EE496 - VHDL Design & Synthesis Assignment 2

Section 1: Combinational Logic

1.1 ALU Design

The first step of this assignment is to design an Arithmetic Logic Unit (ALU) using vhdl. The ALU is to be a 16-bit ALU with the following functionality:

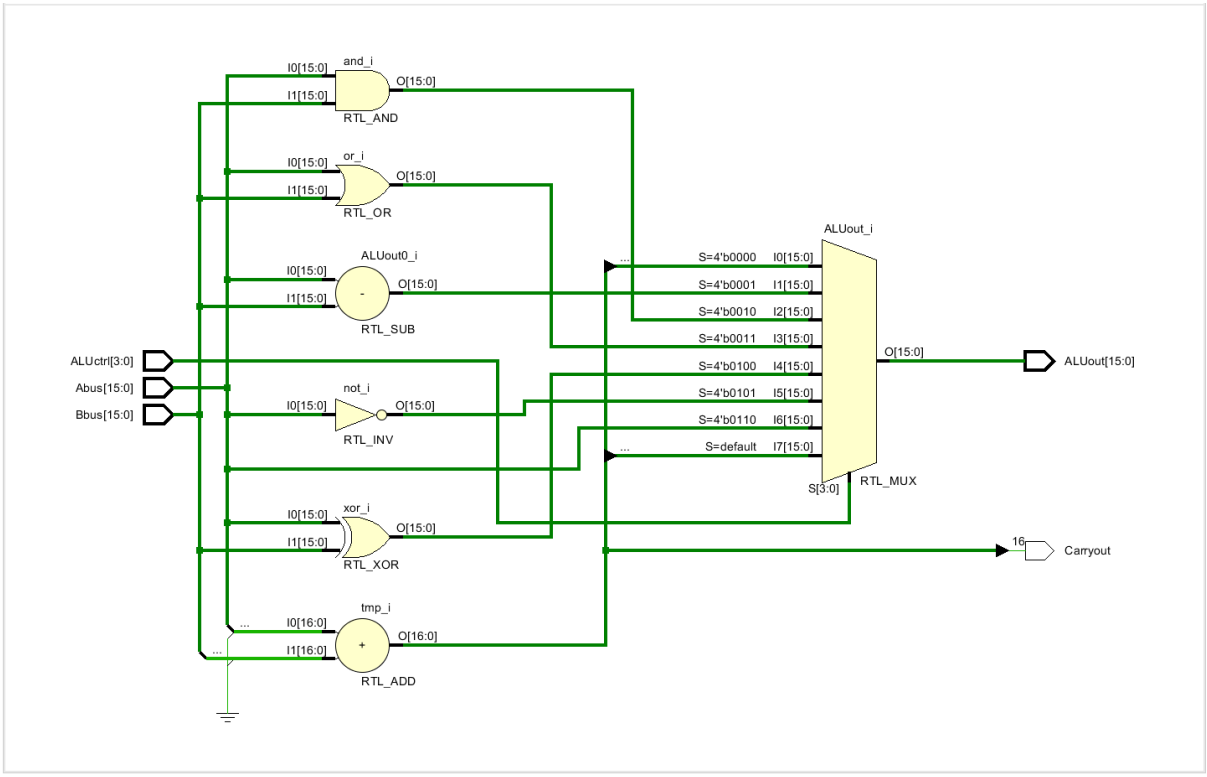
ALUctrl	Mnemonic	Instruction Function
0000	ADD	ALUout = Abus + Bbus
0001	SUB	ALUout = Abus - Bbus
0010	AND	ALUout = Abus & Bbus
0011	OR	ALUout = Abus | Bbus
0100	XOR	ALUout = Abus ^ Bbus
0101	NOT	ALUout = ~ Abus
0110	MOV	ALUout = Abus

This achieved using a singular vhdl behavioural file including all this functionality. The design has 3 inputs (A, B, and ALU control) and a single output (ALU out). For this implementation a carryout was also included in case it may be needed later. The following code achieved the required functionality.

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

entity ALU is
   Port ( ALUctrl : in STD_LOGIC_VECTOR (3 downto 0);
          Abus : in unsigned (15 downto 0);
          Bbus : in unsigned (15 downto 0);
          ALUout : out unsigned (15 downto 0);
          Carryout : out STD_LOGIC);
end ALU;
```

```
architecture Behavioral of ALU is
   signal tmp: unsigned (16 downto 0);
begin

 process(Abus,Bbus,ALUctrl)
 begin
  case(ALUctrl) is
  when "0000" => -- Addition
   ALUout <= tmp(15 downto 0);  -- ALU_out <= A + B ;
  when "0001" => -- Subtraction
   ALUout <= Abus - Bbus ;
  when "0010" => -- Logical and
   ALUout <= Abus and Bbus;
  when "0011" => -- Logical or
   ALUout <= Abus or Bbus;
  when "0100" => -- Logical xor
   ALUout <= Abus xor Bbus;
  when "0101" => -- Logical not
   ALUout <= not Abus;
  when "0110" => -- Logical mov
   ALUout <= Abus;

  when others => ALUout <= tmp(15 downto 0) ; -- ALUout <= A + B;
  end case;
 end process;

