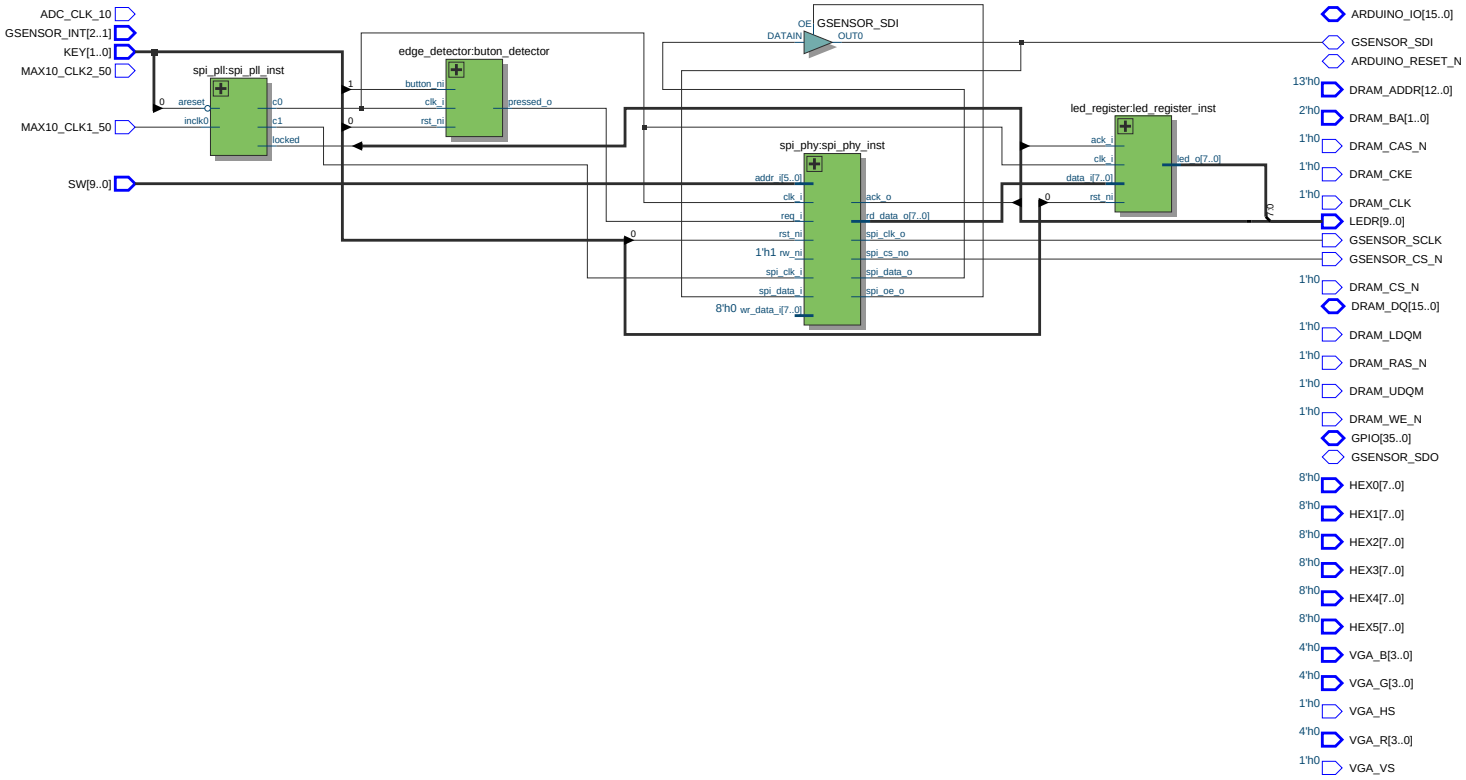
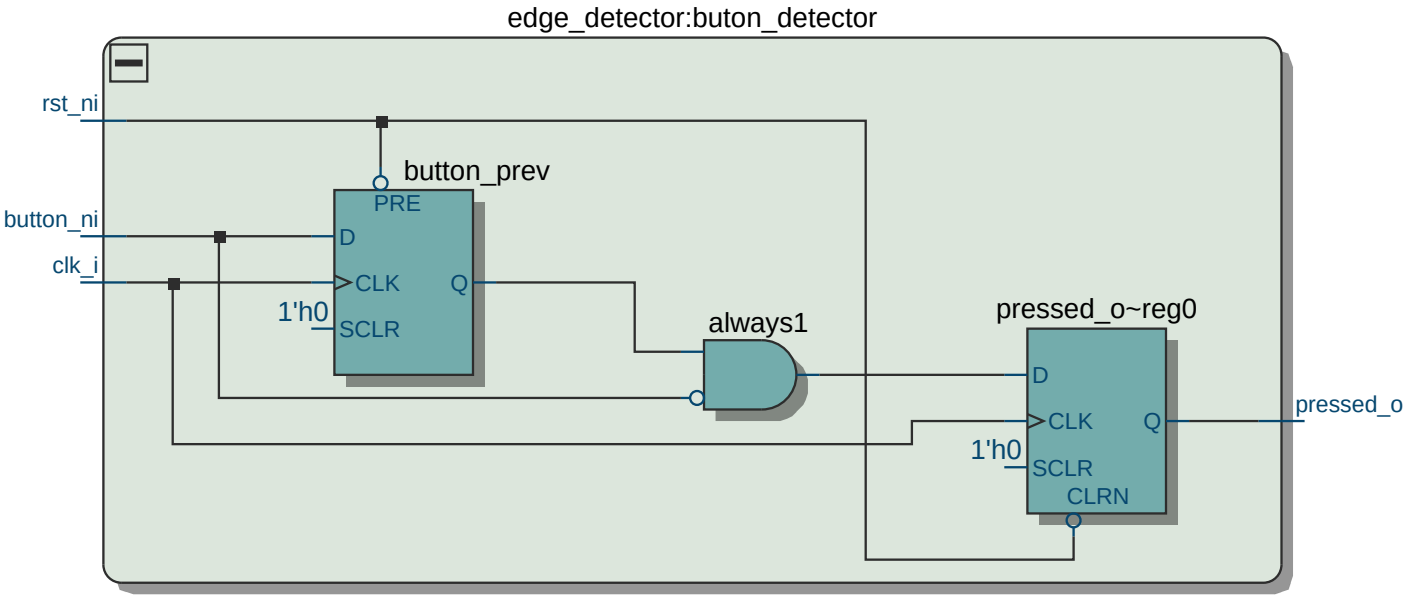
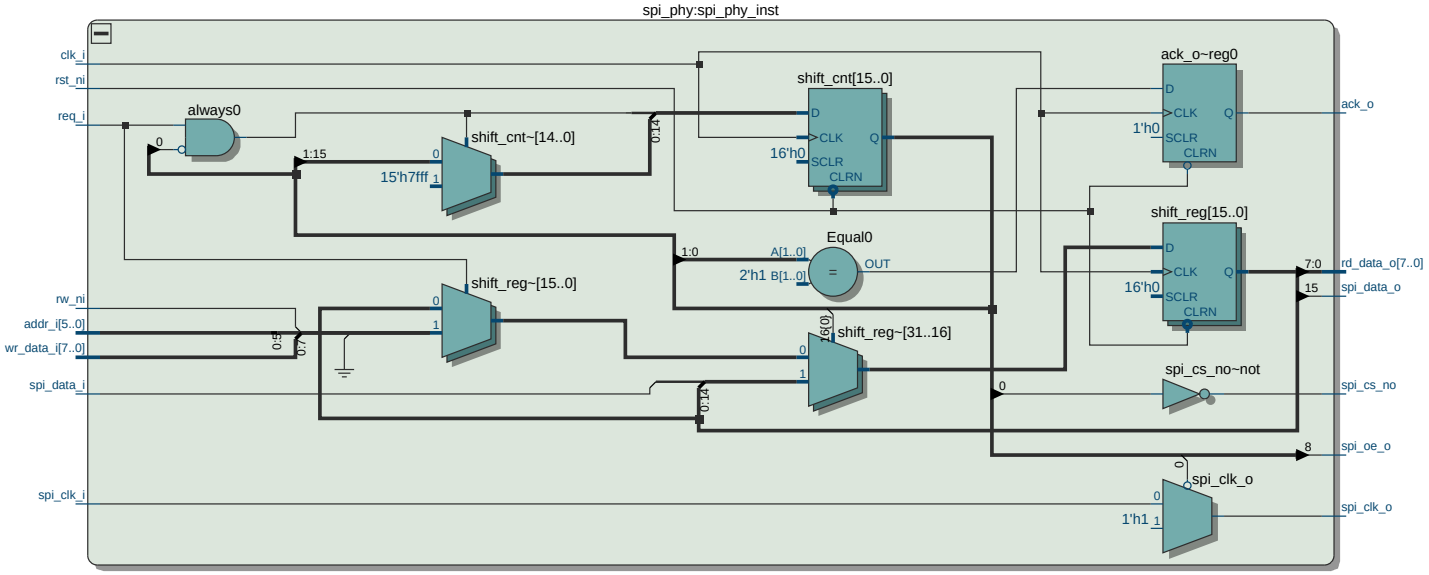
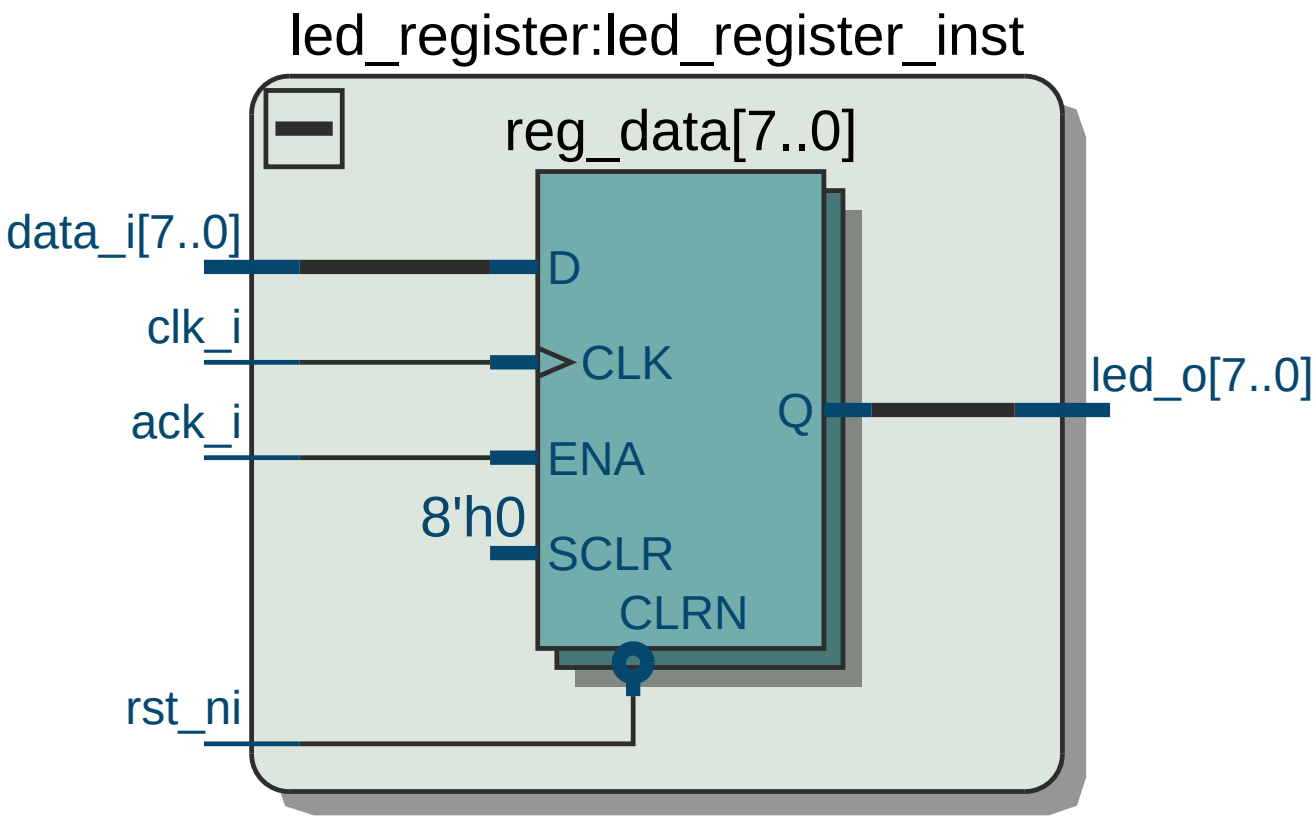
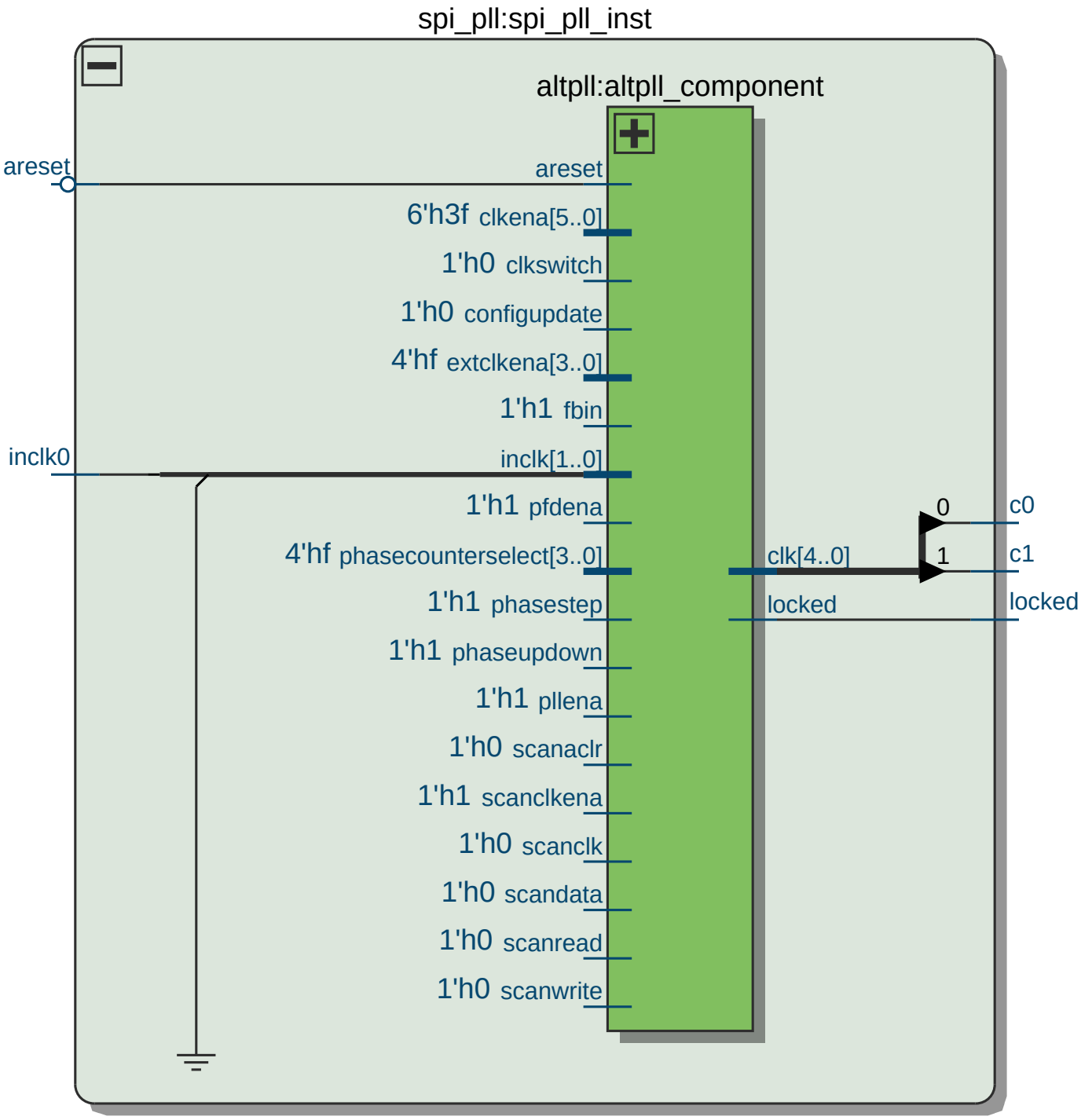
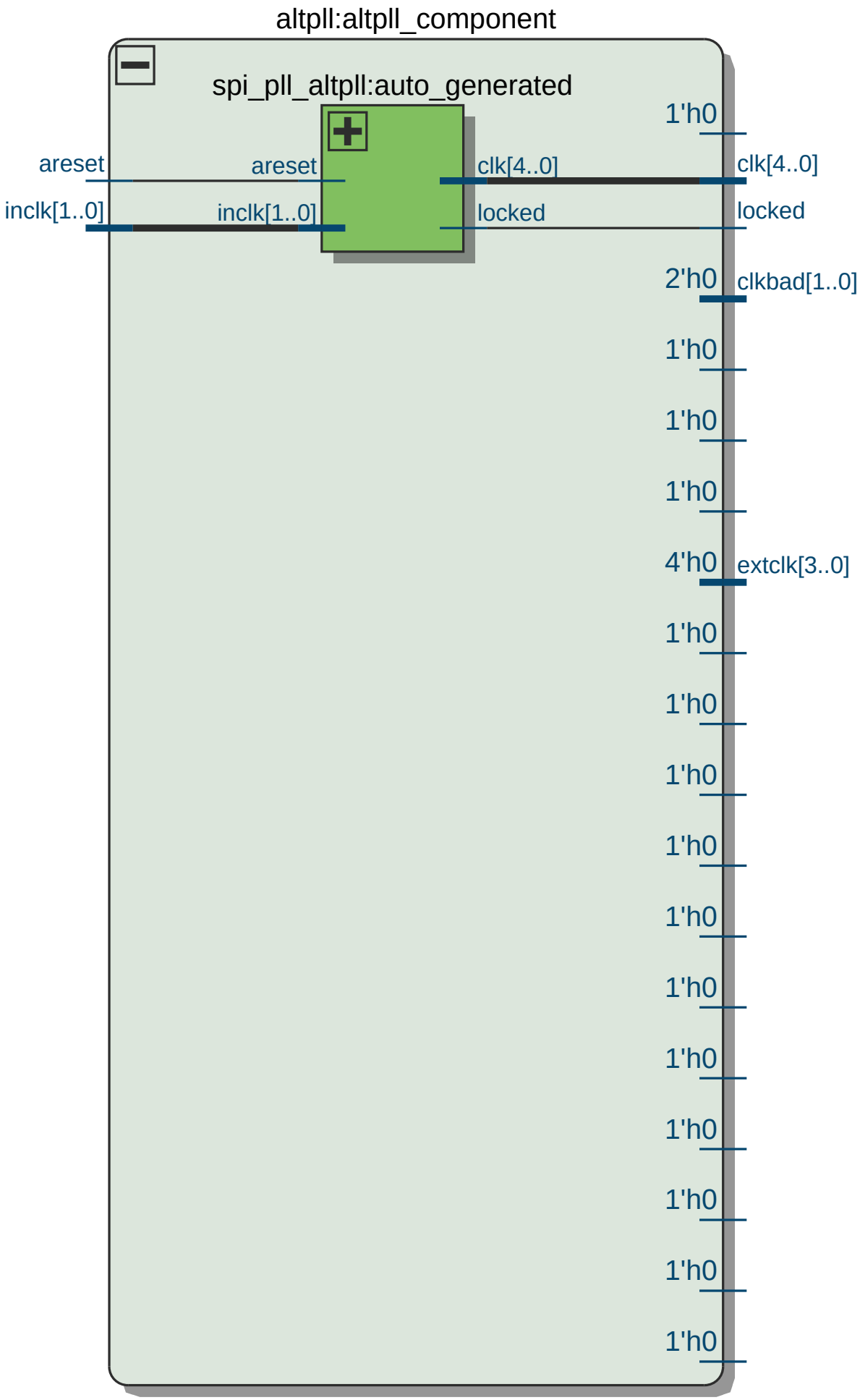

ADC_CLK_10
GSENSOR_INT[2..1]
KEY[1..0]
MAX10_CLK2_50
MAX10_CLK1_50
SW[9..0]
spi_pll:spi_pll_inst
areset
inclk0
c0
c1
locked
edge_detector:buton_detector
button_ni
clk_i
rst_ni
pressed_o
GSENSOR_SDI
OE
DATAIN
OUT0
spi_phy:spi_phy_inst
addr_i[5..0]
clk_i
req_i
rst_ni
rw_ni
spi_clk_i
spi_data_i
wr_data_i[7..0]
ack_o
rd_data_o[7..0]
spi_clk_o
spi_cs_no
spi_data_o
spi_oe_o
led_register:led_register_inst
ack_i
clk_i
data_i[7..0]
rst_ni
led_o[7..0]
ARDUINO_IO[15..0]
GSENSOR_SDI
ARDUINO_RESET_N
DRAM_ADDR[12..0]
DRAM_BA[1..0]
DRAM_CAS_N
DRAM_CKE
DRAM_CLK
LEDR[9..0]
GSENSOR_SCLK
GSENSOR_CS_N
DRAM_CS_N
DRAM_DQ[15..0]
DRAM_LDQM
DRAM_RAS_N
DRAM_UDQM
DRAM_WE_N
GPIO[35..0]
GSENSOR_SDO
HEX0[7..0]
HEX1[7..0]
HEX2[7..0]
HEX3[7..0]
HEX4[7..0]
HEX5[7..0]
VGA_B[3..0]
VGA_G[3..0]
VGA_HS
VGA_R[3..0]
VGA_VS

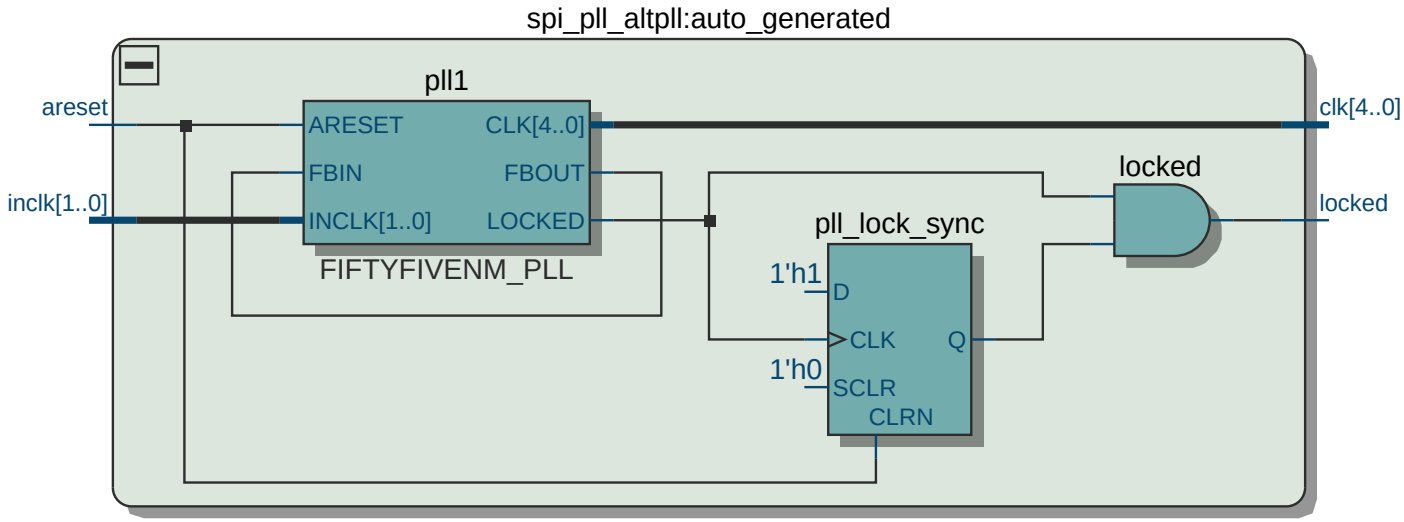

edge_detector:buton_detector
rst_ni
button_ni
clk_i
button_prev
PRE
D
CLK
Q
1'h0
SCLR
always1
pressed_o~reg0
D
CLK
Q
1'h0
SCLR
CLRN
pressed_o


spi_phy:spi_phy_inst
clk_i
rst_ni
req_i
always0
shift_cnt~[14..0]
15'h7fff
shift_cnt[15..0]
16'h0
ack_o~reg0
1'h0
ack_o
shift_reg[15..0]
Equal0
2'h1
OUT
