UNIVERSITY OF ALASKA ANCHORAGE

CSCE A470

CAPSTONE PROJECT

Cable Headend Virtualization Test Application

Author:

Paul Kelly

Supervisor:

Dr. J. Randy Moulic

Anchorage AK, May 2015

Computer Science & Engineering Department
UNIVERSITY of ALASKA ANCHORAGE

kelly.paul.r@gmail.com

Version 1.0

Contents

Chapter 1
Introduction

1.1 Introduction

Hardware virtualization might also be thought of as dynamic hardware, or hardware with plasticity. One disadvantage that hardware has historically had over software is that software is malleable and able to be easily adapted to changes and improvements in infrastructure. Hardware on the other hand usually has to be replaced or physically modified to remain compatible with changing infrastructure. This can be expensive and time-consuming. Advances in FPGA (Field Programmable Gate Array) boards have made them smaller, and has increased their computational power. This in turn has made it easier to virtualize the functions of entire components. These virtual components can be modified and interchanged now much in the same way as software applications. This adds an advantage to companies with large, constantly changing infrastructures, such as digital cable video distributors. Another advantage is that they can reduce the amount of components in their large and complex systems by virtualizing multiple components on a single FPGA board. This project aims to prove that the integration of FPGA boards in a digital cable video distribution system is possible and advantageous.

To test this hardware, it will be necessary to implement a test signal. The generator for this signal can also be constructed virtually in the FPGA board.

Figure 1.1: An FPGA board manufactured by Altera. [1]

1.2 Application

The implementation of the virtual components on the FPGA will be modeled from the current digital cable delivery system.

Ideally, digital cable delivery systems function with these components:

1. receiver/decoder units;
2. video encoding units;
3. video processing units; and
4. video transcoding units.

We would use the FPGA board to integrate all of these separate units into one. The system we will develop will perform the following steps:

1. receive a Digital Video Broadcast Satellite (DVB-S) signal, simulated only in this case;
2. decode the content;
3. select the channel requested by the provider;
4. transcode the video into MPEG-2;
5. encode it for the cable network;
6. convert the video into an IP stream; and
7. send it out to the user.

A simplified version of this is outlined in the 4-step diagram in figure 1.2 below.

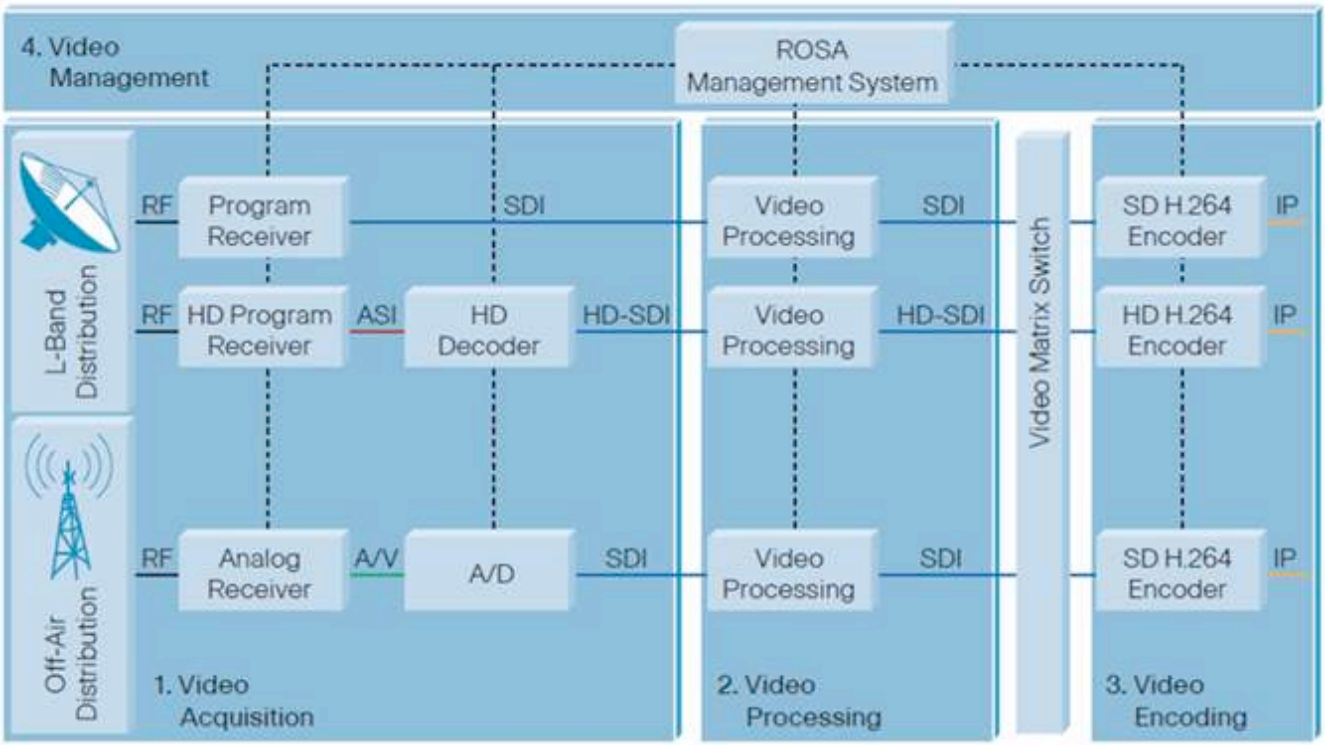


Figure 1.2: Four-step diagram of a cable headend system [2].

1.3 Motivation

The motivation for this project is to streamline the delivery of cable video.

Virtualizing multiple hardware components on one board reduces the amount of equipment a cable company must purchase, house, and maintain, reducing the overall cost of the system.

Replacing equipment is also made easier. Normally, each cable channel delivered requires its own box, which has been configured specifically to deliver that channel. If one of those boxes should fail and there are no backups available on site, then there is a prolonged interruption in the system. The advantage of using FPGAs in this case is their ability to be easily re-configured.

Virtual hardware allows the system to become more responsive to user demand. As user demand for channels increases or decreases, a virtual system can be reconfigured to give more or less bandwidth to those channels. Power consumption can also be rescaled.

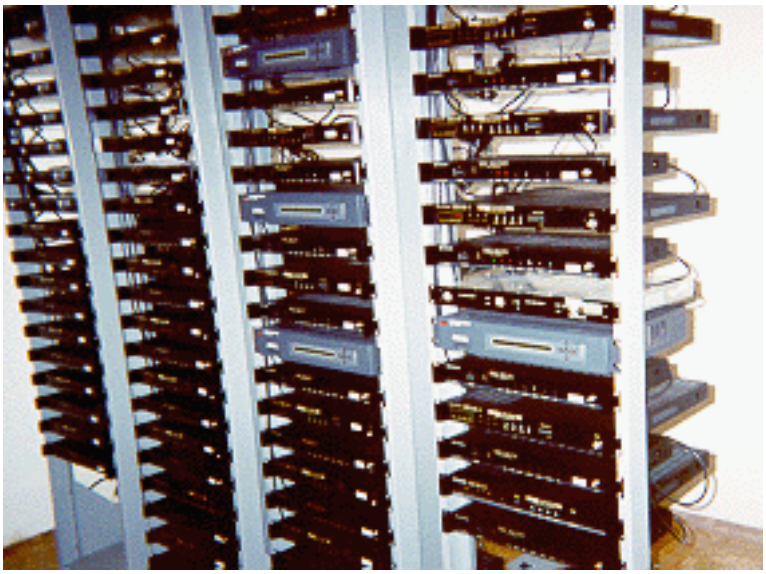


Figure 1.3: A typical cable headend shelve-mounted system. [3]

1.4 Recent Developments

Recent developments in FPGA technology have reduced the size of the chips. Two major brands of these boards are the Altera and Xilinx boards. While these brands now boast gate sizes as small as 16 nm, these are cost prohibitive. The 20 nm gate-sized chips are more cost effective, and with a processing speed at above 10 Gbps [4] are fast enough for our proof of concept trials.

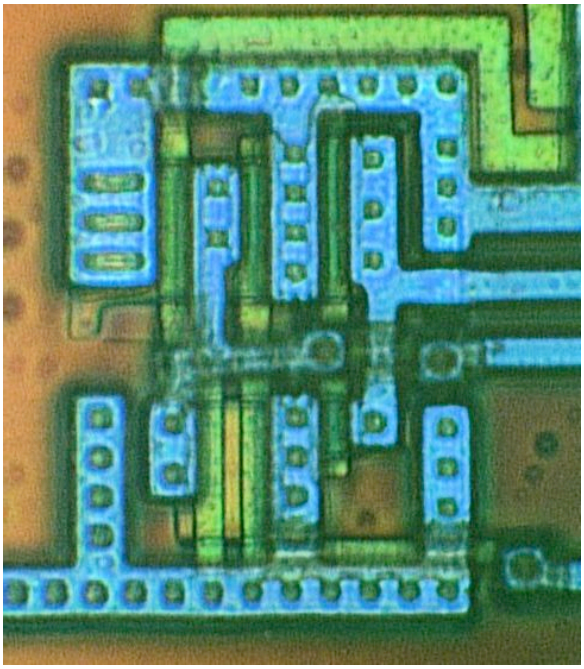


Figure 1.4: CMOS AND gate imaged by a confocal microscope. [5]

Figure 1.5: A Xilinx FPGA board. [6]

Chapter 2

System Integration and Modeling / Methodology

2.1 Altera FPGA Board

FPGA boards come from a few notable manufacturers. We have chosen to use an Altera FPGA board. The most suited model to our purposes is the Cyclone IV. It comes equipped with a port that can accept an auxiliary board. That auxiliary board can accept input and output SDI cable connections. The test signal would be generated by a virtualized Cyclone IV component and sent to the output. Our virtual headend receiver would also exist within the same board and be listening to the SDI input.

