Задача 1

Неправильно указана логика переходов:

- порядок перехода должен быть обратный

- какое-либо вычисление только на последнем бите, в остальных случаях только сдвиг

Меняем:

```
assign next_scr_reg[0] = scr_reg[5] ^ scr_reg[6];
 assign next_scr_reg[1] = scr_reg[0];
 assign next_scr_reg[2] = scr_reg[1];
 assign next_scr_reg[3] = scr_reg[2];
 assign next_scr_reg[4] = scr_reg[3];
 assign next_scr_reg[5] = scr_reg[4] ^ scr_reg[3];
 assign next_scr_reg[6] = scr_reg[5];
```

на

```
assign next_scr_reg[6] = scr_reg[0] ^ scr_reg[1];
 assign next_scr_reg[5] = scr_reg[6];
 assign next_scr_reg[4] = scr_reg[5];
 assign next_scr_reg[3] = scr_reg[4];
 assign next_scr_reg[2] = scr_reg[3];
 assign next_scr_reg[1] = scr_reg[2];
 assign next_scr_reg[0] = scr_reg[1];
```

Неправильно указана логика выхода:

- используется не текущее, а следующее состояние, а выход вместо XOR вычисляется как ИЛИ

Меняем:

```
assign scr_seq = (scr_en) ? next_scr_reg : '0;
 assign scr_data_out[0] = data_in[0] || scr_seq[0];
```

на (с учетом того, что если сигнал шифрования равен 0, то данные пробрасываются, чего не отражено в схеме)

```
  assign scr_seq = scr_reg[SCR_WIDTH-1];
 assign scr_data_out = scr_en ? data_in ^ scr_seq: data_in;
```

- сигнал состояния выхода должен вычисляться не по текущему такту разрешающего выхода, а по следующему

Меняем:

```
always @(posedge clk)
   if(kill)
```

```
    begin
        data_out    <= '0;
        data_out_en <= '0;
    end
  else if(data_in_en)
    begin
        data_out    <= scr_data_out;
        data_out_en <= data_in_en_reg;
    end
  else
    begin
        data_out    <= '0;
        data_out_en <= '0;
    end
```

на

```
always @(posedge clk)
  if(kill)
    begin
        data_out    <= '0;
        data_out_en <= '0;
    end
  else if(data_in_en_reg)
    begin
        data_out    <= scr_data_out;
        data_out_en <= data_in_en_reg;
    end
  else
    begin
        data_out    <= '0;
        data_out_en <= '0;
    end
```

Далее, напишем тестбенч демонстрирующий работу устройства, подав те же сигналы что и в примере и построим диаграмму в GTK:

```
// Топ-модуль верификационного окружения
// скреблера(шифратора на (linear feedback shift register, LFSR)).
```

