Lab 3

Multiplexed Display

1/29/2023

For: Troy Scevers

CST 231 – DIGITAL SYSTEM DESIGN I

By: Cari Blaker

Table of Contents

Abstract

This project is a twelve bit up/down counter (that also happens to be parameterized) with enable and reset features. The DE1-SoC board uses switches for enable, up/down toggling, and reset. It uses a multiplexer to output one of four seven segment digits at a time to the CL3641AH, at a fast enough speed that, to the human eye, it appears all four displays are lit at once. The lab requires two clock divisions to achieve the desired output.

Introduction

The purpose of this lab is to practice interfacing. Using both the DE1-SoC board and the CL3641AH seven-segment display, this lab creates an up/down counter with enable and reset, interfacing the two devices. The DE1-SoC board is used to control the counter displayed on the CL3641AH.

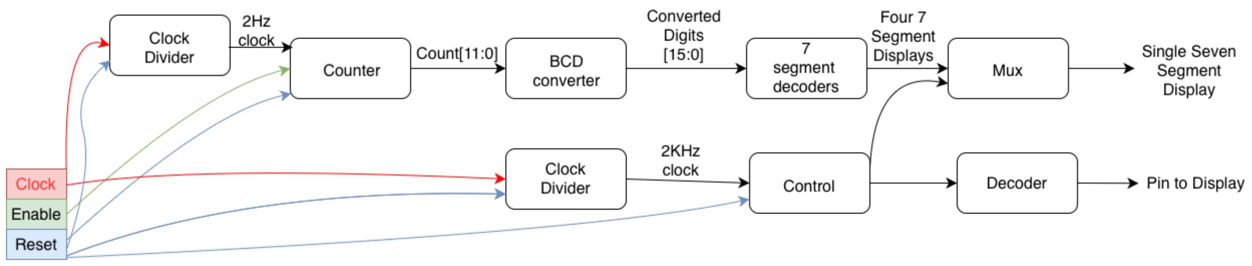


Figure 1: Lab Block Diagram

Design

Physical hardware design

The DE1-SoC board switches zero through two are used for reset, enable, and up/down selection.

The DE1-SoC board's GPIO[0] output pins are used to connect the breadboard and wire up the CL3641AH. Each GPIO pin outputs 3.3V (Vcc). The CL3641AH's forward voltage is 1.8V (Vf). The current, per dice, of the CL3641AH should be between 30 mA and 200 mA. To calculate minimum and maximum current per LED, each current amount is divided by seven to obtain:

$$I1 = \frac{0.03A}{7} = 4.3\, mA$$

Figure 2: Min Current Per Pin Calculation

And:

$$I2 = \frac{0.2A}{7} = 28.6\, mA$$

Figure 3: Max Current Per Pin Calculation

To calculate resistance, the following equation was used twice—once with I1, once with I2.:

$$\frac{Vcc - Vf}{I} = R$$

Figure 4: Resistor Calculation

This obtains the resistance bounds per pin into the CL3641AH of:

$$53\,\Omega \; < \; R < 348\,\Omega$$

Figure 5: Resistor Bounds

Therefore, each resistor was chosen to be 100 Ω. Of the twelve pins on the CL3641AH, eleven are used in the lab. Pin three is a decimal point and not necessary for the function of the lab, and is therefore left out. Eleven resistors are required, one for each pin. Eleven female to male connecting wires are required to connect the GPIO pins on the DE1-SoC to the breadboard where the CL3641AH is connected. Figure 6 in Appendix A shows the pin out diagram from the data sheet for the CL3641AH. Figure 7 in Appendix A shows the GPIO 0 pin out diagram from the data sheet for the DE1-SoC board.

Synthesized hardware design

The major functional modules of this lab's design hierarchy are the top level, the front end, and the back end. The top level connects the front and back.

Design Structure

A diagram of the hierarchy of the lab is provided below in Figure 8. The top level connects the front end and the back end. The back end includes all of the logic to *maintain* a proper count. The front end includes all of the logic to *display* the proper count.

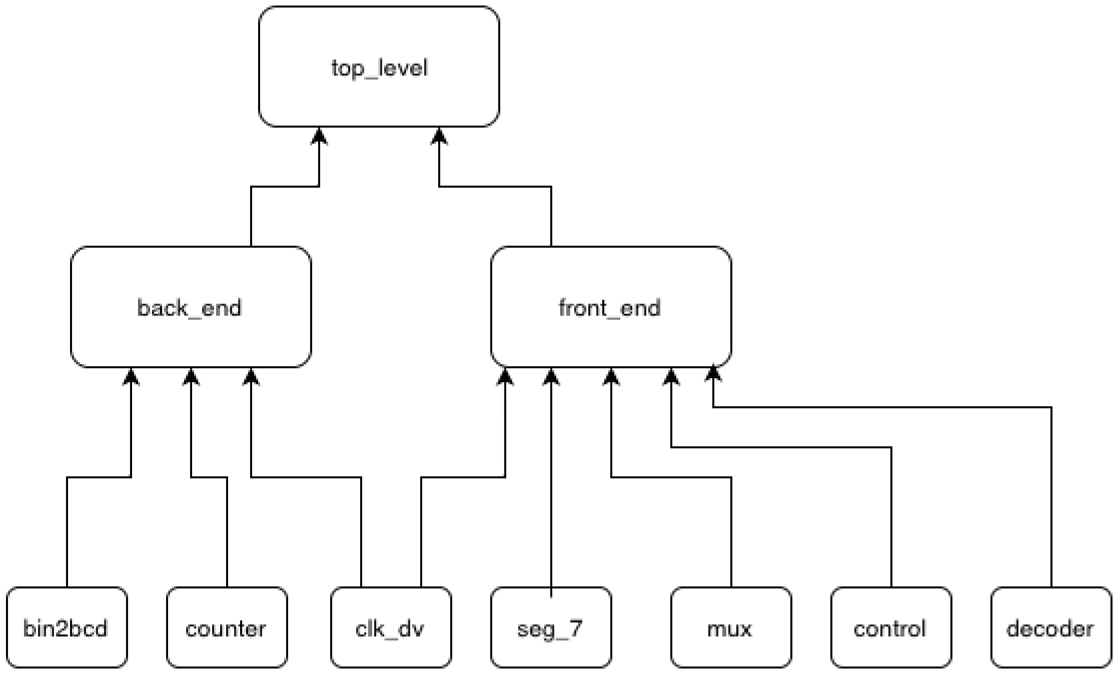


Figure 6: Design Hierarchy Block Diagram

Separate instantiations and outputs of the clock divider module drive the activities of both the front end and the back end.

Modules

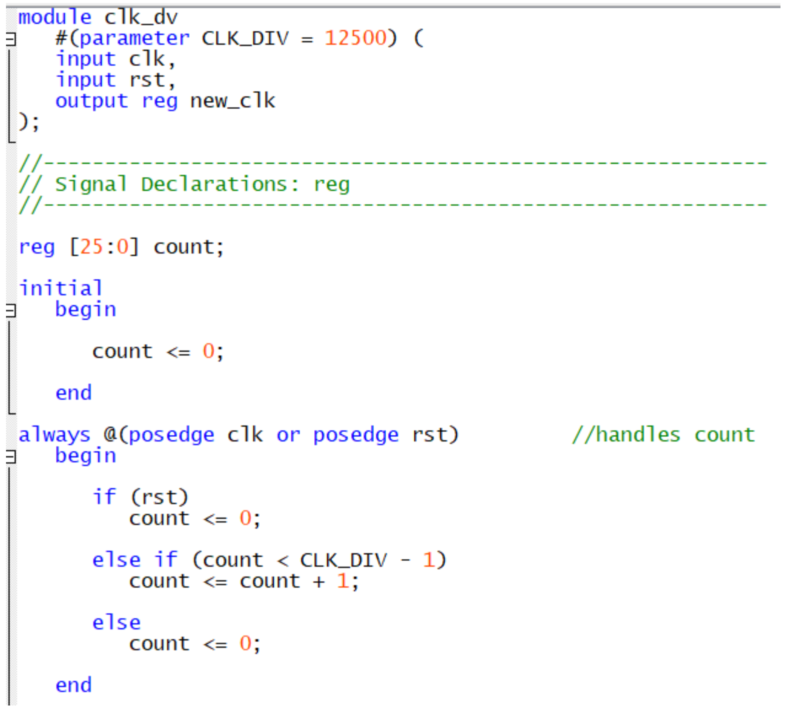
Module 1–clk_dv

The DE1-SoC provides multiple 50 MHz clocks. The lab uses one of these to instantiate two different clock dividers–one speed for counting in the back_end module, and another speed to be used by the front_end module for display. The counter, ideally, counts slowly enough that each number is witnessed. The display, ideally, muxes through the four displays quickly enough to not be witnessed. An ideal display's speed is 500 Hz per display. The CL3641AH has four seven segment displays, so the clock is divided into a 2 KHz clock speed for display in the front_end module. Two counts per second is a perfectly reasonable speed to count, so the counting clock is divided into a 2 Hz clock in the back_end module. Both of these divisions are done using the following equation, where x is the parameter provided to the clock divider module:

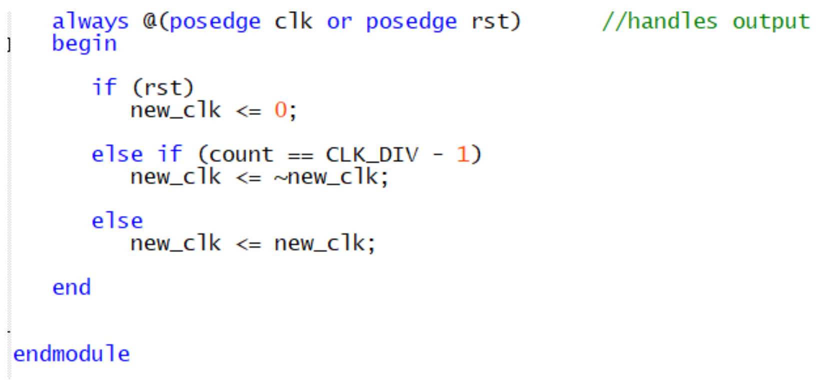
$$\frac{\mathit{Speed\ Provided}}{x} = \frac{\mathit{Speed\ Desired}}{2}$$

Figure 7: Clock Division Equation

This module's inputs are clock and reset. The clock divider module resets asynchronously, but in time with the reset of the count. The clk_dv module takes in the clock provided and divides it by the parameter given, and the output clock is twice the speed of that division due to duty cycle. Figure 12 in Appendix B is the clock divider's RTL Diagram. The module's code is given below:



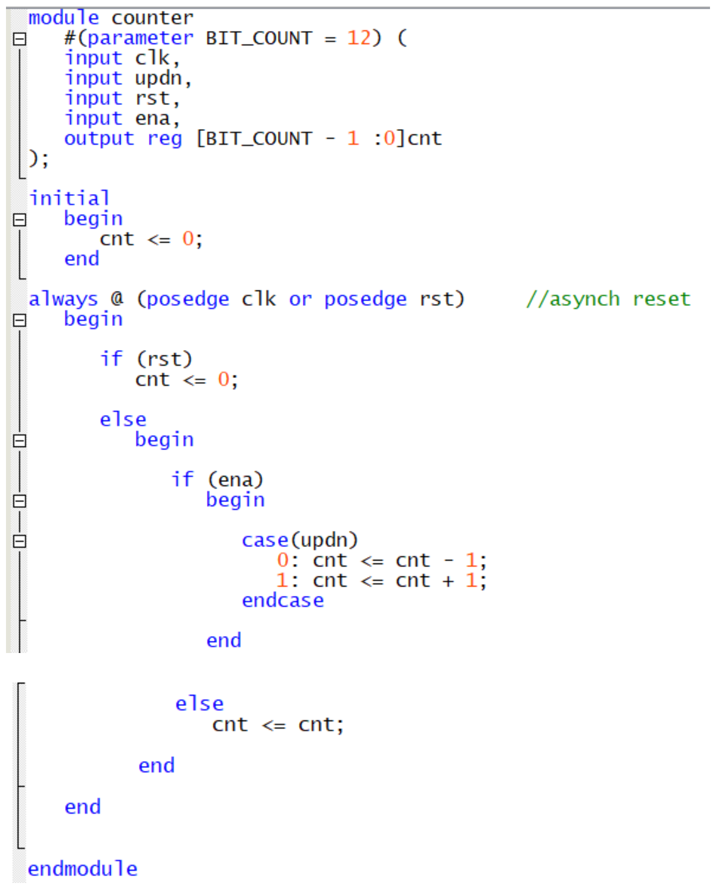
```
module clk_dv
    #(parameter CLK_DIV = 12500) (
    input clk,
    input rst,
    output reg new_clk
);

//-----------------------------------------------------------
// Signal Declarations: reg
//-----------------------------------------------------------

reg [25:0] count;

initial
    begin

        count <= 0;

    end

always @(posedge clk or posedge rst)          //handles count
    begin

        if (rst)
            count <= 0;

        else if (count < CLK_DIV - 1)
            count <= count + 1;

        else
            count <= 0;

    end
```



```
   always @(posedge clk or posedge rst)      //handles output
   begin

      if (rst)
         new_clk <= 0;

      else if (count == CLK_DIV - 1)
         new_clk <= ~new_clk;

      else
         new_clk <= new_clk;

   end

endmodule
```

Module 2–counter

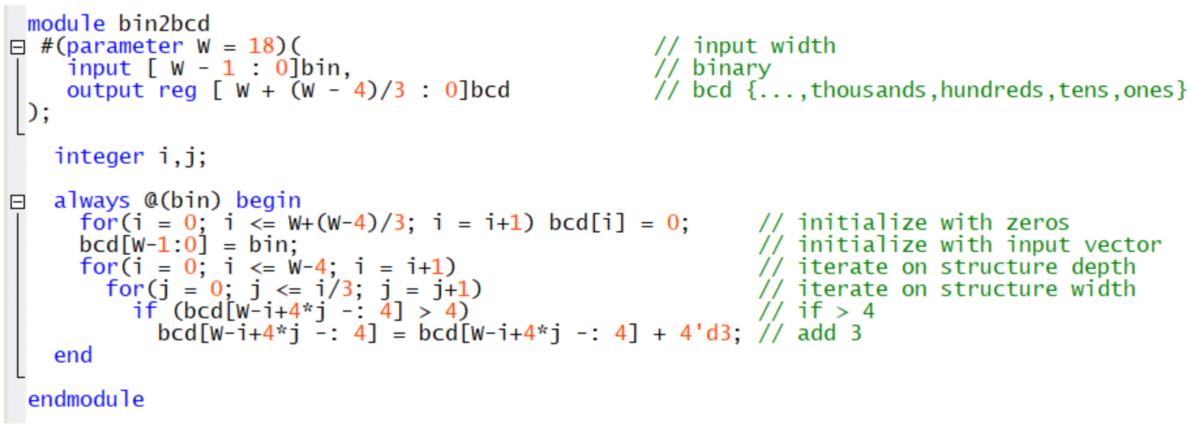
The counter's inputs are enable, reset, and updn. The enable allows the count to continue, the reset (active high) starts the count back at zero, and updn is high to count up, low to count down. The module is parameterized to provide any size count–however, the lab requirements are a twelve bit count, so the module is standardized as such. Its only output is the twelve bit count. The code for the module is as follows:



```
module counter
   #(parameter BIT_COUNT = 12) (
   input clk,
   input updn,
   input rst,
   input ena,
   output reg [BIT_COUNT - 1 :0]cnt
);

initial
   begin
      cnt <= 0;
   end

always @ (posedge clk or posedge rst)     //asynch reset
   begin

      if (rst)
         cnt <= 0;

      else
         begin

            if (ena)
               begin

                  case(updn)
                     0: cnt <= cnt - 1;
                     1: cnt <= cnt + 1;
                  endcase

               end


            else
               cnt <= cnt;

         end

   end

endmodule
```

Module 3–bin2bcd

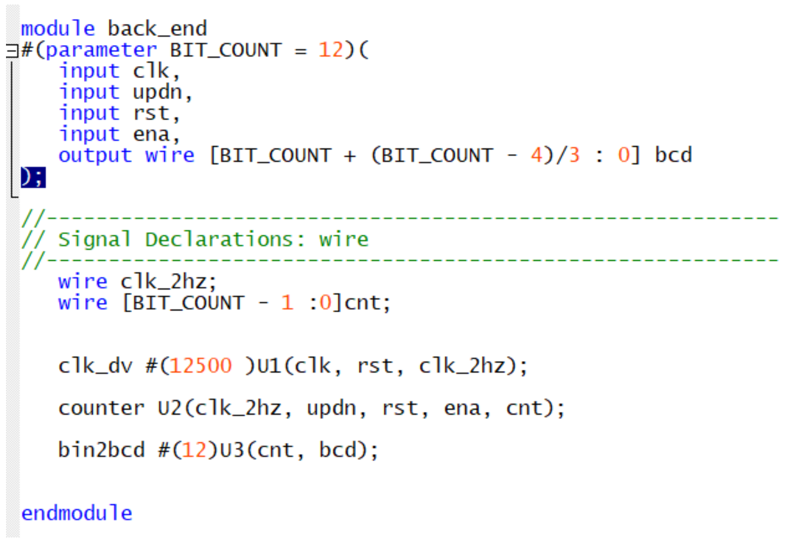
The binary to BCD converter module takes in a parameterized number of binary (in this case, twelve), and converts it to the correct number of four-bit binary coded decimal groupings. This module's code was copied from Wikipedia's "Double Dabble" page. Figure 15 in Appendix B is the RTL for the binary to BCD converter. See the module code below:



```
module bin2bcd
 #(parameter W = 18)(                                   // input width
   input [ W - 1 : 0]bin,                               // binary
   output reg [ W + (W - 4)/3 : 0]bcd                   // bcd {...,thousands,hundreds,tens,ones}
);

  integer i,j;

  always @(bin) begin
    for(i = 0; i <= W+(W-4)/3; i = i+1) bcd[i] = 0;      // initialize with zeros
    bcd[W-1:0] = bin;                                    // initialize with input vector
    for(i = 0; i <= W-4; i = i+1)                        // iterate on structure depth
      for(j = 0; j <= i/3; j = j+1)                      // iterate on structure width
        if (bcd[W-i+4*j -: 4] > 4)                       // if > 4
          bcd[W-i+4*j -: 4] = bcd[W-i+4*j -: 4] + 4'd3; // add 3
  end

endmodule
```

