Remote Access to FPGA

Vincent Giannone

Draft Number 4

11/27/2013

CNT 4104 Software Project in Computer Networks

Instructor: Dr. Janusz Zalewski

Computer Science & Software Engineering Programs

Florida Gulf Coast University

Ft. Myers, FL 33965

1. Introduction

The purpose of this project is to allow a user to remotely upload and execute designs on a Field Programmable Gate Array (FPGA) [A1]. There have been two similar attempts in previous iterations of this course, Robert Porter and Olexiy Kovtunenko, *Remote FPGA Web Lab User* [1] and *Web Based FPGA Lab* [2]. These reports form the background for the current project. This introduction describes the nature of FPGA, which may be unfamiliar to the reader. Section 2 defines the problem, Section 3 describes its solution and Section 4 outlines the implementation and experiments.

1.1 FPGA Circuitry

A Field Programmable Gate Array, abbreviated FPGA, is a field programmable Integrated Circuit (IC). The term “programmable” is analogous to “configurable” in the sense that an FPGA board or chip can realize many different functions after being manufactured. FPGAs are used for their inexpensive implementation of user defined logic networks in hardware. Figure 1 models a primitive, black box view of how an FPGA realizes a function by the connection of a host processor, via a USB cable, to an FPGA that executes the user defined logic.

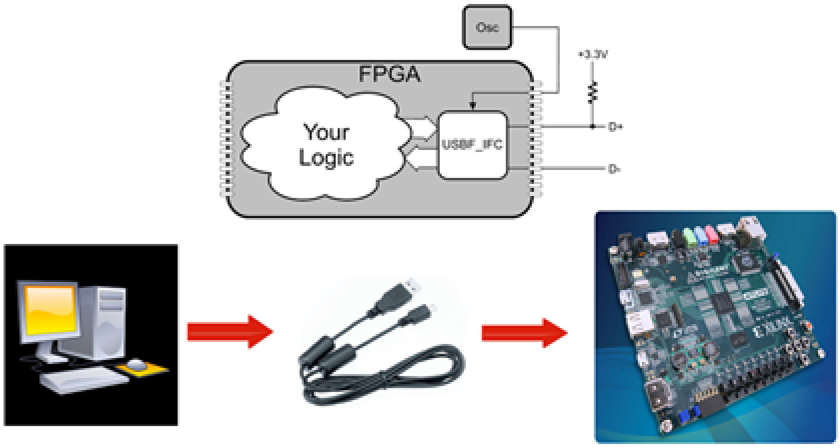


Figure 1. Diagram of intended FPGA usage of user defined logic.

Various brands of FPGA devices, dependent on the company who designs and manufacturer's them, implement their products uniquely, while the architecture of the devices are still fundamentally the same. These manufacturers are referred to as FPGA families. Some of the more well known families of FPGAs include Xilinx, Actel and Altera. The only difference between each of these companies, in terms of an FPGAs architecture, is the physical means for implementing programmability, the interconnection arrangement and the basic functionality of the logic blocks. The basic architecture of an FPGA includes Programmable Logic Devices (PLDs), Logic Gates such as XOR, and NAND gates, Random Access Memory (RAM) and other devices such as a Device Clock Manager (DCM).

The implementation of FPGA architecture is shown in Figure 2. The data are received as input from an Input/Output block (IOB). These data are then operated upon by a function executed by an array of Configurable Logic Blocks (CLBs), which are connected by configurable interconnections, wiring in most cases, between the CLBs, forming a logic block matrix. Once the data have been operated upon, they are sent as output to another I/O block. The red lines in Figure 2 connect data to the CLBs. After the CLBs perform the configured operations, a result is sent to an output IOB.

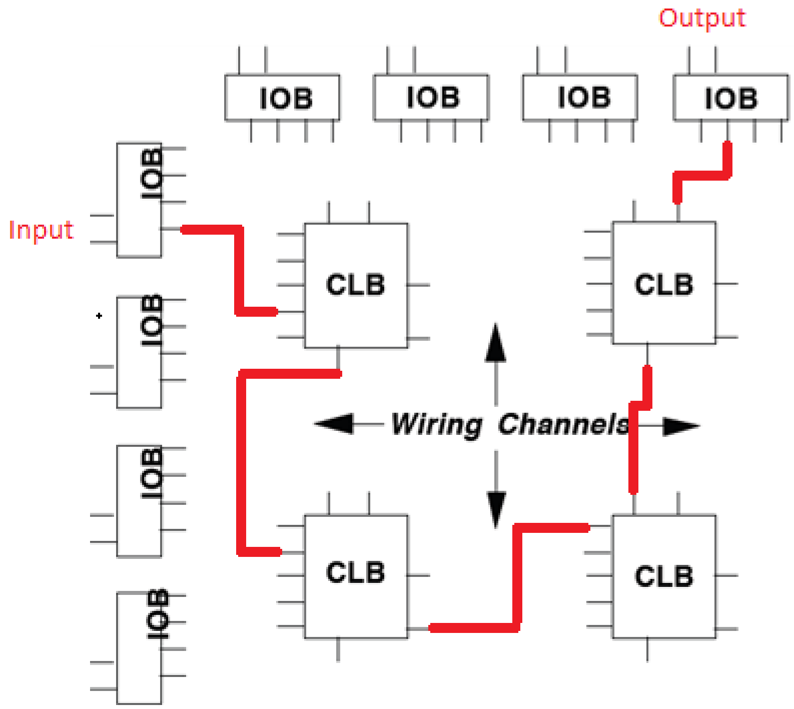


Figure 2. A simple illustration of a IOB, CLB, and programmable interconnect arrangement.

Shown as a single cell architecture, Figure 3 illustrates a simplified version of a CLB as a logic diagram. The values of a,b, and c are termed as the input data, presumably from an IOB, which are sent to a 3-input look-up table. The value d is a control data input and clk is a digital clock signal input.

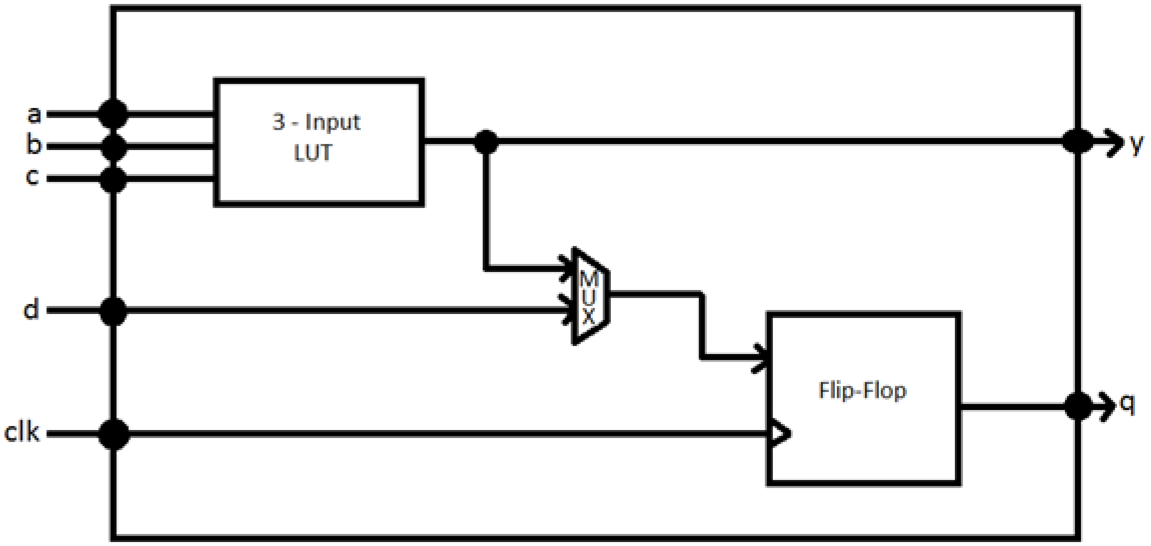


Figure 3. A simple Configurable Logic Block diagram

All CLBs, minimally, contains several elements that encapsulate, on average, a flip-flop (FF), multiplexer (MUX) and a Look-Up Table (LUT); all of which are illustrated in Figure 3. This is not to say that a CLB may not contain other elements, though. Some CLBs may contain adders and other pre-programmed cores. As for the Xilinx FPGA architecture, their CLBs contain two slices. Figure 4 illustrates the concept of a slice.

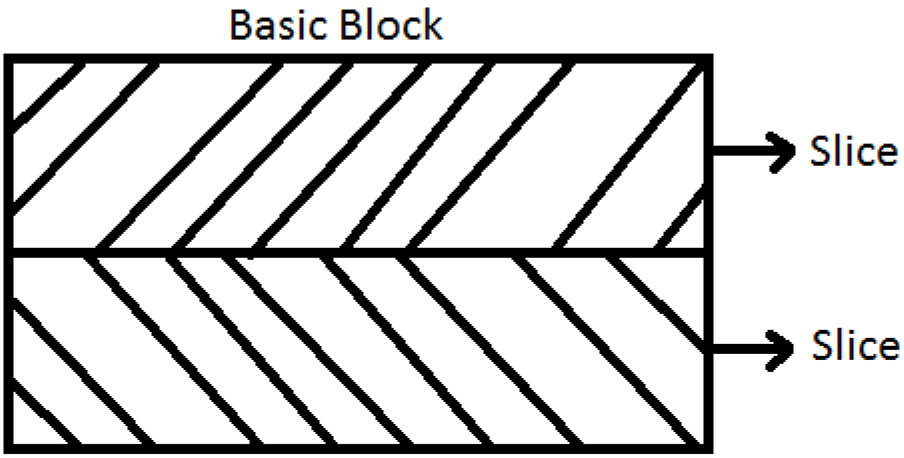


Figure 4. Basic CLB Block with two slices

Each slice contains two 4-input function generators, also called a LUT, carry logic, and two storage elements, illustrated in Figure 5. Each function generator output drives the CLB output and d input of the Flip-Flop.

