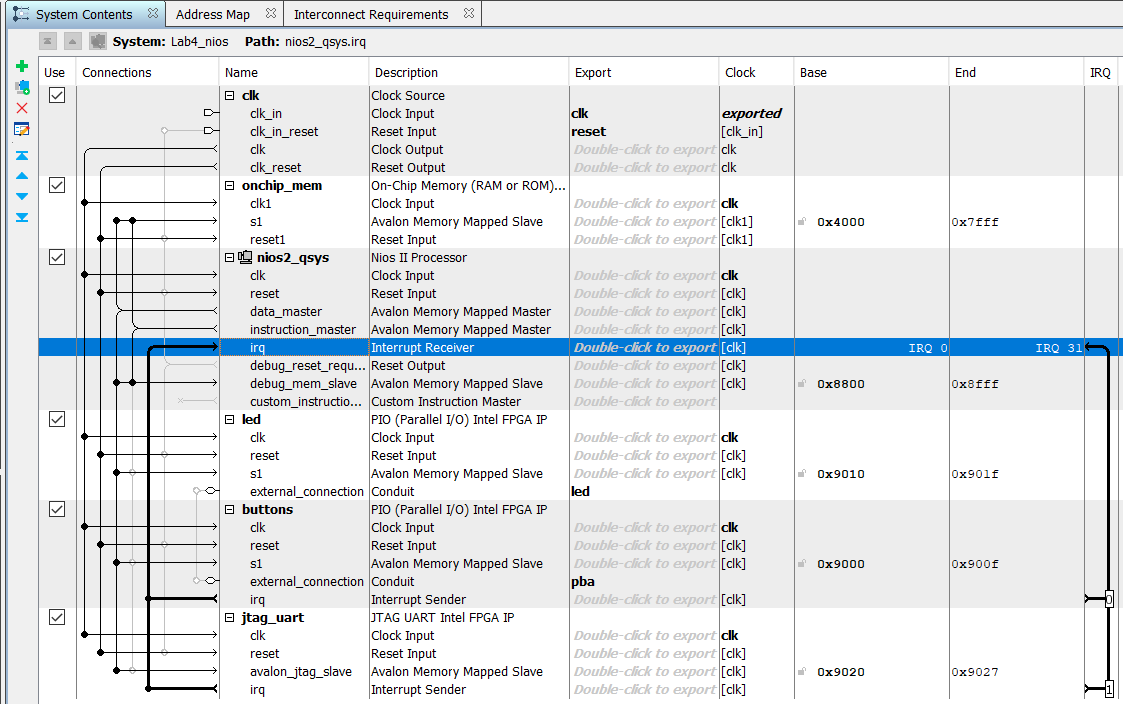


Рис. 3.4. Подключение прерываний к процессору.

Выполним генерацию модуля и создадим файл верхнего уровня:



Выполним компиляцию и выполним назначения для входов и выходов:

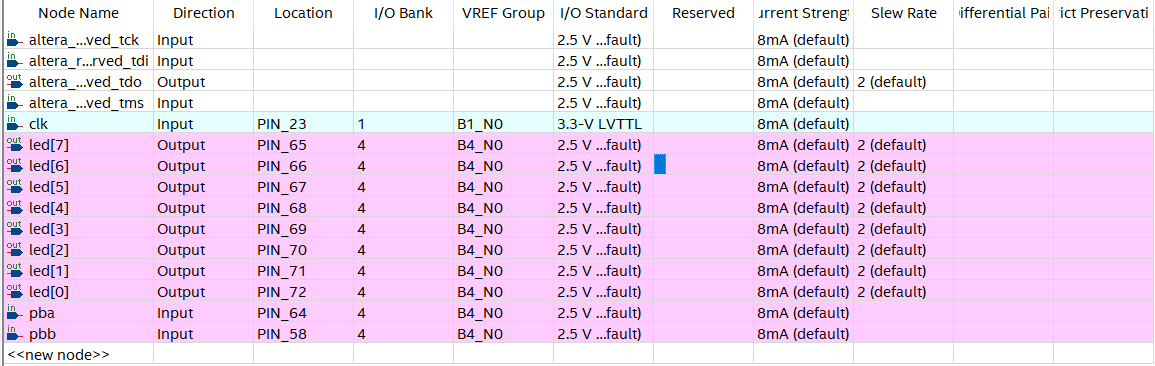


Рис. 3.5. Назначения в Pin Planner.

Также добавим файл для временных характеристик:



Выполним полную компиляцию и проверим, выполняются ли временные характеристики:

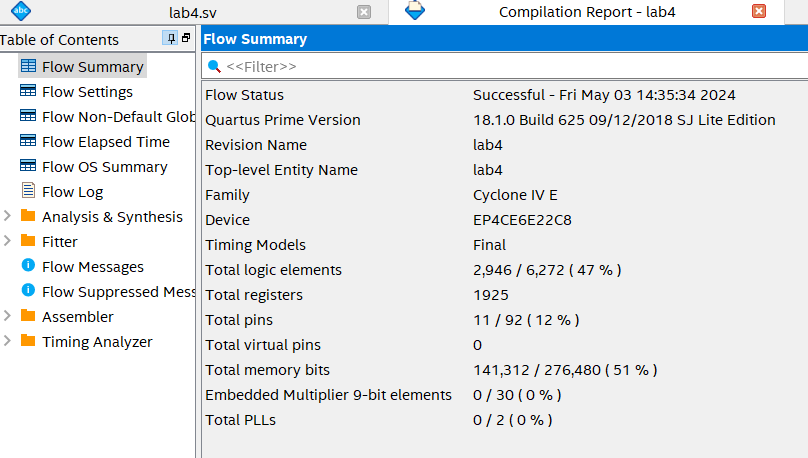
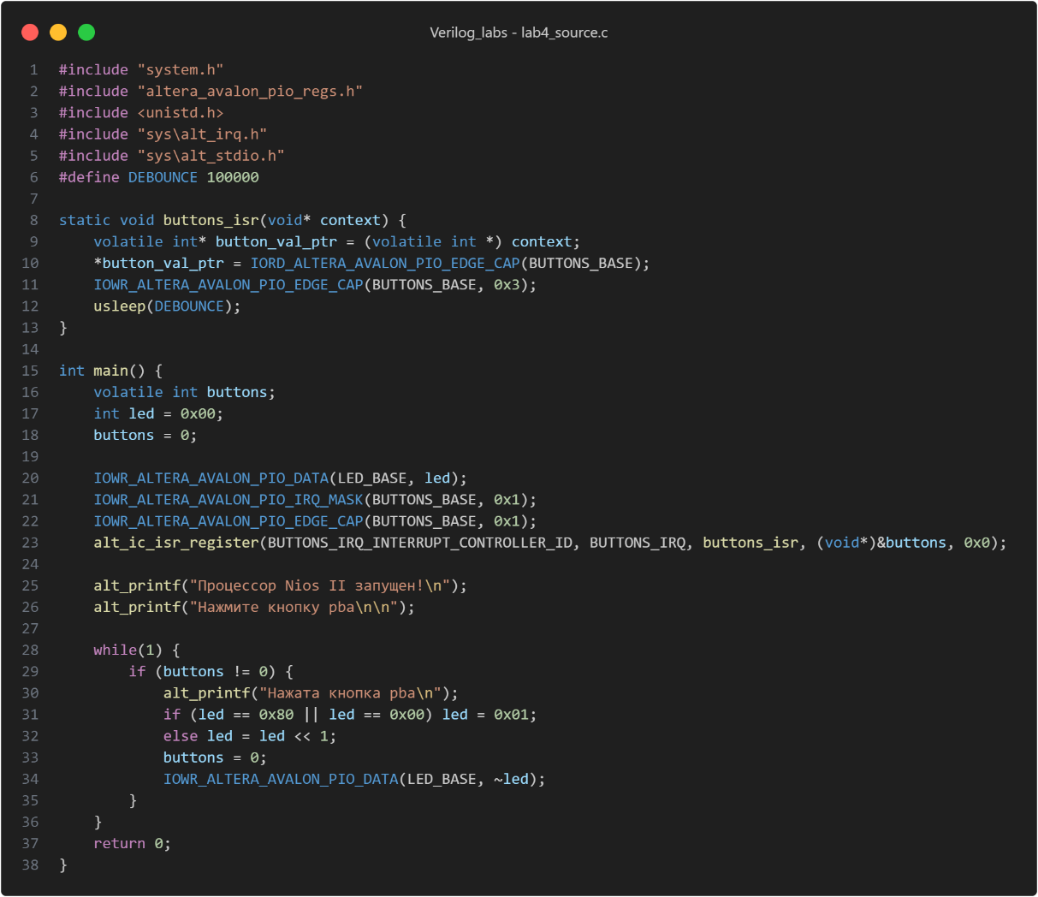


