WTFpga?

Developed by Joe FitzPatrick
FOSSified by Piotr Esden-Tempski and Clifford Wolf
Presented by Josh Johnson

August 13, 2019

1 Introduction

Welcome to the workshop! This is a hands-on crash-course in Verilog and FPGAs in general. It is self-guided and self-paced. Josh is here to answer questions, not drone on with text-laden slides like usual.

While microcontrollers run code, FPGAs let you define wires that connect things together, as well as logic that continuously combines and manipulates the values carried by those wires. Verilog is a hardware description language that lets you define how the FPGA should work.

Because of this, FPGAs are well suited to timing-precise or massively-parallel tasks. If you need to repeatedly process a consistent amount of data with minimal delay, an FPGA would be a good choice. Signal and graphics processing problems, often done with GPUs if power and cost are no object, are often easy to parallelize and FPGAs allow you to widen your pipeline until you run out of resources. As your processing becomes more complicated, or your data becomes more variable, microcontrollers can become a better solution.

The objective of this workshop is to do something cool with FPGAs in only two hours. In order to introduce such a huge topic in such a short time, LOTS of details will be glossed over. Two hours from now you're likely to have more questions about FPGAs than when you started - but at least you'll know the important questions to ask if you choose to learn more.

2 What We Won't Learn

In order to introduce Verilog and FPGAs in such a short time, we're going to skip over several things that will be important when you build your own FPGA-based designs, but are not necessary to kickstart your tinkering:

- Synchronous Logic: We're dealing entirely with human (AKA slow) input and output today. Running at maximum performance requires synchronizing all of the logic using a common clock, and optimizing the logic to fit that enforced timing.
- IP Cores: FPGA vendors pre-build or automatically generate code to let you easily interface your FPGA to interfaces like RAM, network, or PCIe. We'll stick to LEDs and switches today.
- Simulation: Didn't work right the first time? Simulation lets you look at all the signals in your design without having to use hardware or potentially expensive observation equipment.
- Testbenches: For effective simulation, you need to write even more Verilog code to stimulate the inputs to your system.

3 Meet the Hardware

We will be using the not creatively named iCE40-feather board for this workshop. It contains an iCE40UP5K FPGA, USB programmer and USB to UART converter, LiPo battery charge control, and plenty of LEDs in an Adafruit Feather compatible form factor. To extend its capabilities for this workshop, we will be attaching a FeatherWing which has a dual seven segment display, and eight DIP switches.

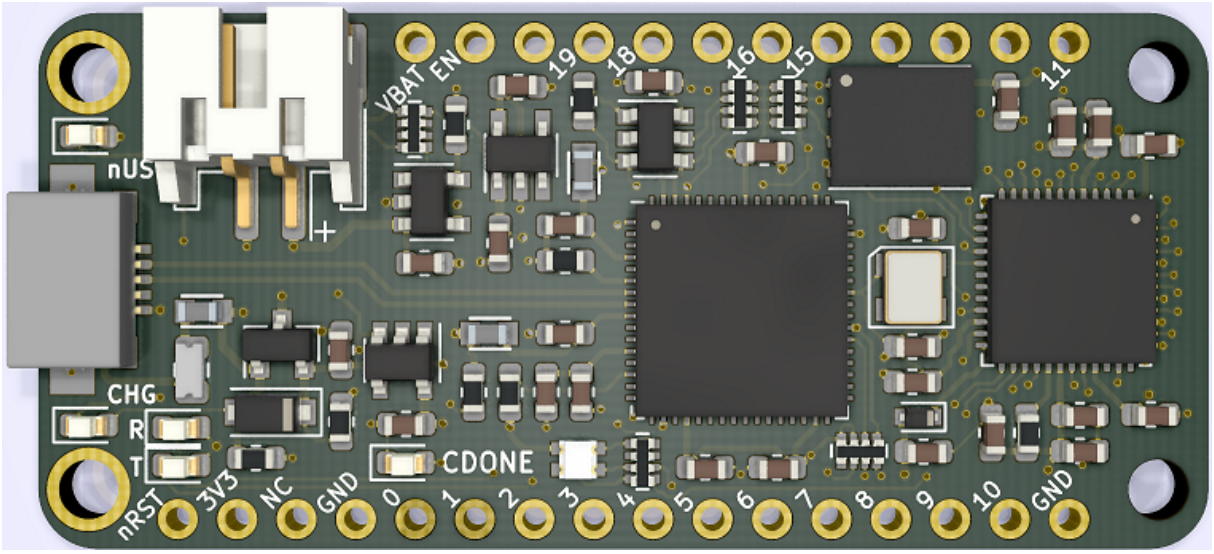

Figure 1: The iCE40-feather FPGA board being utilised.