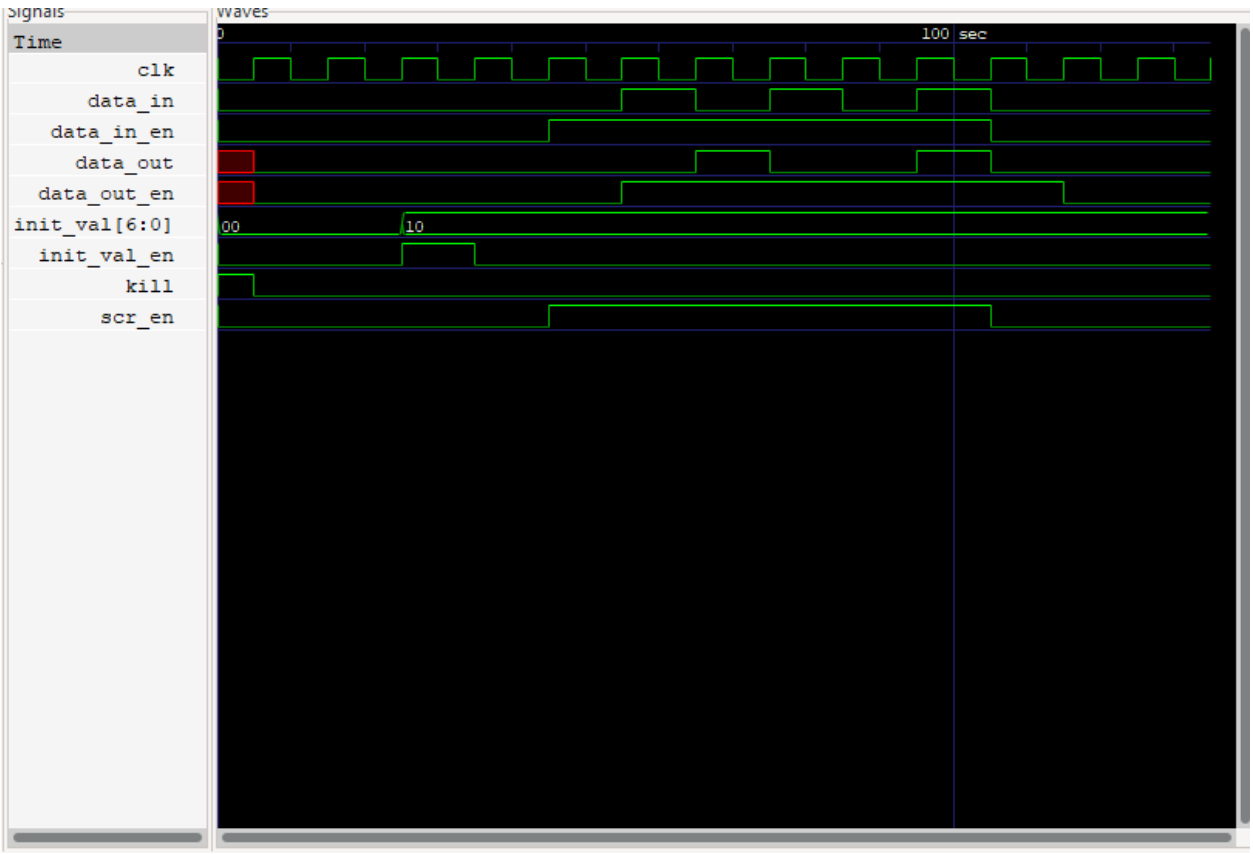
```
//  ------------------- TODO --------------------
// | В данном задании все изменения необходимо |
// | вносить в этот файл                       |
//  ------------------- TODO --------------------
module tb_scr_1dim_core ();
   parameter DATA_WIDTH    = 1;
   parameter SCR_WIDTH     = 7;
   // Определение тактового сигнала и
   // сигнала сброса.
   logic clk;
   logic kill;
   // Определение проводов, которые будут
   // подключены к портам модуля.
         logic                    scr_en;
         logic  [DATA_WIDTH-1:0]      data_in;
         logic                    data_in_en;
         logic  [SCR_WIDTH-1:0]       init_val;
         logic                    init_val_en;
         logic  [DATA_WIDTH-1:0]      data_out;
         logic                    data_out_en;
         logic  [DATA_WIDTH-1:0]      scr_seq;
         logic  [SCR_WIDTH-1:0]       scr_reg;
   // Подключение тестируемого модуля.
   scr_1dim_core #(
      .DATA_WIDTH     (DATA_WIDTH ),
      .SCR_WIDTH     (SCR_WIDTH)
   ) DUT (
      .clk             ( clk       ),
      .kill            ( kill      ),
      .scr_en          ( scr_en     ),
      .data_in         ( data_in    ),
      .data_in_en       ( data_in_en ),
      .init_val_en      ( init_val_en),
      .init_val         ( init_val),
      .data_out         ( data_out   ),
      .data_out_en      ( data_out_en )
   );
```

```
initial begin
   // Настройка waveform
   // Необходимо для создание временных диаграмм
   $dumpfile("wave.vcd");
   $dumpvars(0, tb_scr_1dim_core);
end
// Генерация тактового сигнала.
initial begin
   clk <= 1'b0;
   forever begin
      #5 clk = ~clk;
   end
end
// Генерация сигнала сброса.
initial begin
   kill <= 1'b1;
   @(posedge clk);
   kill = 1'b0;
end
initial begin
   $display("1");
   init_val_en = 0;
   init_val = 7'h00;
   scr_en = 0;
   data_in_en = 0;
   data_in = 0;
   #25;
   init_val_en = 1;
   init_val = 7'h10;
   #10;
   init_val_en = 0;
   #10;
   scr_en = 1;
   data_in_en = 1;
   #10;
   data_in = 1;
   #10;
```

```
        data_in = 0;
        #10;
        data_in = 1;
        #10;
        data_in = 0;
        #10;
        data_in = 1;
        #10;
        data_in = 0;
        data_in_en = 0;
        scr_en = 0;
            //  ------------------ TODO ---------------------
        // | Тестовое воздействие необходимо писать здесь|
        //  ------------------ TODO ---------------------

        // Завершение.
        #30;
        $finish();
    end
    initial begin
        $monitor("time=%3d, clk=%b, kill=%b, init_val_en=%b, init_val=%h, scr_en=%b, data_in_en=%b,
data_in=%b||data_out=%b, data_out_en=%b", $time, clk,kill,init_val_en, init_val, scr_en, data_in_en, data_in, data_out,
data_out_en);
    end


endmodule
```