Рис. 3.6. Результат компиляции.

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:



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:

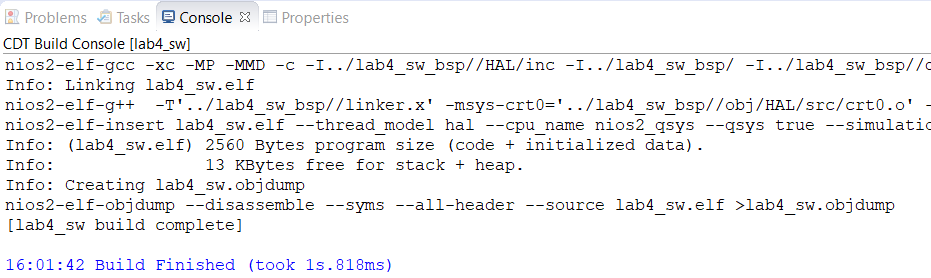


Рис. 3.7. Результат компиляции.

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

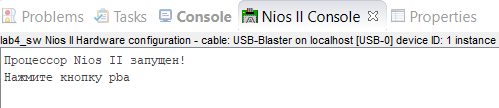


Рис. 3.8. Консоль проекта после записи программы.

Нажмем кнопку несколько раз, будет следующий результат:

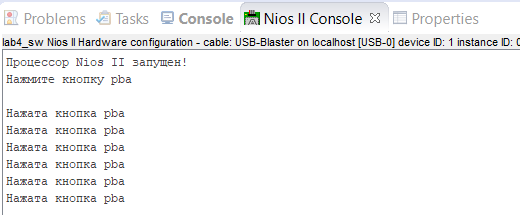


Рис. 3.9. Консоль после нескольких запусков программы.

Как мы видим программа работает корректно в соответствии с ожиданиями.

# Проект 3:

## 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; 0x0с; 0x30; 0xс0 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.

## Решение:

Выполним создание проекта в QP со следующими настройками:

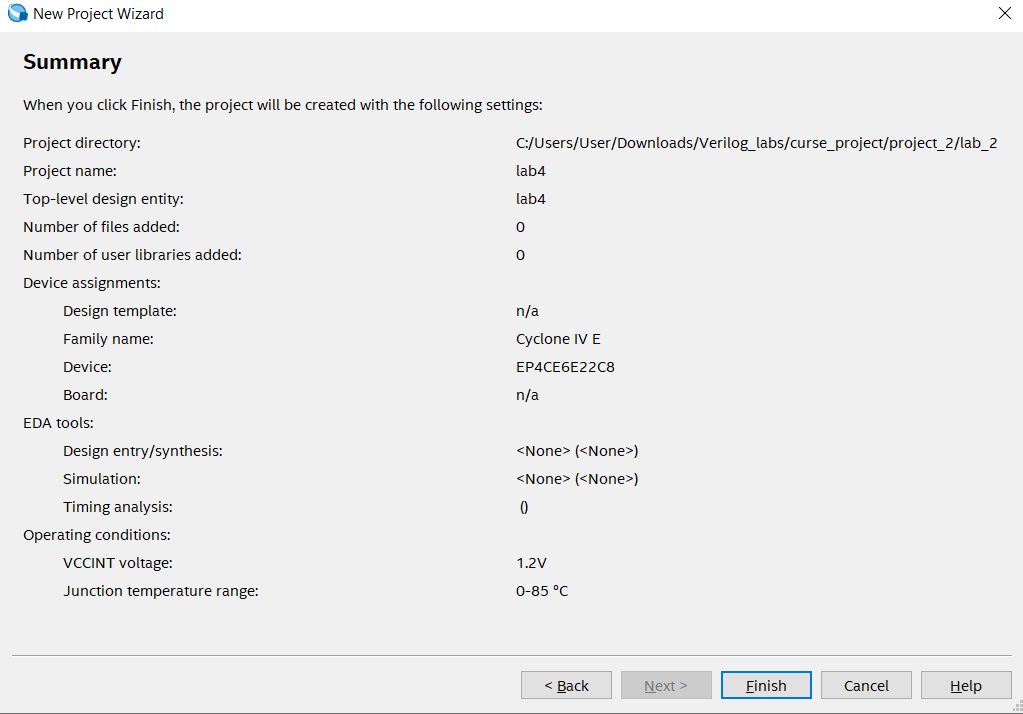


Рис. 4.2. Настройки для QP.

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:

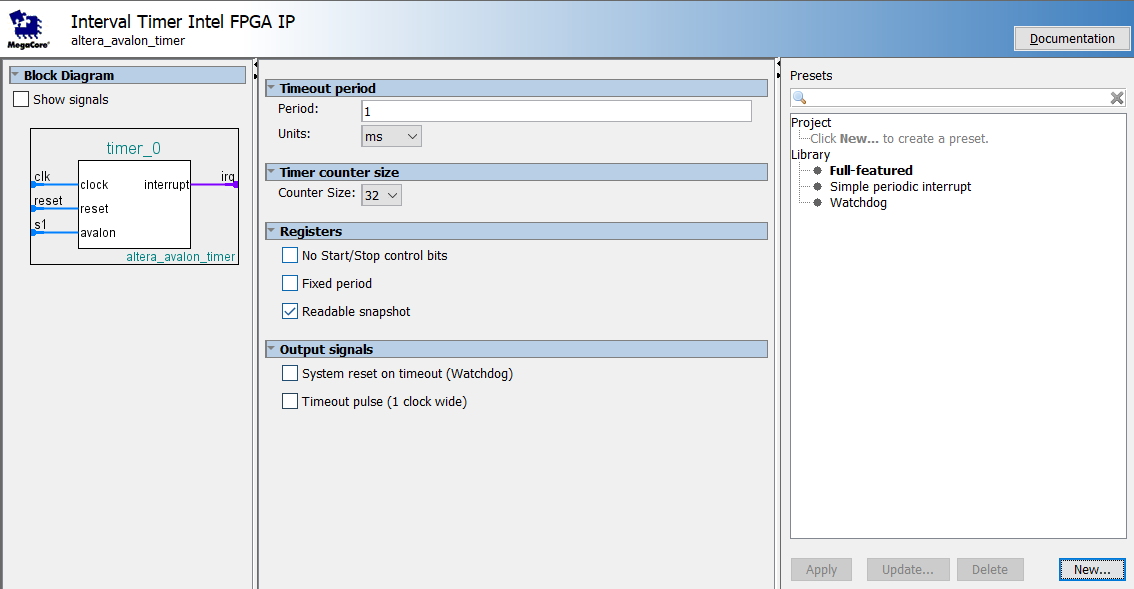


Рис. 4.3. Настройки компонента timer.

