Санкт-Петербургский государственный университет
Кафедра системного программирования

Интеграция SCR1 в платформу LiteX для запуска на Colorlight i9

МУКОВЕНКОВ Роман Сергеевич, группа 23.Б11-мм

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

Санкт-Петербург
2025

Введение

- Применение FPGA для верификации ASIC
- Использование HDL для описания аппаратных систем с возможностью перепрограммирования
- Преимущества: высокая производительность благодаря параллелизму, конвейеризации и минимальным накладным расходам
- Популярность RISC-V архитектур на HDL и возможность коммерческого использования SCR1 благодаря открытой лицензии Solderpad Hardware

Синтез SCR1

- Первый российский микроконтроллер MIK32 АМУР использует SCR1
- Существует по крайней мере четыре отлаженных СнК от компании Syntacore и один от Луконенко Никиты Игоревича, студента третьего курса кафедры системного программирования
- Были созданы с помощью ручного или полуручного подбора компонентов: контроллера памяти, мостов между шинами, UART модуля
- Используют написанный Syntacore bootloader: sc-bl
- А что если нам хочется полностью автоматизировать подбор компонентов для всех FPGA «в один клик»? Для этого нам нужен LiteX

Преимущества LiteX перед ручным методом

- Написан на Python, имеет библиотеку компонентов, как у проприетарных инструментов вендоров
- Можно описывать свою SoC на Python и добавлять компоненты из этой библиотеки
- Далее, процессор и FPGA абстрагируются
- Поддержка порядка 198 плат, абстрагированных в LiteX, при успешной абстракции процессора
- Будем писать часть, отвечающую за абстракцию ресурсов процессора SCR1, проверим запуском на Colorlight i9

Постановка задачи

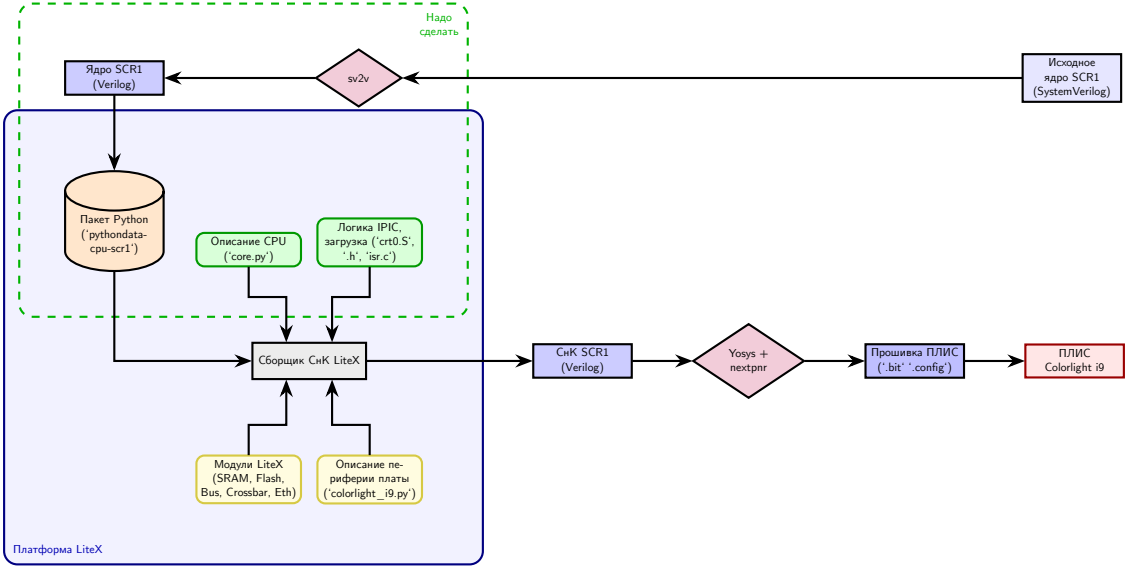
Целью работы является запуск ядра SCR1 на плате Colorlight i9

Задачи:

