DCU School of Electronic Engineering Assignment Submission

Student Name(s):	**Michael Lenehan**
Student Number(s):	**15410402**
Programme:	**B.Eng in Electronic and Computer Engineering**
Module Code:	**EE496**
Lecturer:	**X. Wang**
Project Due Date:	**30/11/2018**

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited andacknowledged within the text of my work. I understand thatplagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references.

I have not copied or paraphrased an extract of any length from any source without identifying the source and using quotation marks as appropriate. Any images, audio recordings, video or other materials have likewise been originated and produced by me or are fully acknowledged and identified.

This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the referencing guidelines found at http://www.library.dcu.ie/citing&refguide08.pdf and/or recommended in the assignment guidelines.

I understand that I may be required to discuss with the module lecturer/s the contents of this submission.

I/me/my incorporates we/us/our in the case of group work, which is signed by all of us.

Signed: ***Michael Lenehan***

Assignment 2

Michael Lenehan

Purpose

The purpose of this assignment is to complete the design of a 16-bit cryptographic coprocessor. The design process is divided amongst the two sections of this assignment. Firstly, the combinational components of the design must be completed. These components include an ALU, capable of performing addition and subtraction operations, along with a number of logical operations, a Shifter, which can shift or rotate the input bits, and a Lookup module, which substitutes the input bits for a value specified in the provided lookup table.

The second section of this assignment focuses on the synchronous design elements, i.e. the memory registers used to store the values output from the combinational logic components. Initially both input and output register components must be createed and tested. Once working, A register file is used to model the memory of the completed system. The completed register file component must then be combined with the combinational logic components, giving the completed crypto coprocessor

Procedure

2.1 Combinational Logic Design

The Combinational Logic of the Crypto Coprocessor allows for the non-memory associated functionality of the coprocessor. The components required give the cryptographic functionality, such as the arithmetic and logical operations of the ALU.

2.1.1 ALU Design

The ALU design is completed according to the provided diagram and operations table found below:

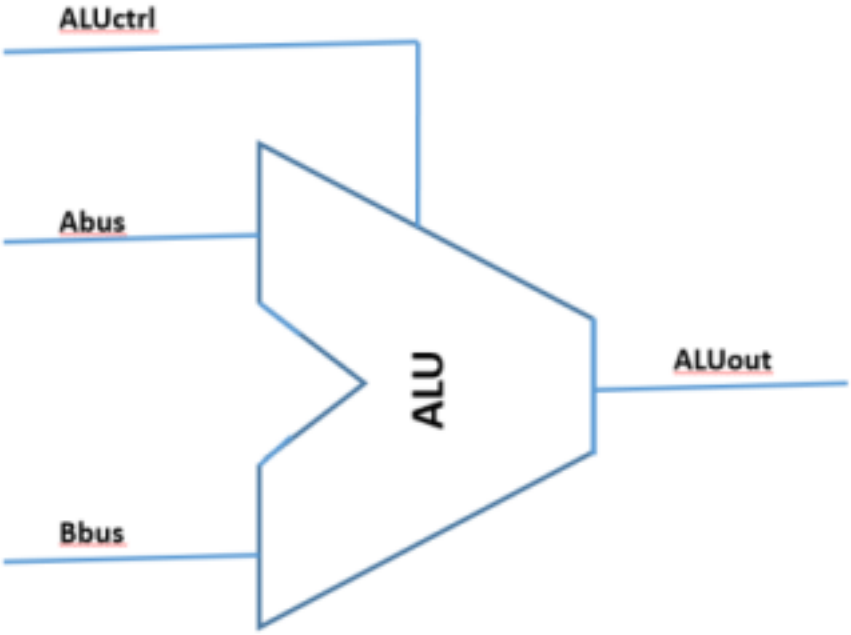

Figure 1: ALU Component

Table 1: ALU Operations

ALUctrl	Mnemonic	Function
0000	ADD	ALUout = Abus + Bbus
0001	SUB	ALUout = Abus - Bbus
0010	AND	ALUout = Abus & Bbus
0011	OR	ALUout = Abus | Bbus
0100	XOR	ALUout = Abus ^ Bbus
0101	NOT	ALUout = !Abus
0110	MOV	ALUout = Abus

The ALU takes three inputs; a control signal (ALUctrl), and two 16-bit input values (Abus, Bbus). An operation is performed on the 16-bit input values according to the operation specified by the control signal, with the output of the operation being output on the ALUout signal.

2.1.2 Shifter Design

The Shifter component is completed according to the provided diagram and operations table, found below:

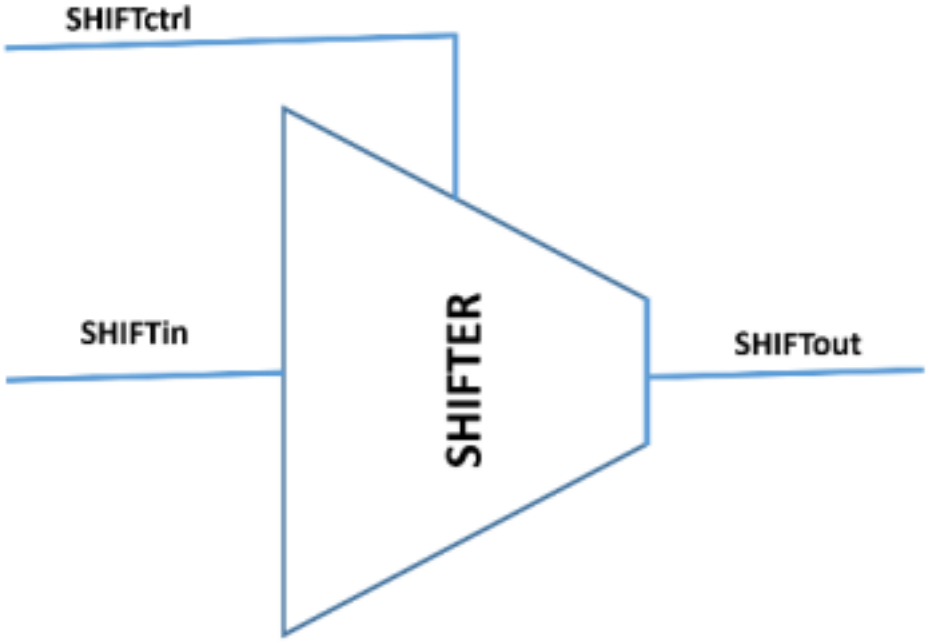

Figure 2: Shifter Component

Table 2: Shifter Operations

SHIFTctrl	Mnemonic	Function
1000	ROR4	Rotate Right 4 bits
1001	ROL4	Rotate Left 4 bits
1010	SLL4	Shift Left Logic 4 bits
1011	SRL4	Shift Right Logic 4 bits

The Shifter takes two inputs; a control signal (SHIFTctrl), and a 16-bit input value (SHIFTin). The shifter then performs an operation, as specified by the SHIFTctrl input, passing the output to the SHIFTout signal.

2.1.3 Lookup Component Design

The Lookup Table component is completed according to the diagram, and operation tables, found below:

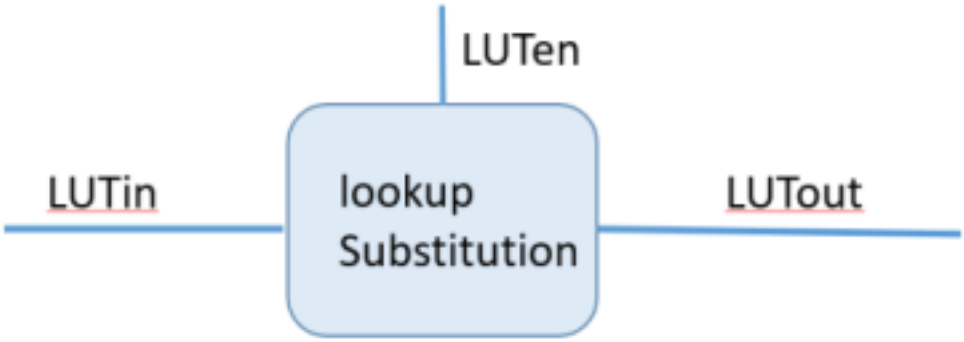

Figure 3: Lookup Table Component

The expanded diagram below displays the the index of the bits which may be passed through the component unchanged. Bits 7-4 are substituted as specified in S-Box 1, with bits 3-0 substituted as specified in S-Box 2.

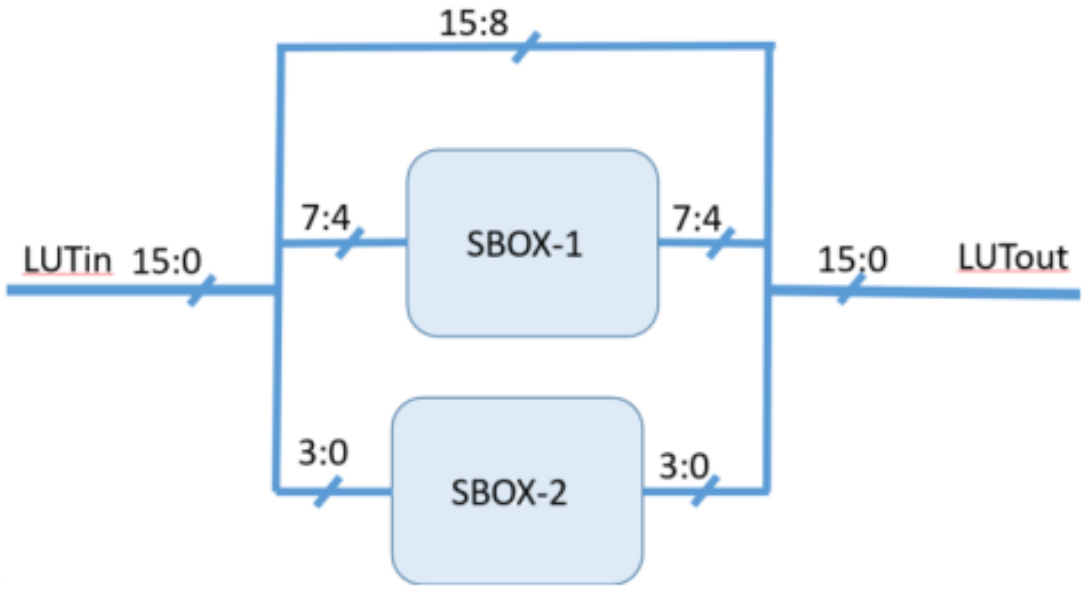

Figure 4: Lookup Table Component Expanded

Table 3: S-Box 1 Substitutions

Input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Output	1	11	9	12	13	6	15	3	14	8	7	4	10	2	5

Table 4: S-Box 2 Substitutions

Input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Output	15	0	13	7	11	14	5	10	9	2	12	1	3	4	8

The Lookup Table component takes two inputs; a control signal (LUTen), and a 16-bit input signal (LUTin). The substitution of the bits of the least significant byte of the input signal is performed according to the above tables. The output signal (LUTout) is then assigned the value of the unchanged most significant byte of the input, along with the substituted least significant bits.

2.1.4 Structural Model

The Structural model of the Combinational Logic is completed by combining the ALU, Shifter, and LUT components. The input control signal must be decoded to send input ABus and BBus signals to the appropriate component, and to output the appropriate result from the multiplexer.