 tmp <= ('0' & Abus) + ('0' & Bbus);
 Carryout <= tmp(16); -- Carryout flag
end Behavioral;
```

This process outputs the correct value for each of the functions listed above, and the following is the elaborated design of this code:



1.2 Shifter in VHDL

The next step is to design a shifter in VHDL. To do this the process is similar to before, a new vhdl file is used and the entire functionality is designed within the one file. The shifter must have the following functionality:

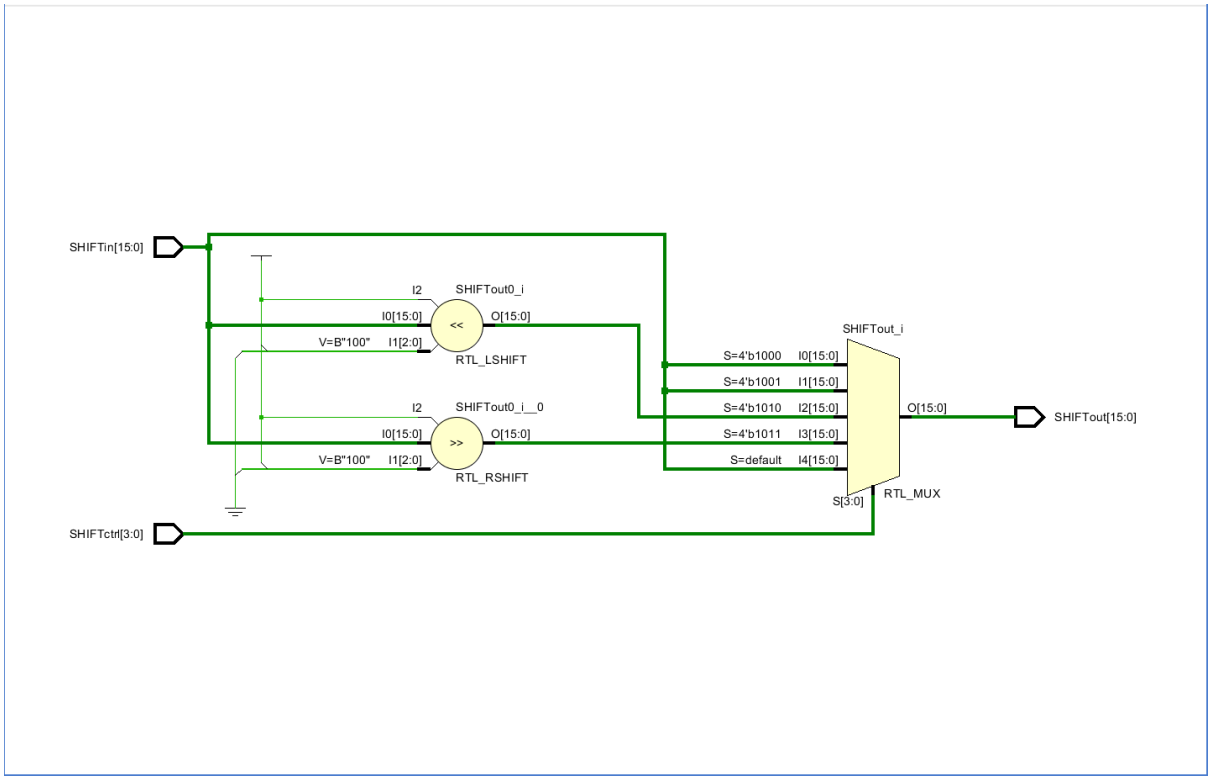
SHIFT ctrl	Instruction Function
1000	Rotate right 4 bits
1001	Rotate left 4 bits
1010	Shift left logic 4 bits
1011	Shift right logic 4 bit

The code this time uses two inputs, shifter in, and shifter control, as well as a single output, shifter out. The shifter must work with 16-bit numbers so both shifter in and out are 16 bit busses. Shift control is a 4-bit bus. The ieee.numeric_std functions for shifting and rotating are used. Follows is the code to implement this functionality:

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

entity SHIFTER is
    Port ( SHIFTin : in unsigned (15 downto 0);
           SHIFTctrl : in unsigned (3 downto 0);
           SHIFTout : out unsigned (15 downto 0));
end SHIFTER;

architecture Behavioral of SHIFTER is

begin
 process(SHIFTin,SHIFTctrl)
 begin
 case(SHIFTctrl) is
  when "1000" => -- Rotate right 4 bits
   SHIFTout <= SHIFTin ror(4);
  when "1001" => -- Rotate left 4 bits
   SHIFTout <= SHIFTin rol(4);
  when "1010" => --  Shift left 4 bits
   SHIFTout <= SHIFTin sll(4);
  when "1011" => --  Shift right 4 bits
   SHIFTout <= SHIFTin srl(4);


  when others => SHIFTout <= SHIFTin ; -- Shift out = shift in;

  end case;
end process;

end Behavioral;
```

This code will take a 16-bit input and output the number rotated or shifted by 4 bits in either direction. If the control signal input does not match any of the available functions it will simply output the same number with no changes.

Follows is the schematic as created by Vivado:



1.3 Non-Linear Lookup Operations in VHDL

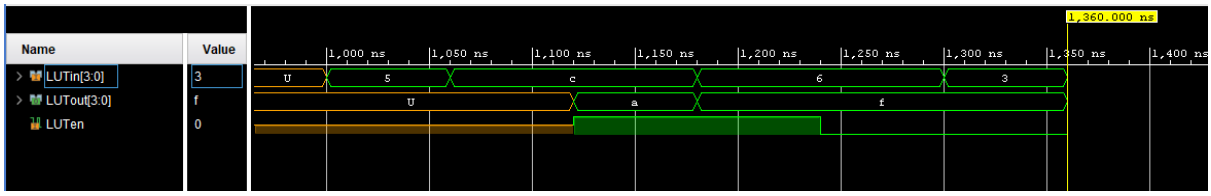
The next design that must be implemented is that of a non-linear lookup operation. To do this two substitution blocks will be used both of which implement lookup tables to give a non-linear but constant output for each input. For the purposes of this assignment this will be done on only the least significant byte of the input 16-bit value. In this implementation each s-block will be their own design within their own vhdl file. The lookup table is implemented with a switch statement and only active if the lookup enable line is active.