shift_reg~[15..0]
rw_ni
addr_i[5..0]
wr_data_i[7..0]
spi_data_i
shift_reg~[31..16]
spi_cs_no~not
spi_cs_no
rd_data_o[7..0]
spi_data_o
spi_oe_o
spi_clk_o
spi_clk_i
spi_clk_o
1'h1


led_register:led_register_inst
reg_data[7..0]
data_i[7..0]
clk_i
ack_i
8'h0
rst_ni
D
CLK
ENA
SCLR
CLRN
Q
led_o[7..0]


spi_pll:spi_pll_inst
altpll:altpll_component
areset
areset
6'h3f clkena[5..0]
1'h0 clkswitch
1'h0 configupdate
4'hf extclkena[3..0]
1'h1 fbin
inclk0
inclk[1..0]
1'h1 pfdena
4'hf phasecounterselect[3..0]
1'h1 phasestep
1'h1 phaseupdown
1'h1 pllena
1'h0 scanaclr
1'h1 scanclkena
1'h0 scanclk
1'h0 scandata
1'h0 scanread
1'h0 scanwrite
clk[4..0]
locked
0
1
c0
c1
locked


altpll:altpll_component
spi_pll_altpll:auto_generated
areset
inclk[1..0]
areset
inclk[1..0]
clk[4..0]
locked
1'h0
clk[4..0]
locked
2'h0
clkbad[1..0]
1'h0
1'h0
1'h0
4'h0
extclk[3..0]
1'h0
1'h0
1'h0
1'h0
1'h0
1'h0
1'h0
1'h0
1'h0
1'h0
1'h0
1'h0


spi_pll_altpll:auto_generated
pll1
areset
inclk[1..0]
ARESET
FBIN
INCLK[1..0]
CLK[4..0]
FBOUT
LOCKED
FIFTYFIVENM_PLL
pll_lock_sync
1'h1
D
CLK
Q
1'h0
SCLR
CLRN
locked
clk[4..0]
locked

```
// ============================================================================
//   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
131   logic sys_clk;
132   logic spi_clk;
133
134   logic sdo;
135   logic sdo_en;
136
137   logic button_pressed; // button detector button_pressed
138
