Howard University
College of Engineering, Architecture and Computer Science
Department of Electrical and Computer Engineering

A TECHNICAL REPORT ON HARDWARE TROJAN DETECTION

Written by:

Darren Earle

Amanuel Getahun

Taylor White

Advisor: Hassan Salmani, PhD

April 28, 2016

Executive Summary

This report presents a method that accurately detects hardware Trojans in computer chip's integrated circuit. Two solution approaches were developed and compared based off of how easy it was to use and repeat the method, how cost effective it was, its ability to maintain the integrity of the system, and how accurate it was in detecting hardware Trojans. The first solution approach uses heat dissipation analysis and the second approach uses timing analysis. Ultimately, it was decided that timing analysis was the better approach because it was easier to use, more cost effective, and more accurate.

TABLE OF CONTENTS

1.0 Introduction

Hardware Trojans are malicious modifications to a circuit that can lead to security breaches or random malfunctions in an electronic device. This report presents two possible solution approaches to detect hardware Trojans. In determining the best solution the following criteria was taken into consideration: the method developed must be easily repeatable, cost effective, keep the integrity of the system, and have accurate detection results. In addition this report will further explain the negative affects hardware Trojans have on an integrated circuit; the selected solution approach, how it was tested, evaluated, and implemented; the project goals; conclusions; and recommendations for future works.

2.0 Project Overview

2.1 Background

Over the past decade security has been a major priority in the United States. From a technological standpoint the U.S. has been on constant alert trying to prevent any intruders from getting into the hardware of computers and phones and leaking any valuable information they find into the wrong hands. These “intruders” are called hardware Trojans and they hide in the integrated circuits of common electronic devices going virtually undetected, which is what makes them so dangerous. Different than their better-known cousins Software Trojans, these Trojans specifically attack the integrated circuit of a computer chip such as a microprocessor, which can be found in computers, cell phones and smart devices. A hardware Trojan is made out of transistors, just like all integrated circuits, and these transistors can be arranged to make logic gates such as AND gates, OR gates or NOT gates, to name a few. Depending on where they’re placed in the circuit and what their intentions are, they may or may not disrupt the functionality of the chip. These Trojans can alter the chip so that it can fail at a crucial time or generate false signals (Mitra, Wong, Wong, 2015). However, the problem arises when the Trojans don’t ever disrupt the chips functionality and they go completely unnoticed, quietly leaking personal information. Hardware Trojans impact the “fabless” semiconductor companies, like AMD, who have to send their chips to chip manufacturing companies to be fabricated, or made. Some chip manufacturing companies, or people within those companies, have malicious intentions and will add one or more hardware Trojans to the chip by strategically placing it into the chip's integrated circuit. Once the chip is fabricated and sent back to the semiconductor company, the hardware Trojan can easily go undetected during their verification process because the modification to the circuit is so small and insignificant. Therefore, these "fabless" companies won't be able to tell if there is an intruder lurking around in their chips' circuits. Once these chips are placed into electronic devices they are shipped out and sold to the consumer. If a consumer has a chip with a hardware Trojan in it, their valuable information can potentially be leaked into the wrong hands. An accurate hardware Trojan detection method needs to be developed so "fabless" companies won't have to deal with their chips being tampered with and so consumers' private information can remain secured.

2.2 Problem Statement

A hardware Trojan is a malicious modification to the circuitry of an integrated circuit. These Trojans can be used to disable or destroy chips and its components as well as bypass or disable the security measures of a system.

2.3 Current Status of Art

Since hardware Trojans are relatively new any detection techniques or methods that are out there are currently in the research phase. Therefore, there is no industry-wide hardware Trojan detection method that is being currently used. The closest "technique" that prevents hardware Trojans is the United States Department of Defense's Trusted Foundry Program. There is a list of foundries that have been accredited by the DOD to work on chips that are deemed "trusted", or without Trojans. However, these foundries are producing chips that are 10 or more years behind the current manufacturing process and they're only focused on chips for military applications. This leaves many other nonmilitary applications susceptible to hardware Trojans. Therefore, the DOD's foundry program doesn't affectively solve the problem at hand.