Follows is the code for the first s-block (SBoxOne):

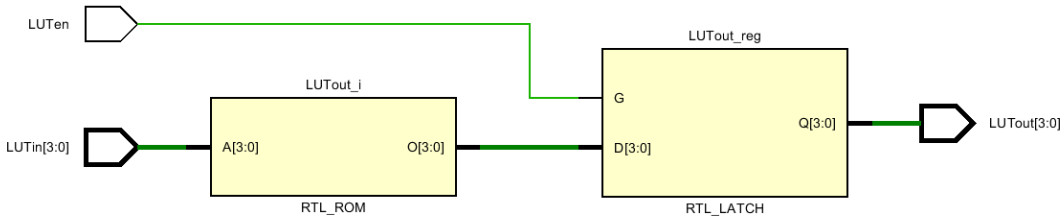
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity SBoxOne is
   Port ( LUTin : in unsigned (3 downto 0);
          LUTout : out unsigned (3 downto 0);
          LUTen : in STD_LOGIC);
end SBoxOne;

architecture Behavioral of SBoxOne is
```

```
begin

 process(LUTin,LUTen)
 begin

if LUTen = '1' then
  case(LUTin) is
  when "0000" => -- 0
   LUTout <= "0001";
  when "0001" => -- 1
   LUTout <= "1011";
  when "0010" => --  2
   LUTout <= "1001";
  when "0011" => --  3
   LUTout <= "1100";

  when "0100" => -- 4
   LUTout <= "1101";
  when "0101" => -- 5
   LUTout <= "0110";
  when "0110" => --  6
   LUTout <= "1111";
  when "0111" => --  7
   LUTout <= "1100";

  when "1000" => -- 8
   LUTout <= "1110";
  when "1001" => -- 9
   LUTout <= "1000";
  when "1010" => --  10
   LUTout <= "0111";
  when "1011" => --  11
   LUTout <= "0100";

  when "1100" => -- 12
   LUTout <= "1010";
  when "1101" => -- 13
   LUTout <= "0010";
  when "1110" => --  14
   LUTout <= "0101";
  when "1111" => --  15
   LUTout <= "0000";

  when others => Lutout <= "0000" ; -- this should not be possible
  end case;

end if;
end process;

end Behavioral;
```

This code allows for any 4-bit input to have a predefined output. It also will not update the output unless the enable line is 1. A couple of initial tests show this functionality:



We can see here that when the input is a value and the enable line is not active the output does not change. Once the enable is active the value changes to what was defined in the lookup table. It then updates as the input changes but once the enable is set to 0 the output no longer updates with changes to input.



We can also see that the schematic generated by Vivado shows the output as a latch with Lookup enable being the enable for the latch.

Near identical code is used to implement the second s-block. The only difference being the file name and the output values defined in the switch statement:

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

entity SBoxTwo is
   Port ( LUTin : in unsigned (3 downto 0);
          LUTout : out unsigned (3 downto 0);
          LUTen : in STD_LOGIC);
end SBoxTwo;

architecture Behavioral of SBoxTwo is

begin

 process(LUTin,LUTen)
 begin

if LUTen = '1' then
```

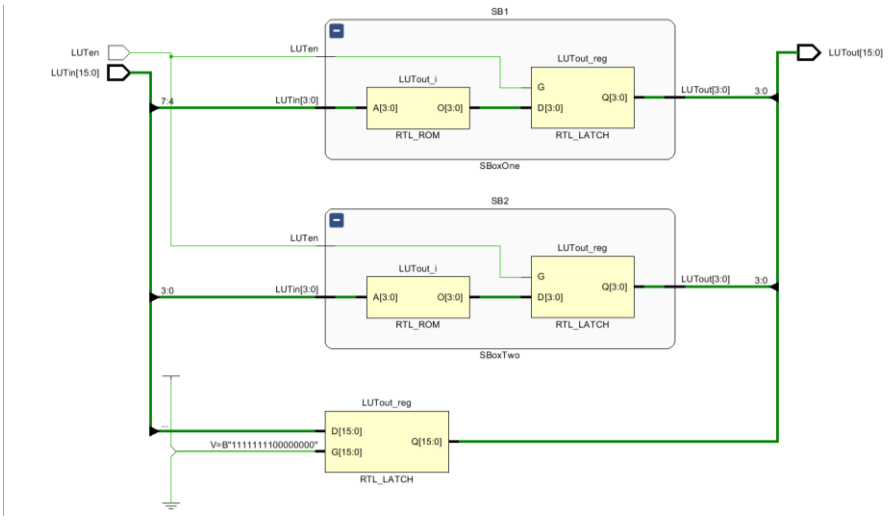
```
 case(LUTin) is
 when "0000" => -- 0
  LUTout <= "1111";
 when "0001" => -- 1
  LUTout <= "0000";
 when "0010" => --  2
  LUTout <= "1101";
 when "0011" => --  3
  LUTout <= "0111";

 when "0100" => -- 4
  LUTout <= "1011";
 when "0101" => -- 5
  LUTout <= "1110";
 when "0110" => --  6
  LUTout <= "0101";
 when "0111" => --  7
  LUTout <= "1010";

 when "1000" => -- 8
  LUTout <= "1001";
 when "1001" => -- 9
  LUTout <= "0010";
 when "1010" => --  10
  LUTout <= "1100";
 when "1011" => --  11
  LUTout <= "0001";

 when "1100" => -- 12
  LUTout <= "0011";
 when "1101" => -- 13
  LUTout <= "0100";
 when "1110" => --  14
  LUTout <= "1000";
 when "1111" => --  15
  LUTout <= "0110";

 when others => Lutout <= "0000" ; -- this should not be possible
 end case;

end if;
end process;

end Behavioral;
```

This code runs and performs identically to the initial s block, bar the output value which is the point of these non-linear operations. The next step is to design a code that implements these blocks to run the operation on a 16-bit input.