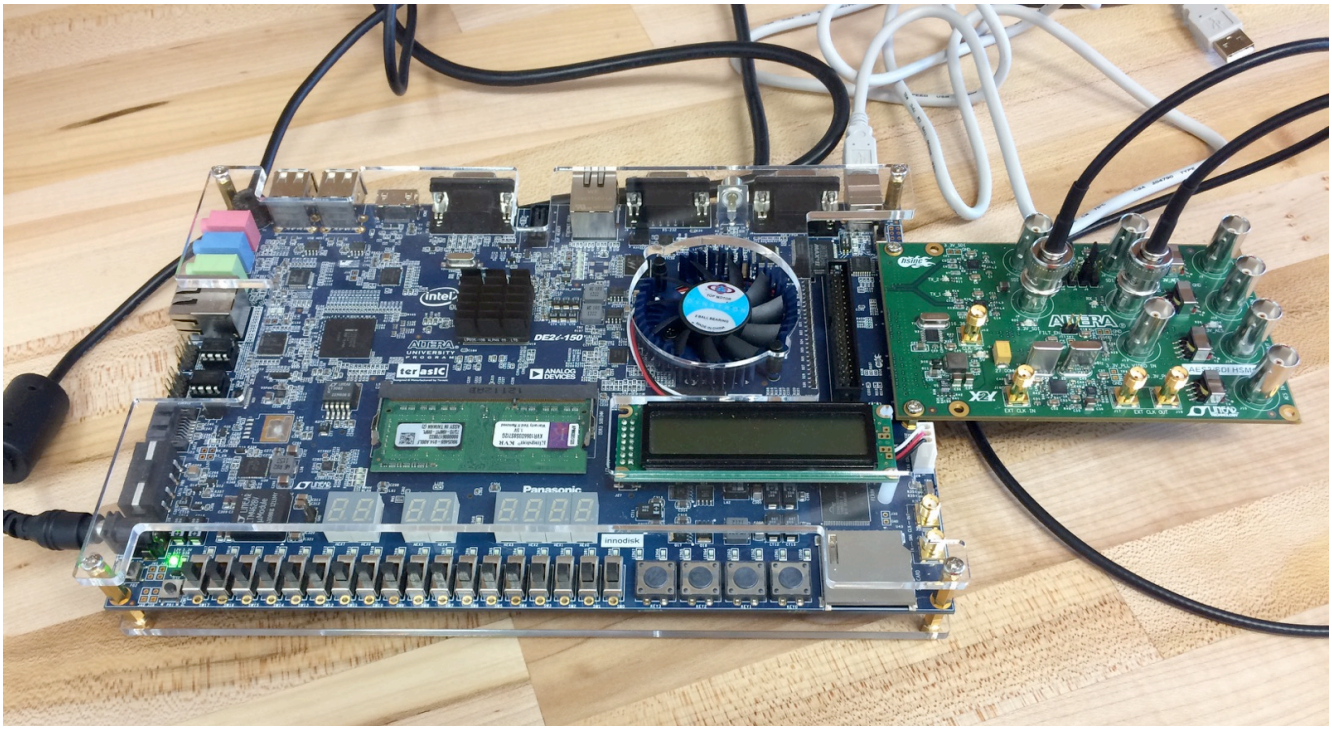


Figure 2.2: Picture of our Altera Board with Auxiliary Board connected, and SDI cables connected to their ports.

2.2 Quartus Lite

Quartus Lite is the software used to program Altera FPGA boards. The behavior of individual modules can be programed logically using VHDL code or Verilog code. The modules can then be represented as GUI's in terms of their inputs and outputs. The file containing these GUIs is called a schematic file, and it represents a single device or module itself. The inputs and outputs of each module can then be connected to each other or become an input or output for the entire device represented by the schematic. The module represented in the schematic could also be compressed into a single GUI within a larger module. Quartris Lite could then program the FPGA board to simulate any of these modules.

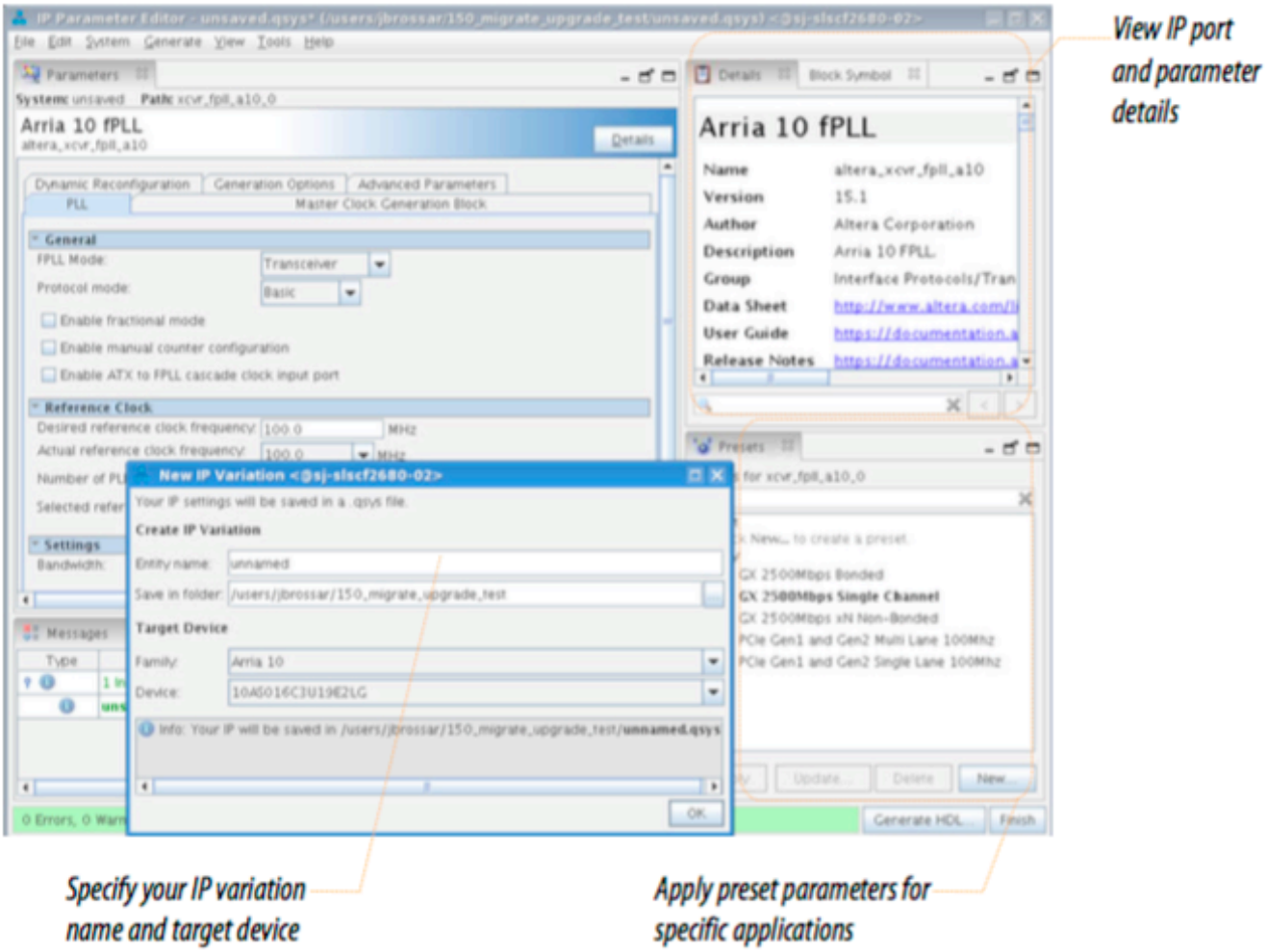


Figure 2.2: Screenshot of Quartus software taken from an Altera user manual. [7]

2.3 Test Signal

The test signal will comprise one module that would be run on the FPGA board concurrently with the receiver, encoder, and decoder modules. There is already a premade module produced by Altera that with minor modifications would produce the test signal. The signal could be designed according to whatever specifications we choose. My job is the encoding of the signal so that we can test the decryption ability.

2.4 Agile Methodology

Agile methodology holds to a few tenants. For one, the process is only loosely sequential. The process also involves frequent consultations with the client. It has a lot of the same steps as Waterfall and other traditional methods. However, instead of exercising those steps sequentially, Agile requires that the steps be visited and revisited with every iteration.

