Московский государственный технический университет
имени Н.Э. Баумана

Методические указания

А.Ю. Попов,
С.В. Ибрагимов,
Е.Н.Дубровин

Лабораторная работа

Разработка ускорителей вычислений на платформе Xilinx Alveo

Москва, 2021

Цель работы

Изучение архитектуры гетерогенных вычислительных систем и технологии разработки ускорителей вычислений на базе ПЛИС фирмы Xilinx. В ходе лабораторной работы предлагается изучить основные сведения о платформе Xilinx Alveo U200, разработать RTL (Register Transfer Language, язык регистровых передач)) описание ускорителя вычислений по индивидуальному варианту, выполнить генерацию ядра ускорителя, выполнить синтез и сборку бинарного модуля ускорителя, разработать и отладить тестирующее программное обеспечение на серверной хост-платформе, провести тесты работы ускорителя вычислений.

1. Технология разработки ускорителей вычислений на модулей Xilinx Alveo

Ускорителями вычислений принято называть специальные аппаратные устройства, способные выполнять ограниченный ряд задач с большей параллельностью и за меньшее время в сравнении с универсальными микропроцессорными ЭВМ . Как правило, ускоритель представляет собой структуру, включающую большое количество примитивных микропроцессорных устройств, объединенных шинами связей. Топология связей и набор команд примитивных процессорных устройств определяется назначением ускорителя и позволяет высвободить место на кристалле для увеличения их количества. Таким образом достигается большая параллельность работы. Наибольшее преимущество дают так называемые потоковые ускорители, которые способны обрабатывать данные в темпе их поступления, и передавать по независимому интерфейсу далее, на последующую обработку.

В настоящее время применение ускорителей вычислений охватывает ряд важных областей: финансовые вычисления, ускорение запросов к базам данных, машинное обучение, видео-аналитика. В ряде случаев удается достичь ускорения более чем в 90 раз по сравнению с универсальными ЭВМ, построенными на микропроцессорах Intel x86.

Создание ускорителей вычислений является трудоемким процессом, так как охватывает не только аппаратную разработку самого устройства, но и предполагает оптимизацию архитектуры ЭВМ для обеспечения наибольшей пропускной способности каналов передачи операндов и результатов, а также минимизации задержек и вычислительных затрат при ожидании работы ускорителей. Можно условно разделить ускорители на два класса: ускорители на основе СБИС и на основе ПЛИС. Первый класс имеет преимущества при крупносерийном выпуске, а значит может быть применим только для востребованных технологий, таких как GPGPU. Второй класс использует в качестве рабочей платформы современные крупноразмерные ПЛИС типа FPGA. Такой вариант имеет преимущества по оптимизации инфраструктуры устройства и системы, так как современные ПЛИС предполагают возможность репрограммирования. Однако, ПЛИС проигрывают СБИС по тактовым частотам и рассеиваемой мощности. В данной лабораторной работе мы изучим технологию создания ускорителей вычислений на основе ПЛИС. Основной плат ускорителя Xilinx Alveo U200 (рис. 1) является ПЛИС xcu200-fsgd2104-2-e архитектуры

Рисунок 1 — Плата ускорителя Xilinx Alveo U200

Xilinx UltraScale, выполненная по 16-нанометровой технологии. Плата обеспечивает взаимодействие с хост-системой через интерфейс PCIe gen3 x16, и помимо ПЛИС содержит 4 планки памяти DIMM DDR4 по 16 ГБ, и два QSFP разъема для подключения 100ГБ Ethernet сети. Для работы с ускорительной платой разработано специальное окружение XRT (Xilinx Runtime), включающее компоненты пользовательского пространства и драйвера ядра. XRT поддерживает как карты ускорителей на основе PCIe, так и встроенную архитектуру на основе MPSoC (для встраиваемых плат с ПЛИС Xilinx), обеспечивающую стандартизованный программный интерфейс для Xilinx FPGA. Благодаря этому достигается высокая унификация компонентов технологии и высокая степень переносимости кода ускорителей и ПО. XRT состоит из двух частей:

- программного стека, работающего на базе ОС Linux в хост системе;
- аппаратного модуля в xrt в виде софт ядра ПЛИС.

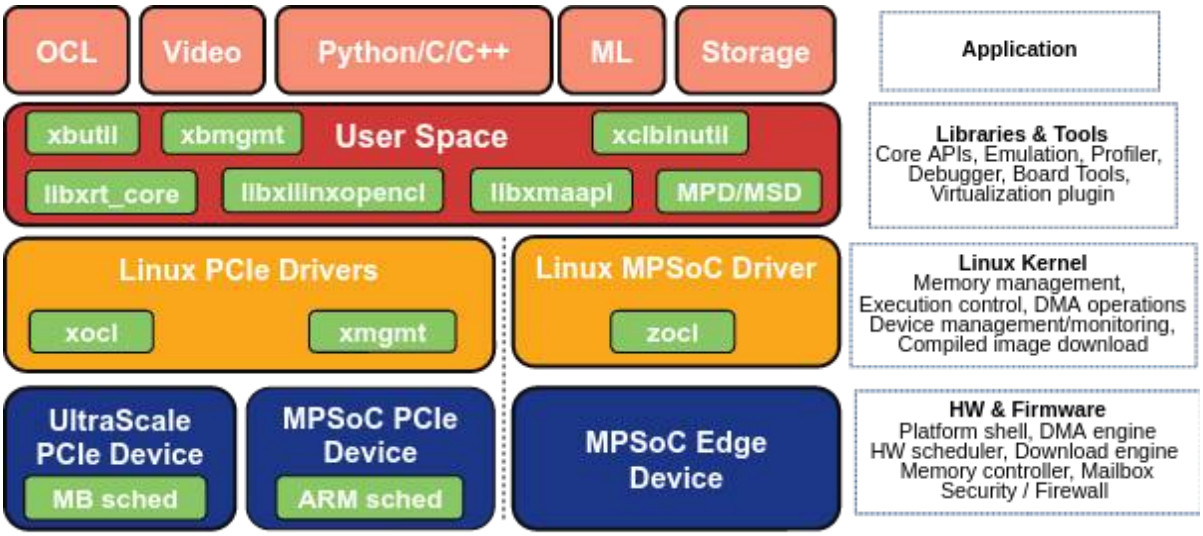


Рисунок 2 — Программный стек XRT

Основными компонентами, необходимыми для управления платой Alveo U200 со стороны пользователя, являются:

- утилита xbutil, позволяет выполнять загрузку пользовательского проекта на ПЛИС4;
- утилита xbmgmt служит для обновления версий XRT, сохраняемых в специальной EEPROM;
- утилита xclbinutil предназначена для формирования конфигурационных образов для ПЛИС.

Программный стек содержит соответствующее стандарту OpenCL API для доступа к ускорителю. Это позволяет применять для разработки языки программирования Python/C/C++, и большое количество OpenCL-совместимых библиотек, таких как: Pytorch, Tensorflow, Caffe2 и многих других.

В оборудовании, используемом для проведения лабораторной работы, использована так называемая XDMA сборка XRT (существует также QDMA сборка, Amazon EC2 F1 сборка, Nimbix Cloud сборка), которая предполагает следующий сценарий взаимодействия ускорителя и пользовательского ПО:

1. Пользовательское ПО сканирует и инициализирует доступные ускорительные платы, совместимые с XRT, определяет доступные ресурсы, создает программное окружение пользовательского аппаратного ядра ускорителя (далее используется термин kernel).
2. Ресурсы локальной памяти ускорительной платы отображаются в пространство памяти хост системы.
3. Инициализируются каналы DMA для прямого доступа к памяти ускорителя.
4. Данные, подлежащие обработке, копируются из ОЗУ в локальную память ускорителя

посредством DMA.

5. Ядру ускорителя (или нескольким ядрам) посредством записи управляющих регистров, передаются параметры вычислений. Пользователь может увеличивать количество параметров по своему усмотрения. Типичным случаем является передача указателей на начало буферов исходных операндов и буфера результата, а также количество обрабатываемых значений.
6. Хост-система выдает сигнал Start ядрам ускорителей, после чего начинается обработка внутри платы Xilinx Alveo.
7. По завершении обработки kernel устанавливает флаг DONE, что вызывает прерывание по шине PCIe.
8. Драйвер обрабатывает прерывание и сообщает пользовательскому ПО о завершении обработки.
9. Пользовательское ПО инициализирует DMA передачу результатов из локальной памяти ускорителя в ОЗУ хост-системы.