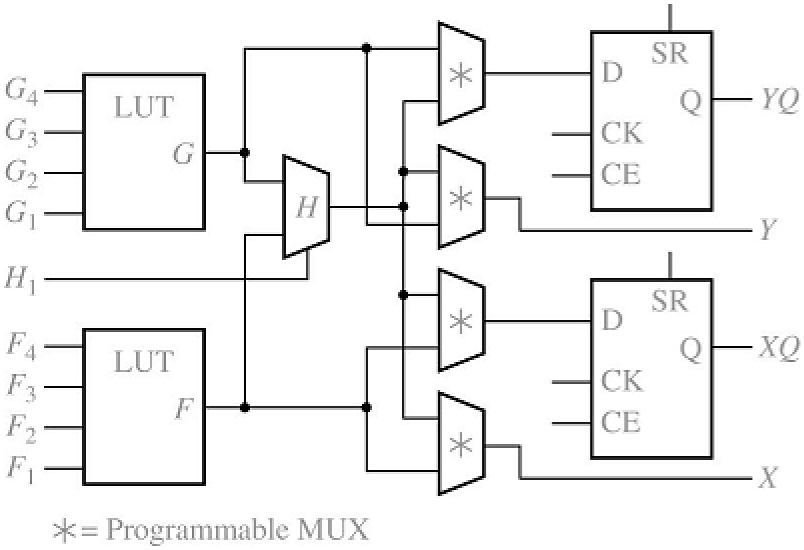
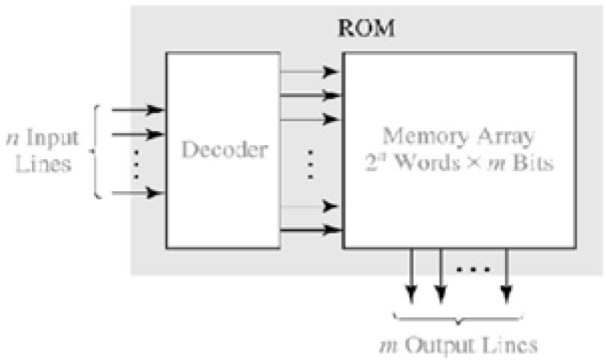
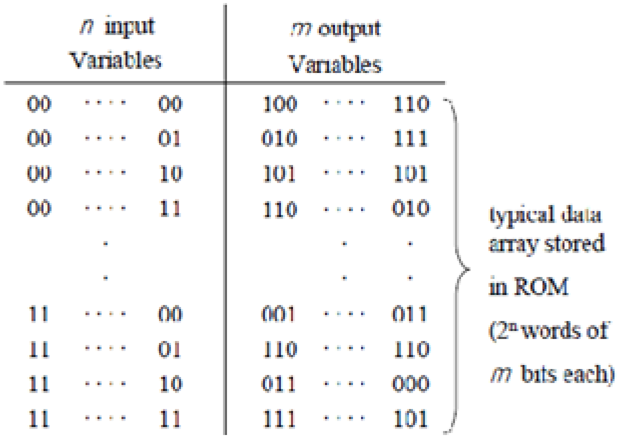


Figure 5. Two slice construct in a Xilinx CLB

A Look-up Table, abbreviated LUT, is a type of storage that reads bitwise input values, logically equivalent to a Boolean function, comparing them to a programmed truth table. The LUT acts as a function generator, retrieving an output for any set of inputs, as operated upon by the stored function. If the Boolean function was not stored using a configurable LUT, then it is possible that an output won't generate the same output when a design is configured. Internally, a LUT is similar to a Read-Only Memory (ROM) device that is Programmable (PROM). Figure 6 and Figure 7 illustrate the similarities of the internal logic for a ROM device and a LUT, respectively. A ROM device configures its contents once, at the time it's manufactured, and a LUT configures its contents each time a new design is loaded to an FPGA, executed by a CLB.



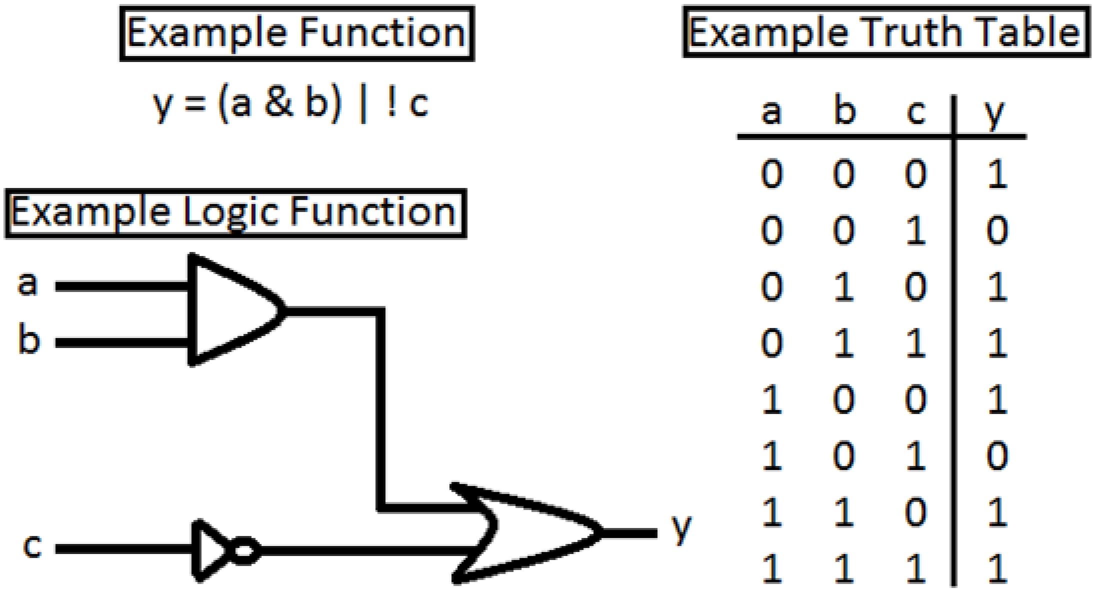


n input Variables	m output Variables
00 ···· 00	100 ···· 110
00 ···· 01	010 ···· 111
00 ···· 10	101 ···· 101
00 ···· 11	110 ···· 010
.	. .
.	. .
11 ···· 00	001 ···· 011
11 ···· 01	110 ···· 110
11 ···· 10	011 ···· 000
11 ···· 11	111 ···· 101

typical data array stored in ROM (2^n words of m bits each)

Figure 6(a). ROM Block Diagram **Figure 6(b). ROM Truth Table**

Figure 6(a) shows a block diagram for a ROM storage device. Since the inputs may vary for any applied ROM, the diagram uses n inputs for m outputs. Figure 6(b) is the corresponding truth table for an n input to m output ROM. As an example of how a LUT operates, Figure 7 illustrates a function that must be realized in a CLB. The truth tables output values y, dependent on the inputs a, b, and c is what defines the function a CLB realizes.



a	b	c	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Figure 7. A Boolean Function, Gate Diagram and Truth Table for a realized function on an FPGA.

These values of y are programmed into the LUT, using Static Random Access Memory (SRAM) cells, that can store either a 0 or a 1. Figure 8 illustrates this construct, in a matrix fashion, as a SRAM connects various values, in order to manipulate the data accordingly.

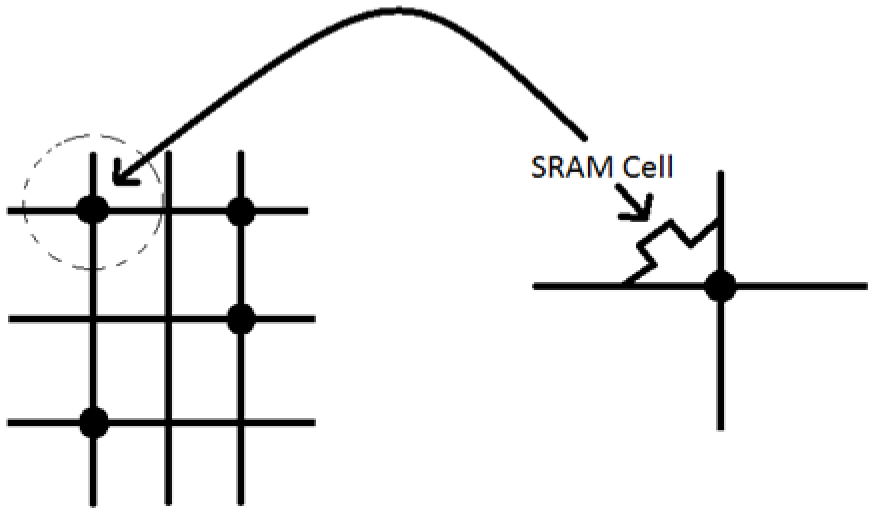


Figure 8. Depiction of an SRAM Cell

Figure 8 also illustrates the use of a Cross-Point connection. A Cross-Point connection is where the two lines overlap, making a connection if the SRAM cell has an input of 1 and breaking the connection if the SRAM cell has an input of 0. These connections are performed by pass transistors, transmission gates, and MUXs.

Once the function is realized by the LUT, using the SRAM cells in Figure 8, its operating values act as an input to a MUX, which then selects an output value of the, based on the inputs. Figure 9 illustrates this MUX operation as a diagram that represents the example function in Figure 7.

