EECS150: Lab 1, FPGA Editor

UC Berkeley College of Engineering
Department of Electrical Engineering and Computer Science

1 Time Table

ASSIGNED	Friday, August 27nd
DUE	Week 3: August 7th – 8th, during your assigned lab section

2 Objectives

This lab is meant to expose you to the bare-bones FPGA platform that you will be using for the rest of the semester. Throughout the semester, you will be introduced to a number of tools that hide the details of the FPGA you are working with so that you can instead focus on your design. Specifically, you will learn how to represent circuits as text. A series of tools will then transform your textual circuit into real hardware on the FPGA. While this approach greatly increases productivity, it is often easy to forget that you are, in fact, designing components that will be mapped to actual hardware. After working through this lab, you will take away with you knowledge of the hardware platform that you are using so that you can make informed design and optimization decisions throughout the rest of the course.

3 Introduction to the CAD Flow

Figure 1 shows the CAD tool flow that you will be using in this lab.

The starting point is the $*$.ncd[1] design file. As you will learn in future labs, this file is generated by the MAP and PAR (place and route) processes that play a part in translating your textual circuit descriptions down to actual hardware. Don't worry about MAP/PAR for now; we will be working with their output (the $*$.ncd file) and later steps in this lab.

We will now (in Sections 3.1, 3.2 and 3.3) explain each of these CAD steps.

3.1 FPGA Editor

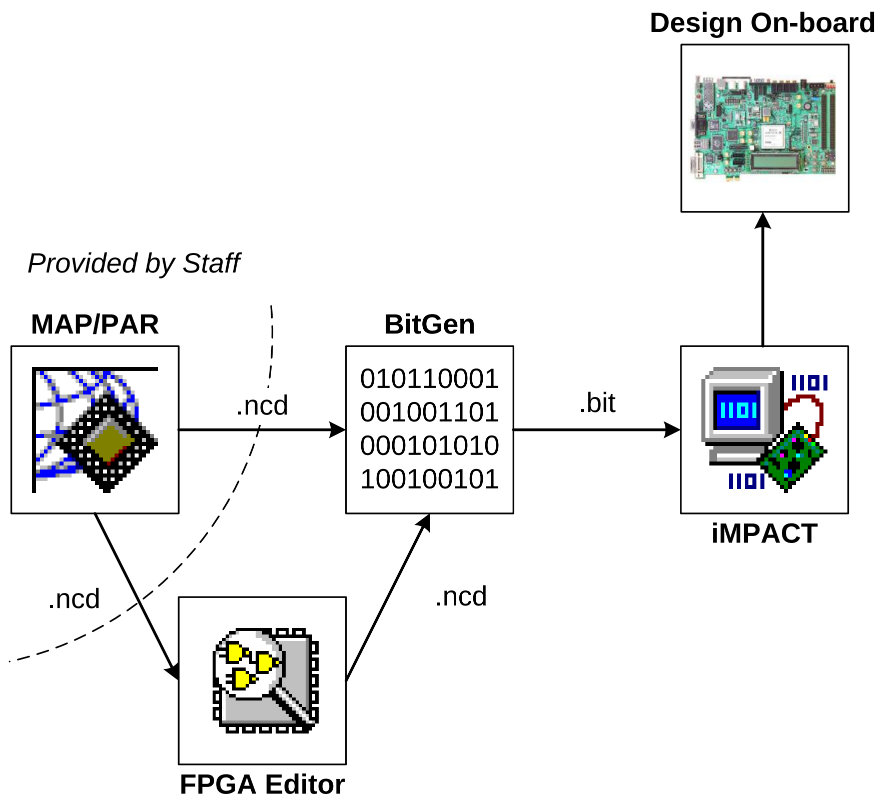
FPGA Editor is a program that allows you to make changes to a circuit that has been mapped to an actual FPGA. The FPGA, as has been discussed, is a matrix of components and wired interconnects. FPGA Editor allows you to look at exactly which of these components are currently in use, how they are being used, and how they have been connected.

FPGA Editor takes a $*$.ncd formatted file as its input. Specifically, the $*$.ncd file represents a circuit that is mapped to a specific FPGAs resources and components. Aside from knowing what this file is supposed to contain, its details are unimportant. What is important is that FPGA Editor modifies the $*$.ncd file based on changes that you decide to make to the circuit. After these changes are made, the modified $*$.ncd is used by the **BitGen** process to manufacturer a $*$.bit file.

Notice in Figure 1 that the output of MAP/PAR also connects directly to **BitGen** as opposed to going through FPGA Editor. This is the typical path that your designs will take this semester: we will

[1] Native circuit description.

Figure 1 FPGA Editor tool flow.



trust the so-called PAR process to do its job better than we can, and not check on its output through FPGA Editor ourselves.

3.2 BitGen

While the ∗.ncd is a file that describes a circuit mapped to an FPGA, it isn't in a format that the FPGA can understand. The idea behind **BitGen** is simple: translate the ∗.ncd into a simpler format that is supposed to be loaded directly to the FPGA. The result of this process is a ∗.bit file that corresponds to the ∗.ncd

3.3 Program Hardware

Once the ∗.bit file has been created, the last step in the story is loading the contents of the ∗.bit file onto the actual FPGA. This is conceptually a very simple step, however, it requires detailed knowledge of the programming cable and the FPGA. We will use a specialized tool to program the board for us.

In this class we'll be using a Xilinx Platform Cable USB II, which is really nothing more than a rather fancy, expensive wire to connect the program, **iMPACT**, to the FPGA. **iMPACT** will then download the ∗.bit file into the FPGA, thereby completing the implementation process.

Once this step is complete, our design is running on actual hardware and we can observe/debug/use it on the XUPv5 boards.

4 PreLab

Please make sure to complete the prelab before you attend your lab section. This week's lab will be very long and frustrating if you do not do the prelab ahead of time.

1. **Read the Configurable Logic Blocks** section of the Virtex-5 FPGA User Guide.
    (a) Pay close attention to the Virtex-5 SLICE (particularly the SLICEL), CLB (Configurable Logic Blocks), where to find the LUTs within the SLICE, and the FPGA fabric that connects all of the pieces together.
    (b) **Answer questions 1a through 1e on the back of this lab (the answers are in the reading!).**
2. **Read Section 3 above**, please make sure you understand it and ask questions ahead of time if necessary.
3. Get a cs150-XX account form from your TA and change your Unix **and** Windows password to a new password of your choosing. (This can be done at the beginning of your first lab session.)
    (a) You must change your Windows password so that you can login to the lab computers in 125 Cory (which are Windows computers).
    (b) You must also change your Unix password for when you are given an SVN[2] account later in the semester.

