

# Проектирование систем на кристалле с программируемой архитектурой

## Лабораторная работа №3

### Цели работы:

1. Изучить принципы работы СнК с процессорной ячейкой на базе RISC-V;
2. Имплементировать модуль, разработанный в предыдущих Л/Р в модуль верхнего уровня представленной СнК;
3. Написать программу на языке программирования C, реализующую обращение из процессорного модуля в имплементированный APB3-Slave модуль;
4. Скомпилировать программу и запустить симуляцию проекта в САПР Synopsys VCS, при необходимости добиться корректной работоспособности APB3-Slave модуля.

### Теоретическая часть

Данная лабораторная работа нацелена на изучение принципа работы общей симуляции большого проекта СнК с интегрированным процессором на базе RISC-V, а также общий принцип проведения «in-system» тестов проекта. Структура рассматриваемого проекта представлена на рисунке 1.

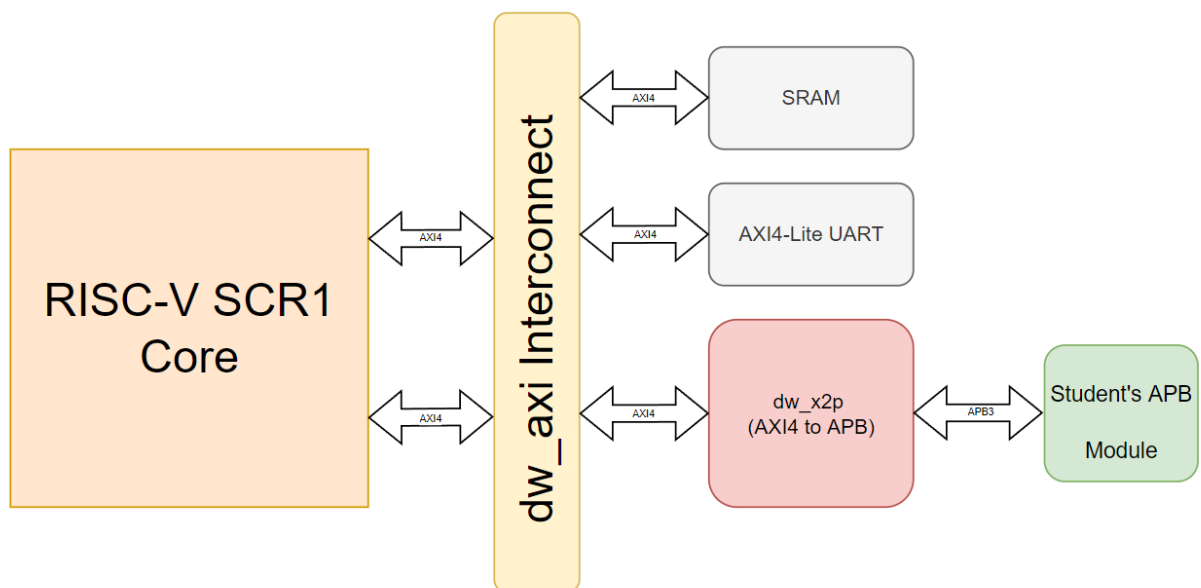


Рисунок 1 – Структура учебного СнК проекта для Л/Р 3

Авторы:

Любавин Кирилл Дмитриевич

Калистратов Олег Александрович

2022

Конструкции языка программирования C, с которыми необходимо ознакомиться:

**Стандартная библиотека `stdint.h`** широко используется при написании Embedded ПО для СнК/ПО/FPGA. Позволяет в явном виде задавать переменные различного размера (в битах). К примеру, список доступных переменных из данной библиотеки, которые мы будем использовать в процессе выполнения данной Л/Р.