To do this a new vhdl design is implemented using both s blocks as components, this simply inputs the least significant 4 bits to the s block 2 and the next 4 least significant bits to s block 1. The least significant 4 bits of the output is the output of s block one and the next four least significant bits of the output are the output of s block 1 and the final 8 bits are simply the most significant 8 bits of the input.

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


entity LUSub is
   Port ( LUTin : in unsigned (15 downto 0);
          LUTen : in STD_LOGIC;
          LUTout : out unsigned (15 downto 0));
end LUSub;

architecture Behavioral of LUSub is

component SBoxOne
port(   LUTin : in unsigned (3 downto 0);
       LUTout : out unsigned (3 downto 0);
       LUTen : in STD_LOGIC);
end component;
component SBoxTwo
port(   LUTin : in unsigned (3 downto 0);
       LUTout : out unsigned (3 downto 0);
       LUTen : in STD_LOGIC);
end component;

signal sb1in : unsigned (3 downto 0);

begin

SB1: SBoxOne port map (LUTin(7 downto 4), LUTout(7 downto 4), LUTen);
SB2: SBoxTwo port map (LUTin(3 downto 0), LUTout(3 downto 0), LUTen);

process(LUTin, LUTen)
begin

   LUTout(15 downto 8) <= LUTin(15 downto 8);


end process;

end Behavioral;
```

This gives the following elaborated design:



1.4 Structural VHDL model

The next step is to put all the components designed together in a single structure. This will therefore need to accept both a 16-bit A input and a 16-bit B input, a control input and a 16-bit output. The control input will be used to control what element of the structure the result comes from and what function the components will perform. A simple way to do this is to use the fact that the control signals for the ALU and Shifter are already values that do not overlap, therefore we can check if the control signal is within the ALU range or the Shifter range and output from the active component. The binary number 0111 was not used for either so this can be set as the activating control signal for the lookup table.

```
  if CTRL = "0111" then
     LUTenable <= '1';
     Result <= LUoutput;
     --Lookup table component;

  elsif CTRL(3) = '0' then
  --ALU code here
     ALUcontrol(3 downto 0) <= CTRL(3 downto 0);
     Result <= ALUoutput;

  elsif CTRL(2) = '0' then
  --Shifter component code
       SHIFTcontrol <= CTRL;
       Result <= Shifteroutput;

  else
     Result <= (others => '0');

  end if;
```

To check which of the ranges the control signal is the most significant bit and second most significant bit can be used. If the most significant bit is 0 we know that it is in the range of 0 – 0111 which includes all of the ALU functions and now the LUT also, however if this is not the case and the second most significant bit is 0, i.e. the most significant bits are 10 this range includes 1000 – 1011 which includes all shifter functionality. Therefore we can use these to output the correct result.

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


entity FullModel is
   Port ( CTRL : in unsigned (3 downto 0);
          Abus : in unsigned (15 downto 0);
          Bbus : in unsigned (15 downto 0);
          Result : out unsigned (15 downto 0));
end FullModel;

architecture Behavioral of FullModel is

component LUSub
port(   LUTin : in unsigned (15 downto 0);
       LUTen : in STD_LOGIC;
       LUTout : out unsigned (15 downto 0));
end component;
```

```
component ALU
port( ALUctrl : in unsigned (3 downto 0);
     Abus : in unsigned (15 downto 0);
     Bbus : in unsigned (15 downto 0);
     ALUout : out unsigned (15 downto 0));
end component;

component SHIFTER
Port ( SHIFTin : in unsigned (15 downto 0);
      SHIFTctrl : in unsigned (3 downto 0);
      SHIFTout : out unsigned (15 downto 0));
end component;

signal LUTenable : STD_LOGIC;
signal ALUcontrol : unsigned(3 downto 0);
signal SHIFTcontrol : unsigned(3 downto 0);
signal ALUoutput : unsigned(15 downto 0);
signal Shifteroutput : unsigned(15 downto 0);
signal LUoutput : unsigned(15 downto 0);
signal Cout : STD_LOGIC;

begin

myALU : ALU port map(ALUcontrol, Abus, Bbus, ALUoutput);
myShifter : SHIFTER port map(Bbus, SHIFTcontrol, Shifteroutput);
myLUT : LUSub port map(Abus, LUTenable, LUoutput);

process
begin
wait for 10ns;
LUTenable <= '0';
SHIFTcontrol <= "0000";

   if CTRL = "0111" then
      LUTenable <= '1';
      Result <= LUoutput;
      --Lookup table component;

   elsif CTRL(3) = '0' then
   --ALU code here
      ALUcontrol(3 downto 0) <= CTRL(3 downto 0);
      Result <= ALUoutput;

   elsif CTRL(2) = '0' then
   --Shifter component code
        SHIFTcontrol <= CTRL;
        Result <= Shifteroutput;

   else
      Result <= (others => '0');

   end if;

end process;
end Behavioral;
```

1.5 VHDL Test bench for Structural Design

The final step of this section is to design a test bench for this VHDL model. It would be extremely time consuming to test for every possible input in this situation so instead the test bench is designed to test each function for two different sets of inputs. The correct output was hardcoded into the test bench to test the output against. For this implementation a for loop was used and within each loop a single function of the model was tested, first for the input A=00CF and B=004D and then again for inputs A=CA72 and B=35B5.