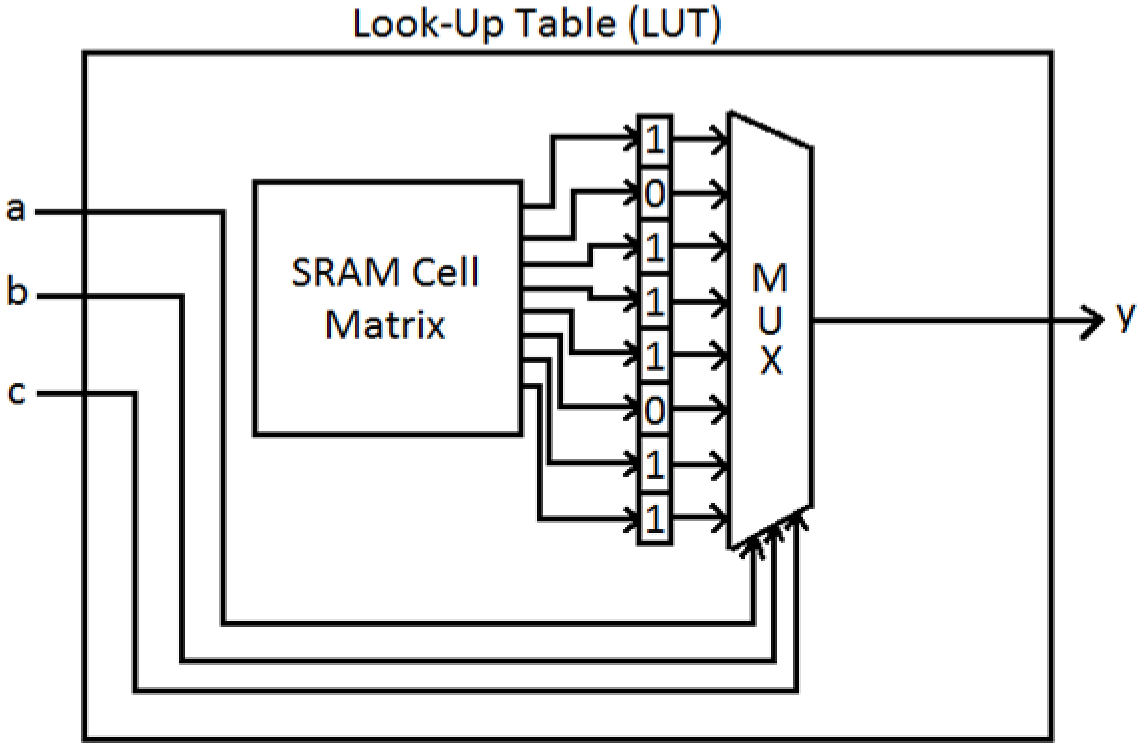
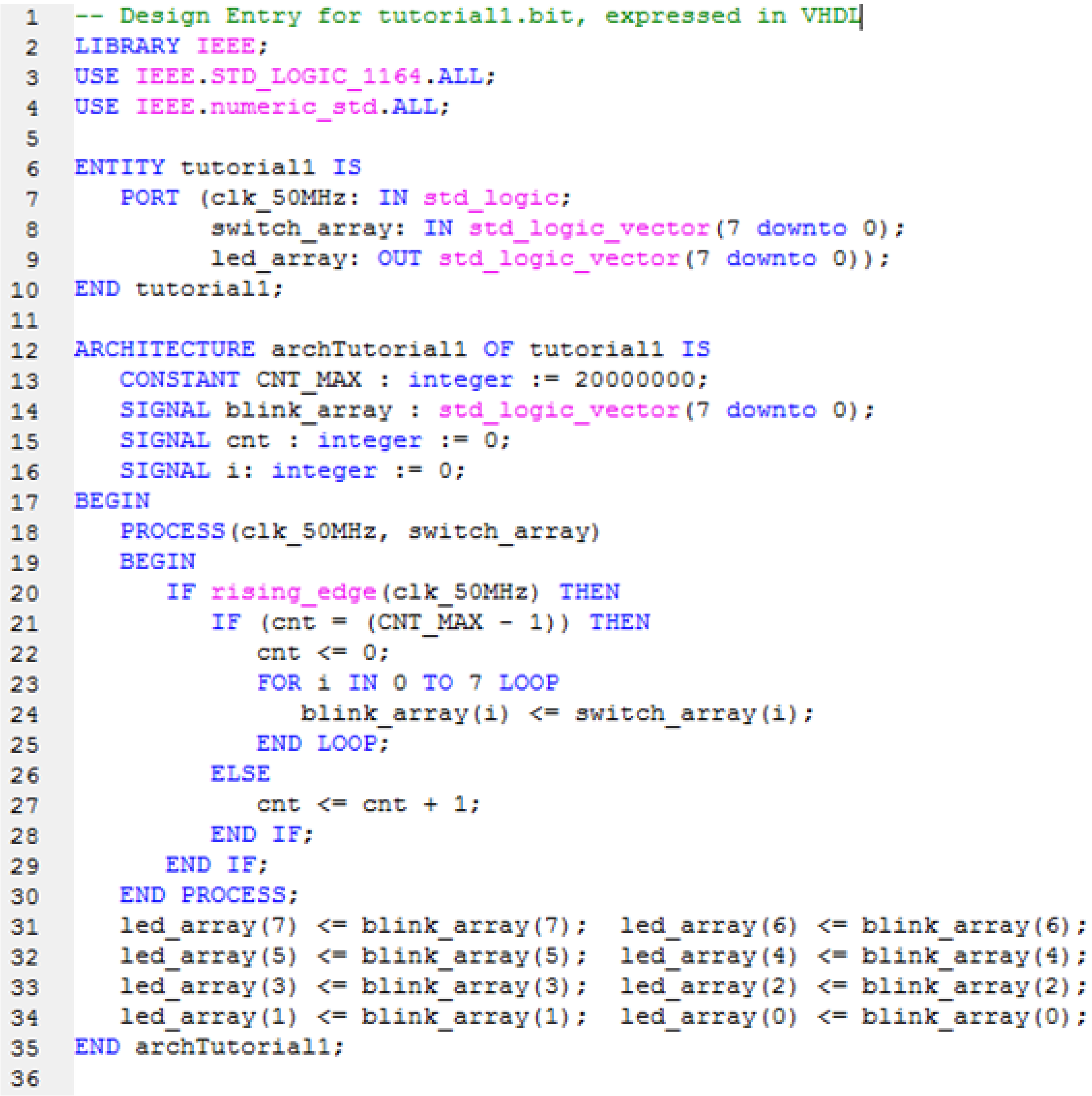


Figure 9. A LUT Diagram, depicting a MUX taking in a function

The programmable nature of a CLB, where its function is defined by the LUT, is implemented by three methods. These methods include the previously discussed SRAM based cells, as well as an implementation of the Anti-Fuse Technology, Erasable Programmable Read Only Memory (EPROM) Technology, and the Electrically Erasable Programmable Read Only Memory (EEPROM) technology. Each method exists due to the different implementation of an FPGAs architecture, dependent on the manufacturer. For example, Xilinx and Altera use the SRAM based technology, where as Actel and Quicklogic use Anti-Fuse Technology. SRAM based and Anti-Fuse programming are the most commonly implemented methods of programmability for an FPGA. These methods are also transparent to a the programmable interconnections of an FPGA.

1.2 Programming FPGAs

When programming an FPGA a user must submit a design entry. The design entry can be represented as either VHDL code or as a Register Transfer Level (RTL) schematic. There is editing software included in Xilinx ISE, and other FPGA IDEs, where one may create a source file to represent their design at either the top-level or bottom-level. This report will cover the Xilinx tools and the series of steps which lead to the programming of an FPGA. Figure 10 illustrates a Hardware Design Language (HDL) design entry, written in VHDL, for an example program called tutorial1.



```vhdl
-- Design Entry for tutorial1.bit, expressed in VHDL
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY tutorial1 IS
   PORT (clk_50MHz: IN std_logic;
         switch_array: IN std_logic_vector(7 downto 0);
         led_array: OUT std_logic_vector(7 downto 0));
END tutorial1;

ARCHITECTURE archTutorial1 OF tutorial1 IS
   CONSTANT CNT_MAX : integer := 20000000;
   SIGNAL blink_array : std_logic_vector(7 downto 0);
   SIGNAL cnt : integer := 0;
   SIGNAL i: integer := 0;
BEGIN
   PROCESS(clk_50MHz, switch_array)
   BEGIN
      IF rising_edge(clk_50MHz) THEN
         IF (cnt = (CNT_MAX - 1)) THEN
            cnt <= 0;
            FOR i IN 0 TO 7 LOOP
               blink_array(i) <= switch_array(i);
            END LOOP;
         ELSE
            cnt <= cnt + 1;
         END IF;
      END IF;
   END PROCESS;
   led_array(7) <= blink_array(7);  led_array(6) <= blink_array(6);
   led_array(5) <= blink_array(5);  led_array(4) <= blink_array(4);
   led_array(3) <= blink_array(3);  led_array(2) <= blink_array(2);
   led_array(1) <= blink_array(1);  led_array(0) <= blink_array(0);
END archTutorial1;
```

Figure 10. Design Entry expressed in VHDL for tutorial1

Similar to other High Level Languages (HLLs) such as Java and C++, VHDL embraces such logical constructs as the IF-ELSE statement and FOR loop. These logical constructs manipulate the data retrieved from the IOBs accordingly, in order to output the correct data. The file extension that uniquely identifies it as VHDL format is tutorial1.vhd, shown in Figure 11.

tutorial1.vhd

Figure 11. tutorial1.vhd File Extension Example

Another type of Design Entry source file that will satisfy tutorial1 is a schematic. A schematic is a visual representation of the user defined function that must be realized on the FPGA. Schematics may be used to express a design at both the top-level and the bottom-level.

When using a schematic as a design entry, its format which is in the file with the tutorial1.sch extension shown in Figure 12, is eventually converted to an HDL structured netlist. This is done in order to satisfy the synthesis process.

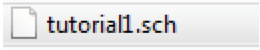

tutorial1.sch

Figure 12. tutorial1.sch File Extension Example

After a user has edited and confirmed that there are no syntax errors, the Design entry must undergo synthesis. Synthesis is a process that begins with the input of an error free design entry file. The synthesis process takes the contents of this file, structurally formatted as an HDL, and creates a bottom-level logic abstraction equivalent, called a netlist file, using a library containing primitives. These primitives are considered to be the simplest design elements available in a library and may include Small Scale Integration (SSI) constructs such as AND gates, OR gates, latches and flip-flops. But as design entries become much more complicated with the constant increase in size of digital systems and what is needed for them to function, these generic primitives have begun to also include Medium Scale Integration (MSI) constructs such as shift registers and full-adders.

When using the Xilinx Synthesis Technology (XST) Tool, a design entry is synthesized to a netlist file just the same, but is written with Xilinx Primitives, where this series of steps is illustrated in Figure 13. These primitives that are available in the Xilinx libraries theoretically function the same as any other AND gates, OR gates, latches and flip-flops, but can only be read by Xilinx FPGA device that you are targeting. A netlist file synthesized using a third-party tool will not be recognized by a Xilinx FPGA.

