# References on Web Jump out to the interwebby thing

PDF Resources: Designing the Hardware

[1] [Laboratory Exercise 2 Numbers and Displays](ftp://ftp.altera.com/up/pub/Altera_Material/11.1/Laboratory_Exercises/Digital_Logic/DE1/vhdl/lab2_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)

[4] [VHDL Tutorial](http://ati.ttu.ee/~alsu/VHDL_TUTORIAL_Ashenden.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](#_Toc428896644)

[Introduction 2](#_Toc428896645)

[Part I 4](#_Toc428896646)

[Part II 10](#_Toc428896647)

[Part III 18](#_Toc428896648)

[Part IV 24](#_Toc428896649)

[Part V 32](#_Toc428896650)

[Part VI 33](#_Toc428896651)

[You may now SCREAM!! 33](#_Toc428896652)

# 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).

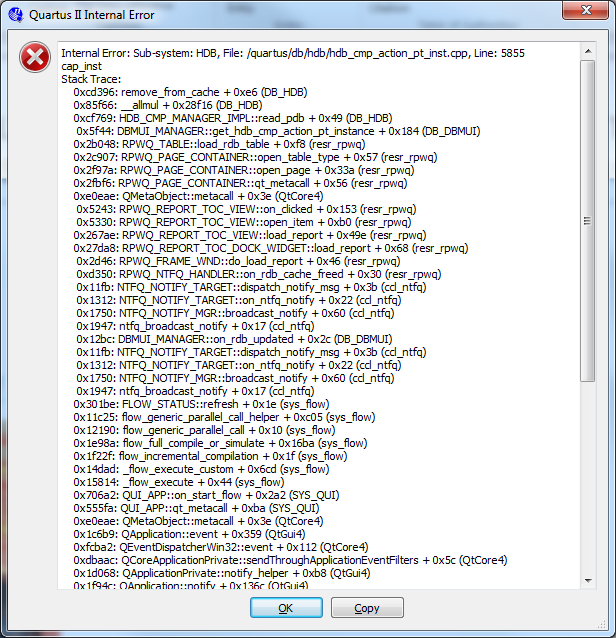


Figure 1: Same old problem

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

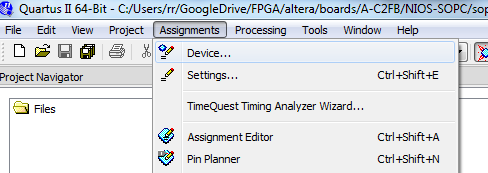


Figure 2. An important missing step.

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.

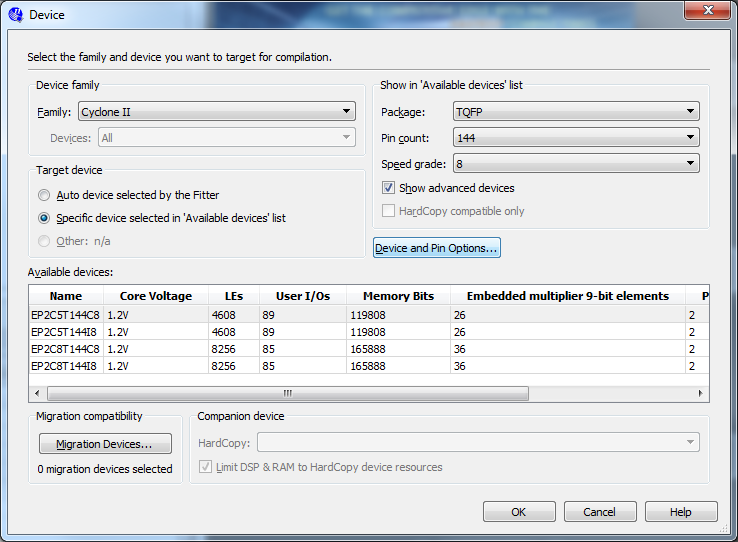


Figure 3. We need to do something with our unused pins!

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

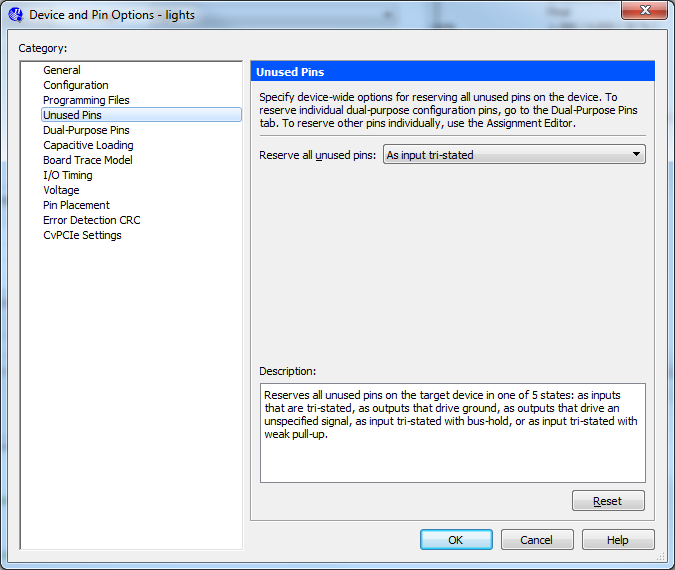


Figure 4. Tell the pins to be “quiet”

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

We run straight into a problem with this part of Laboratory 2 as it requires 10 switches so we know we need another way of doing this. As we did in Part V of Digital Labs Part 1, we can use constants to get around the absence of switches. Or, we can opt to drop one of the LED circuits – we have four switches yes and the problem uses four switches per segment. Yes, we’ll do that.

Other than that, we just sketch out our LED segment lighting and then use LogicFriday to build a truth table. **WARNING:** Notice the inputs in Table 1 are **inverted** to take into account that the switches have pull-up resistors tying them to HIGH.

Table 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | LED\_A | 0 | PIN\_93 | 0 | | 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 | 0 | | LED\_G | 6 | PIN\_79 | 1 |   “1000000” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png**1111**=>0 |
| |  |  |  |  | | --- | --- | --- | --- | | 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 |   “1111001” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png1110=>1 |
| |  |  |  |  | | --- | --- | --- | --- | | LED\_A | 0 | PIN\_93 | 0 | | LED\_B | 1 | PIN\_92 | 0 | | LED\_C | 2 | PIN\_87 | 1 | | LED\_D | 3 | PIN\_86 | 0 | | LED\_E | 4 | PIN\_55 | 0 | | LED\_F | 5 | PIN\_58 | 1 | | LED\_G | 6 | PIN\_79 | 0 |   “0100100” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png1101=>2 |
| |  |  |  |  | | --- | --- | --- | --- | | LED\_A | 0 | PIN\_93 | 0 | | LED\_B | 1 | PIN\_92 | 0 | | LED\_C | 2 | PIN\_87 | 0 | | LED\_D | 3 | PIN\_86 | 0 | | LED\_E | 4 | PIN\_55 | 1 | | LED\_F | 5 | PIN\_58 | 1 | | LED\_G | 6 | PIN\_79 | 0 |   “0110000” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png1100=>3 |
| |  |  |  |  | | --- | --- | --- | --- | | 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 | 0 | | LED\_G | 6 | PIN\_79 | 0 |   “0011001” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png1011=>4 |
| |  |  |  |  | | --- | --- | --- | --- | | LED\_A | 0 | PIN\_93 | 0 | | LED\_B | 1 | PIN\_92 | 1 | | LED\_C | 2 | PIN\_87 | 0 | | LED\_D | 3 | PIN\_86 | 0 | | LED\_E | 4 | PIN\_55 | 1 | | LED\_F | 5 | PIN\_58 | 0 | | LED\_G | 6 | PIN\_79 | 0 |   “0010010” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png1010=>5 |
| |  |  |  |  | | --- | --- | --- | --- | | LED\_A | 0 | PIN\_93 | 0 | | LED\_B | 1 | PIN\_92 | 1 | | LED\_C | 2 | PIN\_87 | 0 | | LED\_D | 3 | PIN\_86 | 0 | | LED\_E | 4 | PIN\_55 | 0 | | LED\_F | 5 | PIN\_58 | 0 | | LED\_G | 6 | PIN\_79 | 0 |   “0000010” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png1001=>6 |
| |  |  |  |  | | --- | --- | --- | --- | | LED\_A | 0 | PIN\_93 | 0 | | 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 |   “1111000” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png1000=>7 |
| |  |  |  |  | | --- | --- | --- | --- | | LED\_A | 0 | PIN\_93 | 0 | | 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 | 0 | | LED\_G | 6 | PIN\_79 | 0 |   “0000000” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png0111=>8 |
| |  |  |  |  | | --- | --- | --- | --- | | LED\_A | 0 | PIN\_93 | 0 | | 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 | 0 | | LED\_G | 6 | PIN\_79 | 0 |   “0011000” | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png0110=9 |
| |  |  |  |  | | --- | --- | --- | --- | | LED\_A | 0 | PIN\_93 | 1 | | LED\_B | 1 | PIN\_92 | 1 | | LED\_C | 2 | PIN\_87 | 1 | | LED\_D | 3 | PIN\_86 | 1 | | LED\_E | 4 | PIN\_55 | 1 | | LED\_F | 5 | PIN\_58 | 1 | | LED\_G | 6 | PIN\_79 | 1 |   otherwise | http://organicmonkeymotion.files.wordpress.com/2014/04/exp72.png |

The truth table becomes:

Minimized:

F0 = A B C D' + B' C D + A' C' + A' B';

F1 = B' C D' + A' C' + B' C' D + A' B';

F2 = B C' D + A' B' + A' C';

F3 = A' D' + B C D' + B' C D + B' C' D' + A' C';

F4 = A' C' + B' C + D';

F5 = A B D' + A' B' + B C' + C' D';

F6 = A B C + B' C' D' + A' C' + A' B';

So coding that up we get Figure 5.

