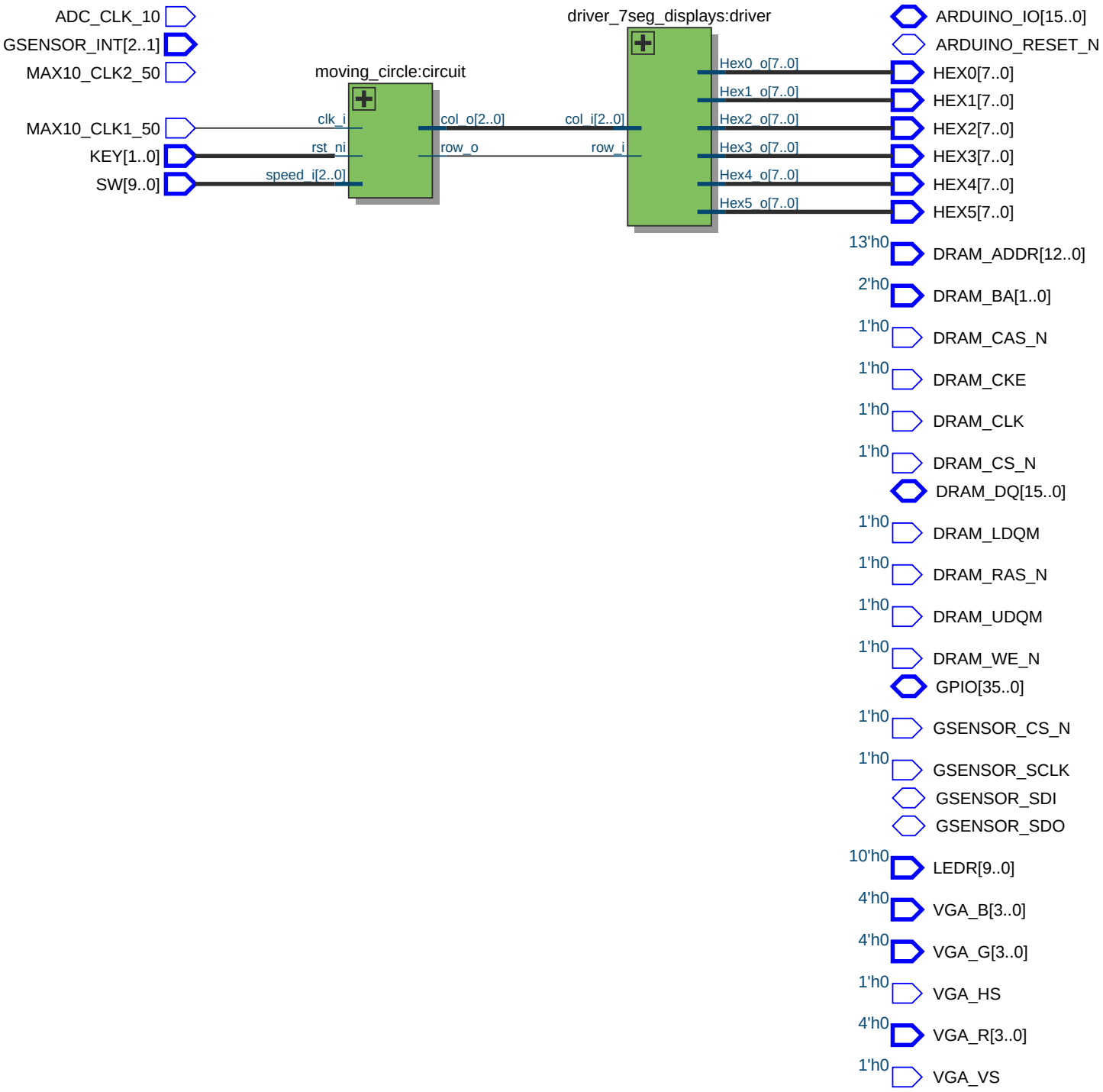
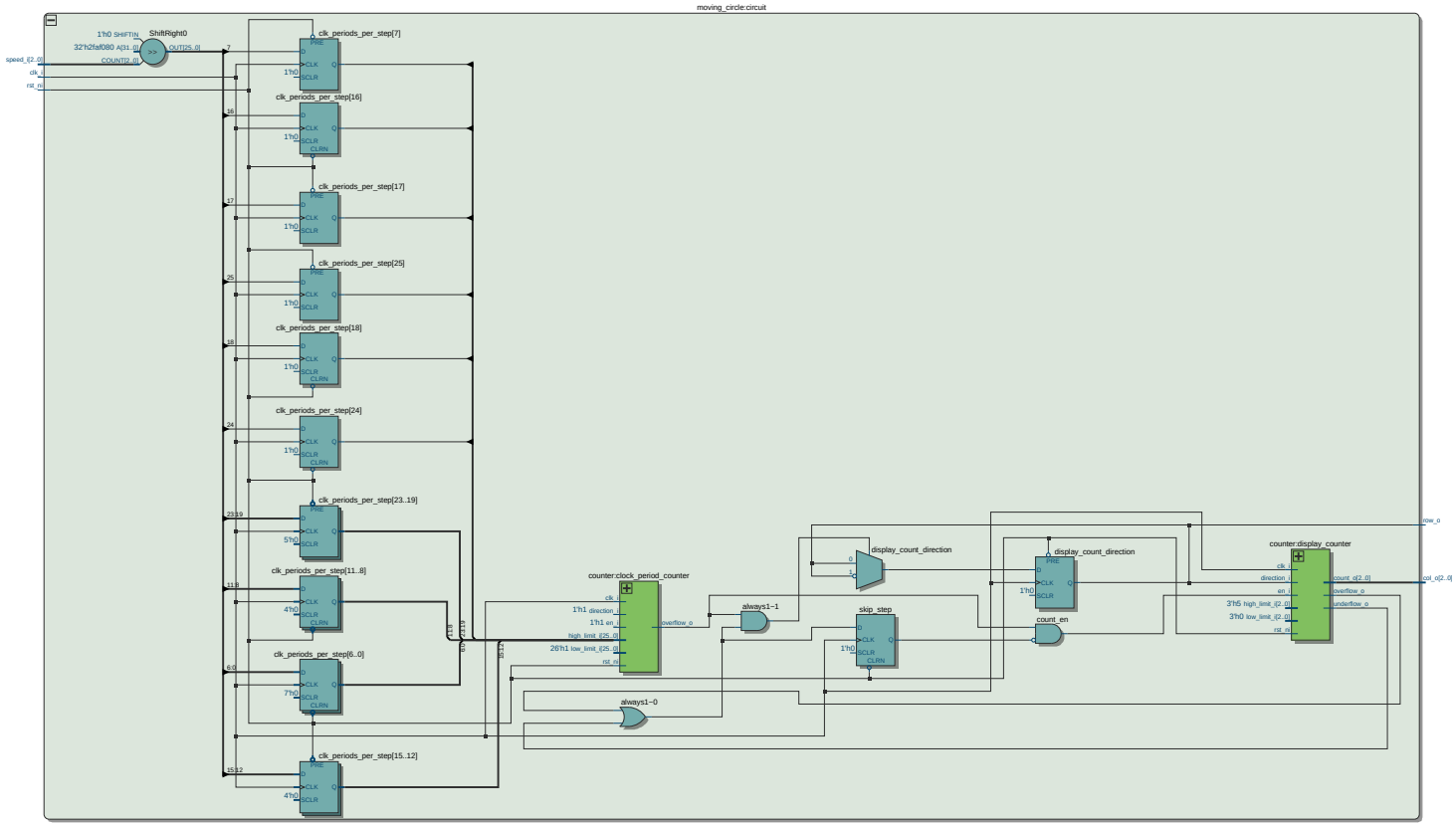
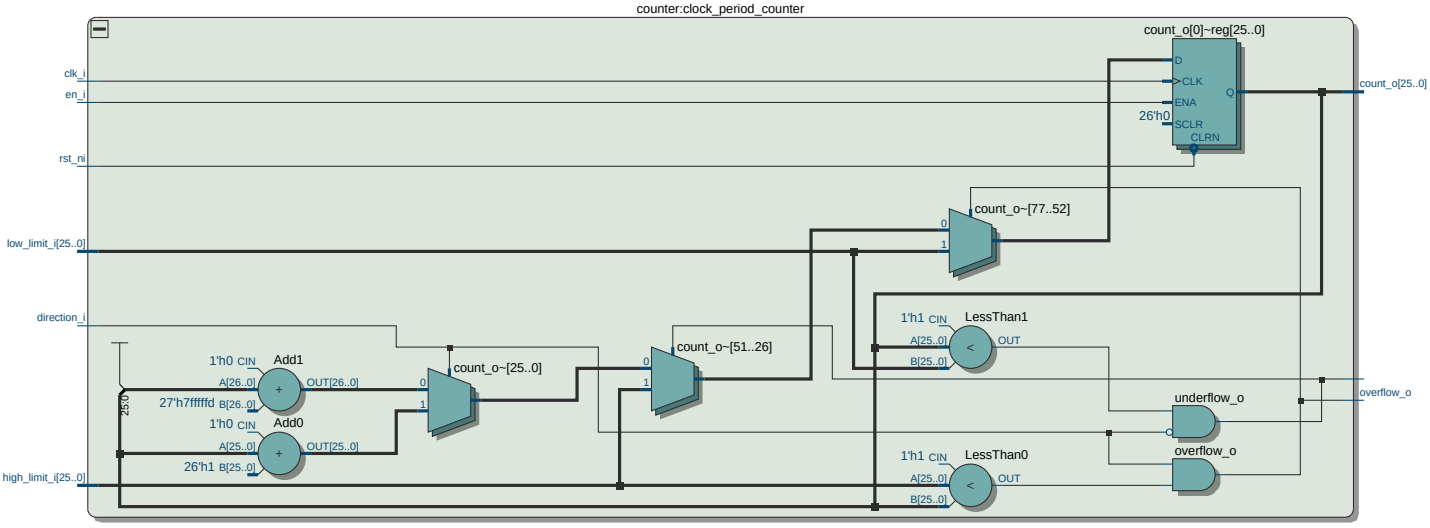
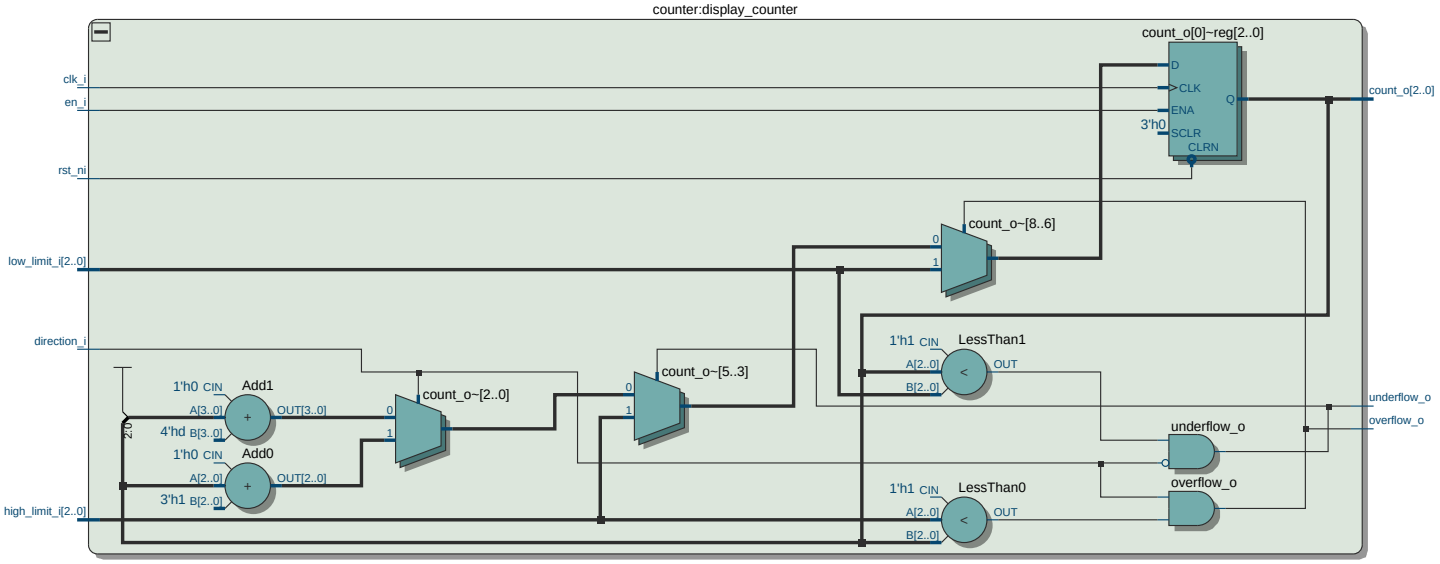
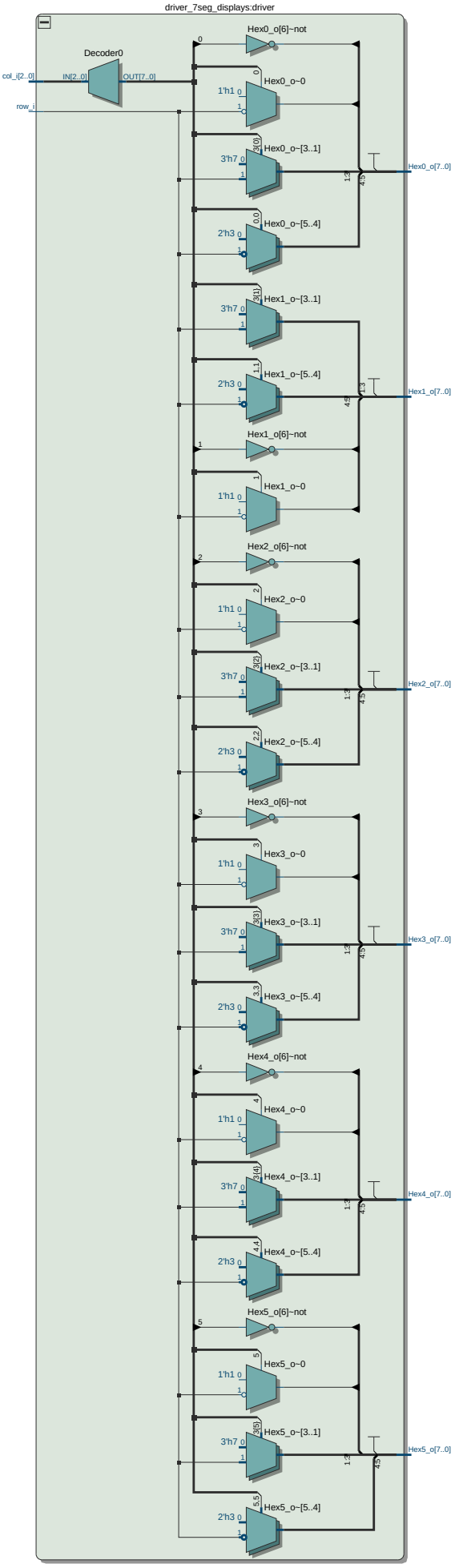

ADC_CLK_10
GSENSOR_INT[2..1]
MAX10_CLK2_50
MAX10_CLK1_50
KEY[1..0]
SW[9..0]
moving_circle:circuit
clk_i
rst_ni
speed_i[2..0]
col_o[2..0]
row_o
driver_7seg_displays:driver
col_i[2..0]
row_i
Hex0_o[7..0]
Hex1_o[7..0]
Hex2_o[7..0]
Hex3_o[7..0]
Hex4_o[7..0]
Hex5_o[7..0]
ARDUINO_IO[15..0]
ARDUINO_RESET_N
HEX0[7..0]
HEX1[7..0]
HEX2[7..0]
HEX3[7..0]
HEX4[7..0]
HEX5[7..0]
13'h0
DRAM_ADDR[12..0]
2'h0
DRAM_BA[1..0]
1'h0
DRAM_CAS_N
1'h0
DRAM_CKE
1'h0
DRAM_CLK
1'h0
DRAM_CS_N
DRAM_DQ[15..0]
1'h0
DRAM_LDQM
1'h0
DRAM_RAS_N
1'h0
DRAM_UDQM
1'h0
DRAM_WE_N
GPIO[35..0]
1'h0
GSENSOR_CS_N
1'h0
GSENSOR_SCLK
GSENSOR_SDI
GSENSOR_SDO
10'h0
LEDR[9..0]
4'h0
VGA_B[3..0]
4'h0
VGA_G[3..0]
1'h0
VGA_HS
4'h0
VGA_R[3..0]
1'h0
VGA_VS


moving_circle:circuit
1'h0 SHIFTIN
32'h2faf080 A[31..0]
ShiftRight0
OUT[25..0]
COUNT[2..0]
speed_i[2..0]
clk_i
rst_i
clk_periods_per_step[7]
clk_periods_per_step[16]
clk_periods_per_step[17]
clk_periods_per_step[25]
clk_periods_per_step[18]
clk_periods_per_step[24]
clk_periods_per_step[23..19]
clk_periods_per_step[11..8]
clk_periods_per_step[6..0]
clk_periods_per_step[15..12]
counter:clock_period_counter
always1~1
always1~0
display_count_direction
skip_step
count_en
counter:display_counter
row_o
col_o[2..0]


counter:clock_period_counter
count_o[0]~reg[25..0]
D
CLK
ENA
SCLR
CLRN
Q
26'h0
count_o[25..0]
clk_i
