ELEX 7660 Lab 1

7-SEGMENT LED DISPLAY DECODER

Nicholas Scott AKA “White Cheddar”
A01255181 | JAN 14TH, 2024

Contents

Table of Figures

Part 1 – Coding and Simulation

In this section, I will present the code I wrote, and the simulation results they yielded.

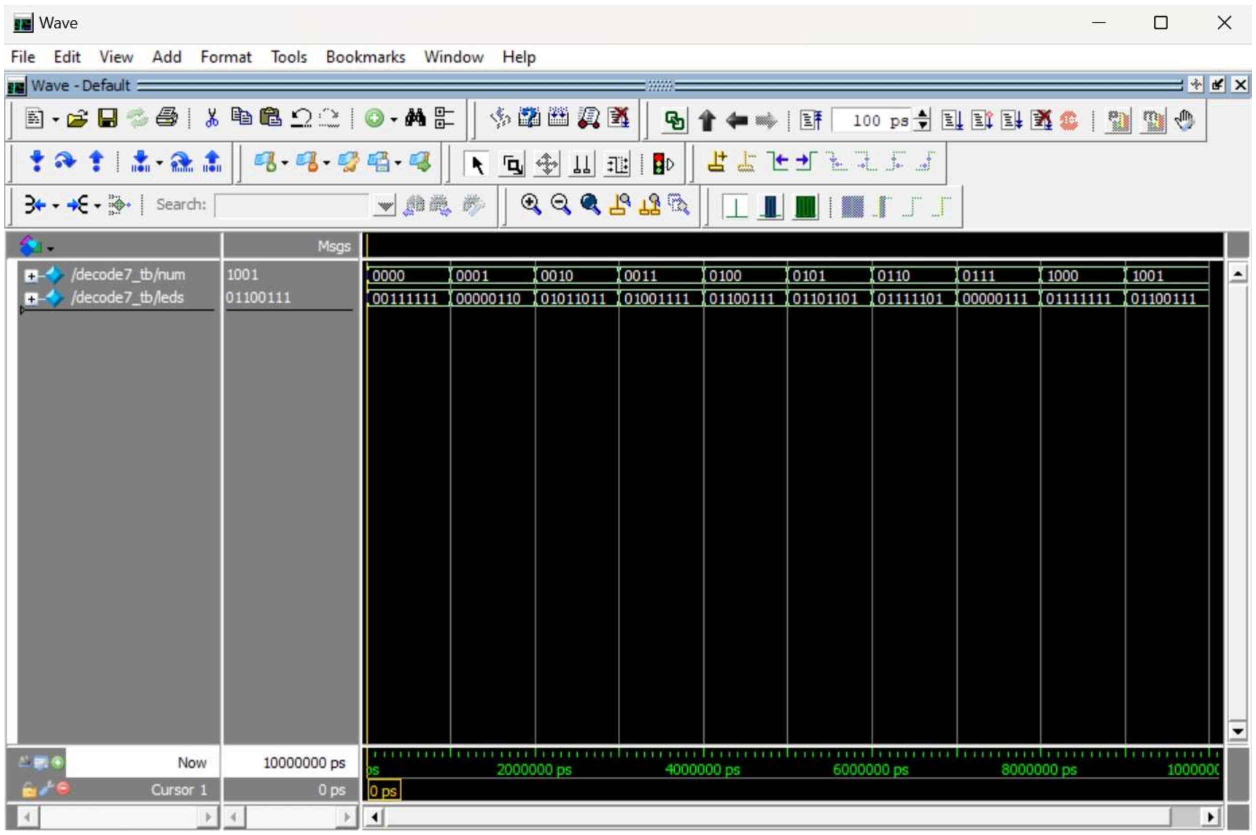
decode7

I started with the most difficult module. I'm a wild man.

```
// ELEX 7660 Lab 1
// Nicholas Scott AKA "White Cheddar"
// A01255181
// Instructor: Sweet Bobby T

module decode7 (input logic [3:0] num,
                output logic [7:0] leds) ;

    // assign leds to the necessary bit configuration to display num on 7-segment (in hexidecimal)
    always_comb
        case(num)
            0 : leds = 'b00111111;
            1 : leds = 'b00000110;
            2 : leds = 'b01011011;
            3 : leds = 'b01001111;
            4 : leds = 'b01100111;
            5 : leds = 'b01101101;
            6 : leds = 'b01111101;
            7 : leds = 'b00000111;
            8 : leds = 'b01111111;
            9 : leds = 'b01100111;
            10 : leds = 'b01110111;     // displays hex character 'A'
            11 : leds = 'b01111100;     // displays hex character 'b'
            12 : leds = 'b00111001;     // displays hex character 'C'
            13 : leds = 'b01011110;     // displays hex character 'd'
