References on Web

PDF Resources: Designing the Hardware

[1] Laboratory Exercise 1 Switches, Lights, and Multiplexers

[2] Translated manual for Master 21EDA board

[3] Altera DE1 Board

Tools

[4] Digital Electronics Education Design Suite (DEEDS)

[5] LogicFriday

Contents

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:

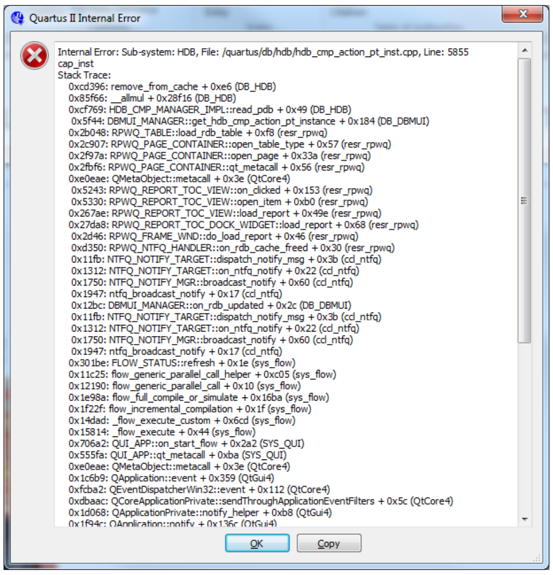
If I have been stumped by something I will use the image to the left to let you know a little investigation was in order.

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).



```
Quartus II Internal Error

Internal Error: Sub-system: HDB, File: /quartus/db/hdb/hdb_cmp_action_pt_inst.cpp, Line: 5855
cap_inst
Stack Trace:
   0xcd396: remove_from_cache + 0xe6 (DB_HDB)
   0x85f66: __allmul + 0x28f16 (DB_HDB)
   0xcf769: HDB_CMP_MANAGER_IMPL::read_pdb + 0x49 (DB_HDB)
    0x5f44: DBMUI_MANAGER::get_hdb_cmp_action_pt_instance + 0x184 (DB_DBMUI)
   0x2b048: RPWQ_TABLE::load_rdb_table + 0xf8 (resr_rpwq)
   0x2c907: RPWQ_PAGE_CONTAINER::open_table_type + 0x57 (resr_rpwq)
   0x2f97a: RPWQ_PAGE_CONTAINER::open_page + 0x33a (resr_rpwq)
   0x2fbf6: RPWQ_PAGE_CONTAINER::qt_metacall + 0x56 (resr_rpwq)
   0xe0eae: QMetaObject::metacall + 0x3e (QtCore4)
    0x5243: RPWQ_REPORT_TOC_VIEW::on_clicked + 0x153 (resr_rpwq)
    0x5330: RPWQ_REPORT_TOC_VIEW::open_item + 0xb0 (resr_rpwq)
   0x267ae: RPWQ_REPORT_TOC_VIEW::load_report + 0x49e (resr_rpwq)
   0x27da8: RPWQ_REPORT_TOC_DOCK_WIDGET::load_report + 0x68 (resr_rpwq)
    0x2d46: RPWQ_FRAME_WND::do_load_report + 0x46 (resr_rpwq)
    0xd350: RPWQ_NTFQ_HANDLER::on_rdb_cache_freed + 0x30 (resr_rpwq)
    0x11fb: NTFQ_NOTIFY_TARGET::dispatch_notify_msg + 0x3b (ccl_ntfq)
    0x1312: NTFQ_NOTIFY_TARGET::on_ntfq_notify + 0x22 (ccl_ntfq)
    0x1750: NTFQ_NOTIFY_MGR::broadcast_notify + 0x60 (ccl_ntfq)
    0x1947: ntfq_broadcast_notify + 0x17 (ccl_ntfq)
    0x12bc: DBMUI_MANAGER::on_rdb_updated + 0x2c (DB_DBMUI)
    0x11fb: NTFQ_NOTIFY_TARGET::dispatch_notify_msg + 0x3b (ccl_ntfq)
    0x1312: NTFQ_NOTIFY_TARGET::on_ntfq_notify + 0x22 (ccl_ntfq)
    0x1750: NTFQ_NOTIFY_MGR::broadcast_notify + 0x60 (ccl_ntfq)
    0x1947: ntfq_broadcast_notify + 0x17 (ccl_ntfq)
   0x301be: FLOW_STATUS::refresh + 0x1e (sys_flow)
   0x11c25: flow_generic_parallel_call_helper + 0xc05 (sys_flow)
   0x12190: flow_generic_parallel_call + 0x10 (sys_flow)
   0x1e98a: flow_full_compile_or_simulate + 0x16ba (sys_flow)
   0x1f22f: flow_incremental_compilation + 0x1f (sys_flow)
   0x14dad: _flow_execute_custom + 0x6cd (sys_flow)
   0x15814: _flow_execute + 0x44 (sys_flow)
   0x706a2: QUI_APP::on_start_flow + 0x2a2 (SYS_QUI)
   0x555fa: QUI_APP::qt_metacall + 0xba (SYS_QUI)
   0xe0eae: QMetaObject::metacall + 0x3e (QtCore4)
   0x1c6b9: QApplication::event + 0x359 (QtGui4)
   0xfcba2: QEventDispatcherWin32::event + 0x112 (QtCore4)
   0xdbaac: QCoreApplicationPrivate::sendThroughApplicationEventFilters + 0x5c (QtCore4)
   0x1d068: QApplicationPrivate::notify_helper + 0xb8 (QtGui4)
   0x1f94c: QApplication::notify + 0x136c (QtGui4)
```

Figure 1: Same old problem

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

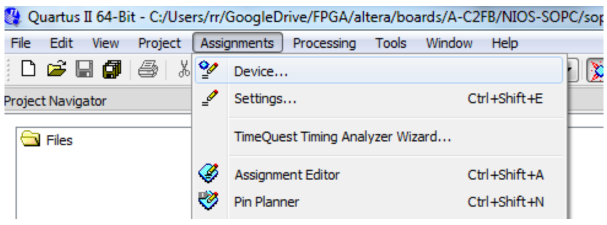
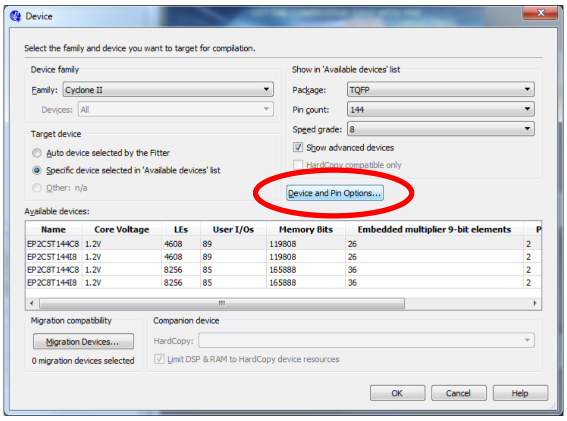

Quartus II 64-Bit - C:/Users/rr/GoogleDrive/FPGA/altera/boards/A-C2FB/NIOS-SOPC/sop
File
Edit
View
Project
Assignments
Processing
Tools
Window
Help
Project Navigator
Files
Device...
Settings... Ctrl+Shift+E
TimeQuest Timing Analyzer Wizard...
Assignment Editor Ctrl+Shift+A
Pin Planner Ctrl+Shift+N

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.



Device

Select the family and device you want to target for compilation.

Device family
Family: Cyclone II
Devices: All

Show in 'Available devices' list
Package: TQFP
Pin count: 144
Speed grade: 8
[✓] Show advanced devices
[ ] HardCopy compatible only

Target device
( ) Auto device selected by the Fitter
(●) Specific device selected in 'Available devices' list
( ) Other: n/a

Device and Pin Options...

Available devices:

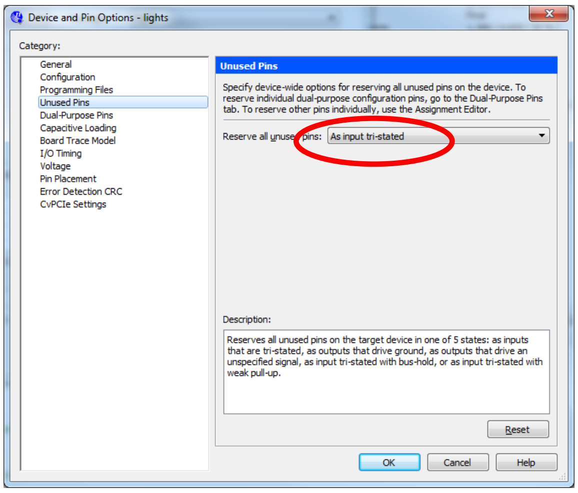
Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded multiplier 9-bit elements	P
EP2C5T144C8	1.2V	4608	89	119808	26	2
EP2C5T144I8	1.2V	4608	89	119808	26	2
EP2C8T144C8	1.2V	8256	85	165888	36	2
EP2C8T144I8	1.2V	8256	85	165888	36	2

Migration compatibility
Migration Devices...
0 migration devices selected

Companion device
HardCopy:
[✓] Limit DSP & RAM to HardCopy device resources

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

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



Specify device-wide options for reserving all unused pins on the device. To reserve individual dual-purpose configuration pins, go to the Dual-Purpose Pins tab. To reserve other pins individually, use the Assignment Editor.

Reserve all unused pins: As input tri-stated

Description:

Reserves all unused pins on the target device in one of 5 states: as inputs that are tri-stated, as outputs that drive ground, as outputs that drive an unspecified signal, as input tri-stated with bus-hold, or as input tri-stated with weak pull-up.

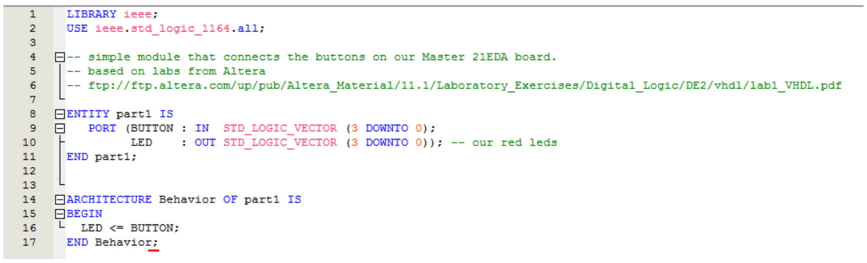
Figure 4. Tell the pins to be "quiet"

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

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



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- simple module that connects the buttons on our Master 21EDA board.
-- based on labs from Altera
-- ftp://ftp.altera.com/up/pub/Altera_Material/11.1/Laboratory_Exercises/Digital_Logic/DE2/vhdl/lab1_VHDL.pdf

ENTITY part1 IS
   PORT (BUTTON : IN  STD_LOGIC_VECTOR (3 DOWNTO 0);
         LED    : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)); -- our red leds
END part1;


ARCHITECTURE Behavior OF part1 IS
BEGIN
  LED <= BUTTON;
END Behavior;
```

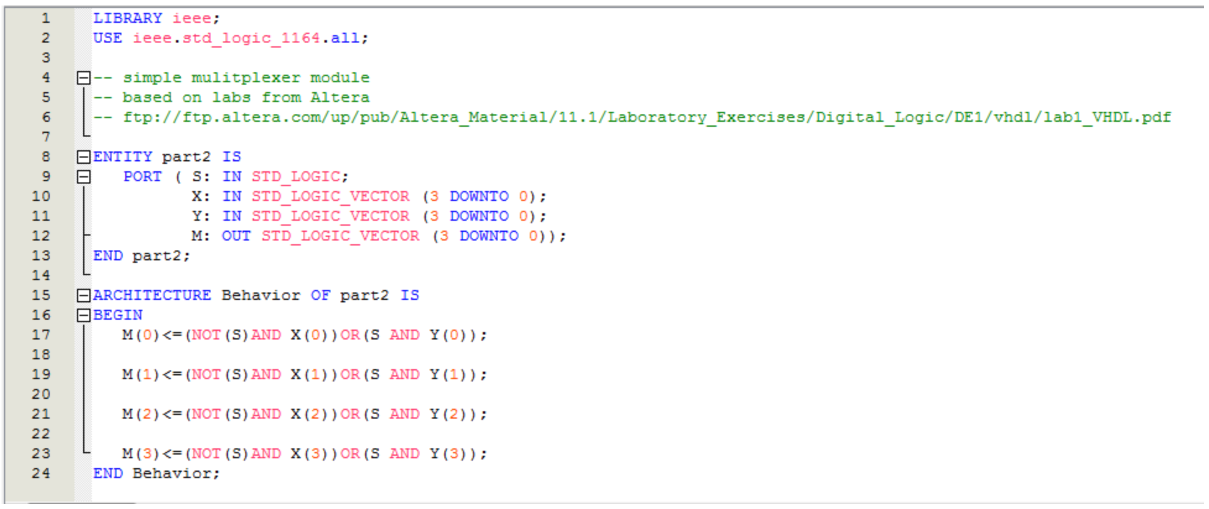
Figure 5: Our VHDL code in place of lab's Figure 1.

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).



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- simple mulitplexer module
-- based on labs from Altera
-- ftp://ftp.altera.com/up/pub/Altera_Material/11.1/Laboratory_Exercises/Digital_Logic/DE1/vhdl/lab1_VHDL.pdf

ENTITY part2 IS
   PORT ( S: IN STD_LOGIC;
          X: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
          Y: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
          M: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END part2;

ARCHITECTURE Behavior OF part2 IS
BEGIN
   M(0)<=(NOT(S)AND X(0))OR(S AND Y(0));

   M(1)<=(NOT(S)AND X(1))OR(S AND Y(1));

   M(2)<=(NOT(S)AND X(2))OR(S AND Y(2));

   M(3)<=(NOT(S)AND X(3))OR(S AND Y(3));
END Behavior;
```

Figure 6: 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".

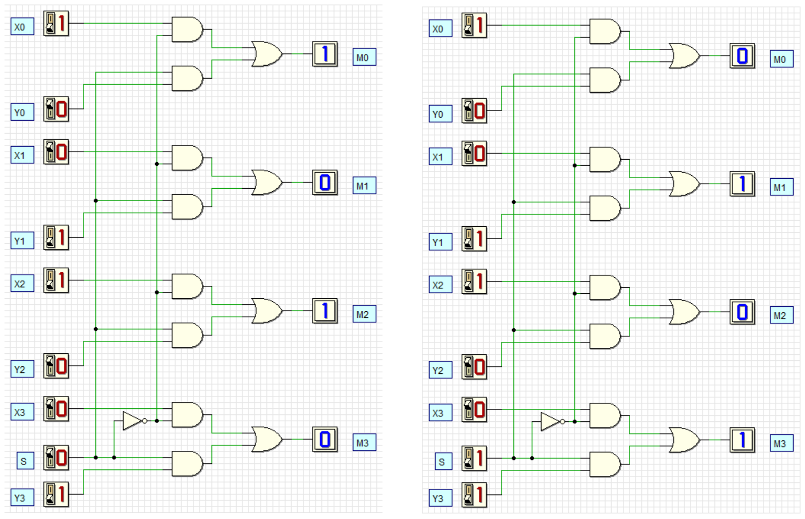
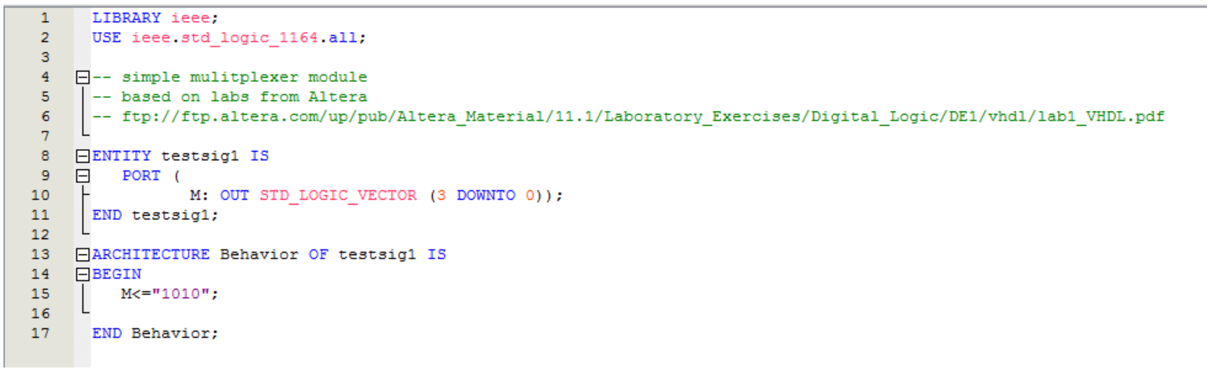


