5-2016

Conversion of Digital Circuits Labs

Caleb N. Taber
ETSU

College of Business and Technology

Honors Thesis Signature Approval Form

Conversion of Digital Circuits Lab

Sprint ,4/6/2017>

The members of the Thesis Committee

approve the Senior Honors Thesis for

Caleb Taber _______________________________________

Thesis Committee Chair

Dr.Hugh Blanton _______________________________________

CBAT Thesis Committee Member

Dr. Paul Simms _______________________________________

External Thesis Committee Member

Cathy Whaley _______________________________________

CBAT Honors Director

Dr. Suzanne Smith ___________________________________

CONVERSION OF DIGITAL CIRCUITS LABS

Taber, Caleb N.

ETSU COLLEGE OF BUSINESS AND TECHNOLOGY

Contents

Abstract: 3
Problem and Background 5
Review of Literature 7
Research Methodology: 10
Results 12
Labs 12
Conclusions 12
Appendix A: Setting Up the Basys 3 14
Step 1: 14
Step 2: 14
Step 3: 15
Step 4: 15
Step 5: 16
Step 6: 16
Finishing Labs in Vivado: 17
Step 1: 17
Step 2: 18
Appendix B: VHDL programming 20
Appendix C: Labs 26
Lab 1: Logic Gates 26
Making the gates 27
Lab 2: Karnugh Maps 29
Lab 3: NAND Gates 31
Lab 5: BCD Down Counter 33
Appendix D: Sample mental model test 35
Works Cited 37

Abstract

The engineering technology department at ETSU currently lacks a modern method to teach digital circuits. The aim of this thesis is to convert our current digital circuits labs to equivalent labs suited to run on the Basys 3. The Basys has several advantages over the aging NI Elvis boards (and now just breadboards) currently in use. The first advantage is that the Basys gives students a taste of FPGA programming without being overwhelmingly; like the systems currently in place for the digital signal processing class. The Basys is also a more modern system; our current integrated circuit and breadboard system is from the 70's and has little to do with the modern world of electronics.

There are several major difficulties with moving towards the Basys 3. It requires several tweaks to the current computer security setting of the lab computers. The other issue to be solved is that very few people in the department have even an inkling of how to program in VHDL and most of them are outgoing students. This lack of skills could be a threat to the class but I have included an appendix and a few recommendations for books on the subject to ensure that system development can continue.

The other objective of this project was to see if there were ways to incorporate new educational techniques into the engineering technology curriculum. While there have been no actual tests on students, the groundwork has been laid to use some new ideas in the classroom. All of these new systems are designed to get students to think about how devices actually work and develop models to help them fully understand what is being taught.

Problem and Background

The current systems used for teaching digital circuits are currently broken. Most of the NI Elvis boards, the current training device for most of the electronics classes, are simply outdated. They can no longer be used with the current software as the serial ports used for connection to computers have been depreciated.

The Baysis 3 FPGA development board overcomes many of the weakness of the NI Elvis trainer. The first advantage is that the Baysis is much more sturdily built and has fewer points of failure that can have wire broken into them. The Basys is quite well made. One board used for this thesis has seen daily use for over two years with no ill effects. Overall the Basys 3 is a tank compared to the NI Elvis, with a much smaller footprint. While this sounds like a sales pitch, our current methods really are quite outdated and this new solution has been long overdue.

The Baysis 3 allows students to get some experience in programming. An emphasis on programming will soon be of greater importance as companies are increasingly doing away with component level logic and moving towards board based designs. The Basys 3 also uses a more modern software suite which would allow students to have a greater understanding of modern programming topics instead of the current system where programming is barely studied in the classroom. The VHDL language used is also one of the lesser studied languages in college curriculums, and there is currently no VHDL language class at ETSU. Adding any type of hardware design would be helpful for students pursuing careers in hardware design.

VHDL was originally developed in 1980 as a way for the military to program ICs included in equipment. It has been updated until recently, but early versions lacked the standard types and had commands that were deemed unnecessary (IEEE, 2002). It has since been used to program everything from missiles to logic boards. It is also one of the most used hardware description languages, along with Verilog which is also useable on the Basys 3.

The goal of this thesis is to design a new lab curriculum and new labs for the digital circuits class; including converting most of the labs from NI Elvis form to a version that can use Baysis 3 and VHDL to teach digital circuit concepts while maintaining coverage of integrated circuit concepts. While this seems like an easy task, it assumes that someone in the department knows VHDL and even how to set up the Baysis 3. I have left several recommendations with staff for new learning materials. There will remain a need for continuing research for VHDL information and perhaps a continuation of this thesis.

A secondary goal of developing a new curriculum is to try to improve other classes in the engineering technology department. Some observations have revealed that most electronics and programming classes develop what is known as a double curve, where many of the students excel and another large percent of the class learn nothing. Larger studies with programming classes show that this phenomenon is not confined to ETSU; it is a universal issue that exists, regardless of student enthusiasm, teaching style, or previous grades. According to current research, the reason this occurs is that some students are able to create a mental model of how digital logic (or electronic principals, or digital signal processing) works and some students are unable to (Dehnadi & Bornat, 2006). The first change would be to introduce a pre-test over basic logic to see which students quickly develop mental models and which may need extra help. Introducing the Baysis 3, along with classroom lectures and one physical breadboard lab may allow for students to experience circuit design from several different angles and allow these students who have fallen behind to develop mental models.

Review of Literature

Due to the newness of the Baysis 3, there is little literature available specially for the board. So far, all that has been published by the manufacturer is a specifications sheet, along with several articles on a website that hosts electronics tutorials. This lack of any literature or information delayed research while Digilent and National Instruments published new information. Thankfully, there is a community of users and programmers who have published several resources which can be used to find out how to set up the Baysis 3. These often fail to provide any advice on how to actually use the Baysis 3, which is where several other important sources came in.

Digilent's documentation assumes that the reader is already proficient in VHDL and only gives explanations for how to handle certain hardware issues. This provided some valuable information to explain how to program the seven segment display and also made sure to spell all the features of the Basys 3. The reference sheet was too technical for a digital signals lab but could be useful for more advanced work. There is no need for the digital circuits class to use any inputs or outputs besides the switches and seven segment display.

Other less orthodox resources were required to convert the labs. The first step was using a video tutorial from California polytechnic's CPE133 class by Andrew Danowitz called *vivado and basys3 getting started*. This was an important video for getting started, as it shows exactly how to set up the board for programming, the specifics of which are somewhat glossed over in Digilent's own literature. The final set of hardware instructions I used was a tutorial from the Digilent sales team called *How-to-use-Verilog-and-Basys-3-to-do-stop-watch*. While this is an unusual choice for a thesis it actually has some valuable code, which was necessary for use in one lab. While this code was not directly copied, I was able use it to understand a harder point of how the system timer functions on the Basys 3 board.

The most important source I have found is the book Free Range VHDL. VHDL is a difficult language to find resources for because it is only used for hardware design. The lack of resources for VHDL is so severe this book is essentially the only resource I have found. There may be other resources but this is the only one that could be easily located. It is fortunate that Free Range VHDL is a well written resource. I have successfully taught an entire lecture using this textbook. I heavily recommend it as a textbook for the entire class and also as an invaluable reference. The book has been tested. It has been used in this research project, and in a single class a book by the same authors was used by Dr.Tan. Free Range VHDL is also available as a free, legal PDF which may help make it more popular among students.

Another similar source is the page of a Professor at UC Irvine named Dr. Jayram, Moorkanikara Nageswaran who archived several VHDL reference guides and textbooks on his personal webpage. While many of the guides do not seem to have the most proper citations, many of the websites have gone offline. The guides he has saved are somewhat out of date, but not so much as to be useless. Many of the obsolete commands are marked as such in the text and workarounds are often offered. The only real issues with his guides are that that not all of the documents are properly cited and many of the guides are not specific to the Basys 3.

The only area where literature is readily available is related to teaching electronics. Numerous papers have been written on how the double curve arises and on how to most effectively teach an electronics class. The first paper I found in the ETSU archives was “Teaching

Electronics and Laboratory Automation Using Microcontroller Boards" which, despite being an article in a chemistry journal, does show that that microcontrollers can be useful lab aids. While there are few opportunities to use the external ports for the Baysis 3 in digital circuits it does have some possible uses outside this class as a way to collect data from labs and control other machinery.

