

**РАЗРАБОТКА ПРОЦЕССОРНОЙ СИСТЕМЫ
НА БАЗЕ СОФТ-ПРОЦЕССОРА MICROBLAZE
В СРЕДЕ XILINX VIVADO IDE/HLX**

Автор:
KeisN13

Рецензенты:
Intekus
Nagornayaod

Материал подготовлен при поддержке:

КТЦ «Инлайн груп» - дистрибутор фирмы Xilinx (www.plis.ru)

ATP Center Xilinx – Сертифицированный тренинг центр Xilinx (www.plis2.ru)

Оглавление

Аннотация	3
Введение.....	3
HW/SW части процессорной системы	4
Создание нового проекта	5
Создание HW-части	12
Разработка SW части	43
Заключение	61
Библиографический список	62
Список тренингов.....	62

Аннотация

В этой статье описаны основные этапы разработки процессорной системы на базе софт-процессора MicroBlaze [1, 2] в среде Xilinx Vivado [3]. Будут рассмотрены только основные моменты и шаги, которые позволят Вам быстро собрать процессорную систему из ресурсов ПЛИС/FPGA, а также получить общее представление о ходе её построения и подключения периферии.

Введение

Разработка систем на FPGA давно перестала ограничиваться простым написанием кода на языках описания аппаратуры (HDL); и по мере возрастания количества логических ресурсов и сложности проектов, подходы к проектированию систем на FPGA многократно пересматривались. Одним из витков развития стало внедрение в проекты софт-процессоров – по сути обычных микропроцессоров, но собранных на ресурсах FPGA. Такой подход даёт разработчикам гибкость системы и ещё больше объединяет области сотрудничества HW (hardware – аппаратных) и SW (software – программных) инженеров. К сожалению, несмотря на относительную несложность разработки софт-процессорных систем, многие пытаясь «подняться» эту тему, сталкиваются с трудностями освоения, потому что не знают где найти нужную информацию, а если и находят, то не знают с чего начать.

Цель статьи – дать общее представление об этапах сборки процессорной системы на базе софт-процессора Microblaze, используя среду Xilinx Vivado.

К сожалению, в рамках одной статьи сложно описать все многообразие процесса построения софт-процессорных систем на FPGA и детально изложить все сопровождающие его «тонкие» моменты, но с чего-то начать необходимо. В последующих статьях мы рассмотрим подключение Ethernet, различных контроллеров памяти, внешних интерфейсов, разберём работу компонентов,

связывающих MicroBlaze с периферией и многое другое. Но это будет чуточку позже, а для начала...

Примечание: надо сказать, что при создании soft-процессорной системы, точнее для её отладки Вам всё же придётся иметь какую-нибудь макетную плату с FPGA Xilinx 7-го семейства [4] или же уметь пользоваться QEMU эмулятором. Простое моделирование через симулятор XSIM здесь не есть решение проблемы. Моделирование запустится, но сколько понадобится времени чтобы посмотреть, что светодиод моргает – я сказать не могу; возможно – дни и недели машинного времени. Поэтому крайне рекомендую обзавестись недорогой макетной платой. экспериментировал на Arty Board от компании Digilent [5], именно на ней и будет проходить текущее и дальнейшее «обучение». Рекомендую читателям обзавестись ею, поскольку кристалл, установленный на ней, поддерживается бесплатной (Web) версией Vivado, но при этом с платой поставляется полноценная лицензия. В России плата обошлась мне в 153 доллара, заказывал через компанию Регион-Вирта [6], поскольку они могут работать с физическими лицами. В интернет-магазинах (включая AliExpress) в то время плата стоила гораздо дороже. Возможно, сейчас ситуация изменилась – проверьте.

HW/SW части процессорной системы

Разработка любой процессорной системы, построенной на ресурсах FPGA, состоит из двух фундаментальных частей: сборки аппаратной платформы HW – hardware, и разработки исполняемой программы SW – software.

HW-часть разрабатывается в среде Xilinx Vivado в модуле IP Integrator [7] (Vivado IPI) и представляет собой создание собственно экземпляра (или нескольких – для многопроцессорной системы) ядра MicroBlaze, соединение его с необходимой периферией и распределение адресного пространства. Разработка кода для MicroBlaze выполняется в Xilinx SDK [8] на ассемблере или C/C++.

Процесс сборки HW-части в Vivado IPI во многом похож, на аналогичный в предшествующих средах Xilinx – ISE и PlanAhead (где для этого использовалась утилита XPS – Xilinx Platform Studio), но имеет ряд отличий от него. Сохранились общие принципы, различия же встречаются, по большей части, в представлении системы. В одной из последующих статей мы разберём процесс разработки в XPS. Со стороны же программной (SW-части) никаких отличий нет: для работы с программной частью также используется Xilinx SDK.

Разработка аппаратной платформы начинается с запуска Vivado и создания проекта.

Создание нового проекта

Воспользуемся одним из способов открытия стартового окна Vivado, используя TCL Shell (Пуск – Xilinx Design Tools – Vivado 201x.x – Vivado 201x.x Tcl Shell (см. рис. 1)).

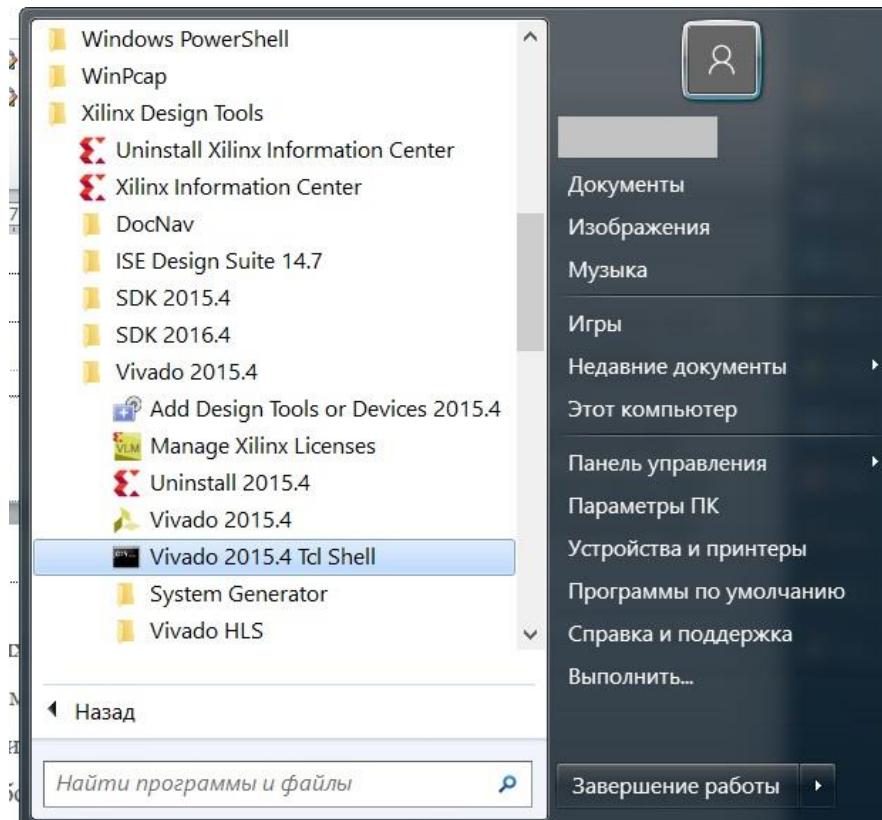


Рисунок 1 Расположение Tcl Shell в меню Пуск

В этом проекте я использую Vivado 2015.4, потому что в ней нет некоторых ошибок, присутствующих в последующих версиях. Для себя считаю её оптимальным вариантом по занимаемому на жёстком диске объёму, стабильности, функциональности и быстродействию. В более новых версиях этапы разработки HW-части либо полностью, либо почти полностью аналогичны или тольк в точь повторяют описанную здесь последовательность действий.

В открывшемся окне пишем команду `start_gui` (рис. 2). Нажимаем Enter и ждём окончания выполнения команды.

```
Vivado 2015.4 Tcl Shell - C:\Xilinx\Vivado\2015.4\bin\vivado.bat -mode tcl

***** Vivado v2015.4 (64-bit)
**** SW Build 1412921 on Wed Nov 18 09:43:45 MST 2015
**** IP Build 1412160 on Tue Nov 17 13:47:24 MST 2015
** Copyright 1986-2015 Xilinx, Inc. All Rights Reserved.

Vivado% start_gui
```

Рисунок 2 Tcl Shell и запуск Vivado с помощью команды `start_gui`

После завершения выполнения команды появится стандартное окно Vivado (рис. 3). В зависимости от версии интерфейс может несколько отличаться; описание и скриншоты в статье будут соответствовать версии 2015.4.

Приступаем к созданию нового проекта.

Нажимаем Create New project (рис. 3)

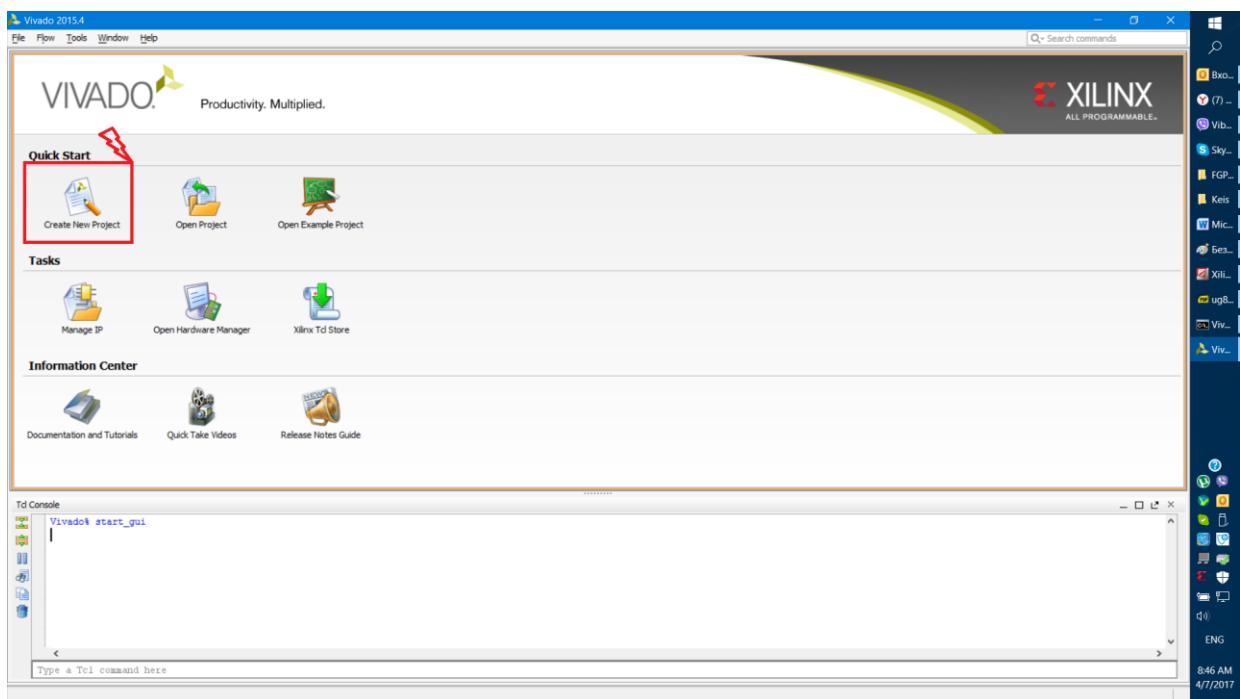


Рисунок 3 Окно Vivado 2015.4, красным выделена кнопка создания нового проекта

После нажатия кнопки появится мастер создания нового проекта, в котором нажимаем кнопку Next (рис. 4).

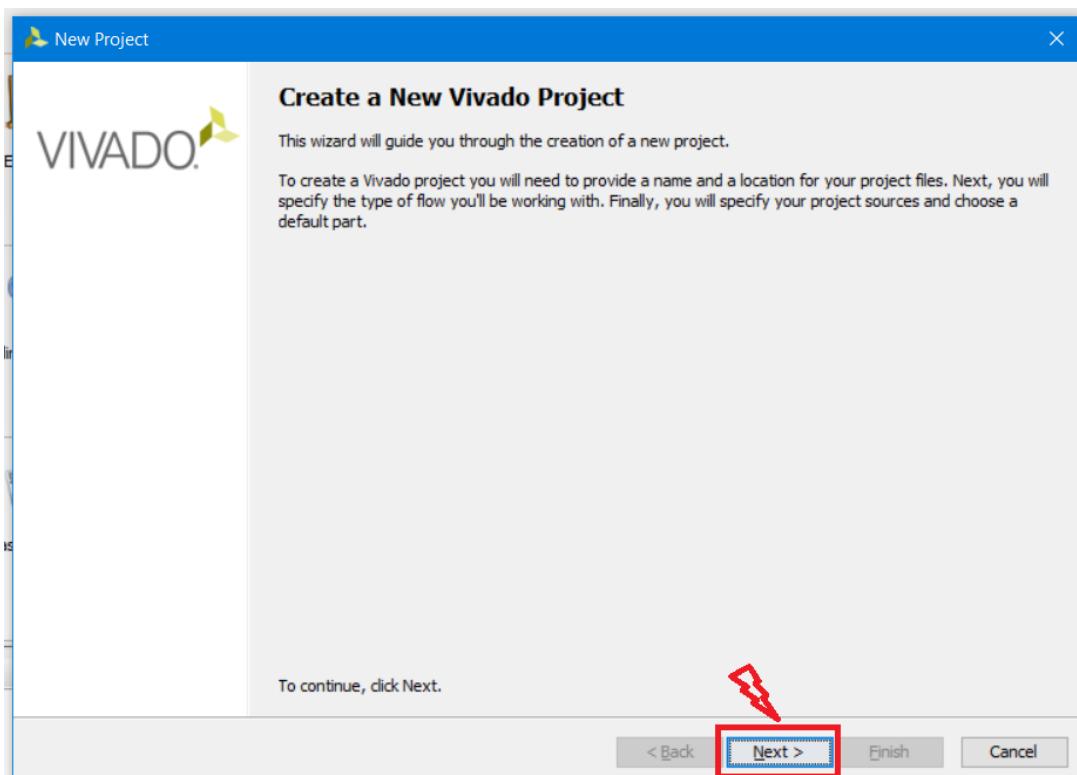


Рисунок 4 Окно мастера создания нового проекта

На странице Project Name (рис. 5) указываем:

1. Название проекта – в поле Project name пишем *Microblaze_Lesson_1*
2. Место расположения проекта – в поле Project location указываем ту папку, которую Вам будет удобно использовать. Не используйте в названии пути русские буквы, пробелы, тире и специальные символы:

C:/Projects/FGPA_Systems/Microblaze_Lesson_1

3. Снимаем галочку Create project subdirectory – иначе по указанному нами пути среда создаст подпапку с введённым нами именем проекта и поместит его файлы уже туда.
4. Нажимаем кнопку Next

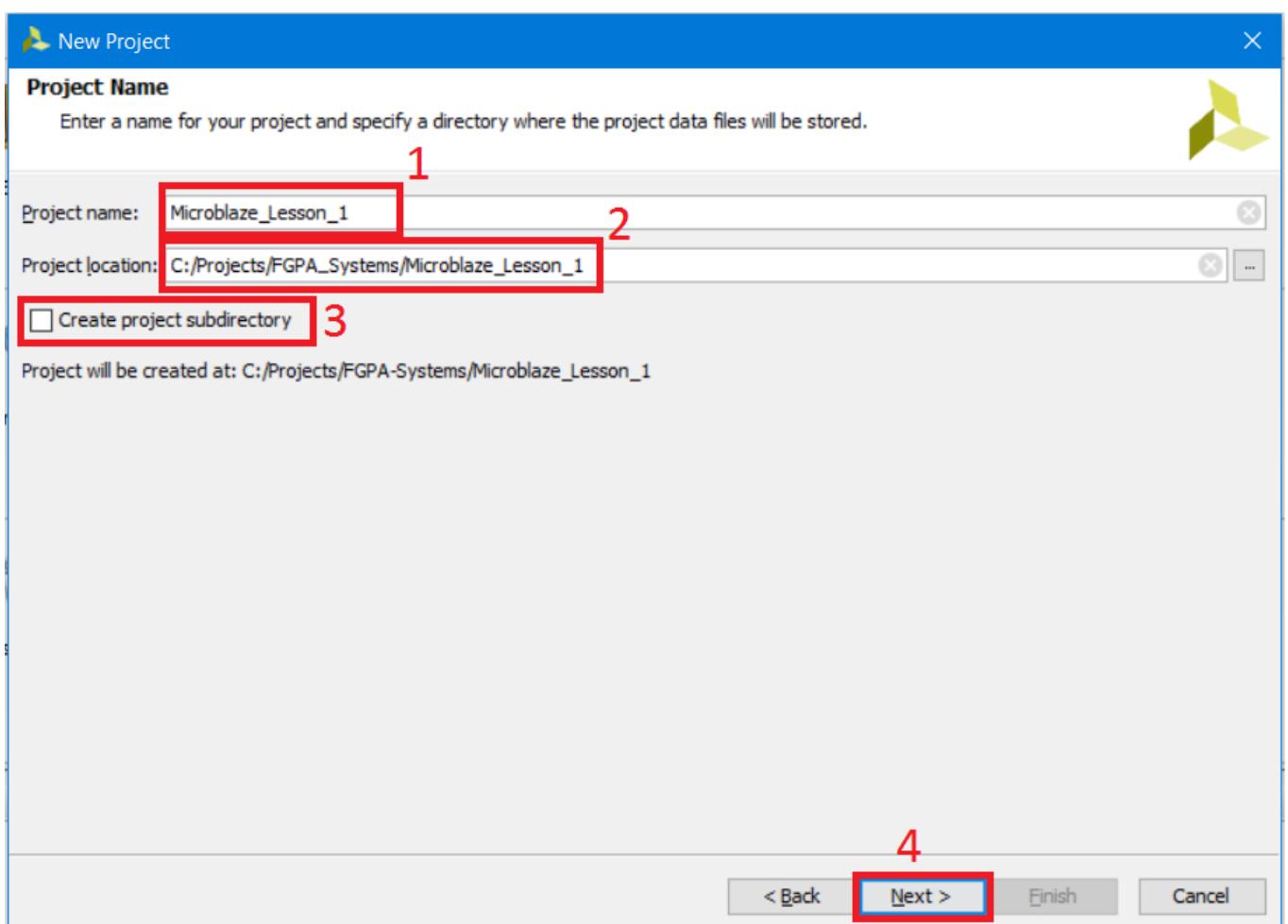


Рисунок 5 Окно мастера создания нового проекта: название проекта, путь к проекту

На странице Project Type (рис. 6):

1. Выбираем тип проекта RTL Project, потому что мы будем создавать с нуля обычновенный проект с добавлением IP-ядер и последующей имплементацией.
2. Ставим галочку Do not specify..., потому что при создании этого проекта никакие файлы мы в него добавлять не будем
3. Нажимаем кнопку Next

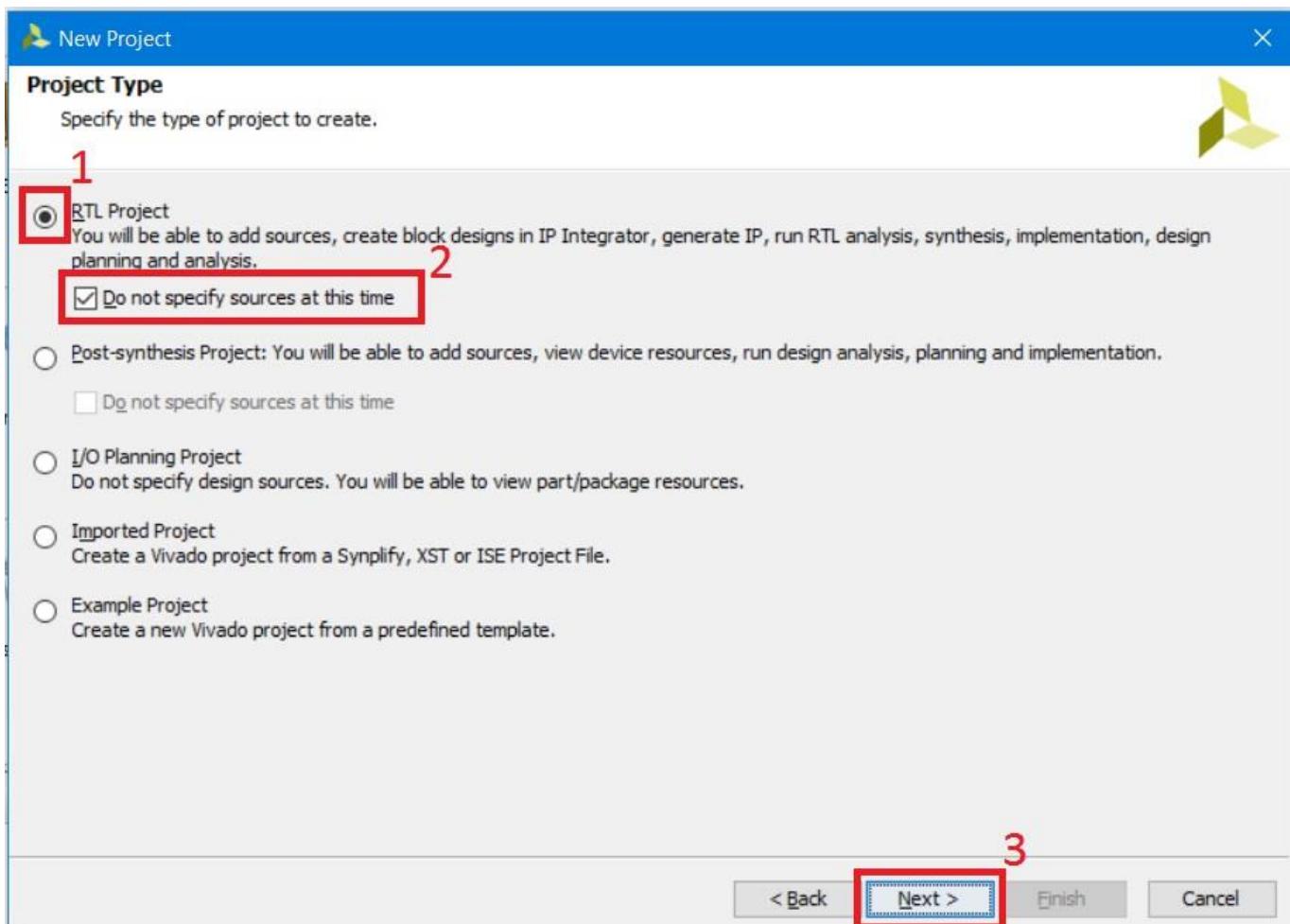


Рисунок 6 Окно мастера создания нового проекта: выбор типа создаваемого проекта

На странице Default Part выбираем (рис. 7):

1. Вкладку Parts с перечнем FPGA, доступных для текущей версии Vivado.
2. В поле поиска Search пишем название кристалла, стоящего на плате. На Arty Board установлена FPGA xc7a35ticsg324-1L
3. В списке выбираем нашу FPGA
4. Нажимаем кнопку Next

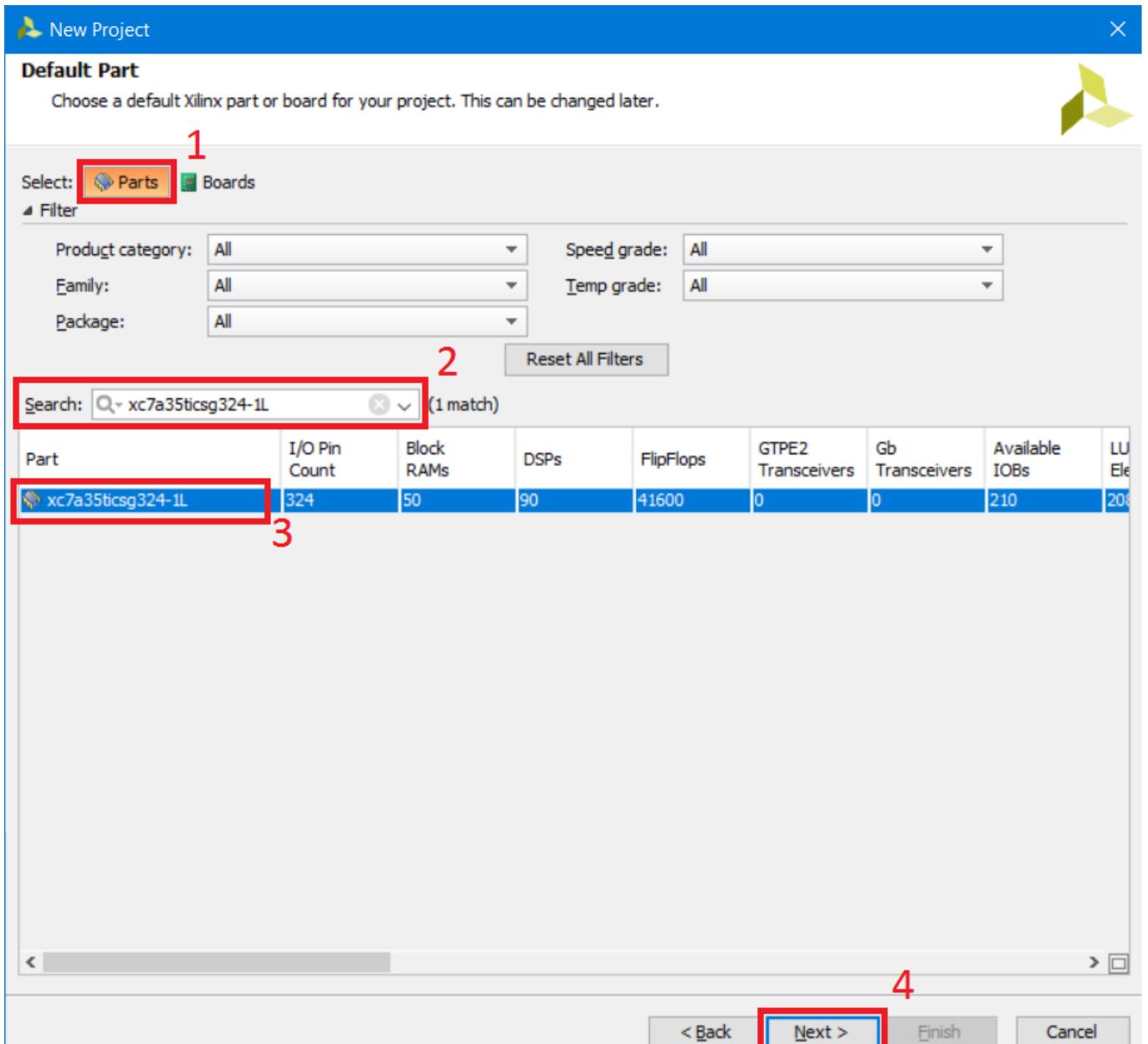


Рисунок 7 Окно мастера создания нового проекта: выбор FPGA

На этом создание нового проекта закончено. В окне New Project Summary (рис. 8) отображается суммарная информация о созданном проекте. Нажимаем кнопку Finish.