Module 4–back_end

This module ties the three previous modules together to perform the counting operations at the desired speed of 2 Hz. The clock divider is instantiated with the parameter 12,500,000 to provide the desired speed–using the previously established equation from Figure 7. Its output clock is then an input for the instantiated counter, which gets the same reset, as well as the updn, and enable of the module. The counter's output count is sent to the binary to BCD converter, which is the output for this higher level module. The module's inputs are the system clock, updn, reset, and enable, as seen in the module code:

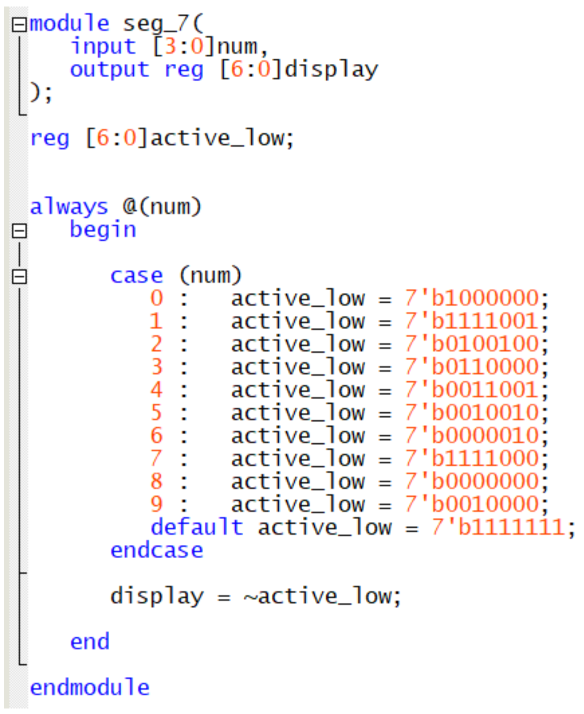


```
module back_end
#(parameter BIT_COUNT = 12)(
   input clk,
   input updn,
   input rst,
   input ena,
   output wire [BIT_COUNT + (BIT_COUNT - 4)/3 : 0] bcd
);

//------------------------------------------------------------
// Signal Declarations: wire
//------------------------------------------------------------
   wire clk_2hz;
   wire [BIT_COUNT - 1 :0]cnt;


   clk_dv #(12500 )U1(clk, rst, clk_2hz);

   counter U2(clk_2hz, updn, rst, ena, cnt);

   bin2bcd #(12)U3(cnt, bcd);


endmodule
```

The back end module's RTL Diagram can be found in Appendix B, Figure 16.

Module 5–seg_7

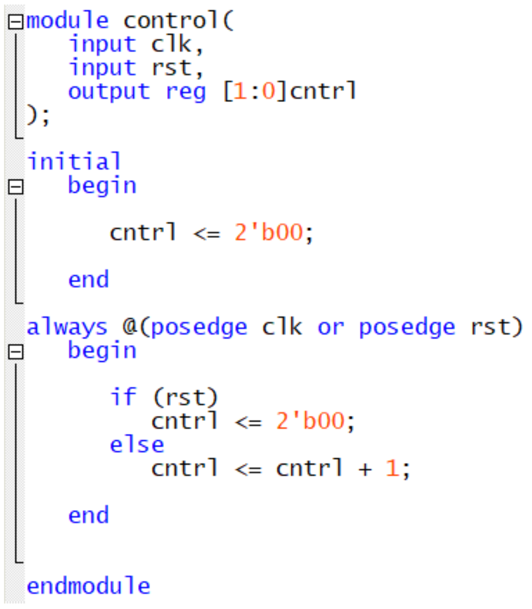
This module is the seven segment decoder. It takes in a four-bit binary number and decodes it for seven segment display, in an active high configuration, outputting seven bits. This module was taken from the last lab and inverted for active high configuration. Seven Segment display RTL and waveforms can be found in Appendix B, Figures 17 and 18. The module code follows:



```
module seg_7(
    input [3:0]num,
    output reg [6:0]display
);

reg [6:0]active_low;


always @(num)
    begin

        case (num)
            0 :    active_low = 7'b1000000;
            1 :    active_low = 7'b1111001;
            2 :    active_low = 7'b0100100;
            3 :    active_low = 7'b0110000;
            4 :    active_low = 7'b0011001;
            5 :    active_low = 7'b0010010;
            6 :    active_low = 7'b0000010;
            7 :    active_low = 7'b1111000;
            8 :    active_low = 7'b0000000;
            9 :    active_low = 7'b0010000;
            default active_low = 7'b1111111;
        endcase

        display = ~active_low;

    end

endmodule
```

Module 6–control

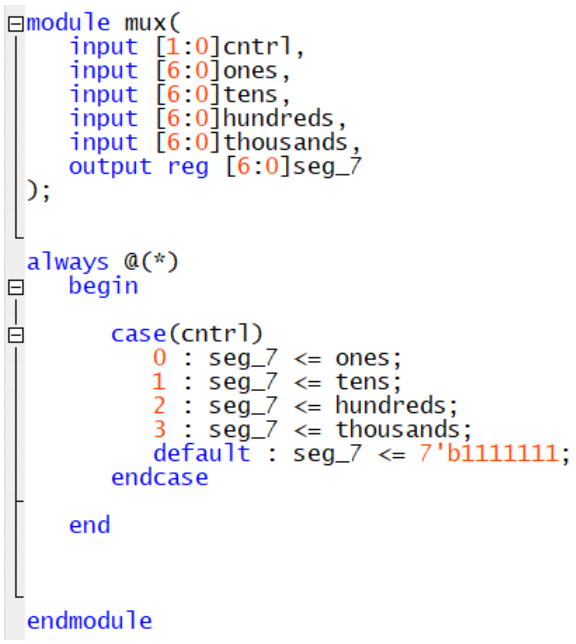
This module's inputs are the divided clock for the front end and the same reset from the rest of the program. It is a simple adder, whose output is a two digit binary number, cycling through the numbers zero to three at the high speed of the input clock. This output signifies which digit is to be displayed (but is not yet decoded for display). The code is copied here:



```
module control(
    input clk,
    input rst,
    output reg [1:0]cntrl
);

initial
    begin

        cntrl <= 2'b00;

    end

always @(posedge clk or posedge rst)
    begin

        if (rst)
            cntrl <= 2'b00;
        else
            cntrl <= cntrl + 1;

    end

endmodule
```

Module 7–mux

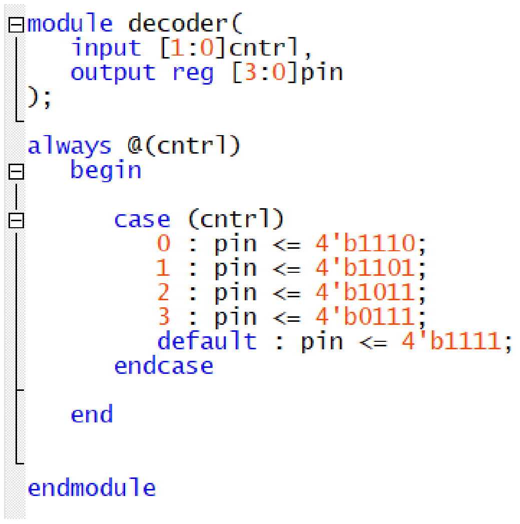
This module receives as inputs four seven-segment display digits: the ones, tens, hundreds, and thousands digits of the counted number. Its other input is the two bit control line. The control line determines which of the digits will be the mux's output. The smallest control amount corresponds to the smallest output digit, and likewise with the highest. Find the module code below:



```
module mux(
    input [1:0]cntrl,
    input [6:0]ones,
    input [6:0]tens,
    input [6:0]hundreds,
    input [6:0]thousands,
    output reg [6:0]seg_7
);

always @(*)
    begin

        case(cntrl)
            0 : seg_7 <= ones;
            1 : seg_7 <= tens;
            2 : seg_7 <= hundreds;
            3 : seg_7 <= thousands;
            default : seg_7 <= 7'b1111111;
        endcase

    end


endmodule
```

