Random Number Generator

PURPOSE – In this lab you will design a Random Number Generator.

1. Introduction
In many real applications, such as electronic measurements (signature analyzers), communications, random (or better said pseudo random) number generators are needed. This laboratory is dedicated to designing a random number generator whose block diagram is shown in the figure below:

This circuit is known as an autonomous Linear Feedback Shift Register (LFSR). In particular, the circuit presented here is called type 1 LSFR. There are also other types of LSFR’s but they are out of the scope of this lab. It has to be mentioned that this circuit is a pseudo random number generator due to its discrete nature.

The vector \( c = \{c_1,c_2,\ldots,c_n\} \) is called the seed of the random number generator. It is fixed and should remain constant over time. It may be provided to the circuit as an input, which does not change over time. \( c_n \) has to be always '1'. The \( c_i \) bits are binary constants such that \( c_i = '1' \) implies that a connection exists, while \( c_i = '0' \) implies that no connection exists. Not any binary combination for the vector \( c \) is a good one. Usually, the vector \( c \) is taken from existing tables for any particular value of \( n \). An encircled '+' symbolizes a two input XOR gate. The vector \( q=\{q_1,q_2,\ldots,q_n\} \) is the random number. These is also a one bit start input which turns on all the FF’s and starts the cycle of generating numbers. This should be implemented in the FF module, similar to a reset input, except it sets the FF’s to 1 instead of 0. For simplicity, you merely hold down start (the pushbutton) to continue the random number generation. That is, if start = 0, \( \{Q_1,Q_2,\ldots,Q_n\} = \{1,1,\ldots,1\} \), and if start = 1, then the FF’s operate normally. Verilog code for the D FF is provided below:
//start of D FF module
module dff ( D, Q, clk, start );

input D, clk, start;
output Q;
reg Q;

always @ (posedge clk)
begin
    if ((start == 1'b0))
        Q <= 1'b1;
    else if ((clk == 1'b1))
        Q <= D;
end
endmodule
//end of D FF module

Important Note: The state q={0,0,…,0} is called a locked state because entering it will cause the entire system to get stuck at q={0,0,…,0}, so this state is never entered under normal operation, which is why ‘start = 0’ sets the FF outputs to 1 and not 0.

2. Writing the Verilog description
You will design a random number generator that can generate different 8-bit (n = 8) number sequences based on a set of control inputs (ctrl). Each bit of ctrl decides if a XOR gate is used or not (if it is not, that connection should be an open circuit).

3. Simulation
Simulate your design for various values of ctrl and see which ctrl combinations produce the longest unique sequence before it repeats. Once you have done this, show your TA the output simulation verifying its working operation, and they will give you the ctrl combination which produces the maximum sequence length (all 255 possible outputs, in exactly 255 clock cycles).

4. Implementation and verification
You will have to write seven segment display code and a .ucf file to display the random number on two digits of the seven segment display. For simplicity, we will break the random number q={q1,q2,q3,q4,q5,q6,q7,q8} into two four bit numbers, n1={q8,q7,q6,q5} and n1={q4,q3,q2,q1}. Your display code will have to display these two four bit numbers n1 and n2 in hexadecimal on two of the seven segment digits simultaneously (so you need to have multiplexing code in your display module). As the clk input you will use the clk signal available via P54, but this should be divided, via the clock_divider from Lab 5, before passing it into the clk input of all the FF’s so that they do not count at an excessive speed. As the initial start input you will assign push button P69.
For the input ctrl (vector of 8 bits) you will use the eight DIP-switches available. You should LOC ctrl(1) (which controls c1) to P2, ctrl(2) (which controls c2) to P3 etc. Synthesize, implement and download your random number generator to the board and show to your TA its functioning for full credit.

**Important Note:** It is recommended that you use one of the seeds that you found from part 3 which has a short repeat cycle so that you can verify the working operation on the board, before using optimal seed.

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

**SUMMARY** -- In this lab you designed and verified an 8-bit random number generator.

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