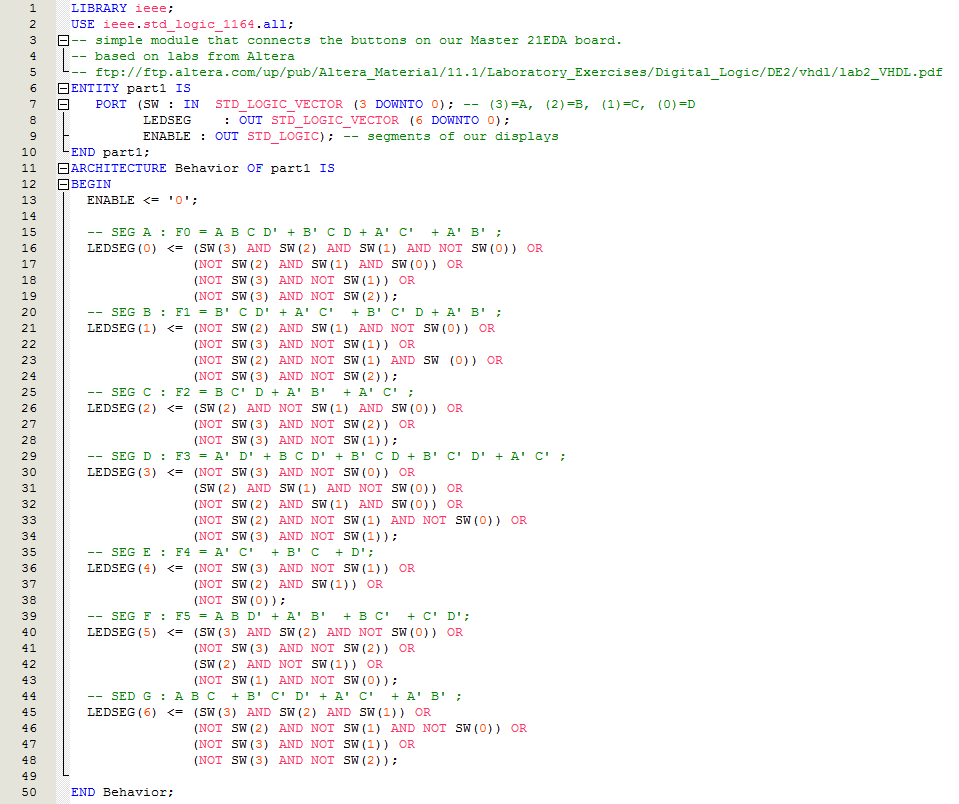


Figure 5: Voila!

Don’t forget the pin assignments at Figure 6.

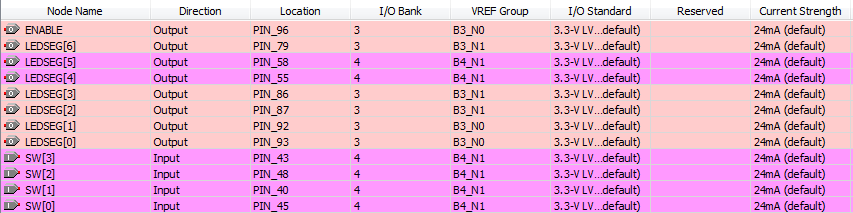


Figure 6

You should get a RTL somewhat like that at Figure 7.

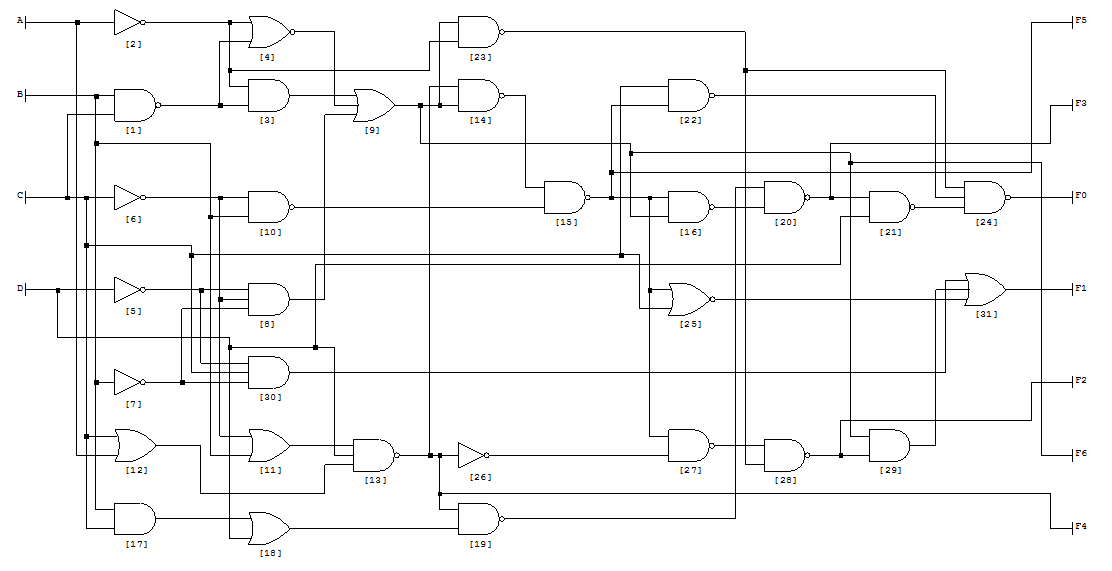


Figure 7

If you bother to code it up in LogicFriday I just used the 8 LED output widget as in Figure 8. If your wire up F0..F6 from Figure 7 then the LED match up with the “0011000” etc. segment enable maps of the logic table so you can interpret them.

So, with everything working you should have the 7-segment display that is enabled off PIN\_96 light up as per design.

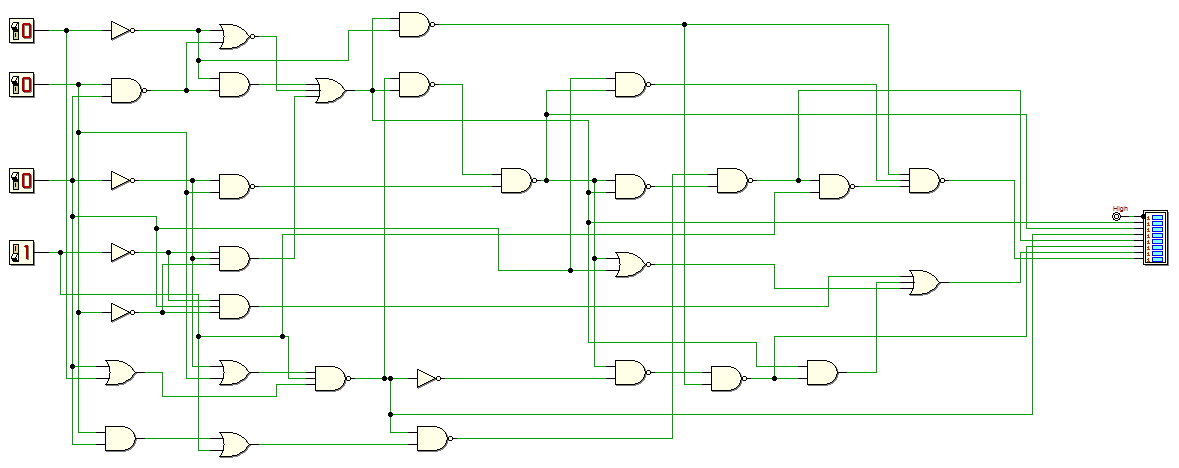


Figure 8

# Part II

We really simply need to note something for “Circuit A” from **Figure 1**.

