ELEX 7660: Lab 2

Matrix Keypad Decoder

Nicholas Huttemann	25/01/2019	Set: T

Table of Contents

Table of Figures

1 Modules

1.1 Top level lab2.sv

```
// ELEX 7660 ELEX 7660 201710 Lab2
// 4x4 keypad decoder and LED display
// Ed.Casas 2017-1-11

module lab2 ( output logic [3:0] kpc,  // column select, active-low
              (* altera_attribute = "-name WEAK_PULL_UP_RESISTOR ON" *)
              input logic  [3:0] kpr,  // rows, active-low w/ pull-ups
              output logic [7:0] leds, // active-low LED segments
              output logic [3:0] ct,   // " digit enables
              input logic  reset_n, CLOCK_50 ) ;

   logic clk ;                  // 2kHz clock for keypad scanning
   logic kphit ;                // a key is pressed
   logic [3:0] num ;            // value of pressed key

   assign ct = { {3{1'b0}}, kphit } ;
   pll pll0 ( .inclk0(CLOCK_50), .c0(clk) ) ;

   colseq    colseq_0 (.*) ;
   kpdecode  kpdecode_0  (.*);
   decode7   decode7_0 (.*) ;

endmodule

// megafunction wizard: %ALTPLL%
// ...
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
// ...

module pll ( inclk0, c0);

        input     inclk0;
        output    c0;

        wire [0:0] sub_wire2 = 1'h0;
        wire [4:0] sub_wire3;
        wire  sub_wire0 = inclk0;
        wire [1:0] sub_wire1 = {sub_wire2, sub_wire0};
        wire [0:0] sub_wire4 = sub_wire3[0:0];
        wire  c0 = sub_wire4;

        altpll altpll_component ( .inclk (sub_wire1), .clk
          (sub_wire3), .activeclock (), .areset (1'b0), .clkbad
          (), .clkena ({6{1'b1}}), .clkloss (), .clkswitch
          (1'b0), .configupdate (1'b0), .enable0 (), .enable1 (),
          .extclk (), .extclkena ({4{1'b1}}), .fbin (1'b1),
          .fbmimicbidir (), .fbout (), .fref (), .icdrclk (),
          .locked (), .pfdena (1'b1), .phasecounterselect
          ({4{1'b1}}), .phasedone (), .phasestep (1'b1),
          .phaseupdown (1'b1), .pllena (1'b1), .scanaclr (1'b0),
          .scanclk (1'b0), .scanclkena (1'b1), .scandata (1'b0),
          .scandataout (), .scandone (), .scanread (1'b0),
          .scanwrite (1'b0), .sclkout0 (), .sclkout1 (),
          .vcooverrange (), .vcounderrange ());
```

```
        defparam
                altpll_component.bandwidth_type = "AUTO",
                altpll_component.clk0_divide_by = 25000,
                altpll_component.clk0_duty_cycle = 50,
                altpll_component.clk0_multiply_by = 1,
                altpll_component.clk0_phase_shift = "0",
                altpll_component.compensate_clock = "CLK0",
                altpll_component.inclk0_input_frequency = 20000,
                altpll_component.intended_device_family = "Cyclone IV E",
                altpll_component.lpm_hint = "CBX_MODULE_PREFIX=lab1clk",
                altpll_component.lpm_type = "altpll",
                altpll_component.operation_mode = "NORMAL",
                altpll_component.pll_type = "AUTO",
                altpll_component.port_activeclock = "PORT_UNUSED",
                altpll_component.port_areset = "PORT_UNUSED",
                altpll_component.port_clkbad0 = "PORT_UNUSED",
                altpll_component.port_clkbad1 = "PORT_UNUSED",
                altpll_component.port_clkloss = "PORT_UNUSED",
                altpll_component.port_clkswitch = "PORT_UNUSED",
                altpll_component.port_configupdate = "PORT_UNUSED",
                altpll_component.port_fbin = "PORT_UNUSED",
                altpll_component.port_inclk0 = "PORT_USED",
                altpll_component.port_inclk1 = "PORT_UNUSED",
                altpll_component.port_locked = "PORT_UNUSED",
                altpll_component.port_pfdena = "PORT_UNUSED",
                altpll_component.port_phasecounterselect = "PORT_UNUSED",
                altpll_component.port_phasedone = "PORT_UNUSED",
                altpll_component.port_phasestep = "PORT_UNUSED",
                altpll_component.port_phaseupdown = "PORT_UNUSED",
                altpll_component.port_pllena = "PORT_UNUSED",
                altpll_component.port_scanaclr = "PORT_UNUSED",
                altpll_component.port_scanclk = "PORT_UNUSED",
                altpll_component.port_scanclkena = "PORT_UNUSED",
                altpll_component.port_scandata = "PORT_UNUSED",
                altpll_component.port_scandataout = "PORT_UNUSED",
                altpll_component.port_scandone = "PORT_UNUSED",
                altpll_component.port_scanread = "PORT_UNUSED",
                altpll_component.port_scanwrite = "PORT_UNUSED",
                altpll_component.port_clk0 = "PORT_USED",
                altpll_component.port_clk1 = "PORT_UNUSED",
                altpll_component.port_clk2 = "PORT_UNUSED",
                altpll_component.port_clk3 = "PORT_UNUSED",
                altpll_component.port_clk4 = "PORT_UNUSED",
                altpll_component.port_clk5 = "PORT_UNUSED",
                altpll_component.port_clkena0 = "PORT_UNUSED",
                altpll_component.port_clkena1 = "PORT_UNUSED",
                altpll_component.port_clkena2 = "PORT_UNUSED",
                altpll_component.port_clkena3 = "PORT_UNUSED",
                altpll_component.port_clkena4 = "PORT_UNUSED",
                altpll_component.port_clkena5 = "PORT_UNUSED",
                altpll_component.port_extclk0 = "PORT_UNUSED",
                altpll_component.port_extclk1 = "PORT_UNUSED",
                altpll_component.port_extclk2 = "PORT_UNUSED",
                altpll_component.port_extclk3 = "PORT_UNUSED",
                altpll_component.width_clock = 5;


endmodule
```