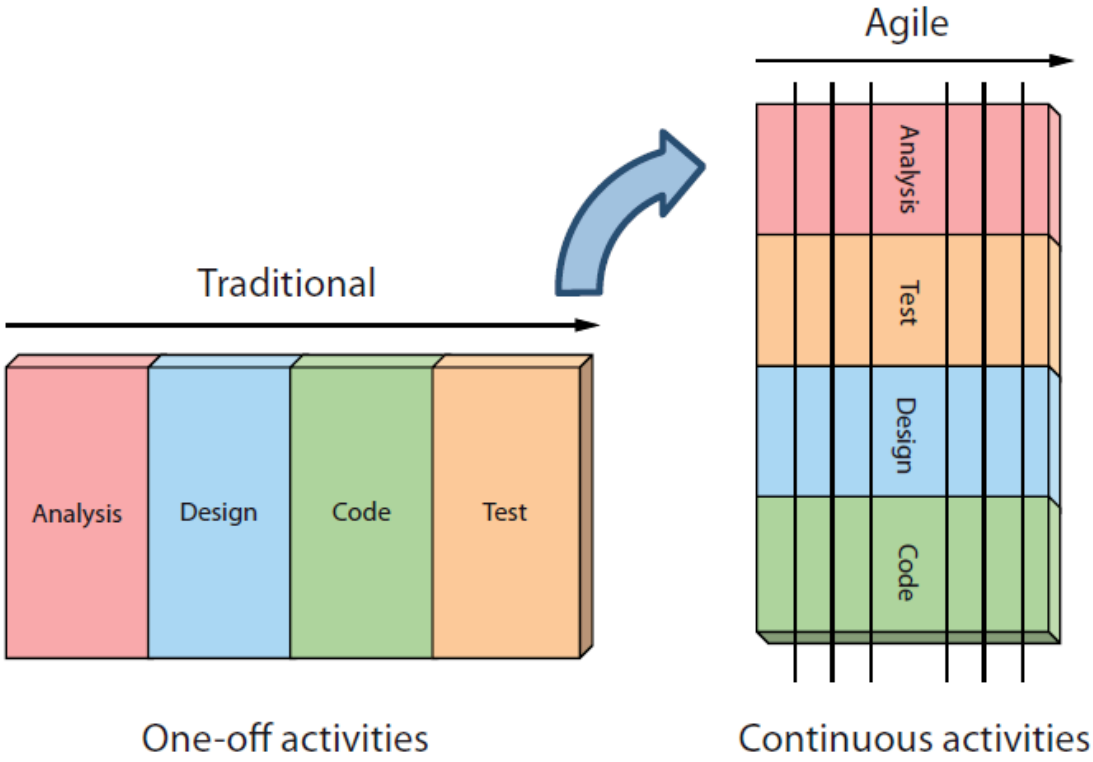


Figure 2.3: Agile projected timeline. [8]

While the process might be implemented to some degree with this project, it only loosely applies. The traditional Agile process is set up for a project in which most of the work is software based, and in which the desires of the customer could be in flex. In this case, while there is some coding involved, most of the work in on a much lower level than traditional software development.

Agile allows for planning some flexibility in the different aspects of the project including scope, time, quality, cost, and other aspects as the developer and client deem appropriate. In this case, time is fixed. It is known that this entire project cannot be completed within the timeframe of one semester. Financial cost is not really a large concern since the developers are not being paid a salary. What might be important is the cost of equipment. Neither the university, nor our supervisor Dr. Moulic has specified a budget to us. However, it is understood that costs should be minimal to accomplish the desired proof of concept.

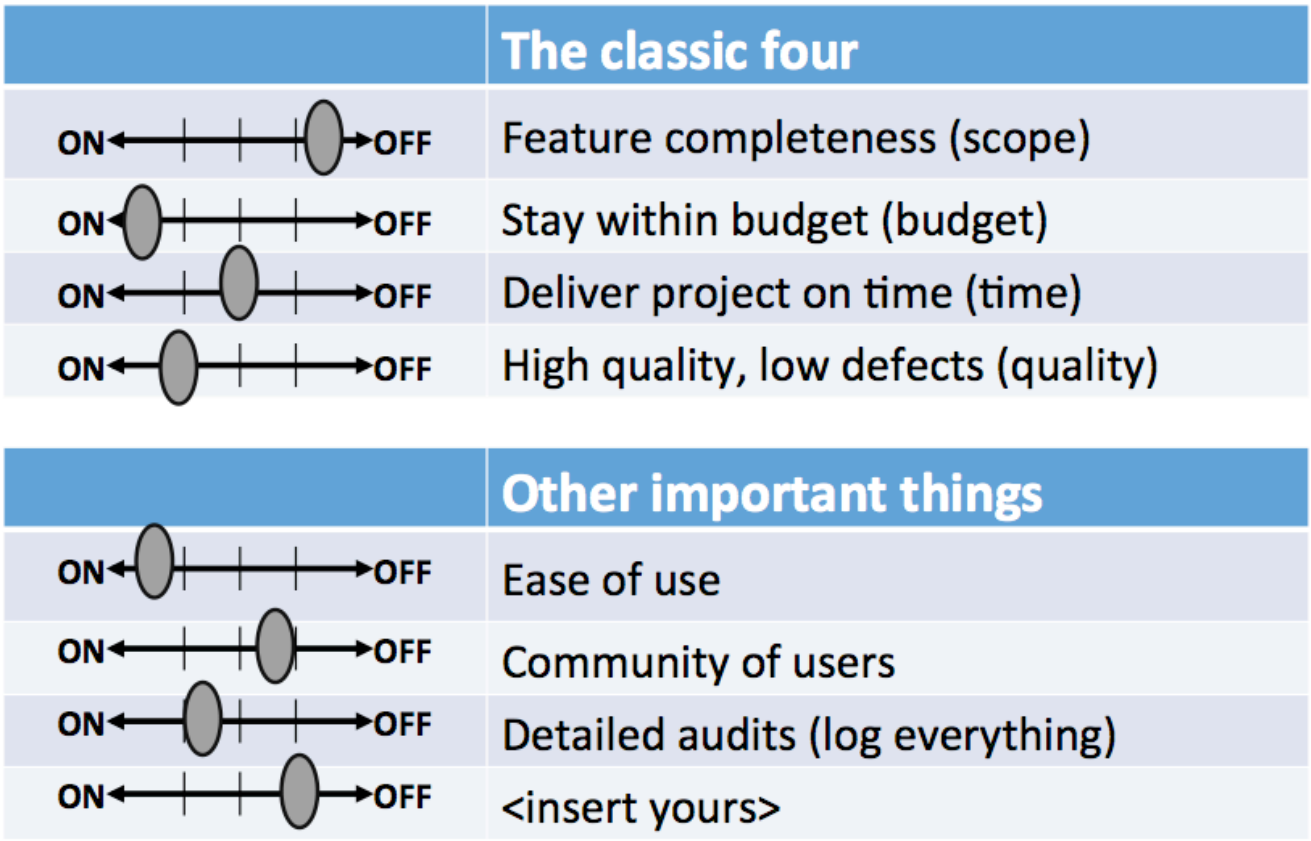


Figure 2.4: Example of how one might prioritize a typical Agile project. This slider comes from a PowerPoint presentation in part prepared by Agile. [9]

2.5 Gantt Chart

I did not enter this project from the beginning, so there is some delay in the start time. However, this is my ideal timeline illustrated below.

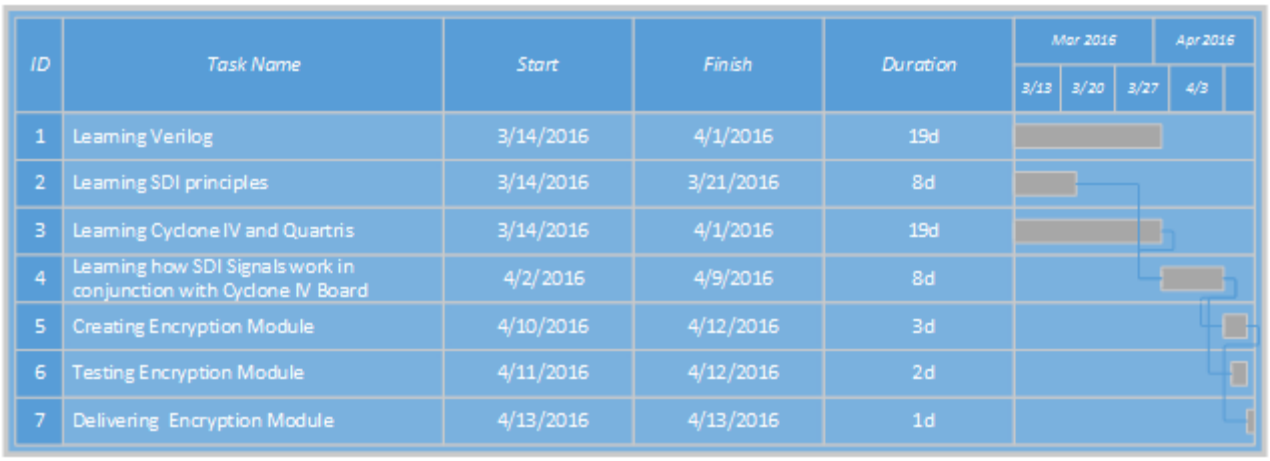


Figure 2.5 Gantt Chart of my project timeline resulting the delivery of the encrypted test signal.