Также необходимо подключить таймер к процессору:

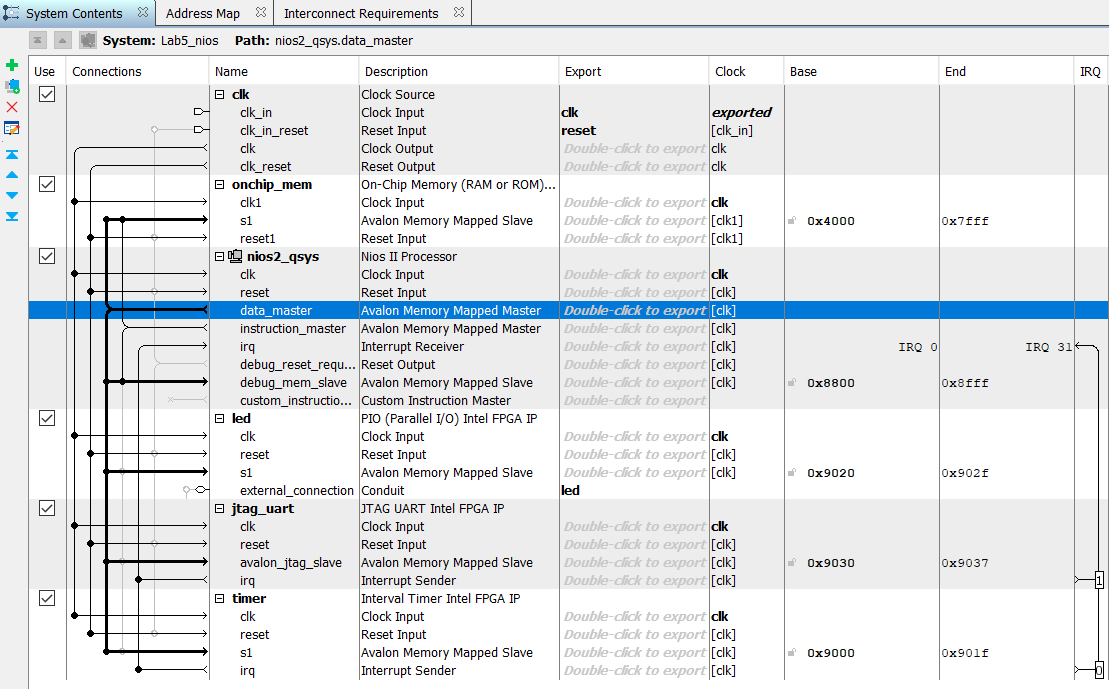
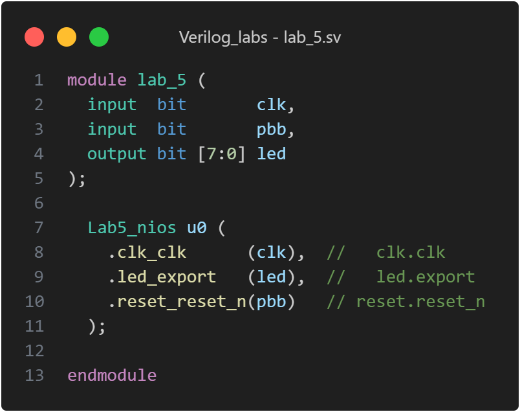


Рис. 4.4. Подключение таймера к процессору.

Выполним генерацию модуля и создадим файл верхнего уровня:



Выполним компиляцию и выполним назначения для входов и выходов:

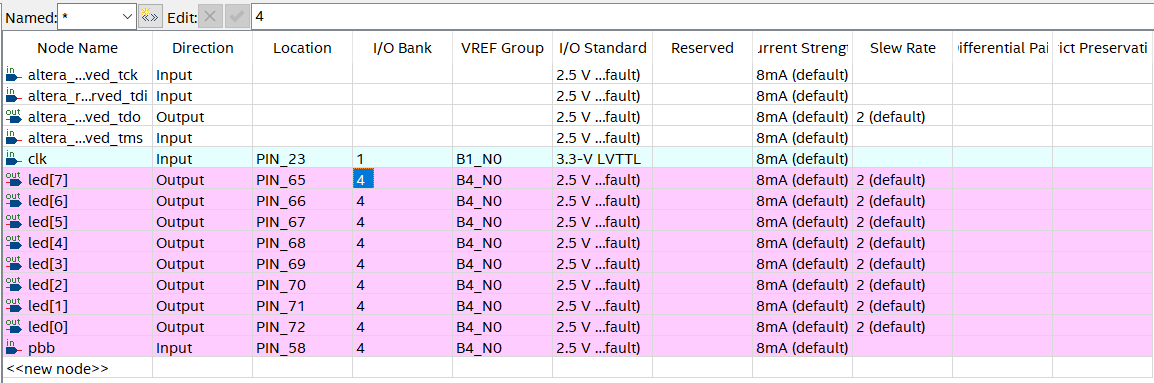


Рис. 4.5. Назначения в Pin Planner.

Также добавим файл для временных характеристик:



Выполним полную компиляцию и проверим, выполняются ли временные характеристики:

