Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 23.Б11-мм

Интеграция SCR1 в платформу LiteX для запуска на Colorlight i9

МУКОВЕНКОВ Роман Сергеевич

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
ст. преподаватель кафедры ИАС, Смирнов К. К.

Санкт-Петербург
2025

Оглавление

Введение

Использование FPGA, программируемых логических интегральных схем, в верификации работоспособности ASIC, интегральных схем конкретного применения, — повсеместная практика ввиду возможности перепрограммирования FPGA таким же исходным кодом, написанным на языках описания аппаратуры (далее — HDL). Такой способ верификации позволяет избежать затратных издержек, связанных с остановкой производства и приспособлением фабрик полупроводников к конкретному ASIC при любом изменении. К тому же, если небольшой тираж устройства не оправдывает расходы начала многочисленного производства, то компании могут выпускать коммерческие решения с использованием FPGA, а не ASIC. Программный код, написанный на аппаратном уровне, работает быстрее из-за возможности настоящей параллелизации, конвейеризации, отсутствия конкуренции за ресурсы и накладных расходов на управление потоками. Поэтому в случаях, когда необходимо следовать быстро изменяющимся стандартам и сохранять аппаратную производительность, используют FPGA, например в телекоммуникации [3].

Реализация архитектуры набора команд (ISA, далее — архитектура) на HDL для последующего производства позволяет создавать процессоры, способные к выполнению любого машинного кода архитектуры [1]. Самая популярная на сегодняшний день архитектура процессоров является x86-64, но по причине необходимости лицензии от Intel или AMD для создания воспроизведения, проекты с открытым исходным кодом включают себя другие архитектуры. RISC-V, одна из таких архитектур, допускает коммерческое пользование. Данные реализации архитектур на HDL называют ядрами. Главное преимущество ядра SCR1 на основе архитектуры RISC-V от компании Syntacore — лицензия Solderpad Hardware, что позволяет коммерчески использовать SCR1 при её соблюдении [6]. Для того, чтобы запустить ядро на FPGA, необходимо написать на HDL систему на кристалле (далее — СнК), которая будет соединять это ядро с конкретными аппаратными ресурсами, а также

скомпилировать программу на машинном языке архитектуры [2].

На текущий момент для различных FPGA у SCR1 существует четыре отлаженных СнК от компании Syntacore [6] и один от Луконенко Никиты Игоревича, студента третьего курса кафедры системного программирования [5]. Создание таких СнК — это скрупулёзный процесс, требующий подбора правильно взаимодействующих с аппаратурой компонентов. Некоторые производители FPGA, такие как Xilinx и Intel, имеют данные решения в своём ПО, специально созданном для дизайна систем [2]. Однако этот процесс тоже требует усилий и времени, а обладателям FPGA других производителей требуется самостоятельно подбирать эти компоненты или писать их самому.

Платформа LiteX, написанная на языке Python, — попытка облегчить процесс создания СнК путём абстрагирования ресурсов FPGA плат и требований ядра, которая позволяет автоматически подбирать необходимые компоненты из своей библиотеки [4]. Значительным преимуществом является и упрощённая работа с передаваемыми на FPGA плату файлами с машинным кодом. Любое же нововведение в LiteX, для примера новая функциональность модуля технологии передачи данных Ethernet, автоматически будет применяться и ко всем ядрам и платам. Успешное интегрирование ядра в LiteX будет сопровождаться значительной, если не полной, поддержкой всех FPGA плат, ресурсы которых были абстрагированы в платформе.

Хоть и эффективность, достигаемая индивидуальным решением, недостижима, работающее решение можно легко корректировать, изменяя конфигурацию ядра, FPGA платы или самого LiteX. Возможно очень простое добавление и других модулей, таких как HDMI и Ethernet. Из-за важности такого взаимодействия и потенциальной поддержки порядка 198 различных FPGA плат в данной работе рассматривается начальная попытка интегрировать SCR1 в платформу LiteX с целью запуска ядра на Colorlight i9, плате FPGA с чипом от производителя Lattice.

1. Постановка задачи

Целью работы является запуск ядра SCR1 на плате Colorlight i9. Для её выполнения в рамках текущей учебной практики были поставлены следующие задачи:

1. изучить работу платформы LiteX в объёме, необходимом для успешной интеграции SCR1;

2. рассмотреть работу уже интегрированного в платформу LiteX ядра VexRiscv для проверки взаимодействия Colorlight i9 и платформы LiteX;

3. реализовать описание SCR1 в LiteX в соответствующем Python файле, подготовить исходные файлы для синтеза.

4. интегрировать базовую версию обработчика прерываний, называемого IPIC, в LiteX

2. Обзор

Для правильной интеграции сперва необходимо ознакомление с доступными в LiteX инструментами синтеза для конкретного FPGA чипа. На плате Colorlight i9 установлен ECP5 LFE5U-45F от Lattice. Для данного чипа LiteX предоставляет поддержку двух toolchain: Lattice Diamond и Yosys в связке с nextpnr. Для доступа к toolchain производителя требуется лицензия. Предоставленный GUI интерфейс с библиотекой компонентов позволяет интуитивно изменять сложные решения и получать подробную информацию для быстрой отладки. Ввиду недоступности Lattice Diamond и распространенности toolchain Yosys в связке с nextpnr, было выбрано open-source решение. Исключительно CLI-ориентированная работа с инструментами не предоставляет библиотеки компонентов, что делает интеграцию ядер в LiteX ещё более важной. При этом результат часто не уступает по эффективности от проприетарного.