3.0 Design Requirements

In order for the hardware Trojan detection method we develop to be effective we need it to be able to detect any Trojans within an integrated circuit, even the smallest of ones. Therefore, we came up with the following criteria based off of the industry's needs to help us determine which solution approach would be best to implement.

- **Easy to use**
    - Easily repeatable, systematic
- **Cost effective**
    - Affordable
- **Keeps the integrity of system (chip)**
    - Withholds system's functions
- **Accurate Detection**
    - Reliable
    - Able to detect small Trojans

4.0 Solution Approaches

4.1 Approach 1: Heat Dissipation Analysis

Today's circuits consist of millions of transistors; the more transistors there are the more complex a circuit becomes, and the more heat that is generated within that circuit. This is due to the fact that more energy is required to power the circuit. Therefore, power and heat are directly proportional in circuits; the more complex a circuit is, the more power that is needed to drive the overall system, the more power that is needed to drive the overall system, the more heat the system generates. Since hardware Trojans are the addition of these transistors, it can be inferred that more heat will be generated within a circuit. By using heat dissipation analysis we should be able to determine whether or not there is a Trojan within a circuit. We will use an Infrared (IR) camera to get the heat maps of two FPGA boards. Both boards will have identical circuits; however, one will have a hardware Trojan, an extra logic gate, while the other one will be Trojan free. We will compare the heat maps of both boards to see if the circuit with the Trojan will produce significantly more heat than the Trojan free circuit.

4.2 Approach 2: Timing Analysis

Integrated circuits consist of many logic gates. It takes data a certain amount of time to flow through the input and output of each logic gate, this is known as propagation delay. The clock signal is what synchronizes the propagation of data in an integrated circuit. Data can flow down many different paths within an integrated circuit; these paths are called net paths. Each net path has a minimum amount of time in which data can propagate, meaning data has to propagate through that net path in a certain amount of time. If that minimum time is ever exceeded, it will slow down the system’s functionality, thus making the system aware that something is wrong. When a hardware Trojan is placed into a net path in an integrated circuit it can add more time to that path and possibly exceed its minimum amount of time needed to propagate data. By using timing analysis on Xilinx ISE we will get the timing of two circuits, one with a Trojan, and one that will be Trojan free. We will then compare the two circuits to see if there is any difference in their timing delays.

5.0 Design Selection

By using our design requirement criteria we narrowed down our options to find the best solution approach that will solve the problem based on our decision matrix, which can be found in Appendix 1.

5.1 Selected Approach: Timing Analysis

We decided to go with Timing Analysis as the selected solution approach because it meets the needs of the industry; the method will be easy to use, cost effective, keep the integrity of the system and will accurately detect all additions made to the circuit no matter how small. In comparison Heat Dissipation Analysis is only able to maintain the integrity of the system, but it’s not easy to use because of the IR camera, it’s not cost effective due to the price of the IR camera and the FPGA boards, and it’s way too vulnerable to inaccuracies such as the room temperature possibly affecting the results and the camera’s inability to detect small additions within a circuit.

6.0 Implementation, Testing, and Evaluation

6.1 Implementation

Our decision to select and use the timing analysis approach inevitably lead us to measuring the timing paths of two identical circuits, where one had a hardware Trojan and the other was Trojan-free. We were most concerned with the short paths of the circuit because these are the only paths where a hardware Trojan can be successfully integrated. The reason being that if it were implemented in any other path it would easily violate the clock frequency making it known to the system that a Trojan is there, thus preventing the Trojan from carrying out its malicious intentions. Since Trojans are dependent on short paths, and the clock is what defines short paths, we kept track of all the clock-dependent elements, i.e. flip-flops, and short paths in the circuit that we tested. We did this by coding two Python programs; the “Flip-Flop Finder” found all of the flip-flops in the circuit, how many there were, and their inputs and outputs while the “Short Path Finder” found all of the short paths in the circuit based on the fact that short paths are defined to be any path within an integrated circuit that is less than or equal to 75% of the minimum clock period. After all of the flip-flops and short paths were identified, we decided that the best approach to ensure that the short paths would not be fouled was to realize these paths altogether as one long path by bypassing the flip-flops that were connected to the short paths only. The reason behind this decision was that short paths’ timing delays could be significantly smaller than the minimum clock period. This becomes problematic when “fabless” semiconductor companies need to measure the short paths’ timing delays during the verification process after fabrication, since measuring each small short path becomes impractical and potentially inaccurate. By combining the paths into one long path, they only need to be concerned with measuring the combined path during the verification process and compare that timing delay, chip after fabrication with the potentially Trojan-containing circuit, with their expected delay, chip before fabrication with ideal their ideal circuit time delay.