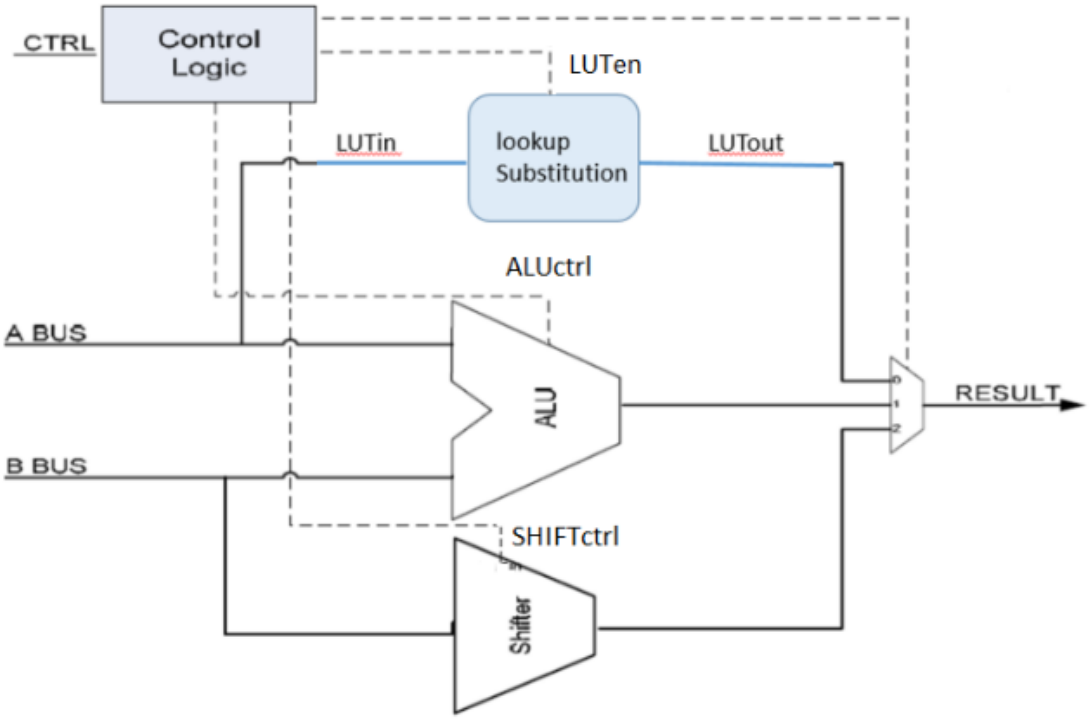

Figure 5: Structural VHDL Model

2.1.5 Testing

Testing of the Combinational Logic must be completed. Using a testbench, and a stimulus generator, a series of input control signals and A and B Bus are passed into the system. The output of these operations must be compared with the expected results to ensure that the system is operating correctly.

2.2 Synchronous Logic Design

The Synchronous Logic Design of the Crypto Coprocessor implements the memory elements of the design. Initially using an input and output register, this is expanded to using a register file, which provides synchronous writes with either synchronous or asynchronous reads.

2.2.1 Registers

The Input and Output Register components are implemented as specified in the provided diagram, found below:

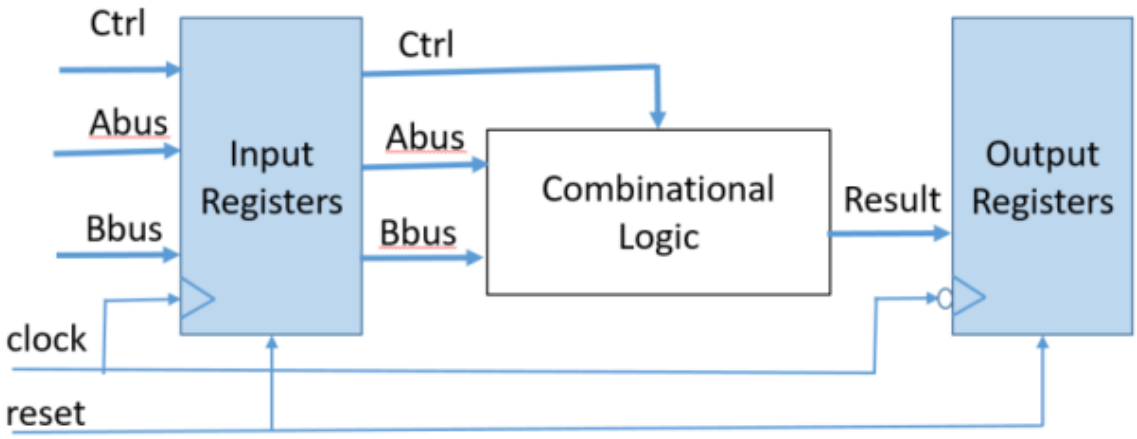

Figure 6: Register Components

The Input Register component has five inputs; a control signal (Ctrl), two 16-bit input signals (Abus, Bbus), a rising edge triggered clock input, and a reset input. The register outputs the control signal, and both the A and B Bus to the Combinational Logic.

The Output Register has three inputs; A 16-bit input (Result) which takes in the output of the Combinational Logic, a falling edge triggered clock input, and a reset input.

2.2.2 Memory - Register File

The Register File component is implemented as specified in the provided diagram, found below:

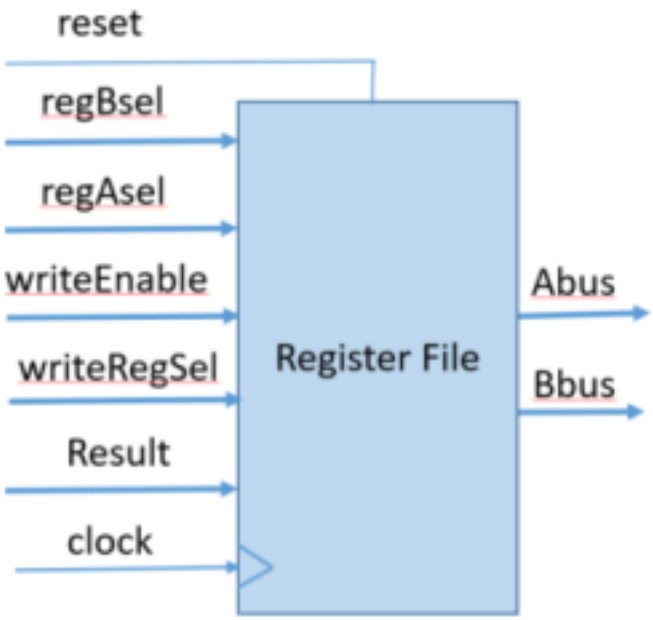

Figure 7: Register File Component

The Register File is used to implement a memory structure in the synchronous logic design. Using a synchronous write and either a synchronous or asynchronous read operation on a rising or falling edge. The register file contains 16 16-bit values, which can be output from the register file on the A or B Bus.

The Register File has 7 inputs, and 2 outputs; 3 1-bit inputs (Reset, writeEnable, and Clock), 3 4-bit inputs (regBsel, regAsel, and writeRegSel), and 1 16-bit input (Result). There are 2 16-bit outputs (Abus and Bbus).

2.2.3 Complete Crypto Coprocessor

The complete Crypto Coprocessor is implemented as specified in the provided diagram, found below:

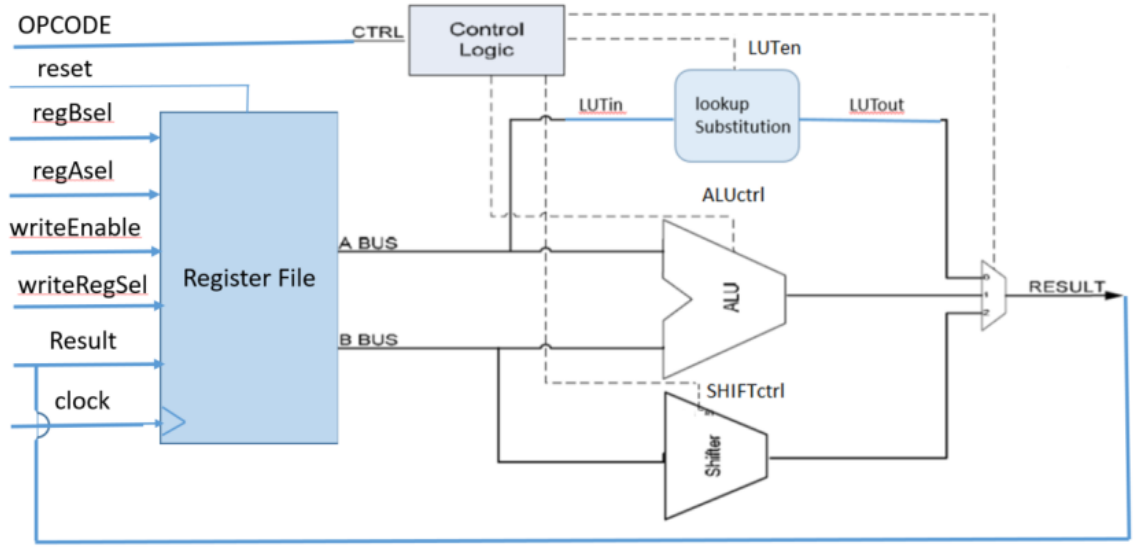

Figure 8: Complete Crypto Coprocessor VHDL Model

The complete Crypto Coprocessor consists of the Register File component, and the Combinational Logic components. The crypto coprocessor takes an input instruction, which specifies the registers to be read from, and passes the values to the combinational logic. Once the specified operation is complete, the output result from the combinational logic is passed back into the register file, and stored in the specified register.

The Crypto Coprocessor has 3 inputs; two 1-bit inputs (reset and clock), and a 16-bit instruction input. This instruction input contains the required opcode for the combinational logic, along with the regAsel, regBsel and writeRegSel values required for the Register File component.

2.2.4 Testing

Testing of the Crypto Coprocessor must be completed. Using a testbench and stimulus generator, input Instruction, reset, and clock values are passed to the system. The output of the system must then be compared with the expected result to ensure that the system is operating correctly.

Results

The following results were obtained from the completion of the the combinational and synchronous sections of the assignment.

3.1 Combinational Logic

The combinational logic consists of the Arithmetic Logic Unit, the Shifter component, the Lookup Substitution component, and the Control Signal Decoder.

3.1.1 ALU

The ALU must perform a number of arithmetic or logical operations on the input 16-bit values. These operations include arithmetic addition and subtraction, and logical AND, OR, XOR, NOT, and Move operations.

Using a case statement, the control signal may be used to determine the operation to be performed. The code, found in the Appendix "ALU Code" section, generates the following schematic:

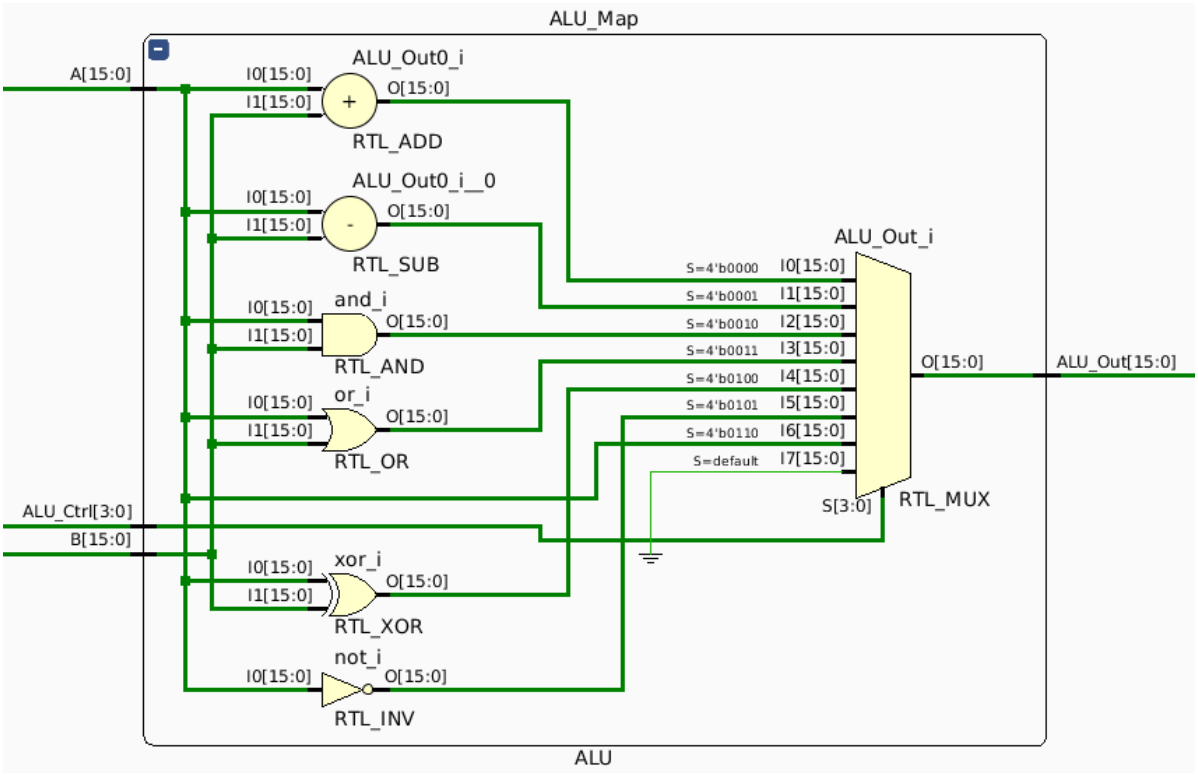

Figure 9: Generated ALU Compnent Schematic

3.1.2 Shifter

The Shifter component must perform bitwise rotation or shifting operations. As the ports used in this component are of type “unsigned” the ROR, ROL, SLL, and SRL functions may not be used, as they may have unexpected behaviour. As such the “rotate_right()”, “rotate_left()”, “shift_right()”, and “shift_left()” functions must be used.