Another paper which was relevant to teaching digital circuits was Bridging the gap between textbook and real applications a book that could have been custom written for our engineering technology program. It addresses our specific problems, namely that industry uses FPGA's and programming while our textbooks and lectures still use 74 series integrated circuits to teach. This paper outlines how to run a basic class with FPGA development including labs and how to best teach class skills. My proposed curriculum does include some major deviations from the one recommended in Bridging the gap between textbook and real applications. The main difference I propose is to remove most of the discrete integrated circuit based labs they recommend and simply replace them with one experiment with a 555-integrated timer. This way students still receive IC training, but less time is relegated to technology from the 70's. We do not need to worry about needing to explain VHDL since this would be the first class most students take with any programming. Using Dr.Gupong Wang's model, along with our current classes as template, may assure the best results. However, this method still must deal with the problem of the double curve. It may seem odd that this one subject is covered so thoroughly but between 30 and 60 percent of students fail programming classes. I also wish to improve our curriculum as a whole using The Camel Has Two Humps by Saeed Dehnadi and Richard Bornat. The Camel Has Two Humpsis mostly dedicated to how there seems to be a universal tendency for computer science classes to develop a double curve; this is where rather than one normal curve of grades there are two smaller ones with half the class failing. Thankfully the paper also explains how to administer tests to check how students understand their assignments. The test was also included in the paper itself along with the logic used to make the test, creating a version tailored to this class and VHDL should be a trivial task. It may even be possible to add in some writing and coding exercises to ensure that students properly understand how programming works.

Using these resources, I was able to successfully convert every lab over to the Basys 3 using VHDL. Some documents were omitted, namely several documents from Xilinx since they were used to configure the security exceptions on the school's computers rather than actually doing any work with the Basys 3.

Research Methodology

The bulk of the research done for this thesis was done to determine new educational methods and ways to improve scores. The actual programming and conversion of the labs required little research and mostly involved teaching myself how to actually program the Basys 3 as well as teaching myself the art of VHDL programming. I have included an Appendix A and that summarizes what I have learned about the practical aspects of programming the Basys 3. Appendix C contains the labs that have been actually converted to use with the Basys 3.

The bulk of the actual research has gone into curriculum planning. While this was originally outside of the scope of the project, it became apparent that the structure of the class would need to be fundamentally altered. Currently the digital circuits class is based around lab work using breadboards, but changing over to the Basys will turn the class into a programming class with a hardware lab thrown in. To address these changes, new challenges will need to be predicted and addressed.

The first step is to ensure that the professor knows how to program VHDL. This should be an obvious first step but several readthroughs of Free Range VHDL and having the professor work their way through the labs themselves. Previous experience with programming languages may or may not be of any help. Converting the labs will also help the instructor to know what issues students may encounter while preforming the labs.

The first major change to the curriculum is to introduce pre-tests for programming. Currently there is little that can be done for the problem of the double curve. Changing curriculum seems to have little effect, neither does finding a better instructor or finding more enthusiastic students. At current the only real plan is early academic intervention and trying to identify which students may need help. The best way to do this is to test students either before class starts or on the first day. The purpose of these tests is to determine if students can create mental models (Dehnadi & Bornat, 2006). Making mental models is by far the most important skill in programming since once the algorithm for a program is made all the other issues are just semantics. A model of this test is included in Appendix D.

Once students are identified, they can be put into groups based on how they form models. According to current research, there will most likely be a double bell curve and the students in the lower group will need more help than others (roughly half the class). Making extra tutoring sessions mandatory may have a detrimental effect on performance, since it may cause students to feel stigmatized and less confident. A better method may be to allow students to retake tests to improve their grades. Then use part of the professor's office hours as an optional study session (or a grad student's time) to make sure that students can get help confidentially.

The new information I can contribute mainly lies in the area of translation between gates and actual programming. NAND universality functions in VHDL but it is very difficult to read. The code:

```
A <= B or C
```

Becomes

```
A <= (B NAND B) NAND (C NAND C)
```

This is readable here to some extent but rapidly becomes cumbersome when trying to write actual code. It also requires additional memory to write out the extra code thus making the code inefficient. While this method should still be taught, it should not be used as a standard.

Results

Labs

Labs have been included at the back of this paper in an appendix for easy printing.

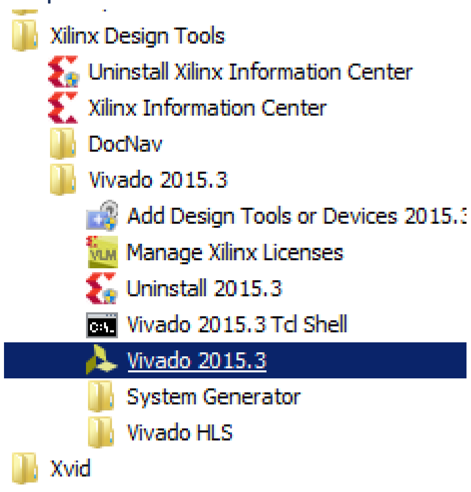
Conclusions

The switch from breadboards to Basys 3 will most likely be a massively popular one among students. It was much easier from a design perspective to essentially be able to go from circuit diagram to written logic and then into programming than it was to physically build a circuit. Feedback was also much easier as Vivado provides that for students rather than relying on human error and sight to try to find which wire does not connect properly. Having the compiler also means that programming issues can be more easily discovered and corrected without instructor intervention. I was also able to install Vivado onto my personal laptop and a number of other computers with ease, so in theory students can treat labs as homework and do anything that does not require the Basys 3 itself.

The switch to the Basys should also be popular among instructors. In addition to the advantages previously mentioned FPGA labs are much easier to organize. Instead of needing to direct students to several boxes of components, all that needs to be handed out are the boards and USB cords. The only real issue is to ensure that every student actually understands how to program the Basys and one student isn't just carrying the entire group.

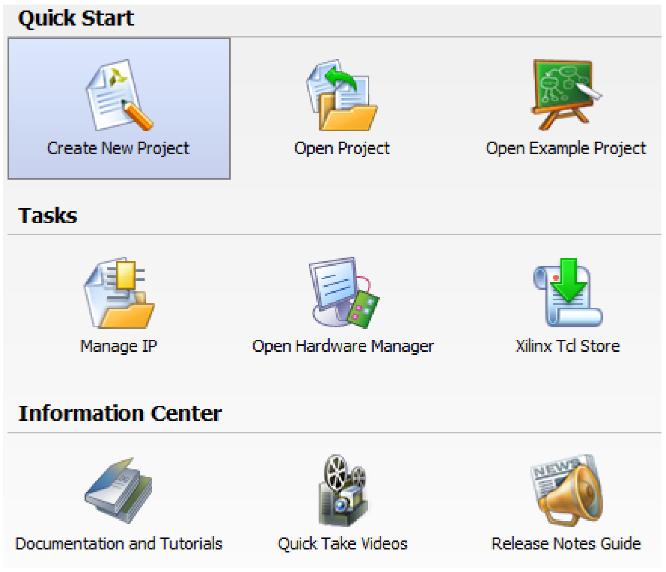
Appendix A: Setting Up the Basys 3[1]

Step 1:



Open the start menu then go to all programs> xilinux design tools > Vivado 2015.3 > Vivado 2015.3 (actual program).

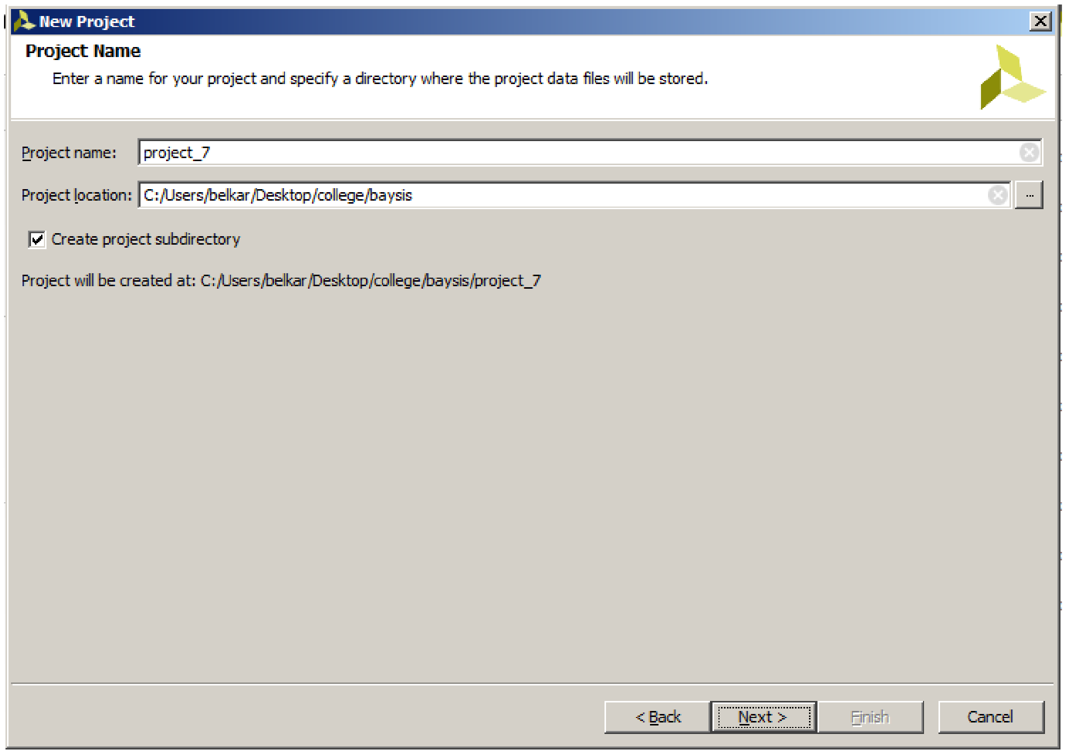
Step 2:



Click create new project

[1] This document is needed for every lab so students may print off and save it.

Step 3:



Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: project_7

Project location: C:/Users/belkar/Desktop/college/baysis

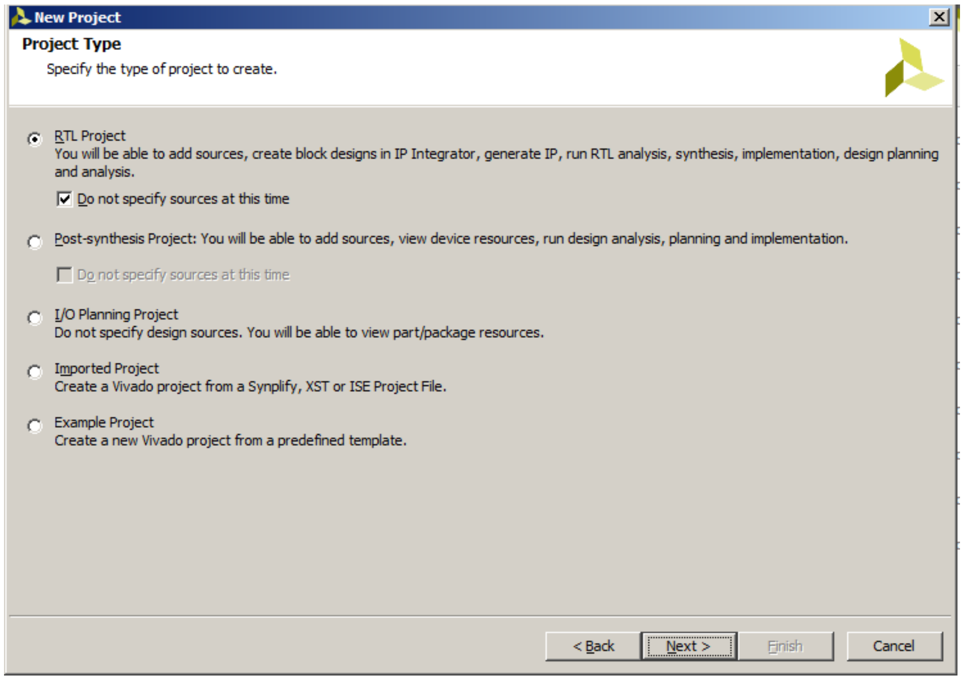
[✓] Create project subdirectory

Project will be created at: C:/Users/belkar/Desktop/college/baysis/project_7

Click next

Once you see this screen verify all information is correct and hit next.

Step 4:



Project Type

Specify the type of project to create.

● RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.

[✓] Do not specify sources at this time

○ Post-synthesis Project: You will be able to add sources, view device resources, run design analysis, planning and implementation.

☐ Do not specify sources at this time

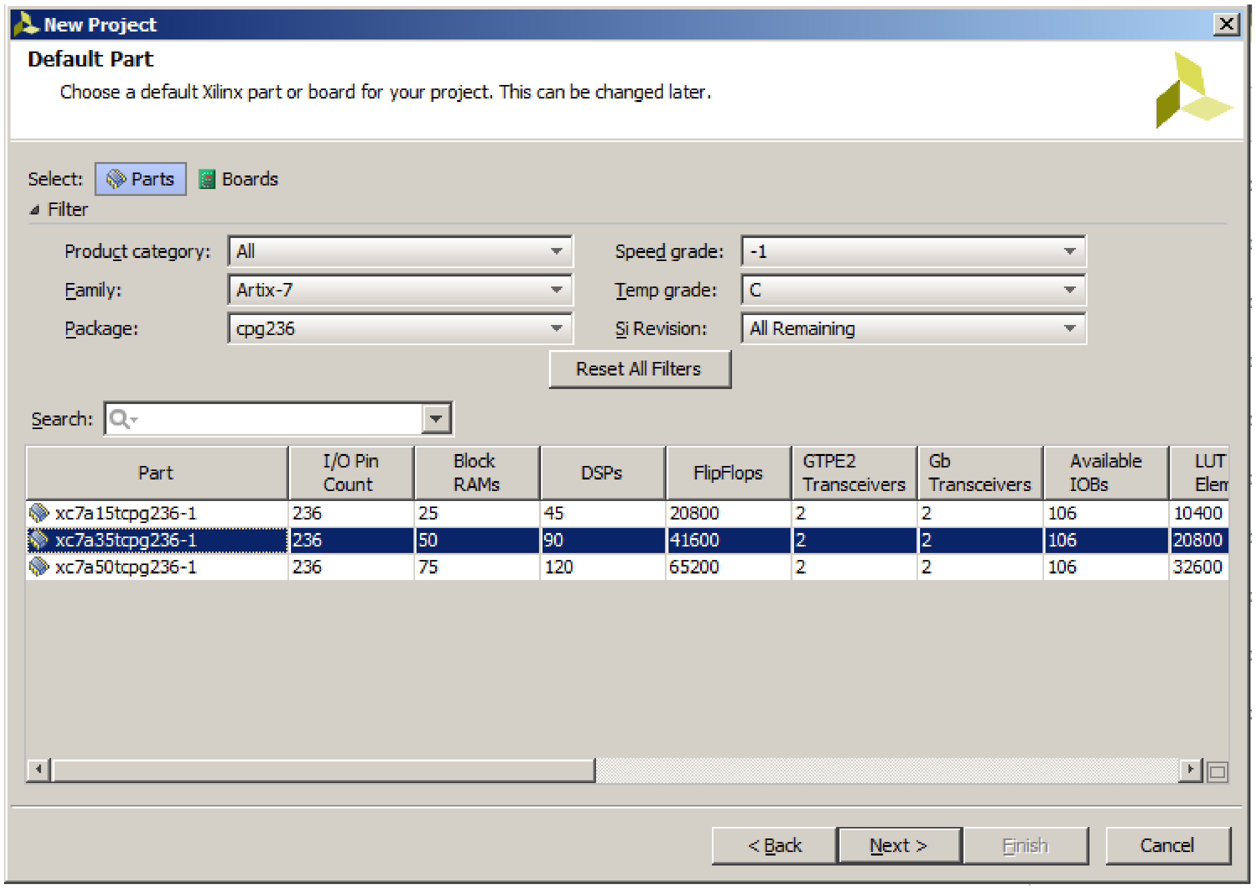
○ I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

○ Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

○ Example Project
Create a new Vivado project from a predefined template.

