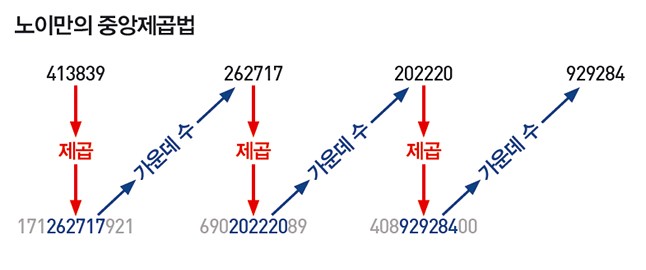
Linear Feedback Shift Register를 이용한 의사난수(pseudo random number)생성회로 구성, 그리고 overflow와 합동을 이용한 LFSR의 한계 극복

스마트시스템소프트웨어학과

20150318 오인규

서론

난수생성 알고리즘은 1949년 폰 노이만에 의해 처음으로 공식 발표되었다. ‘중앙제곱법(Mid-Square method)’이라는 이름의 이 알고리즘은 컴퓨터를 이용해 난수를 만드는 첫 번째 방법이었다. 시드값을 제곱한 뒤, 나온 수의 가운데 일부분을 제곱해가며 난수를 만드는 폰 노이만의 중앙제곱법은 경우에 따라서는 같은 수가 계속 반복될 수도 있고, 가운데 수가 0이 되는경우 더 이상 난수를 만들어내지 못한다는 단점과, 시드가 n자리 수 일 때 난수의 주기가 8^n 보다 항상 짧거나 어떤 숫자로 수렴한다는 단점은 다른 난수생성 알고리즘의 등장을 야기했다.



중앙제곱법 이후에 선형합동법, 메르센 트위스터 등의 난수생성 알고리즘이 등장하였으나, 이 보고서에서는 다루지 않겠다.

이번 보고서에서 다룰 Linear Feedback Shift Register은 회로 설계에 적합하며, 하드웨어 기술언어로 작성하기에도 적합하다고 판단되어 수 많은 난수생성법 중에 택하게 되었다.

Linear Feedback Shift Regist(이하 LFSR)은 XOR연산과 Shift연산을 이용하여 난수를 생성하는 방법이다.

LFSR에 의하여 생성된 난수는 이전 상태값과 LFSR다항식(이 보고서에서는 XOR을 사용함)에 의존적인 성질이 있다. 이 성질에 기인하여 현재 상태값과 LFSR다항식을 알고있는 상태라면 다음 난수를 예측할 수 있다는 단점이 존재하지만 이는 모든 의사난수 생성알고리즘이 갖고있는 문제점이기도 하다.

동작원리

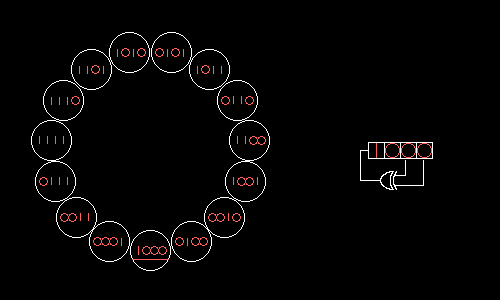
4비트 난수를 생성을 통해서 LFSR의 동작원리를 설명하고자 한다.

SEED값을 0xA, LFSR다항식을 list[3] ^ list[4] 라고 설정하겠다.

설명의 편의를 위해 index를 leftmost를 1로 rightmost를 4로 읽겠다.