A case statement chooses the operation to perform on the 16-bit input based on the input control signal. The code, found in the Appendix “Shifter Code” section, generates the following schematic:

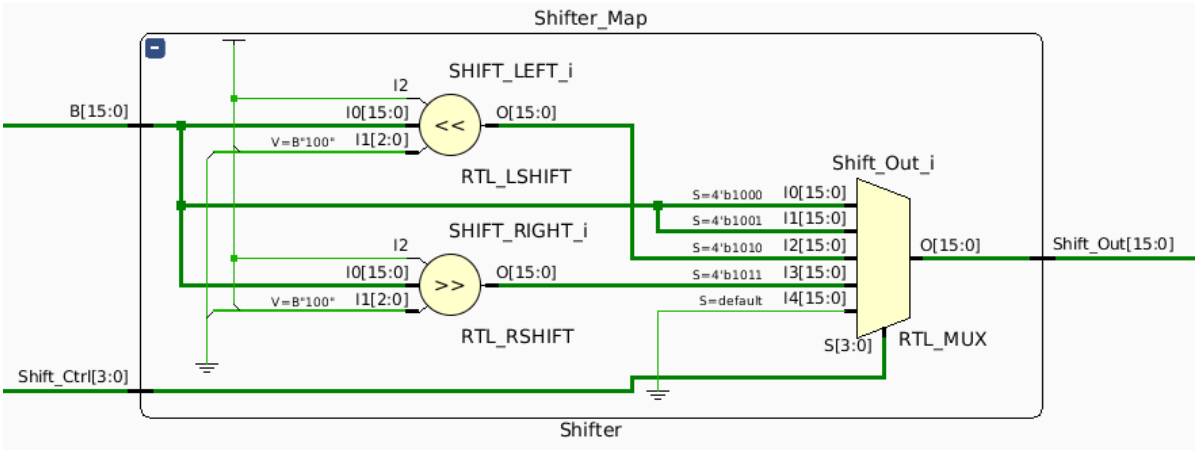

Figure 10: Generated Shifter Component Schematic

3.1.3 LUT

The Lookup Substitution component performs substitutions based in the input 16-bit values. The upper byte of the input value must pass through unnafected, with each of the lower four bits having their values substituted. Using two case statements, the upper 4-bits of the lower byte, and the lower 4-bits of the lower byte have their values substituted. These new values are then recombined with the unchanged upper byte, and output.

The code, found in the Appendix "LUT Code" section, generates the following schematic:

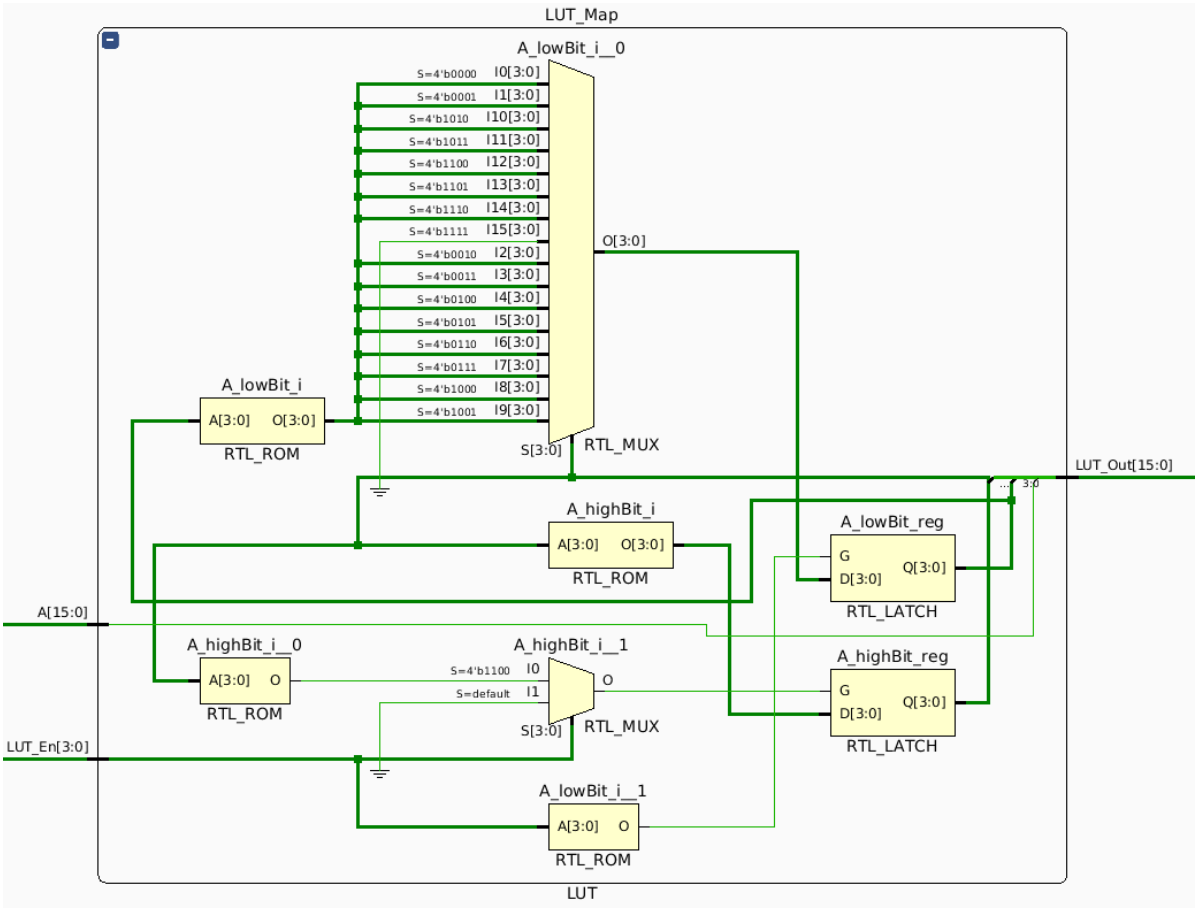

Figure 11: Generated Lookup Table Component Schematic

3.1.4 Control Decoder

The Control signal decoder component takes the input control signal, and outputs both the original control signal for each of the other components, and a value from 0 to 2, which allows an output multiplexer to output the value of the desired operation.

A case statement is used to choose the value from 0 to 3 to assign to each of the input control signal values. The value of 0 indicates a Lookup substitution operation, the value 1 indicates an ALU operation, and the value 2 indicates a Shifter operation.

The code, found in the Appendix "Control Decoder Code" section, generates the following schematic:

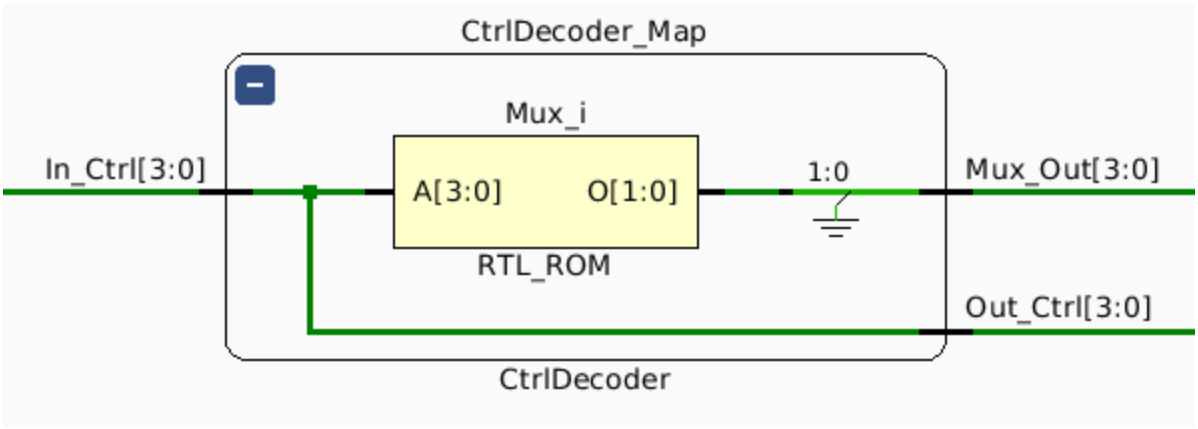

Figure 12: Generated Control Signal Decoder Schematic

3.1.5 Combinational Logic

The combinational logic combines the aforementioned components, passing in two 16-bit values (the A and B Bus), and a 4-bit control value. The appropriate operation is performed on the input values, with the appropriate result being output.

The code, found in the Appendix "Combinational Logic Code" section, generates the following schematic:

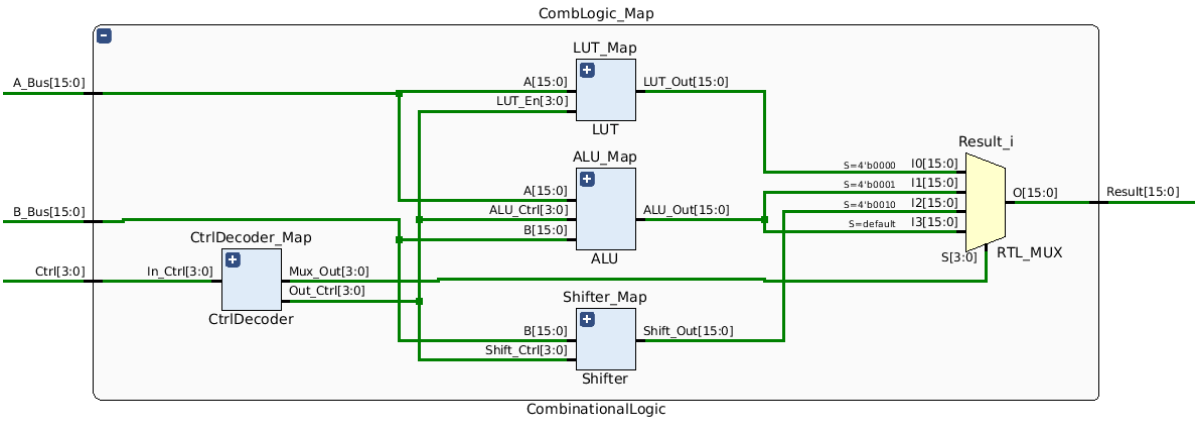

Figure 13: Generated Combinational Logic Component Schematic

3.2 Synchronous Logic

The synchronous logic consists of either a number of registers (Input and Output), or a register file, which can store a number of results, and perform operations on the specified results.