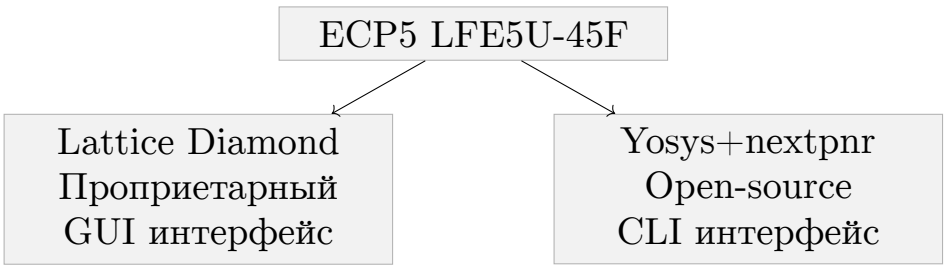

Рис. 1: Выбор toolchain для FPGA ECP5

Определение переменных к вводам и выводам, pin assignment или pinout, для платы Colorlight i9 уже расписано в платформе LiteX в директории platforms. Это входы и выходы, находящиеся всегда на плате — осциллятор, светодиод, флэш-память, SDRAM, и подключаемые внешние — два подключения Ethernet, а также HDMI, UART, Pmod, SD Card.

Помимо этого, конкретные технические компоненты и настройки выбраны в директории targets для правильного выбора из внутренней библиотеки LiteX. Особенно просто можно подключить заранее определенный компонент через опции.

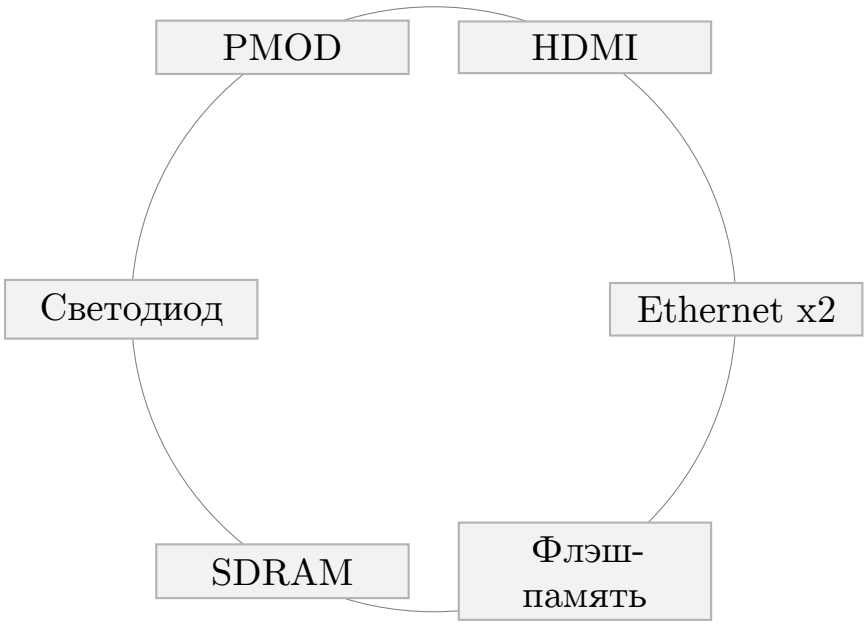

Рис. 2: Периферия Colorlight i9

В главной директории LiteX находятся директории с файлами pythondata-cpu всеразличных ядер, которые являются Python-пакетами. Установив такой пакет через pip, пользователь может обращаться к внутренней функции LiteX для получения файлов ядра

Часть, описывающая ядро, находится в директории cpu. Отдельная директория ядра содержит файлы, определяющие его характеристики и специфичное поведение, связанное с инициализацией сегментов пространства памяти и обработкой запросов на прерывание. В файле core.py определяется тип ядра, ширина данных, кросс-компилятор и флаги для него, само пространство памяти, адрес сброса. Также, определяются количество используемых шин и их интерфейсы. Эти шины в Python объявляются как объекты, которые содержат в себе все необходимые для интерфейса шины переменные и другие объекты. С помощью объектов реализуются и сигналы осциллятора, сброса, прерываний, отладки. Таким образом, в определенном словаре всем названным входам и выходам в высшем модуле Verilog назначается соответствующий объект, что обеспечивает абстракцию.