5 Lab Procedure

In this lab, you will use a tool called FPGA Editor to manipulate logic and nets in a simple design. Throughout the lab, you will verify that your work in FPGA Editor is correct by programming your FPGA with your design file and seeing it come to life in hardware. For simple verification purposes, all exercises in this lab will connect DIP switches to LEDs with some logic placed in between the two. For example, the design file we have provided for you as a starting point implements the logical **or** of two DIP switches and shows the result on an LED (as shown in Figure 5.3). In other words, when at least one of the two DIP Switches is turned to the 'On' position, the LED will light up. If neither of the DIP Switches are 'on,' the LED will remain off.

5.1 Toolflow: FPGA Editor

Before we start working with FPGA Editor, we will first cover how to move your design from FPGA Editor to actual hardware. You will be moving your designs through various tools throughout the course of the lab and you will become intimately familiar with the process throughout the semester.

Now, to get started:

1. Navigate to the CS150 Website and download the Lab1.zip file from the Labs page.
2. Unzip the contents of Lab1.zip into C:\Users\cs150−xx\Lab1Files[3].
    - You should have unzipped:
        (a) DefaultDesign.ncd
        (b) DefaultDesign.pcf
        (c) AugmentedDesign.ncd
        (d) AugmentedDesign.pcf
        (e) Lightshow.ncd
        (f) Lightshow.pcf

5.2 Introduction to FPGA Editor

Before we can dive into manipulating actual hardware, we must become acquainted with FPGA Editor's user interface. To get started, we will first setup an FPGA Editor project.

[2]If you do not know what version control or SVN is right now, don't worry. You will be given detailed instructions on how to set it up and what it is later in the semester.

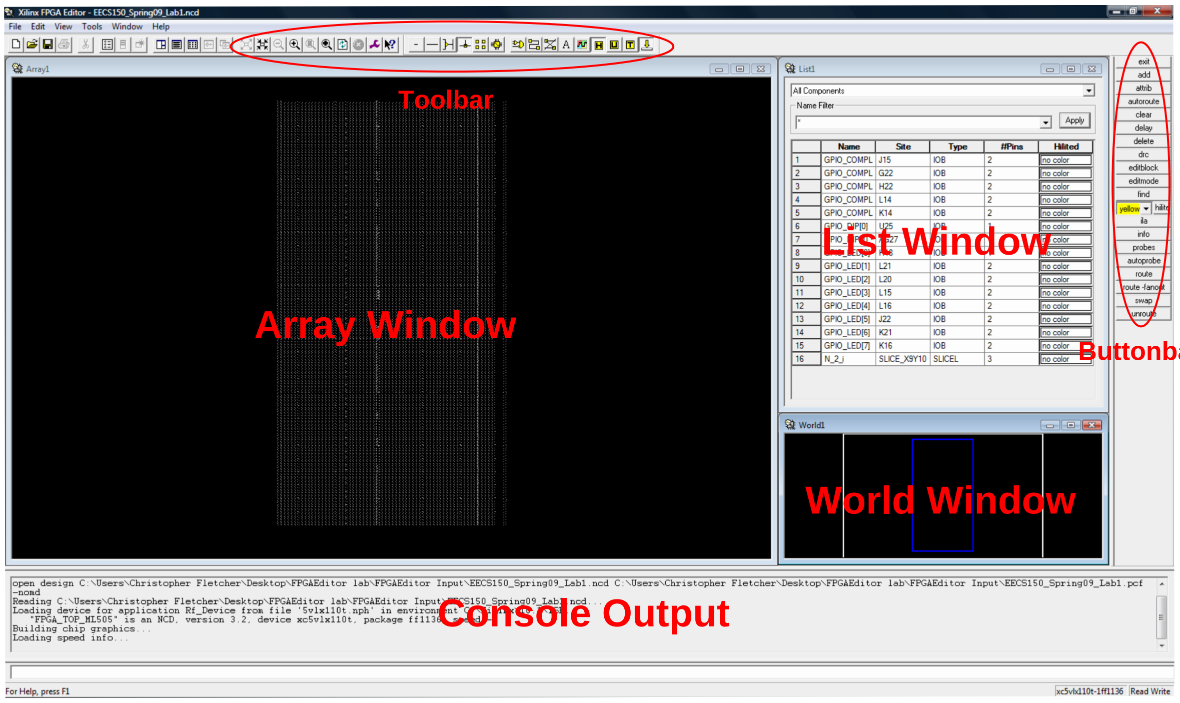
[3]For the remainder of the lab, we will assume that all contents of Lab1.zip have been unzipped to the same directory.

5.2.1 Setting up an FPGA Editor Project

1. **Double-Click** the **FPGA Editor** icon on the desktop to start FPGA Editor.[4]
2. When FPGA Editor opens, click Click **File** → **Open**.
   (a) Under **Design or Hard Macro?** select **Design**.
   (b) For your **Design File**, navigate to DefaultDesign.ncd, which you unzipped from Lab1.zip.
   (c) If the ∗.ncd and ∗.pcf files you unzipped were placed in the same directory, the **Physical Constraints File** section should have auto-completed itself for you by the point. If it didn't auto-complete, navigate to DefaultDesign.pcf.
   (d) Under **Edit Mode**, select **Read Write**.
   (e) Click **Ok** at the bottom of the dialog (leave the section on **Block RAM** data blank).

You now have a ready-to-go project and will be presented with the user interface shown in Figure 2.

Figure 2 A complete FPGA Editor project.



