**Министерство науки и высшего образования Российской Федерации  
федеральное государственное автономное образовательное учреждение высшего образования  
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»  
(Университет ИТМО)**

**Факультет программной инженерии и компьютерной техники**

**Курсовой проект**

**Дисциплина «Проектирование систем на кристалле»**

Выполнили: Морозов О. В.  
Кравцов С. Д.

Группа: P41193

Санкт-Петербург  
2020

СОДЕРЖАНИЕ

[Задание 3](#_Toc44386330)

[Исследование спецификации RISC-V 4](#_Toc44386331)

[Проектирование аппаратного модуля с плавающей точкой с помощью системы высокоуровневого синтеза 9](#_Toc44386332)

[Отладка спроектированного модуля в среде RTL-симулятора 13](#_Toc44386333)

[Оценка конфигурируемости системы 15](#_Toc44386334)

[Интеграция модуля FPU в проект SCR1 18](#_Toc44386335)

[Проведение тестов в среде Verilator 19](#_Toc44386336)

[Вывод 20](#_Toc44386337)

[Перечень дополнительных материалов 21](#_Toc44386338)

[Приложение А 22](#_Toc44386339)

[Приложение Б 25](#_Toc44386340)

[Приложение В 26](#_Toc44386341)

# Задание

**Цель работы**: Исследования возможностей HLS для добавления аппаратного расширителя операций с числами с плавающей точкой к проекту процессора SCR1.

Создаваемый расширитель должен исполнять арифметические операции из расширения RV-F.

Тестирование работы расширителя в проекте SCR1 планируется проводить в Verilator. Создания тестового модуля будет осуществляться в симуляторе Vivado для проверки работы расширителя: корректность работы команд. Для работы будет использован инструмент Vivado HLS.

Задача разделена на следующие этапы:

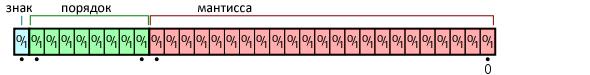
1. Составление интерфейса модуля для подключения к SCR1.
2. Составление программы для HLS. Получение verilog модуля. - Семён
3. Создание тестбенча для модуля в Vivado. Проверка функциональности. - Олег
4. Подключение расширения к SCR1 и проведение system-тестов. - Олег
5. Подготовка отчета, защита проекта.

Материалы проекта доступны по ссылке: <https://github.com/semen121v/Course-project-SCR1-with-FPU>

# Исследование спецификации RISC-V

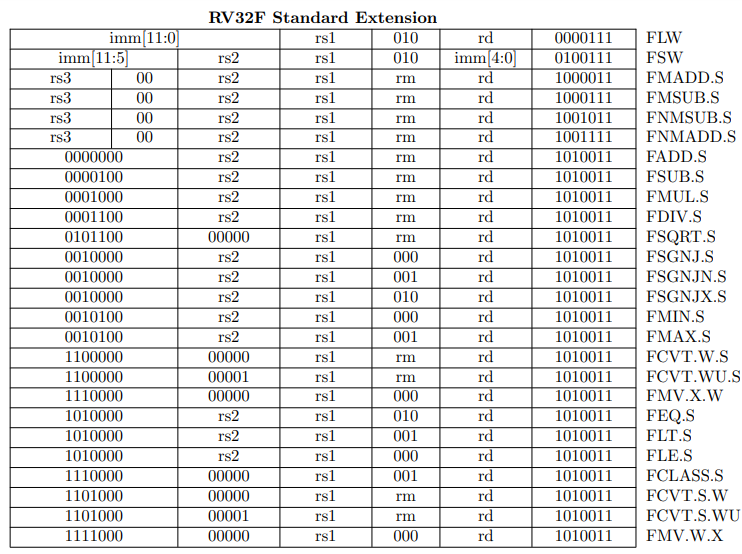
В RISC-V для работы с числами с плавающей точкой используются 32 регистра, обозначаемых f0–f31, размером 32 бита. Расширение, включающее в себя операции с плавающей запятой над числами одинарной точности, обозначается RV-F, обозначение происходит от названия формата данных Float. Формат данных и преобразования для выполнения операций над ними соответствуют стандарту IEEE 754-2008.

IEEE 754-2008 – это стандарт, описывающий формат чисел с плавающей точкой, представление специальных значений (ноль, бесконечность, NaN), преобразования числа при выполнении математических операций, исключительные ситуации.

  
Формат числа с плавающей запятой.

Графически формат числа с плавающей запятой представлен на рисунке. Математическая формула, описывающая этот формат, записывается так:

, где S – знак, B – основание, E – порядок, M – мантисса. Для двоичного представления формула будет выглядеть так:

  
Полный набор команд RV-F.

* FRCSR/FSCSR – псевдоинструкции чтения и записи CSR модуля FPU.
* FLW/FSW – загрузка и сохранение слова, инструкции не изменяют значение.
* FADD/FSUB/FMUL/FDIV/FSQRT – инструкции сложения, вычитания, умножения, деления, вычисления корня квадратного.
* FMADD/FMSUB – операция сложить/вычесть и умножить, формулы

и соответственно.