Module 8–decoder

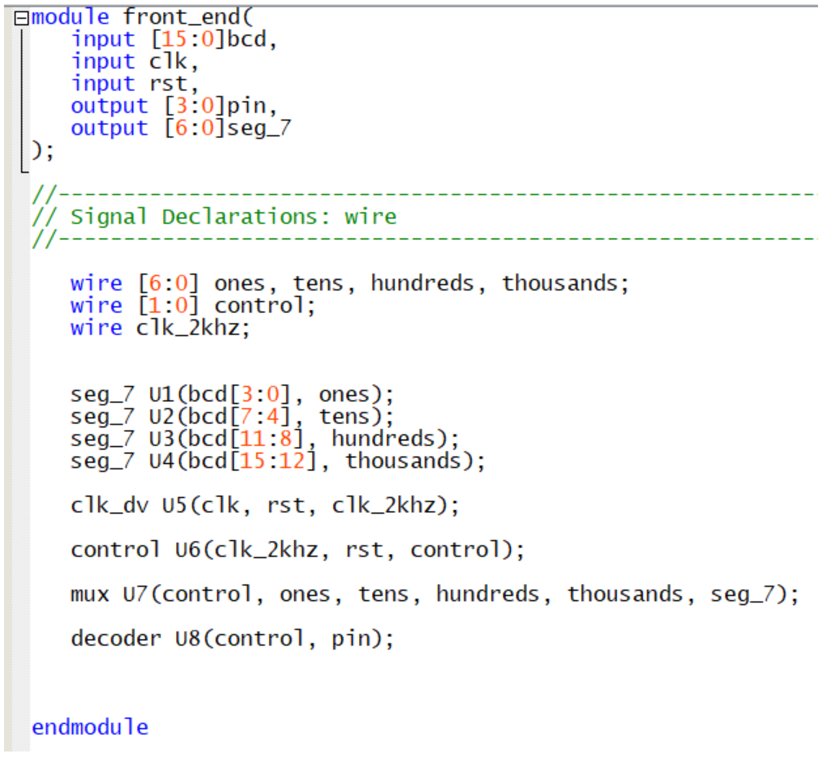
This module's input is the control's output (the same control line as the mux). The decoder decodes those two bits, and outputs the four-bit active low signal (because the CL3641AH has a common cathode configuration for the four digit selector pins, as seen in Appendix A, Figure 8) to ultimately turn on the proper display for the proper digit at the proper time. Here is this module's code:



```
module decoder(
    input [1:0]cntrl,
    output reg [3:0]pin
);

always @(cntrl)
    begin

        case (cntrl)
            0 : pin <= 4'b1110;
            1 : pin <= 4'b1101;
            2 : pin <= 4'b1011;
            3 : pin <= 4'b0111;
            default : pin <= 4'b1111;
        endcase

    end


endmodule
```

Module 9–front_end

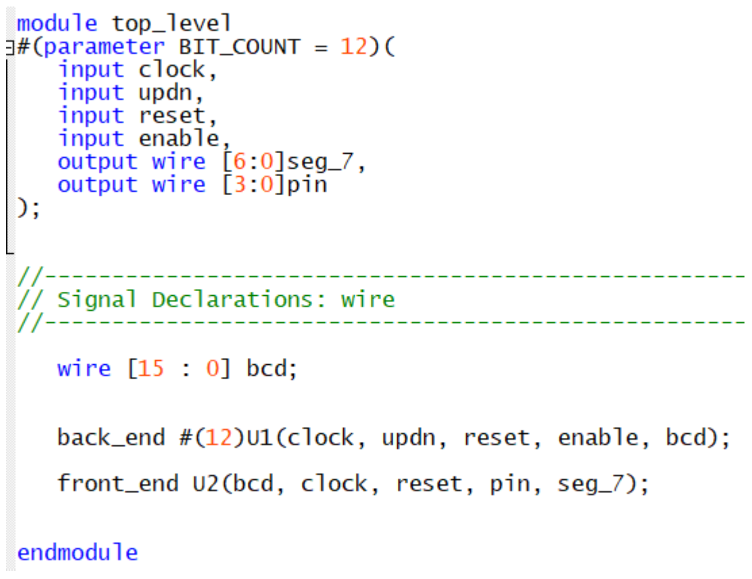
This module ties the four previous modules together, plus the clock divider module, to perform the necessary operations for displaying the count to the CL3641AH. Three wires are declared to tie the modules together: four seven-bit wires for the ones, tens, hundreds, and thousands digits of seven-segment display of the counted number, a 2 bit wire for the control, and a wire for the divided 2 KHz clock. Four seg_7 modules are instantiated to create the input for the mux, the clock is divided and sent to the control line, whose output feeds the mux and the decoder. As the control line quickly manages both the mux and the decoder simultaneously, the proper seven segment digit is sent to the proper pin at a rate faster than the eye can notice it is not all happening at once. The front end module's RTL diagram can be found in Appendix B, Figure 24. The module's code is below:



```
module front_end(
    input [15:0]bcd,
    input clk,
    input rst,
    output [3:0]pin,
    output [6:0]seg_7
);

//-------------------------------------------------------------
// Signal Declarations: wire
//-------------------------------------------------------------

    wire [6:0] ones, tens, hundreds, thousands;
    wire [1:0] control;
    wire clk_2khz;


    seg_7 U1(bcd[3:0], ones);
    seg_7 U2(bcd[7:4], tens);
    seg_7 U3(bcd[11:8], hundreds);
    seg_7 U4(bcd[15:12], thousands);

    clk_dv U5(clk, rst, clk_2khz);

    control U6(clk_2khz, rst, control);

    mux U7(control, ones, tens, hundreds, thousands, seg_7);

    decoder U8(control, pin);


endmodule
```

Module 10–top_level

The top-level module for the design is aptly named, simply tying the back end and the front end together. It is parameterized, in case this design were to be used with different hardware and could display a much larger or smaller count. Its inputs are the system clock, reset, enable, updn, and its outputs are a seven bit seven segment display, and the four bit pin selector. One wire is required, which is a sixteen-bit (this would need to be adjusted for other size parameters) BCD number, that connects the output of the back-end to the input of the front-end instantiations. The module code follows:



```
module top_level
#(parameter BIT_COUNT = 12)(
   input clock,
   input updn,
   input reset,
   input enable,
   output wire [6:0]seg_7,
   output wire [3:0]pin
);

//---------------------------------------------------
// Signal Declarations: wire
//---------------------------------------------------

   wire [15 : 0] bcd;

   back_end #(12)U1(clock, updn, reset, enable, bcd);

   front_end U2(bcd, clock, reset, pin, seg_7);

endmodule
```

Simulation and Testing

The first module tested is the clock divider. The parameter can be adjusted to 2, to observe the output be exactly the same as the clock input (which should be set to any clock speed). The parameter can be adjusted to 4 to observe the output as the input clock speed, divided in half. The clock divider's RTL can be found in Appendix B.

Next, the counter is tested. The clock is set to any clock speed. The enable, reset, and up/down inputs are each tested with lows and highs. Reset is asynchronous, up/down and enable are both synchronous. If reset is high, output should immediately be and remain zero until it is low again. If enable is high, the count should increase/decrease based on the position of up/down. When logic one, updn should count up, and a logic zero updn counts down. If enable is low, the count should stay where it is until enable is high again. Appendix B (Figure 13 and Figure 14) contains the expected test waveforms for the counter module, as well as its RTL Diagram..

The seven segment display module can then be tested. Cycling the input from zero to nine, The expected output should correspond with bit 0 as segment A of the seven segment display, through to bit 6 as segment G, in an active high format (because the CL3641AH has a common anode format for the seven segment pins, as seen in Figure 8). The waveforms are provided in Appendix B (Figure 17 and Figure 18).

The control is tested just as a counter. Its output should go up one each rising clock edge until it reaches 3, and cycle back to zero. Figure 19 in Appendix B shows the RTL Diagram of the control module.

The mux should be tested by cycling through the four possible control numbers and being sure the output is the corresponding ones, tens, hundreds, or thousands seven segment display input. The multiplexer's RTL and waveforms can be found in Appendix B, Figure 20 and Figure 21.

The decoder is tested by cycling through the four possible control numbers and being sure the proper corresponding active low signal is its output. Figure 22 and Figure 23 in Appendix B show the RTL Diagram and waveforms of the decoder module.

The fully compiled and synthesized result can then be tested. The top-level is tested by hooking up the board, in the same manner as the schematic (Figure 11). This should behave the same way as the counter was tested in simulation previously, except the count can be watched from the CL3641AH, the reset is operated from switch 0, enable switch 1, and updn switch 2. As long as the count is displayed what appears to be continuously and is, indeed, counting, the other output is also functioning as expected. The top level module's RTL diagram is Figure 25 in Appendix B.

Problems

At first, the decoder was coded for an active high configuration. After fixing the decoder, some of the pins were disarranged when connected to the GPIO. For example, the G and C LEDs on the CL3641AH were logically reversed. The pin assignments were then switched, and the problem solved.

Results and Conclusion

The problems faced in the lab could have been avoided by doing a thorough inspection of data sheets before coding. Occasionally, mistakes will be made, of course, so trial and error cannot always be avoided–as in the case of the G and C LED accidental switch. After these small trials and tribulations, a successful interface between the DE1-SoC board and the CL3641AH is accomplished.