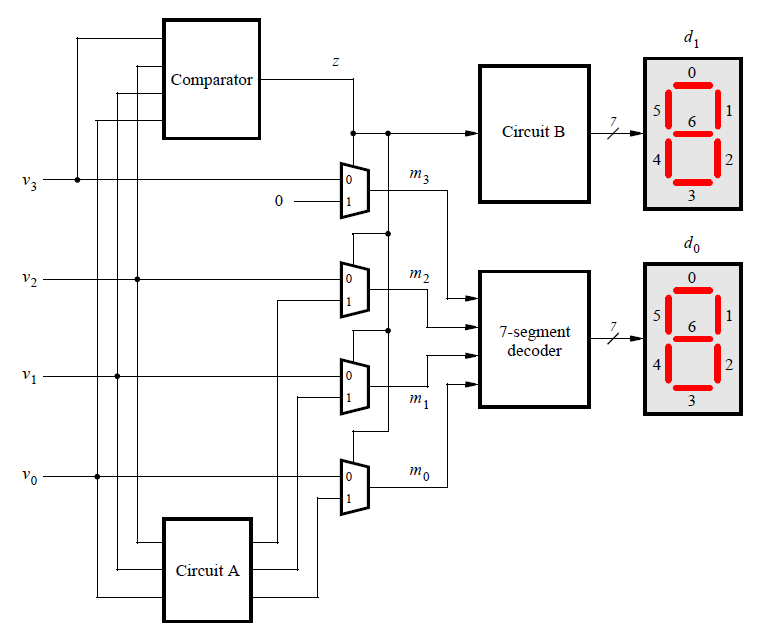


Figure 9: Figure 1

“Circuit A” looks like it needs to morph v2..v0 somehow when z is high. z being high when z>9.

Let’s paint this out.

We’ve four KEYS to give us 00 through 15 on the two displays.

**WARNING:** The KEYS’ column is to take into account that the keys (or buttons) on our board are pulled HIGH (so we want the table **inverted’** to drive our design). Looking at Table 2 below we see that for “Circuit A”, once past VAL “09”, we want to map I[210] (101 down to 000) to the values under F[210] (111 down to 010).

Table 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| KEYS | KEYS’ | VAL | COMP | This is the actual values we want at the multiplexor output >9 | | |
| 0000 | **1111** | 00 | 0 |
| 0001 | 1110 | 01 | 0 |
| 0010 | 1101 | 02 | 0 |
| 0011 | 1100 | 03 | 0 |
| 0100 | 1011 | 04 | 0 |
| 0101 | 1010 | 05 | 0 |
| 0110 | 1001 | 06 | 0 |
| 0111 | 1000 | 07 | 0 |
| 1000 | 0111 | 08 | 0 |
| 1001 | 0110 | 09 | 0 | I[210] F[210] | | |
| 1010 | 0101 | 10 | 1 | 101 | 0 | 111 |
| 1011 | 0100 | 11 | 1 | 100 | 1 | 110 |
| 1100 | 0011 | 12 | 1 | 011 | 2 | 101 |
| 1101 | 0010 | 13 | 1 | 010 | 3 | 100 |
| 1110 | 0001 | 14 | 1 | 001 | 4 | 011 |
| 1111 | 0000 | 15 | 1 | 000 | 5 | 010 |

To wit, we need to power up Logic Friday (note the instruction in the Altera LAB is to avoid using IF/THEN and CASE etc.).

Thus Figure 10:

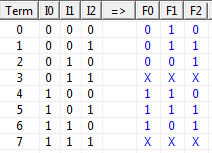


Figure 10

Fun bit is that it makes more sense to have reverse the column labels for I0..I2 because I2 is actually the 20 column and I0 is the 23 column. No matter just reverse things in your head (of course there is no stopping one from setting the columns in LogicFriday). Note we are not using all the values so we can set ‘X’ or “Don’t Care”.

Entered by truth table:

F0 = I0 I1' I2' + I0 I1' I2 + I0 I1 I2';

F1 = I0' I1' I2' + I0' I1' I2 + I0 I1' I2' + I0 I1' I2;

F2 = I0' I1' I2 + I0' I1 I2' + I0 I1' I2 + I0 I1 I2';

Minimized:

F0 = I0;

F1 = I1';

F2 = I2 + I1;

Coded up as Figure 11 below.

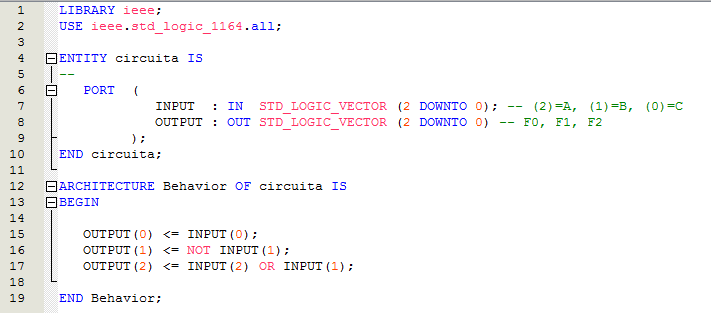


Figure 11

The “Comparator” is straight forward (again taking into account the input buttons being pulled HIGH) and thus Figure 12:

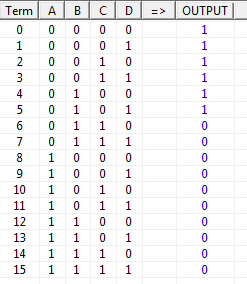


Figure 12

Entered by truthtable:

OUTPUT = A' B' C' D' + A' B' C' D + A' B' C D' + A' B' C D + A' B C' D' + A' B C' D;

Minimized:

OUTPUT = A' C' + A' B';

All that and all it becomes is Figure 13 below.

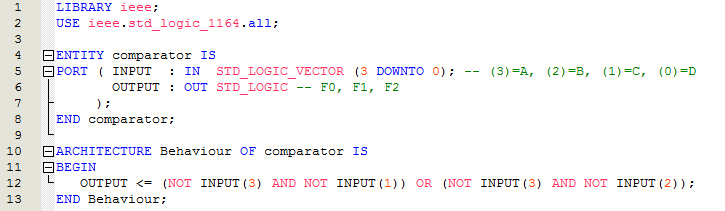


Figure 13

The 7 segment code is from the previous Part I however, we drop the inbuilt enable as we are using the DE1\_disp module from LAB 1 Part VI solution. We have fixed the “ghost” character we incurred with the original code by using the following code over the page. We’ll assign d0 and d1 (from Figure 9: Figure 1) to HEX0 and HEX1 inputs respectively (Figure 14). We’ll drive the other two inputs to blank out the 3rd and 4th displays.

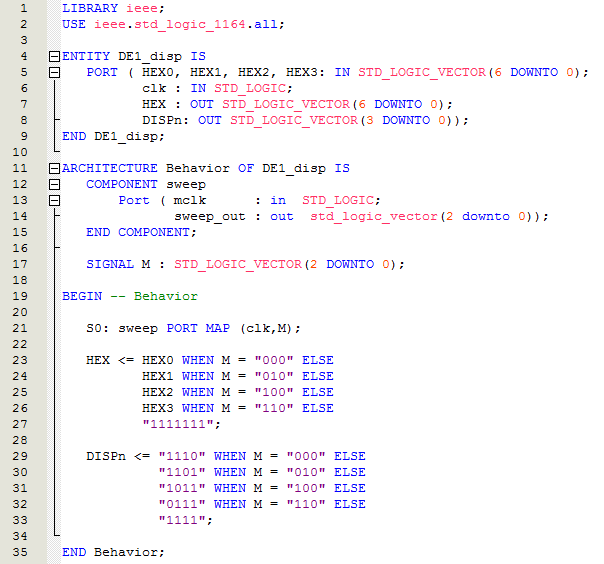


Figure 14

Circuit B is simply either 7-segment ‘0’ WHEN z=’0’ ELSE 7-segment ‘1’ WHEN z=’1’ (using VHDL lingo). The upshot is that Brian (Brian Drummond on Stack Overflow) pointed out the actual problem was the saturation of the bipolar transistor on the board – meaning once it was charged up (enabled) it then took time to drain and the “ghost” was the transistor still driven open (driving circuit to ground) into the next digit time slot.

Without ceremony, “Circuit B” at Figure 15 over the page.

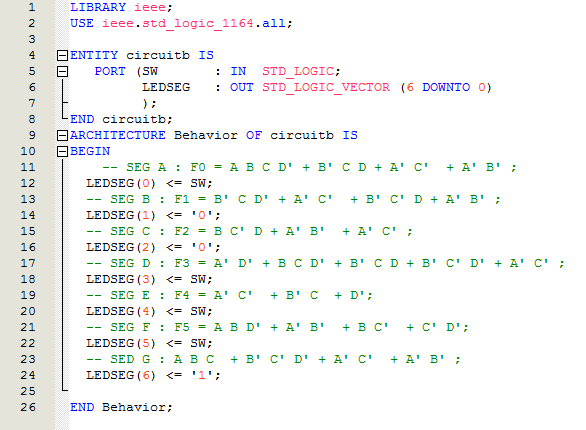


Figure 15

To sum up then, we need to assemble a number of components. The definitions being below at Figure 16.

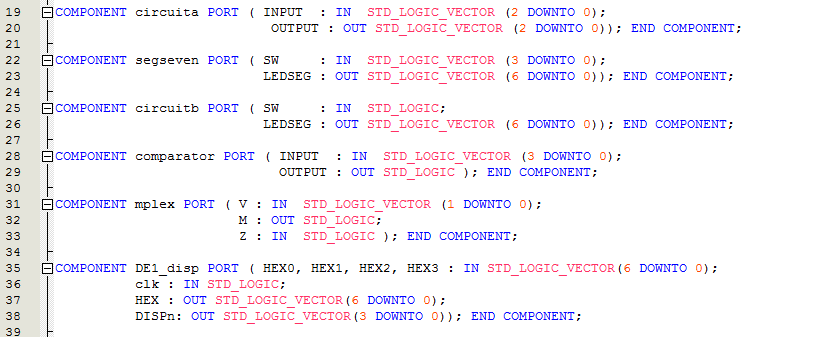


Figure 16

**NOTE:** the code for “segseven” is actually the code from Figure 5 of Part I. Just rename it from “part1” to “segseven” after moving the file to your Part II project etc.