* FNMSUB/FNMADD – операция вычесть/сложить и умножить противоположного знака, формулыи .
* FMIN/FMAX – сравнивает два значения из регистров и записывает меньшее/большее значение.
* FEQ/FLT/FLE – инструкции, выполняющие операции «сравнение», «меньше» и «меньше или равно».
* FSGNJ/FSGNJN/ FSGNJX – инструкция смены знака: первая записывает значение из rs1 со знаком в rs2, вторая с противоположным знаком в rs2, третья вычисляет знак через XOR между знаками rs1 и rs2.
* FMV.X.W/FMV.W.X – перенос значения из float-регистра в integer-регистр и из integer-регистра во float-регистр соответственно.
* FCVT.W.S – конвертирует из float представления в 32-бит integer.
* FCVT.S.W – конвертирует из 32-бит integer в float представление.
* FCVT.WU.S/FCVT.S.WU – соответствует инструкциям FCVT.W.S/FCVT.S.W, но не учитывает знак числа.
* FCLASS – инструкция определяет класс float числа. Таблица 1 демонстрирует работу инструкции.

Таблица 1. Диапазоны классов чисел с плавающей точкой одинарной точности.

|  |  |  |
| --- | --- | --- |
| Результат | Расшифровка | Значение rs1 |
| 0 | Отрицательная бесконечность | 0x7F800000 |
| 1 | Отрицательные нормализованные числа | от 0x80080000 до 0xFF7FFFFF |
| 2 | Отрицательные денормализованные числа | от 0x80000001 до 0x807FFFFF |
| 4 | -0 | 0x80000000 |
| 8 | +0 | 0x00000000 |
| 16 | Положительные денормализованные числа | от 0x00000001 до 0x007FFFFF |
| 32 | Положительные нормализованные числа | от 0x00800000 до 0x7F7FFFFF |
| 64 | Положительная бесконечность | 0xFF800000 |
| 128 | -NaN | от 0xFF800000 до 0xFFFFFFFF |
| 256 | NaN | от 0x7F800001 до 0x7FFFFFFF |

В таблице 2 приведены команды, которые реализуются на С в рамках выполнения работы. Инструкции FRCSR, FSCSR, FLW, FSW не реализуются по причине того, что не исполняются непосредственно арифметическим модулем FPU, а лишь взаимодействуют с регистровым файлом и памятью. Инструкции FDIV и FSQRT из-за того, что в большинстве реализаций стандарта RISC-V они либо выделяются в отдельный от FPU модуль, либо не присутствуют вовсе.

Таблица 2. Формат вызова инструкций RISC-V, реализуемых в разрабатываемом модуле

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Операция | funct7 | funct3 | rs2 | Результат |
| FADD | 0 | 0 | reg | Float |
| FSUB | 4 | 0 | reg | Float |
| FMUL | 8 | 0 | reg | Float |
| FMIN | 20 | 0 | reg | Float |
| FMAX | 20 | 1 | reg | Float |
| FSGNJ | 16 | 0 | reg | Float |
| FSGNJN | 16 | 1 | reg | Float |
| FSGNJX | 16 | 2 | reg | Float |
| FEQ | 80 | 2 | reg | Integer |
| FLT | 80 | 1 | reg | Integer |
| FLE | 80 | 0 | reg | Integer |
| FCVT.W.S | 96 | 0 | 0 | Float |
| FCVT.S.W | 96 | 1 | 0 | Integer |
| FCVT.WU.S | 96 | 0 | 1 | Float |
| FCVT.S.WU | 96 | 1 | 1 | Integer |
| FCLASS | 112 | 1 | 0 | Integer |

Также для расширения RV-F определены специальные значения:

* +0 и -0 – значения 0x00000000 и 0x80000000;
* +бесконечность и -бесконечность – значения 0xFF800000 и 0x7F800000;
* NaN – диапазоны от 0xFF800000 до 0xFFFFFFFF и от 0x7F800001 до 0x7FFFFFFF;

К операциям, приводящим к появлению NaN в качестве ответа, относятся:

* все математические операции, содержащие NaN в качестве одного из операндов, кроме FMIN/FMAX.
* для FMIN/FMAX если оба операнда равны NaN результат NaN, если один из операндов NaN, то результат равен не-NaN операнду.
* [деление ноля на ноль](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BD%D0%B0_%D0%BD%D0%BE%D0%BB%D1%8C);
* деление бесконечности на бесконечность;
* умножение нуля на бесконечность;
* сложение бесконечности с бесконечностью противоположного знака.

Ни одно число не равно NaN, NaN не равно само себе.

# Проектирование аппаратного модуля с плавающей точкой с помощью системы высокоуровневого синтеза

Для реализации модуля FPU были изучены спецификации ядра RISC-V. В результате, были определены все инструкции модуля FPU для RISC-V. Стоит сразу отметить, что от реализации функций деления и нахождения корня пришлось отказаться, так как эти инструкции реализуются в отдельных модулях.

Структура проекта на C, включает в себя три файла:

* double.c - главный файл проект, в котором содержится функция main, из которой вызывается модуль FPU.
* FPU.c - непосредственно файл, в котором реализованы функции с числами с плавающей точкой.
* FPU.h - заголовочный файл, в котором объявлены основные библиотеки и структуры проекта.

Файл FPU.c

В данном файле реализованы функци работы с числами с плавающей запятой, а также функция реализации инициализации структуры, выходных данных.

Код файла приведен в приложение А.