Appendix – A: Pin Maps

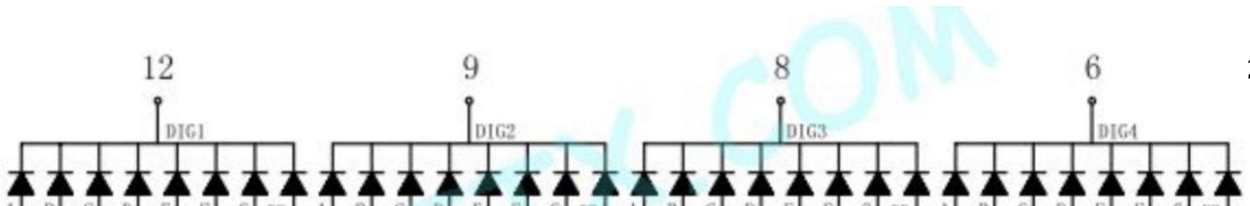


Figure 8: CL3641AH Pin Out Diagram

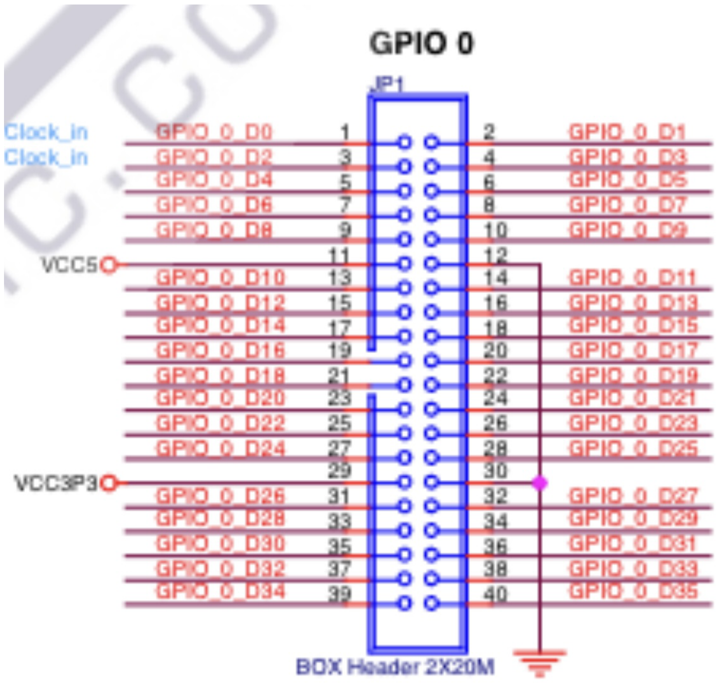
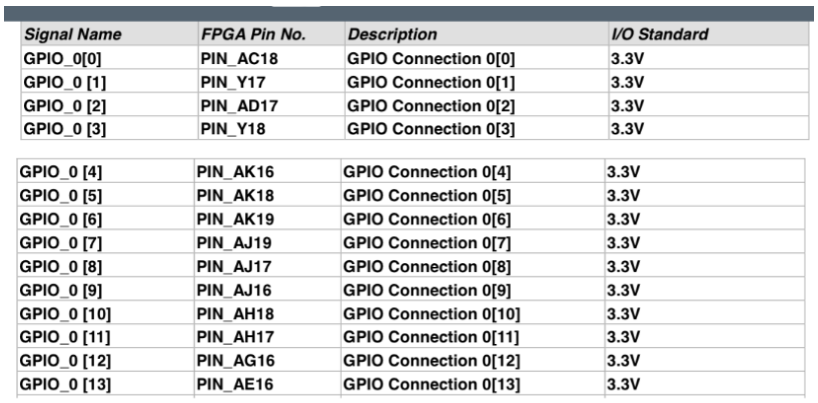


Figure 9: DE1-SoC GPIO 0 Pin Out Diagram



Signal Name	***FPGA Pin No.***	***Description***	***I/O Standard***
GPIO_0[0]	PIN_AC18	GPIO Connection 0[0]	3.3V
GPIO_0 [1]	PIN_Y17	GPIO Connection 0[1]	3.3V
GPIO_0 [2]	PIN_AD17	GPIO Connection 0[2]	3.3V
GPIO_0 [3]	PIN_Y18	GPIO Connection 0[3]	3.3V
GPIO_0 [4]	PIN_AK16	GPIO Connection 0[4]	3.3V
GPIO_0 [5]	PIN_AK18	GPIO Connection 0[5]	3.3V
GPIO_0 [6]	PIN_AK19	GPIO Connection 0[6]	3.3V
GPIO_0 [7]	PIN_AJ19	GPIO Connection 0[7]	3.3V
GPIO_0 [8]	PIN_AJ17	GPIO Connection 0[8]	3.3V
GPIO_0 [9]	PIN_AJ16	GPIO Connection 0[9]	3.3V
GPIO_0 [10]	PIN_AH18	GPIO Connection 0[10]	3.3V
GPIO_0 [11]	PIN_AH17	GPIO Connection 0[11]	3.3V
GPIO_0 [12]	PIN_AG16	GPIO Connection 0[12]	3.3V
GPIO_0 [13]	PIN_AE16	GPIO Connection 0[13]	3.3V