In order to combine the short paths together while also withholding the integrity of the circuit, a multiplexer (MUX) was used. The functionality of a multiplexer allowed us to choose between either concatenating the short paths by bypassing the clock-dependent elements, or maintain the circuit's normal operation. Specifically, a 2-to-1 multiplexer, two inputs and one output, was used between each short path that was found. One input to the multiplexer was straight from the combinational output of a short path, while the other input was from the output of the flip-flop. A detailed schematic of this connection can be found in Appendix 2. This was done to allow for normal operation of the circuit, "Functionality mode", and to check for Trojans in the circuit, "Test mode". The selector, or "Authenticator" is what determined the mode the circuit would be in and therefore what input was to be outputted. When "Functionality mode" was toggled, the multiplexer allowed the data to flow through the flip-flop to the next combinational circuit, allowing for normal circuit behavior to continue. When "Test mode" was toggled, each multiplexer sent the output of the short path's combinational circuit through to its output, which was connected to the input of the next short path's combinational circuit, thus combining all of the short paths in the circuit into one long path.

6.2 Testing

We used timing analysis to compare the timing delays of the circuit with and without a Trojan. We placed the Trojan in a short path, which was found by the "Short Path Finder" program, because that's where hardware Trojans are placed in the real-world. We compared the two circuits while they were in "Test mode" to see that the timing delays were different, proving that our method worked and the Trojan was detected.

6.3 Evaluation

As expected when the circuit with the Trojan and the Trojan-free circuit were in "Test mode" their timing delays were different. By adding the MUX to bypass the flip-flops that were connected to the short paths we were able to concatenate the short paths and turn them into one long path, making it easier for the Trojan to be identified. By comparing the timing delays of both circuits in "Test mode" we were able to see that the timing delay in the circuit with the Trojan was longer than the timing delay in the Trojan-free circuit. Evaluation results can be found in Appendix 3.

7.0 Project Deliverables

7.1 Project's Spring 2016 Target Goal

Our goal for Spring 2016 is to design, develop, and implement a method/technique that would accurately detect/prevent hardware Trojans in an integrated circuit that will satisfy the needs of the industry.

7.2 Project's Final Goal

Part of our final goal to go along with the actual detection method is to develop a medium to implement this exact idea; realizing a circuit, running timing analysis to determine short paths and clock-dependent elements connected to those short paths, implementing a multiplexer to combine the short paths by bypassing those clock-dependent elements, and running timing analysis on the combined path. The medium we decided to develop was a Python program that automates this process. The program will prompt the user for circuit specifications such as input names, clock names, etc. After the user provides the necessary information, the program will run timing analysis and perform the multiplexer implementation around the necessary short paths in order to combine them. Afterwards, the program will have the measured timing delay of the concatenated short paths.