3.2.1 Registers

The Input Register component takes a 4-bit control signal, 2 16-bit A and B Bus signals, and 2 1-bit signals - the clock and reset. The input register uses an asynchronous reset, and is rising edge triggered. The Input Register then passes the control, A Bus and B Bus signals to the combinational logic component.

The Output Register component takes in a 16-bit Result signal, which is the output of the combinational logic, and 2 1-bit signals - the clock and reset. The Output Register also implements an asynchronous reset, and is falling edge triggered.

The code, found in the Appendix "Registers Code" section, generates the following schematic:

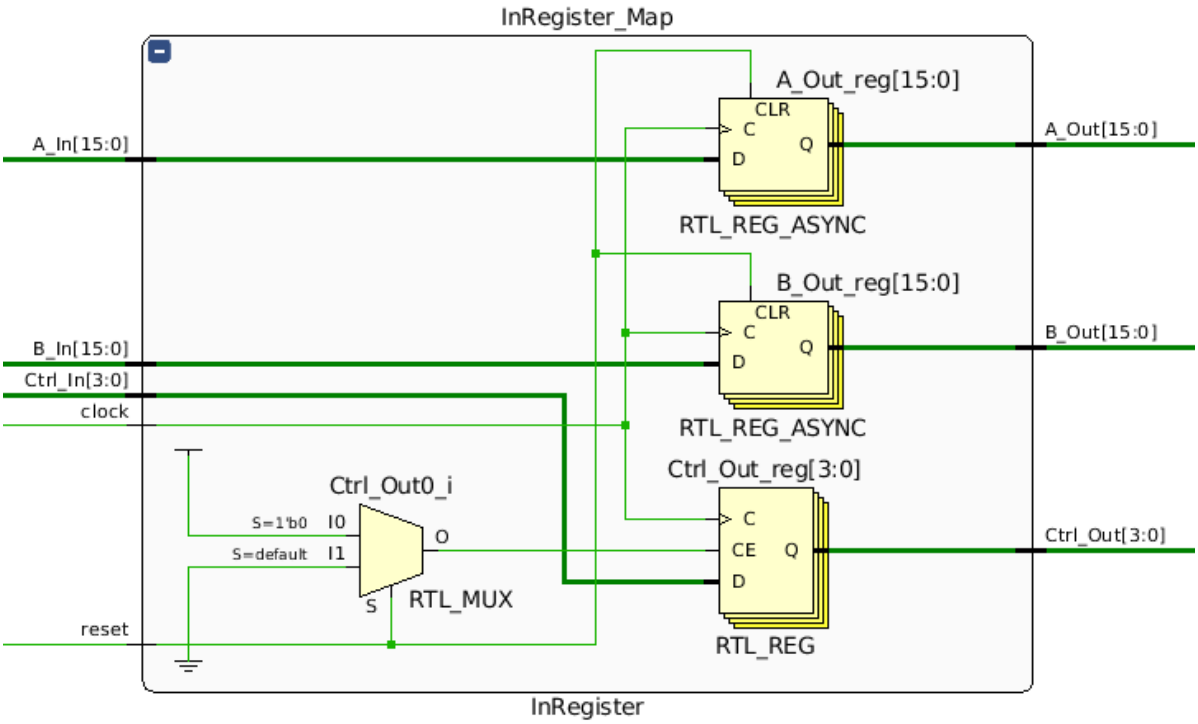

Figure 14: Generated Input Register Schematic

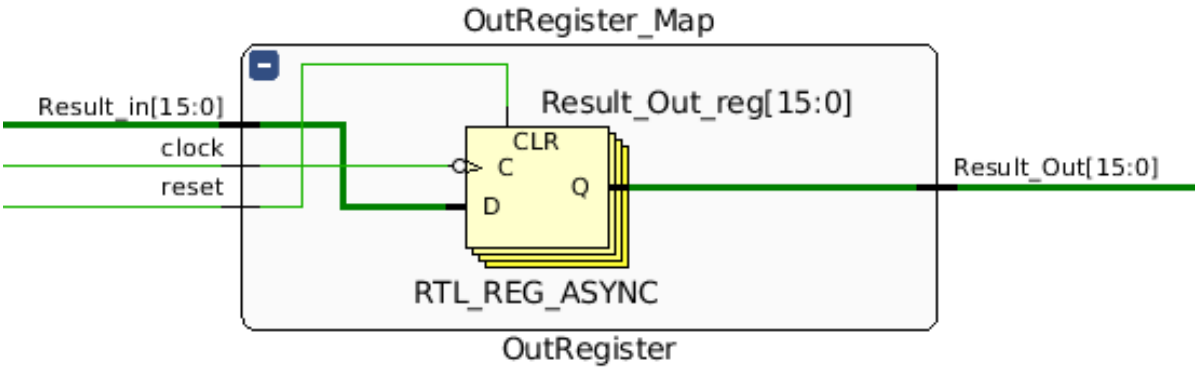

Figure 15: Generated Output Register Schematic

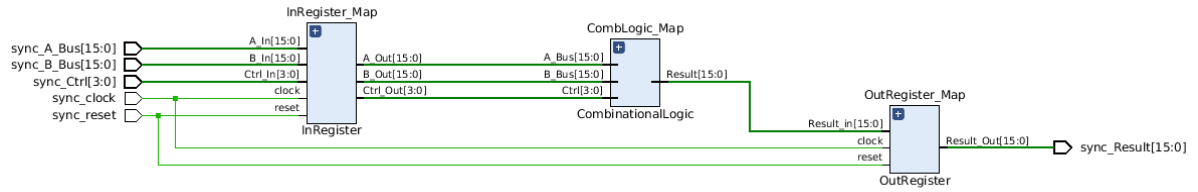

Figure 16: Generated Synchronous Logic with Input and Output Registers

3.2.2 Register File

The Register File Component takes a 16-bit result signal, 3 1-bit clock, writeEnable, and reset signals, and 3 4-bit select signals. The select signals are used to specify the registers which must be loaded and the register which must be written to. The read registers are passed to the 16-bit Abus and Bbus signals.

The Register File implements a synchronous write, and synchronous read, which is rising edge triggered. The reset functionality is also asynchronous. The code, found in the Appendix “Register File Code” section, generates the following schematic:

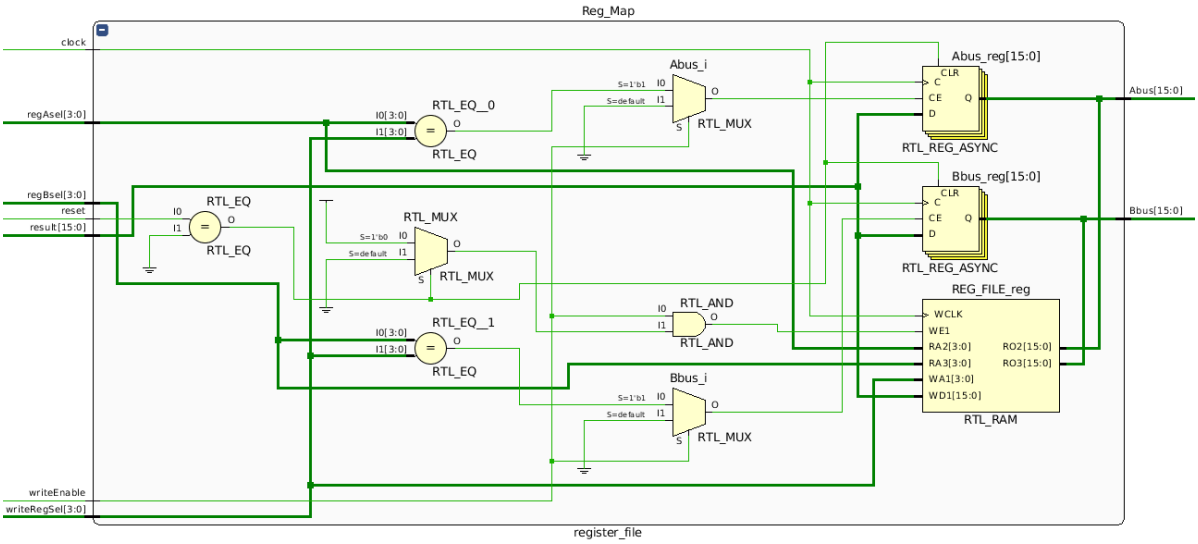

Figure 17: Generated Register File Schematic

3.2.3 Complete Crypto Coprocessor

The Crypto Coprocessor combines all aforementioned Combinational Logic with the Register File Component. A 16-bit Instruction signal is input, which is broken into 4 4-bit signals; the OpCode (Control Signal), regAsel, regBsel, and writeRegSel signals.

The specified A and B registers are read, and passed to the A and B Bus. These 16-bit values are passed into the combinational logic, alongside the OpCode signal. The combinational logic performs the operation as specified by the OpCode, with the 16-bit Result being output. This Result value is then passed back to the Register File components Result input signal, and the specified register has this value written to it.

The code, found in the Appendix “Crypto Coprocessor Code” section, generates the following schematic:

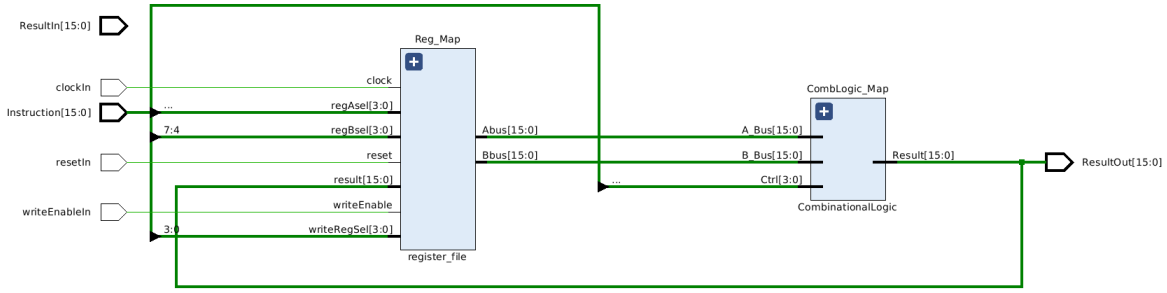

Figure 18: Generated Crypto Coprocessor Schematic

3.3 Testing

Testing was completed using a combination of a testbench and stimulus generator. A separate testbench and stimulus generator combination was used for the combinational logic, synchronous logic, and complete crypto coprocessor.

3.3.1 Combinational Logic

In order to test that the combinational logic was operating as required, a testbench, which, using a stimulus generator, passed specified inputs to the A and B buses. The Result output was then compared with an expected output, logging a "passed" or "failed" message to the console.

The A bus, B bus, and expected results used in the test are contained in the following table (in hexidecimal representation):

Table 5: Inputs and Expected Outputs of Combinational Logic

Index	OpCode	RA	RB	Expected Result
1	0	F0F1	F0F1	E1E2
2	1	F0F2	F0F1	0001
3	2	B2A1	C3D1	8281
4	3	234A	A21D	A35F
5	4	234A	A21D	8157
6	5	F21D	–	0DE2
7	6	ABCD	–	ABCD
8	8	–	12BC	C12B
9	9	12BC	2BC1	

Index	OpCode	RA	RB	Expected Result
10	A	–	BD14	D140
11	B	–	BD14	0BD1
12	C	B2A1	–	B270

The code for the testbench and stimulus generator can be found in the Appendix.