Альтернативная сборка QDMA (Queue Direct Memory Access), доступная на картах ускорителей Alveo, предоставляет разработчикам прямое потоковое соединение с низкой задержкой между хостом и ядрами. Оболочка QDMA включает высокопроизводительный DMA, который использует несколько очередей, оптимизированных как для передачи данных с высокой пропускной способностью, так и для передачи данных с большим количеством пакетов. Только QDMA позволяет передавать поток данных непосредственно в логику FPGA параллельно с их обработкой. Оболочка XDMA требует, чтобы данные сначала были полностью перемещены из памяти хоста в память FPGA (DDRx4 DIMM или PLRAM), прежде чем логика FPGA сможет начать обработку данных, что влияет на задержку на запуска задачи.

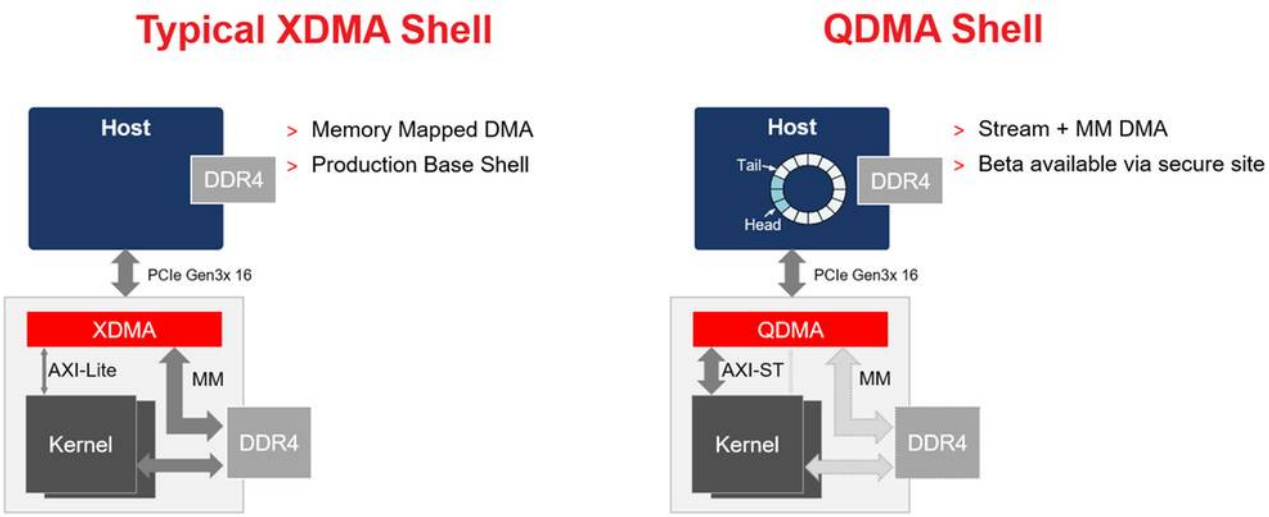


Рисунок 3 — Сравнение XDMA и QDMA сборок платформы XRT

Потоковая передача напрямую в работающие ускорительные ядра (так называемый Free-Running-Mode) позволяет быстро и без излишней буферизации передавать операнды и результаты вычислений на хост по потоковому интерфейсу AXI4 Stream. Решение QDMA подходит для приложений, в которых вычисления строятся на передачи сравнительно небольших пакетов, но при этом требуется высокая производительность и минимальная задержка отклика.

2. Описание архитектуры разрабатываемого ускорителя

В ходе лабораторной работы будет использован базовый шаблон так называемого RTL

проекта VINC, который может быть создан в IDE Xilinx Vitis и САПР Xilinx Vivado. Шаблон VINC выполняет попарное сложение чисел исходного массива и сохраняет результаты во втором массиве. Проект VINC включает:

- Проект ПО хоста, выполняющий инициализацию аппаратного ядра и его тестирование через OpenCL вызовы.
- Синтезируемый RTL проект ядра ускорителя на языках Verilog и SystemVerilog.
- Функциональный тест ускорителя VINC на языке SystemVerilog.

Все перечисленные проекты создаются автоматически посредством запуска мастера RTL проектов в IDE Xilinx Vitis, и могут далее модифицироваться как через тот же мастер, так и в ручном режиме в САПР Xilinx Vivado, или обычном текстовом редакторе. Ряд проектных процедур необходимо запустить из консоли ОС Linux.

Проект VINC представляет собой аппаратное устройство, связанное шиной AXI4 MM (Memory mapped) с DDR[i] памятью, и получающее настроечные параметры по интерфейсу AXI4 Lite от программного обеспечения хоста (см. рисунок 4). В рамках всей системы используется единое 64-х разрядное адресное пространство, в котором формируются адреса на всех AXI4 шинах.

В каждой карте U200 имеется возможность подключить ускоритель к любому DDR[i] контроллеру в том регионе, где будет размещен проект. Всего для пользователя доступны 3 динамических региона: SLR0,1,2, для которых выделены каналы локальной памяти DDR[0], DDR[2], DDR[3] соответственно. Вся подключенная память DDR[0..3] доступна со стороны статического региона, в котором размещена аппаратная часть XRT. Память DDR[1] доступна для использования как в статическом регионе, так и в динамическом регионе SLR1. Предполагается, что эта память может служить для организации эффективной подсистемы памяти ускорительной карты: буферизации данных, передаваемых между хост-системой и ускорителем.

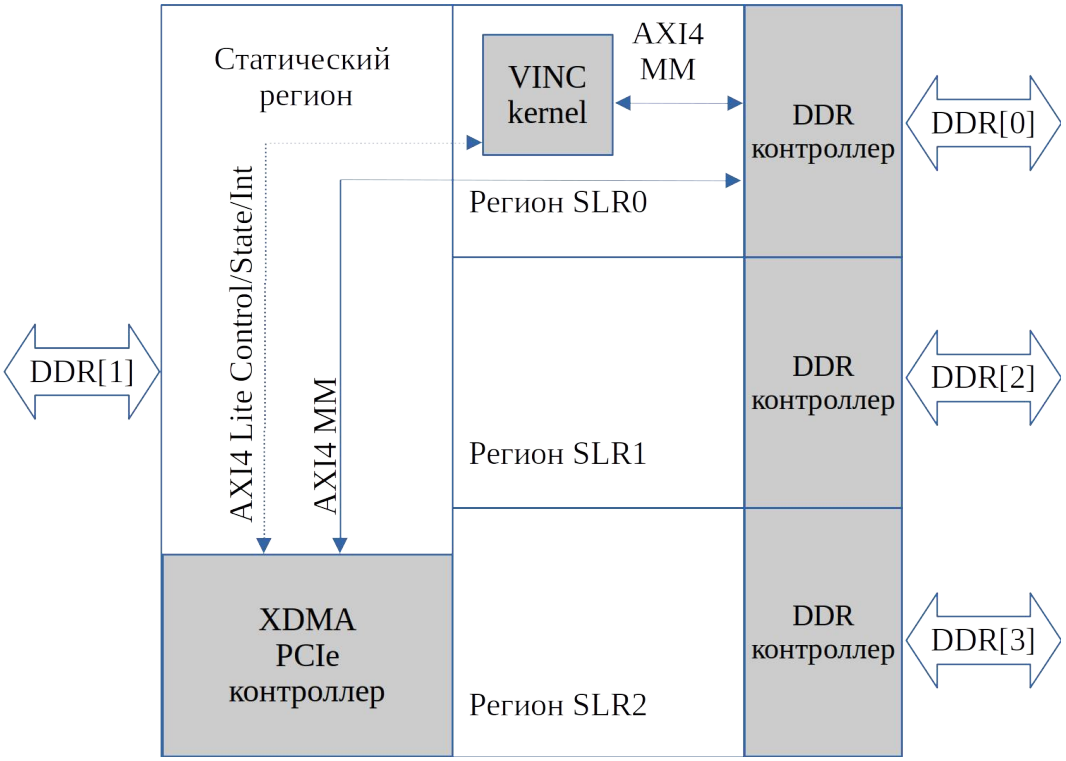


Рисунок 4 — Размещение проекта на ПЛИС xcu200-fsgd2104-2-e карты Alveo U200

Для организации прямого доступа к памяти DDR со стороны хоста также используется AXI4MM шина, соединяющая XDMA PCIe контроллер с контроллером памяти. Таким образом возможно реализовать ускоритель с типовой архитектурой XDMA, описанной