Файл crt0.S, написанный на языке ассемблера архитектуры ядра, находится в той же директории и реализует специфичные для агностического LiteX BIOS вещи — включение запросов на прерывание, их

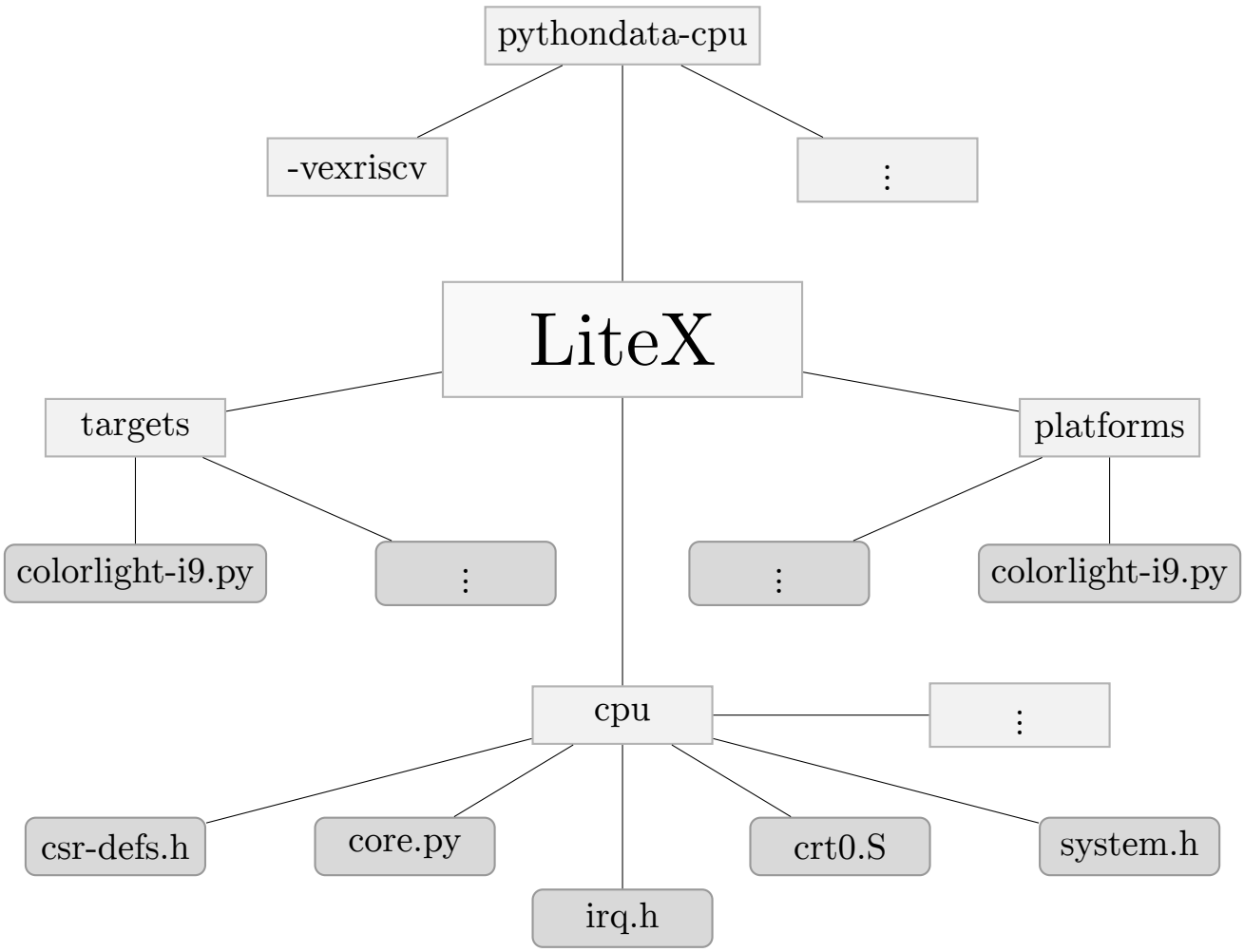

Рис. 3: Абстрагированная структура директорий в LiteX

обработка, инициализация сегментов памяти, в частности стека.

Крайне важным инструментом для отладки запустившегося ядра является симулятор LiteX, который полностью симулирует запуск СнК, а с опцией trace дополнительно создает файл изменения всех внутренних сигналов с расширением vcd. Размер такого файла достигает несколько гигабайтов за пару секунд работы СнК. Такие файлы можно рассматривать в инструменте GTKWave, чтобы попытаться исправить свой СнК.

В процессе интеграции ядра в платформу LiteX также потребовалось воспользоваться инструментом sv2v, который транслирует SystemVerilog в Verilog. Это связано с ограничением функциональности инструмента синтеза Yosys, поддержка SystemVerilog у которого ограничена и доступна лишь в проприетарной версии.

3. Описание решения

Для включения ядра необходима была кнопка. Был выбран поворотный энкодер, на который можно нажать, чтобы послать сигнал. При выборе модулей, необходимо знать электрическое напряжение входов и выходов платы и самой части. Colorlight i9 поддерживает напряжение в 3.3 V, поэтому все приобретенные части имели данное напряжение.

Для того, чтобы передавать UART сигнал, необходим был и преобразователь USB $\leftrightarrow$ UART. Была выбрана модель CH340G-2.

Далее, были установлены Yosys и nextpnr с project trellis. Для переноса решения в LiteX используется также и ecpdap.