Choose RTL Project

Step 5:



New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Select: Parts Boards

Filter

Product category: All
Family: Artix-7
Package: cpg236
Speed grade: -1
Temp grade: C
Si Revision: All Remaining

Reset All Filters

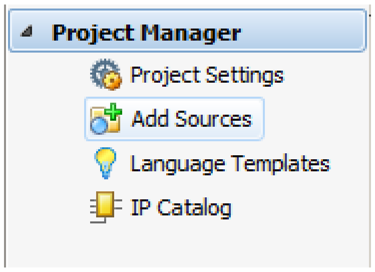
Search:

Part	I/O Pin Count	Block RAMs	DSPs	FlipFlops	GTPE2 Transceivers	Gb Transceivers	Available IOBs	LUT Elem
xc7a15tcpg236-1	236	25	45	20800	2	2	106	10400
xc7a35tcpg236-1	236	50	90	41600	2	2	106	20800
xc7a50tcpg236-1	236	75	120	65200	2	2	106	32600

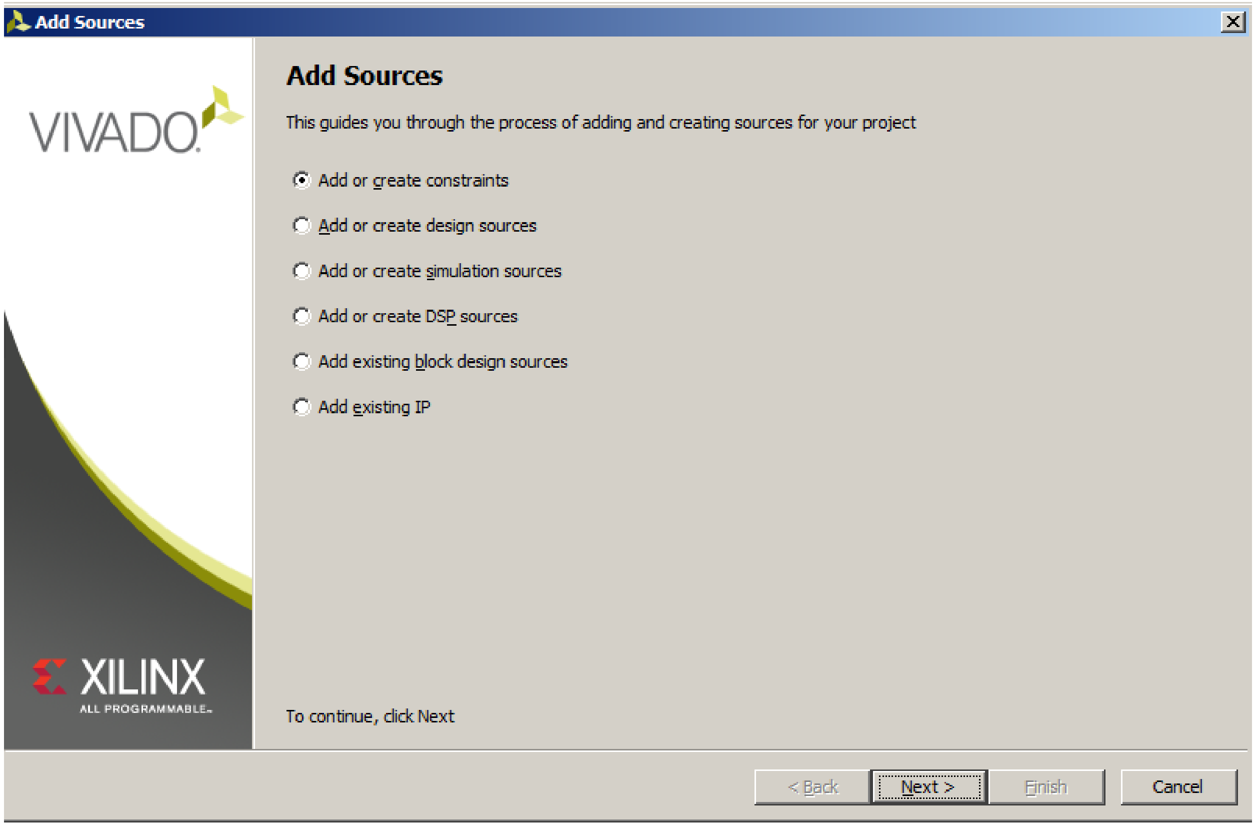
Make sure all settings on this page match with the ones for your project or it will not work. Pick xc7a35tcpg236-1, it is processor for the Basys 3.Then click next and then hit finish on the next screen. (Digilent, 2016)

Step 6:

You should now be on the main page for vivado



Click add sources


Add Sources

Add Sources

This guides you through the process of adding and creating sources for your project

- [✓ Add or create constraints]
- Add or create design sources
- Add or create simulation sources
- Add or create DSP sources
- Add existing block design sources
- Add existing IP

To continue, click Next

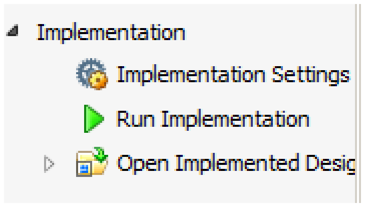
Then click add or create constraints. Then on the next page add your copy of the basys 3 master xdc file which can be found near the top of this webpage. It simply needs to be placed in the file for now. It will be modified depending on the project.

This information was taken from (Danowitz, 2015)

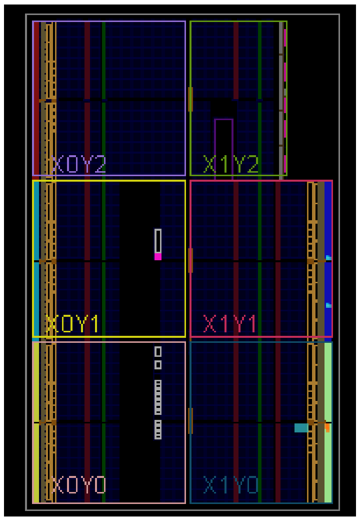
Finishing Labs in Vivado:

Step 1:

Once you have your code written go to implementation and click run implementation



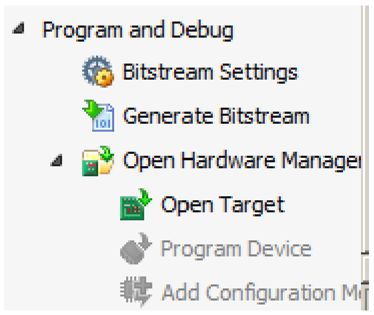
This runs everything else from higher up on the list and gives errors that will prevent the code being written. If the implementation was successful you should see an image similar to this appear in the code window



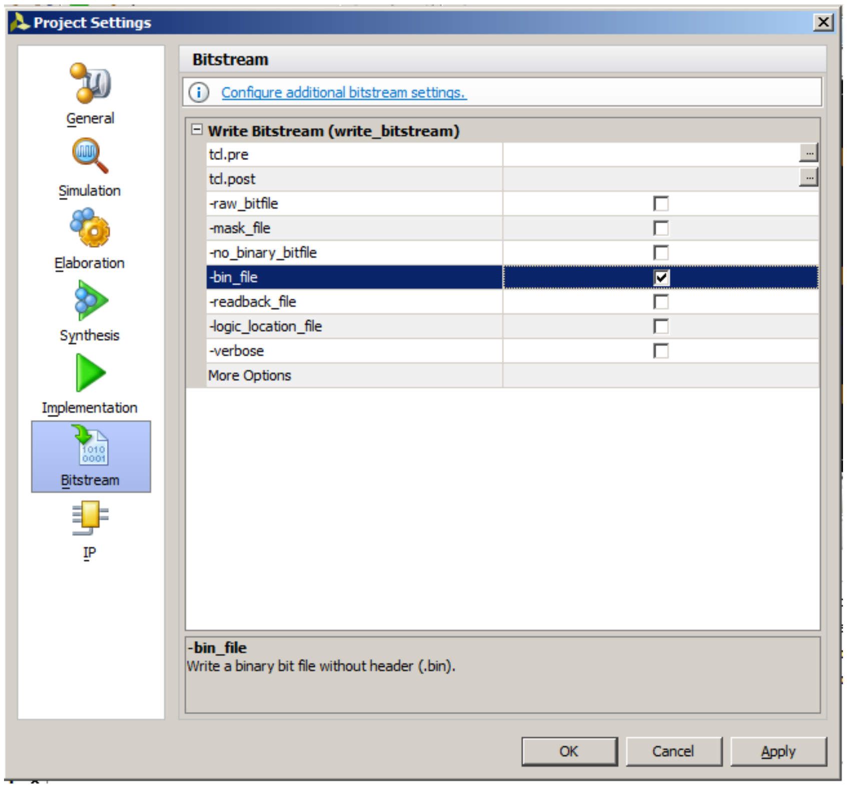
This is the actual design of the circuit within the board and is more useful for more complicated programs on the Basys 3.

Step 2:

Next go to the bottom section of the left-hand side of the screen



Step one is to click on Bit stream settings and check the bin file option.



Bitstream

Configure additional bitstream settings.

