University of Windsor
Electrical and Computer Engineering

Project Report

06-88-443 Embedded System Design

Instructor: Arash Ahmadi

Project 1: Single Purpose Processor

Group Member: Daksh Patel	**ID: 104 030 031**
Group Member: Nyasha Kapfumvuti	**ID: 104 121 166**
Group Member: Khalifa Badamasi	**ID: 103 674 900**

Date: Mar 5th, 2018

Introduction

The objective of this project is to implement a single purpose processor on an FPGA that can calculate a specific function $\frac{d^2y}{dx^2} = b \times f(x) \times \frac{dy}{dx} + a \times g(y) \times y$. Functions f(x) and g(x) and constants 'a' and 'b' were based on the last two digits of our student numbers and the corresponding table given to us. Based on the student number of one of the members in the group our function ended up being $\boldsymbol{\frac{d^2y}{dx^2} = \frac{6x^4dy}{dx} + 6x^4y.}$ This function needed to be discretized and implemented in HDL code for use on an FPGA. This project will show us how to work on scheduling diagrams, block diagrams for the datapath, state table/diagram of the controller and also HDL coding.

Procedure

1) Derive the discrete form of the equation and explain how it was derived.
2) Draw the scheduling diagram.
3) Design and draw the block diagrams of the datapath and state table/diagram of the controller.
4) Write the HDL code of the design and synthesis it on an FPGA.
5) Provide a table indicating LUT usage and maximum frequency.
6) Discuss the difference of your design if a pipelined datapath was used.

Discussion/Analysis

The discrete form is derived in the Appendix Figure 8. Figure 9 is the initial scheduling diagram with Figure 1 showing out progress. Figure 10 is the initial datapath and 2 shows our progress in understanding the project. The tables show our theory for using states and clocks to for the controller portion of the code. Table 1 explains the read and write enable logic using the decoder and clock. The Table 2 finishes the state logic. Table 3 shows the register state logic that we came up with. The figures and tables explain all the operations.

The codes used currently and the previous version are given below. We switched from Verilog to VHDL due to finding resources to help understand instantiating better. In Verilog we kept receiving an error for instantiating modules. The VHDL code is able to perform the addition and multiplication actions through the instantiating method at the gate level. Figure 4 shows the look up table. Figure 5 shows the simulation results. Figure 6 and 7 show the RTL Diagrams.

We used a 3 to 8 decoder to generate controls for our read/write enables of the registers ALU and multiplier to match the sequential operations. The clock drives the decoder, which in turn drivers the system controls and that leads to all the operations. Further selection for the addition and multiplication is accomplished using a bus implementation for our datapath. Wiring with tristate buffers allow us to enable dataflow from the correct registers to the ALU and multiplier based on the Read Enable section of the following table as well as from the ALU and multiplier to the destination register based on the Write Enable Section. This ensures that appropriate data transfers based on the operation, selected inputs and outputs. The figures and tables below explain our design process and the implementation we tried to finish.

Figure 2 was not fully implemented but properly thought out. Originally the plan was to use Tri-State Buffers and a proper Bus but resulted to the idea shown in Figure 2 because we did not understand how to implement a Bus and Tri State buffer. Turned out the Figure 2 implementation also seems to not work out so we simply implemented each function in a state machine. Which the regular code would have used regardless so we believe this to be as close to gate level we are capable of implementing due to time constraints.

Data/Diagrams

Table 1: State Table One Half

State Determined by Clock			Read EN										Write En								
Clock	Dec In	Dec Out	x	dx	u	y	3	R1	R2	R3	R4	R5	x	dx	u	y	R1	R2	R3	R4	R5
0	000	00000001	1	1													1	1			
1	001	00000010			1	1		1	1				1				1				
2	010	00000100						1		1									1		
3	011	00001000					1			1									1		
4	100	00010000		1	1					1					1					1	
5	101	00100000				1					1									1	
6	110	01000000									1		1			1					
7	111	10000000										1									1

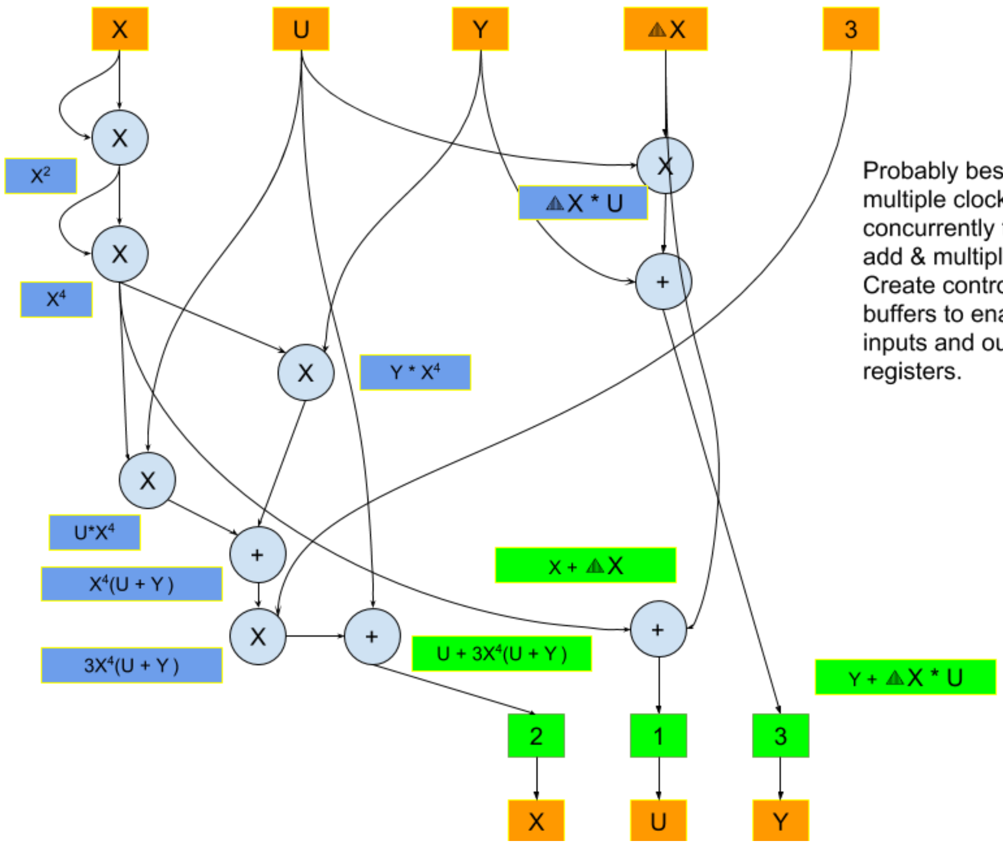
Table 2: State Table Second Half

State Determined by Clock			Multi	Mult Op	Add	Add Op	Reg Transfer	Results	Explanation
Clock	Dec In	Dec Out							
0	000	00000001	1	R1 <= X * X	1	R2 = X + DX		X^2, X + DX	Output 1 IN R2
1	001	00000010	1	R1 <= R1 * R1	1	R3 = U +Y	x <= R2	X^4, Ui + Y	R2 moved to x
2	010	00000100	1	R3 <= R1 * R3				X^4(U +Y)	

3	011	00001000	1	R3 <= 3 * R3				3 *(X^4 * (U +Y))	Output 3 IN R3
4	100	00010000	1	R4 <= U * DX		R3 <= U + R3	u <= R3	U*DX, Ui +U	R3 moved to u
5	101	00100000			1	R4 <= R4 + Y	y <= R4	Y+U*DX	Output 2 IN R4, R4 moved to y
6	110	01000000				R5<= R5 + 1			Count Increment
7	111	10000000							Output

Table 3: Register Select Logic

Register	**Read EN Logic**	**Write EN Logic**
	Combo logic	Combo logic
x	D0	D1
dx	D0 + D4	
u	D1 + D4	D4
y	D1 + D5	D6
3	D3	
R1	D1 + D2	D0 +D1
R2	D1	D0
R3	D2 + D3 + D4	D2 + D3
R4	D5 + D6	D4 +D5
R5	D7	D7



Probably best to use multiple clock cycles to concurrently to carry out add & multiply operations. Create control to tristate buffers to enable correct inputs and outputs from registers.

