# References on Web Jump out to the interwebby thing

PDF Resources: Designing the Hardware

[1] [Laboratory Exercise 1 Switches, Lights, and Multiplexers](ftp://ftp.altera.com/up/pub/Altera_Material/11.1/Laboratory_Exercises/Digital_Logic/DE1/vhdl/lab1_VHDL.pdf)

[2] [Translated manual for Master 21EDA board](http://organicmonkeymotion.files.wordpress.com/2014/01/altera-fpga-development-board-user-manual.pdf)

[3] [Altera DE1 Board](ftp://ftp.altera.com/up/pub/Altera_Material/12.1/Boards/DE1/DE1_User_Manual.pdf)

Tools

[4] [Digital Electronics Education Design Suite](http://www.esng.dibe.unige.it/Deeds/) (DEEDS)

[5] [LogicFriday](http://sontrak.com/)

Contents

[References on Web  1](#_Toc392105456)

[Introduction 2](#_Toc392105457)

[Part I 4](#_Toc392105458)

[Part II 5](#_Toc392105459)

[Part III 9](#_Toc392105460)

[Part IV 17](#_Toc392105461)

[Part V 25](#_Toc392105462)

[POST SCRIPT 35](#_Toc392105463)

[Part VI 36](#_Toc392105464)

[You may now SCREAM!! 43](#_Toc392105465)

# Introduction

The **Headings in red** in this document will mirror the headings in the Altera® tutorial Ref [1] so you can easily map between documents. You **WILL NEED** Ref [1] at least as this document is only providing the gotchas when walking through Ref [1]. Additional **Headings in blue** are internal to this document – used to break things up as you would expect headings to do.

Read the previous paragraph again. You are reading this document along with the Altera® tutorial [1].

Remember also from the blog, we are now using Quartus® II version 11.1 – driven by the chip on the board, the 144-pin EP2C5T144C8 Cyclone II. The predominant difference is the transitioning from SPOC to Qsys as system on chip designer. Both are available in 11.1, which suits us because there is a lot of free info on web for SOPC based design.

Legend:

If I have been stumped by something I will use the image to the left to let you know a little investigation was in order.

If an important “Ah Ha!” moment occurred, I will also let you know.

If you’re to go to the web I will give the hint.

STOP, we are swapping tutorials

Now don’t forget something very important. Quartus ® II is clunky. Recall from the blog the crashing. What you will find is you may need to delete project and start again a couple of times so be prepared both spiritually and emotionally. You will find the Altera® tutorial leaves things out (which we will try to catch). You will also find, as I did, the tool may not even crash, but will not react to menu selections etc. Just take a deep breath and SCREAM, get over it and try again. Of course, that was while we were using 10.1, the switch to 11.1 may have changed that – we’ll see … whoops, yes there we are (Figure 1).

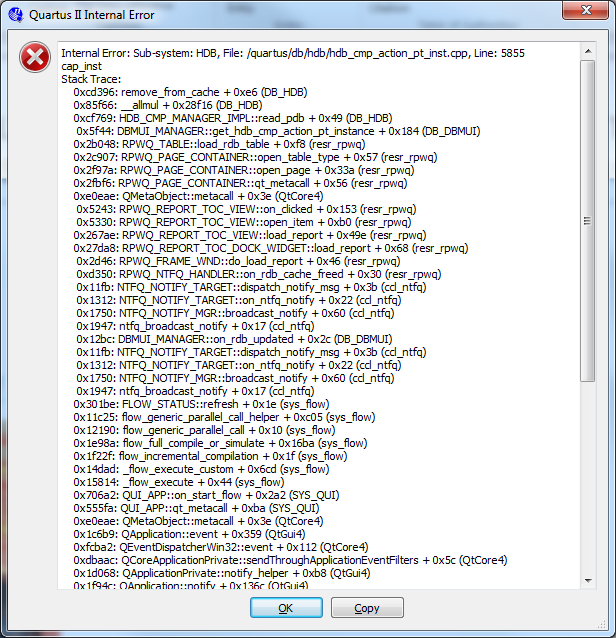


Figure : Same old problem

Don’t forget; as we build projects for each part of the lab remember to set unused pins.

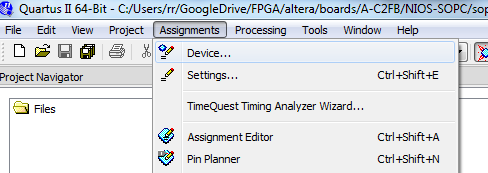


Figure . An important missing step.

Open “Device” dialog (Figure 2) and you will see a button “Device and Pin Options …”, select that (Figure 3). This button doesn’t exist on the dialog when the project is created so you will need to do this as separate step – right now.

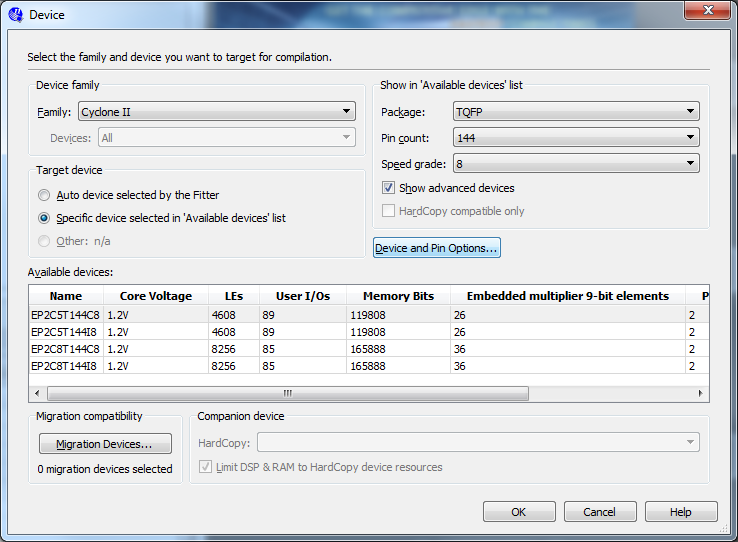


Figure . We need to do something with our unused pins!

Change unused pins to tri-stated inputs (Figure 4).

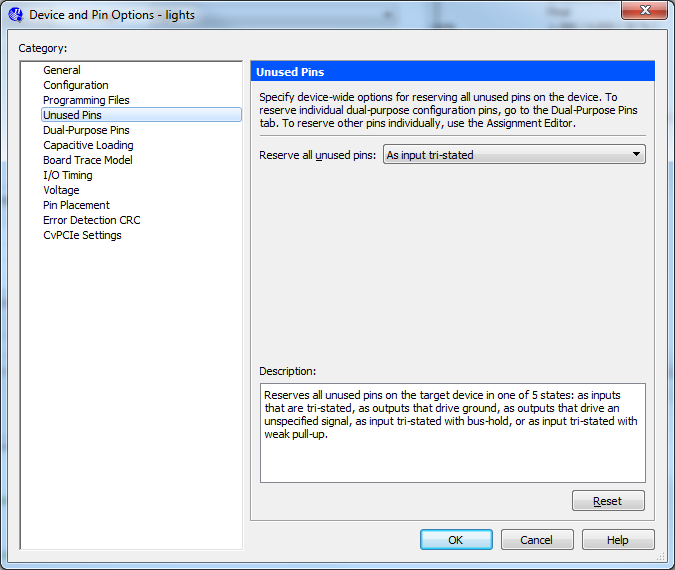


Figure . Tell the pins to be “quiet”

Note also, I will be calling out figures, occasionally, from the Altera® tutorial so I will use Figure x for figures internal to this document and ***Figure y*** when referring to figures in the Altera® tutorial. Similarly, I will use ***Step x***. ***Table x.*** etc. to help remind you to go to the Altera® tutorial.

Ready, set, let’s go.

# Part I

Follow the instructions in Ref [1] but use code the below in Figure 5 which replaces ***Figure 1.*** :

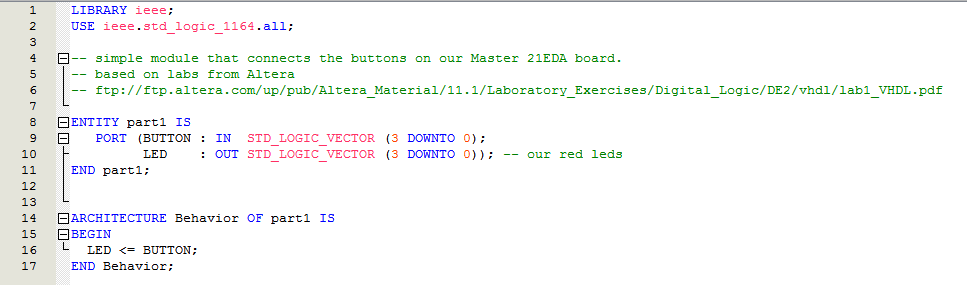


Figure : Our VHDL code in place of lab’s Figure 1.

Note that the tutorial has an error and there is a missing “;” (see red underline Figure 5). Note the change (3 DOWNTO 0) as we only have 4 buttons on our board.

Behaviour will be LEDS OFF until the associated button is pressed.

# Part II

Literal translation of exercise into VHDL (using idiom suggested) is below (Figure 6).

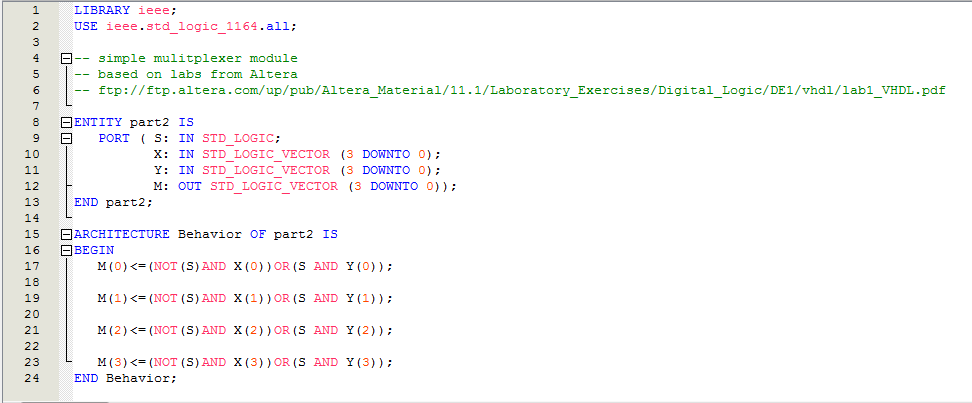


Figure : The Answer for Part II

Our board has too few buttons for this experiment so we need a test harness. The multiplexer “S” signal can be a button, the other inputs need to be set by some test logic. The design of the test logic in DEEDS [4] Digital Circuit Simulator is below (Figure 7). Note change in outputs when we change value of “S”.

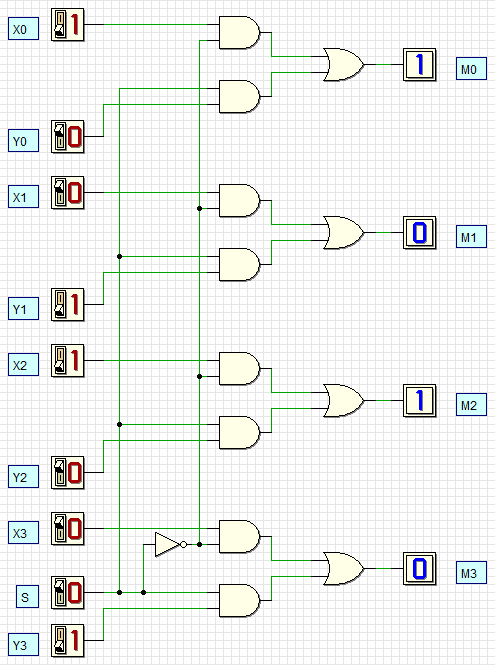
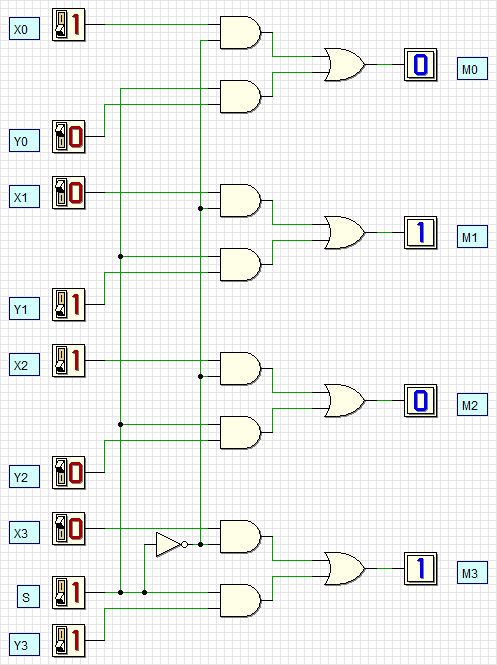
 

Figure : Test harness design for Part II

A pair of signal sources are needed (Figure 8 and Figure 9).