Chapter 3

Design and Testing / User Interface

3.1 User Interface

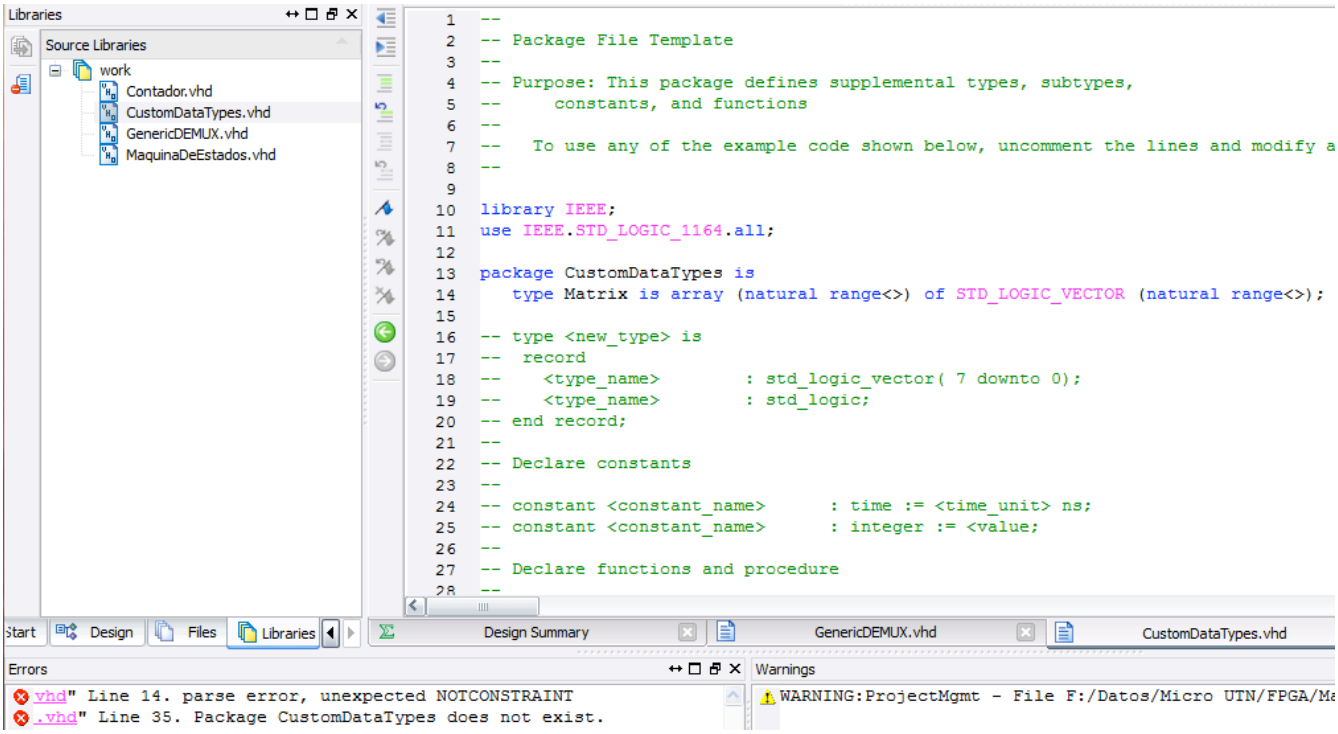
There is no perceived need to develop a specific user interface for this technology. The cable headend systems themselves are hardware. An FPGA, once installed would theoretically have no further interaction with an operator. The only time that an FPGA would need an interface is when it is being programmed and configured.

Figure 3.1: A typical cable headend device for a single channel. [10]

The Quartus and Xilinx programs, which are used to program the FPGAs, already have their own user interfaces. The interfaces used by these programs perform three main tasks.

The programs provide an integrated development environment (IDE) from which to code the logic and behavior of certain modules and components. Both Quartus and Xilinx can be configured to process the logic in either VHDL code or in Verilog code. The documents where this code is written are formatted as ordinary text files with their extension indicating

which language they are written in (“.v” denotes documents written in Verilog and “.vhdl” denotes documents written in VHDL).




```
1  --
2  -- Package File Template
3  --
4  -- Purpose: This package defines supplemental types, subtypes,
5  --     constants, and functions
6  --
7  --   To use any of the example code shown below, uncomment the lines and modify a
8  --
9
10 library IEEE;
11 use IEEE.STD_LOGIC_1164.all;
12
13 package CustomDataTypes is
14    type Matrix is array (natural range<>) of STD_LOGIC_VECTOR (natural range<>);
15
16 -- type <new_type> is
17 --  record
18 --    <type_name>        : std_logic_vector( 7 downto 0);
19 --    <type_name>        : std_logic;
20 -- end record;
21 --
22 -- Declare constants
23 --
24 -- constant <constant_name>      : time := <time_unit> ns;
25 -- constant <constant_name>      : integer := <value;
26 --
27 -- Declare functions and procedure
28 --
```

Figure 3.2: Example of Xilinx IDE for VHDL coding. [11]

The second main task is to provide a graphical user interface (GUI) from which physical abstractions of components and modules can be configured. Some components and modules already exist within the libraries of Xilinx and Quartus. Others can be synthesized from user-coded VHDL or Verilog files. The GUI provides the user access to the inputs and outputs of the modules while the internal logic is governed by the programmed logic of the abstraction.

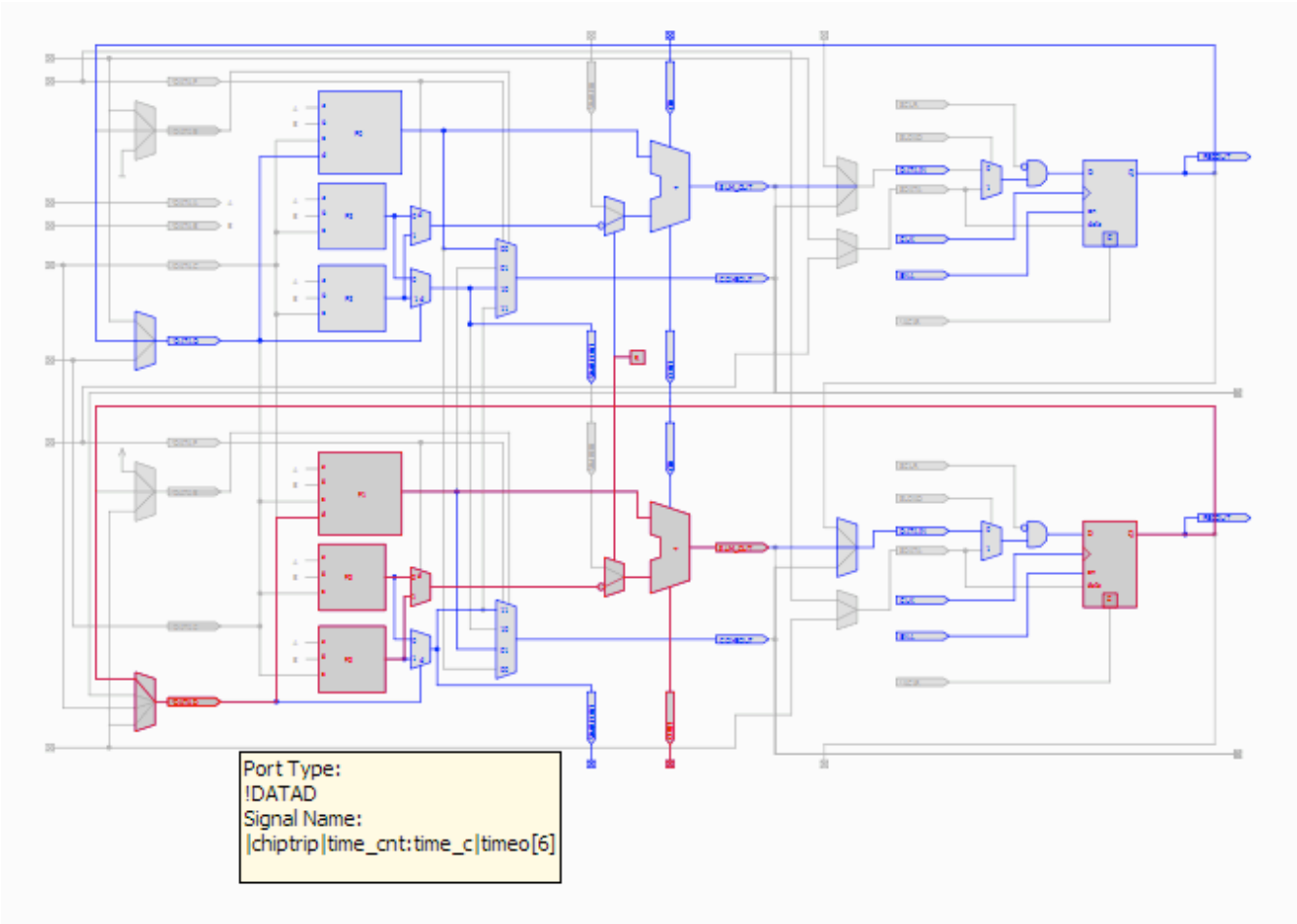


Figure 3.3: Example of Schematic view in Altera. [12]

The third main interface of the Quartus and Xilinx programs is the test bench. This interface allows the user to test behavior of the module being implemented without uploading the head module to the FPGA board. Outputs and inputs are modeled in relation to each other and the device clock so that the user can confirm that the desired behavior is being implemented.