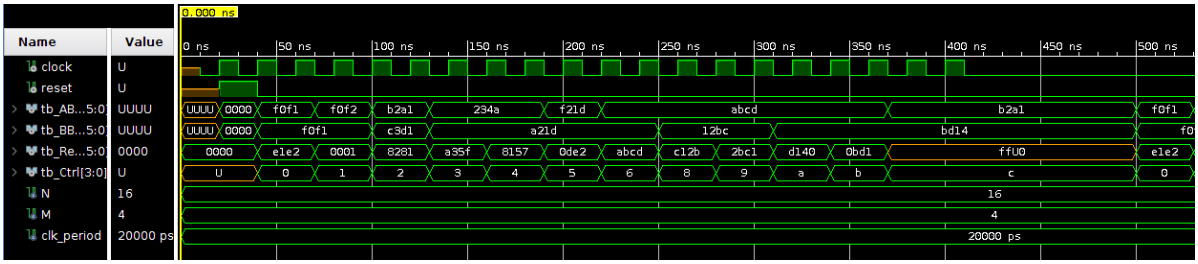

Figure 19: Scope Output for Combinational Logic

On testing, it appeared that the Lookup Substitution component was not working as expected, output incorrect values. Unfortunately the root of this issue was not found, and as such, this issue propagated throughout the assignment.

The output log file, found in the Appendix "Combinational Logic Log" section, shows the "passed Test" message for each iteration of testing.

3.3.2 Synchronous Logic

The synchronous logic was tested in the same way as the combinational logic. A testbench and stimulus generator compare expected results with the output of the second register. The stimulus generator was modified to initiate the "clktmp" variable with a value of '0'. The table used in the combinational logic section was again used for testing purposes, giving the same outputs as above.

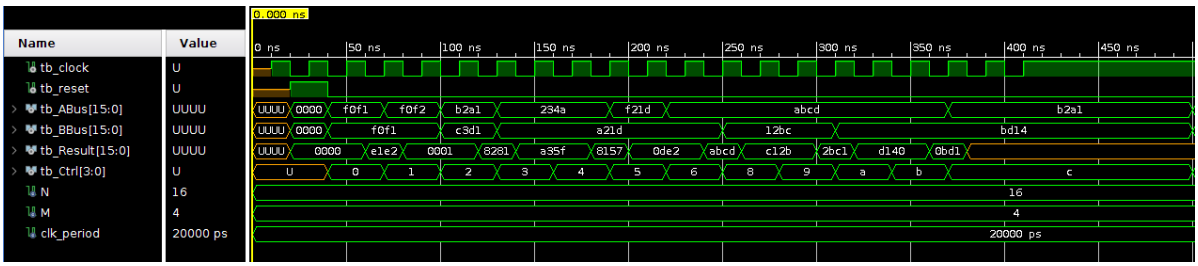

Figure 20: Scope Output for Synchronous Logic

On testing, the correct values were output for the first iteration of the testing, however on subsequent iterations, the output were incorrect. The output log file, found in the Appendix "Synchronous Logic Log" section, shows the "passed Test" message for each test in the first iteration.

3.3.3 Complete Crypto Coprocessor Test Program

The completed Crypto Coprocessor was tested by passing in a number of 16-bit instruction words. These words refer to the opcode, register to assign to the A_Bus, register to assign to the B_Bus, and register to assign the result of the operation to.

A testbench and stimulus generator were again used to pass in the instruction word, and to compare the outputs to the expected output.

The following table gives the operations performed in the testing process:

Table 6: Inputs and Expected Outputs of Completed Crypto Coprocessor

Operation	A Register	B Register	Result Register
ADD	R5	R4	R12
XOR	R1	R8	R7
ROR4	–	R12	R0
SLL4	–	R9	R3
ADD	R0	R7	R10
SUB	R7	R3	R12
NOP	–	–	–
AND	R12	R10	R9
NOP	–	–	–
LUT	R9	–	R2

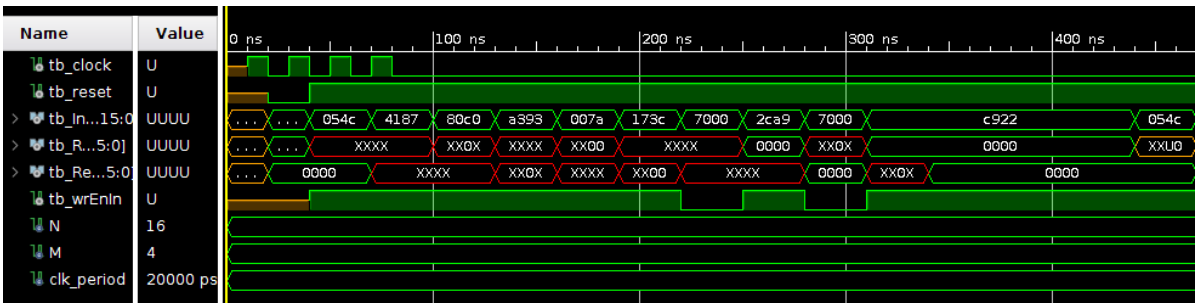

Figure 21: Scope Output for Crypto Coprocessor

On testing, no correct output could be achieved using the stimulus generator. A number of undefined values pass through the system, resulting in undefined output result values. The output log file, found in the Appendix "Crypto Coprocessor Log" section, shows the "failed Test" message for each iteration of the test.

Conclusion

This assignment afforded an opportunity to become more failiar with the testing process in VHDL. The use of testbenches and stimulus generators showed how correct operation of components and systems can be confirmed.

The assignment also provided experience in using timing, clock signals, and memory elements. These aspects must be used to correctly implement the registers and register file, which must have reset, and synchronous/asynchronous read or write capabilities.

Appendix

All code and log file items found within this appendix can be found in the attached zip file.

5.1 Code

5.1.1 ALU Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use ieee.NUMERIC_STD.all;
-----------------------------------------------
---------- ALU 8-bit VHDL ---------------------
-----------------------------------------------
entity ALU is
  generic (N : positive := 16; M : positive :=4);
  port(
        A, B      : in  unsigned(N-1 downto 0);   -- 2 N-bit inputs
        ALU_Ctrl  : in  unsigned(M-1 downto 0);   -- 1 M-bit function selection
        ALU_Out   : out unsigned(N-1 downto 0)   -- 1 N--bit output
       );
end ALU;
architecture Behavioral of ALU is
begin
  process(A,B,ALU_Ctrl)
    begin
      case(ALU_Ctrl) is
        when "0000" => -- Addition
          ALU_out <= A + B ;
        when "0001" => -- Subtraction
          ALU_out <= A - B ;
        when "0010" => -- AND
          ALU_out <= A and B;
        when "0011" => -- OR
          ALU_out <= A or B;
        when "0100" => -- XOR
          ALU_out <= A xor B;
        when "0101" => -- NOT
          ALU_out <= not A;
        when "0110" => -- Move
          ALU_out <= A;
        when others => ALU_out <= x"0000";
      end case;
  end process;
end Behavioral;
```

5.1.2 Shifter Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.NUMERIC_STD.all;
-----------------------------------------------
```

```
---------- Shifter 16-bit VHDL --------------------
-------------------------------------------------
entity Shifter is
  generic (N : positive := 16; M : positive :=4);
  port(
        B           : in  unsigned(N-1 downto 0);     --   N-bit input
        Shift_Ctrl  : in  unsigned(M-1 downto 0);  -- 1 M-bit function selection
        Shift_Out   : out unsigned(N-1 downto 0)   -- 1 N--bit output
      );
end Shifter;
architecture Behavioral of Shifter is
begin
  process(B,Shift_Ctrl)
    begin
      case(Shift_Ctrl) is
      when x"8" => -- Rotate Right
        Shift_out <= rotate_right(B, M);  -- shift left logic
      when x"9" => -- Rotate Left
        Shift_out <= rotate_left(B, M);
      when x"A" => --  Shift Left
        Shift_out <= shift_left(B, M);
      when x"B" => -- Shift Right
        Shift_out <= shift_right(B, M);
      when others => Shift_out <= x"0000";
    end case;
  end process;
end Behavioral;
```

5.1.3 LUT Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.NUMERIC_STD.all;
-------------------------------------------------
---------- LUT 16-bit VHDL --------------------
-------------------------------------------------
entity LUT is
  generic (N : positive := 16; M : positive :=4);
  port(
        A           : in  unsigned(N-1 downto 0);     --   N-bit input
        LUT_En   : in  unsigned(M-1 downto 0);  -- 1 M-bit function selection
        LUT_Out   : out unsigned(N-1 downto 0)  -- 1 N--bit output
```

```
      );
end LUT;
architecture Behavioral of LUT is
begin
  process(A,LUT_En)
    variable A_lowBit : unsigned(M-1 downto 0) := A(3 downto 0);
    variable A_highBit : unsigned(M-1 downto 0) := A(7 downto 4);
    begin
      case(LUT_en) is
       when "1100" =>
         case(A_lowBit) is
           when "0000" =>
             A_lowBit := "1111";
           when "0001" =>
             A_lowBit := "0000";
           when "0010" =>
             A_lowBit := "1101";
           when "0011" =>
             A_lowBit := "0111";
           when "0100" =>
             A_lowBit := "1011";
           when "0101" =>
             A_lowBit := "1110";
           when "0110" =>
             A_lowBit := "0101";
           when "0111" =>
             A_lowBit := "1010";
           when "1000" =>
             A_lowBit := "1001";
           when "1001" =>
             A_lowBit := "0010";
           when "1010" =>
             A_lowBit := "1100";
           when "1011" =>
             A_lowBit := "0001";
           when "1100" =>
             A_lowBit := "0011";
           when "1101" =>
             A_lowBit := "0100";
           when "1110" =>
             A_lowBit := "1000";
           when "1111" => A_lowBit := "0110";
           when others => A_lowBit := "0000";
         end case;
```

```
        case(A_highBit) is
          when "0000" =>
            A_highBit := "0001";
          when "0001" =>
            A_highBit := "1011";
          when "0010" =>
            A_highBit := "1001";
          when "0011" =>
            A_highBit := "1100";
          when "0100" =>
            A_highBit := "1101";
          when "0101" =>
            A_highBit := "0110";
          when "0110" =>
            A_highBit := "1111";
          when "0111" =>
            A_highBit := "0011";
          when "1000" =>
            A_highBit := "1110";
          when "1001" =>
            A_highBit := "1000";
          when "1010" =>
            A_highBit := "0111";
          when "1011" =>
            A_highBit := "0010";
          when "1100" =>
            A_highBit := "1010";
          when "1101" =>
            A_highBit := "0010";
          when "1110" =>
            A_highBit := "0101";
          when "1111" => A_lowBit := "0000";
          when others => A_lowBit := "0000";
        end case;
      when others => LUT_out <= x"0000";
    end case;
    LUT_out(15) <= A(15);
    LUT_out(14) <= A(15);
    LUT_out(13) <= A(15);
    LUT_out(12) <= A(15);
    LUT_out(11) <= A(15);
    LUT_out(10) <= A(15);
    LUT_out(9) <= A(15);
    LUT_out(8) <= A(15);
```

```
    LUT_out(7) <= A_highBit(3);
    LUT_out(6) <= A_highBit(2);
    LUT_out(5) <= A_highBit(1);
    LUT_out(4) <= A_highBit(0);
    LUT_out(3) <= A_lowBit(3);
    LUT_out(2) <= A_lowBit(2);
    LUT_out(1) <= A_lowBit(1);
    LUT_out(0) <= A_lowBit(0);
  end process;
end Behavioral;
```

5.1.4 Control Decoder Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity CtrlDecoder is
generic (M : positive :=4);
Port ( In_Ctrl : in unsigned (M-1 downto 0);
       Out_Ctrl : out unsigned (M-1 downto 0);
       Mux_Out  : out unsigned(M-1 downto 0)
     );
end CtrlDecoder;

architecture Behavioral of CtrlDecoder is

  signal Mux : unsigned(M-1 downto 0);

  begin
    process(In_Ctrl, Mux)
      begin
        case(In_Ctrl) is
          when x"0" => Mux <= x"1";
          when x"1" => Mux <= x"1";
          when x"2" => Mux <= x"1";
          when x"3" => Mux <= x"1";
          when x"4" => Mux <= x"1";
          when x"5" => Mux <= x"1";
          when x"6" => Mux <= x"1";
          when x"8" => Mux <= x"2";
          when x"9" => Mux <= x"2";
```

```
          when x"A" => Mux <= x"2";
          when x"B" => Mux<= x"2";
          when x"C" => Mux <= x"0";
          when others => Mux <= x"3";
        end case;
        Out_Ctrl <= In_Ctrl;
        Mux_Out <= Mux;
     end process;

end Behavioral;
```