- `uint8_t`        8-битовый без знака
- `uint16_t`      16-битовый без знака
- `uint32_t`      32-битовый без знака
- `uint64_t`      64-битовый без знака

**Volatile** — ключевое слово языков C/C++, которое информирует компилятором, что значение переменной может меняться из вне и что компилятор не будет оптимизировать эту переменную. Объекты, объявленные как `volatile`, не используются в определенных оптимизациях, так как их значения могут изменяться в любое время. При запросе объекта с ключевым словом `volatile` система всегда считывает его текущее значение, даже если оно запрашивалось в предшествовавшей инструкции. Кроме того, значение объекта записывается непосредственно при присваивании.

```
1  volatile uint32 * statusPtr = 0xF1230000;
```

Здесь **`statusPtr`** указывает на участок памяти, который в любой момент может быть перезаписан. Наша программа, в которой объявлен и проинициализирован этот указатель, не знает, когда это может произойти. От нее тут ничего не зависит. Но благодаря ключевому слову `volatile` можно быть уверенным, что при каждом обращении по этому адресу мы будем получать актуальное изменяемое значение.

**Указатели** представляют собой объекты, значением которых служат адреса других объектов (переменных, констант, указателей) или функций. Указатели - это неотъемлемый компонент для управления памятью в языке Си.

Для работы с указателями в Си определены две операции:

- операция \* (звездочка) — позволяет получить значение объекта по его адресу — определяет значение переменной, которое содержится по адресу, содержащемуся в указателе;
- операция & (амперсанд) — позволяет определить адрес переменной.

Пример использования:

```

1  #include <stdint.h>
2
3  void main()
4  {
5      uint32_t x = 10;
6      uint32_t *p;
7      p = &x;
8
9      printf("Address = %p \n", p);    // Address = 0060FEA8
10     printf("x = %d \n", *p);        // x = 10
11
12     return 0;
13 }
```

**Структуры** — это совокупность переменных, объединенных одним именем, предоставляющая общепринятый способ совместного хранения информации. Объявление структуры приводит к образованию шаблона, используемого для создания объектов структуры. Переменные, образующие структуру, называются членами структуры. Члены структуры также часто называются элементами или полями. Пример описания структуры для устройства из Л/Р 2 и закреплении за ним определённого адреса памяти представлен на рисунке 2.

```

1  #include <stdint.h>
2
3  #define APB3_DEVICE_BASE_ADDRESS    0x400
4
5  typedef struct {
6      uint32_t    input_data;
7      uint32_t    result;
8      uint32_t    flags;
9      uint32_t    status;
10 } my_device_t;
11
12
13 void main() {
14
15     volatile my_device_t* my_device = (my_device_t*) APB3_DEVICE_BASE_ADDRESS;
16
17 }
```

Рисунок 2 – Пример описания структуры и присваивание переменной этой структуры с привязкой к адресу памяти.

## Выполнение лабораторной работы

1. Необходимо подключить компилятор riscv-unknown-elf-gcc в переменную окружающей среды. Для этого необходимо в файл .bashrc (командой gedit ~/.bashrc) добавить следующую строчку:

```
export PATH=/local/pkims06/SOC_PROGRAMMING_ARCH/riscv-gcc-10.2.0/bin:/$PATH
```

2. Скопировать директорию, содержащую проект СнК к себе в локальную директорию. Директория располагается по следующему пути:

```
/local/pkims06/SOC_PROGRAMMING_ARCH/soc_programming
```

3. Скопировать разработанный в предыдущих Л/Р модуль и интерфейс в локальную директорию из пункта 2 в папку soc\_programming/src/rtl/

4. Интегрировать ваш модуль в файл rtl/top.sv в обозначение комментарием место (Рис. 1). Также необходимо добавить ваш интерфейс APB3 (на примере APB3\_sec) и связать его уже с имеющимся исходным APB3-интерфейсом.



Рисунок 3 – Место инстанцирования модуля и пример его инстанцирования

5. Добавить в файл rtl/rtl\_files.f относительные (не абсолютные) пути к файлам из предыдущих ЛР (Рис. 4)


