EE-316 Lab 6 Coversheet

Name: Joshua Dong

EID: jid295

Section: MW 11-12

Problem: 17.D

Attachment Checklist:

- Part 1
  1. State diagram, block diagram and VHDL code.
  2. Printout of output waveform with the given input(s).
- Part 2
  1. VHDL code.
  2. Printout of output waveform with given input sequence.

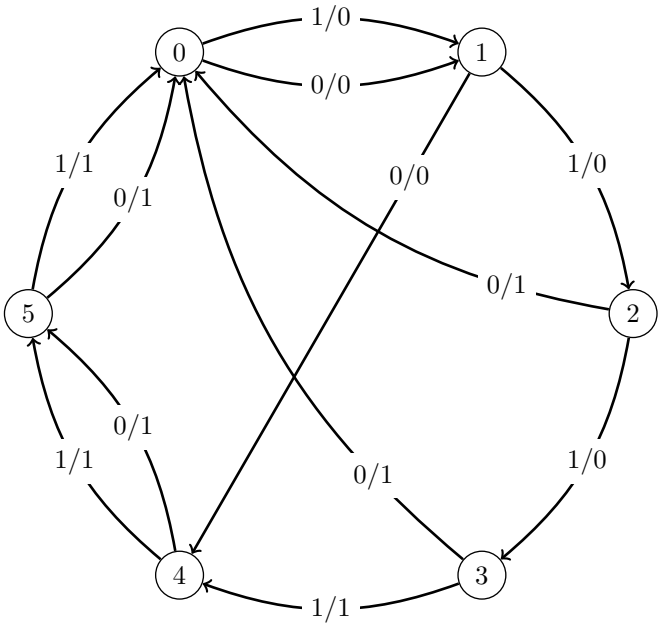
Clearly mark the output sequence and false outputs.

Part 1

Problem Statement

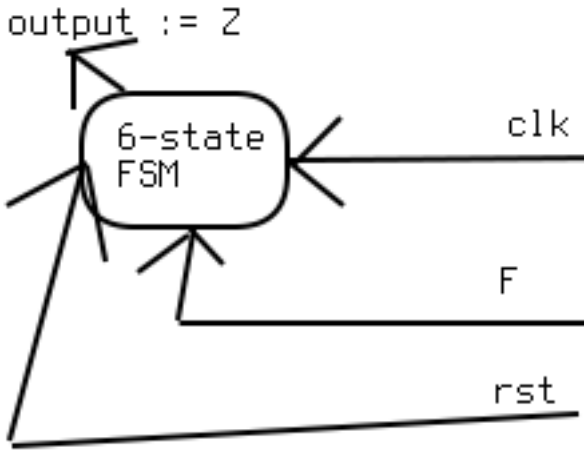
Write a VHDL module to implement a circuit that can generate a clock signal whose time period is a multiple of the input clock. A control signal F determines the multiplying factor. If F = 0, the output signal has a time period twice that of the input clock. If F = 1, the output signal has a time period three times that of the input clock. The portion of the clock cycle when the clock is 1 may be longer than the portion when it is 0, or vice versa. Use a counter with an active-high synchronous clear input.

State Diagram



Block Diagram

The code is simply an implementation of an FSM with 6 states.



Code

```
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
use IEEE.numeric_std.all;


entity SM is
    port(F, clk, rst: in std_logic;
         Z, Q1, Q2, Q3: out std_logic);
end SM;

architecture SM_a of SM is
    -- much cleaner than flip flops, but there is still a
   homomorphism
    signal state, nextstate: integer range 0 to 5 := 0;
    signal state_bits: std_logic_vector(2 downto 0);
begin
    process (state, F)
    begin
        Z <= '1';
        case state is
            when 0 =>
                Z <= '0';
                nextstate <= 1;
            when 1 =>
                Z <= '0';
                -- If F=0, output signal x2
                if (F = '0') then
                    nextstate <= 4;
                -- If F=1, output signal x3
                else
                    nextstate <= 2;
                end if;
            when 2 =>
                Z <= '0';
                -- If F=0, output signal x2
                if (F = '0') then
                    nextstate <= 0; -- reset
                -- If F=1, output signal x3
                else
                    nextstate <= 3;
                end if;
            when 3 =>
                -- If F=0, output signal x2
                if (F = '0') then
                    nextstate <= 0; -- reset
                -- If F=1, output signal x3
                else
                    nextstate <= 4;
                end if;
            when 4 =>
```

```
                nextstate <= 5;
            when 5 =>
                nextstate <= 0;
        end case;
        -- the state homomorphism
        -- just for the lab requirement
        state_bits <= std_logic_vector(to_unsigned(state, 3));
        Q1 <= state_bits(0);
        Q2 <= state_bits(1);
        Q3 <= state_bits(2);
    end process;

    -- use every clock edge (rising and falling).
    -- don't trigger on rst since it's a synchronous input
    process (clk)
    begin
        -- active-high synchronous clear
        if (rst = '1') then
            state <= 0;
        else
            state <= nextstate;
        end if;
    end process;
end SM_a;
```