В файле 5 функций реализуют инструкции обработки чисел с плавающей запятой. При написании кода, было решено использовать функции библиотеки math.h, так как их использование повышает производительность и эффективность кода. В результате выполнения кода, из функции FPU, возвращается структура, в которой прописаны все результаты работы модуля FPU.

При написание функций модуля FPU, использовались специальные функции и макросы библиотеки math.h такие как:

* abs - функция возвращает модуль целого числа (тип int).
* fabs - функция вычисляет абсолютное значение (модуль) и возвращает его. Так как код пишется на Си, то в данной функции, определён только один прототип данных(double).
* fmin - функция возвращает меньший из двух аргументов x и y (оба аргумента типа float).
* fmax - функция возвращает больший из двух аргументов x и y (оба аргумента типа float).
* Islees - проверяет соотношение x < y, не генерируя исключительную ситуацию, если один из аргументов не число (NAN).
* Islessequal - проверяет соотношение x <= y, не генерируя исключительную ситуацию, если один из аргументов не число (NAN).
* Isnan - проверяемое число с плавающей точкой. Возвращает 0, если проверяемый аргумент – конечное число или бесконечность.Отличное от нуля значение, если проверяемый аргумент не число (NAN).
* signbit - макрос, который возвращает 0, если число x положительное (больше или равно нулю), иначе отличное от нуля значение.
* fpclassify - возвращает значение, которое указывает на класс числа с плавающей точкой.
* Isgreatereial - макрос, который возвращает 1, если x >= y, и 0, если x < y или один из аргументов NAN

Файл FPU.h

В данном файле, объявлены библиотеки, которые используются в программе, а так же объявлены две структуры:

* typedef struct s\_float - структура, входных данных в модуль FPU.
* typedef struct return\_float - структура выходных данных из модуля FPU.

Код файла приведен в приложение Б.

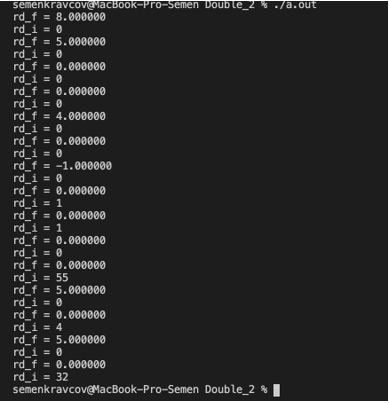
Файл double.h

Основной файл, из которого вызывается модуль FPU, а так же инициализируются массив, входных данных. При помощи данного массива, происходить тестирование модуля FPU.

Код файла приведен в приложение C.

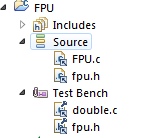
В файле используется функция malloc библиотеки stdlib.h, для выделения памяти под массив структур из 15 элементов в функции static t\_floats\* inizialize\_1(). Полученный массив, возвращается обратно при помощи указателя на голову массива. Затем, при помощи цикла: for, происходить вызов функции FPU, которая имитирует работу модуля FPU, 15 раз.

Результаты работы программы:

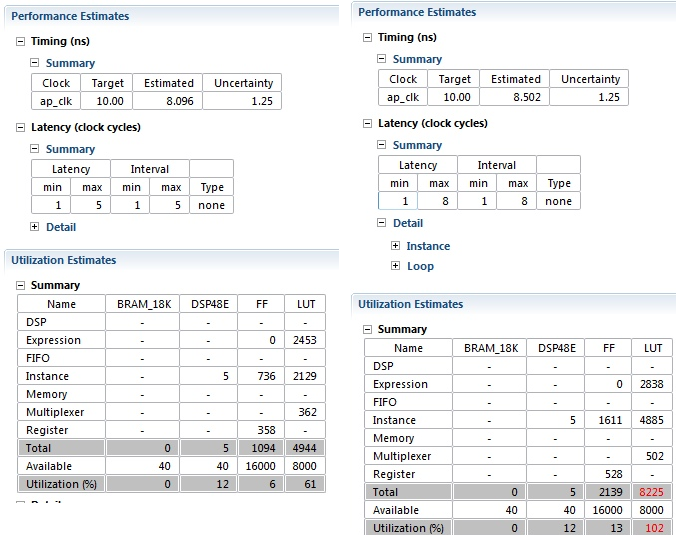


\*Стоит отметить, что в полученных результатах представлены не все данные из модуля FPU. Для этого достаточно в printf заменить rd\_ на rd\_i и заново запустить программу.

В результате, проделанной работы, был написан и протестирован код на языке программирования C, которые реализует поведения модуля FPU. Полученный код, можно использовать на следущей стадии проектирования модуля FPU на ядре RISC\_V.

  
Структура проекта в Vivado HLS.