            14 : leds = 'b01111001;     // displays hex character 'E'
            15 : leds = 'b01110001;     // displays hex character 'F'
        endcase

endmodule
```

This code simply implements the truth table from the Lab Prep. Craig always taught us that if you write good, legible code, you won't need many comments. This code is very simple, and I only felt the need to comment the lines that implemented the hex digits A-F, as it is may not be obvious what is happening there.

Simulation of this code yielded the following waveforms:



```
/decode7_tb/num   1001      0000     0001     0010     0011     0100     0101     0110     0111     1000     1001
/decode7_tb/leds  01100111  00111111 00000110 01011011 01001111 01100111 01101101 01111101 00000111 01111111 01100111

Now       10000000 ps    ps   2000000 ps   4000000 ps   6000000 ps   8000000 ps   1000000
Cursor 1  0 ps           0 ps
```

Figure 1 - 'decode7' Simulation Waveforms

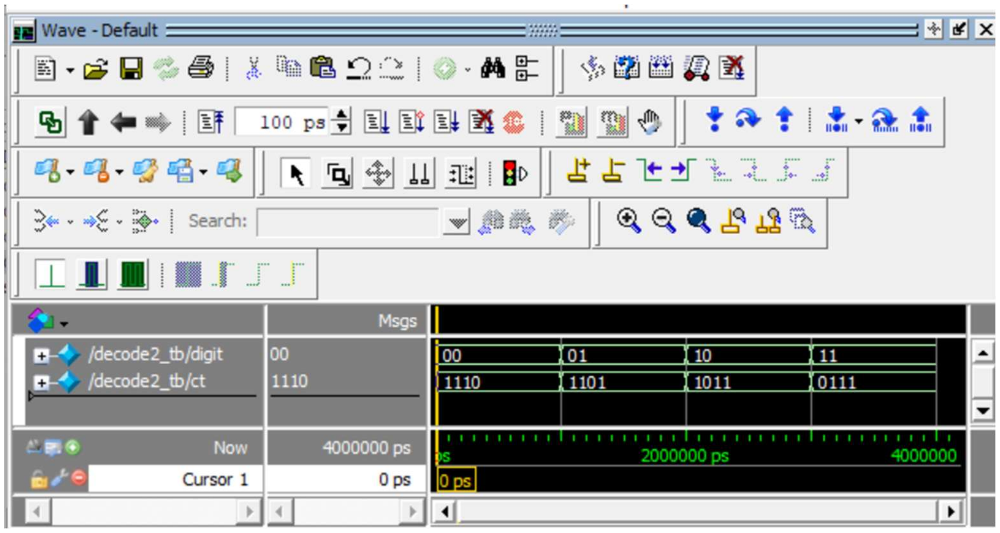
As shown in *Figure 1*, the code implements the truth table as expected. It is important to note that the test bench only tested digits 0-9, which will do for this lab, but for future labs it will be important to remember that hex digits A-F have not been tested yet.

