Brandon Hoang

EGCP 447

December 6, 2019

Assignment #3: LFSR and BFSM

I. Introduction

The LFSR is intended to generate pseudo-random numbers algorithmically, and a BFSM is an FSM that can mask it’s actual function by adding additional states that can exit to the ‘actual’ states through proper inputs. The LFSR randomizes the initial state of the BFSM whenever it is reset.

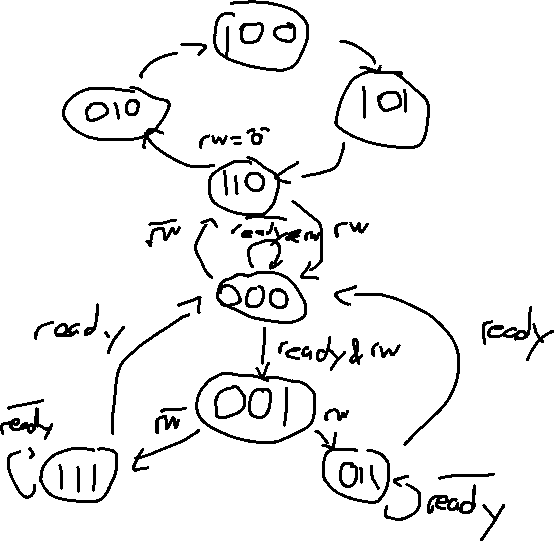
II. Procedure/Discussion

The LFSR consists of 3 flip flops with the reset input swapped with a initialization value and enable input. The LFSR is initialized by setting a init value and toggling the enable to ‘1’. Once the enable is set back to ‘0’, the circuit will cycle through the input’s pseudo-random sequence. The behavior of the Flip flop sequence is dependent on the amount of XOR gates and which D inputs they connect to. In this design, the XOR gate affects the middle bit of the Q output. The circuit works as the flip flops function normally without InitEnable set to ‘1’ and have their Q outputs replaced with InitValue when InitEnable is ‘1’.

The LFSR is a standard (external) LFSR with a characteristic polynomial of f(x) = 1 + x^1 + x^3. An initialization value of “101” was used to test the LFSR. The pseudo-random sequence is 101>001>010>100>011>110>111>101>…

The BFSM includes the original FSM from the very first part plus an additional 4 lock states. The outputs of these new lock states are oe = ‘1’ and we = ‘1’, and they transition to each other unconditionally through a loop. The reset input is also replaced by a similar initialization enable and value input to the LFSR. The circuit functions correctly since it only builds onto the existing FSM design by adding new states and additional case statements.

To unlock the BFSM, rw must be set to ‘1’ when the current state is “110”. The sequence of inputs is irrelevant as long as rw = ‘1’ when state is “110”. For an initial state of “010”, this means the unlock is “XXX1”. To lock, the opposite must occur where rw = ‘0’ and the state must be set to “000”. If a write operation occurs, then the rw must be set back to ‘1’ before the next state logic of state “000” (idle) occurs if the FSM is to stay in the unlocked states. To be honest there is no security related rationale for the lock and unlock conditions. They were set simply to make debugging the system straightforward. The state logic of the not transitional added states can be given logic of their own to complicate the unlocking process. New inputs can also be added to the overall system as well.