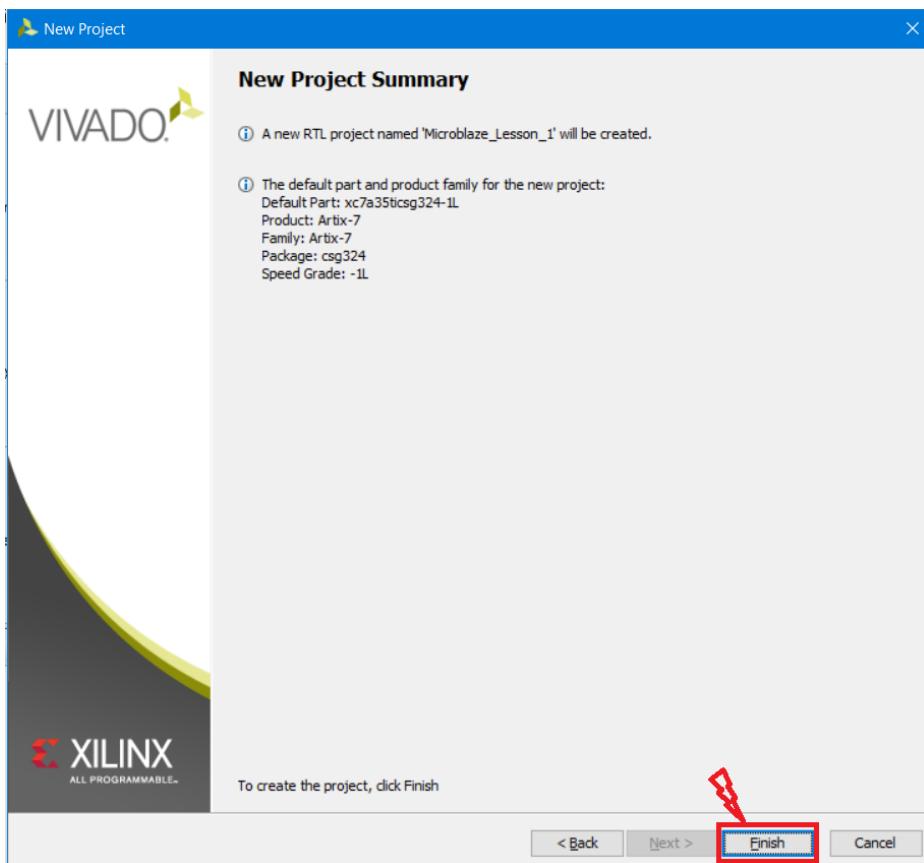


Рисунок 8 Окно мастера создания нового проекта: суммарные сведения о создаваемом проекте

Если Вы что-то захотите поменять в проекте, это можно сделать в любой момент во вкладке Tools – Project settings (рис. 9)

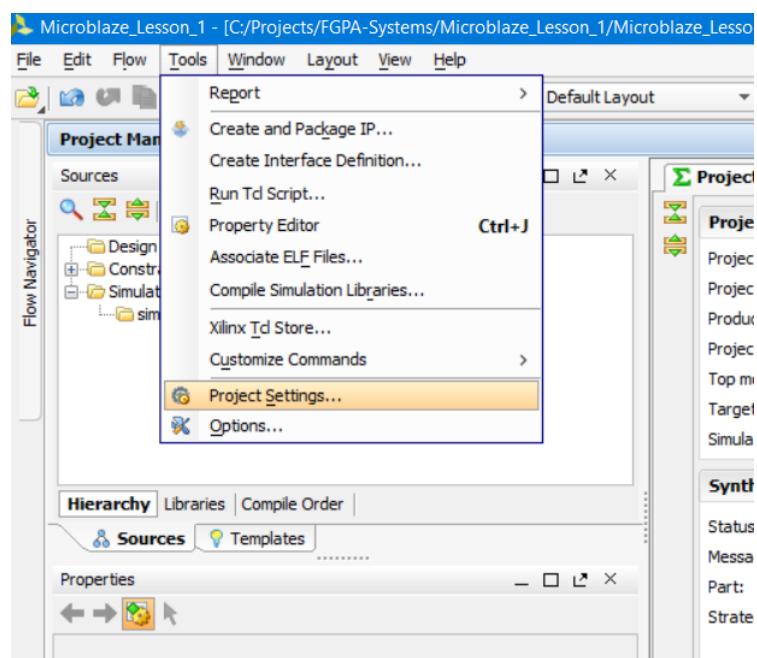


Рисунок 9 Кнопка для изменения настроек проекта

Создание HW-части

Для разработки HW-части софт-процессорной системы в Vivado используется IP Integrator [7], а сам процесс построения основан на добавлении и соединении готовых IP-ядер, которые могут быть как от Xilinx, так и от других фирм, а также разработанными Вами самостоятельно. Для запуска IP Integrator нужно нажать кнопку создания нового блочного проекта Create Block Design, которая находится в панели Flow Navigator – группа IP Integrator, пункт Create Block Design (рис. 10).

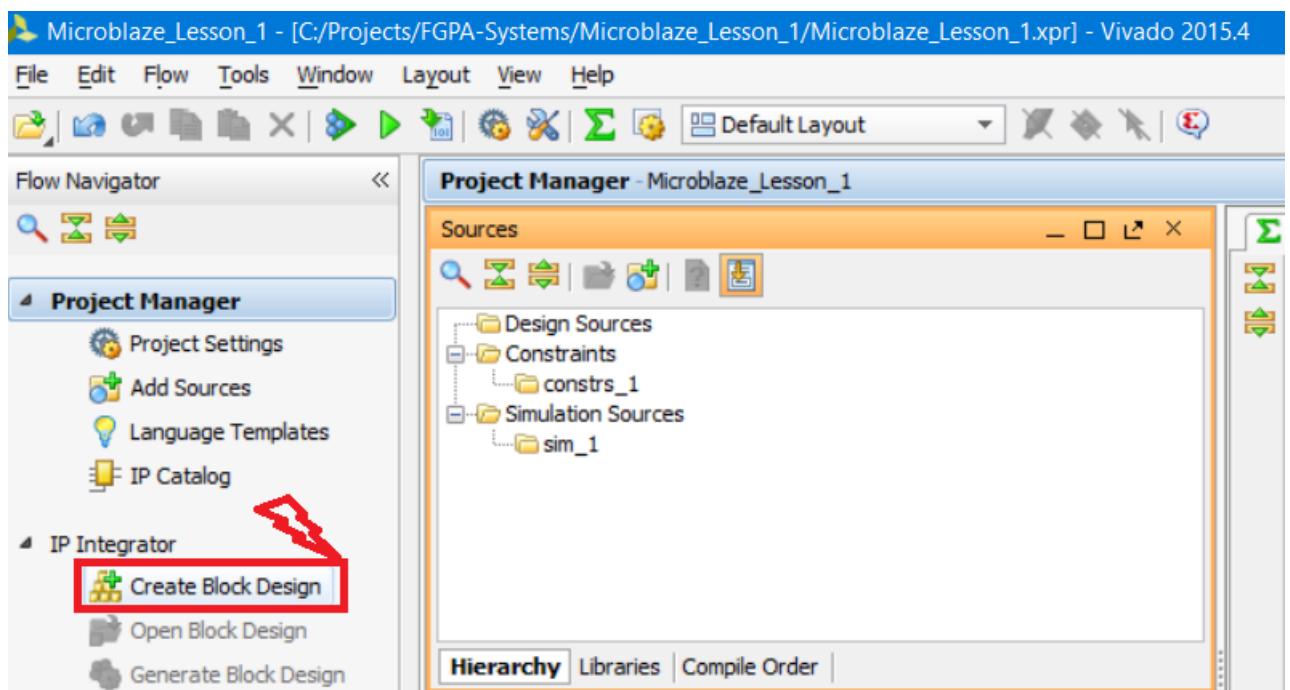


Рисунок 10 Кнопка создания нового блочного проекта

После нажатия на кнопку создания нового блочного проекта появится окно (рис. 11), которое позволяет задать его имя, определить папку для его размещения, если предлагаемая по умолчанию Вас не устраивает, и определить к какой подгруппе основной панели (дизайн или моделирование) будет относиться этот блок. Изменим:

1. Имя проекта на *system*
2. Нажимаем кнопку OK

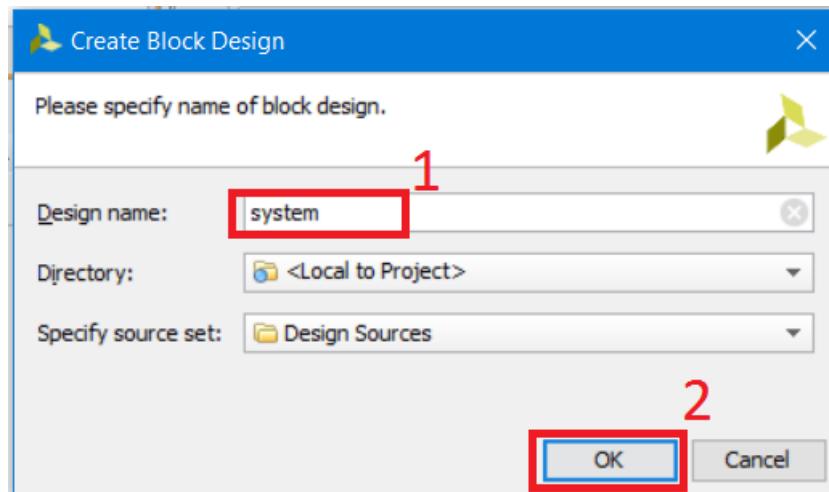


Рисунок 11 Окно настроек создаваемого блочного проекта

После нажатия на кнопку OK Vivado перейдёт в режим IP Integrator: (рис. 12).

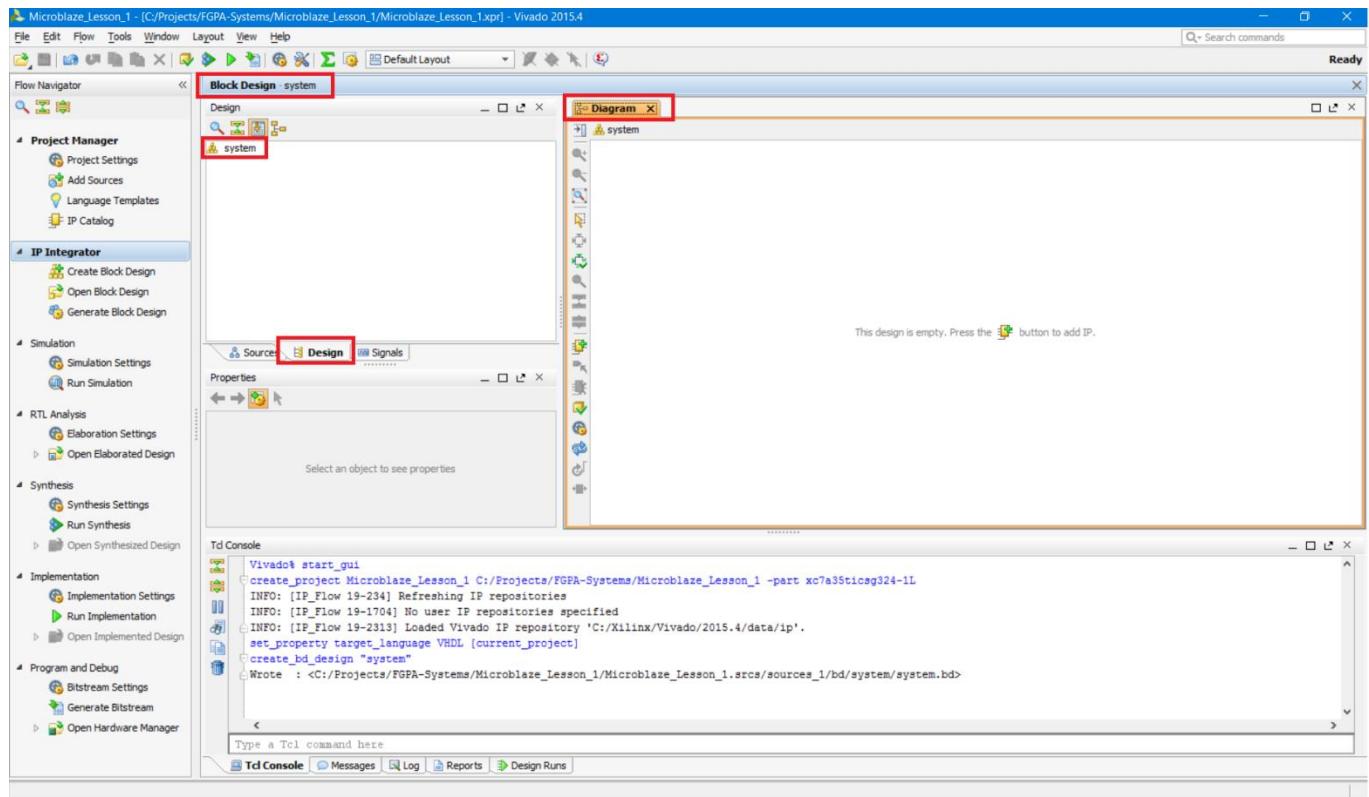


Рисунок 12 Окно Block Design с различными доступными вкладками

Теперь мы можем начать собирать процессорную систему. Процесс этот во многом автоматизирован, и мы этим воспользуемся. Первое, что нам необходимо добавить в софт-процессорную систему – это сам софт-процессор MicroBlaze. Есть

несколько вариантов его исполнения и добавления, но для проектирования таких простых систем, как наша, обычно используется мастер настройки. Для добавления IP-блоков на схему проекта (вкладка Diagram) можно воспользоваться кнопкой Add IP , расположенной в панели инструментов слева во вкладке Diagram или нажать Ctrl+I. После нажатия на кнопку откроется каталог блоков, которые можно добавить на поле Diagram (рис. 13).

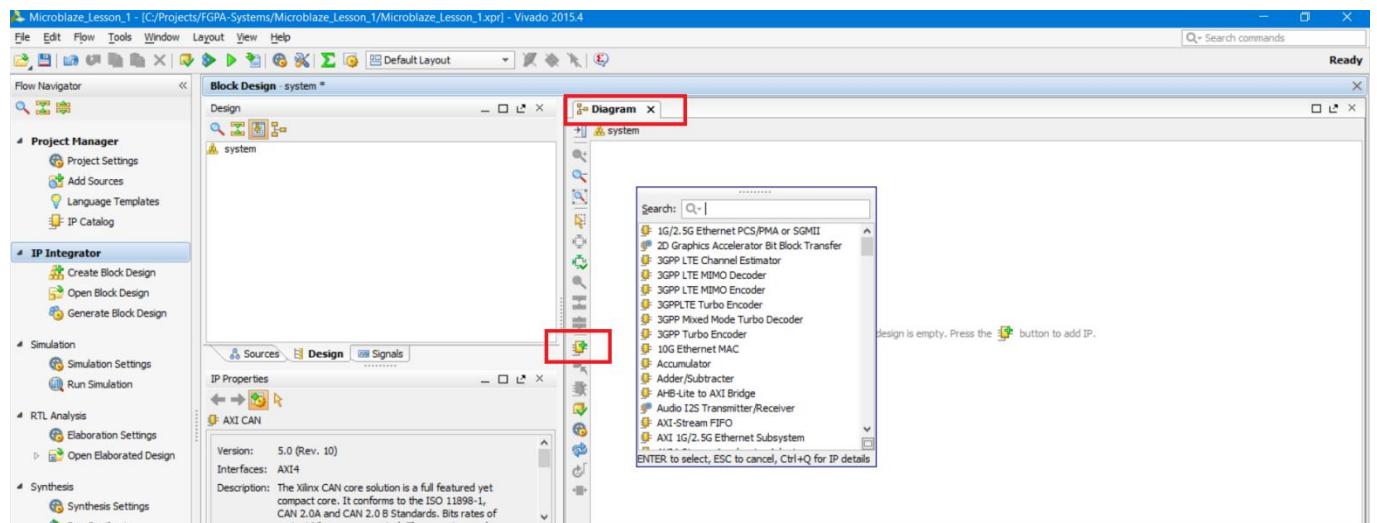


Рисунок 13 Окно со списком доступных IP блоков, после нажатия на кнопку Add IP

В появившемся окне со списком доступных IP блоков (рис. 14) в поле поиска Search пишем *MicroBlaze* и выбираем из найденных позиций MicroBlaze. Остальные два это MicroBlaze Debug Module (MDM) – отладчик для MicroBlaze, который будет позже добавлен автоматически, и микроконтроллерная система на базе MicroBlaze – фактически, ещё один вариант кастомизации процессорной системы.

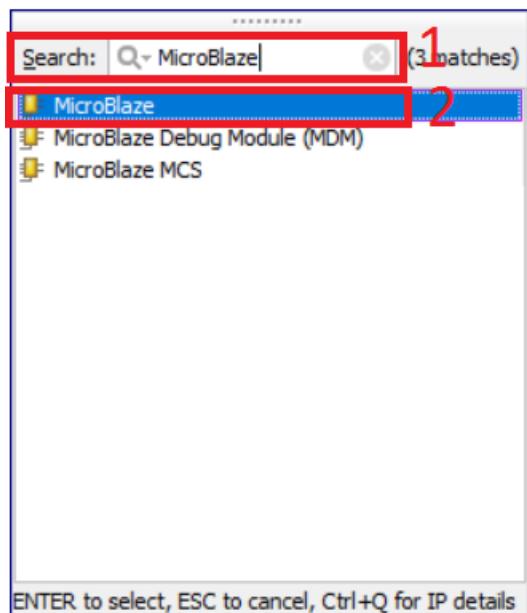


Рисунок 14 Окно со списком доступных IP блоков, с отфильтрованным списком MicroBlaze

Дважды кликаем по MicroBlaze, после чего IP блок появляется на поле (рис. 15). Обратите внимание, что также появилось окно настройки адресов (об этом позже), и «свесилась» зелёная строка помощи, которая появляется, если Vivado может как-то автоматизировать процесс (в нашем случае появилась надпись-ссылка Run Block Automation) и сам модуль MicroBlaze возник на поле Diagram.

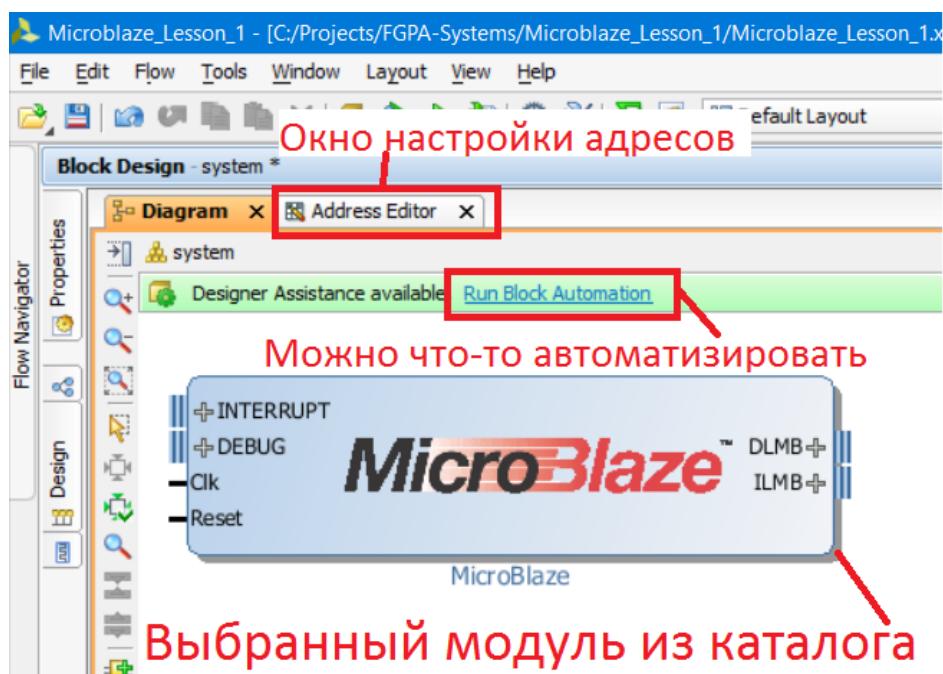


Рисунок 15 Появление дополнительных вкладок, опций автоматизации и IP блока после его добавления

Воспользуемся предлагаемой автоматизацией, нажав на ссылку Run Block Automation, после чего появится окно с мастера экспресс настроек процессорной системы (рис. 16). После вызова мастера доступны именно экспресс-настройки, то есть те, которые используются наиболее часто. Расширенные настройки доступны по двойному щелчку на IP-блоке MicroBlaze, но это выходит за рамки текущей статьи.

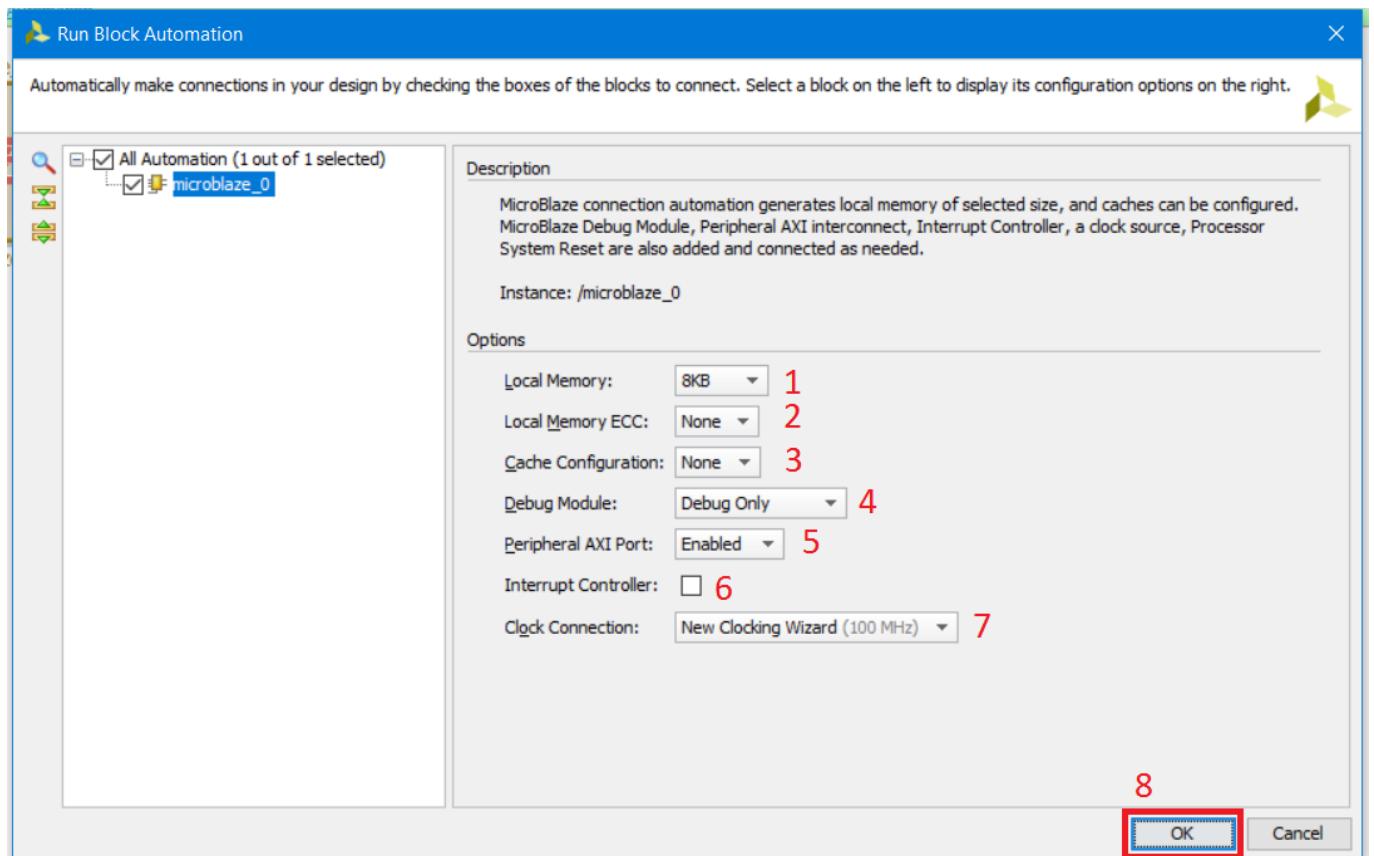


Рисунок 16 Мастер экспресс настрое процессорной системы

Из предлагаемых настроек доступны:

1. Количество памяти для программы и данных
2. Управление механизмом коррекции ошибок
3. Конфигурация кэша
4. Конфигурация отладчика (тот самый MDM из списка)
5. Задействование/отключение порта сопряжения с периферией (AXI-порта)
6. Включение/выключение контроллера прерываний

7. Выбор сигнала тактирования

Оставим все настройки в значениях по умолчанию и нажмём кнопку OK, т. к. для нас этих параметров достаточно. После окончания работы мастера на поле появится множество новых блоков, которые мы разберём ниже. Для оптимизации рабочего поля нажмите кнопку перестроить  в панели инструментов (рис. 17).

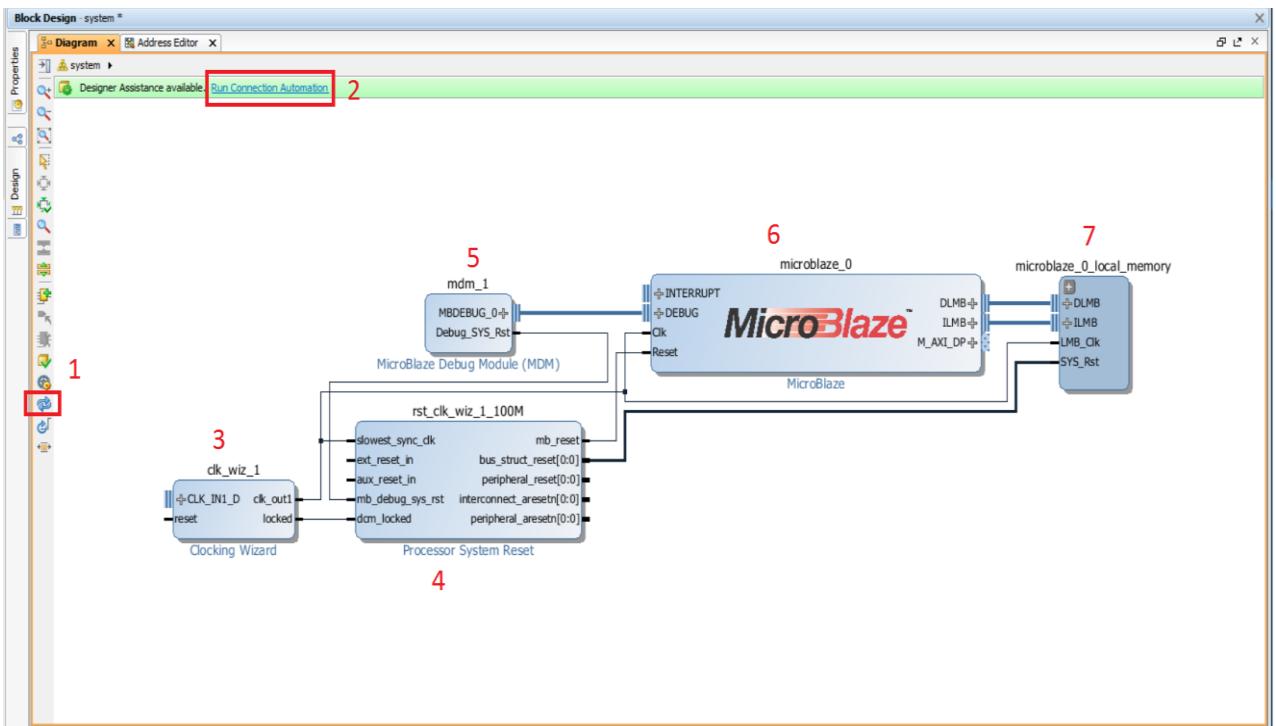


Рисунок 17 Результат работы мастера экспресс-настроек после оптимизации рабочего поля

Как видите, картина изменилась, и на рабочем поле появились:

1. Кнопка оптимизации рабочего поля
2. Предложение по автоматическому подключению одних блоков к другим (нажмите на неё и посмотрите, что предлагается подключить; после просмотра нажмите Отмена – Cancel)
3. Блок управления тактовой частотой и синхронизацией
4. Блок сброса процессорной системы
5. Модуль отладки (отладчик)
6. Ядро софт-процессора MicroBlaze
7. Локальная память данных и программ

Теперь добавим необходимую периферию. В начале статьи мы определились, что будем мигать светодиодом, и выдавать сообщения по UART. Значит, нужно добавить IP-блоки, которые будут обеспечивать этот функционал. Для мигания будем использовать IP-блок ввода/вывода общего назначения GPIO [9] , а для вывода сообщений – UART Lite [10].

