PSEUDO RANDOM NUMBER GENERATOR USING LINEAR FEEDBACK SHIFT REGISTER

Adithya Hosapate,Havish,Sai Ashish,Dhanush

April 26, 2017

I. Introduction

- A linear-feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state.
- The most commonly used linear function of single bits is exclusive-or (XOR). Thus, an LFSR is a shift register whose input bit is driven by the XOR of some bits of the overall shift register value.

II. WORKING

i. LSFR as a pseudo random number generator

- The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the stream of values produced by the register is completely determined by its current (or previous) state.
- Likewise, because the register has a finite number of possible states, it must eventually enter a repeating cycle. However, an LFSR with a well-chosen feedback function can produce a sequence of bits that appears random and has a very long cycle (2^n-1).

ii. Feedback Mechanism

- The previous states on which the input depends are known as taps.
- The taps are XOR' ed sequentially with the output bit and then fed back into the leftmost bit.

iii. Output Properties

- The sequence of bits in the rightmost position is called the output stream
- Ones and zeroes occur in "runs". The output stream 1110010, for example, consists of four runs of lengths 3, 2, 1, 1, in order.
- In one period of a maximal LFSR, 2^n-1 runs occur.
- Exactly half of these runs are one bit long, a quarter are two bits long, up to a single run of zeroes n - 1 bits long, and a single run of ones n bits long. This distribution almost equals the statistical expectation value for a truly random sequence.
- LFSR output streams are deterministic. If the present state and the positions of the XOR gates in the LFSR are known, the next state can be predicted. This is not possible with truly random events.
- With maximal-length LFSRs, it is much easier to compute the next state, as there are only an easily limited number of them for each length.
- The output stream is reversible; an LFSR with mirrored taps will cycle through the output sequence in reverse order.
- The value consisting of all zeros cannot appear. Thus an LFSR of length n cannot be used to generate all 2^n values.

III. CODE

i. Structural Implementation

```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
port(d, clk: in  std_logic;
q   : out std_logic);
end dff;

architecture behav of dff is
begin
process
begin
wait until rising_edge(clk);
q <= d;
end process;
end behav;
library ieee;
use ieee.std_logic_1164.all;


entity shift_reg is
port( d,clk: in  std_logic;
seed : out std_logic;
Q_1,Q_2,Q_3,Q_4,
Q_5,Q_6,Q_7,Q_8: out std_logic);
end shift_reg;

architecture behav of shift_reg is

signal q1,q2,q3,q4
,q5,q6,q7,q8: std_logic;
component dff is
port(d, clk: in  std_logic;
q: out std_logic);
end component;

begin
m1:dff port map(d=>d, clk=>clk, q=>q1);
m2:dff port map(d=>q1, clk=>clk, q=>q2);
m3:dff port map(d=>q2, clk=>clk, q=>q3);
m4:dff port map(d=>q3, clk=>clk, q=>q4);
m5:dff port map(d=>q4, clk=>clk, q=>q5);
m6:dff port map(d=>q5, clk=>clk, q=>q6);
m7:dff port map(d=>q6, clk=>clk, q=>q7);
m8:dff port map(d=>q7, clk=>clk, q=>q8);
Q_1<=q1;
Q_2<=q2;
Q_3<=q3;
Q_4<=q4;
Q_5<=q5;
Q_6<=q6;
Q_7<=q7;
Q_8<=q8;
seed<=Q4 xor Q5 XOR Q6 XOR Q8;
end behav;
```

ii. **Testbench**

```
library ieee;
use ieee.std_logic_1164.all;

entity shr_tb is
end shr_tb;

architecture TB of shr_tb is

component shift_reg is
port(d, clk: in  std_logic;
seed : out std_logic;
Q_1,Q_2,
Q_3,Q_4,
Q_5,Q_6,
Q_7,Q_8: out std_logic);
end component;

signal d,seed,
clk,Q_1,Q_2,Q_3,Q_4,
Q_5,Q_6,Q_7,Q_8:std_logic;

begin
mapping: shift_reg port map(d,
clk,seed,
Q_1,Q_2,
Q_3,Q_4,
Q_5,Q_6,
Q_7,Q_8);
clk_GEN: process
begin
clk <= '0';
wait for 5 ns;
clk <= '1';
wait for 5 ns;
end process;

process
variable c :integer:=0;
begin
if (c=0) then
d <= '1';
wait for 100 ns;
c:=1;
elsif (seed='0') then
d<='0';
wait for 1 ps;
elsif (seed='1') then
d<= '1';
wait for 1 ps;
end if;
end process;
end TB;
```

iii. Console Commands

```
ghdl -a Structural.vhdl Structural_tb.vhdl
ghdl -r shr_tb --vcd=structure.vcd
```

iv. **Behavioral Implementation**

```
library ieee;
use ieee.std_logic_1164.all;

entity sr is
port(clk: in  std_logic;
q0,q4,q5,q6,q7  : out std_logic;
q1   : out std_logic;
q2   : out std_logic;
q3   : out std_logic);

end sr;

architecture behav of sr is

begin

process(clk)
variable s :
std_logic_vector(7 downto 0)
:="11111111";
variable d1 : std_logic;
begin
d1 := s(0) xor s(2)
xor s(3) xor s(4) ;
if(rising_edge (clk)) then
s(0) := s(1);
s(1) := s(2);
s(2) := s(3);
s(3) := s(4);
s(4) := s(5);
s(5) := s(6);
s(6) := s(7);
s(7) := d1;
end if;
q7 <= s(0);
q6 <= s(1);
q5 <= s(2);
q4 <= s(3);
q3 <= s(4);
q2 <= s(5);
q1 <= s(6);
q0 <= s(7);
end process;
end behav;
```

