ELEC8600-1-R-2026W
Reconfigurable Computing

Assignment #1:
Digital System Modeling and Simulation using VHDL and Intel Quartus Prime CAD tool

Submitted by:

Lawaz Islam

Student ID: 110189519

Kamran Shahid

Student ID: 110184367

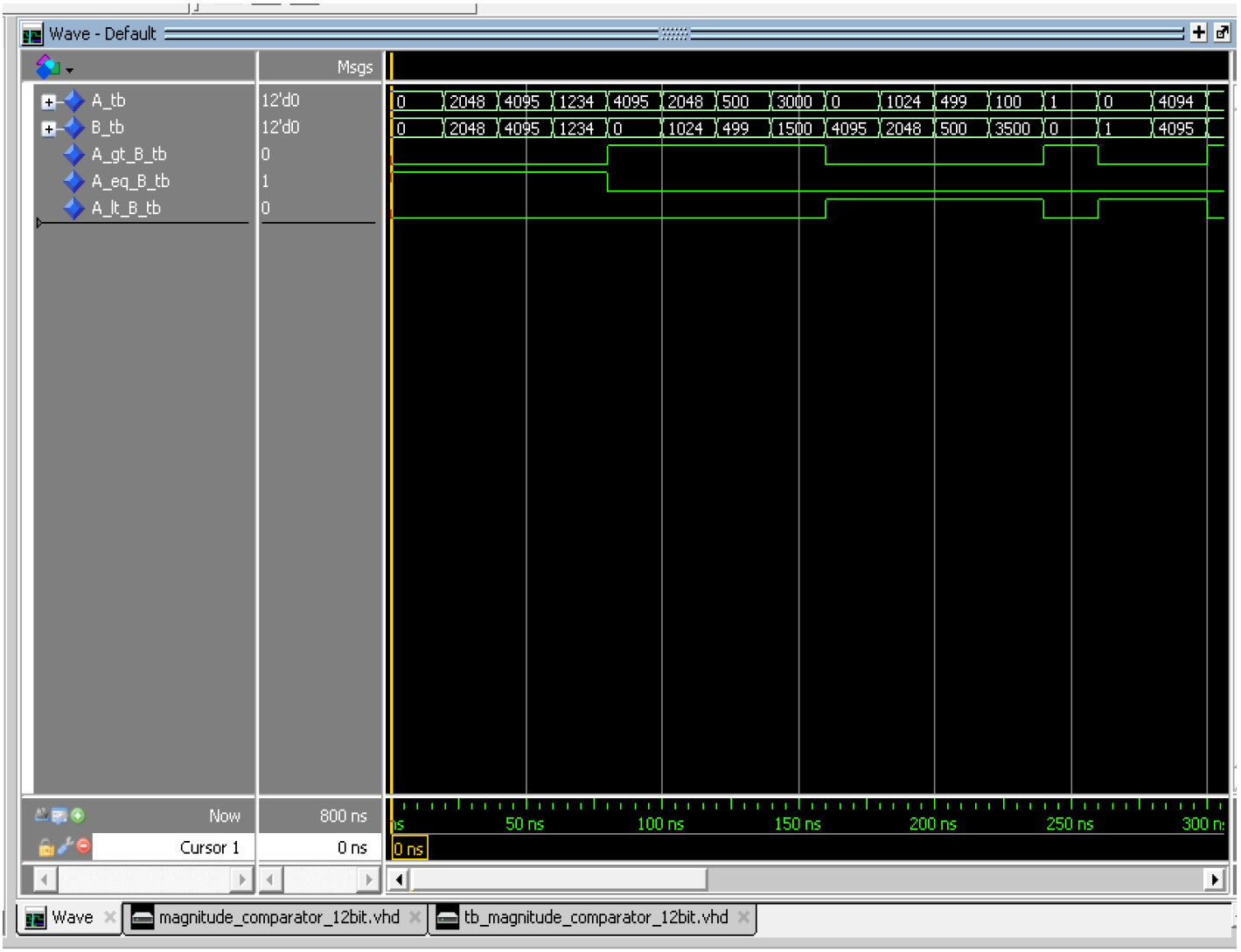
Part I

MAIN MODULE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity magnitude_comparator_12bit is
    port (
        A      : in  STD_LOGIC_VECTOR(11 downto 0);
        B      : in  STD_LOGIC_VECTOR(11 downto 0);
        A_gt_B : out STD_LOGIC;
        A_eq_B : out STD_LOGIC;
        A_lt_B : out STD_LOGIC
    );
end entity magnitude_comparator_12bit;
architecture behavioral of magnitude_comparator_12bit is
begin
    compare_proc : process(A, B)
    begin
        -- Default all outputs to '0'
        A_gt_B <= '0';
        A_eq_B <= '0';
        A_lt_B <= '0';
        if A > B then
            A_gt_B <= '1';
```