Figure 1: Scheduling Diagram

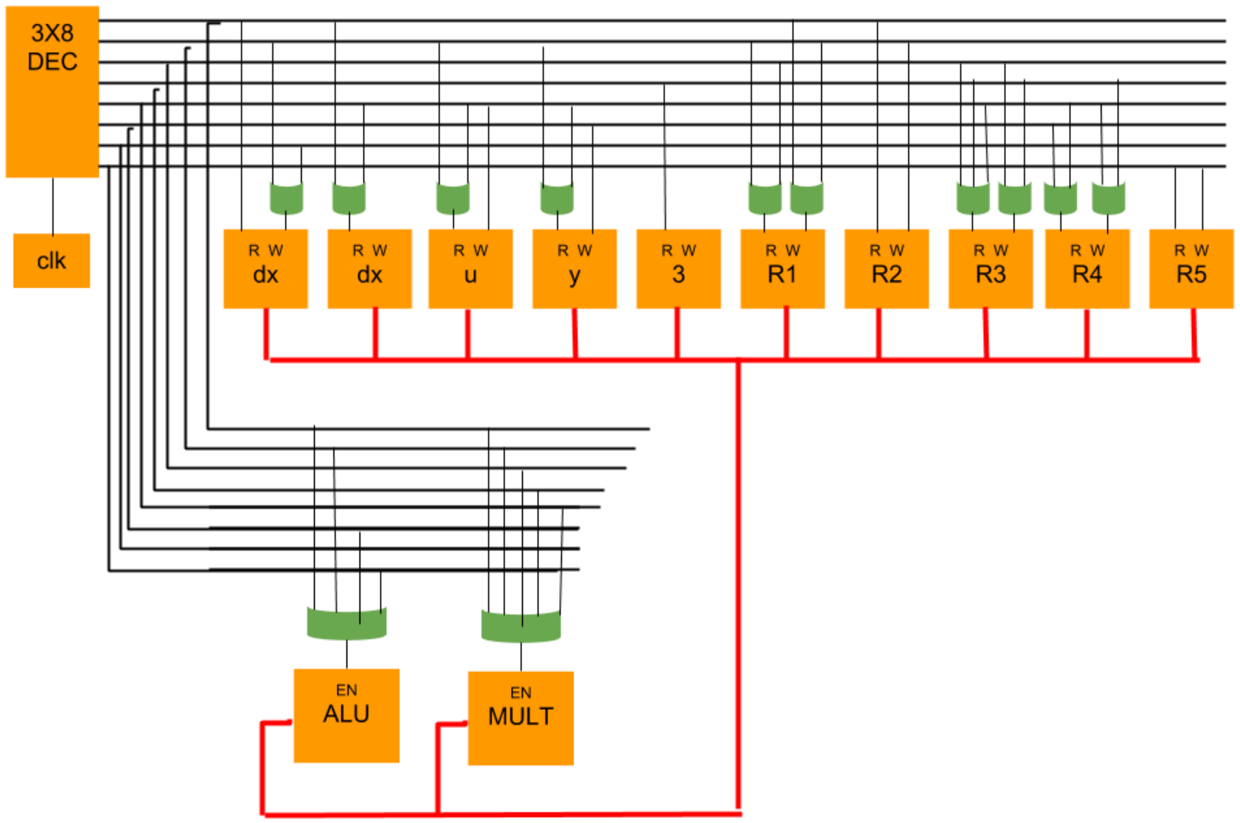
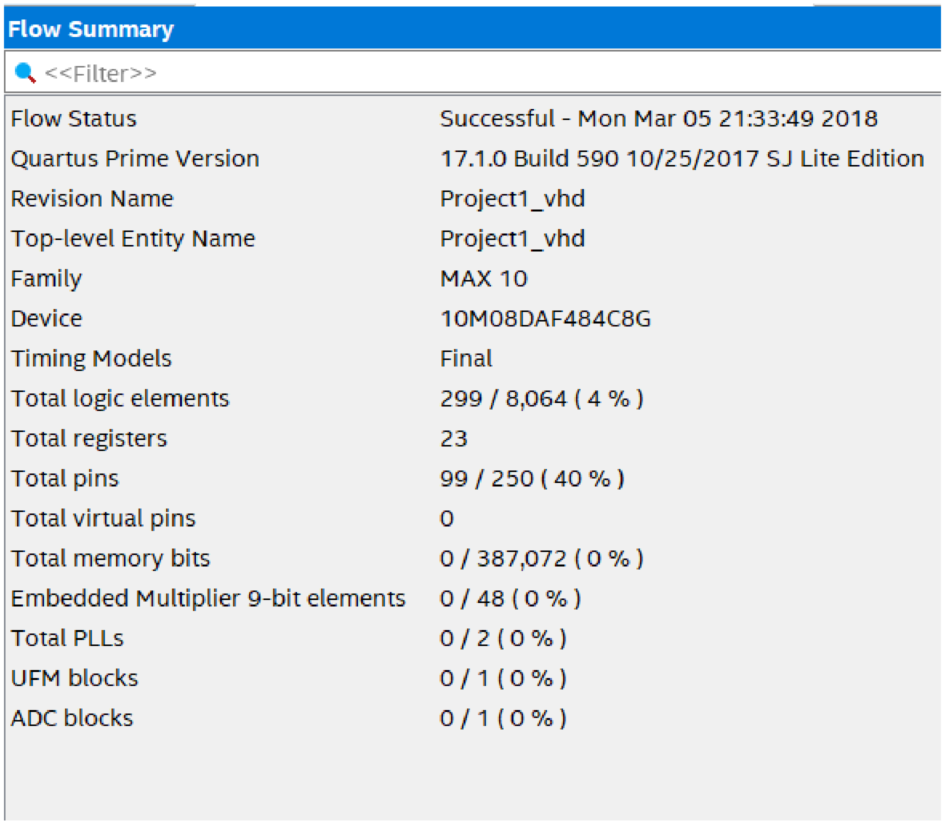


Figure 2: Datapath Diagram

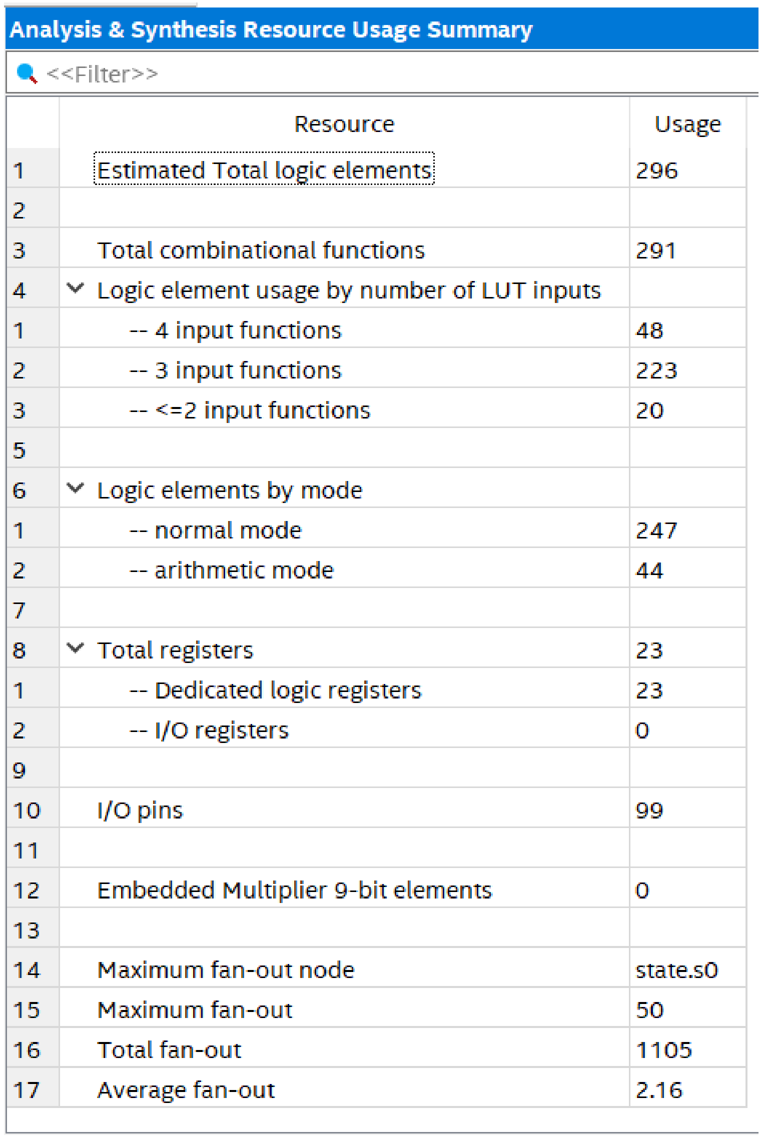


Flow Summary

<<Filter>>

Flow Status	Successful - Mon Mar 05 21:33:49 2018
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name	Project1_vhd
Top-level Entity Name	Project1_vhd
Family	MAX 10
Device	10M08DAF484C8G
Timing Models	Final
Total logic elements	299 / 8,064 (4 %)
Total registers	23
Total pins	99 / 250 (40 %)
Total virtual pins	0
Total memory bits	0 / 387,072 (0 %)
Embedded Multiplier 9-bit elements	0 / 48 (0 %)
Total PLLs	0 / 2 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 1 (0 %)

Figure 3: Synthesis Summary

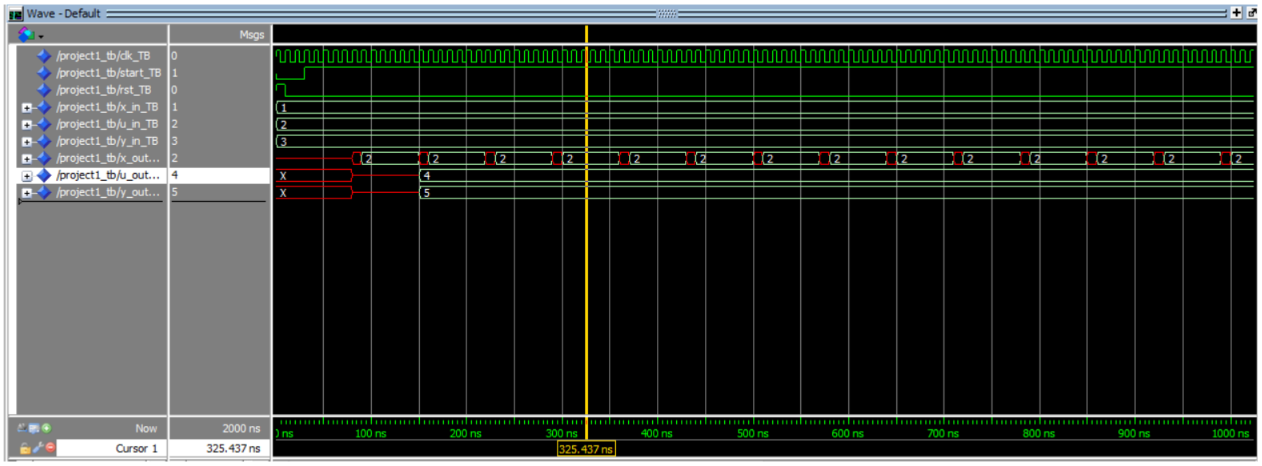


Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimated Total logic elements	296
2		
3	Total combinational functions	291
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	48
2	-- 3 input functions	223
3	-- <=2 input functions	20
5		
6	Logic elements by mode	
1	-- normal mode	247
2	-- arithmetic mode	44
7		
8	Total registers	23
1	-- Dedicated logic registers	23
2	-- I/O registers	0
9		
10	I/O pins	99
11		
12	Embedded Multiplier 9-bit elements	0
13		
14	Maximum fan-out node	state.s0
15	Maximum fan-out	50
16	Total fan-out	1105
17	Average fan-out	2.16

Figure 4: Look Up Table



```
Wave - Default
                              Msgs
/project1_tb/clk_TB           0
/project1_tb/start_TB         1
/project1_tb/rst_TB           0
/project1_tb/x_in_TB          1
/project1_tb/u_in_TB          2
/project1_tb/y_in_TB          3
/project1_tb/x_out...         2
/project1_tb/u_out...         4
/project1_tb/y_out...         5

Now                           2000 ns
Cursor 1                      325.437 ns
```