Задача 2

Инвертированный асинхронный сброс -> убрать отрицание

Неправильная логика выходного переноса -> должна быть дизъюнкция всех попарных конъюнкций

Итого меняем:

```
always @(posedge clk_i) begin
    if( ~areset_i ) begin
      {Cout_o, S_o} <= 'b0;
    end
    else begin
      S_o   <= A_i ^ B_i ^ Cin_i;
      Cout_o <= ( A_i | B_i ) | ( A_i & Cin_i ) |
                ( B_i & Cin_i );
    end
  end
```

на

```
always @(posedge clk_i) begin
    if( areset_i ) begin
      {Cout_o, S_o} <= 'b0;
    end
    else begin
      S_o   <= A_i ^ B_i ^ Cin_i;
      Cout_o <= ( A_i && B_i ) || ( A_i && Cin_i ) ||
                ( B_i && Cin_i );
    end
  end
```

Напишем тестбенч как в примере и проведем симуляцию в GTK:

```
// Топ-модуль верификационного окружения

//  ------------------ TODO -------------------
// | В данном задании все изменения необходимо |
// | вносить в этот файл                       |
//  ------------------ TODO -------------------

module tb_fulladder;

  // Определение тактового сигнала и
  // сигнала сброса.
  logic clk_i;
  logic areset_i;

  // Определение проводов, которые будут
  // подключены к портам блока.
  logic A_i;
  logic B_i;
  logic Cin_i;
  logic Cout_o;
  logic S_o;

  // Подключение тестируемого модуля.
```

```
fulladder_sync DUT (
    .clk_i    ( clk_i    ),
    .areset_i ( areset_i ),
    .A_i      ( A_i      ),
    .B_i      ( B_i      ),
    .Cin_i    ( Cin_i    ),
    .Cout_o   ( Cout_o   ),
    .S_o      ( S_o      )
);


initial begin
    // Настройка waveform
    // Необходимо для создание временных диаграмм
    $dumpfile("wave.vcd");
    $dumpvars(0, tb_fulladder);
end
initial begin
    clk_i <= 1'b0;
    forever begin
        #5 clk_i = ~clk_i;
    end
end
initial begin

    //  ------------------ TODO ----------------------
    // | Тестовое воздействие необходимо писать здесь|
    //  ------------------ TODO ----------------------

    // Завершение.
    A_i = 0;
    B_i = 0;
    Cin_i = 0;
    areset_i = 1;
    #10;
    areset_i = 0;
    #5;
    A_i = 1;
```

```
    #10;
    A_i = 0;
    B_i = 1;
    #10;
    A_i = 1;
    #10;
    A_i = 0;
    B_i = 0;
    Cin_i = 1;
    #10;
    A_i = 1;
    #10;
    A_i = 0;
    B_i = 1;
    #10;
    A_i = 1;
    #15;
    $finish();
  end


endmodule
```