22IS303- COMPUTER ORGANIZATION AND ARCHITECTURE

UNIT - V

IMPLEMENTING RISC-V IN AN FPGA

INTRODUCTION

A field-programmable gate array (FPGA) is a reconfigurable integrated circuit (IC) that lets you implement a wide range of custom digital circuits. Throughout the series, we will examine how an FPGA works as well as demonstrate the basicbuilding blocks of implementing digital circuits using the Verilog hardware description language (HDL).

RISC-V is an open-source instruction set architecture (ISA) that allows anyone to implement a central processing unit (CPU) or system-on-a-chip (SOC) design without paying a licensing fee. As a result, it is popular with FPGA enthusiasts as the starting point for a softcore processor implementation. iN THIS EXAMPLE, we will load a pre-existing RISC-V implementation onto our FPGA and modify the design to allow for button inputs.

IceStick Tutorial

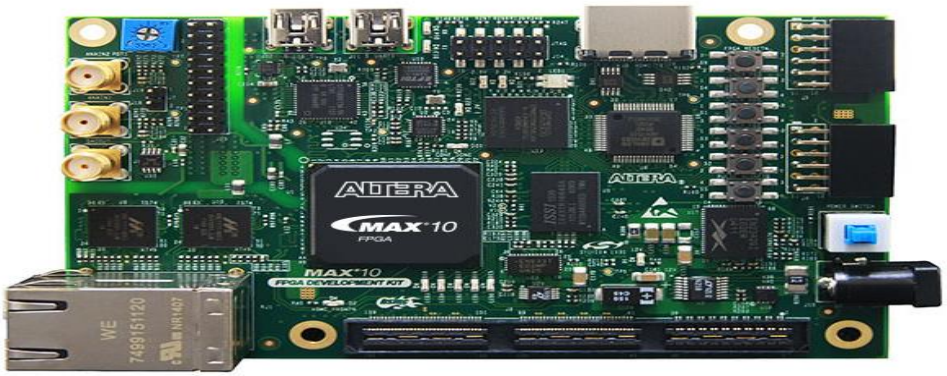

Fig 1.1 FPGA Development Tools

This tutorial will show you how to install FPGA development tools, synthesize a

RISC-V core, compile and install programs and run them on a IceStick. This lets you experience FPGA design and RISC-V using one of the cheapest FPGA devices (around $40).

Install open-source FPGA development toolchain

Before starting, you will need to install the open-source FPGA development toolchain (Yosys, NextPNR etc...), instructions.

(Optional) configure femtosec and femtorv32

Edit learn-fpga/FemtoRV/RTL/CONFIGS/icestick_config.v
This file lets you define what type of RISC-V processor you will create. For IceStick, we use the femtorv32-quark, that has a minimal LUT count and that can run code from the SPI flash. It is best suited for the IceStick that has a small number of LUTs (1380) and very little BRAM (8 kB, but only 6 kB available as RAM, because two kB are used by the registers).
The file learn-fpga/FemtoRV/RTL/CONFIGS/icestick_config.v also lets you select the device drivers present in the associated system-on-chip:

Device	Description	Comment
NRV_IO_LEDS	5 leds	keep it
NRV_IO_UART	serial connection through USB	keep it
NRV_IO_SSD1351	small OLED screen (128x128)	comment-out if you do not have it
NRV_IO_SSD1331	small OLED screen (96x64)	comment-out if you do not have it
NRV_IO_MAX7219	led matrix (10x10)	comment-out if you do not have it
NRV_MAPPED_SPI_FLASH	flash mapped in memory space	keep it

We activate the LEDs (for visual debugging) and the UART (to talk with the system through a terminal-over-USB connection). We use 6144 bytes of RAM. It is not very much, but we cannot do more on the IceStick. You will see that with 6k of RAM, you can still program nice and interesting RISC-V demos. If you do not have an OLED screen or a led matrix screen, you can comment out the corresponding lines in the file. There are other options in the file. Normally you will not need to change them. If you want to know, they are listed below:

Option	Description
NRV_FREQ	Frequency of the processor in MHz
NRV_RAM	Amount of RAM (6KB max)
NRV_RESET_ADDR	Address where to jump at startup and reset (SPI flash + 64KB offset)
NRV_IO_HARDWARE_CONFIG	Hardware register that can be queried by code to see configured devices
NRV_RUN_FROM_SPI_FLASH	Do not initialize BRAM with firmware (firmware will be read from SPI flash)

The default value (66 MHz) corresponds to what is validated by Yosys/NextPNR. If you want you can try overclocking a bit, up to 80 - 90 MHz (but you may experience stability problems then). Note that frequency can only take some predefined values, listed in RTL/PLL/frequencies.txt. You can add your own value there if need be, but then you will need to use RTL/PLL/gen_pll.sh to re-generate the board-specificVerilog for the PLL.

On the IceStick, we use the femtorv32-quark configuration. It deactivates some functionalities (hardware mul, counters), and it is a bit slower (simplified state machine at a cost of a higher CPI), but it has the ability of running the code directly from the SPI flash. It is very useful on the IceStick, because we got 2 MBs of SPI flash, which can be used to store large programs.