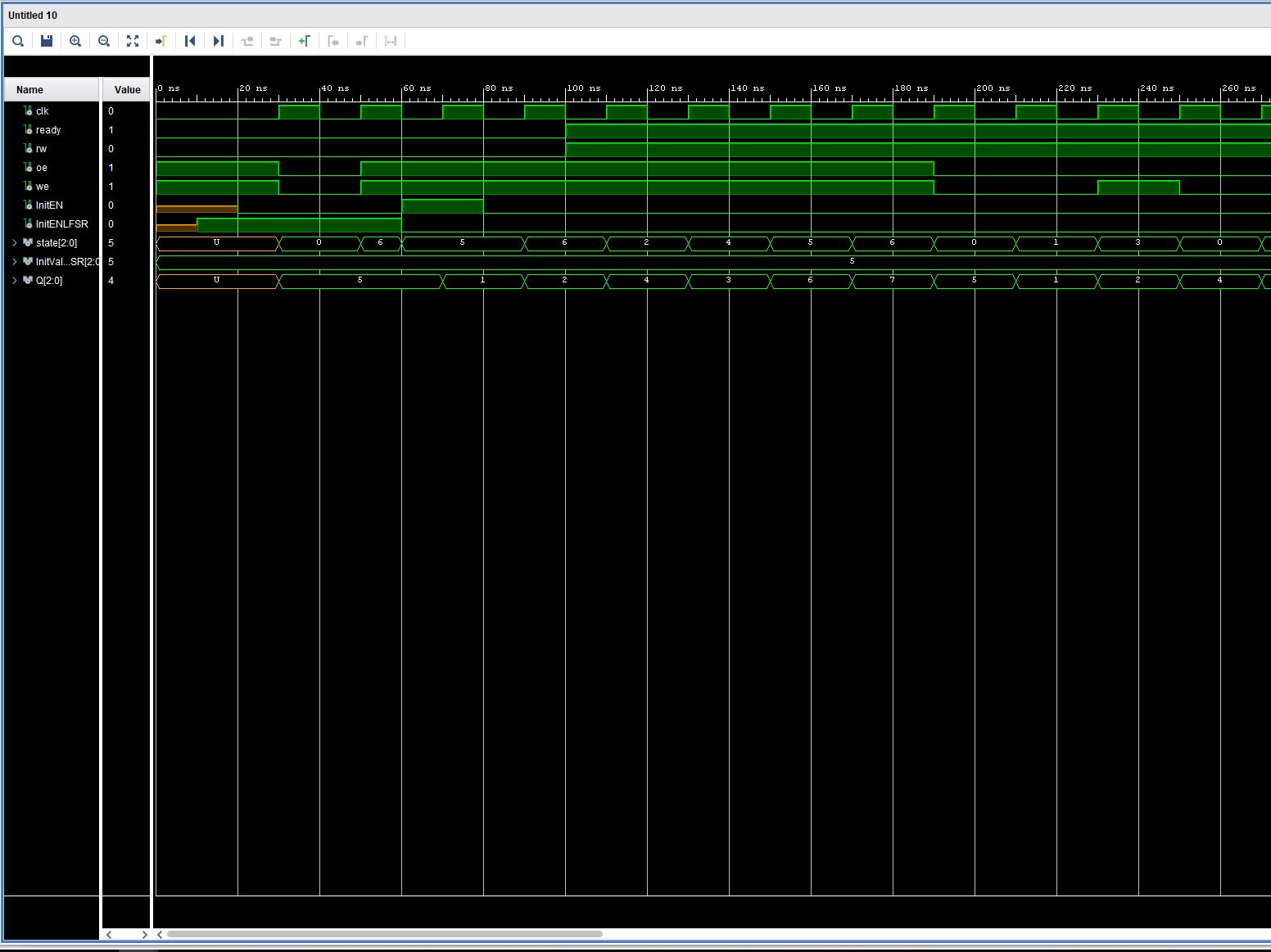


The design of the test bench code is set to test both the process of locking and unlocking. The initialization value of the LFSR is set as “101” and the initialization enable of the BFSM is set to ‘1’ as soon as the LFSR stores the initialization values. As such the BFSM starts in state “101”. Aside from defaulting to “000” before the enable is set to high (an artifact of the original FSM logic at the time of the simulation, that line is removed as of the submission copy), the initialization works as intended. The rw input is set to ‘1’ on the falling edge of the clk during state “110”, but the FSM remains locked despite this. This could be a strange timing phenomenon because of the next state logic as on the next instance of state “110”, the FSM unlocks properly and transitions to the idle state (“000”). The simulation then shows a correct read operation.