К сожалению, было достаточно поздно выявлено, что вход на плате, отвечающий за получение UART сигнала, не работал и всегда выдавал наружу GND, что приводило к неправильной работе преобразователя. Была совершена попытка отладить проблему. Вместо данных с выхода преобразователя был подключен сигнал с кнопки. Кнопка могла подавать сигнал на данный вход. Было понятно, что вход на плате был неисправен. Необходимо было, чтобы другой вход отвечал за принятие сигнала, а потому необходимо было поменять и физическое расположение провода. Проблема решилась копированием файла в platforms colorlight-i5.py, поддерживающего i5 и i9, в colorlight-i9.py и изменением в нём входа, отвечающего за принятие UART сигнала. В такое же название был переименован и одноимённый файл в targets.

3.1. Запуск демо на VexRiscv

Демонстрация интуитивной работы с LiteX. После перехода в директорию targets, необходимо просто запустить соответствующий файл платы с расширением py с опциями build, load, cpu-type и cpu-variant. Опция build создает с помощью установленного по-умолчанию toolchain Yosys и nextpnr необходимый битстрим, который будет передаваться на плату, а load загружает его через JTAG соединение, которое представляет собой обычное USB соединение из-за привязки JTAG к USB на плате Colorlight i9. По-умолчанию загружаться на плату будет самое

поддерживаемое CнK с ядром VexRiscv, поэтому для проверки работы платформы было выбрано именно оно. Была использована максимальная конфигурация.

Спустя пару минут, процесс передачи завершается, а светодиод на плате начинает мигать. Для дальнейшего шага необходимо соединение UART, а также кнопка для передачи сигнала сброса, которая инициирует посылку UART сообщения на компьютер. Нажав на кнопку сброса, пользователь может увидеть, как светодиоды на преобразователе загорелись. Контакт передается на компьютер. Но чтобы принять этот сигнал и посылать свои сообщения, необходима программа терминал на компьютере для взаимодействия. LiteX предоставляет свой терминал, его можно открыть через

```
litex_term /dev/ttyUSBX,
```

где X — номер подключенного преобразователя как устройства Linux.

Теперь можно увидеть, как терминал принимает сигнал от Colorlight i9 и отображает его, как меню. Мы можем вводить команды. Команда help позволяет увидеть всё, что мы можем сделать в LiteX BIOS. Ставить значение у светодиода, манипулировать памятью, запускать что-либо из памяти, проверять скорость чтения и записи. Работоспособность проверена.

Также, при сбросе ядра можно сразу передать свой файл с расширением bin. Демо LiteX является отличной стартовой точкой для разработки собственной программы, запускаемой на данном ядре. Создав этот файл с помощью команды litex_bare_metal_demo, пользователь может передать его через litex_term с опцией kernel. Потом необходимо нажать на кнопку сброса. В демо предоставляются возможности протестировать вывод программы Hello World на языке C, посмотреть на режимы моргания светодиода и на анимированный вращающийся пончик из символов ASCII. Пончик вычисляется на Colorlight i9, а потом отправляется на компьютер через UART.

3.2. Подготовка файлов SCR1 для LiteX

Сперва необходимо было удостовериться, что Yosys может принимать исходные файлы ядра SCR1 и выводить правильный результат. Yosys имеет лишь ограниченную поддержку SystemVerilog и полностью поддерживает Verilog, а ядро SCR1 написано на языке SystemVerilog.

Была проведена попытка синтезировать ядро с этой ограниченной поддержкой. Необходимо было полностью расписать все необходимые пути исходных файлов в файле с расширением ys. Однако Yosys стал выдавать ошибки. Была предпринята попытка их разрешить.

Сперва не поддерживался синтаксический сахар и косметика, такие как return, заменяющий присвоение значения имени функции, и написание названия самой функции через двоеточие после endfunction. В некоторых случаях приходилось подставлять параметры, так как Yosys считал их использование в range неконстантным.

Потом было встречено то, что никак не поддерживалось в Yosys — использование typedef struct packed вне собственного модуля выдавало непонятную ошибку, связанную с range. Были собраны исходные файлы Yosys с отладочными сообщениями, чтобы понять и исправить, почему assert в парсере для range, когда обращались к полям этого типа, не был успешным. Но о невозможности решения своей проблемы стало известно, когда было замечено, что оно просто не поддерживается.

Следующим инструментом для цели был synlig — попытка заставить Yosys поддерживать SystemVerilog. Однако и здесь встретилась более ранняя проблема — ничего не определялось из include файлов.

Была предпринята попытка воспользоваться sv2v для трансляции SystemVerilog в Verilog, а результирующие файлы обработать с помощью Yosys. Данный инструмент воспроизвел логику исходных файлов, а Yosys не выдавал ошибок или предупреждений. Тогда был создан Makefile, который делал это автоматически, а потом запускал nextpnr. В итоге, после обработки sv2v получались синтезируемые исходные файлы. В конце концов, из двух доступных для SCR1 шин AXI4 и AHB-Lite была выбрана шина AXI4, так как она является более производитель-

ной. При этом задача, заключающаяся в сложности её интеграции с СнК, делегируется LiteX.