Добавление IP блоков аналогично добавлению MicroBlaze. Поэтому нажимаем кнопку  , и в поле поиска пишем *gpio*, выбираем модуль AXI GPIO и нажимаем Enter (рис. 18)

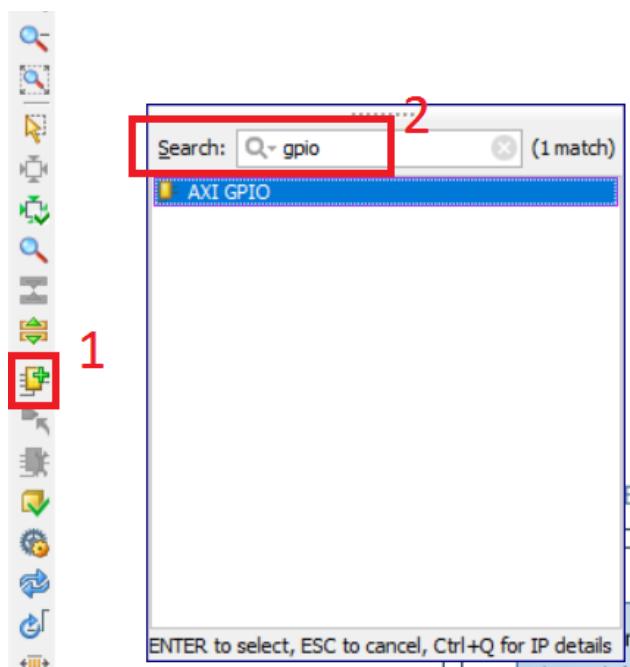


Рисунок 18 Поиск GPIO в каталоге доступных IP блоков

На схему должен добавиться блок, изображённый на рис. 19.

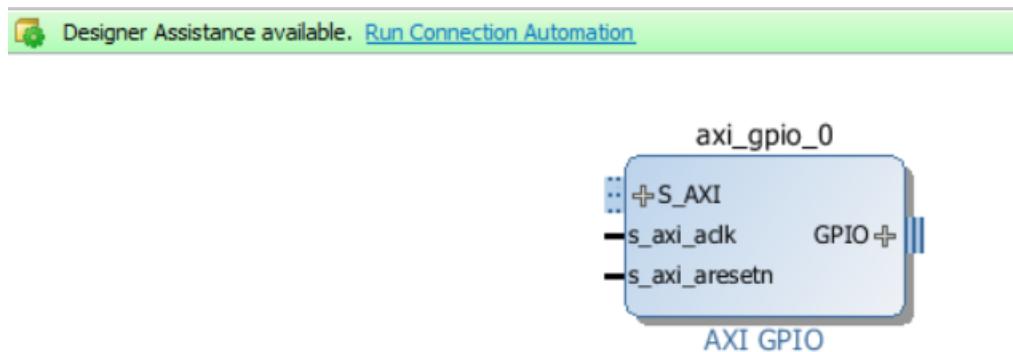


Рисунок 19 Блок AXI GPIO

После добавления блока AXI GPIO необходимо его настроить, задать разрядность портов, определить направление и количество каналов. Настройка производится двойным щелчком по блоку либо выбором Customize Block из контекстного меню, открывающегося по щелчку правой кнопкой мыши. Модуль axi_gpio_0 будет управлять светодиодом, периодически включая и выключая его. Значит axi_gpio_0 должен быть сконфигурирован следующим образом (рис. 20)

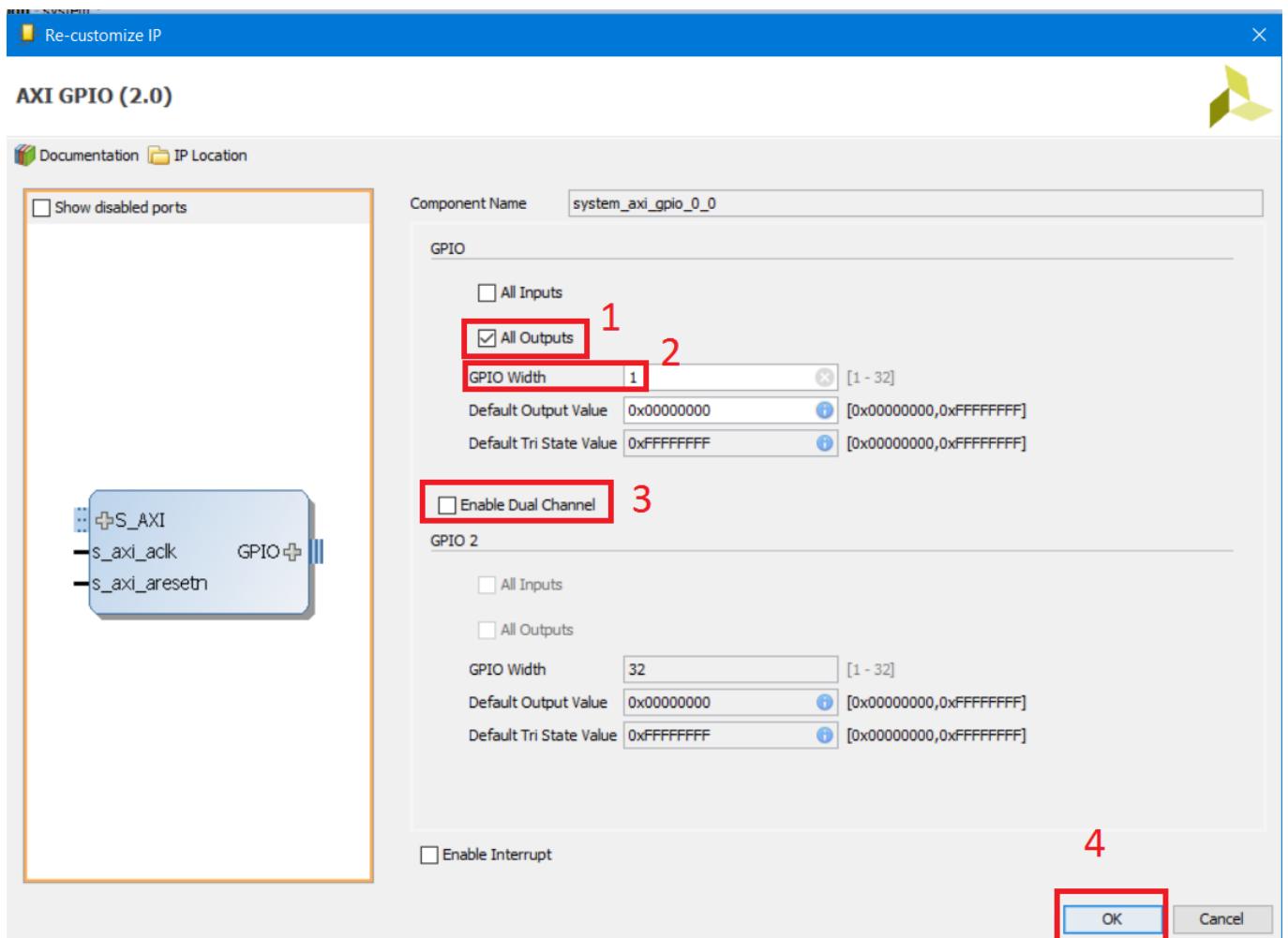


Рисунок 20 Настройка блока AXI GPIO

1. Задаём направление портов блока AXI GPIO – выбираем, что все выводы являются выходами, т. к. мы управляем светодиодом, а не он нами.
2. Ставим разрядность порта 1 – всего будет подключён 1 светодиод.
3. Второй канал нам не нужен, отключаем его.

4. Нажимаем кнопку OK.

Нажав плюсик около интерфейса GPIO модуля axi_gpio_0 увидим, что порт gpio_io_o имеет разрядность 1 (вектор [0:0]) – рис. 21.



Рисунок 21 Вид блока AXI GPIO после настройки

Теперь давайте попробуем подключить наш настроенный блок AXI GPIO к MicroBlaze. Предлагаю довериться автоматике и посмотреть, что нам создаст Vivado. Нажимаем на строку Run connection Automation (см рис. 19) и ждём появления окна помощника доступных подключений (рис. 22)

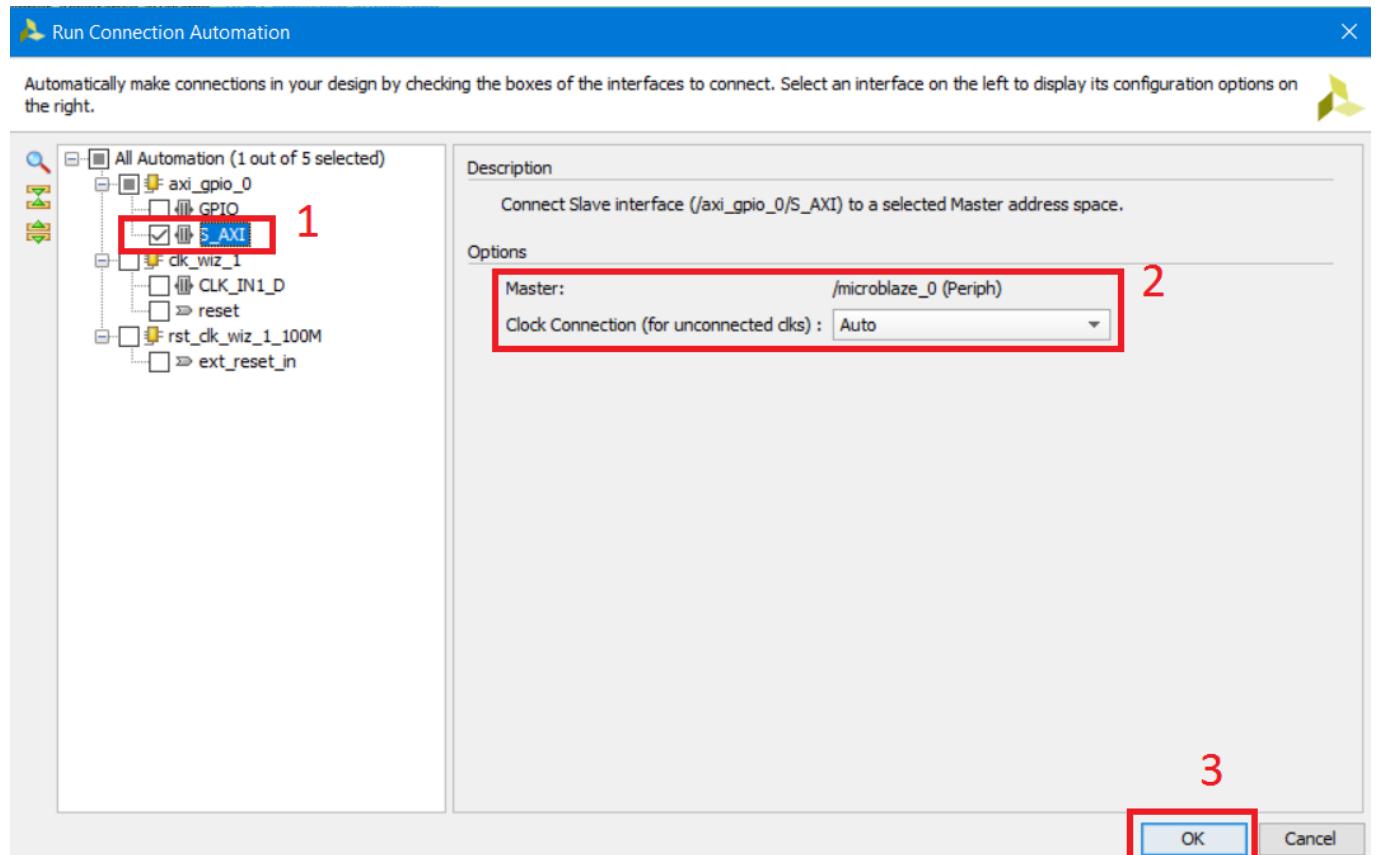


Рисунок 22 Окно помощника подключений

В окне помощника подключений:

1. Выберите подключение `axi_gpio_0` и поставьте галочку `S_AXI` (далее мы рассмотрим и поясним это подключение).
2. Установите подключение тактового сигнала как `Auto`.
3. Нажмите `OK`.

После завершения подключения нажмите кнопку оптимизации рабочего пространства  в панели инструментов.

Как вы можете заметить, появился дополнительный модуль AXI Interconnect [11] (рис. 23). Кратко опишу его назначение. Дело в том, что взаимодействие между процессором и периферией происходит по шине AXI [12] (о ней и её видах должна быть отдельная статья, если кто-то захочет в этом помочь – напишите мне). Нашине есть мастер (обычно это процессор) и слэйв (Slave), в нашей литературе это ведущий и ведомый соответственно. Мастер отправляет команды слейву. Однако AXI не позволяет подключить к мастеру более одного слейва напрямую. Именно напрямую нельзя, но можно через коммутатор, который и называется AXI Interconnect – назначение этого модуля обеспечить подключение между несколькими мастерами и несколькими слейвами. Пока в нашей системе один мастер – `microblaze_0` и один слейв – `axi_gpio_0`.

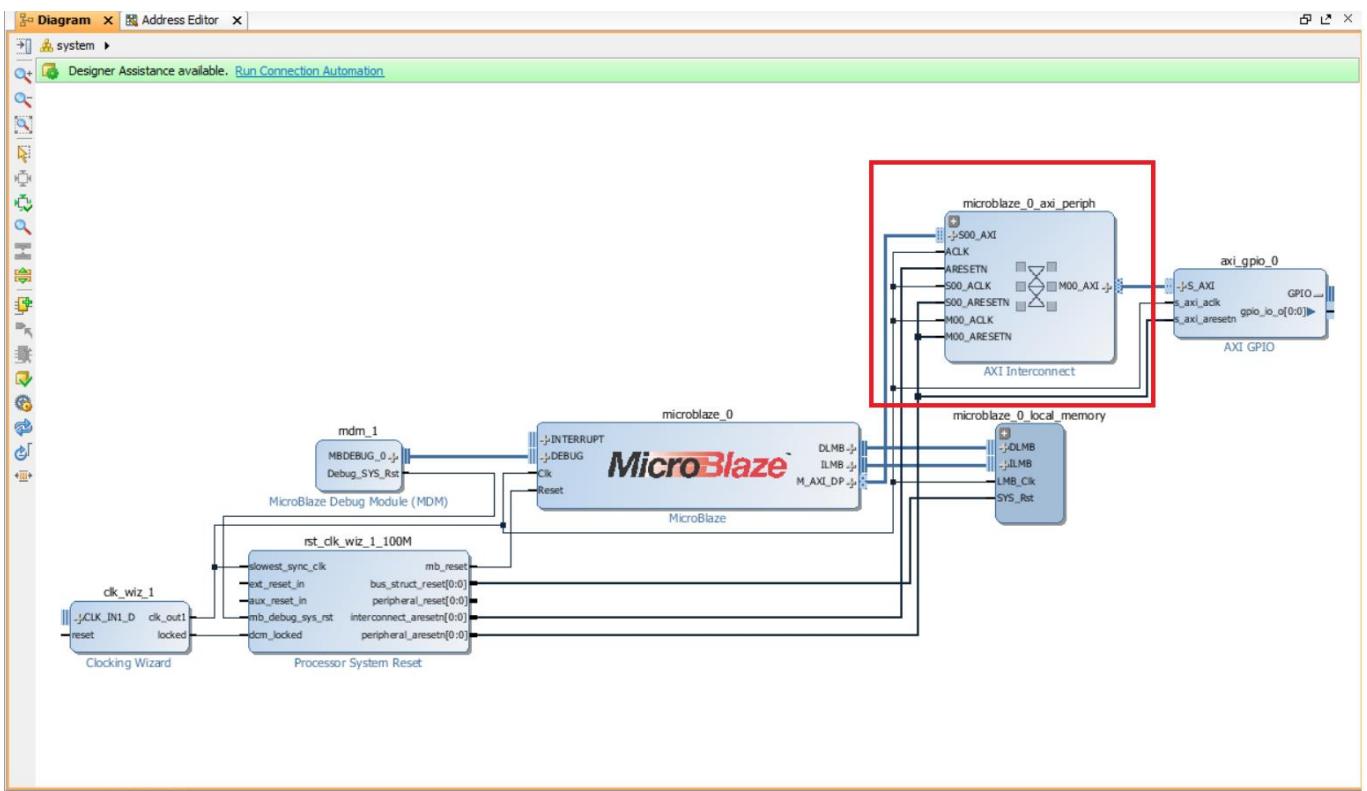


Рисунок 23 Результат работы помощника подключений: автоматическая вставка модуля AXI Interconnect, его подключение к MicroBlaze и AXI GPIO

После работы мастера синяя строка сверху не пропала. Это потому что ещё есть что автоматизировать, но делать этого мы пока не будем. Теперь давайте попробуем добавить модуль UART Lite в нашу систему.

Надеюсь, что последовательность запомнили. Нажимаем кнопку , вводим в строке поиска Uartlite, дважды кликаем и ждём. Если все сделали корректно, должен появиться модуль как на рис. 24.

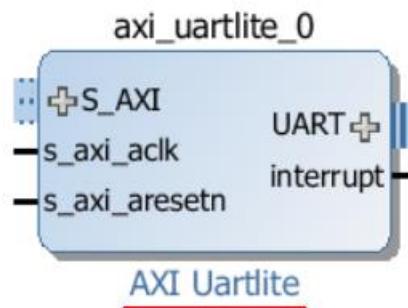


Рисунок 24 Блок Uartlite (именно Uartlite, а не тот, который не Uartlite)

При желании просмотрите доступные настройки для модуля и если есть необходимость, измените. Настройки обычные: скорость (9600), количество стоп бит (1), чётность (нет), количество бит (8).

Для подключения Uartlite к нашей системе, давайте вновь воспользуемся автоматизацией, которую предлагает Vivado, нажав строку Run Connection Automation. Для модуля axi_uart_0 выберите S_AXI и нажмите OK (рис. 25)

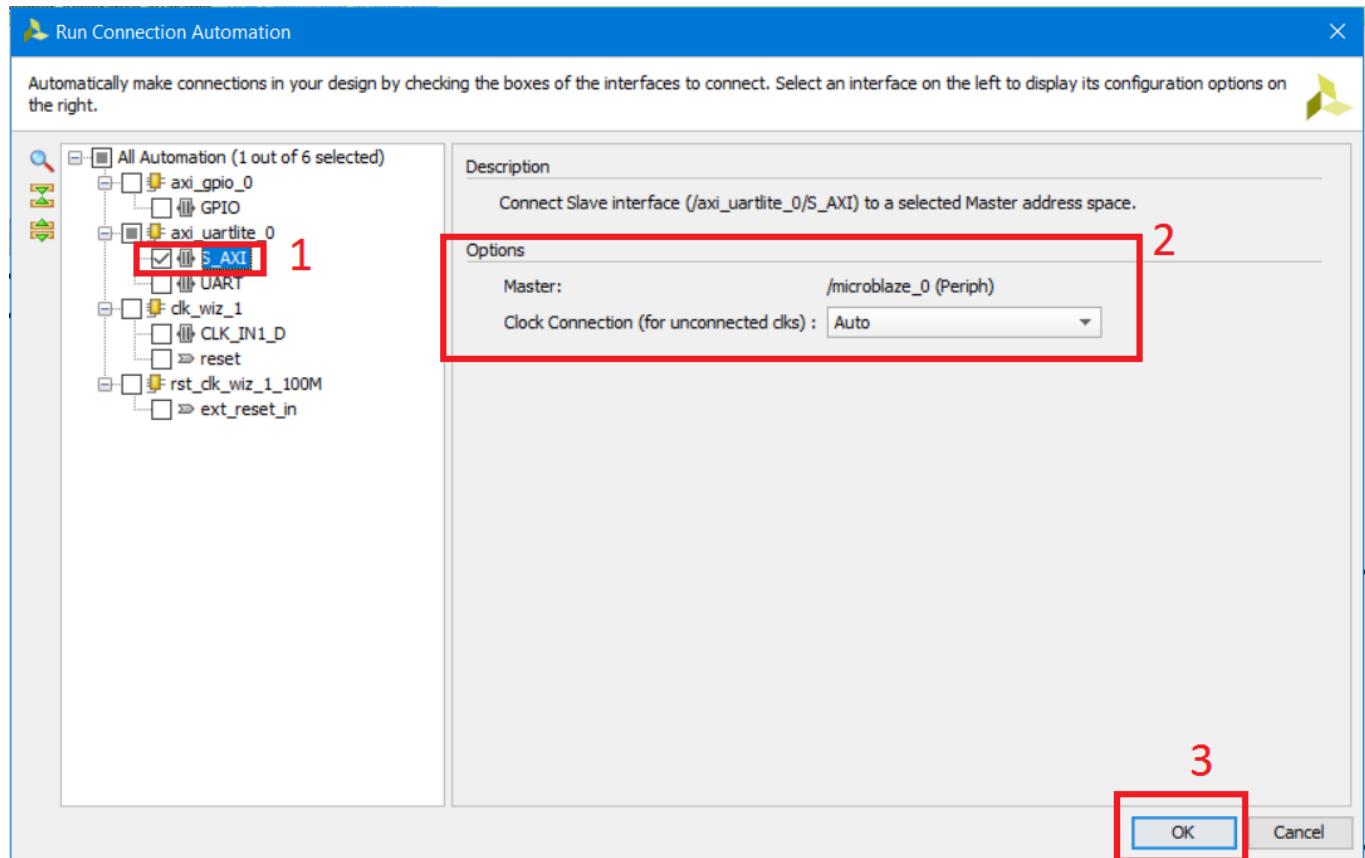


Рисунок 25 Настройка помощника подключений для Uartlite

После завершения подключения нажмите кнопку оптимизации рабочего пространства в панели инструментов и обратите внимание на изменения в AXI Interconnect (если не помните что было, обратитесь к рис. 23).

Что произошло и почему? В AXI Interconnect добавился дополнительный мастер-порт для подключения второго слэйва (Uartlite – это ведомое устройство). MicroBlaze имеет один мастер-порт, который называется M_AXI_DP. Мастер один, а слейва два, но подключать напрямую можно только один слейв. Для подключения

нескольких слейвов мы используем AXI Interconnect. Поэтому в AXI Interconnect добавился ещё один порт (рис. 26). В общем случае, AXI Interconnect может обеспечивать взаимное подключение нескольких мастеров и нескольких слейвов.

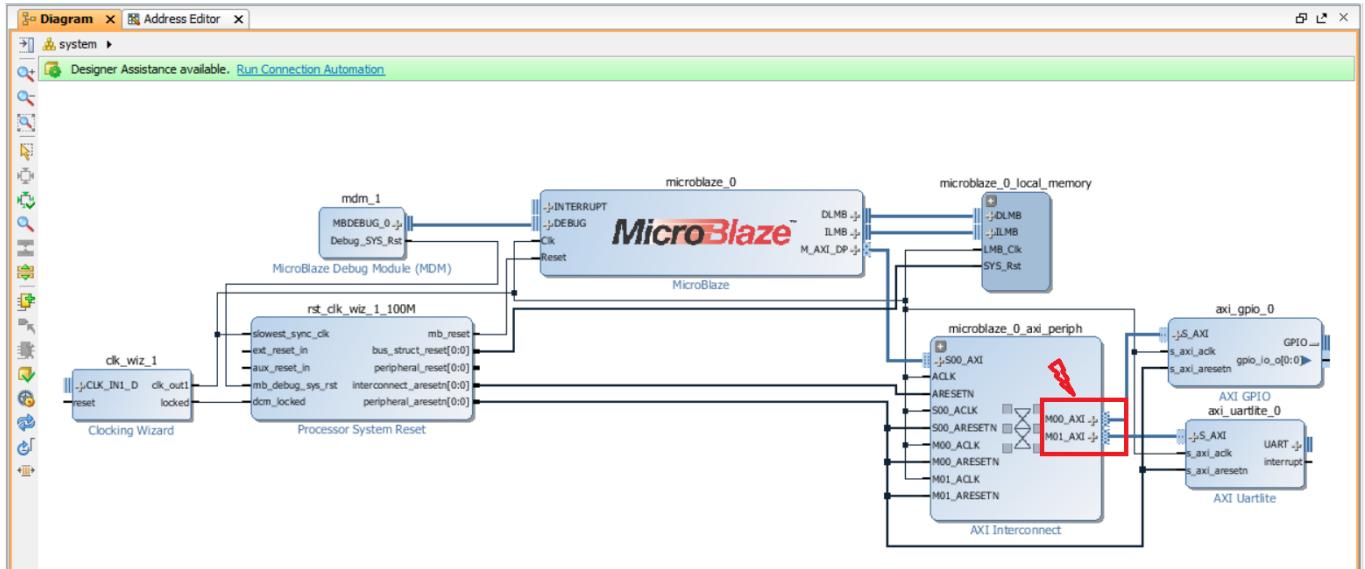


Рисунок 26 После окончания работы помощника подключений, количество слейвов устройств на шине стало равным двум. Поэтому в AXI Interconnect добавился ещё один порт

Часть дела сделана. Теперь давайте настроим модуль управления тактовой частотой и синхронизацией clk_wiz_1. Для Ваших плат значения могут быть другими – надеюсь, что Вы сможете найти в документации значение частоты системного тактового генератора, тип идущего от него сигнала, и ножку(и) FPGA к которым он подключен. Ниже приведены настройки для Arty Board (рис. 27). Настройка любого модуля производится двойным щелчком по нему.