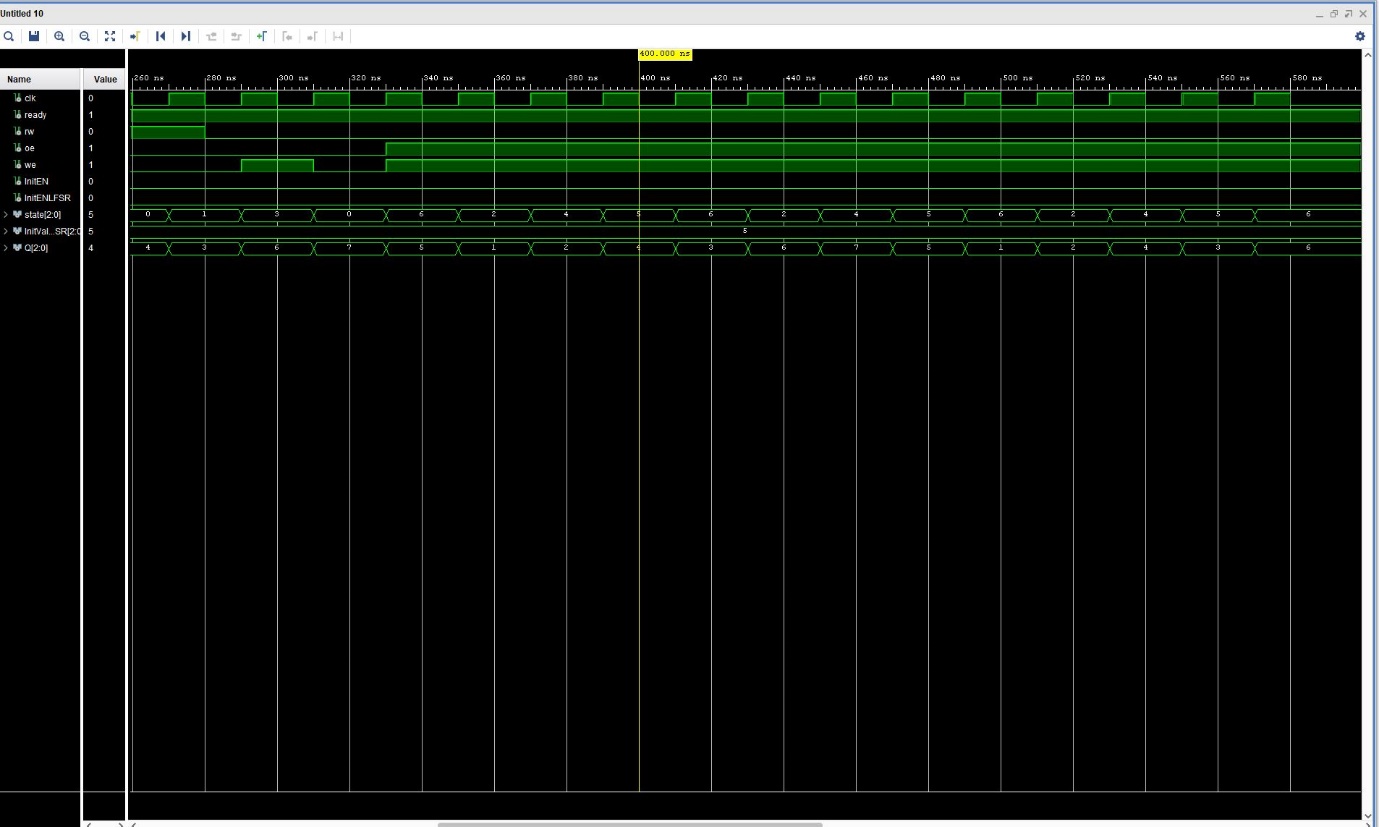
To show the FSM locking itself, rw is set to ‘0’ when the FSM is in the middle of the read operation. Once the state becomes idle, the next state becomes “110” which is the start of the lock state loop. rw is maintained at ‘0’ for 10+ clock cycles and maintains the locked state loop. As such the intended function of the locking is successful.