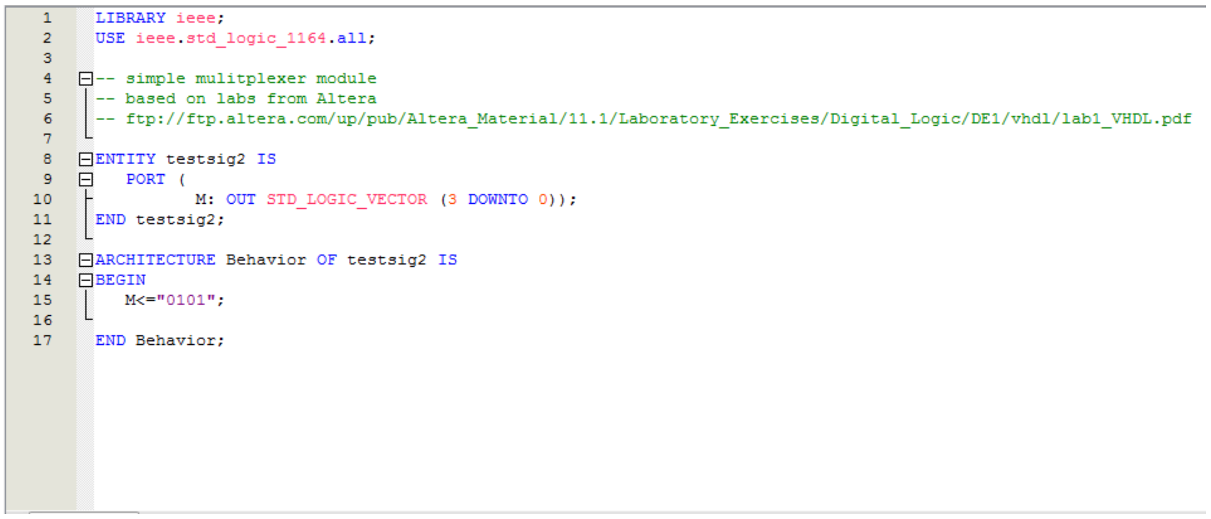
Figure 7: Test harness design for Part II

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



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- simple mulitplexer module
-- based on labs from Altera
-- ftp://ftp.altera.com/up/pub/Altera_Material/11.1/Laboratory_Exercises/Digital_Logic/DE1/vhdl/lab1_VHDL.pdf

ENTITY testsig1 IS
   PORT (
          M: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END testsig1;

ARCHITECTURE Behavior OF testsig1 IS
BEGIN
   M<="1010";

END Behavior;
```

Figure 8: One lot of test signals



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- simple mulitplexer module
-- based on labs from Altera
-- ftp://ftp.altera.com/up/pub/Altera_Material/11.1/Laboratory_Exercises/Digital_Logic/DE1/vhdl/lab1_VHDL.pdf

ENTITY testsig2 IS
   PORT (
          M: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END testsig2;

ARCHITECTURE Behavior OF testsig2 IS
BEGIN
   M<="0101";

END Behavior;
```

Figure 9: 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).

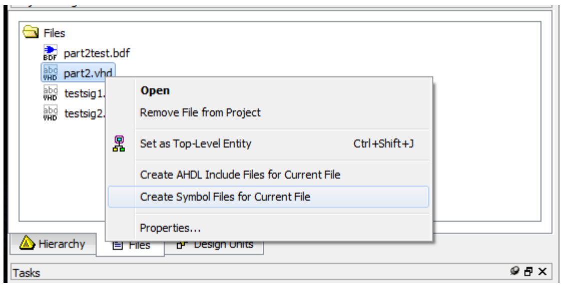


Figure 10: for each *.vhd file

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

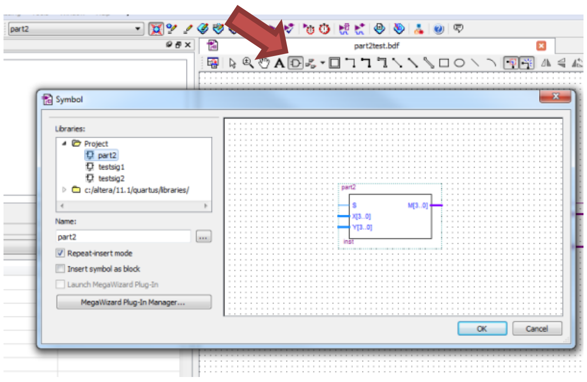


Figure 11: Project Symbol Library

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

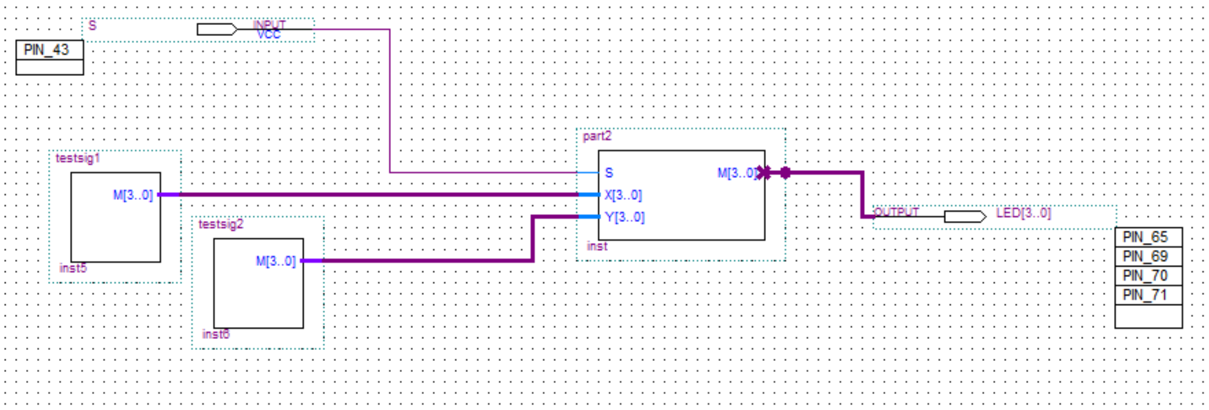
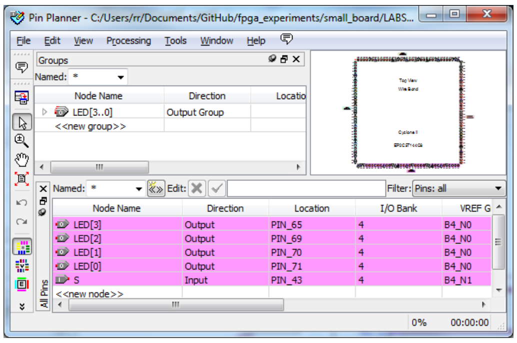


Figure 12: The test harness for Part II

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


Pin Planner - C:/Users/rr/Documents/GitHub/fpga_experiments/small_board/LABS...

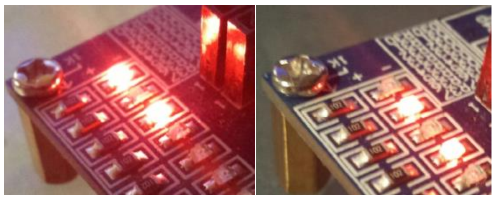
Node Name	Direction	Locatio
LED[3..0]	Output Group	
<<new group>>		

Node Name	Direction	Location	I/O Bank	VREF G
LED[3]	Output	PIN_65	4	B4_N0
LED[2]	Output	PIN_69	4	B4_N0
LED[1]	Output	PIN_70	4	B4_N0
LED[0]	Output	PIN_71	4	B4_N0
S	Input	PIN_43	4	B4_N1
<<new node>>				

0% 00:00:00

Figure 13: 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).

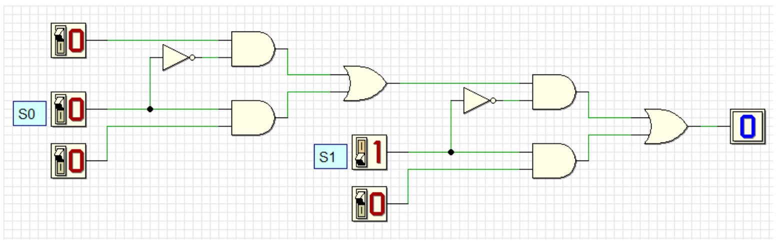
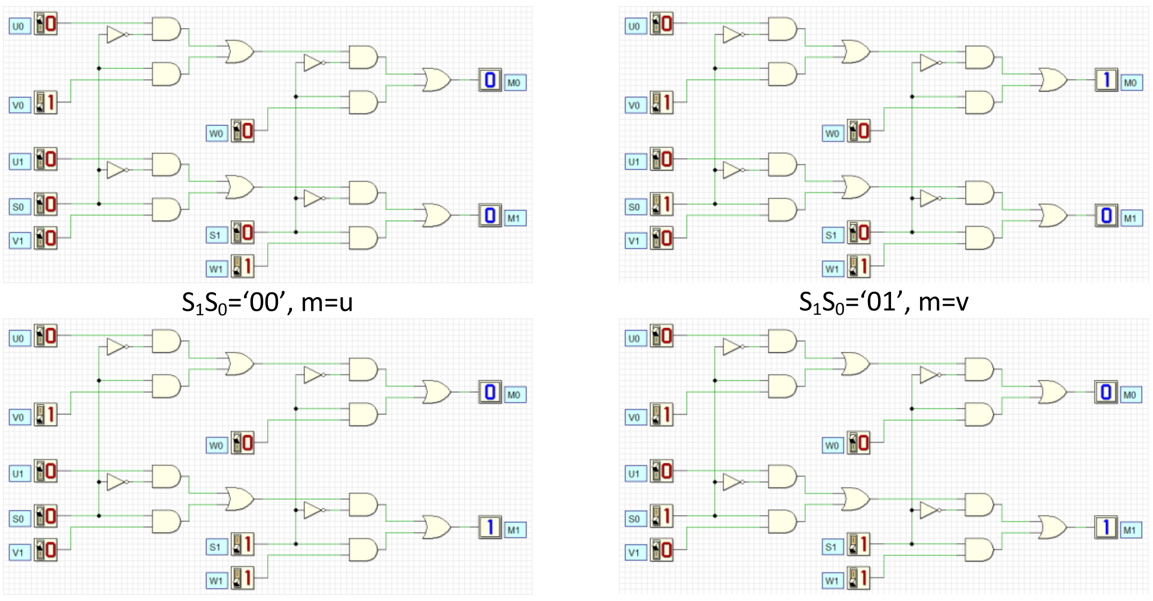


Figure 14: A one-bit wide 3-to-1 multiplexer

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



S_1S_0='00', m=u

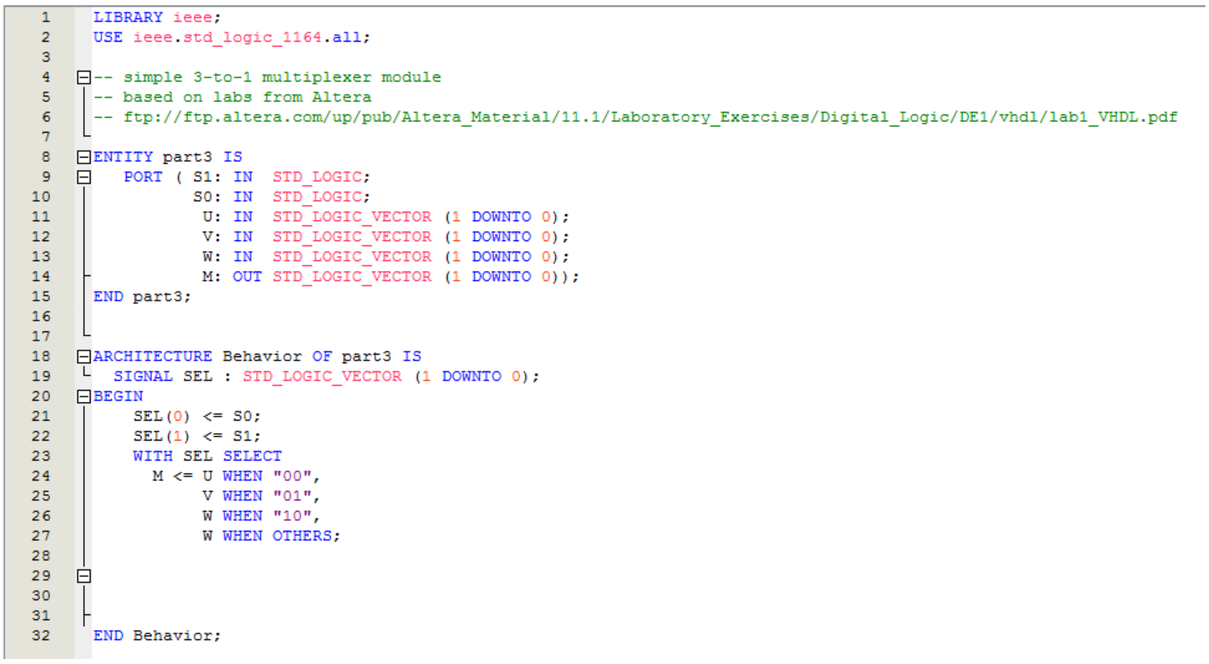
S_1S_0='01', m=v

S_1S_0='10', m=w

S_1S_0='11', m=w

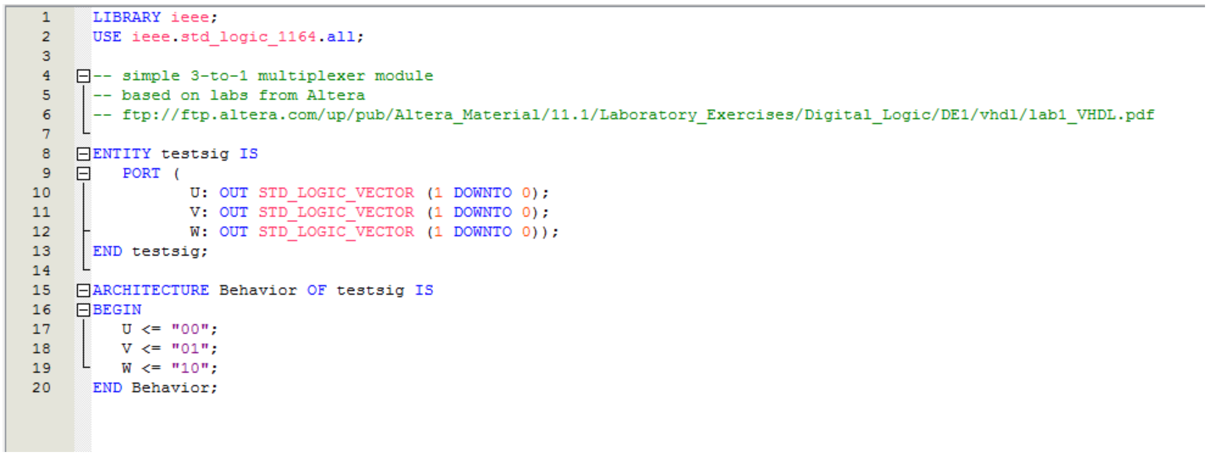
Figure 15: A two-bit wide 3-to-1 multiplexer

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



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- simple 3-to-1 multiplexer module
-- based on labs from Altera
-- ftp://ftp.altera.com/up/pub/Altera_Material/11.1/Laboratory_Exercises/Digital_Logic/DE1/vhdl/lab1_VHDL.pdf

ENTITY part3 IS
   PORT ( S1: IN  STD_LOGIC;
          S0: IN  STD_LOGIC;
           U: IN  STD_LOGIC_VECTOR (1 DOWNTO 0);
           V: IN  STD_LOGIC_VECTOR (1 DOWNTO 0);
           W: IN  STD_LOGIC_VECTOR (1 DOWNTO 0);
           M: OUT STD_LOGIC_VECTOR (1 DOWNTO 0));
END part3;


ARCHITECTURE Behavior OF part3 IS
  SIGNAL SEL : STD_LOGIC_VECTOR (1 DOWNTO 0);
BEGIN
    SEL(0) <= S0;
    SEL(1) <= S1;
    WITH SEL SELECT
      M <= U WHEN "00",
           V WHEN "01",
           W WHEN "10",
           W WHEN OTHERS;




END Behavior;
```

Figure 16: The Answer for Part III



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- simple 3-to-1 multiplexer module
-- based on labs from Altera
-- ftp://ftp.altera.com/up/pub/Altera_Material/11.1/Laboratory_Exercises/Digital_Logic/DE1/vhdl/lab1_VHDL.pdf

ENTITY testsig IS
   PORT (
          U: OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
          V: OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
          W: OUT STD_LOGIC_VECTOR (1 DOWNTO 0));
END testsig;

ARCHITECTURE Behavior OF testsig IS
BEGIN
   U <= "00";
   V <= "01";
   W <= "10";
END Behavior;
```