	Name	Site	Type	#Pins	Hilited
1	GPIO_COMPL	J15	IOB	2	no color
2	GPIO_COMPL	G22	IOB	2	no color
3	GPIO_COMPL	H22	IOB	2	no color
4	GPIO_COMPL	L14	IOB	2	no color
5	GPIO_COMPL	K14	IOB	2	no color
6	GPIO_DIP[0]	U25	IOB	1	no color
7	GPIO_DIP[1]	AG27	IOB	[illegible]	no color
8	GPIO_LED[0]	H18	IOB	[illegible]	no color
9	GPIO_LED[1]	L21	IOB	2	no color
10	GPIO_LED[2]	L20	IOB	2	no color
11	GPIO_LED[3]	L15	IOB	2	no color
12	GPIO_LED[4]	L16	IOB	2	no color
13	GPIO_LED[5]	J22	IOB	2	no color
14	GPIO_LED[6]	K21	IOB	2	no color
15	GPIO_LED[7]	K16	IOB	2	no color
16	N_2_i	SLICE_X9Y10	SLICEL	3	no color

```
open design C:\Users\Christopher Fletcher\Desktop\FPGAEditor lab\FPGAEditor Input\EECS150_Spring09_Lab1.ncd C:\Users\Christopher Fletcher\Desktop\FPGAEditor lab\FPGAEditor Input\EECS150_Spring09_Lab1.pcf
-nomd
Reading C:\Users\Christopher Fletcher\Desktop\FPGAEditor lab\FPGAEditor Input\EECS150_Spring09_Lab1.ncd...
Loading device for application Rf_Device from file '5vlx110t.nph' in environment [illegible]
   "FPGA_TOP_ML505" is an NCD, version 3.2, device xc5vlx110t, package ff1136, speed [illegible]
Building chip graphics...
Loading speed info...
```

5.2.2 Navigating an FPGA Editor Project

The main project view shown in Figure 2 presents several windows that you will use throughout the lab. The **Array Window** shows you a schematic of the FPGA you are working with. It will highlight different **Components** of the FPGA, that are currently in use, as well as **Nets** or **Wires**[5], depending on which view mode you are currently using. The **List Window** allows you to find and sort components and nets by their names. The **World Window** shows you a mini-map of the entire FPGA and indicates, with a boxed outline, what region is currently being shown in the **Array Window**. The **Console Output** (at

[4]If FPGA Editor is not on your desktop, you can navigate to it through **Start** → **Programs** → **Xilinx ISE Design Suite 10.1** → **Accessories** → **FPGA Editor.**

[5]FPGA Editor makes a distinction between what is known as a **net** and what you are probably more familiar with: a **wire**. A net is an abstract connection between different components, usually shown as a straight line from component to component. In truth, connections between components in an FPGA are seldom straight lines, as the FPGA is a matrix of interconnects. Thus, the physical connection between two components in your design is called a wire. After two items in your design are **routed**, wires are derived from nets.

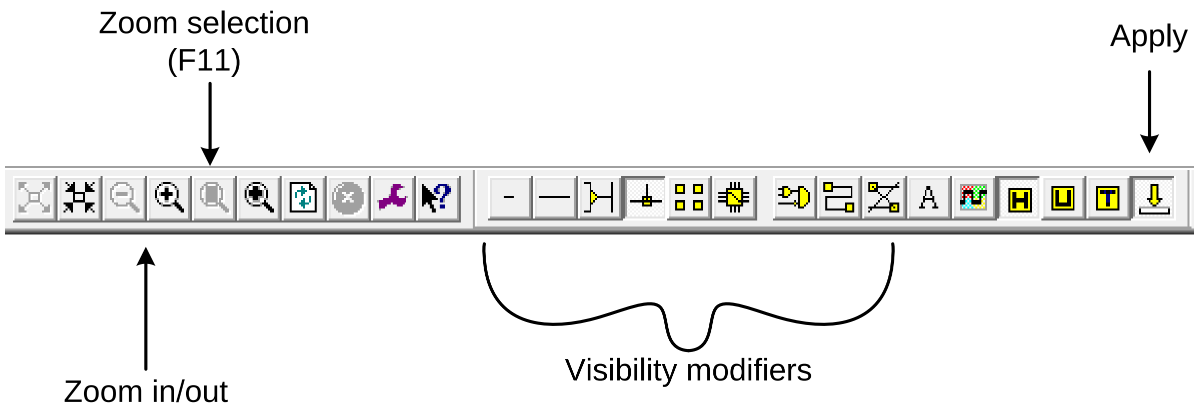
the bottom of the view) shows error messages and other indicators that you may find useful throughout the design process. The **Buttonbar** (to the far right of the screen) features a flurry of useful buttons that we will use throughout the lab. The **Toolbar** (at the top of the view) features several worthwhile buttons and is shown in Figure 3.

Most of the buttons on the toolbar are self-explanatory. You should be aware of two in particular:

1. The **Zoom Selection** button will automatically zoom you to whichever net/component you have selected in the **List Window**. This is very useful as finding components in the **Array Window** is very difficult due to the FPGA's complexity.

2. The **Apply** button (to the far right of the toolbar) must be pressed/enabled.

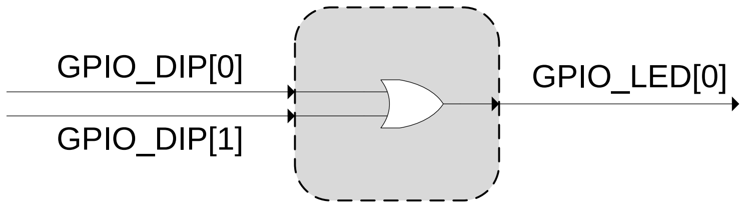
Figure 3 The main FPGA Editor toolbar.



5.3 The Default Design

Now that you are familiar with the main FPGA Editor view, we can take a look at the default design that came with DefaultDesign.ncd. This design is very simple. As mentioned in Section 5, the default design implements the abstract circuit shown in Figure 4. On the FPGA board, GPIO_LED[...] stands for **General Purpose Input/Output LED**. The ... indicates that there are a vector of LEDs to choose from (8 on the XUPV5). Likewise, GPIO_DIP[...] stands for the general purpose **DIP Switches** found on in the bottom right corner of the board. Since there are 8 DIP Switches on the DIP Switch panel in the bottom right, this is also a vector whose indices range from 0-7. Thus, the default circuit performs the **or** of the bottom two DIP Switches and shows the result on the LED indexed by 0.

Figure 4 The "Default Design" circuit.



We will now explore this circuit down to the LUT that implements the **or** of the two DIP Switches.

1. In the **List Window**, locate the component that is of type **SLICEL** and write down its **Site** on the check-off sheet for this lab (Question 2a).