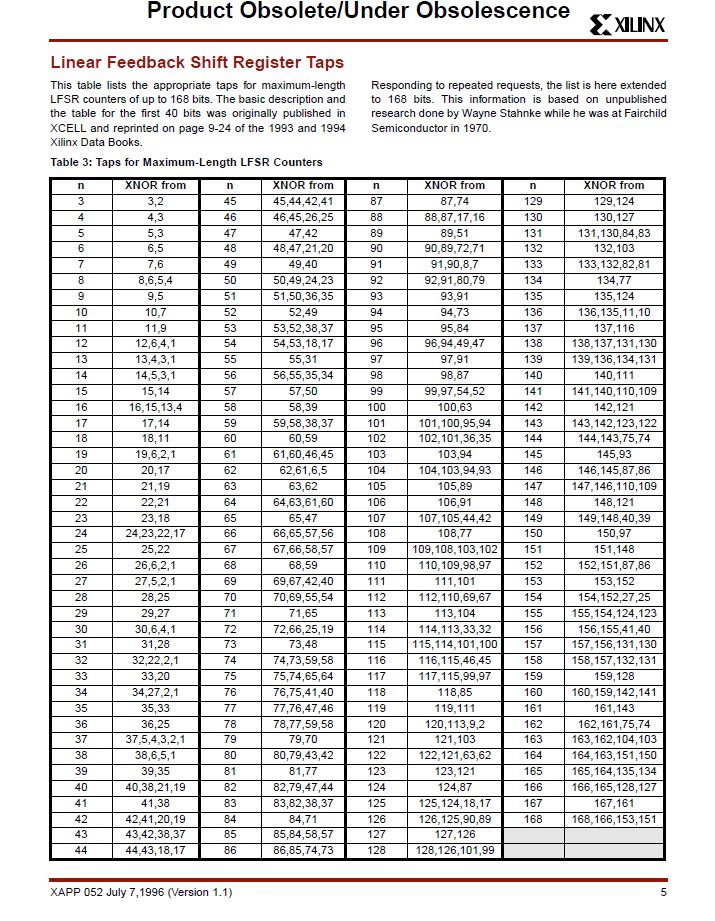
1. list = 0b1010인 상태이다
2. list[3] 와 list[4], 즉 1과 0 의 배타적 논리합을 통해 조각을 만든다. 1 ^ 0 은 1 이다 조각은 1이 된다.
3. 과정 2를 통해서 얻은 조각을 list[1]에 넣는다. 이때 list[1]은 list[2]로 list[2]는 list[3]으로, list[3]은 list[4]로 밀려간다.(Shift)
4. 0b1010이 들어있던 list에는 이제 0b1101이 들어있다.

아래의 사진은 위의 과정을 사이클로 만든 그림이다.