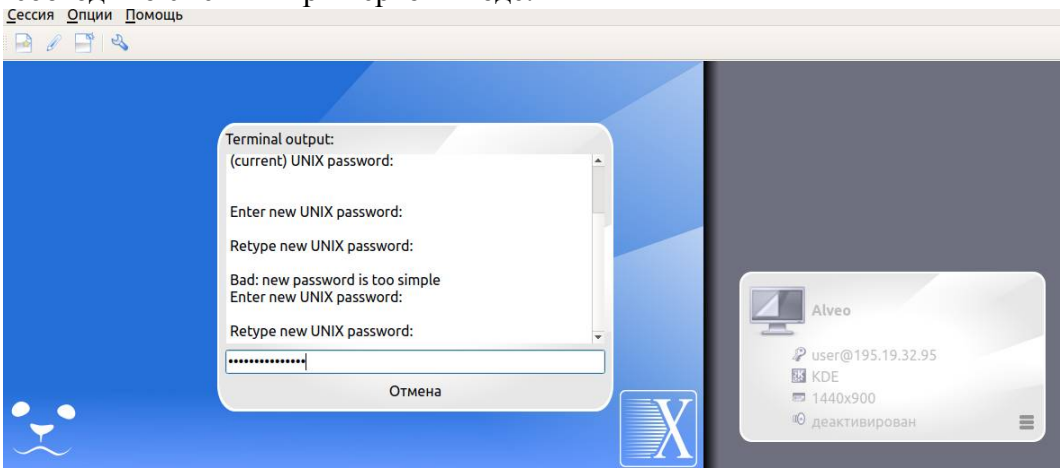
ранее на рисунке 3.

Выбор одного из регионов для размещения проектов осуществляется на этапе так называемой линковки конфигурационного файла при помощи компилятора v++(фактически: компоновки, размещение и трассировки нескольких проектов в единый конфигурационный файл).

3. Практическая часть

3.1. Подключение к серверу Alveo

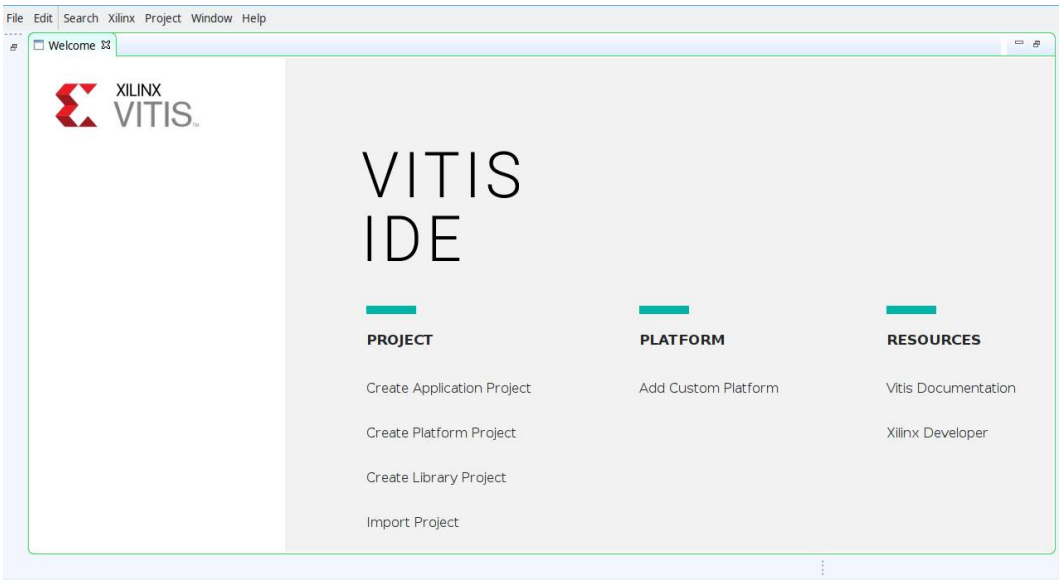
Для разработки проектов будет использован сервер с картой Alveo U200. На сервере установлены все необходимые средства разработки, и организован множественный доступ к рабочим столам пользователей.
Для доступа к серверу необходимо использовать клиент X2Go (https://wiki.x2go.org/doku.php/doc:installation:x2goclient), который доступен для основных операционных систем: Windows, Linux, OsX, FreeBSD Unix.
Адрес сервера и параметры входа будут выданы преподавателем на лабораторной работе.
В окне программы X2Go создайте новое соединение с указанными параметрами.
Пароль необходимо сменить при первом входе:



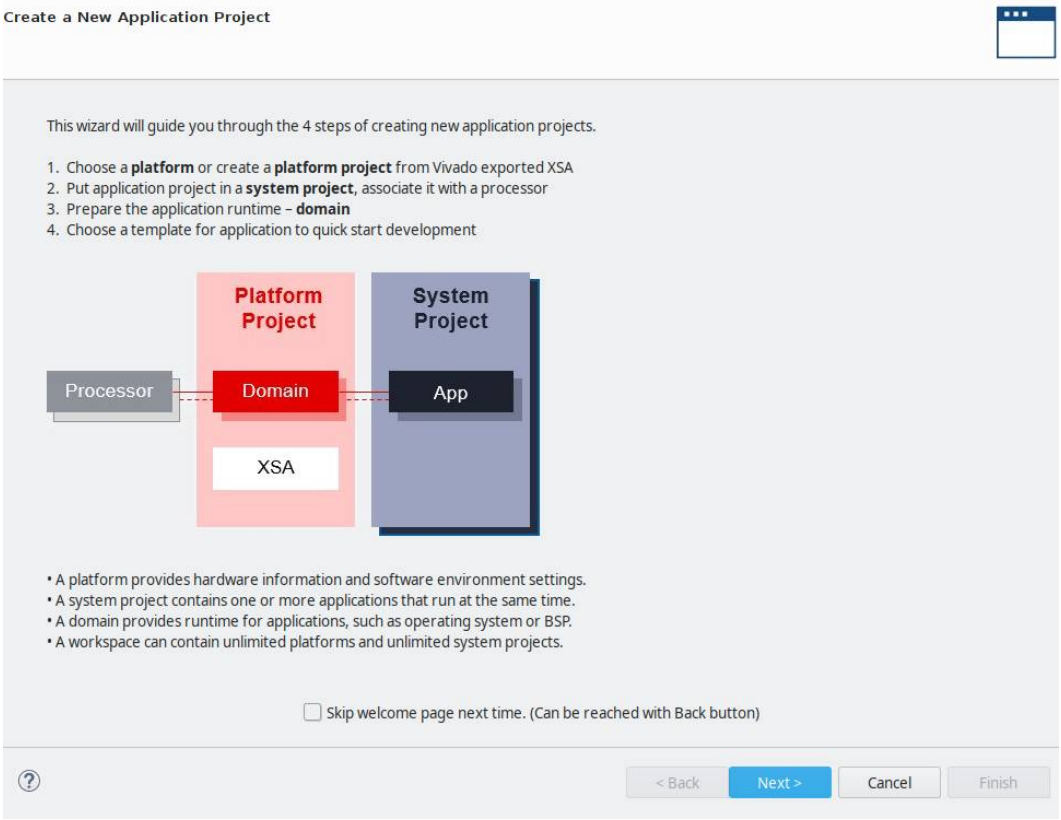
Далее вы будете подключены к рабочему столу KDE операционной системы Ubuntu 18.04:

3.2. Создание проекта VINC

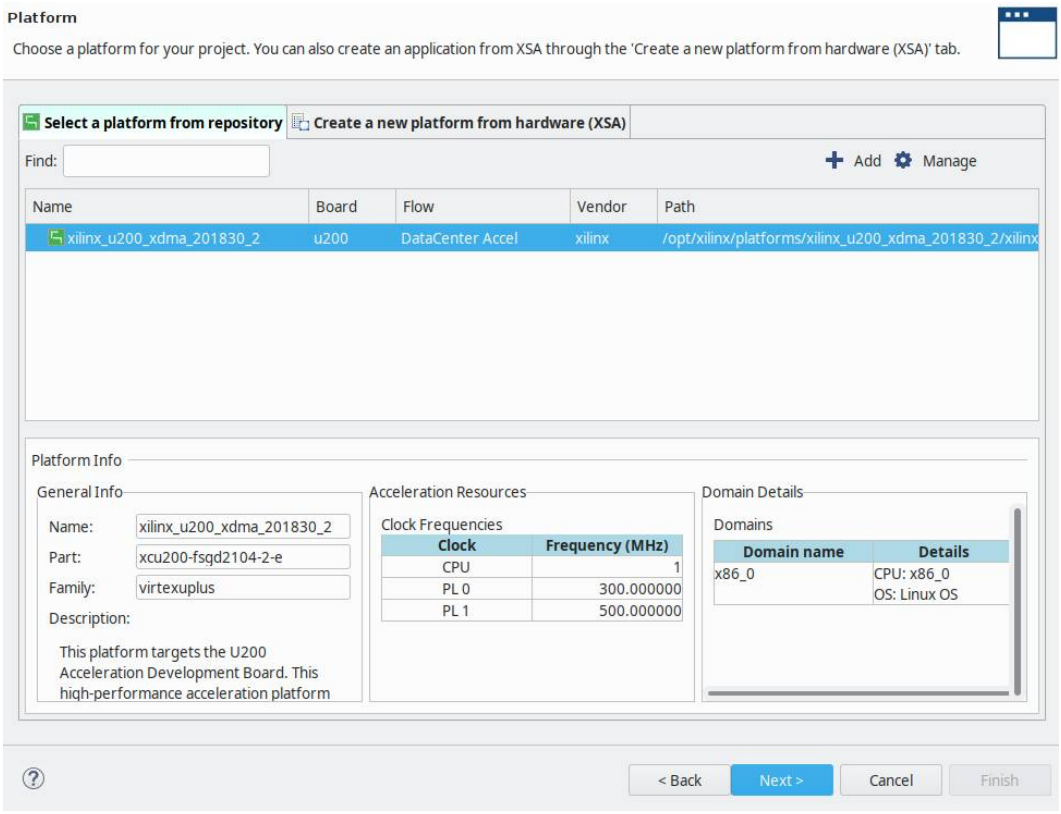
Запустите Vitis IDE.



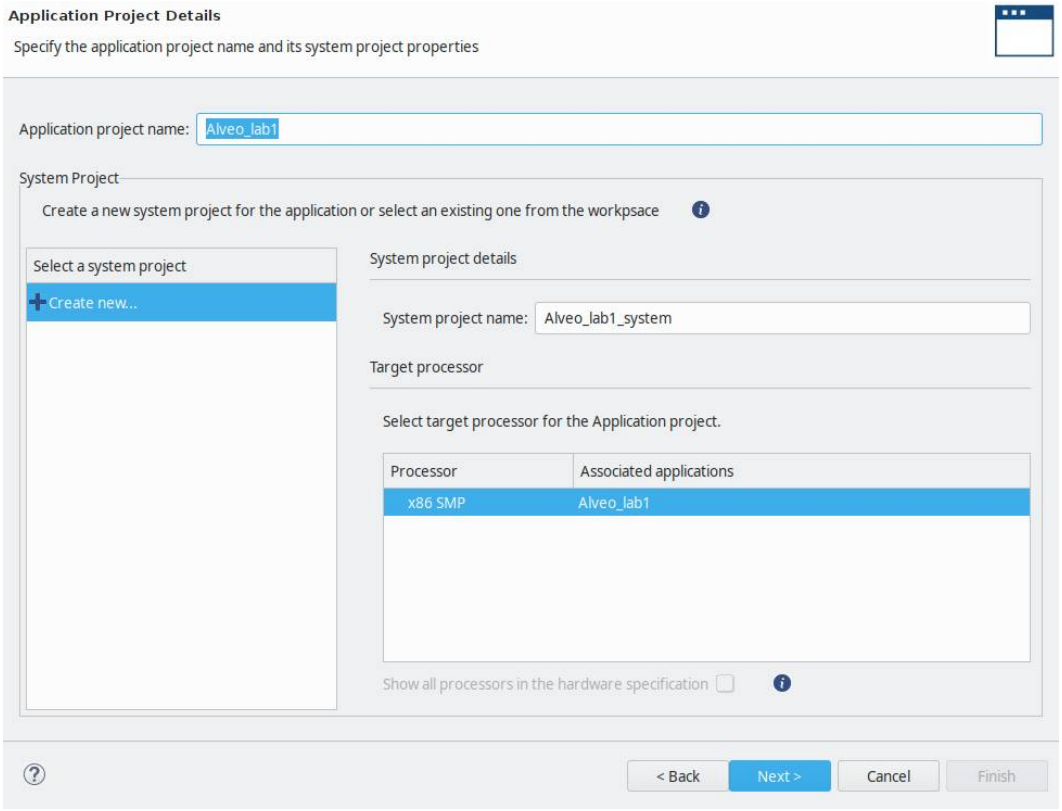
Создайте новый проект в пункте: Create Application Project.
В открывшемся окне нажмите: Next.



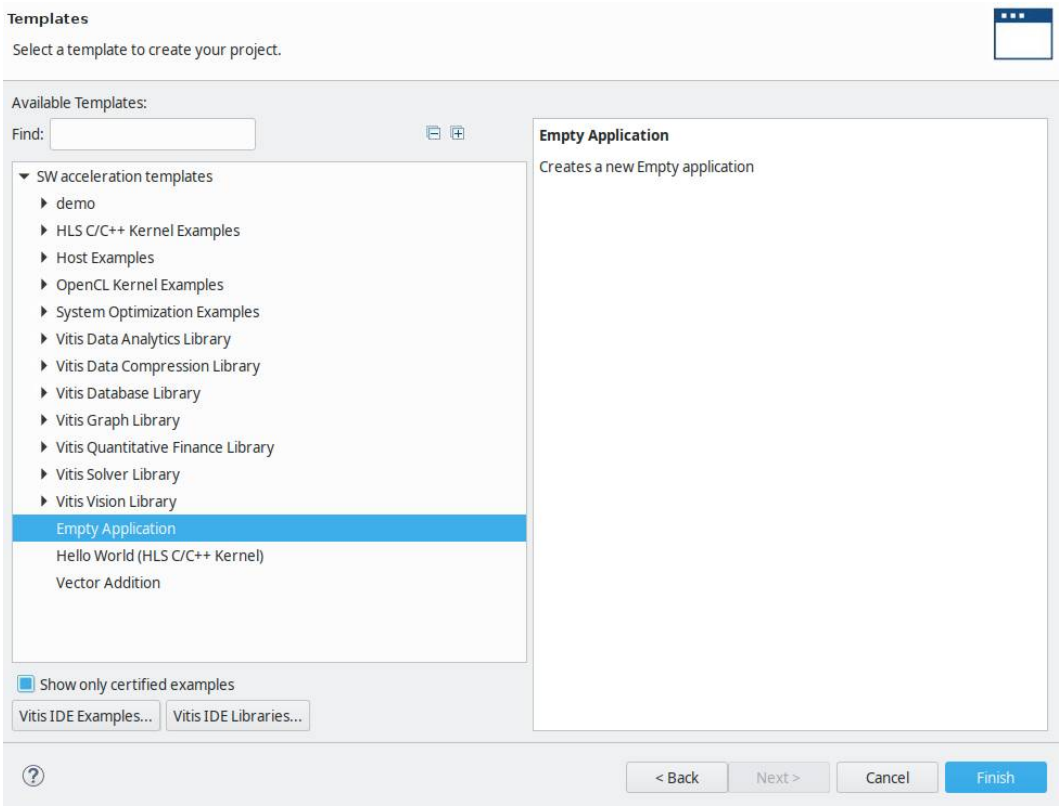
В следующем окне выберите платформу xilinx_u200_xdma_201830_2.