en_i
rst_ni
low_limit_i[25..0]
direction_i
high_limit_i[25..0]
count_o~[77..52]
count_o~[51..26]
count_o~[25..0]
0
1
1'h0 CIN
Add1
A[26..0]
27'h7fffffd B[26..0]
OUT[26..0]
1'h0 CIN
Add0
A[25..0]
26'h1 B[25..0]
OUT[25..0]
25:0
1'h1 CIN
LessThan1
A[25..0]
B[25..0]
OUT
1'h1 CIN
LessThan0
A[25..0]
B[25..0]
OUT
underflow_o
overflow_o
overflow_o


counter:display_counter
count_o[0]~reg[2..0]
D
CLK
ENA
3'h0
SCLR
CLRN
Q
count_o[2..0]
clk_i
en_i
rst_ni
low_limit_i[2..0]
direction_i
high_limit_i[2..0]
count_o~[8..6]
count_o~[5..3]
count_o~[2..0]
1'h0 CIN
Add1
A[3..0]
4'hd B[3..0]
OUT[3..0]
1'h0 CIN
Add0
A[2..0]
3'h1 B[2..0]
OUT[2..0]
2'h2
1'h1 CIN
LessThan1
A[2..0]
B[2..0]
OUT
1'h1 CIN
LessThan0
A[2..0]
B[2..0]
OUT
underflow_o
overflow_o
underflow_o
overflow_o


driver_7seg_displays:driver
Decoder0
col_i[2..0]
IN[2..0]
OUT[7..0]
row_i
Hex0_o[6]~not
Hex0_o~0
Hex0_o~[3..1]
Hex0_o~[5..4]
Hex0_o[7..0]
Hex1_o~[3..1]
Hex1_o~[5..4]
Hex1_o[6]~not
Hex1_o~0
Hex1_o[7..0]
Hex2_o[6]~not
Hex2_o~0
Hex2_o~[3..1]
Hex2_o~[5..4]
Hex2_o[7..0]
Hex3_o[6]~not
Hex3_o~0
Hex3_o~[3..1]
Hex3_o~[5..4]
Hex3_o[7..0]
Hex4_o[6]~not
Hex4_o~0
Hex4_o~[3..1]
Hex4_o~[5..4]
Hex4_o[7..0]
Hex5_o[6]~not
Hex5_o~0
Hex5_o~[3..1]
Hex5_o~[5..4]
Hex5_o[7..0]
1'h1
3'h7
2'h3

```
// ============================================================================
//   Ver  :| Author            :| Mod. Date :| Changes Made:
//   V1.1 :| Alexandra Du       :| 06/01/2016:| Added Verilog file
// ============================================================================