Write Bitstream (write_bitstream)

tcl.pre	
tcl.post	
-raw_bitfile	☐
-mask_file	☐
-no_binary_bitfile	☐
-bin_file	☑
-readback_file	☐
-logic_location_file	☐
-verbose	☐
More Options	

-bin_file
Write a binary bit file without header (.bin).

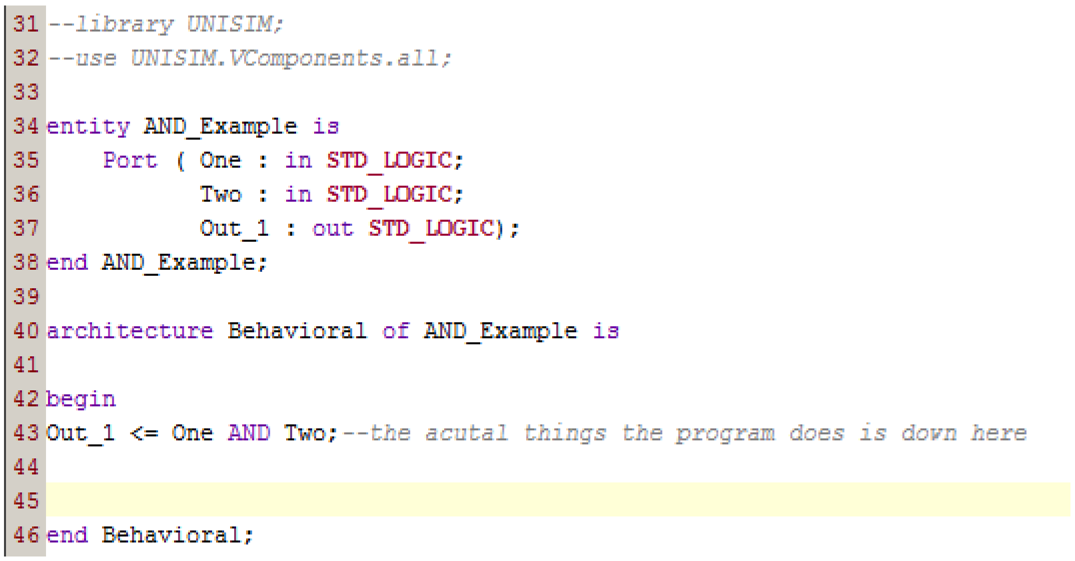
And make sure the –bin_file box is checked, this will ensure that the file writes faster. Then go back and click on open hardware target, if your Basys 3 is plugged into the computer it should be the only available target. Then click program device and wait for it to finish. Information from (Alex Wong)

Appendix B VHDL programming2

VHDL is a strange language to use. It appears to some people to be some kind of simplified version of the C used to program Arduino boards and other small development devices. This view is how I originally approached the project and it led to several months of frustration. VHDL is a hardware design language, which is unlike any other type of programming language. The first difference is that VHDL is a concurrent language. This means that unlike Java, which complies and then executes programs line by line, or Python, which just goes line by line, VHDL just does the whole program at once. This concurrence means that the loops most programming languages take for granted are often unusable or at the very least a kind of bottleneck for VHDL. There are ways to force VHDL to use certain aspects of other programming languages, but it is usually best to keep it simple and for most of the digital circuits class only basic commands will be used.

VHDL is first and foremost a hardware design language. forgetting this fact often leads to programs which do not work correctly and piles of mangled code. The first step in any program is to first design the circuit to be made. For example, a simple AND gate circuit; once the circuit is designed it needs to be broken down into its logic and a program started.

[2] This Appendix is an abridgement of several relevant parts of Free Range VHDL



```
31 --library UNISIM;
32 --use UNISIM.VComponents.all;
33
34 entity AND_Example is
35     Port ( One : in STD_LOGIC;
36            Two : in STD_LOGIC;
37            Out_1 : out STD_LOGIC);
38 end AND_Example;
39
40 architecture Behavioral of AND_Example is
41
42 begin
43 Out_1 <= One AND Two;--the acutal things the program does is down here
44
45
46 end Behavioral;
```

Figure 1Here is a simple example program that creates one and gate.

The first two statements on lines 31 and 32 are the end of some boilerplate that always starts at the beginning of every design file created by VHDL. It is a useful thing to fill out to keep track of your programs but in no way, impacts the program directly. It is always advisable to add comments too difficult to understand code with. Doing so will allow you to understand your own code. Beyond that are the entity and architecture statements. These two statements make up the actual directions for the board to follow. The entity declaration sets up the players in the program. It is linked to another file called the constraints which was covered in appendix A. The architecture statement tells the program what do with the things declared in the entity statement and may also hold its own type of entity called a variable.

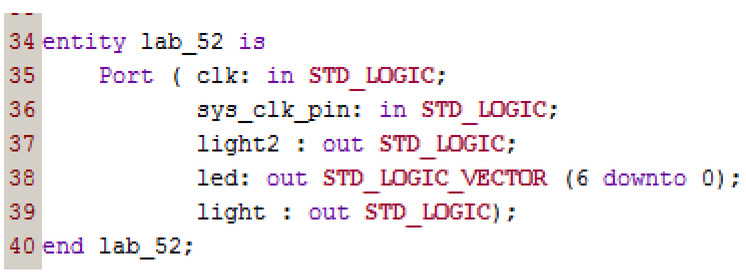
The next area of the example is called the entity declaration. Most of this code can be automatically generated when first creating the design file, but manually typing in entity names is also fine. Entities declared in this space can have 3 types, in, out or in/out. How the first two function should be obvious. In represents read states such as switches, and Out represents out states like LEDs. In/out can be both read from and written to, and due to that advantage, it seems logical that every data type should be one. I thought that at first, but there are many times where it causes logical errors to have output lights accidently read from and physical switches can't be written. Excessive use of in/out statements may also cause errors that cause your bitstream not to be written.

The next statement is the architecture statement. It is where the program actually happens. After the begin statement, the actual code that controls the program goes in. In the case of this program, it is just the one line:

Out_1 <= One AND Two

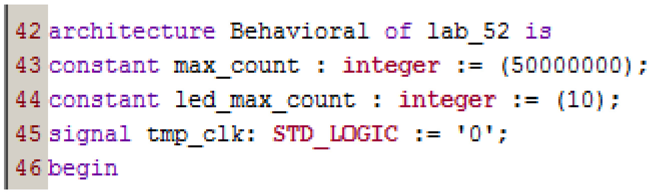
Which causes Out_1 to be written to based on an AND gate between one and two. The use of the word AND, and other logical gates are created the same way. The really important thing to look at is the symbol "<=". This symbol means that the port being pointed at "gets" whatever statement is on the other side. This is not an equals sign, which is actually := in VHDL, but can be thought of in a similar manner.

The above example is the essentially the "hello world" of VHDL programming. A more complex example sheds light on additional features of VHDL.



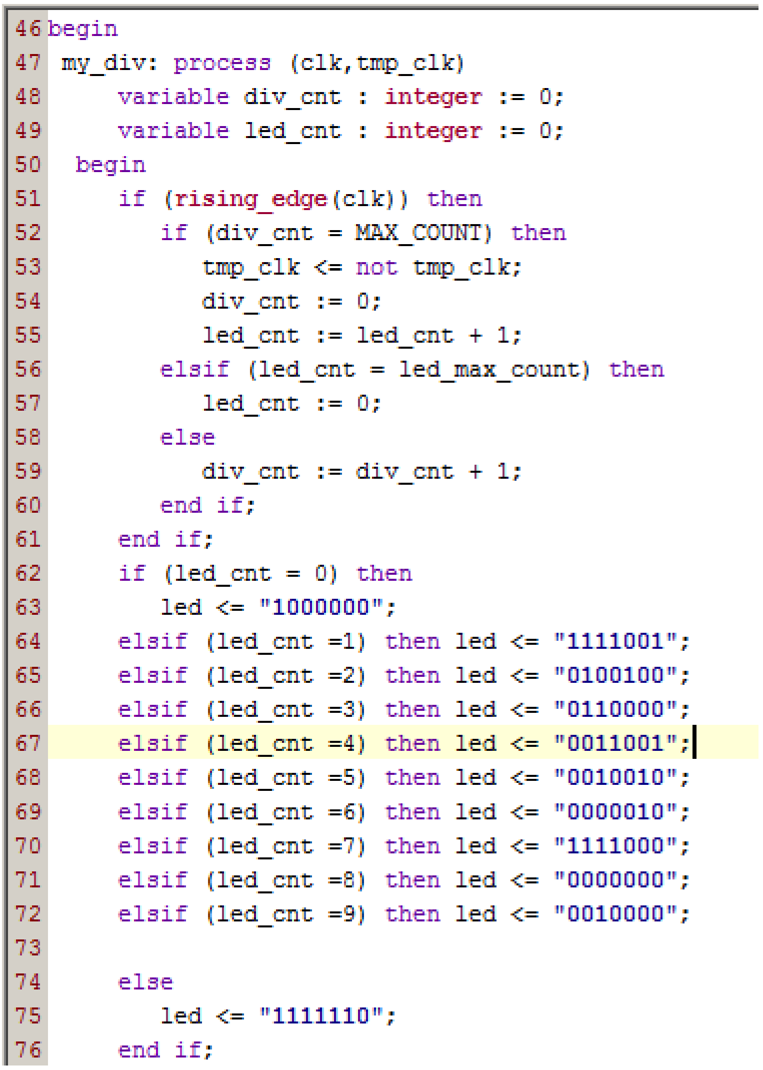
```
entity lab_52 is
    Port ( clk: in STD_LOGIC;
           sys_clk_pin: in STD_LOGIC;
           light2 : out STD_LOGIC;
           led: out STD_LOGIC_VECTOR (6 downto 0);
           light : out STD_LOGIC);
end lab_52;
```