1.2 Colseq

```
// colseq.sv - ELEX 7660 - Sequentially pulls each column of the keypad low until a
matching keypress is detected.
// Nicholas Huttemann 2018-01-22

module colseq (input logic  [3:0] kpr ,
               input logic  clk, reset_n,
               output logic [3:0] kpc = 'b0111);

// The four states corresponding to each column
int states [4] = '{'b0111, 'b1011, 'b1101, 'b1110};
logic [1:0] state = 0;
logic hold;

always_comb
      if (kpr == 'b1111)               // If there was a keypress, hold
            hold <= 0;
      else
            hold <= 1;

always_ff @ (posedge clk, negedge reset_n) begin
      if (~reset_n)
            kpc <= 'b0111;
      else
            if (~hold) begin           // If not holding, cycle through the columns
                  kpc = states[state];
                  if (state == 3)
                        state = 0;
                  else
                        state++;
            end
end
endmodule
```

1.3 Kpdecode

```
// kpdecode.sv - ELEX 7660 - Given the detected row ('kpr') and column ('kpc')
location of a press on the keypad,
// set 'num' to the decoded key value and drive the 7-segment with 'kphit'.
// Nicholas Huttemann 2018-01-25

module kpdecode (input logic  [3:0] kpc,
                 input logic  [3:0] kpr,
                 output logic [3:0] num,
                 output logic kphit);

// Define a virtual keypad as a 2D array. Changed from int to logic.
logic [3:0] keypad [4][4] = '{'{1,2,3,10},
                              '{4,5,6,11},
                              '{7,8,9,12},
                              '{14,0,15,13}};
logic [1:0] row;
logic [1:0] col;
always_comb begin
```

```
      if (kpr != 'b1111) begin  // A key has been pressed, determine its value
            kphit = 1;
            case (kpc)                    // Which column was detected?
                  'b0111 : col = 0;
                  'b1011 : col = 1;
                  'b1101 : col = 2;
                  'b1110 : col = 3;
                 default: col = 0;
            endcase
            case (kpr)                    // Which row was detected?
                  'b0111 : row = 0;
                  'b1011 : row = 1;
                  'b1101 : row = 2;
                  'b1110 : row = 3;
                 default: row = 0;
            endcase

      end
      else begin                          // No key was pressed
            kphit = 0;
            row = 0;
            col = 0;
      end

      num = keypad[row][col];             // The decoded value is located in
'keypad' at the detected row and column index.

end
endmodule
```

