Revision History
January, 2008 : Original document creation (M. Hamlett)

1. Introduction
Field Programmable Gate Arrays (FPGAs) provide numerous advantages to designers. They give you the option of quick time-to-market, lower initial costs, and the option to change your design over time. The purpose of this tutorial is to give you a brief overview of the design flow process when your end-target is an FPGA. While FPGA designs do not require the use of Hardware Description Languages, most designers do use HDL’s extensively. The end product in an FPGA device is called the bitstream which is then loaded onto the FPGA. This tutorial will take you through how to create that bitstream from an existing Verilog design.

2. Learning Objectives
- To get a brief overview of FPGA tool flow by taking a provided example through the flow
- The gain a bigger picture of the possibilities and advantages of using FPGAs.

3. Tools
This tutorial focuses on the Xilinx tools for FPGA design. In particular, we will be using a product called Xilinx ISE 9.2. You will access this software using the Virtual Computing Lab. The Virtual Computing Lab (VCL) maintains a large number of machines that students can use remotely. We will be using Remote Desktop to remotely access these machines, so for the purposes of this tutorial, we assume that you are somewhere on campus and are using a Windows machine.

Figure 1: VCL Reservation Creation
4. Getting Started: Virtual Computing Lab & Remote Desktop

Accessing Virtual Computing Lab
1. Visit the following site: http://vcl.ncsu.edu
2. Click on “Make a VCL Reservation” in the upper left of the site
3. Log in with your Unity ID/password
4. Select the “Xilinx 9.2 (w/ModelSim+Matlab)” from the dropdown box, also select
   Now for when to use the application. This tutorial should take no more than 2 hours
to complete, so select the appropriate Duration.
5. Your screen should resemble Figure 1.
6. Click on “Create Reservation”
7. After some delay, a “Connect!” button will appear, click on this.
8. Record the Remote Computer Address, Login and password provided to you.

Using Remote Desktop
1. The Remote Desktop application can be found in one of two locations:
   • In the Start Menu, Under Accessories
   • If using a campus machine, it is also available in the Novell Application
     Launcher.
2. Once you have located and started Remote Desktop, a dialog box will ask you which
   computer to connect to. Input the IP Address of the machine provided by the VCL.
   See Figure 2
3. Click on Connect, and input the Login and password you recorded when prompted.
   You should end up at a new Windows desktop.
4. Click on the icon “Map to Unity K Drive Space”. Log in with your standard Unity ID
   and password.
5. Create a new folder for this Tutorial somewhere in your K: drive. Download all
   support files for this tutorial into this folder. This includes all the Verilog files
   provided with the tutorial.
6. Double click on the “Xilinx ISE 9.2i” icon. This will start the Xilinx FPGA design
   environment.

IMPORTANT: Make sure you create your project, and save all your work on your K:
   drive. ALL THINGS YOU STORE ON THE DESKTOP, OR “MY DOCUMENTS”
   FOLDER WILL BE LOST WHEN YOU LOGOUT!
5. Overview of the design flow process

By now, you are familiar with part of the design flow process as it relates to Synopsys. You’ve taken a design concept, implemented it in Verilog and synthesized it with the Synopsys Design Compiler. Designing for FPGAs is not significantly different than what you’ve already done. As long as your Verilog RTL files contain synthesizable logic, Xilinx should have no problem using it. The major steps involved are:

1. Design Input
2. Synthesis
3. Design Constraining
4. Implementation
5. Bitstream Creation

Design Input

The design input stage is where you take a design you’ve planned out, and start inputting it into the Xilinx tools. A design can be input using a variety of methods. You can code it in either Verilog or VHDL, or alternatively you can use a “Schematic” view to input your design graphically. For purposes of this class we input our design as Verilog RTL. Once your design is complete, you can check the syntax of your RTL during this stage and fix any errors that may result.

Synthesis

The synthesis phase of the design is where the Xilinx tools take your Verilog RTL, and translate it into physical hardware. The tools use a variety of logic minimization techniques to produce a rough mapping of your design on the FPGA. Xilinx FPGAs are based on Slices. Each slice contains a few Multiplexors, Look up Tables, and other logic. One of the major differences between design targeting a FPGA and an ASIC is that you have to take into account fixed resources. When you run Synthesis for an ASIC in Synopsys, the software determines a netlist which eventually determines which standard cells your design needs, how to connect them, and how many you need. Synthesis for an FPGA involves the tool determining how it can map your design to these fixed “Slice” structures in the most efficient manner possible. At the end of the Synthesis process, the tools have a rough idea of the resource utilization and maximum clock frequency of your design.