The above entity declaration is similar to the one from the first example, with the exception of the led port. It has two notable difference between it and the other ports. First it does not use the STD_LOGIC_VECTOR type. Rather it uses STD_LOGIC and has the extra 6 downto 0 at the end. This makes led a special kind of port called a bundle which is basically a way to declare ports in bulk. Doing this is much faster that having to type out 7 lines of led port declarations and will save time later in the coding process. Remember that you can go either upto or downto but must either start or end with a zero.



```
architecture Behavioral of lab_52 is
constant max_count : integer := (50000000);
constant led_max_count : integer := (10);
signal tmp_clk: STD_LOGIC := '0';
begin
```

There are also ways to declare new items within the architecture statement. Below the beginning of the architecture statement and before the begin are variables. Variables are like ports that do not connect to the board. They exist as abstract programming and mostly exist to make more complex programs possible. The first keyword describes what kind of variable is to be made. The first two are constants and cannot be changed from within the program. The final is a signal that is the equivalent to a Boolean data type in most other programming languages. There are other data types but they are of limited use in this lab.



```
begin
 my_div: process (clk,tmp_clk)
     variable div_cnt : integer := 0;
     variable led_cnt : integer := 0;
  begin
     if (rising_edge(clk)) then
        if (div_cnt = MAX_COUNT) then
           tmp_clk <= not tmp_clk;
           div_cnt := 0;
           led_cnt := led_cnt + 1;
        elsif (led_cnt = led_max_count) then
           led_cnt := 0;
        else
           div_cnt := div_cnt + 1;
        end if;
     end if;
     if (led_cnt = 0) then
        led <= "1000000";
     elsif (led_cnt =1) then led <= "1111001";
     elsif (led_cnt =2) then led <= "0100100";
     elsif (led_cnt =3) then led <= "0110000";
     elsif (led_cnt =4) then led <= "0011001";
     elsif (led_cnt =5) then led <= "0010010";
     elsif (led_cnt =6) then led <= "0000010";
     elsif (led_cnt =7) then led <= "1111000";
     elsif (led_cnt =8) then led <= "0000000";
     elsif (led_cnt =9) then led <= "0010000";

     else
        led <= "1111110";
     end if;
```

[3]The architecture statement for a program is even more complex. The first statement is my_div:process. The div process is one of the more interesting ideas in VHDL it basically puts a smaller program that is closer to a normal programming language like C into the program you are currently writing. This may seem like an odd concept but it allows us to write non-concurrent areas within VHDL. The two ports within the brackets at the end of the process statement are what is allowed to be used within the process statement. Inside process statements variables can actually be declared before the start of the process statement's own begin statement. Then like the first example, the program actually begins.

Process statements allow for the use of if statements. If statements allow branching logic within the process statement so that one thing can happen depending on the results of a Boolean expression rather than just the state of one port. For statements and other mainstays

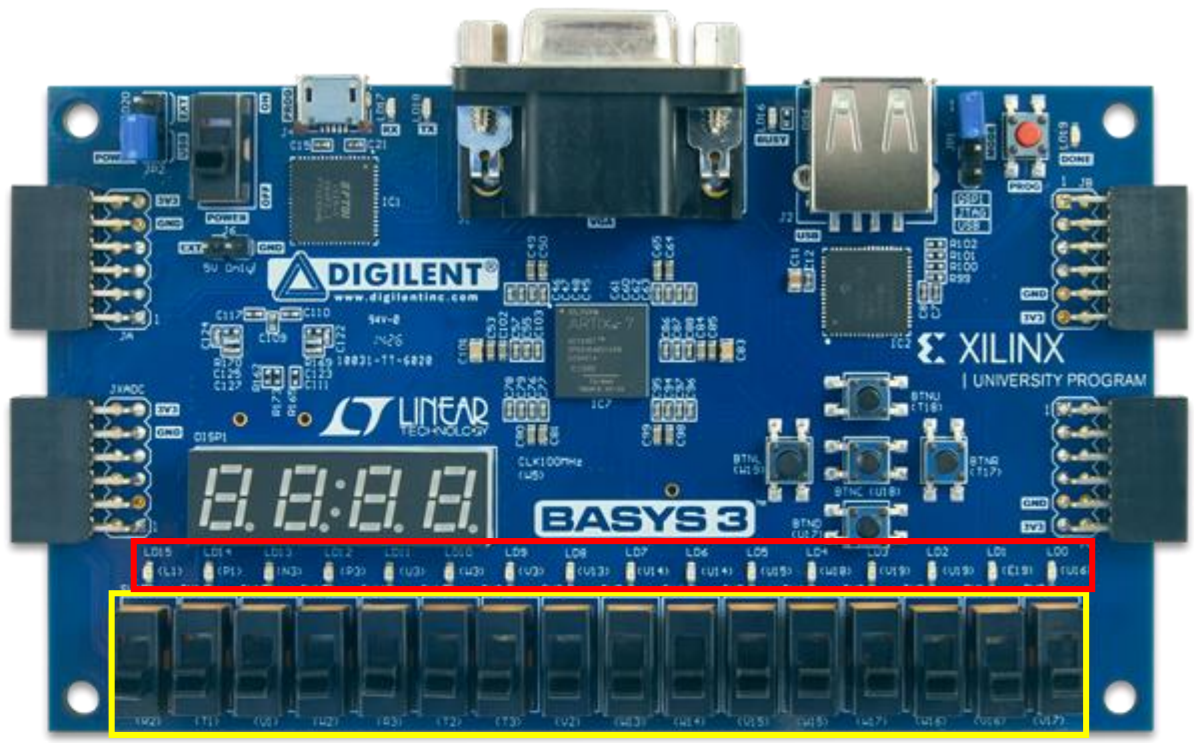
[3] (Alex Wong)

of regular programming languages are also useable within a process statement, with the exception of the while statement which is depreciated for use in the Basys 3.

Appendix C: Labs

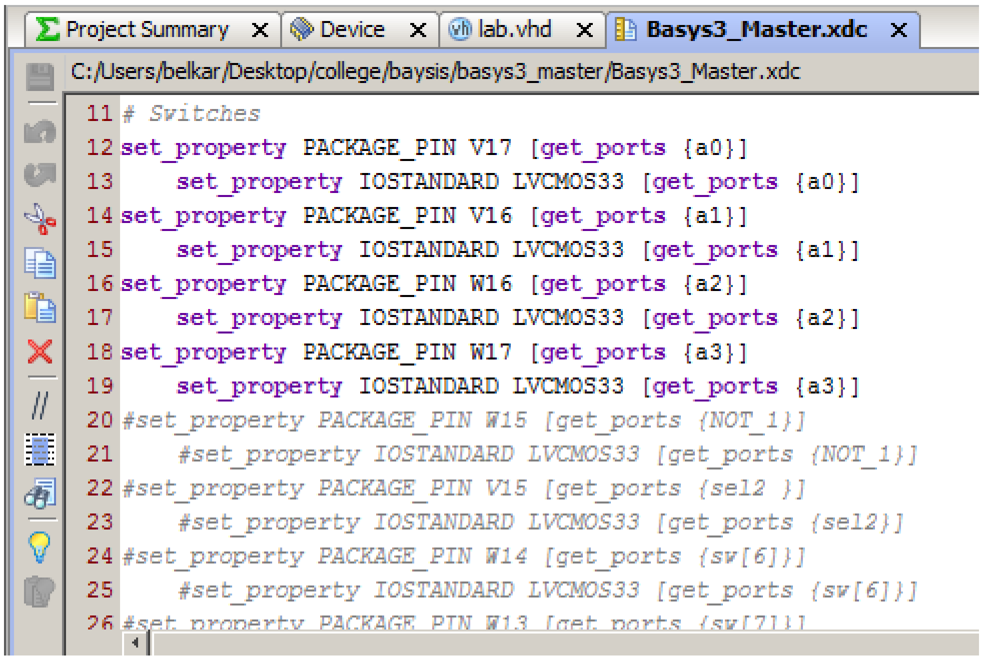
Lab 1: Logic Gates

In this lab, we will learn the most basic building block of VHDL Programs, logic gates