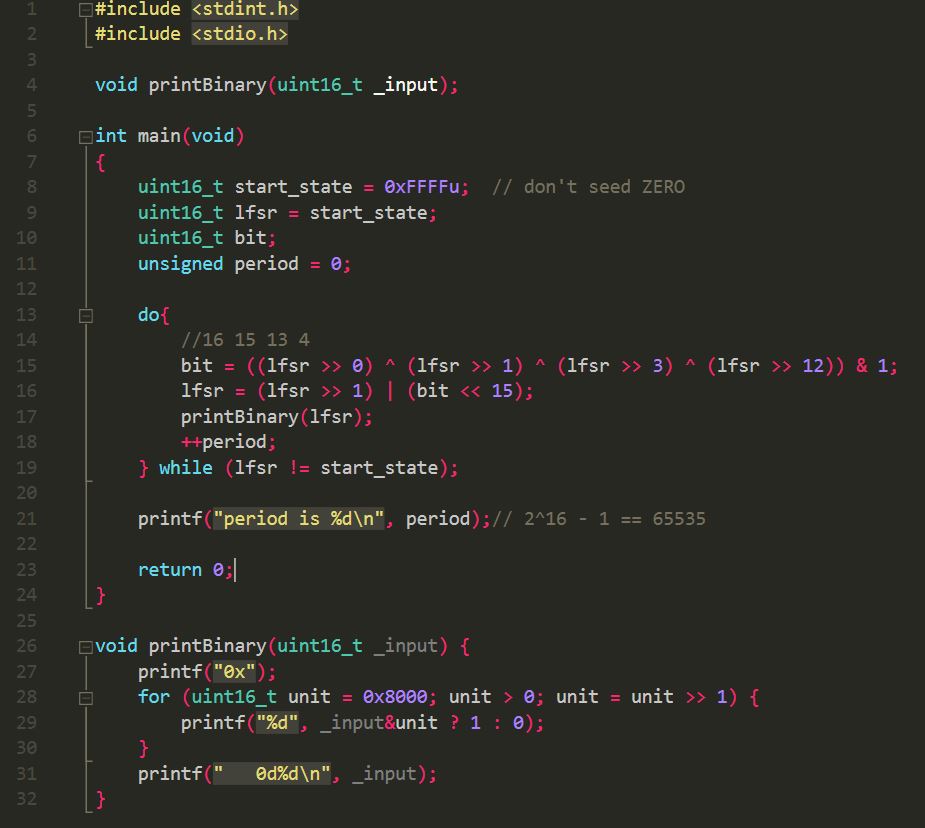
4비트 난수생성 사이클

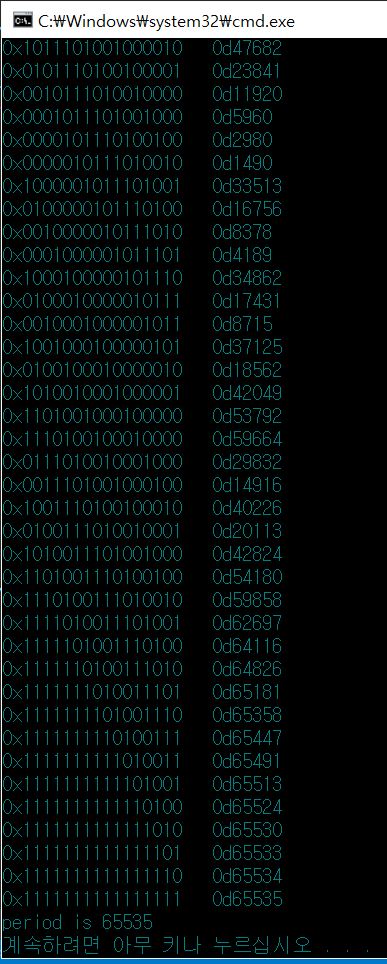
위에서 LFSR다항식을 list[3] ^ list[4]로 잡았는데 3과 4를 tap이라고 명명한다. Tap의 개수가 2의 배수이고, tap들의 집합이 relative prime 일때 LFSR의 주기는 최대가 된다.

최대주기 = (2^비트길이) -1 이며, 위의경우 최대주기는 2^4 -1인 15이다. 위의 그림에서 상태가 15개 이므로 3과 4는 주기를 최대로 만드는 적절한 tap이라고 할 수 있다.

 xilinx사의 문서(Efficient Shift Registers, LFSR, Counters, and Long Pseudo Random Sequence Generators)에서 얻은 비트길이별 tap의 위치이다.

C언어 구현

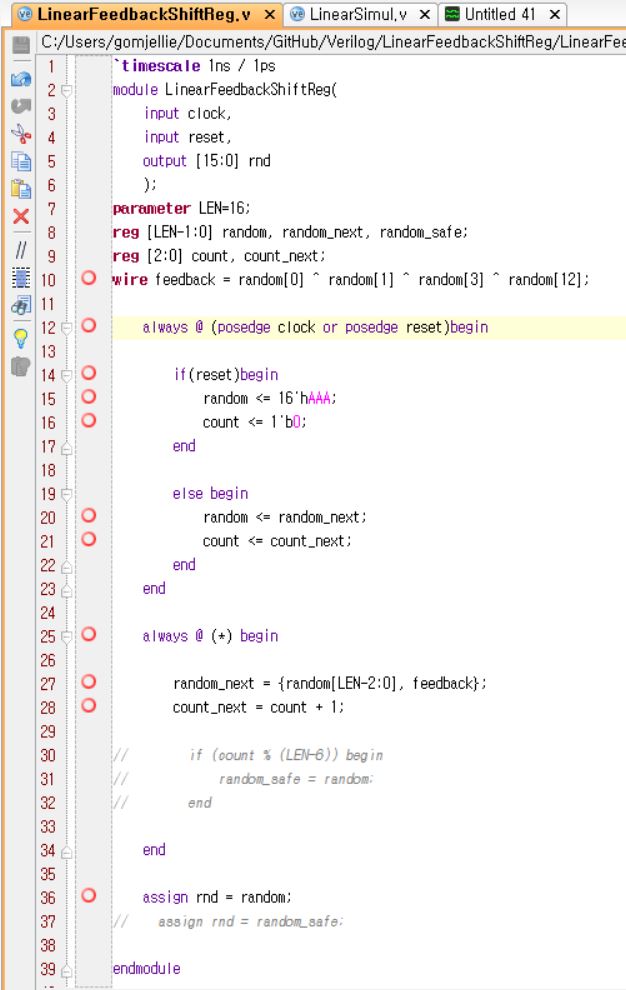
C언어로 구현한 LFSR알고리즘 코드이다.