If the test was successful there is an putput to the console specifying the test and its success.

So for the addition of the first inputs the code is as follows:

```
Abus <= "0000000011001111";  --CF --207
   Bbus <= "0000000001001101";  --4D --77
   wait for 40ns;

   if CTRL = "0000" then
      if Result = "0000000100011100" then
         report "Opperation 1 Test 1 is successful";
      else
         report "Opperation 1 Test 1 is failed";
      end if;
   end if;
```

This sets the inputs and then checks if the function is addition based on the control value, and if so will test if the addition was successful.

This is then repeated for every function and repeated again for the second set of inputs so the final code is as follows:

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity FullModel_tb is
end;

architecture bench of FullModel_tb is

 component FullModel
    Port ( CTRL : in unsigned (3 downto 0);
          Abus : in unsigned (15 downto 0);
          Bbus : in unsigned (15 downto 0);
          Result : out unsigned (15 downto 0));
 end component;

 signal CTRL: unsigned (3 downto 0);
 signal Abus: unsigned (15 downto 0);
 signal Bbus: unsigned (15 downto 0);
 signal Result: unsigned (15 downto 0);
```

```
begin

  uut: FullModel port map ( CTRL   => CTRL,
                            Abus   => Abus,
                            Bbus   => Bbus,
                            Result => Result );

process
  begin

    Abus <= "0000000000000000";
    Bbus <= "0000000000000000";
    CTRL <= "0000";
wait for 10ns;



sub_loop: for I in 0 to 12 loop

    Abus <= "0000000011001111";  --CF --207
    Bbus <= "0000000001001101";  --4D --77
    wait for 40ns;

    if CTRL = "0000" then
       if Result = "0000000100011100" then
          report "Opperation 1 Test 1 is successful";
       else
          report "Opperation 1 Test 1 is failed";
       end if;
    end if;

     if CTRL = "0001" then
       if Result = "0000000010000010" then
          report "Opperation 2 Test 1 is successful";
       else
          report "Opperation 2 Test 1 is failed";
       end if;
    end if;

    if CTRL = "0010" then
      if Result = (Abus and Bbus) then
         report "Opperation 3 Test 1 is successful";
      else
         report "Opperation 3 Test 1 is failed";
      end if;
   end if;

   if CTRL = "0011" then
      if Result = "0000000011001111" then
         report "Opperation 4 Test 1 is successful";
      else
         report "Opperation 4 Test 1 is failed";
      end if;
   end if;

   if CTRL = "0100" then
      if Result = "0000000010000010" then
         report "Opperation 5 Test 1 is successful";
      else
```

```
        report "Opperation 5 Test 1 is failed";
    end if;
  end if;

  if CTRL = "0101" then
    if Result = "1111111100110000" then
      report "Opperation 6 Test 1 is successful";
    else
      report "Opperation 6 Test 1 is failed";
    end if;
  end if;

  if CTRL = "0110" then
    if Result = "0000000011001111" then
      report "Opperation 7 Test 1 is successful";
    else
      report "Opperation 7 Test 1 is failed";
    end if;
  end if;

  if CTRL = "0111" then
    if Result = "0000000010100110" then
      report "Opperation 8 Test 1 is successful";
    else
      report "Opperation 8 Test 1 is failed";
    end if;
  end if;

  if CTRL = "1000" then
    if Result = "1101000000000100" then
      report "Opperation 9 Test 1 is successful";
    else
      report "Opperation 9 Test 1 is failed";
    end if;
  end if;

  if CTRL = "1001" then
    if Result = "0000010011010000" then
      report "Opperation 10 Test 1 is successful";
    else
      report "Opperation 10 Test 1 is failed";
    end if;
  end if;

  if CTRL = "1010" then
    if Result = "0000010011010000" then
      report "Opperation 11 Test 1 is successful";
    else
      report "Opperation 11 Test 1 is failed";
    end if;
  end if;

  if CTRL = "1011" then
    if Result = "0000000000000100" then
      report "Opperation 12 Test 1 is successful";
    else
      report "Opperation 12 Test 1 is failed";
    end if;
end if;
```

```
Abus <= "1100101001110010";  --CA72 --51,876
Bbus <= "0011010110110101";  --35B5 --13,749
wait for 40ns;

if CTRL = "0000" then
   if Result = "0000000000100111" then
      report "Opperation 1 Test 2 is successful";
   else
      report "Opperation 1 Test 2 is failed";
   end if;
end if;

 if CTRL = "0001" then
   if Result = "1001010010111101" then
      report "Opperation 2 Test 2 is successful";
   else
      report "Opperation 2 Test 2 is failed";
   end if;
end if;

if CTRL = "0010" then
  if Result = "0000000000110000" then
     report "Opperation 3 Test 2 is successful";
  else
     report "Opperation 3 Test 2 is failed";
  end if;
end if;

if CTRL = "0011" then
  if Result = "1111111111110111" then
     report "Opperation 4 Test 2 is successful";
  else
     report "Opperation 4 Test 2 is failed";
  end if;
end if;

if CTRL = "0100" then
  if Result = "1111111111000111" then
     report "Opperation 5 Test 2 is successful";
  else
     report "Opperation 5 Test 2 is failed";
  end if;
end if;

if CTRL = "0101" then
  if Result = "0011010110001101" then
     report "Opperation 6 Test 2 is successful";
  else
     report "Opperation 6 Test 2 is failed";
  end if;
end if;

if CTRL = "0110" then
  if Result = "1100101001110010" then
     report "Opperation 7 Test 2 is successful";
  else
     report "Opperation 7 Test 2 is failed";
  end if;
```