```

```
139   logic [7:0] data_o; // data out from the SPI module
140
141
142   //=======================================================
143   //  Structural coding
144   //=======================================================
145
146   spi_pll  spi_pll_inst (
147      .areset  ( ~KEY[0]      ),
148      .inclk0  ( MAX10_CLK1_50 ),
149      .c0      ( sys_clk      ),
150      .c1      ( spi_clk      ),
151      .locked  ( LEDR[9]      )
152   );
153
154
155   spi_phy spi_phy_inst (
156      .clk_i         ( sys_clk      ),
157      .spi_clk_i     ( spi_clk      ),
158      .rst_ni        ( KEY[0]       ),
159
160      .req_i         ( button_pressed), // ~KEY[1]
161      .rw_ni         ( 1'b1         ),
162      .addr_i        ( SW[5:0]      ),
163      .wr_data_i     ( 8'b0         ),
164      .ack_o         ( LEDR[8]      ),
165      .rd_data_o     ( data_o       ),
166
167      .spi_cs_no     ( GSENSOR_CS_N ),
168      .spi_clk_o     ( GSENSOR_SCLK ),
169      .spi_data_i    ( GSENSOR_SDI  ),
170      .spi_data_o    ( sdo          ),
171      .spi_oe_o      ( sdo_en       )
172   );
173
174   edge_detector buton_detector(
175      .clk_i         ( sys_clk         ),
176      .rst_ni        ( KEY[0]          ),
177      .button_ni     ( KEY[1]          ),
178      .pressed_o     ( button_pressed  )
179   );
180
181   led_register led_register_inst(