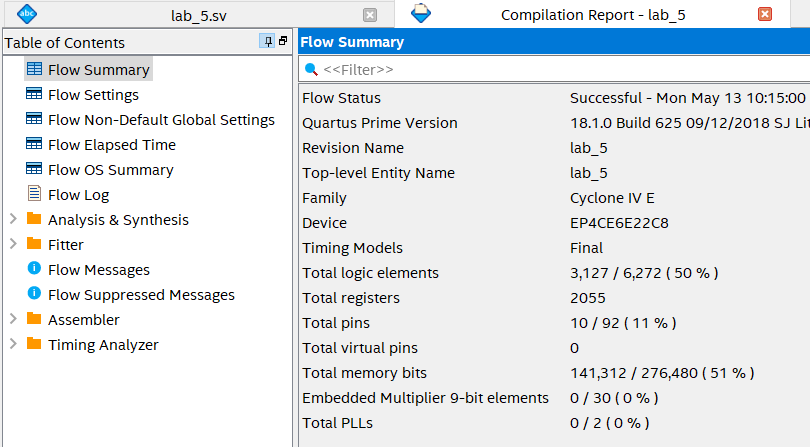
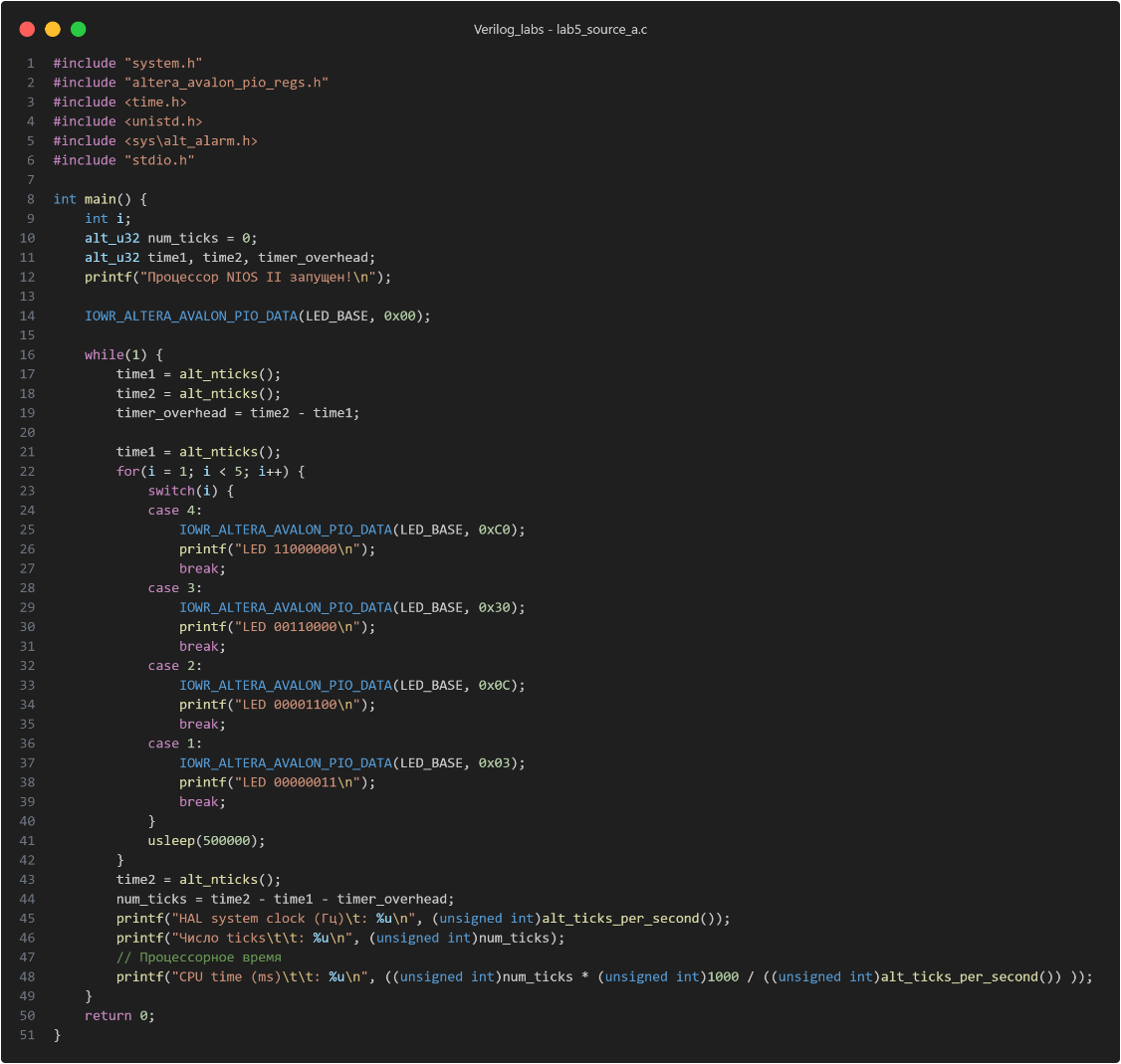


Рис. 4.6. Результат компиляции.

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:



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:

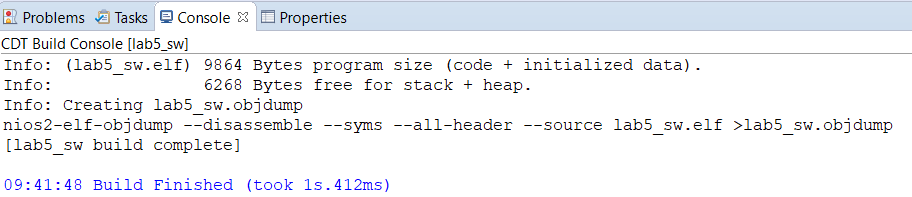


Рис. 4.7. Результат компиляции.

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

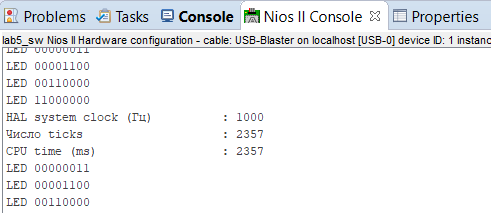


Рис. 4.8. Результат запуска программы на плате.

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:



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:

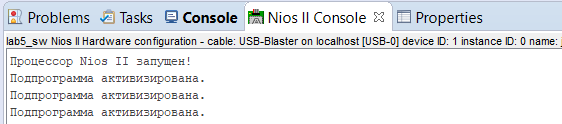


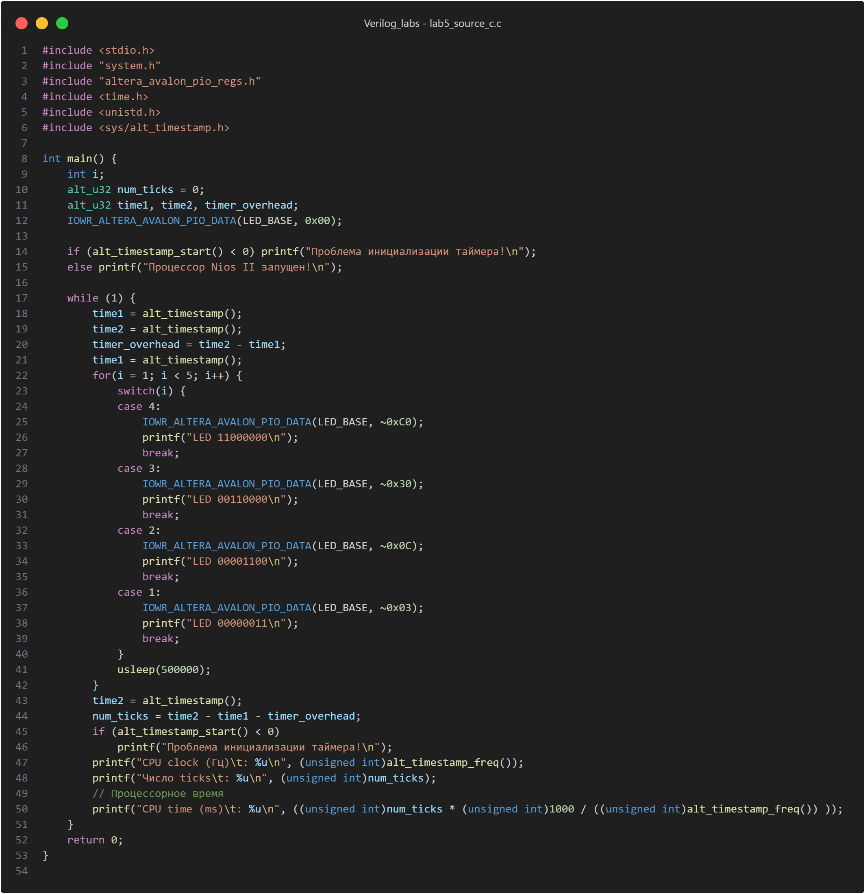
Рис. 4.9. Результат запуска программы на плате.