```
        elsif A = B then
            A_eq_B <= '1';
        else
            A_lt_B <= '1';
        end if;
    end process compare_proc;
end architecture behavioral;
```

OUTPUT:



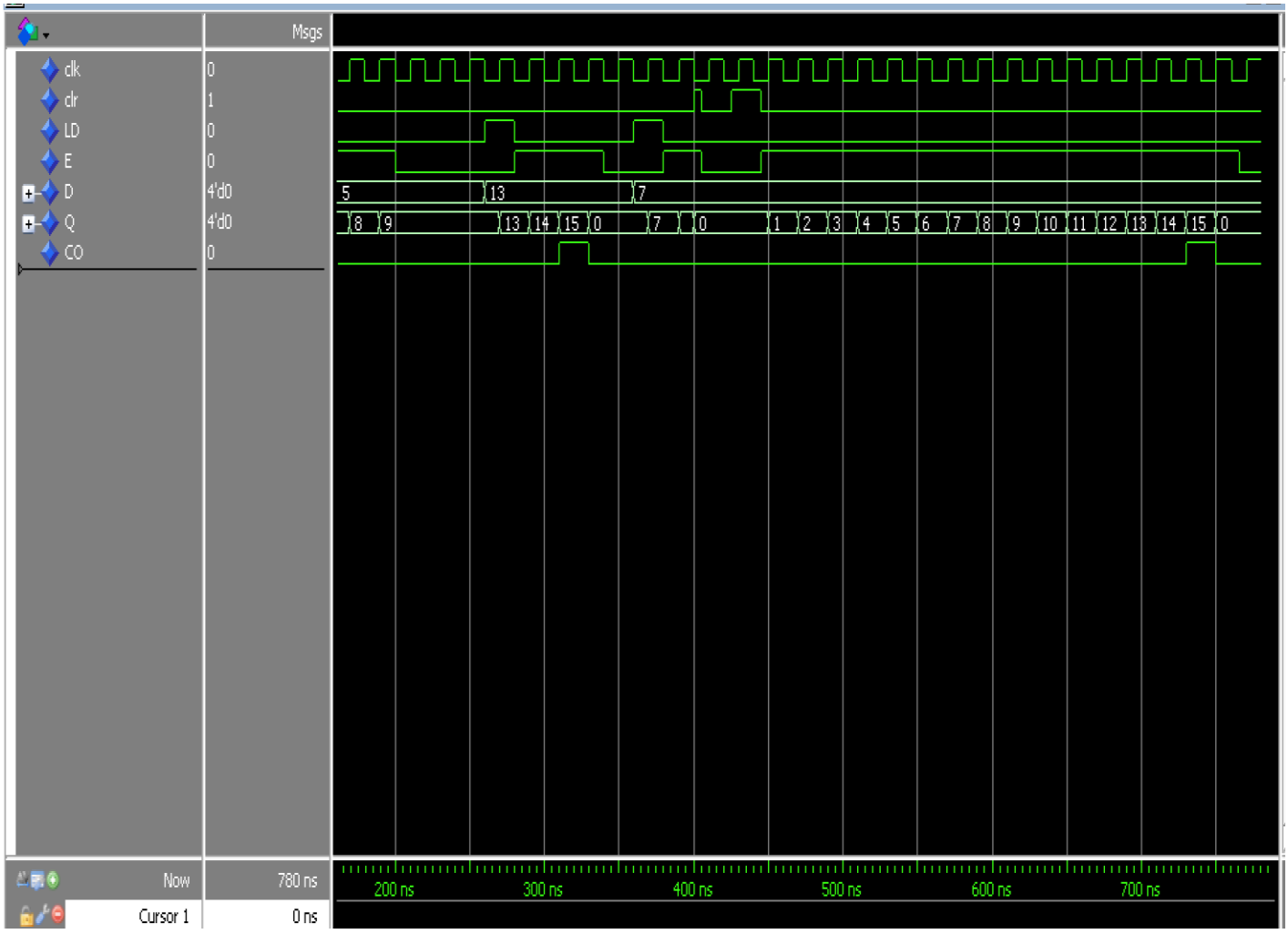
Part II

MAIN MODULE:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity counter_4bit is
    port (
        clk  : in  std_logic;
        clr  : in  std_logic;
        E    : in  std_logic;
        LD   : in  std_logic;
        D    : in  std_logic_vector(3 downto 0);
        Q    : out std_logic_vector(3 downto 0);
        CO   : out std_logic
    );
end counter_4bit;
architecture behavioral of counter_4bit is
    signal count : unsigned(3 downto 0);
begin
    process(clk, clr)
    begin
        if clr = '1' then
            count <= "0000";
        elsif rising_edge(clk) then
            if LD = '1' then
                count <= unsigned(D);
            elsif E = '1' then
```

```
            count <= count + 1;
        else
            count <= count;
        end if;
    end if;
  end process;
  Q  <= std_logic_vector(count);
  CO <= '1' when count = "1111" else '0';
end behavioral;
```

OUTPUT:



Part III

MAIN MODULE:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity counter_4bit is
    port (
        clk : in  std_logic;
        clr : in  std_logic;
        E   : in  std_logic;
        LD  : in  std_logic;
        D   : in  std_logic_vector(3 downto 0);
        Q   : out std_logic_vector(3 downto 0);
        CO  : out std_logic
    );
end entity counter_4bit;
architecture behavioral of counter_4bit is
    signal count : unsigned(3 downto 0);
begin
    process(clk, clr)
    begin
        if clr = '1' then
            count <= "0000";
        elsif rising_edge(clk) then
            if LD = '1' then
                count <= unsigned(D);
            elsif E = '1' then
```

```
            count <= count + 1;
        end if;
    end if;
  end process;
  Q  <= std_logic_vector(count);
  CO <= '1' when count = "1111" else '0';
end architecture behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity magnitude_comparator_12bit is
  port (
    A      : in  std_logic_vector(11 downto 0);
    B      : in  std_logic_vector(11 downto 0);
    A_gt_B : out std_logic;
    A_eq_B : out std_logic;
    A_lt_B : out std_logic
  );
end entity magnitude_comparator_12bit;
architecture behavioral of magnitude_comparator_12bit is
begin
  process(A, B)
  begin
    A_gt_B <= '0';
    A_eq_B <= '0';
    A_lt_B <= '0';
```

```
        if    A > B then A_gt_B <= '1';
        elsif A = B then A_eq_B <= '1';
        else             A_lt_B <= '1';
        end if;
    end process;
end architecture behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity counter_12bit is
    port (
        clk : in  std_logic;
        clr : in  std_logic;
        E   : in  std_logic;
        LD  : in  std_logic;
        D   : in  std_logic_vector(11 downto 0);
        Q   : out std_logic_vector(11 downto 0)
    );
end entity counter_12bit;

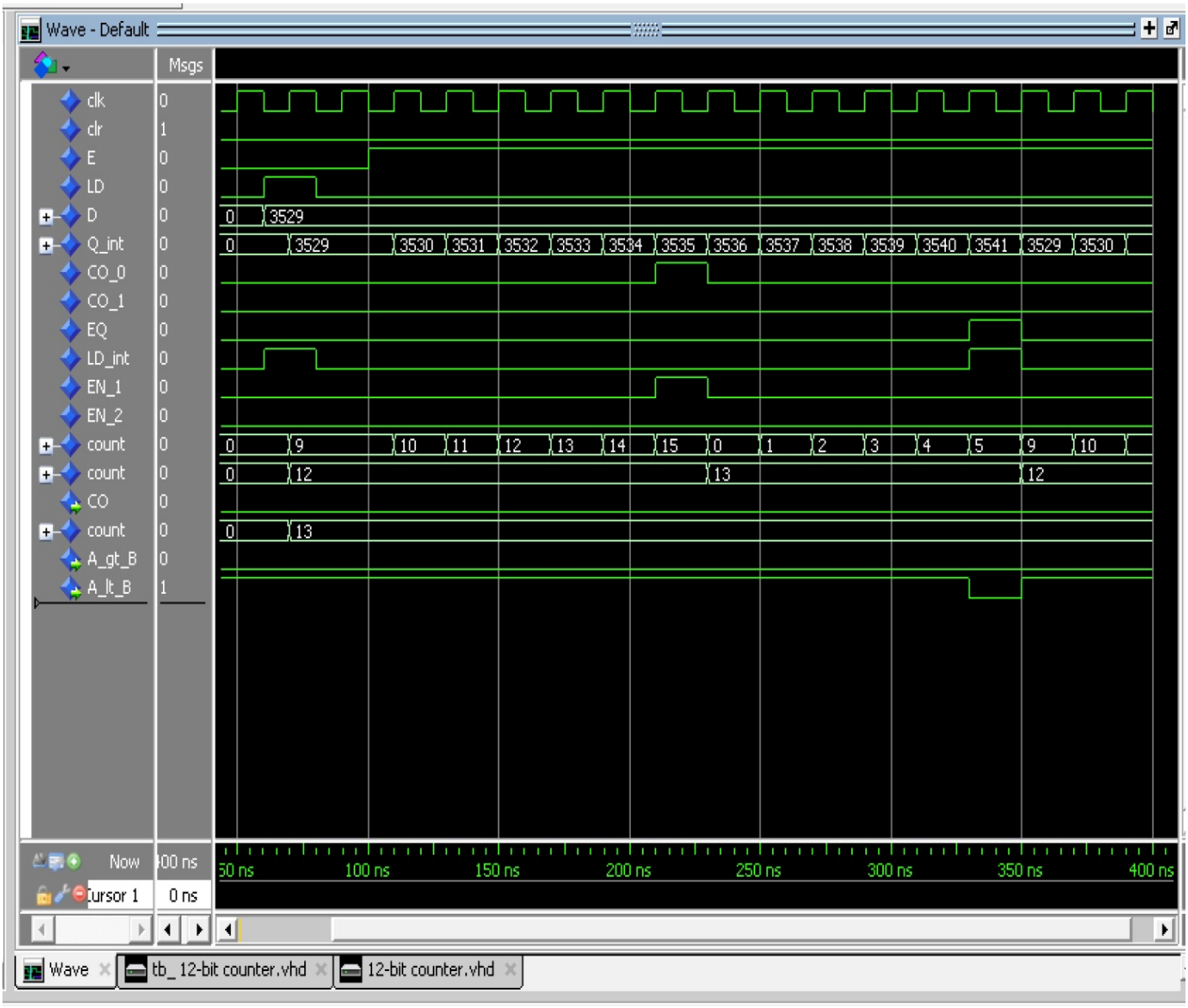
architecture structural of counter_12bit is
    component counter_4bit is
        port (
            clk : in  std_logic;
            clr : in  std_logic;
            E   : in  std_logic;
            LD  : in  std_logic;
```