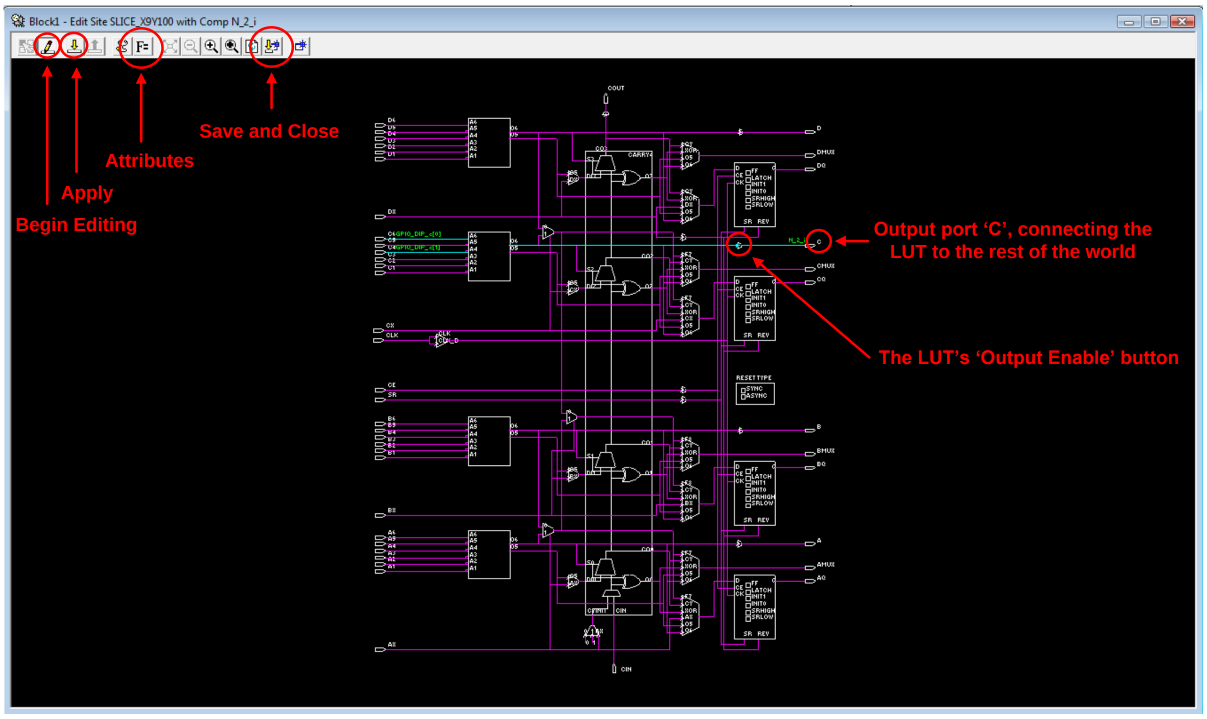
2. **Left-Click** on the SLICEL component and press **F11** to zoom to its physical location on the FPGA.

3. **Left-Click** the SLICEL in the **Array Window** that appears when you press F11.

4. **Left-Click** the **editblock** button on the **Buttonbar** at the far right of the screen.

You should now see a window (shown in Figure 5) showing the internals of the SLICEL you selected. **Before continuing, make sure that the window background is black.** A black background indicates that you can write/modify the SLICEL. If the background is gray, refer to Section 5.2 to set the **Edit Mode** to **Read Write**. If the **Edit Mode** is set correctly, make sure the **Begin Editing** button in the SLICEL window toolbar (shown in Figure 5) is pressed.

Figure 5 A Virtex-5 SLICEL.


Block1 - Edit Site SLICE_X9Y100 with Comp N_2_i
Save and Close
Attributes
Apply
Begin Editing
Output port 'C', connecting the LUT to the rest of the world
The LUT's 'Output Enable' button

On the left-side of the SLICEL, you will see the four 6LUTs that come with the SLICE. Each of these LUTs has two outputs. Our default design only uses one output from one of the LUTs. Find the name of the LUT output port (not the output port from the SLICE) that corresponds to this output line and fill in its **name** on the check-off sheet for this lab (Question 2b).

Right now, we have fairly decent visibility of our design. We know exactly which SLICE implements our simple design on the FPGA. We also know which LUT in the SLICE is doing all of the work. Let's dig deeper and verify that the logic function implemented by the LUT is exactly what we expect (namely an **or**).

1. In Figure 5, find the **Attributes** button on the SLICE toolbar and press it.

2. Scroll down in the rows of attributes until you see the line that clearly implements our circuit (it should look like two symbols (the two inputs) and some operator between them).

3. Write down the contents of the row on the check-off sheet for this lab (Question 2c).

You now have complete visibility of your design. You have found, down to the LUT function generator itself, how our simple circuit is implemented on the FPGA. We will now employ the **BitGen** and **iMPACT** tools to see this circuit come alive in hardware.

5.4 Toolflow: BitGen and iMPACT

In order to push our design to hardware, we must run **BitGen** to create a ∗.bit file and then run **iMPACT** to program the FPGA with the ∗.bit file.

5.4.1 BitGen

1. Close the SLICE window that you were looking at in Section 5.3.

2. In the main FPGA Editor view, click **Tools** → **Probes**.

   (a) In the **Probes** dialog box, look to the right and click on the **Bitgen...** button.

   (b) In the **Run Bitgen** dialog box, click **Ok**.

If the process was a success, the console output window at the bottom of the screen should show **Creating bit map...** for a while and then read **Done.**

5.4.2 iMPACT

To open **iMPACT** through FPGA Editor:

1. In the main FPGA Editor view, click **Tools** → **Probes**.

   (a) In the **Probes** dialog box, look to the right and click on the **Download** button.
      - The **iMPACT** UI shown in Figure 6 should now be open.

   (b) Make sure that the **Platform Cable USB II** is connected to the **JTag** port on the XUPV5 board and that the XUPV5 board is on.
      - The little light on the **Platform Cable USB II** will turn green when the cable detects that it is connected to a **powered Xilinx chip**.
      - The power switch for the XUPV5 board is in the upper right of the board.

   (c) In the **iMPACT** UI, click **Initialize Chains**.

   (d) When prompted to **Assign New Configuration File**, click **Bypass** four (4) times.
      - Of the 5 chips in the main **iMPACT** window, you want to **Bypass** until the right-most chip is colored green.

   (e) When the right-most chip is green, select DefaultDesign.bit in the **Assign New Configuration File** dialog and click **Open**.

   (f) When the **Device Programming Properties** dialog opens, click **Ok**.

   (g) **Right-Click** on the right-most chip and press **Program**.

   (h) Again, when the **Device Programming Properties** dialog opens, click **Ok**.

After several seconds a blue banner reading **Program Succeeded** should appear. This means that your board has been programmed successfully!