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:



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

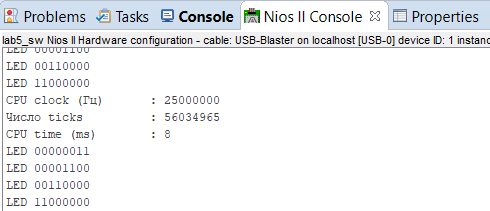


Рис. 4.10. Результат запуска программы на плате.

После запуска программы также начали переключаться и светодиоды.

Программа была успешно продемонстрирована преподавателю практики.

# Проект 3:

## Структура проекта:

Структура разрабатываемого проекта приведена ниже:

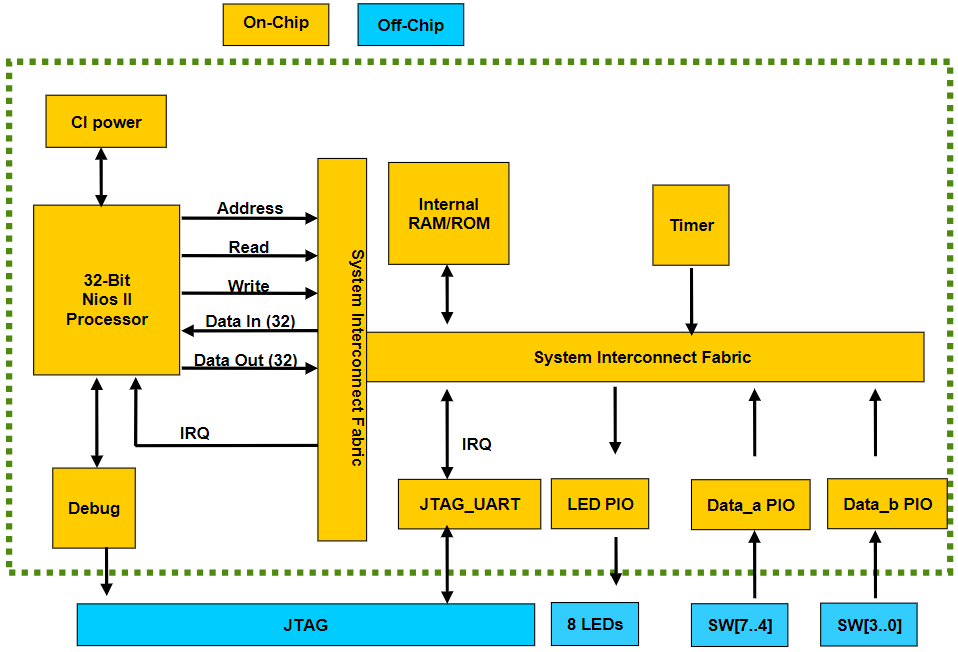


Рис. 5.1. Структура разрабатываемого проекта.

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.

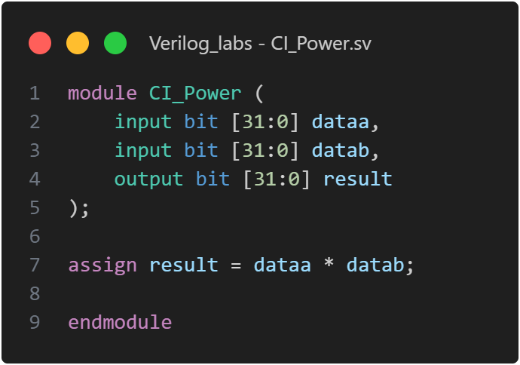
Этап 2.

* 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.

## Решение:

Выполним создание проекта в QP с стандартными настройками.

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



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

Создадим новый компонент, добавим CI\_Power в PD:

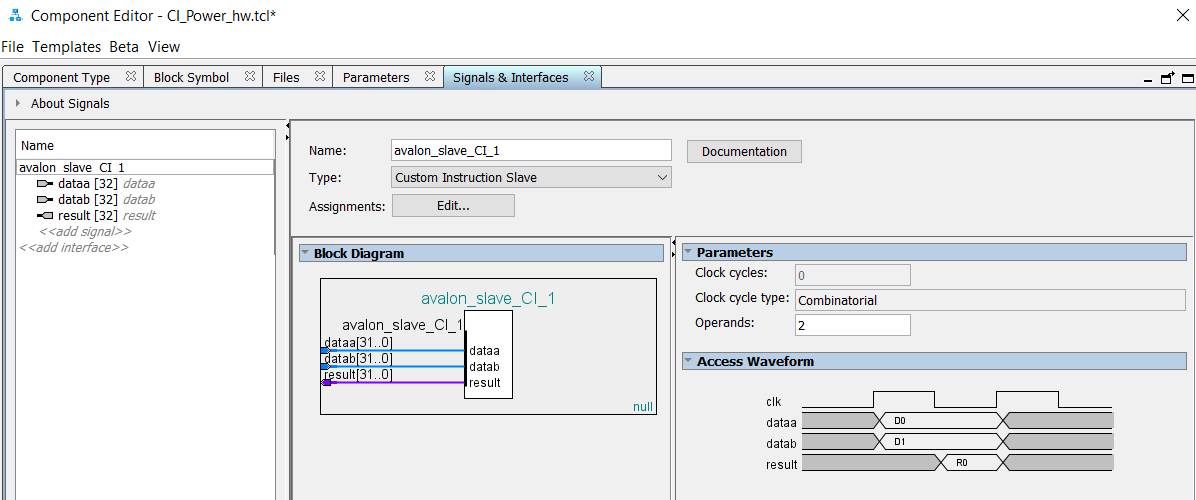


Рис. 5.2. Добавление модуля Custom Instruction.

Его блок будет выглядеть следующим образом:

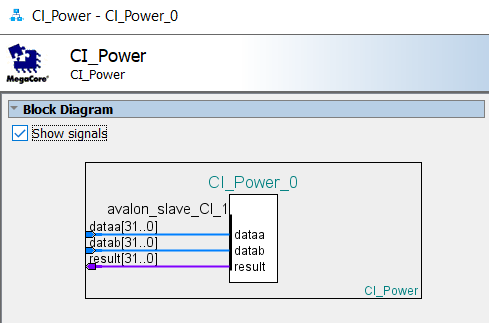


Рис. 5.3. Блок модуля Custom Instruction.

