Final

- **DE10_LITE_Golden_Top:**

```
//
=============================================================================
=
//   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
    output          [7:0]       HEX3,
`endif
`ifdef ENABLE_HEX4
    output          [7:0]       HEX4,
`endif
`ifdef ENABLE_HEX5
    output          [7:0]       HEX5,
`endif

    ////////////// KEY: 3.3 V SCHMITT TRIGGER //////////
`ifdef ENABLE_KEY
    input           [1:0]       KEY,
`endif

    ////////////// LED: 3.3-V LVTTL //////////
`ifdef ENABLE_LED
    output          [9:0]       LEDR,
`endif

    ////////////// SW: 3.3-V LVTTL //////////
`ifdef ENABLE_SW
    input           [9:0]       SW,
`endif

    ////////////// VGA: 3.3-V LVTTL //////////
`ifdef ENABLE_VGA
    output          [3:0]       VGA_B,
    output          [3:0]       VGA_G,
    output                      VGA_HS,
    output          [3:0]       VGA_R,
    output                      VGA_VS,
`endif

    ////////////// Accelerometer: 3.3-V LVTTL //////////
`ifdef ENABLE_ACCELEROMETER
    output                      GSENSOR_CS_N,
    input           [2:1]       GSENSOR_INT,
    output                      GSENSOR_SCLK,
    inout                       GSENSOR_SDI,
    inout                       GSENSOR_SDO,
`endif

    ////////////// Arduino: 3.3-V LVTTL //////////
`ifdef ENABLE_ARDUINO
```

```
    inout           [15:0]      ARDUINO_IO,
    inout                       ARDUINO_RESET_N,
`endif

    ////////////// GPIO, GPIO connect to GPIO Default: 3.3-V LVTTL //////////
`ifdef ENABLE_GPIO
    inout           [35:0]      GPIO
`endif
);

localparam DIGITS = 6;
localparam WIDTH = 20; //$clog2(10**DIGITS - 1);

//=======================================================
//  REG/WIRE declarations
//=======================================================

logic [DIGITS-1:0][3:0] bcd;
logic [DIGITS-1:0][7:0] hex;
logic [DIGITS-1:0] blank;
logic [WIDTH-1:0 ] result;

//=======================================================
//  Structural coding
//=======================================================

assign {HEX5, HEX4, HEX3, HEX2, HEX1, HEX0} = hex;

acumulator #(
    .WIDTH ( WIDTH )
) acumulator_inst (
    .clk_i    ( MAX10_CLK1_50 ) ,
    .rst_ni   ( KEY[1] ) ,
    .add_ni   ( KEY[0] ) ,
    .number_i ( SW ) ,
    .result_o ( result )
);

bin_to_bcd #(
    .DIGITS ( DIGITS ),
    .WIDTH  ( WIDTH  )
)

bin_to_bcd_inst(
    .bin_i ( result  ),
    .bcd_o ( bcd )
);

genvar i;
  generate
    for (i=0; i<DIGITS; i++) begin : GEN_7SEG
        single_digit_7seg_driver
        digit_7seg_driver_inst(
            .bcd_i   ( bcd[i]     ),
            .seg7_o  ( hex[i]     ),
            .blank_i ( (i==0) ? 1'b0 : (i==DIGITS-1) ? 1'b1 : blank[i+1] ),
            .blank_o ( blank[i]   )
        );
    end
  endgenerate
endmodule
```

- **bin_to_bcd:**

```
module bin_to_bcd #(
    DIGITS = 2,
    WIDTH = $clog2(10**DIGITS - 1)
) (
    input[WIDTH-1:0] bin_i,
    output reg [DIGITS-1:0][3:0] bcd_o
);


always @ (*) begin
    integer i;
    reg [WIDTH - 1:0] tmp;
    tmp = bin_i;
    for (i=0; i<DIGITS; i=i+1) begin
        bcd_o[i] = tmp % 10;
        tmp = tmp / 10;
    end
end

endmodule
```