Необходимо было, чтобы LiteX видел данные файлы. Для этого нужно было создать директорию pythondata-cpu-scr1 и, по примеру других ядер, заполнить информацией для создания Python-пакета. Далее, необходимо просто воспользоваться командой,

```
pip install .
```

Создание файла ядра прошло таким же образом успешно. Был создан core.py, характеризующий процессор. Потом пришлось соединять названия входов и выходов модуля scr1_top_axi с объектами LiteX в словаре.

Ресурс	**VexRiscv**	**SCR1**	**FazyRV**	**picorv32**	**firev**	**Colorlight i9**
DP16KD	49 (45%)	100 (92%)	39 (36%)	41 (37%)	39 (36%)	108 (100%)
TRELLIS_FF	2884 (6%)	4138 (9%)	1771 (4%)	2621 (5%)	2067 (4%)	43848 (100%)
TRELLIS_COMB	6465 (14%)	17407 (39%)	4167 (9%)	6442 (5%)	5351 (12%)	43848 (100%)

Таблица 1: Сравнение использования ресурсов для SCR1 и др. процессоров

3.3. Интеграция обработчика прерываний IPIC

Однако некоторые файлы написать оказалось нелегко. Предстояло написать правильные crt0.S и сопутствующие header файлов, определяющие логику обработчика прерываний и инициализацию сегментов. Сперва была предпринята попытка адаптировать ассемблерный файл SCR1 для LiteX. Файл, используемый с такой же целью в BIOS от Syntacore, слишком сильно интегрирован с его другими компонентами, которые конфликтуют с агностической частью LiteX BIOS. Для поддержки было рассмотрено изменение скрипта компоновщика, находящегося во внутренних файлах LiteX, на похожий в Syntacore BIOS. Файл с расширением bin получался слишком большим, шестнадцать мегабайт, и не помещался в ROM (Read Only Memory). Он по неизвестным причинам заполнялся очень большим количеством нулей.

Вставив заголовочные файлы и crt0.S от ядра cv32e40p на его место, было симулировано отправление через UART меню при нажатии кнопки reset, что означало успешную работу процессора, его взаимодействия с отправкой данных по UART и инициализации сегмента data, в котором располагалось передаваемое меню.

Включая в стартовое сообщение арифметические операции с переменными, объявленными volatile, мы можем удостовериться в успешном выполнении этих операций на FPGA и их передаче на компьютер: сложении, умножении, делении, вычитании.

На плату нельзя было передавать данные через клавиатуру. При просмотре waveform было видно, что при любом нажатии не генерировалось никакое прерывание, а буфер из введённых символов заполнялся полностью без какой-либо обработки. Был сделан вывод, что обработчик прерываний работал неверно.