Figure 17: Test signals for Part III

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

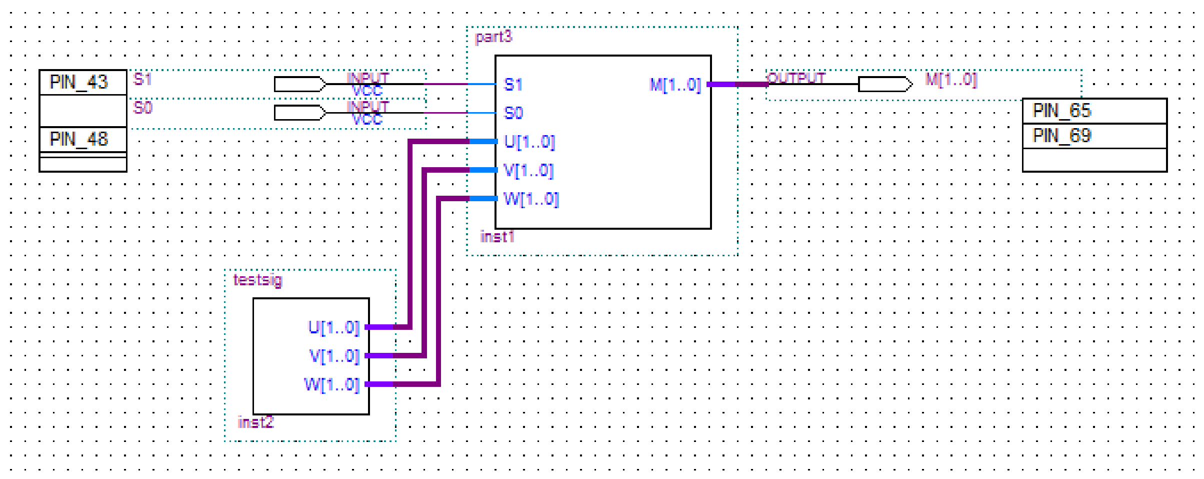


Figure 18: The test harness for Part III

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

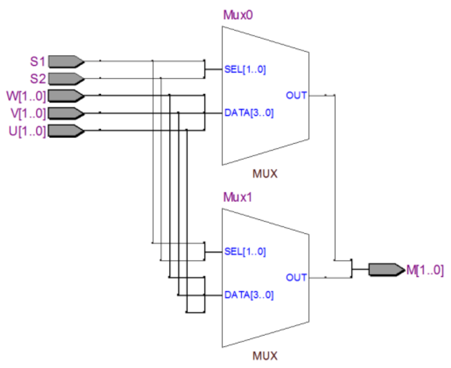


Figure 19: two 1 bit wide 3-to-1 multiplexers

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.

s_1 s_0	m
0 0	u
0 1	v
1 0	w
1 1	w

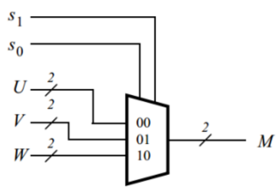


Figure 20: Target behaviour

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

Table 1: 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:

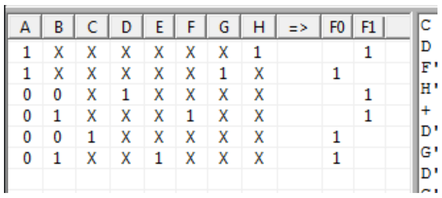
Factored:

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 ;



A	B	C	D	E	F	G	H	=>	F0	F1
1	X	X	X	X	X	X	1			1
1	X	X	X	X	X	1	X		1	
0	0	X	1	X	X	X	X			1
0	1	X	X	X	1	X	X			1
0	0	1	X	X	X	X	X		1	
0	1	X	X	1	X	X	X		1	

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.

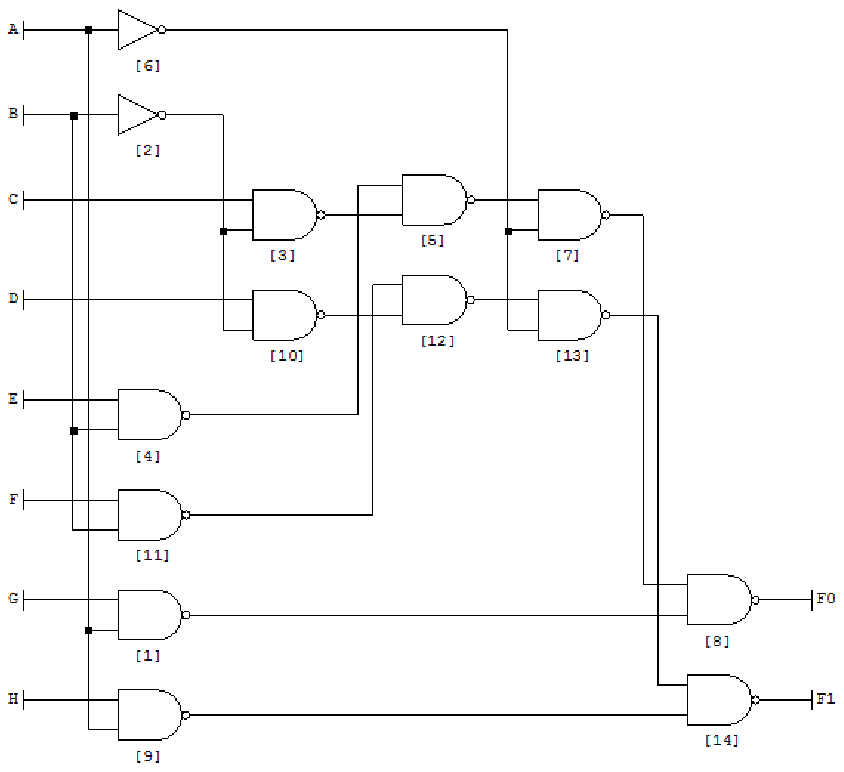
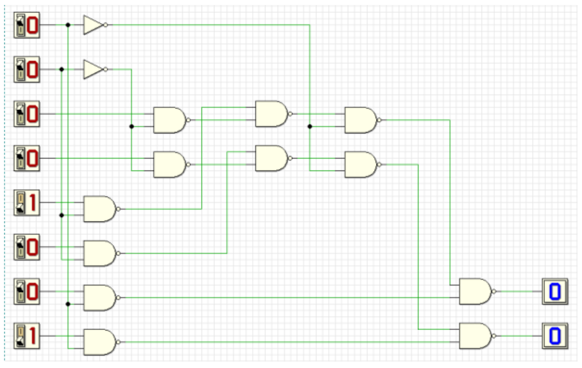
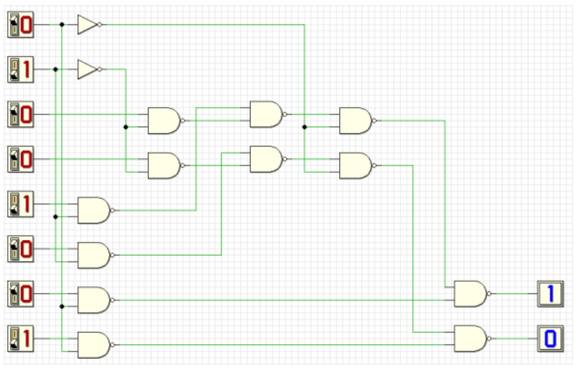


Figure 21: Circuit design for Part III by LogicFriday

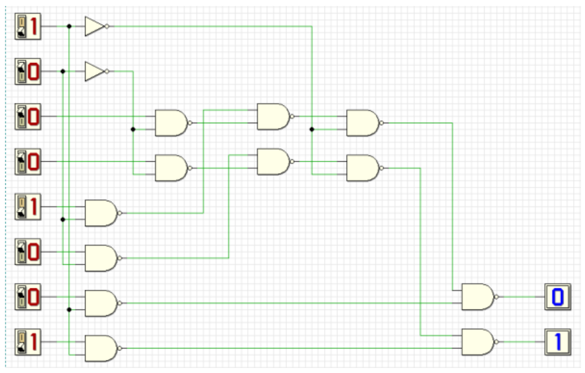
Checking it out with DEEDS [4] we get Figure 22 below (see file part3b.pbs).



S_1S_0='00', m=u



S_1S_0='01', m=v


S_1S_0='10', m=w

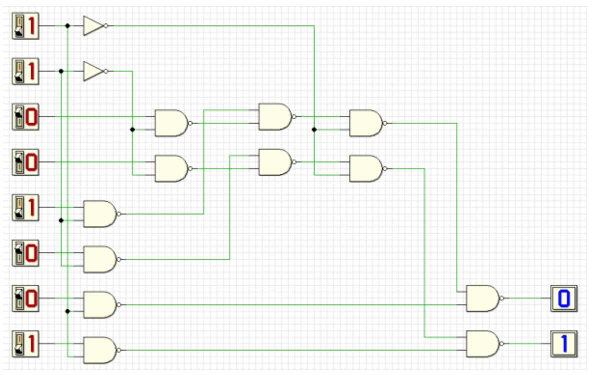

S_1S_0='11', m=w

Figure 22: 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:

$M(0) = S_1 W_0 + S_1' S_0' V_0 + S_1' S_0 W_0$;

$M(1) = S_1 W_1 + S_1' S_0' V_1 + S_1' S_0 W_1$;

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

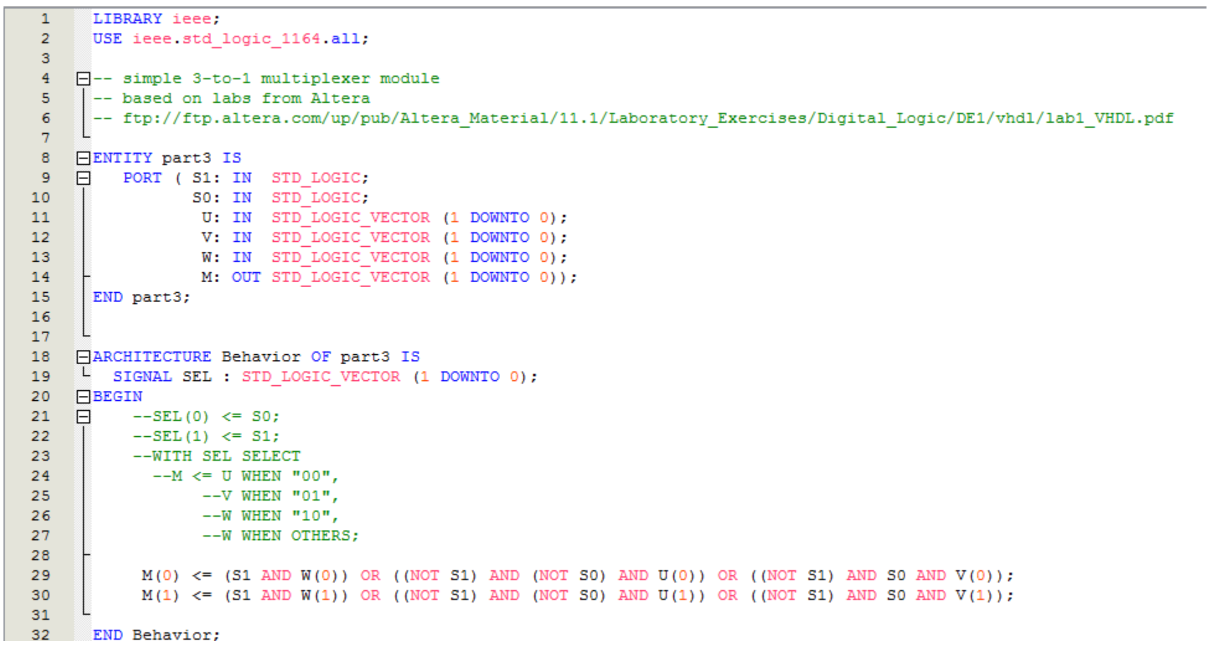

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- simple 3-to-1 multiplexer module
-- based on labs from Altera
-- ftp://ftp.altera.com/up/pub/Altera_Material/11.1/Laboratory_Exercises/Digital_Logic/DE1/vhdl/lab1_VHDL.pdf

ENTITY part3 IS
   PORT ( S1: IN  STD_LOGIC;
          S0: IN  STD_LOGIC;
           U: IN  STD_LOGIC_VECTOR (1 DOWNTO 0);
           V: IN  STD_LOGIC_VECTOR (1 DOWNTO 0);
           W: IN  STD_LOGIC_VECTOR (1 DOWNTO 0);
           M: OUT STD_LOGIC_VECTOR (1 DOWNTO 0));
END part3;


ARCHITECTURE Behavior OF part3 IS
  SIGNAL SEL : STD_LOGIC_VECTOR (1 DOWNTO 0);
BEGIN
    --SEL(0) <= S0;
    --SEL(1) <= S1;
    --WITH SEL SELECT
      --M <= U WHEN "00",
           --V WHEN "01",
           --W WHEN "10",
           --W WHEN OTHERS;

     M(0) <= (S1 AND W(0)) OR ((NOT S1) AND (NOT S0) AND U(0)) OR ((NOT S1) AND S0 AND V(0));
     M(1) <= (S1 AND W(1)) OR ((NOT S1) AND (NOT S0) AND U(1)) OR ((NOT S1) AND S0 AND V(1));