- **single_digit_7seg_driver:**

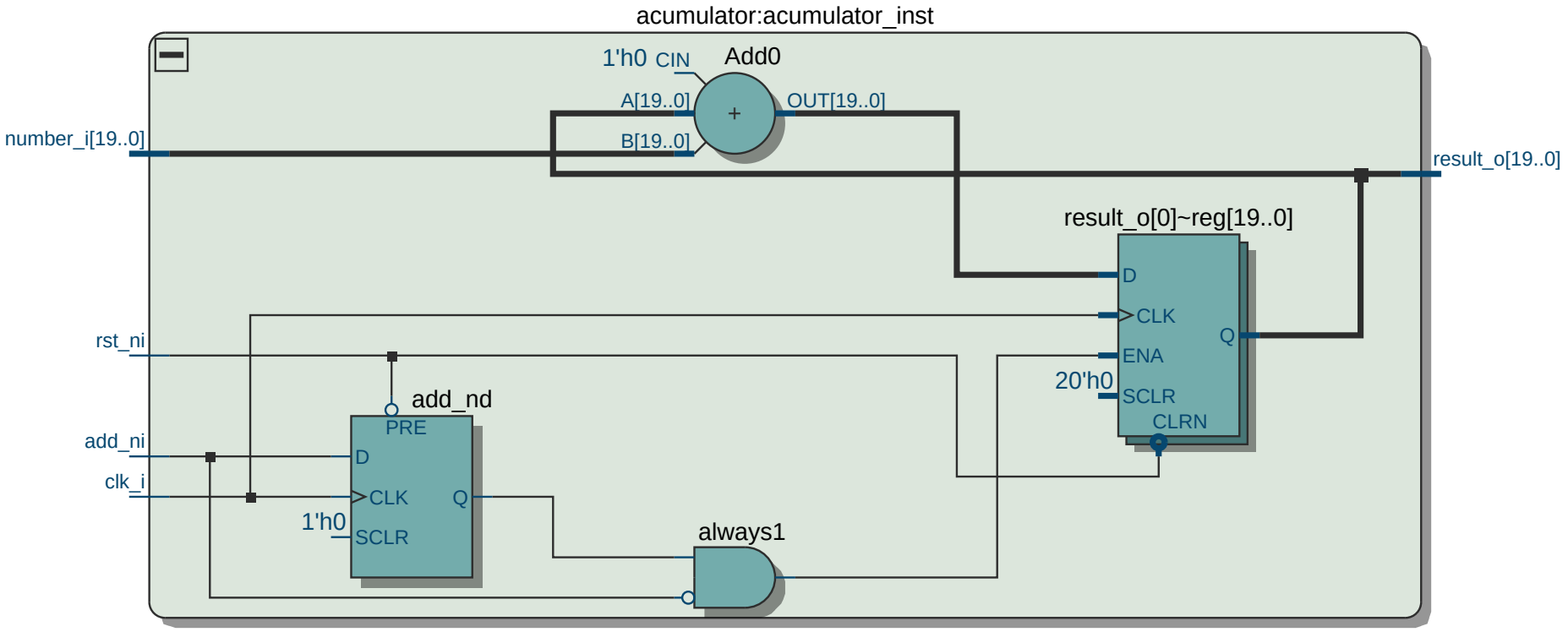
```
module single_digit_7seg_driver (
    input           [3:0] bcd_i,
    output reg  [7:0] seg7_o,
    input                   blank_i,
    output              blank_o
);


always @ (*) begin
    case (bcd_i)
        0: seg7_o = 8'b00111111;
        1: seg7_o = 8'b00000110;
        2: seg7_o = 8'b01011011;
        3: seg7_o = 8'b01001111;
        4: seg7_o = 8'b01100110;
        5: seg7_o = 8'b01101101;
        6: seg7_o = 8'b01111101;
        7: seg7_o = 8'b00000111;
        8: seg7_o = 8'b01111111;
        9: seg7_o = 8'b01101111;
        default: seg7_o = 8'hxx;
    endcase
    seg7_o = {8{!blank_o}} & seg7_o;
    seg7_o = ~seg7_o;
end

assign blank_o = !(|bcd_i) && blank_i;

endmodule
```