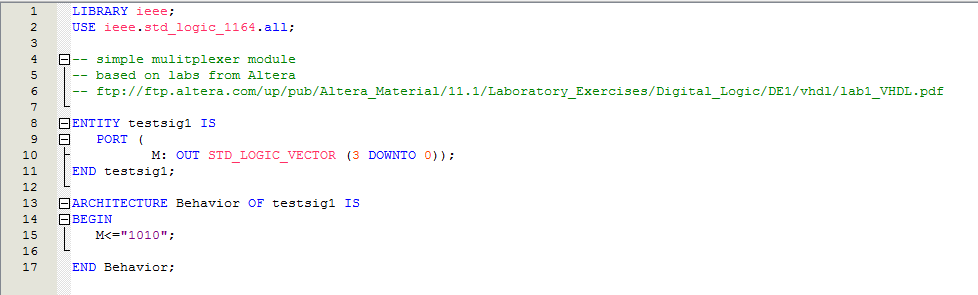


Figure : One lot of test signals

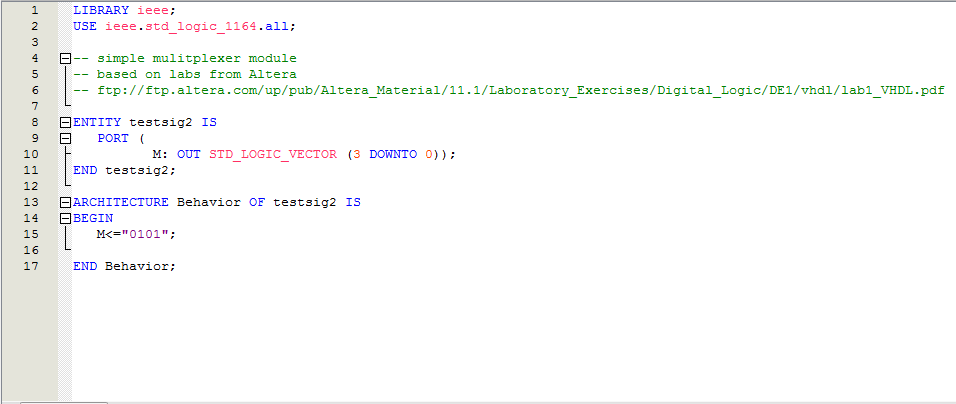


Figure : Second lot of test signals

Create symbols for each of the components by going to the files tab, right clicking a file and selecting the “” menu item (Figure 10).

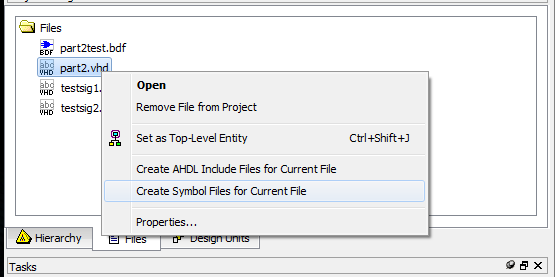


Figure : for each \*.vhd file

This adds symbols in the Project symbol library for you to add to the schematic editor (Figure 11).

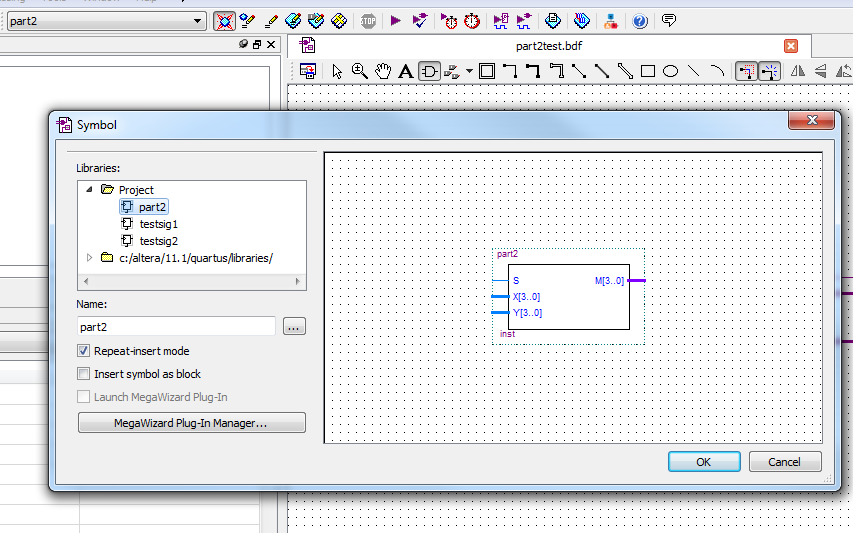


Figure : Project Symbol Library

Then build the test harness as a schematic (Figure 12).

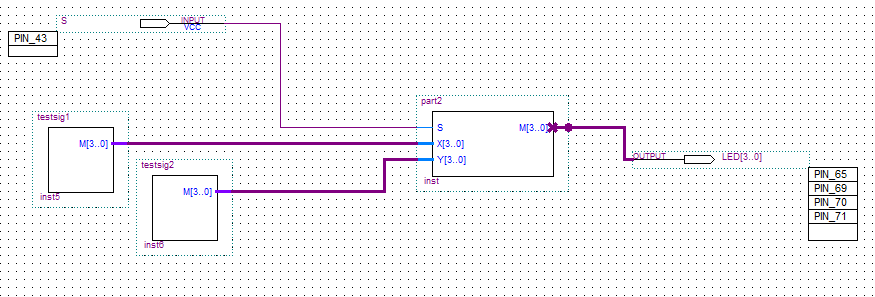


Figure : The test harness for Part II

And don’t forget the pin connections (Figure 13).

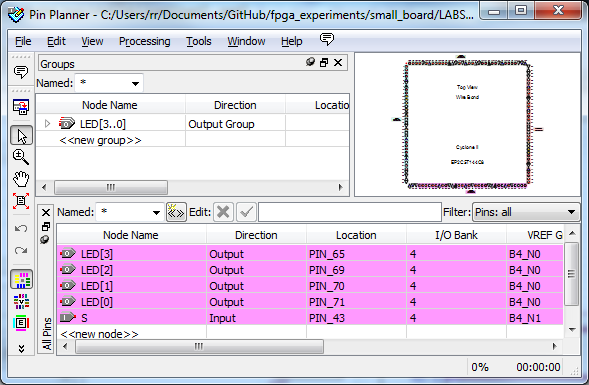
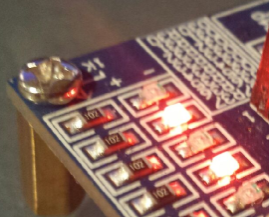
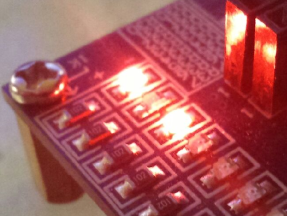


Figure : Pin layout for Part II

Voila!! Now press button on pin 43 on and off a few times to cycle between the two patterns.



# Part III

***Figure 4a*** of Lab 1 is modelled below in DEEDS [4] (Figure 14) (see file part3.pbs).

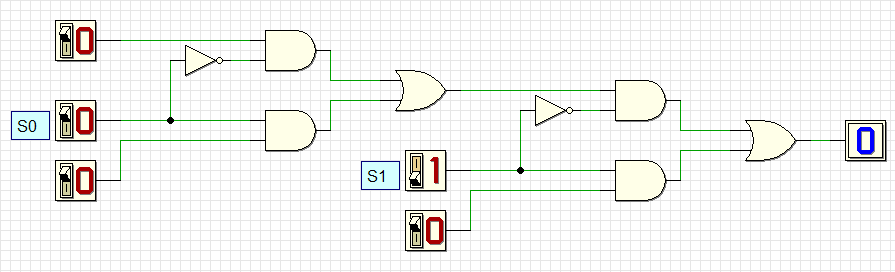


Figure : A one-bit wide 3-to-1 multiplexer

After playing with that, ***Figure 5*** is then modelled in DEEDS [4] below (Figure 15) (see file part3a.pbs)..

|  |  |
| --- | --- |
| S1S0=‘00’, m=u | S1S0=‘01’, m=v |
| S1S0=‘10’, m=w | S1S0=‘11’, m=w |

Figure : A two-bit wide 3-to-1 multiplexer

Literal translation of exercise into VHDL using a select statement is below (Figure 16) with test signal generator (Figure 17).

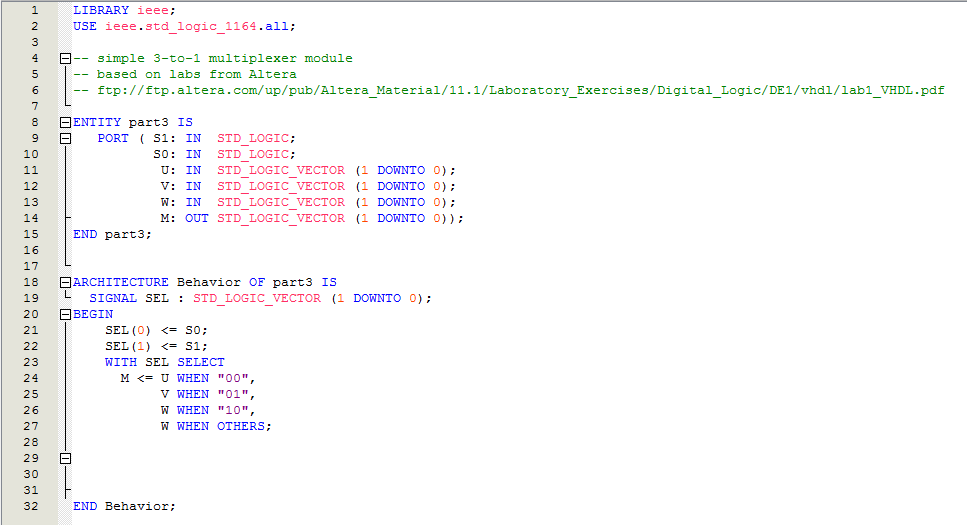


Figure : The Answer for Part III

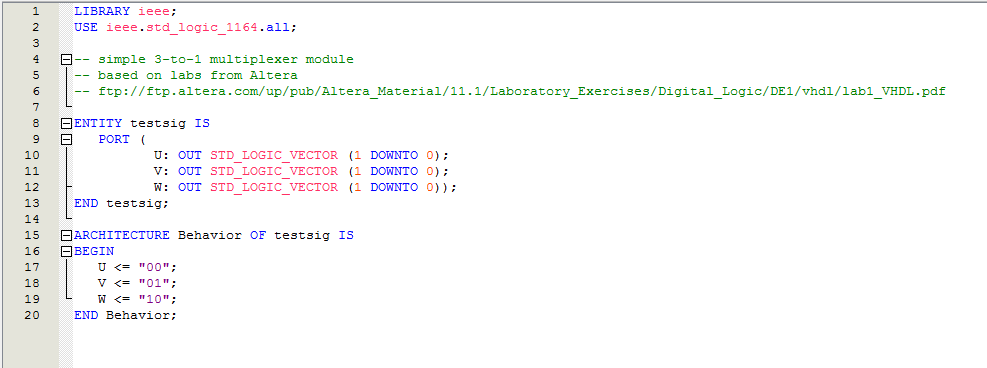


Figure : Test signals for Part III

Then the test harness as a schematic is below (Figure 18).