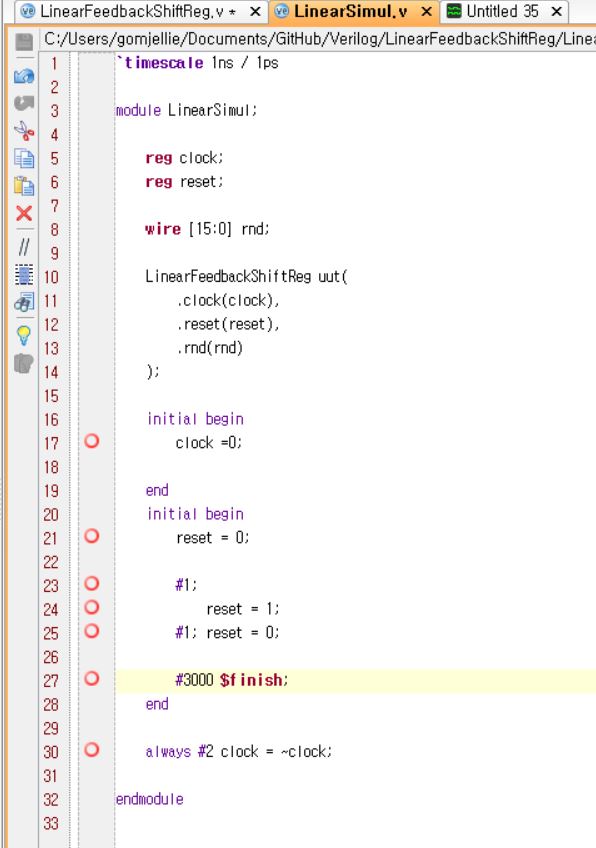
실행결과

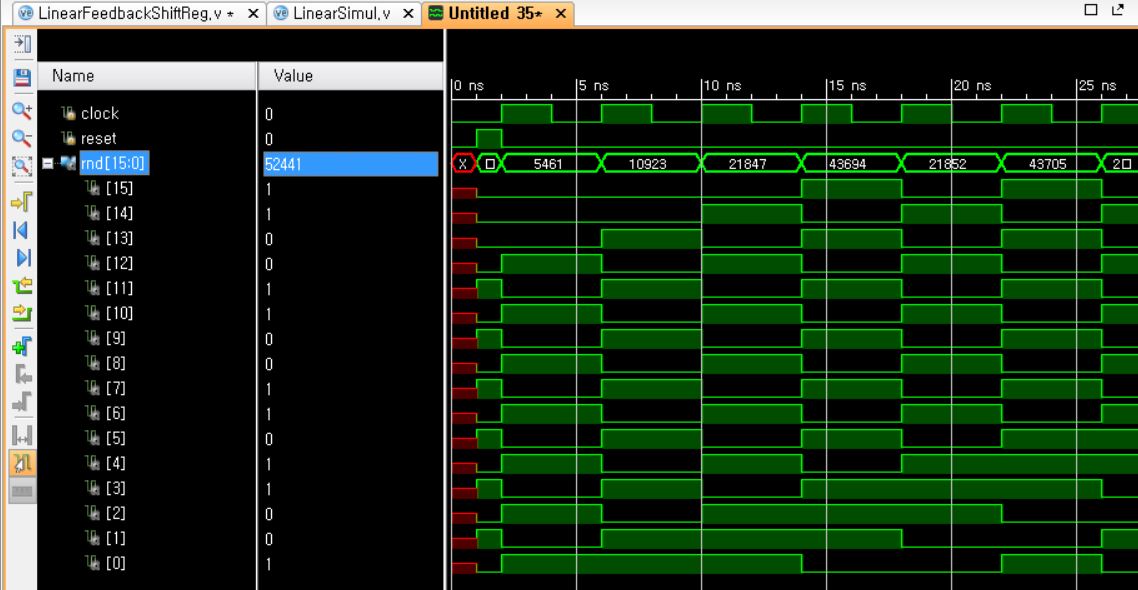
65535개의 의사난수를 출력하고 종료되는 모습이다.

tap값을 적절하게 골랐기 때문에 최대주기인 2^16-1 가 출력된다.