//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

`define ENABLE_ADC_CLOCK
`define ENABLE_CLOCK1
`define ENABLE_CLOCK2
`define ENABLE_SDRAM
`define ENABLE_HEX0
`define ENABLE_HEX1
`define ENABLE_HEX2
`define ENABLE_HEX3
`define ENABLE_HEX4
`define ENABLE_HEX5
`define ENABLE_KEY
`define ENABLE_LED
`define ENABLE_SW
`define ENABLE_VGA
`define ENABLE_ACCELEROMETER
`define ENABLE_ARDUINO
`define ENABLE_GPIO

module DE10_LITE_Golden_Top(

      ///////////// ADC CLOCK: 3.3-V LVTTL //////////
`ifdef ENABLE_ADC_CLOCK
      input                       ADC_CLK_10,
`endif
      ///////////// CLOCK 1: 3.3-V LVTTL //////////
`ifdef ENABLE_CLOCK1
      input                       MAX10_CLK1_50,
`endif
      ///////////// CLOCK 2: 3.3-V LVTTL //////////
`ifdef ENABLE_CLOCK2
      input                       MAX10_CLK2_50,
`endif

      ///////////// SDRAM: 3.3-V LVTTL //////////
`ifdef ENABLE_SDRAM
      output          [12:0]      DRAM_ADDR,
      output           [1:0]      DRAM_BA,
      output                      DRAM_CAS_N,
      output                      DRAM_CKE,
      output                      DRAM_CLK,
      output                      DRAM_CS_N,
      inout           [15:0]      DRAM_DQ,
      output                      DRAM_LDQM,
      output                      DRAM_RAS_N,
      output                      DRAM_UDQM,
      output                      DRAM_WE_N,
`endif

      ///////////// SEG7: 3.3-V LVTTL //////////
`ifdef ENABLE_HEX0
      output           [7:0]      HEX0,
`endif
`ifdef ENABLE_HEX1
