Lab 6: Intro to FPGAs

UC Davis Physics 116B
Rev 2/25/2019

There's a saying when dealing with complex electronic systems: "If you can make the LED blink, you're 90% of the way there.", so in this lab you will make the LEDs blink on the Diligent Xilinx prototype board. Doing so involves four distinct steps:

- Writing VHDL code to specify the desired logical behavior.
- Constraining the mapping between the internal logic signals and the pins that connect to the board.
- Compiling your code and generating the configuration (.bit) file.
- Downloading the configuration to the one of the Xilinx chips on the board.

Once you figure out how do do this, it will be very straightforward to generalize your knowledge to much more complex applications.

The Digilent Xilinx prototype board

Digilent is one of several vendors that produce "prototype" boards incorporating a Xilinx chip or chips, as well as a variety of interfaces. In principle, these are used to develop code to be used in other applications of the chips, but the boards are versatile enough that they can be used directly even in fairly complex applications.

This particular board contains three Xilinx chips

- Xilinx XC3S500E Spartan-3E FPGA. This has 232 user pins and over 10,000 logic cells. It will be the primary target for our configurations.
- Xilinx 4 Mbit Platform Flash configuration PROM. Configurations may optionally be loaded to this, so they will automatically be uploaded to the XC3S500E when the board powers up.
- Xilinx 64-macrocell XC2C64A CoolRunner CPLD. Used to coordinate different boot modes for the XC3S500E, but can also be used for user applications.

The board also contains many other features and interfaces, including onboard memory, various standard interfaces (RS-232, ethernet, etc), and LCD display panel, and a VGA port. Of particular interest for our labs will be

- Input buttons and switches

- Display LEDs
- A four-input DAC
- A four-output ADC

Implementing an example program

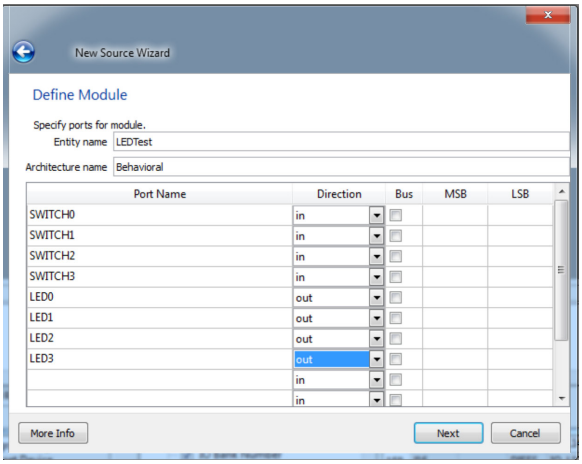
If you don't see a shortcut for "Project Navigator", then open "Start->All Programs->Xilinx Design Tools->ISE Design Suite 14.4 ->64-bit Project Navigator" to open the Xilinx IDE.

Start a new project by doing "File->New Project...". Give the project an unused name and click "Next". On the next screen (project settings), go to the "Evaluation Development Board" pulldown and select "Spartan-3E Starter Board. Verify that this fills in