182      .clk_i         ( sys_clk      ),
183      .rst_ni        ( KEY[0]       ),
184      .data_i        ( data_o       ),
185      .ack_i         ( LEDR[8]      ),
186      .led_o         ( LEDR[7:0]    )
187   );
188
189
190   assign GSENSOR_SDI = sdo_en ? sdo : 1'bz;
191   //un modul nou pentru detector de front    facut
192
193   //de implementat un registru care sa retina valoarea citita   FACUT
194
195   endmodule
196
```

```
1   module edge_detector(
2       input wire clk_i,          // ceasul sistemului
3       input wire rst_ni,         // reset
4       input wire button_ni,      // semnalul de la buton
5       output reg pressed_o       // iese 1 DOAR 1 ciclu la apasare
6   );
7
8       reg button_prev;
9
10      always @(posedge clk_i or negedge rst_ni) begin
11          if (~rst_ni) begin
12              button_prev <= 1'b1;
13          end else begin
14              button_prev <= button_ni;
15          end
16      end
17
18      always @(posedge clk_i or negedge rst_ni) begin
19          if (~rst_ni) begin
20              pressed_o <= 1'b0;
21          end else begin
22              // detectam front de coborare: 1 -> 0
23              if (button_prev == 1'b1 && button_ni == 1'b0)
24                  pressed_o <= 1'b1;
25              else
26                  pressed_o <= 1'b0;
27          end
28      end
29
30
31  endmodule
32
```

```
module spi_phy (