      output           [7:0]      HEX1,
`endif
`ifdef ENABLE_HEX2
      output           [7:0]      HEX2,
`endif
`ifdef ENABLE_HEX3
```

```
70      output           [7:0]     HEX3,
71    `endif
72    `ifdef ENABLE_HEX4
73      output           [7:0]     HEX4,
74    `endif
75    `ifdef ENABLE_HEX5
76      output           [7:0]     HEX5,
77    `endif
78
79      ////////////// KEY: 3.3 V SCHMITT TRIGGER //////////
80    `ifdef ENABLE_KEY
81      input            [1:0]     KEY,
82    `endif
83
84      ////////////// LED: 3.3-V LVTTL //////////
85    `ifdef ENABLE_LED
86      output           [9:0]     LEDR,
87    `endif
88
89      ////////////// SW: 3.3-V LVTTL //////////
90    `ifdef ENABLE_SW
91      input            [9:0]     SW,
92    `endif
93
94      ////////////// VGA: 3.3-V LVTTL //////////
95    `ifdef ENABLE_VGA
96      output           [3:0]     VGA_B,
97      output           [3:0]     VGA_G,
98      output                     VGA_HS,
99      output           [3:0]     VGA_R,
100     output                     VGA_VS,
101   `endif
102
103     ////////////// Accelerometer: 3.3-V LVTTL //////////
104   `ifdef ENABLE_ACCELEROMETER
105     output                     GSENSOR_CS_N,
106     input            [2:1]     GSENSOR_INT,
107     output                     GSENSOR_SCLK,
108     inout                      GSENSOR_SDI,
109     inout                      GSENSOR_SDO,
110   `endif
111
112     ////////////// Arduino: 3.3-V LVTTL //////////
113   `ifdef ENABLE_ARDUINO
114     inout           [15:0]     ARDUINO_IO,