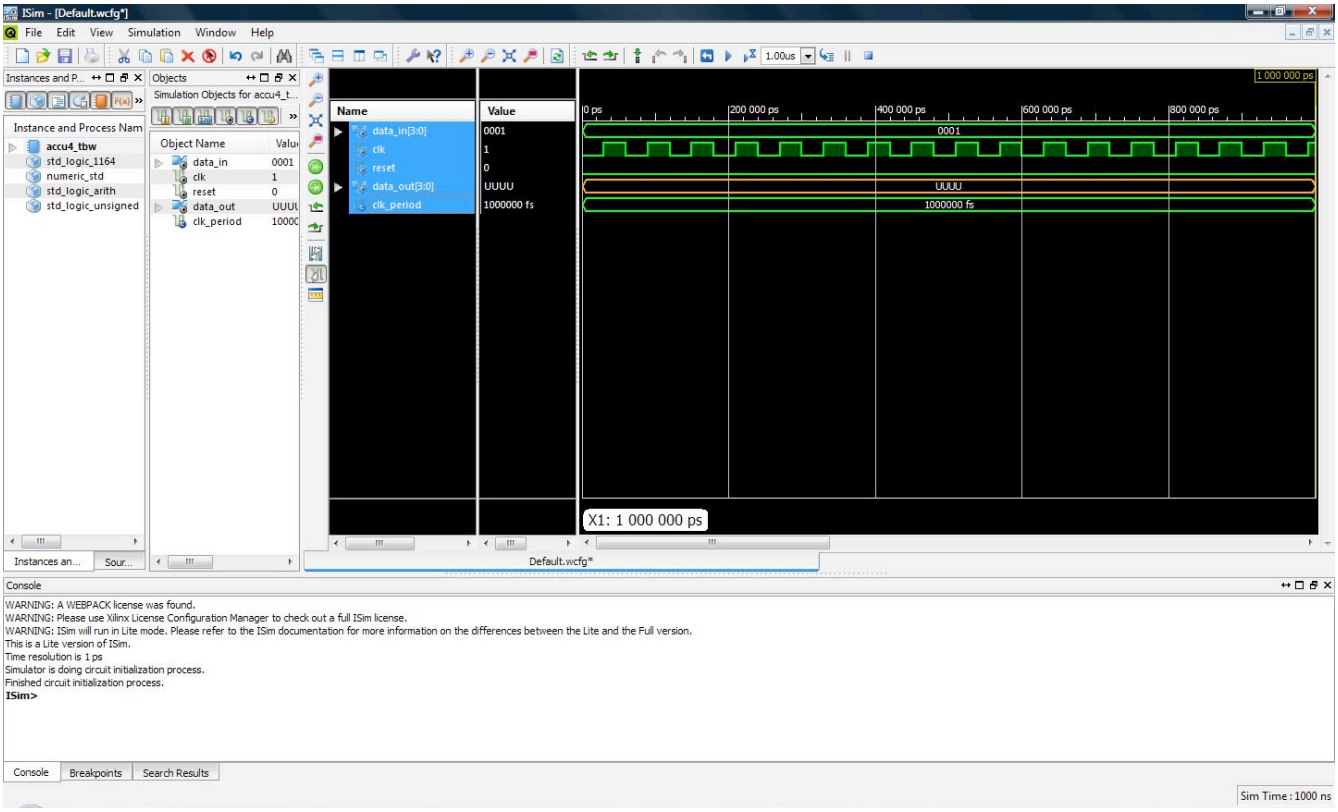


Figure 3.4: Example of Xilinx test bench. [13]

3.2 Testing and Techniques

There are four popular testing techniques that are used in conjunction with each other in software engineering. These include Unit Testing, Integration Testing, System Testing, and Operational Testing. Not all of these can be strictly followed this project, or for my part of the project: the encoding of the test signal. We do plan on implementing some of these tests

Unit Testing is usually the testing of certain abstractions within written code called classes and methods. These test classes usually create instances of the classes they are testing. They then call methods belonging to these classes, usually providing an input. The unit tests are usually written to expect a very specific output given that input, and if they do not get it, then the that specific test is said to have failed. Most modern IDEs support these unit test classes, or unit test suites; which contain multiple unit test classes. However, the closest thing that exists in the Quartus environment is the test bench. In the test bench a Verilog or VHDL file is written referencing the module being tested. It sets that module's inputs to certain values. However, instead of passing or failing tests, the outputs are simply monitored and the user is left to determine if the output was expected and/or desired. All modules should be tested in this way before being uploaded to the FPGA board.

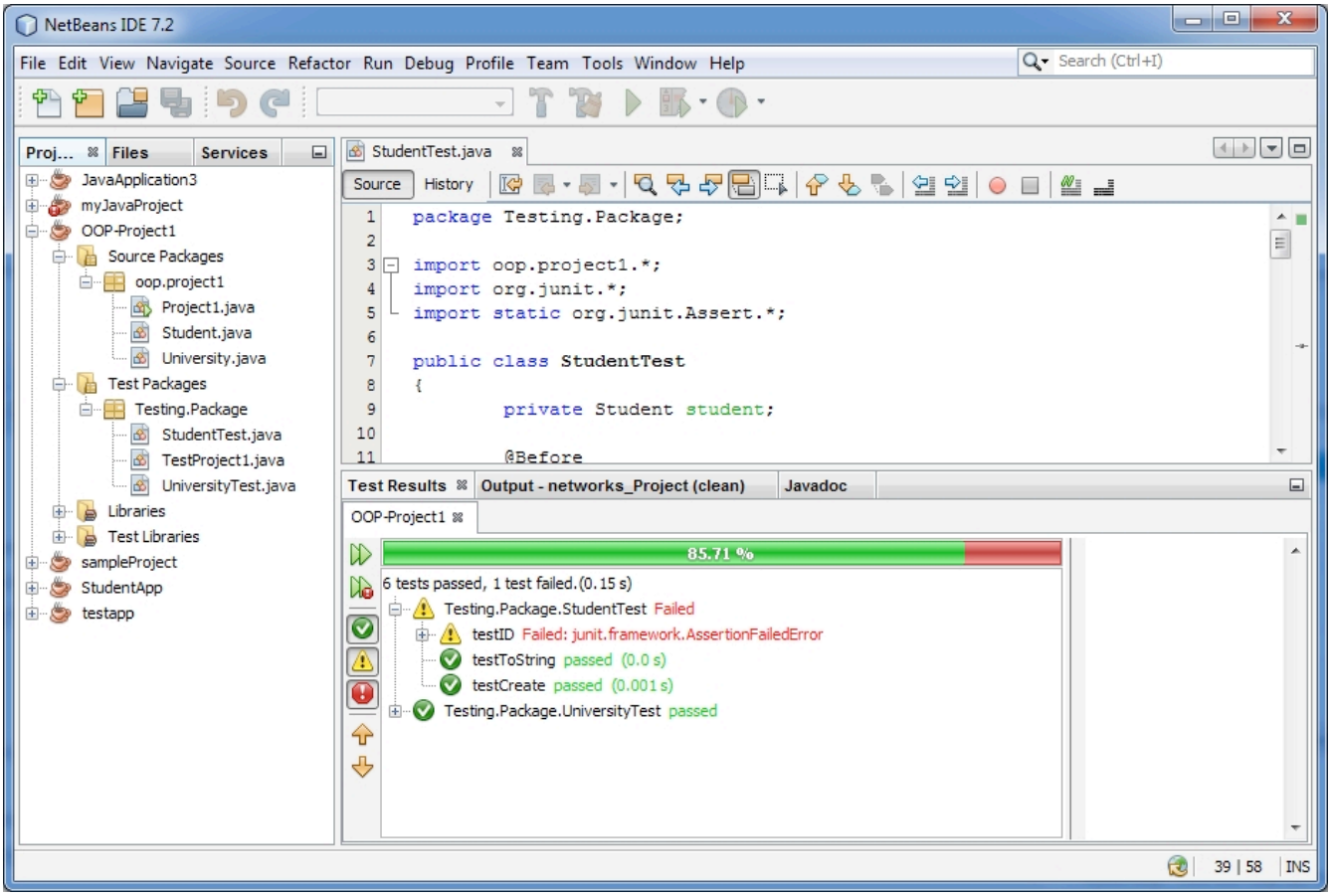


Figure 3.5: Example test suite being ran using the IDE Netbeans. [14]

Integration testing is testing that examines the interactions between different components of software. The closest this could be applied in our project would be creating test benches for individual groups of modules after they have been unit tested in the test bench. I will only be testing one module, and that is the module that encodes the test signal. It should be integrated separately with the modules that generate the signal and receive it to verify that the signal is encoded and passed along.