Note the switches outlined in yellow and the LED's outlined in red. These will act as the inputs and outputs for the logic gates in this lab. Image taken from (Digilent, 2016)

After following the steps outlined in the *Getting Started Guide*, you will need to actually set the constraints file to be used with the Basys Lab. The best way to do this is to make a copy of the master XDC for each Lab since altering the XDC in a latter lab may cause an older lab to be unusable. It is also unadvisable to just uncomment everything in the XDC file and just use the regular names as it will cause a number of warnings to show up during synthesis. It is also advisable to rename your signals for both readability and because by default they are written for use in a bundle and not renaming them may give errors.


Project Summary
Device
lab.vhd
Basys3_Master.xdc
C:/Users/belkar/Desktop/college/baysis/basys3_master/Basys3_Master.xdc

```
# Switches
set_property PACKAGE_PIN V17 [get_ports {a0}]
    set_property IOSTANDARD LVCMOS33 [get_ports {a0}]
set_property PACKAGE_PIN V16 [get_ports {a1}]
    set_property IOSTANDARD LVCMOS33 [get_ports {a1}]
set_property PACKAGE_PIN W16 [get_ports {a2}]
    set_property IOSTANDARD LVCMOS33 [get_ports {a2}]
set_property PACKAGE_PIN W17 [get_ports {a3}]
    set_property IOSTANDARD LVCMOS33 [get_ports {a3}]
#set_property PACKAGE_PIN W15 [get_ports {NOT_1}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {NOT_1}]
#set_property PACKAGE_PIN V15 [get_ports {sel2 }]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sel2}]
#set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
#set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
```

Figure 2Image from (Digilent 2016)

To use a signal in your circuit, just remove the # from in front of the set_property and rename the switch. For this lab, you will need to uncomment two switches and one LED for each gate you make. You will be asked to make these outputs and inputs when making the simulation file, the names you put in there can be changed at any point, so do not worry if the names are different; it just causes an error.

Making the gates

To actually make the gates is a relatively simple process as nearly every gate follows the same syntax

LED <= Input_1 (gate type) Input_2;

(Note those are not the actual names)

For this lab you will need to program each of these gates onto one program

AND

OR

NOT (only needs one input)

NAND

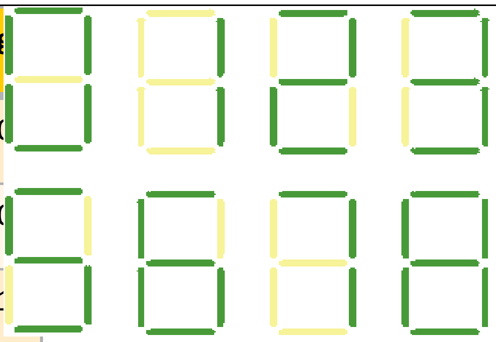
NOR

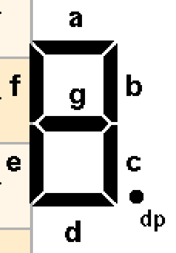
Lab 2: Karnugh Maps

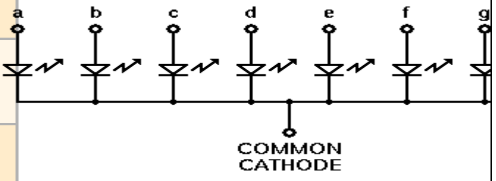
In this lab we will copy the function of a karnugh map onto the Basys to simulate the B segment of a 7 segment display going through its function as it goes from one to nine.

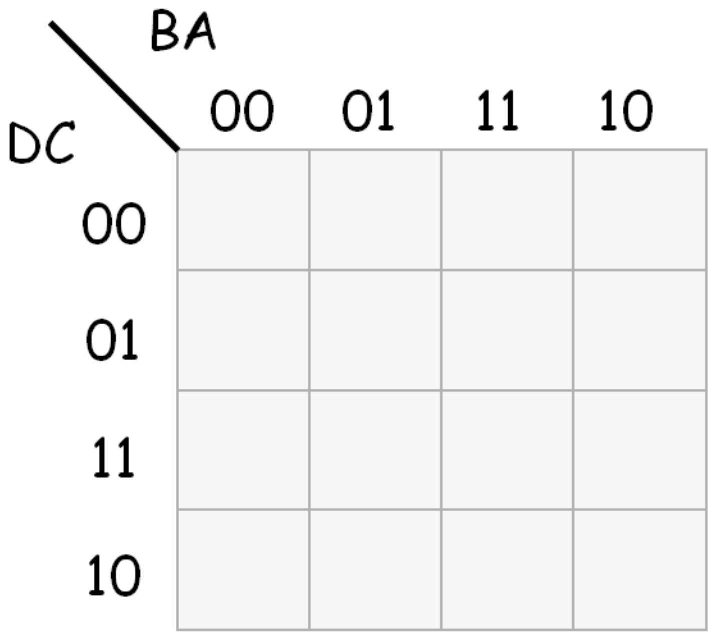
First off fill out the karnugh map for segment B using the information below

D	C	B	A		a	b	c	d	e	f	g
0	0	0	0		1	1	1	1	1	1	0
0	0	0	1		0	1	1	0	0	0	0
0	0	1	0		1	1	0	1	1	0	1
0	0	1	1		1	1	1	1	0	0	1
0	1	0	0		0	1	1	0	0	1	1
0	1	0	1		1	0	1	1	0	1	1
0	1	1	0		1	0	1	1	1	1	1
0	1	1	1		1	1	1	0	0	0	0
1	0	0	0		1	1	1	1	1	1	1
1	0	0	1		1	1	1	1	0	1	1
1	0	1	0		X	X	X	X	X	X	X
1	0	1	1		X	X	X	X	X	X	X
1	1	0	0		X	X	X	X	X	X	X
1	1	0	1		X	X	X	X	X	X	X
1	1	1	0		X	X	X	X	X	X	X
1	1	1	1		X	X	X	X	X	X	X









To fill this out just put down a 1 on squares where the light would be on and zero when it is off. Then circle the areas where there are "islands" of ones. Circle them and remember that the Karnugh map represents a torus (in common terms a doughnut) so circles can wrap around the map. Once everything has been properly circled see what changes between areas of a circle, these changes are those that are unimportant. Once you have found the unimportant changes you should have the full logic for the individual segment. This ABCD can be drawn into a circuit and used for the next step in the lab.

Next you will need to set up the XDC file for this project, you will need an input for each letter and one output for the B segment. Use the logic found in the karnugh map

HINT: Add a not around the entire equation for B to get it going. The common cathode causes the circuit to go low when high.