Figure 10: DE1-SoC GPIO 0 Pin in Diagram

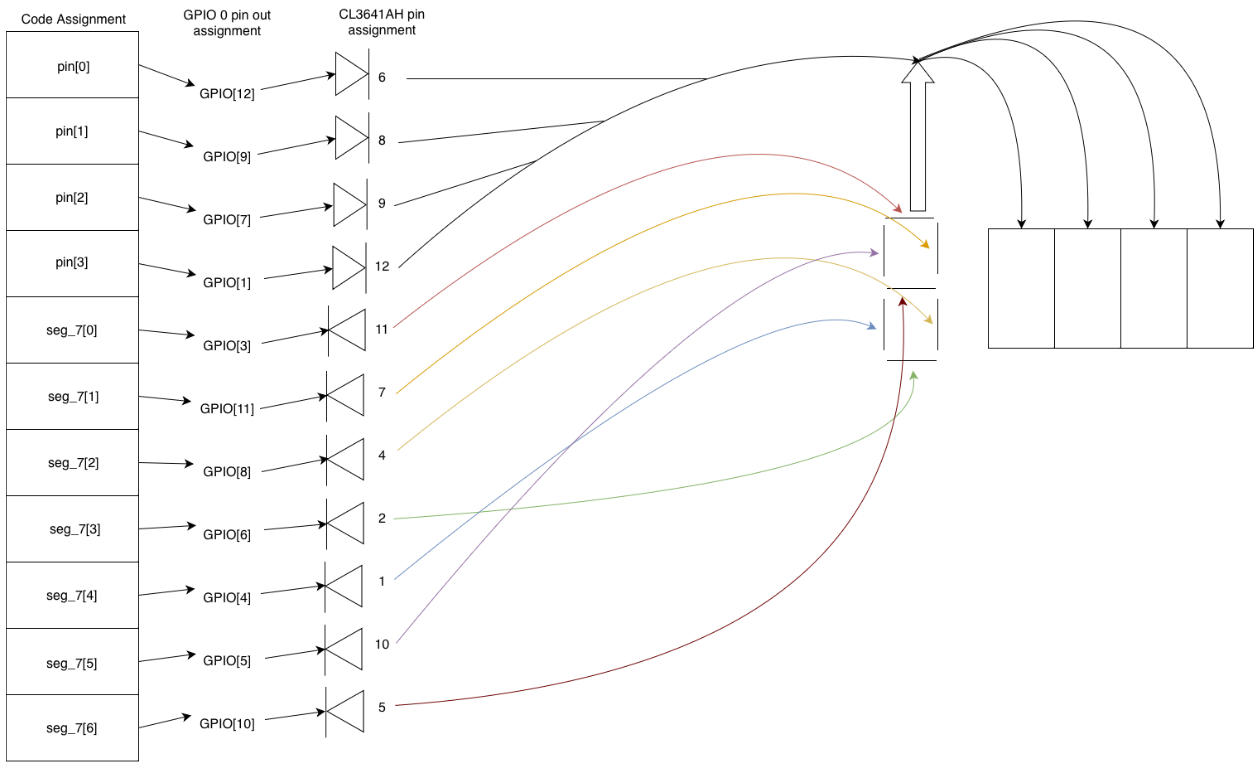


Figure 11: Connection Schematic from program code to GPIO 0 to CL3641AH

Appendix – B: RTL Diagrams and Waveforms

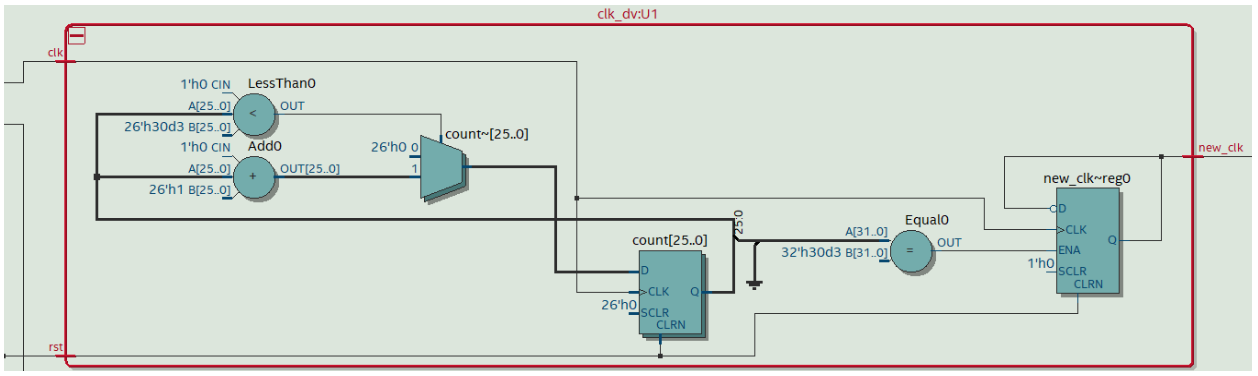


Figure 12: Clock Divider RTL

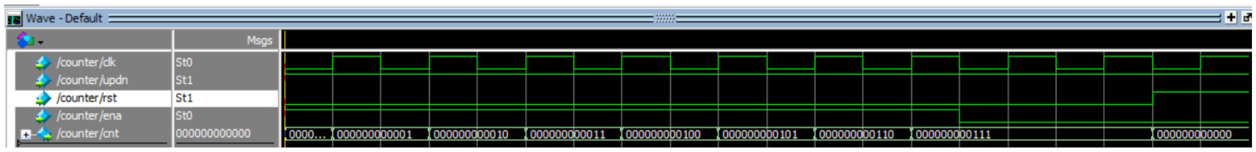


Figure 13: Counter waveforms

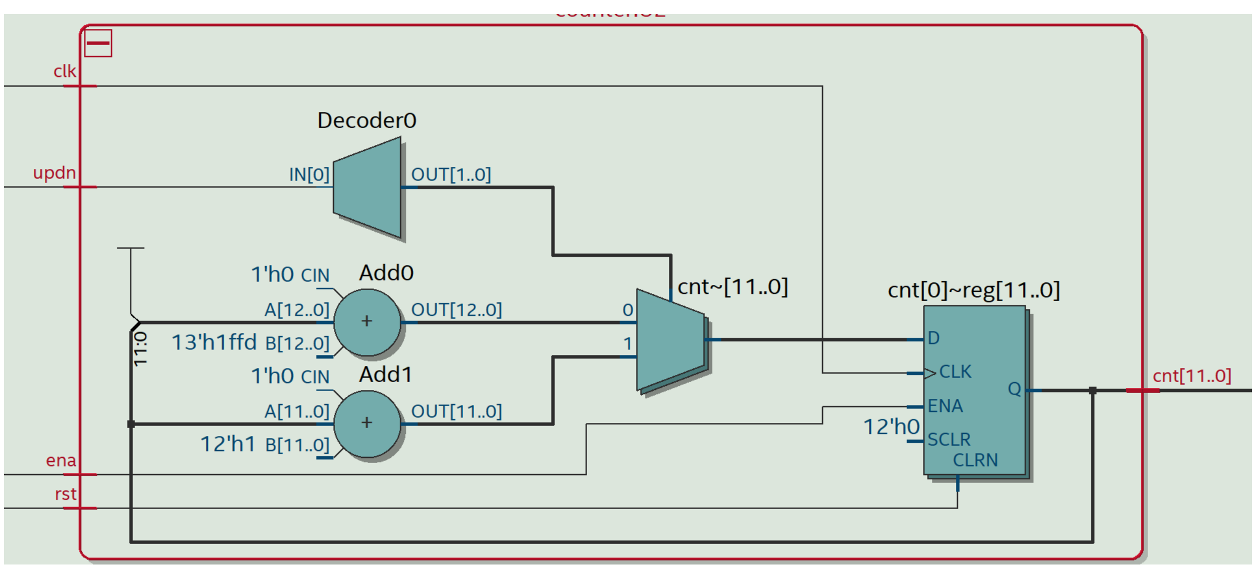


Figure 14: Counter RTL Diagram

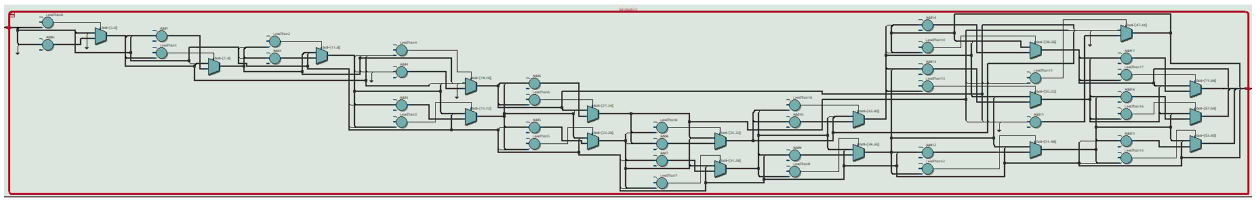


Figure 15: Binary to BCD converter RTL Diagram

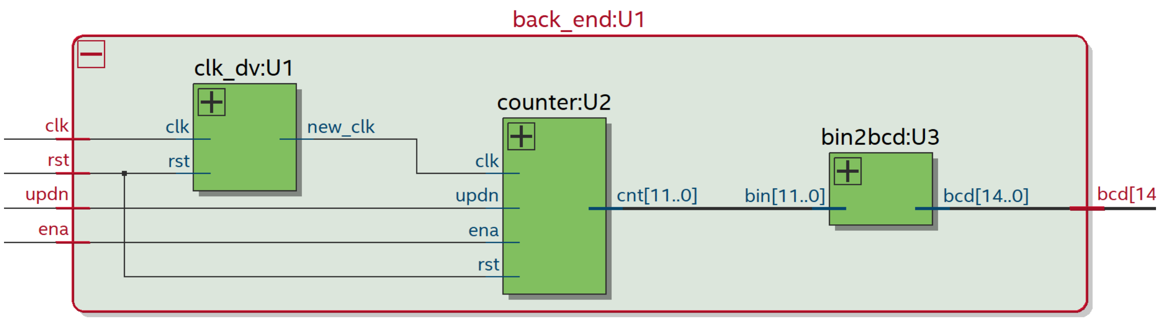


Figure 16: Back End Module RTL Diagram

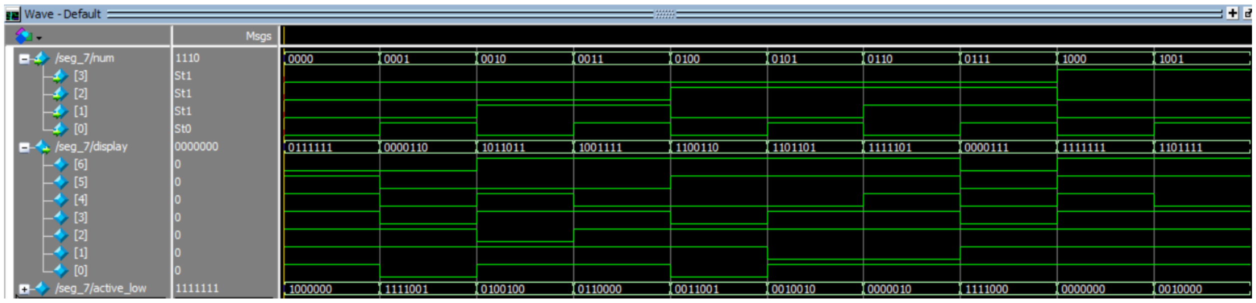


Figure 17: Seven Segment Display Waveforms

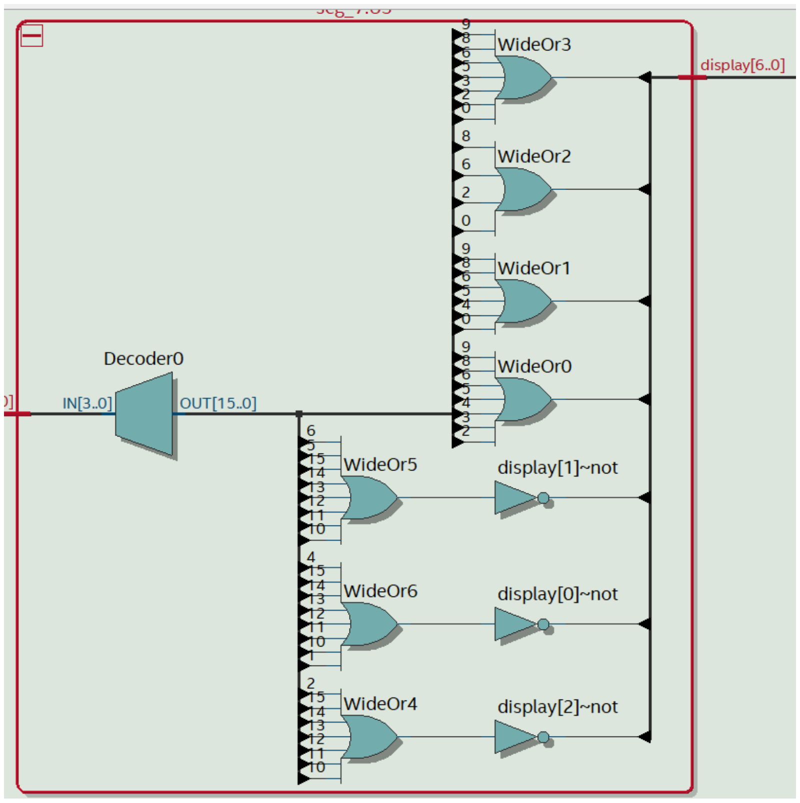


Figure 18: Seven Segment Display RTL Diagram

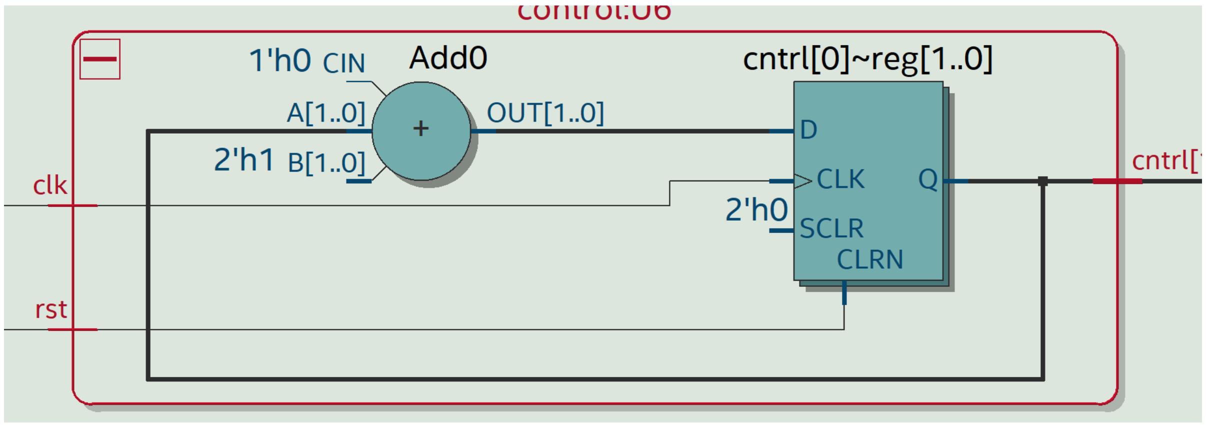
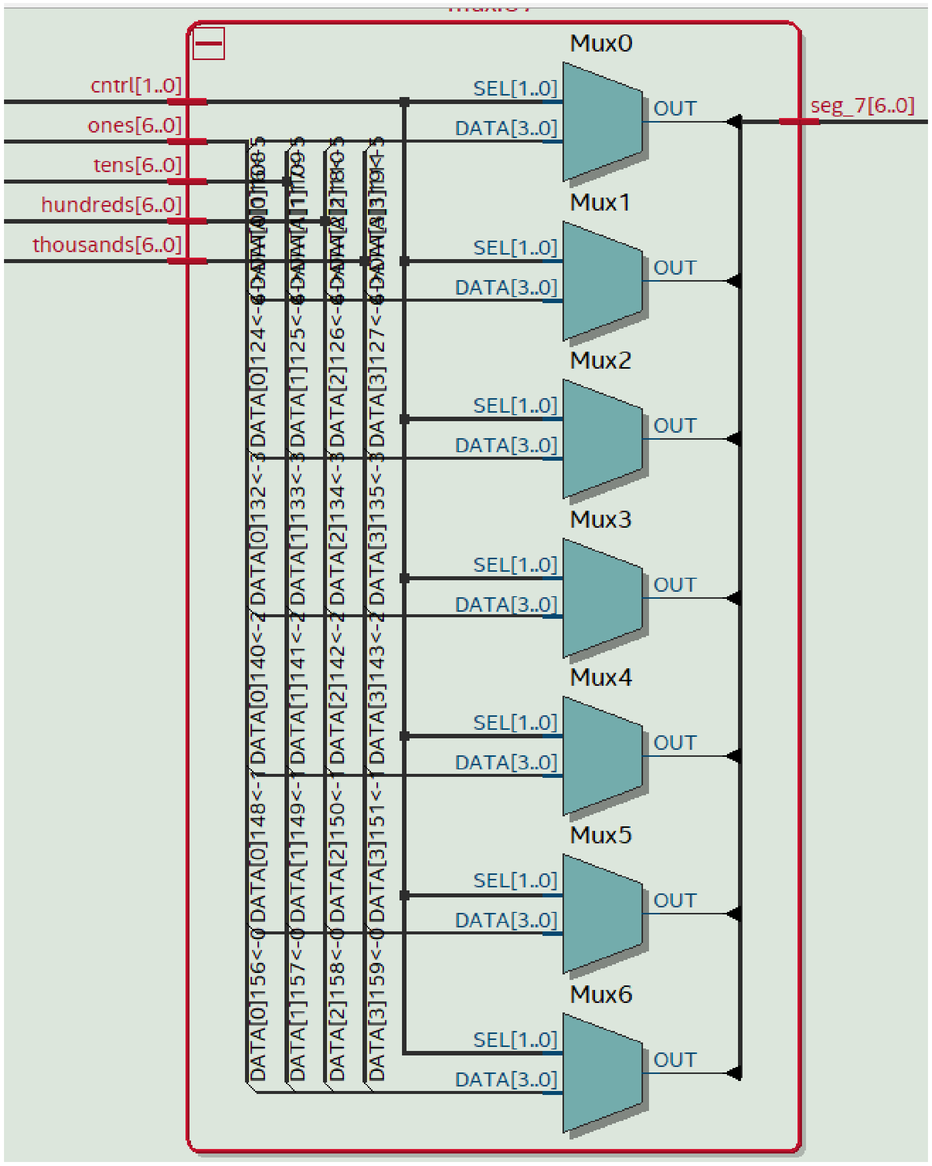
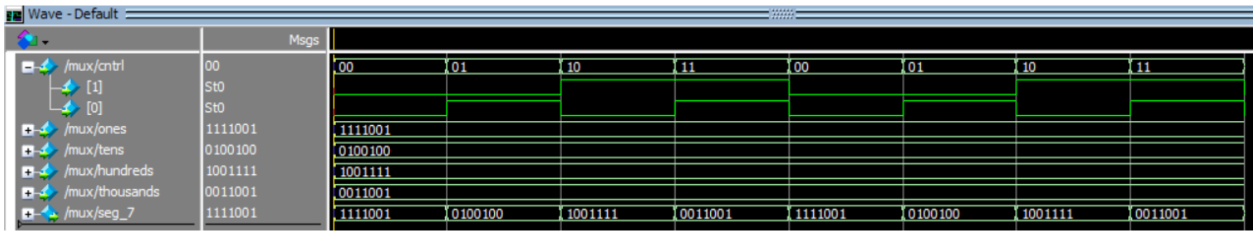