Далее укажите название проекта

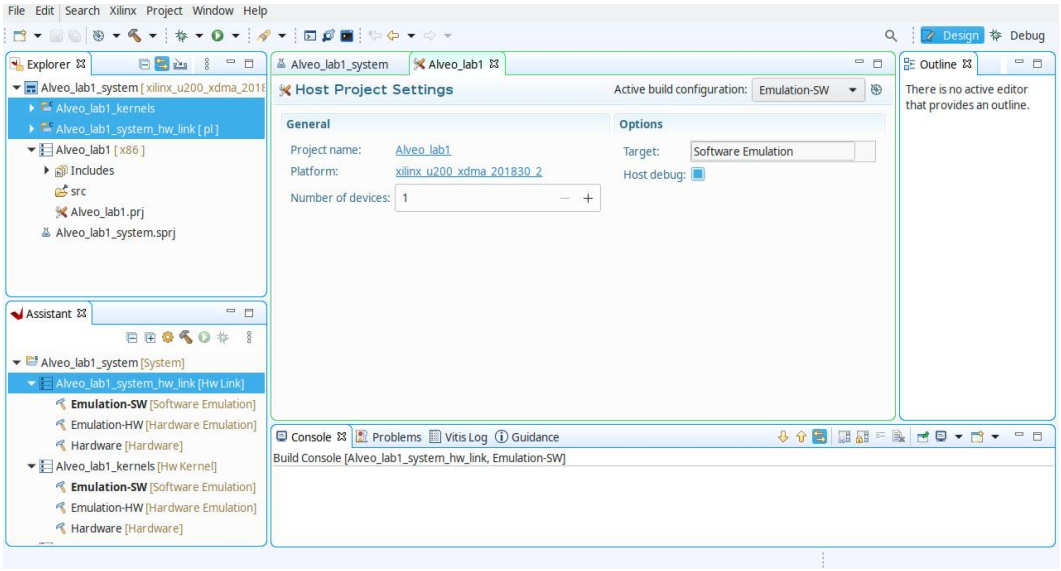


Далее выберите пункт: Empty Application и Finish.

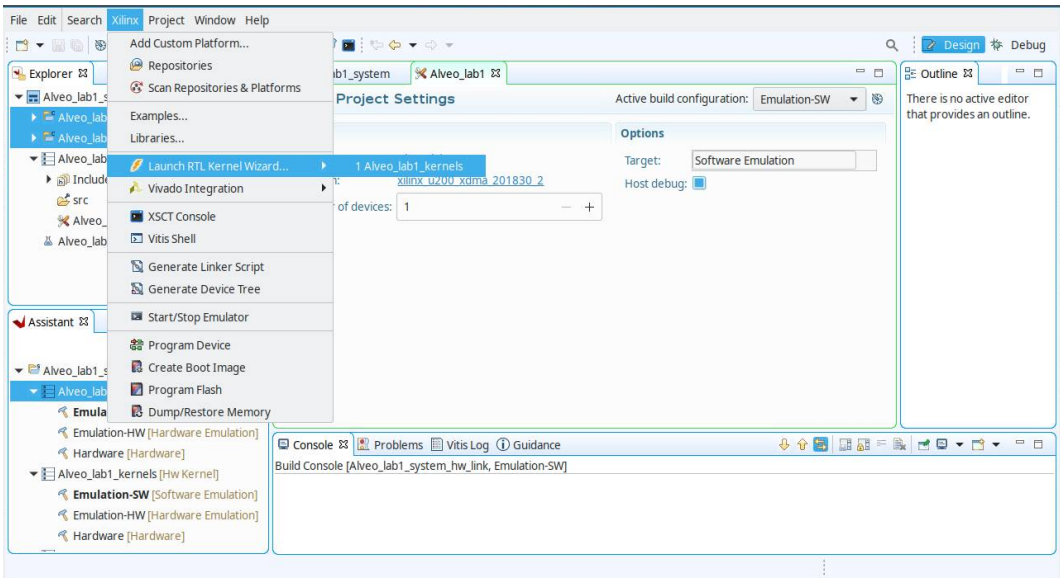


В итоге будет создан шаблон проекта, состоящий из следующих частей:

- <Название проекта>_kernels — исходные описания ядер проекта и результаты сборки объектных модулей *.xo.
- <Название проекта>_system_hw_link — результаты линковки ядер в единый конфигурационный файл *.xclbin
- <Название проекта> - исходные описания программного обеспечения хоста.

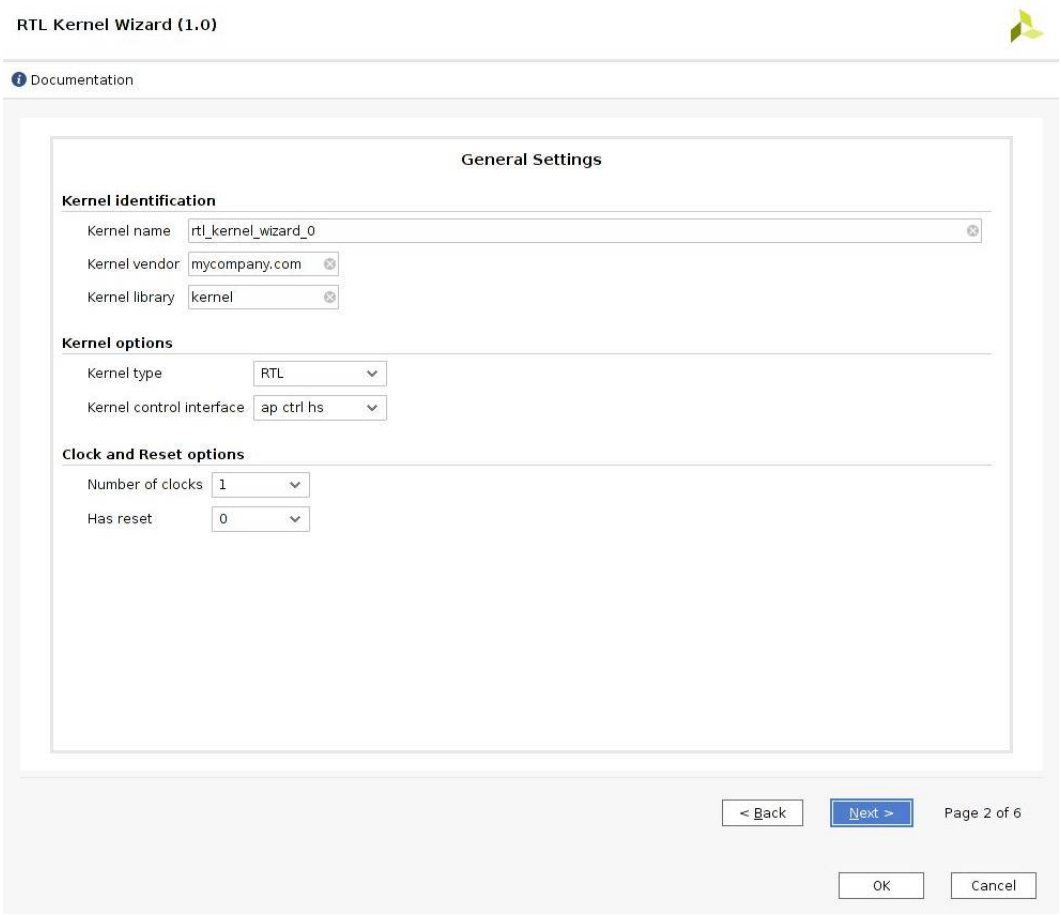


Запустите мастер RTL проекта VINC. Для этого в меню Xilinx выберите пункт Launch RTL Kernel Wizard. Далее введите название вашего проекта и нажмите кнопку Next.