The system test will likely require the top module to be uploaded to the FPGA board. A C++ program will also be running on the board to decrypt the signal my module has encrypted. This will be monitored as a bit stream, though not analyzed.

This leaves operational acceptance testing. In software engineering this is the testing that verifies that the requirements of the project will be met while being ignorant of the construction of the code, or in our case the components. This is the final test that will be done to verify that the system functions as designed, and as a user would expect. Our acceptance test should verify through LED lights on the FPGA board that a signal is being transmitted. A USB attached to the Cyclone IV FPGA will pass the final resulting signal as a bit stream to be monitored by a computer.

3.3 The Agile Method

The Agile system is a system based on both managing and coding a project to produce a solution that is both useful and acceptable to the client or end users. Agile is aptly named because agility is built into all aspects of the process.

On the project management side, regular meetings are held with the clients. Requirements of the project remain fluid to some degree as both the project manager, the developers, and the client develop a better understanding of the desired outcome. An important part of Agile project management include keeping a team working in the same location on a regular basis. This increases communication and efficiency. Another important part is bringing the developers in on the client meetings, eliminating the middle man whenever possible.

Agile coding involves keeping the code as organized, readable, and flexible as possible. This means in part not relying so much on comments written into the code. This tends to clutter the code, and makes the code less adaptable. For example, if a client were to change a requirement, requiring changes to the code, the comments left in the code could not only become obsolete, they could actually confuse understanding about what the code is designed to do, making debugging more difficult. Instead Agile suggests structuring the code and naming variables such that it becomes apparent what the code in a given section is designed to do.

Chapter 4

Results and Discussion

4.1 Context

The original scope of this project was a proof of concept. The goal of the final project is to prove that the functionalities of cable headend devices could be virtualized in a cable headend system. This did not just include the functionalities of receiving the signal and transcoding it into a MPEG format, but also the functionality of decrypting an incoming signal and re-encrypting the resulting signal. To accomplish all of these proofs was anticipated from the beginning to take multiple semesters. It was understood that the team was only entering on the first semester of this project, and that future teams would pick up where we left off.

My responsibility with the project consisted of attempting to encrypt the signal by passing the test signal through a virtually created encoder. The other team members were responsible for generating the signal I would be decrypting, for creating the virtualized headend system, and for the decryption. The signal was to pass through my encoder, out through an SDI port, and back in through an input SDI port.

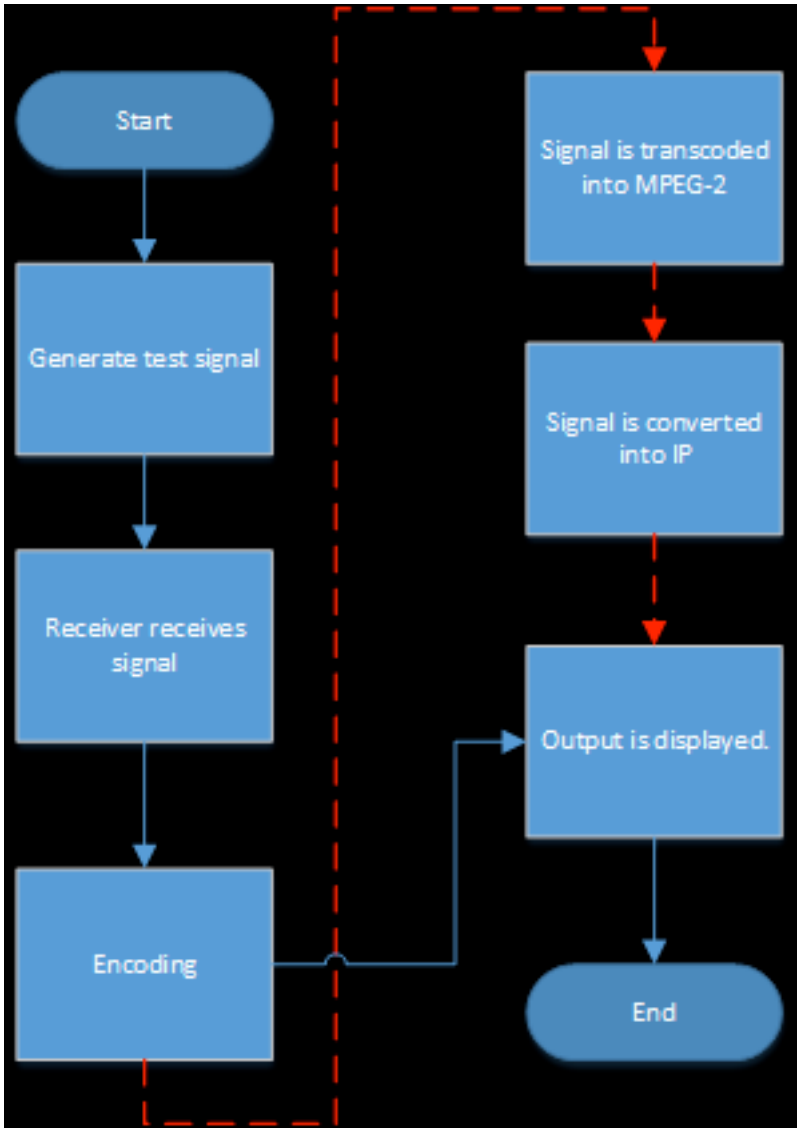


Figure 4.1: A diagram indicating the process flow of the system. The signal transcoding into MPEG-2 and conversion into IP was not attempted for this semester, however this would the normal behavior of a cable headend system.

4.2 Results

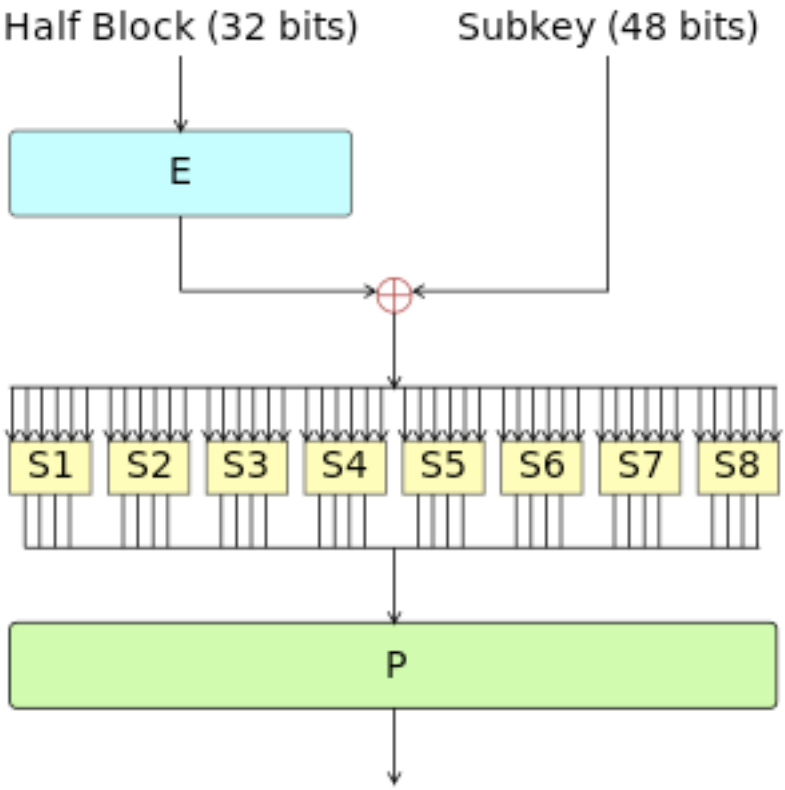


Figure 4.2: Visual depiction of the DES decryption algorithm. A variation of this algorithm called Triple DES is the industry standard and was in the process of being virtualized at the end of this semester. [15]

While at this time the encoder remains incomplete, the other team members were able to verify the operation of their components.

The FPGA board:
The Quartus software indicates that to generate the signal generation and virtual receiver the Altera board only used 5% of its available gates. This includes 2% containing the virtual soft-core processor running the C++ program.