Figure 19: Control RTL Diagram



cntrl[1..0]
ones[6..0]
tens[6..0]
hundreds[6..0]
thousands[6..0]
Mux0
Mux1
Mux2
Mux3
Mux4
Mux5
Mux6
SEL[1..0]
DATA[3..0]
OUT
seg_7[6..0]
DATA[0]156<-0 DATA[0]148<-1 DATA[0]140<-2 DATA[0]132<-3 DATA[0]124<-4
DATA[1]157<-0 DATA[1]149<-1 DATA[1]141<-2 DATA[1]133<-3 DATA[1]125<-4
DATA[2]158<-0 DATA[2]150<-1 DATA[2]142<-2 DATA[2]134<-3 DATA[2]126<-4
DATA[3]159<-0 DATA[3]151<-1 DATA[3]143<-2 DATA[3]135<-3 DATA[3]127<-4

Figure 20: Mux RTL Diagram



```
Wave - Default
                    Msgs
/mux/cntrl          00       00       01       10       11       00       01       10       11
  [1]               St0
  [0]               St0
/mux/ones           1111001  1111001
/mux/tens           0100100  0100100
/mux/hundreds       1001111  1001111
/mux/thousands      0011001  0011001
/mux/seg_7          1111001  1111001  0100100  1001111  0011001  1111001  0100100  1001111  0011001
```

