module primary\_lsfr12 (

input clk,

input reset,

input write,

input pushin,

input [146:0] InitialData12,

output [146:0] rnd1

);

//Linear feedback shift registers

reg [146:0] lfsr12, 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

lfsr12 <= #1 0;

//case1

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

end

else

begin

if (write)

begin

lfsr12 <= InitialData12;

//case2

//lfsr1 <= 185'h08AAC66E37215874F559A0ACF14362FC0D24CD61E1D5512;

count1 <= 0;

end

else if (pushin)

begin

lfsr12 <= #1 random\_next1;

count1 <= #1 count\_next1;

end

end

end

always @ (\*)

begin

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

random\_next1 = lfsr12; //default state stays the same

count\_next1 = count1;

random\_done1 = 0;

random\_next1 = { (lfsr12[130]^lfsr12[136]^lfsr12[142]^lfsr12[141]^lfsr12[146]^lfsr12[135]^lfsr12[141]^lfsr12[146]^lfsr12[140]^lfsr12[146]^lfsr12[145]) ,

(lfsr12[129]^lfsr12[135]^lfsr12[141]^lfsr12[146]^lfsr12[140]^lfsr12[146]^lfsr12[145]^lfsr12[134]^lfsr12[140]^lfsr12[146]^lfsr12[145]^lfsr12[139]^lfsr12[145]^lfsr12[144]) ,

(lfsr12[128]^lfsr12[134]^lfsr12[140]^lfsr12[146]^lfsr12[145]^lfsr12[139]^lfsr12[145]^lfsr12[144]^lfsr12[133]^lfsr12[139]^lfsr12[145]^lfsr12[144]^lfsr12[138]^lfsr12[144]^lfsr12[143]) ,

(lfsr12[127]^lfsr12[133]^lfsr12[139]^lfsr12[145]^lfsr12[144]^lfsr12[138]^lfsr12[144]^lfsr12[143]^lfsr12[132]^lfsr12[138]^lfsr12[144]^lfsr12[143]^lfsr12[137]^lfsr12[143]^lfsr12[142]) ,

(lfsr12[126]^lfsr12[132]^lfsr12[138]^lfsr12[144]^lfsr12[143]^lfsr12[137]^lfsr12[143]^lfsr12[142]^lfsr12[131]^lfsr12[137]^lfsr12[143]^lfsr12[142]^lfsr12[136]^lfsr12[142]^lfsr12[141]^lfsr12[146]) ,

(lfsr12[125]^lfsr12[131]^lfsr12[137]^lfsr12[143]^lfsr12[142]^lfsr12[136]^lfsr12[142]^lfsr12[141]^lfsr12[146]) , (lfsr12[124:122]),

(lfsr12[121]^lfsr12[146]) ,(lfsr12[120]^lfsr12[145]) ,(lfsr12[119]^lfsr12[144]) ,(lfsr12[118]^lfsr12[143]) ,

(lfsr12[117]^lfsr12[146]^lfsr12[142]) ,(lfsr12[116]^lfsr12[145]^lfsr12[141]^lfsr12[146]) ,

(lfsr12[115]^lfsr12[144]^lfsr12[140]^lfsr12[146]^lfsr12[145]) ,(lfsr12[114]^lfsr12[143]^lfsr12[139]^lfsr12[145]^lfsr12[144]) ,

(lfsr12[113]^lfsr12[142]^lfsr12[138]^lfsr12[144]^lfsr12[143]) ,(lfsr12[112]^lfsr12[141]^lfsr12[146]^lfsr12[137]^lfsr12[143]^lfsr12[142]) ,

(lfsr12[111]^lfsr12[140]^lfsr12[146]^lfsr12[145]^lfsr12[136]^lfsr12[142]^lfsr12[141]^lfsr12[146]) ,

(lfsr12[110]^lfsr12[139]^lfsr12[145]^lfsr12[144]^lfsr12[135]^lfsr12[141]^lfsr12[146]^lfsr12[140]^lfsr12[146]^lfsr12[145]) ,

(lfsr12[109]^lfsr12[138]^lfsr12[144]^lfsr12[143]^lfsr12[134]^lfsr12[140]^lfsr12[146]^lfsr12[145]^lfsr12[139]^lfsr12[145]^lfsr12[144]) ,

(lfsr12[108]^lfsr12[137]^lfsr12[143]^lfsr12[142]^lfsr12[133]^lfsr12[139]^lfsr12[145]^lfsr12[144]^lfsr12[138]^lfsr12[144]^lfsr12[143]) ,

(lfsr12[107]^lfsr12[136]^lfsr12[142]^lfsr12[141]^lfsr12[146]^lfsr12[132]^lfsr12[138]^lfsr12[144]^lfsr12[143]^lfsr12[137]^lfsr12[143]^lfsr12[142]) ,