Figure 5: Simulation Results

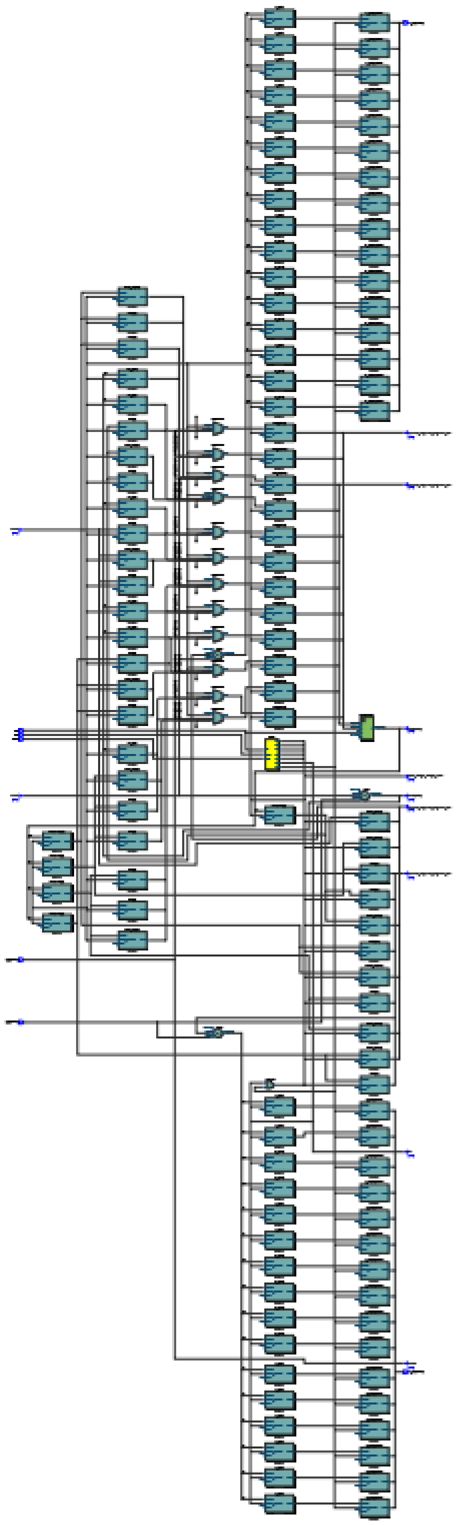


Figure 6: RTL Simulation 1

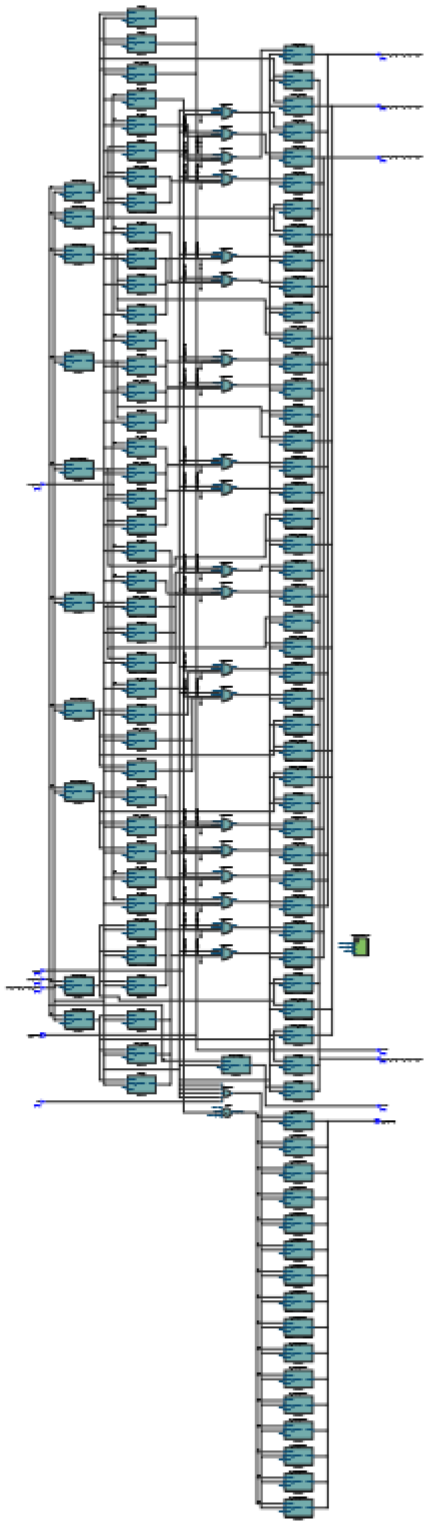


Figure 7: RTL Simulation 2

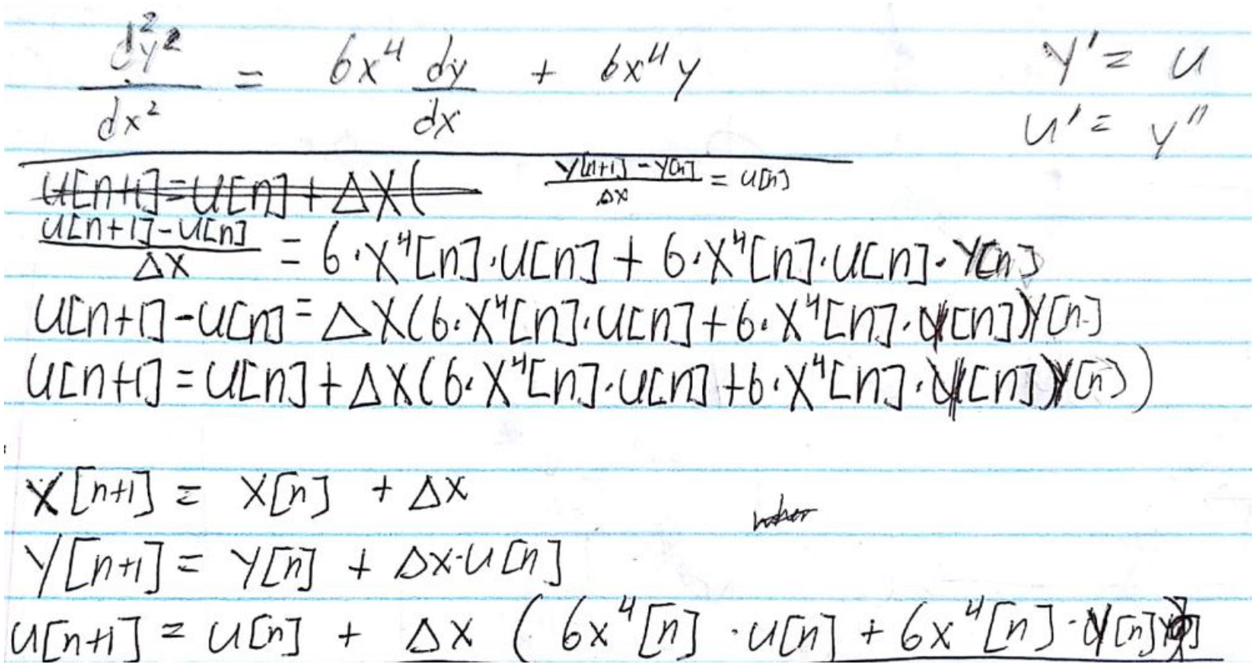
Table 4: Differences Between Pipeline and Non-Pipelined Approaches

Non-pipelined Approach	**Pipelined Approach**
More load on processor	Less load on the processor
Better performance (quicker completion time, less steps required); less latency	Worse performance (slower completion time, more step required); more latency
Same or more hardware used, more transistors	Same or less hardware used, less transistors

Conclusion

The project has been completed to the best of our abilities. We believe we have designed something that is very efficient and still fast. The implementation of it needs work to properly loop. The current code is using a mix of the below and above datapaths to try and receive results. The testbench shows the first iteration values but cannot output anymore due to trouble connecting the input and output wires.

Appendix



$$\frac{d^2y}{dx^2} = 6x^4\frac{dy}{dx} + 6x^4y \qquad y' = u \qquad u' = y''$$

~~U[n+1]=U[n]+ΔX(~~

$$\frac{Y[n+1]-Y[n]}{\Delta x} = u[n]$$

$$\frac{U[n+1]-U[n]}{\Delta X} = 6\cdot X^4[n]\cdot U[n] + 6\cdot X^4[n]\cdot U[n]\cdot Y[n]$$

$$U[n+1]-U[n] = \Delta X(6\cdot X^4[n]\cdot U[n] + 6\cdot X^4[n]\cdot \cancel{Y}[n])Y[n]$$

$$U[n+1] = U[n] + \Delta X(6\cdot X^4[n]\cdot U[n] + 6\cdot X^4[n]\cdot \cancel{Y}[n])Y[n])$$

$$X[n+1] = X[n] + \Delta X$$

when

$$Y[n+1] = Y[n] + \Delta x\cdot U[n]$$

$$U[n+1] = U[n] + \Delta X\ (6x^4[n]\cdot U[n] + 6x^4[n]\cdot \cancel{Y}[n]\cancel{Y}[n]$$