END Behavior;
```

Figure 23: Part II by equation

This gives us the following RTL (Figure 24).

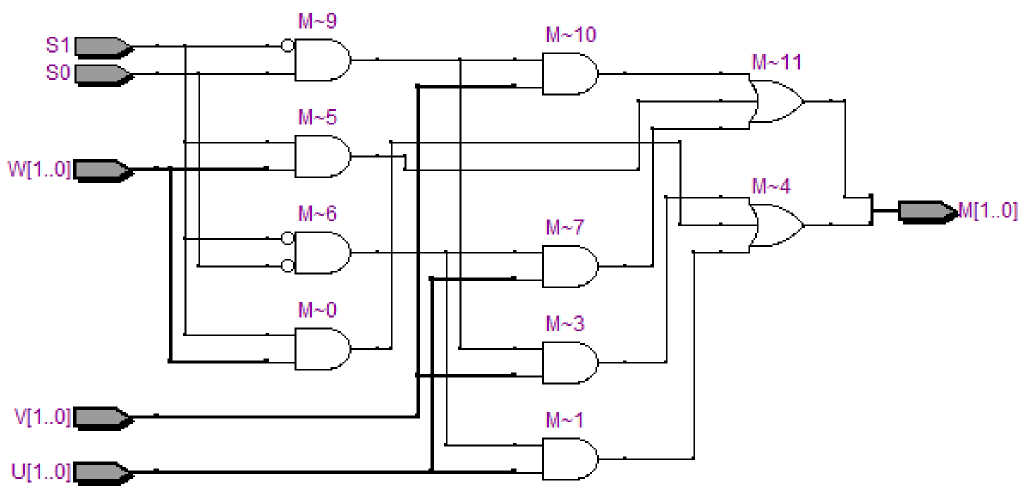
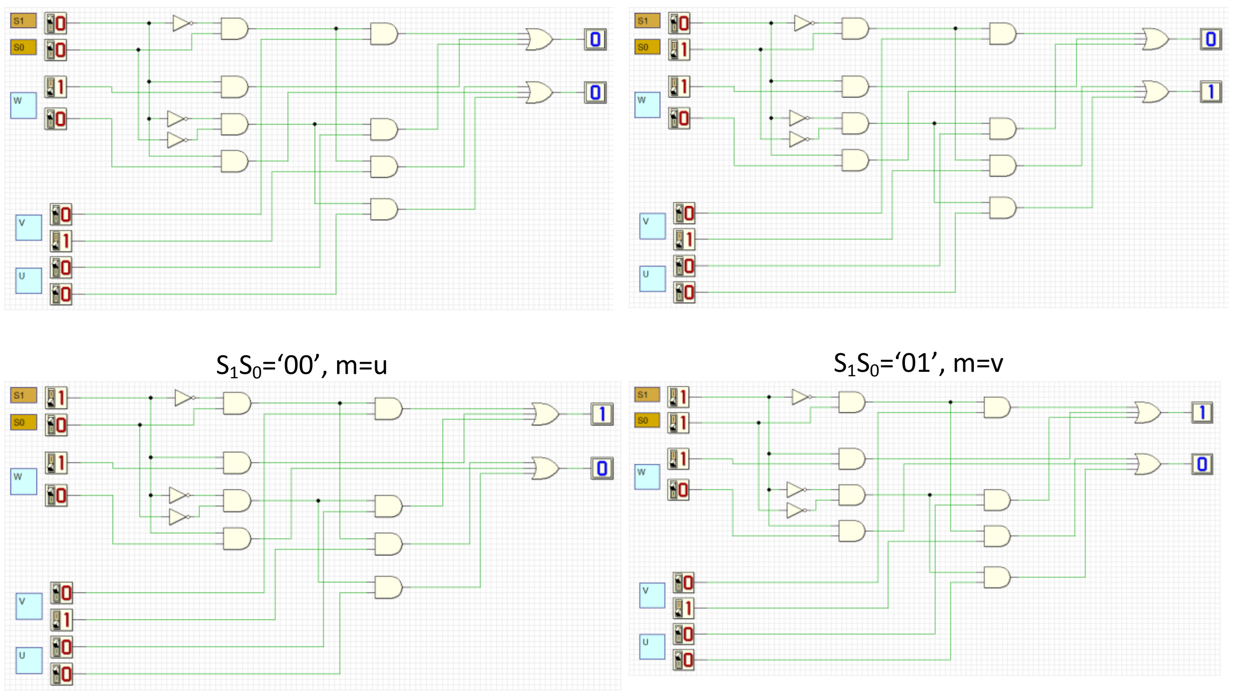


Figure 24: RTL for Part III by equation

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



S_1S_0='00', m=u

S_1S_0='01', m=v

S_1S_0='10', m=w

S_1S_0='11', m=w

Figure 25: 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

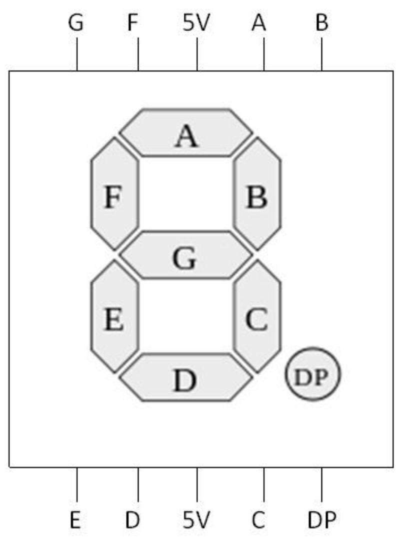


Figure 26: Typical convention for labelling an LED

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

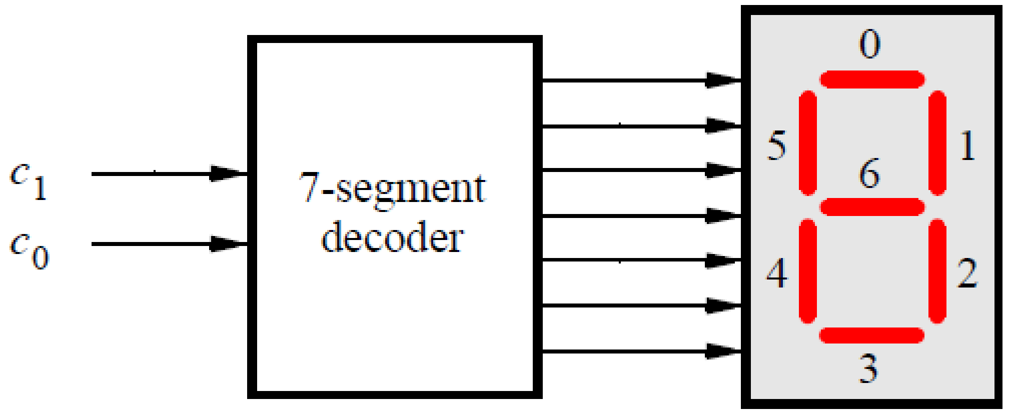


Figure 27: *Figure 6. A 7-segment decoder.*

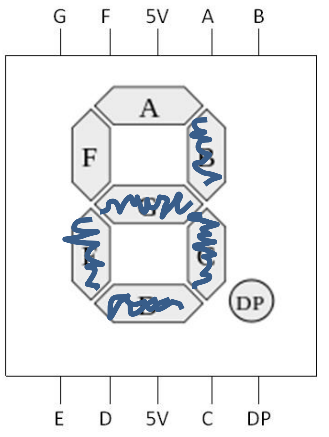
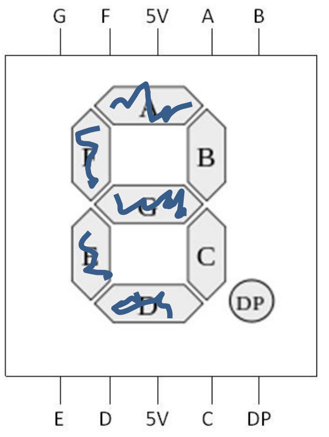
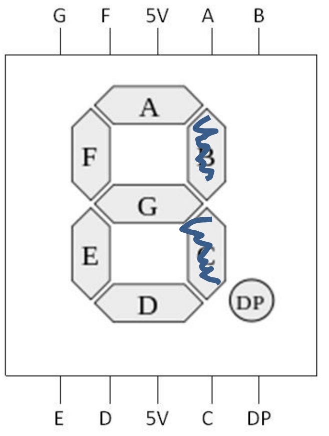
LED_A	0	PIN_93
LED_B	1	PIN_92
LED_C	2	PIN_87
LED_D	3	PIN_86
LED_E	4	PIN_55
LED_F	5	PIN_58
LED_G	6	PIN_79
LED_H(DP)	7	PIN_113

c_1c_0	Character
00	d
01	E
10	1
11	

Figure 28: *Table 1. Character codes.*

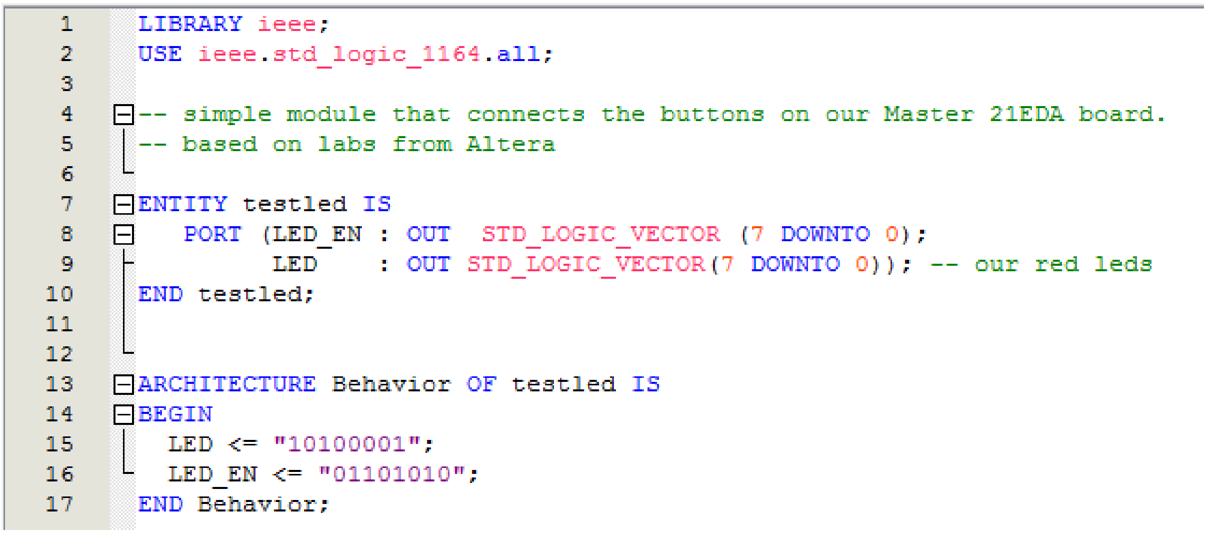
We map out the segments we need for "dE1" below (Table 2)[1].

Table 2: 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"				
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"				
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"				

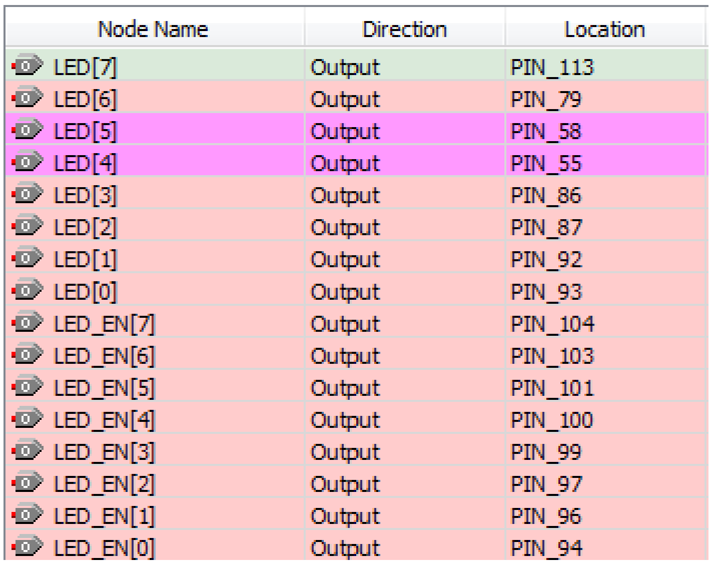
[1] We could actually change the displayed chars from "dE1" to "EdA" for our board, do so if you like, just change the Table 2 entries.

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



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- simple module that connects the buttons on our Master 21EDA board.
-- based on labs from Altera

ENTITY testled IS
   PORT (LED_EN : OUT  STD_LOGIC_VECTOR (7 DOWNTO 0);
           LED    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)); -- our red leds
END testled;


ARCHITECTURE Behavior OF testled IS
BEGIN
  LED <= "10100001";
  LED_EN <= "01101010";
END Behavior;
```

Figure 29: Test segment lighting



Node Name	Direction	Location
LED[7]	Output	PIN_113
LED[6]	Output	PIN_79
LED[5]	Output	PIN_58
LED[4]	Output	PIN_55
LED[3]	Output	PIN_86
LED[2]	Output	PIN_87
LED[1]	Output	PIN_92
LED[0]	Output	PIN_93
LED_EN[7]	Output	PIN_104
LED_EN[6]	Output	PIN_103
LED_EN[5]	Output	PIN_101
LED_EN[4]	Output	PIN_100
LED_EN[3]	Output	PIN_99
LED_EN[2]	Output	PIN_97
LED_EN[1]	Output	PIN_96
LED_EN[0]	Output	PIN_94

Figure 30: Pin allocations

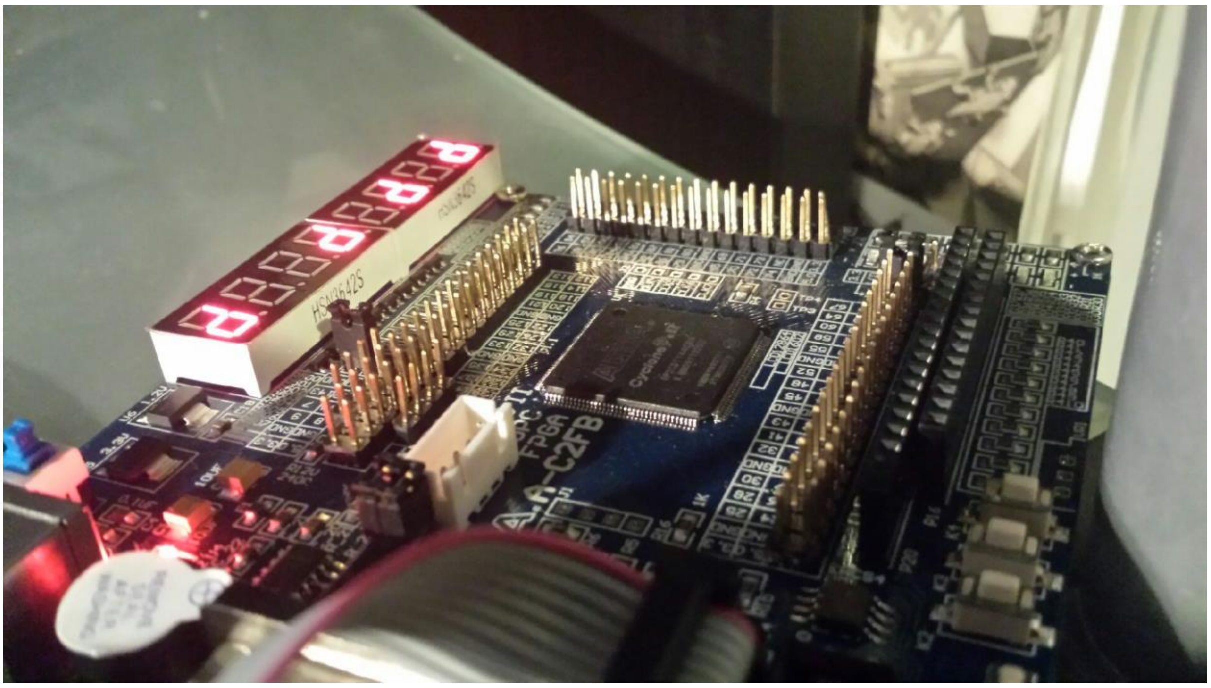
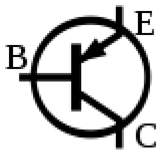


Figure 31: Resulting segment lighting

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").

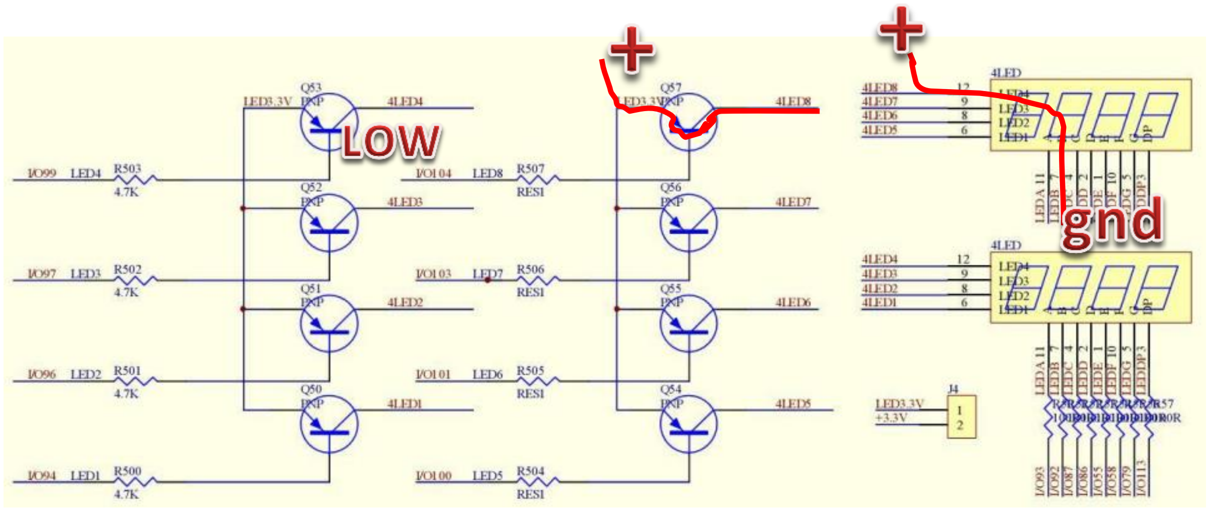
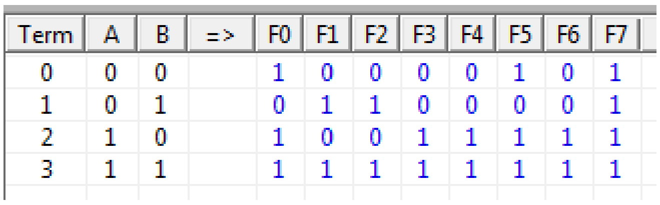


Figure 32: Low on "N" of PNP turns transistor "ON"

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



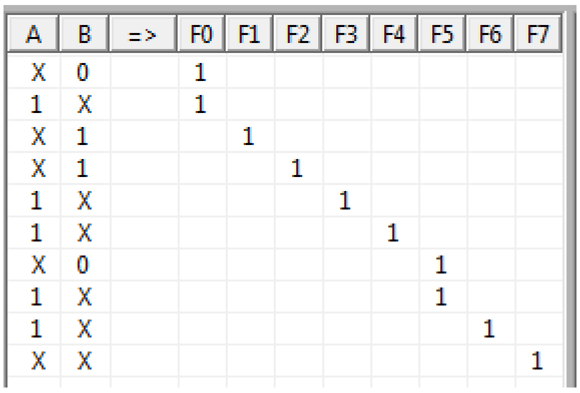
Term	A	B	=>	F0	F1	F2	F3	F4	F5	F6	F7
0	0	0		1	0	0	0	0	1	0	1
1	0	1		0	1	1	0	0	0	0	1
2	1	0		1	0	0	1	1	1	1	1
3	1	1		1	1	1	1	1	1	1	1