The first 5 relating to the design in Figure 9: Figure 1 and the sixth being our tailoring of the Master 21EDA so that the 7-segment LED displays act somewhat like the ones on the DE1.

Wired up (in code) the result looks fine. More or less mimicking the design at Figure 9: Figure 1 as one would hope.

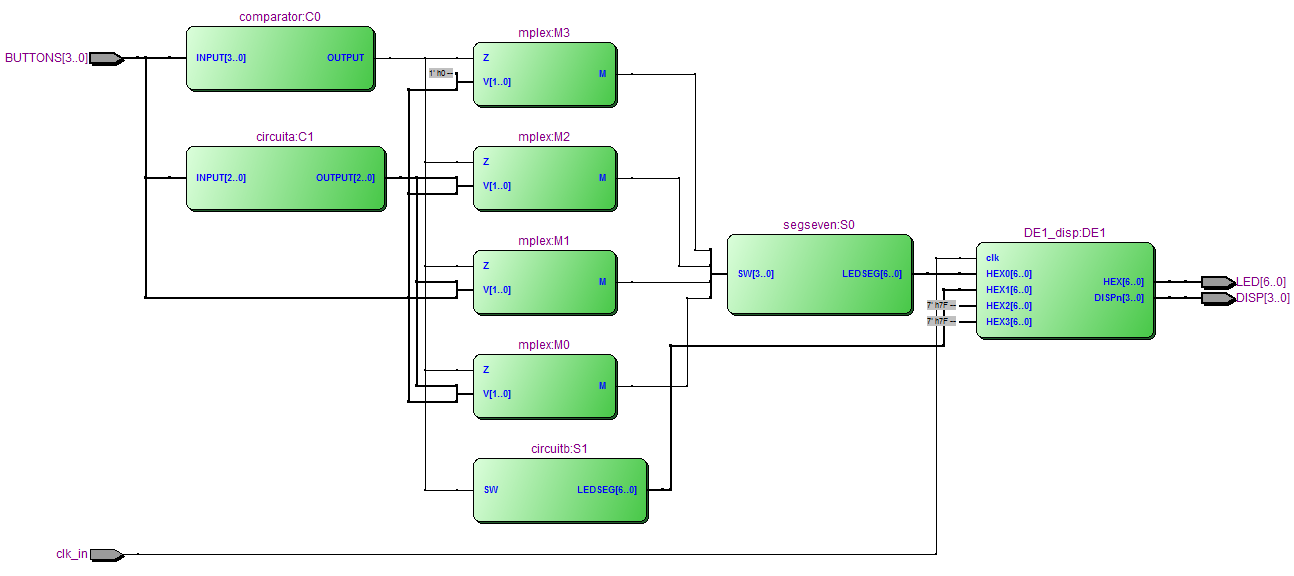


Figure 17

To wire it up we literally need some (signal) wires and so: we define:

s\_m(3..0), HOLD\_LOW, s\_z, s\_ao(2..0), s\_ai(2..0), HEX\_0(6..0), HEX1(6..0), BLANK(6..0)

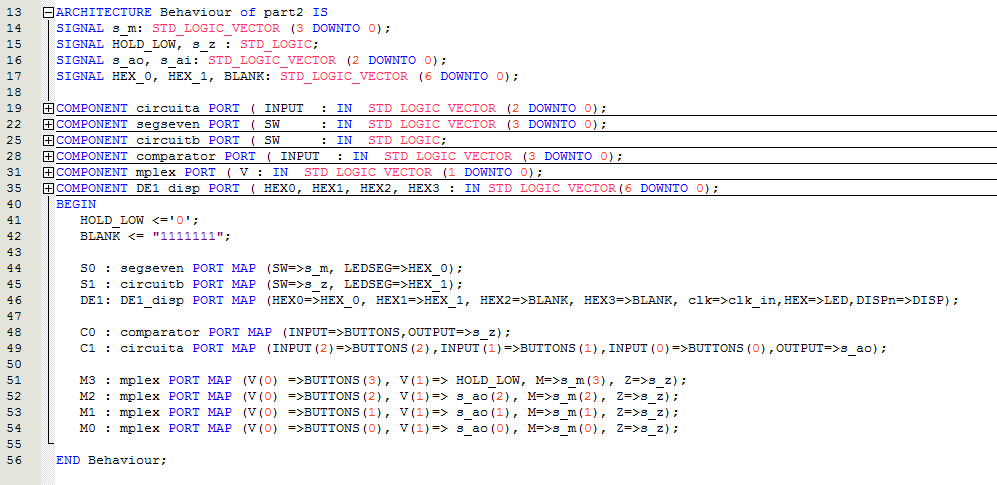
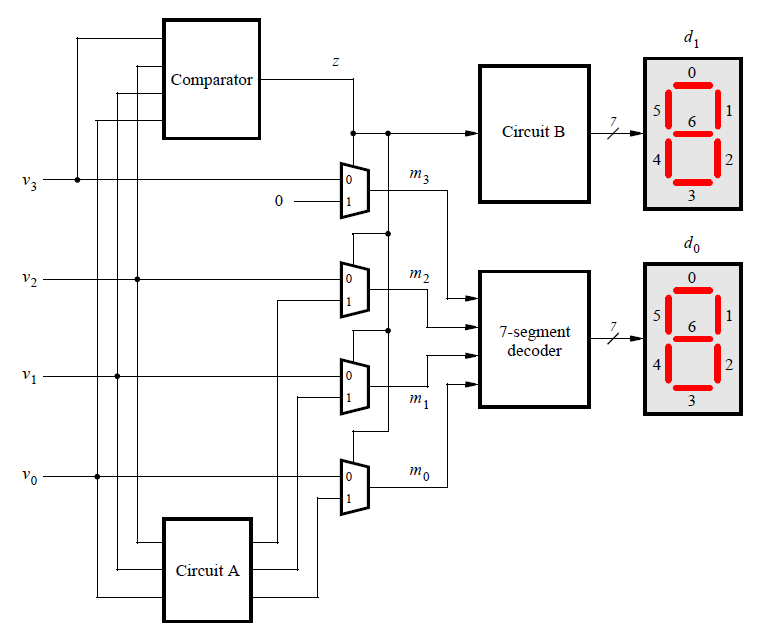


Figure 18

To help decipher the wiring have a look at Figure 19 over the page.



s\_ai(2..0)

HOLD\_LOW

s\_m(3..0)

DE1\_disp

HEX0

HEX1

s\_ao(2..0)

s\_z

BUTTONS(3..0)

Figure 19

The final step, once compiled, is to wire up the design to pins as in Figure 20 below.

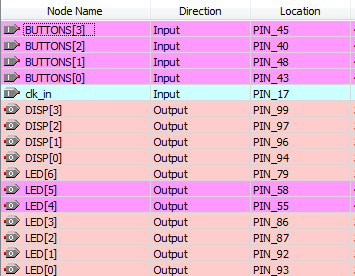


Figure 20

Voila!

No ghost!

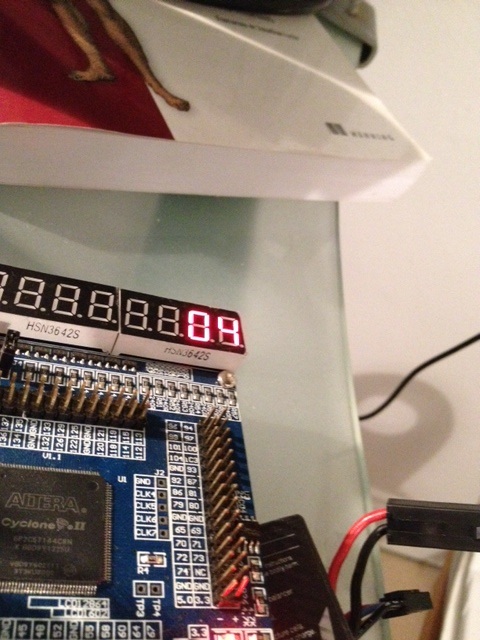


Figure 21

# Part III

A ripple carry OR full adder in Figure 22:

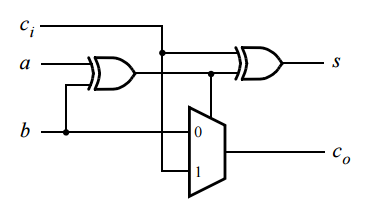


Figure 22

Begets Figure 23 in Deeds’ Digital Circuit Simulator:

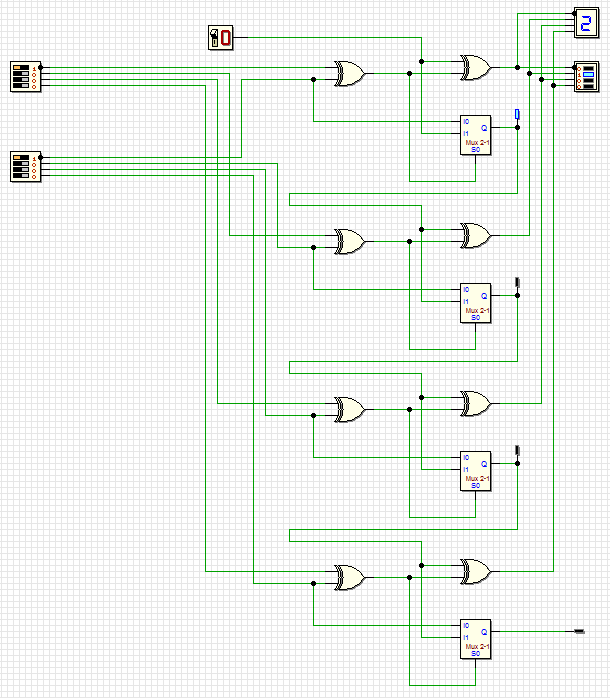


Figure 23

You can see above where there are test points on the carry flags and 1+1=2 with a carry flagged as one would expect. Now have a play. Is there a carry on 1+2? On 2+2? On 1+3? On 3+1?