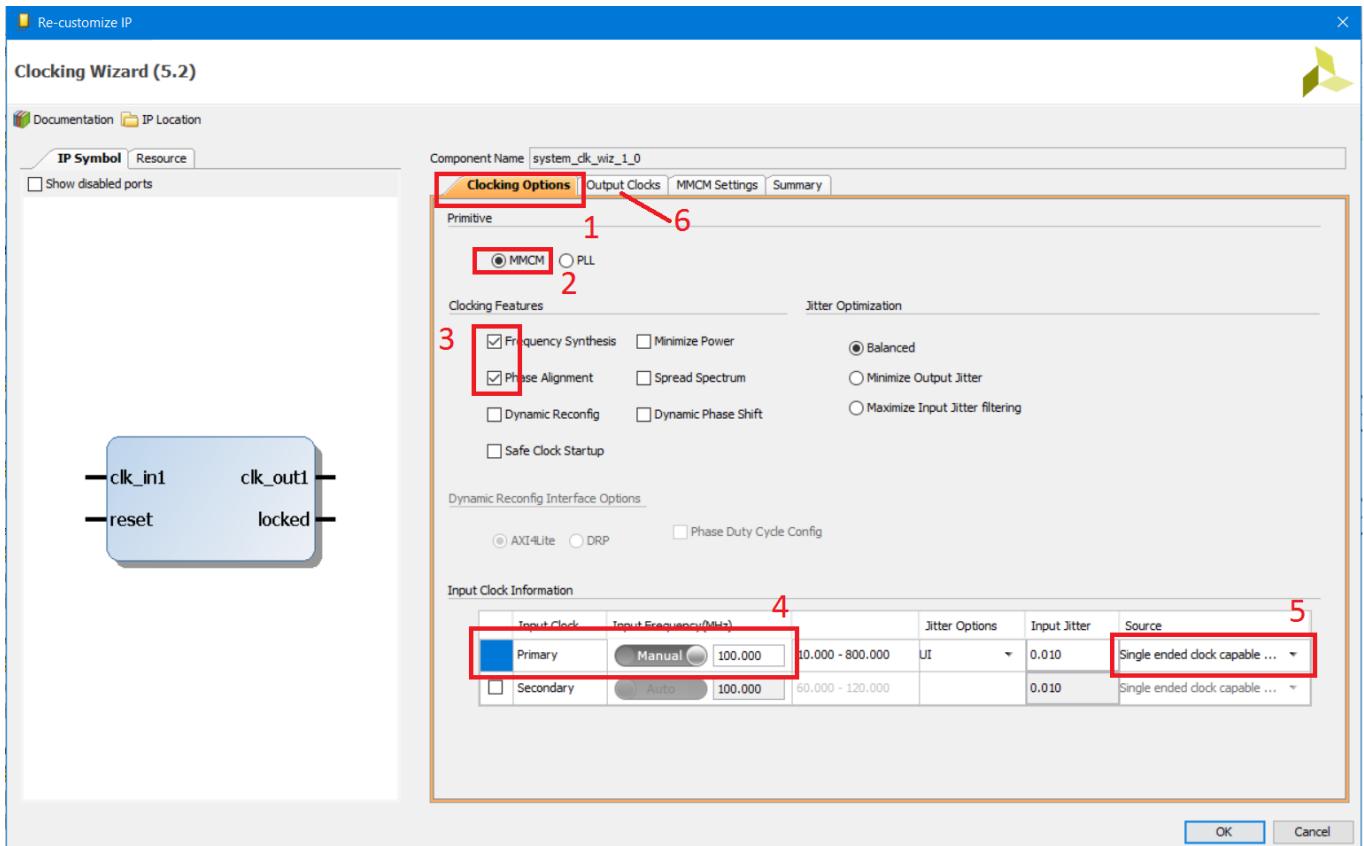
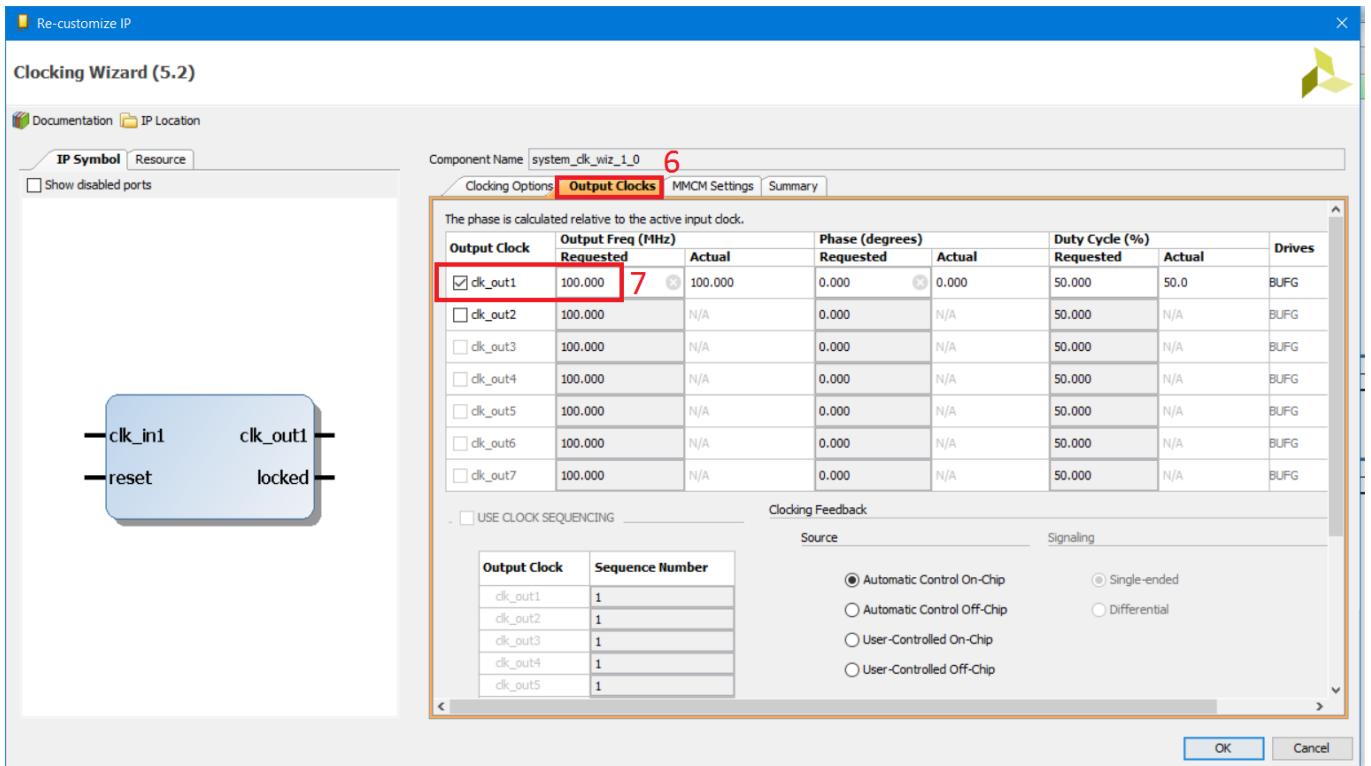
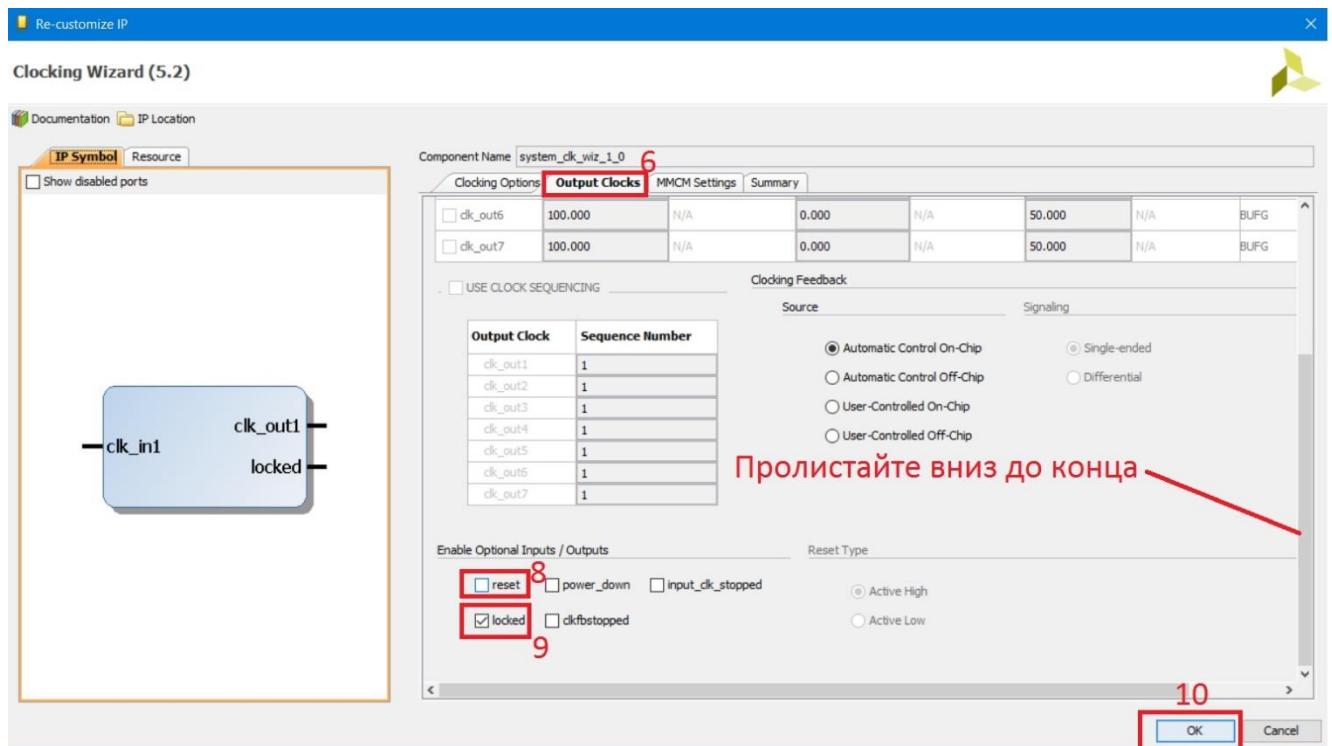


Рисунок 27 Окно настройки модуля управления тактовой частотой и синхронизацией: вкладка Clocking Options
В окне настроек clk_wiz_1:

1. Выберите вкладку Clocking Options.
2. Тип модуля – MMCM.
3. Включены возможности синтеза частоты и фазовой подстройки.
4. Основная тактовая частота – 100МГц.
5. Тип сигнала с системного тактового генератора на плате – однополярный.
6. Затем выберите вкладку Output Clocks (рис. 28).
7. На ней – установите выходное значение частоты (тактовая частота процессора) – также 100 МГц.
- Пролистайте вниз до конца
8. Снимите галочку с сигнала reset.
9. Установите галочку locked.
10. Нажмите OK.



Пролистайте вниз



Пролистайте вниз до конца

Рисунок 28 Окно настройки модуля управления тактовой частотой и синхронизацией: вкладка Clocking Options

Установленные настройки могут повлиять на внешний вид clk_wiz_1 (для Arty Board он примет вид, как на рис. 29).



Рисунок 29 Изменение внешнего вида модуля после проделанных настроек (для Arty Board)

Если оставить незадействованные выводы модуля сброса неподключенными, то может возникнуть ситуация, при которой процессор будет всегда в состоянии сброса. Потому выполним явное их подключение к логической «1» – неактивному для них логическому уровню. Для этого:

1. Добавьте блок constant.
2. Установите в нем значение 1 (если по умолчанию оно другое).
3. Подключите выход блока Constant к входам aux_reset_in и ext_reset_in блока rst_clk_wiz_1_100M.

Подключение осуществляется следующим образом: подведите мышку к выходу блока Constant (указатель должен принять вид карандашка), нажмите на выход и, не отпуская левую кнопку мыши, подведите соединение к входу aux_reset_in – а затем повторите те же действия для ext_reset_in. В очередной раз нажмите кнопку оптимизации рабочего пространства  в панели инструментов. Соединение должно принять вид, как на рис. 30.

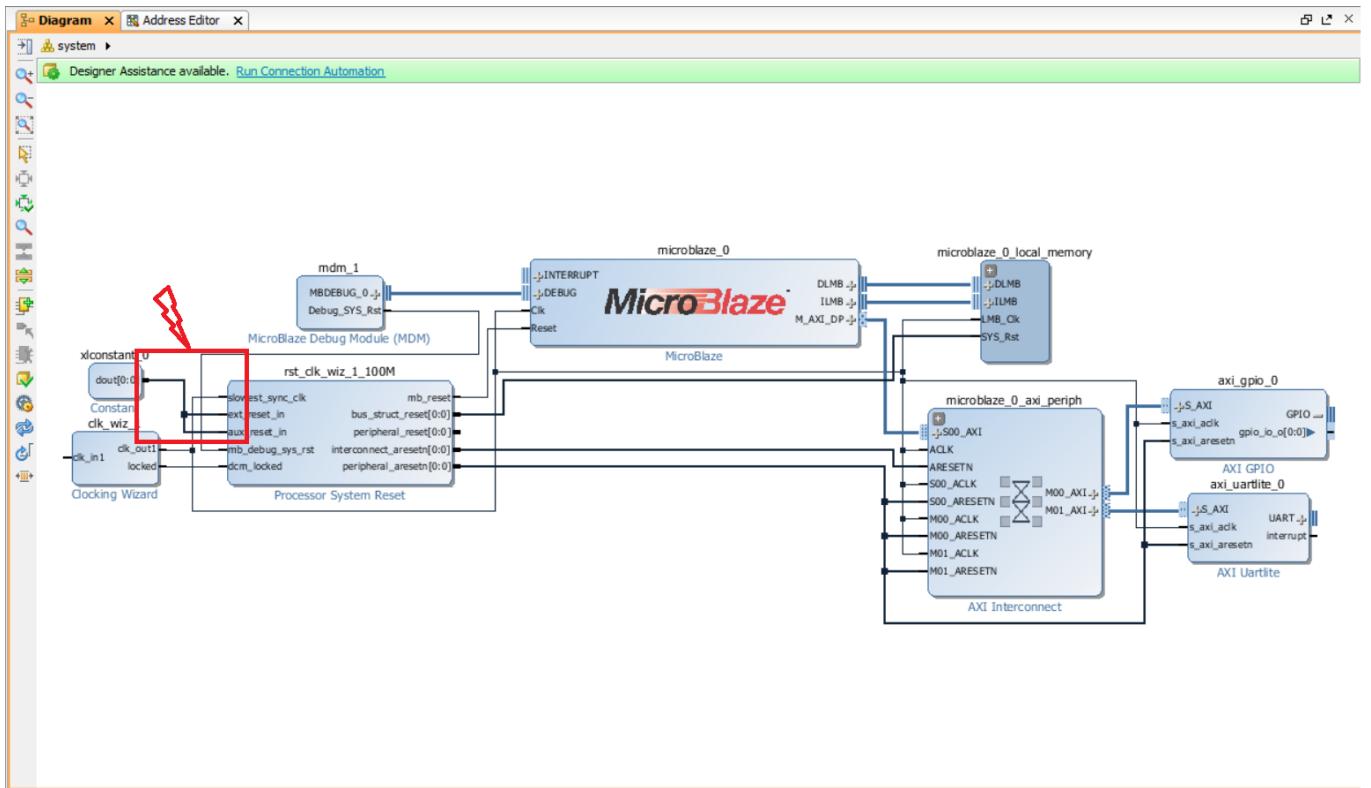


Рисунок 30 Соединение выхода блока Constant с входами aux_reset_in и ext_reset_in блока rst_clk_wiz_1_100M

Теперь необходимо создать внешние входы и выходы для нашей процессорной системы. Если вы обратили внимание, сейчас не подключены `clk_in1` и выходы экземпляров модулей `axi_gpio_0` и `axi_uartlite_0`. Сейчас мы должны обозначить, какие порты являются внешними и должны выходить из нашей процессорной системы наружу.

Назначим вход `clk_in1` модуля Clocking Wizard внешним. Для этого кликнем по нему правой кнопкой мыши и выберем Make External (рис.31)

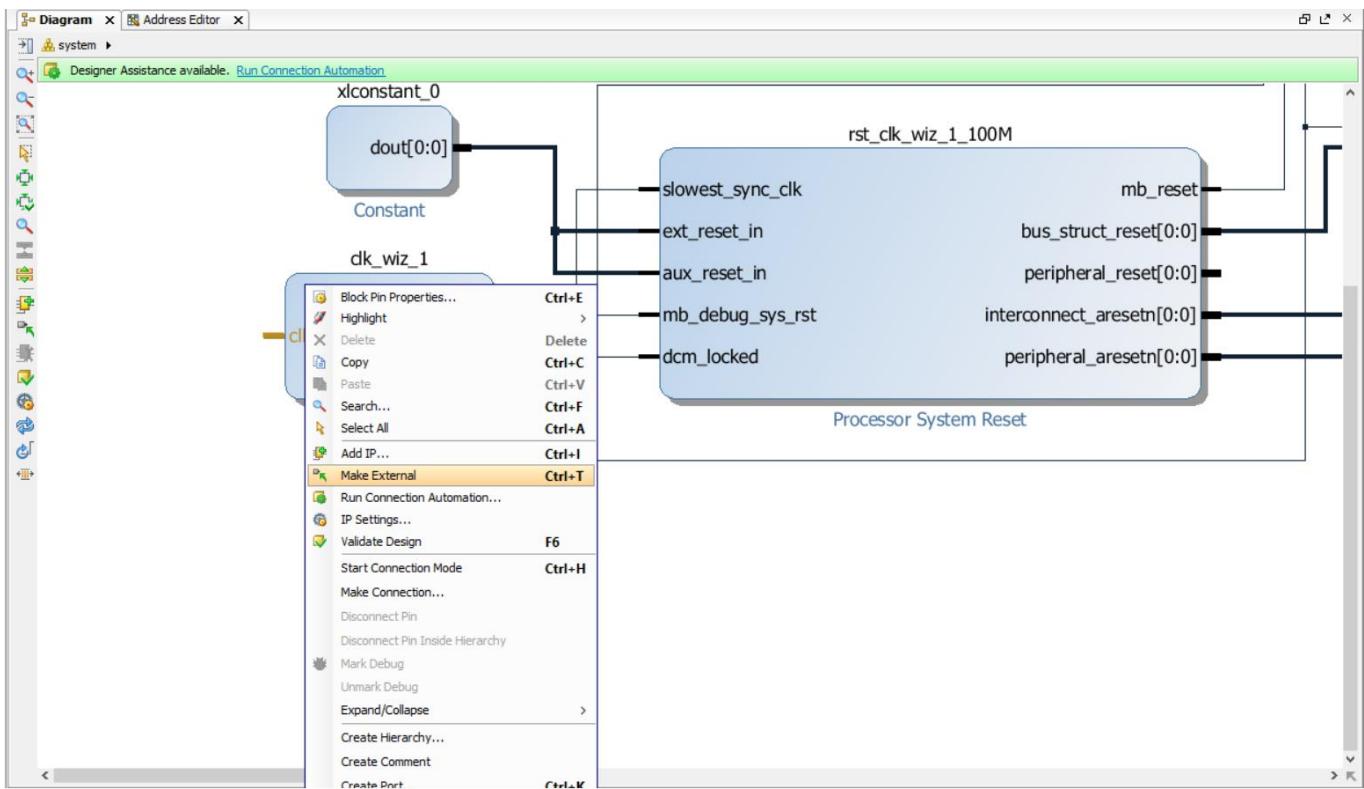


Рисунок 31 Объявление входа clk_in1 модуля Clocking Wizard внешним

После этого к входу clk_in1 будет подключён порт, с таким же именем (рис. 32)

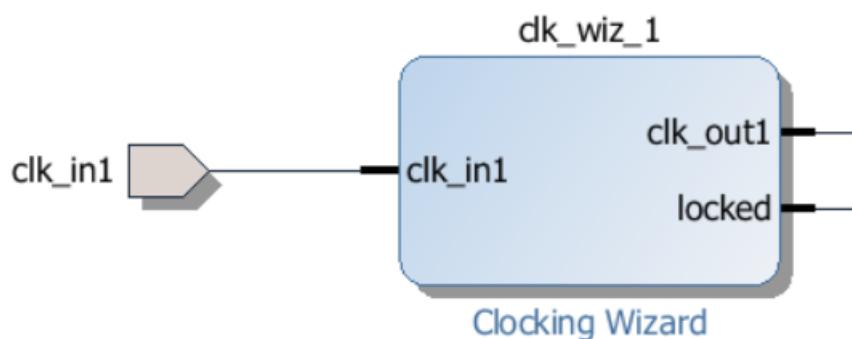


Рисунок 32 Внешний порт clk_in1, созданный автоматически, подключён к входу clk_in1 модуля clk_wiz_1

Аналогично поступаем и с выходом GPIO модуля axi_gpio_0 и шиной UART модуля axi_uartlite_0. Сделать внешним можно как шину, так и отдельный порт из шины; а если портов вшине несколько (как вшине UART), то направление портов

будет определено автоматически (см. 33). Некоторые незадействованные порты можно оставить неподключёнными – например, interrupt модуля axi_uartlite_0.

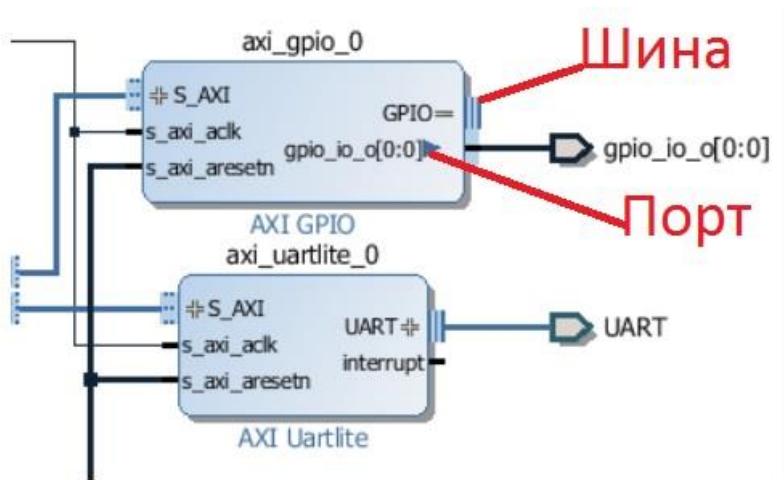


Рисунок 33 Объявление внешними порт управления светодиодом и шины UART

Помните про вкладку Address Editor (рис.34)? Самое время обратиться к ней и понять её назначение.

Наши модули соединены по шине AXI, полное название AXI Memory Map, это значит, что все модули, подключённые к шине должны иметь уникальный адрес, чтобы процессор смог к ним обратиться и «не перепутать» один модуль с другим. Назначение уникальных адресов (распределение адресного пространства) как раз и происходит во вкладке Address Editor, где в ручном, полуавтоматическом или автоматическом режиме можно установить диапазон адресов для конкретных модулей.

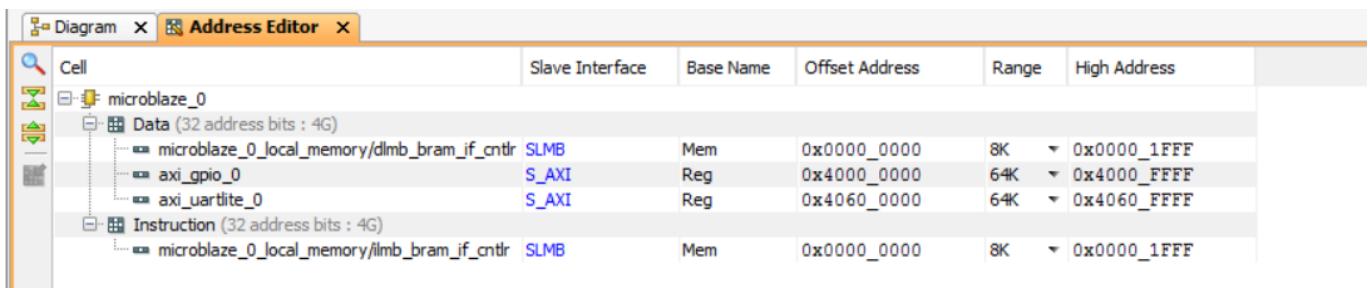


Рисунок 34 Вкладка Address Editor

Поскольку мы пользовались помощником соединений и мастерами настроек, адресное пространство уже полностью сконфигурировано, но если бы мы делали подключение шин в ручном режиме, то увидели бы следующую картину (рис. 35).

Эта картинка только для демонстрации: я временно удалил шину AXI, соединяющую экземпляры AXI Interconnect и GPIO, а потом соединил её вручную. Из-за ручного соединения назначения адреса не произошло. Но мы можем назначить адрес, нажав на кнопку автоматического назначения адресов на панели инструментов вкладки Address Editor. После этого, картинка станет прежней (то есть как на рис. 34).

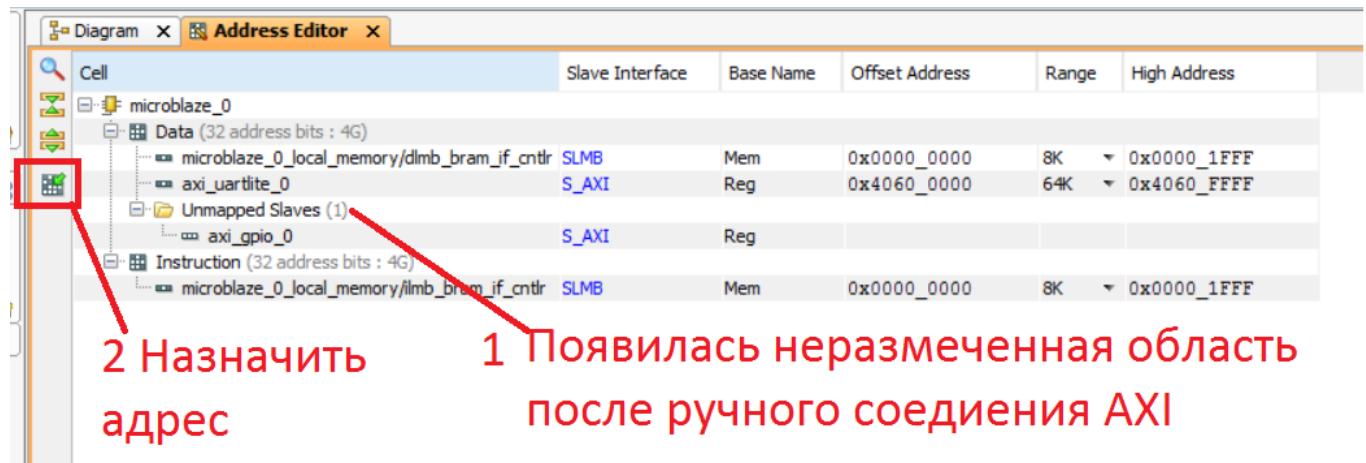


Рисунок 35 В случае ручного соединения шин AXI появляется неразмеченная область. Нажатие кнопки запускает автоматическое распределение адресного пространства

К сожалению, для нашей будущей программы не достаточно памяти, которую мы задали, когда пользовались мастером экспресс-настроек MicroBlaze (рис. 16). Нам необходимо увеличить количество памяти для данных и инструкций с 8К до 16К. Сделать это можно простым выбором из выпадающего списка доступного количества памяти, нажав на стрелочку в соответствующем поле (рис. 36). Изменить необходимо размер памяти и инструкций. В обоих полях должно быть значение 16К

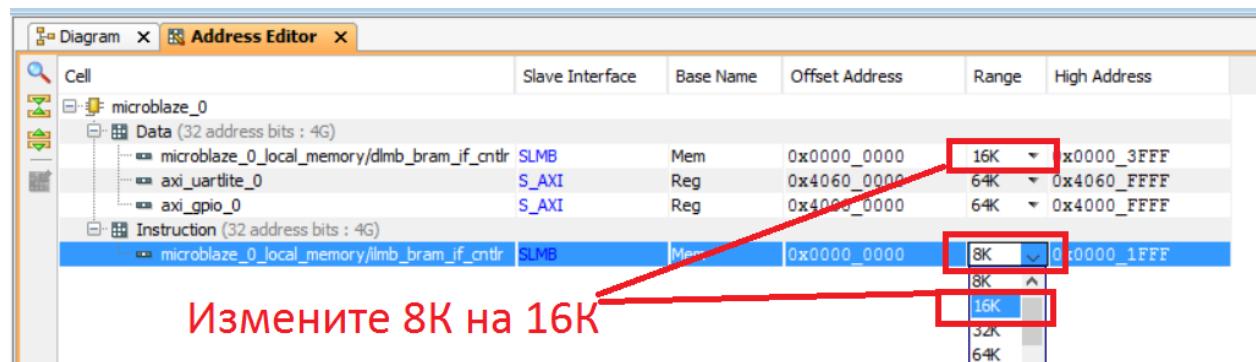


Рисунок 36 Изменение количества памяти, выделяемой для процессорной системе

Мы проделали много действий, закончили собирать систему и назначили все адреса. Но все ли корректно и правильно? В панели инструментов на вкладке Diagram есть одна из самых важных кнопок, называется она Validate Design (2 на рис. 37). Нажатие этой кнопки запускает инструмент проверки ошибок сборки HW-части процессорной системы (рис. 37). Нажмите эту кнопку и дождитесь результата. Если появилось окно, как на рис. 37, то поздравляю Вас, сборка HW-части проекта закончена. Если же появились ошибки – внимательно прочитайте их и постарайтесь самостоятельно исправить, вернувшись по тексту к соответствующим пунктам либо проделав всю последовательность с начала ещё раз, более внимательно и аккуратно.