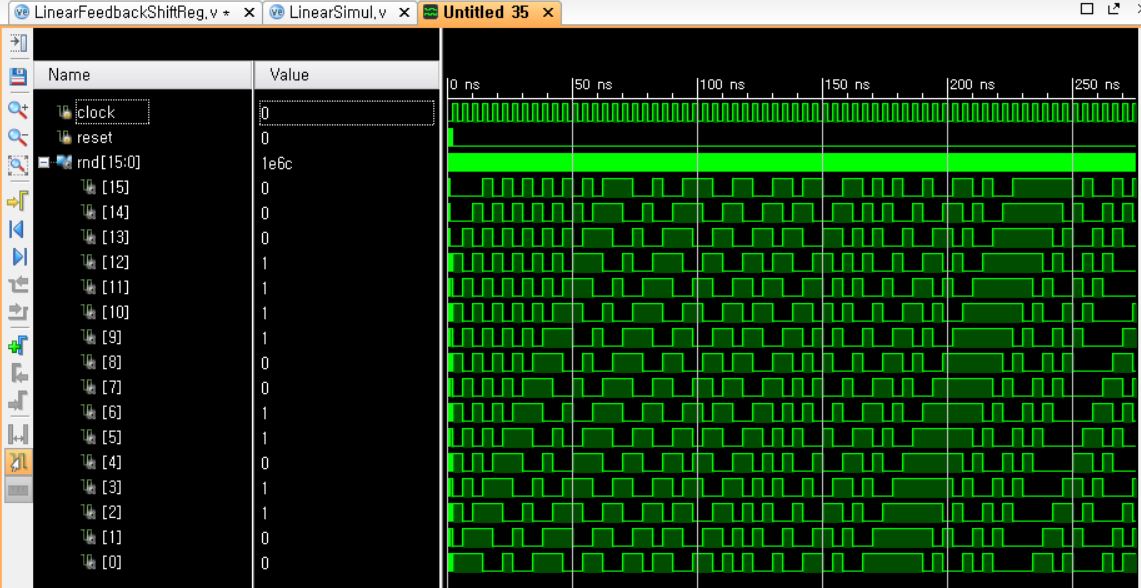
Verilog 구현

verilogCode

simulCode



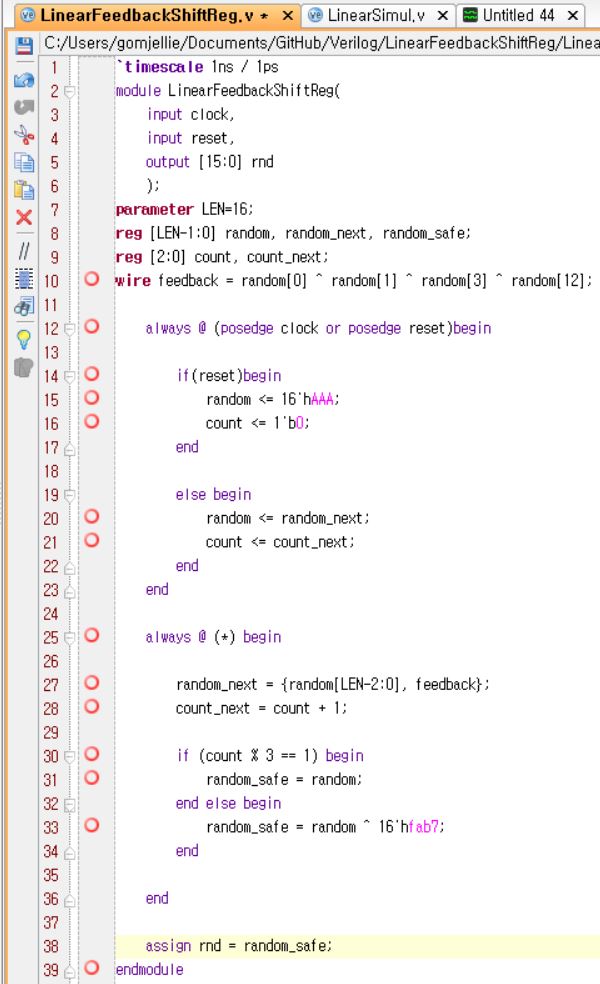
5461 -> 10923 -> 21847 -> 43694로 이어지는 난수는 10진수로 관찰할 때 규칙성을 찾을 수 없지만