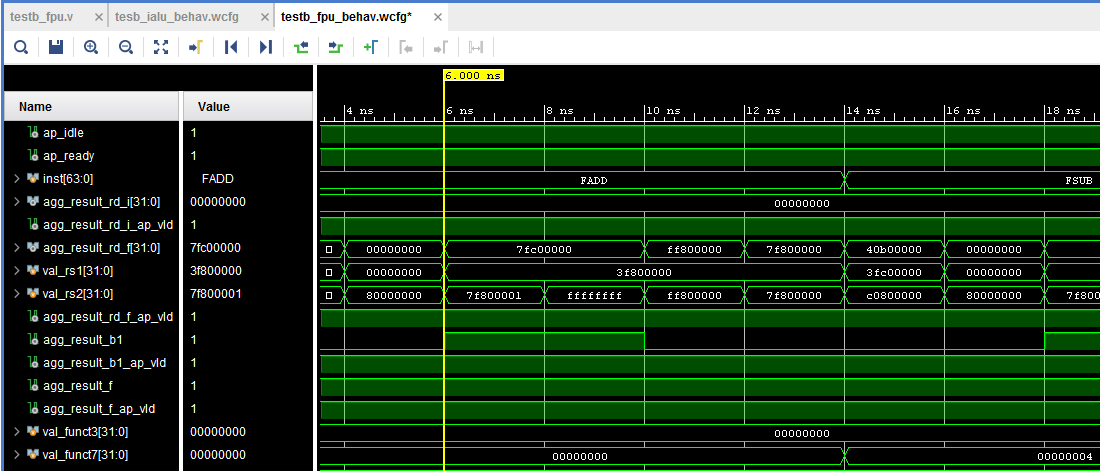
В результате выполнения работы было установлено, что встроенные функции для работы с числами с плавающей запятой из библиотеки math.h дают как меньшую задержку вычислений (5 тактов для самой долгой инструкции против 8 без math.h). Сравнение показателей видно на рисунке.

  
Сравнение результатов синтеза. Справа – без использования функций библиотеки math.h, слева – с ними.

# Отладка спроектированного модуля в среде RTL-симулятора

Для отладка синтезированного модуля проводилась в проекте arty\_scr1. Тестирование включало в себя отладку в среде симулятора Vivado, так возможность синтеза вычислителя. В результате выполнения этапа получена рабочая прошивка для ПЛИС.

Для тестирования был составлен отладочный файл testb\_fpu.v. Каждая итерация тестов отправляет номер инструкции, значения операндов, после чего запускает работу FPU и считывает флаги ошибки, значения регистров результата.

  
Часть интерфейса симулятора с временной диаграммой, показывающей корректное выполнение инструкций FMUL и FSUB.

Юнит-тесты проверяют арифметические действия, то те операции, в результате которые будет получено обычное число, а не специальное значение, и краевые случаи, то есть результаты операций со специальными значениями. К специальным значениям относятся 0, бесконечность, NaN и их отрицательные аналоги. Поскольку проверить все комбинации входных значений в рамках проекта невозможно, арифметические действия были проверены на базовую функциональность: работу с отрицательными и дробными числами. Краевые условия проверялись на взаимодействия числа с отрицательным нулем (0+(-0)), взаимодействие с числа с NaN, –NaN, бесконечностью и отрицательной бесконечностью. Результаты сравнивались с результатами аналогичных операций на C, для перевода десятичных значений использовался инструмент float-to-hex [7].

На этом этапе было определено, что модуль не делает различия между NaN и –NaN, а hex-представления NaN выглядит как 0x7FC00000.

# Оценка конфигурируемости системы

Для оценки конфигурируемости системы было созданы 4 реализации: комбинационная, однотактная, многотактная и конвейерная. Для того, чтобы изменить конфигурацию, необходимо добавить Verilog-файлы в проект из папки FPU\_versions и собрать из tcl-скриптов IP-модули. Для того, чтобы запустить tcl-скрипт в консоли Vivado необходимо ввести команду:

source *путь\_к\_файлу*/*файл*.tcl

Директива **pipeline** позволяет конвейерезировать исполнение функции. Систаксис функции:

#pragma HLS pipeline II=<int>

, где II – интервал инициации, количество тактов между вводом новой порции данных.

Директива была применена ко всей функции FPU:

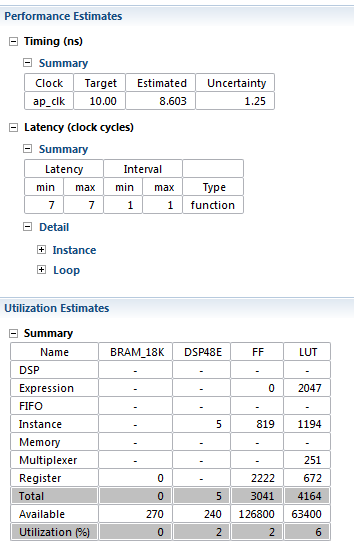
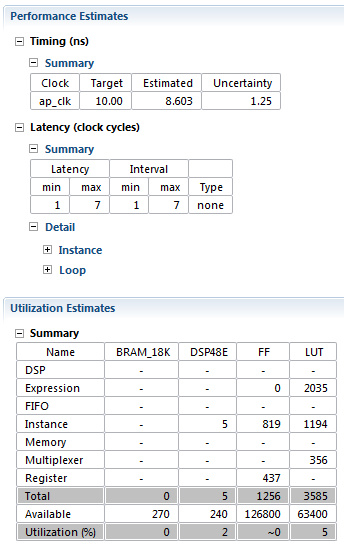
return\_floats **FPU**(t\_floats val){

**#pragma** HLS pipeline II=1  
...

В результате был успешно был синтезирован конвейер с интервалом 1 такт. Глубина конвейера – 8 уровней:

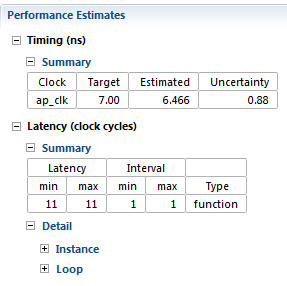
INFO: [SCHED 204-61] Pipelining function 'FPU'.  
INFO: [SCHED 204-61] Pipelining result : Target II = 1, Final II = 1, Depth = 8.