This will be our test generator to check our LED outputs against inputs.

Now of course this is cheating! But LogicFriday allows you to draw your circuit as in Figure 24. When you submit you get your equations – minimized of course.

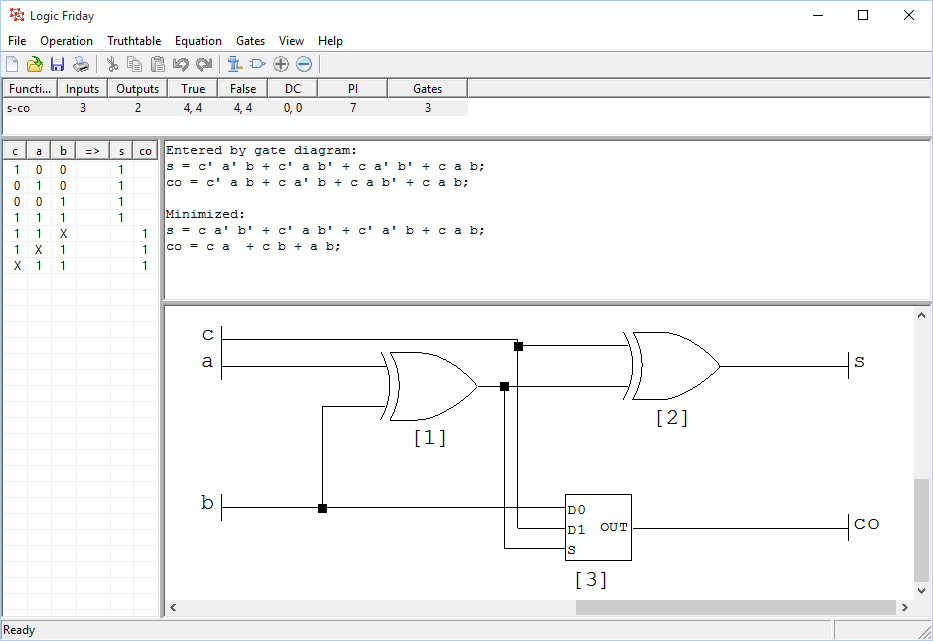


Figure 24

It is then straight forward to code up as in Figure 25.

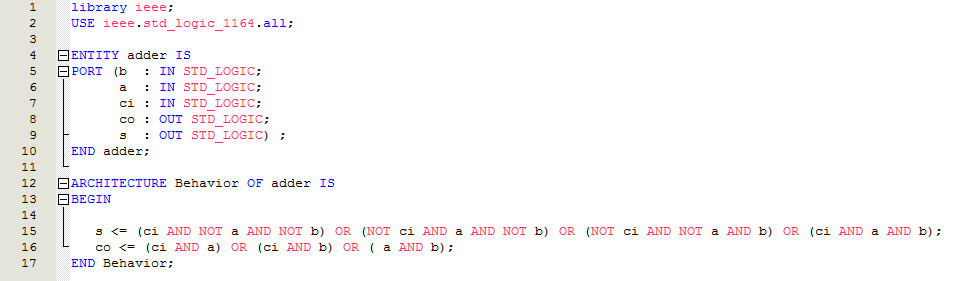


Figure 25

Now we weave Figure 23 into code at Figure 26.

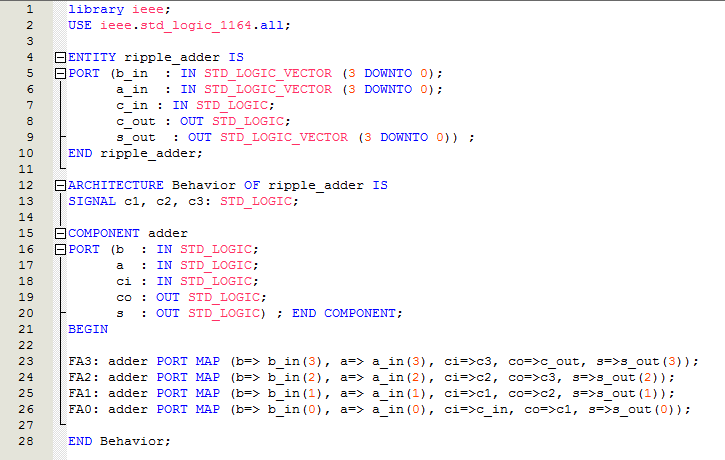


Figure 26

Now, remember we are short switches so we need to cheat and use the schematic capture. We need 4 times 1 bit lpm\_constant (Figure 27) for our ‘b\_in[3..0]’ input. Well feed input ‘a\_in[3..0]’ from the switches. May as well have an adjustable carry input as well. The schematic looks like Figure 28 over the page.

 LPM\_Constants explanation is at: <http://quartushelp.altera.com/current/master.htm>

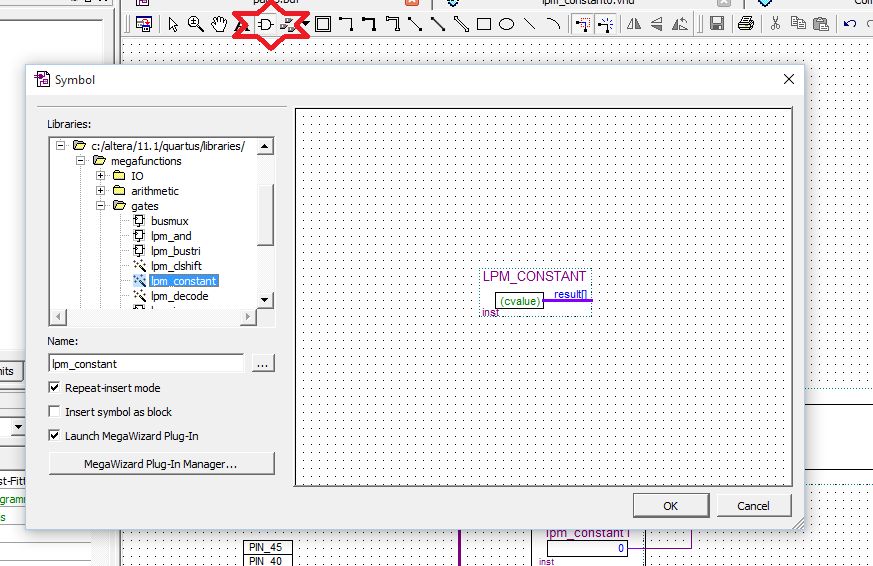


Figure 27

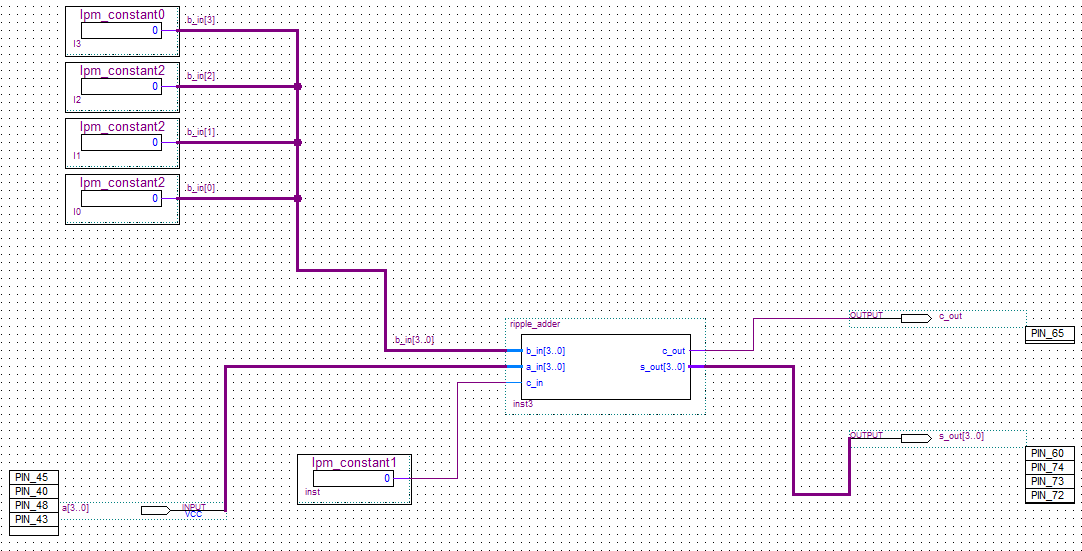


Figure 28

Note the naming on the “b\_in[3..0]” buss in Figure 28. At the buss end you still need to name the buss as it didn’t do what I thought might be the intuitive thing to do, which is to pick up the name of the inputs. Likely makes sense though since, being a buss, it might have a raft of input/output points and thus likely its own name.