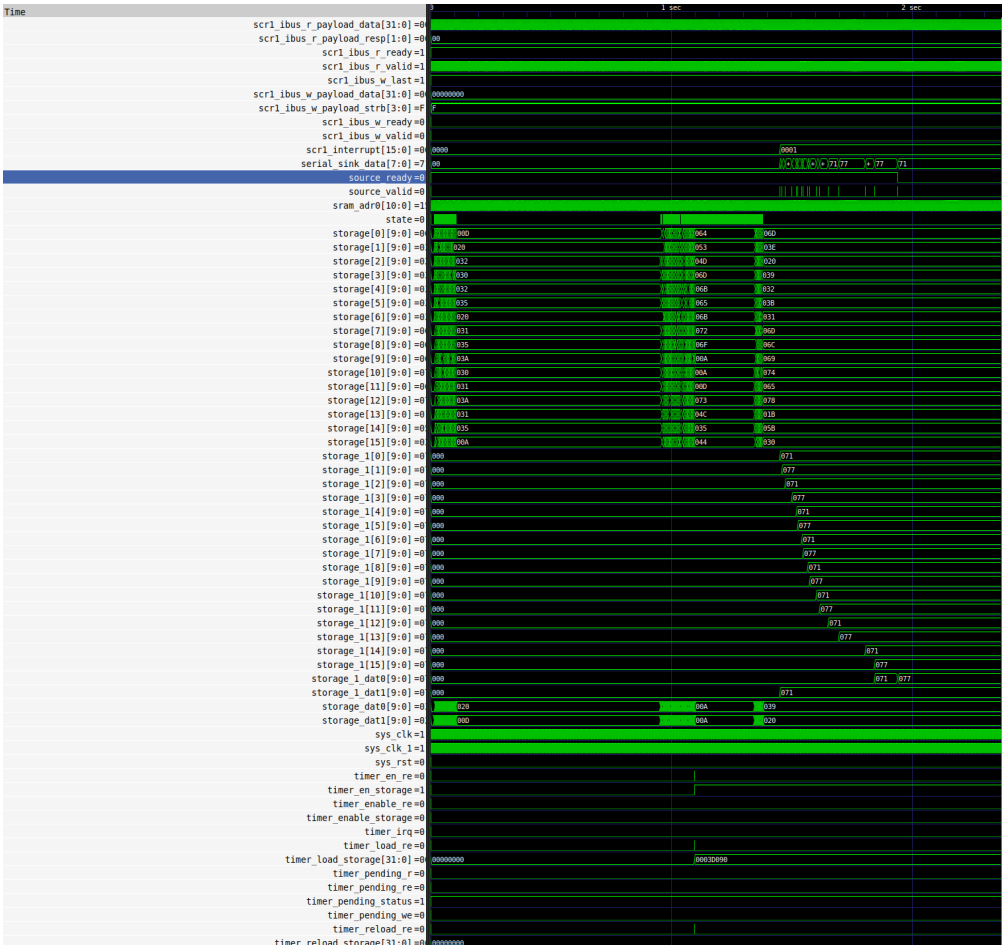

Рис. 4: Waveform, сгенерированная LiteX

Отладка кода через litex_sim с опцией trace навела на верный путь. Был рассмотрен путь irq_lines, который отвечал за прерывания и обрабатывался в модуле IPIC (Integrated Programmable Interrupt Controller), аналогов поддержки которого в LiteX не было. Была использована спецификация к SCR1 к модулю IPIC и Machine Mode CSRs, был вставлен и модифицирован код интерфейса IPIC Syntacore из ответвления Zephyr RTOS. Итеративное исправление ошибок и анализ спецификации позволил адаптировать функции в LiteX.

За обработку прерываний отвечала функция LiteX isr.c, где каждый процесс при девиации поведения объявлял собственную реализацию. Наиболее похожая реализация с IPIC была с PLIC (Platform-Level Interrupt Controller), анализ которой привёл к пониманию общей структуры.

Сперва необходимо было инициализировать IPIC. Для этого существовала функция Syntacore. crt0.S файл должен будет вызывать её, прежде чем переходить к включению глобальных прерываний. Для единственного UART был включен IRQ с вектором 0 (репрезентация в irq_table) на линии 0 (репрезентация физически в irq_lines). За основу crt0.S (нет аналогов, взаимодействующих с IPIC) был взят vexriscv, но мы выключили таймер, оставив только внешние прерывания (передали в mie 0x800 вместо 0x880) и вызвали нашу инициализацию.

После инициализации в main.c происходит включение глобальных прерываний через установление mstatus.MIE единицей. Теперь каждое нажатие на клавиатуру будет вызывать прерывание (нажатие настроено LiteX), в результате чего выполнение переходит на адрес, указанный в mtvec. В нашем случае, это trap_entry, сохраняющий, восстанавливающий регистры и вызывающий isr.

isr использует функции интерфейса soi и eoi, вызывающиеся в начале и конце прерывания. soi возвращает необходимый вектор и делает так, чтобы управление не возвращалось до выполнения eoi. Возвращаемый вектор выступает в роли индекса в таблице прерываний, которую заполнил LiteX автоматически при вызове irq_attach в uart_init.

Функции интерфейса LiteX irq_setmask, irq_getmask и irq_pending

либо не нужны в данной простой реализации единственного прерывания, либо вовсе не требуются. В обобщённой реализации isr от них зависел получаемый вектор, но для SCR1 для этого используется функция soi. Для простоты они были отключены.

И симулятор, и физическая плата стали успешно обрабатывать запросы от UART. Инициализирована память, из неё можно читать и в неё можно записывать, из неё запускать код. Конкретно, работают SPI Flash и SDRAM память. Можно управлять светодиодом. При передаче firmware litex_bare_metal_demo были замечены небольшие проблемы: сначала ничего не выводилось. Надо было указать адрес SDRAM через параметр kernel-adr (в нашем случае 0x10000000), куда необходимо было поместить исполняемый код. Потом, был подкорректирован порядок инициализации UART и включения глобальных прерыванийв main файле демо. Агностичность была сохранена.

Когда вращался пончик, сначала результат представлял собой случайные символы. Поле пересборки файла демо полностью проблема решилась. Пончик вращался, но с более медленной скоростью по сравнению с VexRiscv. В текущей реализации максимальная частота критического пути составляет 23.13 МГц при желаемой частоте в 60 МГц, что может означать нестабильную работу при встрече критического пути. Скорее всего, это связано с размером ядра, заданной максимальной конфигурации. Запущены тесты скорости памяти и сделано поверхностное сравнение. Последующие запуски теста не отличались по скорости.

Параметр	**VexRiscv**	**SCR1**	**FazyRV**	**picorv32**	**firev**
Желаемая частота, МГц	60	60	60	60	60
Частота критического пути, МГц	45.32	23.13	47.93	51.27	54.7
Послед. чт. 2 МиБ SDRAM, МиБ/с	30.2	17.2	10	17.5	24.5
Послед. зап. 2 МиБ SDRAM, МиБ/с	22.1	13.1	8.3	12.5	16.2
Послед. чт. 4 КиБ Flash, МиБ/с	2.8	2.6	2.6	1.7	3.1
Случ. чт. flash 4 КиБ, МиБ/с	1.3	0.768	0.046	0.769	0.17
Скорость пончика	Быстрая	Средняя	Оч. медл.	Медл.	Оч. медл