```
  end if;

  if CTRL = "0111" then
     if Result = "1100101000111101" then
        report "Opperation 8 Test 2 is successful";
     else
        report "Opperation 8 Test 2 is failed";
     end if;
  end if;

  if CTRL = "1000" then
     if Result = "0101001101011011" then
        report "Opperation 9 Test 2 is successful";
     else
        report "Opperation 9 Test 2 is failed";
     end if;
  end if;

  if CTRL = "1001" then
     if Result = "0101101101010011" then
        report "Opperation 10 Test 2 is successful";
     else
        report "Opperation 10 Test 2 is failed";
     end if;
  end if;

  if CTRL = "1010" then
     if Result = "0101101101010000" then
        report "Opperation 11 Test 2 is successful";
     else
        report "Opperation 11 Test 2 is failed";
     end if;
  end if;

  if CTRL = "1011" then
     if Result = "0000001101011011" then
        report "Opperation 12 Test 2 is successful";
     else
        report "Opperation 12 Test 2 is failed";
     end if;
end if;



    wait for 40ns;
    CTRL <= CTRL + 1;
end loop sub_loop;

end process;

end;
```

The following is the output to console if the design is working correctly:

```
Note: Opperation 1 Test 1 is successful
Time: 50 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 1 Test 2 is successful
Time: 90 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 2 Test 1 is successful
Time: 170 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 2 Test 2 is successful
Time: 210 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 3 Test 1 is successful
Time: 290 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 3 Test 2 is successful
Time: 330 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 4 Test 1 is successful
Time: 410 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 4 Test 2 is successful
Time: 450 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 5 Test 1 is successful
Time: 530 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 5 Test 2 is successful
Time: 570 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 6 Test 1 is successful
Time: 650 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 6 Test 2 is successful
Time: 690 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 7 Test 1 is successful
Time: 770 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 7 Test 2 is successful
Time: 810 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 8 Test 1 is successful
Time: 890 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 8 Test 2 is successful
Time: 930 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 9 Test 1 is successful
Time: 1010 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 9 Test 2 is successful
Time: 1050 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 10 Test 1 is successful
Time: 1130 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 10 Test 2 is successful
Time: 1170 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 11 Test 1 is successful
Time: 1250 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 11 Test 2 is successful
Time: 1290 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 12 Test 1 is successful
Time: 1370 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
Note: Opperation 12 Test 2 is successful
Time: 1410 ns  Iteration: 0  Process: /FullModel_tb/line__30  File: C:/Users/ciara/Documents/College/HDL/project_2/project_2.srcs/sim_1/new/Fullmodel_bench.vhd
INFO: [USF-XSim-96] XSim completed. Design snapshot 'FullModel_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1600ns
```

Test Results:

Test Index	*OpCode (Hex)*	*A input*	*B input*	*Expected Output*	*Actual Output*	*Pass/Fail*
1	0	CF	4D	11C	11C	Pass
2	0	CA72	35B5	27	27	Pass
3	1	CF	4D	82	82	Pass
4	1	CA72	35B5	94BD	94BD	Pass
5	2	CF	4D	4D	4D	Pass
6	2	CA72	35B5	30	30	Pass
7	3	CF	4D	CF	CF	Pass
8	3	CA72	35B5	FFF7	FFF7	Pass
9	4	CF	4D	82	82	Pass
10	4	CA72	35B5	FFC7	FFC7	Pass
11	5	CF	4D	FF30	FF30	Pass
12	5	CA72	35B5	358D	358D	Pass
13	6	CF	4D	CF	CF	Pass
14	6	CA72	35B5	CA72	CA72	Pass

15	12	CF	4D	A6	A6	Pass
16	12	CA72	35B5	CA3D	CA3D	Pass
17	8	CF	4D	D004	D004	Pass
18	8	CA72	35B5	535B	535B	Pass
19	9	CF	4D	4D0	4D0	Pass
20	9	CA72	35B5	5B53	5B53	Pass
21	10	CF	4D	4D0	4D0	Pass
22	10	CA72	35B5	5B50	5B50	Pass
23	11	CF	4D	4	4	Pass
24	11	CA72	35B5	35B	35B	Pass

Section 2: Synchronous Logic Design

2.1 Registers

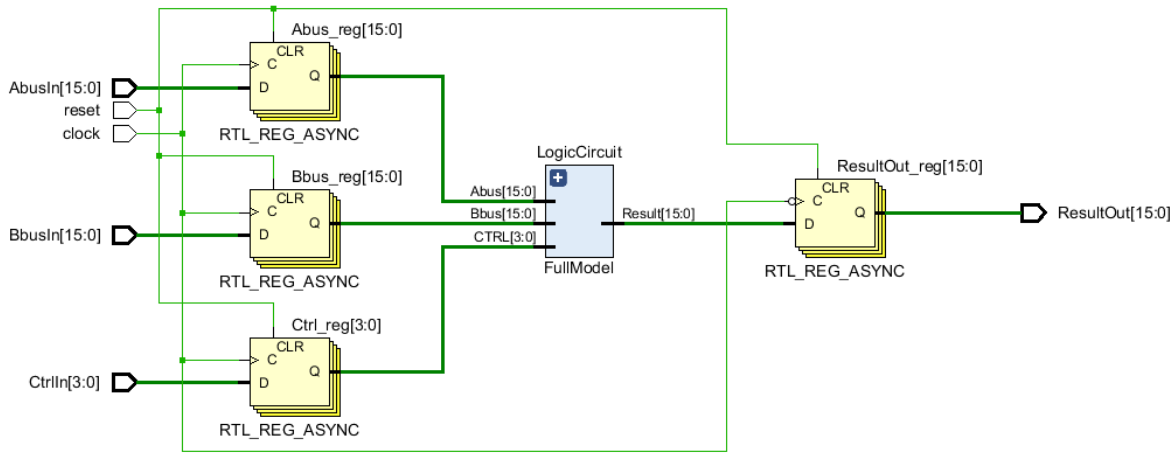
With the combinational logic of the cryptography hardware complete the next step is too look at making this hardware more functional with synchronous system design. To start the Input and Output should be only released via a clock signal. In this implementation the inputs will be released into the combinational logic and the output from the combinational logic will be registered on the falling edge of the clock signal. This means that over a single clock period the input and output of the hardware will update.

To do this a new VHDL file is created that specifies this functionality, this file stores the values as inputs and outputs entering and leaving the combinational logic and are only set on the rising and falling edge of the clock. The reset will also set these values to zero regardless of the clock signal.

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

entity Registers is
   Port ( AbusIn : in unsigned (15 downto 0);
          BbusIn : in unsigned (15 downto 0);
          CtrlIn : in unsigned (3 downto 0);
          reset : in STD_LOGIC;
          clock : in STD_LOGIC;
          ResultOut : out unsigned (15 downto 0));
end Registers;

architecture Behavioral of Registers is
component FullModel
Port ( CTRL : in unsigned (3 downto 0);
          Abus : in unsigned (15 downto 0);
          Bbus : in unsigned (15 downto 0);
          Result : out unsigned (15 downto 0));
end component;
```

```
signal Result : unsigned(15 downto 0);
signal Abus : unsigned (15 downto 0);
signal Bbus : unsigned (15 downto 0);
signal Ctrl : unsigned (3 downto 0);
begin

LogicCircuit : FullModel port map(Ctrl, Abus, Bbus, Result);

process(clock, reset)
begin
    if reset = '1' then
       ResultOut <= "0000000000000000";
       Abus <= "0000000000000000";
       Bbus <= "0000000000000000";
       Ctrl <= "0000";
    elsif rising_edge(clock) then
          Abus <= AbusIn;
          Bbus <= BbusIn;
          Ctrl <= CtrlIn;
    end if;

    if reset = '1' then
        ResultOut <= "0000000000000000";
    elsif falling_edge(clock) then
          ResultOut <= Result;
    end if;

end process;

end Behavioral;
```