Design Constraining

At this point in the design flow, the tools know how they are going to map the logic you need into slices. The tools do not however know how to constrain your design. Another resource FPGAs provide other than slices is pins. Pins allow you to get inputs/stimulus in from the outside world and allow you to send out information. Each port in your top
level design should correspond to one of the FPGAs' pins. If you do not explicitly specify which pin a port should correspond to, the tools will randomly assign ports to pins. This is usually not desired. There are a variety of types of pins available on FPGAs. Some pins should only be used for Clock or reset signals. Other pins are for different input/output standards such as TTL or LVDS. The details of pin constraints are beyond the scope of this tutorial, we will however show you how to make simple pin constraints to target a specific development board. There are other types of constraints you can make during this phase, such as Timing and Area constraints. These are beyond the scope of this tutorial.

Implementation

After Synthesis and Design Constraining, Implementation determines which specific slices will implement your given logic. Implementation also works out the routing of signals from your input/output pins to the logic in your slices. Some refer to this stage as “Place And Route” because you Place your logic in specific parts of your FPGA here, and route your signals.

Bitstream Creation

The final step in the design process is to take your Implemented design and translate it into a format that can be used by your FPGA. This format is called the Bitstream, and it is a proprietary binary file that gets loaded onto the FPGA. Xilinx FPGAs are SRAM-based technology, meaning that once you load a bitstream on the FPGA, it only remains in the FPGA while power is applied. Each time power is applied to the FPGA, the design must be reloaded. Most designers handle this by placing a small PROM (Programmable ROM) before the FPGA that on power up, will automatically program the FPGA in just a few milliseconds. Another part of the Bitstream creation process allows you to create a special bitstream for these “PROMS”. 
6. Creating a new Project

Here we will create a new, blank project to match the design specifications given in the appendix. We assume you have successfully logged into the VCL and have Xilinx ISE 9.2 open.

1. Click on File, New Project.
2. Name the project whatever you like, Here we name this project trafficControl, make sure you set the Project Location to the folder you already created for this tutorial on your K: drive. Top-Level Source Type should be set to HDL.

![Image of New Project Wizard]

3. The next screen asks you to specify the target FPGA for your design. We will target a Virtex2P, XC2VP30. Make sure you specify the package type as FF896, Speed Gate -7. Your dialog should look similar to the one below when done.

![Image of New Project Wizard with Virtex2P settings]

4. Click Next to the Create a New Source Screen
5. On the “Add Existing Sources” Dialog, click on the “Add Source” button and select all the Verilog files provided with this tutorial. This should include: `top.v`, `controller.v`, `datapath.v`, `lightController.v`
6. Click Finish, your Window should then look like Figure 3.

![Figure 3]
Take a moment, and look around the Interface. Most of the screen is occupied by the project Status report. Since we haven’t done anything in the project yet, the report is empty. On the left of the screen are two panels. One is called the Sources panel, the other is called the Processes panel.

Notice in the sources panel that you see the design in a hierarchical structure. Meaning, underneath the top.v (top module) is the datapath and controller. Inside the datapath are the flasher and lightController modules. If you wanted to create a new source, you would do so through the Project -> New Source menu option. Once created, the source would appear in the Sources pane at the proper level in the hierarchy. Also notice how under the datapath, the flasher module appears with a Question Mark. This is because we haven’t defined a flasher module for this project.

The processes tab displays the various options available to you for your selected source. For instance, if you click on top.v you see a variety of options including Synthesize, Implement and Generate Programming File. For sources that are not set to the top module, you see fewer options, but can still do things like “Check Syntax”
7. Adding a New Source

We should now fix this missing flasher module that is currently showing up as a question mark in the Xilinx tools. Follow these steps to learn how to add new sources to your project.

1. Click on the Project Menu, then click on New Source.

2. In the New Source Wizard, name your File *flasher*
   On the left side of the screen, choose “Verilog Module”

3. The next screen in the wizard will allow you to specify the ports for the module.
   Go ahead and make 5 input ports (Clock, reset, red, yellow, green)
   Make 1 output port, flashOut
   Your screen should look as follows:

   ![New Source Wizard](image)

4. Click Next and finish

5. You will now see a shell of the flash module on the screen with just the input/output ports defined. Copy and paste the remainder of the flash module into this file….This can be found in the appendix.

6. Save the file, you should notice the Question mark go away for both flashNS and flashEW. This indicates that the software now knows the code for these modules.
8. Synthesis

We will now take our completed design, and run it through Synthesis.