    input        rst_ni    ,
    input        clk_i     ,  // max 5MHz for ADXL345 sensor
    input        spi_clk_i ,  // same frequency as clk_i, 220 deg phase offset

    // Parallel request-acknowledge interface (sampled with clk_i)
    input        req_i     ,
    input        rw_ni     ,  // 1=read, 0=write
    input  [5:0] addr_i    ,
    input  [7:0] wr_data_i ,
    output logic ack_o     ,
    output [7:0] rd_data_o ,

    // SPI interface
    output       spi_cs_no ,
    output       spi_clk_o ,
    output       spi_data_o,
    input        spi_data_i,
    output       spi_oe_o     // control 3-state buffer for 3-wire SPI mode
    // spi oe output enable
);

localparam SHREG_WIDTH = 16;

logic [SHREG_WIDTH-1 : 0] shift_reg;
logic [SHREG_WIDTH-1 : 0] shift_cnt;

always @ (posedge clk_i or negedge rst_ni) begin
    if (!rst_ni)
        shift_cnt <= 'b0;
    else if (req_i && spi_cs_no)
        shift_cnt <= {SHREG_WIDTH{1'b1}};
    else
        shift_cnt <= shift_cnt >> 1;
end

assign spi_cs_no = !shift_cnt[0];
assign spi_clk_o = spi_cs_no ? 1'b1 : spi_clk_i;
assign spi_oe_o  = shift_cnt[8];

always @ (posedge clk_i or negedge rst_ni) begin
    if (!rst_ni)
        shift_reg <= 'b0;
    else if (!spi_cs_no)
        shift_reg <= (shift_reg << 1) | spi_data_i;
    else if (req_i)
        shift_reg <= {rw_ni, 1'b0, addr_i, wr_data_i};
end

assign spi_data_o = shift_reg[SHREG_WIDTH-1];