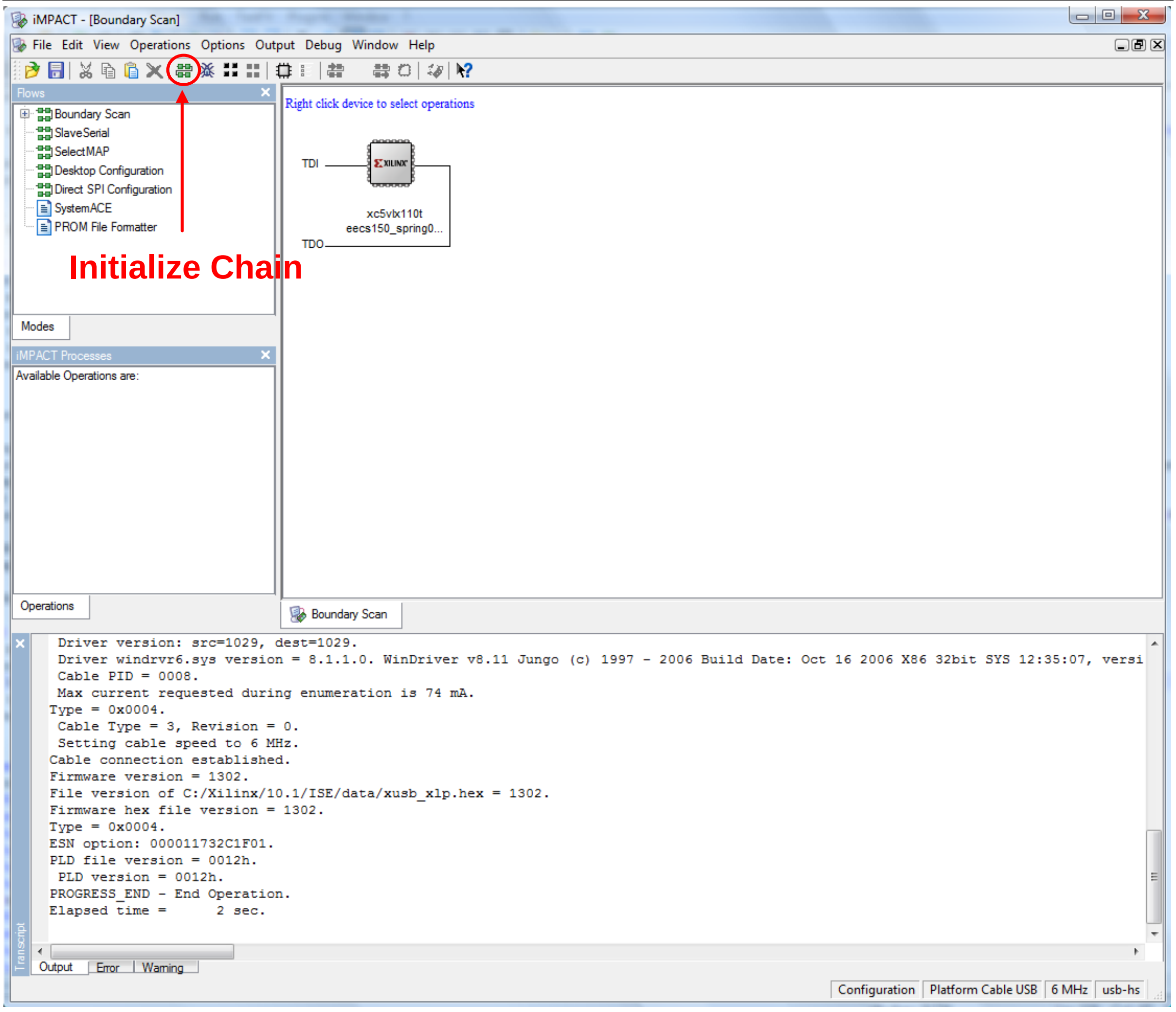
Now that your board has been programmed, we will verify that the default design does what it should do.

1. In the bottom-right of the XUPV5, locate the GPIO DIP SW bank of switches and flip the switches labeled ‘1’ or ‘2’.

2. Verify that an LED lights up when one or both of these switches are turned to the ‘ON’ position.

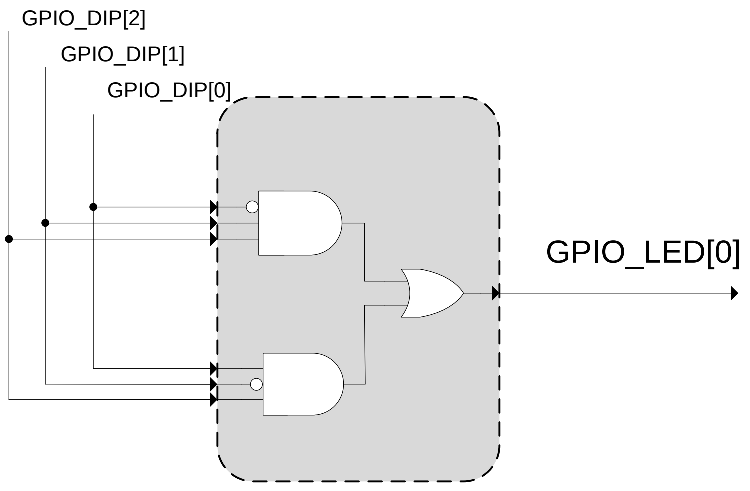
3. Be prepared to point to which LED lit up to your TA for check-off (Question 2d).

Figure 6 The iMPACT user interface.



```
 Driver version: src=1029, dest=1029.
 Driver windrvr6.sys version = 8.1.1.0. WinDriver v8.11 Jungo (c) 1997 - 2006 Build Date: Oct 16 2006 X86 32bit SYS 12:35:07, versi
 Cable PID = 0008.
 Max current requested during enumeration is 74 mA.
Type = 0x0004.
 Cable Type = 3, Revision = 0.
 Setting cable speed to 6 MHz.
Cable connection established.
Firmware version = 1302.
File version of C:/Xilinx/10.1/ISE/data/xusb_xlp.hex = 1302.
Firmware hex file version = 1302.
Type = 0x0004.
ESN option: 000011732C1F01.
PLD file version = 0012h.
 PLD version = 0012h.
PROGRESS_END - End Operation.
Elapsed time =      2 sec.
```

Figure 7 The new logic function featuring a 3rd DIP switch.



