SecuringHardware.com

WTFPGA?

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

The objective 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.

Joe FitzPatrick
@securelyfitz
joefitz@securinghardware.com
http://www.securinghardware.com/WTFpga

Meet the hardware

This is a basic FPGA demonstration/learning board.

<board image, annotated>

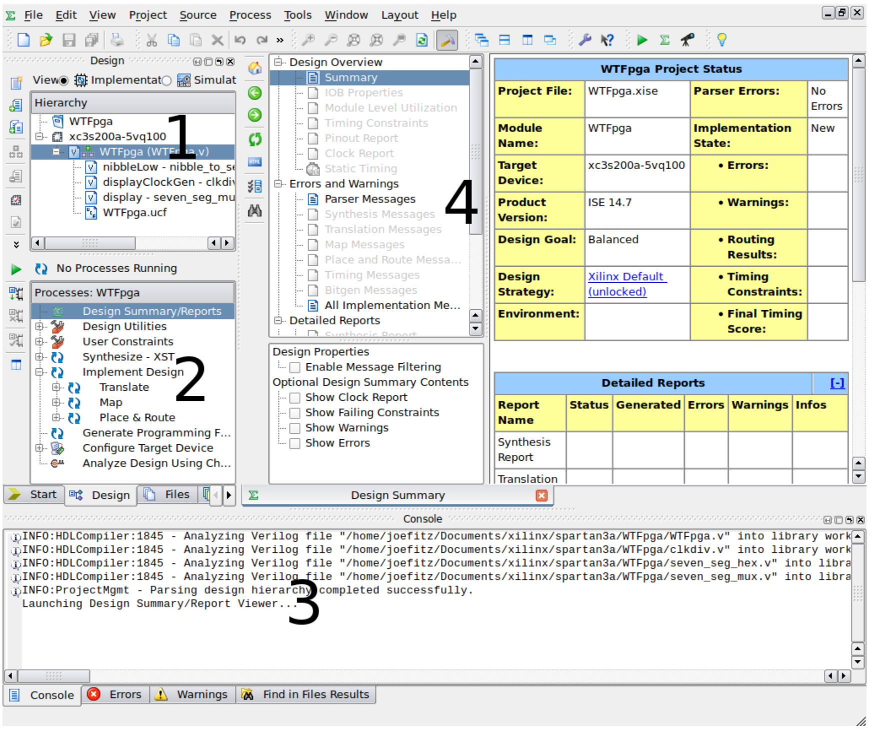
1. Xilinx XC3S200A FPGA
2. 8 DIP switches
3. 4 Pushbutton Keys
4. 8 LEDs
5. 8 7-segment Displays
6. Power Connector
7. JTAG programming header

1. Take a look at the board, and identify each of the components listed above. There are few other components not listed that we won’t use in this workshop.
2. Power on your laptop and connect the power cable from a USB port to the power connector on the device. The FPGA will automatically load a demo configuration from the onboard serial flash. Play around:
   - 2.1. What do the DIP switches do?
   - 2.2. What do keys 0-3 do?
   - 2.3. What does the reset button do?

Configuring an FPGA

Now that we are familiar with the hardware as-is, let’s walk through the process of using the Xilinx tools to make sure we can configure the board.

1. Connect the JTAG interface
    1.1. Connect the keyed 10-pin header to the black FPGA board
    1.2. Connect the keyed 20-pin header to the red JTAG board
    1.3. Connect the JTAG board to the laptop with the USB Mini cable
    1.4. Be careful to keep the boards apart and away from metal objects that might short them.
2. Power on your laptop and start the Xilinx Tools. Since these are repurposed chromebooks, there are a few extra steps:
    2.1. Power on the laptop
    2.2. Click ‘browse as guest’ to bypass login
    2.3. Press <ctrl-alt-t> to open a crosh shell.
    2.4. Type ‘shell’. This will switch you over to a full linux desktop.
    2.5. Double click on the icon labeled ISE on the desktop to open up the xilinx tools with the default project.



WTFpga Project Status

Project File:	WTFpga.xise	**Parser Errors:**	No Errors
Module Name:	WTFpga	**Implementation State:**	New
Target Device:	xc3s200a-5vq100	**• Errors:**	
Product Version:	ISE 14.7	**• Warnings:**	
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:		**• Final Timing Score:**	

Detailed Reports [-]

Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report					
Translation					

```
INFO:HDLCompiler:1845 - Analyzing Verilog file "/home/joefitz/Documents/xilinx/spartan3a/WTFpga/WTFpga.v" into library work
INFO:HDLCompiler:1845 - Analyzing Verilog file "/home/joefitz/Documents/xilinx/spartan3a/WTFpga/clkdiv.v" into library work
INFO:HDLCompiler:1845 - Analyzing Verilog file "/home/joefitz/Documents/xilinx/spartan3a/WTFpga/seven_seg_hex.v" into libra
INFO:HDLCompiler:1845 - Analyzing Verilog file "/home/joefitz/Documents/xilinx/spartan3a/WTFpga/seven_seg_mux.v" into libra
INFO:ProjectMgmt - Parsing design hierarchy completed successfully.
Launching Design Summary/Report Viewer...
```

1. Project Hierarchy
2. Processes View
3. Console
4. Design Summary/ Code Editor

3. All we will do for now is load and run this design to our FPGA board. In the Processes View on the bottom left, there is a ‘Configure Target Device’ item. Double Click on it.
    3.1. First, the software will Synthesize the design - turn the Verilog code into basic logical operations, optimizing it in the process
    3.2. Next, the tools will Implement the design. This takes the optimized list of registers and assignments, and maps them to the logical blocks available on the specific FPGA device we have configured

    3.3. When that completes, the fully laid out implementation needs to be packaged into a format for programming to the device. There are a number of options, but we will use a .bit bitstream file for programming over JTAG
    3.4. Finally, the .bit file needs to be sent to the device over the jtag adapter. When this happens, the demo configuration will be cleared and the new design will take it’s place

4. What does this new configuration do?

Assignments

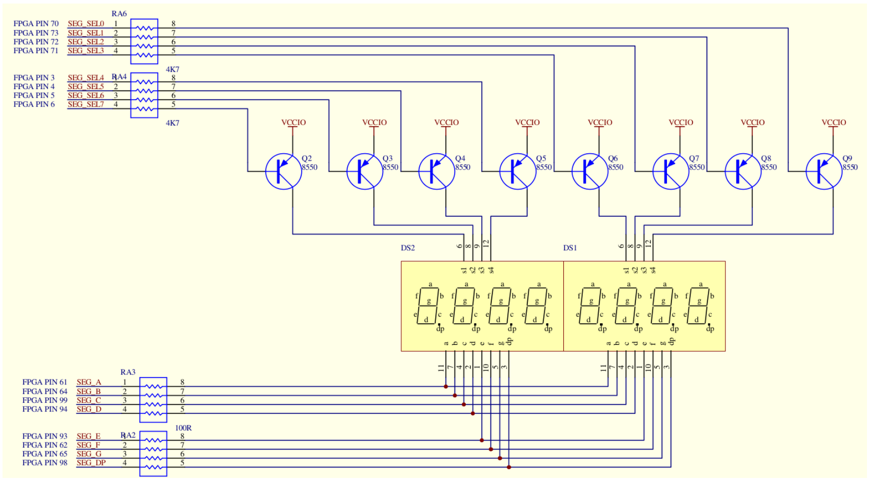
Now that we know how to use the tools to configure our device, let’s start writing some simple verilog code. Programming languages give you different ways of storing and passing data between blocks of code. Hardware Description Languages (HDLs) allow you to write code that defines how things are connected

1. Double click on ‘WTFpga.v’ in the hierarchy view in the upper left. It will open the file for editing
2. First, we are actually going to add a wire that we can use to connect key 0 to LED 0:
   - 2.1. Find the comment on where to add wires.
   - 2.2. Add a new wire called wire0: “wire wire0;”
3. Next, we need to connect the inputs and outputs to the wire:
   - 3.1. Find the comment showing you where to enter assignments.
   - 3.2. There is already an assignent for LED. replace it with a new oone to output the value of wire0: “assign led =wire0;”
   - 3.3. Assign wire0 to the value of the button, key[0]: “assign wire0=key[0];”
   - 3.4. Note that we’re not writing sequential commands. All of these things happen at once. It doesn’t actually matter what order the assign statements occur.
4. Now, let’s configure the board:
   - 4.1. Double click on ‘Configure Target Device’ in the view on the bottom left.
   - 4.2. The software should synthesize, implement, generate, and configure in order.
   - 4.3. If you have issues, look in the console for possible build errors. If you have trouble, ask for help!
   - 4.4. Once you have programmed the device, try pressing key0. What happens?

5. Conveniently, there are 8
6. switches and 8 LEDs on this board. Let’s repeat the process, and this time connect 8 wires at once. Also, we don’t need to explicitly create a wire to connect things, we can do it direclty:
    6.1. replace the previous assignment for led with “assign led[7:0]=sw[7:0];”
    6.2. Double click on “Configure Target Device”
    6.3. Test your system. Did it do what you expected?

Combinational Logic

Simple assignments demonstrate the parallel nature of FPGAs, but combinational logic makes it much more useful. We're going to write a small module (like a procedure) that will convert the binary value shown on the LEDs into a hex digit on the 7-segment display.