8.0 Conclusion

Two solution approaches to detect hardware Trojans were presented in this report. The first solution approach was to use heat dissipation analysis and compare the heat maps of two FPGA boards to see if the circuit with the Trojan produced more heat than the Trojan free circuit. The second solution approach was to use timing analysis and compare the timing delays of two circuits to see if the circuit with a Trojan had a longer timing delay than the Trojan-free circuit. We decided to use the timing analysis solution approach instead of the heat dissipation analysis because it met all of the needs of our customer the industry; it was easy to use, cost effective, kept the integrity of the system and able to accurately detect any small additions to the circuit. By implementing a 2-to-1 MUX with a “Functionality mode” and “Test mode” we were able to let the circuit operate normally in one mode and bypass all of the clock dependent elements in the circuit that were connected to the short paths and concatenate those short paths to turn it into one long path to make it easier for a Trojan to be identified in the other mode. The simple addition of the MUX to a circuit makes our method easy to use, cost effective, able to keep the integrity of the system, and able to accurately detect small Trojans. Our hardware Trojan detection method would be very helpful in the real world because it’s simple, repeatable, and accurate.

9.0 Recommendations for Future Works

Our team has learned many lessons throughout this project. Below is a bulleted list of what we advise the future members of this project to do in order to be more effective as a team and accomplish their goals.

- **Divide tasks among members based on their strengths**
    - Increases the likelihood of the task being completed on time or ahead of time
    - Members would be able to help others on tasks if they finished early.
- **Give enough room to make mistakes**
    - “Pad” your deadlines, meaning give yourselves more than enough time to accomplish your tasks
        - If you think you can complete something in a week, give yourself 2 weeks, just in case the unexpected happens.
    - This will help to avoid pushing back deadlines
- **More team meetings**
    - Will help you to stay on track with meeting the deadlines

10.0 References

Subhasish Mitra, H.-S. Philip Wong, and Simon Wong. "Stopping Hardware Trojans in Their Tracks." *IEEE Spectrum*. 20 Jan. 2015. Web.

"Index of /~maksim/benchmarks/iscas99/vhdl." *Index of /~maksim/benchmarks/iscas99/vhdl*. Web.

Appendix 1: Decision Matrix

Decision Matrix

	Heat Dissipation Analysis	**Timing Analysis**
Ease of use	No prior experience with IR Camera.	Xilinx ISE will be used to design the circuit and to run timing analysis. It is a program we're already familiar with and is used as a reliable tool in the industry.
Cost Effective	Tools are expensive: IR Camera \$400-\$500 2 FPGA Board \$120	Tools are essentially free. (Xilinx ISE)
Integrity of System	System's functionality is maintained	System's functionality is maintained
Detection Accuracy	Vulnerable to inaccuracies: Room temperature. Unable to detect small additions in a circuit.	Able to pick up on the smallest additions to a circuit.

Appendix 2: Schematic of MUX in Circuit

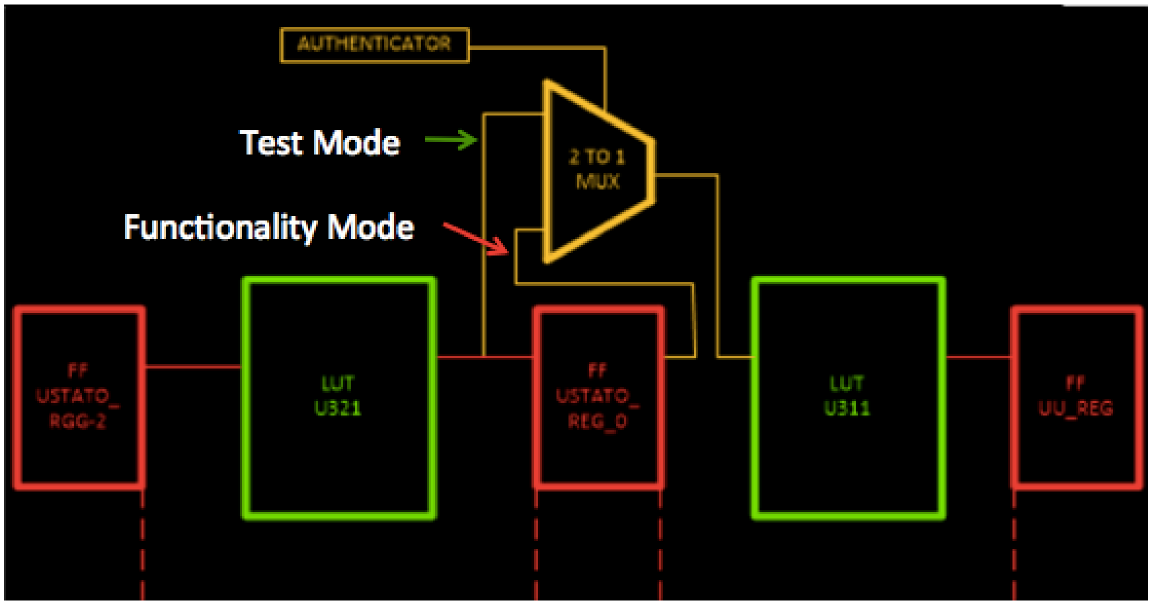
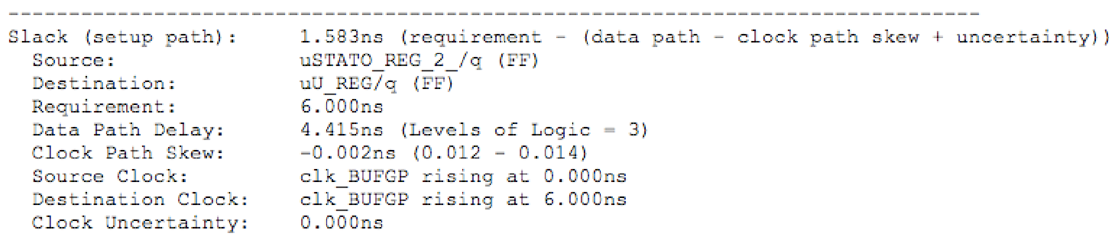