Количество регистров возросло в 3 раза.



Сравнение результатов. Слева – без использования директивы, слева – с директивой.

Минимальный период тактового сигнала удалось получить при глубине конвейера 12.



Минимальный период.

Директива **latency** позволяет задать максимальную и минимальную задержку в тактах. Синтаксис:

#pragma HLS latency min=<int> max=<int>

Чтобы получить нулевую задержку, период тактового сигнала пришлось снизить с 10нс до 40нс. Для этого в начале каждой функции была задана директива с максимальной задержкой равной 0. При задержке 1 такт, минимальный период уменьшается до 30нс.

return\_floats **FPU**(t\_floats val){

**#pragma** HLS latency max=0  
...

**static** return\_floats **FCLASS**(t\_floats val, return\_floats val\_out){

**#pragma** HLS latency max=0  
...

**static** return\_floats **FCVTWS\_FCVTSW\_FCVTWUS\_FCVTSWU**(t\_floats val, return\_floats val\_out){

**#pragma** HLS latency max=0  
...

**static** return\_floats **FSGNJ\_FSGNJN\_FSGNJX**(t\_floats val, return\_floats val\_out)

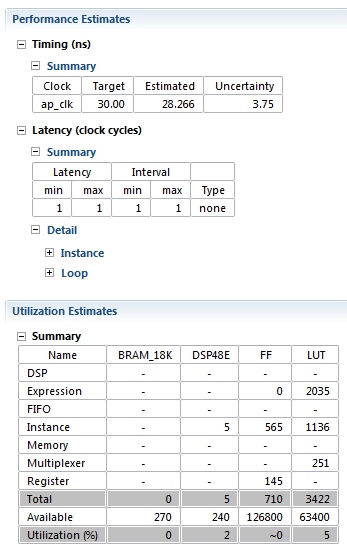
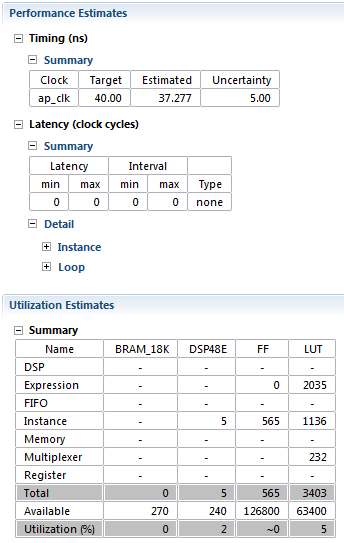
{

**#pragma** HLS latency max=0  
...

**static** **int** **FEQ**(**float** rs1, **float** rs2)

{

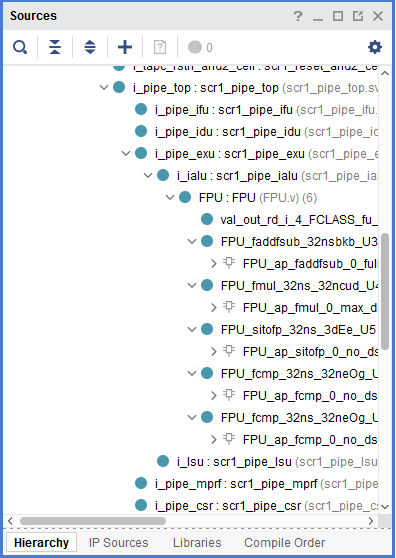
**#pragma** HLS latency max=0  
...



Задержка 0 (слева), задержка 1 (справа).

# Интеграция модуля FPU в проект SCR1

Для интеграции с SCR1 выбрана комбинационная реализация, поскольку все инструкции базового набора в ialu выполнены в виде комбинационной схемы. Структура FPU, интегрированного в SCR1 представлена на рисунке ниже.

  
Структура SCR1 с FPU.

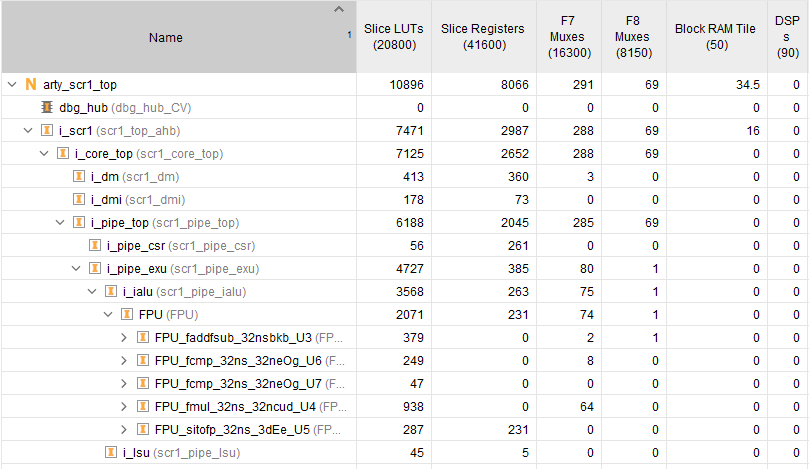
**Внесённые изменения**:

Были изменены idu, ialu, scr1\_riscv\_isa\_decoding.