5.5 Augmenting the Design

Now that we know how to look at the bare-bones of our design, we are going to make some modifications to show off the power of the LUT. Specifically, we are going to add a DIP Switch to our design, connect it to the LUT that was computing **or** in Section 5.3, and make the logic function implemented by the LUT a bit more interesting. We would like to change the logic function to that shown in Figure 7

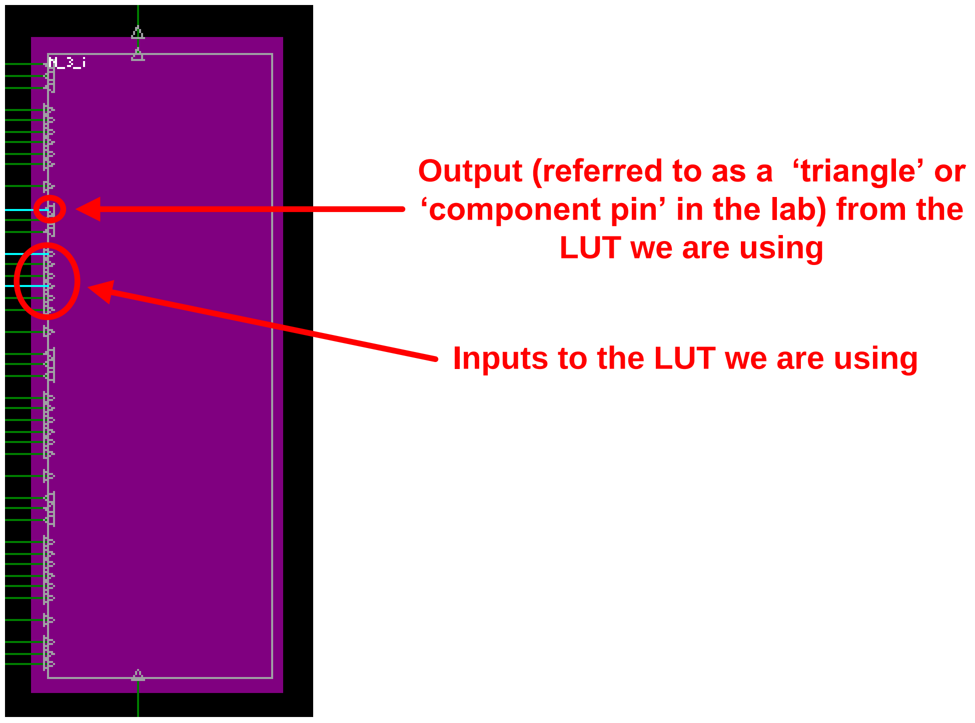
Take a moment to appreciate the power of this modification. It takes no more LUTs to implement the relatively complex function shown in Figure 7 as it does to implement the simple **or** gate shown in Figure 4. This is because LUTs are **function generators** that can accommodate any function representable with the number of input ports they support. Furthermore, how a LUT is used doesn't change the delay through it. **Know that the LUTs in the Virtex-5 parts you are using are 6LUTs**[6]

Now we will make the changes to our design to accommodate the new requirements.

1. Open FPGA Editor using AugmentedDesign.ncd as your design file.
    - If you need help opening FPGA Editor, refer to Section 5.2.
    - AugmentedDesign.ncd should look identical to DefaultDesign.ncd at first glance. The only difference is that we have setup the I/O parameters for the third DIP switch for you, a process that requires knowing several hard-to-explain settings.
2. In the **List Window**, find the component whose type is a SLICEL (this SLICE will be implementing the same **or** function that it was originally).
3. Click on the SLICEL component and use **Zoom Selection** (F11) to zoom to the SLICE in the **Array Window**.

So far, what we have done is familiar from Section 5.3. Before opening the SLICE, however, we are going to attach the third DIP Switch to the SLICE.

Figure 8 The input and output ports of the SLICEL.


Output (referred to as a 'triangle' or 'component pin' in the lab) from the LUT we are using
Inputs to the LUT we are using

[6]6LUTs have 6 **inputs**. This convention holds for other LUTs such as the 4LUT (4 inputs) or 5LUT (5 inputs).

Shown in Figure 8 are the nets connected to the SLICEL and the input/output pins[7] for the SLICEL. Circled are 6 of the **triangles**/pins pointing towards the right. These are the 6 **inputs** to the 6LUT we are using. Two of the 6 have light-blue lines extending into them. These are the nets that connect, on the other end, to GPIO_DIP[0] and GPIO_DIP[1].[8] This is expected: this SLICE implements the **or** of these two DIP Switches.

We are now going to attach the third DIP Switch to the LUT in this slice.

1. At the top of the **List Window**, select **Unrouted Nets** from the drop-down box.
2. Verify that the only item left in the **List Window** is one called GPIO_DIP_c[2].
3. Click on one of the unused **triangles** amongst the 6 that are circled in Figure 8.
   (a) Make sure to only select the **triangle** and not the line itself.
      - You can check yourself because if the triangle is selected, it alone will turn red. If the line is selected, on the other hand, the line will turn red.

   (b) Make sure that the triangle you selected is a component pin and not just a site pin.
      - You can check this in the Console Output (see Figure 2) section of the FPGA Editor view. If the triangle you selected is only a site pin, the message you will see when you select the triangle will look something like site .pin = SLICE_X17Y100.CXX (this is **NOT** a triangle that will work). If the triangle is a component pin, the message will look more like site .pin = SLICE_X17Y100.CXX, comp.pin = N_XX_i.CXX. Notice that the message is longer and contains the text comp.pin. **If, in the next steps, you get the error message "WARNING:FPGAEditor:783 - There is no comp pin to add to a net." your triangle is not a component pin and will not work.**
4. While the triangle is still selected, **hold down** the **Ctrl** key and click the unrouted net labeled GPIO_DIP_c[2] in the **List Window**.
5. With both the triangle and the unrouted net selected, click the **route** button on the **Buttonbar** (shown in Figure 2) at the far right of the screen.
6. Verify that you see a console output message that reads:
   ERROR:FPGAEditor:361 − Nothing found to route. Ironically, this means that everything is ok.
7. Click on the **autoroute** button, also found on the button bar.

At this point, you should see the port you selected connected with a light-blue line, just like the other two that were there originally. Now that we have connected the third DIP Switch to the SLICE, its time to change the logic within the LUT that implements the logic.

1. Follow the instructions given in Section 5.3 to get to the screen where you can modify the **Attributes** for a specific LUT.
2. On the attributes line for the LUT that implements **or**, make the changes necessary so that the LUT implements the logic described in Figure 7.
   - When making logic changes, you may use the following symbols:
     (a) **and**: *
     (b) **or**: +
     (c) **not**: ~
     (d) **xor**: @

[7]Specifically, the pins on a component such as the SLICEL are known as component pins or casually referred to as "**triangles**."