Synthesize and program

You can now synthesize the design and send it to the device. Plug the device in a USB port, then:
$make ICESTICK
The first time you run it, it will download RISC-V development tools (takes a while). Now, install a terminal emulator:
$sudo apt-get install python3-serial
(or sudo apt-get install screen, both work). To see the output, you need to connect to it (using the terminal emulator):
$make terminal

Examples with the serial terminal (UART)

The directories FIRMWARE / EXAMPLES and FIRMWARE/ASM_EXAMPLES contain programs in C and assembly that you can run on the device. On the IceStick, only those that use 6K or less will work (list below).
To compile a program:
$cd FIRMWARE/EXAMPLES
$make hello.prog
The .prog target generates the program and sends it to the device's SPI flash, using iceprog. To see the result, use:
$cd ../..
$make terminal
There are several C and assembly programs you can play with (list below). To learn more about RISC-V assembly, see the RISC-V specifications, in particular the instruction set and the programmer's manual.

Fig 1.2 RISC V Assembly Program

Program	Description
ASM_EXAMPLES/blinker_shift.S	the blinker program, using shifts
ASM_EXAMPLES/blinker_wait.S	the blinker program, using a delay loop
ASM_EXAMPLES/test_serial.S	reads characters from the serial over USB, and sends them back
ASM_EXAMPLES/mandelbrot_terminal.S	computes the Mandelbrot set and displays it in ASCII art
EXAMPLES/hello.c	displays a welcome message
EXAMPLES/sieve.c	computes prime numbers

Examples with the LED matrix

Fig 1.3 LED Matrix

It is cheap (less than $1) and easy to find (just google search max7219 8x8 led matrix). Make sure pin labels (CLK,CS,DIN,GND,VCC) correspond to the image, then insert it in the J2 connector of the IceStik as shown on the image. You can also build an OysterPMOD (with both a led matrix and OLED screen using a single PMOD connector).

FemtoSOC configuration

Now we need to activate hardware support for the led matrix (and deactivate the UART). To do that, configure devices
in FemtoRV/RTL/CONFIGS/icestick_config.v as follows:

```
...
`define NRV_IO_MAX7219 // Mapped IO, 8x8 led matrix
...
```

Then you need to re-synthethize and send the bitstream to the IceStick:

```
$make ICESTICK
```

Now you can compile the hello world program (FIRMWARE/EXAMPLES/hello.c). Edit it and uncomment the following line:

```
femtosoc_tty_init();
```

This line automatically redirects printf() to the configured device (here the led matrix). Now compile the program and send it to the device:

```
$cd FIRMWARE/EXAMPLES
$make hello.prog
```

When the led matrix is configured, printf() is automatically redirected to the scroller display routine. The sieve.c program will also behave like that.
There are other examples that you can play with:

Program	Description
ASM_EXAMPLES/test_led_matrix.S	display two images on the led matrix in ASM
EXAMPLES/life_led_matrix.c	Game of life on a 8x8 toroidal world

If you want to write your own program: in C, you first need to switch the display on using MAX7219_init(), then you can use the
function MAX7219(col,data) where col is the column index in 1..8 (and not 0..7 !!!), and data an 8-bit integer indicating which led should be lit. Take a look at FIRMWARE/EXAMPLES/life_led_matrix.c for reference.

Examples with the OLED screen

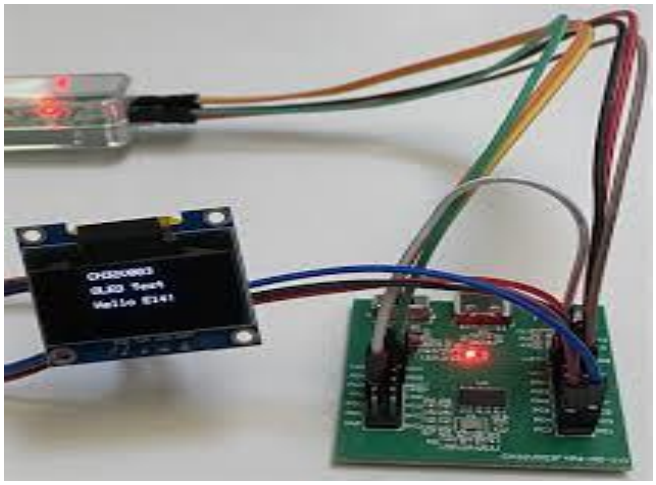

Fig 1.4 OLED Screen

With its 64 pixels, our led matrix is somewhat limited and lacks colors... Let us generate more fancy graphics. For this, you will need a *SSD1351 128x128 oled display*. It costs around $15 (there exists cheaper screens, such as 240x240 IPS screens driven by the ST7789, but they really do not look as good, and they are not compatible, believe me the SSD1351 is worth the price). Make sure you get one of good quality (if it costs less than $5 then I'd be suspicious, some users reported failures with such low-cost versions). Got mine from Waveshare. Those from Adafruit were reported to work as well.