The test bench both simulates the function of the circuit and connects the LFSR and BFSM together. It is both a top module and test bench in one code file. Both the port maps of the LFSR and BFSM are declared as components and signals for each input/output are assigned. The Q output of the LFSR and the InitValue Input of the BFSM are connected by a common signal.

Unlocking case:



Locking:



III. Conclusion

I learned from this assignment how the LFSR interfaces with the BFSM to obscure the behavior of the actual FSM. The LFSR only determines the initial state of the BFSM if it is ‘reset’. The current design is barebones and is meant to be easy to verify the functionality of, so redoing it would probably mean adding more complex state logic and lock/unlock input combinations. The current design works as intended with barebones functionality so adding additional logic should be simple. A diagram of the LFSR and BFSM connected to each other as an example should probably be included as well, as the amount of bits for the LFSR output needed to be figured out.

Appendix:

LFSR FF:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity LFSRFF is

Port ( clk : in STD\_LOGIC;

InitEN : in STD\_LOGIC;

InitVal : in STD\_LOGIC;

D : in STD\_LOGIC;

Q : out STD\_LOGIC

);

end LFSRFF;

architecture Behavioral of LFSRFF is

begin

process (clk, InitVal)

begin

if (InitEN = '1') then

Q <= InitVal;

elsif (rising\_edge(clk)) then

Q <= D;

end if;

end process;

end Behavioral;

LFSR:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity LFSR is

Port ( clk : in STD\_LOGIC;

InitEN : in STD\_LOGIC;

InitVal : in STD\_LOGIC\_VECTOR(2 downto 0);

Q : out STD\_LOGIC\_VECTOR(2 downto 0)

);

end LFSR;

architecture Behavioral of LFSR is

component LFSRFF is

Port ( clk : in STD\_LOGIC;

InitEN : in STD\_LOGIC;

InitVal : in STD\_LOGIC;

D : in STD\_LOGIC;

Q : out STD\_LOGIC

);

end component;

signal q1, q2, q3, xor1 : std\_logic;

begin

FF0: LFSRFF port map (clk, InitEN, InitVal(0), q3, q1 );

xor1 <= q1 xor q3;

FF1: LFSRFF port map (clk, InitEN, InitVal(1), xor1, q2);

FF2: LFSRFF port map (clk, InitEN, InitVal(2), q2, q3);

Q(0) <= q1;

Q(1) <= q2;

Q(2) <= q3;

end Behavioral;

BFSM:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity BFSM is

Port ( clk : in STD\_LOGIC;

ready : in STD\_LOGIC;

rw : in STD\_LOGIC;

InitEN: in STD\_LOGIC;

InitVal: in STD\_LOGIC\_VECTOR (2 downto 0);

state: out STD\_LOGIC\_VECTOR (2 downto 0);

oe : out STD\_LOGIC;

we : out STD\_LOGIC);

end BFSM;

architecture Behavioral of BFSM is

signal state\_reg, state\_next: std\_logic\_vector (2 downto 0);

-- 000 <= Idle

-- 001 <= Decision

-- 011 <= read

-- 111 <= Write

begin

process (clk, InitEN)

begin

if (InitEN = '1') then

state\_reg <= InitVal;

elsif (rising\_edge(clk)) then

state\_reg <= state\_next;

end if;

end process;

process (state\_reg)

begin

case state\_reg is

when "000" =>

if ( ready = '1' and rw = '1' ) then

state\_next <= "001";

elsif (ready = '0' and rw = '1') then

state\_next <= "000";

else

state\_next <= "110";

end if;

when "001" =>

if ( rw = '1' ) then

state\_next <= "011";

else

state\_next <= "111";

end if;

when "011" =>

if (ready = '1') then

state\_next <= "000";

else

state\_next <= "011";

end if;

when "111" =>

if ( ready = '1') then

state\_next <= "000";

else

state\_next <= "111";

end if;

when "010" =>

state\_next <= "100";

when "100" =>

state\_next <= "101";

when "101" =>

state\_next <= "110";

when "110" =>

if (rw = '1') then

state\_next <= "000";

else

state\_next <= "010";

end if;

end case;

end process;

process (state\_reg)

begin

case state\_reg is

when "000" =>

oe <= '0';

we <= '0';

when "001" =>

oe <= '0';

we <= '0';

when "011" =>

oe <= '0';

we <= '1';

when "111" =>

oe <= '1';

we <= '0';

when others =>

oe <= '1';

we <= '1';

end case;

end process;

state <= state\_reg;

end Behavioral;