[8]Above the two used inputs, you will also see a solitary output (denoted by a triangle pointing in the opposite direction as the inputs, extended by a light blue line). This output is tied to GPIO_LED[0] and finishes the connection from the DIP switches to the SLICEL to the LED bank. This net is already connected for you - don't modify it.

- When changing the logic within the LUT, pay close attention to which of the LUTs input ports (A6, A5, . . .) correspond to which DIP Switches.
- Do not modify the **Config** line in the **Attributes** window. When you **Apply** your changes (see below), the **Config** line will automatically update itself to reflect your changes.

3. When you are finished modifying the LUT logic, click the **Apply** button *followed by* the **Save and Close** button in the LUT view (see Figure 5).

When you finish making your changes, its time to move our new design to hardware and verify that it worked.

1. Follow the instructions in Section 5.4 that describe how to run **BitGen** and **iMPACT**.
2. Verify that your design works in hardware by manipulating the three DIP Switches.
3. Show your working design to your TA for check-off (Question 3a). **Since there is one more section in this lab that requires you to demo a design in hardware to your TA, save your ∗.bit file from this section someplace safe for the time being.**

After you have successfully re-programmed your FPGA board and have verified your augmented circuit, take a moment to think about some optimizations that you can make to future designs based on what you now know about FPGA Architecture. Specifically, consider the 6LUT. We have added a relatively significant amount of logic to our simple **or** gate yet used no additional logic resources in the FPGA. This is because the 6LUT is indifferent to the complexity of your logic functions, but limits us to using its 6 inputs. We get to create logic functions that use 1 to 6 inputs with the same amount of FPGA resources regardless of whether we use 1 or 6 (or somewhere in-between) inputs.

On the flip side, as our logic functions require more than 6 inputs, we must employ more than one 6LUT. From the PreLab, we learned how to implement 7 and 8 input functions in a single SLICEL. Specifically, if we have a 7 input function, we must use two 6LUTs (takes up $\frac{1}{2}$ of the LUTs in a SLICEL). If we have an 8 input function, we must use four 6LUTs (takes up all of the LUTs in a SLICEL). Given that our designs in this lab have only used $\frac{1}{4}$ of the LUTs in a SLICEL, the resources we use, when upping the number of inputs to our logic functions, increase substantially. Take this to heart in your future designs. In general, optimize your work based on general good digital design principles and the platform you are targeting (in this case the Virtex-5 LX110T FPGA).

5.6 LED Lightshow

Now that you know how to modify the logic function in a LUT and how to manipulate nets, its time for the grand finale, called the **Lightshow**. The lightshow (whose circuit is shown in Figure 9 is a simple circuit that lights up multiple LEDs in sequence, effectively creating a lightshow. The catch is that this circuit only functions properly when an Enable signal is high. Your task in this section is to fix the Enable signal, program the board with the fixed circuit and demo the lightshow to your TA.

To begin, open a new instance of FPGA Editor with the lightshow design:

1. Open FPGA Editor using Lightshow.ncd as your design file.
   - If you need help opening FPGA Editor, refer to Section 5.2.

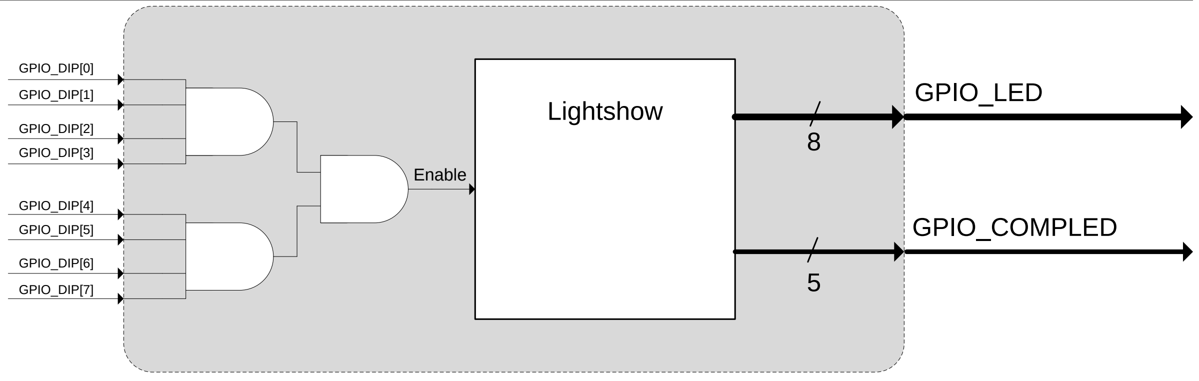
If you zoom into the **Array Window** once the design is loaded, and play with the toolbar visibility modifiers (see Figure 3), you will see that this circuit is far more complex than anything you have seen before. Although Figure 9 abstracts the complexity of the circuit behind a block, the circuit itself looks like anything but a block. Like any other circuit instantiated on an FPGA, the lightshow is composed of numerous interconnects between CLBs and the SLICEs contained within them.

As mentioned above, your task is only to implement the Enable signal logic. The rest of the lightshow (the portion in the box labelled "**Lightshow**" in Figure 9) has already been completed for you. The trick is that we have to figure out exactly where the Enable signal is generated so that we can make it perform the function that we want it to.

Fortunately for us, FPGA Editor is kind enough to reserve a sensible name for the nets and components that we are interested in. If you look through the **List Window** (when **All Components** is selected in the drop down), you will find a SLICEL called "**Enable**." This is the SLICE that generates the Enable signal from Figure 9.

If you take a closer look at the SLICE called **Enable**, you will notice that its **B** output port is connected to the Enable signal. Right now, this output is only driven by 2 DIP switches (specifically, it is connected to by nets called GPIO_DIP_net[0] and GPIO_DIP_net[1] which themselves connect to the I/O pads for the DIP switches). **Your task is to connect the rest of the DIP switches up to the Enable output such that your Enable signal is driven by the same logic as what is shown in Figure 9.**

Figure 9 The lightshow circuit.



In order to match the specifications in Figure 9, your solution must implement an 8 input **and** gate. To make this section more interesting, you are required to do this in **two ways** - with:

1. Two 6LUTs. This method will use one or two SLICELs.
2. Four 6LUTs. This method *must* use one SLICEL.

Before you begin, please see Section 5.6.1, below. Once you have implemented the Enable logic shown in Figure 9, you will once again push your solution to hardware for verification. Step-by-step:

1. Follow the instructions in Section 5.4 that describe how to run **BitGen** and **iMPACT**.
2. Verify that your design works in hardware by manipulating **all 8** DIP Switches.
3. Show one version of your working design to your TA for check-off (Question 4a).
4. Show FPGA Editor screenshots, of each implementation[9], to your TA. (Following Questions 4b-4c).
5. Answer thought question 4d.