These little screens need 7 wires. The good news is that no soldering is needed, just get a 2x6 pins connector such as the one on the image, connect the wires as shown to the connector, then the connector to the IceStick. If the colors of the wires do not match, use the schematic on the right to know which wire goes where.

You can also build an OysterPMOD (with both a led matrix and OLED screen using a single PMOD connector).

Now you need to reconfigure icestick_config.v as follows:

...

`define NRV_IO_SSD1351 // Mapped IO, 128x128x64K OLed screen

Then you need to re-synthethize and send the bitstream to the IceStick:

$make ICESTICK
Let us compile a test program:

$ cd FIRMWARE/EXAMPLES
$ make gfx_test.prog
If everything goes well, you will see an animated colored pattern on the screen. Note that the text-mode demos (hello.c and sieve.c) still work and now display text on the screen. There are other programs that you can play with:

(The black diagonal stripes are due to display refresh; they are not visible normally).

Program	**Description**
ASM_EXAMPLES/test_OLED.S	displays an animated pattern.
ASM_EXAMPLES/mandelbrot_OLED.S	displays the Mandelbrot set.
EXAMPLES/cube.c	displays a rotating 3D cube.
EXAMPLES/mandelbrot.c	displays the Mandelbrot set (C version).
EXAMPLES/riscv_logo.c	a rotozoom with the RISCV logo (back to the 90's).
EXAMPLES/spirograph.c	rotating squares.
EXAMPLES/gfx_test.c	displays an animated pattern (C version).
EXAMPLES/gfx_demo.c	demo of graphics functions (old chaps, remember EGAVGA.bgi ?).
EXAMPLES/test_font_OLED.c	test font rendering.
EXAMPLES/sysconfig.c	displays femtosoc and femtorv configurations.

The LIBFEMTORV32 library includes some basic font rendering, 2D polygon clipping and 2D polygon filling routines. Everything fits in the available 6kbytes of memory !

Storing stuff on the SPI Flash

The IceStick stores the configuration of the FPGA in a flash memory, and there is plenty of unused room in it: *if it does not fit in one chip, we can overflow in the neighborhing chip !*. This flash memory is a tiny 8-legged chip that talks to the external world using a serial protocol (SPI). In fact we are already using it ! It is where our programs are stored, and FemtoRV32 directly executes them from there. We are also going to store some data there.

Let us copy the data to the SPI flash:

```
$ iceprog -o 1M FIRMWARE/EXAMPLES/DATA/scene1.bin
```

This copies the data starting from a 1Mbytes offset (the lower addresses are used to store the configuration of the FPGA and to store our programs, so do not overwrite them). The data file scene1.bin is the original one, taken from the ST_NICCC demo. Now you can compile the demo program and send it to the IceStick:

```
$ cd FIRMWARE/EXAMPLES
$ make ST_NICCC_spi_flash.prog
```

RAY TRACING

Command

$ make tinyraytracer.prog

It will display the classical shiny spheres and checkerboard scene. It takes around 20 minutes to do so, be patient! This is because not only is our processor not super-fast, but also it does not have hardware multiplication, and it executes floating point software routines from the SPI Flash! It would be faster to copy them in RAM, but they do not fit (remember, we only have 6 KB of RAM).

For smaller programs, it is possible to fit a part of the routines in RAM, take a look at FIRMWARE/EXAMPLES/riscv_logo_2.c (if you have both the led matrix and SSD screen, it will tell you what it is doing). It shows the effect of the magic keyword RV32_FASTCODE: this keyword is expanded as an annotation, indicating to the linker that the corresponding function should be copied in RAM instead of SPI Flash. Since RAM is much faster to access than SPI Flash, you can do that for some small functions that are called often. It is for instance the case of some graphic functions, used by our version of the ST_NICCC demo.
However, the limits of our tiny system are quicky reached. If you want to go further, an easy way is to get an ULX3S. It costs a bit more ($130) but it is worth the price (the on-board ECP5 FPGA is HUGE as compared to the one of the IceStick). Now time to read the ULX3S tutorial !

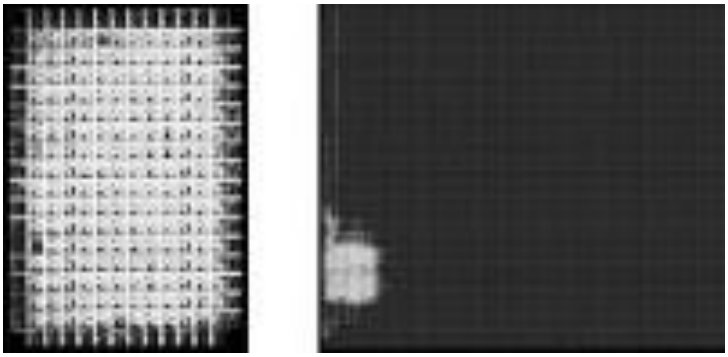

Fig 1.5 Core installed in IceStick

This image shows our core, installed on a ICE40HX1K (that equips the IceStick) and on an ECP5 85K (that equips the ULX3S). Plenty of room on the ULX3S !