(lfsr12[106]^lfsr12[135]^lfsr12[141]^lfsr12[146]^lfsr12[140]^lfsr12[146]^lfsr12[145]^lfsr12[131]^lfsr12[137]^lfsr12[143]^lfsr12[142]^lfsr12[136]^lfsr12[142]^lfsr12[141]^lfsr12[146]) ,

(lfsr12[105]^lfsr12[134]^lfsr12[140]^lfsr12[146]^lfsr12[145]^lfsr12[139]^lfsr12[145]^lfsr12[144]) ,

(lfsr12[104]^lfsr12[133]^lfsr12[139]^lfsr12[145]^lfsr12[144]^lfsr12[138]^lfsr12[144]^lfsr12[143]) ,

(lfsr12[103]^lfsr12[132]^lfsr12[138]^lfsr12[144]^lfsr12[143]^lfsr12[137]^lfsr12[143]^lfsr12[142]) ,

(lfsr12[102]^lfsr12[131]^lfsr12[137]^lfsr12[143]^lfsr12[142]^lfsr12[136]^lfsr12[142]^lfsr12[141]^lfsr12[146]) , (lfsr12[101:50]),

(lfsr12[49]^lfsr12[146]), (lfsr12[48]^lfsr12[145]), (lfsr12[47]^lfsr12[144]), (lfsr12[46]^lfsr12[143]), (lfsr12[45]^lfsr12[142]), (lfsr12[44]^lfsr12[141]^lfsr12[146]) ,

(lfsr12[43]^lfsr12[140]^lfsr12[146]^lfsr12[145]) ,(lfsr12[42]^lfsr12[139]^lfsr12[145]^lfsr12[144]) ,(lfsr12[41]^lfsr12[138]^lfsr12[144]^lfsr12[143]) ,(lfsr12[40]^lfsr12[137]^lfsr12[143]^lfsr12[142]) ,

(lfsr12[39]^lfsr12[136]^lfsr12[142]^lfsr12[141]^lfsr12[146]) ,(lfsr12[38]^lfsr12[135]^lfsr12[141]^lfsr12[146]^lfsr12[140]^lfsr12[146]^lfsr12[145]) ,

(lfsr12[37]^lfsr12[134]^lfsr12[140]^lfsr12[146]^lfsr12[145]^lfsr12[139]^lfsr12[145]^lfsr12[144]) ,

(lfsr12[36]^lfsr12[133]^lfsr12[139]^lfsr12[145]^lfsr12[144]^lfsr12[138]^lfsr12[144]^lfsr12[143]) ,

(lfsr12[35]^lfsr12[132]^lfsr12[138]^lfsr12[144]^lfsr12[143]^lfsr12[137]^lfsr12[143]^lfsr12[142]) ,

(lfsr12[34]^lfsr12[131]^lfsr12[137]^lfsr12[143]^lfsr12[142]^lfsr12[136]^lfsr12[142]^lfsr12[141]^lfsr12[146]) , (lfsr12[33:0]), (lfsr12[146:142]),

(lfsr12[141]^lfsr12[146]) ,

(lfsr12[140]^lfsr12[146]^lfsr12[145]) ,(lfsr12[139]^lfsr12[145]^lfsr12[144]) ,(lfsr12[138]^lfsr12[144]^lfsr12[143]) ,(lfsr12[137]^lfsr12[143]^lfsr12[142]) ,

(lfsr12[136]^lfsr12[142]^lfsr12[141]^lfsr12[146]) ,(lfsr12[135]^lfsr12[141]^lfsr12[146]^lfsr12[140]^lfsr12[146]^lfsr12[145]) ,

(lfsr12[134]^lfsr12[140]^lfsr12[146]^lfsr12[145]^lfsr12[139]^lfsr12[145]^lfsr12[144]) ,

(lfsr12[133]^lfsr12[139]^lfsr12[145]^lfsr12[144]^lfsr12[138]^lfsr12[144]^lfsr12[143]) ,

(lfsr12[132]^lfsr12[138]^lfsr12[144]^lfsr12[143]^lfsr12[137]^lfsr12[143]^lfsr12[142]) ,

(lfsr12[131]^lfsr12[137]^lfsr12[143]^lfsr12[142]^lfsr12[136]^lfsr12[142]^lfsr12[141]^lfsr12[146]) };

count\_next1 = count1 + 1;

if (count1 == 1)

begin

count1 = 0;

random\_done1 = lfsr12; //assign the random number to output after 13 shifts

end

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

end

assign rnd1 = lfsr12;

endmodule