1. Click on the top module (top.v).
2. Under the processes list on the left you will see an option for Synthesize. Double click on this to start the Synthesis process.
3. Wait for synthesis to complete….you will know it is finished when you see the “Process “Synthesize” completed successfully” message.
4. Look through the various status messages that came out through the window
   
   Question: What is the minimum clock period reported by Synthesis?

5. Notice that beside synthesis now is a Triangle with an exclamation point inside. This is the tools way of saying Synthesis has completed with warnings. We will ignore the warnings given for this exercise.

9. Design Constraints

Our design contains the following input ports:
- Clock
- Reset
- Sensor0
- Sensor1
- Sensor2
- Sensor3

It also contains the following output ports:
- Light0
- Light1
- Light2
- Light3

While the purpose of this tutorial is not to load our final design onto hardware, we do want to illustrate the steps that would be required. For now, it is assumed that we are
targeting a Xilinx XUP development board, which has a variety of peripherals available to it. Among those available are on-board clocks, push-buttons and LEDs. The table below shows the relevant connection specs that were taken from the board’s user’s guide.

### Xilinx XUP Connections

<table>
<thead>
<tr>
<th>Signal</th>
<th>Direction</th>
<th>FPGA Pin</th>
<th>I/O Type</th>
<th>Drive</th>
<th>Slew</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_CLOCK</td>
<td>Input</td>
<td>AJ15</td>
<td>LVCMOS25</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>LED_0</td>
<td>Output</td>
<td>AC4</td>
<td>LVTTTL</td>
<td>12mA</td>
<td>SLOW</td>
</tr>
<tr>
<td>LED_1</td>
<td>Output</td>
<td>AC3</td>
<td>LVTTTL</td>
<td>12mA</td>
<td>SLOW</td>
</tr>
<tr>
<td>LED_2</td>
<td>Output</td>
<td>AA6</td>
<td>LVTTTL</td>
<td>12mA</td>
<td>SLOW</td>
</tr>
<tr>
<td>LED_3</td>
<td>Output</td>
<td>AA5</td>
<td>LVTTTL</td>
<td>12mA</td>
<td>SLOW</td>
</tr>
<tr>
<td>PB_ENTER</td>
<td>Input</td>
<td>AG5</td>
<td>LVTTTL</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>PB_UP</td>
<td>Input</td>
<td>AH4</td>
<td>LVTTTL</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>PB_DOWN</td>
<td>Input</td>
<td>AG3</td>
<td>LVTTTL</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>PB_LEFT</td>
<td>Input</td>
<td>AH1</td>
<td>LVTTTL</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>PB_RIGHT</td>
<td>Input</td>
<td>AH2</td>
<td>LVTTTL</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

We will assign the top level ports of our code to the Xilinx Pins as follows:

- Clock → SYSTEM_CLOCK
- Reset → PB_ENTER
- Sensor0 → PB_UP
- Sensor1 → PB_RIGHT
- Sensor2 → PB_DOWN
- Sensor3 → PB_LEFT
- Light0 → LED_0
- Light1 → LED_1
- Light2 → LED_2
- Light3 → LED_3

1. Click on the top level module (top.v)
2. In the processes tab, expand the options under “User Constraints” by clicking on the triangle to the left
3. Double click on the option for “Assign Package Pins”
4. A dialog will ask you if you want to “automatically create a UCF and add it to the project at this time?”, click Yes.
5. Xilinx PACE will now start, on the right of the screen will be a photo of your FPGA in terms of I/O.
6. PACE is a very powerful way to assign large numbers of I/O constraints. We will not cover all of
its features here. Maximize the window called “Design Object List – I/O Pins”

7. Your window should look like the following…

8. Using the data contained in the table, fill in the appropriate values. Grey boxes indicate that setting is not applicable. Ignore the Termination, Delay and Local Clock columns

9. When finished, your window should look something like this:

10. Click on File, then Save.
11. Select the default bus delimiter “Synplify Verilog Default”
12. Close Xilinx PACE.

This concludes the design constraint portion of the tutorial. Constraints are important when doing FPGA Prototyping because more often than not you start off using some sort of development board. To use the various peripherals on the board, you have to understand where they connect into your FPGA and make the appropriate constraints.
10. Implementation

After Synthesis and Design Constraints comes the Implementation phase of the design process. Remember, this is when the tools figure out exactly which slices they will use for specific portions of logic.