5.1.5 Combinational Logic Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.NUMERIC_STD.all;

entity CombinationalLogic is
  generic (N : positive := 16; M : positive :=4);
  port(
        A_Bus, B_Bus    : in unsigned(N-1 downto 0);
        Ctrl            : in unsigned(M-1 downto 0);
        Result          : out unsigned(N-1 downto 0)
      );
end CombinationalLogic;

architecture structural_view of CombinationalLogic is

    component CtrlDecoder
    port( In_Ctrl : in unsigned (M-1 downto 0);
          Out_Ctrl : out unsigned (M-1 downto 0);
          Mux_Out  : out unsigned(M-1 downto 0)
        );
    end component;

    component ALU
    port(
          A, B        : in  unsigned(N-1 downto 0);  -- 2 N-bit inputs
          ALU_Ctrl    : in  unsigned(M-1 downto 0);  -- 1 M-bit function selection
          ALU_Out     : out unsigned(N-1 downto 0)  -- 1 N--bit output
        );
    end component;
```

```
    component Shifter
    port(
          B           : in  unsigned(N-1 downto 0);     --   N-bit input
          Shift_Ctrl  : in  unsigned(M-1 downto 0);  -- 1 M-bit function selection
          Shift_Out   : out unsigned(N-1 downto 0)  -- 1 N--bit output
        );
    end component;
    component LUT
    port(
          A           : in  unsigned(N-1 downto 0);     --   N-bit input
          LUT_En      : in  unsigned(M-1 downto 0);  -- 1 M-bit function selection
          LUT_Out     : out unsigned(N-1 downto 0)  -- 1 N--bit output
        );
    end component;

    signal ALU_Result, Shift_Result, LUT_Result : unsigned(N-1 downto 0);
    signal Ctrl_In : unsigned(M-1 downto 0);
    signal Mux : unsigned(M-1 downto 0);

begin

  CtrlDecoder_Map : CtrlDecoder
  port map( In_Ctrl => Ctrl,
            Out_Ctrl => Ctrl_In,
            Mux_Out  => Mux
          );

  ALU_Map : ALU
    port map(
              A => A_bus,
              B => B_Bus,
              ALU_Ctrl => Ctrl_In,
              ALU_Out => ALU_Result
            );
  Shifter_Map : Shifter
    port map(
              B => B_Bus,
              Shift_Ctrl => Ctrl_In,
              Shift_Out => Shift_Result
            );
  LUT_Map : LUT
    port map(
              A => A_Bus,
              LUT_En => Ctrl_In,
```

```
            LUT_Out => LUT_Result
          );

    process(A_Bus, B_Bus, Mux, LUT_Result, ALU_Result, Shift_Result)
       begin
        case(Mux) is
          when x"0" => Result <= LUT_Result;
          when x"1" => Result <= ALU_Result;
          when x"2" => Result <= Shift_Result;
          when others => Result <= ALU_Result;
        end case;
    end process;

end ;
```

5.1.6 Input Register Code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.NUMERIC_STD.all;

entity InRegister is
  generic (N : positive := 16; M : positive :=4);
  port( reset, clock : in std_logic;
        A_In, B_In : in unsigned(N-1 downto 0);
        Ctrl_In : in unsigned(M-1 downto 0);
        A_Out, B_Out: out unsigned(N-1 downto 0);
        Ctrl_Out : out unsigned(M-1 downto 0)
      );
  end InRegister;
--flip-flop with asynchronous reset
architecture beh of InRegister is
  begin
    asr_clk_en: process (reset,clock)
  begin
    if (reset = '1') then   --asynchronous reset
      A_Out <= (others => '0'); --x"0000";
      B_Out <= (others => '0'); -- x"0000";
    elsif (clock'event and clock='1') then --rising_edge(clk)
        A_Out <= A_In;
        B_Out <= B_In;
        Ctrl_Out <= Ctrl_In;
```

```
    end if;
  end process asr_clk_en;
end beh;
```

5.1.7 Output Register Code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.NUMERIC_STD.all;

entity OutRegister is
  generic (N : positive := 16; M : positive :=4);
  port( reset, clock: in std_logic;
        Result_in : in unsigned (N-1 downto 0);
        Result_Out : out unsigned(N-1 downto 0)
      );
  end OutRegister;
--flip-flop with asynchronous reset & set
architecture beh of OutRegister is
  begin
    asr_clk_en: process (reset, clock)
  begin
    if (reset = '1') then    --asynchronous reset
      Result_Out <= (others => '0'); --x"0000";
    elsif (clock'event and clock='0') then --falling_edge(clk)
        Result_Out <= Result_in;
    end if;
  end process asr_clk_en;
end beh;
```

5.1.8 Synchronous Logic Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.NUMERIC_STD.all;

entity synchronousSystem is
 generic (N : positive := 16; M : positive :=4);
 port(  sync_reset, sync_clock : in std_logic;
        sync_A_Bus, sync_B_Bus : in unsigned(N-1 downto 0);
        sync_Result : out unsigned(N-1 downto 0);
```

```
        sync_Ctrl : in unsigned(M-1 downto 0)
      );
end synchronousSystem;

architecture Behavioral of synchronousSystem is

  signal In_A, In_B : unsigned(N-1 downto 0);
  signal In_Ctrl : unsigned(M-1 downto 0);
  signal In_Result : unsigned(N-1 downto 0);

  component InRegister
    port ( reset, clock : in std_logic;
           A_In, B_In : in unsigned(N-1 downto 0);
           Ctrl_In : in unsigned(M-1 downto 0);
           A_Out, B_Out: out unsigned(N-1 downto 0);
           Ctrl_Out : out unsigned(M-1 downto 0)
         );
  end component;
  component CombinationalLogic
    port ( A_Bus, B_Bus    : in unsigned(N-1 downto 0);
           Ctrl            : in unsigned(M-1 downto 0);
           Result          : out unsigned(N-1 downto 0)
         );
  end component;
  component OutRegister
    port ( reset, clock: in std_logic;
           Result_in : in unsigned (N-1 downto 0);
           Result_Out : out unsigned (N-1 downto 0)
         );
  end component;
begin
  InRegister_Map : InRegister
    port map( reset => sync_reset,
              clock => sync_clock,
              A_In => sync_A_Bus,
              B_In => sync_B_Bus,
              Ctrl_In => sync_Ctrl,
              A_Out => In_A,
              B_Out => In_B,
              Ctrl_Out => In_Ctrl
            );

  CombLogic_Map : CombinationalLogic
    port map( A_Bus => In_A,
```

```
              B_Bus => In_B,
              Ctrl => In_Ctrl,
              Result => In_Result
            );

  OutRegister_Map : OutRegister
    port map( reset => sync_reset,
              clock => sync_clock,
              Result_in => In_Result,
              Result_Out => sync_Result
            );

end Behavioral;
```

5.1.9 Register File Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity register_file is
 generic (N : positive := 16; M : positive :=4);
 port( Abus           : out unsigned(15 downto 0);
       Bbus           : out unsigned(15 downto 0);
       result         : in unsigned(15 downto 0);
       writeEnable    : in std_logic;
       regAsel        : in unsigned(3 downto 0);
       regBsel        : in unsigned(3 downto 0);
       writeRegSel    : in unsigned(3 downto 0);
       reset          : in std_logic;
       clock          : in std_logic
     );
end register_file;

architecture Behavioral of register_file is

  type memory is array(0 to 15) of unsigned(15 downto 0);
  signal REG_FILE : memory := ( 0 => x"0001",
                                1 => x"c505",
                                2 => x"3c07",
                                3 => x"d405",
                                4 => x"1186",
```

```
                                5 => x"f407",
                                6 => x"1086",
                                7 => x"4706",
                                8 => x"6808",
                                9 => x"baa0",
                                10=> x"c902",
                                11 => x"100b",
                                12 => x"C000",
                                13=> x"c902",
                                14 => x"100b",
                                15 => x"B000",
                                others => (others => '0')
                              );
  begin
    asr_clk_en: process (reset,clock)
    begin
      if (reset = '0') then   --asynchronous reset
        Abus <= (others => '0'); --x"0000";
        Bbus <= (others => '0'); -- x"0000";
      elsif (clock'event and clock='1') then --rising_edge(clk)
          if(writeEnable = '1') then
            reg_file(to_integer(writeRegSel)) <= result;
            if(regAsel = writeRegSel) then
              Abus <= result;
            end if;
            if(regBsel = writeRegSel) then
              Bbus <= result;
            end if;
          else null;
          end if;
      end if;
    end process asr_clk_en;
    Abus <= reg_file(to_integer(regAsel));
    Bbus <= reg_file(to_integer(regBsel));
end Behavioral;
```

5.1.10 Crypto Coprocessor Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
```

```
entity CompleteCrypto is
 generic (N : positive := 16; M : positive :=4);
 port( Instruction : in unsigned(N-1 downto 0);
       resetIn, clockIn : in std_logic;
       writeEnableIn  : in std_logic;
       ResultIn : in unsigned(N-1 downto 0);
       ResultOut : out unsigned(N-1 downto 0)
     );
end CompleteCrypto;

architecture Behavioral of CompleteCrypto is

  signal A_In, B_In : unsigned(N-1 downto 0);
  signal OpCode, regASelIn, regBSelIn, writeRegSelIn : unsigned(M-1 downto 0);
  signal Result_In : unsigned(N-1 downto 0);
  signal writeEnableTmp : std_logic;

  component register_file
  port(  Abus          : out unsigned(15 downto 0);
         Bbus          : out unsigned(15 downto 0);
         result        : in unsigned(15 downto 0);
         writeEnable   : in std_logic;
         regAsel       : in unsigned(3 downto 0);
         regBsel       : in unsigned(3 downto 0);
         writeRegSel   : in unsigned(3 downto 0);
         reset         : in std_logic;
         clock         : in std_logic
       );
  end component;

  component CombinationalLogic
    port ( A_Bus, B_Bus    : in unsigned(N-1 downto 0);
           Ctrl            : in unsigned(M-1 downto 0);
           Result          : out unsigned(N-1 downto 0)
         );
    end component;

begin

  Reg_Map : register_file
    port map(  Abus => A_In,
               Bbus => B_In,
               result => Result_In,
               writeEnable => writeEnableIn,
```

```
            regAsel => regASelIn,
            regBsel => regBSelIn,
            writeRegSel => writeRegSelIn,
            reset => resetIn,
            clock => clockIn
          );

  CombLogic_Map : CombinationalLogic
    port map( A_Bus => A_In,
              B_Bus => B_In,
              Ctrl => OpCode,
              Result =>  Result_In
            );

  process(Instruction)
    begin
      OpCode <= Instruction(N-1 downto 3*M);
      regASelIn <= Instruction((3*M)-1 downto 2*M);
      regBSelIn <= Instruction((2*M)-1 downto M);
      writeRegSelIn <= Instruction(M-1 downto 0);
      writeEnableTmp <= writeEnableIn;
      ResultOut <= Result_In;
--    case(OpCode) is
--      when x"7" => writeEnableTmp <= '0';
--      when others => writeEnableTmp <= '1';
--    end case;
    --Result_In <= Result_Out;
  end process;

end Behavioral;
```

5.2 Testbench and Stimulus Generators

5.2.1 Combinational Logic Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.NUMERIC_STD.all;

entity tb_combLogic is
  generic(N : positive := 16; M : positive :=4);
end tb_combLogic;
```