Top (test bench file):

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 12/06/2019 01:38:22 PM

-- Design Name:

-- Module Name: top - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity top is

-- Port ( );

end top;

architecture Behavioral of top is

component LFSR is

Port ( clk : in STD\_LOGIC;

InitEN : in STD\_LOGIC;

InitVal : in STD\_LOGIC\_VECTOR(2 downto 0);

Q : out STD\_LOGIC\_VECTOR(2 downto 0)

);

end component;

component BFSM is

Port ( clk : in STD\_LOGIC;

ready : in STD\_LOGIC;

rw : in STD\_LOGIC;

InitEN: in STD\_LOGIC;

InitVal: in STD\_LOGIC\_VECTOR (2 downto 0);

state: out STD\_LOGIC\_VECTOR (2 downto 0);

oe : out STD\_LOGIC;

we : out STD\_LOGIC);

end component;

signal clk, ready, rw, oe, we, InitEN, InitENLFSR : std\_logic;

signal state, InitValLFSR, Q : std\_logic\_vector(2 downto 0);

begin

uut1 : LFSR PORT MAP (

clk => clk,

InitEN => InitENLFSR,

InitVal => InitValLFSR,

Q => Q

);

uut2 : BFSM PORT MAP (

clk => clk,

ready => ready,

rw => rw,

InitEN => InitEN,

InitVal => Q,

state => state,

oe => oe,

we => we

);

tb : PROCESS

BEGIN

clk <= '0';

InitValLFSR <= "101";

ready <= '0';

rw <= '0';

wait for 10ns;

InitENLFSR <= '1';

wait for 10ns;

InitEN <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

InitENLFSR <= '0';

InitEN <= '1';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

InitEN <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

ready <= '1';

rw <= '1';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

rw <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

clk <= '0';

wait;

END PROCESS;

end Behavioral;

FSM:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity FSM is

Port ( clk : in STD\_LOGIC;

rst : in STD\_LOGIC;

ready : in STD\_LOGIC;

rw : in STD\_LOGIC;

oe : out STD\_LOGIC;

we : out STD\_LOGIC);

end FSM;

architecture Behavioral of FSM is

signal state\_reg, state\_next: std\_logic\_vector (2 downto 0);

-- 000 <= Idle

-- 001 <= Decision

-- 011 <= read

-- 111 <= Write

begin

process (clk, rst)

begin

if (rst = '1') then

state\_reg <= "000";

elsif (rising\_edge(clk)) then

state\_reg <= state\_next;

end if;

end process;

process (state\_reg)

begin

case state\_reg is

when "000" =>

if ( ready = '1' ) then

state\_next <= "001";

else

state\_next <= "000";

end if;

when "001" =>

if ( rw = '1' ) then

state\_next <= "011";

else

state\_next <= "111";

end if;

when "011" =>

if (ready = '1') then

state\_next <= "000";

else

state\_next <= "011";

end if;

when "111" =>

if ( ready = '1') then

state\_next <= "000";

else

state\_next <= "111";

end if;

when others =>

state\_next <= "000";

end case;

end process;

process (state\_reg)

begin

case state\_reg is

when "000" =>

oe <= '0';

we <= '0';

when "001" =>

oe <= '0';

we <= '0';

when "011" =>

oe <= '0';

we <= '1';

when "111" =>

oe <= '1';

we <= '0';

end case;

end process;

end Behavioral;