assign rd_data_o = shift_reg[7:0];

always @ (posedge clk_i or negedge rst_ni) begin
    if (!rst_ni)
        ack_o <= 1'b0;
    else
        ack_o <= shift_cnt[1:0] == 2'b01;
end

endmodule

```

```
1   module led_register(
2       input   logic           clk_i,
3       input   logic           rst_ni,
4       input   logic  [7:0]    data_i,
5       input   logic           ack_i,
6       output  logic  [7:0]    led_o
7   );
8
9
10  //registru intern pentru a retine datele de la SPI
11  logic [7:0] reg_data;
12
13  always @( posedge clk_i or negedge rst_ni ) begin
14      if(~rst_ni) begin
15          reg_data <= 8'b0;
16      end else if(ack_i) begin
17          reg_data <= data_i;
18      end
19  end
20
21  assign led_o = reg_data; // iesirea registrului este conectata la LED-uri
22
23
24
25  endmodule
```

```
1  //altpll bandwidth_type="AUTO" CBX_DECLARE_ALL_CONNECTED_PORTS="OFF" clk0_divide_by=50
   clk0_duty_cycle=50 clk0_multiply_by=1 clk0_phase_shift="0" clk1_divide_by=50
   clk1_duty_cycle=50 clk1_multiply_by=1 clk1_phase_shift="611111" compensate_clock="CLK0"
   device_family="MAX 10" inclk0_input_frequency=20000 intended_device_family="MAX 10"
   lpm_hint="CBX_MODULE_PREFIX=spi_pll" operation_mode="normal" pll_type="AUTO"
   port_clk0="PORT_USED" port_clk1="PORT_USED" port_clk2="PORT_UNUSED" port_clk3="PORT_UNUSED"