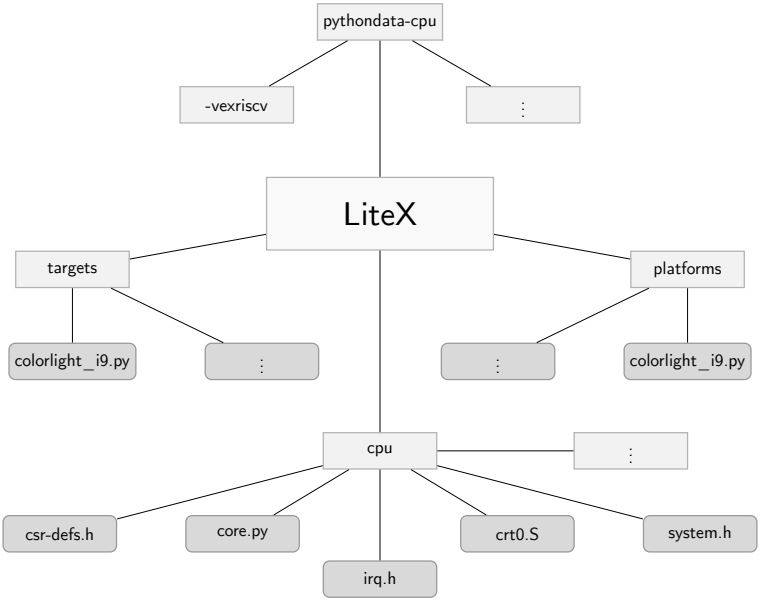
- Изучить работу платформы LiteX в объёме, необходимом для успешной интеграции SCR1
- Рассмотреть работу уже интегрированного в платформу LiteX ядра VexRiscv для проверки взаимодействия Colorlight i9 и платформы LiteX
- Реализовать описание SCR1 в LiteX в соответствующем Python файле, подготовить исходные файлы для синтеза
- Интегрировать базовую версию обработчика прерываний, называемого IPIC, в LiteX

В конце мы запустим SCR1 с LiteX firmware!

Что требуется сделать

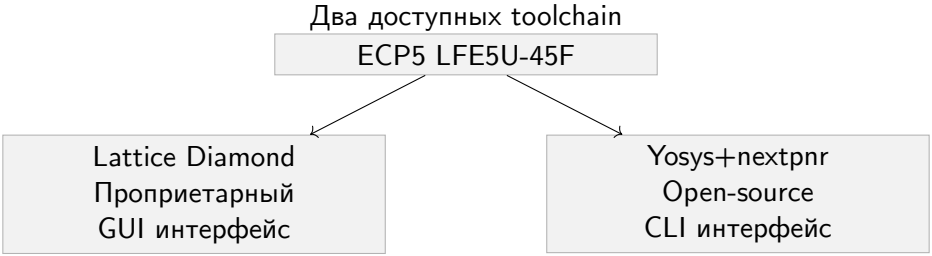


LiteX для абстрагирования процессоров



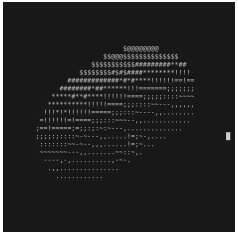
Какие инструменты доступны?

- Плата Colorlight i9 содержит ECP5 LFE5U-45F от Lattice
- Выбран open-source toolchain (Yosys + nextpnr) ввиду его доступности



Демо на VexRiscv

- Для проверки работы платформы для Colorlight i9 синтезировано и отправлено ядро VexRiscv
- Команды:
  - `colorlight_i9.py --build` – сборка битстрима
  - `colorlight_i9.py --load` – загрузка через JTAG
  - `litex_term /dev/ttyUSBX` – терминал взаимодействия через UART
  - `litex_term /dev/ttyUSBX --kernel=<file>.bin` – послать firmware через UART
- Результат: светодиод мигает, через UART появляется меню LiteX BIOS, доступны команды (Hello World на языке C, анимация ASCII-«пончика», управление LED)



Подготовка исходных файлов

- Yosys не полностью поддерживет SystemVerilog, в отличие от Verilog!
- Попытка синтезировать исходные файлы с ограниченной поддержкой вне LiteX
  - Заменить синтаксический сахар (return заменяет присвоение значения имени функции и завершает её)
  - Убрать косметику (endfunction : <название_функции>)
  - Вставить «неконстантные» параметры (использование в range)
- struct packed вне собственного модуля выдавало непонятную ошибку, связанную с range.
  - Были собраны исходные файлы Yosys с отладочными сообщениями, чтобы понять, почему какой-то assert падал
  - К сожалению, typedef struct packed не поддерживались вне своего модуля. Менять логику было крайне проблематично. О проблеме известно, но решение ломает другое использование https://github.com/YosysHQ/yosys/issues/4653
- Был опробован synlig — другой парсер для Yosys, но ничего не определялось из include файлов
- Был выбран инструмент sv2v, который транслировал SystemVerilog в Verilog файлы.
- Транслированные файлы переносились в папку pythondata-cpu-scr1, где устанавливались с помощью `pip install .`