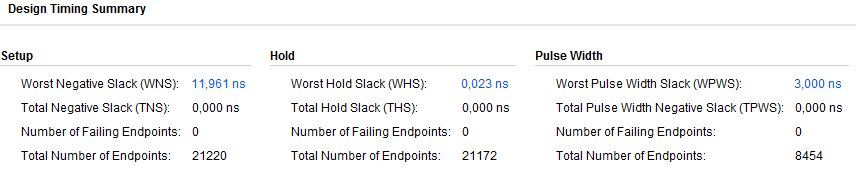
* Модуль idu – декодер инструкций, добавлены ответвления case, соответствующие OPCODE FPU, а также кодам функций (func3 и func7).
* Модуль ialu – подключён модуль FPU и добавлен декодер номера команд. Декодер необходим в силу несоответствия интерфейсов ialu и fpu: в fpu требует коды func3 и func7, в то время как ialu исполняет команду по порядковому номеру в idu (сигнал cmd).
* В файле scr1\_riscv\_isa\_decoding добавлен OPCODE FPU в перечисление type\_scr1\_rvi\_opcode\_e и номера инструкций в перечисление type\_scr1\_ialu\_cmd\_sel\_e.

**Конфигурация**:

Синтез происходил в конфигурации RV32I, заданной в scr1\_arch\_description с 4-стадийным конвейером. Результаты синтеза можно видеть в таблице 3 и на рисунке ниже. Синтез происходил для ПЛИС Artix7. В настройках IP-core были отключено использование DSP-блоков.



Потребление ресурсов ПЛИС.



Результаты статического анализа.

Таблица 3. Характеристики разработанной аппаратуры

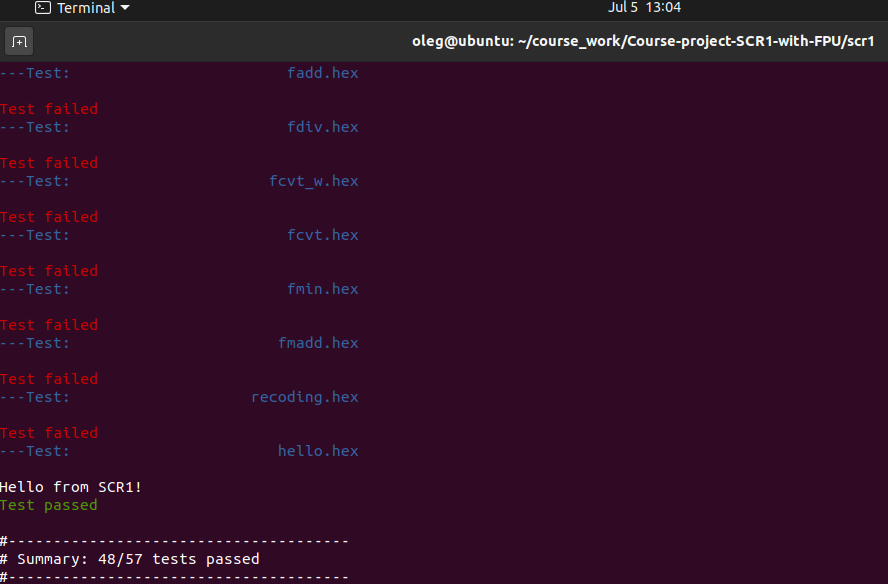
|  |  |  |
| --- | --- | --- |
| Характеристика | FPU | SCR1 |
| LUT / % | 2071/9.96% | 10896/52.38% |
| FF / % | 231/0.56% | 8066/19.39% |
| Тактовая частота | 25 МГц | |
| WNS/WHS | 11.921/0.023 | |

# Проведение тестов в среде Verilator

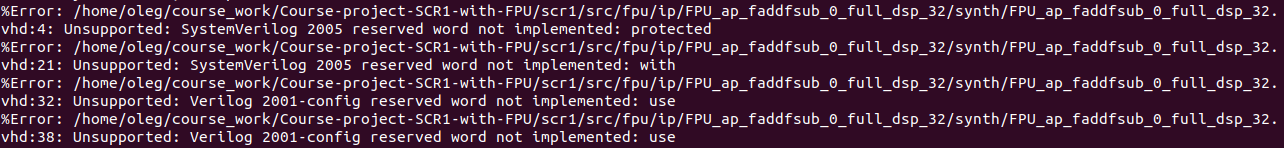
Среда отладки SCR1 поддерживает работу симуляторов ModelSim, Synopsys VCS, Cadence NCSim, Verilator. Про системных тестов быбран симулятор Verilator, поскольку является основным для проекта SCR1, а также потому, что он уже использовался нами в других работах курса.

Отладка процессора в Vivado оказалась невозможна, поскольку симулятор Vivado не смог обработать тестовый топ-файл (ahb\_tb), однако была проведена отладка ialu в тестовом файле ialu\_testb.

В файле scr1/sim/tests/riscv\_isa/rv32\_tests.inc были подключены файлы с тестами RVF. Для проверки корректности тестов было закомментировано объявление FPU в ialu, запущены тесты в verilator. Все тесты скомпилировались и сработали.

  
Тесты и модифицированные файлы

В файле scr1/src/core.files были добавлены файлы FPU. При запуске тестов возникла ошибка компиляции – используемые IP-модулями функции не поддерживаются Verilator, поэтому протестировать проект в этой среде оказалось невозможно.