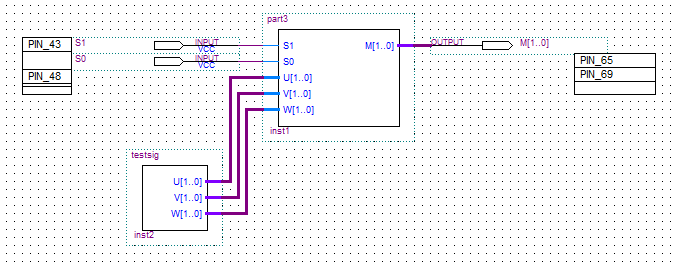


Figure : The test harness for Part III

Go figure, “part3” is interpreted as two 1 bit wide 3-to-1 multiplexers when you check the RTL (Figure 19).

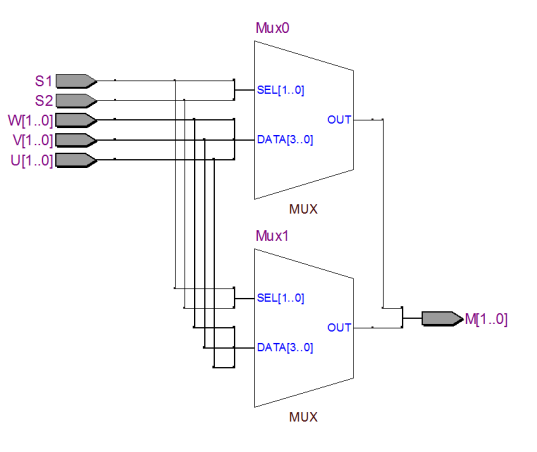


Figure : two 1 bit wide 3-to-1 multiplexers

Using the buttons attached to pins 43 and 48 we should see the LEDs on pins 65 and 69 extinguished until we press one or the other and both LEDs should be lit when both buttons are pressed – as per design intent.

The hard way is taking ***Figure 5*** from the lab (Figure 20 below) literally and doing a truth table.

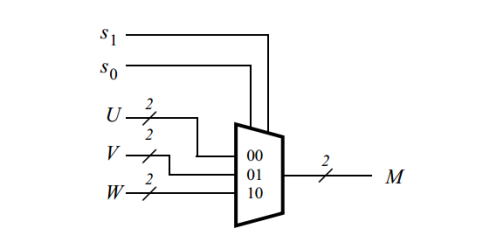


Figure : Target behaviour

We will use LogicFriday [5], converting signal names to those in Table 1 below.

Table : Convert Part III signal names to defaults in Logic Friday

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S1 | S0 | U(0) | U(1) | V(0) | V(1) | W(0) | W(1) | M(0) | M(1) |
| A | B | C | D | E | F | G | H | F0 | F1 |

So using LogicFriday [5] the full truth table (with 8 inputs and 2 outputs) is:

Imported from file:

F0 = A' B' C D' E' F' G' H' + A' B' C D' E' F' G' H + A' B' C D' E' F' G H' + A' B' C D' E' F' G H + A' B' C D' E' F G' H' + A' B' C D' E' F G' H + A' B' C D' E' F G H' + A' B' C D' E' F G H + A' B' C D' E F' G' H' + A' B' C D' E F' G' H + A' B' C D' E F' G H' + A' B' C D' E F' G H + A' B' C D' E F G' H' + A' B' C D' E F G' H + A' B' C D' E F G H' + A' B' C D' E F G H + A' B' C D E' F' G' H' + A' B' C D E' F' G' H + A' B' C D E' F' G H' + A' B' C D E' F' G H + A' B' C D E' F G' H' + A' B' C D E' F G' H + A' B' C D E' F G H' + A' B' C D E' F G H + A' B' C D E F' G' H' + A' B' C D E F' G' H + A' B' C D E F' G H' + A' B' C D E F' G H + A' B' C D E F G' H' + A' B' C D E F G' H + A' B' C D E F G H' + A' B' C D E F G H + A' B C' D' E F' G' H' + A' B C' D' E F' G' H + A' B C' D' E F' G H' + A' B C' D' E F' G H + A' B C' D' E F G' H' + A' B C' D' E F G' H + A' B C' D' E F G H' + A' B C' D' E F G H + A' B C' D E F' G' H' + A' B C' D E F' G' H + A' B C' D E F' G H' + A' B C' D E F' G H + A' B C' D E F G' H' + A' B C' D E F G' H + A' B C' D E F G H' + A' B C' D E F G H + A' B C D' E F' G' H' + A' B C D' E F' G' H + A' B C D' E F' G H' + A' B C D' E F' G H + A' B C D' E F G' H' + A' B C D' E F G' H + A' B C D' E F G H' + A' B C D' E F G H + A' B C D E F' G' H' + A' B C D E F' G' H + A' B C D E F' G H' + A' B C D E F' G H + A' B C D E F G' H' + A' B C D E F G' H + A' B C D E F G H' + A' B C D E F G H + A B' C' D' E' F' G H' + A B' C' D' E' F' G H + A B' C' D' E' F G H' + A B' C' D' E' F G H + A B' C' D' E F' G H' + A B' C' D' E F' G H + A B' C' D' E F G H' + A B' C' D' E F G H + A B' C' D E' F' G H' + A B' C' D E' F' G H + A B' C' D E' F G H' + A B' C' D E' F G H + A B' C' D E F' G H' + A B' C' D E F' G H + A B' C' D E F G H' + A B' C' D E F G H + A B' C D' E' F' G H' + A B' C D' E' F' G H + A B' C D' E' F G H' + A B' C D' E' F G H + A B' C D' E F' G H' + A B' C D' E F' G H + A B' C D' E F G H' + A B' C D' E F G H + A B' C D E' F' G H' + A B' C D E' F' G H + A B' C D E' F G H' + A B' C D E' F G H + A B' C D E F' G H' + A B' C D E F' G H + A B' C D E F G H' + A B' C D E F G H + A B C' D' E' F' G H' + A B C' D' E' F' G H + A B C' D' E' F G H' + A B C' D' E' F G H + A B C' D' E F' G H' + A B C' D' E F' G H + A B C' D' E F G H' + A B C' D' E F G H + A B C' D E' F' G H' + A B C' D E' F' G H + A B C' D E' F G H' + A B C' D E' F G H + A B C' D E F' G H' + A B C' D E F' G H + A B C' D E F G H' + A B C' D E F G H + A B C D' E' F' G H' + A B C D' E' F' G H + A B C D' E' F G H' + A B C D' E' F G H + A B C D' E F' G H' + A B C D' E F' G H + A B C D' E F G H' + A B C D' E F G H + A B C D E' F' G H' + A B C D E' F' G H + A B C D E' F G H' + A B C D E' F G H + A B C D E F' G H' + A B C D E F' G H + A B C D E F G H' + A B C D E F G H;

F1 = A' B' C' D E' F' G' H' + A' B' C' D E' F' G' H + A' B' C' D E' F' G H' + A' B' C' D E' F' G H + A' B' C' D E' F G' H' + A' B' C' D E' F G' H + A' B' C' D E' F G H' + A' B' C' D E' F G H + A' B' C' D E F' G' H' + A' B' C' D E F' G' H + A' B' C' D E F' G H' + A' B' C' D E F' G H + A' B' C' D E F G' H' + A' B' C' D E F G' H + A' B' C' D E F G H' + A' B' C' D E F G H + A' B' C D E' F' G' H' + A' B' C D E' F' G' H + A' B' C D E' F' G H' + A' B' C D E' F' G H + A' B' C D E' F G' H' + A' B' C D E' F G' H + A' B' C D E' F G H' + A' B' C D E' F G H + A' B' C D E F' G' H' + A' B' C D E F' G' H + A' B' C D E F' G H' + A' B' C D E F' G H + A' B' C D E F G' H' + A' B' C D E F G' H + A' B' C D E F G H' + A' B' C D E F G H + A' B C' D' E' F G' H' + A' B C' D' E' F G' H + A' B C' D' E' F G H' + A' B C' D' E' F G H + A' B C' D' E F G' H' + A' B C' D' E F G' H + A' B C' D' E F G H' + A' B C' D' E F G H + A' B C' D E' F G' H' + A' B C' D E' F G' H + A' B C' D E' F G H' + A' B C' D E' F G H + A' B C' D E F G' H' + A' B C' D E F G' H + A' B C' D E F G H' + A' B C' D E F G H + A' B C D' E' F G' H' + A' B C D' E' F G' H + A' B C D' E' F G H' + A' B C D' E' F G H + A' B C D' E F G' H' + A' B C D' E F G' H + A' B C D' E F G H' + A' B C D' E F G H + A' B C D E' F G' H' + A' B C D E' F G' H + A' B C D E' F G H' + A' B C D E' F G H + A' B C D E F G' H' + A' B C D E F G' H + A' B C D E F G H' + A' B C D E F G H + A B' C' D' E' F' G' H + A B' C' D' E' F' G H + A B' C' D' E' F G' H + A B' C' D' E' F G H + A B' C' D' E F' G' H + A B' C' D' E F' G H + A B' C' D' E F G' H + A B' C' D' E F G H + A B' C' D E' F' G' H + A B' C' D E' F' G H + A B' C' D E' F G' H + A B' C' D E' F G H + A B' C' D E F' G' H + A B' C' D E F' G H + A B' C' D E F G' H + A B' C' D E F G H + A B' C D' E' F' G' H + A B' C D' E' F' G H + A B' C D' E' F G' H + A B' C D' E' F G H + A B' C D' E F' G' H + A B' C D' E F' G H + A B' C D' E F G' H + A B' C D' E F G H + A B' C D E' F' G' H + A B' C D E' F' G H + A B' C D E' F G' H + A B' C D E' F G H + A B' C D E F' G' H + A B' C D E F' G H + A B' C D E F G' H + A B' C D E F G H + A B C' D' E' F' G' H + A B C' D' E' F' G H + A B C' D' E' F G' H + A B C' D' E' F G H + A B C' D' E F' G' H + A B C' D' E F' G H + A B C' D' E F G' H + A B C' D' E F G H + A B C' D E' F' G' H + A B C' D E' F' G H + A B C' D E' F G' H + A B C' D E' F G H + A B C' D E F' G' H + A B C' D E F' G H + A B C' D E F G' H + A B C' D E F G H + A B C D' E' F' G' H + A B C D' E' F' G H + A B C D' E' F G' H + A B C D' E' F G H + A B C D' E F' G' H + A B C D' E F' G H + A B C D' E F G' H + A B C D' E F G H + A B C D E' F' G' H + A B C D E' F' G H + A B C D E' F G' H + A B C D E' F G H + A B C D E F' G' H + A B C D E F' G H + A B C D E F G' H + A B C D E F G H;

Mercifully, LogicFriday [5] will factor/minimize it down to:

Factored:

F0 = A' (H' + H) (G' + G) (F' + F) (B' C E' + B C' E) (D' + D) + (A' C E (H' + H) (G' + G) (F' + F) (D' + D) + A G (H' + H) (F' + F) (E' + E) (D' + D) (C' + C)) (B' + B);

F1 = A' (H' + H) (G' + G) (B' D F' + B D' F) (E' + E) (C' + C) + (A' D F (H' + H) (G' + G) (E' + E) + A H (G' + G) (F' + F) (E' + E) (D' + D)) (C' + C) (B' + B);

|  |  |
| --- | --- |
| Minimized:  F0 = A G + A' B' C + A' B E ;  F1 = A H + A' B' D + A' B F ; |  |

Using the LogicFriday [5] mapper of truth table to gates we get Figure 21 below. The way LogicFriday works requires that you select the gate type to be used when turning the minimised truth table into a circuit. I selected NAND and NOR gates.