```
        D   : in  std_logic_vector(3 downto 0);
        Q   : out std_logic_vector(3 downto 0);
        CO  : out std_logic
      );
  end component;
  component magnitude_comparator_12bit is
    port (
      A     : in  std_logic_vector(11 downto 0);
      B     : in  std_logic_vector(11 downto 0);
      A_gt_B : out std_logic;
      A_eq_B : out std_logic;
      A_lt_B : out std_logic
    );
  end component;
  signal Q_int  : std_logic_vector(11 downto 0);
  signal CO_0   : std_logic;
  signal CO_1   : std_logic;
  signal EQ     : std_logic;
  signal LD_int : std_logic;
  signal EN_1   : std_logic;
  signal EN_2   : std_logic;
  constant TC : std_logic_vector(11 downto 0) := "110111010101";
begin
  LD_int <= LD or EQ;
  EN_1   <= E and CO_0;
  EN_2   <= E and CO_0 and CO_1;
  C0 : counter_4bit
```

```
    port map (
        clk => clk, clr => clr,
        E   => E,      LD => LD_int,
        D   => D(3 downto 0),
        Q   => Q_int(3 downto 0),
        CO  => CO_0
    );
C1 : counter_4bit
    port map (
        clk => clk, clr => clr,
        E   => EN_1,   LD => LD_int,
        D   => D(7 downto 4),
        Q   => Q_int(7 downto 4),
        CO  => CO_1
    );
C2 : counter_4bit
    port map (
        clk => clk, clr => clr,
        E   => EN_2,   LD => LD_int,
        D   => D(11 downto 8),
        Q   => Q_int(11 downto 8),
        CO  => open
    );
COMP : magnitude_comparator_12bit
    port map (
        A      => Q_int,
        B      => TC,
```

```
        A_gt_B => open,
        A_eq_B => EQ,
        A_lt_B => open
    );
  Q <= Q_int;
end architecture structural;
```

OUTPUT:



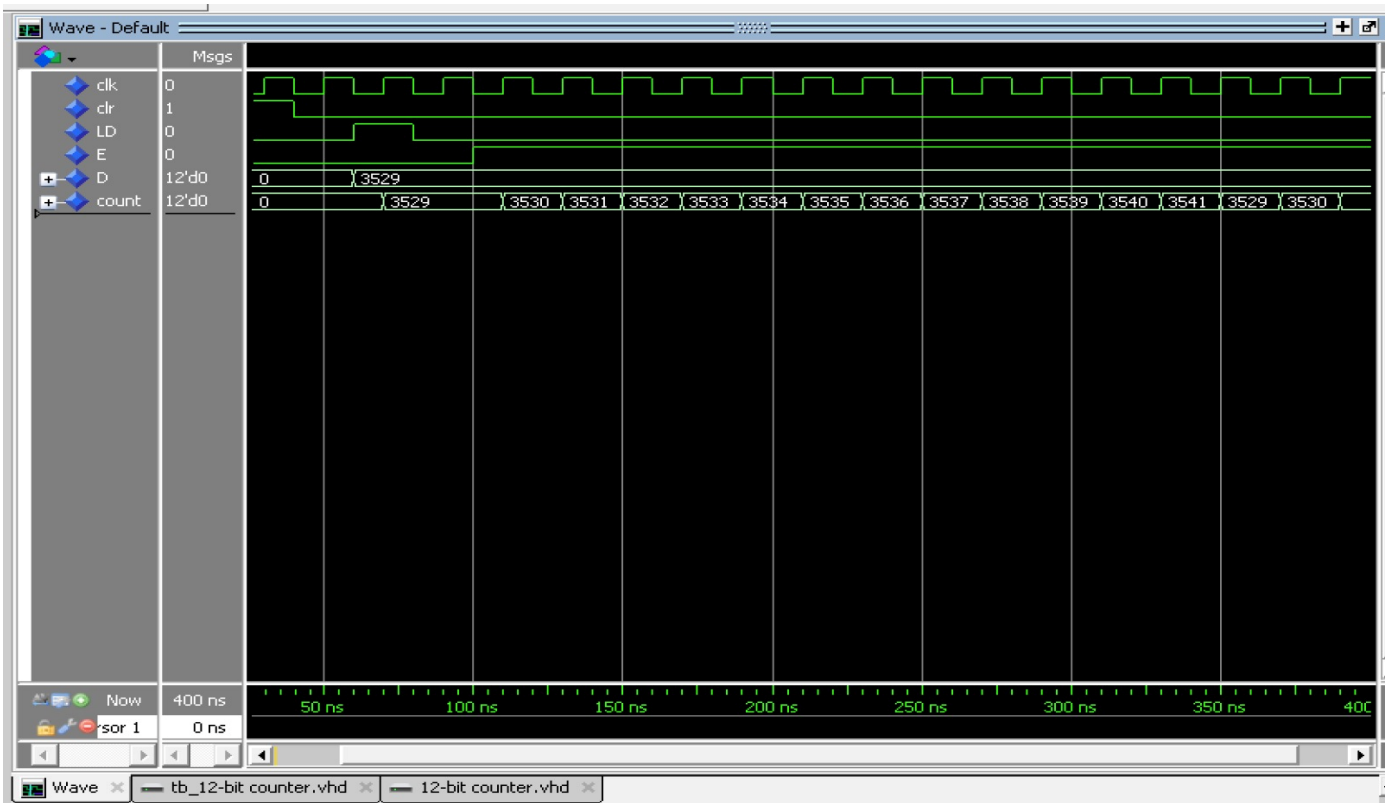
Part IV

MAIN MODULE:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity counter_12bit is
    port (
        clk : in  std_logic;
        clr : in  std_logic;
        E   : in  std_logic;
        LD  : in  std_logic;
        D   : in  std_logic_vector(11 downto 0);
        Q   : out std_logic_vector(11 downto 0)
    );
end entity counter_12bit;
architecture behavioral of counter_12bit is
    constant TC    : unsigned(11 downto 0) := to_unsigned(3541, 12);
    signal   count : unsigned(11 downto 0) := (others => '0');
begin
    process(clk, clr)
    begin
        if clr = '1' then
            count <= (others => '0');
        elsif rising_edge(clk) then
            if LD = '1' then
                count <= unsigned(D);
```

```
        elsif E = '1' then
            if count = TC then
                count <= unsigned(D);
            else
                count <= count + 1;
            end if;
        end if;
    end if;
  end process;
  Q <= std_logic_vector(count);
end architecture behavioral;
```