Figure 1: Schematic of MUX in circuit

In Figure 1, the “Authenticator” is the selector that selects the path that will be outputted. The green arrow is pointing to the path that the MUX will output when “Test Mode” is selected. The red arrow is pointing to the path that the MUX will output when “Functionality Mode” is selected. It can be seen in Figure 1 that the selected path in “Test Mode” bypasses the flip-flop, as intended, and goes into the input of the next circuit element. It can also be seen that the selected path in “Functionality Mode” goes through the flip-flop and then goes into the input of the next circuit element, the way it normally functions.

Appendix 3: Evaluation Results

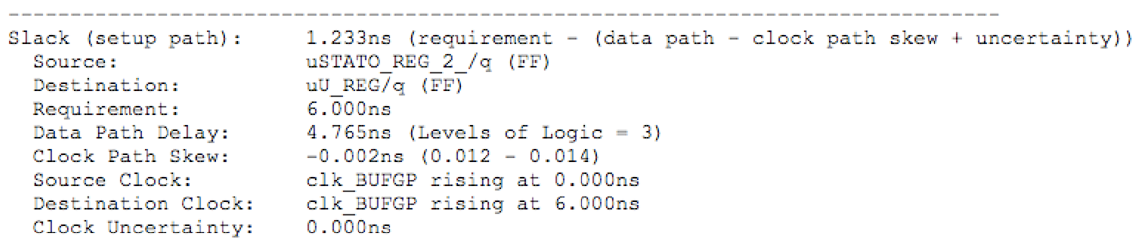
Trojan Free Circuit in “Test mode”



```
--------------------------------------------------------------------------------
Slack (setup path):     1.583ns (requirement - (data path - clock path skew + uncertainty))
  Source:               uSTATO_REG_2_/q (FF)
  Destination:          uU_REG/q (FF)
  Requirement:          6.000ns
  Data Path Delay:      4.415ns (Levels of Logic = 3)
  Clock Path Skew:      -0.002ns (0.012 - 0.014)
  Source Clock:         clk_BUFGP rising at 0.000ns
  Destination Clock:    clk_BUFGP rising at 6.000ns
  Clock Uncertainty:    0.000ns
```

The timing delay in the Trojan-free circuit is 4.415ns.