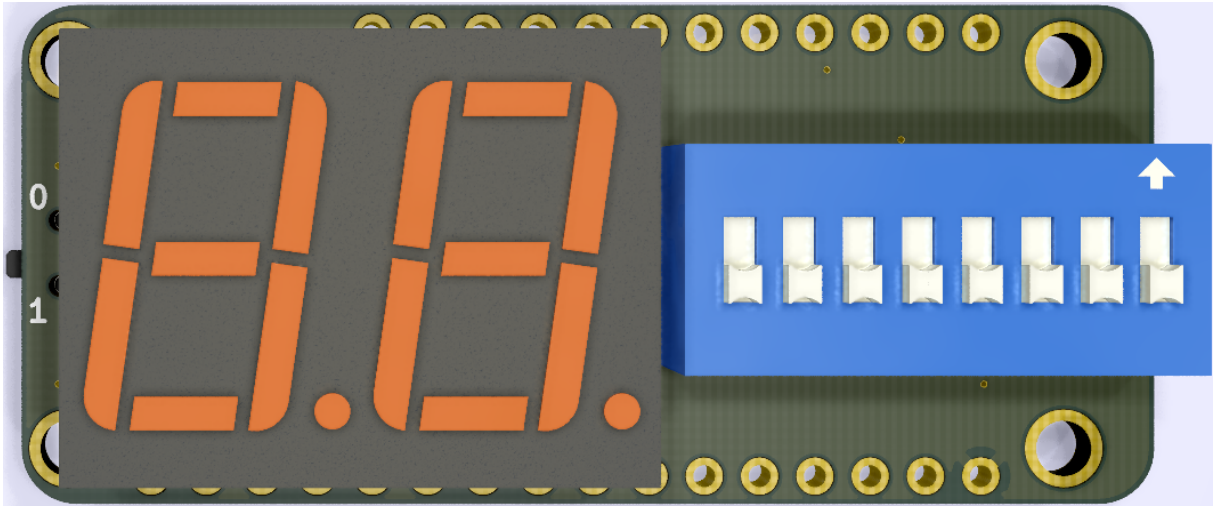

Figure 2: The seven segment and DIP switch FeatherWing.

4 Getting to Blinky

Due to time constraints, it is suggested that you install the toolchain and get a blinking LED on your iCE40-feather before attending the workshop. If you have not already installed the required Yosys, NextPNR, and IceStorm tools, follow the instructions in `install.md`. If you do not have the FPGA dev board, read the instructions in `README.md` and get in contact with Josh.

With the toolchain installed and hardware in hand, it is now time to blink some LEDs! If you have not already done so, clone the repository and open the blink directory.

git clone https://github.com/joshajohnson/WTFpga
cd WTFpga/blink

Now its time to walk through the process of synthesizing an FPGA design of your own and uploading it to the FPGA. We will be using the amazing open source tools called IceStorm, NextPNR and Yosys.

- Ensure that the FPGA is connected via a USB cable to your computer.
- Open the command line in the `WTFpga/blink` folder.
- Build and upload the gateware by typing **make prog** into the terminal.
- Ensure that the led marked nUSR is blinking. If not it's time to begin troubleshooting!
- Once you get the LED blinking, open the file `blink.v` and have a look around. See if you can change the frequency of the blinking LED by altering the code.
- After changing `blink.v` and saving it, run **make prog** again and confirm the frequency changes.

If you have made it this far, congratulations! You have successfully programmed your FPGA, and can now attend the WTFpga workshop knowing that you'll hit the ground running. If you are having issues programming the board, get in contact with Josh and he will lend a hand troubleshooting.