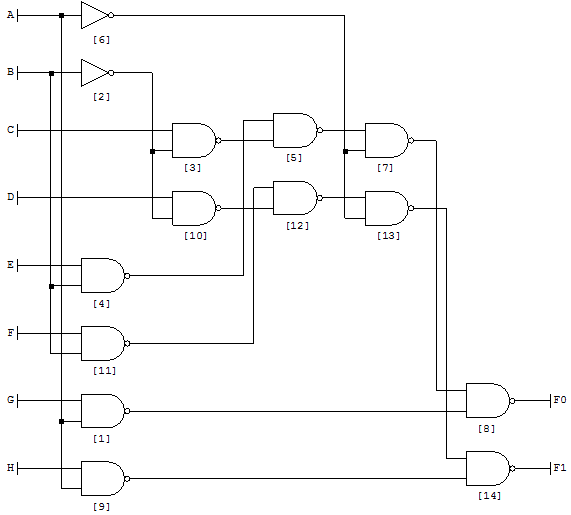


Figure : Circuit design for Part III by LogicFriday

Checking it out with DEEDS [4] we get Figure 22 below (see file part3b.pbs).

|  |  |
| --- | --- |
| S1S0=‘00’, m=u | S1S0=‘01’, m=v |
| S1S0=‘10’, m=w | S1S0=‘11’, m=w |

Figure : Logic Table to circuit conversion

What we need to do now is go back and code up the minimized equation in VHDL:

F0 = A G + A' B' C + A' B E ;

F1 = A H + A' B' D + A' B F ;

Which will become:

M(0) = S1 W0 + S1' S0' V0 + S1' S0 W0 ;

M(1) = S1 W1 + S1' S0' V1 + S1' S0 W1 ;

Code this into VDHL with the following (Figure 23).

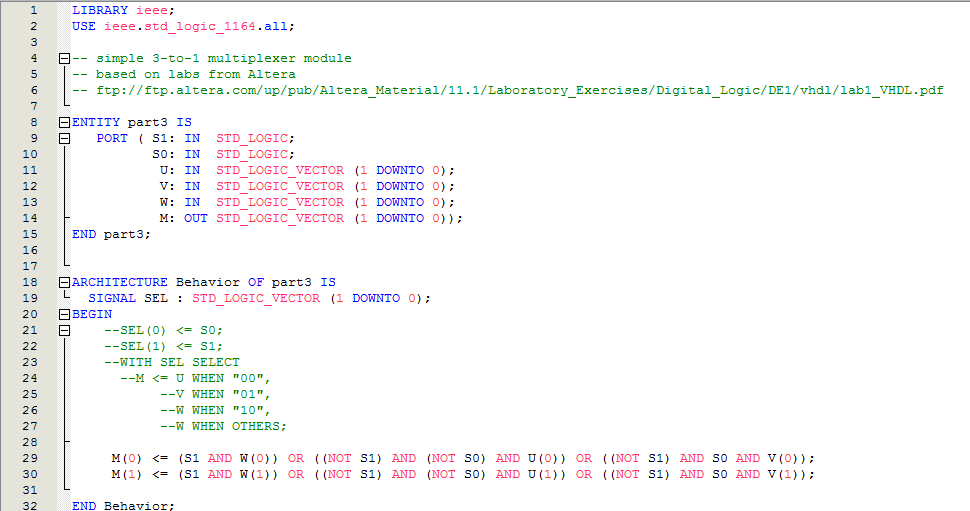


Figure : Part II by equation

This gives us the following RTL (Figure 24).

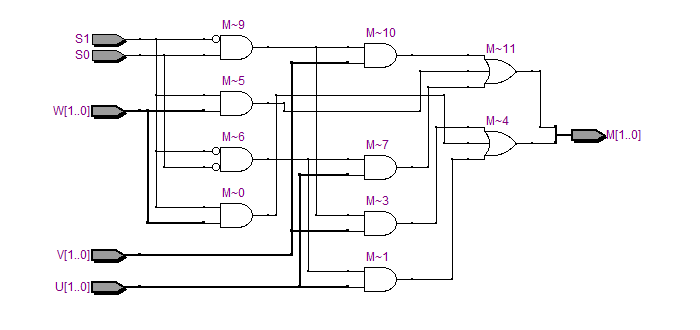


Figure : RTL for Part III by equation

Just for fun we can code that RTL solution into DEEDS [4] to examine that approach (Figure 25) (see file part3d.pbs).

|  |  |
| --- | --- |
| S1S0=‘00’, m=u | S1S0=‘01’, m=v |
| S1S0=‘10’, m=w | S1S0=‘11’, m=w |

Figure : RTL output of equation based approach code in DEEDS

If all goes well the formula based approach should behave the same way as the SELECT based approach.

# Part IV

On the DE1 board these labs were written for the four 7-segments displays called HEX00, HEX01, HEX02 and HEX03 (ref [3]).

With our board we have eight 7 segment LED with the following pins used to enable them (we’ll use LED\_EN8 in this part):

|  |  |  |
| --- | --- | --- |
| LED\_EN1 | OUTPUT | PIN\_94 |
| LED\_EN2 | OUTPUT | PIN\_96 |
| LED\_EN3 | OUTPUT | PIN\_97 |
| LED\_EN4 | OUTPUT | PIN\_99 |
| LED\_EN5 | OUTPUT | PIN\_100 |
| LED\_EN6 | OUTPUT | PIN\_101 |
| LED\_EN7 | OUTPUT | PIN\_103 |
| LED\_EN8 | OUTPUT | PIN\_104 |

The segments are then driven by the following pins with their relation to the display at Figure 26:

|  |  |  |
| --- | --- | --- |
| LED\_A | OUTPUT | PIN\_93 |
| LED\_B | OUTPUT | PIN\_92 |
| LED\_C | OUTPUT | PIN\_87 |
| LED\_D | OUTPUT | PIN\_86 |
| LED\_E | OUTPUT | PIN\_55 |
| LED\_F | OUTPUT | PIN\_58 |
| LED\_G | OUTPUT | PIN\_79 |
| LED\_H(DP) | OUTPUT | PIN\_113 |

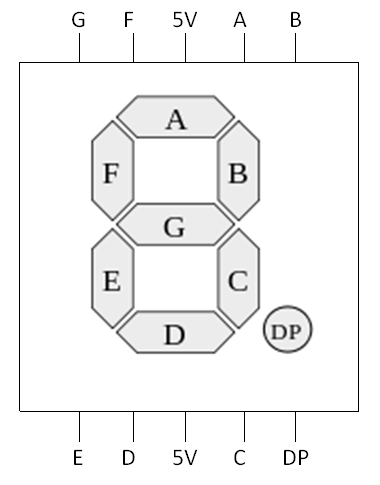


Figure : Typical convention for labelling an LED

The problem statement is below (Figure 27 and Figure 28).



Figure : *Figure 6. A 7-segment decoder.*

|  |  |  |
| --- | --- | --- |
| LED\_A | 0 | PIN\_93 |
| LED\_B | 1 | PIN\_92 |
| LED\_C | 2 | PIN\_87 |
| LED\_D | 3 | PIN\_86 |
| LED\_E | 4 | PIN\_55 |
| LED\_F | 5 | PIN\_58 |
| LED\_G | 6 | PIN\_79 |
| LED\_H(DP) | 7 | PIN\_113 |



Figure : *Table 1. Character codes.*

We map out the segments we need for “dE1” below (Table 2)[[1]](#footnote-1).

Table : The segment maps

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | LED\_A | 0 | PIN\_93 | 1 | | LED\_B | 1 | PIN\_92 | 0 | | LED\_C | 2 | PIN\_87 | 0 | | LED\_D | 3 | PIN\_86 | 0 | | LED\_E | 4 | PIN\_55 | 0 | | LED\_F | 5 | PIN\_58 | 1 | | LED\_G | 6 | PIN\_79 | 0 | | LED\_H(DP) | 7 | PIN\_113 | 1 |   “10100001” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png |
| |  |  |  |  | | --- | --- | --- | --- | | LED\_A | 0 | PIN\_93 | 0 | | LED\_B | 1 | PIN\_92 | 1 | | LED\_C | 2 | PIN\_87 | 1 | | LED\_D | 3 | PIN\_86 | 0 | | LED\_E | 4 | PIN\_55 | 0 | | LED\_F | 5 | PIN\_58 | 0 | | LED\_G | 6 | PIN\_79 | 0 | | LED\_H(DP) | 7 | PIN\_113 | 1 |   “10000110” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png |
| |  |  |  |  | | --- | --- | --- | --- | | LED\_A | 0 | PIN\_93 | 1 | | LED\_B | 1 | PIN\_92 | 0 | | LED\_C | 2 | PIN\_87 | 0 | | LED\_D | 3 | PIN\_86 | 1 | | LED\_E | 4 | PIN\_55 | 1 | | LED\_F | 5 | PIN\_58 | 1 | | LED\_G | 6 | PIN\_79 | 1 | | LED\_H(DP) | 7 | PIN\_113 | 1 |   “11111001” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png |

A little experiment first, code up that in Figure 29, don’t forget the pin allocations Figure 30.