1.4 Decode7

```
// decode7.sv - ELEX 7660 – Converts ‘num’, an integer 0 – 15, into an 8-bit vector
‘leds’ that will light up corresponding segments on a 7-segment LED.
// Nicholas Huttemann 2018-01-22
module decode7 ( input logic [3:0] num,
                 output logic [7:0] leds);

always_comb
      case(num)
            0 : leds = 8'b11000000;
            1 : leds = 8'b11111001;
            2 : leds = 8'b10100100;
            3 : leds = 8'b10110000;
            4 : leds = 8'b10011001;
            5 : leds = 8'b10010010;
            6 : leds = 8'b10000010;
            7 : leds = 8'b11111000;
            8 : leds = 8'b10000000;
            9 : leds = 8'b10010000;
          4'ha: leds = 8'b10001000;
          4'hb: leds = 8'b10000011;
          4'hc: leds = 8'b11000110;
          4'hd: leds = 8'b10100001;
          4'he: leds = 8'b10000110;
          4'hf: leds = 8'b10001110;
       default: leds = 8'b11000000;
      endcase
endmodule
```

1.5 Simulation Results

```
VSIM 3> run -all
# on reset kpc = 0111 (should be 0111)
# PASS: key  0 => num  1 (should be  1)
# PASS: num  1 => led f9 (should be f9)
# PASS: key  1 => num  2 (should be  2)
# PASS: num  2 => led a4 (should be a4)
# PASS: key  2 => num  3 (should be  3)
# PASS: num  3 => led b0 (should be b0)
# PASS: key  3 => num  a (should be  a)
# PASS: num 10 => led 88 (should be 88)
# PASS: key  4 => num  4 (should be  4)
# PASS: num  4 => led 99 (should be 99)
# PASS: key  5 => num  5 (should be  5)
# PASS: num  5 => led 92 (should be 92)
# PASS: key  6 => num  6 (should be  6)
# PASS: num  6 => led 82 (should be 82)
# PASS: key  7 => num  b (should be  b)
# PASS: num 11 => led 83 (should be 83)
# PASS: key  8 => num  7 (should be  7)
# PASS: num  7 => led f8 (should be f8)
# PASS: key  9 => num  8 (should be  8)
# PASS: num  8 => led 80 (should be 80)
# PASS: key 10 => num  9 (should be  9)
# PASS: num  9 => led 90 (should be 90)
# PASS: key 11 => num  c (should be  c)
# PASS: num 12 => led c6 (should be c6)
# PASS: key 12 => num  e (should be  e)
# PASS: num 14 => led 86 (should be 86)
# PASS: key 13 => num  0 (should be  0)
# PASS: num  0 => led c0 (should be c0)
# PASS: key 14 => num  f (should be  f)
# PASS: num 15 => led 8e (should be 8e)
# PASS: key 15 => num  d (should be  d)
# PASS: num 13 => led a1 (should be a1)
# ** Note: $stop    : C:/Users/A00920439/Desktop/Lab2/lab2_tb.sv(110)
#    Time: 113500 ns  Iteration: 1  Instance: /lab2_tb
# Break in Module lab2_tb at C:/Users/A00920439/Desktop/Lab2/lab2_tb.sv line 110
```