Circuit with Trojan in “Test mode”



```
--------------------------------------------------------------------------------
Slack (setup path):     1.233ns (requirement - (data path - clock path skew + uncertainty))
  Source:               uSTATO_REG_2_/q (FF)
  Destination:          uU_REG/q (FF)
  Requirement:          6.000ns
  Data Path Delay:      4.765ns (Levels of Logic = 3)
  Clock Path Skew:      -0.002ns (0.012 - 0.014)
  Source Clock:         clk_BUFGP rising at 0.000ns
  Destination Clock:    clk_BUFGP rising at 6.000ns
  Clock Uncertainty:    0.000ns
```

The timing delay in the circuit with the Trojan is 4.765ns.

The timing difference is 4.765ns – 4.415ns = **0.350ns**

Because there is a 0.350ns difference in timing between the two circuits, this proves that our hardware Trojan detection method works.

Source Code Listing

Flip-Flop Finder Program

```
import sys
import re

filename=input("Enter File Name(Must be .txt): ")

file=open("C:\\Users\\amanuel\\Desktop\\"+filename+".txt")
f=open("C:\\Users\\amanuel\\Desktop\\"+filename+".txt")
g=open("C:\\Users\\amanuel\\Desktop\\"+filename+".txt")
a=open("C:\\Users\\amanuel\\Desktop\\"+filename+".txt")
b=open("C:\\Users\\amanuel\\Desktop\\"+filename+".txt")
c=open("C:\\Users\\amanuel\\Desktop\\"+filename+".txt")
m=open("C:\\Users\\amanuel\\Desktop\\"+filename+".txt")
p=open("C:\\Users\\amanuel\\Desktop\\"+filename+".txt")
y=open("C:\\Users\\amanuel\\Desktop\\"+filename+".txt")

inputs=[]                      #list for imputs
outputs=[]                    #list for outputs
signals=[]                     #list for signals
constants=[]                    #list for constants
variables=[]                    #list for variables
ff_inputs=[]                   #list for flipflop inputs
ff_outputs=[]                  #list for flipflop outputs
allinfo=[]
temp=[]
count_list=[]

#finds main inputs/outputs
for line in file:
    ent=re.search(r"entity\s*(\w+)\s*is\s*",line)
    end=re.search(r"end\s*(\w+)",line)
    if ent != None:
        for line in file:
           one=1
           count_c=0
           componenet=re.search(r"\s*(\w+)\s*:\s*(\w+)\s+(\w+\(?.*\)?)",line)
           if componenet != None:
              if componenet.group(2) == "in " :
                   inputs.append(componenet.group(1))
              elif componenet.group(2)== "out ":
                   outputs.append(componenet.group(1))
              if componenet.group(3)!= None:
                  typ=componenet.group(3)
                  match1 =
re.search(r"(\w+)\s*\(\s*(\d+)\s+(downto|to)\s+(\d+)\s*\)",typ)
                  if match1 != None:
                     if match1.group(3) == "downto" :
                        range=int(match1.group(2)) - int(match1.group(4))+1
                        com_val={componenet.group(1):range}
                        for key, value in com_val.items():
                            allinfo.append(key)
                            allinfo.append(value)
                        while count_c<range:
                          print(componenet.group(2)+ "put "+ "Name:"  +
componenet.group(1) +"<"+str(count_c)+">" +" Type: " +match1.group(1) + " Size:
" + str(range))
                          count_c=count_c+1

                  else :
                    range=1
                    com_val={componenet.group(1):range}
                    for key, value in com_val.items():
                            allinfo.append(key)
```

```
                               allinfo.append(value)
                     print(componenet.group(2)+ "put "+ "Name:"  +
componenet.group(1) +" Type: " +componenet.group(3).strip(";") + " Size: " +
str(range))

              if end!=None:                    #stops at the end of entity
                  break

for line in f:
            count_s=0
            signal=re.search(r"signal\s+(\w+)\s*:\s*(\w+)\s*(\w+\(?.*\)?)",line)
            if signal!=None:
                signame=signal.group(1)
                signals.append(signame)
                sigtype=(signal.group(2)+signal.group(3))
                match2 =
re.search(r"(\w+)\s*\(\s*(\d+)\s+(downto|to)\s+(\d+)\s*\)",sigtype)
                if match2 != None:
                     if match2.group(3) == "downto" :
                        range=int(match2.group(2)) - int(match2.group(4))+1
                        com_val={signame:range}
                        for key, value in com_val.items():
                            allinfo.append(key)
                            allinfo.append(value)
                        while count_s<range:
                           print("Signal "+ "Name:"  + signame +"<"+
str(count_s) + ">" +" Type: " +match2.group(1) + " Size: " + str(range))