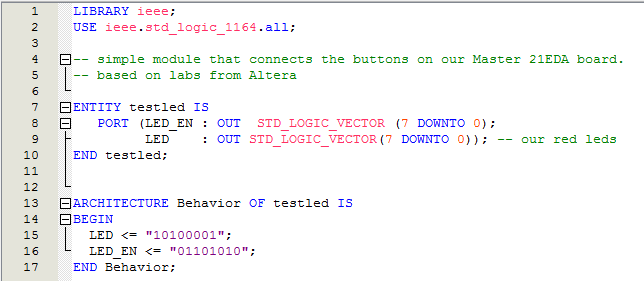


Figure : Test segment lighting

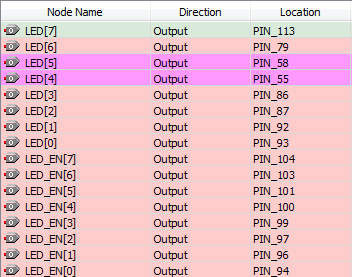


Figure : Pin allocations

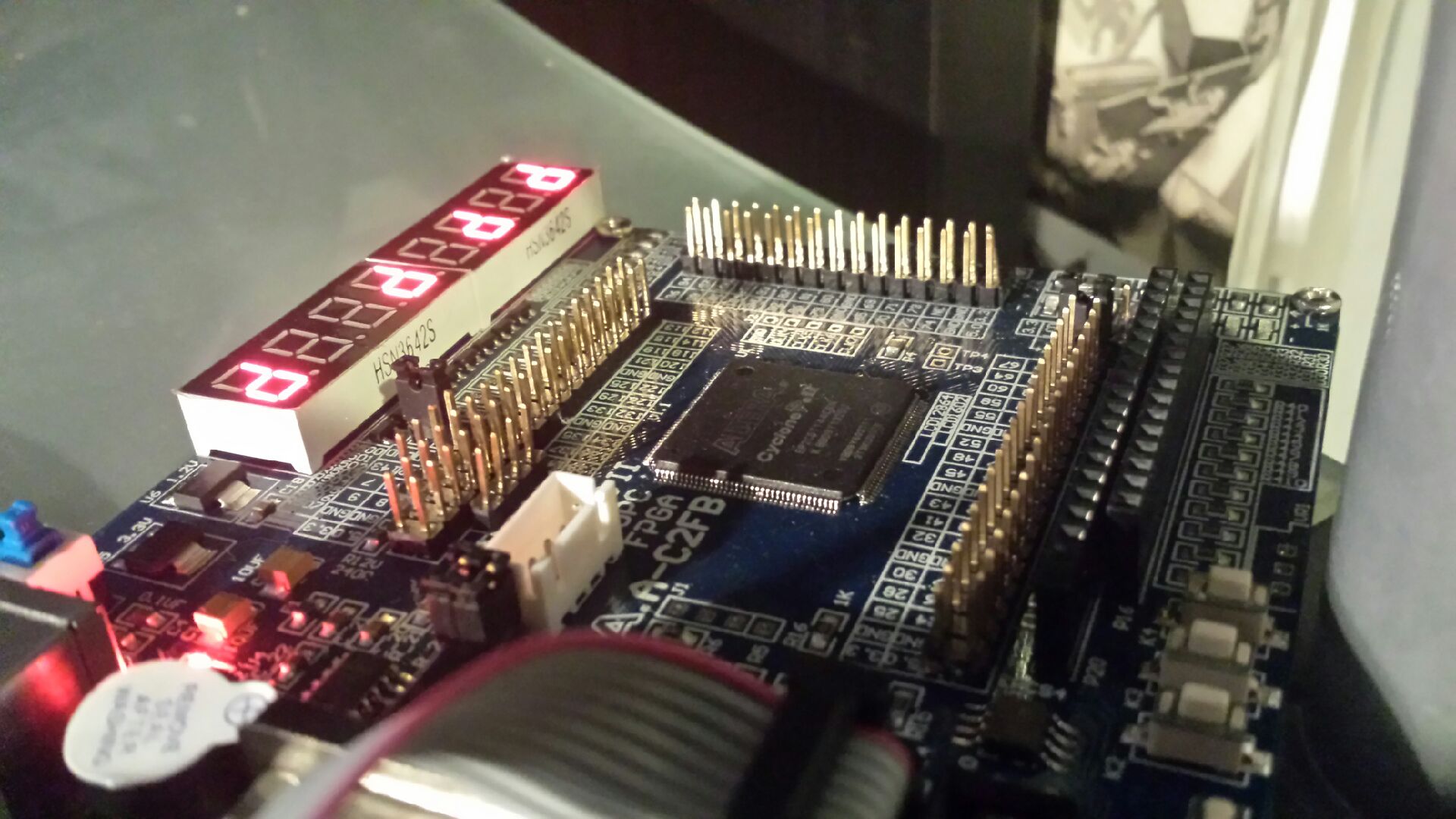
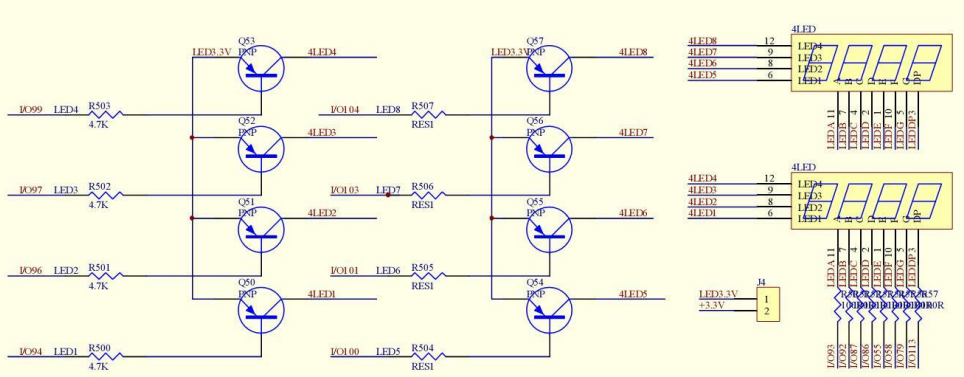


Figure : Resulting segment lighting

Note in Figure 31, where enable is LOW (“0”) segment are lit. Note also where segment is LOW (“0”) segment is lit. This makes sense; the LOW enable drives a PNP transistor which then supplies 3.3V to the segments.

A PNP transistor is "on" when its base (“B”) is pulled low relative to the emitter (“E”).

**+**



**gnd**

**LOW**

**+**

Figure : Low on "N" of PNP turns transistor "ON"

Using LogicFriday then we model the problem as a truth table (Figure 33).

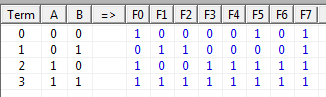


Figure : Lazy LogicFriday

This generates a set of equations which we can then minimize, thus:

Entered by truthtable:

F0 = A' B' + A B' + A B;

F1 = A' B + A B;

F2 = A' B + A B;

F3 = A B' + A B;

F4 = A B' + A B;

F5 = A' B' + A B' + A B;

F6 = A B' + A B;

F7 = 1;

Minimized:

F0 = B' + A ;

F1 = B;

F2 = B;

F3 = A ;

F4 = A ;

F5 = B' + A ;

F6 = A ;

F7 = 1;

Tablewize this looks like the figure below (Figure 34).

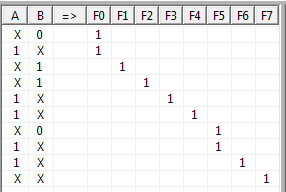


Figure : Minimized truth table

So coding it up (remembering to add button inputs) we get the code below (Figure 35). Note the LED\_EN setting - we are using LED8 which is described in our board’s manual as “the left one”. You can then see the relationship between our seven segment display layout and the bit positions – neat yes?

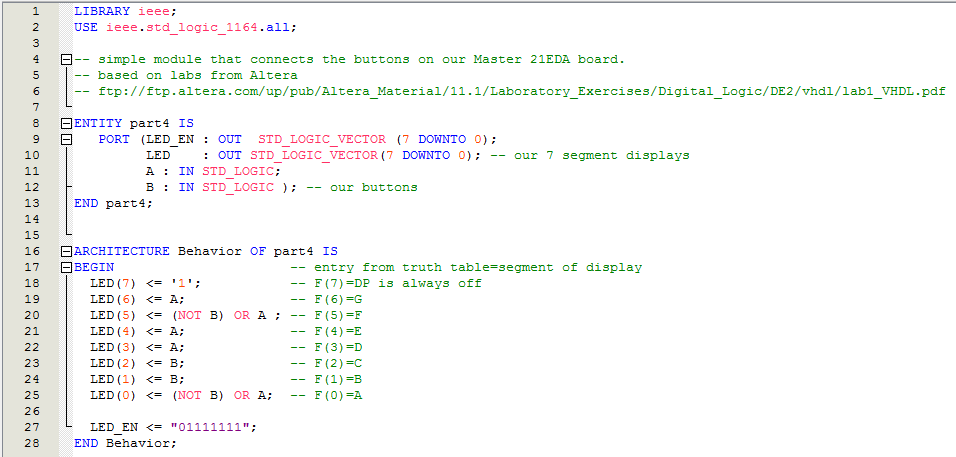


Figure : Part IV solution

Full pin allocation is below (Figure 36).

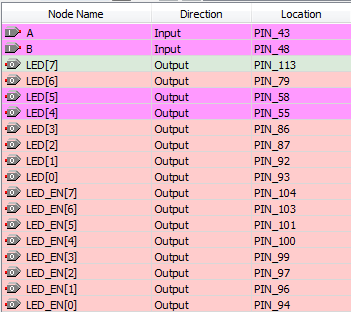


Figure : Full pin allocation

The display selected should have all segments extinguished when the FPGA is loaded with image. This makes sense because when the buttons (K1 and K2) are not pressed the corresponding inputs (PIN\_43 and PIN\_48) are held high by resisters – giving ‘11’ as the inputs for “A” and “B” (refer back to Figure 28).

A peak at the gate model built by LogicFriday (limited to inverter and 2 input NAND gates) is below (Figure 37).

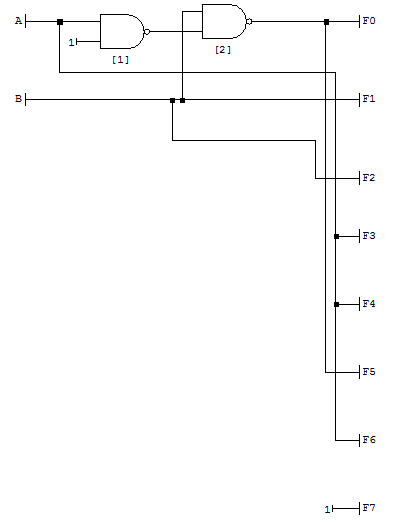


Figure : Simple enough

Just for fun here is the RTL model from Quartus II ® (Figure 38).

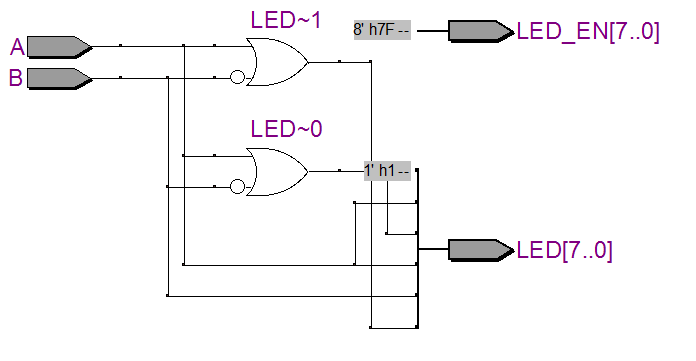


Figure : Part IV RTL

# Part V

Okay, so now the bad news. Out board diverges from DE1 in an important way here. Our board (left panel Figure 39) has shared segment drivers (for all eight displays) each display with a separate enable. The DE1 (right panel Figure 39) has individual segment drivers and essentially no enables.

Both work essentially the same, the anode is held high with the cathode driven low to light the segment. We enable the anode by driving a signal low on our PNP transistors to drive the anode to +3.3V and light the segment by pulling the relevant cathode low (see Figure 32). On the DE1 board, all the anodes are held at VCC (+ve) and the cathodes are pulled low across 330ohm resisters (RN7 and RN8 for example).

|  |  |
| --- | --- |
| **Master 21EDA Board** | **DE1 7-SEGMENT Design** |
|  |  |

Figure : Spot the difference

So, that means something different for our circuit design. Target behaviour as per ***Table 1. Character codes*** of Altera ® tutorial [1] (Figure 40).



Figure : The desired outputs

What they want for the DE1 is something like the following (Figure 41):

|  |  |
| --- | --- |
| Note: See the change in allocation of DE1 SW(x-x) across the mux2bit\_3to1, namely:  ‘d’ = SW5-4  ‘E’ = SW3-2  ‘1’ = SW1-0  This trick allows the DE1 owners to set the ‘dE1’ in the switches and let the 3 instances of mux2bit\_3to1 sort out the rotation. |  |

Figure : The DE1 Solution

Coding up for the DE1 board you get the following (Figure 42):

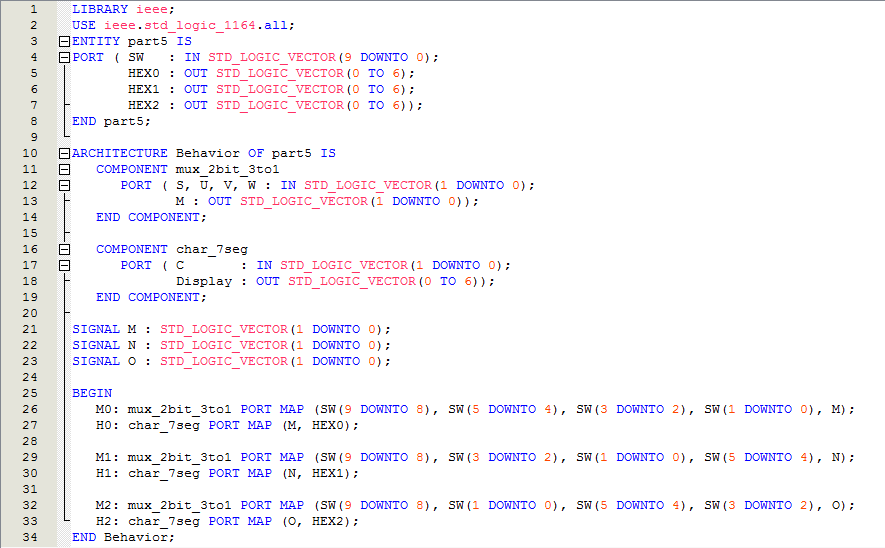


Figure : Coded up for DE1 board

Remember, however, the DE1 has 4 seven segment displays - driving the segments directly via HEX0, HEX1, HEX2 and HEX3. That is, 28 segment driving pins, of which Part IV lab uses 21 (look again at Figure 39 or at ref [3]).

Code up the DE1 solution (Figure 42) and look at the RTL (Figure 43). Use the mouse to hover over connections to see this working. For example, hover over the V[1..0] of M2. The ‘d’, or SW[5:4] is where we expect it (look back at previous page and Figure 42 line 32) – that is the middle character.

Try hovering over other mux inputs.