115     inout                      ARDUINO_RESET_N,
116   `endif
117
118     ////////////// GPIO, GPIO connect to GPIO Default: 3.3-V LVTTL //////////
119   `ifdef ENABLE_GPIO
120     inout           [35:0]     GPIO
121   `endif
122   );
123
124
125
126   //=======================================================
127   //  REG/WIRE declarations
128   //=======================================================
129
130   logic [3:0] display_count;
131   logic display_count_direction;
132
133
134   //=======================================================
135   //  Structural coding
136   //=======================================================
137
138
```

```
139   driver_7seg_displays driver (
140       .col_i          (display_count),
141       .row_i          (display_count_direction),
142       .Hex0_o         (HEX0),
143       .Hex1_o         (HEX1),
144       .Hex2_o         (HEX2),
145       .Hex3_o         (HEX3),
146       .Hex4_o         (HEX4),
147       .Hex5_o         (HEX5)
148   );
149
150   moving_circle circuit (
151      .clk_i         (MAX10_CLK1_50),
152      .rst_ni        (KEY[0]),
153      .speed_i       (SW[2:0]),
154      .row_o         (display_count_direction),
155      .col_o         (display_count)
156   );
157
158   endmodule
159
```

```
module moving_circle #(
    CLK_FREQ = 50_000_000, //50 Mhz  50000000
    STEPS_PER_SEC = 1,
    NO_DISPLAYS = 6
)(
    input clk_i,
    input rst_ni,
    input [2:0] speed_i,

    output logic row_o,
    output logic [3-1:0] col_o