                           count_s=count_s+1

                else :
                    range=1
                    com_val={signame:range}
                    for key, value in com_val.items():
                            allinfo.append(key)
                            allinfo.append(value)
                    print("Signal "+ "Name:"  + signame + " Type: " +
(signal.group(2)+signal.group(3).strip(";")) + " Size: " + str(range))

for line in m:
    constant=re.search(r"constant\s+(\w+)\s*:\s*(\w+)\s*(\w+\(?.*\)?)",line)
    if constant!=None:
                constant_name=constant.group(1)
                constants.append(constant_name)
                contype=(constant.group(2))
                vall=constant.group(3).split(":=")
                valu=vall[1].split(";")
                value=valu[0]
                print("Constant "+ "Name:" + constant_name + " Type: "+ contype
+ " Value :" + str(value))
                const_val={constant_name:value}
                for key, value in const_val.items():
                            allinfo.append(key)
                            allinfo.append(value)

for line in p:
    variable=re.search(r"variable\s+(\w+)\s*:\s*(\w+)\s*(\w+\(?.*\)?)",line)
    if variable!=None:
        var_name=variable.group(1)
        variables.append(var_name)
        vartype=variable.group(2)

var_size=re.search(r"range\s*(\d+)\s+(downto|to)\s+(\d+)",variable.group(3))
        if var_size!=None:
            if var_size.group(2)=="downto":
                range=int(var_size.group(1))-int(var_size.group(3))
                var_val={var_name:range}
                print("Variable "+"Name:"+var_name+" Type:"+vartype+"
Range:"+str(range))
                for key, value in var_val.items():
                            allinfo.append(key)
                            allinfo.append(value)
```

```
for line in c:
        ff_count=0
        f_count=0
        ff=re.search(r"'event" or r"rising_edge",line)
        if ff != None:
            for line in c:
                ff_in_out=re.search(r"\s*(\w+)\s*<=\s*(\w+)",line)
                if ff_in_out != None:
                    ffoutput=ff_in_out.group(1)
                    ffinput=ff_in_out.group(2)
                    ff_inputs.append(ffinput)
                    ff_outputs.append(ffoutput)
                    if ffinput in allinfo:
                        val=int(allinfo.index(ffinput))
                        ff_size=allinfo[val+1]
                        ff_count=ff_size*1
                        count_list.append(ff_count)
                    while f_count<ff_count:
                       print("Flip Flop Input: "+ffinput+"<"+str(f_count)+">"+ "
Flip Flop Output: "+ffoutput+"<"+str(f_count)+">")
                       f_count=f_count+1

                if line.startswith("end process"):  #stops at the end of entity
                  break

            for line in y:
                case_statment=re.search(r"case\s+(\w+)\s*(\w+)",line)
                if case_statment != None:
                    ff_count=ff_count+1
                    count_list.append(ff_count)
                    case_name=case_statment.group(1)
                    for line in y:
                      when=re.search(r"when\s+(\w+)\s*",line)
                      if when != None:
                          case_is=when.group(1)
                          print(case_is)

                          for line in y:

cff_in_out=re.search(r"\s*(\w+)\s*<=\s*\S*(\w+\s*\(?.*\)?)\S*",line)
                              if cff_in_out != None:
                                  cff_output=cff_in_out.group(1)
                                  cf_input=cff_in_out.group(0).split("<=")
                                  cff_input=str(cf_input[1])
                                  print("When " + case_name + " is "+case_is+"
Flip Flop Input: "+cff_input + " Flop Flop Output: "+cff_output)
                                  when_case={case_name:case_is}
                                  ff_info={cff_input:cff_output}
                                  for key, value in when_case.items():
                                     allinfo.append(key)
                                     allinfo.append(value)
                                  for key, value in ff_info.items():
                                      allinfo.append(key)
                                      allinfo.append(value)
                                  if line.startswith("when"):
                                     break

                if line.startswith("end case"):                   #stops at the
end of entity
                  break

#print(allinfo)
#print(count_list)
print("Total # of Flip Flops: " + str(sum(count_list)))
```