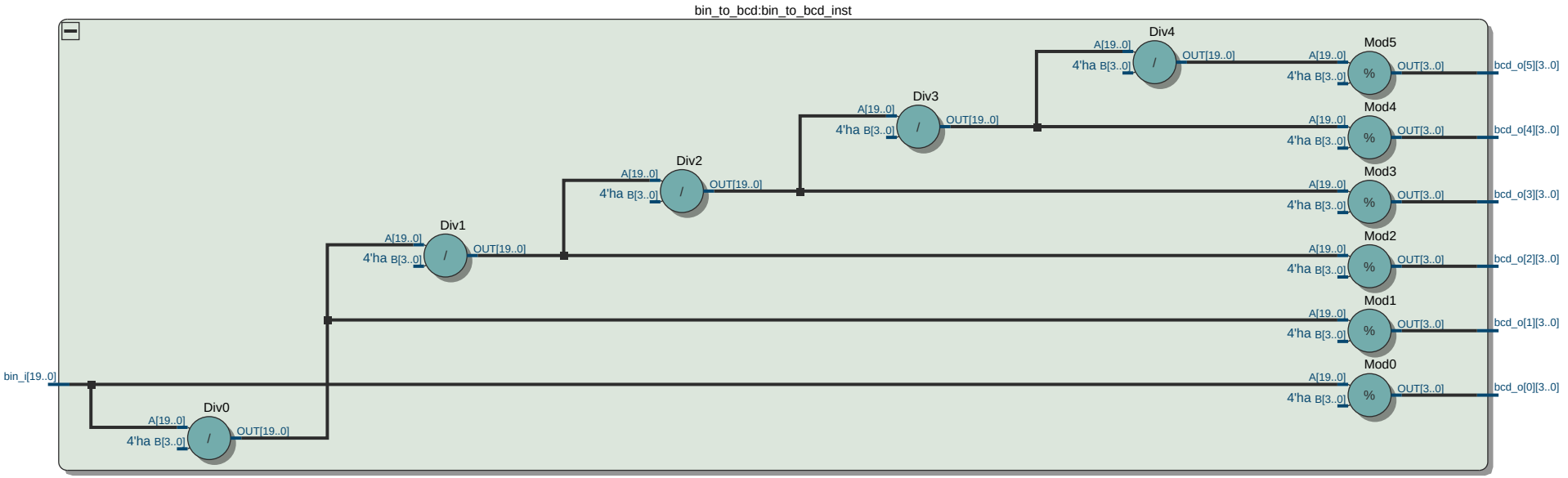
- **acumulator:**

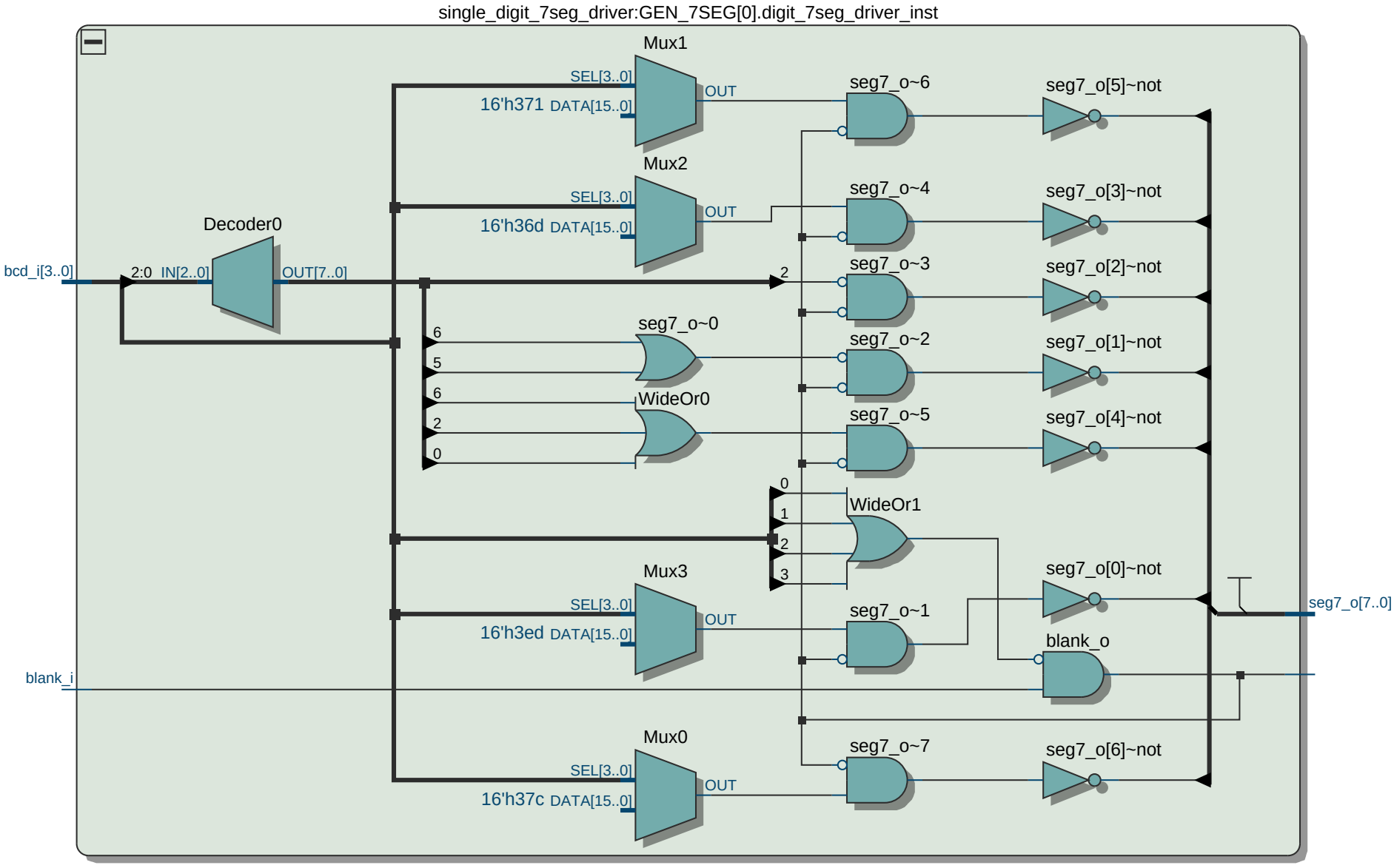
```
module acumulator #(
```

```
    WIDTH = 20
) (
    input                   clk_i   ,
    input                   rst_ni  ,
    input                   add_ni  ,
    input  [WIDTH-1:0]  number_i,
    output [WIDTH-1:0]  result_o
);

logic add_nd;

always @ (posedge clk_i or negedge rst_ni) begin
    if (!rst_ni)
        add_nd <= 1'b1;
    else
        add_nd <= add_ni;
end

always @ (posedge clk_i or negedge rst_ni) begin
    if (!rst_ni)
        result_o <= 'b0;
    else if (add_nd && !add_ni)
        result_o <= result_o + number_i;
end

endmodule
```