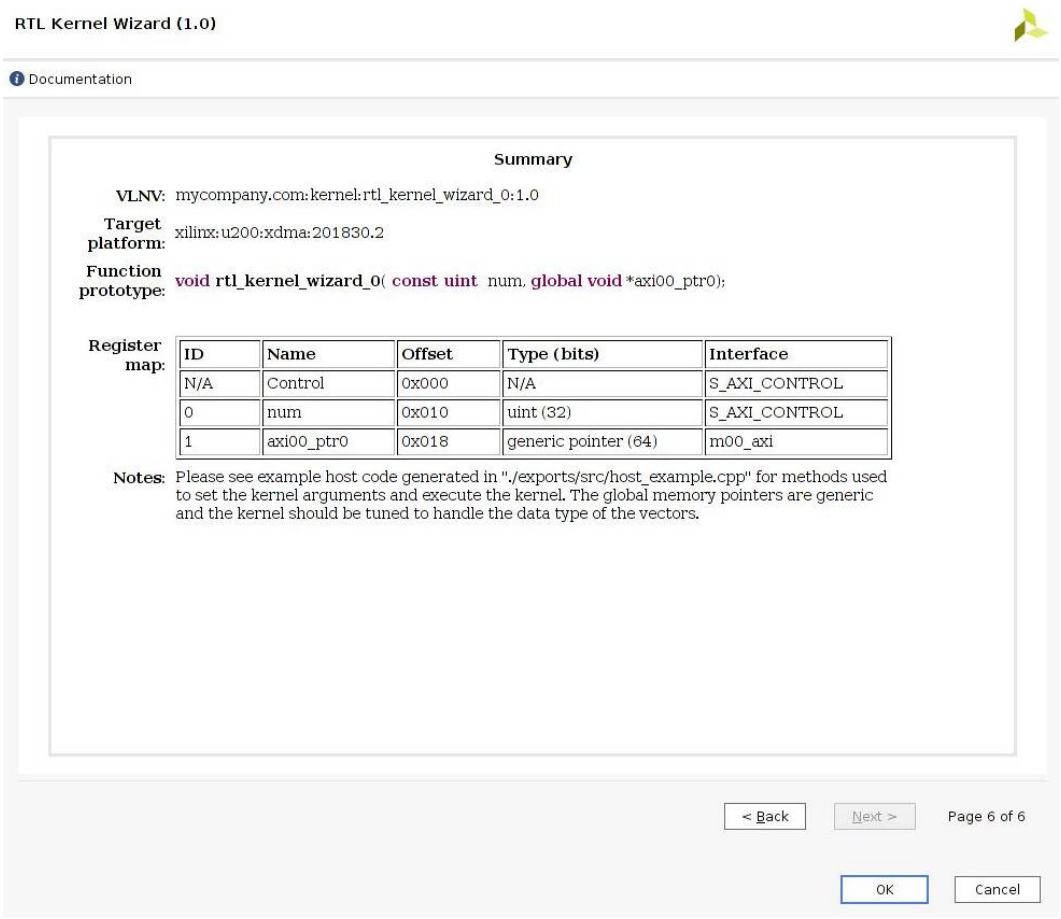
Далее укажите параметры:

- Number of Clocks — 1
- Has Reset — 0
- Kernel control interface — ap ctrl hs

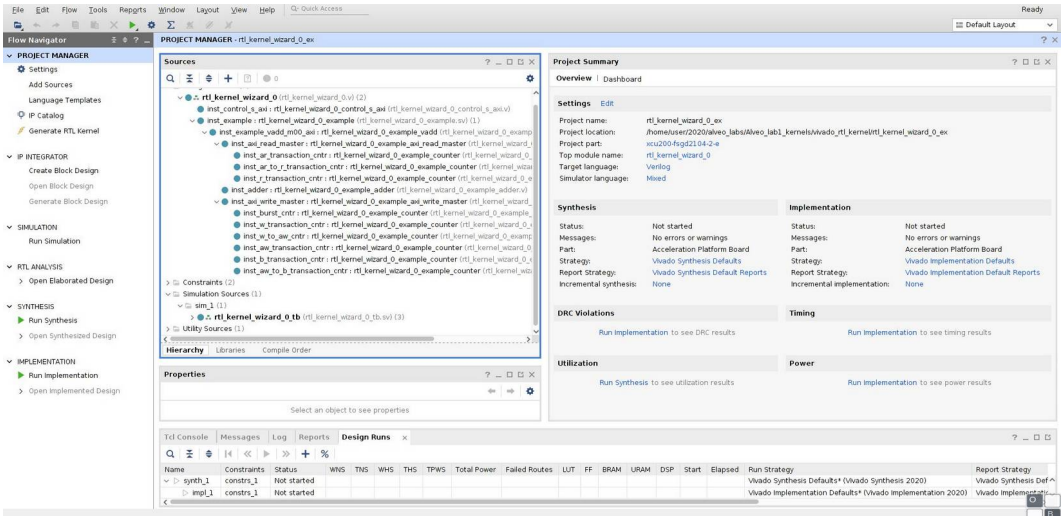


В следующем окне задайте имя настраиваемого параметра ядра.
Далее, в окне настройки AXI4 MM интерфейсов ядра введите название параметра — указателя на область памяти для размещения данных.

В следующем окне укажите количество потоковых интерфейсов: Number of AXI4-Stream Interfaces = 0.

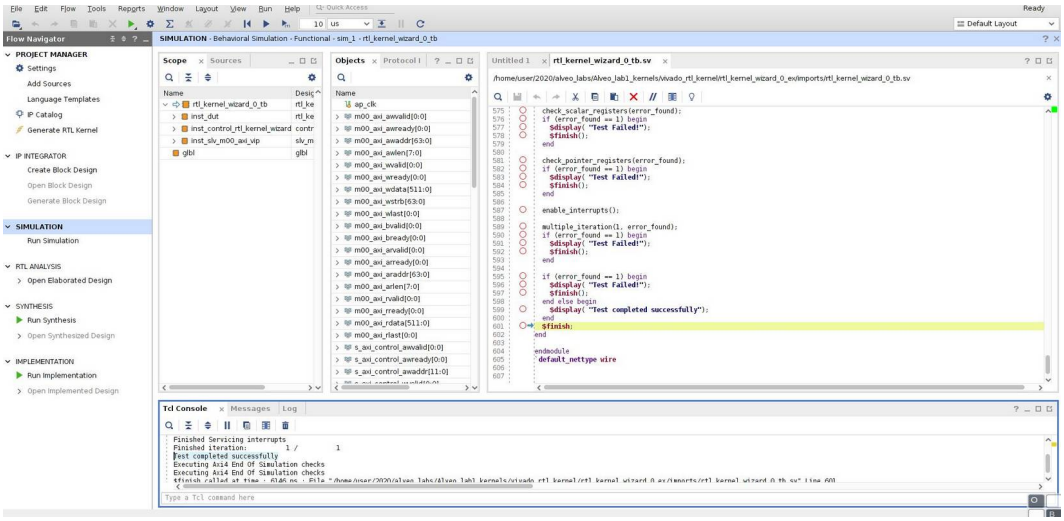


В итоге будет создан проект ядра ускорителя и он будет открыт САПР Xilinx Vivado для редактирования.

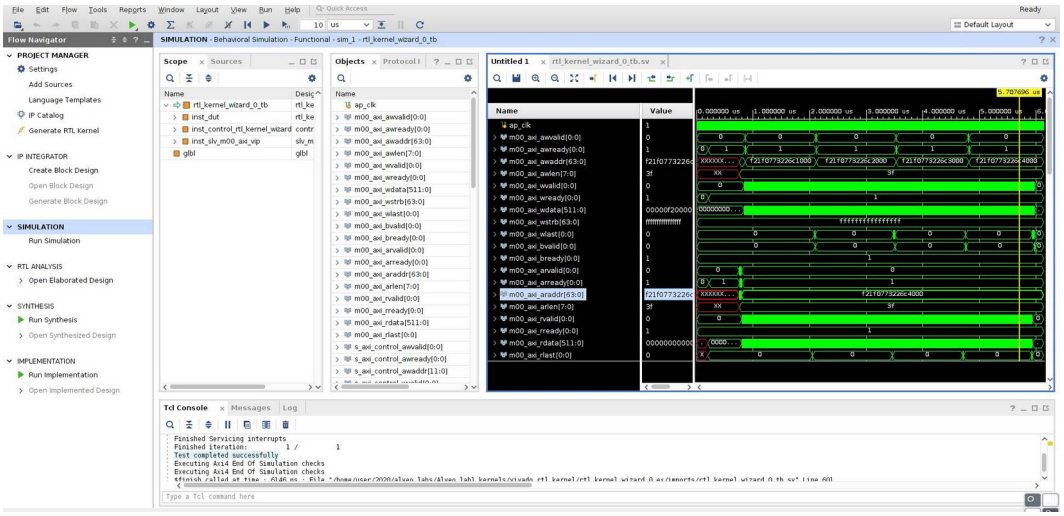


Ознакомьтесь с содержимым модулей проекта. Большинство модулей написаны на языке SystemVerilog. Пояснения по синтаксису языка вы можете получить на странице: http://www.asic-world.com/systemverilog/
Модуль rtl_kernel_wizard_0_tb в разделе Simulation Sources содержит тест проекта.
Запустите симуляцию проекта в меню: Simulation → Run Simulation → Run Behavioural

Simulation. Запустите тест и дождитесь конца симуляции (директива $finish).