The trick is at the other end, on the 1-bit constants is using the ‘wire’ within the buss we are interested in at each constant – which represent I3=8, I2=4, I1=2 and I0=1.

To make this clearer read Table 3 below.

Table 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Binary Value | Pin | Ripple a\_in | Constant ID | Ripple b\_in | Ripple Out |
| 8 | PIN\_45 | a\_in[3] | I3 | b\_in[3] | s\_out[3] |
| 4 | PIN\_40 | a\_in[2] | I2 | b\_in[2] | s\_out[2] |
| 2 | PIN\_48 | a\_in[1] | I1 | b\_in[1] | s\_out[1] |
| 1 | PIN\_43 | a\_in[0] | I0 | b\_in[0] | s\_out[0] |

In any event now we are cooking. Open the “In-System Memory Content Editor” shown at Figure 29.

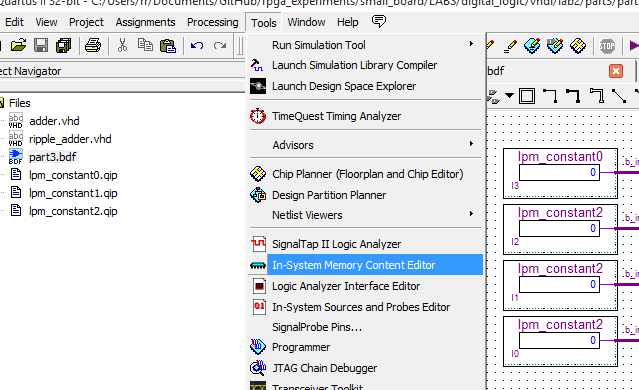


Figure 29

Refer to Quartus II Handbook Version 11.1 Volume 3: Verification Section IV. System Debugging Tools for information on how to use the “In-System Memory Content Editor”. Especially “15. In-System Modification of Memory and Constants”.

Once you are familiar with the tool, or at least while you have the manual open, you can modify the b\_in[] values as displayed in Figure 30.

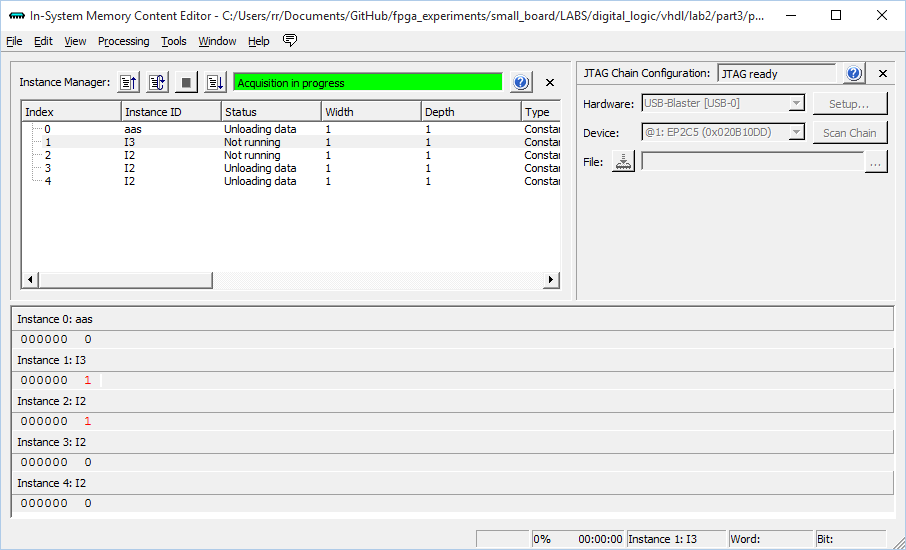


Figure 30

Now if your kindergarten math is up to scratch you can now play with adding binary. Again, you can cross check with simulation of the circuit in Deeds at Figure 23.

# Part IV

|  |
| --- |
| You are to design a circuit that adds two BCD digits. The inputs to the circuit are BCD numbers *A* and *B*, plus a carry-in, cin. The output should be a two-digit BCD sum S1S0. Note that the largest sum that needs to be handled by this circuit is S1S0 = 9 + 9 + 1 = 19. Perform the steps given below. |

Wow! Design a means to add two Binary Coded Decimal (BCD) numbers!

 <https://en.wikipedia.org/wiki/Binary-coded_decimal>

So if your digital math is not up to speed, at little help is at hand.

 <http://www2.elo.utfsm.cl/~lsb/elo211/aplicaciones/katz/chapter5/chapter05.doc4.html>

Short answer is, apparently, one will add 6 if previous sum > 9. When you read the explanation from the link above the table below might help. Remember ‘X’ means “don’t care” (see also Figure 31 ).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| # | B3 | B2 | B1 | B0 | Combinations | Representation | GATE |
| 9 | 1 | 0 | 0 | 1 |  |  |  |
| 10 | 1 | 0 | 1 | 0 | B3& B1 | 1X1X | A2 |
| 11 | 1 | 0 | 1 | 1 | B3& B1 | 1X1X | A2 |
| 12 | 1 | 1 | 0 | 0 | B3& B2 | 11XX | A1 |
| 13 | 1 | 1 | 0 | 1 | B3& B2 | 11XX | A1 |
| 14 | 1 | 1 | 1 | 0 | (B3& B2)|( B3& B1) | 11XX|1X1X | A1|A2 |
| 15 | 1 | 1 | 1 | 1 | (B3& B2)|( B3& B1) | 11XX|1X1X | A1|A2 |

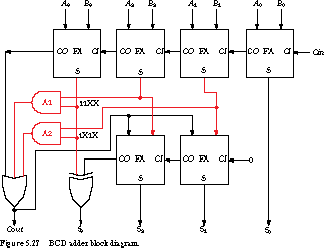


Figure 31

Note also the constraint that maximum number to output need only be 19. This helps as it simply means S0 is 4 bit and Carry Out of the circuit can along with a display driver we have used previously, paint a “1” or a “0” in the 7-segment display.

Now all we have to do is simply rewire the ‘adder’ from Part III with wires and ‘glue’.

WHAT! There is an XOR gate in there! Well, turns out, while we have been relying on binary AND and OR operators previously, there is also a raft of other binary operators in VHDL, namely: AND, OR, NAND, NOR, XOR, XNOR.

For a brush up read:

<http://whatis.techtarget.com/definition/logic-gate-AND-OR-XOR-NOT-NAND-NOR-and-XNOR>

Any old how, sketching out the wiring on the diagram we get Figure 32.

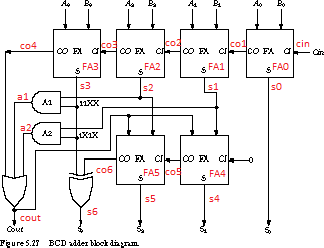


Figure 32

Wired up in code this looks somewhat like Figure 33.

Now all we have to really do is wire things up to give us firstly the bcd\_adder. We will have six adders and some “glue”. The glue is around a1, a2 cout and s6 of Figure 32.