2진수로 보면 1비트씩 밀려 내려가는 규칙을 쉽게 발견 할 수 있다.

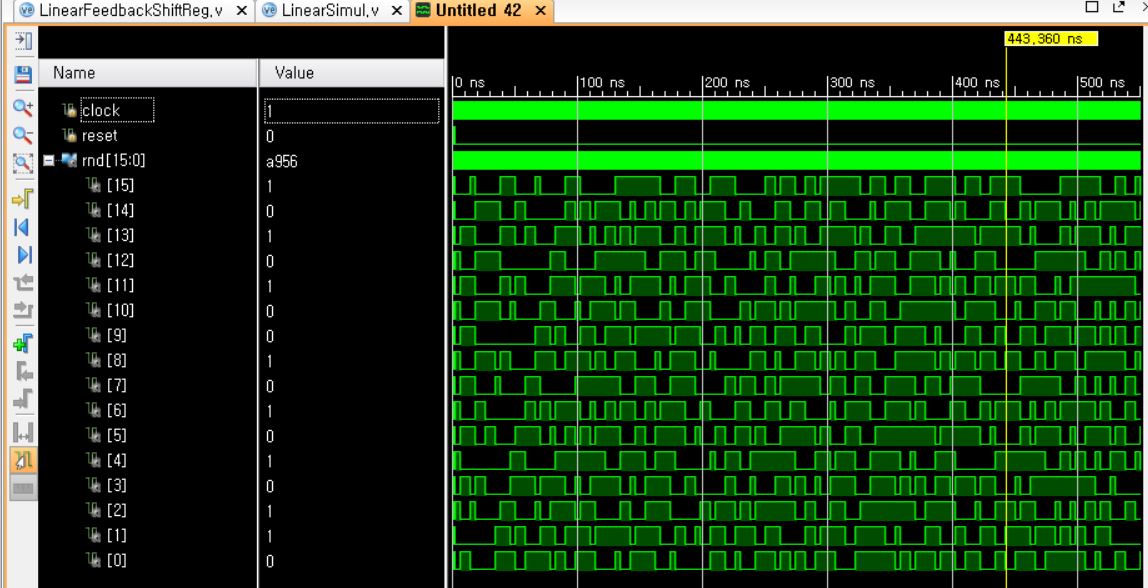
생성된 의사난수 수열을 분석하면서 LFSR다항식을 알게 되면 다음 수를 예측할 수 있기 때문에 개선이 필요하다.