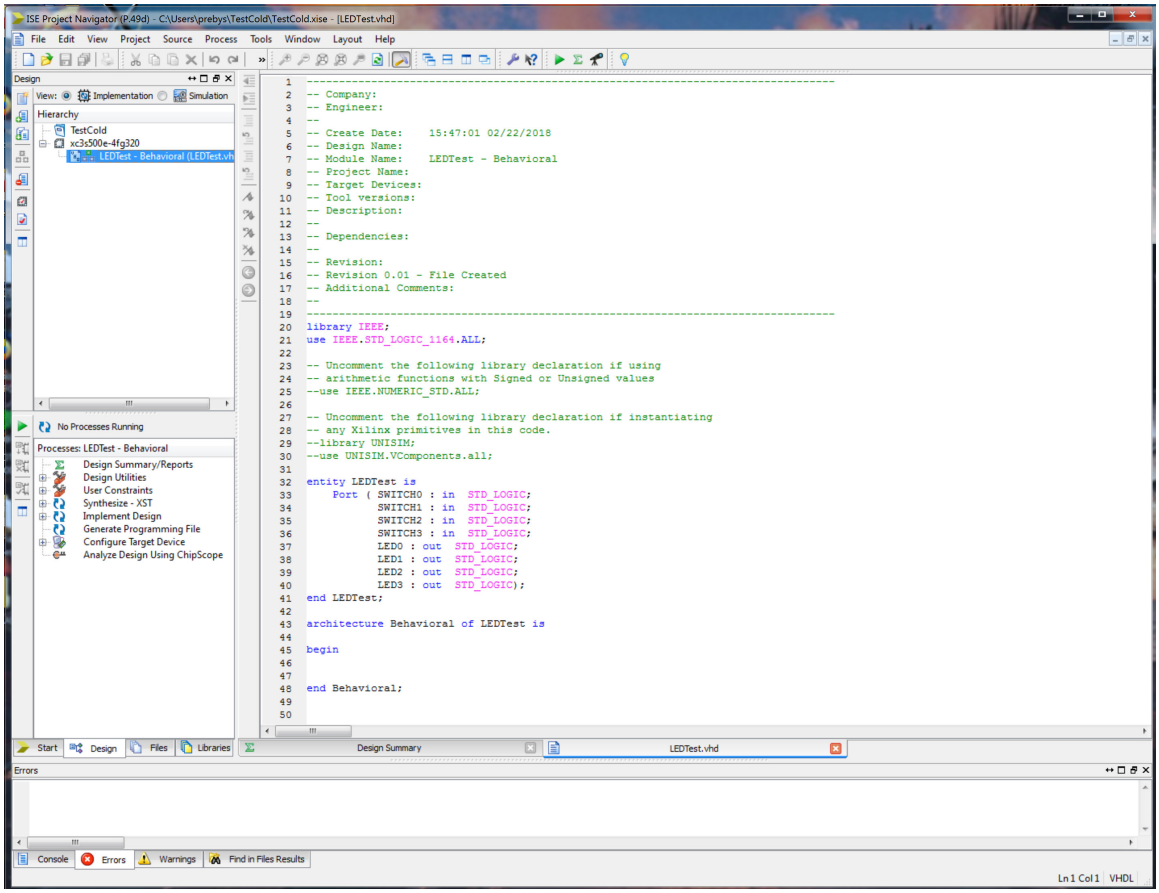
- Family: Spartan 3E
- Device: XC3S500E
- Package: FG320

Click "Next" and then "Finish".

Under Hierarchy, right click on Xc3s500-4fg320 and select "New Source->VHDL Module" Give it a name. We'll assume "LEDTest" This should pull up a "Define Module" window. Specify 4 inputs and 4 outputs, as shown



Click "Next" and "Finish". This will create the framework of a routine, and your window should look like this



Your implementation code goes between “begin” and “end” under “architecture”. Tie each switch to the corresponding LED by typing

```
LED0 <= SWITCH0;
```

etc

Now you need to constrain the signals to the appropriate pins. You can find all the pin definitions in the reference manual for the board, which should be on the computer.

In the Hierarchy window, right click on LEDTest - Behavioral (NOT on the name of the Xilinx chip!), select “New Source...”->“Implementation Constraints File”, give it the name “pins”, click “Next” then“Finish”. If you expand LEDtest, you should see a file “pins.ucf” beneath it. Double click on it to edit it, and enter the pin constraints as shown.

```
1  NET "SWITCH0" LOC = "L13";
2  NET "SWITCH1" LOC = "L14";
3  NET "SWITCH2" LOC = "H18";
4  NET "SWITCH3" LOC = "N17";
5
6  NET "LED0" LOC = "F12";
7  NET "LED1" LOC = "E12";
8  NET "LED2" LOC = "E11";
9  NET "LED3" LOC = "F11";
```

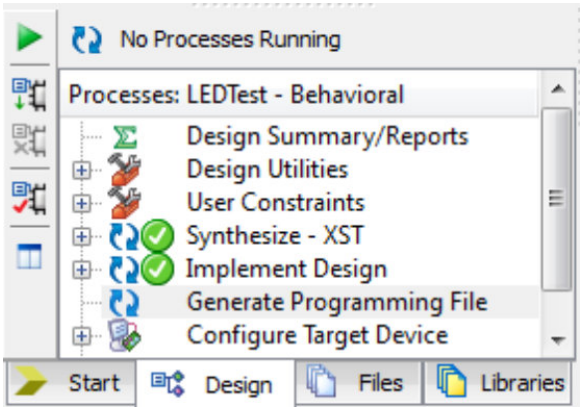
Click the “Save” icon.