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

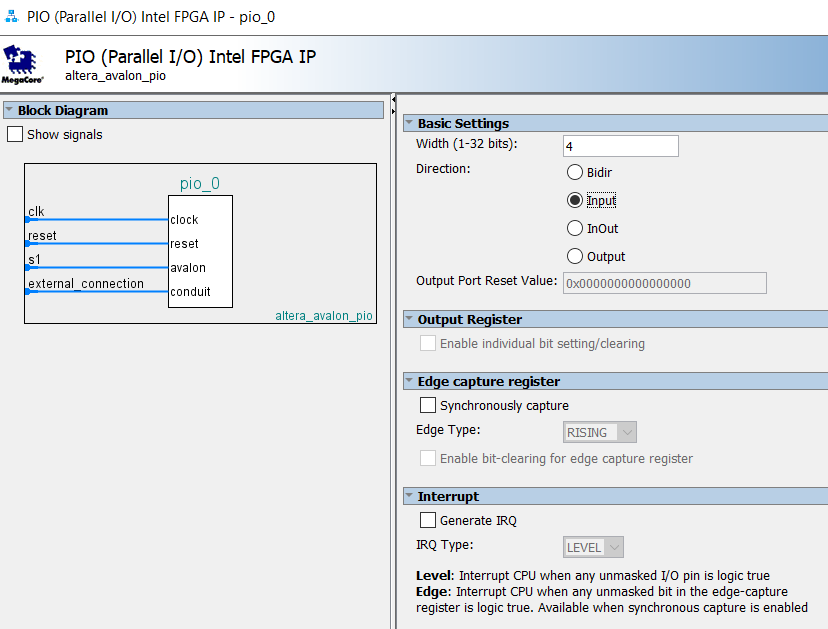


Рис. 5.4. Настройки модуля ввода 1.

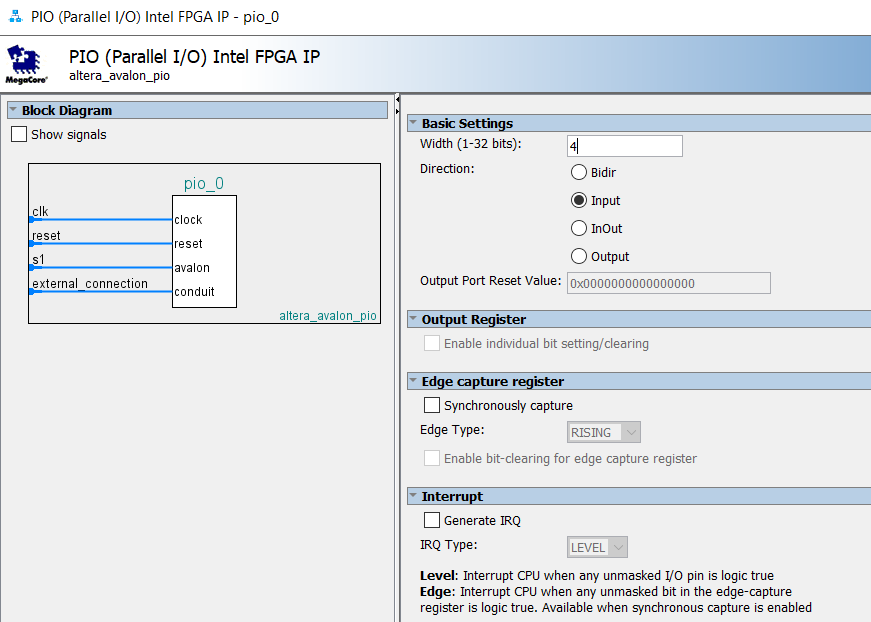


Рис. 5.5. Настройки модуля ввода 2.

Выполним все подключения и получим следующую схему в PD:

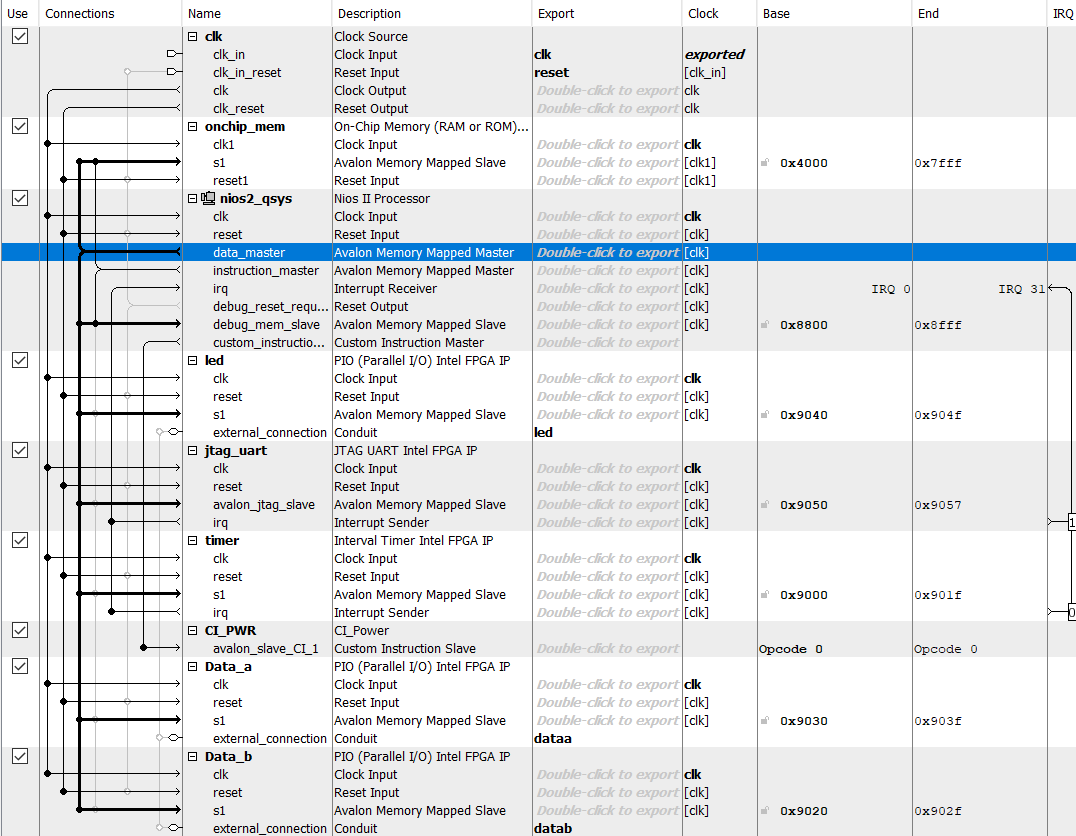


Рис. 5.6. Подключения в PD.

Schem разработанного устройства приведена ниже:

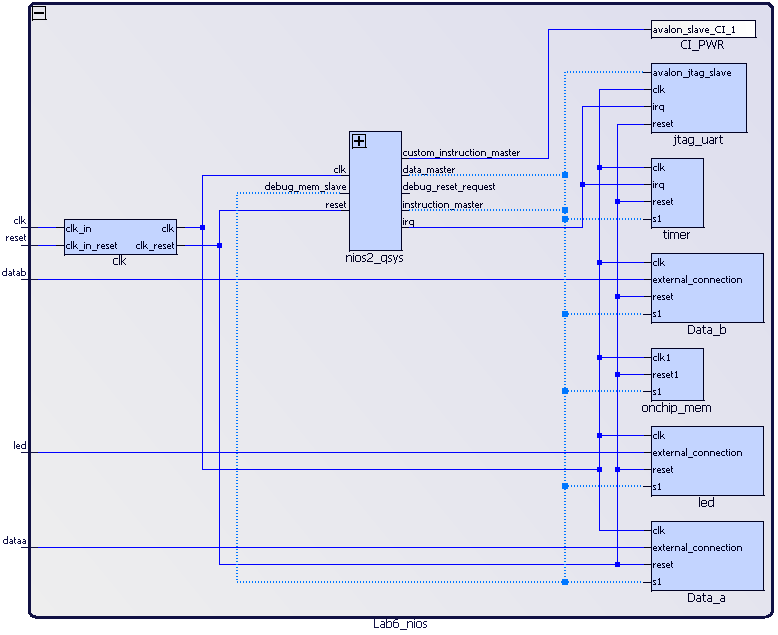


Рис. 5.7. Схема разработанного устройства.