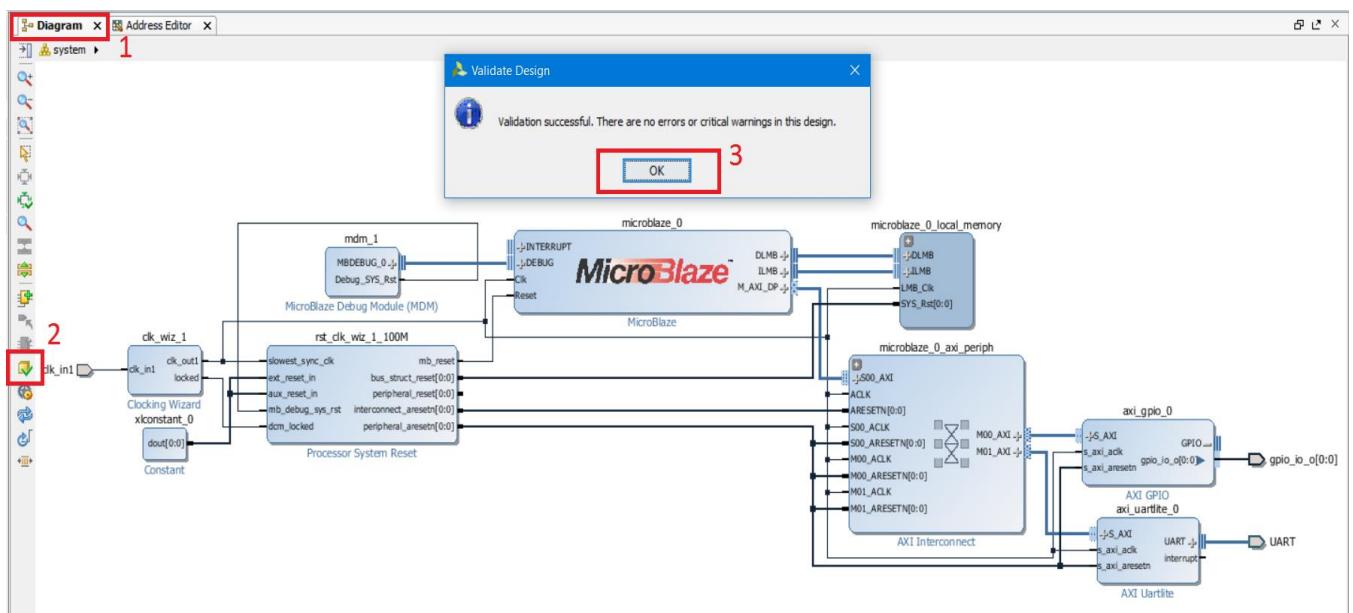


Рисунок 37 Проверка на ошибки сборки HW части процессорной системы.

Нажмите на основной панели Vivado кнопку сохранения, а затем – кнопку Project Manager, чтобы выйти из IP Integrator и вернуться в основной режим работы Vivado (рис. 38):

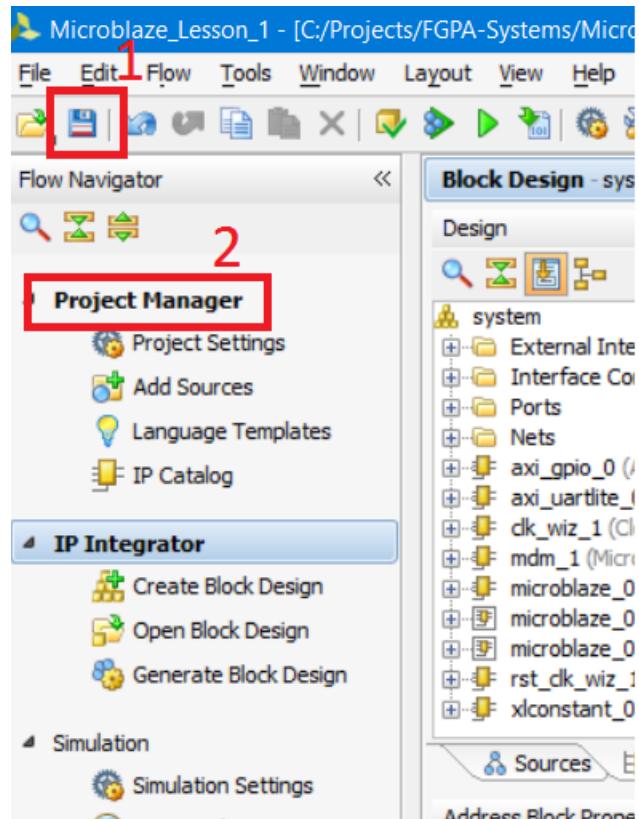


Рисунок 38 Сохранение и выход из IP Integrator

Далее следуют стандартные этапы проектирования: синтез, имплементация, генерация файла прошивки (битстрима). Но для блочного проекта обязательно нужно сделать обёртку (wrapper); это показано на рис. 39. Нажмите правой кнопкой на созданном блочном проекте и выберите Create HDL Wrapper. В появившемся окне нажмите OK.

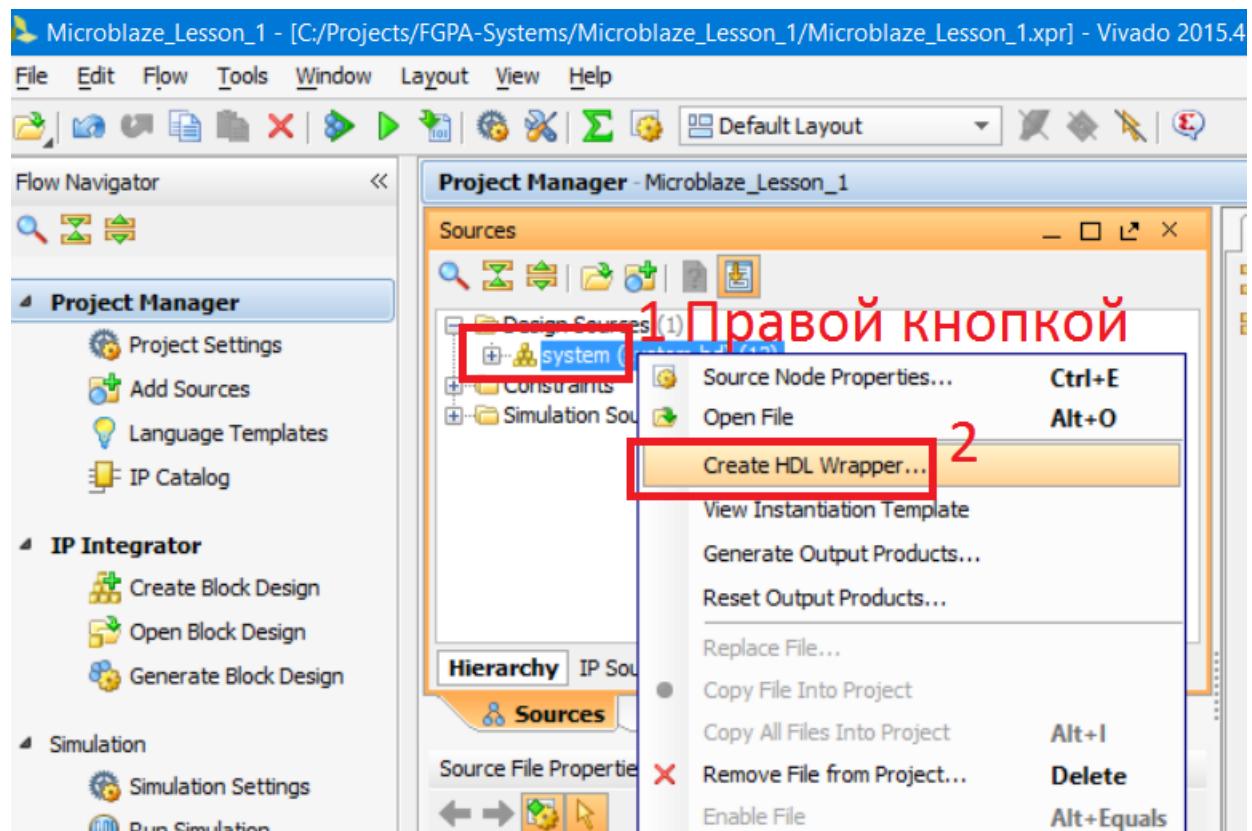


Рисунок 39 Создание обёртки для блочного проекта.

Обёртка – это простой VHDL- или Verilog-файл (с расширениями «vhd» или «v» соответственно), в который включена наша процессорная система, как часть иерархии. Таким образом, нашей собранной процессорной системой мы сможем оперировать, как простым модулем, добавляя её в качестве подмодуля в модули верхнего уровня. После создания обёртки наш модуль можно наконец-то запустить на синтез, нажав на кнопку Run Synthesis (рис. 40)

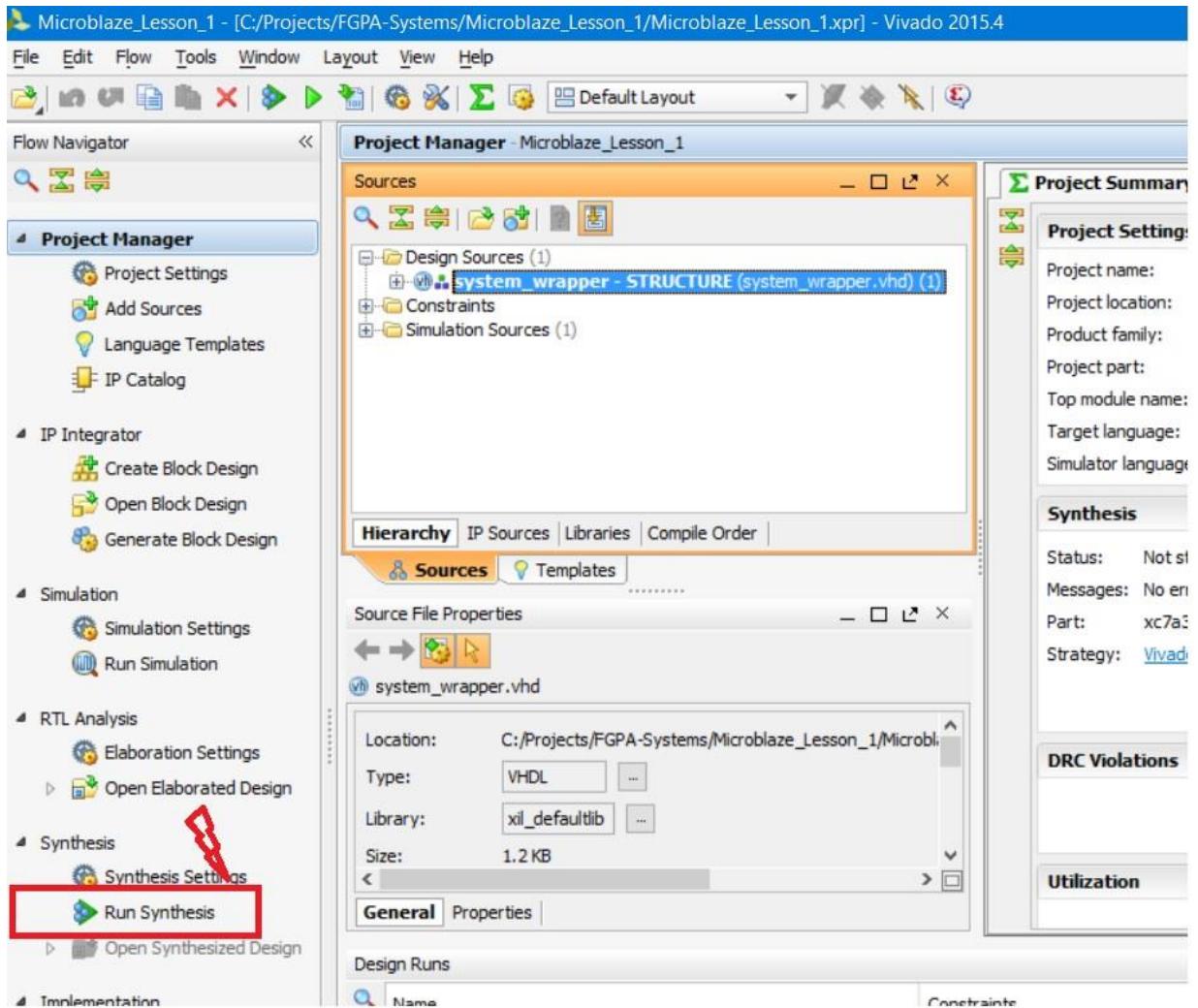


Рисунок 40 Запуск синтеза проекта

После окончания синтеза появится окно выбора действия (рис. 41):

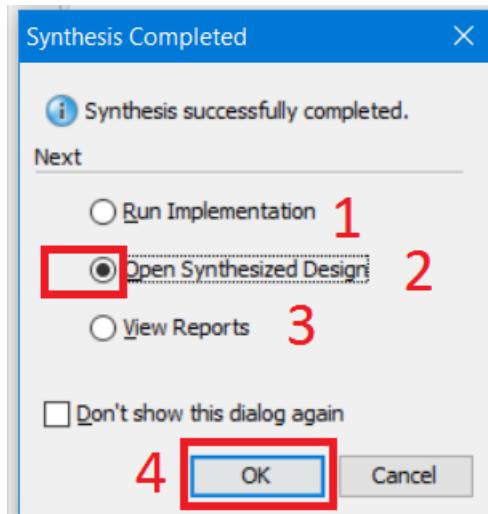


Рисунок 41 Действие после синтеза

Окно, появляющееся после синтеза, предлагает выполнить:

1. Запустить имплементацию.
2. Открыть синтезируемый проект (Выберите этот пункт).
3. Просмотреть отчёты.

Выберите второй пункт – открыть синтезированный проект. Если Вы машинально закрыли окно на рис. 41, то можно открыть синтезированный проект другим способом – нажав на кнопку Open Synthesized Design, которая находится под кнопкой запуска синтеза (рис. 42). Обратите внимание, что кнопка Open Synthesized Design активна, если только у Вас есть результат синтеза; если проект не синтезирован, кнопка будет неактивна (см. рис. 40, где результатов синтеза ещё нет)

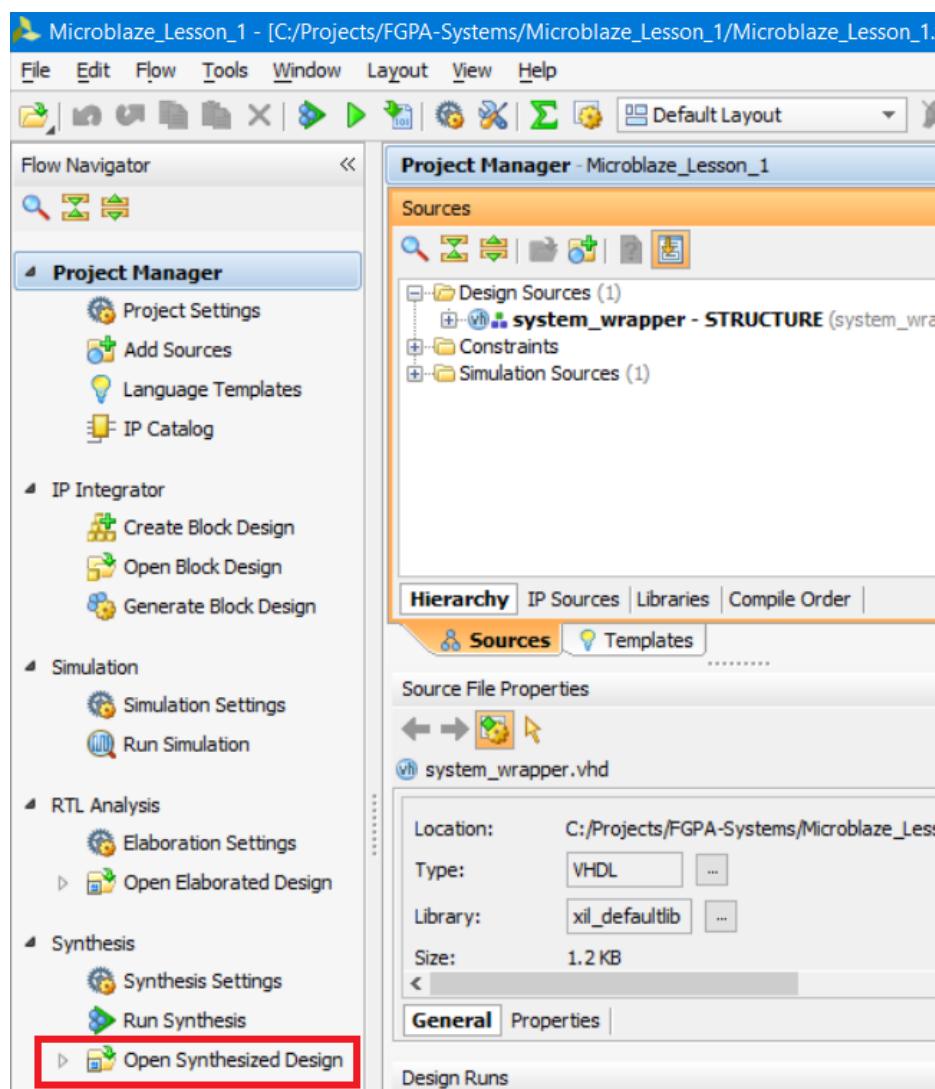


Рисунок 42 Кнопка открытия синтезированного проекта

Открытие синтезированного проекта нужно для двух вещей:

1. Проанализировать проект (если есть желание).
2. Выполнить назначение портам нашего проекта физических ножек FPGA.

До текущего момента физически мы ещё не указывали, к каким ножкам подключить нашу HW-часть. Сделаем это сейчас в модуле I/O Planning. Чтобы его запустить, необходимо при открытом уже синтезированном проекте выбрать в меню Layout пункт I/O Planning (рис. 43).

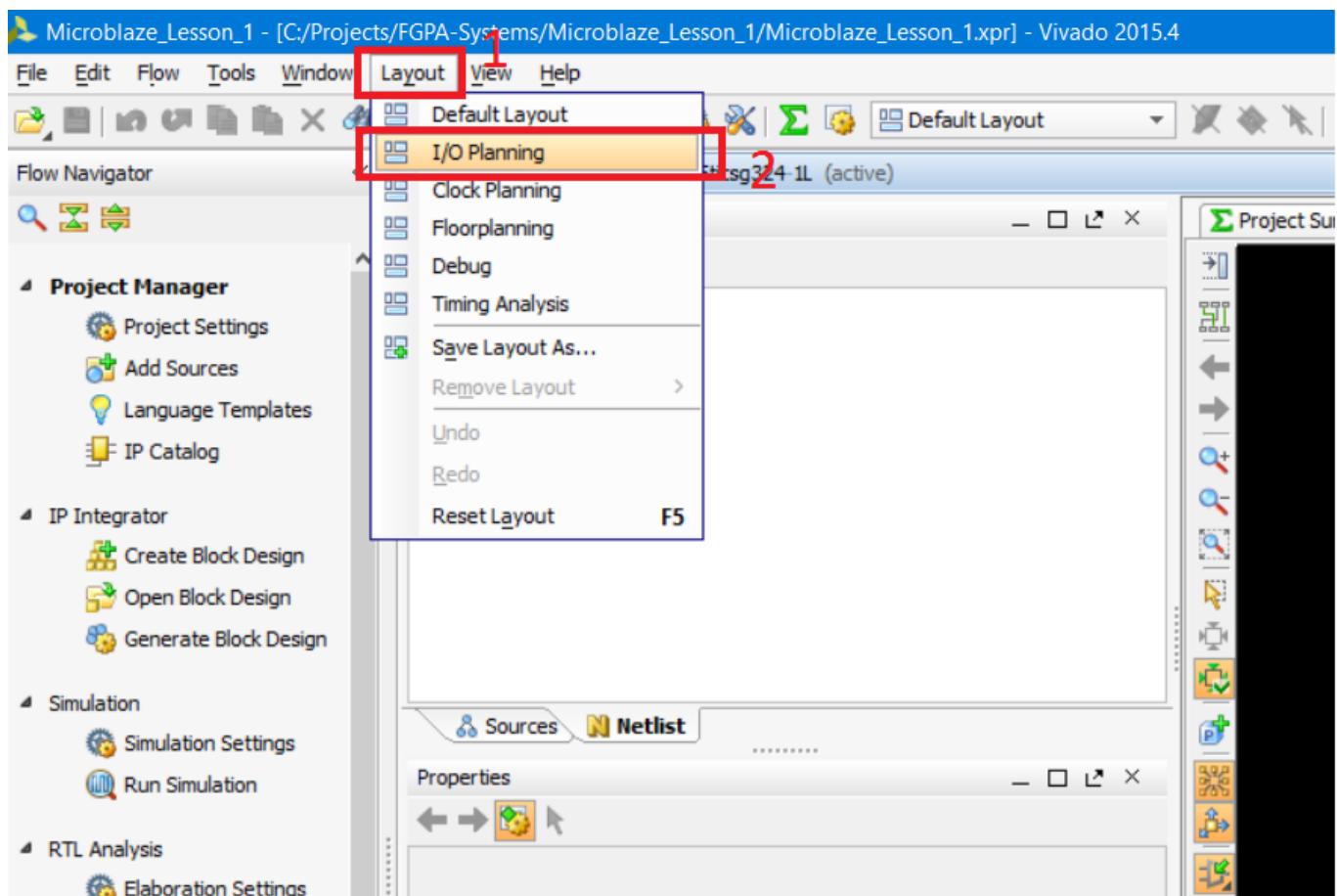


Рисунок 43 Открытие модуля планировщика выводов FPGA (обязательно должен быть открыт синтезированный проект, в противном случае планировщик может быть недоступен).

Если всё сделано корректно, то должен открыться планировщик выводов (рис. 44).

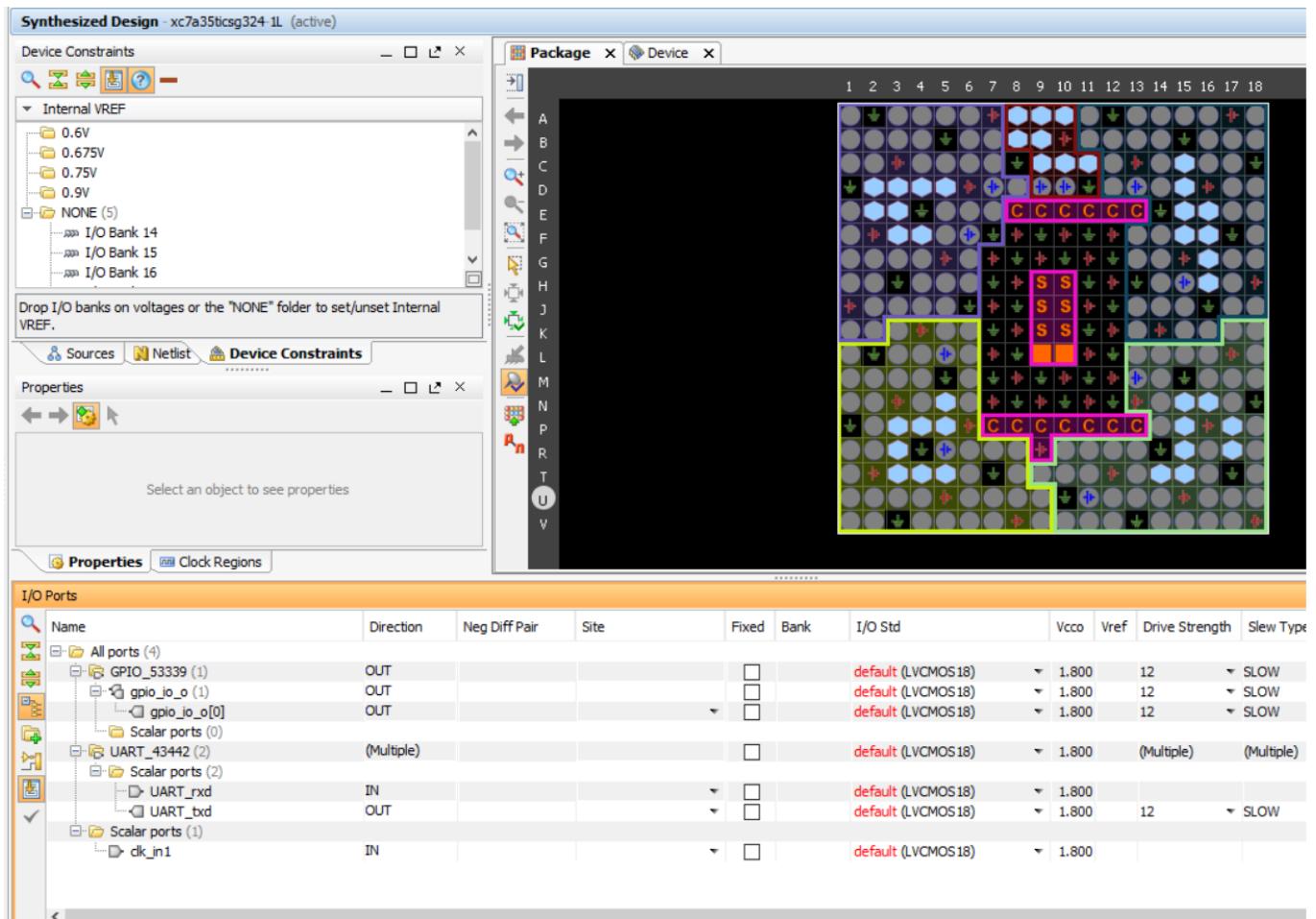


Рисунок 44 Вид окна планировщика выводов с раскрытым списком портов модуля верхнего уровня нашего проекта

Теперь мы можем выполнить назначение физических ножек FPGA portам нашей процессорной системы. Сделать это можно многими способами. Один из них – явно заполнить необходимые поля. Нас будут интересовать только поля Site – название контакта FPGA и I/O Std – стандарт ввода-вывода, определяющей напряжения логических уровней и другие электрические параметры. Обратите внимание, что если Ваша плата – не Arty Board, то содержимое этих двух колонок может быть другим; надеюсь, Вы сможете прочитать принципиальную электрическую схему Вашей платы и указать во всех полях правильные значения. Для Arty Board они показаны на рис. 45.

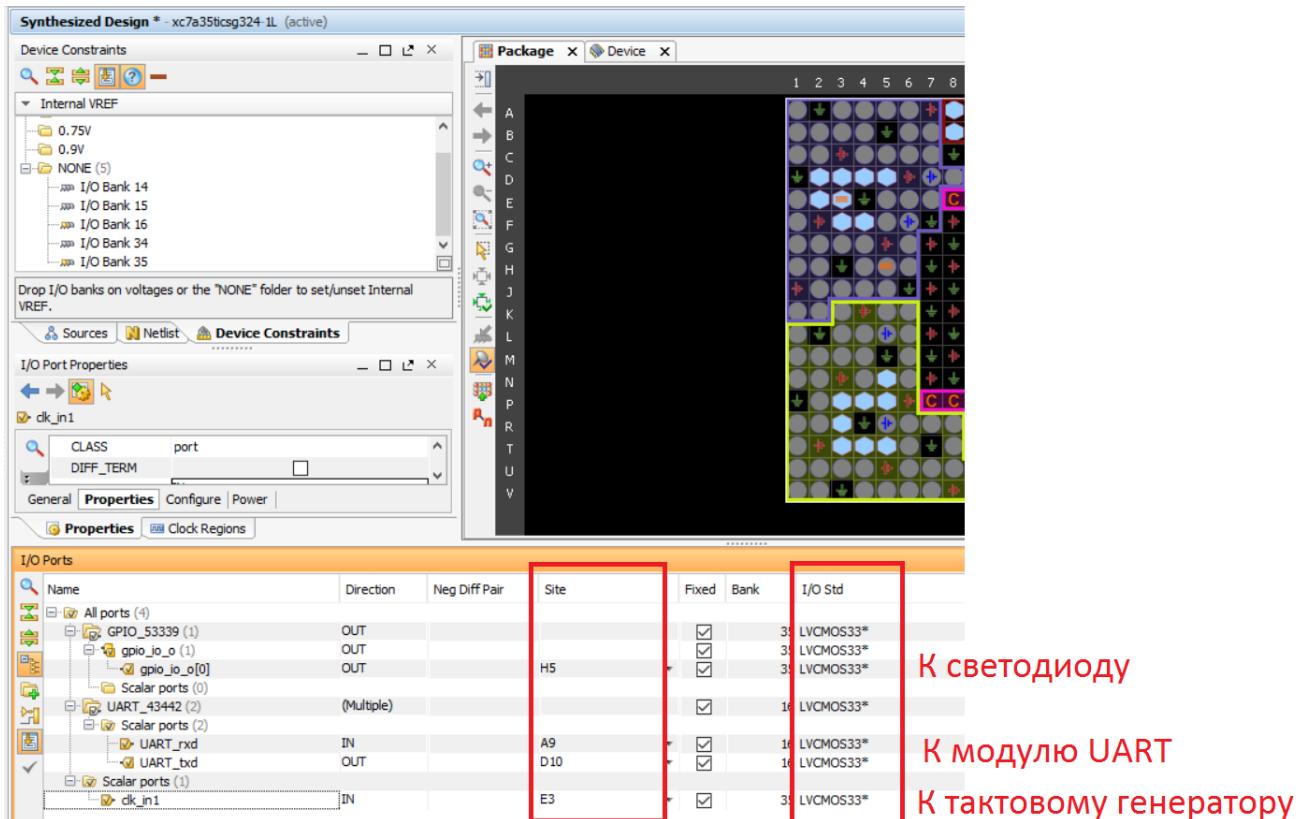


Рисунок 45 Подключение портов к ножкам FPGA для Arty Board

Обратите внимание, что значки портов после выполнения назначений стали жёлтыми, а не серыми.

