module primary\_lsfr1 (

input clk,

input reset,

input write,

input pushin,

input [184:0] InitialData1,

output [184:0] rnd1

);

//Linear feedback shift registers

reg [184:0] lfsr1, random\_next1, random\_done1;

//Count for the number of shifts

reg [3:0] count1, count\_next1;

always @ (posedge clk or posedge reset)

begin

if (reset)

begin

lfsr1 <= #1 0;

//case1

//lfsr1 <= 185'h4751245563371bb82b2b5aacd05678a1b17e06c62eb0dace; //An LFSR cannot have an all 0 state, thus reset to 4751245563371bb82b2b5aacd05678a1b17e06c62eb0dace

end

else

begin

if (write)

begin

lfsr1 <= InitialData1;

//case2

//lfsr1 <= 185'h08AAC66E37215874F559A0ACF14362FC0D24CD61E1D5512;

count1 <= 0;

end

else if (pushin)

begin

lfsr1 <= #1 random\_next1;

count1 <= #1 count\_next1;

end

end

end

always @ (\*)

begin

//-----------Combinational code for shift register 1 --> 13 bits ----------//

random\_next1 = lfsr1; //default state stays the same

count\_next1 = count1;

random\_done1 = 0;

random\_next1 = { (lfsr1[171:134]), (lfsr1[133]^lfsr1[184]) ,(lfsr1[132]^lfsr1[183]) ,(lfsr1[131]^lfsr1[182]) ,(lfsr1[130]^lfsr1[181]) ,

(lfsr1[129]^lfsr1[184]^lfsr1[180]), (lfsr1[128]^lfsr1[183]^lfsr1[179]), (lfsr1[127]^lfsr1[182]^lfsr1[178]),

(lfsr1[126]^lfsr1[181]^lfsr1[177]), (lfsr1[125]^lfsr1[180]^lfsr1[176]), (lfsr1[124]^lfsr1[179]^lfsr1[175]),

(lfsr1[123]^lfsr1[178]^lfsr1[174]), (lfsr1[122]^lfsr1[177]^lfsr1[173]), (lfsr1[121]^lfsr1[176]^lfsr1[172]),

(lfsr1[120]^lfsr1[184]^lfsr1[175]), (lfsr1[119]^lfsr1[183]^lfsr1[174]), (lfsr1[118]^lfsr1[182]^lfsr1[173]),

(lfsr1[117]^lfsr1[181]^lfsr1[172]),

(lfsr1[116]^lfsr1[180]) ,(lfsr1[115]^lfsr1[179]) ,(lfsr1[114]^lfsr1[178]) ,(lfsr1[113]^lfsr1[177]) ,

(lfsr1[112]^lfsr1[176]) ,(lfsr1[111]^lfsr1[175]) ,(lfsr1[110]^lfsr1[174]) ,(lfsr1[109]^lfsr1[173]) ,

(lfsr1[108]^lfsr1[172]) ,(lfsr1[107:39]), (lfsr1[38]^lfsr1[184]) ,(lfsr1[37]^lfsr1[183]) ,(lfsr1[36]^lfsr1[182]) ,

(lfsr1[35]^lfsr1[181]) ,(lfsr1[34]^lfsr1[180]) ,(lfsr1[33]^lfsr1[179]) ,(lfsr1[32]^lfsr1[178]) ,

(lfsr1[31]^lfsr1[177]) ,(lfsr1[30]^lfsr1[176]) ,(lfsr1[29]^lfsr1[175]) ,(lfsr1[28]^lfsr1[174]) ,

(lfsr1[27]^lfsr1[173]) ,(lfsr1[26]^lfsr1[172]) , (lfsr1[25:14]), (lfsr1[13]^lfsr1[184]) ,(lfsr1[12]^lfsr1[183]) ,

(lfsr1[11]^lfsr1[182]) ,(lfsr1[10]^lfsr1[181]) , (lfsr1[09]^lfsr1[180]) ,(lfsr1[08]^lfsr1[179]) ,

(lfsr1[07]^lfsr1[178]) ,(lfsr1[06]^lfsr1[177]) ,(lfsr1[05]^lfsr1[176]) ,(lfsr1[04]^lfsr1[175]) ,

(lfsr1[03]^lfsr1[174]) ,(lfsr1[02]^lfsr1[173]) ,(lfsr1[01]^lfsr1[172]) , (lfsr1[0]), (lfsr1[184:172])};

count\_next1 = count1 + 1;

if (count1 == 1)

begin

count1 = 0;

random\_done1 = lfsr1; //assign the random number to output after 13 shifts

end

//--------------------------------------------End of combination logic for shift register 1----------------------------------//

end

assign rnd1 = lfsr1;

//assign rnd1 = random\_next1;

endmodule