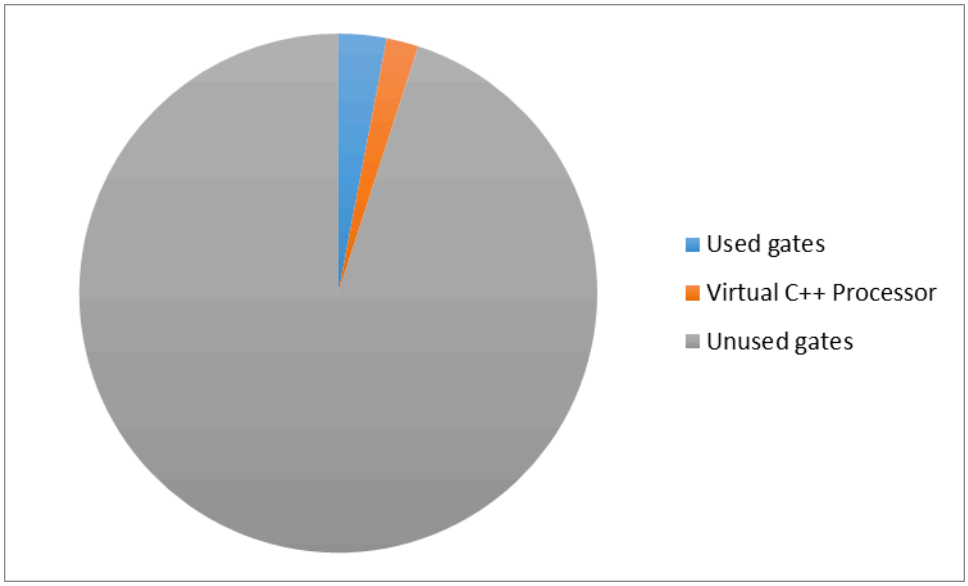


Figure 4.3: How many gates were used in the FPGA with only a test signal generator and signal receiver virtualized, and the decryption program running.

The Decryption Program:
The decryption program was compared against test data from an actual signal from a cable headend system from GCI. This data indicated that splitting the input signal in two resulted in a decrease of 13% in decryption speed.

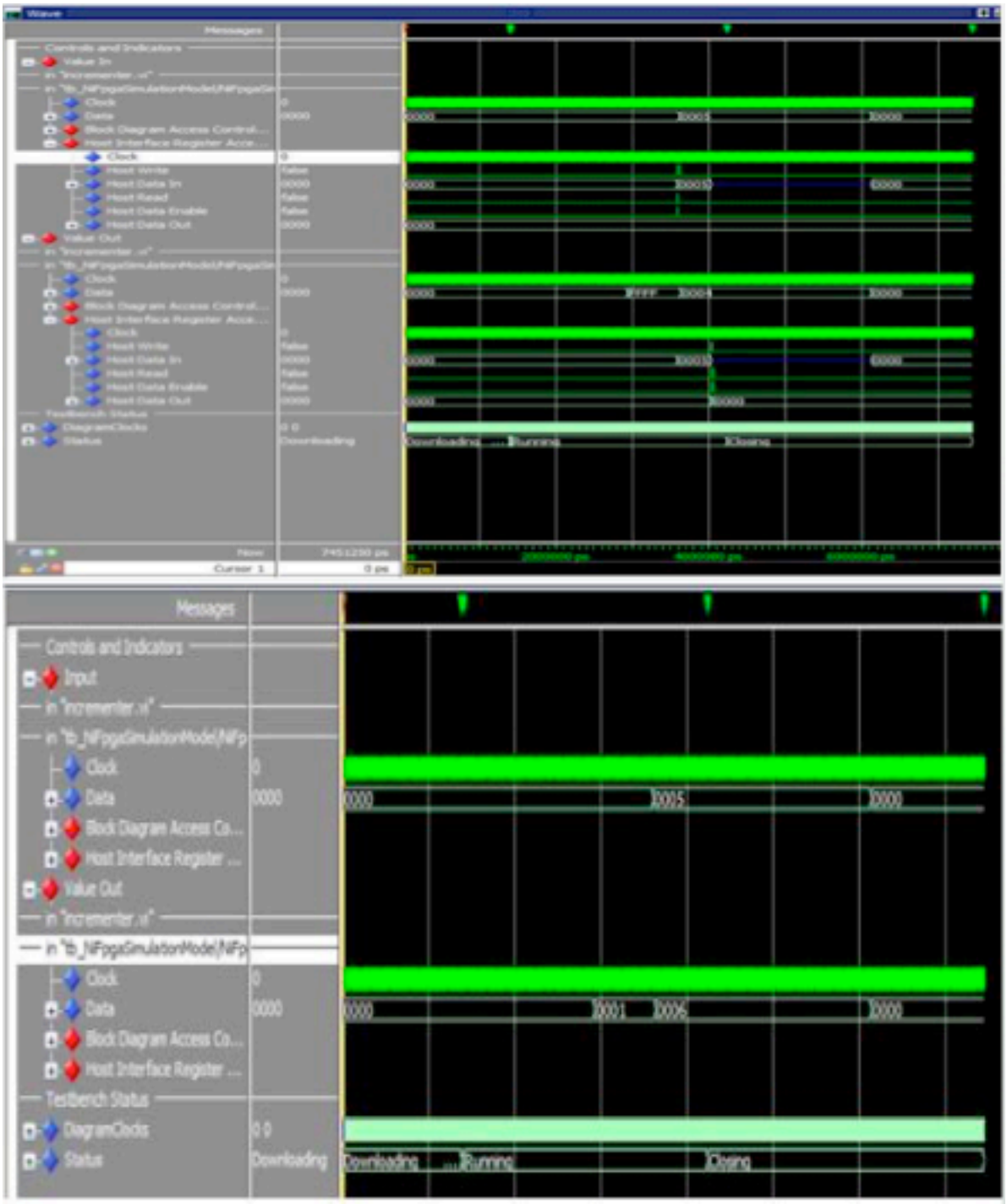


Figure 4.4: Data from the C++ processor showing an unsplit signal on the bottom, and a split signal on the top.

4.3 Discussion

While an entire 5-component headend device was not simulated in the course of this semester, initial data suggests that this will be a feasible project worth continued study. With only 5% of the gates being used in the Cyclone IV, this leaves plenty of room for additional modules of the headend system to be virtualized as well. It may even be possible to implement multiple headend devices in a single FPGA system depending on the needs of the transcoding and IP modules.

Chapter 5
Summary and Conclusion

5.1 Implications

The ability to virtualize the hardware in a cable headend system has the potential to make cable television delivery more efficient. By virtualizing multiple components onto a single board the amount of needed hardware is reduced. This also increases the flexibility of cable headend stations that no longer need specialized hardware configured to deliver only a single specific channel. Furthermore, compacting the functionality of multiple devices into one virtualized construct could reduce the latency between as many as five separate machines.

This added efficiency reduces costs to cable providers. This could translate both into higher profits for cable providers, and/or into reduced costs for cable subscribers.

5.2 Recommendations for future development

This is hopefully the first of many semesters that students will develop this technology. There are two other components that need to be developed. These include the transcoding of the signal into an MPEG-2 video format, and converting that video into IP packets.

While we were forced to use a Cyclone IV because of changes to international import regulations, it is recommended that interested students purchase an Arria V well in advance of the academic semester. The Arria V FPGA unit comes equipped with a dedicated co-processor which would allow the user to run programs and interface directly from the board.

Also, a better system for testing the integration of these separate technologies is recommended.