Short Path Finder Program

```
from sys import argv
```

```
import re
import math
short_paths=[]
all_info=[]

x = open("C:/Users/Taylor/Documents/b01FullTimingAnalysis.txt","r")
y = open("C:/Users/Taylor/Documents/b01FullTimingAnalysis.txt","r")

#function for finding paths' information and the path total
def findPaths():
  netPaths_count = 0
  short_path_count=0
  for line in x:
        if 'Minimum period is' in line:
            minPeriod = line.split()                 #splits line at spaces
            minPeriod = minPeriod[3]                 #gets the min. period value
ns
            minPeriod = minPeriod[0:5]               #gets min. period value
without ns at end
            minPeriod = float(minPeriod)             #changes string into float
            minPeriodPercentage = (minPeriod * 0.75) #finds 75% of min. period
and put   #into minPeriodPercentage
            min_period_perc=(round(minPeriodPercentage, 3))
            print("Delay must be less than ", min_period_perc)        #print
rounded minPeriodPercentage value to 3 digits
  for line in y:
     source=re.search(r"(\s+)Source:\s+(\w+)s*\(?.*\)?",line)
     destination=re.search(r"(\s+)Destination:\s+(\w+)s*\(?.*\)?",line)
     data_path_delay=re.search(r"Data Path Delay:\s+(\d+\.\d+)",line)
     if source != None:
         sourcename=(source.group(2))   #sourcename
         s=source.group(0)              #entire line with source info
         s1=s.split("(")                #split group after "("   gets source
type
         s2=s1[1].split(")")            #split group after ")"   gets source
type
         stype=s2[0]                    #source type is stored into stype
         print(sourcename)              #prints source name
         print(stype)                   #print source type
         netPaths_count=netPaths_count+1
         all_info.append(sourcename)
         all_info.append(stype)
     if destination!=None:
         dest_name=(destination.group(2))

         d=destination.group(0)         #entire line with destination info
         d1=d.split("(")                #split group after "(" gets destination
#type
         d2=d1[1].split(")")            #split group after ")" gets destination
#type
         dtype=d2[0]                    #destination type is stored into dtype
         print(dest_name)               #prints destination name
         print(dtype)                   #prints destination type
         all_info.append(dest_name)
         all_info.append(dtype)
     if data_path_delay!=None:
         pathdelay=(data_path_delay.group(1))
         path_delay_value=float(pathdelay)
         all_info.append(path_delay_value)
         print(path_delay_value)
         if (path_delay_value<=min_period_perc):
            shortpath=path_delay_value
            short_paths.append(path_delay_value)
            short_path_count=short_path_count+1
            if path_delay_value in all_info:     #looks for short paths in list
               tfs=int(all_info.index(path_delay_value))  #gets index of the
#short path value in list
               src=tfs-4       #goes back four places within list to find
#source name of the short path
               des=tfs-2       #goes back 2 places within list to find
#destination name of the short path
```

```
            print("Short Path==>"+" Source Name:"+all_info[src]+" Destination
Name:"+all_info[des]+" Path Delay:"+str(shortpath))
     print("Source Name: " + sourcename+" Type: "+stype+" Destination name:
"+dest_name+" Type: "+dtype+" Path Delay: "+pathdelay)
  #print("Total Short Path: "+all_info[src]+" "+all_info[des]+"
"+str(shortpath))
  print("Total Path: "+str(netPaths_count))
  print("Total Short Path: "+str(short_path_count))
  print("All Short Path Delays: "+str(short_paths))
  print(all_info)

findPaths() #calls findPaths function
```