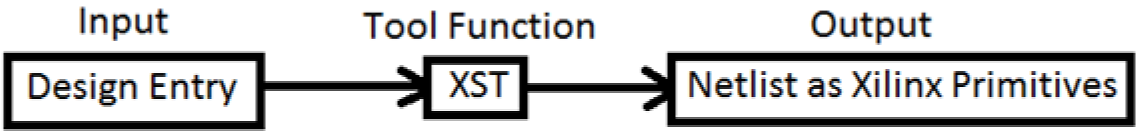


Figure 13. Simple XST Operation Output Diagram

The netlist file generated as output by the XST Tool has a format extension .ngc, and is shown in Figure 14 in regards to the program tutorial1.

Figure 14. tutorial1.ngc File Format Extension Example

Since a netlist file is unreadable in its format by the GUI, one may view their synthesized design as a pre-optimized Register Transfer Level (RTL) schematic, which is also an output of XST Tool.

Not to be confused with its design entry counterpart, though they may look the same, an RTL schematic derives from the netlist file, where as the design entry schematic comes directly from the user and has a different extension. It is considered pre-optimized since it is not written using Xilinx Primitives. Figure 15 graphically illustrates the pre-optimized RTL schematic in regards to the program tutoria1, synthesized by the XST Tool.

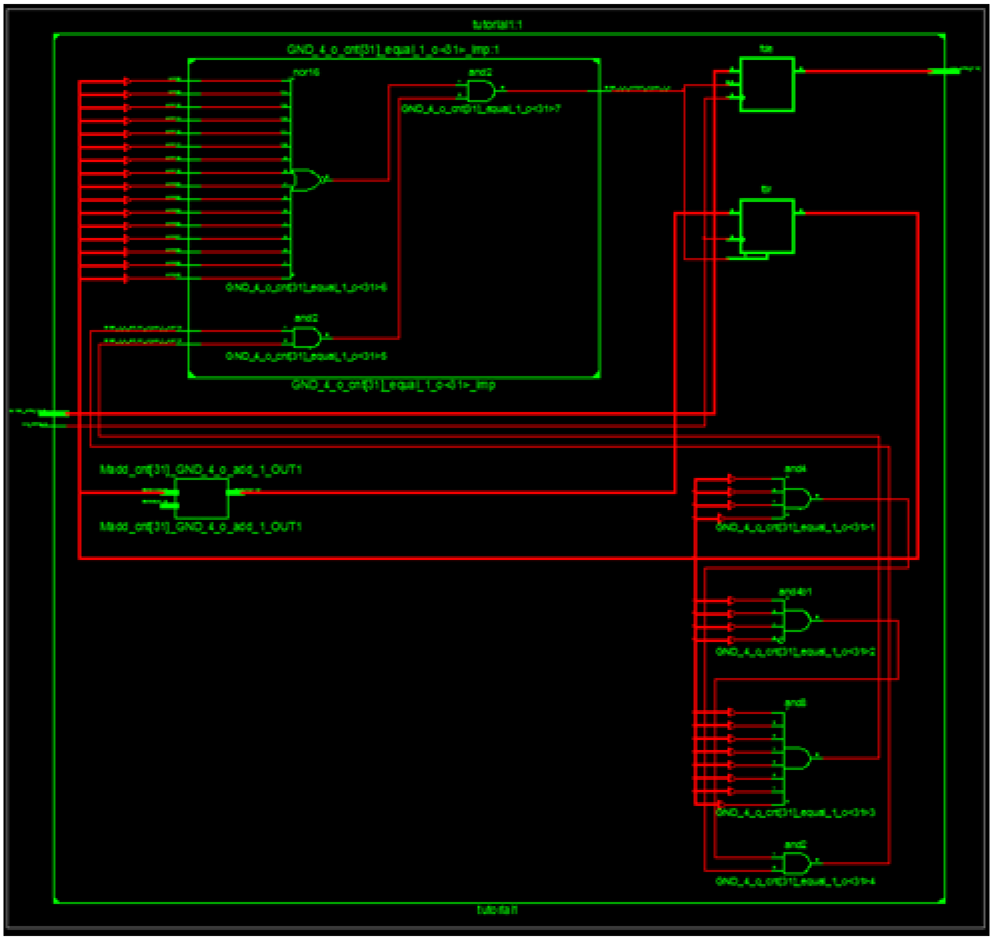

tutorial1:1
GND_4_o_cnt[31]_equal_1_o<31>_imp:1
nor16
and2
GND_4_o_cnt[31]_equal_1_o<31>7
GND_4_o_cnt[31]_equal_1_o<31>6
and2
GND_4_o_cnt[31]_equal_1_o<31>5
GND_4_o_cnt[31]_equal_1_o<31>_imp
Madd_cnt[31]_GND_4_o_add_1_OUT1
Madd_cnt[31]_GND_4_o_add_1_OUT1
and4
GND_4_o_cnt[31]_equal_1_o<31>1
and4b1
GND_4_o_cnt[31]_equal_1_o<31>2
and8
GND_4_o_cnt[31]_equal_1_o<31>3
and2
GND_4_o_cnt[31]_equal_1_o<31>4
tutorial1

Figure 15. Graphical pre-optimized RTL Schematic of tutorial1

Figure 16 shows the file format extension .ngr for the pre-optimized RTL Schematic file in regards to the program tutorial1, making it unique and only readable to Xilinx FPGA Families of devices.

tutorial1.ngr

Figure 16. tutorial1.ngr File Format Extension Example

After the design entry has been synthesized, the next step is translation. Translation is a process that converts a netlist file and constraints file, written in non-native primitives, to a format that is readable to the targeted Xilinx FPGA device. The Native Generic Database (NGD) Build Tool reads all of the input design netlists and constraints files and writes the results into a single merged file, using the Xilinx primitives included in their libraries, describing the logic and design constrains of the function being realized.

In the case that one may want to target a Xilinx FPGA Device if the XST Tool was used to convert the design entry to a netlist file the NGD Tool would only have to merge the netlist files to a single file, since the XST Tool automatically writes it's netlists using Xilinx primitives. This translation process is shown in Figure 17.

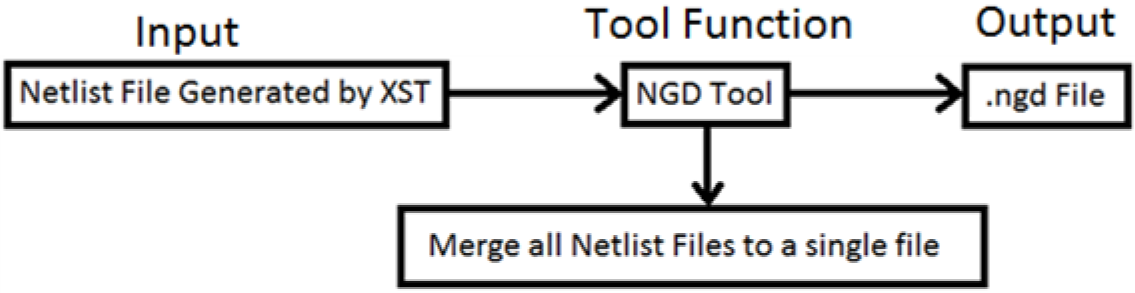

Input
Tool Function
Output
Netlist File Generated by XST
NGD Tool
.ngd File
Merge all Netlist Files to a single file

Figure 17. Translation process for netlist files generated using XST

If the design entry files were synthesized to a netlist file using a third-party synthesis tool or is expected to translate a direct schematic file, the NGD Tool would first have to convert convert the netlists into Xilinx primitives and then merge the files into a single file. Figure 18 contrasts that of Figure 17, showing what the NGD tool must do to make the .ngd File for the reasons mentioned above and in the figure.

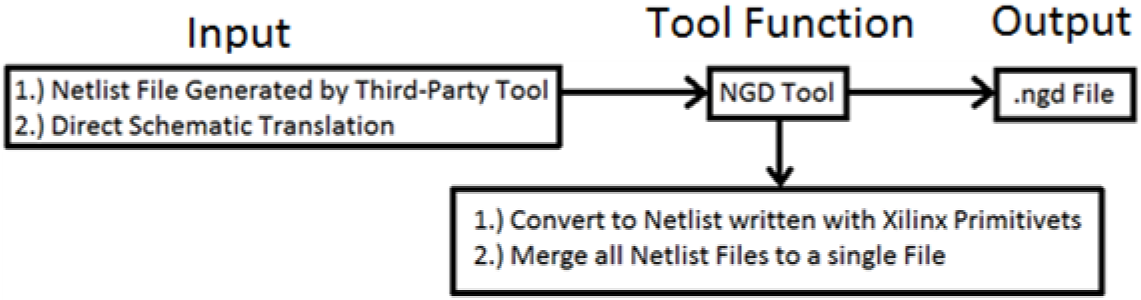

Input
Tool Function
Output
1.) Netlist File Generated by Third-Party Tool
2.) Direct Schematic Translation
NGD Tool
.ngd File
1.) Convert to Netlist written with Xilinx Primitivets
2.) Merge all Netlist Files to a single File

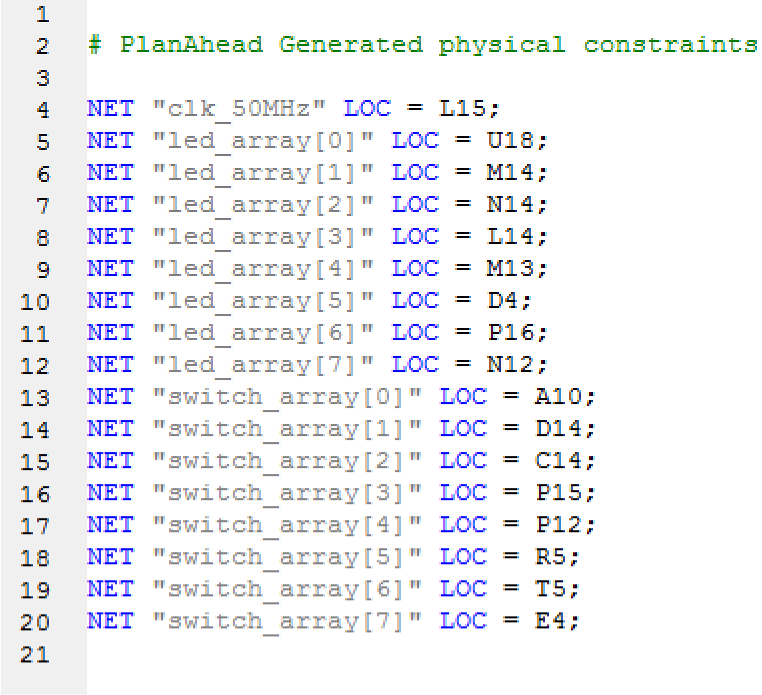
Figure 18. Translation process for netlist files not written in Xilinx Primitives