Figure 33: 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;
```

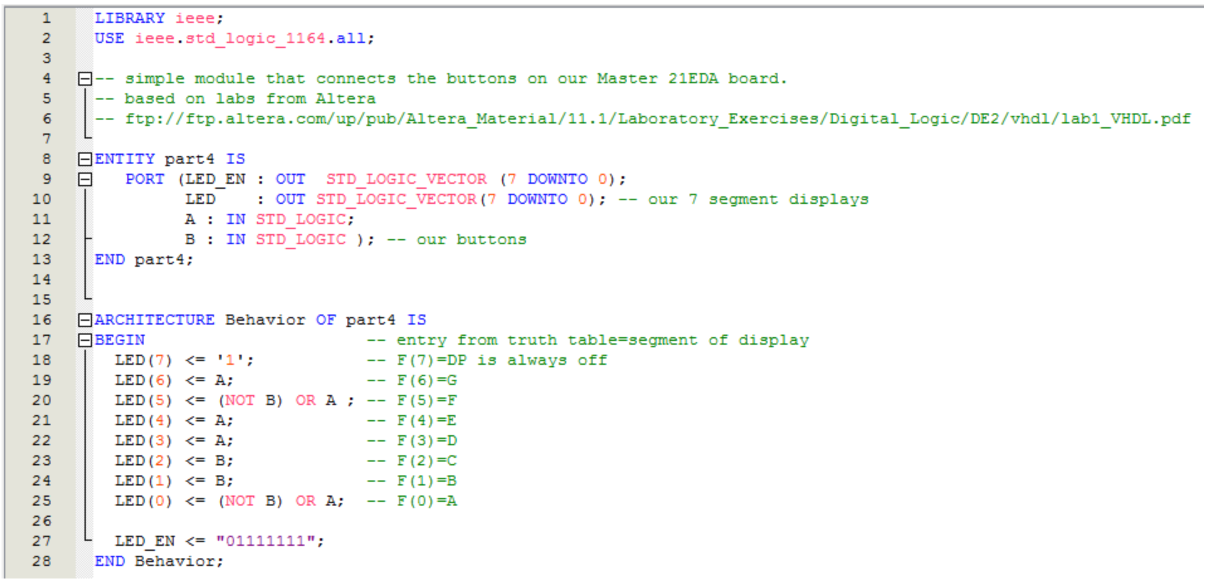
Tablewize this looks like the figure below (Figure 34).



A	B	=>	F0	F1	F2	F3	F4	F5	F6	F7
X	0		1							
1	X		1							
X	1			1						
X	1				1					
1	X					1				
1	X						1			
X	0							1		
1	X							1		
1	X								1	
X	X									1

Figure 34: Minimized truth table

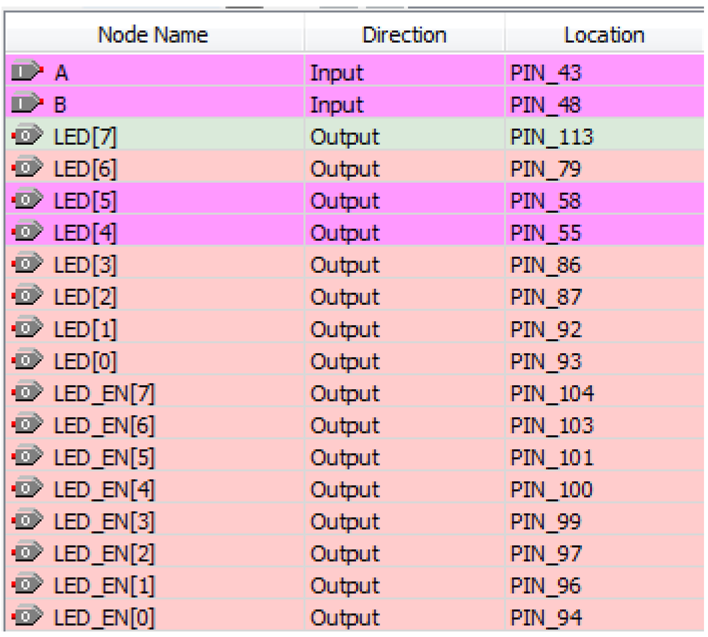
So coding it up (remembering to add button inputs) we get the code below (Figure 35). Note the LED_EN setting - we are using LED8 which is described in our board's manual as "the left one". You can then see the relationship between our seven segment display layout and the bit positions – neat yes?



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-- simple module that connects the buttons on our Master 21EDA board.
-- based on labs from Altera
-- ftp://ftp.altera.com/up/pub/Altera_Material/11.1/Laboratory_Exercises/Digital_Logic/DE2/vhdl/lab1_VHDL.pdf

ENTITY part4 IS
   PORT (LED_EN : OUT  STD_LOGIC_VECTOR (7 DOWNTO 0);
         LED    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0); -- our 7 segment displays
         A : IN STD_LOGIC;
         B : IN STD_LOGIC ); -- our buttons
END part4;


ARCHITECTURE Behavior OF part4 IS
BEGIN                          -- entry from truth table=segment of display
  LED(7) <= '1';               -- F(7)=DP is always off
  LED(6) <= A;                 -- F(6)=G
  LED(5) <= (NOT B) OR A ;     -- F(5)=F
  LED(4) <= A;                 -- F(4)=E
  LED(3) <= A;                 -- F(3)=D
  LED(2) <= B;                 -- F(2)=C
  LED(1) <= B;                 -- F(1)=B
  LED(0) <= (NOT B) OR A;      -- F(0)=A

  LED_EN <= "01111111";
END Behavior;
```

Figure 35: Part IV solution

Full pin allocation is below (Figure 36).



Node Name	Direction	Location
A	Input	PIN_43
B	Input	PIN_48
LED[7]	Output	PIN_113
LED[6]	Output	PIN_79
LED[5]	Output	PIN_58
LED[4]	Output	PIN_55
LED[3]	Output	PIN_86
LED[2]	Output	PIN_87
LED[1]	Output	PIN_92
LED[0]	Output	PIN_93
LED_EN[7]	Output	PIN_104
LED_EN[6]	Output	PIN_103
LED_EN[5]	Output	PIN_101
LED_EN[4]	Output	PIN_100
LED_EN[3]	Output	PIN_99
LED_EN[2]	Output	PIN_97
LED_EN[1]	Output	PIN_96
LED_EN[0]	Output	PIN_94

Figure 36: 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).

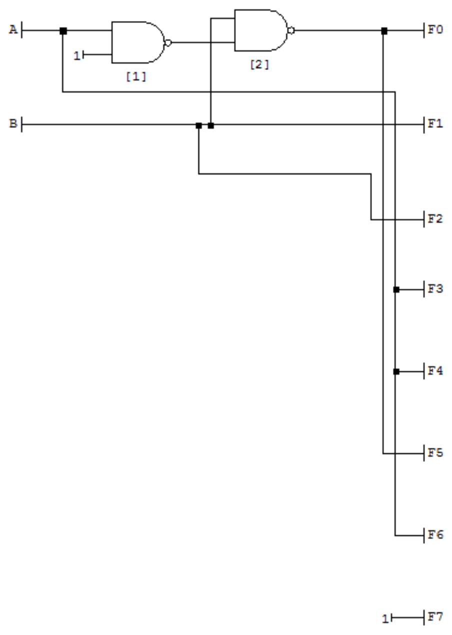


Figure 37: Simple enough

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

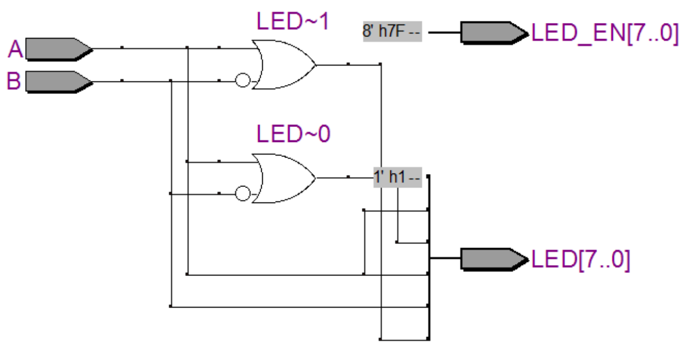


Figure 38: Part IV RTL

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).

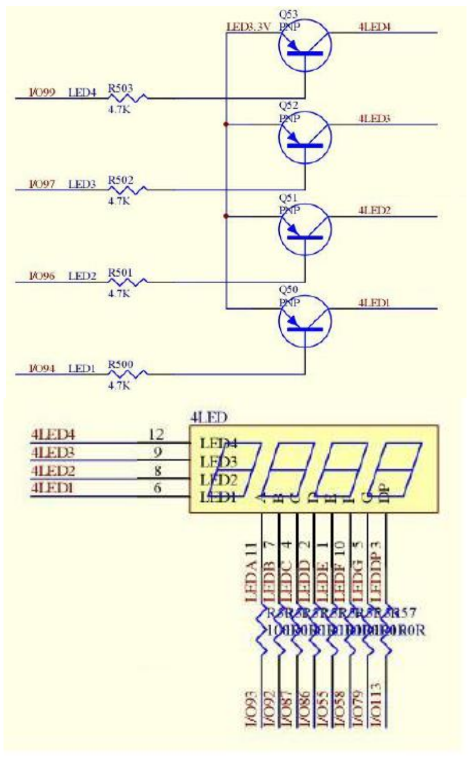
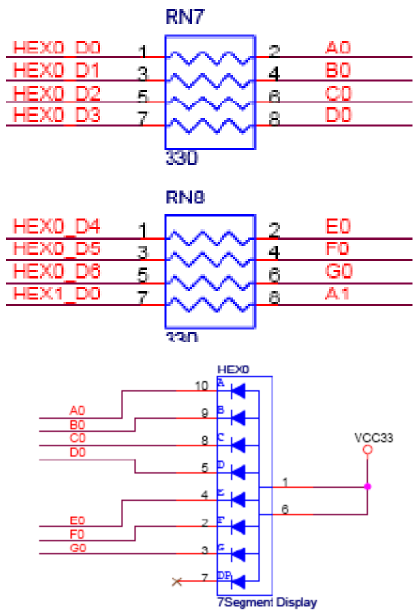
Master 21EDA Board	DE1 7-SEGMENT Design
	

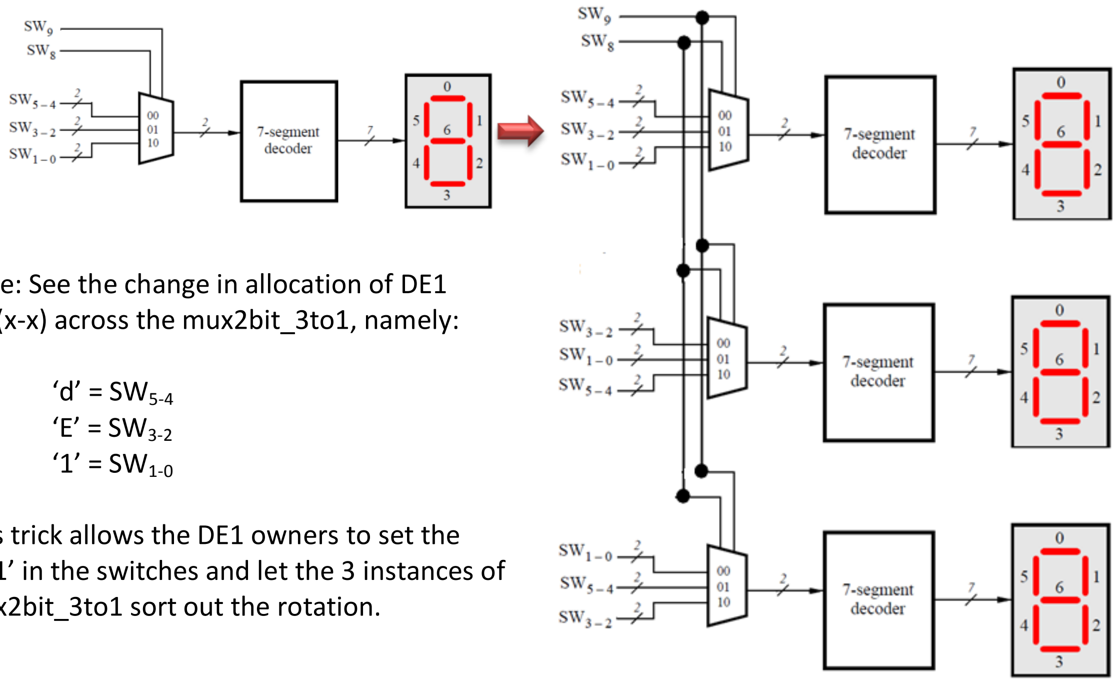
Figure 39: Spot the difference

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

SW_9 SW_8	Character pattern
00	d E 1
01	E 1 d
10	1 d E

Figure 40: 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' = SW_{5-4}
'E' = SW_{3-2}
'1' = SW_{1-0}

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 41: The DE1 Solution

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

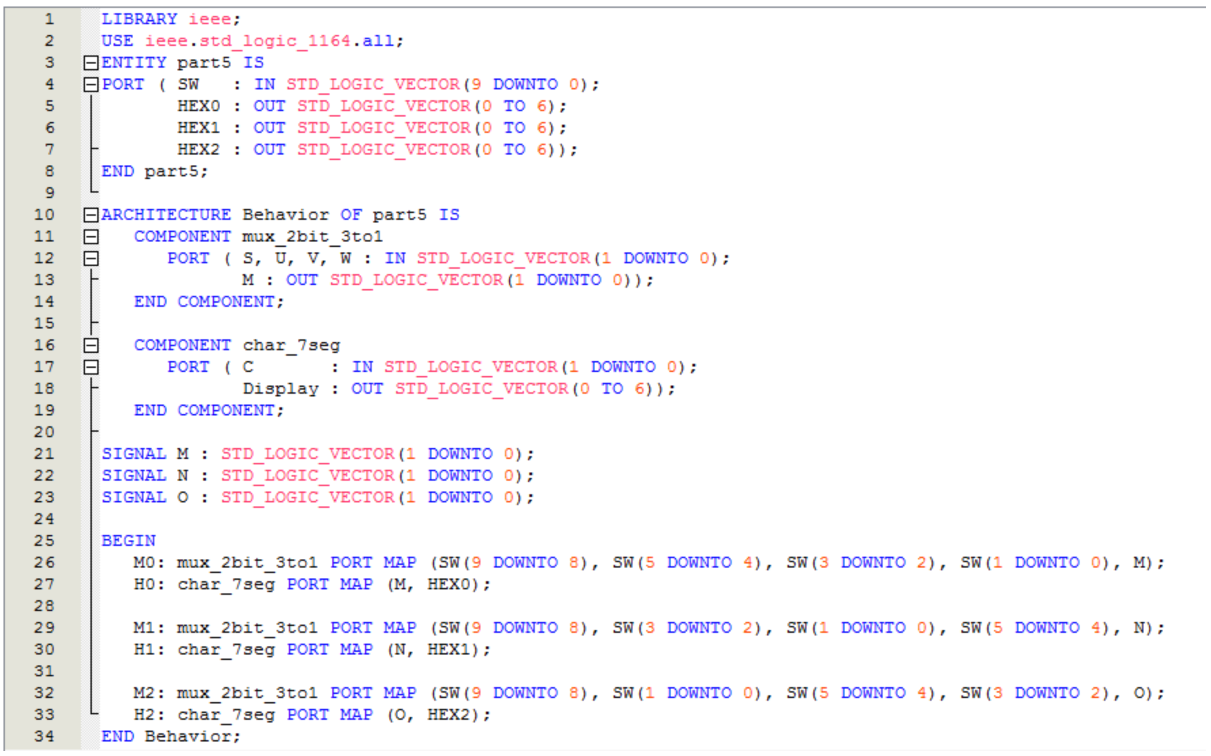

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY part5 IS
PORT ( SW   : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
       HEX0 : OUT STD_LOGIC_VECTOR(0 TO 6);
       HEX1 : OUT STD_LOGIC_VECTOR(0 TO 6);
       HEX2 : OUT STD_LOGIC_VECTOR(0 TO 6));
END part5;

ARCHITECTURE Behavior OF part5 IS
   COMPONENT mux_2bit_3to1
      PORT ( S, U, V, W : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
             M : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
   END COMPONENT;

   COMPONENT char_7seg
      PORT ( C       : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
             Display : OUT STD_LOGIC_VECTOR(0 TO 6));
   END COMPONENT;

SIGNAL M : STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL N : STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL O : STD_LOGIC_VECTOR(1 DOWNTO 0);

BEGIN
   M0: mux_2bit_3to1 PORT MAP (SW(9 DOWNTO 8), SW(5 DOWNTO 4), SW(3 DOWNTO 2), SW(1 DOWNTO 0), M);
   H0: char_7seg PORT MAP (M, HEX0);

   M1: mux_2bit_3to1 PORT MAP (SW(9 DOWNTO 8), SW(3 DOWNTO 2), SW(1 DOWNTO 0), SW(5 DOWNTO 4), N);
   H1: char_7seg PORT MAP (N, HEX1);