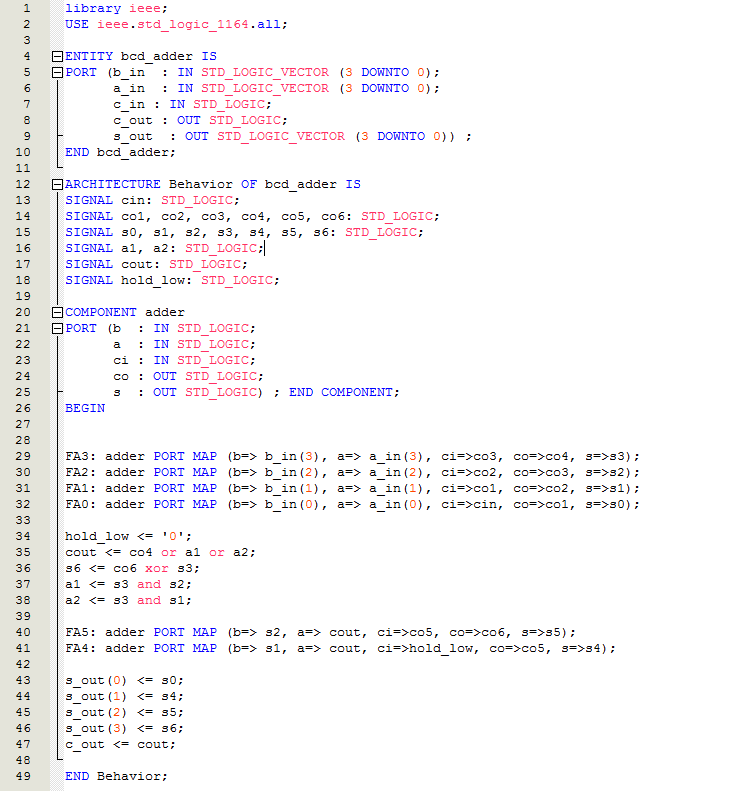


Figure 33

Other code we will need we can steal from Part II and use “circuitb” and “DE1\_disp” (including “sweep”). However, we cannot use the segseven code as is as it assumed inputting from the switches which are pulled high by a resistor. We need to re-write that code so that the inputs range from the un-inverted 0000..1001 and not the **inverted** 1111..0110 of Table 1. Suffice to say minimised we get:

Minimized:

A = In3' In2' In1' In0 + In2 In1' In0' + In3 In1 + In3 In2;

B = In2 In1' In0 + In3 In1 + In2 In1 In0' + In3 In2;

C = In2' In1 In0' + In3 In2 + In3 In1;

D = In3 In0 + In2 In1' In0' + In2' In1' In0 + In3 In1 + In2 In1 In0;

E = In2 In1' + In3 In1 + In0;

F = In3 In2 + In3' In2' In0 + In2' In1 + In1 In0;

G = In3' In2' In1' + In2 In1 In0 + In3 In2 + In3 In1;

New code for segseven is at Figure 34 (using un-inverted inputs).

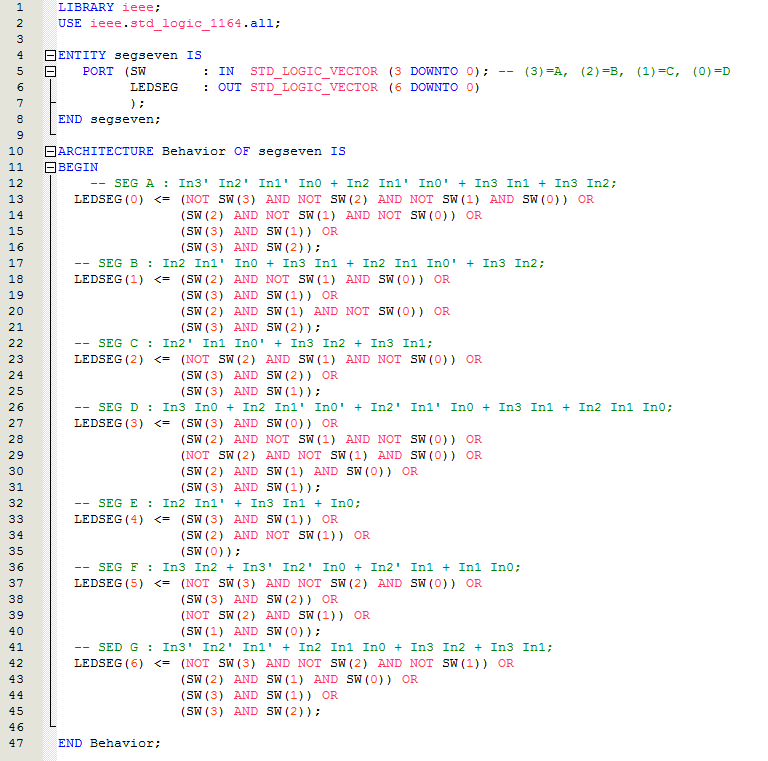


Figure 34

Now, remember our board does not have enough switches to provide all inputs so we’ll need a schematic as our design top element. Before that though we will build most of the circuit up in VHDL. This looks rather like the code for “part4\_code” in Figure 35.

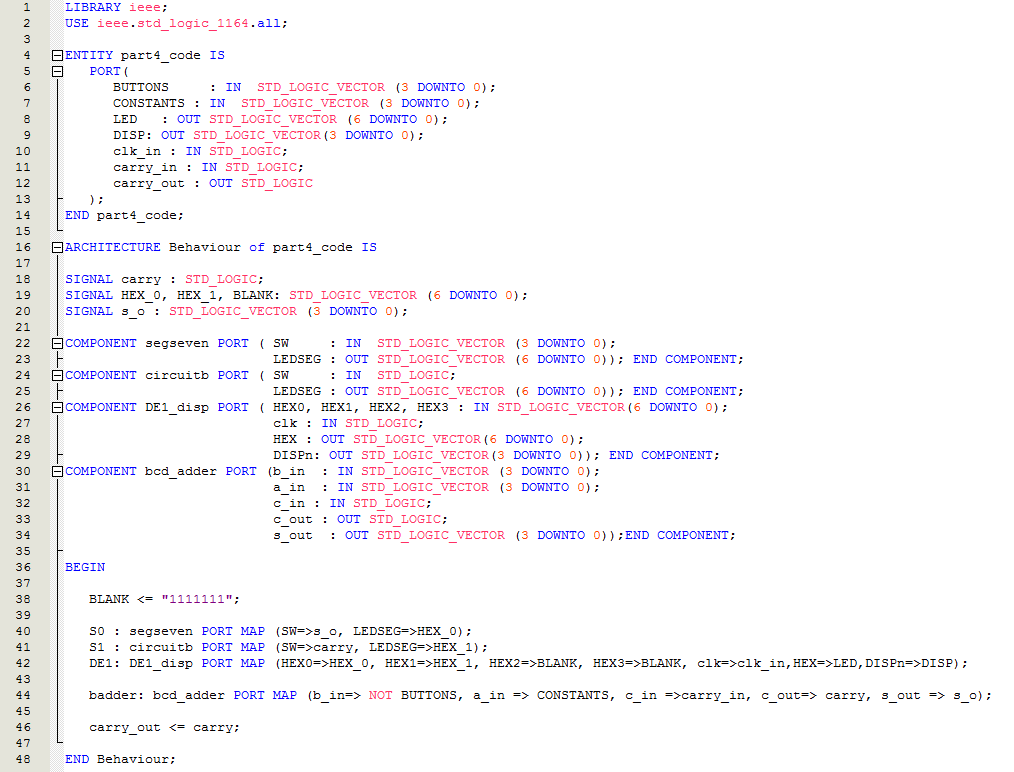


Figure 35

We then create a symbol for “part4\_code” as per Figure 36

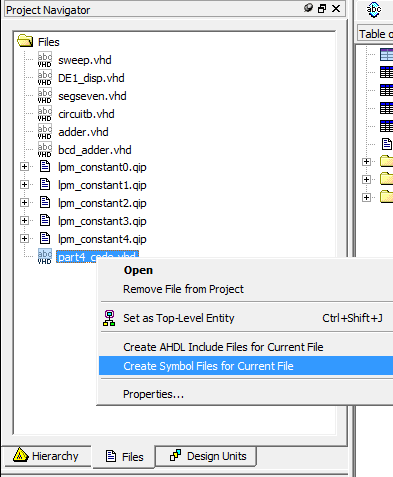


Figure 36

Now we can create the top element “part4” (Figure 37) and to be sure make it the top element (Figure 38).

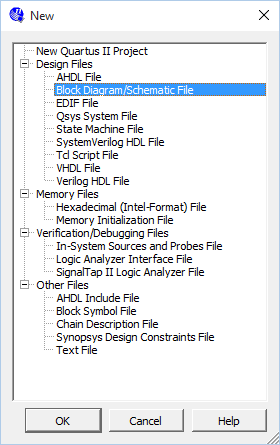


Figure 37

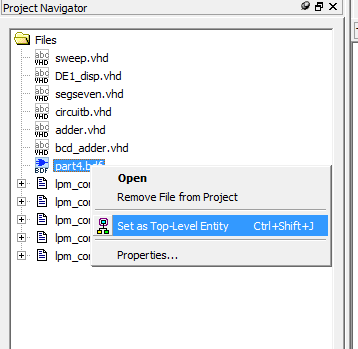


Figure 38

Now, we take our symbol for part4\_code (Figure 39) and build a circuit around it (Figure 40).