Справа сверху отображает созданная нами инструкция в белом прямоугольнике.

Выполним генерацию и создадим файл верхнего уровня:

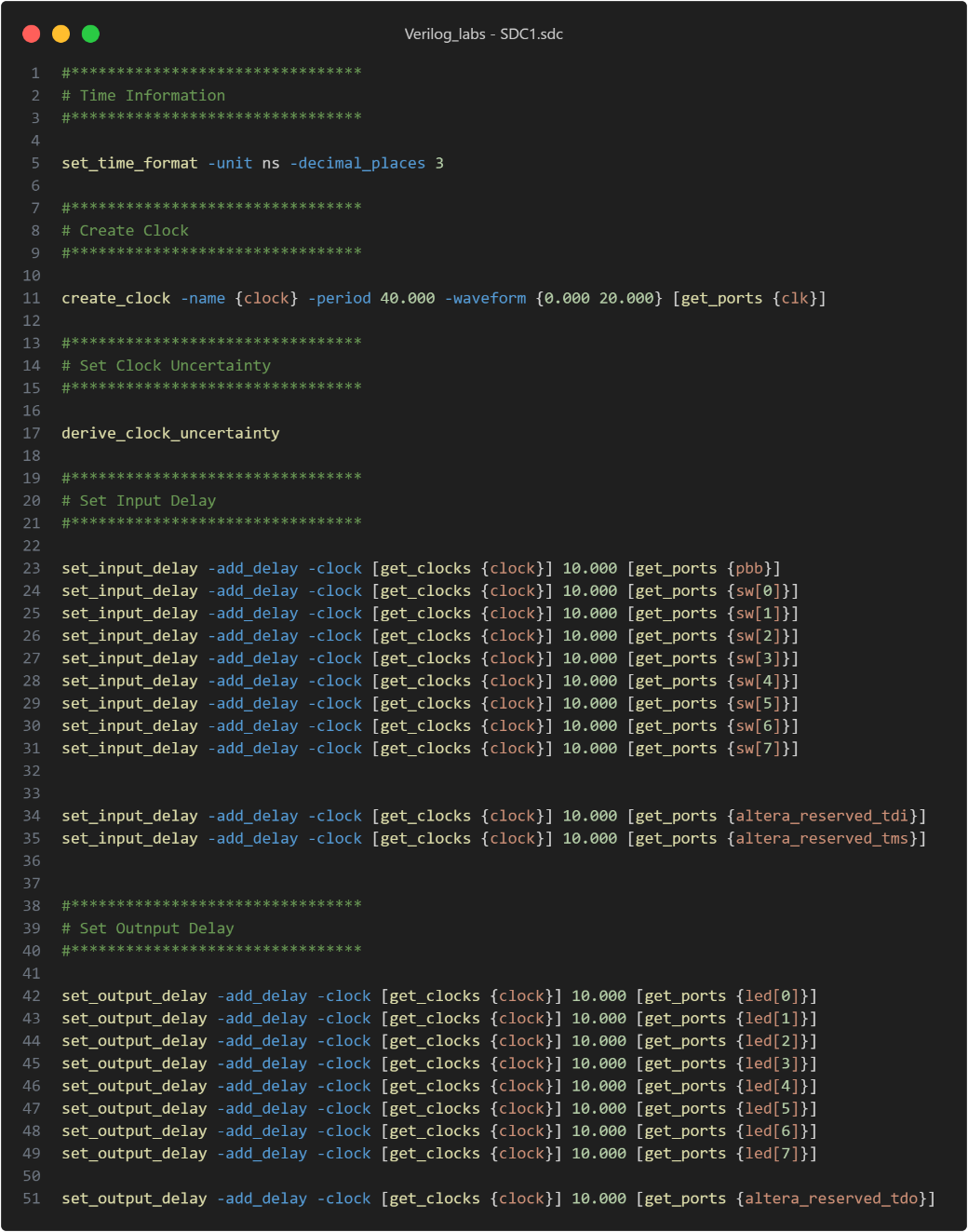


Выполним компиляцию и выполним назначения для входов и выходов:



Рис. 5.8. Назначения в Pin Planner.

Также добавим файл для временных характеристик:



Выполним полную компиляцию и проверим, выполняются ли временные характеристики:

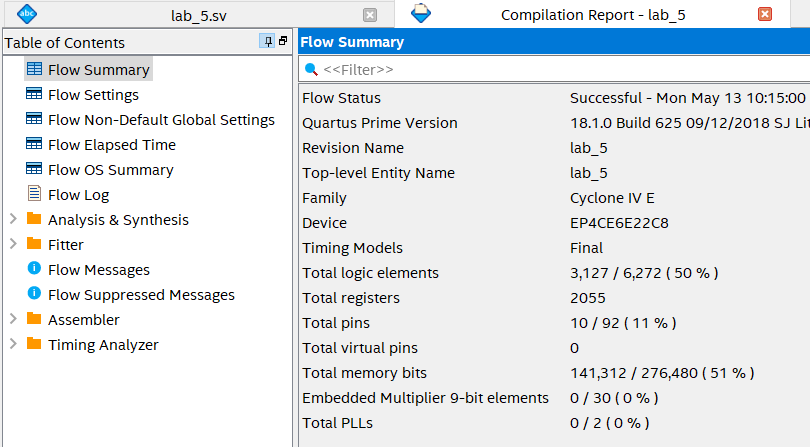
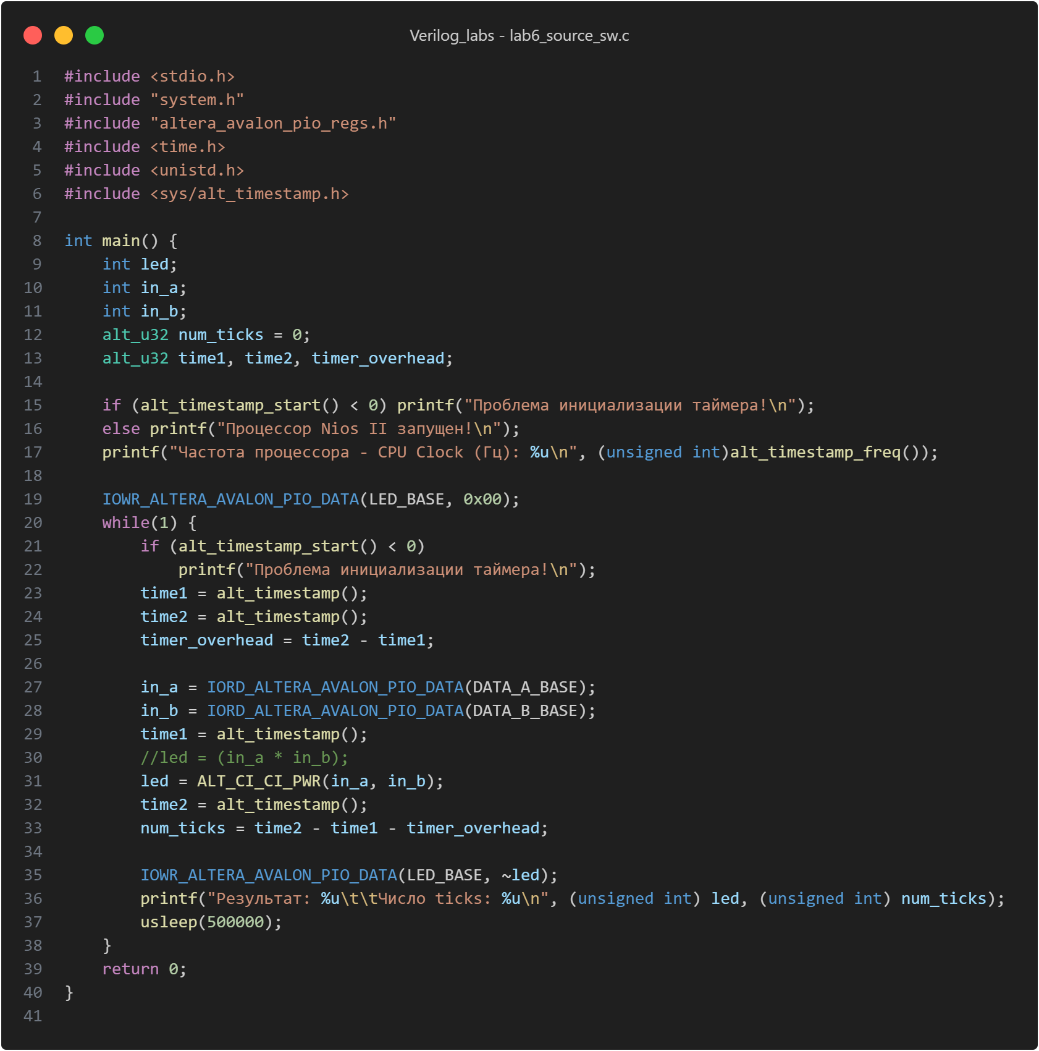