Overflow와 합동을 이용한 개선

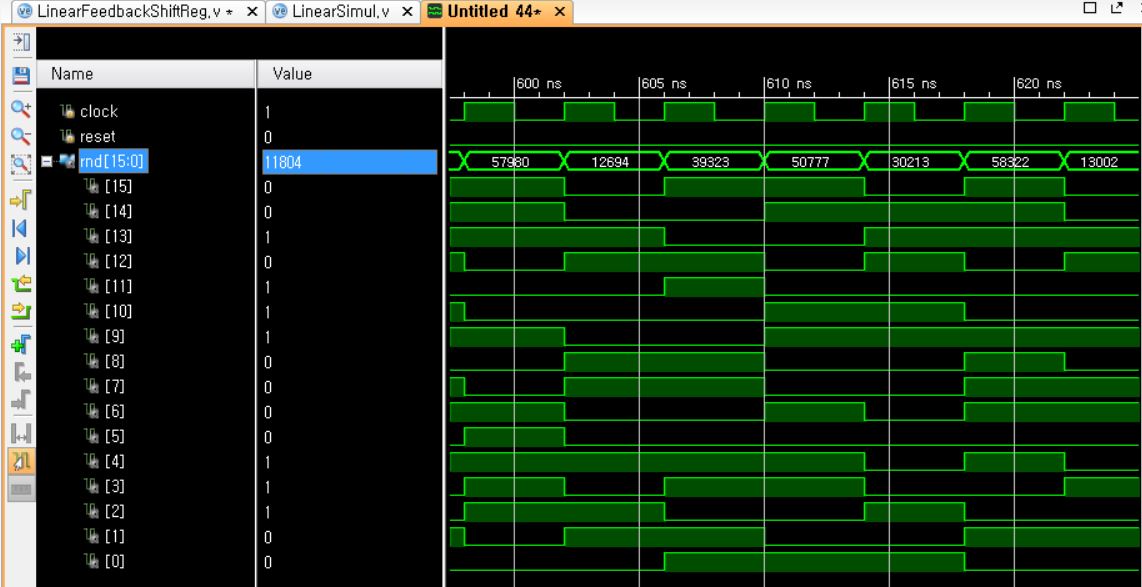


Count를 3비트로 설정해서 8을 주기로 값의 손실이 생기고, count(mod3)이 1과 합동일 때마다 random\_safe를 생성하여 출력으로 내보낸다.

Count(mod3)이 1또는 2일경우 random ^ 0xfab7 을 random\_safe에 할당한다. 임의로 지정한 수 0xfab7은 일종의 암호화 키 역할을 한다.



규칙을 발견하기 어려워 졌다.



Zedboard상에서 7segment로 구현

`Verilog code

timescale 1ns / 1ps

module Timer(

input clk, load, cnt,

[3:0] M,

//output [3:0] Q,

output reg led, reg [7:0] seg, reg [5:0] digit

);

reg [3:0] A;

reg ld;

wire tc;

wire [3:0] B;

wire [3:0] D;

wire [3:0] Q;

reg clk\_1hz;

reg [26:0]clk\_cnt;

Reg\_4 Reg(A,load,clk\_1hz,B);

Mux\_4 Mux(A,B,load,D);

Down\_ct count(D,load,cnt,clk\_1hz,Q);

//LED\_7 (led, seg, digit

always@(\*)

A = M - 1;

always@(posedge clk) begin//???

if(clk\_cnt == 27'd1000\_0000\_0) //???

clk\_cnt <= 27'd0;

else

clk\_cnt <= clk\_cnt + 1'b1;

end

always@(\*)//???

clk\_1hz <= clk\_cnt[25];

always@(\*) begin

digit <= 6'b100000;

case(Q)

4'b0000 : seg <= 8'b0011\_1111;//0

4'b0001 : seg <= 8'b0000\_0110;//1

4'b0010 : seg <= 8'b0101\_1011;//2

4'b0011 : seg <= 8'b0100\_1111;//3

4'b0100 : seg <= 8'b0110\_0110;//4

4'b0101 : seg <= 8'b0110\_1101;//5

4'b0110 : seg <= 8'b0111\_1100;//6

4'b0111 : seg <= 8'b0010\_0111;//7

4'b1000 : seg <= 8'b0111\_1111;//8

4'b1001 : seg <= 8'b0110\_0111;//9

4'b1010 : seg <= 8'b1111\_0111;//A

4'b1011 : seg <= 8'b1111\_1100;//B

4'b1100 : seg <= 8'b1011\_1001;//C

4'b1101 : seg <= 8'b1101\_1110;//D

4'b1110 : seg <= 8'b1111\_1001;//E

4'b1111 : seg <= 8'b1111\_0001;//F

default : seg <= 8'b0011\_1111;

endcase

if (tc==1)

led <= 1;

end

endmodule

module Reg\_4(A,load,clk\_1hz,B);

input [3:0] A;

input load, clk\_1hz;

output reg [3:0] B;

always@(posedge clk\_1hz) begin

if(load)

B <= A;

else

B <= B;

end

endmodule

module Mux\_4(A,B,load,D);

input [3:0] A;

input [3:0] B;

input load;

output reg [3:0] D;

always@(\*) begin

if(load)