```
architecture struct of tb_combLogic is

  constant clk_period : time := 20ns;

  signal clock, reset : std_logic;
  signal tb_ABus, tb_BBus, tb_Result : unsigned(N-1 downto 0);
  signal tb_Ctrl : unsigned(M-1 downto 0);

  component CombinationalLogic is
    port( A_Bus    : in unsigned(N-1 downto 0);
          B_Bus    : in unsigned(N-1 downto 0);
          Ctrl             : in unsigned(M-1 downto 0) := x"0";
          Result           : out unsigned(N-1 downto 0)
        );
  end component;
  component stimgen
    generic (clock_period : time := clk_period);
    port( A_Bus : out unsigned(N-1 downto 0);
          B_Bus : out unsigned(N-1 downto 0);
          Ctrl  : out unsigned (M-1 downto 0);
          clock : out STD_LOGIC;
          reset : out STD_LOGIC;
          result : in unsigned  (N-1 downto 0)
         );
    end component;

begin
  uut: CombinationalLogic
    port map( A_Bus => tb_ABus,
              B_Bus => tb_BBus,
              Ctrl => tb_Ctrl,
              Result => tb_Result
            );

   my_stimgen: stimgen
     port map( A_Bus => tb_ABus,
               B_Bus => tb_BBus,
               Ctrl => tb_Ctrl,
               clock => clock,
               reset => reset,
               result => tb_Result
              );
```

```
end struct;
```

5.2.2 Stimulus Generator 1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.NUMERIC_STD.all;

entity stimgen is

    generic (CLOCK_PERIOD : time);
    Port ( A_Bus : out unsigned(15 downto 0);
           B_Bus : out unsigned(15 downto 0);
           Ctrl : out unsigned(3 downto 0);
           clock : out STD_LOGIC;
           reset : out STD_LOGIC;
           result : in unsigned  (15 downto 0)
          );
end stimgen;

architecture Behavioral of stimgen is

  signal DONE : boolean := false;
  signal FAILED: boolean := false;
begin
    -- Ctrl generator
    Clock1: process
        variable clktmp : std_logic := '1';
    begin
        while DONE /= true and FAILED /= true loop
            wait for CLOCK_PERIOD/2;
            clktmp := not clktmp;
            Clock <= clktmp;
        end loop;
        wait;
    end process;

    -- Apply stimulus signals and check result (LED signal values)
    CheckResult: process

    begin
        wait for CLOCK_PERIOD;
```

```
        -- Put system in reset
        Reset <= '1';
        A_Bus <= x"0000";
        B_Bus <= x"0000";

        wait for CLOCK_PERIOD;
        -- Take system out of reset
        Reset <= '0';
        wait for 0ns; -- force update of signal values

-- test three rounds
        for i in 0 to 3 loop

 -- test 1:
        Ctrl <= x"0";
        A_Bus <= x"F0F1";
        B_Bus <= x"F0F1";
        wait for 1.5*CLOCK_PERIOD;
        if  result = x"E1E2" then
           report "Passed test1";
        else
           report "failed test1" severity error;
           FAILED <= true;
        end if;

 -- test 2:
        Ctrl <= x"1";
        A_Bus <= x"F0F2";
        B_Bus <= x"F0F1";
        wait for 1.5*CLOCK_PERIOD;
         if result = x"0001" then
            report "Passed test2";
         else
            report "failed test2" severity error;
            FAILED <= true;
         end if;

-- test 3:
        Ctrl <= x"2";
        A_Bus <= x"B2A1";
        B_Bus <= x"C3D1";
        wait for 1.5*CLOCK_PERIOD;
        if result = x"8281" then
           report "Passed test3";
```

```
        else
           report "failed test3" severity error;
           FAILED <= true;
        end if;

-- test 4:
        Ctrl <= x"3";
        A_Bus <= x"234A";
        B_Bus <= x"A21D";
        wait for 1.5*CLOCK_PERIOD;
   if result <= x"A35F" then
           report "Passed test4";
        else
           report "failed test4" severity error;
           FAILED <= true;
        end if;

-- test 5:
        Ctrl <= x"4";
        A_Bus <= x"234A";
        B_Bus <= x"A21D";
        wait for 1.5*CLOCK_PERIOD;
        if result = x"8157" then
           report "Passed test5";
        else
           report "failed test5" severity error;
           FAILED <= true;
        end if;

 -- test 6:
        Ctrl <= x"5";
        A_Bus <= x"F21D";
        wait for 1.5*CLOCK_PERIOD;
        if result = x"0DE2" then
           report "Passed test6";
        else
           report "failed test6" severity error;
           FAILED <= true;
        end if;

  -- test 7:
        Ctrl <= x"6";
        A_Bus <= x"ABCD";
        wait for 1.5*CLOCK_PERIOD;
```

```
        if result = x"ABCD" then
           report "Passed test7";
        else
           report "failed test7" severity error;
           FAILED <= true;
        end if;

    -- test 8:
        Ctrl <= x"8";
        B_Bus <= x"12BC";
        wait for 1.5*CLOCK_PERIOD;
        if result = x"C12B" then
            report "Passed test8";
        else
            report "failed test8" severity error;
            FAILED <= true;
        end if;

    -- test 9:
        Ctrl <= x"9";
        B_Bus <= x"12BC";
        wait for 1.5*CLOCK_PERIOD;
        if result = x"2BC1" then
           report "Passed test9";
        else
           report "failed test9" severity error;
           FAILED <= true;
        end if;

    -- test 10:
        Ctrl <= x"A";
        B_Bus <= x"BD14";
        wait for 1.5*CLOCK_PERIOD;
        if result = x"D140" then
           report "Passed test10";
        else
           report "failed test10" severity error;
           FAILED <= true;
        end if;

    -- test 11:
        Ctrl <= x"B";
        B_Bus <= x"BD14";
        wait for 1.5*CLOCK_PERIOD;
```

```
        if result = x"0BD1" then
           report "Passed test11";
        else
           report "failed test11" severity error;
           FAILED <= true;
        end if;

   -- test 12:
        Ctrl <= x"C";
        A_Bus <= x"B2A1";
        wait for 1.5*CLOCK_PERIOD;
        if result = x"B270" then
           report "Passed test12";
        else
           report "failed test12" severity error;
           FAILED <= true;
        end if;

        wait for 5*CLOCK_PERIOD;

        end loop;  -- end round loop

        wait for 5*CLOCK_PERIOD;


        DONE <= true;
        Report "Test completed";
        wait;
    end process;

end Behavioral;
```

5.2.3 Synchronous Logic Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.NUMERIC_STD.all;

entity tb_synchReg is
  generic(N : positive := 16; M : positive :=4);
end tb_synchReg;
```

```
architecture struct of tb_synchReg is

  constant clk_period : time := 20ns;

  signal tb_clock, tb_reset : std_logic;
  signal tb_ABus, tb_BBus, tb_Result : unsigned(N-1 downto 0);
  signal tb_Ctrl : unsigned(M-1 downto 0);

  component synchronousSystem is
    port( sync_reset, sync_clock : in std_logic;
          sync_A_Bus, sync_B_Bus : in unsigned(N-1 downto 0);
          sync_Result : out unsigned(N-1 downto 0);
          sync_Ctrl : in unsigned(M-1 downto 0)
        );
  end component;
  component stimgen2
    generic (clock_period : time := clk_period);
    port( A_Bus : out unsigned(N-1 downto 0);
          B_Bus : out unsigned(N-1 downto 0);
          Ctrl  : out unsigned (M-1 downto 0);
          clock : out STD_LOGIC;
          reset : out STD_LOGIC;
          result : in unsigned  (N-1 downto 0)
         );
    end component;

begin
  uut: synchronousSystem
    port map( sync_reset => tb_reset,
              sync_clock => tb_clock,
              sync_A_Bus => tb_ABus,
              sync_B_Bus => tb_BBus,
              sync_Ctrl => tb_Ctrl,
              sync_Result => tb_Result
            );

   my_stimgen: stimgen2
     port map( A_Bus => tb_ABus,
               B_Bus => tb_BBus,
               Ctrl => tb_Ctrl,
               clock => tb_clock,
               reset => tb_reset,
               result => tb_Result
             );
```

```
end struct;
```

5.2.4 Stimulus Generator 2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.NUMERIC_STD.all;

entity stimgen2 is

    generic (CLOCK_PERIOD : time);
    Port ( A_Bus : out unsigned(15 downto 0);
           B_Bus : out unsigned(15 downto 0);
           Ctrl : out unsigned(3 downto 0);
           clock : out STD_LOGIC;
           reset : out STD_LOGIC;
           result : in unsigned  (15 downto 0)
          );
end stimgen2;

architecture Behavioral of stimgen2 is

  signal DONE : boolean := false;
  signal FAILED: boolean := false;
begin
    -- Ctrl generator
    Clock1: process
        variable clktmp : std_logic := '0';
    begin
        while DONE /= true and FAILED /= true loop
            wait for CLOCK_PERIOD/2;
            clktmp := not clktmp;
            Clock <= clktmp;
        end loop;
        wait;
    end process;

    -- Apply stimulus signals and check result (LED signal values)
    CheckResult: process

    begin
```

```
        wait for CLOCK_PERIOD;
        -- Put system in reset
        Reset <= '1';
        A_Bus <= x"0000";
        B_Bus <= x"0000";

        wait for CLOCK_PERIOD;
        -- Take system out of reset
        Reset <= '0';
        wait for 0ns; -- force update of signal values

-- test three rounds
        for i in 0 to 3 loop

 -- test 1:
        Ctrl <= x"0";
        A_Bus <= x"F0F1";
        B_Bus <= x"F0F1";
        wait for 1.5*CLOCK_PERIOD;
        if  result = x"E1E2" then
           report "Passed test1";
        else
           report "failed test1" severity error;
           FAILED <= true;
        end if;

 -- test 2:
        Ctrl <= x"1";
        A_Bus <= x"F0F2";
        B_Bus <= x"F0F1";
        wait for 1.5*CLOCK_PERIOD;
         if result = x"0001" then
            report "Passed test2";
         else
            report "failed test2" severity error;
            FAILED <= true;
         end if;

-- test 3:
        Ctrl <= x"2";
        A_Bus <= x"B2A1";
        B_Bus <= x"C3D1";
        wait for 1.5*CLOCK_PERIOD;
        if result = x"8281" then
```

```
            report "Passed test3";
        else
            report "failed test3" severity error;
            FAILED <= true;
        end if;

-- test 4:
        Ctrl <= x"3";
        A_Bus <= x"234A";
        B_Bus <= x"A21D";
        wait for 1.5*CLOCK_PERIOD;
    if result <= x"A35F" then
            report "Passed test4";
        else
            report "failed test4" severity error;
            FAILED <= true;
        end if;

-- test 5:
        Ctrl <= x"4";
        A_Bus <= x"234A";
        B_Bus <= x"A21D";
        wait for 1.5*CLOCK_PERIOD;
        if result = x"8157" then
            report "Passed test5";
        else
            report "failed test5" severity error;
            FAILED <= true;
        end if;

 -- test 6:
        Ctrl <= x"5";
        A_Bus <= x"F21D";
        wait for 1.5*CLOCK_PERIOD;
        if result = x"0DE2" then
            report "Passed test6";
        else
            report "failed test6" severity error;
            FAILED <= true;
        end if;

  -- test 7:
        Ctrl <= x"6";
        A_Bus <= x"ABCD";
```

```
      wait for 1.5*CLOCK_PERIOD;
      if result = x"ABCD" then
         report "Passed test7";
      else
         report "failed test7" severity error;
         FAILED <= true;
      end if;

   -- test 8:
      Ctrl <= x"8";
      B_Bus <= x"12BC";
      wait for 1.5*CLOCK_PERIOD;
      if result = x"C12B" then
          report "Passed test8";
      else
          report "failed test8" severity error;
          FAILED <= true;
      end if;

   -- test 9:
      Ctrl <= x"9";
      B_Bus <= x"12BC";
      wait for 1.5*CLOCK_PERIOD;
      if result = x"2BC1" then
         report "Passed test9";
      else
         report "failed test9" severity error;
         FAILED <= true;
      end if;

   -- test 10:
      Ctrl <= x"A";
      B_Bus <= x"BD14";
      wait for 1.5*CLOCK_PERIOD;
      if result = x"D140" then
         report "Passed test10";
      else
         report "failed test10" severity error;
         FAILED <= true;
      end if;

   -- test 11:
      Ctrl <= x"B";
      B_Bus <= x"BD14";
```

```
        wait for 1.5*CLOCK_PERIOD;
        if result = x"0BD1" then
           report "Passed test11";
        else
           report "failed test11" severity error;
           FAILED <= true;
        end if;

   -- test 12:
        Ctrl <= x"C";
        A_Bus <= x"B2A1";
        wait for 1.5*CLOCK_PERIOD;
        if result = x"B270" then
           report "Passed test12";
        else
           report "failed test12" severity error;
           FAILED <= true;
        end if;

        wait for 5*CLOCK_PERIOD;

        end loop;  -- end round loop

        wait for 5*CLOCK_PERIOD;


        DONE <= true;
        Report "Test completed";
        wait;
    end process;

end Behavioral;
```