six.vhdl

Sample Testbench

```
library ieee;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;


--  A testbench has no ports.
entity six_tb is
    end six_tb;

architecture behav of six_tb is
    component SM
        port(F, clk, rst: in std_logic;
        Z, Q1, Q2, Q3: out std_logic);
    end component;

    --  Specifies which entity is bound with the component.
    for six_0: SM use entity work.SM;
        signal F, clk, rst: std_logic;
        signal Z: std_logic;
    begin
    six_0: SM port map (F=>F, clk=>clk, rst=>rst, Z=>Z);

    process
```

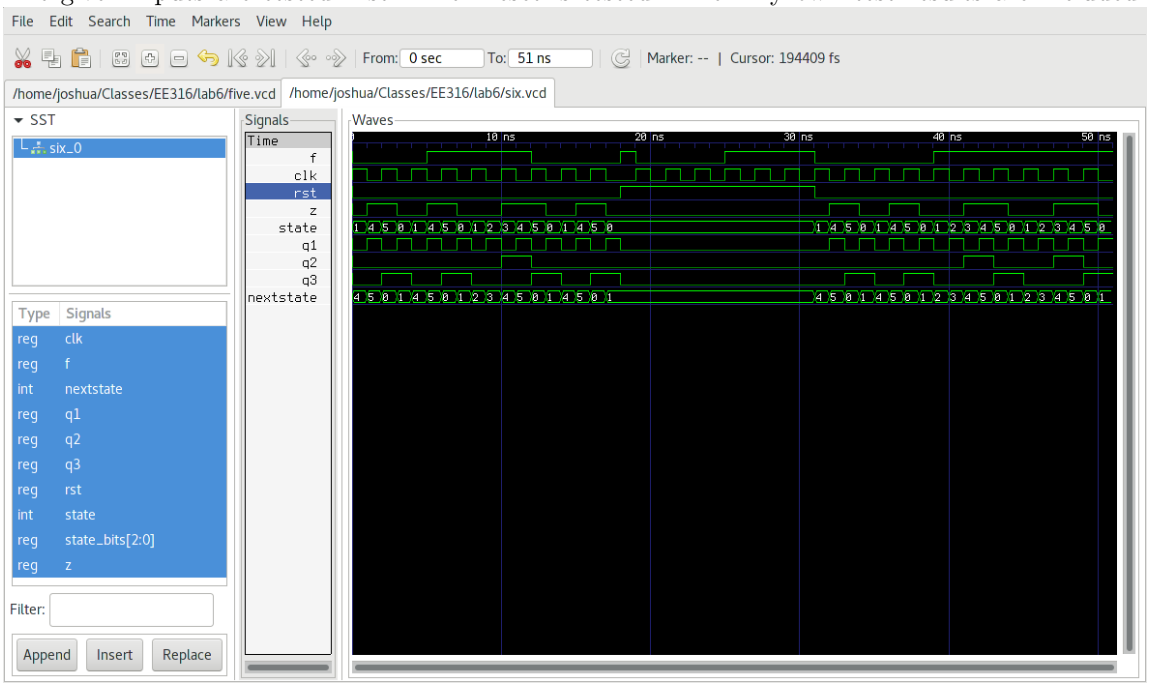
```
type pattern_type is record
    --  inputs
    F, clk, rst : std_logic;
    --  The expected outputs of the adder.
    Z : std_logic;
end record;
--  The patterns to apply.
type pattern_array is array (natural range <>) of pattern_type;
constant patterns : pattern_array :=
-- F, clk, rst, Z
-- provided tests
-- F = (0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0)
(('0', '1', '0', '0'),
('0', '0', '0', '1'),
('0', '1', '0', '1'),
('0', '0', '0', '0'),
('0', '1', '0', '0'),
('1', '0', '0', '1'),
('1', '1', '0', '1'),
('1', '0', '0', '0'),
('1', '1', '0', '0'),
('1', '0', '0', '0'),
('1', '1', '0', '1'),
('1', '0', '0', '1'),
('0', '1', '0', '1'),
('0', '0', '0', '0'),
('0', '1', '0', '0'),
('0', '0', '0', '1'),
('0', '1', '0', '1'),
('0', '0', '0', '0'),
-- my own tests (reset and basic vave)
('1', '0', '1', '0'),
('0', '1', '1', '0'),
('0', '0', '1', '0'),
('0', '1', '1', '0'),
('0', '0', '1', '0'),
('0', '1', '1', '0'),
('0', '0', '1', '0'),
('1', '1', '1', '0'),
('1', '0', '1', '0'),
('1', '1', '1', '0'),
('1', '0', '1', '0'),
('1', '1', '1', '0'),
('1', '0', '1', '0'),

('0', '1', '0', '0'),
('0', '0', '0', '1'),
('0', '1', '0', '1'),
('0', '0', '0', '0'),
('0', '1', '0', '0'),
('0', '0', '0', '1'),
('0', '1', '0', '1'),
('0', '0', '0', '0'),
```

```
    ('1', '1', '0', '0'),
    ('1', '0', '0', '0'),
    ('1', '1', '0', '1'),
    ('1', '0', '0', '1'),
    ('1', '1', '0', '1'),
    ('1', '0', '0', '0'),
    ('1', '1', '0', '0'),
    ('1', '0', '0', '0'),
    ('1', '1', '0', '1'),
    ('1', '0', '0', '1'),
    ('1', '1', '0', '1'),
    ('1', '0', '0', '0'));
begin
    --  Check each pattern.
    for i in patterns'range loop
        --  Set the inputs.
        F <= patterns(i).F;
        clk <= patterns(i).clk;
        rst <= patterns(i).rst;
        --  Wait for the results.
        wait for 1 ns;
        --  Check the outputs.
        assert Z = patterns(i).Z
            report "incorrect output" severity error;
    end loop;
    assert false report "end of test" severity note;
    --  Wait forever; this will finish the simulation.
    wait;
end process;
end behav;
```

