

ECE 497: Special Project

Weekly Report

Week 04

Alexander Lukens Karl Hallsby

Illinois Institute of Technology

February 18, 2021

What We Did

- ▶ Attempted to flash default chip to Alex's FPGA.
- ▶ Successfully passed USB connection to FPGA through to VirtualBox VM, interfaced with FPGA in Vivado
- ▶ Conducted "Hello World" project on FPGA to ensure that bitstream was being sent to FPGA correctly.
- ▶ Used FPGA Prototyping Flow in Chipyard to generate a bitstream for the Arty FPGA board. Strangely, the default "example" project for the Arty did not pass all timing constraints.
 - ▶ Will require additional investigation
- ▶ When creating bitstream in Chipyard, Vivado runs several tests on the design and produces detailed reports in a Chipyard folder.

- ▶ Successfully created custom config file using building blocks provided by Chipyard, produced 8-core “small Rocket core” SoC with otherwise default settings
- ▶ Utilized RISC-V toolchain to compile basic test program (hello world plus simple for loop)
- ▶ Working getting *our* code out to a separate directory, for Git storage.

7.3. Baremetal RISC-V Programs

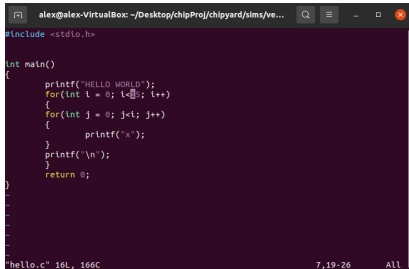
To build baremetal RISC-V programs to run in simulation, we use the riscv64-unknown-elf cross-compiler and a fork of the libgloss board support package. To build such a program yourself, simply invoke the cross-compiler with the flags "-fno-common -fno-builtin-printf -specs=htif_nano.specs" and the link with the arguments "-static -specs=htif_nano.specs". For instance, if we want to run a "Hello, World" program in baremetal, we could do the following.

```
#include <stdio.h>

int main(void)
{
    printf("Hello, World!\n");
    return 0;
}
```

```
$ riscv64-unknown-elf-gcc -fno-common -fno-builtin-printf -specs=htif_nano.specs -c hello.c
$ riscv64-unknown-elf-gcc -static -specs=htif_nano.specs hello.o -o hello.riscv
$ spike hello.riscv
Hello, World!
```

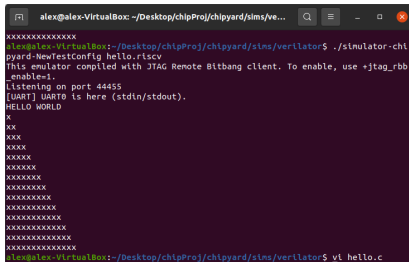
Figure: Commands to compile for RISC-V Simulator



```
alex@alex-VirtualBox: ~/Desktop/chipProj/chipyard/sims/ve...  
#include <stdio.h>  
  
int main()  
{  
    printf("HELLO WORLD");  
    for(int i = 0; i<5; i++)  
    {  
        for(int j = 0; j<5; j++)  
        {  
            printf("x");  
        }  
        printf("\n");  
    }  
    return 0;  
}
```

hello.c" 16L, 166C 7,19-26 All

Figure: hello.c code



```
alex@alex-VirtualBox: ~/Desktop/chipProj/chipyard/sims/ve...  
alex@alex-VirtualBox:~/Desktop/chipProj/chipyard/sims/verilator$ ./simulator-chi  
pyard-NewTestConfig hello.riscv  
This emulator compiled with JTAG Remote Bitbang client. To enable, use +jtag_rbb  
_enable=1.  
Listening on port 4455  
[UART] UART0 is here (stdin/stdout).  
HELLO WORLD  
x  
xx  
xxx  
xxxx  
xxxxx  
xxxxxx  
xxxxxxx  
xxxxxxxx  
xxxxxxxxx  
xxxxxxxxx  
xxxxxxxxx  
xxxxxxxxx  
xxxxxxxxx  
xxxxxxxxx  
xxxxxxxxx  
xxxxxxxxx  
xxxxxxxxx  
xxxxxxxxx  
alex@alex-VirtualBox:~/Desktop/chipProj/chipyard/sims/verilator$ vt hello.c
```

Figure: hello.c output from simulator

What We Learned

- ▶ The repository is incredibly complicated
- ▶ **VERY** deep directory nesting (Partly due to Scala/Java project directory conventions).
- ▶ Putting the generated chip on an FPGA seems to be much more difficult than originally thought.
- ▶ Generating a non-default chip can be very easy or very hard.
 - ▶ Some of the options that must be overridden to ensure a different chip is built and simulated/benchmarked are not easy to understand or find.
 - ▶ Generating these SoC configurations requires **LOTS** of memory.

Next Steps

- ▶ Continue trying to write the default chip out to Alex's FPGA and test.
- ▶ Using Alex's discovery in Vivado, collect gate counts of components within the default chip.
- ▶ Practice generating other non-default chips to understand all the options used when generating a new chip.
- ▶ Hopefully, start defining a new, custom, chip using what we know, and building a very small proof-of-concept.



Alon Amid et al. “Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs.” In: *IEEE Micro* 40.4 (2020), pp. 10–21. ISSN: 1937-4143. DOI: [10.1109/MM.2020.2996616](https://doi.org/10.1109/MM.2020.2996616).