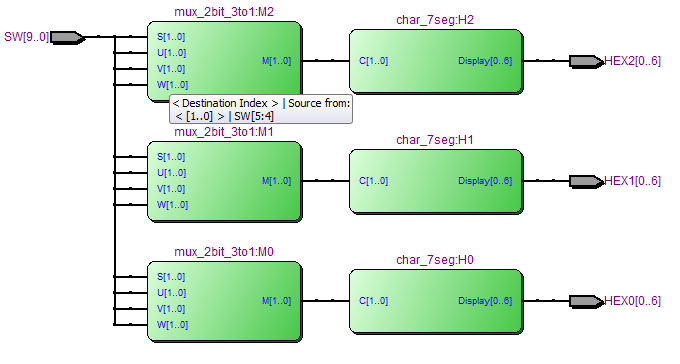


Figure : RTL to DE1 Solution

For our M21EDA board however, the circuit will have to be more like the following (Figure 44):

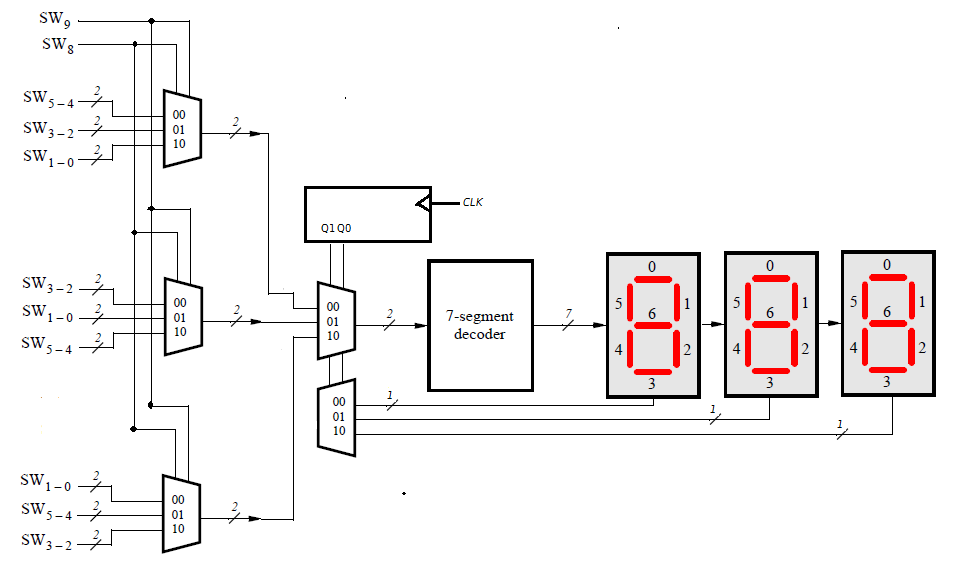


Figure : Our version of Part V Solution

So our version of the experiment uses an extra mux2bit\_3to1, a counter of some description and a 1-of-3 decoder.

Our mux is from Part III (Figure 45).

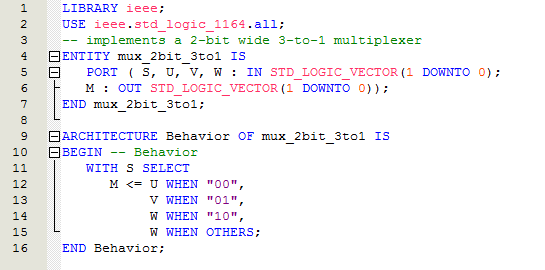


Figure : mux for Part V from Part III

Our 7-segment decoder is changed a little but still recognisable (Figure 46). Change is because we are using a vector instead of two separate signals for the select input (A+B=C), and we are using TO instead of DOWNTO for the segment vector “Display”.

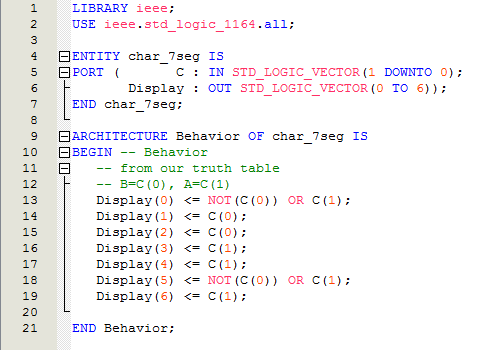


Figure : Slight mod from Part IV

For the main code we’ll have to move away from the original input signature (Figure 47 lines 4..7) – but not too much. I decided to keep the counter and decoder “outside” of the core example code. Notice how we “glued” in the extra mux (at line 34).

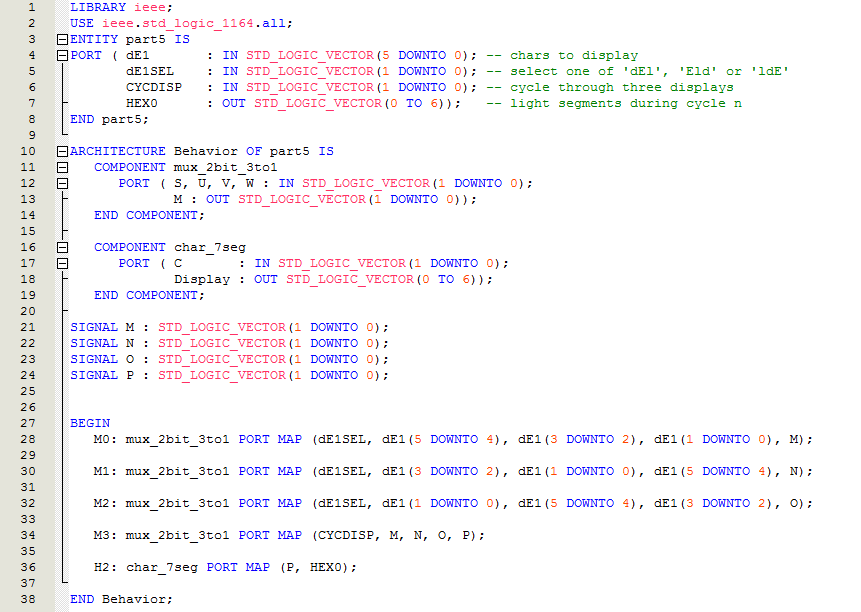


Figure : DE1 code modified for our M21EDA board

Checking the RTL (Figure 48) we get part of our intended design (Figure 49).

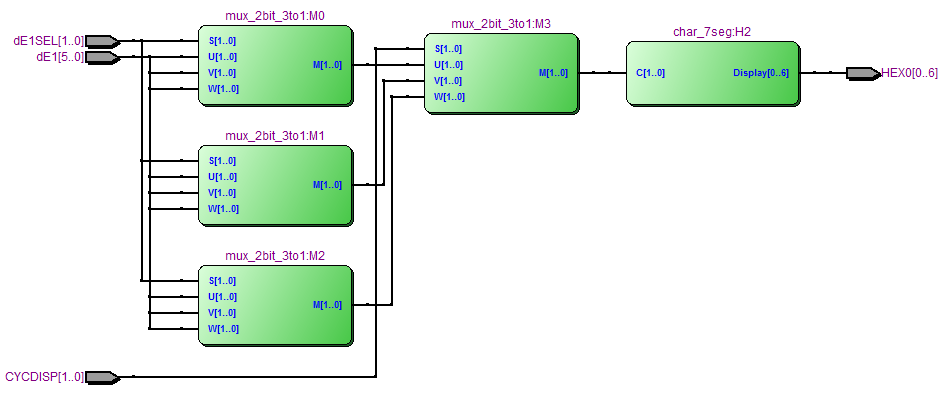


Figure : RTL of our intermediate design

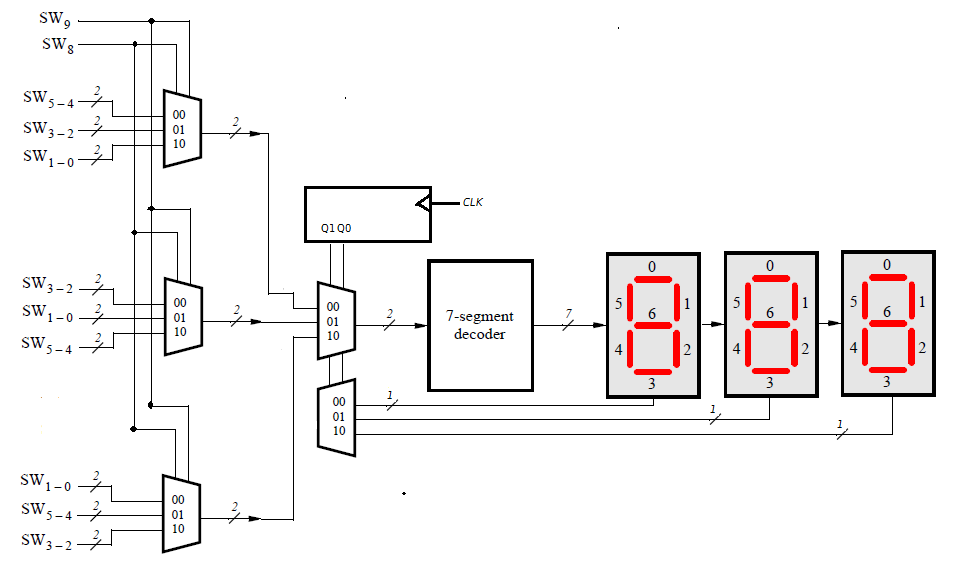
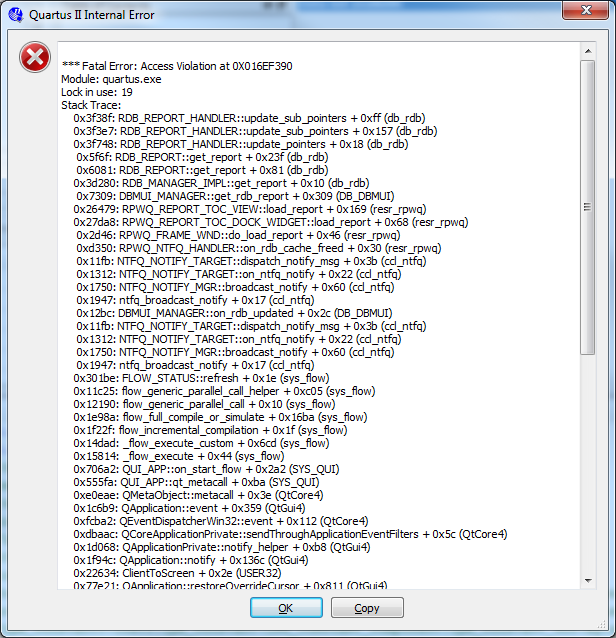


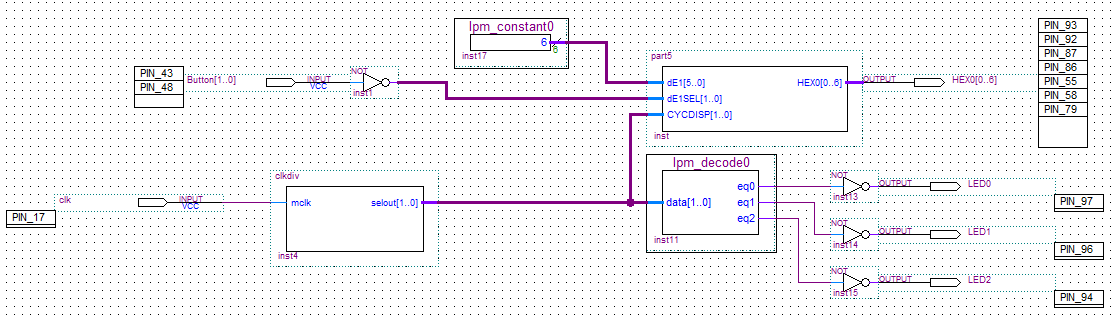
Figure : The implementation so far

Go figure, another random kerplunk!



In any event, the fully assembled experiment is below (Figure 50) – again we had to take some licence/liberty to fit this DE1 example onto the M21EDA board.