OUTPUT:



ELEC8600-1-R-2026W

Reconfigurable Computing

Assignment #2:

Digital System Modeling, Simulation and Synthesis for FPGAs using VHDL and Intel Quartus Prime CAD tool

Submitted by:

Lawaz Islam

Student ID: 110189519

Kamran Shahid

Student ID: 110184367

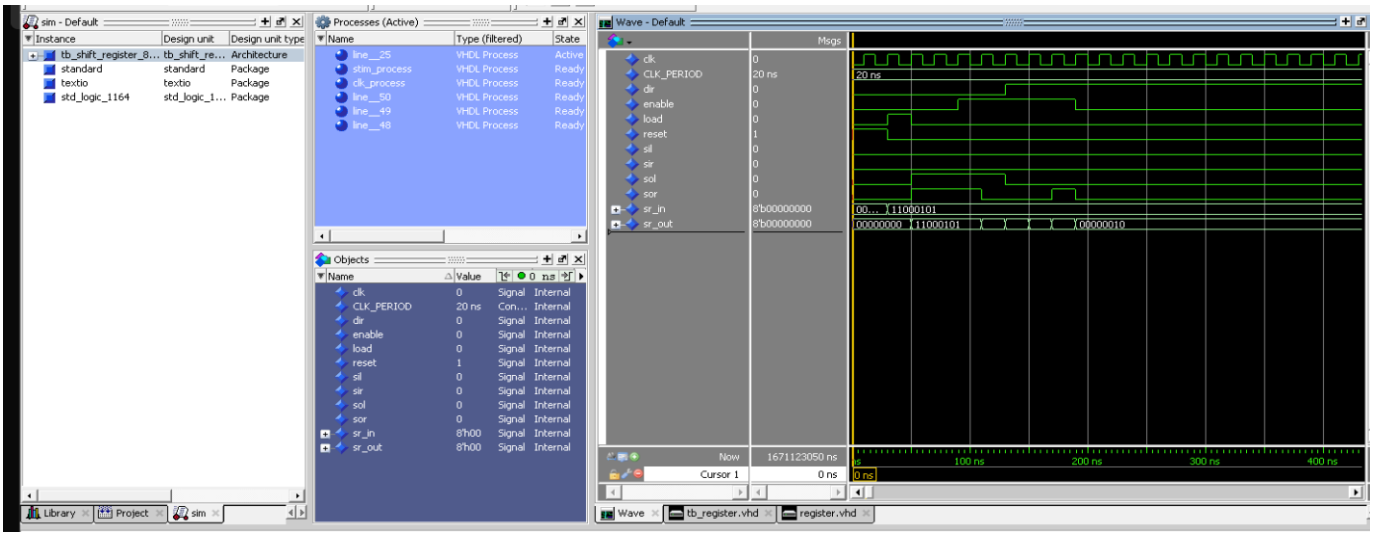
Part I

Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity shift_register_8bit is
  port (
    clk   : in  STD_LOGIC;
    reset  : in  STD_LOGIC;
    load   : in  STD_LOGIC;
    enable : in  STD_LOGIC;
    dir    : in  STD_LOGIC;
    sil    : in  STD_LOGIC;
    sir    : in  STD_LOGIC;
    sr_in  : in  STD_LOGIC_VECTOR(7 downto 0);
    sol    : out STD_LOGIC;
    sor    : out STD_LOGIC;
    sr_out : out STD_LOGIC_VECTOR(7 downto 0)
  );
end entity shift_register_8bit;
architecture behavioral of shift_register_8bit is
  signal reg : STD_LOGIC_VECTOR(7 downto 0);
begin
  process(clk, reset)
  begin
    if reset = '1' then
```

```
        reg <= (others => '0');
      elsif rising_edge(clk) then
        if load = '1' then
          reg <= sr_in;
        elsif enable = '1' then
          if dir = '0' then
            reg <= reg(6 downto 0) & sil;
          else
            reg <= sir & reg(7 downto 1);
          end if;
        end if;
      end if;
   end process;
   sr_out <= reg;
   sol   <= reg(7);
   sor   <= reg(0);
end architecture behavioral;
```

Output:



Part II

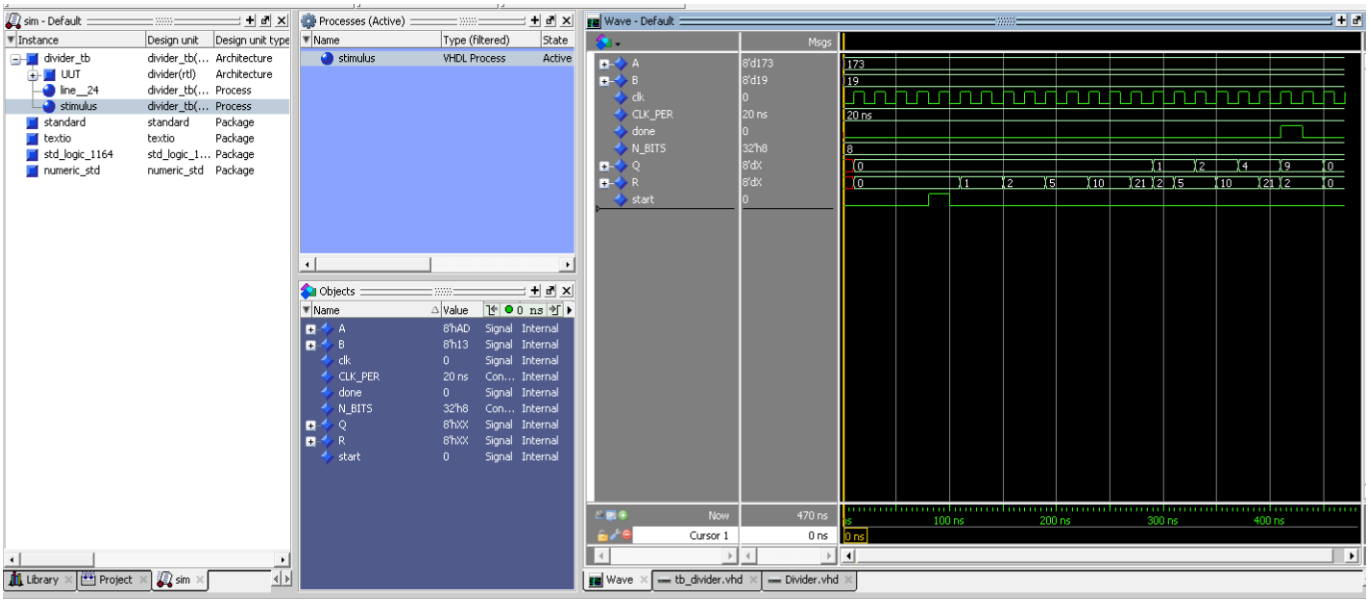
Code:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity divider is
  generic (n : integer := 8);
  port (
    clk   : in  std_logic;
    start : in  std_logic;
    A     : in  std_logic_vector(n-1 downto 0);
    B     : in  std_logic_vector(n-1 downto 0);
    Q     : out std_logic_vector(n-1 downto 0);
    R     : out std_logic_vector(n-1 downto 0);
    done  : out std_logic
  );
end entity divider;
architecture rtl of divider is
  type state_type is (S1, S2, S3, S4);
  signal state : state_type := S1;
  signal RA : unsigned(n-1 downto 0);
  signal RB : unsigned(n-1 downto 0);
  signal RR : unsigned(n-1 downto 0);
  signal RQ : unsigned(n-1 downto 0);
 signal C  : integer range 0 to n-1;
begin
```