   port_clk4="PORT_UNUSED" port_clk5="PORT_UNUSED" port_extclk0="PORT_UNUSED"
   port_extclk1="PORT_UNUSED" port_extclk2="PORT_UNUSED" port_extclk3="PORT_UNUSED"
   port_inclk1="PORT_UNUSED" port_phasecounterselect="PORT_UNUSED"
   port_phasedone="PORT_UNUSED" port_scandata="PORT_UNUSED" port_scandataout="PORT_UNUSED"
   self_reset_on_loss_lock="OFF" width_clock=5 areset clk inclk locked CARRY_CHAIN="MANUAL"
   CARRY_CHAIN_LENGTH=48
2  //VERSION_BEGIN 22.1 cbx_altclkbuf 2022:10:25:15:36:36:SC cbx_altiobuf_bidir
   2022:10:25:15:36:38:SC cbx_altiobuf_in 2022:10:25:15:36:38:SC cbx_altiobuf_out
   2022:10:25:15:36:38:SC cbx_altpll 2022:10:25:15:36:38:SC cbx_cycloneii
   2022:10:25:15:36:38:SC cbx_lpm_add_sub 2022:10:25:15:36:38:SC cbx_lpm_compare
   2022:10:25:15:36:38:SC cbx_lpm_counter 2022:10:25:15:36:38:SC cbx_lpm_decode
   2022:10:25:15:36:36:SC cbx_lpm_mux 2022:10:25:15:36:38:SC cbx_mgl 2022:10:25:15:36:55:SC
   cbx_nadder 2022:10:25:15:36:38:SC cbx_stratix 2022:10:25:15:36:38:SC cbx_stratixii
   2022:10:25:15:36:38:SC cbx_stratixiii 2022:10:25:15:36:38:SC cbx_stratixv
   2022:10:25:15:36:38:SC cbx_util_mgl 2022:10:25:15:36:38:SC  VERSION_END
3  //CBXI_INSTANCE_NAME="DE10_LITE_Golden_Top_spi_pll_spi_pll_inst_altpll_altpll_component"
4  // synthesis VERILOG_INPUT_VERSION VERILOG_2001
5  // altera message_off 10463
6
7
8
9  // Copyright (C) 2022  Intel Corporation. All rights reserved.