   M2: mux_2bit_3to1 PORT MAP (SW(9 DOWNTO 8), SW(1 DOWNTO 0), SW(5 DOWNTO 4), SW(3 DOWNTO 2), O);
   H2: char_7seg PORT MAP (O, HEX2);
END Behavior;
```

Figure 42: 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.

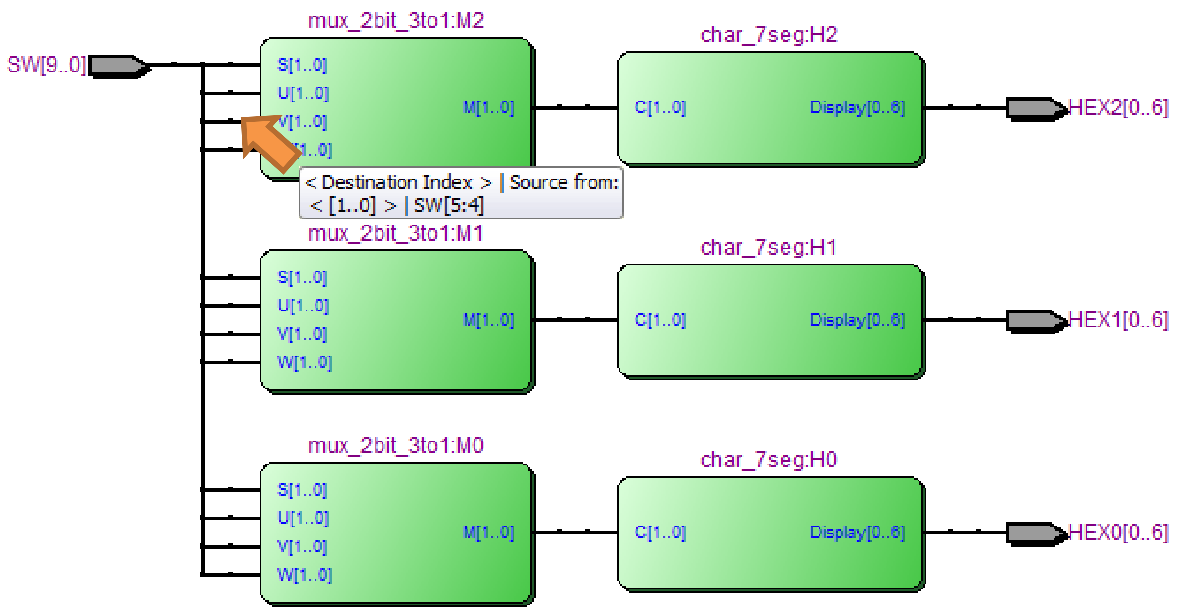


Figure 43: RTL to DE1 Solution

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

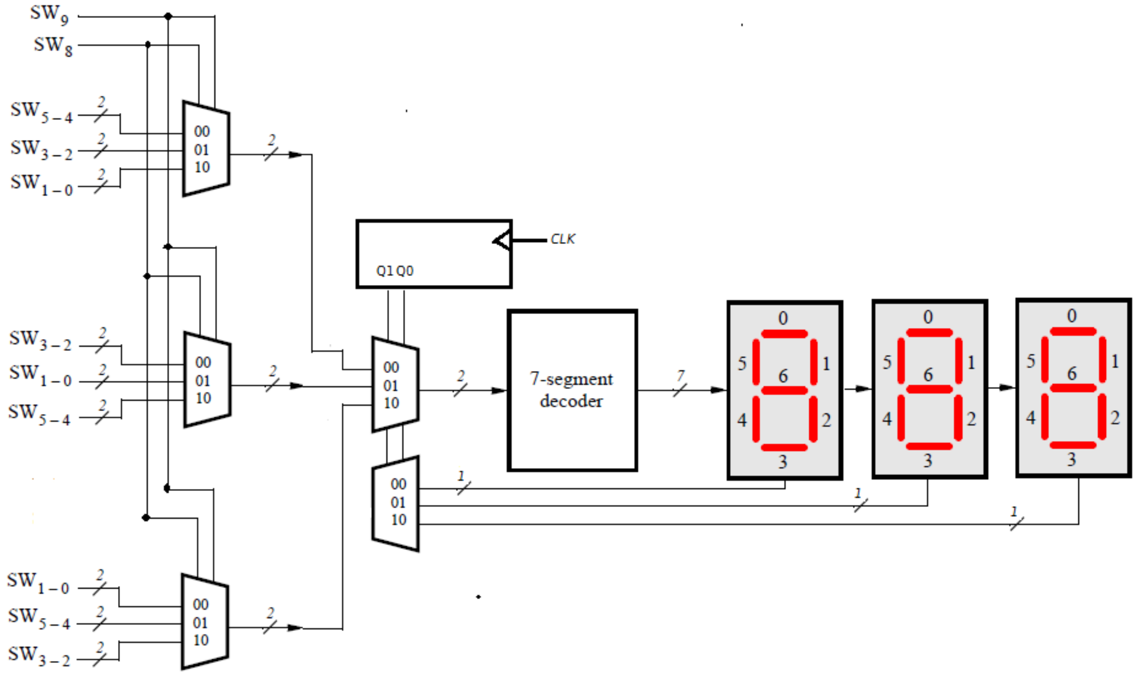
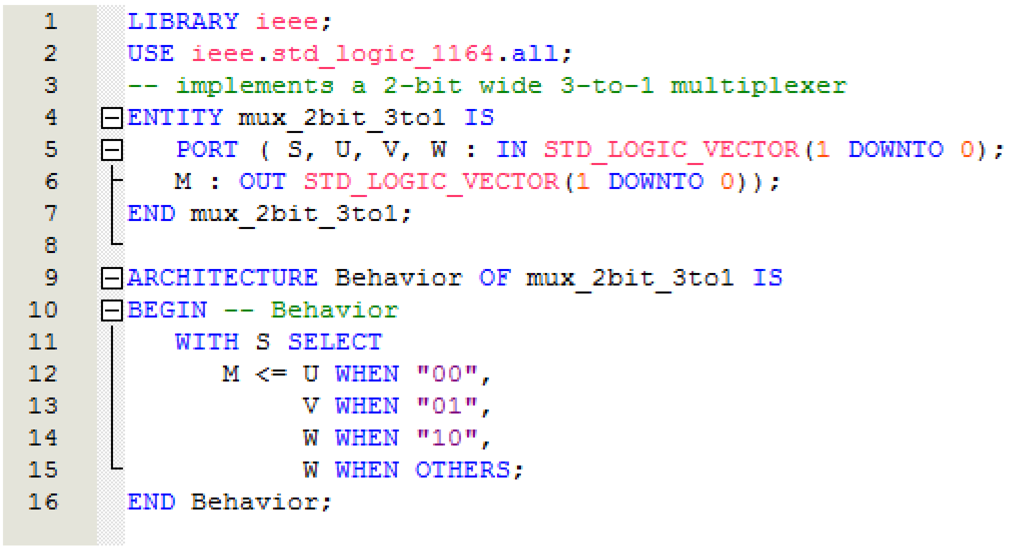


Figure 44: Our version of Part V Solution

So our version of the experiment uses an extra mux2bit_3to1, a counter of some description and a 1-of-3 decoder.

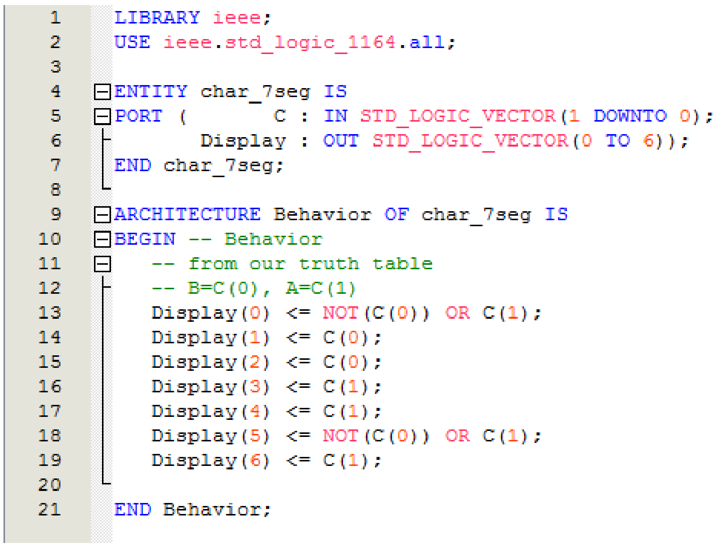
Our mux is from Part III (Figure 45).



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- implements a 2-bit wide 3-to-1 multiplexer
ENTITY mux_2bit_3to1 IS
   PORT ( S, U, V, W : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
   M : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
END mux_2bit_3to1;

ARCHITECTURE Behavior OF mux_2bit_3to1 IS
BEGIN -- Behavior
   WITH S SELECT
      M <= U WHEN "00",
           V WHEN "01",
           W WHEN "10",
           W WHEN OTHERS;
END Behavior;
```

Figure 45: mux for Part V from Part III

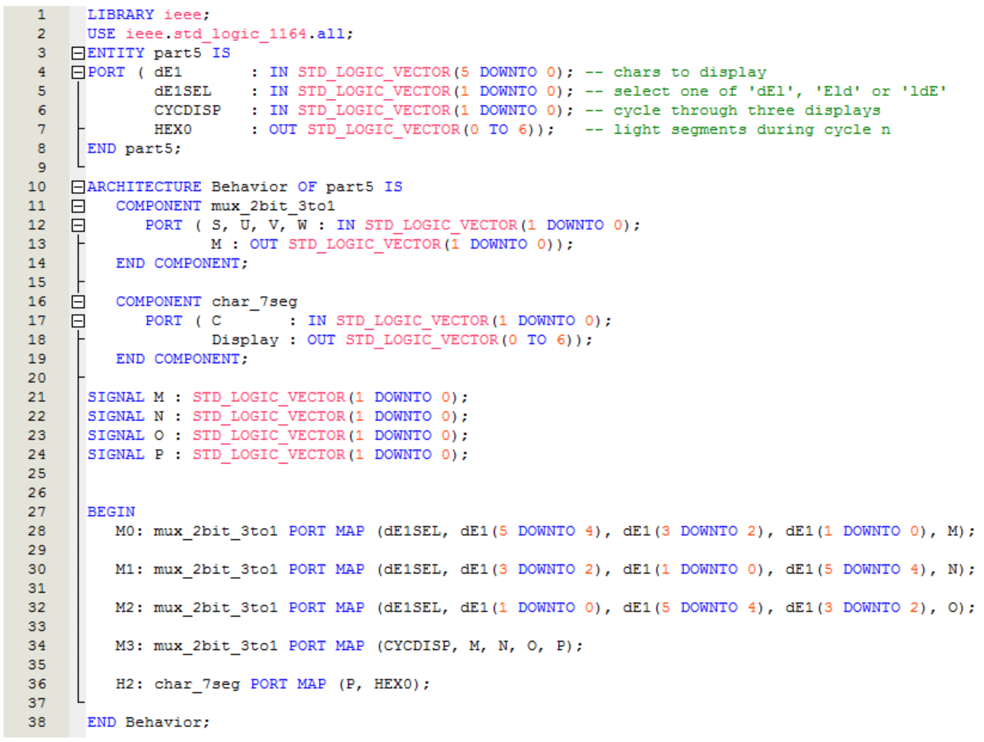
Our 7-segment decoder is changed a little but still recognisable (Figure 46). Change is because we are using a vector instead of two separate signals for the select input (A+B=C), and we are using TO instead of DOWNTO for the segment vector "Display".



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY char_7seg IS
PORT (       C : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
       Display : OUT STD_LOGIC_VECTOR(0 TO 6));
END char_7seg;

ARCHITECTURE Behavior OF char_7seg IS
BEGIN -- Behavior
   -- from our truth table
   -- B=C(0), A=C(1)
   Display(0) <= NOT(C(0)) OR C(1);
   Display(1) <= C(0);
   Display(2) <= C(0);
   Display(3) <= C(1);
   Display(4) <= C(1);
   Display(5) <= NOT(C(0)) OR C(1);
   Display(6) <= C(1);

END Behavior;
```

Figure 46: 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).



```vhdl
 1  LIBRARY ieee;
 2  USE ieee.std_logic_1164.all;
 3  ENTITY part5 IS
 4  PORT ( dE1       : IN STD_LOGIC_VECTOR(5 DOWNTO 0); -- chars to display
 5         dE1SEL    : IN STD_LOGIC_VECTOR(1 DOWNTO 0); -- select one of 'dE1', 'E1d' or '1dE'
 6         CYCDISP   : IN STD_LOGIC_VECTOR(1 DOWNTO 0); -- cycle through three displays
 7         HEX0      : OUT STD_LOGIC_VECTOR(0 TO 6));   -- light segments during cycle n
 8  END part5;
 9
10  ARCHITECTURE Behavior OF part5 IS
11     COMPONENT mux_2bit_3to1
12        PORT ( S, U, V, W : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
13               M : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
14     END COMPONENT;
15
16     COMPONENT char_7seg
17        PORT ( C       : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
18               Display : OUT STD_LOGIC_VECTOR(0 TO 6));
19     END COMPONENT;
20
21  SIGNAL M : STD_LOGIC_VECTOR(1 DOWNTO 0);
22  SIGNAL N : STD_LOGIC_VECTOR(1 DOWNTO 0);
23  SIGNAL O : STD_LOGIC_VECTOR(1 DOWNTO 0);
24  SIGNAL P : STD_LOGIC_VECTOR(1 DOWNTO 0);
25
26
27  BEGIN
28     M0: mux_2bit_3to1 PORT MAP (dE1SEL, dE1(5 DOWNTO 4), dE1(3 DOWNTO 2), dE1(1 DOWNTO 0), M);
29
30     M1: mux_2bit_3to1 PORT MAP (dE1SEL, dE1(3 DOWNTO 2), dE1(1 DOWNTO 0), dE1(5 DOWNTO 4), N);
31
32     M2: mux_2bit_3to1 PORT MAP (dE1SEL, dE1(1 DOWNTO 0), dE1(5 DOWNTO 4), dE1(3 DOWNTO 2), O);
33
34     M3: mux_2bit_3to1 PORT MAP (CYCDISP, M, N, O, P);
35
36     H2: char_7seg PORT MAP (P, HEX0);
37
38  END Behavior;
```

Figure 47: DE1 code modified for our M21EDA board

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

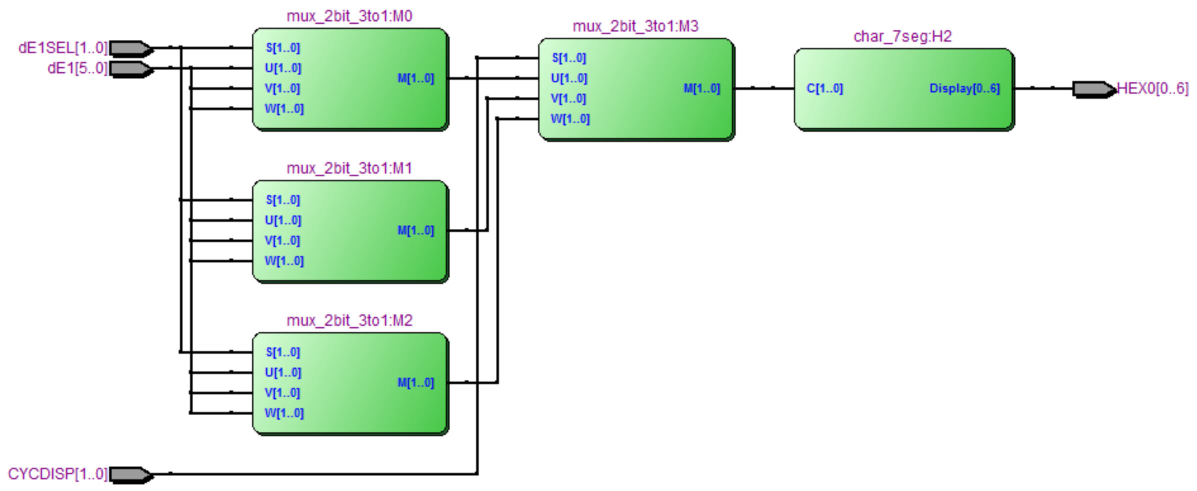


Figure 48: RTL of our intended design

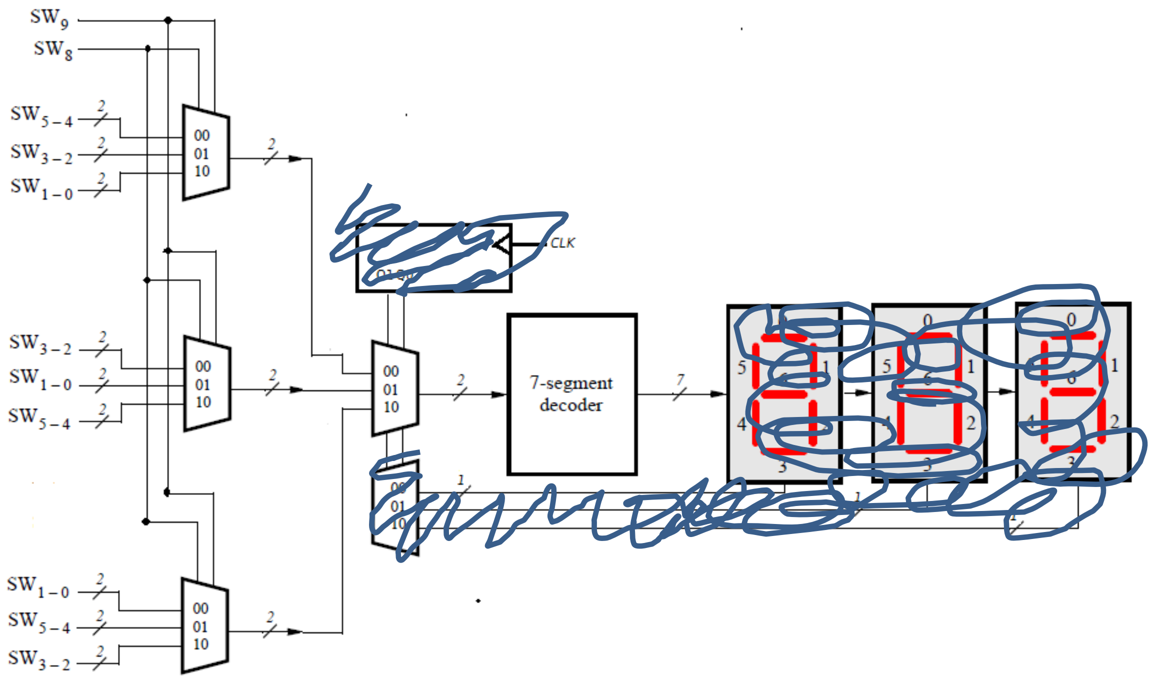
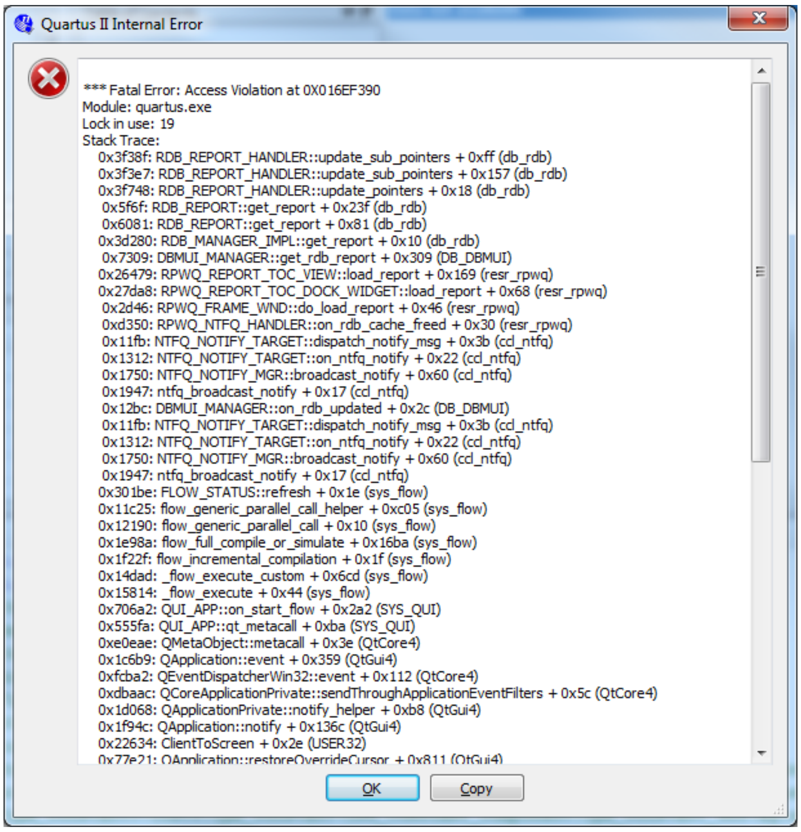


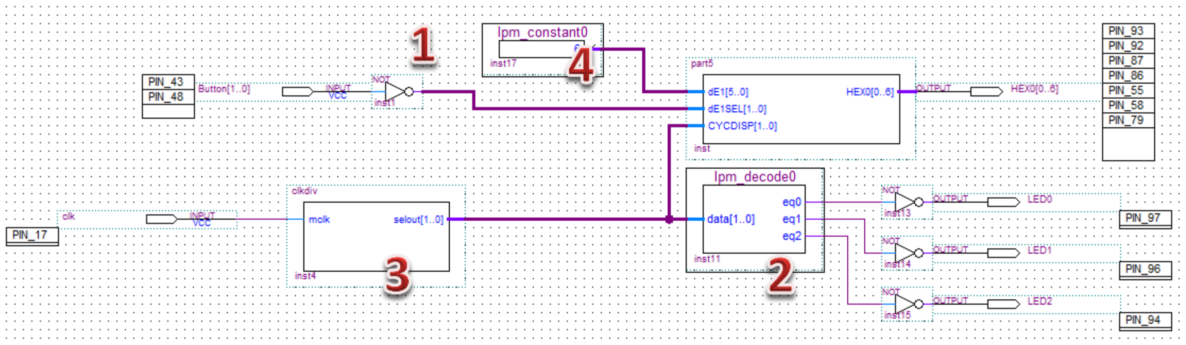
Figure 49: The implementation so far

Go figure, another random kerplunk!