Во вкладке Scope в модуле проекта inst_dut вы можете выбрать и добавить в окно симуляции любые необходимые сигналы. При сохранении конфигурационного файла симуляции все сделанные вами изменения в списке сигналов будут учтены при следующем запуске симуляции.



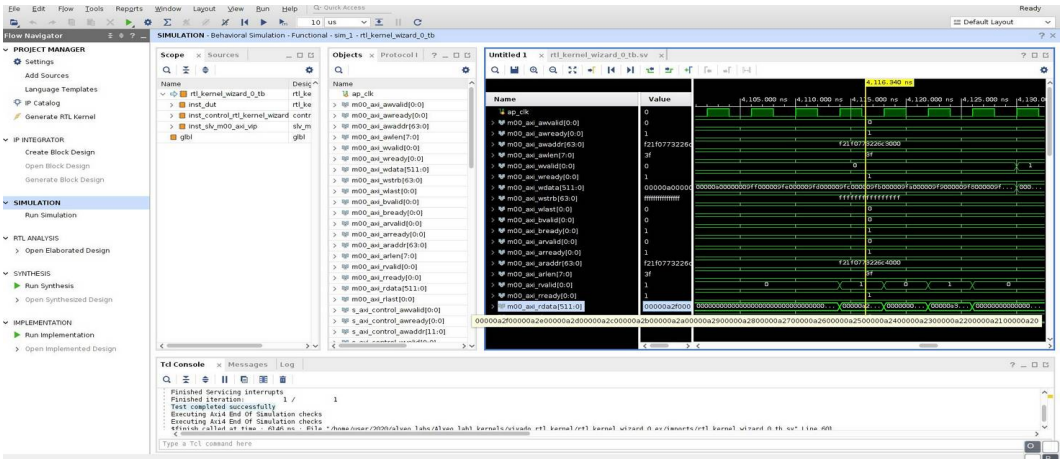
По умолчанию, в диаграмму добавлены сигналы шины AXI4 MM, представляющие собой 5 независимых каналов передачи сообщений:

Канал передачи	Группы сигналов
Канал чтения адреса от ведущего к ведомому	m00_axi_ar*
Канал чтения данных от ведомого к ведущему	m00_axi_r*
Канал записи адреса записи от ведущего к ведомому	m00_axi_aw*
Канал запись данных от ведущего к ведомому	m00_axi_w*
Канал записи ответа от ведомого к ведущему	m00_axi_b*

Каналы позволяют сформировать конвейерные транзакции чтения и записи.
Последовательность событий транзакции чтения можно представить следующим образом: ARVALID→ ARREADY→ RVALID→ RREADY.
Последовательность событий транзакции записи: AWVALID→ AWREADY → WVALID → WREADY → BVALID → BREADY.

Сформируйте следующие диаграммы:

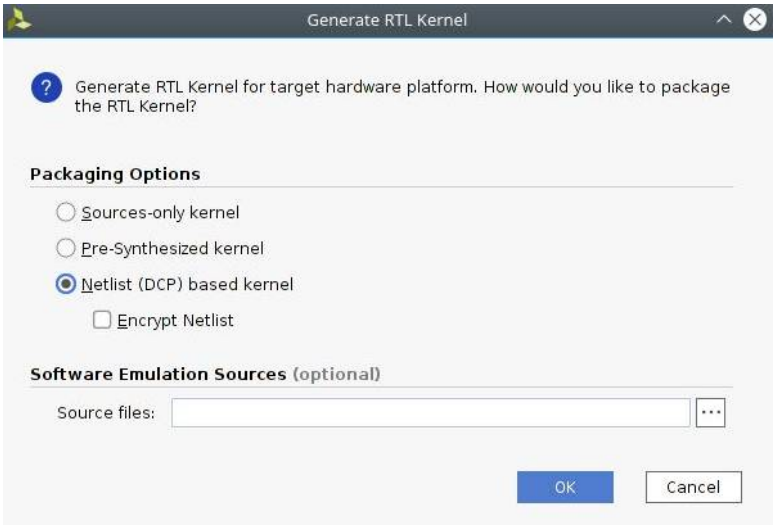
- Одна транзакция чтения данных вектора на шине AXI4 MM из DDR памяти (группы сигналов m00_axi_ar* и m00_axi_r*).
- Одна транзакция записи результата инкремента данных на шине AXI4 MM (группы сигналов m00_axi_aw*, m00_axi_w* и m00_axi_b*).
- Инкремент данных в модуле <Название проекта>_adder.v.



Результат занесите в отчет.

3.3.Сборка проекта

В САПР Xilinx Vivado выберите пункт Generate RTL Kernel. Далее выберите пункт Netlixt (DCP) based kernel (генерация ядра ускорителя в виде синтезированных низкоуровневых описаний).



В результате будет сформирован *.xo файл, представляющий собой архив проекта Vivado с результатами синтеза. Путь к файлу *xo вы может узнать в отчете по сборке или в TCL консоли Vivado.

Далее создайте конфигурационный файл <Название проекта>.cfg для запуска линковки проекта. В конфигурационом файле указывается основная информация для работы компилятора v++:

- Количество и условные имена экземпляров ядер.
- Тактовая частота работы ядра.
- Для каждого ядра: выбор области SLR (SLR[0..2]), выбор DDR (DDR[0..3]) памяти, выбор высокопроизводительной памяти PLRAM(PLRAM[0,1,2]).
- Параметры синтеза и оптимизации проекта.

Пример конфигурационного файла:

```
[connectivity]
# Объявление ядра и экземпляров ядра (nk)
# rtl_kernel_vinc   - Имя модуля верхнего уровня Vivado (см. RTL проект)
# 2               - Количество экземпляров ядра
# vinc2,vinc1      - Условные имена экземпляров ядра (определяются разработчиком)
# Доступ из ПО хоста выполняется по имени <Имя ядра>:{<Имя экземпляра>}.
#  Например: rtl_kernel_vinc:{vinc0}
nk=rtl_kernel_vinc:2:vinc0.vinc1

# Связывание с регионом (slr)
# vinc0           - Имя экземпляра ядра
# SLR0            - Имя региона. Допустимые имена регионов: SLR0,SLR1,SLR2
slr=vinc0:SLR0

# Связывание с интерфейсом памяти (sp)
# vinc0           - Имя экземпляра ядра
# axi4_0,axi4_1    - Имя интерфейса ядра в проекте Vivado (см. RTL проект).
# DDR[0],DDR[1]    - Имя интерфейса памяти. Допустимые имена интерфейса: PLRAM[0..2], DDR[0..3]
sp=vinc0.axi4_0:DDR[0]
sp=vinc0.axi4_1:PLRAM[0]

# Аналогичные директивы для каждого экземпляра ядра
slr=vinc1:SLR1
sp=vinc1.axi4_0:DDR[2]
sp=vinc1.axi4_1:PLRAM[1]

[vivado]
# Настройки параметров фазы имплементации
prop=run.impl_1.STEPS.OPT_DESIGN.ARGS.DIRECTIVE=Explore
prop=run.impl_1.STEPS.PLACE_DESIGN.ARGS.DIRECTIVE=Explore
prop=run.impl_1.STEPS.PHYS_OPT_DESIGN.IS_ENABLED=true
prop=run.impl_1.STEPS.PHYS_OPT_DESIGN.ARGS.DIRECTIVE=AggressiveExplore
prop=run.impl_1.STEPS.ROUTE_DESIGN.ARGS.DIRECTIVE=Explore
```

Конфигурационный файл занесите в отчет.
По умолчанию, если не задан конфигурационный файл, все интерфейсы будут подключены к одной и той-же внешней памяти.
В конфигурационном файле задайте описание для линковки одного ядра ускорителя VINC.
Далее в системной консоли или через ssh подключение к серверу запустите команду:

> screen

По команде будет создана независимая сессия консоли, что позволит отключиться от сервреа без останова процесса линковки.
Для выхода из сессии в основную консоль используйте команду Dettach (Ctrl+A,D).
Для повторного подключения к сессии используйте команду

> screen -r

Запустите линковку проекта в сессии screen:

> v++ -l -t hw -o <Путь к файлу vinc.xclbin> -f xilinx_u200_xdma_201830_2 <Путь к *.xo> --config <Путь к конфигурационному файлу.cfg>