Сохраним результаты назначения, нажав кнопку сохранить (рис. 46). После нажатия на неё появится окно, которое говорит, что сохранение возможно приведёт к необходимости повторного синтеза. Нажимаем OK.

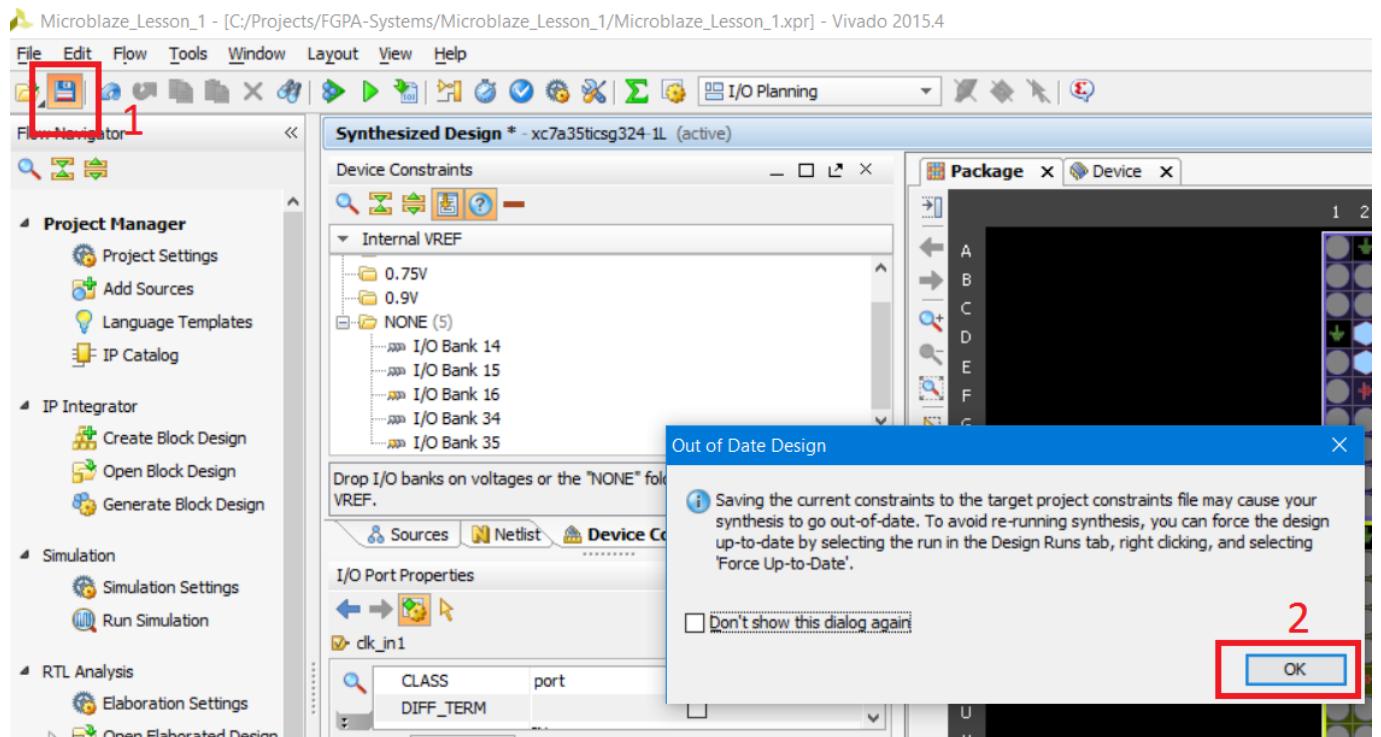


Рисунок 46 Сохранение в планировщике выводов произведённых изменений.

Сохранение параметров назначений ножек производится обычно в файл формата (и расширения) xdc – Xilinx Design Constraints. Его мы ешё не создавали, поэтому Vivado предлагает нам создать новый файл для сохранения (рис. 47).

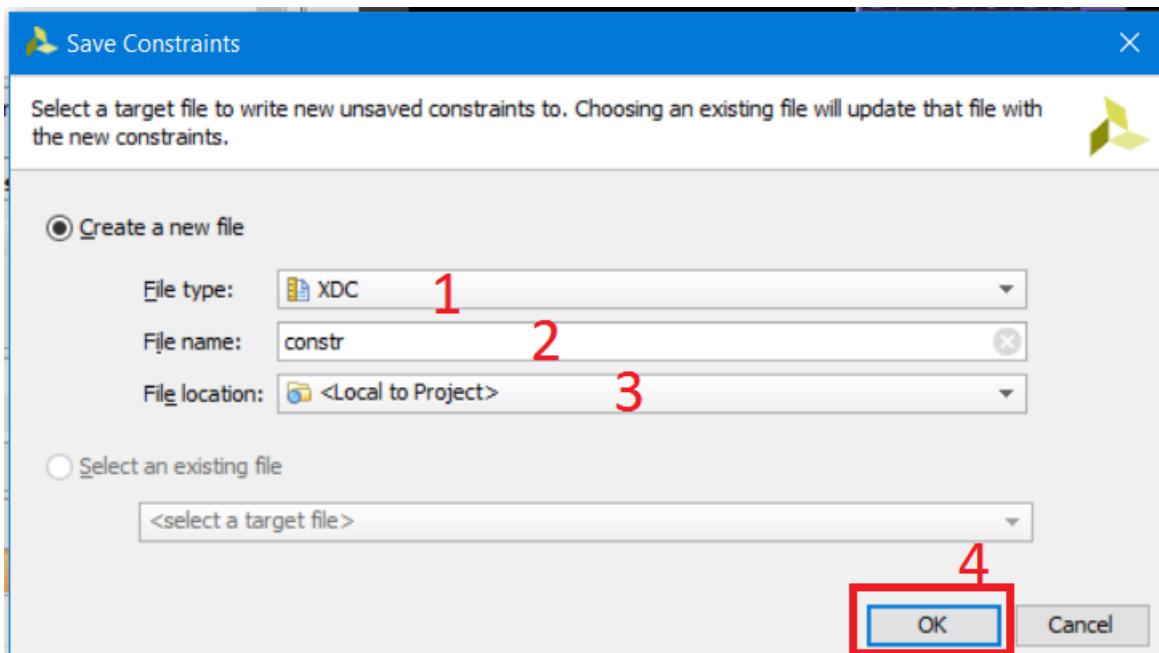


Рисунок 47 Окно создания файла для сохранения результата

В окне, показанном на рис. 47:

1. Тип файла. Выберите XDC – Xilinx Design Constraint – файл с настройками для проекта
2. Имя файла; наберите constr.
3. Расположение файла – оставьте по умолчанию.
4. Нажмите OK.

После сохранения результатов необходимо выполнить синтез проекта заново, о чём говорит надпись Synthesis Out-of-Date в правом верхнем углу (рис. 47).

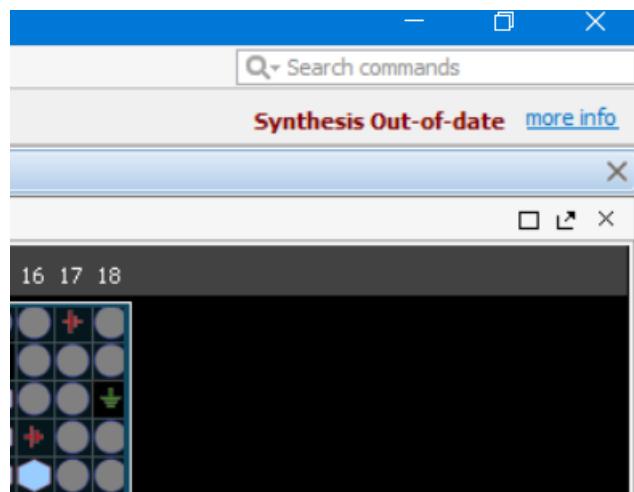


Рисунок 48 Результаты синтеза устарели, синтез необходимо выполнить заново: Synthesis Out-of-date

Мы позволим Vivado заново выполнить синтез, но в этот раз запустим не просто синтез, а сразу генерацию битстрима – bit-файла. При этом Vivado «увидит», что результатов синтеза и имплементации нет, и предложит выполнить их в автоматическом режиме, прежде чем сгенерировать bit-файл (рис. 49).

Нажмите Generate Bitstream и затем, когда Vivado предложит выполнить синтез и имплементацию перед запуском генерации bit-файла – нажмите Yes.

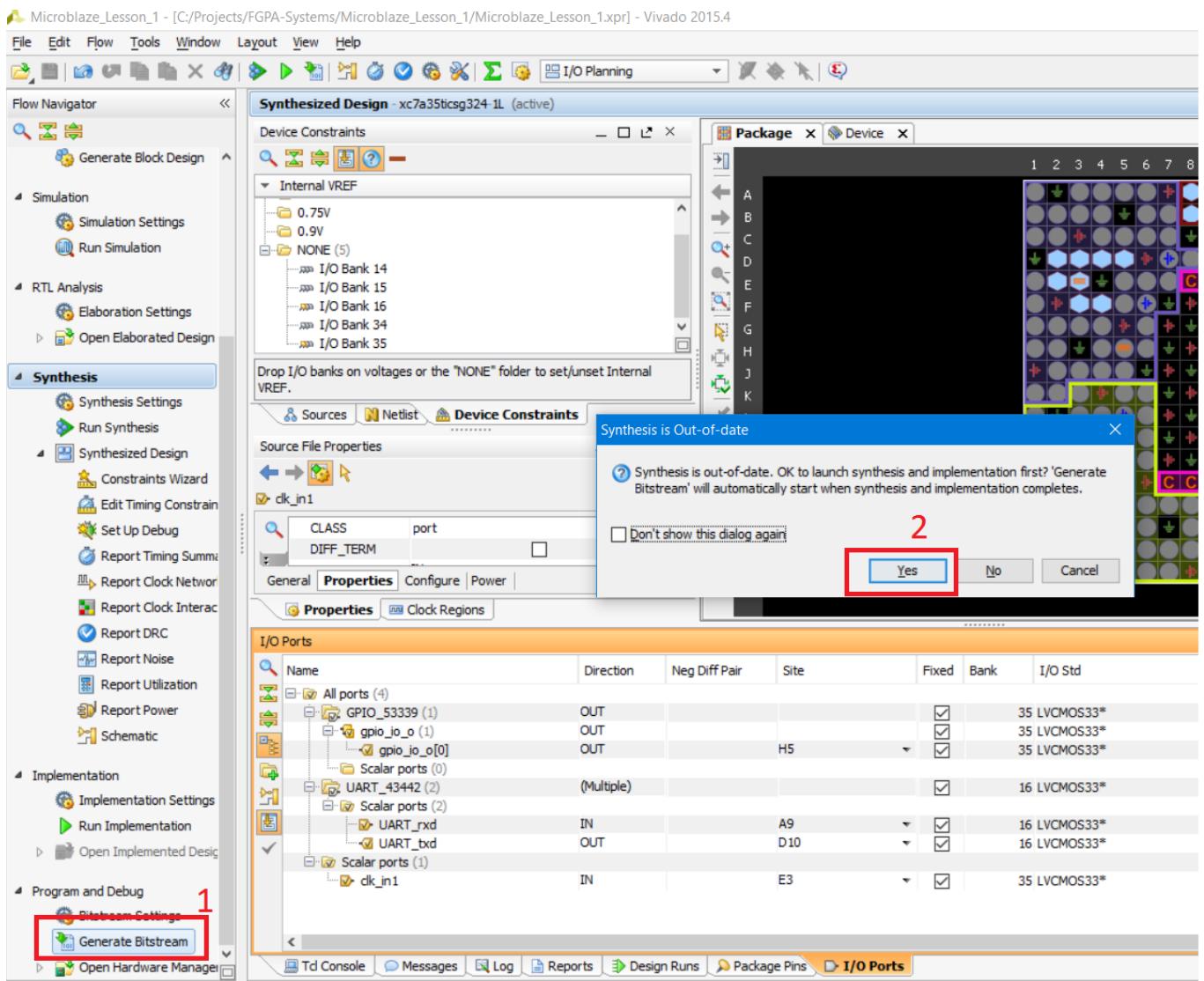


Рисунок 49 Запуск генерации bit-файла и окно, предупреждающее что сначала будут выполнены синтез и имплементация

Теперь нам нужно ждать окончания генерации bit-файла. Это может занять минут 10-15 в зависимости от Вашего компьютера. По окончании генерации bit-файла появится окно (рис. 50), предлагающее выполнить одно из действий. Нам ничего дальше делать не нужно, поэтому его просто закрываем.

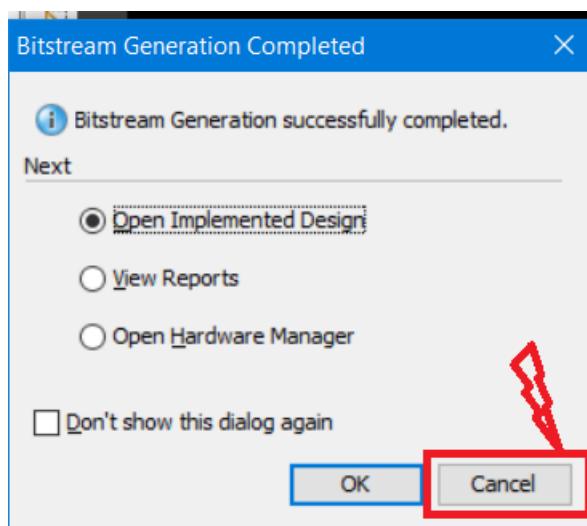


Рисунок 50 Окно завершения генерации bit-файла. Мы делать дальше ничего не будем, поэтому нажимаем кнопку Cancel.

На этом сборка HW-части закончена... Переходим к SW.

Разработка SW части

SW-часть (программная) необходима, чтобы «оживить» нашу собранную процессорную систему. Сейчас это просто кусок «железа», который не выполняет никаких действий. Как уже было сказано выше, разработка программной части выполняется в среде Xilinx SDK, которая есть по сути Eclipse с плагинами от Xilinx. Разумеется, что Xilinx уже автоматизировал часть процесса написания программы и подготовил некоторые исходные файлы и библиотеки, так что писать мы будем не «с нуля». Сейчас нам необходимо передать Xilinx SDK информацию об аппаратной «начинке» нашей процессорной системы: какие использованы устройства, какова их конфигурация и адреса и т.д. В общем – выполнить экспорт нашей аппаратной (HW) части. Сделать это можно, выбрав в левом верхнем углу File-Export-Hardware (рис. 51)

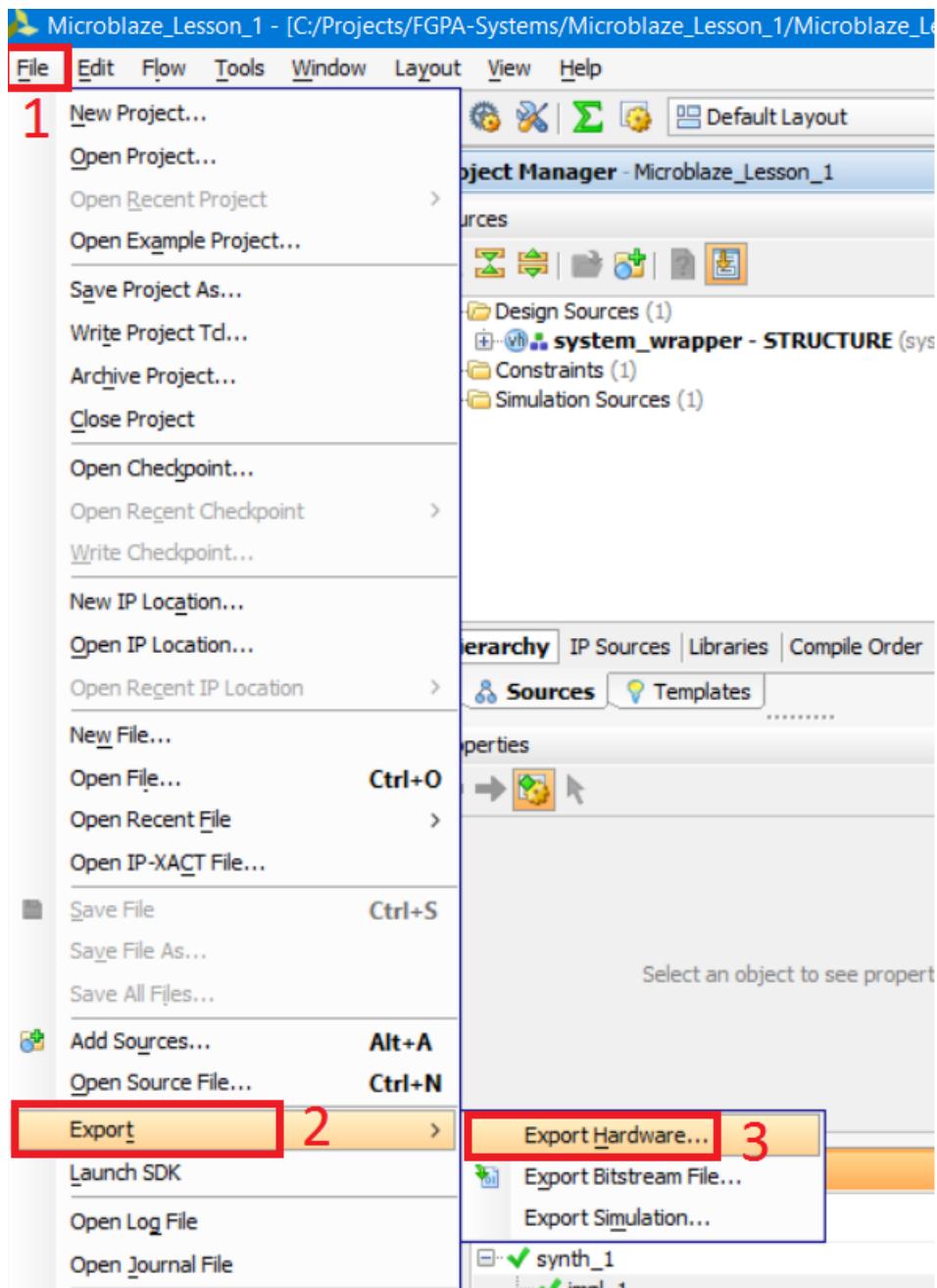


Рисунок 51 Экспорт HW-части проекта в Xilinx SDK

После этого появится окно (рис. 52), в котором указывается, нужно ли экспортировать bit-файл, и какую папку выбрать в качестве рабочей. Оставим все настройки в состояниях по умолчанию; bit-файл экспортировать сейчас не надо, мы добавим его позже. Нажимаем OK.

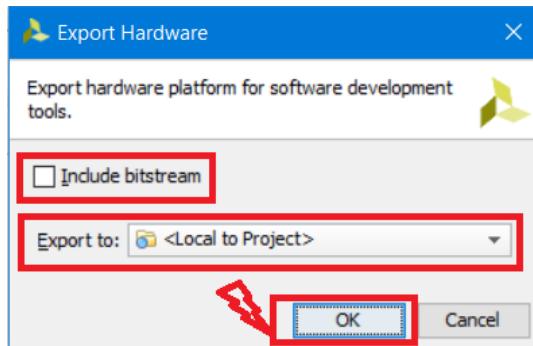


Рисунок 52 Параметры экспорта HW части

Теперь запускаем программу Xilinx SDK. Сделать это можно через главное меню Windows или из-под Vivado – просто выберите в левом верхнем углу File-Launch SDK (рис. 53).

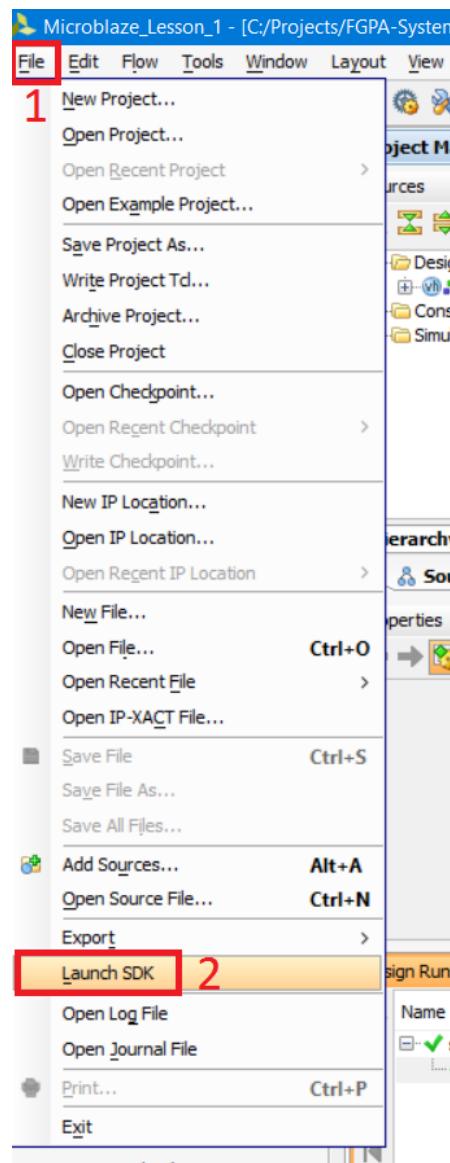


Рисунок 53 Запуск Xilinx SDK из-под Vivado

Теперь Vivado просит указать параметры запуска SDK; отставим их по умолчанию и нажмём OK (рис. 54):

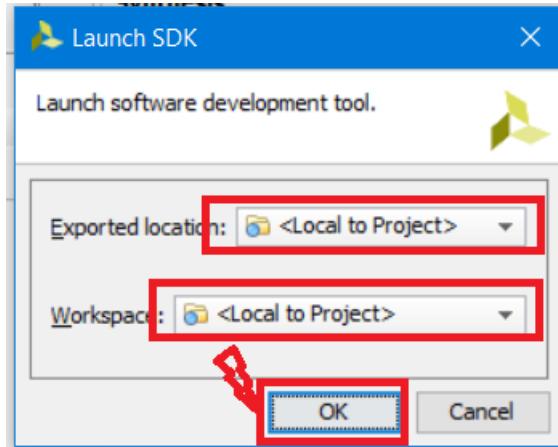


Рисунок 54 Параметры запуска Xilinx SDK

Если все сделано правильно, должно появиться окно, как на рис.55. Это и есть Xilinx SDK.

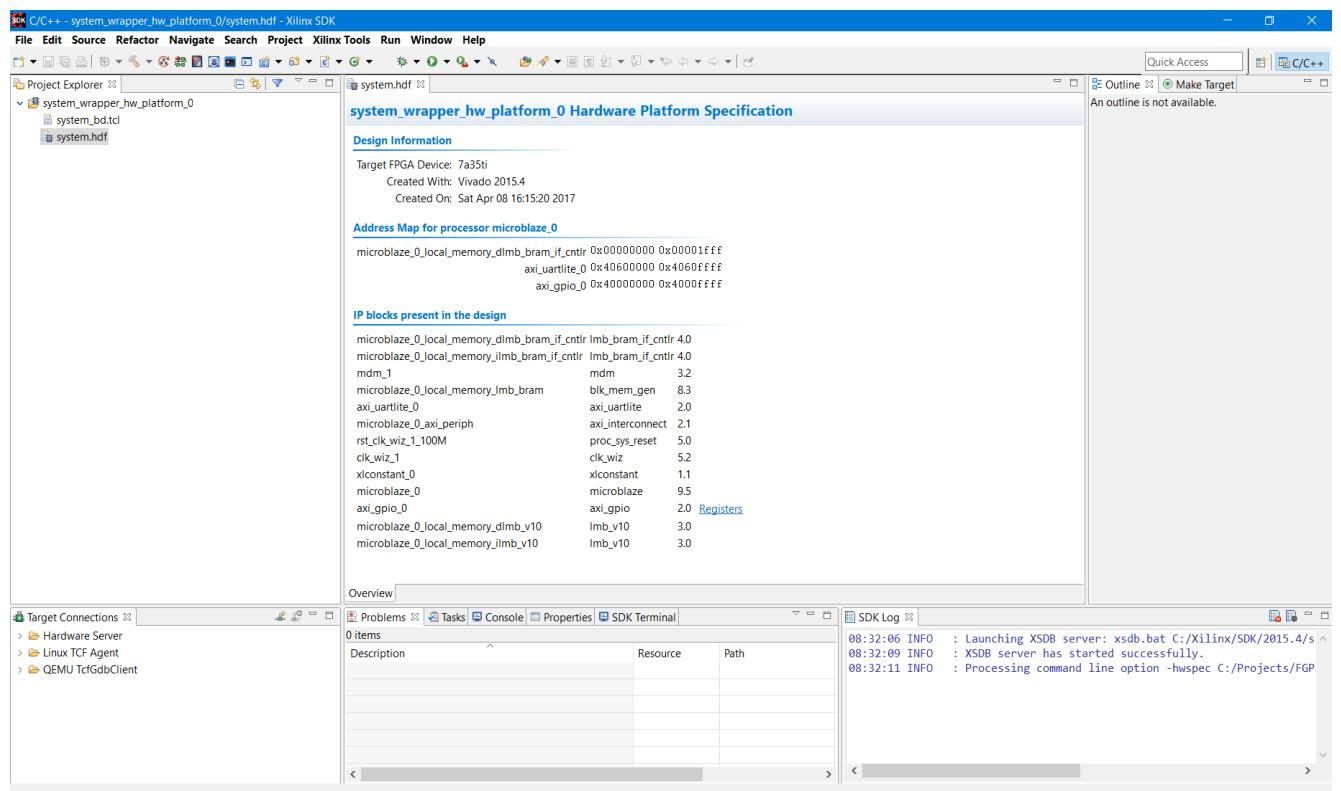


Рисунок 55 Основное окно Xilinx SDK

Сейчас мы не будем разбирать назначение каждого из внутренних окон системы, поскольку это выходит за рамки данной статьи. Единственное, о чём я

могу упомянуть – так это о том, что у Xilinx SDK очень-очень-очень хорошая справка (рис. 56). Если у Вас возникают вопросы, в основном все ответы лежат в справке. Не поленитесь, ознакомиться с ней, если будет время.