six_tb.vhdl

Waveforms

The given inputs are tested first. Then reset is tested. Then my own test results are included.

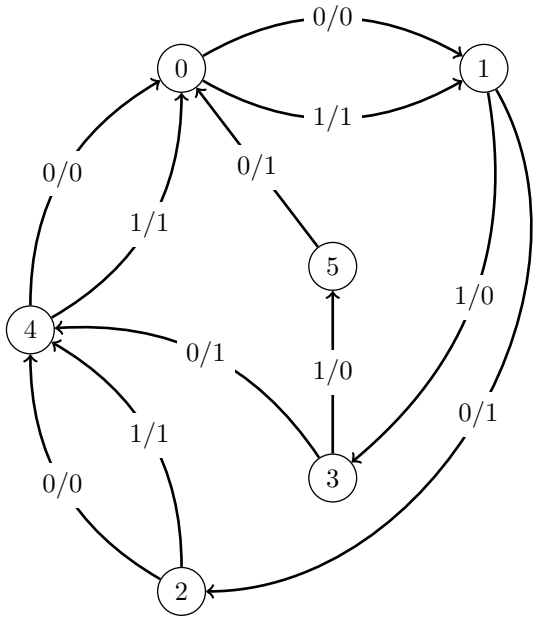


Part 2

Problem Statement

Design a sequential circuit which adds two to a binary number in the range 0000 through 1. The input and output should be serial with the least significant bit first. Find a state table with a minimum number of states. Design the circuit using NAND gates, NOR gates, and three D flip-flops. Any solution which is minimal for your state assignment and uses 10 or fewer gates and inverters is acceptable.