Интеграция SCR1 в LiteX

- Структура класса процессора:
  - Наследование от класса LiteX `CPU`
  - Архитектура rv32i, m и c extensions
- Подключение шин AXI:
  - Две независимые шины для инструкций и данных
  - Сопряжение сигналов AXI:
    - AW/AR/W/B/R каналы
    - Поддержка burst-транзакций
  - К сожалению, синтезировать прямую поддержку AXI не получилось
  - Поэтому в SCR1 LiteX используется Wishbone bridge
  - Параметр –bus-bursting ни на что не влиял
- Память:
  - Адресы основных компонентов:
    - main_ram: 0x10000000
    - CSR и I/O: 0xff000000
    - ROM: 0xfffc0000 (адрес сброса)
    - SRAM: 0xfff00000
    - TCM: 0xf0000000

Интеграция обработчика прерываний IPIC

- SCR1 использует IPIC (Integrated Programmable Interrupt Controller) для обработки прерываний
- LiteX не имеет реализации IPIC, её надо писать самому. Но интерфейс простой!
- Что было сделано:
  - fork Zephyr RTOS от Syntacore имеет реализацию интерфейса IPIC, также пришлось смотреть в спецификацию
  - Инициализация связывает провод-линии прерываний с логическим представлением в таблице — вектором
  - LiteX автоматически заполняет таблицу прерываний, прерывания таймера пока не поддерживаем: в mie 0x800
  - mstatus.MIE включает глобальные прерывания, при нажатии выполнение переходит в адрес в mtvec
  - Сохранение регистров, вызов isr, который берёт вектор из soi и выполняет по этому индексу необходимую функцию
- Результат: UART на FPGA работает, запуск программы с пончиком после некоторых изменений тоже

Сравнение использования ресурсов

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

Таблица: Сравнение использования ресурсов для SCR1 и др. процессоров

- Небольшой поверхностный исследовательский эксперимент!
- Заданные исследовательские вопросы:

  RQ1 : Насколько конкурентоспособен (ресурсы, пропускная способность памяти, ощущаемая скорость пончика) по сравнению с другими СнК?

  RQ2 : Насколько влияет конфигурация в LiteX (шина СнК, тип соединения между шинами) на производительность?
  - ⋆ Поможет определить "бутылочные горлышка"!
- Метрики:
  - занимаемые ресурсы;
  - пропускная скорость памяти;
  - ощущаемая скорость пончика.

Поверхностное сравнение скоростей процессоров

Параметр	**VexRiscv**	**SCR1**	**FazyRV**	**picorv32**	**firev**
Желаемая частота, МГц	60	60	60	60	60
Макс. стабильная частота, 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: Сравнение скорости SCR1 (shared, wishbone) с др. процессорами

SCR1	**SDRAM (МиБ/с)**		**TCM (МиБ/с)**		**Комб. логика**	**Триггеры**
Конфигурация	**Чтение**	**Запись**	**Чтение**	**Запись**	**TRELLIS_COMB**	**TRELLIS_FF**
shared, wishbone	17.2	13.1	33.6	33.6	17407 (39%)	4138 (9%)
crossbar, wishbone	20.6	14.5	34.0	32.0	15613 (35%)	4093 (9%)
crossbar, axi-lite	21.6	15.6	40.1	42.7	17423 (39%)	4299 (9%)

Таблица: Сравнение производительности и ресурсов различных конфигураций SCR1

Обсуждение результатов эксперимента

RQ1 : Процессор SCR1 имеет среднюю производительность по сравнению с другими ядрами, во многом из-за конвертации шин AXI в шины Wishbone, на которых написаны основные компоненты в библиотеке, а также axi-lite. Скорость пончика во многом зависела от наличия модуля быстрого умножения. Только VexRiscv и SCR1 имели данные модули.

RQ2 : Изменение типа соединения памяти и шин с shared на crossbar привело не только к увеличению пропускной способности памяти, но и уменьшению количества используемых ресурсов. К приросту пропускной способности привело и изменение внутреннего стандарта шины с Wishbone на axi-lite, но повысилось количество используемых ресурсов. Оптимизация результата по пропускной скорости памяти и производительности возможна, если добавить в LiteX полную поддержку шины AXI4. Оптимизация результата по ресурсам труднодостижима, но возможна, если пользователю необязательно иметь какой-либо компонент из максимальной конфигурации. К тому же, вариант SCR1 на шине AHB использует меньшее количество ресурсов. Для разработки автоматического СнК SCR1 на AHB можно переиспользовать данный код.