  
Ошибка при подключении IP-модулей.

# Вывод

В ходе работы были изучены особенности проектирования аппаратуры при помощи средств высокоуровневого синтеза Vivado HLS и был спроектирован исполнительный модуль, исполняющий типовые инструкции работы с числами с плавающей запятой.

Спроектированный модуль операций с плавающей точкой выполняет все заявленные в таблице 2 операции, помимо операций перевода числа из представления с плавающей запятой в целочисленное представление, при интеграции также удалена функция определения класса числа. Модуль был протестирован в среде Vivado. System-тесты не были проведены по той причине, что среда Verilator не способна обработать IP-модули Vivado.

К положительным качествам Vivado HLS, как средства разработки, можно отнести возможность реконфигурации решения. Спецификация на языке C может быть легко синтезирована как в виде комбинационной схемы, так и многотактовой или конвейерной схемы, при этом возможно регулировать количество тактов, которые требуется для выполнения операции.

К отрицательным качествам Vivado HLS относятся ограничение на использование результатов синтеза в продуктах других производителей. Многие функции Vivado HLS способен генерировать только в виде готовых IP-блоков, для которые требуется отдельная поддержка.

# Перечень дополнительных материалов

1. Volume 1, Unprivileged Spec v. 20191213 - <https://riscv.org/specifications/isa-spec-pdf/>, спецификация RISC-V
2. Vivado Design Suite User Guide High-Level Synthesis UG902 - <https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug902-vivado-high-level-synthesis.pdf>, пособие по использованию HLS
3. Floating-Point Design with Vivado HLS - <https://www.xilinx.com/support/documentation/application_notes/xapp599-floating-point-vivado-hls.pdf>, пособие по использованию HLS для float-вычислений
4. HLS Pragmas <https://www.xilinx.com/html_docs/xilinx2019_1/sdaccel_doc/hls-pragmas-okr1504034364623.html>, описание директив оптимизации для HLS
5. SCR1 User Manual <https://github.com/syntacore/scr1/blob/master/docs/scr1_um.pdf>, пользовательский мануал для процессора SCR1
6. SCR1 External Architecture Specification <https://github.com/syntacore/scr1/blob/master/docs/scr1_eas.pdf>, спецификация процессора SCR1
7. Float-to-hex <https://gregstoll.com/~gregstoll/floattohex/>, конвертация значений float в шестнадцатеричное представление.

# Приложение А

**Код файла FPU.c**

#include "fpu.h"

static return\_floats inizialize(){

return\_floats val;

val.b1 = 0;

val.rd\_i = 0;

val.rd\_f = 0;

val.f = 1;

return(val);

}

static int FEQ(float rs1, float rs2)

{

return (islessequal(rs1, rs2) && isgreaterequal(rs1, rs2));

}

static return\_floats FSGNJ\_FSGNJN\_FSGNJX(t\_floats val, return\_floats val\_out)

{

if (val.funct3 == 0)

val\_out.rd\_f = copysignf(val.rs1, val.rs2);

else if (val.funct3 == 1)

val\_out.rd\_f = copysignf(val.rs1, -val.rs2);

else if (val.funct3 == 2)

val\_out.rd\_f = (signbit(val.rs2)==signbit(val.rs1))? copysignf(val.rs1, 1) : copysignf(val.rs1, -1);

else

val\_out.f = 0;

return (val\_out);

}

static return\_floats FCVTWS\_FCVTSW\_FCVTWUS\_FCVTSWU(t\_floats val, return\_floats val\_out){

if(val.funct3 == 0){

if (FEQ(val.rs2, 1.0f))

val\_out.rd\_i = (int)fabsf(val.rs1);

else

val\_out.rd\_i = (int)val.rs1;

}

else if (val.funct3 == 1){

if (FEQ(val.rs2, 1.0f))

val\_out.rd\_f = abs(val.i);

else

val\_out.rd\_f = val.i;

}

else

val\_out.rd\_f = 0;

return (val\_out);

}

static return\_floats FCLASS(t\_floats val, return\_floats val\_out){

if (fpclassify(val.rs2) == FP\_ZERO){

if (signbit(val.rs1)) {

if (fpclassify(val.rs1) == FP\_NAN)

val\_out.rd\_i = 128;

else if (fpclassify(val.rs1) == FP\_INFINITE)

val\_out.rd\_i = 0;

else if (fpclassify(val.rs1) == FP\_ZERO)

val\_out.rd\_i = 4;

else if (fpclassify(val.rs1) == FP\_SUBNORMAL)

val\_out.rd\_i = 2;

else if (fpclassify(val.rs1) == FP\_NORMAL)

val\_out.rd\_i = 1;

}

else {

if (fpclassify(val.rs1) == FP\_NAN)

val\_out.rd\_i = 256;

else if (fpclassify(val.rs1) == FP\_INFINITE)

val\_out.rd\_i = 64;

else if (fpclassify(val.rs1) == FP\_ZERO)

val\_out.rd\_i = 8;

else if (fpclassify(val.rs1) == FP\_SUBNORMAL)

val\_out.rd\_i = 16;

else if (fpclassify(val.rs1) == FP\_NORMAL)

val\_out.rd\_i = 32;

}

}

else

val\_out.rd\_f = 0;

return(val\_out);