Рис. 5.9. Результат компиляции.

Как видим все временные характеристики выполняются.

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



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:

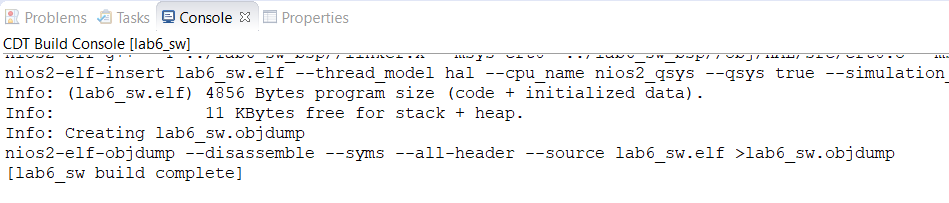


Рис. 5.10. Результат компиляции.

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

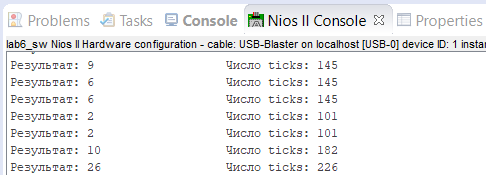


Рис. 5.11. Результат запуска программы на процессоре.

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:

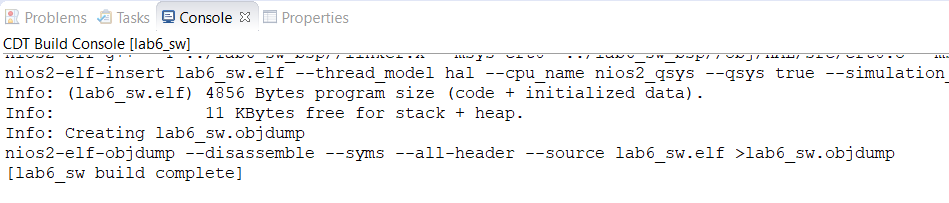


Рис. 5.12. Результат компиляции.

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

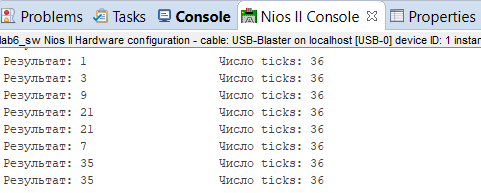
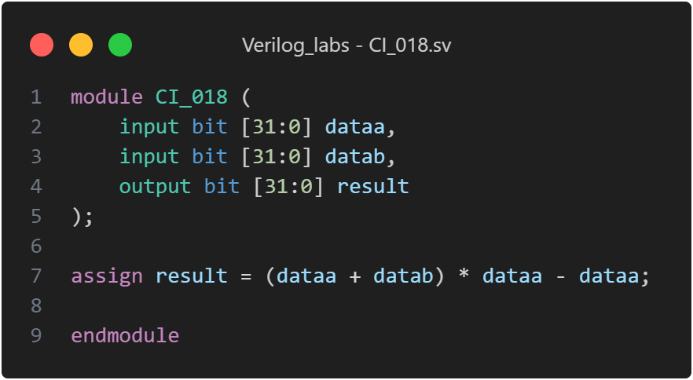


Рис. 5.13. Результат запуска программы на процессоре.

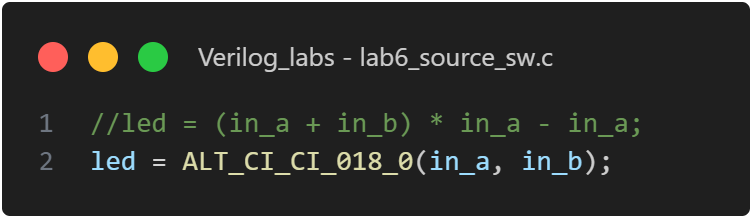
Как видим время выполнения не изменяется и является очень маленьким.

Теперь вместо умножения необходимо реализовать функцию в соответствии с вариантом, а именно: . Для того, чтоб создать Custom Instruction создадим следующий файл:



Далее все действия аналогичны и в отчете будут опущены.

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



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

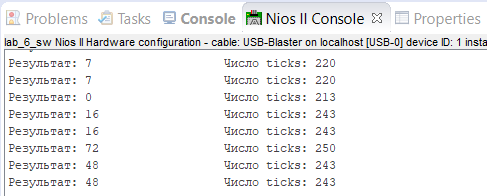


Рис. 5.14. Результат запуска на процессоре.

Однако если повторно скомпилировать, но уже Custom Instruction:

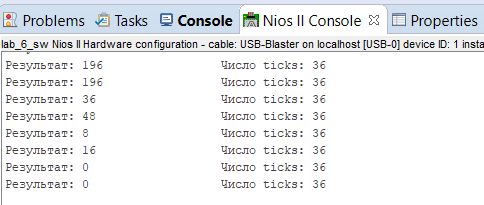


Рис. 5.15. Результат запуска на процессоре.

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.

# 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.