Результаты

В результате работы было сделано:

- Изучена работа платформы LiteX в объёме, необходимом 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
- Интегрирована базовая версия обработчика прерываний, называемого IPIC, в LiteX

Цель выполнена, SCR1 запущено на Colorlight i9

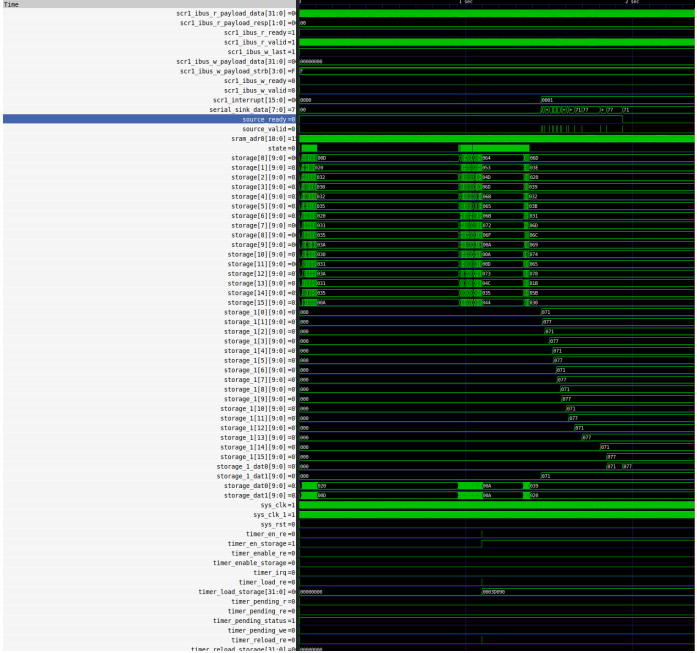


Рис.: Waveform, сгенерированная LiteX

Рис.: LiteX подобрал нужные компоненты: IRQ, CSR, Bus, ROM, SRAM, Interconnect

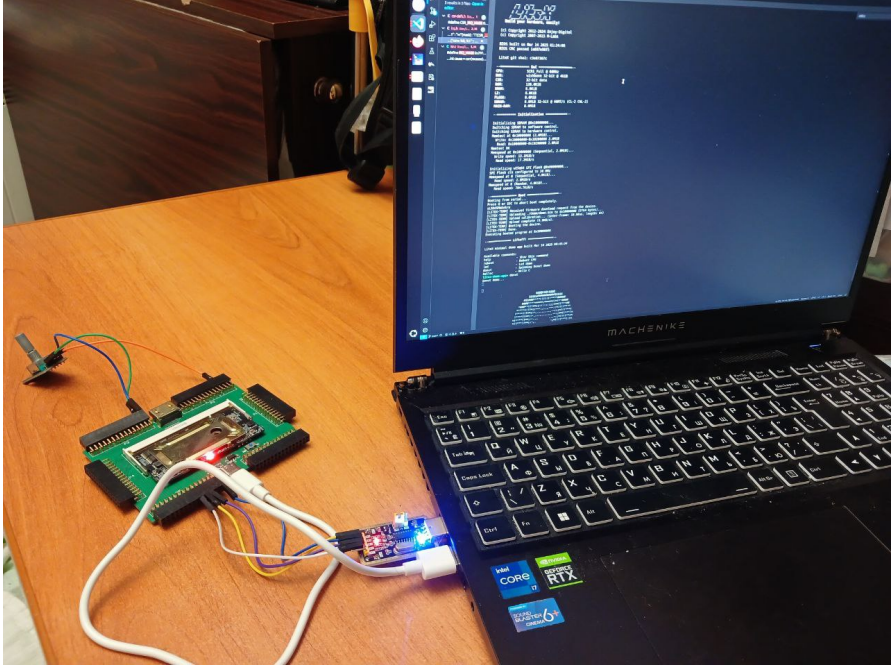


Рис.: Как настроена связь между компьютером и FPGA

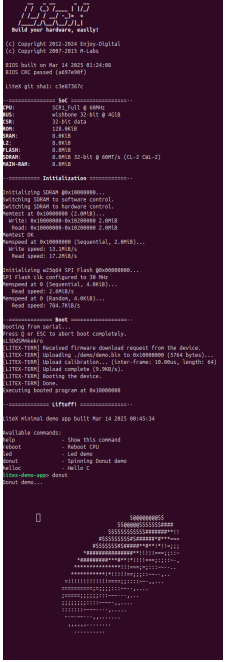


Рис.: Консоль взаимодействия LiteX