v. **Testbench**

```
library ieee;
use ieee.std_logic_1164.all;

entity lfsr_tb is
end lfsr_tb;

architecture tb of lfsr_tb is

component sr is
port(clk: in  std_ulogic;
q0   : out std_logic;
q1   : out std_logic;
q2   : out std_logic;
q3   : out std_logic;
q4   : out std_logic;
q5   : out std_logic;
q6   : out std_logic;
q7   : out std_logic);
end component;

signal clk: std_ulogic;
signal q0,q1,q2,q3,
q4,q5,q6,q7: std_ulogic;

begin
mapping: sr port map(clk, q0,q1,q2,
q3,q4,q5,q6,q7);
process
begin
clk <= '1';
wait for 50 ns;
clk <= '0';
wait for 50 ns;

end process;

end tb;
```

vi. Console Commands

```
ghdl -a behavioral.vhdl behavioral_tb.vhdl
ghdl -r lfsr_tb --vcd=behavior.vcd
```

IV. Mathematical Analysis

i. Characteristic polynomial

- The arrangement of taps for feedback in an LFSR can be expressed as a polynomial mod 2, that is the coefficients of the polynomial must be 1s or 0s.
- This is called the feedback polynomial or reciprocal characteristic polynomial
- The "one" in the polynomial does not correspond to a tap. It corresponds to the input to the first bit (i.e. x^0).
- The powers of the terms represent the tapped bits, counting from the left. The first and last bits are always connected as an input and output tap respectively
- For example, if the taps are at the 8th, 5th, 6th and 4th bits (as shown), the feedback polynomial is
- The characteristic polynomial is $x^8+x^6+x^5+x^4+1$.

ii. Primitive polynomial

For a LFSR to give random sequence of maximum length, the characteristic polynomial must be primitive (irreducible). Conditions for primitive

- No. of taps should be even.
- There must be no common divisor for all taps taken together except 1.
- If the common divisor for all taps taken together is other than 1, then we will not get pseudo random numbers.

V. Periodic Nature of Pseudo Random numbers

- Pseudo Random numbers repeat after certain number of cycles(for example, for 8 bit, the numbers repeat after 255 cycles).
- So, sufficiently long LFSR generates pseudo random numbers in which the time period is too big to be considered.
- But, perfect random numbers never repeat. They are like the occurrence of decimal places in π.

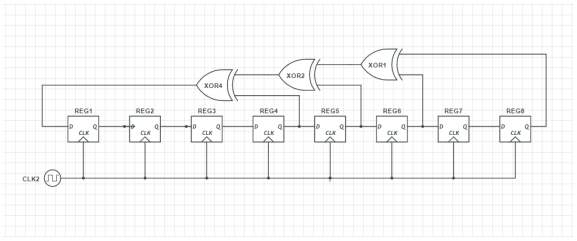

Figure 1: *circuit diagram*

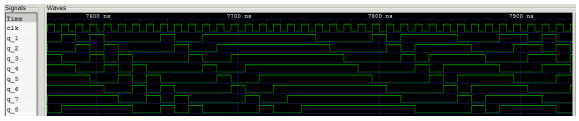

Figure 2: *gtkwave simulation*

VI. Uses of pseudo random numbers

- Sufficiently long pseudo random numbers are used extensively in cryptography.
- They are also useful for data encryption because of their nearly-random nature.
- They are used in scrambling where short runs of 0's and 1's are combined with the output of lfsr and a decoder unscrambles the scrambled signal(useful in digital communications).
- Outputs of LFSR's are used to generate test patterns because they cover all possible states except all zero state.

VII. Tests

- By considering two different seeds which generate different set of random numbers at the same time.
- By considering the number at one instant of time and 255 cycles after that instant of time. If they are same, then the circuit is working properly.

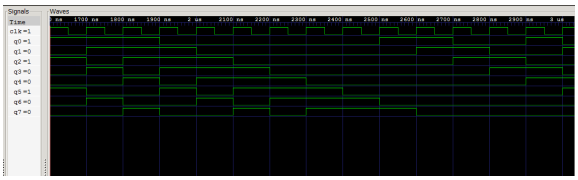

Figure 3: *simulation at 1600 ns for seed=00000001*

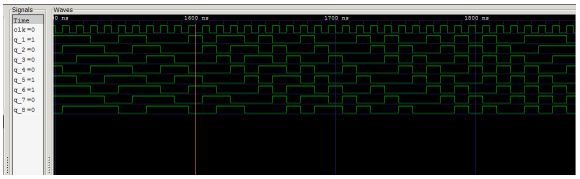

Figure 4: *simulation at 1600 ns for seed=11111111*

- Clearly for different seeds, different bits are generated at the same time.

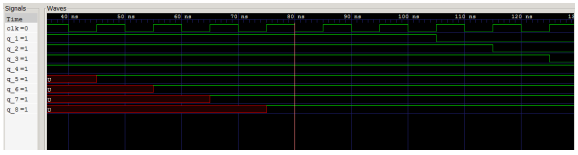

Figure 5: *simulation at 80 ns*

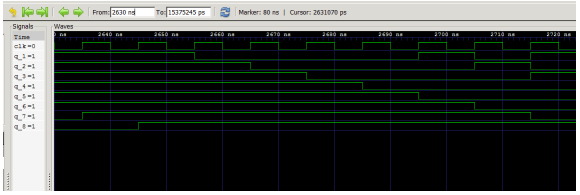

Figure 6: *simulation at 2630 ns*

- After 255(2550 ns) cycles, the outputs generated are the same.