<pre> 1 axilite_uart/axiluart.v 2 axilite_uart/rxuart.v 3 axilite_uart/rxuartlite.v 4 axilite_uart/skidbuffer.v 5 axilite_uart/txuart.v 6 axilite_uart/txuartlite.v 7 axilite_uart/ufifo.v 8 axilite_uart/wbuart.v 9 10 apb3_if.svi 11 axi4_pkg.sv 12 axi4_if.svi 13 fifo_v3.sv 14 axi2mem.sv 15 generic_memory.sv 16 scr1_wrapper.sv 17 top.sv </pre>		<pre> 1 axilite_uart/axiluart.v 2 axilite_uart/rxuart.v 3 axilite_uart/rxuartlite.v 4 axilite_uart/skidbuffer.v 5 axilite_uart/txuart.v 6 axilite_uart/txuartlite.v 7 axilite_uart/ufifo.v 8 axilite_uart/wbuart.v 9 10 apb3_slave/apb3_slave_pkg.sv 11 apb3_slave/apb3_slave.sv 12 apb3_slave/crc-16-ccitt.sv 13 14 apb3_if.svi 15 axi4_pkg.sv 16 axi4_if.svi 17 fifo_v3.sv 18 axi2mem.sv 19 generic_memory.sv 20 scr1_wrapper.sv 21 top.sv </pre>
--	---	---

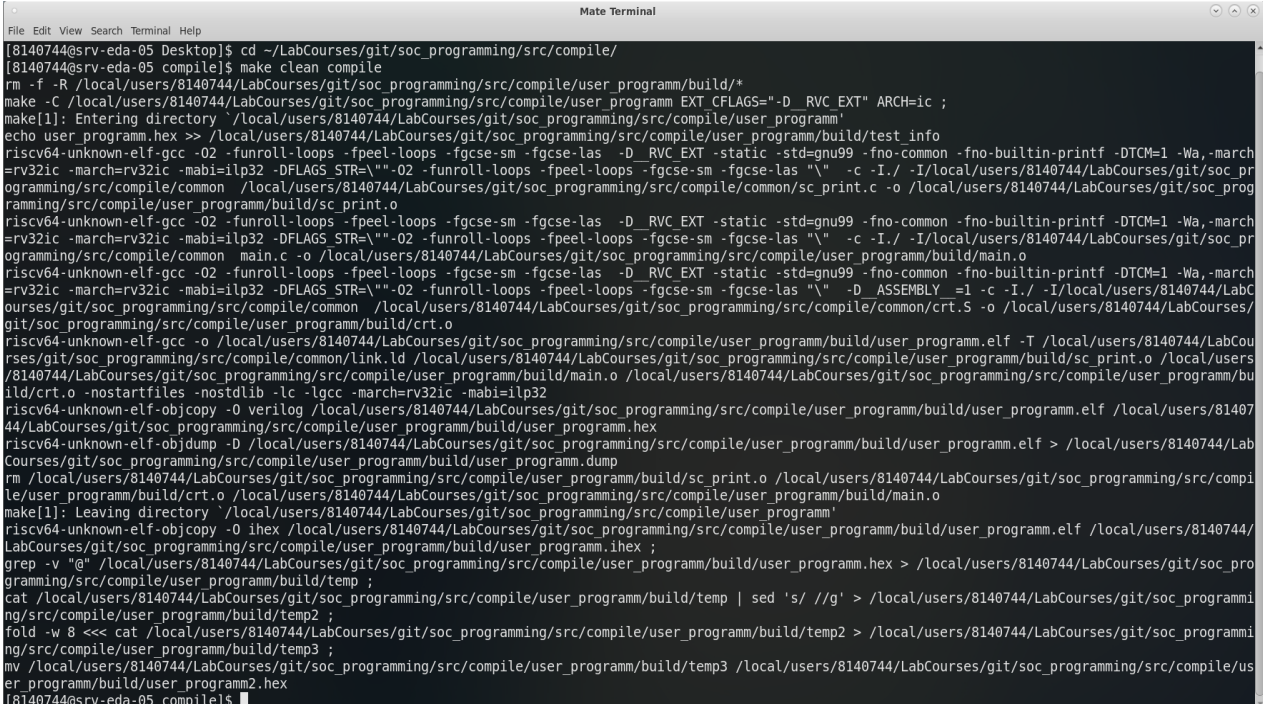
Рисунок 4 – Добавление относительных путей к вашим файлам

6. Написать исполняемый код на ЯП С в файле `compile/user_programm/main.c`

7. Скомпилировать код с помощью следующей команды (выполнять из папки `compile`)