Таблица 2: Сравнение скорости SCR1 с другими процессорами

В качестве следующей работы предлагается подбор оптимальной конфигурации SCR1. TCM, при этом, работает на первоначальном ад-

ресе 0xF0000000. Последовательная скорость записи и чтения 4 КиБ 33.6 МиБ/сек и 33.6 МиБ/сек соответственно. Была проверена скорость sram (8 КиБ), подключенной LiteX. Она составляла 22.6 МиБ/сек на запись и 20.6 МиБ/сек на чтение 128 бит, а TCM (64 КиБ) при этом имеет скорость 32.1 МиБ/сек на запись и 30.6 МиБ/сек на чтение 128 бит. Быстрым, но неверным решением, будет замена адреса sram в memory map на адрес TCM. Так мы сможем ускорить работу процессора, пусть и не столь заметно. При этом оставшиеся произвольные 56 КиБ пользователь может использовать как угодно.

Также, был проверен crossbar с помощью опции –bus-interconnect crossbar. Он работал быстрее и занимал меньше места: посл. запись SDRAM 4 МиБ 14.5 МиБ/сек и посл. чтение SDRAM 4 МиБ 20.6 МиБ/сек, TRELLIS_COMB: 15613 (35%) и TRELLIS_FF: 4093 (9%). Скорость TCM 4 КиБ: посл. чтение 34 МиБ/сек, запись 32 МиБ/сек. Изменение главных шин тоже концептуально просто, шина axi-lite с crossbar выдаёт наилучшую скорость: посл. запись SDRAM 4 МиБ 15.6 МиБ/сек и посл. чтение SDRAM 4 МиБ 21.6 МиБ/сек, Скорость TCM 256 Б: посл. чтение 34.2 МиБ/сек, запись 41.3 МиБ/сек.