The final output of the NGD Tool, as Figure 17 and Figure 18 illustrate, is a file formatted with the extension .ngd, shown in Figure 19.

tutorial1.ngd

Figure 19. tutorial1.ngd File Extension Example

The constraints file that is also included as input to the NGD Tool is a file that defines the IOB pin locations for a design. Figure 20 shows an example of this file.



```

# PlanAhead Generated physical constraints

NET "clk_50MHz" LOC = L15;
NET "led_array[0]" LOC = U18;
NET "led_array[1]" LOC = M14;
NET "led_array[2]" LOC = N14;
NET "led_array[3]" LOC = L14;
NET "led_array[4]" LOC = M13;
NET "led_array[5]" LOC = D4;
NET "led_array[6]" LOC = P16;
NET "led_array[7]" LOC = N12;
NET "switch_array[0]" LOC = A10;
NET "switch_array[1]" LOC = D14;
NET "switch_array[2]" LOC = C14;
NET "switch_array[3]" LOC = P15;
NET "switch_array[4]" LOC = P12;
NET "switch_array[5]" LOC = R5;
NET "switch_array[6]" LOC = T5;
NET "switch_array[7]" LOC = E4;

```

Figure 20. User Constraints File for tutorial1.ucf

In this case, the user did not write this file himself for the tutorial1 program, but was generated by the PlanAhead Tool. Each line assigns a pin location on the FPGA for each port that receives the data for the IOB. These pin locations are unique to the Xilinx Spartan-6 FPGA device architecture.

Figure 21 illustrates the file format extension for this file, which is .ucf.

tutorial1.ucf

Figure 21. tutorial1.ucf File Extension Example

After the translation process comes mapping. Mapping is the process of defining the input and output connectives of a logic function, described by a translated netlist and constraints file, to the physical elements of the system a FPGA is connected to.

When discussing the Xilinx ISE, after the .ngd file is made available by the NGD Tool, the MAP Tool groups the logical elements which define the function into CLBs and IOBs.

The concept of partitioning is fundamental to the Mapping process in the sense that it describes the amount of CLBs and IOBs to be allocated during the placement process, but now these groupings are no longer generic.

The MAP Tool, and the files it generates, are device dependent due to the Translate process and NGD Tool. The Xilinx primitives that the .ngd file is written with make it so that no other FPGA architecture can implement the function it describes, at least properly. Since the NGD Tool incorporates an exception handler and realizes errors, if pin locations entered into the constraints file were not available on the FPGA, the translated netlist file, .ngd, would not be generated.

Figure 22 illustrates how the components of a system are mapped to the unique pin location on the FPGA.

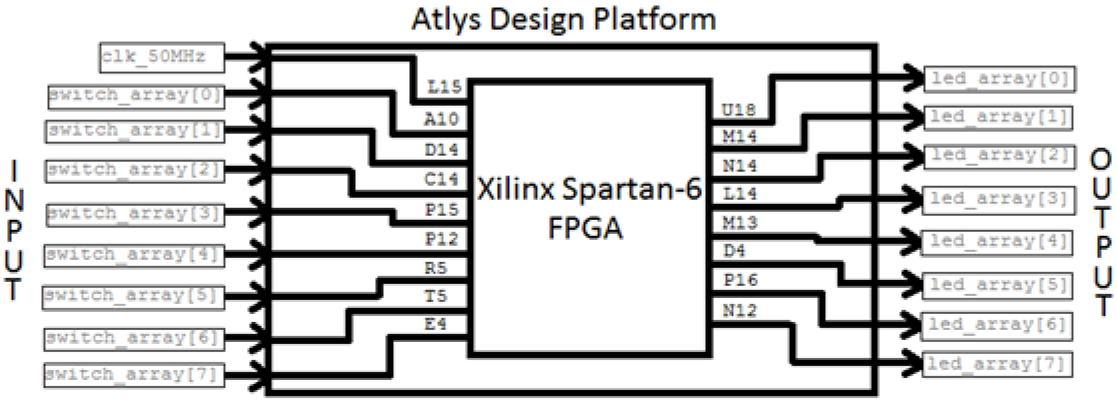


Figure 22. Mapping Process Diagram

The files that the MAP Tool outputs include a Native Circuit Description (NCD) File, with file format extension .ncd, and a Physical Constraints File (PCF) File, with file format extension .pcf. Figure 23 illustrates the Input and Output operation of the MAP Tool.

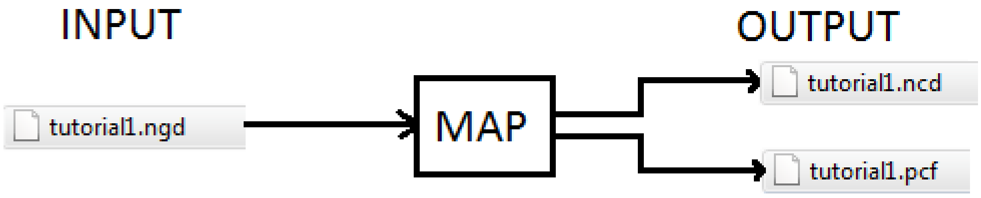


Figure 23. MAP Tool Operation Diagram

The NCD file represents the physical circuit description of the input design as applied to a specific device, which essentially describes which CLBs and IOBs are going to be used. The PCF file contains the physical constraint, which describes the CLBs, IOBs and what interconnects will be used and not used.

Next is the Placement and Routing process. The placement and route process determines the placement of cells and the routing between the cells. Placement determines which CLBs will be used and routing creates the connections between the CLBs.

Eventually the Bit Stream is generated. A bitstream is an encoded bit array sequence sent by a configured FPGA to the system it is connected to. Its purpose is to imitate a native machine code format and apply the functionality encapsulated in it. The information encoded within the array is the binary representation of the netlist and placelist, which describes the placement of CLBs, input/output blocks, three-state buffers, pins, and routing elements.

Finally comes the configuration of the device. Configuration is the process of taking the generated bitstream and loading into an FPGA architecture, in order to emulate the desired functionality. Configuration is entirely dependent on the programmable technology used to make up an architecture. For example, on SRAM-based FPGAs, to configure a device is to load a generated bitstream from a non-volatile device, such as a ROM, to the SRAM-based FPGA. Once loaded, the FPGA will function as described by the configuration of the logic gates described by the bitstream.

2. Definition of the Problem

The purpose of this project is to develop and FPGA programmable station which would allow a user access via a web page available on a server in the FGCU software engineering lab. This web page will prompt the user first to either enter account information to access the content of the site or to create a new account. A SQL database will be used to store account information and reference when a user logs in. The software will check all valid accounts in the database for the information used. Once a user has successfully logged in, they will be prompted to choose which FPGA board they would like to upload a design to, choosing either a Xilinx Spartan-6 comprehensive design board or an Altera DE2 design board. Then the user will choose both a constraints file and VHDL file locally on their computer and upload the two to the server. The web page will then transfer control to an allocated account on ustream [4] where a video web camera will stream a live feed to the site. The server will automatically load the design via USB to the board, configure the logic gates, and begin execution. Each module of the system is shown in Figure 24 below. These modules are the server, FPGA and video web cam.

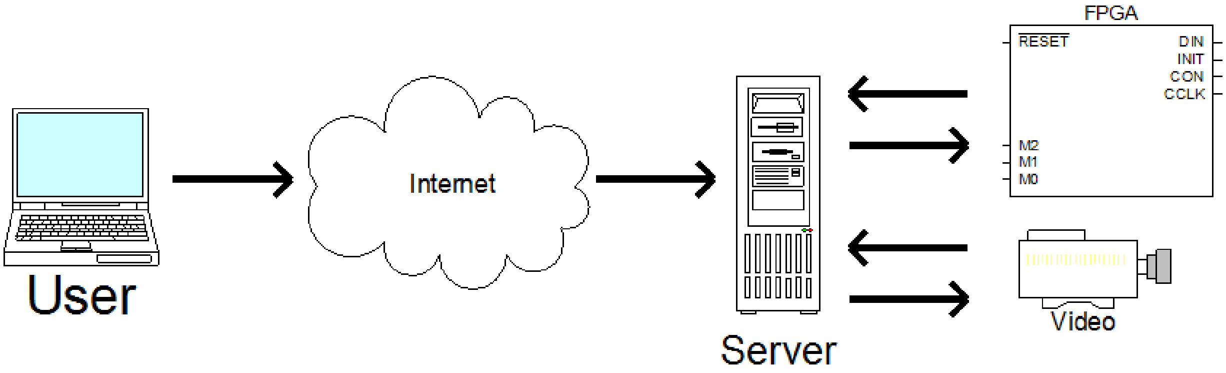
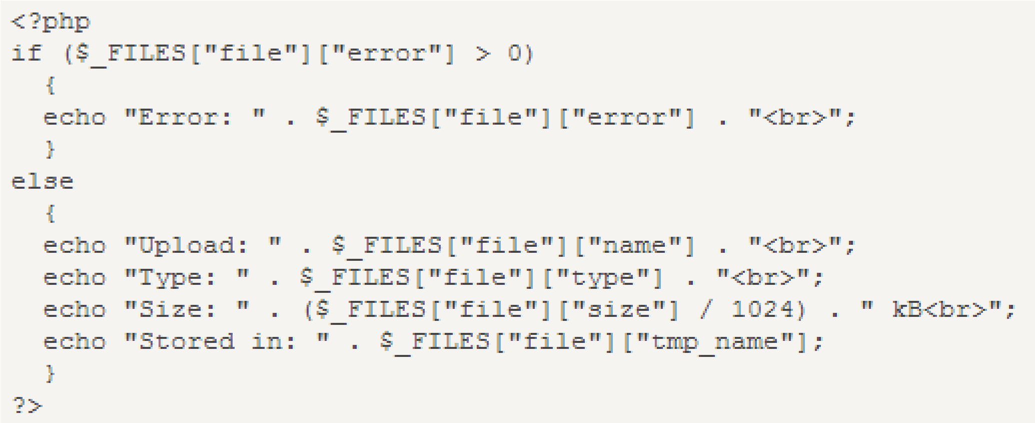