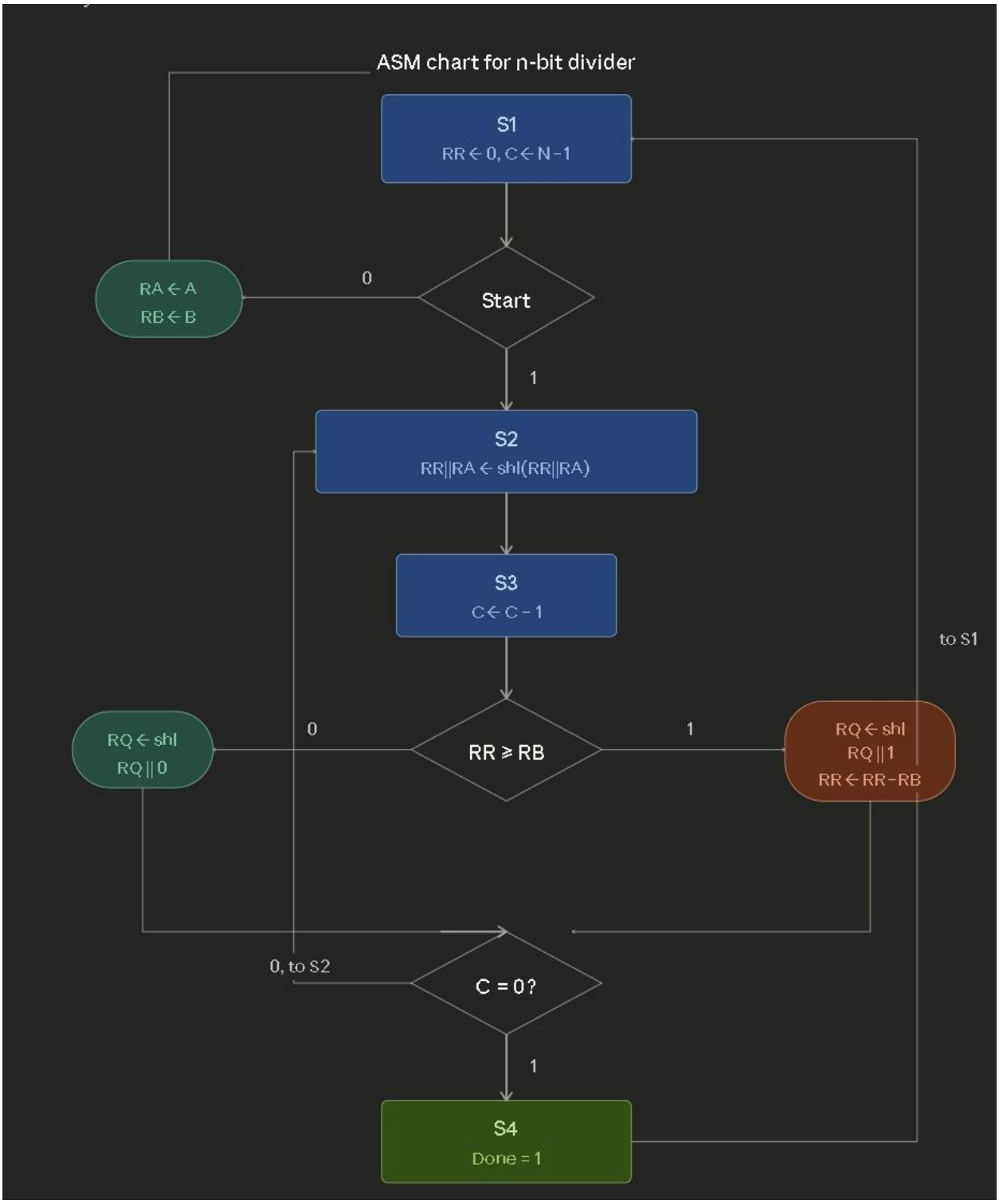
```
Q   <= std_logic_vector(RQ);
  R   <= std_logic_vector(RR);
  done <= '1' when state = S4 else '0';
  process(clk)
    variable con_result : unsigned(2*n-1 downto 0);
  begin
    if rising_edge(clk) then
      case state is
        when S1 =>
          RR   <= (others => '0');
          RQ   <= (others => '0');
          RA   <= unsigned(A);
          RB   <= unsigned(B);
          C    <= n - 1;
          if start = '1' then
            state <= S2;
          end if;
        when S2 =>
          con_result := RR & RA;
          con_result := con_result(2*n-2 downto 0) & '0';
          RR   <= con_result(2*n-1 downto n);
          RA   <= con_result(n-1 downto 0);
          state <= S3;
        when S3 =>
          if RR >= RB then
            RQ <= RQ(n-2 downto 0) & '1';
```

```
            RR <= RR - RB;
          else
            RQ <= RQ(n-2 downto 0) & '0';
          end if;
          if C = 0 then
            state <= S4;
          else
            C    <= C - 1;
            state <= S2;
          end if;
        when S4 =>
          state <= S1;
      end case;
    end if;
  end process;
end architecture rtl;
```