5.2.5 Crypto Coprocessor Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.NUMERIC_STD.all;

entity tb_Crypto is
  generic(N : positive := 16; M : positive :=4);
end tb_Crypto;
```

```
architecture struct of tb_Crypto is

  constant clk_period : time := 20ns;

  signal tb_clock, tb_reset : std_logic;
  signal tb_Instruction,tb_ResultOut, tb_ResultIn : unsigned(N-1 downto 0);
  signal tb_wrEnIn : std_logic;

  component CompleteCrypto is
    port( Instruction : in unsigned(N-1 downto 0);
          resetIn, clockIn, writeEnableIn  : in std_logic;
          ResultIn : in unsigned(N-1 downto 0);
          ResultOut : out unsigned(N-1 downto 0)
        );
  end component;
  component stimgen3
    generic (clock_period : time := clk_period);
    port( Instruction  : out unsigned (N-1 downto 0);
          wen    : out STD_LOGIC;
          clock : out STD_LOGIC;
          reset : out STD_LOGIC;
          resultin  : out unsigned (N-1 downto 0);
          resultout : in unsigned  (N-1 downto 0)
         );
    end component;

begin
  uut: CompleteCrypto
    port map( Instruction => tb_Instruction,
              resetIn => tb_reset,
              clockIn => tb_clock,
              writeEnableIn => tb_wrEnIn,
              ResultIn => tb_Resultout,
              ResultOut => tb_Resultout
            );

   my_stimgen: stimgen3
     port map(
               Instruction => tb_Instruction,
               wen => tb_wrEnIn,
               clock => tb_clock,
               reset => tb_reset,
               resultin => tb_ResultIn,
```

```
            resultout => tb_ResultOut
           );

end struct;
```

5.2.6 Stimulus Generator 3

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.NUMERIC_STD.all;

entity stimgen3 is

    generic (CLOCK_PERIOD : time);
    Port ( Instruction : out unsigned(15 downto 0);
           wen   : out STD_LOGIC;
           clock : out STD_LOGIC;
           reset : out STD_LOGIC;
           resultin  : out unsigned (15 downto 0);
           resultout : in unsigned  (15 downto 0)
          );
end stimgen3;

architecture Behavioral of stimgen3 is

  signal DONE : boolean := false;
  signal FAILED: boolean := false;
begin
    -- Ctrl generator
    Clock1: process
        variable clktmp : std_logic := '0';
    begin
        while DONE /= true and FAILED /= true loop
            wait for CLOCK_PERIOD/2;
            clktmp := not clktmp;
            Clock <= clktmp;
        end loop;
        wait;
    end process;

    -- Apply stimulus signals and check result (LED signal values)
    CheckResult: process
```

```
    begin
        wait for CLOCK_PERIOD;
        -- Put system in reset
        Reset <= '0';
        Instruction <= x"0000";
        ResultIn <= x"0000";

        wait for CLOCK_PERIOD;
        -- Take system out of reset
        Reset <= '1';
        wait for 0ns; -- force update of signal values

-- test three rounds
        for i in 0 to 3 loop

 -- test 1:
        Instruction <= x"054C";
        wen <= '1';
        wait for 1.5*CLOCK_PERIOD;
        if  resultout = x"E1E2" then
           report "Passed test1";
        else
           report "failed test1" severity error;
           FAILED <= true;
        end if;
        resultin <= resultout;

 -- test 2:
        Instruction <= x"4187";
         wen <= '1';
        wait for 1.5*CLOCK_PERIOD;
         if resultout = x"0001" then
            report "Passed test2";
         else
            report "failed test2" severity error;
            FAILED <= true;
         end if;
         resultin <= resultout;


-- test 3:
        Instruction <= x"80C0";
        wen <= '1';
```

```
        wait for 1.5*CLOCK_PERIOD;
        if resultout = x"8281" then
           report "Passed test3";
        else
           report "failed test3" severity error;
           FAILED <= true;
        end if;
        resultin <= resultout;

-- test 4:
        Instruction <= x"A393";
        wen <= '1';
        wait for 1.5*CLOCK_PERIOD;
    if resultout <= x"A35F" then
           report "Passed test4";
        else
           report "failed test4" severity error;
           FAILED <= true;
        end if;
        resultin <= resultout;

-- test 5:
        Instruction <= x"007A";
        wen <= '1';
        wait for 1.5*CLOCK_PERIOD;
        if resultout = x"8157" then
           report "Passed test5";
        else
           report "failed test5" severity error;
           FAILED <= true;
        end if;
        resultin <= resultout;

 -- test 6:
        Instruction <= x"173C";
        wen <= '1';
        wait for 1.5*CLOCK_PERIOD;
        if resultout = x"0DE2" then
           report "Passed test6";
        else
           report "failed test6" severity error;
           FAILED <= true;
        end if;
        resultin <= resultout;
```

```
-- test 7:
     Instruction <= x"7000";
     wen <= '0';
     wait for 1.5*CLOCK_PERIOD;
     if resultout = x"ABCD" then
        report "Passed test7";
     else
        report "failed test7" severity error;
        FAILED <= true;
     end if;
     resultin <= resultout;

-- test 8:
     Instruction <= x"2CA9";
     wen <= '1';
     wait for 1.5*CLOCK_PERIOD;
     if resultout = x"C12B" then
         report "Passed test8";
     else
         report "failed test8" severity error;
         FAILED <= true;
     end if;
     resultin <= resultout;

-- test 9:
     Instruction <= x"7000";
     wen <= '0';
     wait for 1.5*CLOCK_PERIOD;
     if resultout = x"2BC1" then
        report "Passed test9";
     else
        report "failed test9" severity error;
        FAILED <= true;
     end if;
     resultin <= resultout;

-- test 10:
     Instruction <= x"C922";
     wen <= '1';
     wait for 1.5*CLOCK_PERIOD;
     if resultout = x"D140" then
        report "Passed test10";
     else
```

```
            report "failed test10" severity error;
            FAILED <= true;
        end if;
        resultin <= resultout;

        wait for 5*CLOCK_PERIOD;

        end loop;  -- end round loop

        wait for 5*CLOCK_PERIOD;


        DONE <= true;
        Report "Test completed";
        wait;
    end process;

end Behavioral;
```

5.3 Log Files

5.3.1 Combinational Logic Log

run 1000ns

Note: Passed test1 Time: 70 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test2 Time: 100 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test3 Time: 130 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test4 Time: 160 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test5 Time: 190 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test6 Time: 220 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/

Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test7
Time: 250 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult
File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/
Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test8
Time: 280 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult
File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/
Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test9
Time: 310 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult
File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/
Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test10
Time: 340 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult
File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/
Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test11
Time: 370 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult
File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/
Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Error: failed test12
Time: 400 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult
File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/
Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test1
Time: 530 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult
File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/
Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test2
Time: 560 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult
File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/
Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test3
Time: 590 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult
File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/
Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test4
Time: 620 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult
File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/
Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test5
Time: 650 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult
File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/
Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test6
Time: 680 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult
File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/
Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test7
Time: 710 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult
File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/
Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test8
Time: 740 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult
File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/
Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test9
Time: 770 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult

File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test10 Time: 800 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test11 Time: 830 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Error: failed test12 Time: 860 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd Note: Passed test1 Time: 990 ns Iteration: 0 Process: /tb_combLogic/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen.vhd

5.3.2 Synchronous Logic Log

run 1000ns

Note: Passed test1 Time: 70 ns Iteration: 0 Process: /tb_synchReg/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen2.vhd Note: Passed test2 Time: 100 ns Iteration: 0 Process: /tb_synchReg/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen2.vhd Note: Passed test3 Time: 130 ns Iteration: 0 Process: /tb_synchReg/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen2.vhd Note: Passed test4 Time: 160 ns Iteration: 0 Process: /tb_synchReg/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen2.vhd Note: Passed test5 Time: 190 ns Iteration: 0 Process: /tb_synchReg/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen2.vhd Note: Passed test6 Time: 220 ns Iteration: 0 Process: /tb_synchReg/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen2.vhd Note: Passed test7 Time: 250 ns Iteration: 0 Process: /tb_synchReg/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen2.vhd Note: Passed test8 Time: 280 ns Iteration: 0 Process: /tb_synchReg/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/Assignment2/Assignment2.srcs/sources_1/new/stimgen2.vhd Note: Passed test9

Time: 310 ns Iteration: 0 Process: /tb_synchReg/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/ Assignment2/Assignment2.srcs/sources_1/new/stimgen2.vhd Note: Passed test10 Time: 340 ns Iteration: 0 Process: /tb_synchReg/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/ Assignment2/Assignment2.srcs/sources_1/new/stimgen2.vhd Note: Passed test11 Time: 370 ns Iteration: 0 Process: /tb_synchReg/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/ Assignment2/Assignment2.srcs/sources_1/new/stimgen2.vhd Error: failed test12 Time: 400 ns Iteration: 0 Process: /tb_synchReg/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/ Assignment2/Assignment2.srcs/sources_1/new/stimgen2.vhd

5.3.3 Crypto Coprocessor Log

run 1000ns

Error: failed test1 Time: 70 ns Iteration: 0 Process: /tb_Crypto/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/ Assignment2/Assignment2.srcs/sources_1/new/stimgen3.vhd Error: failed test2 Time: 100 ns Iteration: 0 Process: /tb_Crypto/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/ Assignment2/Assignment2.srcs/sources_1/new/stimgen3.vhd Error: failed test3 Time: 130 ns Iteration: 0 Process: /tb_Crypto/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/ Assignment2/Assignment2.srcs/sources_1/new/stimgen3.vhd Error: failed test4 Time: 160 ns Iteration: 0 Process: /tb_Crypto/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/ Assignment2/Assignment2.srcs/sources_1/new/stimgen3.vhd Error: failed test5 Time: 190 ns Iteration: 0 Process: /tb_Crypto/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/ Assignment2/Assignment2.srcs/sources_1/new/stimgen3.vhd Error: failed test6 Time: 220 ns Iteration: 0 Process: /tb_Crypto/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/ Assignment2/Assignment2.srcs/sources_1/new/stimgen3.vhd Error: failed test7 Time: 250 ns Iteration: 0 Process: /tb_Crypto/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/ Assignment2/Assignment2.srcs/sources_1/new/stimgen3.vhd Error: failed test8 Time: 280 ns Iteration: 0 Process: /tb_Crypto/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/ Assignment2/Assignment2.srcs/sources_1/new/stimgen3.vhd Error: failed test9 Time: 310 ns Iteration: 0 Process: /tb_Crypto/my_stimgen/CheckResult File:

/home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/ Assignment2/Assignment2.srcs/sources_1/new/stimgen3.vhd Error: failed test10 Time: 340 ns Iteration: 0 Process: /tb_Crypto/my_stimgen/CheckResult File: /home/mlenehan/Documents/Michael/ECE/ECE4/EE496/Assignment2/Code/ Assignment2/Assignment2.srcs/sources_1/new/stimgen3.vhd