Figure 24. Physical diagram of the system

This project will use a SQL database, PHP and Javascript scripting, HTML, a Logitech web cam, and Xilinx and Altera FPGA development boards. The code from Figure 25, retrieved from w3schools PHP file upload tutorial [5], is a template for uploading designs to the database. Felipe Velosa's report *Maintenance Project: Video Surveillance System* [6] is the basis for configuring and allowing for live streaming from a web cam.



```
<?php
if ($_FILES["file"]["error"] > 0)
  {
  echo "Error: " . $_FILES["file"]["error"] . "<br>";
  }
else
  {
  echo "Upload: " . $_FILES["file"]["name"] . "<br>";
  echo "Type: " . $_FILES["file"]["type"] . "<br>";
  echo "Size: " . ($_FILES["file"]["size"] / 1024) . " kB<br>";
  echo "Stored in: " . $_FILES["file"]["tmp_name"];
  }
?>
```

Figure 25. File upload PHP code from w3schools [4]

The essential part of this project is the development and implementation of a command-line scripting software to configure each of the FPGAs. Altera's *Command-Line Scripting Manual* [7] and Xilinx's *Command Line Tools User Guide* [8] form the basis for this part.

3. Design Solution

Figure 26 shows the physical diagram of the application to be developed that will implement the software described in Section 2, which relies on a SQL database, PHP and Javascript scripting, HTML, a Logitech webcam, and Xilinx and Altera FPGA development boards. The FPGA Scripting Application which will run on a server in the Software Engineering Lab will control all of these components and allow for execution. The entities associated with the application are the UStream Server connected via Internet, Video Camera, Ethernet, Relational Database, FPGA and a User.

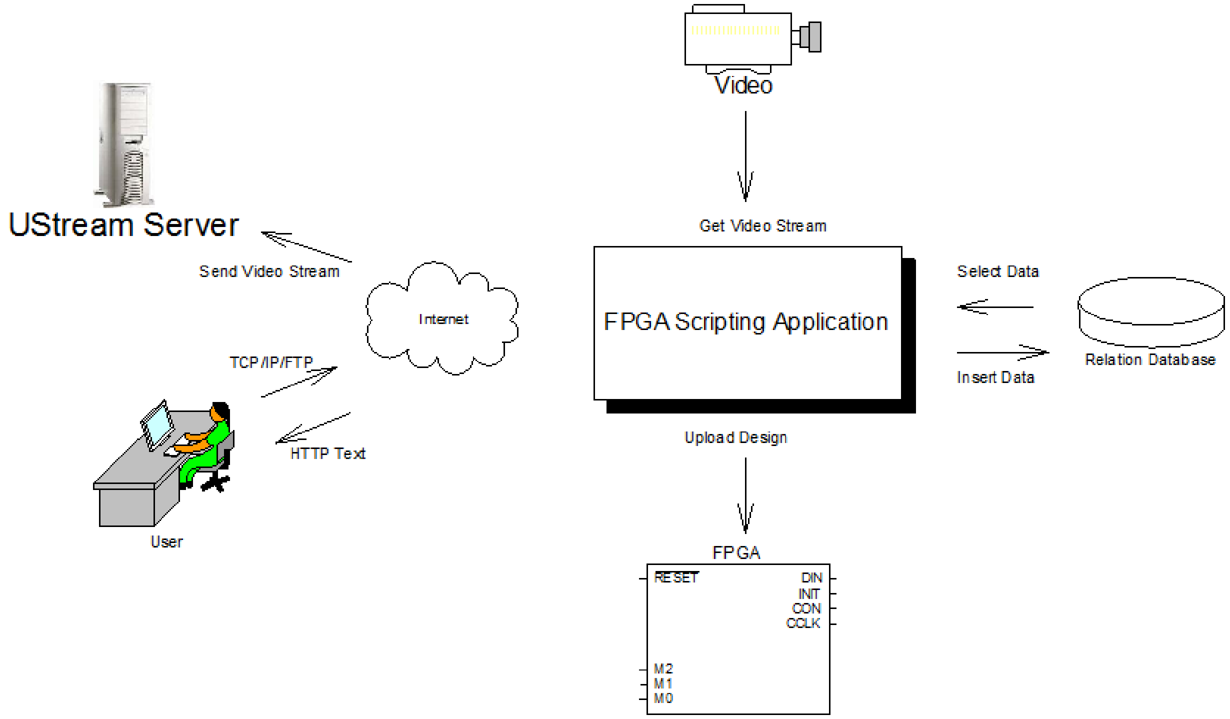


Figure 26. Physical Diagram Illustrating the FPGA Server Structure

Each of these entities communicate with the application using a protocol or following a strict functionality. The user can upload files, view the login and design selection pages using the Transmission Control Protocol (TCP), Internet Protocol (IP), File Transmission Protocol (FTP), and the Hypertext Transfer Protocol (HTTP). The UStream server, which is where the application will send the video stream, obtained from a webcam in the lab, will allow a user to view the execution of the design configured to the FPGA. These designs are available on the relational database the will be stored on the server as well. This server will store the program files, constraint files, and user login information. The FPGA entity is the most important entity of the group, which is the device that will be configured by a user design. The Internet will allow for all communication and data transfers. The application flow chart, which is shown in Figure 27, describes how the application will function and the flow of events controlled by a user using the entities shown in Figure 26.

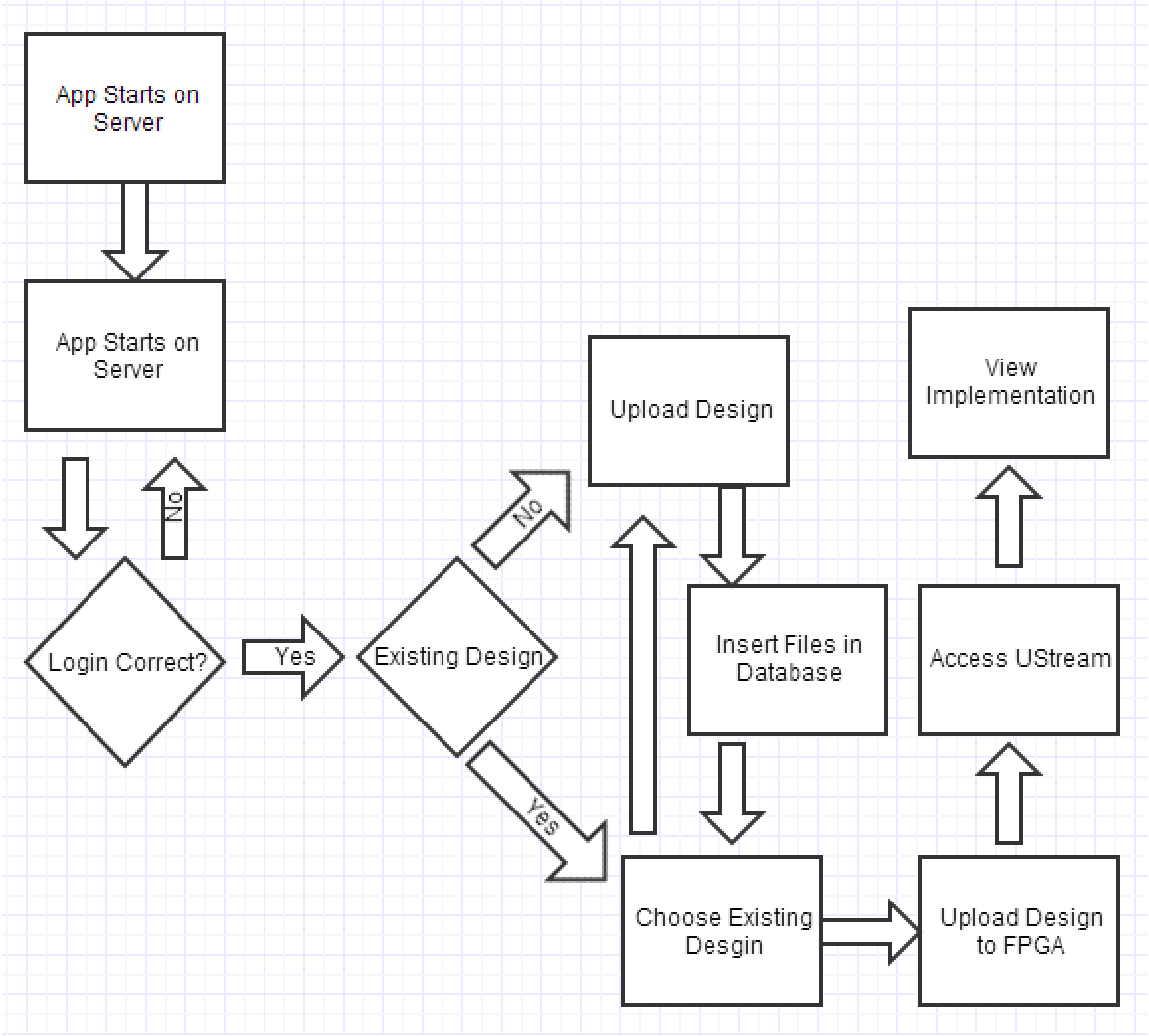


Figure 27. Application Flowchart

The flow of control begins when the application starts; at this point it will wait for a user to open a connection with the server. When a user accesses the server, he will be prompted to login. The user will supply user login information. This login information will be cross compared to existing login information in the database. If it exists, the user will be granted access, if it is not, the user will be denied and be prompted to login in again. Once the user has logged in, he will be prompted to make a decision. The decision is to either upload a design, or choose an existing design from the database. If the user decides to upload a design, he must choose the files locally on his computer; the upload script shown in Figure 25 will be used to implement this function. The user will then have the choice to upload more designs or to choose an existing design available in the database. If the user chooses a design from the database, a script will run that will configure the FPGA with design chosen. UStream will then be accessed with a live video stream of the FPGA shown. The user will view the implementation of the design chosen.