Figure 8: Discrete Function Calculations

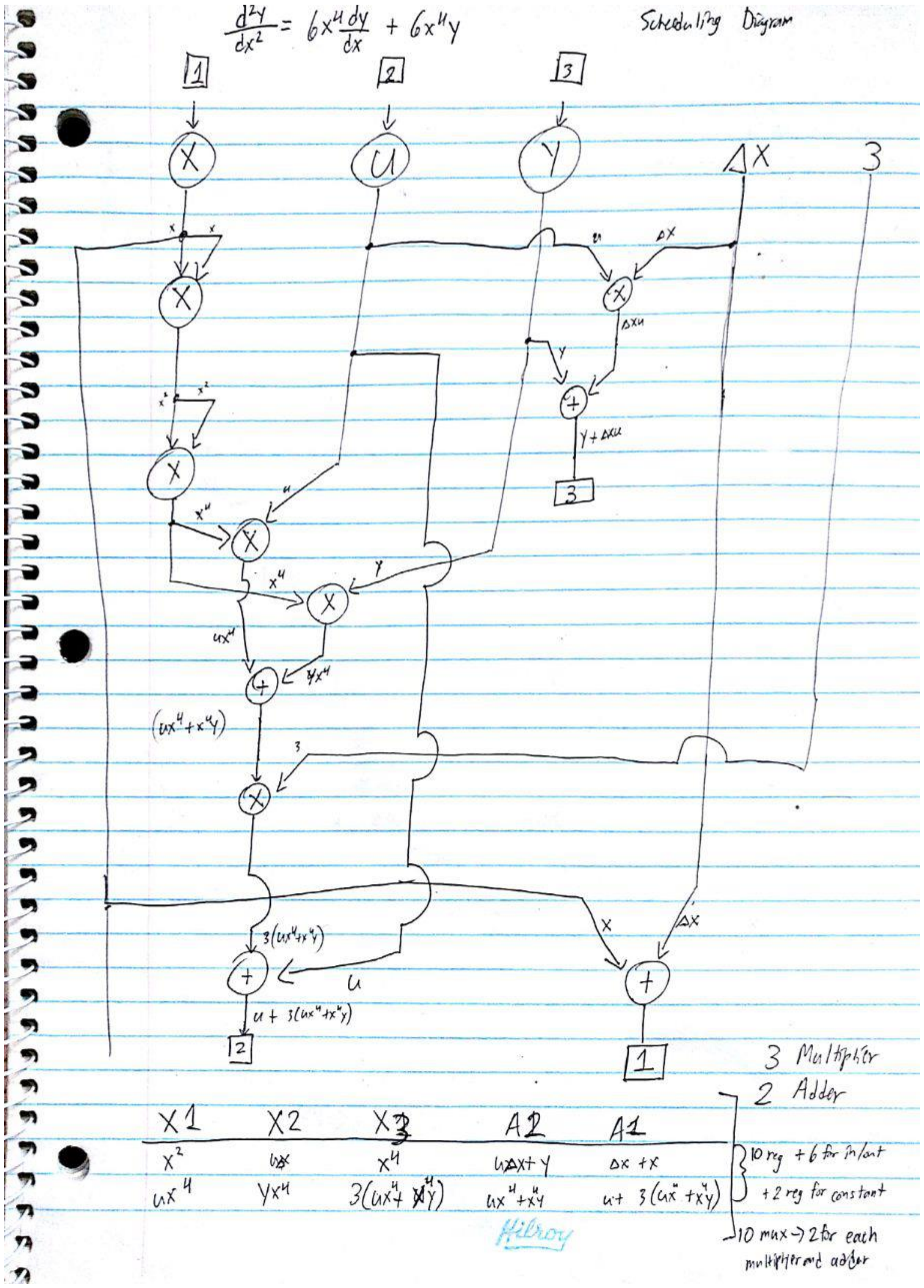


Figure 9: Initial Scheduling Diagram

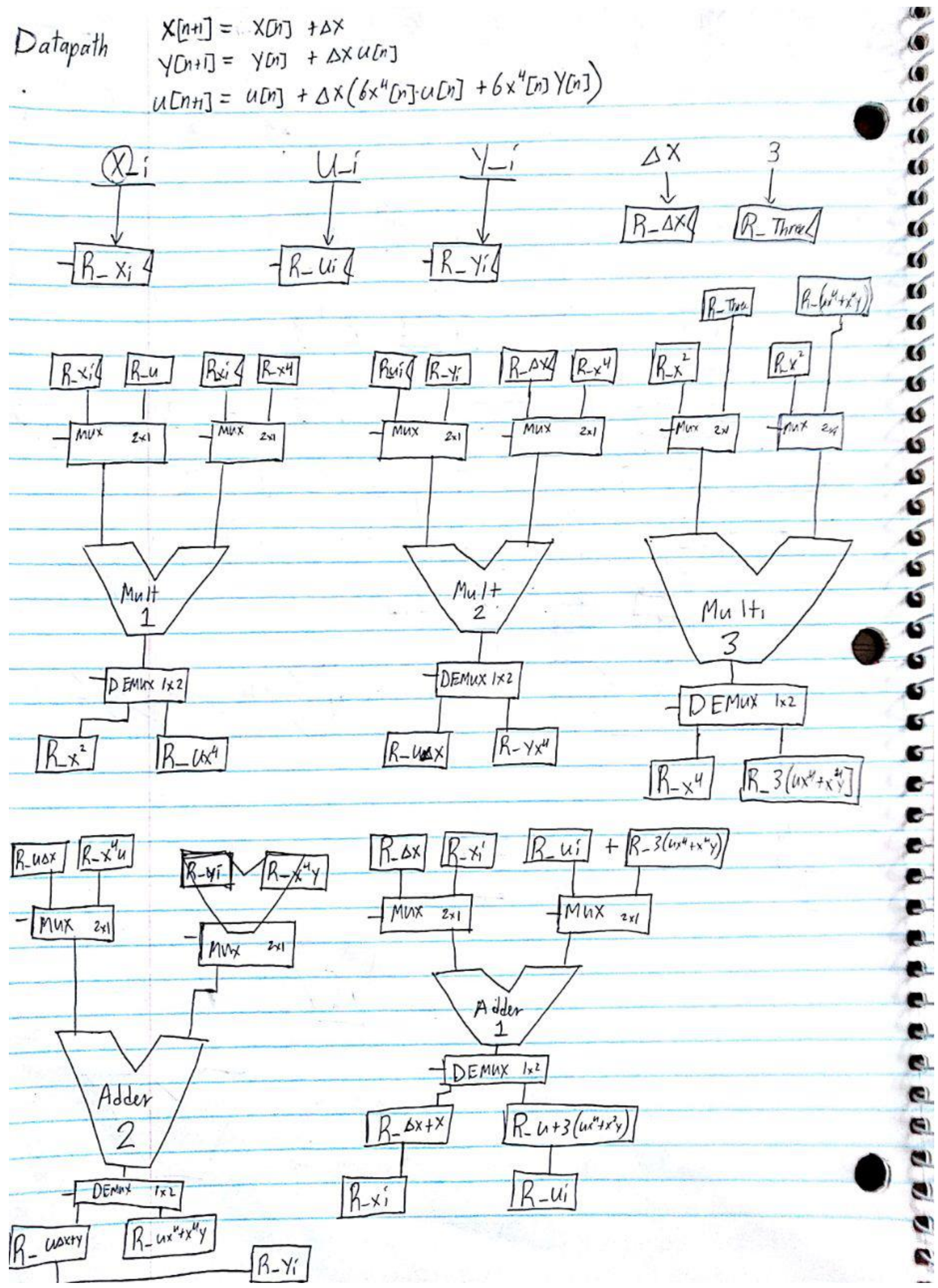


Figure 10: Initial Datapath

Current Simulation

Project 1

```
-- A Moore machine's outputs are dependent only on the current state.
-- The output is written only when the state changes.  (State
-- transitions are synchronous.)

--Group Member: Daksh Patel         ID: 104 030 031
--Group Member: Nyasha Kapfumvuti   ID: 104 121 166
--Group Member: Khalifa Badamasi        ID: 103 674 900
--
--Project 1: VHDL Implementation of Single Purpose Processor


library ieee;
library std;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

--library ieee_proposed;
--use ieee_proposed.fixed_pkg.all;

entity Project1_vhd is

    port(
        clk      : in    std_logic;
        start     : in    std_logic;
        rst   : in    std_logic;
        x_in, u_in, y_in     : in std_logic_vector(15 downto 0);
        x_out, u_out, y_out : out std_logic_vector(15 downto 0)
    );

end entity;

architecture rtl of Project1_vhd is

    -- Build an enumerated type for the state machine
    type state_type is (s0, s1, s2, s3, s4, s5, s6);

    -- Register to hold the current state
    signal state    : state_type;
    --shared variable count : INTEGER range 0 to 10;
    -- Internal Variables
    signal x_o, u_o, y_o, x_i, u_i, y_i :  std_logic_vector(15 downto 0);
```

```vhdl
    signal rx4, rx2, rdx, r3, rudx, r3ux4_yx4, rux4_x4y, ru_y :
std_logic_vector(15 downto 0);
    signal tempMult : std_logic_vector(31 downto 0);


    -- Booth Multiplier Variables
    signal mult_ina, mult_inb : std_logic_vector(15 downto 0);
    signal mload : std_logic := '1';
    signal mready : std_logic;
    signal mult_o : std_logic_vector(31 downto 0);
    -- Booth Component Ports
    component booth port(ain : in std_logic_vector(15 downto 0);
       bin : in std_logic_vector(15 downto 0);
       qout : out std_logic_vector(31 downto 0);
       clk : in std_logic;
       load : in std_logic;
       ready : out std_logic);
    end component;


    -- Full Adder Variables
    signal add_A, add_B: std_logic_vector(15 downto 0);
    signal add_Sum : std_logic_vector(16 downto 0);


    -- Full Adder Ports
    component ripple_carry_adder port(i_add_term1, i_add_term2: in
std_logic_vector(15 downto 0);

       o_result : out std_logic_vector(16 downto 0)
       );
    end component;

    -- Mux Variables
    signal mux1out : std_logic_vector(15 downto 0);
    signal mux2out : std_logic_vector(15 downto 0);
    signal mux3out : std_logic_vector(15 downto 0);
    signal mux1s : std_logic;
    signal mux2s : std_logic;
    signal mux3s : std_logic;

    -- Mux Ports
    component mux_2to1 port (A,B : in std_logic_vector(15 downto 0);
    S:IN std_logic;
    Y:OUT std_logic_vector(15 downto 0));
    end component;

begin
    -- Booth Port Map
    mult1 : booth port map (ain => mult_ina, bin => mult_inb, qout => mult_o, clk
=> clk, load => mload, ready => mready);
    -- Full Adder Port Map
    add1 : ripple_carry_adder port map (i_add_term1 => add_A, i_add_term2 =>
add_B, o_result => add_Sum);
    -- Mux Port Map
    mux1 : mux_2to1 port map (A => x_in, B => x_o, S => mux1s, Y => x_i);
    mux2 : mux_2to1 port map (A => u_in, B => u_o, S => mux2s, Y => u_i);
    mux3 : mux_2to1 port map (A => y_in, B => y_o, S => mux3s, Y => y_i);

    rdx <= "0000000000000001";
    r3 <= "0000000000000110";
--  x_i <= x_in;
--  u_i <= u_in;
--  y_i <= y_in;


    -- Logic to advance to the next state
    process (clk, rst)

    begin
        if rst = '1' then
            state <= s0;
        elsif (rising_edge(clk)) then
        --elsif count > 0 then
            case state is
                when s0=>
                    if start = '1' then
                        state <= s1;
                    else
                        state <= s0;
                    end if;
                when s1=>
                    if start = '1' then
                        state <= s2;
                    else
                        state <= s1;
                    end if;
                when s2=>
                    if start = '1' then
                        state <= s3;
                    else
                        state <= s2;
                    end if;
                when s3 =>
                    if start = '1' then
                        state <= s4;
                    else
                        state <= s3;
                    end if;
                when s4=>
                    if start = '1' then
                        state <= s5;
                    else
                        state <= s4;
                    end if;
                when s5=>
                    if start = '1' then
                        state <= s6;
                    else
                        state <= s5;
                    end if;
                when s6=>
                    if start = '1' then
                        state <= s0;
                    else
                        state <= s6;
                    end if;
            end case;
        end if;
    end process;

    -- Output depends solely on the current state
    process (state)
    begin

        mux1s <= '0';
        mux2s <= '0';
        mux3s <= '0';

        case state is
            when s0 =>
                mult_ina <= x_i;
                mult_inb <= x_i;
                tempMult <= mult_o;
                --tempMult <= x_i*x_i;
                rx2 <= tempMult(15 downto 0);
                --x_out <= x_i + rdx;
                add_A <= x_i;
                add_B <= rdx;

                if add_Sum(16) = '1' then
                    x_o <= add_Sum(16 downto 1);
                else
                    x_o <= add_Sum(15 downto 0);
                end if;

            when s1 =>
                mult_ina <= rx2;
                mult_inb <= rx2;

                --tempMult <= rx2*rx2;
                rx4 <= tempMult(15 downto 0);
                --ru_y <= u_i + y_i;

                add_A <= u_i;
                add_B <= y_i;
                if add_Sum(16) = '1' then
                    ru_y <= add_Sum(16 downto 1);
                else
                    ru_y <= add_Sum(15 downto 0);
                end if;

            when s2 =>
                mult_ina <= rx4;
                mult_inb <= ru_y;

                --tempMult <= rx4*ru_y;
                rux4_x4y <= tempMult(15 downto 0);
            when s3 =>
                mult_ina <= rux4_x4y;
                mult_inb <= r3;

                --tempMult <= rux4_x4y*r3;
                r3ux4_yx4 <= tempMult(15 downto 0);
            when s4 =>
                mult_ina <= u_i;
                mult_inb <= rdx;
```

```
                --tempMult <= u_i*rdx;
                rudx <= tempMult(15 downto 0);
                --u_o <= r3ux4_yx4 + u_i;
                add_A <= u_i;
                add_B <= r3ux4_yx4;
                if add_Sum(16) = '1' then
                    u_o <= add_Sum(16 downto 1);
                else
                    u_o <= add_Sum(15 downto 0);
                end if;

            when s5 =>
                --y_o <= rudx + y_i;
                add_A <= rudx;
                add_B <= y_i;
                if add_Sum(16) = '1' then
                    y_o <= add_Sum(16 downto 1);
                else
                    y_o <= add_Sum(15 downto 0);
                end if;

            when s6 =>
--              y_i <= y_o;
--              x_i <= x_o;
--              u_i <= u_o;
                mux1s <= '1';
                mux2s <= '1';
                mux3s <= '1';

                y_out <= y_o;
                x_out <= x_o;
                u_out <= u_o;

        end case;
    end process;

end rtl;
```