**make clean compile**

В консоли не должно быть каких-либо ошибок компиляции. Пример успешной компиляции представлен на рисунке 5.



```

[8140744@srv-eda-05 Desktop]$ cd ~/LabCourses/git/soc_programming/src/compile/
[8140744@srv-eda-05 compile]$ make clean compile
rm -f -R /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/*
make -C /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm EXT_CFLAGS="-D_RVC_EXT" ARCH=ic ;
make[1]: Entering directory '/local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm'
echo user_programm.hex >> /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/test_info
riscv64-unknown-elf-gcc -O2 -funroll-loops -fpeel-loops -fgcse-sm -fgcse-las -D_RVC_EXT -static -std=gnu99 -fno-common -fno-builtin-printf -DTCM=1 -Wa,-march=rv32ic -march=rv32ic -mabi=ilp32 -DFLAGS_STR="" -O2 -funroll-loops -fpeel-loops -fgcse-sm -fgcse-las "" -c -I./ -I/local/users/8140744/LabCourses/git/soc_programming/src/compile/common /local/users/8140744/LabCourses/git/soc_programming/src/compile/common/sc_print.c -o /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/sc_print.o
riscv64-unknown-elf-gcc -O2 -funroll-loops -fpeel-loops -fgcse-sm -fgcse-las -D_RVC_EXT -static -std=gnu99 -fno-common -fno-builtin-printf -DTCM=1 -Wa,-march=rv32ic -march=rv32ic -mabi=ilp32 -DFLAGS_STR="" -O2 -funroll-loops -fpeel-loops -fgcse-sm -fgcse-las "" -c -I./ -I/local/users/8140744/LabCourses/git/soc_programming/src/compile/common main.c -o /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/main.o
riscv64-unknown-elf-gcc -O2 -funroll-loops -fpeel-loops -fgcse-sm -fgcse-las -D_RVC_EXT -static -std=gnu99 -fno-common -fno-builtin-printf -DTCM=1 -Wa,-march=rv32ic -march=rv32ic -mabi=ilp32 -DFLAGS_STR="" -O2 -funroll-loops -fpeel-loops -fgcse-sm -fgcse-las "" -D_ASSEMBLY=1 -c -I./ -I/local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/user_programm.crt.S -o /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/crt.o
riscv64-unknown-elf-gcc -o /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/user_programm.elf -T /local/users/8140744/LabCourses/git/soc_programming/src/compile/common/link.ld /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/sc_print.o /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/main.o /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/crt.o -nostartfiles -nostdlib -lc -lgcc -march=rv32ic -mabi=ilp32
riscv64-unknown-elf-objcopy -O verilog /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/user_programm.elf /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/user_programm.hex
riscv64-unknown-elf-objdump -D /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/user_programm.elf > /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/user_programm.dump
rm /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/sc_print.o /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/main.o
make[1]: Leaving directory '/local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm'
riscv64-unknown-elf-objcopy -O ihex /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/user_programm.elf /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/user_programm.ihex ;
grep -v "0" /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/user_programm.hex > /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/temp ;
cat /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/temp | sed 's/ //g' > /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/temp2 ;
fold -w 8 <<< cat /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/temp2 > /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/temp3 ;
mv /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/temp3 /local/users/8140744/LabCourses/git/soc_programming/src/compile/user_programm/build/user_programm2.hex
[8140744@srv-eda-05 compile]$

```

Рисунок 5 – Пример лога компиляции программы без ошибок

8. Подключить модуль Synopsys VCS следующей командой:

```
module load synopsys/VCS/R-2020.12
```

9. Перейти в папку simulation и запустить симуляцию проекта в САПР Synopsys VCS с помощью следующей команды:

```
make clean sim
```

**10.** В симуляторе вывести waveform симуляции вашего модуля и продемонстрировать его работоспособность (скриншотами).

### **Используемая литература**

1. Программирование на С и С++. Режим доступа: <http://www.c-cpp.ru/>
2. Syntacore SCR1 RISC-V Core. Режим доступа: <https://github.com/syntacore/scr1>