10 //  Your use of Intel Corporation's design tools, logic functions
11 //  and other software and tools, and any partner logic
12 //  functions, and any output files from any of the foregoing
13 //  (including device programming or simulation files), and any
14 //  associated documentation or information are expressly subject
15 //  to the terms and conditions of the Intel Program License
16 //  Subscription Agreement, the Intel Quartus Prime License Agreement,
17 //  the Intel FPGA IP License Agreement, or other applicable license
18 //  agreement, including, without limitation, that your use is for
19 //  the sole purpose of programming logic devices manufactured by
20 //  Intel and sold by Intel or its authorized distributors.  Please
21 //  refer to the applicable agreement for further details, at
22 //  https://fpgasoftware.intel.com/eula.
23
24
25
26 //synthesis_resources = fiftyfivenm_pll 1 reg 1
27 //synopsys translate_off
28 `timescale 1 ps / 1 ps
29 //synopsys translate_on
30 (* ALTERA_ATTRIBUTE = {"SUPPRESS_DA_RULE_INTERNAL=C104;SUPPRESS_DA_RULE_INTERNAL=R101"} *)
31 module  spi_pll_altpll
32    (
33    areset,
34    clk,
35    inclk,
36    locked) /* synthesis synthesis_clearbox=1 */;
37    input   areset;
38    output   [4:0]  clk;
39    input   [1:0]  inclk;
40    output   locked;
41 `ifndef ALTERA_RESERVED_QIS
42 // synopsys translate_off
43 `endif
44    tri0   areset;
45    tri0   [1:0]  inclk;
46 `ifndef ALTERA_RESERVED_QIS
47 // synopsys translate_on
48 `endif
49
50    reg   pll_lock_sync;
```

```
51      wire  [4:0]   wire_pll1_clk;
52      wire  wire_pll1_fbout;
53      wire  wire_pll1_locked;
54
55      // synopsys translate_off
56      initial
57         pll_lock_sync = 0;
58      // synopsys translate_on
59      always @ ( posedge wire_pll1_locked or  posedge areset)
60         if (areset == 1'b1) pll_lock_sync <= 1'b0;
61         else  pll_lock_sync <= 1'b1;
62      fiftyfivenm_pll   pll1
63      (
64      .activeclock(),
65      .areset(areset),
66      .clk(wire_pll1_clk),
67      .clkbad(),
68      .fbin(wire_pll1_fbout),
69      .fbout(wire_pll1_fbout),
70      .inclk(inclk),
71      .locked(wire_pll1_locked),
72      .phasedone(),
73      .scandataout(),
74      .scandone(),
75      .vcooverrange(),
76      .vcounderrange()
77      `ifndef FORMAL_VERIFICATION
78      // synopsys translate_off
79      `endif
80      ,
81      .clkswitch(1'b0),
82      .configupdate(1'b0),
83      .pfdena(1'b1),
84      .phasecounterselect({3{1'b0}}),
85      .phasestep(1'b0),
86      .phaseupdown(1'b0),
87      .scanclk(1'b0),
88      .scanclkena(1'b1),
89      .scandata(1'b0)
90      `ifndef FORMAL_VERIFICATION
91      // synopsys translate_on
92      `endif
93      );
94      defparam
95         pll1.bandwidth_type = "auto",
96         pll1.clk0_divide_by = 50,
97         pll1.clk0_duty_cycle = 50,
98         pll1.clk0_multiply_by = 1,
99         pll1.clk0_phase_shift = "0",
100        pll1.clk1_divide_by = 50,
101        pll1.clk1_duty_cycle = 50,
102        pll1.clk1_multiply_by = 1,
103        pll1.clk1_phase_shift = "611111",
104        pll1.compensate_clock = "clk0",
105        pll1.inclk0_input_frequency = 20000,
106        pll1.operation_mode = "normal",
107        pll1.pll_type = "auto",
108        pll1.self_reset_on_loss_lock = "off",
109        pll1.lpm_type = "fiftyfivenm_pll";
110     assign
111        clk = {wire_pll1_clk[4:0]},
112        locked = (wire_pll1_locked & pll_lock_sync);
113  endmodule //spi_pll_altpll
114  //VALID FILE
115
```