Instructors note: Here is what I found for this lab

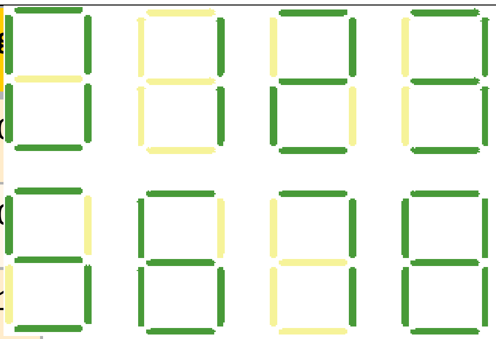
b <= NOT((NOT(a2))or(NOT(a0) AND NOT(a1)) or (a2 and a1));

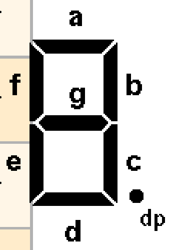
there are most likely many other solutions but this is an easy one to use

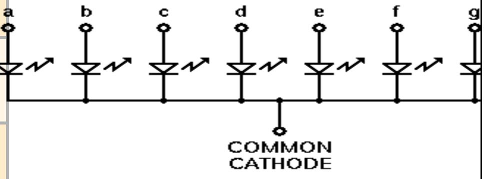
Lab 3: NAND Gates

This lab is essentially the same as Lab 2 but with NAND gates and using the C segment of the display. First off you must once again fill out the Karnugh Map and find the logic for segment C

D	C	B	A		a	b	c	d	e	f	g
0	0	0	0		1	1	1	1	1	1	0
0	0	0	1		0	1	1	0	0	0	0
0	0	1	0		1	1	0	1	1	0	1
0	0	1	1		1	1	1	1	0	0	1
0	1	0	0		0	1	1	0	0	1	1
0	1	0	1		1	0	1	1	0	1	1
0	1	1	0		1	0	1	1	1	1	1
0	1	1	1		1	1	1	0	0	0	0
1	0	0	0		1	1	1	1	1	1	1
1	0	0	1		1	1	1	1	0	1	1
1	0	1	0		X	X	X	X	X	X	X
1	0	1	1		X	X	X	X	X	X	X
1	1	0	0		X	X	X	X	X	X	X
1	1	0	1		X	X	X	X	X	X	X
1	1	1	0		X	X	X	X	X	X	X
1	1	1	1		X	X	X	X	X	X	X

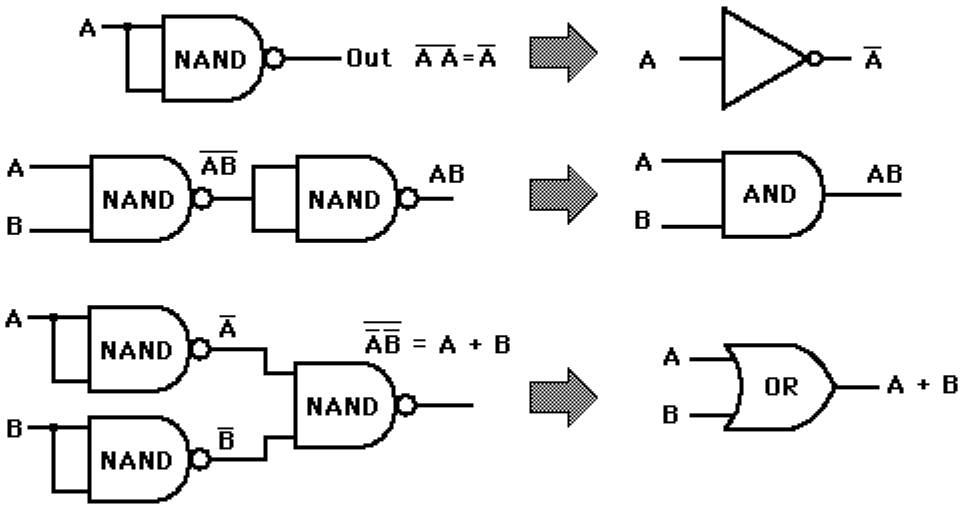






DC \ BA	00	01	11	10
00				
01				
11				
10				

Once this has been found you must convert from regular logic to NAND logic. This is actually a simple task once the relations between regular gates and NAND gates are realized



Converting the equation for C will give you a NAND gate equation which can be programmed into the Basys 3.

Instructors Note:

b <=not(a2 NAND ((a1 NAND a0) NAND ((a0 NAND '1') NAND (a1 NAND '1'))));

This works but using other methods may be used.

Lab 5: BCD Down Counter

This will be our most difficult lab and require learning several new methods for use on the Basys 3. Making a down counter requires the ability to count, an ability our previous circuits have lacked. Making a clock for the Basys 3 is not a straightforward process since it only has its system clock which moves much too fast to be very readable for humans. To do this we must divide the clock into smaller segments, or divide up the running time into segments which can then be used to turn things on and off.

But for timing to exist the program will need to function step by step and not concurrently. To do this VHDL includes the process statement.

```
my_div: process (clk,tmp_clk)

begin

*statements go here*

End process my_div;
```

The process works by essentially making a special area where VHDL acts like a normal programming language. Statements within it are executed one by one rather than all at once like in normal VHDL. This means that if statements can be used. If statements in VHDL work like if statements in any other programming language. They allow the program to do different things based on an input.

```
if (*conditions go here) then

       *statements go here*

     elsif (*conditions go here*) then

       *statements go here*

     else

       *statements go here*

     end if;
```

In the above statement conditions need to be put in the parentheses and which code is to be executed put after the then. The elsif statements (a shortening of else if) is another way to add a condition and there can be an infinite number of elsif statements. The else statement always has to end the statement and should act as an error catcher, if something goes wrong, have the else display something recognizable to aid in troubleshooting.

But the amount to which you divide the clock also has to be stored somewhere. All that we have used in the class so far are called signals; they can hold Boolean values but nothing else and even those can usually only hold one value unless a bundle is declared. But there are two other methods variables, which exist in process statements, and constants, which can exist in the program proper.

Constants are declared in the architecture statement of either the program itself or within a process statement, and for the purpose of our program the length of the duty cycle in milliseconds will be stored in a constant. This will be stored in a flip flop which switches when the count for the led hits the maximum. You can also use this to keep track of which number should be displayed on the Basys 3 by making a more complex if else statement.

One way to do this lab is to make the following

Architecture

2 variables

Process statement

Flip flop to change numbers and keep track of numbers

If statements to display numbers

HINT: It may be faster to name the LEDs as a bundle and pass values to them using a single quote around the binary numbers. There is also a function called rising_edge(*put clock name here*) that can be used to find the rising edge of a clock pulse for if statements.

Appendix D: Sample Mental Model Test

1. A = 1
   B = 0
   C <= A or B
   What will be the value of B

2. If (A = ‘1’){

   B <= A;

   Elsif (b = ‘1’)

   A <= B;

   Else

   C <= 1;

   A. What will A, B and C equal if the staring inputs are A=1 B=0 and C=0

   B. What will A, B and C equal if the staring inputs are A=0 B=0 and C=0

   C. What will A, B and C equal if the staring inputs are A=1 B=1 and C=1

3. Port (LBtn,CBtn,RBtn,TBtn,BBtn : in std_logic;
   CLK : in std_logic;
   SW : in std_logic_vector(15 downto 0);
   disp_en : out std_logic_vector(3 downto 0);
   segments : out std_logic_vector(7 downto 0);
   speakerout : out std_logic);
   end MainModule;

   A. Why does the first line contain 4 ports rather than one like the others?

   B. What is the purpose of the downto statements?

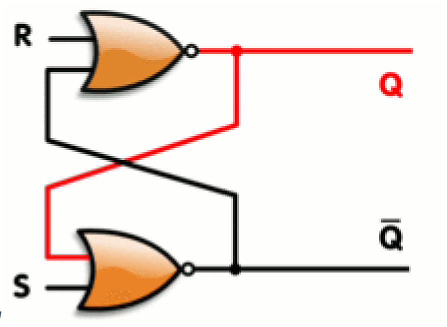


Figure 3Image Courtesy of Wikipedia

Assuming both of the triangular objects output high only If one input is high what happens when

A. R is put high

B. S is put High

C. R and S are put high

Works Cited

Alex Wong. (n.d.). How to use Verilog and Basys 3 to do 3 bit binary counter. Instructables.

Danowitz, A. (2015, July 9). vivado and basys3 getting started. Youtube.

Dehnadi, S., & Bornat, R. (2006, Febuary 22). The camel has two humps. Middlesex, United Kingdom: Middlesex University School of Computing.

Digilent. (2016, April 8). *Basys 3™ FPGA Board Reference Manual.* Retrieved from reference.digilentinc.com: https://reference.digilentinc.com/_media/basys3:basys3_rm.pdf

Hoffbeck, J. P. (2014). Using Practical Examples in Teaching Digital Logic. *Engineering Faculty Publications and* (p. 21). Portland: Shiley School of Engineering.

IEEE. (2002). *IEEE standard VHDL language reference manual.* New York, NY: Institute of Electrical and Electronics Engineers.

(n.d.).SR Latch. *Wikipedia.* Wikipedia.

Tappero, F. &. (2013). *Free Range VHDL.* Free Range Factory .

Wang, G. (2009). Bridging the gap between textbook and real applications: A teaching methodology in digital electronics education. *Computer Applications in Engineering Education*.