CPEG 422/622 Spring 2020

Homework 3

Due March 16th at midnight (through Canvas)
Put your name in the comment part of the code you submitted!

1. Please implement a 24-bit universal shift register with VHDL programming, the register should have clear, parallel load, shift right, shift left, and store function. The entity view is shown below:

```
entity shift_register is
Port {
        clock:  in STD_LOGIC;
        clear:  in STD_LOGIC;
        control:        in STD_LOGIC_VECTOR(1 downto 0);
        serial_in:      in STD_LOGIC;
        parallel_in:    in STD_LOGIC_VECTOR(23 downto 0);
        serial_out:     out STD_LOGIC;
        parallel_out:   out STD_LOGIC_VECTOR(23 downto 0)
};
end shift_register;
```

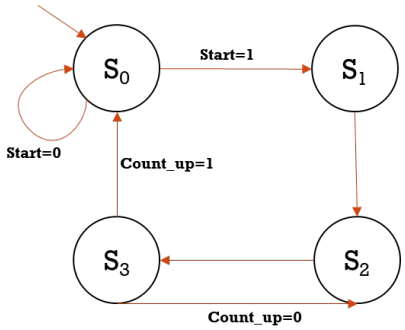
   a) Clear is synchronous reset (clock rising edge sensitive)
   b) Control signals (also synchronized by clock rising edge) defined as:
      00 STORE (hold Serial_out, Parallel_out and register values)
      01 RIGHT (shift right, Serial_in is loaded to the leftmost register, and rightmost bit is sent to Serial_out)
      10 LEFT (shift left, Serial_in is loaded to the rightmost register, and leftmost bit is sent to Serial_out)
      11 PARALLEL (load "Parallel_in" to the register in parallel.)

   Please submit your design source file and testbench.

Grading policy:
No syntax error – 10
Assignment of Q (states) is correctly implemented with case or if statements – 5
Implementation is synchronous, inside a process, triggered by the rising edge of clock, and clock is in the sensitivity list – 5
The four modes are correct – 5 pts each, total 20
Testbench correctly tests the four modes – 10

2. An FSM that controls booth multiplication is shown below, please write a VHDL code to implement this FSM. Submit your source files and testbench.

```
entity booth_control is
-- Port ( );
Port
      (
       clock    : in STD_LOGIC;
       reset:     in STD_LOGIC;
       start    : in STD_LOGIC;
       count_up : in STD_LOGIC;
       init     : out STD_LOGIC;
       loadA    : out STD_LOGIC;
       shift    : out STD_LOGIC;
       done     : out STD_LOGIC
      );
end booth_control;
```



a) reset is synchronous (clock rising edge sensitive)

b) Initial state is S_0, and FSM has four states in total.
S_0: Idle, circuit is ready.
S_1: Init, initialize components.
S_2: LoadA, selectively load result from adder/subtractor to product register.
S_3: Shift, shift product register and multiplier register one bit to the right.

c) State transitions caused by input (synchronized by clock rising edge) are:
S_0: if start=1, $S_0 \rightarrow S_1$; otherwise remains in S_0.
S_1: S_1 always moves to S_2 regardless of the inputs.
S_2: S_2 always moves to S_3 regardless of the inputs.
S_3: if count_up=1, $S_3 \rightarrow S_0$; otherwise if $S_3 \rightarrow S_2$.

d) Outputs at each state are:
S_0: done=1; Init=0; LoadA=0; Shift=0;
S_1: done =0; Init=1; LoadA=0; Shift=0;
S_2: done =0; Init=0; LoadA=1; Shift=0;
S_3: done =0; Init=0; LoadA=0; Shift=1;

Grading policy:
No syntax error – 10
Reset is synchronous – 5
Implementation is synchronous, inside a process, triggered by the rising edge of clock, and clock is in the sensitivity list – 5
Assignment of Q (next state logic) is correctly implemented with case or if statements – 10
Output logic are correctly implemented with case or if statements – 10
Testbench correctly tests FSM functions – 10