Output:

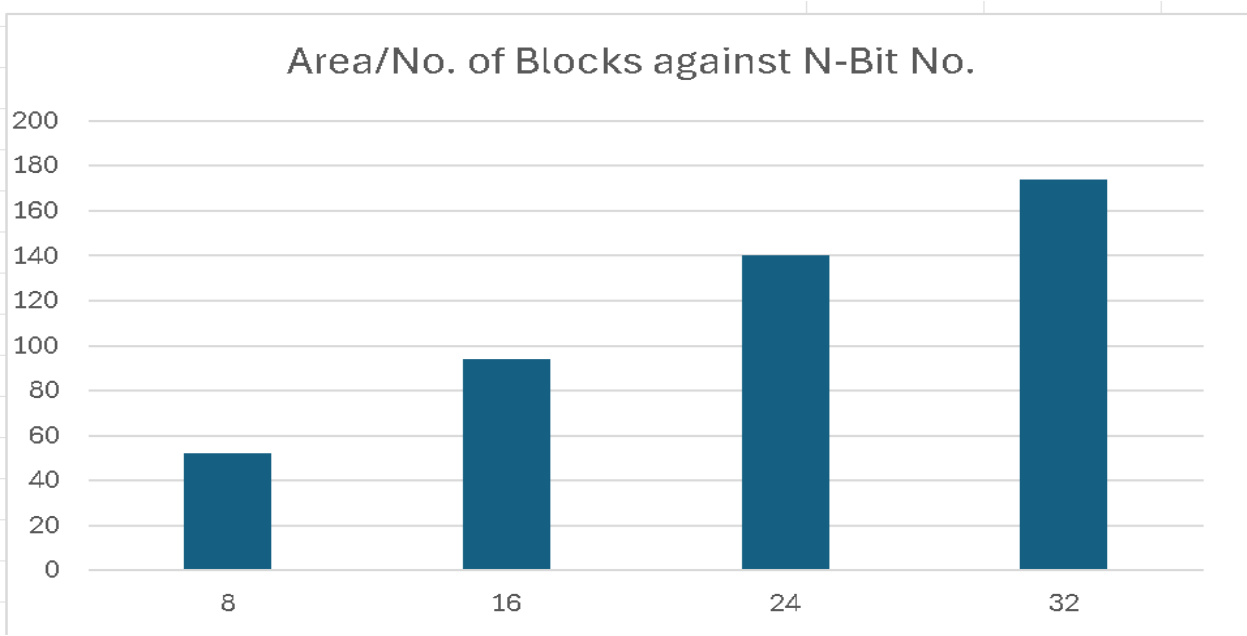


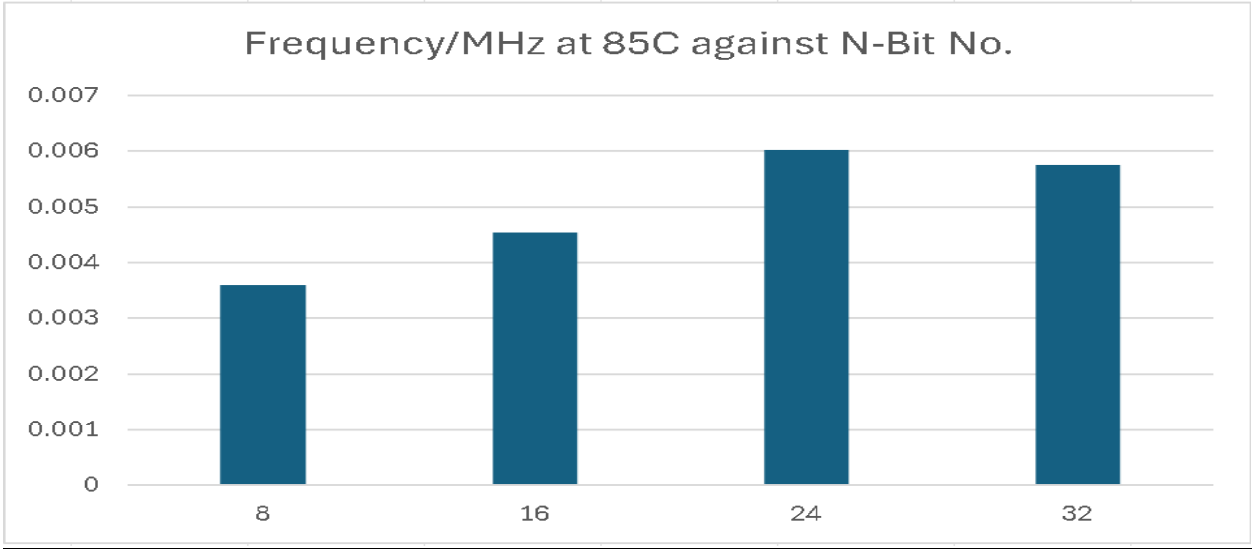
ASM Chart


ASM chart for n-bit divider
S1
RR ← 0, C ← N-1
Start
0
RA ← A
RB ← B
1
S2
RR||RA ← shl(RR||RA)
S3
C ← C - 1
to S1
RR ≥ RB
0
RQ ← shl
RQ || 0
1
RQ ← shl
RQ || 1
RR ← RR-RB
C = 0?
0, to S2
1
S4
Done = 1

Synthesis Report Summary

N-Bit No.	8	16	24	32
Area/No. of Blocks	52	94	140	174
Frequency/MHz at 85C	278.71	220.7	166.22	173.64
Time/μs	0.003588	0.004531	0.006016	0.005759





Theoretically it shows that 32 bit performs faster than 24 bit, but in practical situation 32 bit is expected to perform slower than 24 bit

ELEC8600-1-R-2026W

Reconfigurable Computing

Assignment 3 – Version 2

Digital System for Byte Storage and Processing

Submitted by:

Lawaz Islam

Student ID: 110189519

Kamran Shahid

Student ID: 110184367

University of Windsor

Department of Electrical and Computer Engineering

April 2026

1. Introduction

This report documents the VHDL design, simulation, and synthesis of a digital system for byte storage and processing. The system reads 8 consecutive bytes from an input port, stores them in an internal 8x8-bit memory, computes the difference between each consecutive pair of bytes, streams the results to an output port, and signals completion with a done pulse. The design is modeled from an ASM chart, implemented in VHDL, simulated in Questa, and synthesized using Intel Quartus Prime.