);

// clog2(x) = y returns the smallest integer y such that 2^y >= x
// ex: clog2(8) = 3, clog2(9) = 4

localparam int CLK_PERIODS_PER_STEP = CLK_FREQ / STEPS_PER_SEC;
localparam CLK_DIV_WIDTH = $clog2(CLK_PERIODS_PER_STEP);

localparam DISPLAYS_COUNT_WIDTH = $clog2(NO_DISPLAYS);

logic step;
logic [CLK_DIV_WIDTH-1:0] clk_periods_per_step;

logic [DISPLAYS_COUNT_WIDTH-1:0] display_count;
logic display_count_direction, display_count_overflow, display_count_underflow;

logic skip_step, count_en;


always @ (posedge clk_i or negedge rst_ni)
    if(~rst_ni)
        clk_periods_per_step <= CLK_PERIODS_PER_STEP;
    else
        clk_periods_per_step <= CLK_PERIODS_PER_STEP >> speed_i;
        // de precizat clk_periods_per_step = CLK_PERIODS_PER_STEP/ 2^speed_i
        // 101  >> 010 = 001
        // 50 shiftat cu 3 pozitii
        // 50 = baza2 => 110010 >> 011 = 000110

counter #(
    .WIDTH(CLK_DIV_WIDTH)
) clock_period_counter (
    .clk_i          (clk_i),
    .rst_ni         (rst_ni),
    .direction_i    (1'b1),
    .en_i           (1'b1),
    .low_limit_i    (1),
    .high_limit_i   (clk_periods_per_step),
    .count_o        (),
    .overflow_o     (step),
    .underflow_o    ()
);

counter #(
    .WIDTH(DISPLAYS_COUNT_WIDTH)
) display_counter (
    .clk_i          (clk_i),
    .rst_ni         (rst_ni),
    .direction_i    (display_count_direction), // row
    .en_i           (count_en),
    .low_limit_i    (0),
    .high_limit_i   (NO_DISPLAYS-1),
    .count_o        (display_count),  //col