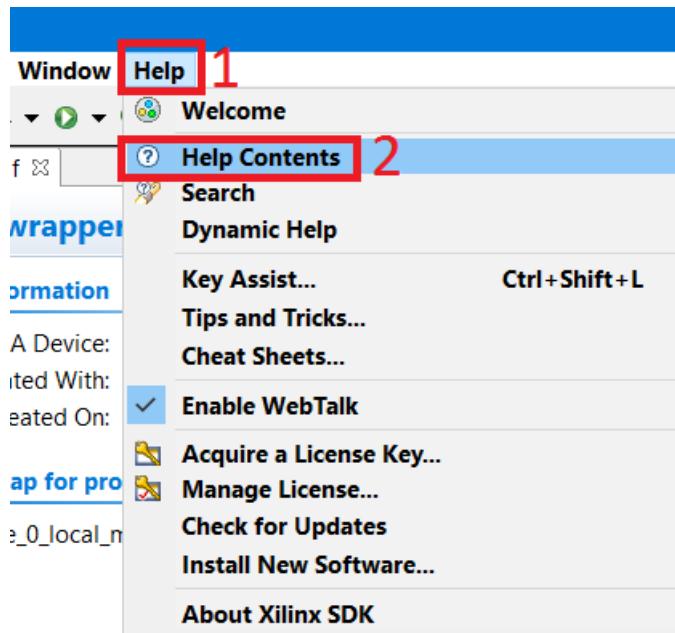


Рисунок 56 Вызов справки Xilinx SDK

Приступим к созданию проекта. Выбираем File-New-Application project

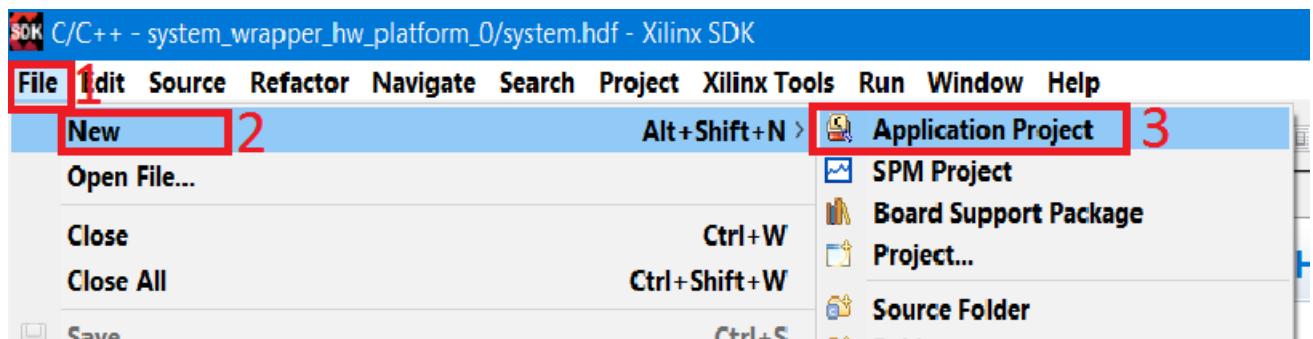


Рисунок 57 Создание нового проекта в Xilinx SDK

Задаём основные параметры проекта (рис. 58):

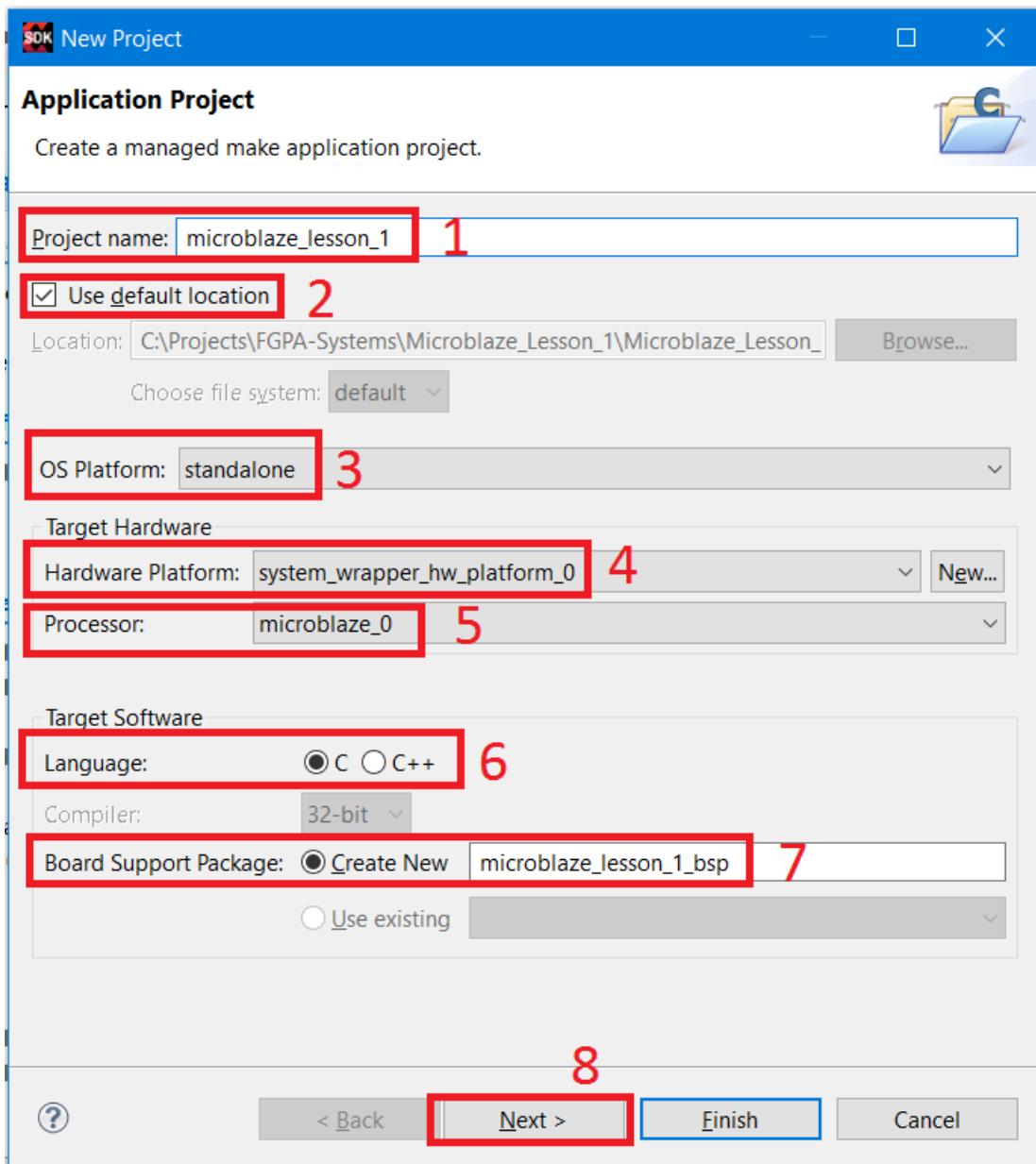


Рисунок 58 Параметры нового проекта

На рис. 58:

1. Название проекта.
2. Место расположения проекта (выставляем используемое по умолчанию).
3. Тип используемой в проекте операционной системы (без ОС – standalone).
4. Аппаратная конфигурация (если процессоров системе несколько, то будет список; у нас – только один).

5. Идентификатор процессора (актуально для многопроцессорных систем).
6. Язык программирования (чистый C).
7. Пакет драйверов (создаём новый на основании информации о HW-части).
8. Нажимаем Next.

Теперь SDK предлагает выбрать один из нескольких готовых вариантов приложения (application), включая Hello, world (рис. 59). Мы же с Вами, для более подробного знакомства со структурой приложения, выберем пустое приложение (Empty Application), нажимаем Finish.

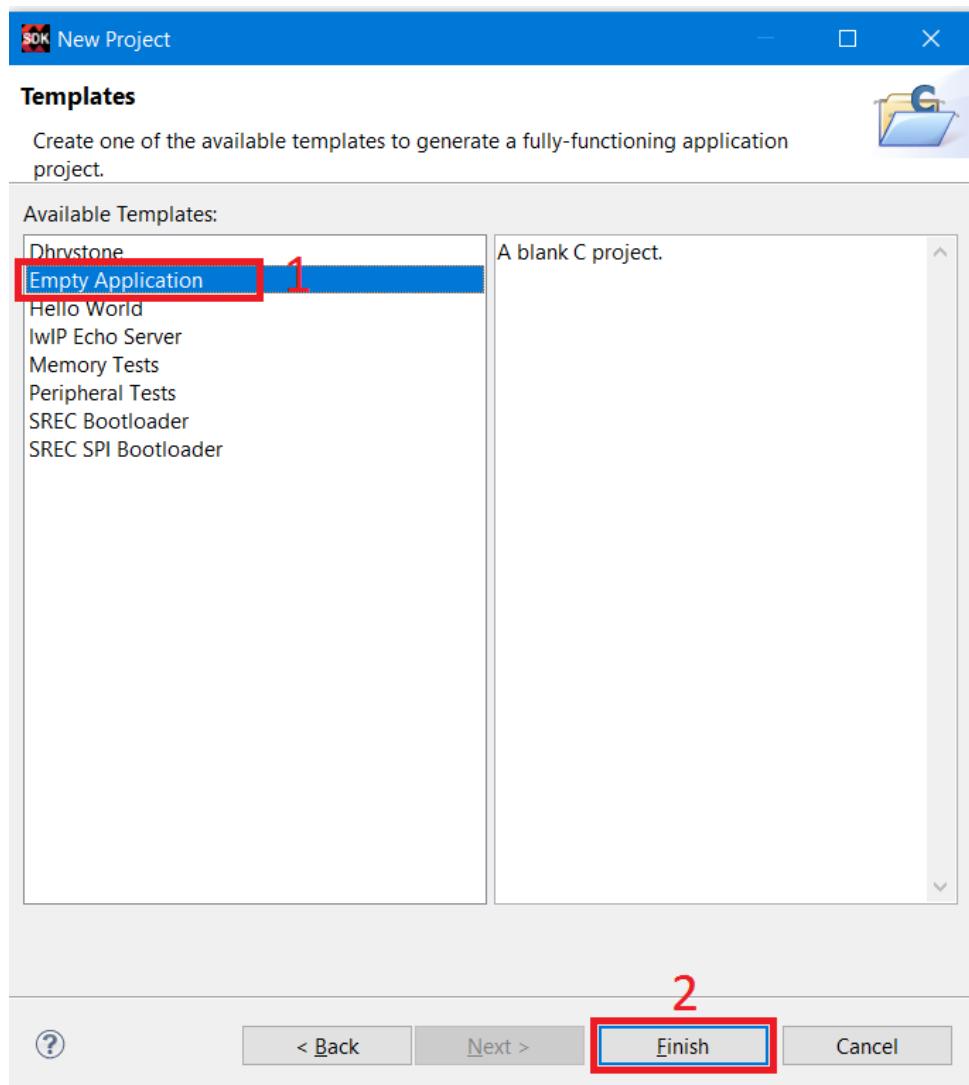


Рисунок 59 Выбор типа проекта – пустой проект

После нажатия на кнопку Finish в проект добавляется несколько файлов и папок (рис. 60).

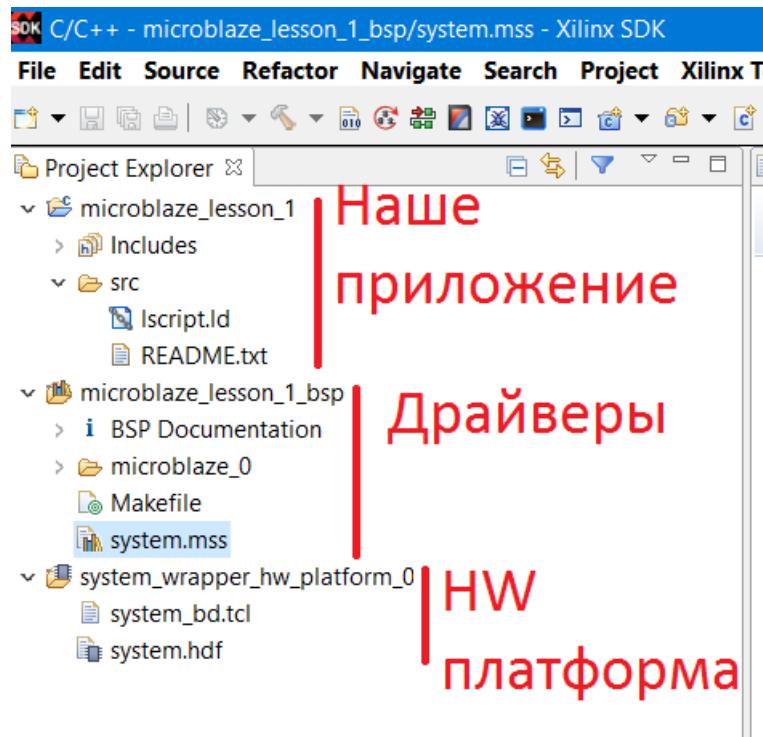


Рисунок 60 После создания проекта добавляются приложение и драйверы.

В нашем приложении ещё нет файла с исполняемой программой. Нам нужно создать его и добавить в проект. Поскольку качестве языка программирования был выбран С, то нужно добавить файл с расширением «.с». Для добавления нового файла в проект нажмите на значку нашего приложения правой кнопкой и выберите New – File (рис. 61).

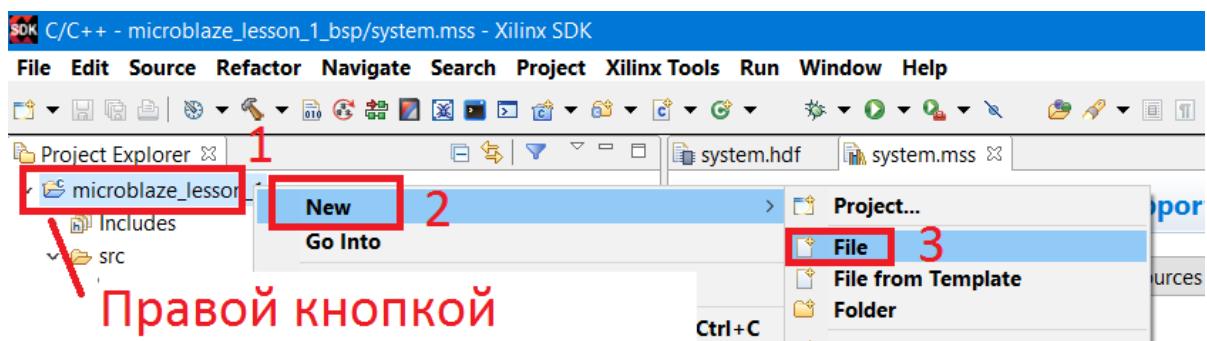


Рисунок 61 Добавление нового файла в приложение

Теперь нужно указать параметры создаваемого файла (рис. 62). Путь к папке, в которой он будет создан, можно указать в строке 1, или же в дереве папок выбрать папку, в которой он будет создан

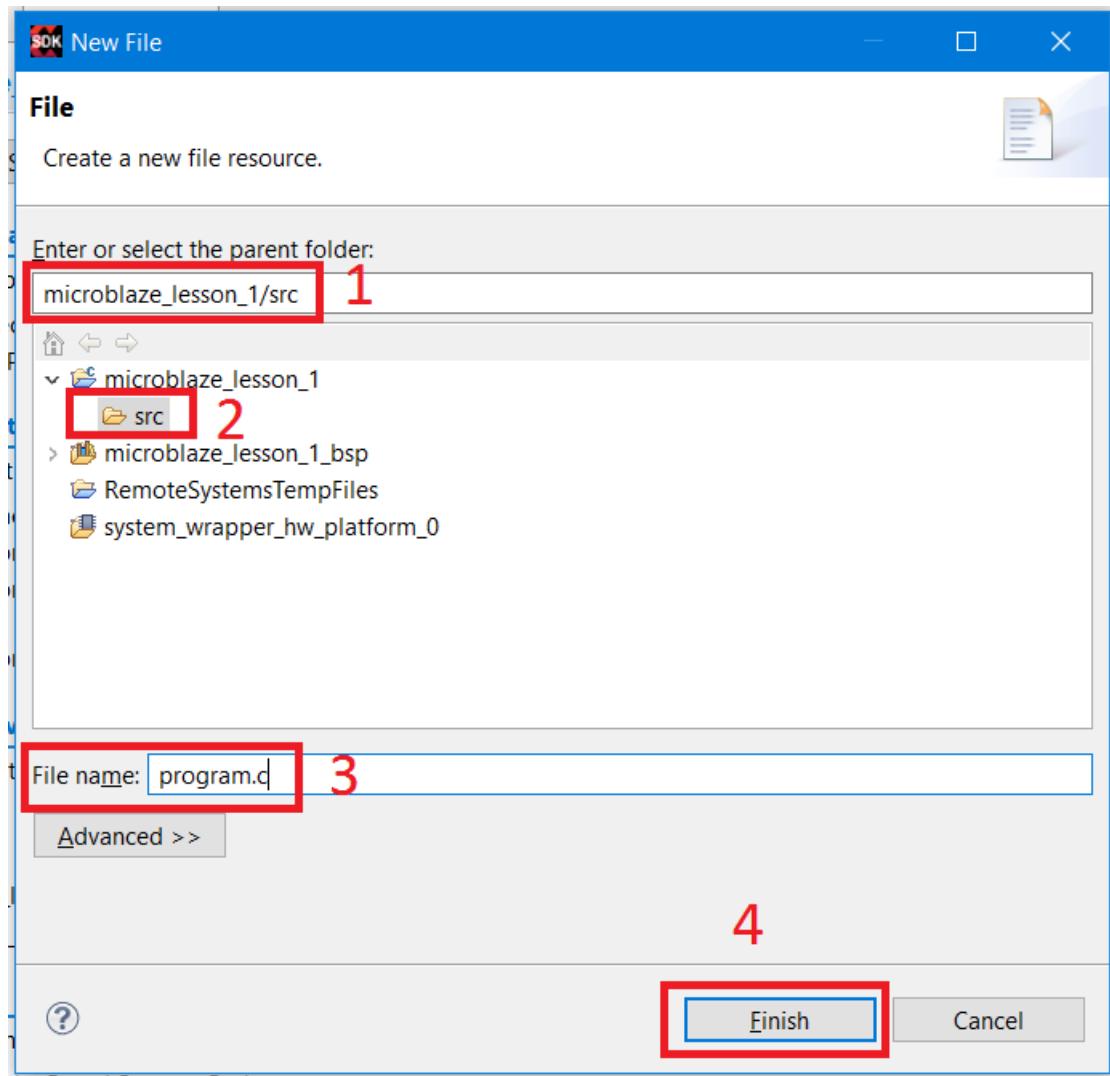


Рисунок 62 Создание и добавление нового файла в проект.

На рис. 62:

1. Путь к файлу относительно каталога приложения.
2. Путь к файлу в дереве директорий.
3. Указываем название файла (program.c).
4. Нажмите Finish.

Созданный файл появится в папке src (рис. 63). Чтобы просмотреть его содержимое, дважды кликните по нему. После создания файла появилась ошибка, о чем говорит красный крестик в папке приложения. Дело в том, что файл – пустой, и в приложении нет функции main(). Нам необходимо написать программу в наш файл.

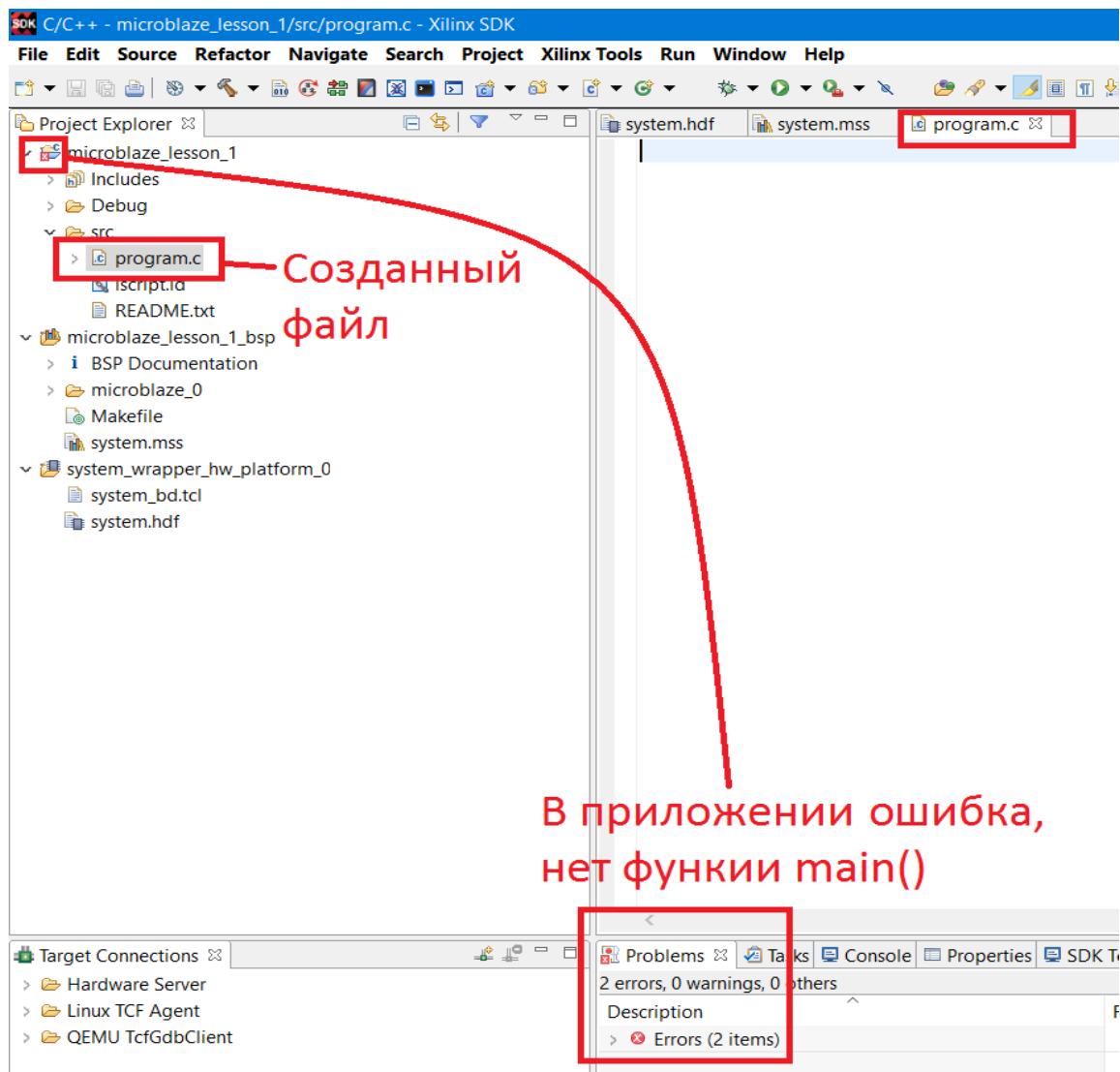


Рисунок 63 Ошибка из-за отсутствия функции main()

Откройте файл program.c и заполните его следующим кодом:

```
#include "xparameters.h" //Библиотека с параметрами IP-блоков
#include "xgpio.h" //Библиотека с функциями GPIO

XGpio gpio; //Создаем "программную" модель GPIO

int main(){

    u32 i = 0; //используем для задержки
    u32 led = 0; //состояние светодиода

    XGpio_Initialize(&gpio, XPAR_GPIO_0_DEVICE_ID); //Находим и инициализируем GPIO

    xil_printf("Hello, world!!!\n\r"); //Автоматическицепляется Uartlite и выводит
сообщение

    while(1){ //Бесконечный цикл мигания
        i++; //увеличиваем счётчик
        if(i == 1000000){ //Если достигнуто значение 1_000_000
            led = !led; //Инвертируем состояние светодиода
            XGpio_DiscreteWrite(&gpio, 1, led); //Записываем состояние светодиода в
GPIO
            i = 0; //Сбрасываем счётчик
        }
    }
    return 0;
}
```

Листинг 1.

Файл xparameters.h является одним из наиболее важных файлов, который содержит адреса и номера устройств на шине нашей процессорной системы. Именно в нём определён номер (ID) нашего GPIO - XPAR_GPIO_0_DEVICE_ID. Если бы GPIO модулей было бы несколько, то для каждого из них было бы отдельное описание параметров адреса, ID и т.д. Файл xparameters.h можно найти (рис. 64) в Project Explorer – название_приложения_bsp – процессор – include

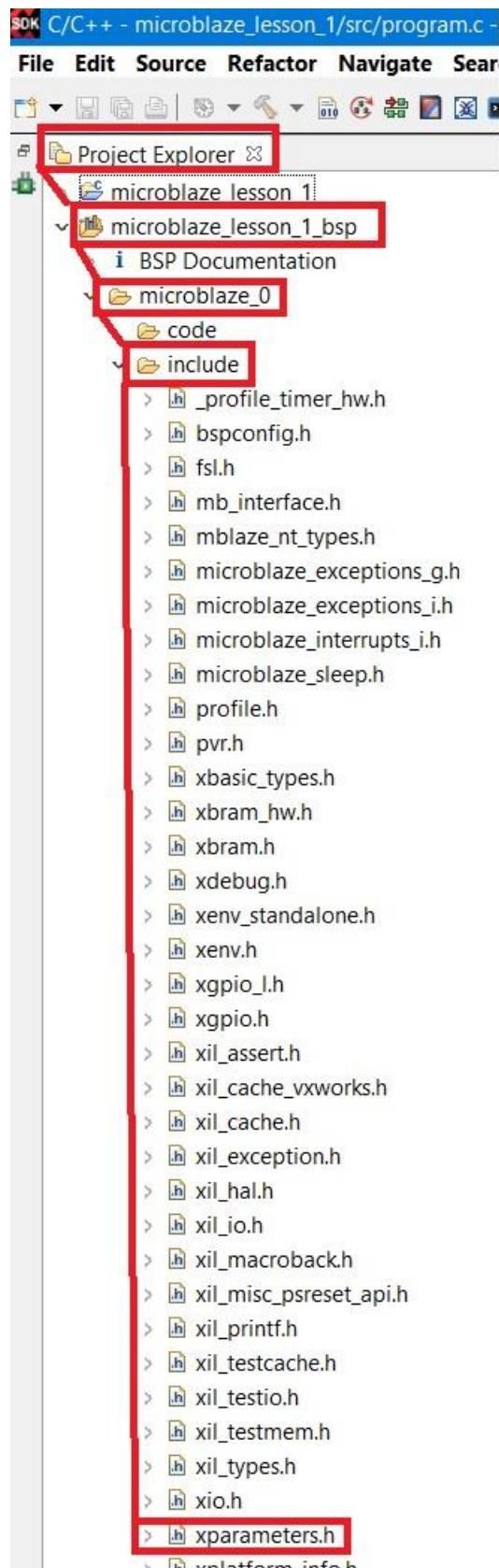


Рисунок 64 Расположение файла xparameters.h

Мы создали программу, теперь можем её попробовать запустить. Мы не будем отлаживаться в пошаговом режиме, а сразу посмотрим результат. Но для начала нам необходимо подключить плату к компьютеру, что бы он определил какой СОМ порт будет задействован для UARTa.

Для просмотра сообщений от UART мы будем использовать SDK terminal. Чтобы открыть SDK terminal, необходимо найти его в каталоге доступных окон. Для этого выберите Window – Show View – Others..., затем наберите в строке поиска SDK terminal, выберите его и нажмите OK (рис. 65).

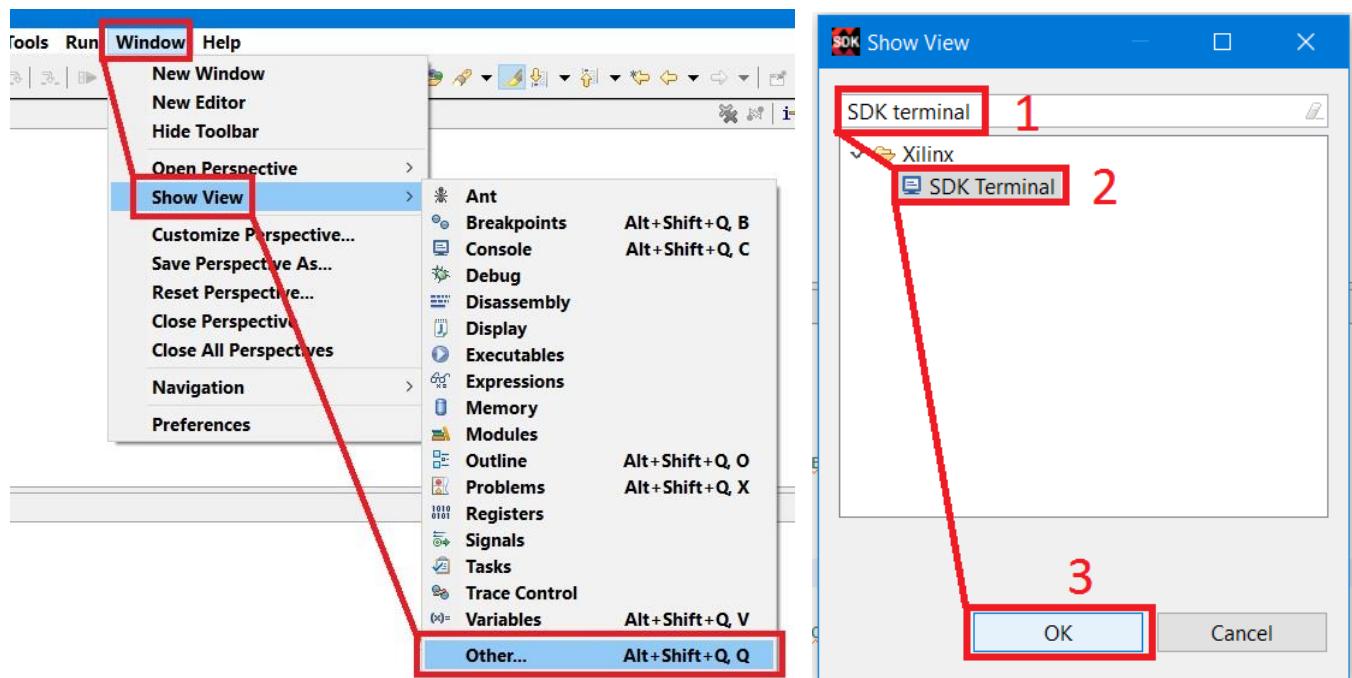


Рисунок 65 Открытие окна SDK terminal

После этого должно появиться окно SDK terminal, которое мы будем использовать, чтобы «ловить» сообщения, посылаемые нашим Uartlite модулем. Теперь давайте настроим SDK terminal (рис. 66). Выберите окно SDK terminal, нажмите кнопку подключения к последовательному порту (зелёный крестик) и установите значения в соответствии с рис. 66, затем нажмите OK.