To run Implementation, do the following:
1. First click on the top module again (top.v)
2. Then double click on Implement Design under the processes menu.
3. Wait for Implementation to finish, you should see the message Process “Generate Post-Place & Route Static Timing” completed successfully when finished.
4. You should also notice two of the Yellow triangles with exclamation points, one beside Synthesize and one beside Implement.
5. Double click on the “View Design Summary” in the processes tab.
6. You will see a overview of the Device Utilization for your design.

Question: How many Slices were used for this design, and how many are available?

7. In the Performance Summary Section of the Design Summary, you will see “All Constraints Met” under Timing Constraints. Click on these words.
8. As you can see, if you do not specify specific constraints for your clock, Xilinx ISE makes a best effort at minimizing Clock Period. What is the best case clock period reported here?

Does the minimum clock period reported after Synthesis match the minimum clock period reported after Implementation? If not, what are some reasons this may be the case?
10. Bitstream Creation

The final step in the design process involves creating the bitstream to go onto the FPGA. In order to create the bitstream, the design must have been Synthesized and Implemented without any errors. To create the bitstream:

1. Click the top level module (top.v)
2. Double click “Generate Programming File” in the Processes tab
3. You know the file has been successfully created when you see the message “Process “Generate Programming File” completed successfully”

The bit file is created as top.bit and is stored in the same directory as the rest of your files. The bit stream is loaded onto the FPGA using a JTAG interface, which in most cases hooks up to the computer by USB cable.

A Xilinx tool called **IMPACT** does the work of loading the bitstream onto the FPGA.

11. Conclusion

This concludes the tutorial. In reality this document has only scratched the surface of the abilities of the FPGA tools. It is possible through the Xilinx ISE to fully simulate your device operation without an FPGA. It is also possible to debug a design on an FPGA in real-time with a Xilinx product known as Chipscope Pro. Finally, the true power of most FPGAs is in the IP (Intellectual Property) that is embedded on the devices. On a common FPGA today, one would find one or more Embedded Processors, Ethernet Controllers, Clock Multipliers and other such devices. You can quickly get started using these features using something known as the Core Generator in ISE. If you wish to learn more on your own, please reference the documents listed below for more information.

12. References

- Xilinx ISE 9 – In-depth Tutorial  
- Tutorials on using Embedded CPU’s in Xilinx FPGAs 
- Altera Quartus Quick Start Guide (Altera is another major FPGA Vendor) 
- Xilinx XUP Development Board Users Guide 
13. Appendix

Design Description for the Provided Example

The design involves the implementation of a traffic light controller with the following specifications:

- 4-way intersection, no turn lanes
- Red, Yellow, Green Lights
- Yellow light lasts for 10 seconds
- Road sensors in each of the 4 intersection roads
- Green Light controlled by road sensors
- When a lane with red light gets a positive road sensor input, controller counts to 5 and then gives the lane a green light
- Because the LEDs we are using on board do not have multiple colors, we add a flashing aspect as follows:
  - Green light has no flash (always on)
  - Yellow light flashes twice a second (50% duty cycle)
  - Red light flashes 4 times a second (50% duty cycle)

Again, this design was created around the I/O available on a specific development board (Xilinx XUP). The board has five push buttons, four for the different sensors and one for reset. The board also provides for 4 LED’s, one for each of the 4 traffic lights.
reg [63:0] countHalfSecond, countQuarterSecond;
reg flashOut;

wire [2:0] ryg;
reg flipYellow, flipRed;

assign ryg = {red, yellow, green};

always@(ryg, flipRed, flipYellow) begin
    case(ryg)
        3'b100 : begin
            flashOut <= flipRed;
        end
        3'b010 : begin
            flashOut <= flipYellow;
        end
        3'b001 : begin
            flashOut <= 0;
        end
        default : begin
            flashOut <= flipRed;
        end
    endcase
end

always@(posedge Clock) begin
    if(!reset) begin
        countHalfSecond <= 0;
        flipYellow <= 0;
    end
    else begin
        if(countHalfSecond > 64'd50000000) begin
            flipYellow <= !flipYellow;
            countHalfSecond <= 0;
        end
        else
            countHalfSecond <= countHalfSecond + 1;
    end
end

always@(posedge Clock) begin
    if(!reset) begin
        countQuarterSecond <= 0;
        flipRed <= 0;
    end
end
end
else begin
  if(countQuarterSecond > 64'd25000000) begin
    flipRed <= ~flipRed;
    countQuarterSecond <= 0;
  end
  else
    countQuarterSecond <= countQuarterSecond + 1;
end
end