Project 1 Testbench

```
library ieee;
library std;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

LIBRARY ieee  ;
LIBRARY std  ;
USE ieee.std_logic_1164.all  ;

ENTITY Project1_tb  IS
END ;

architecture test of Project1_tb is
    component Project1_vhd
        port (
        clk      : in    std_logic;
        start     : in   std_logic;
        rst  : in   std_logic;
        x_in, u_in, y_in    : in std_logic_vector(15 downto 0);
        x_out, u_out, y_out : out std_logic_vector(15 downto 0)
        );
    end component;

  signal clk_TB     : std_logic;
  signal start_TB    : std_logic;
  signal rst_TB  : std_logic;
  signal x_in_TB, u_in_TB, y_in_TB  : std_logic_vector(15 downto 0);
  signal x_out_TB, u_out_TB, y_out_TB : std_logic_vector(15 downto 0);


begin
    -- instantiate the ALU
    inst_Project1_vhd:  Project1_vhd
        port map(

        clk =>clk_TB,
        start =>start_TB,
        rst =>rst_TB,
        x_in =>x_in_TB,
        u_in =>u_in_TB,
        y_in =>y_in_TB,
        x_out =>x_out_TB,
        u_out =>u_out_TB,
        y_out =>y_out_TB
        );

    -- Generate clock stimulus
    clk_gen: process
    begin -- clock period = 10 ns
        clk_TB <= '1';
        wait for 5 ns;
        clk_TB <= '0';
        wait for 5 ns;

        if now >= 2000 ns then -- run for 200 cc
            assert false
             report "simulation is completed (not error)."
             severity error;
            wait;
        end if;
    end process;

    data_gen: process
    begin

      x_in_TB <= "0000000000000001";
      u_in_TB <= "0000000000000010";
      y_in_TB <= "0000000000000011";
            start_TB <= '0';
            rst_TB <= '1';
            wait for 10 ns;
            rst_TB <= '0';
            wait for 10 ns;


            --input_TB <= (others => "00000000"); --all values 0 at beginning
            wait for 10 ns;
            start_TB <= '1';

        wait;

    end process;

end test;
```

Booth Multiplier

```
---------------------------------------------------------------------
--              Digital System Design with VHDL 2nd Edition
--              Mark Zwolinski
--              Pearson Education, 2004, ISBN 0-13-039985-X
--
--              Chapter 6, page 148
--
-- Design unit: booth(rtl) (Entity and Architecture)
--            :
-- File name  : booth.vhd
--            :
-- Description: RTL description of Booth multiplier
--            :
-- Limitations: None
--            :
-- System     : VHDL'93/'02, STD_LOGIC_1164
--            :
-- Author     : Mark Zwolinski
--            : mz@ecs.soton.ac.uk
--
-- Revision   : Version 2.0 10/12/03
---------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all, IEEE.numeric_std.all;
entity booth is
  generic(al : NATURAL := 16;
          bl : NATURAL := 16;
          ql : NATURAL := 32);
  port(ain : in std_logic_vector(al-1 downto 0);
       bin : in std_logic_vector(bl-1 downto 0);
       qout : out std_logic_vector(ql-1 downto 0);
       clk : in std_logic;
       load : in std_logic;
       ready : out std_logic);
end entity booth;

architecture rtl of booth is
begin
  process (clk) is
    variable count : INTEGER range 0 to al;
    variable pa : signed((al+bl) downto 0);
    variable a_1 : std_logic;
    alias p : signed(bl downto 0) is pa((al + bl) downto al);
  begin
    if (rising_edge(clk)) then
      if load = '1' then
        p := (others => '0');
        pa(al-1 downto 0) := signed(ain);
        a_1 := '0';
        count := al;
        ready <= '0';
      elsif count > 0 then
        case std_logic_vector'(pa(0), a_1) is
          when "01" =>
            p := p + signed(bin);
          when "10" =>
            p := p - signed(bin);
          when others => null;
        end case;
        a_1 := pa(0);
        pa := shift_right(pa, 1);
        count := count - 1;
      end if;
      if count = 0 then
        ready <= '1';
      end if;
      qout <= std_logic_vector(pa(al+bl-1 downto 0));
    end if;
  end process;
end architecture rtl;
```

Ripple Carry Adder

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

entity ripple_carry_adder is
  generic (
    g_WIDTH : natural := 16
    );
  port (
    i_add_term1  : in std_logic_vector(g_WIDTH-1 downto 0);
    i_add_term2  : in std_logic_vector(g_WIDTH-1 downto 0);
    --
    o_result   : out std_logic_vector(g_WIDTH downto 0)
    );
end ripple_carry_adder;


architecture rtl of ripple_carry_adder is

  component full_adder is
    port (
      i_bit1  : in  std_logic;
      i_bit2  : in  std_logic;
      i_carry : in  std_logic;
      o_sum   : out std_logic;
      o_carry : out std_logic);
  end component full_adder;

  signal w_CARRY : std_logic_vector(g_WIDTH downto 0);
  signal w_SUM   : std_logic_vector(g_WIDTH-1 downto 0);


begin

  w_CARRY(0) <= '0';                      -- no carry input on first full adder

  SET_WIDTH : for ii in 0 to g_WIDTH-1 generate
    i_FULL_ADDER_INST : full_adder
      port map (
        i_bit1  => i_add_term1(ii),
        i_bit2  => i_add_term2(ii),
        i_carry => w_CARRY(ii),
        o_sum   => w_SUM(ii),
        o_carry => w_CARRY(ii+1)
        );
  end generate SET_WIDTH;

  o_result <= w_CARRY(g_WIDTH) & w_SUM;  -- VHDL Concatenation

end rtl;
```

Full Adder

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity full_adder is
  port (
    i_bit1  : in std_logic;
    i_bit2  : in std_logic;
    i_carry : in std_logic;
    --
    o_sum   : out std_logic;
    o_carry : out std_logic
    );
end full_adder;


architecture rtl of full_adder is

  signal w_WIRE_1 : std_logic;
  signal w_WIRE_2 : std_logic;
  signal w_WIRE_3 : std_logic;

begin

  w_WIRE_1 <= i_bit1 xor i_bit2;
  w_WIRE_2 <= w_WIRE_1 and i_carry;
  w_WIRE_3 <= i_bit1 and i_bit2;

  o_sum   <= w_WIRE_1 xor i_carry;
  o_carry <= w_WIRE_2 or w_WIRE_3;


  -- FYI: Code above using wires will produce the same results as:
  -- o_sum   <= i_bit1 xor i_bit2 xor i_carry;
  -- o_carry <= (i_bit1 xor i_bit2) and i_carry) or (i_bit1 and i_bit2);

  -- Wires are just used to be explicit.

end rtl;
```

Mux 2 to 1 with 16Bit Lanes

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

entity mux_2to1 is
    Port ( S : in  STD_LOGIC;
           A   : in  STD_LOGIC_VECTOR (15 downto 0);
           B   : in  STD_LOGIC_VECTOR (15 downto 0);
           Y   : out STD_LOGIC_VECTOR (15 downto 0));
end mux_2to1;

architecture Behavioral of mux_2to1 is
begin
    Y <= B when (S = '1') else A;
end Behavioral;
```

3 to 8 Decoder