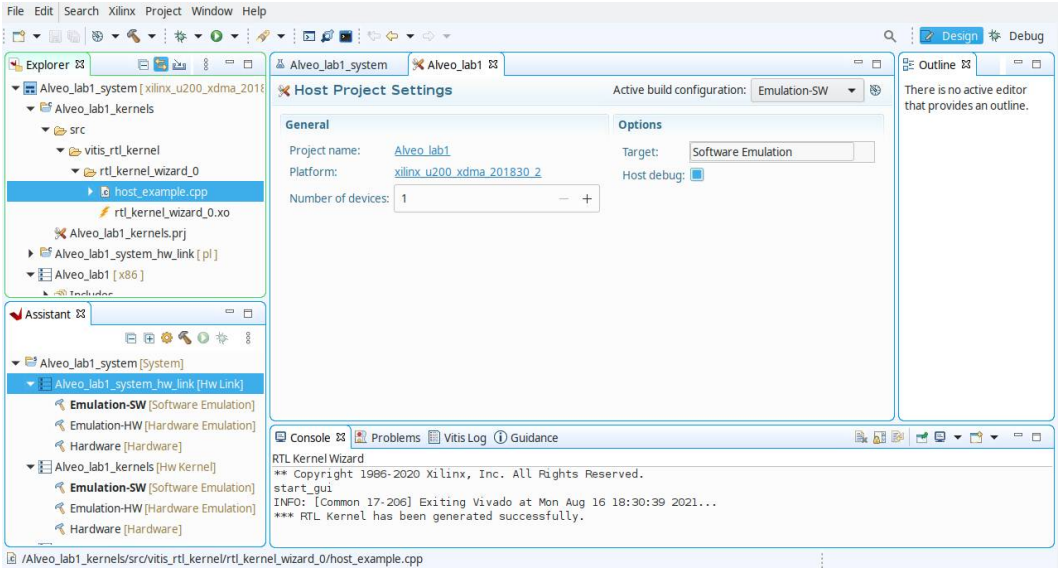
Процесс может занять до 6 часов. В результате будет сформирован файл *.xclbin, который может быть передан в ускорительную карту утилитой xbutil. Однако, принято включать код для программирования ПЛИС непосредственно в код проекта Vitis. Вы можете обнаружить этот вызов в коде проекта:

```
cl_int err;
std::string binaryFile = (argc != 2) ? "vinc.xclbin" : argv[1]; //Путь к xclbin задается аргументом
unsigned fileBufSize;
std::vector<cl::Device> devices = get_xilinx_devices();
devices.resize(1);
cl::Device device = devices[0]; //Выбираем первую найденную карту
cl::Context context(device, NULL, NULL, NULL, &err);
char *fileBuf = read_binary_file(binaryFile, fileBufSize); //Читаем xclbin в буфер
cl::Program::Binaries bins{{fileBuf, fileBufSize}};
cl::Program program(context, devices, bins, NULL, &err); //Программируем ПЛИС
```

Вместе с генерацией файла *.xclbin, генерируется лог файл v++*.log и файл описания ресурсов *.xclbin.info. Ознакомьтесь с указанными файлами и занесите их содержимое в отчет.

3.4. Запуск программного обеспечения на хост-системе.

Скопируйте автоматически созданный программный модуль host_example.cpp в проект <Название проекта>/src. Ознакомьтесь с его содержимым. Выполните сборку проекта Project→Build.



В меню Xilinx войдите в пункт Debug Configuration.
Создайте новую Debug сессию типа TCF Local System debug.
Нажмите кнопку Debug. После этого запустится сессия отладки, которая остановится в начале функции main().
Дальнейшая отладка выполняется стандартными процедурами: установка точек останова, создание списка просматриваемых переменных, просмотра дампа памяти и т.д.

3.5. Индивидуальные задания

1. Измените код модуля <Название проекта>_adder.v RTL таким образом, чтобы ускоритель выполнял указанную в Таблице 1 функцию.

Таблица 1 — Индивидуальные варианты функций ускорителя.

Вариант	Функция	Регион
1	R[i] = A[i]/2 + 10	SLR0,DDR[0]
2	R[i] = A[i]*16 + (10<<7)	SLR1,DDR[1]
3	R[i] = (A[i] -1)*4	SLR1,DDR[2]
4	R[i] = max(A[i], B[i])	SLR2,DDR[3]
5	R[i] = (A[i] + 3) / 2	SLR0,DDR[0]
6	R[i] = A[i] & 0xFEDCBA9876543210	SLR1,DDR[1]
7	R[i] = A[i] | ~ 0xf	SLR1,DDR[2]
8	R[i] = (A[i] + 10) | (255<<32)	SLR2,DDR[3]
9	R[i] = min (A[i] - 4, 5)	SLR0,DDR[0]
10	R[i] = A[i]/8 + 14	SLR1,DDR[1]
11	R[i] = 25 + A[i]/2	SLR1,DDR[2]
12	R[i] = (A[i] & 0xf0f0f0f0f0f0f0f0f0f0) + 10	SLR2,DDR[3]
13	R[i] = A[i]*8 + 10	SLR0,DDR[0]
14	R[i] = A[i]/16 - 11	SLR1,DDR[1]
15	R[i] = (A[i] + 128)/8	SLR1,DDR[2]
16	R[i] = A[i]*4 - 16	SLR2,DDR[3]
17	R[i] = min(A[i] , 500) + 3	SLR0,DDR[0]
18	R[i] = ~(A[i]+2)	SLR1,DDR[1]
19	R[i] = max (A[i],3000)	SLR1,DDR[2]
20	R[i] = min(A[i]-1,120)	SLR2,DDR[3]
21	R[i] = ((~A[i])*8)	SLR0,DDR[0]
22	R[i] = ~A[i] + ~0xf	SLR1,DDR[1]
23	R[i] = min(A[i]+3,1000)	SLR1,DDR[2]
24	R[i] = (A[i] + 100) / 16	SLR2,DDR[3]
25	R[i] = ~A[i] - (1024<<7)	SLR0,DDR[0]
26	R[i] = min(A[i]-200,300)	SLR1,DDR[1]
27	R[i] = max(A[i]*128, 16)	SLR1,DDR[2]
28	R[i] = (A[i] + 1024)*32	SLR2,DDR[3]
29	R[i] = (A[i] + 4)/1024	SLR0,DDR[0]
30	R[i] = 2*A[i] + 30	SLR1,DDR[1]

2. Выполните моделирование ускорителя и зафиксируйте в отчете следующие диаграммы:
   - Транзакция чтения данных вектора на шине AXI4 MM из DDR памяти.
   - Транзакция записи результата инкремента данных на шине AXI4 MM.
   - Инкремент данных в модуле <Название проекта>_adder.v.
3. Выполните линковку ускорителя.
4. Модифицируйте host_example.cpp таким образом, чтобы выполнялась проверка выполнения вашей функции в ускорителе.
5. Запустите программный тест. Результаты проверки работоспособности занесите в отчет.

4. Содержание отчета

1. ФИО, номер группы студента, номер лабораторной работы.
2. Функциональная схема разрабатываемой аппаратной системы (аналогично рисунку 4)
3. Копии экранов моделирования исходного проекта VINC (транзакция чтения данных, транзакция записи результатов, инкремент данных в модуле <Название проекта>_adder.v).
4. Конфигурационный файл линковки.
5. Содержимое файлов v++*.log и *.xclbin.info.
6. Измененный код <Название проекта>_adder.v в соответствии с индивидуальным заданием.
7. Копии экранов модулирования измененного проекта VINC (транзакция чтения данных, транзакция записи результатов, инкремент данных в модуле <Название проекта>_adder.v).
8. Код модифицированного модуля host_example.cpp.
9. Результаты запуска host_example.cpp (в виде листинга вывода на консоль или в виде таблицы с результатами теста).

5. Контрольные вопросы

1. Назовите преимущества и недостатки XDMA и QDMA платформ.
2. Назовите последовательность действий, необходимых для инициализации ускорителя со стороны хост-системы.
3. Какова процедура запуска задания на исполнения в ускорительном ядре VINC.
4. Опишите процесс линковки на основании содержимого файла v++_*.log.

Ссылки на литературу

1. https://www.xilinx.com/products/design-tools/vitis/xrt.html
2. https://www.xilinx.com/html_docs/xilinx2021_1/vitis_doc/index.html
3. http://www.asic-world.com/systemverilog/
4. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug1238-sdx-rnil.pdf