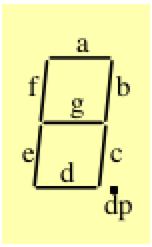
First we should understand the display. There are 8 Seg[] output wires that control which lines are on or off, and there are 8 Seg_sel output wires that choose which digit(s) are on at any given time. In order to display 8 different characters, we need to cycle through them, one at a time, fast enough so that they persist.

All of the code to generate the clock (clkdiv.v) and multiplex the display (seven_seg_mux.v) is already written and included in the project, giving direct access to each of the displays.

1. First, let's connect the stubbed out nibble_to_seven_seg module into our design:
    1.1. Find nibble_to_seven_seg in WTFpga.v. The module is instantiated, but not connected to anything.
    1.2. Connect the low 4 bits of our switches to the nibblein field: " .nibblein(sw[3:0]),"

1.3. Connect the 8 bits of output to the seven segment display: “.segout(seg6)”

2. Next, let’s duplicate it so we can see a full hex byte:
   2.1. Cut-and-paste the entire nibble_to_seven_seg block a second time
   2.2. Rename the instantiation from “lowNibble” to “highNibble”
   2.3. Connect the high 4 bits of the switch with “sw[7:4]”
   2.4. Connect the second display with “seg7”
3. Now, let’s implement nibble_to_seven_seg:
   3.1. Double click on it in the project browser
   3.2. We’re going to use a case statement. They actually work just the same as they do in most programming languages. See the short example in the comments of the code.
   3.3. For each entry of the case statement, we need the expected value of nibblein, and the corresponding value of segout. segout is an array of bits that represent the segments of the display as shown below. When a bit is ‘0’, it is ON and Illumitated, when it’s ‘1’ it is OFF.
   3.4. For example, hex ‘1’ looks like 8’b11111001. We can express this as: “4’h1=8’b11111001;”
   3.5. Figure out what you need to set for each of the hex values using the diagram below
4. When you’re done, try to “Configure Target Device”. Does it work as you expected?



Registers

While there is so much more to combinational logic than we actually touched on, lets move on to a new concept - Registers. Assignments are excellent at connecting blocks together, but they're more similar to passed parameters to a function than actual variables. Registers allow you to capture and store data for repeated or later use.

We will use a register to store the value we set on the DIP switches for later use

1. First, let's add a register to store our value.
    1.1. Find the area for wire and reg definitions
    1.2. Add a register for storedValue: "reg [7:0] storedValue;"
2. Open WTFpga.v and find the "always @" block
    2.1. "always @" is the header for a synchronous block of code. Like software, commands get executed in order inside the always block
    2.2. The list in parenthesis following @ is the Sensitivity List. Whenever one of the signals in the sensitivity list changes, the block is run. It is similar to binding a callback function or mapping an interrupt in software terms
    2.3. Let's add "posedge key[3]" to the sensitivity list. This means each positive edge - every time key[3] goes from low to high - we will execute this block.
    2.4. Now, assign the value of the switches to storedValue inside the always block: "storedValue<=sw;"
    2.5. Note that assignments are different inside the always block - we are setting a register, not assigning a wire anymore!
3. We need to display this value. Duplicate both nibble_to_seven_seg blocks:

3.1. rename them to nibbleHighStored and NibbleLowStored

3.2. connect the nibblein to storedValue instead of sw

3.3. output them to seg[3] and seg[4]

4. Configure your device, and test it out:

   4.1. Two display characters should change any time you switch the DIPs.

   4.2. Press Key 3 to store the current value of the switches, and show it in the middle of the display

   4.3. Does it work how you expected?

Calculating

Lets combine everything we've done so far with a little bit of glue and lipstick to make it into a basic 8-bit calculator. To do this we need to perform some math and display the result.

1. First let's get addition working:
   1.1. Add a wire for the result: "wire [7:0] result;"
   1.2. Assign the result a value "assign result = sw + storedValue;"
2. Display the result on the far right side of the display:
   2.1. Duplicate two more nibble_to_seven_seg blocks
   2.2. connect nibblein to result
   2.3. connect segout to seg0 and seg1
3. Add a + and = sign to the display. We can do this by directly assigning the bits we want on and off to the corresponding segN values:
   3.1. "assign seg5=8'b10111001;"
   3.2. "assign seg2=8'b10110111;"
4. Synthesize your design and configure your device. Did it work?
5. Let's add subtraction. We only need to change two lines to have the device subtract when key[0] is pressed:
   5.1. Use the ternary operator to add or subtract based on the value of key[0]: "assign result = key[0]?sw-storedValue:sw+storedValue;"
   5.2. Use it again to select the operator displayed: "assign seg5=key[0]?8'b10111111:8'b10111001;"
   5.3. Synthesize and configure. Did it work?