```vhdl
---------------------------------------------------------------------
--              Digital System Design with VHDL 2nd Edition
--              Mark Zwolinski
--              Pearson Education, 2004, ISBN 0-13-039985-X
--
--              Chapter 4, Exercise 4.3, page 78, 342
--
-- Design unit: decoder(bool_expr, when_else, with_select)
--            : (Entity and Architectures)
--            :
-- File name  : decode38.vhd
--            :
-- Description: 3 to 8 decoder. Answer to exercise 4.3
--            :
-- Limitations: None
--            :
-- System     : VHDL'93/'02, STD_LOGIC_1164
--            :
-- Author     : Mark Zwolinski
--            : mz@ecs.soton.ac.uk
--
-- Revision   : Version 2.0 03/12/03
---------------------------------------------------------------------
```

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
entity decoder is
  port (a : in std_logic_vector(2 downto 0);
        z : out std_logic_vector(7 downto 0));
end entity decoder;

architecture bool_expr of decoder is
begin
  z(0) <= not a(0) and not a(1) and not a(2);
  z(1) <= a(0) and not a(1) and not a(2);
  z(2) <= not a(0) and a(1) and not a(2);
  z(3) <= a(0) and a(1) and not a(2);
  z(4) <= not a(0) and not a(1) and a(2);
  z(5) <= a(0) and not a(1) and a(2);
  z(6) <= not a(0) and a(1) and a(2);
  z(7) <= a(0) and a(1) and a(2);
end architecture bool_expr;

architecture when_else of decoder is
begin
  z <= "00000001" when a = "000" else
       "00000010" when a = "001" else
       "00000100" when a = "010" else
       "00001000" when a = "011" else
       "00010000" when a = "100" else
       "00100000" when a = "101" else
       "01000000" when a = "110" else
       "10000000" when a = "111" else
       "XXXXXXXX";
end architecture when_else;

architecture with_select of decoder is
begin
  with a select
    z <= "00000001" when "000",
         "00000010" when "001",
         "00000100" when "010",
         "00001000" when "011",
         "00010000" when "100",
         "00100000" when "101",
         "01000000" when "110",
         "10000000" when "111",
         "XXXXXXXX" when others;
end architecture with_select;
```

Demux

```
---------------------------------------------------------------------
--              Digital System Design with VHDL 2nd Edition
--              Mark Zwolinski
--              Pearson Education, 2004, ISBN 0-13-039985-X
--
--              Chapter 4, page 64
--
-- Design unit: mux(mux1, mux2) (Entity and Architectures)
--            :
-- File name  : mux.vhd
--            :
-- Description:
--            :
-- Limitations: None
--            :
-- System     : VHDL'93/'02, STD_LOGIC_1164
--            :
-- Author     : Mark Zwolinski
--            : mz@ecs.soton.ac.uk
--
-- Revision   : Version 2.0 03/12/03
---------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
entity mux is
  port (a, b: out std_logic_vector(15 downto 0);
        s: in std_logic;
        y: in std_logic_vector(15 downto 0));
end entity mux;

architecture mux1 of mux is
begin
  with s select
    y => a when "0",
         b when "1",

         'X' when others;
end architecture mux1;

architecture mux2 of mux is
begin
  y => a when s="0" else
```

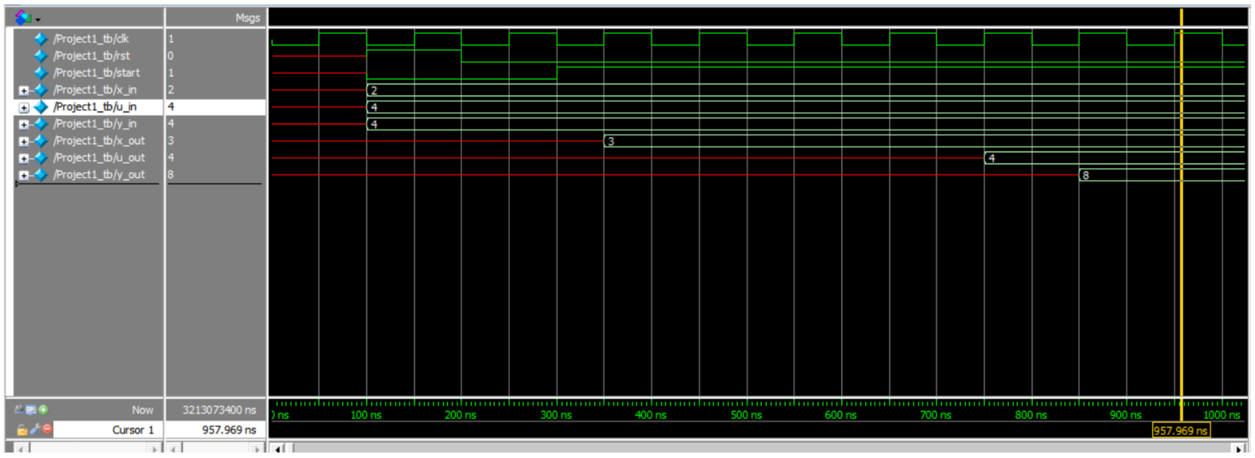
```
       b when s="1" else

       'X';
end architecture mux2;
```

ROM

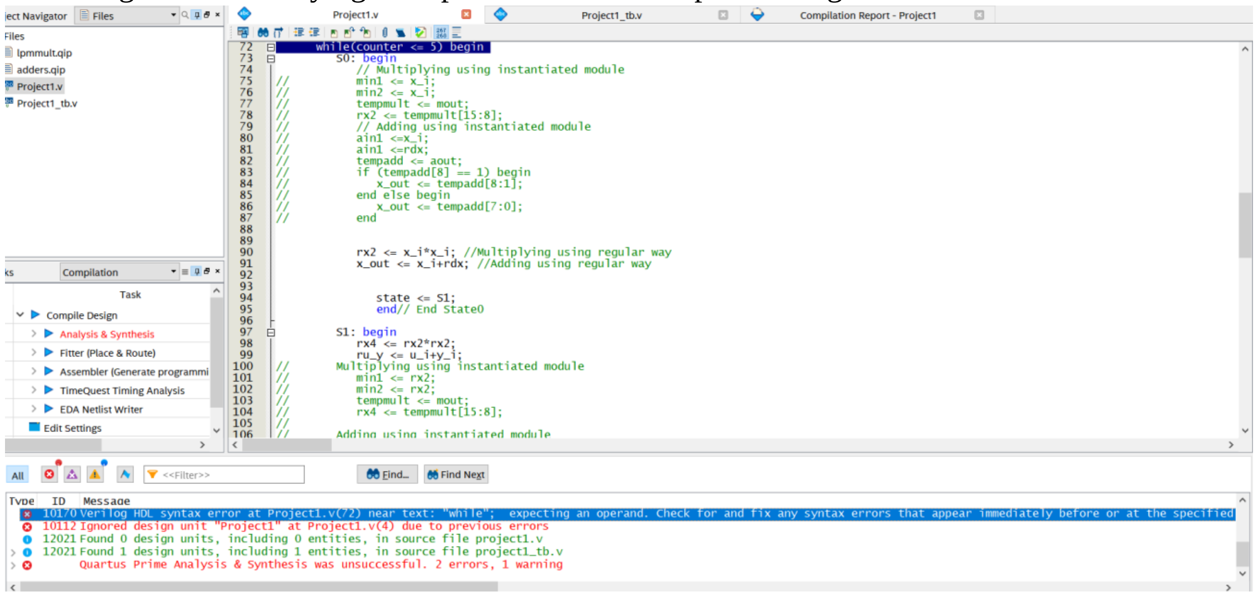
```
--------------------------------------------------------------------
--               Digital System Design with VHDL 2nd Edition
--               Mark Zwolinski
--               Pearson Education, 2004, ISBN 0-13-039985-X
--
--               Chapter 6, page 143
--
-- Design unit: rom16x7(sevenseg) (Entity and Architecture)
--             :
-- File name   : rom.vhd
--             :
-- Description: RTL model of ROM containing decoding patterns for
--             : 7 segment display
--             :
-- Limitations: None
--             :
-- System      : VHDL'93/'02, STD_LOGIC_1164
--             :
-- Author      : Mark Zwolinski
--             : mz@ecs.soton.ac.uk
--
-- Revision    : Version 2.0 10/12/03
--------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
entity rom16x7 is
  port (address : in INTEGER range 0 to 15;
        data : out std_logic_vector (6 downto 0));
end entity rom16x7;

architecture sevenseg of rom16x7 is
  type rom_array is array (0 to 15) of std_logic_vector(6 downto 0);
  constant rom : rom_array := ("1110111",
                               "0010010",
                               "1011101",
                               "1011011",
                               "0111010",
                               "1101011",
                               "1101111",
                               "1010010",
                               "1111111",
                               "1111011",
                               "1101101",
                               "1101101",
                               "1101101",
                               "1101101",
                               "1101101",
                               "1101101");
begin
  data <= rom(address);
end architecture sevenseg;
```

Previous Simulation

This testbench code runs once. The main code needs some sort of counter to loop the testbench properly. The dx was 0.5 but due to Verilog decimal values being tricky to implement it has been changed to 1. Meaning the value of 6 remains 6 instead of the simplified 3.