    .overflow_o     (display_count_overflow),
    .underflow_o    (display_count_underflow)
);

```

```
70  assign count_en = step && !skip_step;
71  assign row_o = display_count_direction;
72  assign col_o = display_count;
73
74
75  //schimbam directia/randul la ovf sau underflow
76  always @ (posedge clk_i or negedge rst_ni)
77      if(~rst_ni)
78          display_count_direction <= 1'b1;
79      else if(step && (display_count_overflow || display_count_underflow))
80          display_count_direction <= ~display_count_direction;
81
82
83  // folosim skip step pentru a astepta sa se schimbe directia
84  // altfel se va schimba directia de 2 ori la fiecare overflow/underflow
85  // si va ramane blocat (se interschimba intre limitele sale) 0 sau 5
86
87  always @ (posedge clk_i or negedge rst_ni)
88      if(~rst_ni)
89          skip_step <= 1'b0;
90      else if(display_count_overflow || display_count_underflow)
91          skip_step <= 1'b1;
92      else
93          skip_step <= 1'b0;
94
95  endmodule
```

```
1   module driver_7seg_displays(
2       input [2:0] col_i,
3       input       row_i,
4       output reg [7:0] Hex0_o, Hex1_o, Hex2_o, Hex3_o, Hex4_o, Hex5_o
5   );
6   //                            76543210
7   localparam up_circle =   8'b10011100;
8   localparam down_circle = 8'b10100011;
9   localparam empty =       8'b11111111;
10
11  reg [7:0] selected_symbol;
12
13  always_comb begin
14      Hex0_o = empty;
15      Hex1_o = empty;
16      Hex2_o = empty;
17      Hex3_o = empty;
18      Hex4_o = empty;