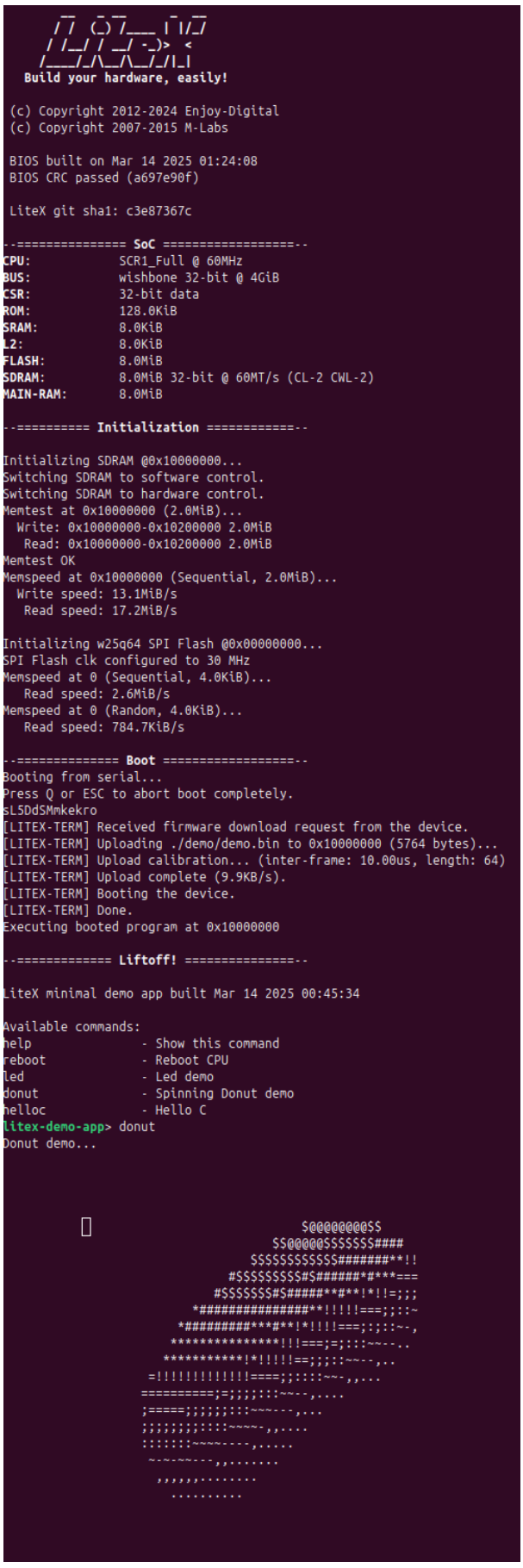

Рис. 5: Консоль взаимодействия LiteX

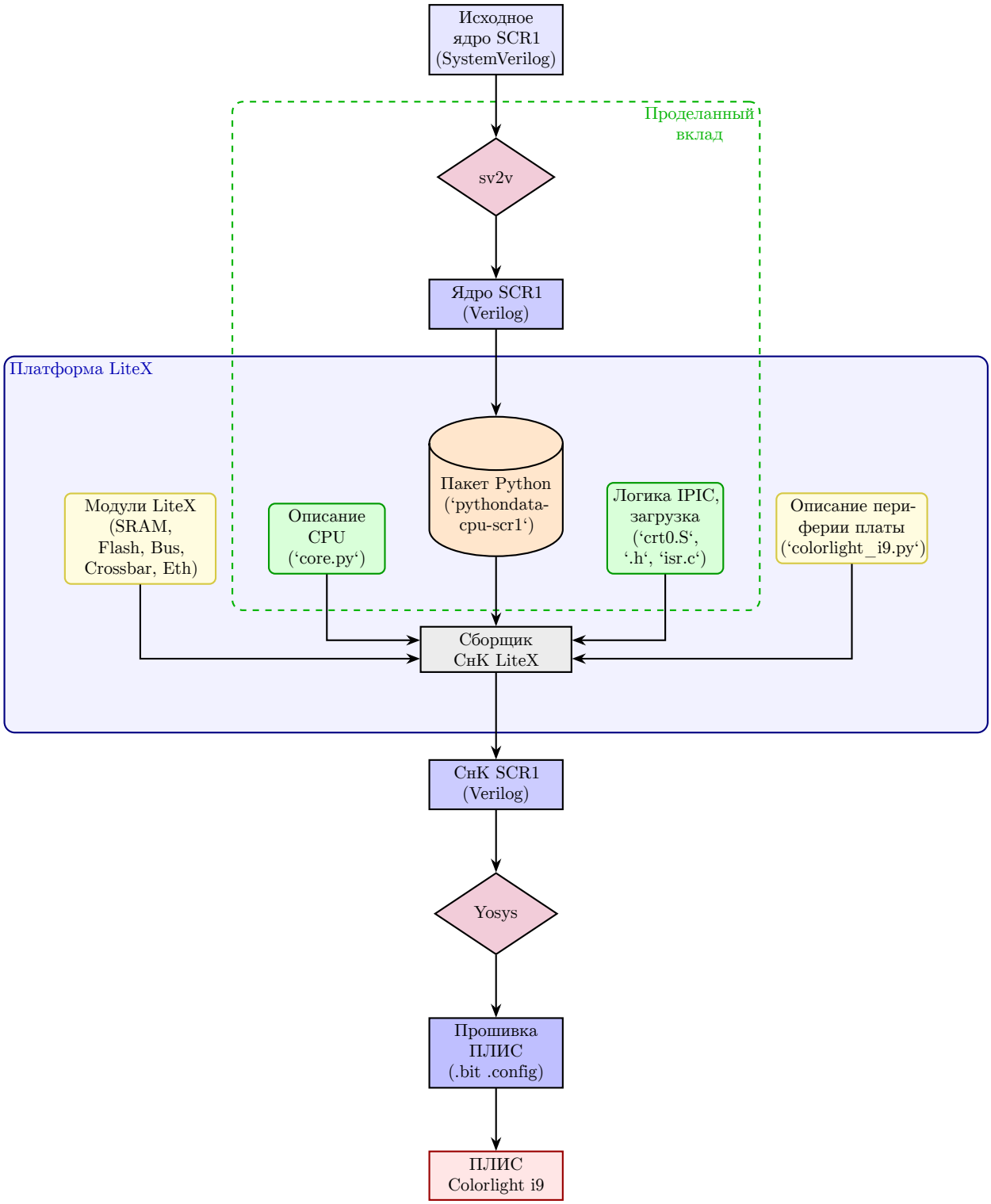

Рис. 6: Проделанная работа, шаги до работы на Colorlight i9

Заключение

Вот каких результатов удалось добиться в ходе учебной практики:

- изучена работа платформы LiteX для успешной интеграции SCR1;
- рассмотрена работа уже интегрированного в платформу LiteX ядра VexRiscv для проверки взаимодействия Colorlight i9 и платформы LiteX;
- реализовано описание SCR1 в LiteX в соответствующем Python файле, подготовлены исходные файлы для синтеза;
- интегрирован IPIC и запущен SCR1 на Colorlight i9 с помощью LiteX

Код доступен в репозитории: https://github.com/Mukovenkov-Roman-Sergeyevich/scr1-colorlight-i9-litex

Список литературы

[1] Charles Roth Lizy John Byeong Kil Lee. Digital Systems Design Using Verilog. — Cengage Learning, 2015. — ISBN: 9781285051079.

[2] Chu Pong P. FPGA Prototyping by VHDL Examples: Xilinx MicroBlaze MCS SoC. — Wiley, 2017. — ISBN: 9781119282747.

[3] Daniel Nenni Paul McLellan. Fabless: The Transformation of the Semiconductor Industry. — CreateSpace Independent Publishing Platform, 2014. — ISBN: 9781497525047.

[4] Enjoy-Digital. LiteX Repository. — URL: https://github.com/enjoy-digital/litex (дата обращения: 27 февраля 2025 г.).

[5] Lukonenko Nikita. SCR1 Tang Primer 20K SoC Repository. — URL: https://github.com/SurfaceYellowDuck/scr1_tang_primer20k (дата обращения: 27 февраля 2025 г.).

[6] Syntacore. SCR1 RISC-V Core Repository. — URL: https://github.com/syntacore/scr1 (дата обращения: 27 февраля 2025 г.).