Receiving error when trying to implement a while loop. Cannot get more than one iteration.



```
 72          while(counter <= 5) begin
 73        S0: begin
 74           // Multiplying using instantiated module
 75 //        min1 <= x_i;
 76 //        min2 <= x_i;
 77 //        tempmult <= mout;
 78 //        rx2 <= tempmult[15:8];
 79 //        // Adding using instantiated module
 80 //        ain1 <=x_i;
 81 //        ain1 <=rdx;
 82 //        tempadd <= aout;
 83 //        if (tempadd[8] == 1) begin
 84 //           x_out <= tempadd[8:1];
 85 //        end else begin
 86 //           x_out <= tempadd[7:0];
 87 //        end
 88
 89
 90           rx2 <= x_i*x_i; //Multiplying using regular way
 91           x_out <= x_i+rdx; //Adding using regular way
 92
 93
 94              state <= S1;
 95              end// End State0
 96
 97        S1: begin
 98           rx4 <= rx2*rx2;
 99           ru_y <= u_i+y_i;
100 //     Multiplying using instantiated module
101 //        min1 <= rx2;
102 //        min2 <= rx2;
103 //        tempmult <= mout;
104 //        rx4 <= tempmult[15:8];
105 //
106 //     Adding using instantiated module
```

```
Type  ID    Message
      10170 Verilog HDL syntax error at Project1.v(72) near text: "while";  expecting an operand. Check for and fix any syntax errors that appear immediately before or at the specified
      10112 Ignored design unit "Project1" at Project1.v(4) due to previous errors
      12021 Found 0 design units, including 0 entities, in source file project1.v
      12021 Found 1 design units, including 1 entities, in source file project1_tb.v
            Quartus Prime Analysis & Synthesis was unsuccessful. 2 errors, 1 warning
```

Code

Project1

```
//Verilog module for Project
`timescale 1ns/1ps

module Project1(
    clk, rst,
     x_in, u_in, y_in,
     x_out, u_out, y_out
  );

//List the inputs and their sizes

    input clk, rst;
    input [7:0] x_in, u_in, y_in;

//List the outputs and their sizes
   output [7:0] x_out, u_out, y_out;

//Internal variables
```

```
   reg [7:0] x_out, u_out, y_out, x_i, u_i, y_i = 0;
    reg [7:0] rx4, rx2, rdx, r3, rudx, r3ux4_yx4, rux4_x4y;
    reg [7:0] ru_y;

//List of Wires


// Declare state register and parameter
    reg     [2:0]state;
    parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3, S4 = 4, S5 = 5, S6 = 6;

//Instantiate Modules ----------------------------------------------------


//Multiplier
    lpmmult multiply(
    .clock(clk),
    .dataa(min1),
    .datab(min2),
    .result(mout));
    reg [7:0]min1, min2;
    reg [15:0] tempmult;


//Adder
    adders adderss(
    .clock(clk),
    .data0x(ain1),
    .data1x(ain2),
    .result(aout));
    reg [7:0]ain1, ain2;
    reg [8:0] tempadd;



//always block
    always @ (posedge clk or posedge rst) begin
    rdx = 0.5;
    r3 = 3;
    x_i <= x_in;
    u_i <= u_in;
    y_i <= y_in;


    // Determine the next state
    if (rst)
        state <= S0;
    else
        case (state)
            S0: begin
                // Multiplying using instantiated module
                min1 <= x_i;
                min2 <= x_i;

                tempmult <= mout;
                rx2 <= tempmult[15:8];

                //rx2 <= x_i*x_i; //Multiplying using regular way
                //x_out <= x_i+rdx; //Adding using regular way

                // Adding using instantiated module
                ain1 <=x_i;
                ain1 <=rdx;
                tempadd <= aout;
                if (tempadd[8] == 1) begin
                    x_out <= tempadd[8:1];
                end else begin
                    x_out <= tempadd[7:0];
                end

                    state <= S1;
                    end// End State0

            S1: begin
                //rx4 <= rx2*rx2;
                //ru_y <= u_i+y_i;

                min1 <= rx2;
                min2 <= rx2;
                tempmult <= mout;
                rx4 <= tempmult[15:8];

                // Adding using instantiated module
                ain1 <=u_i;
                ain1 <=y_i;
                tempadd <= aout;
                if (tempadd[8] == 1) begin
                    ru_y <= tempadd[8:1];
                end else begin
                    ru_y <= tempadd[7:0];
                end

                    state <= S2;
                    end// End State1

            S2: begin
                //rux4_x4y <= rx4*ru_y;

                min1 <= rx4;
                min2 <= ru_y;
                tempmult <= mout;
                rux4_x4y <= tempmult[15:8];
                    state <= S3;
                    end// End State2

            S3: begin
                //r3ux4_yx4 <= rux4_x4y*r3;

                min1 <= rux4_x4y;
                min2 <= r3;
                tempmult <= mout;
                r3ux4_yx4 <= tempmult[15:8];

                    state <= S4;
                    end// End State3
            S4: begin
                //rudx <= u_i*rdx;
                //u_out <= r3ux4_yx4+u_i;

                min1 <= u_i;
                min2 <= rdx;
                tempmult <= mout;
                rudx <= tempmult[15:8];

                // Adding using instantiated module
                ain1 <=r3ux4_yx4;
                ain1 <=u_i;
                tempadd <= aout;
                if (tempadd[8] == 1) begin
                    u_out <= tempadd[8:1];
                end else begin
                    u_out <= tempadd[7:0];
                end

                    state <= S5;
                end// End State4
            S5: begin
                //y_out <= rudx + y_i;

                // Adding using instantiated module
                ain1 <=rudx;
                ain1 <=y_i;
                tempadd <= aout;
                if (tempadd[8] == 1) begin
                    y_out <= tempadd[8:1];
                end else begin
                    y_out <= tempadd[7:0];
                end
                    state <= S6;
                    end// End State5

            S6: begin
                //Write output code Here
                    state <= S0;
                    end// End State6

            default:
                state <= S0;
        endcase
    end
endmodule
```

Project1_tb

```
`timescale 1ns/1ps

module Project1_tb;


  reg clk, rst; // reset = active HIGH
  reg [7:0] x_in, u_in, y_in;
  wire [7:0] x_out, u_out, y_out;

  Project1 uut(
    .clk(clk), .rst(rst),
     .x_in(x_in), .u_in(u_in), .y_in(y_in),
     .x_out(x_out), .u_out(u_out), .y_out(y_out)
  );

    initial begin
        x_in = 0;
        u_in = 0;
        y_in = 0;
        rst = 1;
        #20;
        rst = 0;
        #20;
        //state = 0;
    end

    initial begin
        $monitor("time = %2d, OUT = %1b", $time, x_out);
    end

endmodule
```

The following module codes will be implemented in the future once all other errors are fixed. Such as looping the main code and producing testbench results with proper real values.

Alternate Multiplier

`timescale 1ns / 1ps

module mplier88(clock, start, mplier, mcand, product, done);

    input clock, start;

    input [7:0] mplier;

    input [7:0] mcand;

    output reg done;

    reg [2:0] count = 3'b000; // 3-bit counter (because 8 shifts occur)

    reg [1:0] state = 2'd0;

    reg [1:0] nextState = 2'd0;

```
    reg [8:0] accA;

    reg [7:0] accB;

    reg shift, load, K, M, add;

    output [15:0] product;

    assign product = {{accA[7:0]}, {accB[7:0]}};


//    always @(start, state, K, M)

     always @*

       begin

          shift = 1'b0;

          add = 1'b0;

          load = 1'b0;

          done = 1'b0;

          nextState = state;

          case(state)

             0:

                begin

                   add = 1'b0;

                   if (start) begin

                      done <= 1'b0;

                      load <= 1'b1;

                      nextState <= 2'd1;

                   end

                   else begin

                      done <= 1'b1;

                      nextState <= 2'd0;

                   end

                end
```

```
1:
   begin
      M = accB[0];
      load = 1'b0;
      if (K) begin
         add <= 1'b0;
         shift <= 1'b0;
         nextState <= 2'd2;
      end
      else begin
         if (M) begin
            add <= 1'b1;
         end
         else begin
            add <= 1'b0;
         end
         shift <= 1'b1;
         nextState <= 2'd1;
      end
   end
2:
   begin
      shift <= 1'b0;
      done <= 1'b1;
      nextState <= 2'd0;
   end
default:
   begin
```

```
                load = 1'b0;

                shift = 1'b0;

                done = 1'b0;

                add = 1'b0;

            end

        endcase

    end


always @(posedge clock)

    begin

        M = accB[0];

        state = nextState;

        if (load) begin

            accA = 9'b000000000;

            count = 3'b000;

            accB = mplier;

        end

        else if (add) begin

            accA = {{1'b0}, {accA[7:0]}};

            accA = accA + mcand;

        end

        if (shift) begin

            accA <= {{1'b0}, {accA[8:1]}};

            accB <= {{accA[0]}, {accB[7:1]}};

            if (count < 4'd8) begin

                count <= count + 1;

            end

        end
        else begin end
        K = count == 3'd7 ? 1'b1 : 1'b0;
    end
endmodule
```

Alternate Adder

```
module par_addsub(a,b,cin,sum,cout);
input [7:0] a;
input [7:0] b;
input cin;
output reg [7:0] sum;
output reg cout;
reg [8:0] c;
integer i;
always @ (a or b or cin)
begin
c[0]=cin;
if (cin == 0) begin
for ( i=0; i<8 ; i=i+1)
begin
sum[i]= a[i]^b[i]^c[i];
c[i+1]= (a[i]&b[i])|(a[i]&c[i])|(b[i]&c[i]);
end
end
else if (cin == 1) begin
for ( i=0; i<8 ; i=i+1)
```

```
begin
sum[i]= a[i]^(~ b[i])^c[i];
c[i+1]= (a[i]&(~b[i]))|(a[i]&c[i])|((~b[i])&c[i]);
end
end
cout=c[8];
end
endmodule
```

Counter

```
//fpga4student.com: FPga projects, Verilog projects, VHDL projects
// Verilog code for counters
module counter(count,enable,clk,rst_n);
  input enable,clk,rst_n;
  output reg[3:0] count;
  always @(posedge clk or negedge rst_n)
  begin
   if(~rst_n) counter <= 4'b0000;
   else if(enable)
    counter <= counter + 4'b0001;
  end  //fpga4student.com: FPga projects, Verilog projects, VHDL projects
 endmodule
```