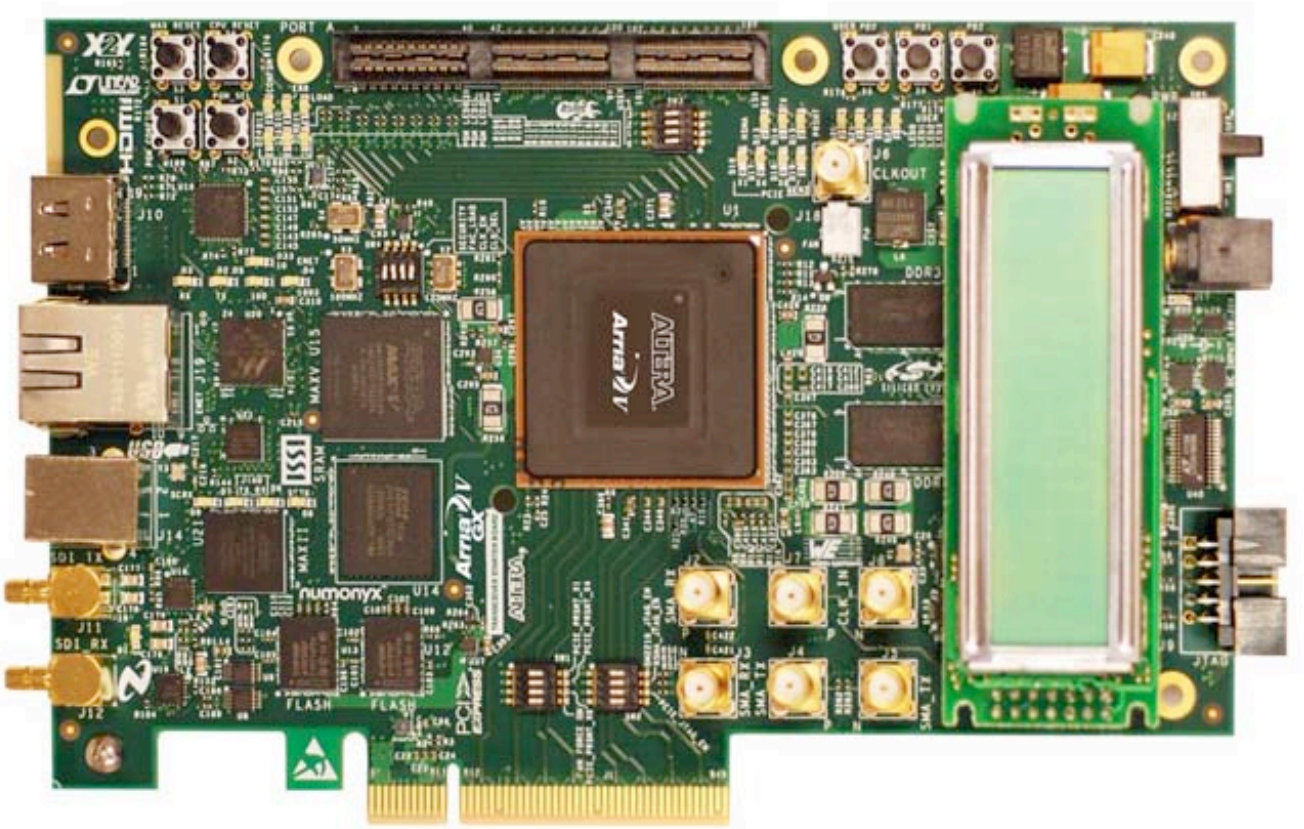


Figure 5.1: An Arria V FPGA board. [16]

5.3 Conclusion

The project was successful in demonstrating that cable headend virtualization has potential. By demonstrating that only 5% of the FPGA gates are needed to replace two out of the five boxes that are currently used, we have proven that the technology works, and that the technology would be an improvement over the current system.

Appendix A

Top module for encoder:

```
`timescale
1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    20:10:26 11/29/2014
// Design Name:
// Module Name:    top
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
`include "firstchange.v"
`include "endchange.v"
`include "subkey_generate_better.v"
`include "Encryption.v"
module top(
    //input [64:1] key,
    input [64:1] data_in,
    //input mode,
         input clk,
        //input rst_n,
    output [64:1] data_out
        );
wire [64:1] key ;
assign key = 64'h0001;
wire mode;
assign mode = 1'b1;
wire rst_n;
assign rst_n = 1'b1;
wire [64:1] fstchgedout;
```

```
wire [56:1] fstchgekout;
firstchange
fstchge_u1(.key(key),.data_in(data_in),.clk(clk),.rst_n(rst_n),.firstchangedout(fstchgedout

wire [48:1] k1,k2,k3,k4,k5,k6,k7,k8,k9,k10,k11,k12,k13,k14,k15,k16;
subkey_generate_better
subkey_generate_u2(.key_in(key),.mode_in(mode),.subkey_out1(k1),.subkey_out2(k2),.subkey_ou

.subkey_out5(k5),.subkey_out6(k6),.subkey_out7(k7),.subkey_out8(k8),.subkey_out9(k9),.subke

.subkey_out11(k11),.subkey_out12(k12),.subkey_out13(k13),.subkey_out14(k14),.subkey_out15(k

wire [32:1] L2,R2,L3,R3,L4,R4,L5,R5,L6,R6,L7,R7,L8,R8,L9,R9,L10,R10,L11,R11,L12,R12,L13,R13
Encryption f1
(.L(fstchgedout[64:33]),.R(fstchgedout[32:1]),.subkey(k1),.Encryp_clk(clk),.Encryp_rst_n(rs
Encryption f2 (.L(L2) ,.R(R2) ,.subkey(k2) ,.Encryp_clk(clk),.Encryp_rst_n(rst_n),.new_R(R3
Encryption f3 (.L(L3) ,.R(R3) ,.subkey(k3) ,.Encryp_clk(clk),.Encryp_rst_n(rst_n),.new_R(R4
Encryption f4 (.L(L4) ,.R(R4) ,.subkey(k4) ,.Encryp_clk(clk),.Encryp_rst_n(rst_n),.new_R(R5
Encryption f5 (.L(L5) ,.R(R5) ,.subkey(k5) ,.Encryp_clk(clk),.Encryp_rst_n(rst_n),.new_R(R6
Encryption f6 (.L(L6) ,.R(R6) ,.subkey(k6) ,.Encryp_clk(clk),.Encryp_rst_n(rst_n),.new_R(R7
Encryption f7 (.L(L7) ,.R(R7) ,.subkey(k7) ,.Encryp_clk(clk),.Encryp_rst_n(rst_n),.new_R(R8
Encryption f8 (.L(L8) ,.R(R8) ,.subkey(k8) ,.Encryp_clk(clk),.Encryp_rst_n(rst_n),.new_R(R9
Encryption f9 (.L(L9) ,.R(R9) ,.subkey(k9) ,.Encryp_clk(clk),.Encryp_rst_n(rst_n),.new_R(R1
Encryption f10(.L(L10),.R(R10),.subkey(k10),.Encryp_clk(clk),.Encryp_rst_n(rst_n),.new_R(R1
Encryption f11(.L(L11),.R(R11),.subkey(k11),.Encryp_clk(clk),.Encryp_rst_n(rst_n),.new_R(R1
Encryption f12(.L(L12),.R(R12),.subkey(k12),.Encryp_clk(clk),.Encryp_rst_n(rst_n),.new_R(R1
Encryption f13(.L(L13),.R(R13),.subkey(k13),.Encryp_clk(clk),.Encryp_rst_n(rst_n),.new_R(R1
Encryption f14(.L(L14),.R(R14),.subkey(k14),.Encryp_clk(clk),.Encryp_rst_n(rst_n),.new_R(R1
Encryption f15(.L(L15),.R(R15),.subkey(k15),.Encryp_clk(clk),.Encryp_rst_n(rst_n),.new_R(R1

/*notice the last time sbox change specially*/
Encryption f16(.L(L16),.R(R16),.subkey(k16),.Encryp_clk(clk),.Encryp_rst_n(rst_n),.new_R(L1

endchange
endchange_u3(.endchange_L(L17),.endchange_R(R17),.endchange_clk(clk),.endchange_rst_n(rst_r

endmodule
```

GitHub Repository

https://github.com/reflectiveLeprechan/hardwarevirtual

References

[1] D. Harris, "Microsoft is building fast, low-power neural networks with FPGAs," 23 February 2015. [Online]. Available: https://gigaom.com/2015/02/23/microsoft-is-building-fast-low-power-neural-networks-with-fpgas/. [Accessed 27 March 2016].

[2] Cisco, "Cisco IPTV Head-end Solution," [Online]. Available: http://www.cisco.com/c/en/us/solutions/service-provider/iptv-head-end-solution/index.html#~architecture. [Accessed 27 March 2016].

[3] "Jon Powell & Associates, Inc.," [Online]. Available: http://jpa1.com.

[4] Altera, "ARRIA 10 FPGA AND SOC," [Online]. Available: https://www.altera.com/products/fpga/arria-series/arria-10/overview.html. [Accessed 28 March 2016].

[5] USENIX Technical Program, "USENIX Technical Program - Paper - Smartcard 99," 18 March 2002. [Online]. Available: https://www.usenix.org/legacy/events/smartcard99/full_papers/kommerling/kommerling_html/. [Accessed 28 March 2016].

[6] Richard Barry, "freeRTOS," [Online]. Available: http://www.freertos.org/portmicroblaze.html.

[7] Altera Inc, "Altera.com," 9 February 2016. [Online]. Available: https://www.altera.com/en_US/pdfs/literature/hb/qts/qts-qps-handbook.pdf. [Accessed 24 April 2016].

[8] S. Butler, "AgileSamuraiPrinciples A," Anchorage, 2015.

[9] S. Butler, "AgileSamuraiPrinciples B," Anchorage, 2015.

[10] Cisco Inc, "Cisco Digital Headend Solution," 2009.

[11] Stack Exchange Inc, "Why Xilinx ISE can't get access to my custom package?," 30 September 2013. [Online]. Available: http://electronics.stackexchange.com/questions/83889/why-xilinx-ise-cant-get-access-to-my-custom-package. [Accessed 26 April 2016].

[12] Altera Corp, [Online]. Available: http://quartushelp.altera.com/14.0/mergedProjects/optimize/image/ape_alm_schematic_view.gif. [Accessed 26 April 2016].

[13] Xilinx Inc, "Accumalator 4 bit test bench problem!," 20 April 2010. [Online]. Available: https://forums.xilinx.com/t5/General-Technical-Discussion/Accumalator-4-bit-test-bench-problem/td-p/67695. [Accessed 25 April 2016].

[14] "Fundamentals of Object Oriented Programming," [Online]. Available: http://oopbook.com/junit-testing/junit-testing-in-netbeans/. [Accessed 25 April 2016].

[15] Wikipedia, "Data Encryption Standard," [Online]. Available: https://en.wikipedia.org/wiki/Data_Encryption_Standard. [Accessed 26 April 2016].

[16] Altera Corporation, [Online]. Available: https://www.altera.com/products/boards_and_kits/dev-kits/altera/kit-arria-v-starter.html. [Accessed 26 April 2016].