```
*** Fatal Error: Access Violation at 0X016EF390
Module: quartus.exe
Lock in use: 19
Stack Trace:
    0x3f38f: RDB_REPORT_HANDLER::update_sub_pointers + 0xff (db_rdb)
    0x3f3e7: RDB_REPORT_HANDLER::update_sub_pointers + 0x157 (db_rdb)
    0x3f748: RDB_REPORT_HANDLER::update_pointers + 0x18 (db_rdb)
     0x5f6f: RDB_REPORT::get_report + 0x23f (db_rdb)
     0x6081: RDB_REPORT::get_report + 0x81 (db_rdb)
    0x3d280: RDB_MANAGER_IMPL::get_report + 0x10 (db_rdb)
     0x7309: DBMUI_MANAGER::get_rdb_report + 0x309 (DB_DBMUI)
    0x26479: RPWQ_REPORT_TOC_VIEW::load_report + 0x169 (resr_rpwq)
    0x27da8: RPWQ_REPORT_TOC_DOCK_WIDGET::load_report + 0x68 (resr_rpwq)
     0x2d46: RPWQ_FRAME_WND::do_load_report + 0x46 (resr_rpwq)
     0xd350: RPWQ_NTFQ_HANDLER::on_rdb_cache_freed + 0x30 (resr_rpwq)
     0x11fb: NTFQ_NOTIFY_TARGET::dispatch_notify_msg + 0x3b (ccl_ntfq)
     0x1312: NTFQ_NOTIFY_TARGET::on_ntfq_notify + 0x22 (ccl_ntfq)
     0x1750: NTFQ_NOTIFY_MGR::broadcast_notify + 0x60 (ccl_ntfq)
     0x1947: ntfq_broadcast_notify + 0x17 (ccl_ntfq)
     0x12bc: DBMUI_MANAGER::on_rdb_updated + 0x2c (DB_DBMUI)
     0x11fb: NTFQ_NOTIFY_TARGET::dispatch_notify_msg + 0x3b (ccl_ntfq)
     0x1312: NTFQ_NOTIFY_TARGET::on_ntfq_notify + 0x22 (ccl_ntfq)
     0x1750: NTFQ_NOTIFY_MGR::broadcast_notify + 0x60 (ccl_ntfq)
     0x1947: ntfq_broadcast_notify + 0x17 (ccl_ntfq)
    0x301be: FLOW_STATUS::refresh + 0x1e (sys_flow)
    0x11c25: flow_generic_parallel_call_helper + 0xc05 (sys_flow)
    0x12190: flow_generic_parallel_call + 0x10 (sys_flow)
    0x1e98a: flow_full_compile_or_simulate + 0x16ba (sys_flow)
    0x1f22f: flow_incremental_compilation + 0x1f (sys_flow)
    0x14dad: _flow_execute_custom + 0x6cd (sys_flow)
    0x15814: _flow_execute + 0x44 (sys_flow)
    0x706a2: QUI_APP::on_start_flow + 0x2a2 (SYS_QUI)
    0x555fa: QUI_APP::qt_metacall + 0xba (SYS_QUI)
    0xe0eae: QMetaObject::metacall + 0x3e (QtCore4)
    0x1c6b9: QApplication::event + 0x359 (QtGui4)
    0xfcba2: QEventDispatcherWin32::event + 0x112 (QtCore4)
    0xdbaac: QCoreApplicationPrivate::sendThroughApplicationEventFilters + 0x5c (QtCore4)
    0x1d068: QApplicationPrivate::notify_helper + 0xb8 (QtGui4)
    0x1f94c: QApplication::notify + 0x136c (QtGui4)
    0x22634: ClientToScreen + 0x2e (USER32)
    0x77e21: QApplication::restoreOverrideCursor + 0x811 (QtGui4)
```

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.



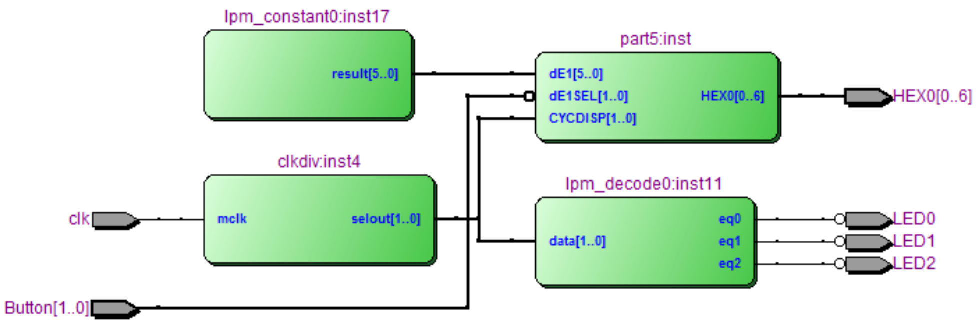


Figure 50: Full Part V solution

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.

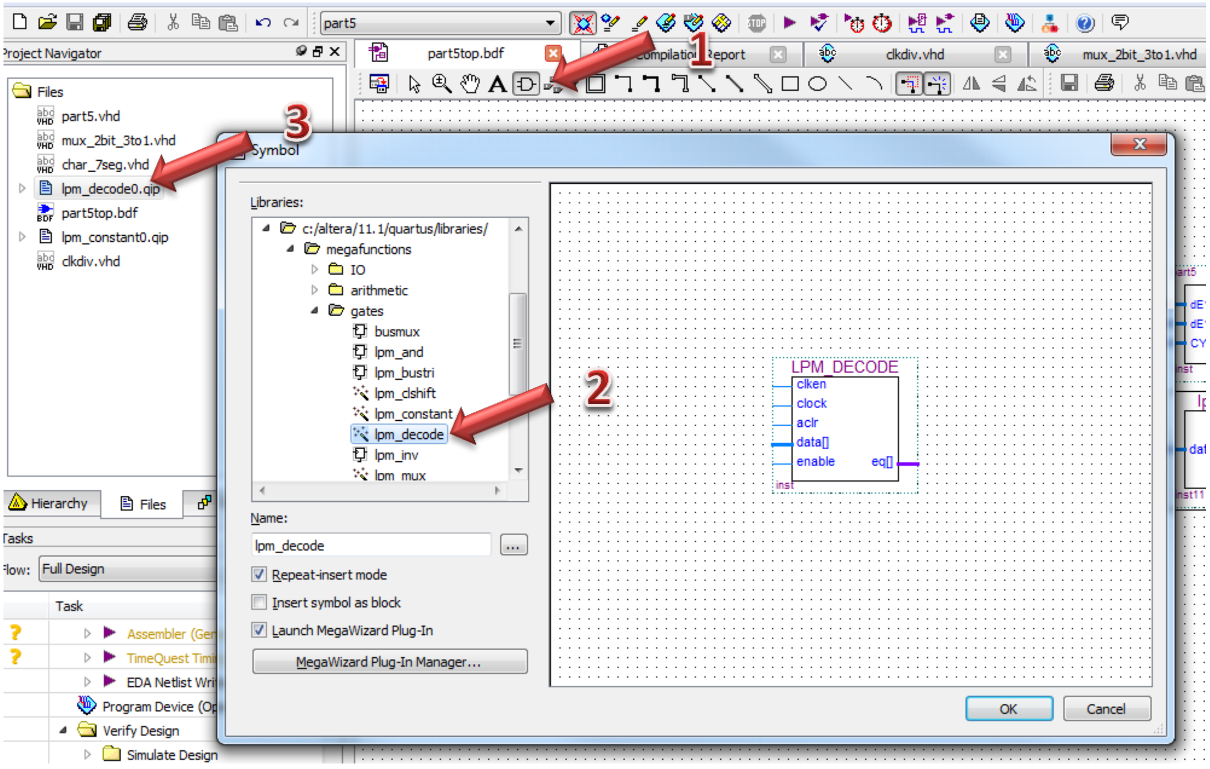
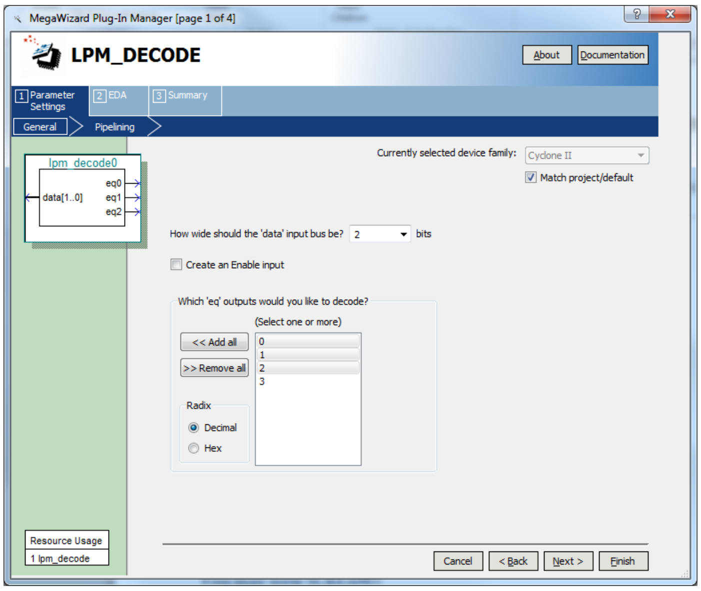


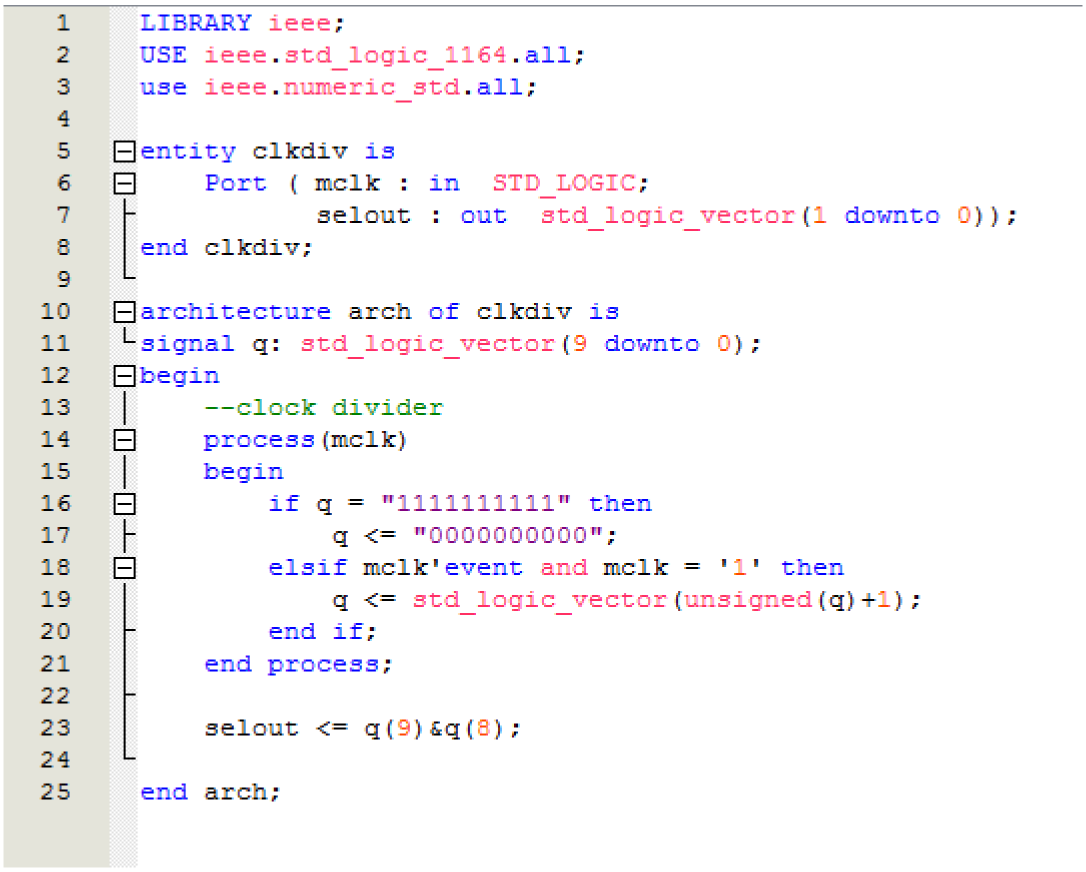
Figure 51: Library wizards



Currently selected device family: Cyclone II
☑ Match project/default
How wide should the 'data' input bus be? 2 bits
☐ Create an Enable input
Which 'eq' outputs would you like to decode?
(Select one or more)
<< Add all
>> Remove all
0
1
2
3
Radix
(●) Decimal
( ) Hex

Figure 52: 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).