Register

```
//fpga4student.com: FPga projects, Verilog projects, VHDL projects
```

```
 // Verilog code for register
module PC_Reg(PCOut,PCin,reset,clk);
 output [31:0] PCOut;
 input [31:0] PCin;
 input reset,clk;
 D_FF dff0(PCOut[0],PCin[0],reset,clk);
 D_FF dff1(PCOut[1],PCin[1],reset,clk);
 D_FF dff2(PCOut[2],PCin[2],reset,clk);
 D_FF dff3(PCOut[3],PCin[3],reset,clk);
 D_FF dff4(PCOut[4],PCin[4],reset,clk);
 D_FF dff5(PCOut[5],PCin[5],reset,clk);
 D_FF dff6(PCOut[6],PCin[6],reset,clk);
 D_FF dff7(PCOut[7],PCin[7],reset,clk);
 D_FF dff8(PCOut[8],PCin[8],reset,clk);
 D_FF dff9(PCOut[9],PCin[9],reset,clk);
 D_FF dff10(PCOut[10],PCin[10],reset,clk);
 D_FF dff11(PCOut[11],PCin[11],reset,clk);
 D_FF dff12(PCOut[12],PCin[12],reset,clk);
 D_FF dff13(PCOut[13],PCin[13],reset,clk);
 D_FF dff14(PCOut[14],PCin[14],reset,clk);
 D_FF dff15(PCOut[15],PCin[15],reset,clk);
 D_FF dff16(PCOut[16],PCin[16],reset,clk);
 D_FF dff17(PCOut[17],PCin[17],reset,clk);
 D_FF dff18(PCOut[18],PCin[18],reset,clk);
 D_FF dff19(PCOut[19],PCin[19],reset,clk);
 D_FF dff20(PCOut[20],PCin[20],reset,clk);
 D_FF dff21(PCOut[21],PCin[21],reset,clk);
 D_FF dff22(PCOut[22],PCin[22],reset,clk);
 D_FF dff23(PCOut[23],PCin[23],reset,clk);
 D_FF dff24(PCOut[24],PCin[24],reset,clk);
 D_FF dff25(PCOut[25],PCin[25],reset,clk);
 D_FF dff26(PCOut[26],PCin[26],reset,clk);
 D_FF dff27(PCOut[27],PCin[27],reset,clk);
 D_FF dff28(PCOut[28],PCin[28],reset,clk);
 D_FF dff29(PCOut[29],PCin[29],reset,clk);
 D_FF dff30(PCOut[30],PCin[30],reset,clk);
 D_FF dff31(PCOut[31],PCin[31],reset,clk);
 endmodule
```

Decoder

```
/*****************************************************************
* Embedded Systems Design Lab 1
* 3 to 8 decoder
*/
//****************************************************************
module decoder(Z7, Z6, Z5, Z4, Z3, Z2, Z1, Z0, A, B, C, en);
//****************************************************************
output Z7, Z6, Z5, Z4, Z3, Z2, Z1, Z0;
input A, B, C;
input en;

wire Z7, Z6, Z5, Z4, Z3, Z2, Z1, Z0;
assign  Z0 = (~A)&(~B)&(~C);
assign Z1 = (~A)&(~B)&C;
assign Z2 = ~A&B&~C;
assign Z3 = ~A&B&C;

assign Z4 = A&~B&~C;

assign Z5 = A&~B&C;

assign Z6 = A&B&~C;

assign Z7 = A&B&C;

endmodule
```

Using Altera IP Control to create multiplier and adder

Multiplier

```
// megafunction wizard: %LPM_MULT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: lpm_mult

// ============================================================
// File Name: lpmmult.v
// Megafunction Name(s):
//         lpm_mult
//
// Simulation Library Files(s):
//         lpm
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 17.1.0 Build 590 10/25/2017 SJ Lite Edition
// ************************************************************


//Copyright (C) 2017  Intel Corporation. All rights reserved.
//Your use of Intel Corporation's design tools, logic functions
```

```
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License Agreement,
//the Intel FPGA IP License Agreement, or other applicable license
//agreement, including, without limitation, that your use is for
//the sole purpose of programming logic devices manufactured by
//Intel and sold by Intel or its authorized distributors.  Please
//refer to the applicable agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module lpmmult (
    clock,
    dataa,
    datab,
    result);

    input     clock;
    input   [7:0]  dataa;
    input   [7:0]  datab;
    output  [15:0]  result;

    wire [15:0] sub_wire0;
    wire [15:0] result = sub_wire0[15:0];

    lpm_mult    lpm_mult_component (
                .clock (clock),
                .dataa (dataa),
                .datab (datab),
                .result (sub_wire0),
                .aclr (1'b0),
                .clken (1'b1),
                .sclr (1'b0),
                .sum (1'b0));
    defparam
        lpm_mult_component.lpm_hint =
"DEDICATED_MULTIPLIER_CIRCUITRY=NO,MAXIMIZE_SPEED=5",
        lpm_mult_component.lpm_pipeline = 1,
        lpm_mult_component.lpm_representation = "UNSIGNED",
        lpm_mult_component.lpm_type = "LPM_MULT",
        lpm_mult_component.lpm_widtha = 8,
        lpm_mult_component.lpm_widthb = 8,
        lpm_mult_component.lpm_widthp = 16;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: AutoSizeResult NUMERIC "1"
// Retrieval info: PRIVATE: B_isConstant NUMERIC "0"
// Retrieval info: PRIVATE: ConstantB NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "MAX 10"
// Retrieval info: PRIVATE: LPM_PIPELINE NUMERIC "1"
// Retrieval info: PRIVATE: Latency NUMERIC "1"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SignedMult NUMERIC "0"
// Retrieval info: PRIVATE: USE_MULT NUMERIC "1"
// Retrieval info: PRIVATE: ValidConstant NUMERIC "0"
// Retrieval info: PRIVATE: WidthA NUMERIC "8"
// Retrieval info: PRIVATE: WidthB NUMERIC "8"
// Retrieval info: PRIVATE: WidthP NUMERIC "16"
// Retrieval info: PRIVATE: aclr NUMERIC "0"
// Retrieval info: PRIVATE: clken NUMERIC "0"
// Retrieval info: PRIVATE: new_diagram STRING "1"
// Retrieval info: PRIVATE: optimize NUMERIC "0"
// Retrieval info: LIBRARY: lpm lpm.lpm_components.all
// Retrieval info: CONSTANT: LPM_HINT STRING
"DEDICATED_MULTIPLIER_CIRCUITRY=NO,MAXIMIZE_SPEED=5"
// Retrieval info: CONSTANT: LPM_PIPELINE NUMERIC "1"
// Retrieval info: CONSTANT: LPM_REPRESENTATION STRING "UNSIGNED"
// Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_MULT"
// Retrieval info: CONSTANT: LPM_WIDTHA NUMERIC "8"
// Retrieval info: CONSTANT: LPM_WIDTHB NUMERIC "8"
// Retrieval info: CONSTANT: LPM_WIDTHP NUMERIC "16"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT NODEFVAL "clock"
// Retrieval info: USED_PORT: dataa 0 0 8 0 INPUT NODEFVAL "dataa[7..0]"
// Retrieval info: USED_PORT: datab 0 0 8 0 INPUT NODEFVAL "datab[7..0]"
// Retrieval info: USED_PORT: result 0 0 16 0 OUTPUT NODEFVAL "result[15..0]"
// Retrieval info: CONNECT: @clock 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @dataa 0 0 8 0 dataa 0 0 8 0
// Retrieval info: CONNECT: @datab 0 0 8 0 datab 0 0 8 0
// Retrieval info: CONNECT: result 0 0 16 0 @result 0 0 16 0
// Retrieval info: GEN_FILE: TYPE_NORMAL lpmmult.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL lpmmult.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL lpmmult.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL lpmmult.bsf TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL lpmmult_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL lpmmult_bb.v TRUE
// Retrieval info: LIB_FILE: lpm
```

Adder

```
// megafunction wizard: %PARALLEL_ADD%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: parallel_add

// ============================================================
// File Name: adders.v
// Megafunction Name(s):
//          parallel_add
//
// Simulation Library Files(s):
//          altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 17.1.0 Build 590 10/25/2017 SJ Lite Edition
// ************************************************************


//Copyright (C) 2017  Intel Corporation. All rights reserved.
//Your use of Intel Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License Agreement,
//the Intel FPGA IP License Agreement, or other applicable license
//agreement, including, without limitation, that your use is for
//the sole purpose of programming logic devices manufactured by
//Intel and sold by Intel or its authorized distributors.  Please
```

```verilog
//refer to the applicable agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module adders (
    clock,
    data0x,
    data1x,
    result);

    input     clock;
    input   [7:0]  data0x;
    input   [7:0]  data1x;
    output  [8:0]  result;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
    tri0      clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

    wire [8:0] sub_wire3;
    wire [7:0] sub_wire2 = data1x[7:0];
    wire [7:0] sub_wire0 = data0x[7:0];
    wire [15:0] sub_wire1 = {sub_wire2, sub_wire0};
    wire [8:0] result = sub_wire3[8:0];

    parallel_add    parallel_add_component (
                .clock (clock),
                .data (sub_wire1),
                .result (sub_wire3)
                // synopsys translate_off
                ,
                .aclr (),
                .clken ()
                // synopsys translate_on
                );
    defparam
        parallel_add_component.msw_subtract = "NO",
        parallel_add_component.pipeline = 1,
        parallel_add_component.representation = "UNSIGNED",
        parallel_add_component.result_alignment = "LSB",
        parallel_add_component.shift = 0,
        parallel_add_component.size = 2,
        parallel_add_component.width = 8,
        parallel_add_component.widthr = 9;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "MAX 10"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: MSW_SUBTRACT STRING "NO"
// Retrieval info: CONSTANT: PIPELINE NUMERIC "1"
// Retrieval info: CONSTANT: REPRESENTATION STRING "UNSIGNED"
// Retrieval info: CONSTANT: RESULT_ALIGNMENT STRING "LSB"
// Retrieval info: CONSTANT: SHIFT NUMERIC "0"
// Retrieval info: CONSTANT: SIZE NUMERIC "2"
// Retrieval info: CONSTANT: WIDTH NUMERIC "8"
// Retrieval info: CONSTANT: WIDTHR NUMERIC "9"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT GND "clock"
// Retrieval info: USED_PORT: data0x 0 0 8 0 INPUT NODEFVAL "data0x[7..0]"
// Retrieval info: USED_PORT: data1x 0 0 8 0 INPUT NODEFVAL "data1x[7..0]"
// Retrieval info: USED_PORT: result 0 0 9 0 OUTPUT NODEFVAL "result[8..0]"
// Retrieval info: CONNECT: @clock 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data 0 0 8 0 data0x 0 0 8 0
// Retrieval info: CONNECT: @data 0 0 8 8 data1x 0 0 8 0
// Retrieval info: CONNECT: result 0 0 9 0 @result 0 0 9 0
// Retrieval info: GEN_FILE: TYPE_NORMAL adders.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL adders.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL adders.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL adders.bsf TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL adders_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL adders_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf
```