KEY[1..0]
MAX10_CLK1_50
SW[9..0]
0
1
acumulator:acumulator_inst
add_ni
clk_i
number_i[19..0]
rst_ni
result_o[19..0]
bin_to_bcd:bin_to_bcd_inst
bin_i[19..0]
bcd_o[0][3..0]
bcd_o[1][3..0]
bcd_o[2][3..0]
bcd_o[3][3..0]
bcd_o[4][3..0]
bcd_o[5][3..0]
single_digit_7seg_driver:GEN_7SEG[0].digit_7seg_driver_inst
bcd_i[3..0]
1'h0
blank_i
seg7_o[7..0]
HEX0[7..0]
13'h0
DRAM_ADDR[12..0]
2'h0
DRAM_BA[1..0]
single_digit_7seg_driver:GEN_7SEG[1].digit_7seg_driver_inst
HEX1[7..0]
single_digit_7seg_driver:GEN_7SEG[2].digit_7seg_driver_inst
blank_o
HEX2[7..0]
single_digit_7seg_driver:GEN_7SEG[3].digit_7seg_driver_inst
HEX3[7..0]
single_digit_7seg_driver:GEN_7SEG[4].digit_7seg_driver_inst
HEX4[7..0]
single_digit_7seg_driver:GEN_7SEG[5].digit_7seg_driver_inst
1'h1
HEX5[7..0]
DRAM_DQ[15..0]


acumulator:acumulator_inst
1'h0
CIN
Add0
A[19..0]
B[19..0]
OUT[19..0]
+
number_i[19..0]
result_o[19..0]
result_o[0]~reg[19..0]
D
CLK
ENA
Q
20'h0
SCLR
CLRN
rst_ni
add_nd
PRE
add_ni
clk_i
1'h0
always1


bin_to_bcd:bin_to_bcd_inst
bin_i[19..0]
Div0
A[19..0]
4'ha B[3..0]
OUT[19..0]
Div1
Div2
Div3
Div4
Mod0
Mod1
Mod2
Mod3
Mod4
Mod5
%
OUT[3..0]
bcd_o[0][3..0]
bcd_o[1][3..0]
bcd_o[2][3..0]
bcd_o[3][3..0]
bcd_o[4][3..0]
bcd_o[5][3..0]


single_digit_7seg_driver:GEN_7SEG[0].digit_7seg_driver_inst
Mux1
SEL[3..0]
16'h371 DATA[15..0]
OUT
Mux2
SEL[3..0]
16'h36d DATA[15..0]
OUT
Decoder0
bcd_i[3..0]
2:0 IN[2..0]
OUT[7..0]
2
6
5
6
2
0
seg7_o~0
WideOr0
0
1
2
3
WideOr1
Mux3
SEL[3..0]
16'h3ed DATA[15..0]
OUT
blank_i
Mux0
SEL[3..0]
16'h37c DATA[15..0]
OUT
seg7_o~6
seg7_o~4
seg7_o~3
seg7_o~2
seg7_o~5
seg7_o~1
seg7_o~7
seg7_o[5]~not
seg7_o[3]~not
seg7_o[2]~not
seg7_o[1]~not
seg7_o[4]~not
seg7_o[0]~not
blank_o
seg7_o[6]~not
seg7_o[7..0]