**1**



**2**

**4**

**3**

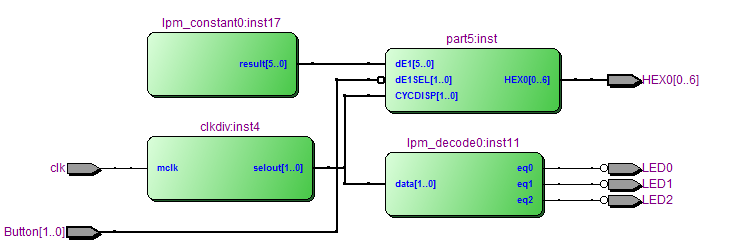
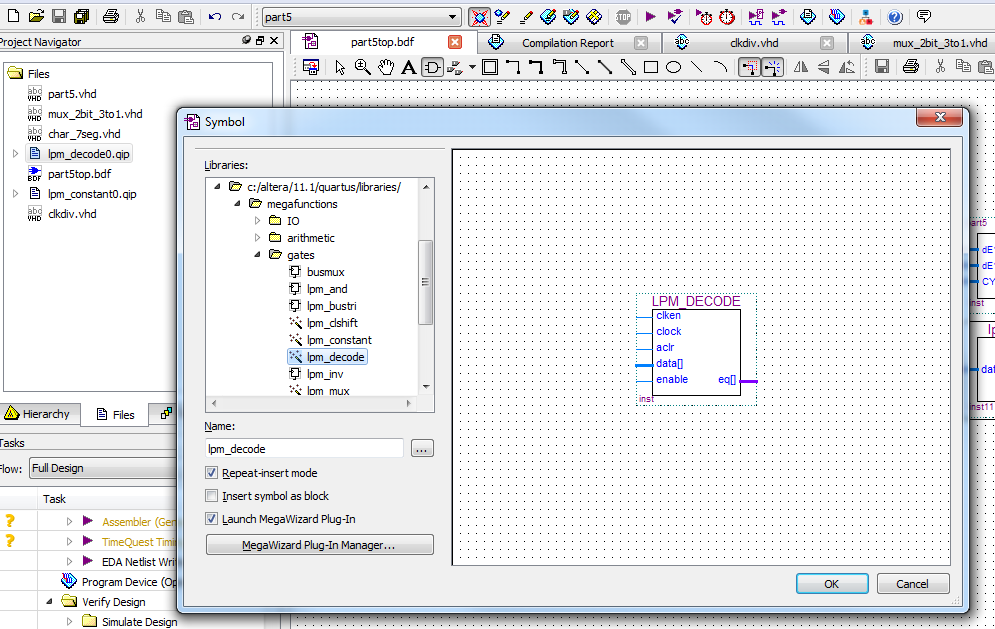


Figure : Full Part V solution

The inverter at the Button input (Figure 50 item 1) is to invert the logic of our buttons (held high until pressed), this means default un-pressed button input is ‘00’ instead of ‘11’.

To play with some of the options available we use items from the libraries (Figure 51 item 1) I built the segment scanner (Figure 50 item 2) from the “megafunction gates” (Figure 51 item 2), which auto generates code (Figure 51 item 3). The decoder options I picked are at Figure 52. I added inverters to the outputs as the enable needs to be low.



**3**

**2**

**1**

1

Figure : Library wizards

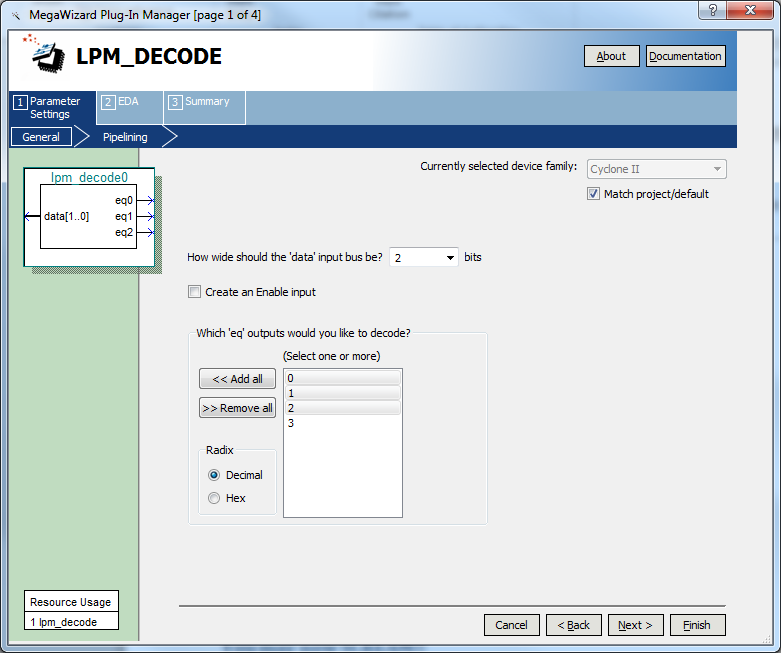


Figure : Wizard options for decoder/scanner

I just coded up a quick and dirty divider counter with display sweep “selout” (Figure 50 item 3, code below at Figure 53) to drive the display sweep decoder (Figure 50 item 2).

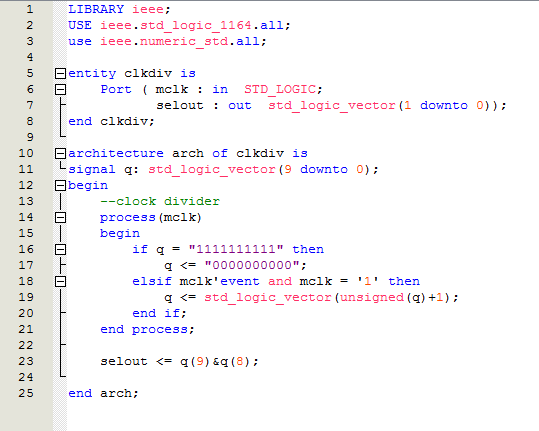


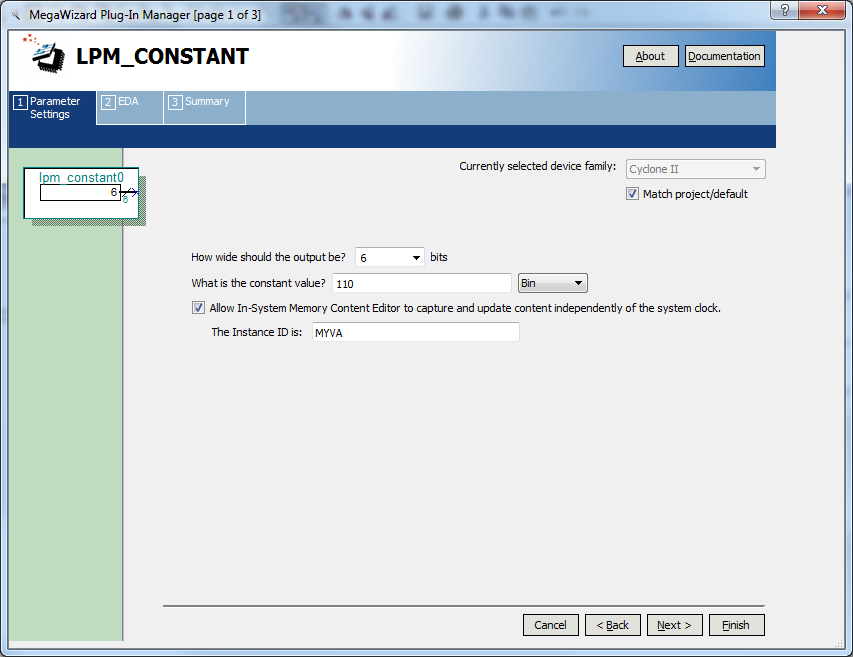
Figure : Slow down clk and provide a display sweep

 From Figure 51 line 19, not immediately obvious is that you cannot increment a std\_logic\_vector. A little web search found that you had to type cast it to unsigned and then re-cast it back to std\_logic\_vecor. This code was from an example on the web that would not compile without that cast applied – don’t assume sample code on the interwebby thing works.

I tried a few library options to get a scanner going, megafuntions, 74193 etc., but settled on hand coding a solution. There will likely be a library solution to this, it just needed a little more investigation.

Note: Inside of a process statement (Figure 51 line 14..21), statements are executed sequentially which allows us to introduce sequencing and control. This is more like a software programming idiom than the concurrent behaviour of VHDL code generally.

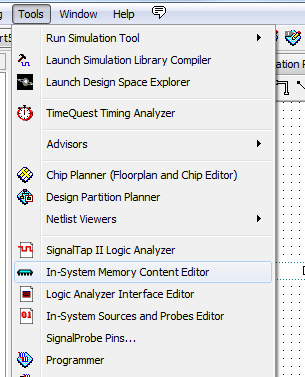
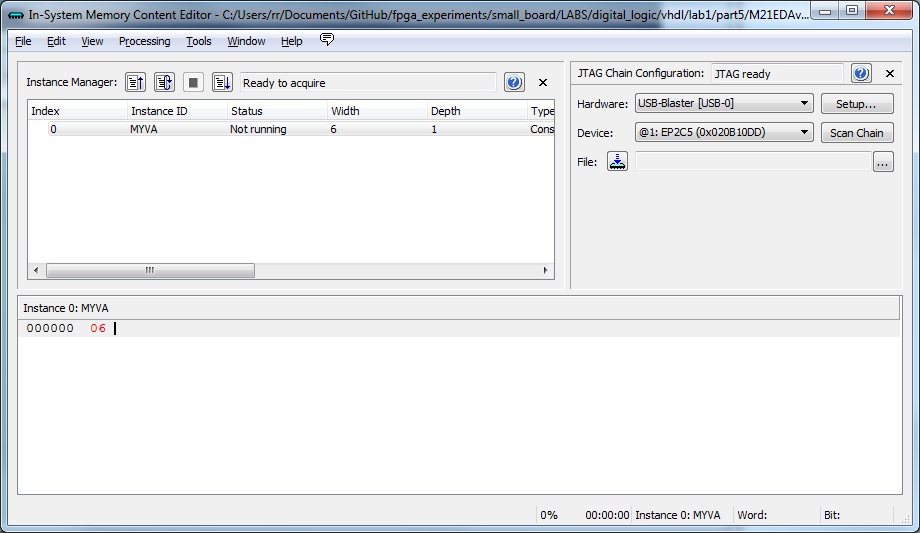
I used a “constant” wizard from the megafunctions (Figure 54) to create the missing DE1 switches (SW[5..0]) that set up the character string for display (Figure 50 item 4). The neat thing is that you can a set a flag (Figure 54 item 1) so a named value (Figure 54 item 2) can be tuneable while running it on the target board using the “In-system Memory Content Editor” (Figure 55 item 1) to read or write to the named value (Figure 55 item 2).



**2**

**1**

Figure : Define SW[5..0] as a constant

**2**

**1**

Figure : Talk to the variables on the running board

## POST SCRIPT

To better map to the DE1 board, for the displays at least, a 7-bit 4-to-1 mux with a 1-of-4 decoder would be a good solution to mimic the DE1 display (Figure 56).

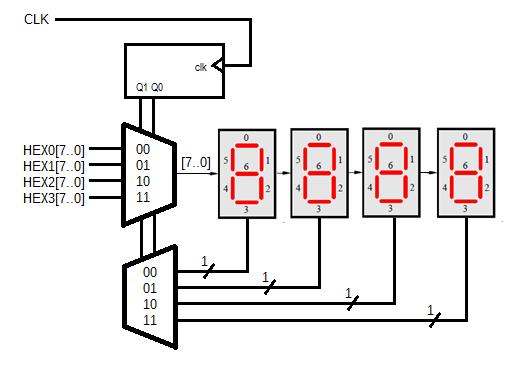


Figure : Mimic DE1 displays

Something for Part VI.

# Part VI

Okay, final stretch.