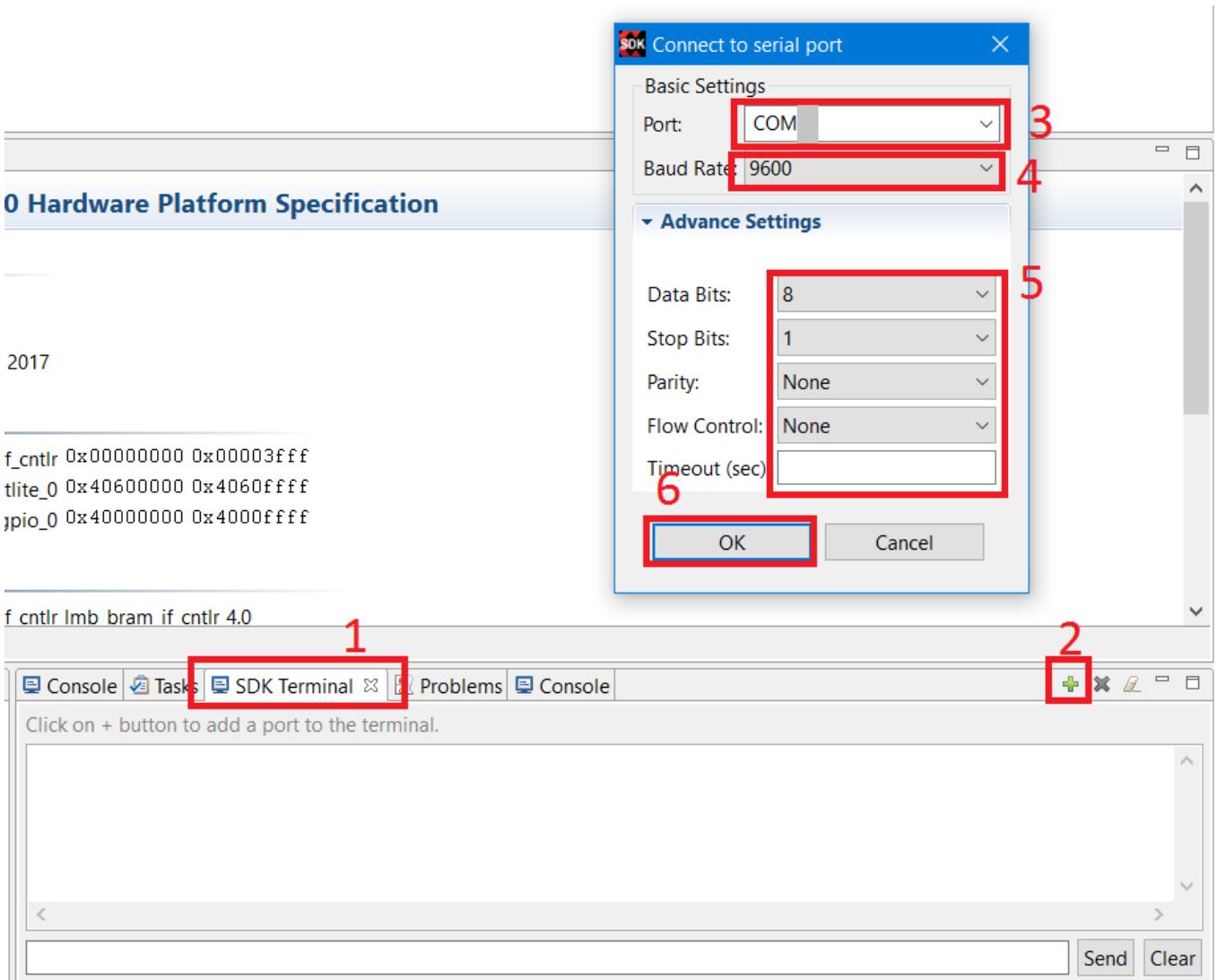


Рисунок 66 Настройка терминала SDK

Настройка терминала выполнена. По завершении настройки терминала, в нём может появиться «мусор». Если он Вам мешает нажмите кнопку очистить , которая находится рядом с кнопкой подключения к порту (зелёный крестик 2 на рис. 66).

Теперь необходимо настроить запуск нашей программы: прошить FPGA и загрузить приложение в процессор. Всё это делается в одном окне, но в разных вкладках. Для настройки запуска (рис. 67) нажмите на треугольник рядом с кнопкой Run

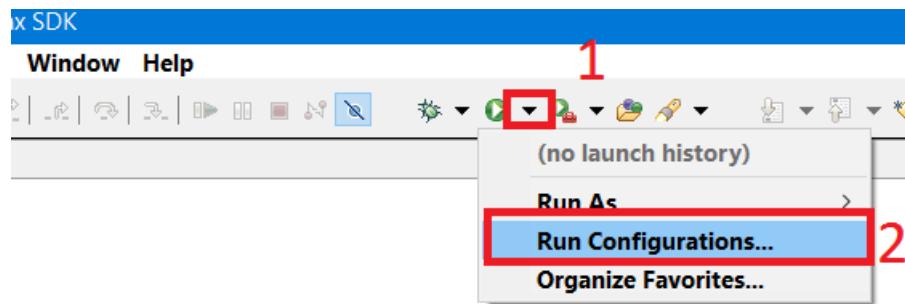


Рисунок 67 Вызов настроек запуска

Окно на рис. 68 предназначено для выбора отладчика или инструмента запуска настройки запуска процессорной системы. Дважды кликните на System Debugger.

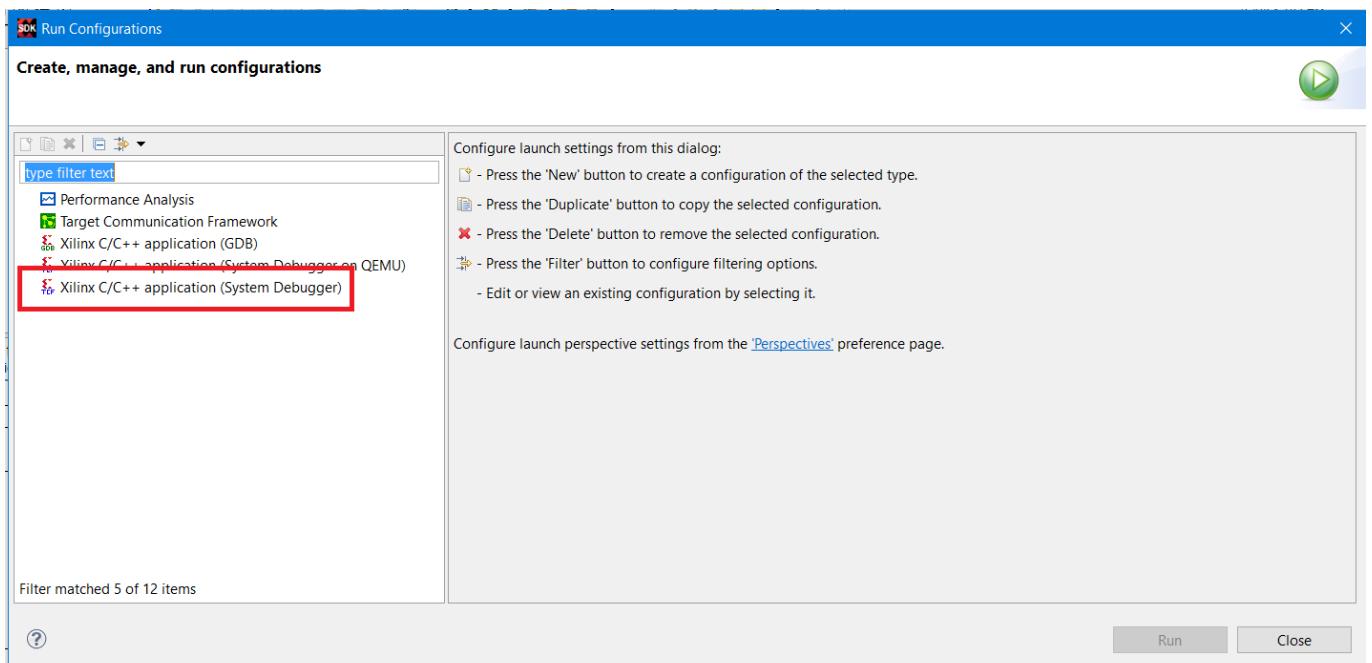


Рисунок 68 Окно выбора инструмента запуска

После этого окно должно приобрести вид, соответствующий рис. 69. Теперь можем приступить к настройке запуска.

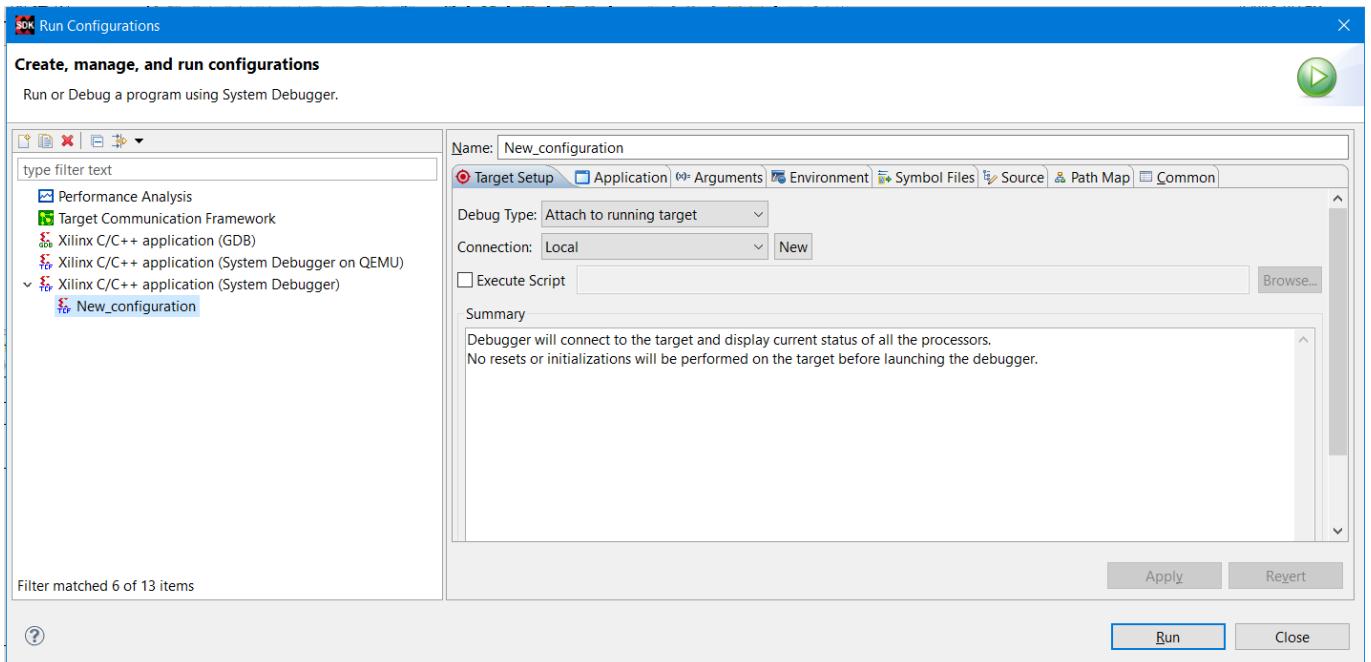


Рисунок 69 Исходное окно настроек запуска

Измените название наших настроек на system_run_1, затем выберите Debug Type: Standalone Application Debug (рис. 70), после чего окно должно измениться.

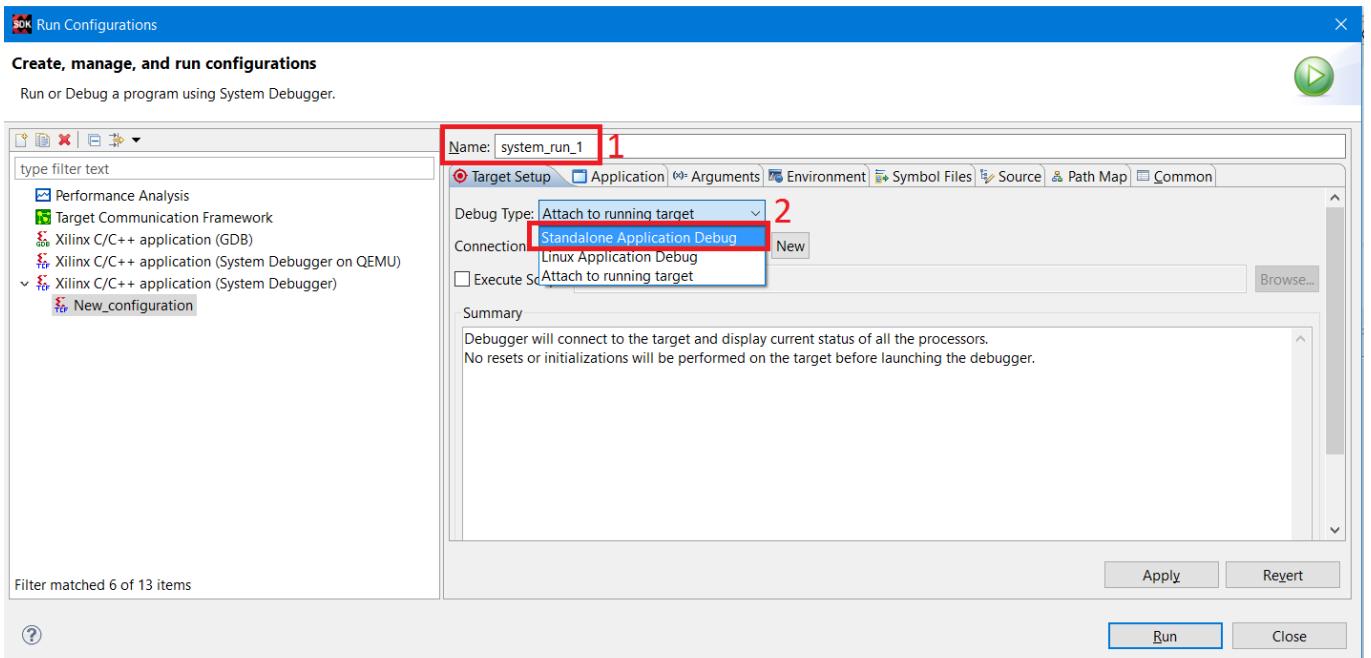


Рисунок 70 Переименование настроек и выбор типа отладки

После изменения, заполните содержимое в соответствии с рис. 71

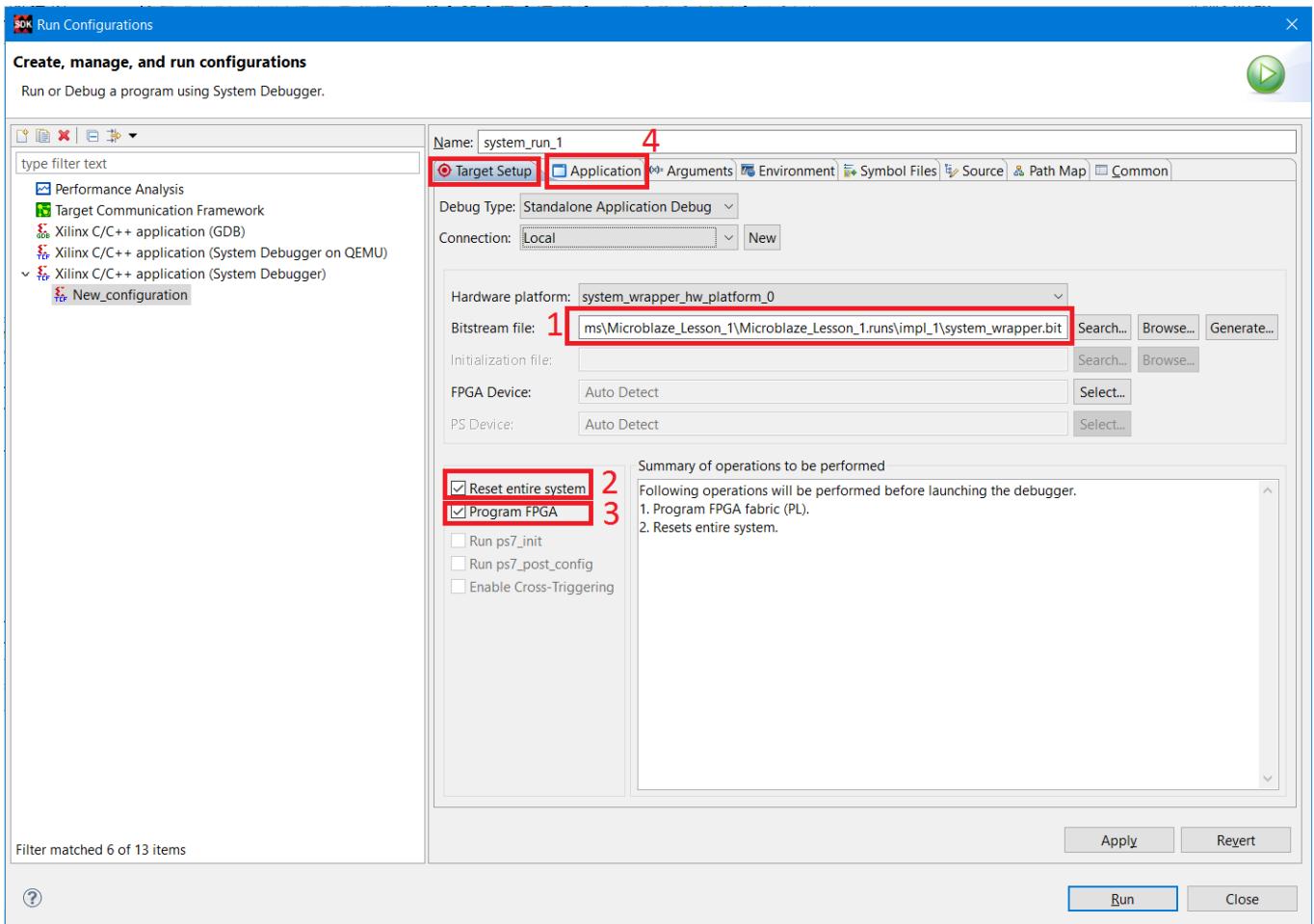


Рисунок 71 Настройка HW параметров запуска

На рис. 71 обозначены:

- 1 Путь к бит файлу. Должен быть указан полный путь. Находится внутри папки с проектом (название_проекта – название_проекта.runs – impl_1 – .bit)
- 2 Сброс системы после загрузки
- 3 Запрограммировать FPGA
- 4 Перейдите во вкладку Application

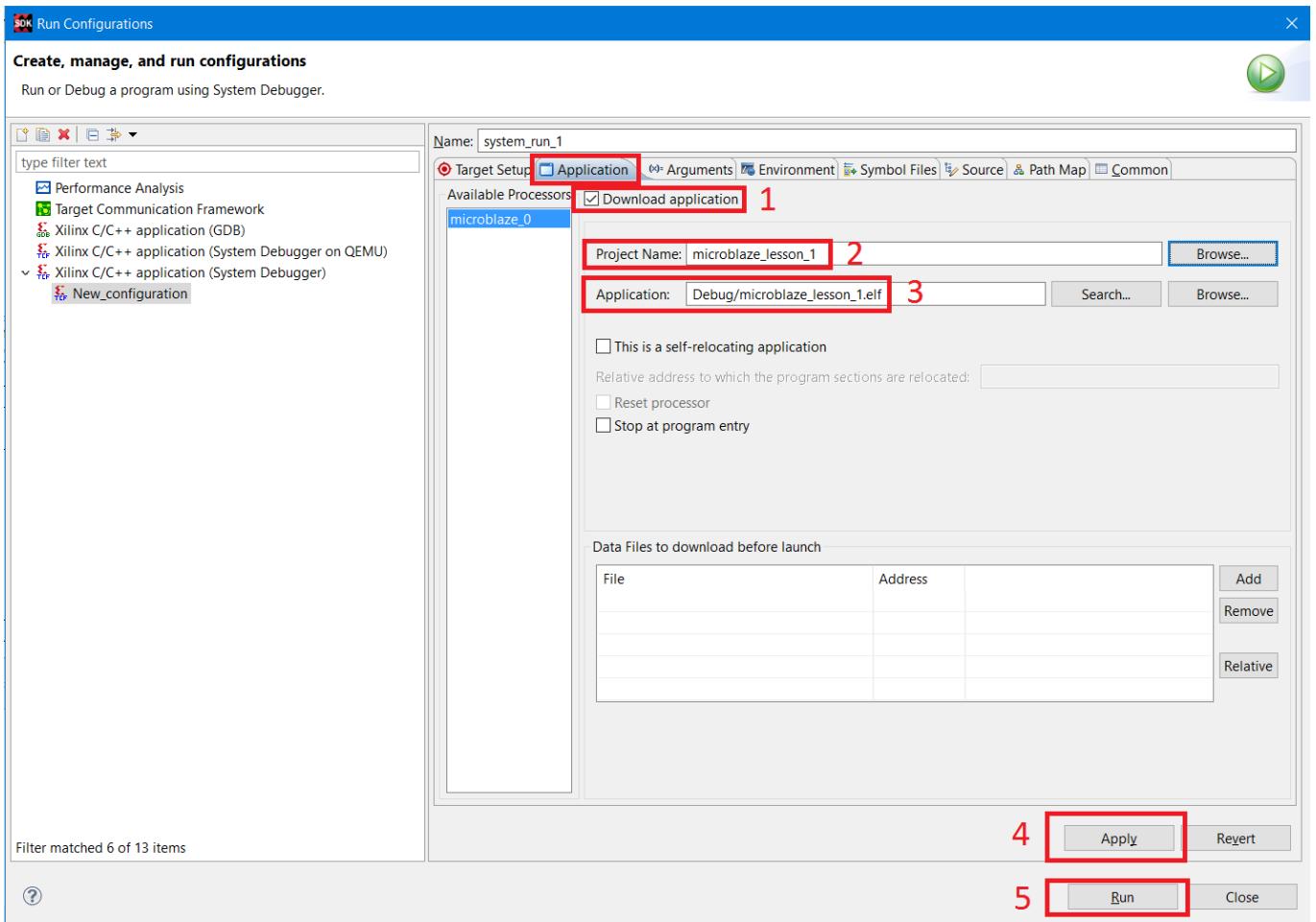


Рисунок 72 Настройка параметров запуска приложения

На рис. 72:

- 1 Загрузить приложение после загрузки FPGA
- 2 Название приложения, которое нужно загрузить
- 3 Путь к исполняемому .elf файлу приложения
- 4 Кнопка применить настройки
- 5 Запуск приложения

После нажатия кнопки Run произойдёт загрузка FPGA и запуск приложения, в результате чего светодиод LD4 на плате Arty Board должен начать мигать, а в SDK terminal должно появиться сообщение «Hello, world!», рис. 73.

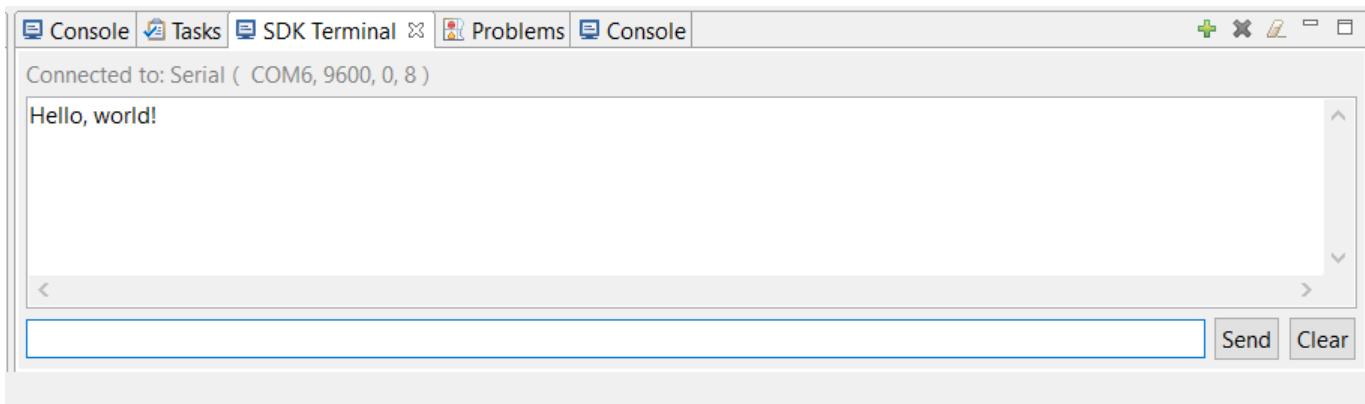


Рисунок 73 Hello, world! В SDK terminal

На этом сборка и запуск процессорной системы, построенной на базе MicroBlaze, закончена.

Заключение

Не смотря на то, что материал занимает более 60 страниц, для небольших проектов сборка процессорной системы на базе MicroBlaze занимает не многим больше 10 минут. Во многом этот процесс автоматизирован, что помогает избежать пользовательских ошибок. Ниже, в библиографическом списке, приведена некоторая подборка документов, которые могут послужить в качестве дополнительного материала для более глубокого понимания, как процесса разработки, так и функциональности используемых IP-блоков, а также приведены ссылки на доступные по данной тематике тренинги в сертифицированном тренинг центре компании Xilinx.

Если кто-то захочет попрактиковаться, то вот Ваше домашнее задание:

1. На Arty Board установлены четыре зелёных светодиода. Сделайте из них бегущие огни или бегущую тень или счётчик. Тоже можно сделать и с трёхцветными светодиодами.
2. Настройте GPIO таким образом, что бы можно было считывать состояния кнопок, установленных на плате, и отображать их состояние в SDK terminal.
3. Сделайте запуск бегущих огней по кнопке – любой на Ваш выбор.

Библиографический список

1. [UG984](#). MicroBlaze Processor Reference Guide. Xilinx Inc.
2. [MicroBlaze на сайте Xilinx](#)
3. [Vivado на сайте Xilinx](#)
4. [DS180](#). 7 Series FPGAs Data Sheet: Overview. Xilinx Inc.
5. [Описание Arty Board](#) на сайте Digilent
6. [Сайт](#) компании Регион-Вирта
7. [UG995](#). Designing IP Subsystems Using IP Integrator. Xilinx Inc.
8. [UG782](#). Software Development Kit (SDK) Help
9. [PG144](#). AXI GPIO - LogiCORE IP Product Guide
10. [PG142](#). AXI UART Lite - LogiCORE IP Product Guide
11. [PG059](#). AXI Interconnect - LogiCORE IP Product Guide
12. [UG1037](#). Vivado AXI Reference Guide

Список тренингов

по MicroBlaze в сертифицированном тренинг центре компании Xilinx:

1. [Построение встраиваемых процессорных систем](#)
2. [Дополнительный курс по построению встраиваемых процессорных систем](#)
3. [Разработка ПО для встраиваемых процессорных систем](#)
4. [Доп. курс по разработке ПО для встраиваемых процессорных систем](#)
5. [Полный список курсов](#)