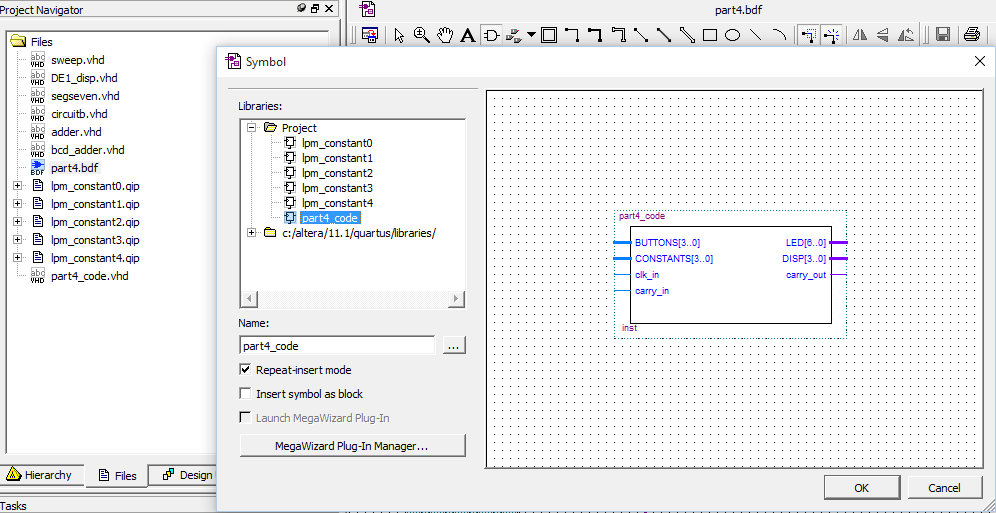


Figure 39

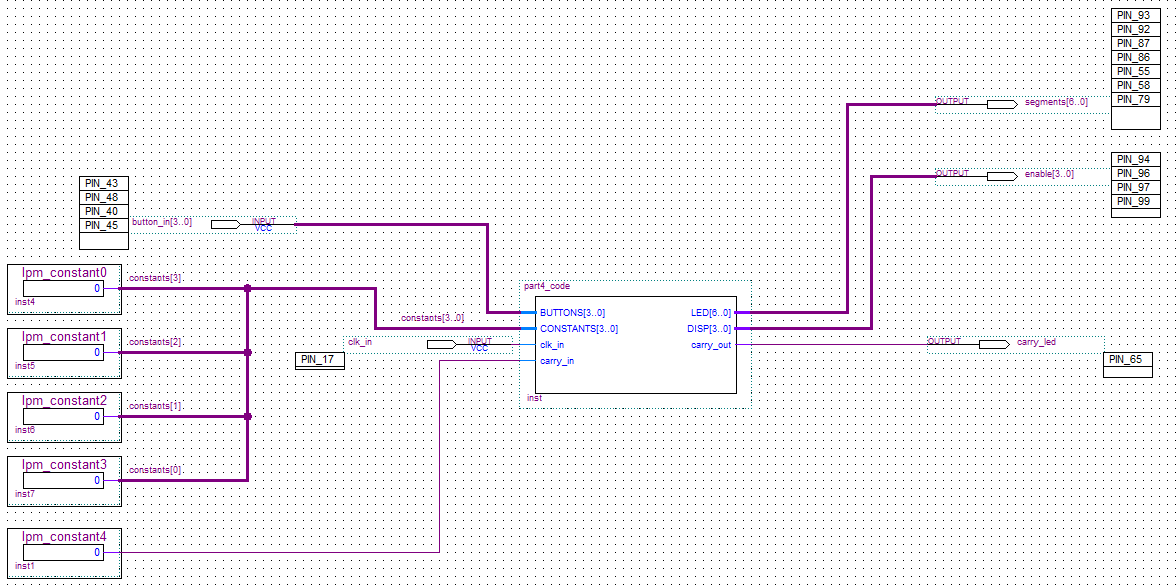


Figure 40

Now after compiling the suggested pins to use are in Figure 41.

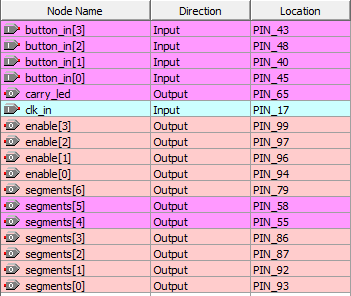


Figure 41

WOW! Look at the RTL for the bcd\_adder at Figure 42! One-4-one of the sketching at Figure 32!

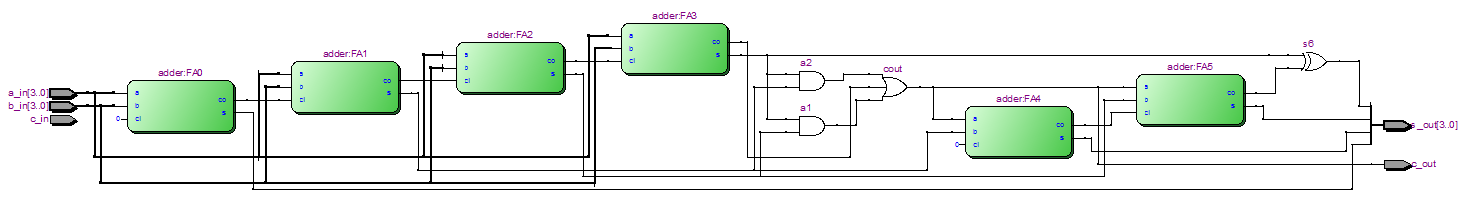


Figure 42

Phew!

# Part V

Aim this lab is to use an algorithm for the BCD adder at Figure 43.

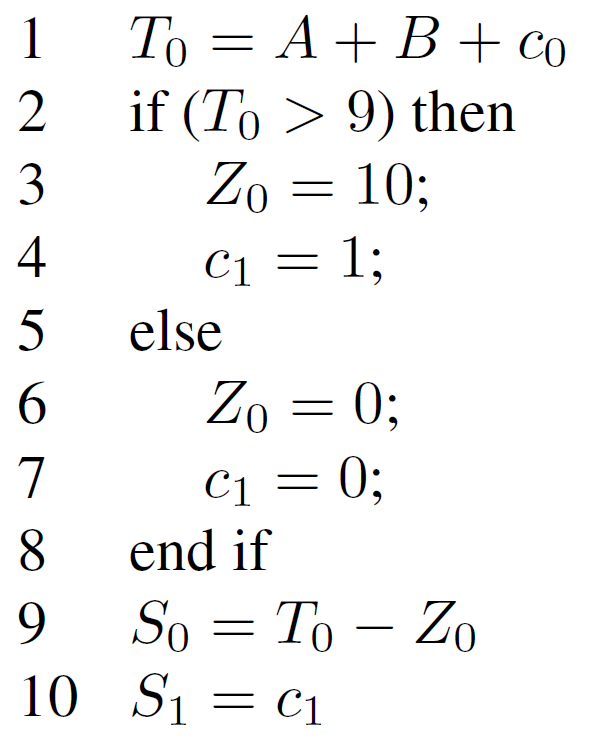


Figure 43

So, FPGA and VHDL math! DOH!

Have a read of [bitweenie](http://www.bitweenie.com/listings/vhdl-type-conversion/) and the discussion on VHDL type conversion. We will likely do something, well, not illegal but clumsy or not preferred. Mostly the argument is we are wanting to fit the BCD adder into a current design.

Otherwise we need convert to “unsigned” and then cast again back to STD\_LOGIC\_VECTOR:

t <= STD\_LOGIC\_VECTOR(unsigned(a\_in) + unsigned(b\_in));

Although there is a “trick” when adding single wire STD\_LOGIC signals:

t <= STD\_LOGIC\_VECTOR(unsigned(a\_in) + unsigned(b\_in) + (c\_in & ""));

The gem, of course, was the ‘(c\_in & “”)’. The ampersand or & is a concatenation operator in VHDL. So, you are building a vector of one or, say {c\_in} as opposed to c\_in. That is the “” is an empty vector so an empty vector plus an entry is a non-empty vector of one entry. Think of vector then as array.

A good explanation, with examples, is on page 41 of a tutorial on VHDL by Peter Ashenden [4].

So, the long and short of it was the code in Figure 44 over the page.

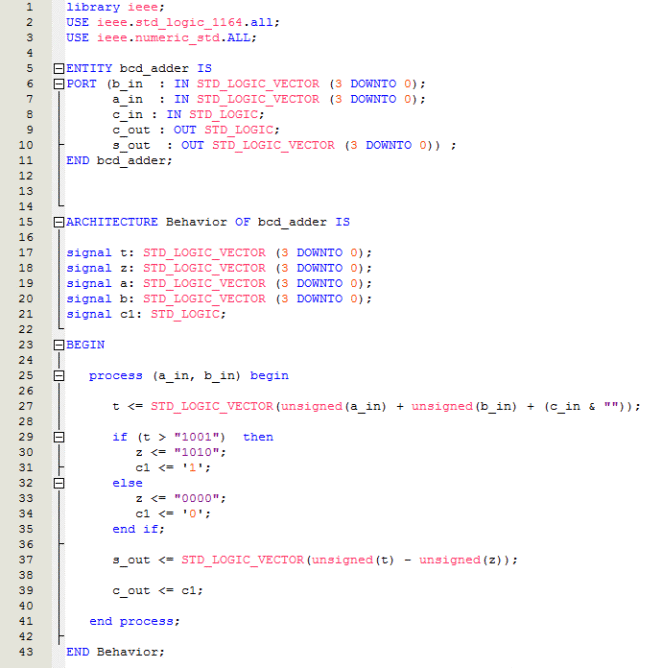


Figure 44

So, starting with the project for Part IV we simply replace the code for the “bcd\_adder” with that at Figure 44 above. You can drop the code for the “adder” as all the functionality is in “bcd\_adder” without having to use the “adder” component used in Part IV.

# Part VI

# You may now SCREAM!!