5.6.1 Lightshow Hints

The lightshow circuit is very complex compared to what you have seen before. Don't try to understand or follow everything that is going on. Instead, focus your attention to the SLICEL named **Enable** and to the **unrouted** nets GPIO_DIP_net[2] ... GPIO_DIP_net[7].

One thing to keep in mind as you are trying to perform the following more complex tasks with FPGA Editor is that the tools are not perfect. If you find yourself spending excessive amounts of time trying to do simple tasks, please make sure you have closely read each of the following hints. If you still find that you are experiencing difficultly, ask the staff for help.

[9]Open the SLICE window and take a screenshot of the SLICE(s) that you modified. Be prepared to explain what you did.

1. Think about how you are going to implement the 2× and 4× 6LUTs versions of the **Enable** function. Don't start until you have a proposed solution.

2. Consider implementing the 2× 6LUT version first. Experience tells us that students have a simpler time with this implementation.

3. GPIO_DIP_net[0] and GPIO_DIP_net[1] have already been connected for you, but the others are **unrouted**. To route them properly, follow the instructions in Section 5.5.

4. At some point, you may want to **Add a Net**. This is useful for connecting arbitrary components together. To add a net:

   (a) Hold down the **Ctrl** key and click each of the **triangles**/component pins that you want to connect together.

   (b) Click **Add** on the button bar as seen in Figure 2.

   (c) Give your net a name[10] of your choosing, and click **Ok**.

   If you want to connect two components that are so far away you can't **Ctrl** + click both of them on the same screen:

   (a) Click one of the triangles (corresponding to one end of the net) and click **Add**.

   (b) Find the other triangle in your design (using F11 or another means of jumping around the FPGA fabric).

   (c) **Ctrl** + click the **unrouted net** which you added in the first step and the triangle that you just jumped to.

   (d) Click **route** in the button bar from Figure 2.

   (e) Click **autoroute** to finish routing.

5. **If you connect a new LUT:** you must make sure that its output is properly driven.

   (a) In Figure 5, locate the **Output Enable** button for the LUT that is being used (it should appear as a triangle pointing to the right).

   (b) Click the corresponding **Output Enable** triangle for your LUT so that the line connecting your LUT's output to the SLICE's output port is **light blue** (such as in Figure 5).

6. Make sure the nets that you connect to different SLICEs have corresponding logic in the LUT that they connect to. For example, if your 6LUT uses the input A6, you **must** connect the pin that corresponds to that input to some net. If there is a mismatch between your nets and the logic within your SLICE, you will see errors such as **Dangling pins** or **Partially routed design**. These errors will prevent BitGen from running properly.

7. Should you use the SLICE's internal multiplexing ({F7A, F7B, F8} MUX), it is important to realize that you **cannot** successfully autoroute nets connected to the selectors for the internal MUXs (AX, BX, CX) **if you have already connected inputs to the LUTs**. You will need to first disconnect the inputs to the LUTs if you wish to route connections to the internal multiplexors.

8. When attempting to delete an element, it is important to distinguish between wires and nodes. Generally speaking, selecting a wire that is part of a net is equivalent to selecting the net. Therefore, selecting a wire and pressing Del will have the effect of deleting the net. When trying to remove one node from the net, you will instead want to select that individual node (triangle) and press Del. If you are still experiencing trouble, try first unrouting the net from which you wish to remove nodes.

9. If you are having trouble determining how to use the internal MUXs, try playing around with the various triangles inside a SLICE while in Block View.

[10] When naming your nets, make sure to not include spaces. Spaces in net names will cause unpredictable errors in FPGA Editor.

10. **Save often!** This, of course, goes for all your work in this class, but even more so for this lab. The undo features of FPGA Editor leave a lot to be desired. You may very well be unable to undo your last command or you may find the program crashing rather frequently. Be patient, save often.

$\Longrightarrow$ **Watch the console output at the bottom of the screen constantly. If you did something wrong, the console is the only mechanism that is going to send out an alert.**$\Longleftarrow$

6 Lab 1 Checkoff

ASSIGNED	Friday, August 27nd
DUE	Week 3: August 7th – 8th, during your assigned lab section

Man Hours Spent	**Total Points**	**TA's Initial**	**Date**	**Time**
	/100		/ /	
Name	**SID**	**Section**		

1. PreLab .... ________ (25%)
   (a) How many SLICEs are in a single CLB (Configurable Logic Block)? .... ________
   (b) How many inputs do each of the LUTs on a Virtex-5 LX110T FPGA have? .... ________
   (c) How many of these LUTs does the LX110T have? .... ________
   (d) How can you implement a logic function of 7 and 8 inputs in a single SLICEL?

   ____

   ____

   (e) What is the difference between a SLICEL and a SLICEM?

   ____

   ____

2. Default Design .... ________ (25%)
   (a) SLICE Site # that is used to implement **or** .... ________
   (b) LUT output port name that is used to implement **or** .... ________
   (c) LUT attributes used to implement **or** .... ________
   (d) Show (or point to) the LED for your TA .... ________

3. Augmented Design .... ________ (25%)
   (a) Show your working design, on hardware, to your TA .... ________

4. Lightshow .... ________ (25%)
   (a) Show your working design, in hardware, to your TA .... ________
   (b) Show the 2× 6LUT screenshot to your TA. .... ________
   (c) Show the 4× 6LUT screenshot to your TA. .... ________
   (d) The 4× 6LUT implementation requires more LUTs than the 2× 6LUT implementation. Why would you ever choose it over the 2× 6LUT implementation?

   ____

   ____

Rev.	**Name**	**Date**	**Description**
E	Austin Doupnik & Mike Eastham	8/27/2010	Moved lab to Fall 2010. Removed old command line information
D	Chris Fletcher	1/20/2010	Moved lab to Spring 2010. Added 2× implementation requirements for the “Lightshow” circuit. Added clarification when launching iMPACT.
C	Kyle Wecker & Chris Fletcher	8/29/2009	Expansion of hint section & Minor terminology updates
B	Chen Sun & Chris Fletcher	1/22/2009	Fixed typos
A	Chris Fletcher & John Wawrzynek	1/2/2009	Wrote new lab