}

return\_floats FPU(t\_floats val){

return\_floats val\_out = inizialize();

if (val.funct3 == 0 && val.funct7 == 0)

val\_out.rd\_f = val.rs1 + val.rs2;

else if (val.funct3 == 0 && val.funct7 == 4)

val\_out.rd\_f = val.rs1 - val.rs2;

else if (val.funct3 == 0 && val.funct7 == 8)

val\_out.rd\_f = val.rs1 \* val.rs2;

else if (val.funct7 == 16)

val\_out = FSGNJ\_FSGNJN\_FSGNJX(val, val\_out);

else if (val.funct3 == 0 && val.funct7 == 20)

val\_out.rd\_f = fminf(val.rs1, val.rs2);

else if (val.funct3 == 1 && val.funct7 == 20)

val\_out.rd\_f = fmaxf(val.rs1, val.rs2);

else if (val.funct3 == 2 && val.funct7 == 80)

val\_out.rd\_i = FEQ(val.rs1, val.rs2);

else if (val.funct3 == 1 && val.funct7 == 80)

val\_out.rd\_i = isless(val.rs1, val.rs2);

else if (val.funct3 == 0 && val.funct7 == 80)

val\_out.rd\_i = islessequal(val.rs1, val.rs2);

else if (val.funct7 == 96)

val\_out = FCVTWS\_FCVTSW\_FCVTWUS\_FCVTSWU(val, val\_out);

else if (val.funct3 == 1 && val.funct7 == 112)

val\_out = FCLASS(val, val\_out);

else

val\_out.f = 0;

if (isnan(val\_out.rd\_f) != 0)

val\_out.b1 = 1;

return (val\_out);

}

# Приложение Б

**Код файла FPU.h**

#ifndef FPU\_H

# define FPU\_H

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <math.h>

typedef struct s\_float

{

float rs1;

float rs2;

int i;

int funct7;

int funct3;

} t\_floats;

typedef struct return\_float

{

int rd\_i;

float rd\_f;

bool b1;

bool f;

} return\_floats;

return\_floats FPU(t\_floats val);

#endif

# Приложение В

**Код файла double.c**

#include "fpu.h"

static t\_floats inizialize(){

t\_floats val;

printf("Enter first number: ");

scanf("%f", &val.rs1);

printf("Enter second number: ");

scanf("%f", &val.rs2);

printf("Enter integer number: ");

scanf("%i", &val.i);

printf("Enter funct7: ");

scanf("%d", &val.funct7);

printf("Enter funct3: ");

scanf("%d", &val.funct3);

return (val);

}

static t\_floats\* inizialize\_1(){

t\_floats \*val = (t\_floats\*)malloc(sizeof(t\_floats) \* 15);

val[0].rs1 = 4;

val[0].rs2 = 4;

val[0].i = 5;

val[0].funct7= 0;

val[0].funct3 =0;

val[1].rs1 = 9;

val[1].rs2 = 4;

val[1].i = 5;

val[1].funct7= 4;

val[1].funct3 =0;

val[2].rs1 = 0;

val[2].rs2 = 4;

val[2].i = 5;

val[2].funct7= 8;

val[2].funct3 =0;

val[3].rs1 = 0;

val[3].rs2 = 4;

val[3].i = 5;

val[3].funct7= 20;

val[3].funct3 =0;

val[4].rs1 = 2;

val[4].rs2 = 4;

val[4].i = 5;

val[4].funct7= 20;

val[4].funct3 =1;

val[5].rs1 = 0;

val[5].rs2 = 4;

val[5].i = 5;

val[5].funct7= 16;

val[5].funct3 =0;

val[6].rs1 = 1;

val[6].rs2 = 4;

val[6].i = 5;

val[6].funct7= 16;

val[6].funct3 =1;

val[7].rs1 = 0;

val[7].rs2 = 4;

val[7].i = 5;

val[7].funct7= 80;

val[7].funct3 =0;

val[8].rs1 = 0;

val[8].rs2 = 4;

val[8].i = 5;

val[8].funct7= 80;

val[8].funct3 =1;

val[9].rs1 = 0;

val[9].rs2 = 4;

val[9].i = 5;

val[9].funct7= 80;

val[9].funct3 =2;

val[10].rs1 = 55;

val[10].rs2 = 0;

val[10].i = 5;

val[10].funct7= 96;

val[10].funct3 =0;

val[11].rs1 = 8;

val[11].rs2 = 0;

val[11].i = 5;

val[11].funct7= 96;

val[11].funct3 =1;

val[12].rs1 = 4;

val[12].rs2 = 1;

val[12].i = 5;

val[12].funct7= 96;

val[12].funct3 =0;

val[13].rs1 = 8;

val[13].rs2 = 1;

val[13].i = 5;

val[13].funct7= 96;

val[13].funct3 =1;

val[14].rs1 = 5;

val[14].rs2 = 0;

val[14].i = 5;

val[14].funct7= 112;

val[14].funct3 =1;

return (val);

}

int main(int ac, char \*\*av){

t\_floats \*val\_in = inizialize\_1();

for (int i = 0; i < 15; i++)

{

return\_floats val\_out = FPU(val\_in[i]);

if (val\_out.f == 0){

printf("такой команды нет\n");

// return (0);

}

printf("rd\_f = %f\n", val\_out.rd\_f);

printf("rd\_i = %d\n", val\_out.rd\_i);

}

return (0);

}