State Graph



VHDL Code

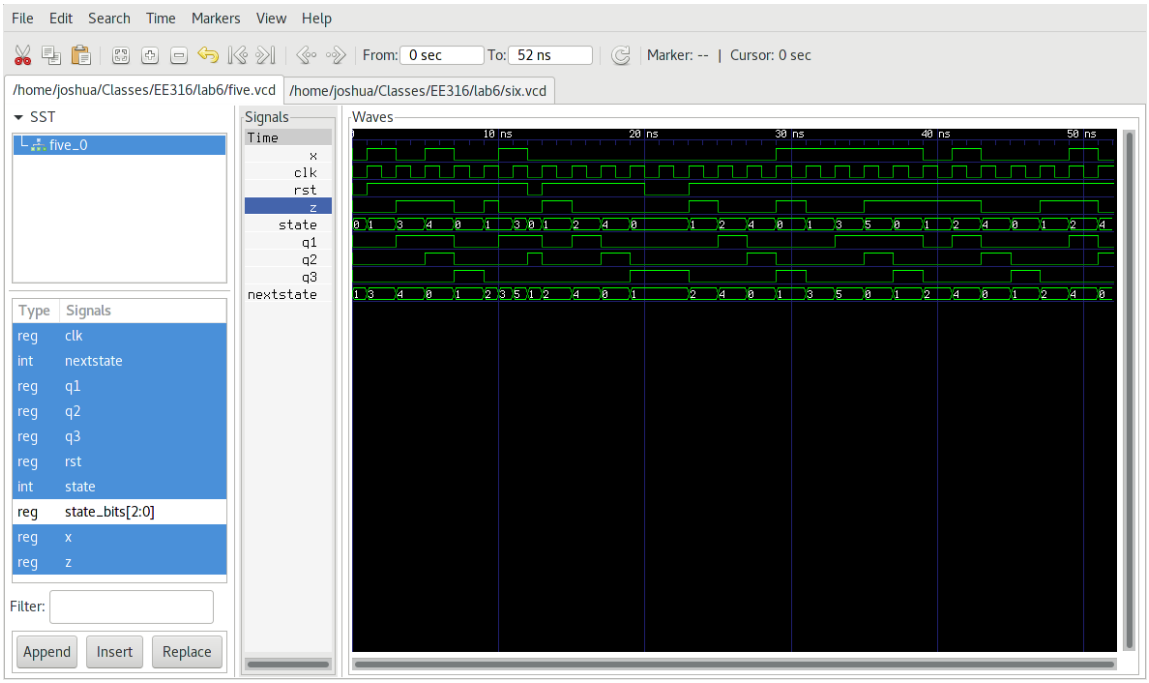
Below is simply an implementation of the state graph:

```
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
use IEEE.numeric_std.all;


entity SM is
    port(rst, clk, X: in std_logic;
         Q1, Q2, Q3, Z: out std_logic);
end SM;

architecture SM_a of SM is
    -- much cleaner than flip flops, but there is still a
   homomorphism
    signal state, nextstate: integer range 0 to 5 := 0;
    signal state_bits: std_logic_vector(2 downto 0);
begin
```

```
    process (state, X)
    begin
        case state is
            when 0 =>
                Z <= X;
                nextstate <= 1;
            when 1 =>
                Z <= not X;
                if (X = '0') then
                    nextstate <= 2;
                else
                    nextstate <= 3;
                end if;
            when 2 =>
                Z <= X;
                nextstate <= 4;
            when 3 =>
                Z <= not X;
                if (X = '0') then
                    nextstate <= 4;
                else
                    nextstate <= 5;
                end if;
            when 4 =>
                Z <= X;
                nextstate <= 0;
            when 5 =>
                Z <= '1';
                nextstate <= 0;
        end case;
        -- the state homomorphism
        state_bits <= conv_std_logic_vector(state,3);
        Q3 <= state_bits(2);
        Q2 <= state_bits(1);
        Q1 <= state_bits(0);
    end process;

    process (clk, rst)
    begin
        -- active-low asynchronous clear
        if (rst = '0') then
            state <= 0;
        -- rising edge trigger
        elsif (clk = '1') then
            state <= nextstate;
        end if;
    end process;
end SM_a;
```

five.vhdl

Waveforms


File Edit Search Time Markers View Help
From: 0 sec
To: 52 ns
Marker: -- | Cursor: 0 sec
/home/joshua/Classes/EE316/lab6/five.vcd
/home/joshua/Classes/EE316/lab6/six.vcd
SST
five_0
Type Signals
reg clk
int nextstate
reg q1
reg q2
reg q3
reg rst
int state
reg state_bits[2:0]
reg x
reg z
Filter:
Append
Insert
Replace
Signals
Time
x
clk
rst
z
state
q1
q2
q3
nextstate
Waves
10 ns
20 ns
30 ns
40 ns
50 ns