It can be seen above that if statements are used to check the state of the clock and reste and the registers are set accordingly. Follows is the Schematic generated from this code:



2.2 Memory

The advantages of synchronous design is the ability to model memory elements in the form of a registry file. To do this a new entity is created with the register file inputs and outputs and the functionality of the memory. In this case the memory has 16 separate 16-bit registers, with the ability to write two one and read from two on the rising edge of a clock signal.

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

entity Register_File is
   port
   (
   Abus : out std_logic_vector(15 downto 0);
   Bbus : out std_logic_vector(15 downto 0);
   result : in  std_logic_vector(15 downto 0);
   writeEnable : in std_logic;
   regASel : in std_logic_vector(3 downto 0);
   regBSel : in std_logic_vector(3 downto 0);
   writeRegSel : in std_logic_vector(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 std_logic_vector(15 downto 0);
signal registers : 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

   regFile: process(clock, reset)
   begin
   if reset = '1' then
      Abus <= "0000000000000000";
      Bbus <= "0000000000000000";
```

```
  else
    if rising_edge(clock) then
      if(writeEnable = '1') then
        registers(to_integer(unsigned(writeRegSel))) <= result;
      end if;
      Abus <= registers(to_integer(unsigned(regASel)));
      Bbus <= registers(to_integer(unsigned(regBSel)));
      if(WriteRegSel = regASel) then
        Abus <= result;
      elsif(WriteRegSel = regBSel) then
        Bbus <= result;
      end if;
    end if;
  end if;
end process;
end behavioral;
```

Here is the code for such a file. The Memory type is simply initializing the values within the register value. The process code is fairly simple and simply on a rising edge will write in the new value to the position specified by the writeRegSel if writing is enabled, and then reads the values output to Abus and Bbus from the positions specified by regAsel, and regBsel.

2.3 Complete Crypto Coprocessor

Finally, the Register file needs to be implemented with the rest of the code in order to model a complete crypto coprocessor. Doing this requires simply interfacing the two components together in a new vhdl file. There is not much code to write as once the two components are linked the clock signal into the register file controls what happens.

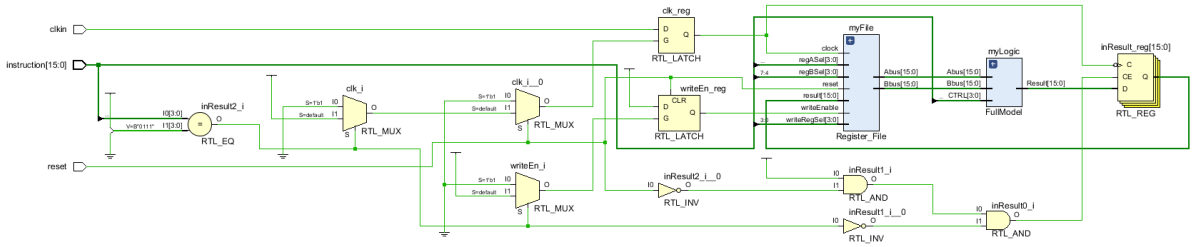
Two things to be aware of is the implementation of a no op code, this means no operation is to take place, this can be implemented by having an empty if statement for when the op code is received and then an else system that allows the clock signal through to the register file, this way if the nop code is active the register file will not get a clock file and will complete no operation.