2. System Specification

2.1 I/O Interface

Signal	Direction	Description
datain[7:0]	In	8-bit data input: one byte sampled per clock during S2 load phase
start	In	Active-high: sampled in S1, transitions FSM to S2 when asserted
reset	In	Synchronous active-high: returns FSM to S1 from any state
clk	In	System clock: all state transitions on rising edge
dataout[7:0]	Out	8-bit output: consecutive byte differences driven during S3
done	Out	Active-high for exactly one clock cycle in S4 before returning to S1

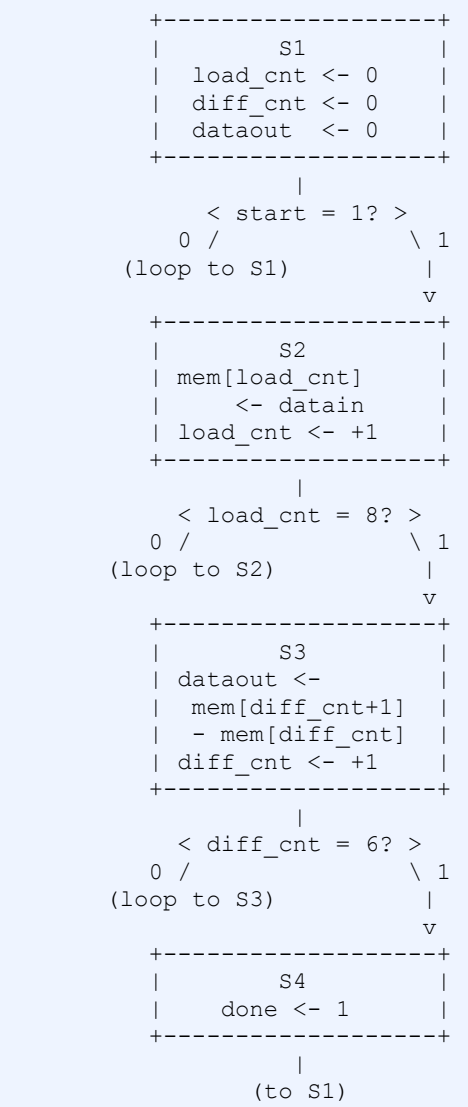
2.2 Operational Sequence

The system operates in four sequential phases. In S1 it idles with counters cleared, waiting for start. In S2 it captures 8 bytes from datain into internal memory, one per clock, until load_cnt reaches 8. In S3 it computes mem[diff_cnt+1] minus mem[diff_cnt] and drives the result onto dataout each cycle until diff_cnt reaches 6. In S4 it asserts done for one cycle and returns to S1.

3. ASM Chart

The ASM chart below models the four-state FSM. Rectangular boxes are states with register operations. Diamonds are decision points with 0 (false) and 1 (true) exits. Synchronous reset overrides all states and forces return to S1.

ASM Chart – byte_processor (Version 2)



Note: reset = 1 forces S1 from any state.

3.1 State Descriptions

S1 (IDLE): Clears load_cnt, diff_cnt, and dataout to 0. Waits for start = 1. Reset in any state forces return here.

S2 (LOAD): Writes datain into mem[load_cnt] and increments load_cnt each clock cycle. Loops until load_cnt reaches 8, then transitions to S3.

S3 (DIFF): Reads mem[diff_cnt+1] and mem[diff_cnt] directly from the internal array and drives their difference onto dataout. Increments diff_cnt each cycle. Loops until diff_cnt reaches 6, covering all 7 pairs. Then transitions to S4.

S4 (DONE): Asserts done = 1 for exactly one clock cycle. Returns to S1 on the next rising edge.

4. VHDL Design

4.1 Architecture

The design is a single self-contained VHDL file with no sub-components. The internal memory is a plain signal array of eight 8-bit std_logic_vector elements, written and read directly within the clocked process. This avoids registered-output pipeline delays that arise with separate memory components.

4.2 Main Design: byte_processor.vhd

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

entity byte_processor is
    port (
        clk     : in  std_logic;
        reset   : in  std_logic;
        start   : in  std_logic;
        datain  : in  std_logic_vector(7 downto 0);
        dataout : out std_logic_vector(7 downto 0);
        done    : out std_logic
    );
end entity byte_processor;

architecture rtl of byte_processor is

    type state_t is (S_IDLE, S_LOAD, S_DIFF, S_DONE);
    signal state : state_t;

    type mem_t is array (0 to 7) of std_logic_vector(7 downto 0);
    signal mem : mem_t;

    signal load_cnt : integer range 0 to 8;
    signal diff_cnt : integer range 0 to 7;

begin

    process (clk)
    begin
        if rising_edge(clk) then
            if reset = '1' then
                state    <= S_IDLE;
                load_cnt <= 0;
                diff_cnt <= 0;
                dataout  <= (others => '0');
                done     <= '0';
            else
                done <= '0';

                case state is

                    when S_IDLE =>
                        load_cnt <= 0;
```

```
                        diff_cnt <= 0;
                        dataout  <= (others => '0');
                        if start = '1' then
                            state <= S_LOAD;
                        end if;

                    when S_LOAD =>
                        mem(load_cnt) <= datain;
                        if load_cnt = 7 then
                            load_cnt <= 0;
                            diff_cnt <= 0;
                            state    <= S_DIFF;
                        else
                            load_cnt <= load_cnt + 1;
                        end if;

                    when S_DIFF =>
                        dataout <= std_logic_vector(
                            unsigned(mem(diff_cnt + 1)) -
                            unsigned(mem(diff_cnt))
                        );
                        if diff_cnt = 6 then
                            state <= S_DONE;
                        else
                            diff_cnt <= diff_cnt + 1;
                        end if;

                    when S_DONE =>
                        done  <= '1';
                        state <= S_IDLE;

                    when others =>
                        state <= S_IDLE;

                end case;
            end if;
        end if;
    end process;

end architecture rtl;
```

5. Testbench

5.1 Test Strategy

The testbench applies a synchronous reset for 5 cycles, then asserts start for one cycle to trigger the load phase. The 8 Version 2 test bytes are driven onto datain one per clock cycle. The simulation runs until done is asserted, confirming all 7 differences have been produced.

5.2 Testbench Code: tb_byte_processor.vhd

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

entity tb_byte_processor is
end entity tb_byte_processor;
```

```
architecture sim of tb_byte_processor is

    component byte_processor
        port (
            clk     : in  std_logic;
            reset   : in  std_logic;
            start   : in  std_logic;
            datain  : in  std_logic_vector(7 downto 0);
            dataout : out std_logic_vector(7 downto 0);
            done    : out std_logic
        );
    end component;

    signal clk_s     : std_logic := '0';
    signal reset_s   : std_logic := '1';
    signal start_s   : std_logic := '0';
    signal datain_s  : std_logic_vector(7 downto 0) := (others => '0');
    signal dataout_s : std_logic_vector(7 downto 0);
    signal done_s    : std_logic;

    constant CLK_PERIOD : time := 20 ns;

begin

    DUT : byte_processor
        port map (
            clk     => clk_s,
            reset   => reset_s,
            start   => start_s,
            datain  => datain_s,
            dataout => dataout_s,
            done    => done_s
        );

    -- clock
    clk_s <= not clk_s after CLK_PERIOD / 2;

    stim : process
    begin
        -- reset for 5 cycles
        reset_s <= '1';
        wait for CLK_PERIOD * 5;
        reset_s <= '0';
        wait for CLK_PERIOD * 2;

        -- assert start
        start_s <= '1';
        wait for CLK_PERIOD;
        start_s <= '0';

        -- feed 8 bytes one per cycle
        datain_s <= x"18"; wait for CLK_PERIOD;
        datain_s <= x"2D"; wait for CLK_PERIOD;
        datain_s <= x"76"; wait for CLK_PERIOD;
        datain_s <= x"84"; wait for CLK_PERIOD;
        datain_s <= x"98"; wait for CLK_PERIOD;
        datain_s <= x"99"; wait for CLK_PERIOD;
        datain_s <= x"A6"; wait for CLK_PERIOD;
        datain_s <= x"F4"; wait for CLK_PERIOD;

        -- wait for done
        wait until done_s = '1';
        wait for CLK_PERIOD * 5;

        report "DONE" severity note;
```

```
        wait;
    end process;

end architecture sim;
```