D <= A;

else

D <= B;

end

endmodule

module Down\_ct(D,load,cnt,clk\_1hz,Q);

input [3:0] D;

input load,cnt,clk\_1hz;

output reg [3:0] Q;

reg tc, ld;//??

reg [3:0] D1;

wire randPart = D1[2] ^ D1[3];

always@(posedge clk\_1hz & cnt) begin//??

if(ld==1)

D1 <= D;

else

D1 <= {D1[2:0],randPart};

//D1 <= D1 - 1'b1;

tc <= ~(D1[0] | D1[1] | D1[2] | D1[3]);

Q <= D1;

end

always@(\*)

ld <= load | tc;

endmodule

XDC파일

set\_property PACKAGE\_PIN F21 [get\_ports {M[3]}]

set\_property PACKAGE\_PIN H22 [get\_ports {M[2]}]

set\_property PACKAGE\_PIN G22 [get\_ports {M[1]}]

set\_property PACKAGE\_PIN F22 [get\_ports {M[0]}]

set\_property PACKAGE\_PIN M15 [get\_ports {load}]

set\_property PACKAGE\_PIN H17 [get\_ports {cnt}]

set\_property PACKAGE\_PIN U14 [get\_ports {led}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {M[3]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {M[2]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {M[1]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {M[0]}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {load}]

set\_property IOSTANDARD LVCMOS18 [get\_ports {cnt}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led}]

set\_property PACKAGE\_PIN AA8 [get\_ports {seg[7]}]

set\_property PACKAGE\_PIN AB9 [get\_ports {seg[6]}]

set\_property PACKAGE\_PIN AB10 [get\_ports {seg[5]}]

set\_property PACKAGE\_PIN AB11 [get\_ports {seg[4]}]

set\_property PACKAGE\_PIN AA9 [get\_ports {seg[3]}]

set\_property PACKAGE\_PIN Y10 [get\_ports {seg[2]}]

set\_property PACKAGE\_PIN AA11 [get\_ports {seg[1]}]

set\_property PACKAGE\_PIN Y11 [get\_ports {seg[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {seg[0]}]

set\_property PACKAGE\_PIN Y9 [get\_ports {clk}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {clk}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digit[5]}]

set\_property PACKAGE\_PIN W10 [get\_ports {digit[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digit[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digit[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digit[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digit[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digit[0]}]

set\_property PACKAGE\_PIN V12 [get\_ports {digit[4]}]

set\_property PACKAGE\_PIN W8 [get\_ports {digit[3]}]

set\_property PACKAGE\_PIN V10 [get\_ports {digit[2]}]

set\_property PACKAGE\_PIN W11 [get\_ports {digit[1]}]

set\_property PACKAGE\_PIN W12 [get\_ports {digit[0]}]

Zedboard 실행결과 및 분석

실행 동영상 url : <https://goo.gl/FdXC0L>

3 6 d a 5 b 7 F e c 8 1 2 4 9 가 반복된다 (주기 15)

3 : 0011

6 : 0110

D : 1101

A : 1010

5 : 0101

B : 1011

7 : 0111

F : 1111

E : 1110

C : 1100

8 : 1000

1 : 0001

2 : 0010

4 : 0100

9 : 1001

앞서 보인 4비트 LFSR사이클 그림과 일치한다

Reference:

**xilinx사의 의사난수생성에 관한 문서**

[**http://www.xilinx.com/support/documentation/application\_notes/xapp052.pdf**](http://www.xilinx.com/support/documentation/application_notes/xapp052.pdf)

programming커뮤니티 stackoverflow의 verilog상에서의 의사난수 생성법에관한 질문과 답변

<http://stackoverflow.com/questions/14497877/how-to-implement-a-pseudo-hardware-random-number-generator>

네이버 캐스트 컴퓨터로 만들어내는 난수

[**http://navercast.naver.com/contents.nhn?rid=22&contents\_id=112416**](http://navercast.naver.com/contents.nhn?rid=22&contents_id=112416)

**Wikipedia** Linear-feedback shift register article

<https://en.wikipedia.org/wiki/Linear-feedback_shift_register>

위의 자료를 참조하였습니다.