Figure 21: Mux Waveforms

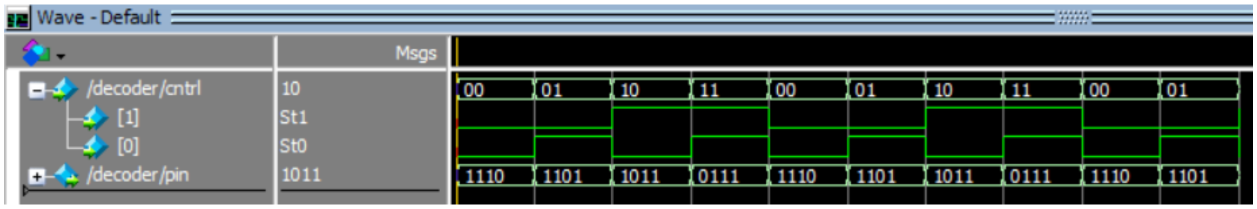


Figure 22: Decoder Waveforms

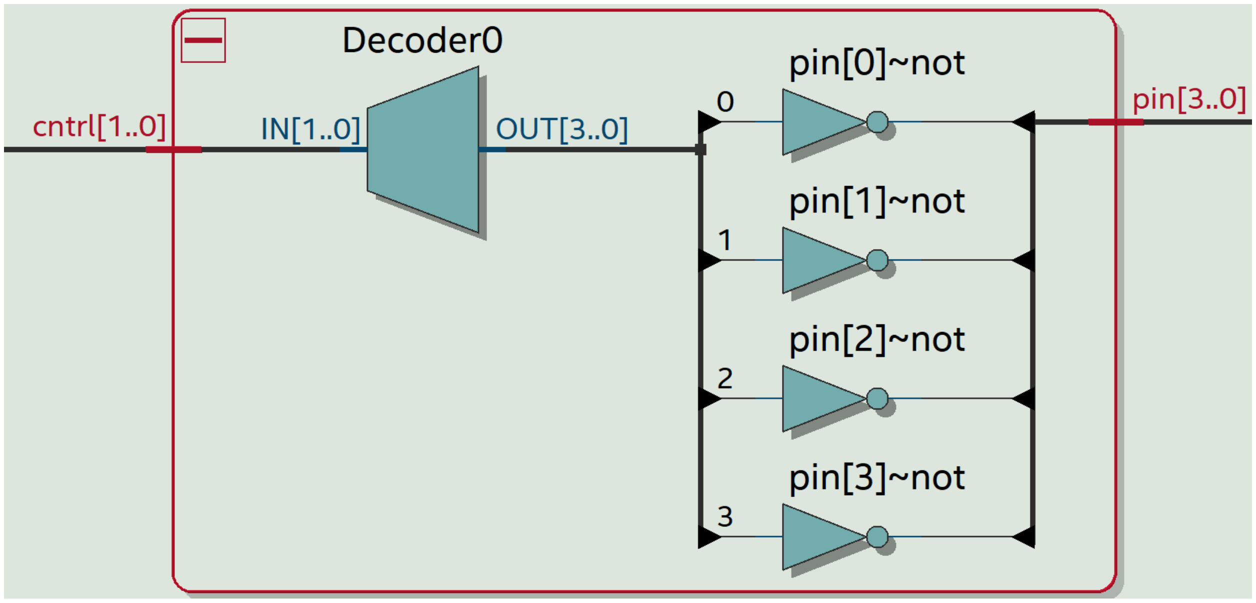


Figure 23: Decoder RTL Diagram

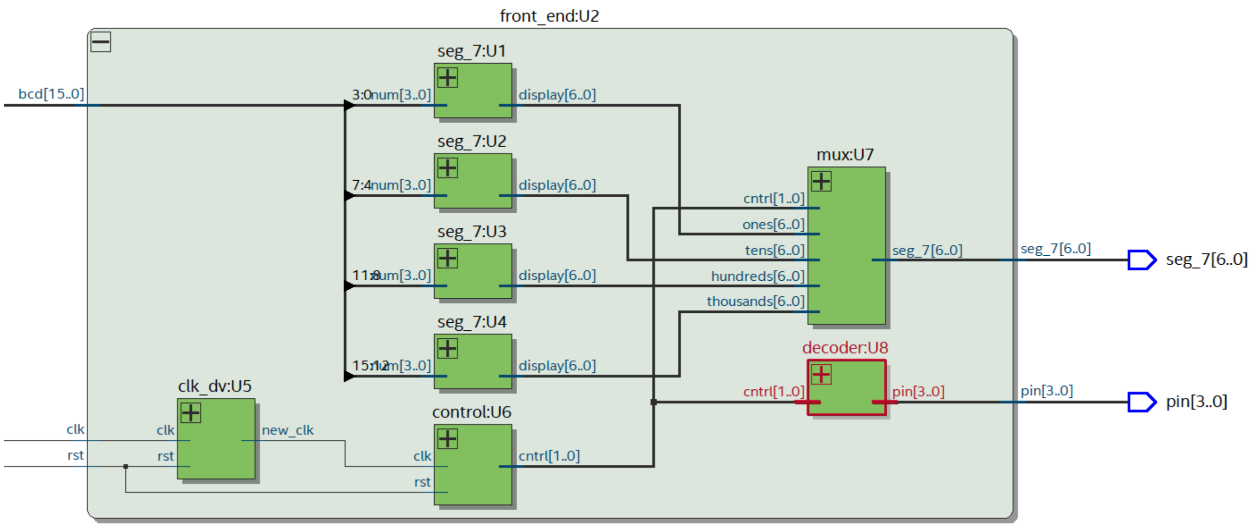


Figure 23: Front End RTL Diagram

Table of Figures

References

[1]   “Double Dabble.” Wikipedia. https://en.wikipedia.org/wiki/Double_dabble (accessed January 28, 2023)

[2]   *DE1-SoC User Manual* (2014). Accessed: January 28, 2023. [Online]. Available: https://inst-fs-iad-prod.inscloudgate.net/files/3607afc4-522a-4f44-a5c1-1580f22dd310/DE1-SoC_User_manual.pdf?download=1&token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE2NzQxOTc2NTcsInVzZXJfaWQiOiIxMjgzOTAwMDAwMDAwMzc5MjYiLCJyZXNvdXJjZSI6Ii9maWxlcy8zNjA3YWZjNC01MjJhLTRmNDQtYTVjMS0xNTgwZjIyZGQzMTAvREUxLVNvQ19Vc2VyX21hbnVhbC5wZGYiLCJqdGkiOiI4YzdhYjE1ZC03NTY5LTQ1YmUtYTQ4NC0xNzc4YjZmZjI1N2MiLCJob3N0Ijoib2l0Lmluc3RydWN0dXJlLmNvbSIsIm9yaWdpbmFsX3VybCI6Imh0dHBzOi8vYTEyODM5LTMwMjY0ODguY2x1c3RlcjkxLmNhbnZhcy11c2VyLWNvbnRlbnQuY29tL2NvdXJzZXMvMTI4Mzl-MTc2OTUvZmlsZXMvMTI4Mzl-MzAyNjQ4OC9jb3Vyc2UlMjBmaWxlcy9ERTEtU29DX1VzZXJfbWFudWFsLnBkZj9kb3dubG9hZF9mcmQ9MVx1MDAyNm5vX2NhY2hlPXRydWVcdTAwMjZyZWRpcmVjdD10cnVlXHUwMDI2c2ZfdmVyaWZpZXI9ZXlKMGVYQWlPaUpLVjFRaUxDSmhiR2NpT2lKSVV6VXhNaUo5LmV5SjFjMlZ5WDJsa0lqb2lNVEk0TXprd01EQXdNREF3TURNM09USTJJaXdpY205dmRGOWhZMk52ZFc1MFgybGtJam9pTVRJNE16a3dNREF3TURBd01EQXdNREF4SWl3aWIyRjFkR2hmYUc5emRDSTZJbTlwZEM1cGJuTjBjblZqZEhWeVpTNWpiMjBpTENKeVpYUjFjbTVmZFhKc0lqcHVkV3hzTENKbVlXeHNZbUZqYTE5MWNtd2lPaUpvZEhSd2N6b3ZMMjlwZEM1cGJuTjBjblZqZEhWeVpTNWpiMjB2WTI5MWNuTmxjeTh4TnpZNU5TOW1hV3hsY3k4ek1ESTJORGc0TDJSdmQyNXNiMkZrUDJSdmQyNXNiMkZrWDJaeVpEMHhYSF
V3TURJMlptRnNiR0poWTJ0ZmRITTlNVFkzTkRJd09UWTROeUlzSW1WNGNDSTZNVFkzTkRJd09UazROMzAuTWtxN0JiQ3dJWnhyTlp0SVNGQk9GNElCOFRVVEZGUVdnR3lwMk5QV21ITUFwdjRPVjRGMlkyYlViVjdhWnlaWF9uMExuWEJBRUJObXVvUVpRZTEzdVEiLCJleHAiOjE2NzQyODQwNTd9.rAUBUNddvg1bZIEEdqQl46VLR4ma64u9MRfpNPFygfuL28WGrF8huwyLI-6aA0d8GSLMLjCmXed53_VN1RCaog

[3] *ALTERA Cyclone V SoC Development & Education Board (DE1-SoC)* (2014). Accessed: January 28, 2023. [Online]. Available: https://oit.instructure.com/courses/17695/files/3026486/download?download_frd=1

[4] *XLITX CL3641AH User Manual.* Accessed: January 28, 2023. https://oit.instructure.com/courses/17695/files/3026487/download?download_frd=1