Figure 1: Simulation Results

2 RTL Diagrams

2.1 Top Level

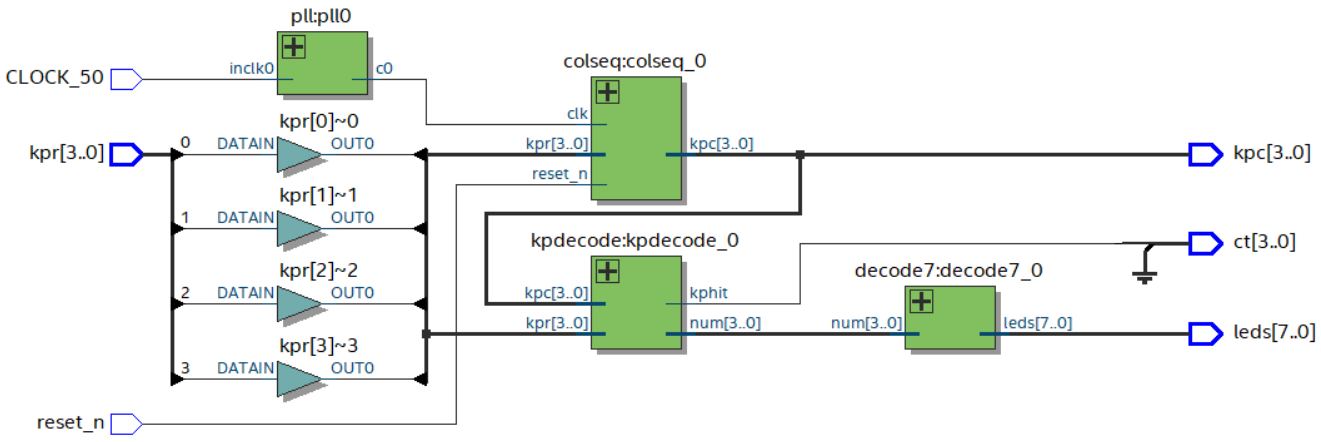

Figure 2: Top level diagram lab2.sv

2.2 Colseq

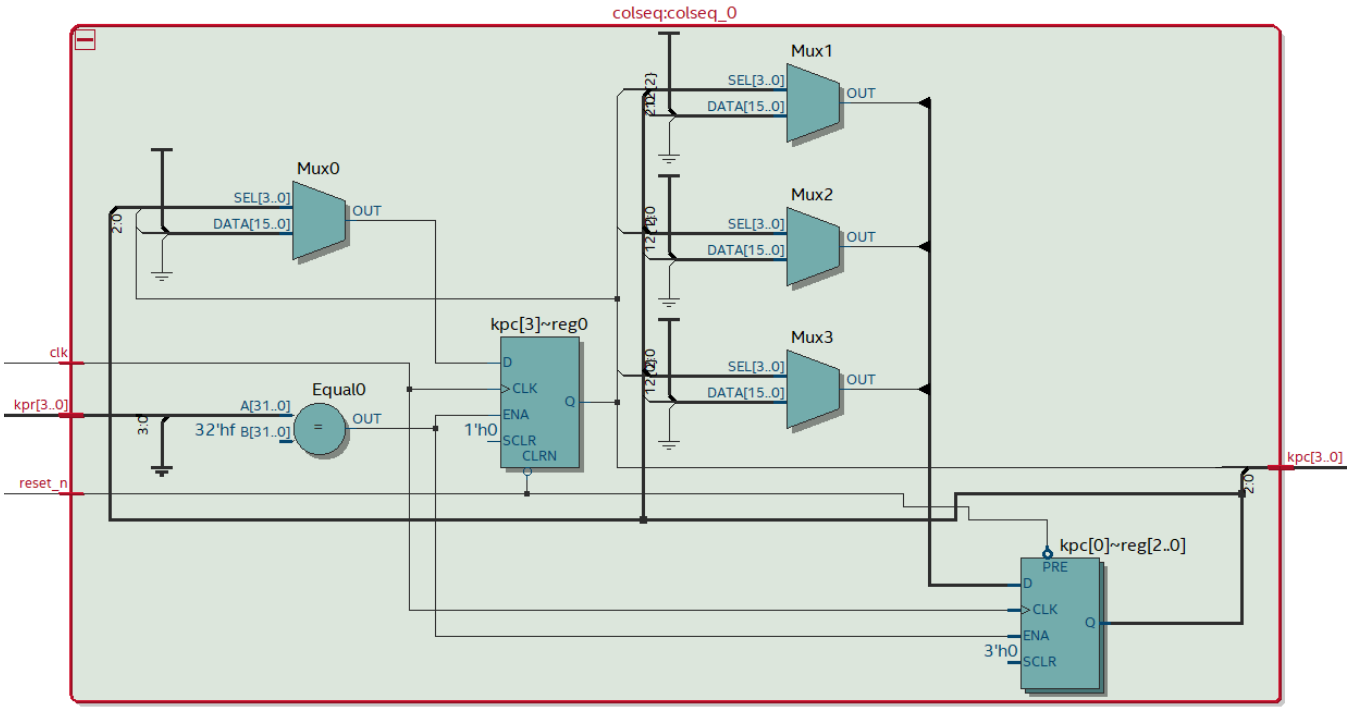

Figure 3: colseq.sv diagram

2.3 Kpdecode

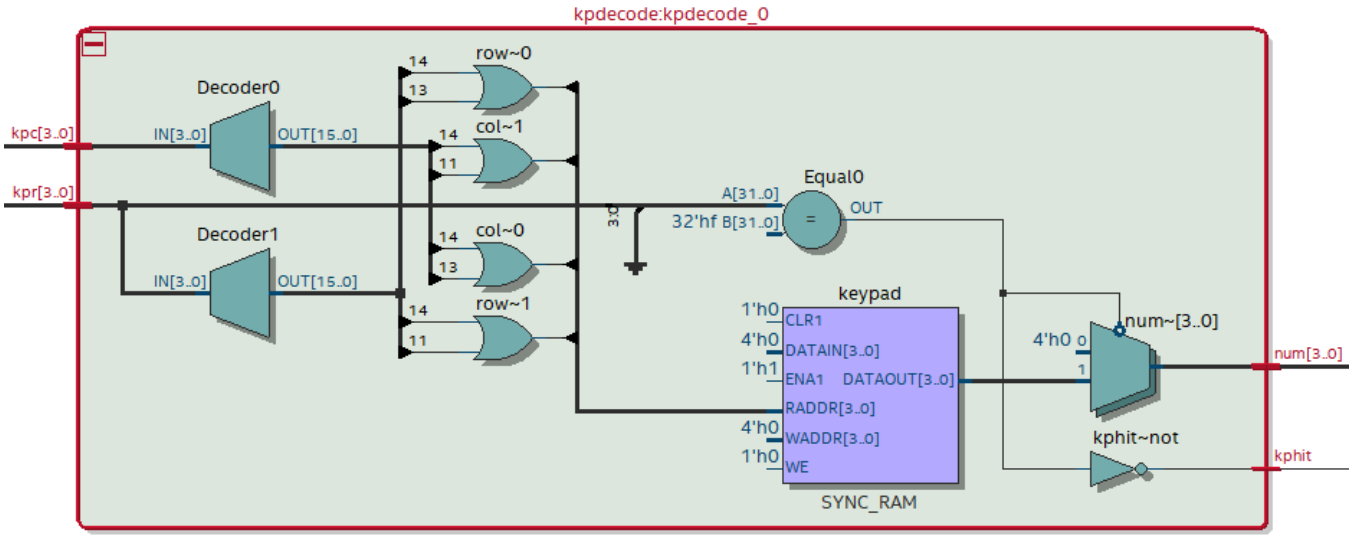

Figure 4: kpdecode.sv diagram

2.4 Decode7

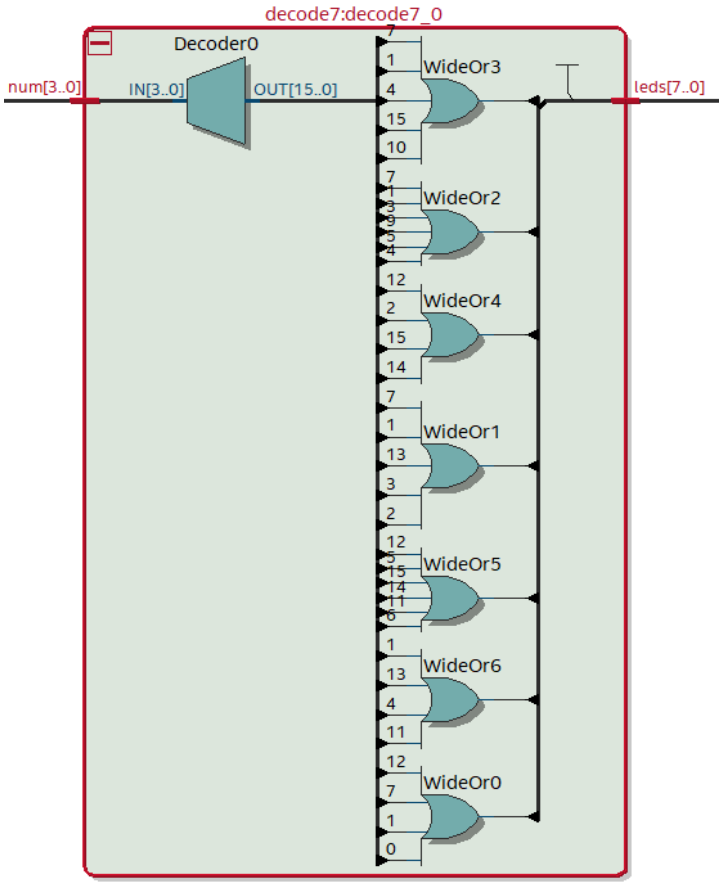

Figure 5: decode7.sv diagram

3 Compilation Summary

Flow Status	Successful - Fri Jan 25 09:56:33 2019
Quartus Prime Version	18.0.0 Build 614 04/24/2018 SJ Lite Edition
Revision Name	lab2
Top-level Entity Name	lab2
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	18 / 22,320 (< 1 %)
Total registers	4
Total pins	22 / 154 (14 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	1 / 4 (25 %)

Figure 6: Compilation Summary