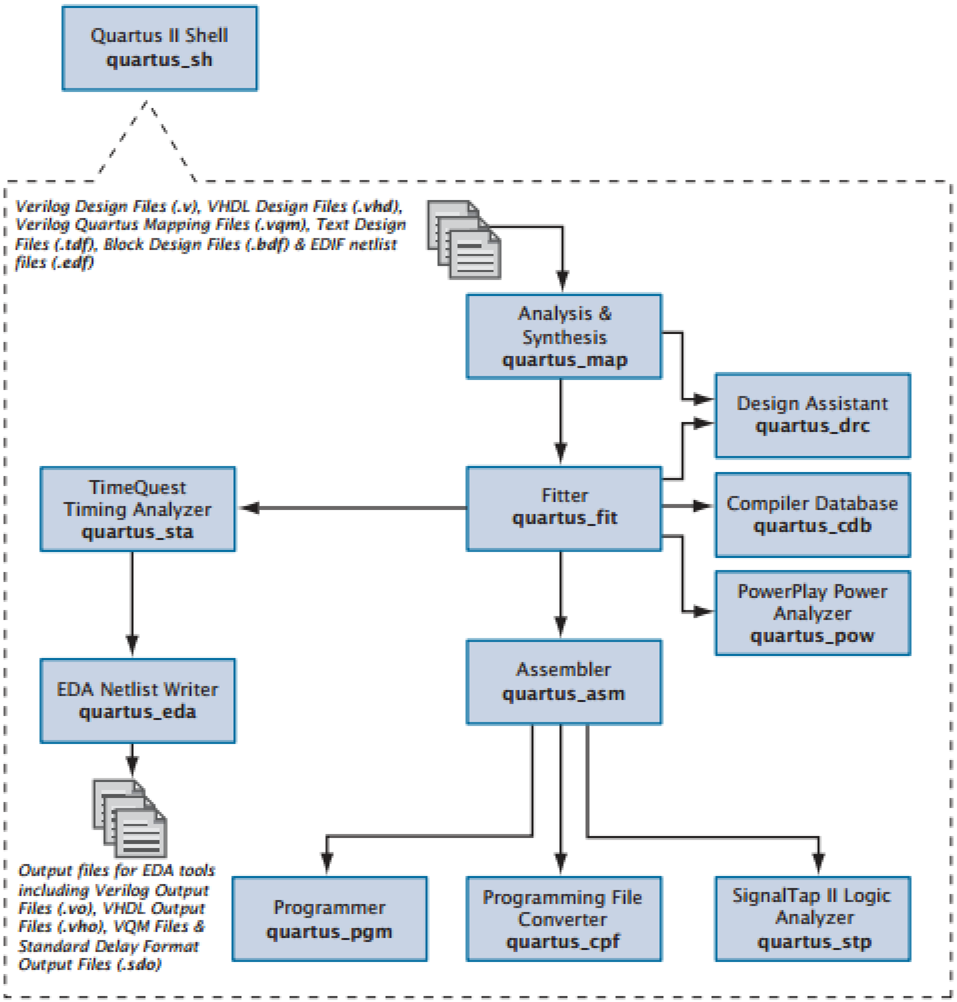

Quartus II Shell
quartus_sh
Verilog Design Files (.v), VHDL Design Files (.vhd), Verilog Quartus Mapping Files (.vqm), Text Design Files (.tdf), Block Design Files (.bdf) & EDIF netlist files (.edf)
Analysis & Synthesis
quartus_map
Design Assistant
quartus_drc
TimeQuest Timing Analyzer
quartus_sta
Fitter
quartus_fit
Compiler Database
quartus_cdb
PowerPlay Power Analyzer
quartus_pow
EDA Netlist Writer
quartus_eda
Assembler
quartus_asm
Output files for EDA tools including Verilog Output Files (.vo), VHDL Output Files (.vho), VQM Files & Standard Delay Format Output Files (.sdo)
Programmer
quartus_pgm
Programming File Converter
quartus_cpf
SignalTap II Logic Analyzer
quartus_stp

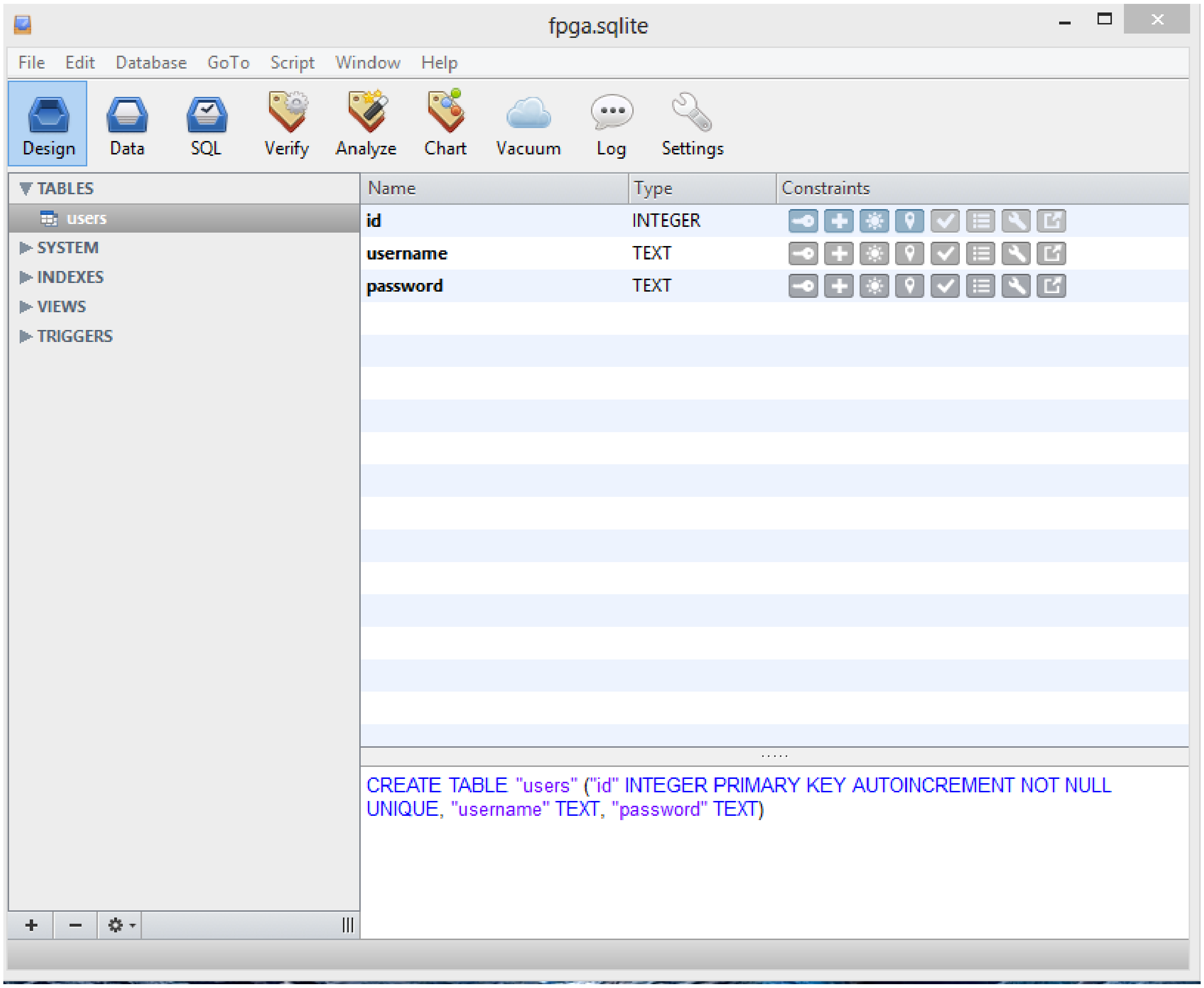
Figure 28. Typical Design Flow [7]

The main problem of this project is the configuration process of the FPGA board. Figure 28, which was taken from the Altera *Command-Line Scripting* manual, shows the design flow of using command-line executables, which is how the Altera FPGA board will be configured. The only other option other than using the command line is to use an automation tool to run the Integrated Development Environments to compile and configure the designs on the devices. This is not an appropriate option and will not be done, since it is unreliable. Suppose the server was interrupted during the automation process, the entire task would become compromised. This is a risk that one should not take. Implementing the process via a batch is more reliable and secure.

4. Implementation and Experimentation

This project borrows extensively from the previous development by Robert Porter and Olexiy Kovtunenko in *Remote FPGA Web Lab User* [1]. There have been a few additions to the website so that it can be developed locally on a computer in the Software Engineering lab.

The recovery and maintenance of the existing web page required two massive changes in the code. The first change was using SQLite, a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The developers originally used Microsoft SQL, but SQLite is a better fit since it is a lightweight, reliable database engine. Figure 29 shows the interface to the management studio of SQLite.

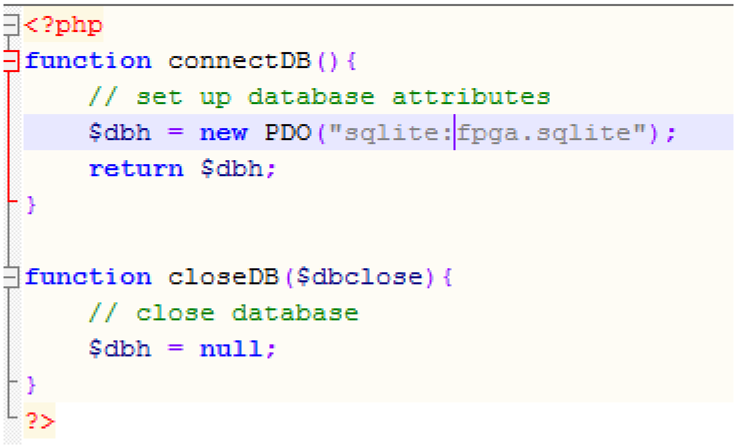


Name	Type	Constraints
id	INTEGER	
username	TEXT	
password	TEXT	

```
CREATE TABLE "users" ("id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL
UNIQUE, "username" TEXT, "password" TEXT)
```

Figure 29. FPGA Table in SQLite Database

The connectivity to the database also changed in the PHP code is shown in Figure 30. PHP Data Objects (PDO) where implemented. PDOs are extensions that define lightweight, consistent interfaces for accessing databases in PHP. The database was appropriately named `fpga.sqlite`.