There seems to be a bug (or a feature) at this point that keeps the program from recognizing this as a constraints file. To force it, do the following:

- On the “pins.ucf” file, right-click->“Remove”, then “Yes”.
- Right-click on “LEDtest - Behavioral”, select “Add source...” and reselect “pins.ucf”.

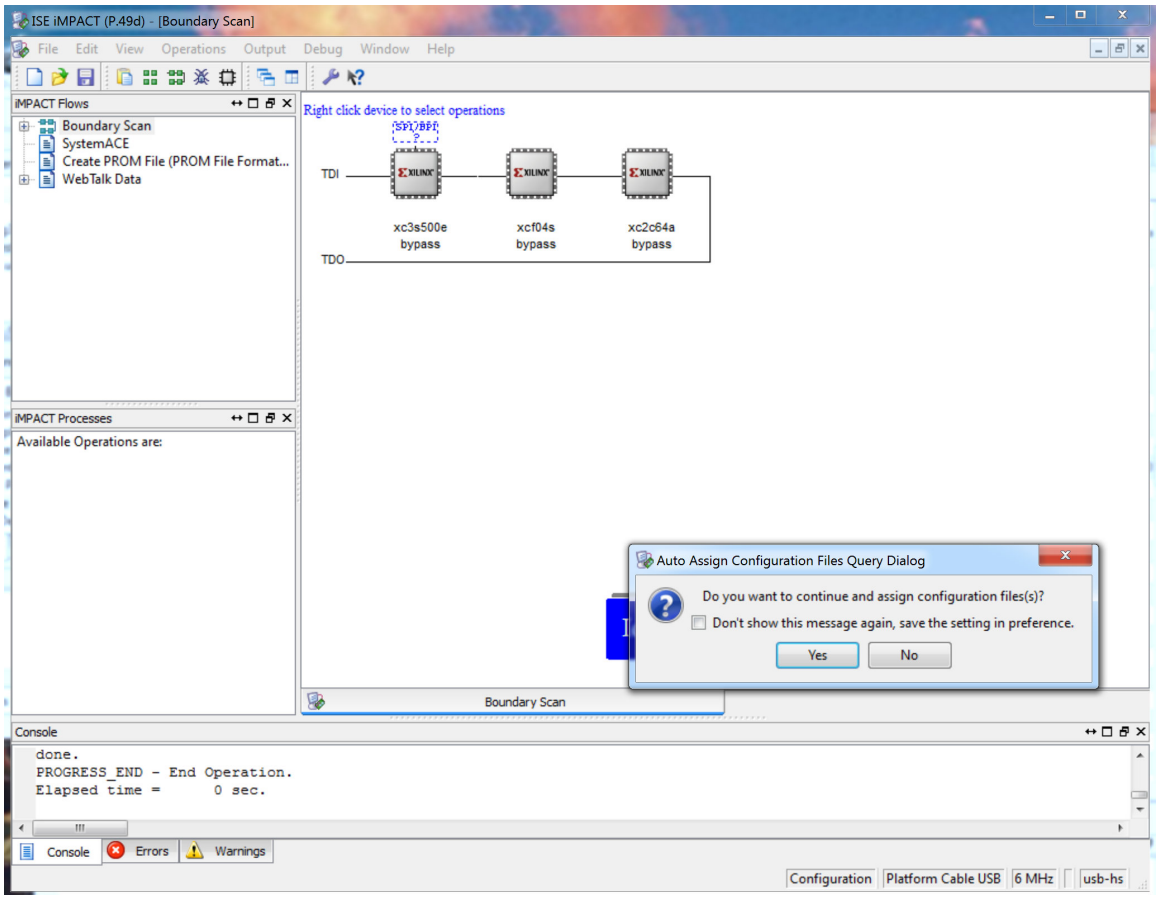
Now it should work.

Now click on “LEDtest” again, and right-click->“Implement top module”. This will cause “Synthesize-XST” and “Implement Design” to execute in the Design window. If it runs successfully, both should have a green check.