6. Simulation Results

6.1 Expected vs Actual Output

Cycle	mem[i]	mem[i+1]	dataout (hex)	decimal
0	0x18	0x2D	**0x15**	21
1	0x2D	0x76	**0x49**	73
2	0x76	0x84	**0x0E**	14
3	0x84	0x98	**0x14**	20
4	0x98	0x99	**0x01**	1
5	0x99	0xA6	**0x0D**	13
6	0xA6	0xF4	**0x4E**	78

6.2 Simulation Waveform

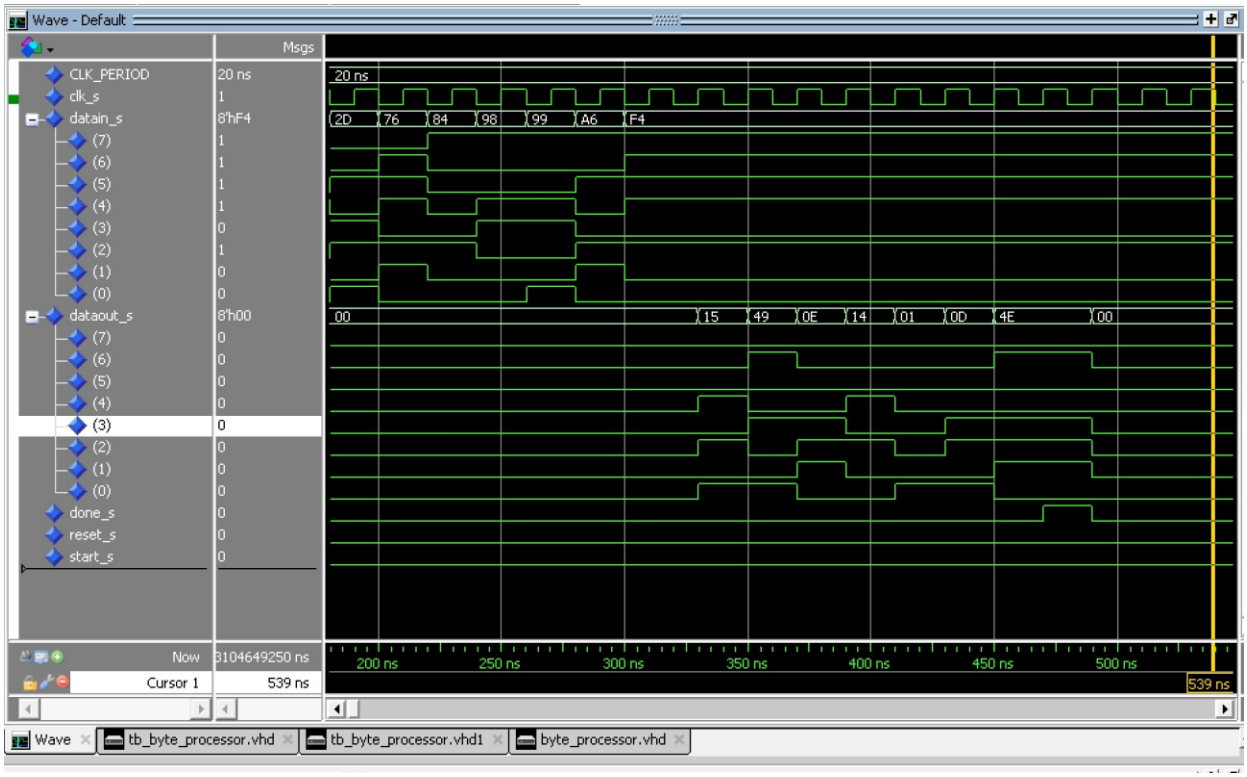


Figure 1: Questa waveform showing datain sequence, dataout values (15, 49, 0E, 14, 01, 0D, 4E) and done pulse

7. FPGA Synthesis Results

Only byte_processor.vhd was included in the Quartus Prime project. The testbench was excluded as it contains non-synthesizable constructs. The target device is a Ultra Max 10 FPGA.

7.1 Resource Usage

Resource	Usage
Estimated total logic elements	**192**
Total combinational functions	128
Total registers	**83**
Dedicated logic registers	83
I/O pins	**20**
Embedded multiplier 9-bit elements	0
Maximum fan-out node	**clk~input**
Maximum fan-out	83
Average fan-out	**2.94**

7.2 Timing

Parameter	Value
Fmax (Slow 1200mV 85C)	**189.83 MHz**
Restricted Fmax	189.83 MHz
Clock name	**clk**
Testbench clock	50 MHz (20 ns period)

The maximum operating frequency of 189.83 MHz is well above the 50 MHz testbench clock, confirming ample timing margin on the target device.

7.3 Synthesis Screenshots

Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimated Total logic elements	192
2		
3	Total combinational functions	128
4	▾ Logic element usage by number of LUT inputs	
1	-- 4 input functions	92
2	-- 3 input functions	28
3	-- <=2 input functions	8
5		
6	▾ Logic elements by mode	
1	-- normal mode	121
2	-- arithmetic mode	7
7		
8	▾ Total registers	83
1	-- Dedicated logic registers	83
2	-- I/O registers	0
9		
10	I/O pins	20
11		
12	Embedded Multiplier 9-bit elements	0
13		
14	Maximum fan-out node	clk~input
15	Maximum fan-out	83
16	Total fan-out	738
17	Average fan-out	2.94

Figure 2: Quartus Prime Analysis and Synthesis Resource Usage Summary

low 1200mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name
1	189.83 MHz	189.83 MHz	clk

Figure 3: Quartus Prime Slow 1200mV 85C Model Fmax Summary

8. Conclusion

The byte_processor design correctly implements the four-state ASM chart in a single self-contained VHDL file. All 7 consecutive byte differences are produced correctly in simulation, matching the expected Version 2 output sequence (15h, 49h, 0Eh, 14h, 01h, 0Dh, 4Eh). The done signal pulses for exactly one clock cycle after the final difference. FPGA synthesis on Ultra Max 10 reports 192 logic elements, 83 registers, and a maximum operating frequency of 189.83 MHz, well above the simulation clock rate.