We’ll go for broke and personalize a little. So, first we will recode displays to read “EdA” and only use 7 (not 8) signals (

Table 3) – since our board is the M21EDA, thus:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | *SW9 SW8* | Character Pattern | | | |
| 00 |  | E | d | A |
| 01 | E | d | A |  |
| 10 | d | A |  | E |
| 11 | A |  | E | d |

Table : Characters for our M21EDA board

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | LED\_A | 0 | PIN\_93 | 0 | | LED\_B | 1 | PIN\_92 | 1 | | LED\_C | 2 | PIN\_87 | 1 | | LED\_D | 3 | PIN\_86 | 0 | | LED\_E | 4 | PIN\_55 | 0 | | LED\_F | 5 | PIN\_58 | 0 | | LED\_G | 6 | PIN\_79 | 0 |   “0000110” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png |
| |  |  |  |  | | --- | --- | --- | --- | | LED\_A | 0 | PIN\_93 | 1 | | LED\_B | 1 | PIN\_92 | 0 | | LED\_C | 2 | PIN\_87 | 0 | | LED\_D | 3 | PIN\_86 | 0 | | LED\_E | 4 | PIN\_55 | 0 | | LED\_F | 5 | PIN\_58 | 1 | | LED\_G | 6 | PIN\_79 | 0 |   “0100001” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png |
| |  |  |  |  | | --- | --- | --- | --- | | LED\_A | 0 | PIN\_93 | 1 | | LED\_B | 1 | PIN\_92 | 0 | | LED\_C | 2 | PIN\_87 | 0 | | LED\_D | 3 | PIN\_86 | 1 | | LED\_E | 4 | PIN\_55 | 1 | | LED\_F | 5 | PIN\_58 | 1 | | LED\_G | 6 | PIN\_79 | 1 | | LED\_H(DP) | 7 | PIN\_113 | 1 |   “0001000” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png |

Coding this up in a truth table gives us the following (Figure 57).

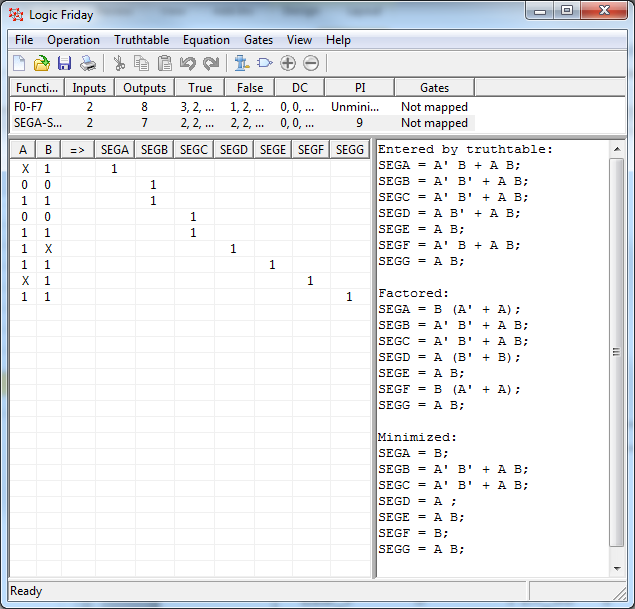


Figure : EdA for our M21EDA board

From which the following code falls out (Figure 58).

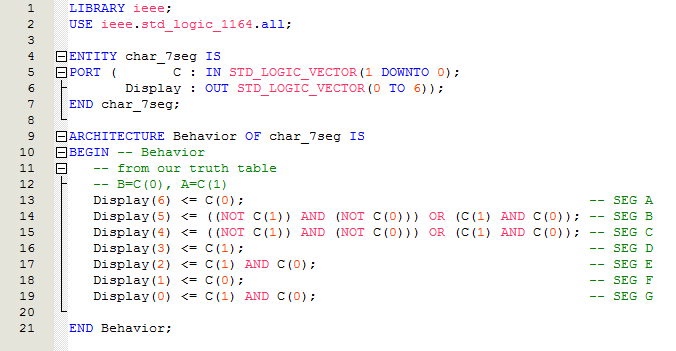


Figure : New char\_7seg

Of course, there is a subtle change to the mux (3-to-1 to 4-to-1) (Figure 59).

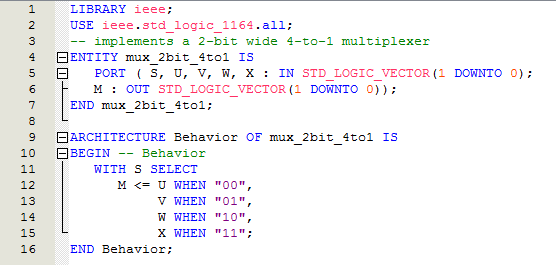


Figure : Modified mux

Basic code to pull it together is the following (Figure 60):

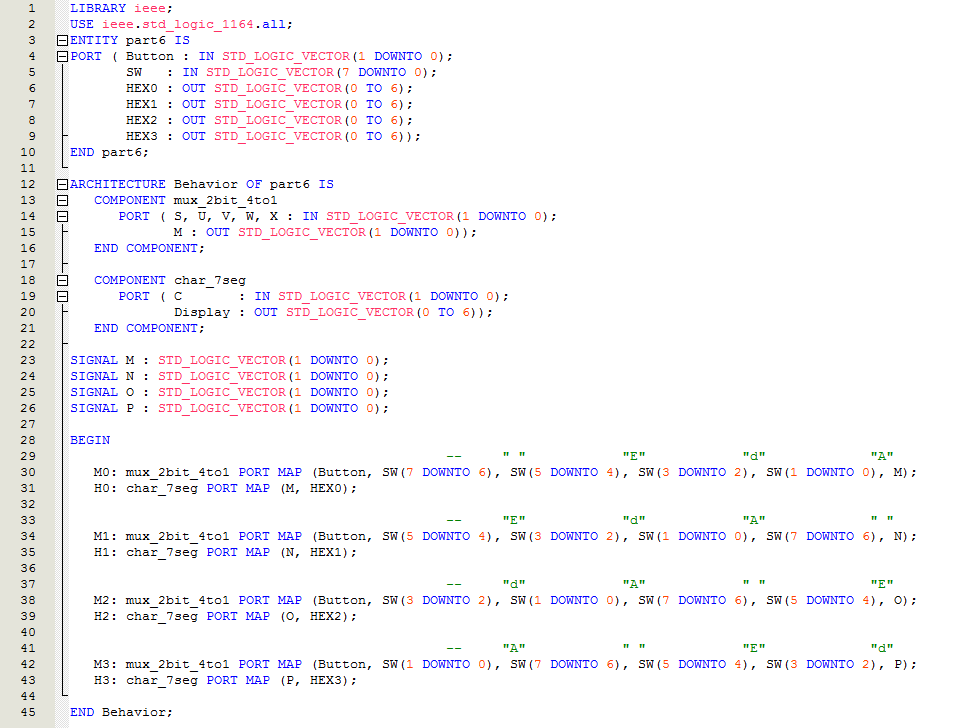


Figure : Main functionality

Notice we again split out the physical buttons from the “switches” that will be used to set the default characters “ MdA” – which will again be “jumbled” between the four mux (Figure 60 lines 30, 34, 38, and 42).

Of course, in Part V we finished with the idea that we might make the M21EDA displays behave more like the DE1 so we can read labs across more readily - which leads to the following code (Figure 61).

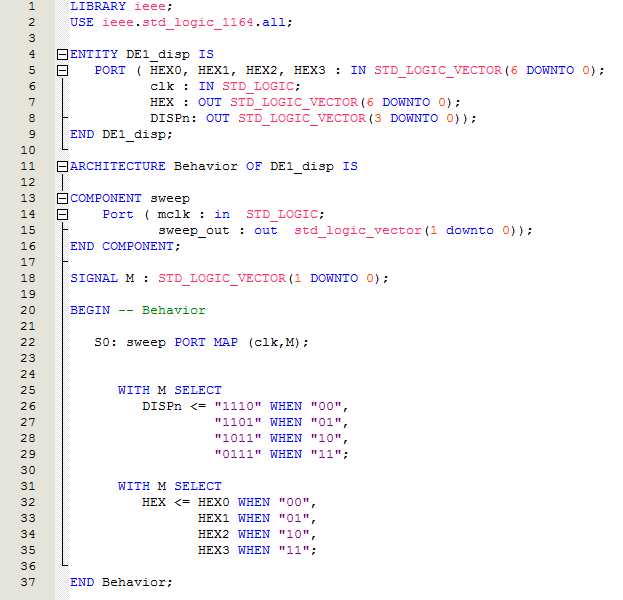


Figure : Impersonate the DE1 displays

The “sweep” component, called by DE1\_disp, is just re-casting of the “clkdiv” component from Part V (Figure 62).

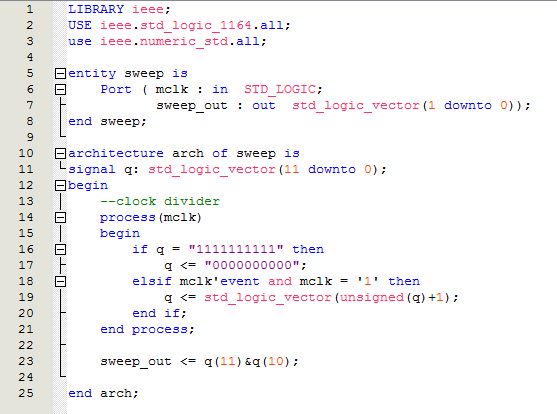


Figure : Reused and Renamed

Pulling it all together with the schematic editor we get the following (Figure 63).

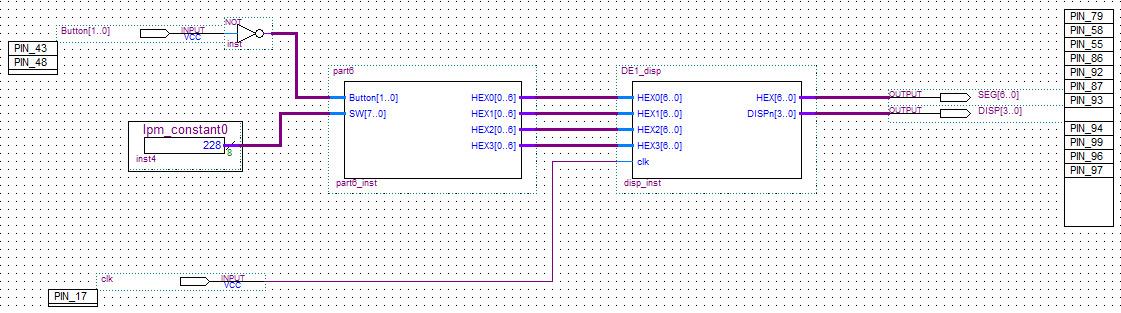


Figure : The final assembly

Don’t forget things like making the schematic the top of the project, and add the inverter to invert the button logic and then voila! (Figure 64).



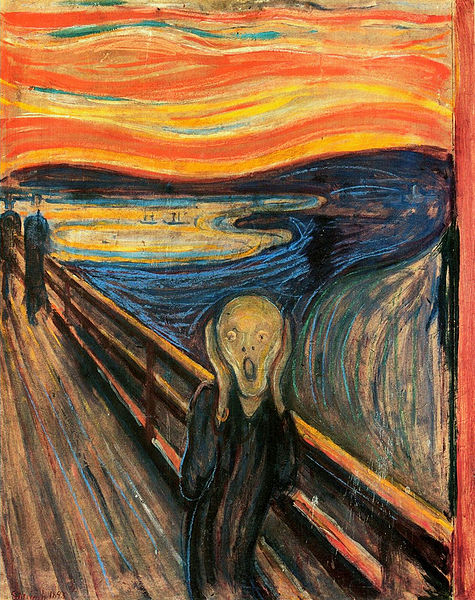
Figure : Final result (well almost)

Almost? If you look closely there is a bug in the DE1\_disp code. Notice the “ghosting”! The “A” is set in the third position but appears dimly in the fourth. The “A” and “F” segments of the second display are ghosted – likely remnants of the “E” in the first position.

What is likely happening is that the sweeping strobe for the display enable, and hence the outputting of the next character, is happening slightly out of whack. Most likely, the display enable is being changed ahead of the character change. This makes sense, given we are coding in hardware, and the so-called concurrency actually means we might see one circuit run ahead of the other.

We will look a fixing the ghosting and then packaging up the DE1\_disp into a re-useable library element.

# You may now SCREAM!!



1. We could actually change the displayed chars from “dE1” to “EdA” for our board, do so if you like, just change the Table 2 entries. [↑](#footnote-ref-1)