decode2

Since I started with **decode7**, the rest of the coding was simple in comparison. The only thinking involved in the remaining code is ensuring the proper sequence dictated by the *digit* input. The lab document claims that when *digit* is 0, the rightmost digit of the display is active, and when *digit* is 3, the leftmost digit is active.

```
// ELEX 7660 Lab 1
// Nicholas Scott AKA "White Cheddar"
// A01255181
// Instructor: Sweet Bobby T

module decode2 (input logic [1:0] digit,
                output logic [3:0] ct);

    // 'ct' enables one digit of 7-segment display based on 'digit' input
    always_comb
        case(digit)
            0 : ct = 'b1110;    // rightmost digit enabled
            1 : ct = 'b1101;
            2 : ct = 'b1011;
            3 : ct = 'b0111;    // leftmost digit enabled
        endcase

endmodule
```

This code yielded the following simulation result:



```
/decode2_tb/digit   00     00   | 01   | 10   | 11
/decode2_tb/ct      1110   1110 | 1101 | 1011 | 0111
Now                 4000000 ps      2000000 ps     4000000
Cursor 1            0 ps
```

Figure 2 - 'decode2' Simulation Waveforms

Again, the simulation proved that the code effectively implements the truth table.

bcitid

The code for this module is very similar to **decode2**. The same logic applies to the sequencing dictated by the two-bit *digit* input.

```
// ELEX 7660 Lab 1
// Nicholas Scott AKA "White Cheddar"
// A01255181
// Instructor: Sweet Bobby T

module bcitid (input logic [1:0] digit,
                output logic [3:0] idnum);

    // assign '5181' to digits 3 to 0, respectively
    always_comb
        case(digit)
            0 : idnum = 1;  // rightmost digit
            1 : idnum = 8;
            2 : idnum = 1;
            3 : idnum = 5;  // leftmost digit
        endcase

endmodule
```

This code yielded the following simulation result:

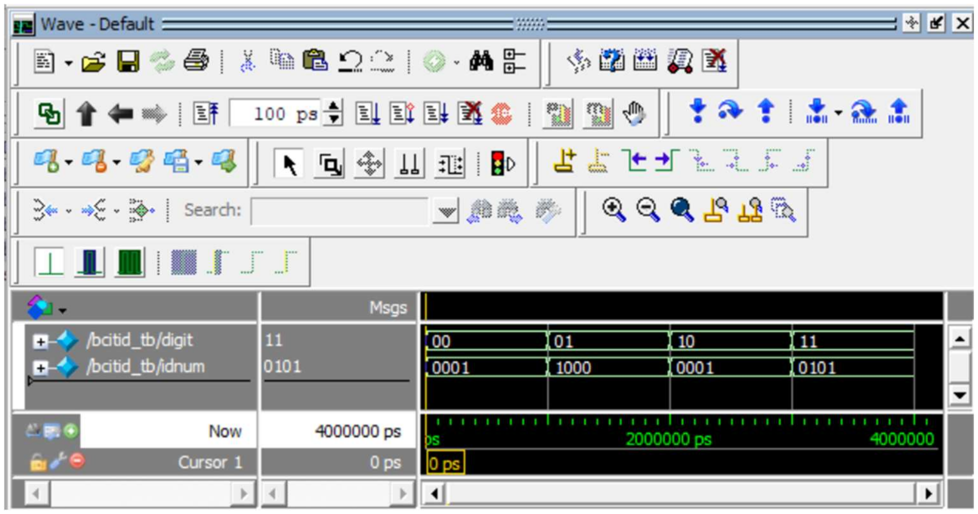
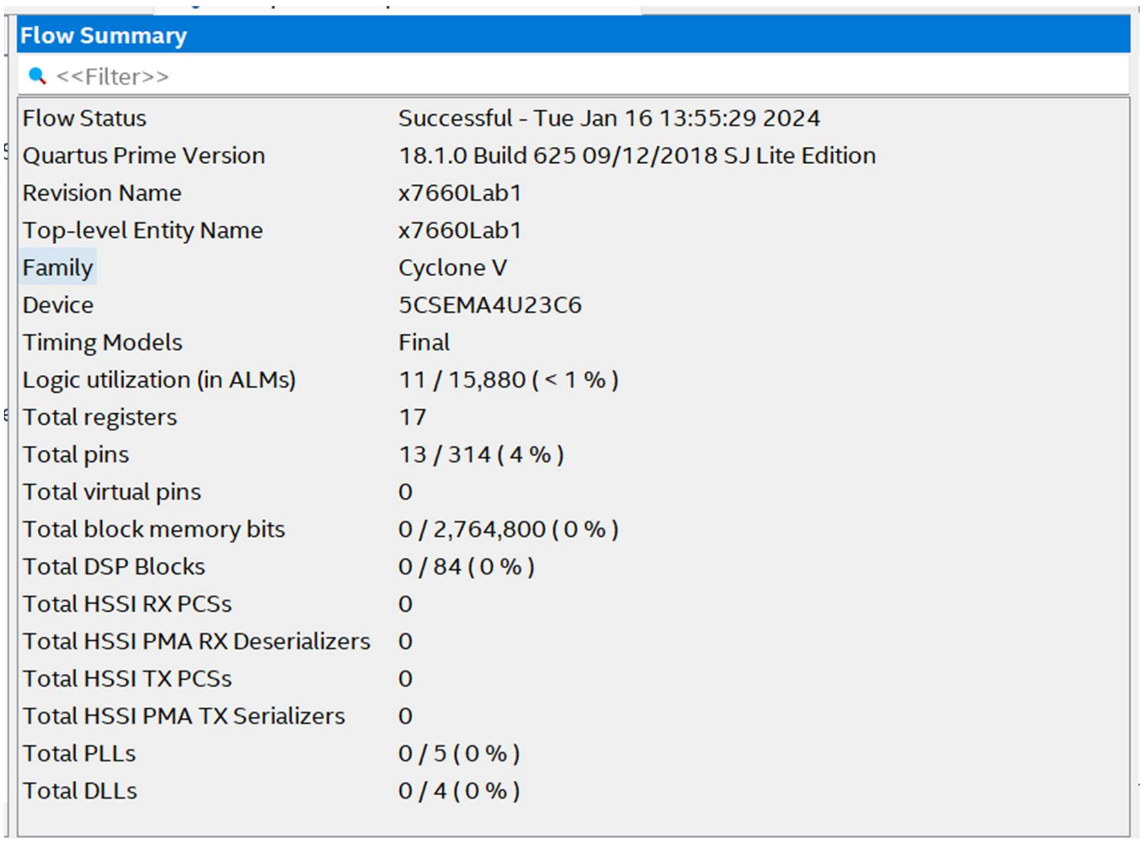


Figure 3 - 'bcitid' Simulation Waveforms

Yet again, the code implements the truth table perfectly. Almost too perfectly if you ask me. Something is bound to go wrong eventually.

Part 2 – Synthesis and Implementation

I imported the files and made some modifications to change the name of the project to *x7660Lab1*. I then was able to compile the code and received the following report:



Flow Summary

<<Filter>>

Flow Status	Successful - Tue Jan 16 13:55:29 2024
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	x7660Lab1
Top-level Entity Name	x7660Lab1
Family	Cyclone V
Device	5CSEMA4U23C6
Timing Models	Final
Logic utilization (in ALMs)	11 / 15,880 (< 1 %)
Total registers	17
Total pins	13 / 314 (4 %)
Total virtual pins	0
Total block memory bits	0 / 2,764,800 (0 %)
Total DSP Blocks	0 / 84 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 5 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 4 - My First Compilation Report

I'm going to have it framed. I then generated the RTL Netlist:

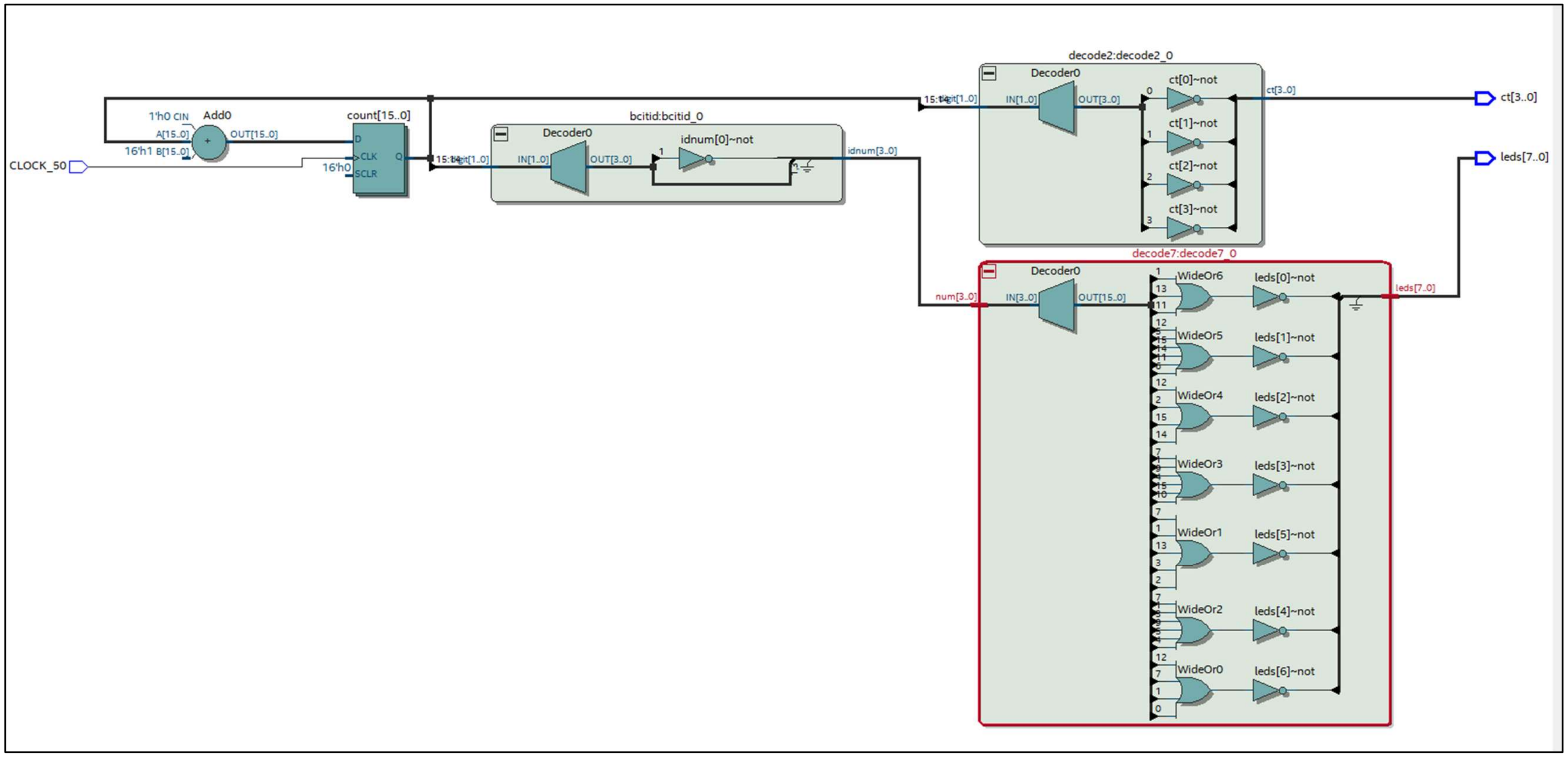


Figure 5 - RTL Netlist

I then uploaded the code and completed the demonstration.