19      Hex5_o = empty;
20
21      selected_symbol = (row_i) ? up_circle : down_circle;
22
23      case (col_i)
24          3'b000: Hex0_o = selected_symbol;
25          3'b001: Hex1_o = selected_symbol;
26          3'b010: Hex2_o = selected_symbol;
27          3'b011: Hex3_o = selected_symbol;
28          3'b100: Hex4_o = selected_symbol;
29          3'b101: Hex5_o = selected_symbol;
30          default: begin
31              Hex0_o = empty;
32              Hex1_o = empty;
33              Hex2_o = empty;
34              Hex3_o = empty;
35              Hex4_o = empty;
36              Hex5_o = empty;
37          end
38      endcase
39  end
40
41
42  endmodule
```

```
 1  module counter #(
 2      WIDTH = 3
 3  )(
 4      input clk_i,
 5      input rst_ni,
 6      input direction_i,
 7      input en_i,
 8      input [WIDTH-1:0] low_limit_i,
 9      input [WIDTH-1:0] high_limit_i,
10      output reg [WIDTH-1:0] count_o,
11      output reg overflow_o,
12      output reg underflow_o
13  );
14
15  always @ (posedge clk_i or negedge rst_ni) begin
16      if(~rst_ni)
17          count_o <= 0;
18      else if(en_i) begin
19          if(overflow_o)
20              count_o <= low_limit_i;
21          else if(underflow_o)
22              count_o <= high_limit_i;
23          else
24              count_o <= (direction_i) ? count_o + 1 : count_o - 1;
25      end
26  end
27
28  assign overflow_o = direction_i && (count_o >= high_limit_i);
29  assign underflow_o = !direction_i && (count_o <= low_limit_i);
30
31  endmodule
32
```