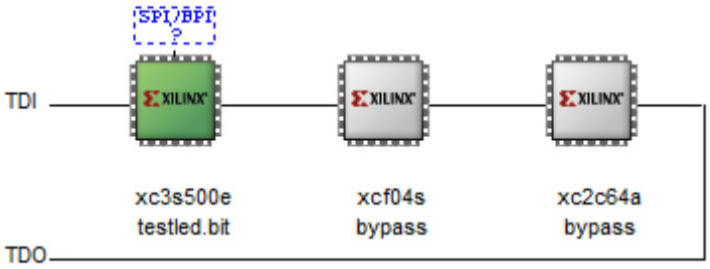


(Very important!) Select the “Design Summary (implemented) tab at the right, click “Pinout Report” and verify that it has mapped the signals correctly. If it hasn’t, try removing and adding the constraints file again.

Now double-click on “Configure Target Device”. Click “OK” at the splash to bring up iMPACT. Double-click on “Boundary Scan” at the left. If the screen that appears at the right is blank, mouse over it and right-click->“Initialize chain”. An image should appear of three Xilinx chips, daisy-chained together.



A splash will come up asking you whether you want to continue and assign configuration files. Click "Yes", and then click "No" on the next splash about loading the configuration to a PROM. At this point a browser will appear. Use this to select your "ledtest.bit" file (it will probably come up in a different directory, so be careful you don't accidentally select an old one from someone else!). It will load this into the first Xilinx chip and move on to the next two. Click "Bypass" for both of those. When you're finished, the first chip should be green and have the name of the bit file under it, while the other two should say "bypass".



Now (finally), right-click->"Program" on the first chip.

Verify that the four switches control the four lights, as you intended.

If you leave the iMPACT window open, you will able to assign different files and reprogram the chip without going through all the initialization steps again.

Further Exercises

Now that you're experts, do the following exercises. Your lab report should include your code. Photos of the screen are acceptable, provided they're legible.

Combinatorial Logic

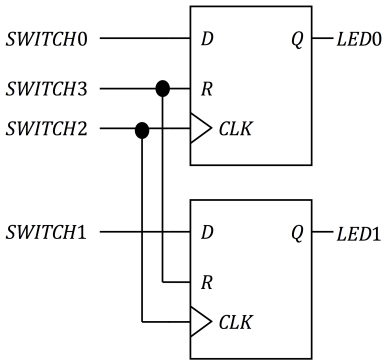
Modify your design as follows

- LED0 represents the logical AND of SWITCH0 and SWITCH1
- LED1 represents the logical OR of all four switches
- LED2 and LED3 represent, respectively, the SUM and CARRY bits from summing SWITCH2 and SWITCH3

Verify that your design works correctly.

Synchronous Logic

Write code to implement the following circuit



When SWITCH2 goes HI, the state of SWITCH0 and SWITCH1 will be loaded to LED0 and LED1. SWITCH3 will *asynchronously* turn those LEDs off as soon as it goes HI. Recall that in class, we gave an example of a simple data latch as shown below:

```
LATCH: process(CLK)
   begin
   if rising_edge(CLK) then
       Q <= D
       QB <= not D
   else
       --default: holds previous value
   end if;
end process;
```

Modify this to handle two inputs/outputs and so that the RESET switch takes priority over the CLK input.

If you leave LED2 and LED3 in your port output list without using them, so go ahead and connect them to SWITCH0 and SWITCH1. That way you can see the state of the inputs to the latches.

If you write this code correctly, the compiler will recognize that you are creating synchronous logic, and tie the click signal to one of the internal high speed clock buffer lines of the Xilinx chip. Only certain pins can be tied directly to these lines and H18 (Switch 2) is not one of them, so it will have to be internally "jumpered" to the clock line. This would preclude clocking your circuit at the highest rate, so by default it will produce a fatal error. Since you're clocking the circuit very slowly, you don't need to worry about this, and you can reduce this error to a warning by adding the line

```
NET "SWITCH2" CLOCK_DEDICATED_ROUTE = FALSE;
```

to your "pins.ucf" file.

Compile and load your code, and verify that you can change the states of SWITCH0 and SWITCH1 without affecting LED0 and LED1 until SWITCH2 transitions form LO to HI, and that SWITCH3 will turn LED0 and LED1 off.