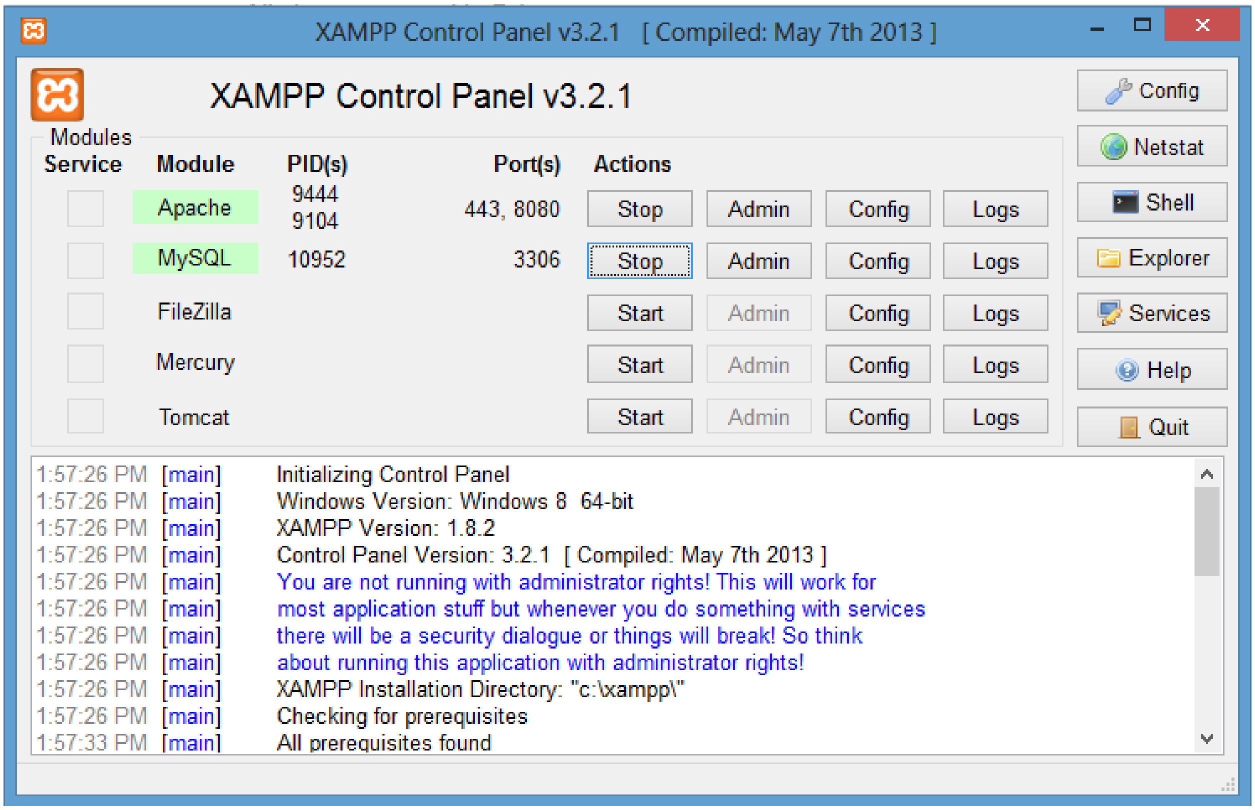


```php
<?php
function connectDB(){
    // set up database attributes
    $dbh = new PDO("sqlite:fpga.sqlite");
    return $dbh;
}

function closeDB($dbclose){
    // close database
    $dbh = null;
}
?>
```

Figure 30. Change Database Connection in PHP code

The other change was how the web page, and its files were served. For the time, while the project is still developing, the web page will developed locally using the XAMPP [9]. XAMPP is a free, open source cross-platform web server solution stack package. The package has many modules, but the webpage only uses the Apache HTTP Server and interpreters for scripts written in the PHP. The control panel is shown in Figure 31.


XAMPP Control Panel v3.2.1   [Compiled: May 7th 2013]
XAMPP Control Panel v3.2.1
Modules
Config
Netstat
Shell
Explorer
Services
Help
Quit

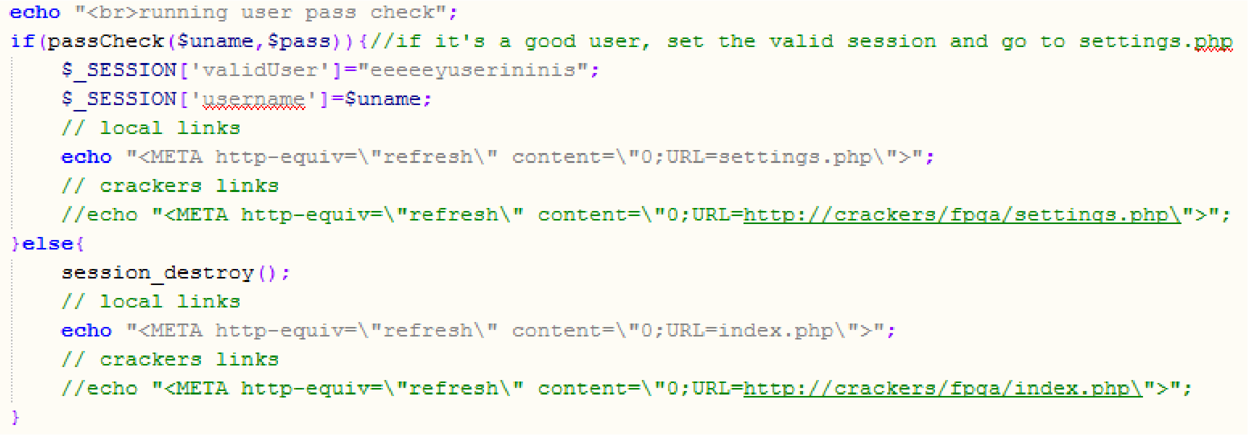
Service	Module	PID(s)	Port(s)	Actions			
□	Apache	9444 9104	443, 8080	Stop	Admin	Config	Logs
□	MySQL	10952	3306	Stop	Admin	Config	Logs
□	FileZilla			Start	Admin	Config	Logs
□	Mercury			Start	Admin	Config	Logs
□	Tomcat			Start	Admin	Config	Logs

```
1:57:26 PM  [main]   Initializing Control Panel
1:57:26 PM  [main]   Windows Version: Windows 8  64-bit
1:57:26 PM  [main]   XAMPP Version: 1.8.2
1:57:26 PM  [main]   Control Panel Version: 3.2.1  [ Compiled: May 7th 2013 ]
1:57:26 PM  [main]   You are not running with administrator rights! This will work for
1:57:26 PM  [main]   most application stuff but whenever you do something with services
1:57:26 PM  [main]   there will be a security dialogue or things will break! So think
1:57:26 PM  [main]   about running this application with administrator rights!
1:57:26 PM  [main]   XAMPP Installation Directory: "c:\xampp\"
1:57:26 PM  [main]   Checking for prerequisites
1:57:33 PM  [main]   All prerequisites found
```

Figure 31. XAMPP Control Panel

All the URL locations referenced in the HTML and PHP code had to be changed to accommodate for this change in how the page was being served. Figure 32 shows an example of how the links were changed. The links originally referenced URLs at the website `http://crackers.com`, but they were changed to local directories.

Note. I will now try to implement, using the website that I have managed to recover and maintain, an emulation software that will allow a user to remotely control the physical components of the FPGA board in the lab. This form of remote control will provide an essential component to the remote lab experience. Now a user is only able to configure the device. If I am able to complete my design, the user will be able to actually use it.



```php
echo "<br>running user pass check";
if(passCheck($uname,$pass)){//if it's a good user, set the valid session and go to settings.php
    $_SESSION['validUser']="eeeeeyuserininis";
    $_SESSION['username']=$uname;
    // local links
    echo "<META http-equiv=\"refresh\" content=\"0;URL=settings.php\">";
    // crackers links
    //echo "<META http-equiv=\"refresh\" content=\"0;URL=http://crackers/fpga/settings.php\">";
}else{
    session_destroy();
    // local links
    echo "<META http-equiv=\"refresh\" content=\"0;URL=index.php\">";
    // crackers links
    //echo "<META http-equiv=\"refresh\" content=\"0;URL=http://crackers/fpga/index.php\">";
}
```

Figure 32. Change in URL code Example

5. Conclusion

The website developed by Robert Porter and Olexiy Kovtunenko in *Remote FPGA Web Lab User* [1] was successfully restored. In future iterations of this project the is to allow the user to interact with the I/O peripherals on the FPGA board via an emulation software available on the web page.

6. References

[1] R. Porter and O. Kovtunenko. *Web Based FPGA Lab*, Florida Gulf Coast University, Ft Myers, FL, September 9, 2008
http://itech.fgcu.edu/faculty/zalewski/CNT4104/CNT4104projects.html

[2] R. Porter and O. Kovtunenko. *Remote FPGA Web Lab User Manual*, Florida Gulf Coast University, Ft Myers, FL, March 30, 2009
http://itech.fgcu.edu/faculty/zalewski/CNT4104/CNT4104projects.html

[3] V. Giannone. *Security at the Chip Level (FPGA)*, Florida Gulf Coast University, Ft Myers, FL, December 21, 2012

[4] Ustream
http://www.ustream.tv/

[5] PHP File Upload
http://www.w3schools.com/php/php_file_upload.asp

[6] F. Velosa. *Maintenance Project: Video Surveillance System*, Florida Gulf Coast University, Ft Myers, FL, April 23, 2013
http://itech.fgcu.edu/faculty/zalewski/CNT4104/CNT4104projects.html

[7] Altera. Command-Line Scripting
http://www.altera.com/literature/hb/qts/qts_qii52002.pdf

[8] Xilinx. Command Line Tools User Guide
http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/devref.pdf

[9] XAMPP. Apache Distribution Containing MySQL, PHP and Perl.
http://www.apachefriends.org/en/xampp.html