```
if (instruction(15 downto 12)) = "0111" then
else
    clk <= clkin;
```

The other is the fact that a write function is complete on the rising edge, therefore the output of the previous action may be overwritten by the current action as it is being wrote into the file, a solution for this is to simply have the result output from the ALU only get updated to the result input of the register file on the falling edge of the clock signal.

```
if falling_edge(clk) then
      inResult <= std_logic_vector(outResult);
    end if;
```

We can see these two work arounds in the form of a clk register on the clock input and a result register from the combinational logic output:



Here is the entire code for this step:

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


entity CompleteCryptoCoprocessor is
   Port ( instruction : in STD_LOGIC_VECTOR (15 downto 0);
          reset : in STD_LOGIC;
          clkin : in std_logic);
end CompleteCryptoCoprocessor;

architecture Behavioral of CompleteCryptoCoprocessor is

component FullModel
   Port ( CTRL : in unsigned (3 downto 0);
          Abus : in unsigned (15 downto 0);
          Bbus : in unsigned (15 downto 0);
          Result : out unsigned (15 downto 0));
end component;

component Register_File
   port
(
Abus : out std_logic_vector(15 downto 0);
Bbus : out std_logic_vector(15 downto 0);
result : in  std_logic_vector(15 downto 0);
writeEnable : in std_logic;
regASel : in std_logic_vector(3 downto 0);
regBSel : in std_logic_vector(3 downto 0);
writeRegSel : in std_logic_vector(3 downto 0);
reset : in std_logic;
clock : in std_logic
);
end component;

signal interAbus : std_logic_vector(15 downto 0);
signal interBbus : std_logic_vector(15 downto 0);
signal outResult : unsigned(15 downto 0);
signal inResult : std_logic_vector(15 downto 0);
signal writeEn : std_logic;
signal clk : std_logic;

begin
```

```
myFile : Register_File port map(interAbus, interBbus, inResult, writeEn, instruction(11 downto 8), instruction(7 downto 4), instruction(3 downto 0), reset, clk);
myLogic : FullModel port map(unsigned(instruction(15 downto 12)) , unsigned(interAbus), unsigned(interBbus), outResult);


process(clk, reset)
begin

if reset = '1' then
      writeEn <= '0';
else
   if (instruction(15 downto 12)) = "0111" then

   else
      clk <= clkin;
      writeEn <= '1';
      if falling_edge(clk) then
         inResult <= std_logic_vector(outResult);
      end if;
   end if;
end if;

end process;
end Behavioral;
```

2.4 Test Program

Now with a completed crypto coprocessor it is time to test its functionality, in order to do this we must create a new test bench in order to test the functionality of the coprocessor. To do this a new test bench file is created and the used to test for the following instructions:

```
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
```

This can be done using simple wait commands in process while the clock signal is controlled by a separate process.

```
stimulus: process
  begin

   instruction <= "0000010101001100";
   reset <= '0';
   stop_the_clock <= false;
```

```
  instruction <= "0100000110000111" after 2*clock_period;
  wait for 4*clock_period;
  instruction <= "1000000011000000";
  wait for 2*clock_period;
  instruction <= "1010111110010011";
  wait for 2*clock_period;
  instruction <= "0000000001111010";
  wait for 2*clock_period;
  instruction <= "0001011100111100";
  wait for 2*clock_period;
  instruction <= "0111111110010011";
  wait for 2*clock_period;
  instruction <= "0010110010101001";
  wait for 2*clock_period;
  instruction <= "0111110010101001";
  wait for 2*clock_period;
  instruction <= "1100100100000010";

 wait;
 end process;

 clocking: process
 begin
   while not stop_the_clock loop
    clkin <= '0', '1' after clock_period / 2;
    wait for clock_period;
  end loop;
  wait;
 end process;

end;
```

Once this code is ran it will run through all the commands specified and this is the registry files contents once complete:

registers[0:15][...	d058,c50...	Array
[0][15:0]	d058	Array
[1][15:0]	c505	Array
[2][15:0]	011e	Array
[3][15:0]	aa00	Array
[4][15:0]	1186	Array
[5][15:0]	f407	Array
[6][15:0]	1086	Array
[7][15:0]	ad0d	Array
[8][15:0]	6808	Array
[9][15:0]	0105	Array
[10][15:0]	7d65	Array
[11][15:0]	100b	Array
[12][15:0]	030d	Array
[13][15:0]	c902	Array
[14][15:0]	100b	Array
[15][15:0]	b000	Array

Looking at the commands we can calculate whether these are correct.

ADD R5, R4 R12 so adding the contents of 5 and 4 and sending them to 12, f407 + 1186 = 058D (12 is overwritten by a later command in this situation).

XOR R1, R8 R7 C505 XOR 6808 = AD0D whish we is successfully what is stored in R7.

ROR4 R12 R0 058D rotated right = D058 which is successfully stored in R0.

SLL4 R9 R3 BAA0 shifted left = AA00 which is successfully stored in R3.

ADD R0, R7 R10 D058 + AD0D = 17D65 and 7D65 is successfully stored in R10.

SUB R7, R3 R12 AD0D – AA00 = 030D which is stored in R12.

Nop, was completed successfully with no changes to the regester file.

AND R12, R10 R9 030D AND 7D65 = 0105 which is stored in R9.

The next nop was successfully complete.

LUT R9 R2 0105 translated through the lookup table = 011E which is stored in R2.

All of these functions were completed successfully so the functionality of the coprocessor is complete.

Simulation instructions:

To run the simulations on this project the full coprocessors simulation file is named: CompleteCryptoCoproceesor_tb simply set this as the simulation source and run simulation.

To run the simulation of the combinational logic the simulation file is named FullModel_tb, again set this as the simulation source and run the simulation.

The complete crypto simulation needs to run for 400ns for all results and the full model for 1500ns.