```
 1  LIBRARY ieee;
 2  USE ieee.std_logic_1164.all;
 3  use ieee.numeric_std.all;
 4
 5  entity clkdiv is
 6      Port ( mclk : in  STD_LOGIC;
 7             selout : out  std_logic_vector(1 downto 0));
 8  end clkdiv;
 9
10  architecture arch of clkdiv is
11  signal q: std_logic_vector(9 downto 0);
12  begin
13      --clock divider
14      process(mclk)
15      begin
16          if q = "1111111111" then
17              q <= "0000000000";
18          elsif mclk'event and mclk = '1' then
19              q <= std_logic_vector(unsigned(q)+1);
20          end if;
21      end process;
22
23      selout <= q(9)&q(8);
24
25  end arch;
```

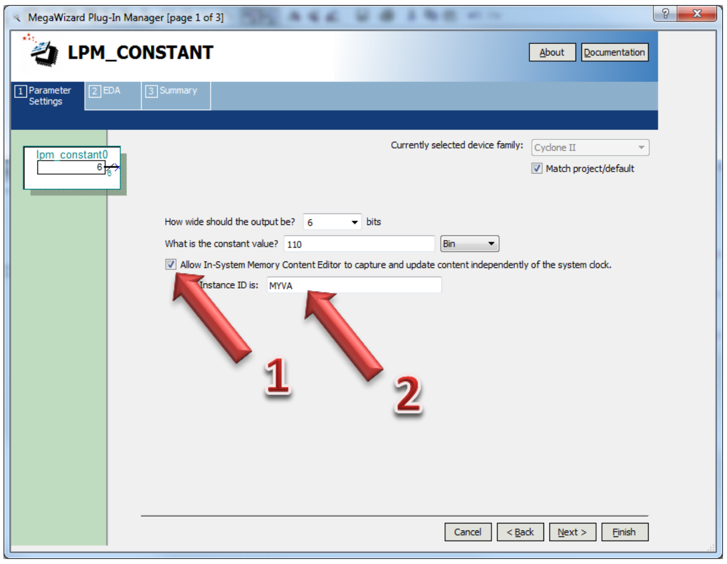
Figure 53: Slow down clk and provide a display sweep

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.

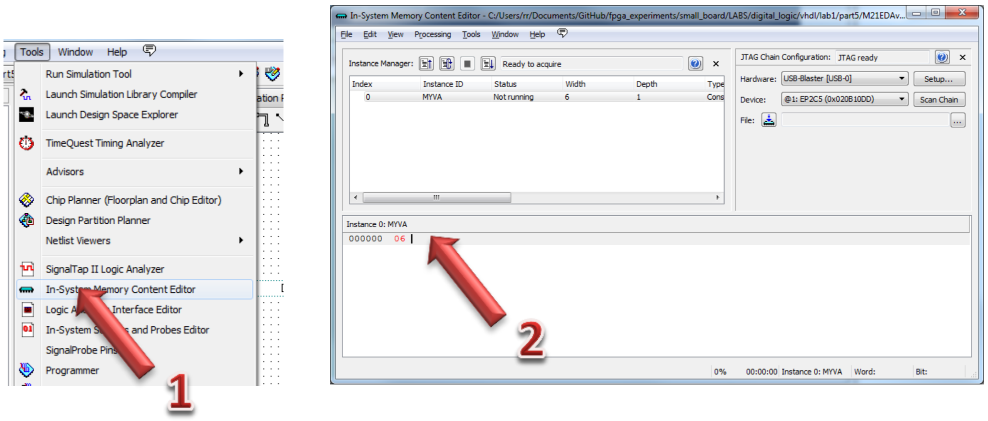
I used a "constant" wizard from the megafunctions (Figure 54) to create the missing DE1 switches (SW[5..0]) that set up the character string for display (Figure 50 item 4). The neat thing is that you can a set a flag (Figure 54 item 1) so a named value (Figure 54 item 2) can be tuneable while running 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).



Currently selected device family: Cyclone II
[✓] Match project/default

How wide should the output be? 6 bits
What is the constant value? 110 Bin
[✓] Allow In-System Memory Content Editor to capture and update content independently of the system clock.
Instance ID is: MYVA

Figure 54: Define SW[5..0] as a constant


Tools
Window
Help
Run Simulation Tool
Launch Simulation Library Compiler
Launch Design Space Explorer
TimeQuest Timing Analyzer
Advisors
Chip Planner (Floorplan and Chip Editor)
Design Partition Planner
Netlist Viewers
SignalTap II Logic Analyzer
In-System Memory Content Editor
Logic Analyzer Interface Editor
In-System Sources and Probes Editor
SignalProbe Pins
Programmer
1
File
Edit
View
Processing
Tools
Window
Help
Setup...
Scan Chain
2

In-System Memory Content Editor - C:/Users/rr/Documents/GitHub/fpga_experiments/small_board/LABS/digital_logic/vhdl/lab1/part5/M21EDAv...

Instance Manager: Ready to acquire

Index	Instance ID	Status	Width	Depth	Type
0	MYVA	Not running	6	1	Cons

JTAG Chain Configuration: JTAG ready
Hardware: USB-Blaster [USB-0]
Device: @1: EP2C5 (0x020B10DD)
File:

Instance 0: MYVA

```
000000  06
```

0% 00:00:00 Instance 0: MYVA Word: Bit:

Figure 55: 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).

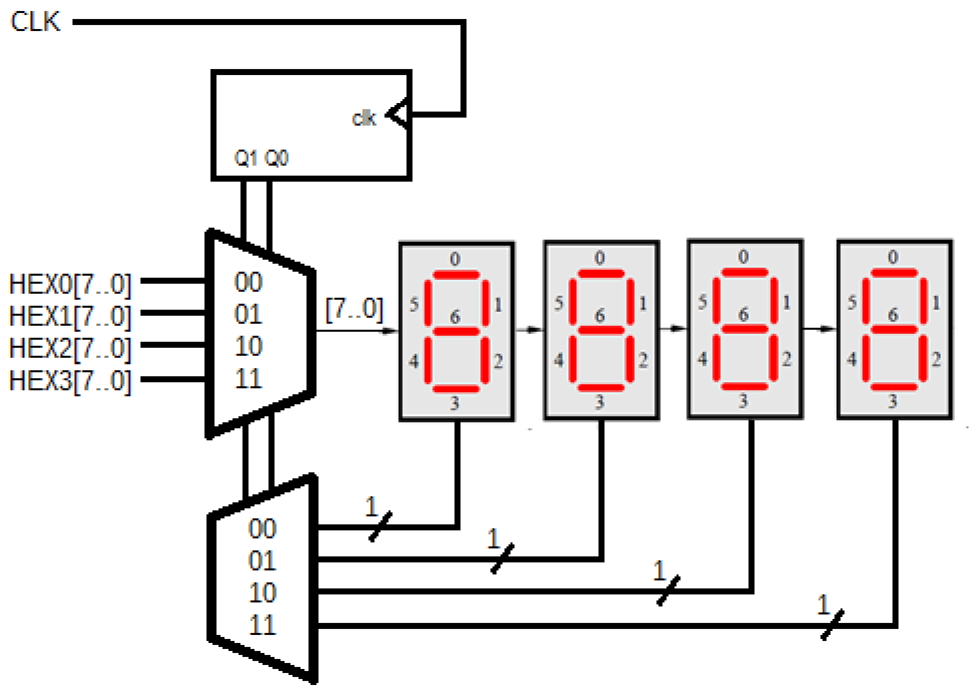


Figure 56: Mimic DE1 displays

Something for Part VI.

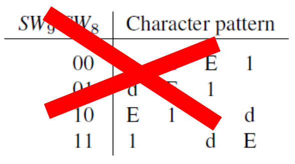
Part VI

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 (

7 (not 8) signals (

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



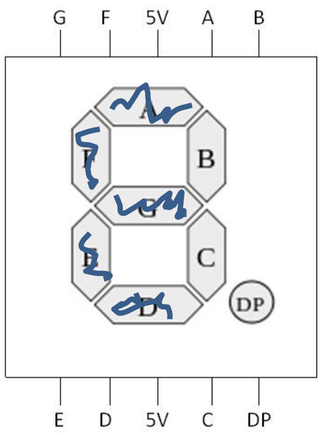
~~$SW_9 SW_8$~~	~~Character pattern~~			
~~00~~		[illegible]	~~E~~	~~1~~
~~01~~	~~d~~	~~E~~	~~1~~	
~~10~~	~~E~~	~~1~~		~~d~~
~~11~~	~~1~~		~~d~~	~~E~~

$SW_9 SW_8$	Character Pattern			
00		E	d	A
01	E	d	A	
10	d	A		E
11	A		E	d

Table 3: Characters for our M21EDA board

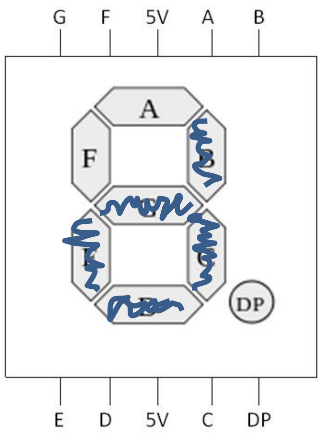
LED_A	0	PIN_93	0
LED_B	1	PIN_92	1
LED_C	2	PIN_87	1
LED_D	3	PIN_86	0
LED_E	4	PIN_55	0
LED_F	5	PIN_58	0
LED_G	6	PIN_79	0

"0000110"



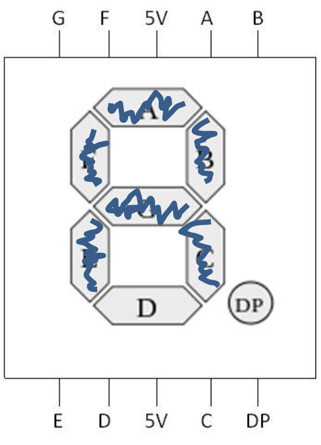
LED_A	0	PIN_93	1
LED_B	1	PIN_92	0
LED_C	2	PIN_87	0
LED_D	3	PIN_86	0
LED_E	4	PIN_55	0
LED_F	5	PIN_58	1
LED_G	6	PIN_79	0

"0100001"

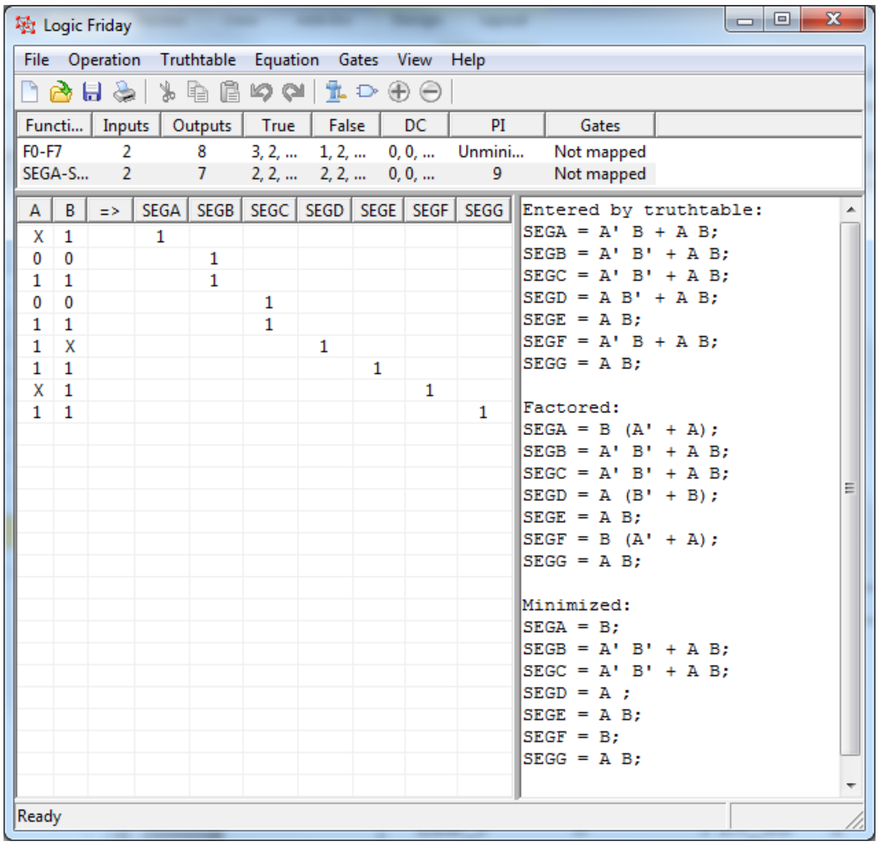


LED_A	0	PIN_93	1
LED_B	1	PIN_92	0
LED_C	2	PIN_87	0
LED_D	3	PIN_86	1
LED_E	4	PIN_55	1
LED_F	5	PIN_58	1
LED_G	6	PIN_79	1
LED_H(DP)	7	PIN_113	1

"0001000"



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



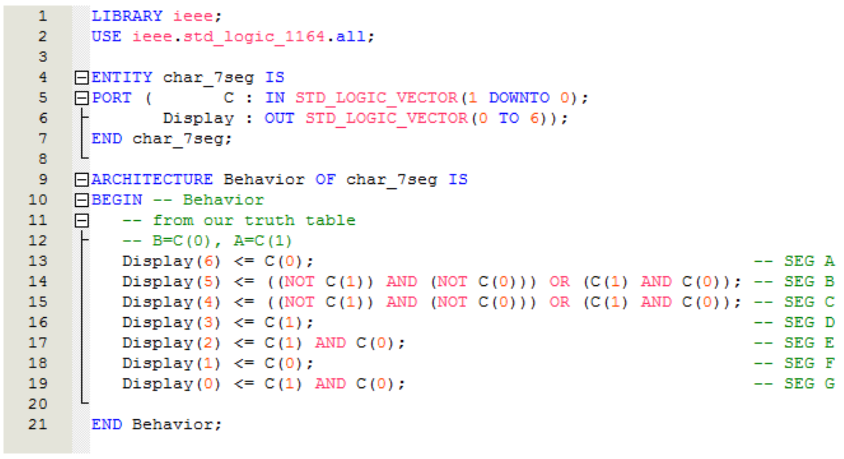
Functi...	Inputs	Outputs	True	False	DC	PI	Gates
F0-F7	2	8	3, 2, ...	1, 2, ...	0, 0, ...	Unmini...	Not mapped
SEGA-S...	2	7	2, 2, ...	2, 2, ...	0, 0, ...	9	Not mapped

A	B	=>	SEGA	SEGB	SEGC	SEGD	SEGE	SEGF	SEGG
X	1		1						
0	0			1					
1	1			1					
0	0				1				
1	1				1				
1	X					1			
1	1						1		
X	1							1	
1	1								1

```
Entered by truthtable:
SEGA = A' B + A B;
SEGB = A' B' + A B;
SEGC = A' B' + A B;
SEGD = A B' + A B;
SEGE = A B;
SEGF = A' B + A B;
SEGG = A B;

Factored:
SEGA = B (A' + A);
SEGB = A' B' + A B;
SEGC = A' B' + A B;
SEGD = A (B' + B);
SEGE = A B;
SEGF = B (A' + A);
SEGG = A B;

Minimized:
SEGA = B;
SEGB = A' B' + A B;
SEGC = A' B' + A B;
SEGD = A ;
SEGE = A B;
SEGF = B;
SEGG = A B;
```

Figure 57: EdA for our M21EDA board

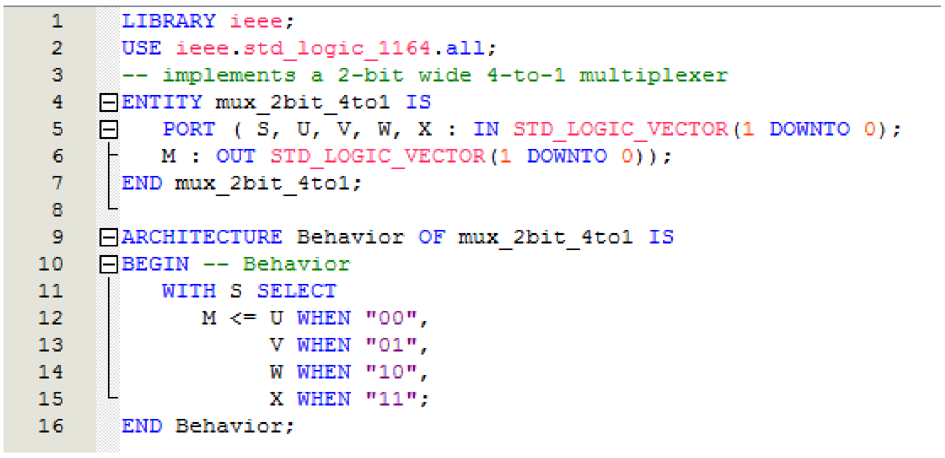
From which the following code falls out (Figure 58).



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY char_7seg IS
PORT (        C : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        Display : OUT STD_LOGIC_VECTOR(0 TO 6));
END char_7seg;

ARCHITECTURE Behavior OF char_7seg IS
BEGIN -- Behavior
    -- from our truth table
    -- B=C(0), A=C(1)
    Display(6) <= C(0);                                         -- SEG A
    Display(5) <= ((NOT C(1)) AND (NOT C(0))) OR (C(1) AND C(0)); -- SEG B
    Display(4) <= ((NOT C(1)) AND (NOT C(0))) OR (C(1) AND C(0)); -- SEG C
    Display(3) <= C(1);                                         -- SEG D
    Display(2) <= C(1) AND C(0);                                -- SEG E
    Display(1) <= C(0);                                         -- SEG F
    Display(0) <= C(1) AND C(0);                                -- SEG G

END Behavior;
```

Figure 58: New char_7seg

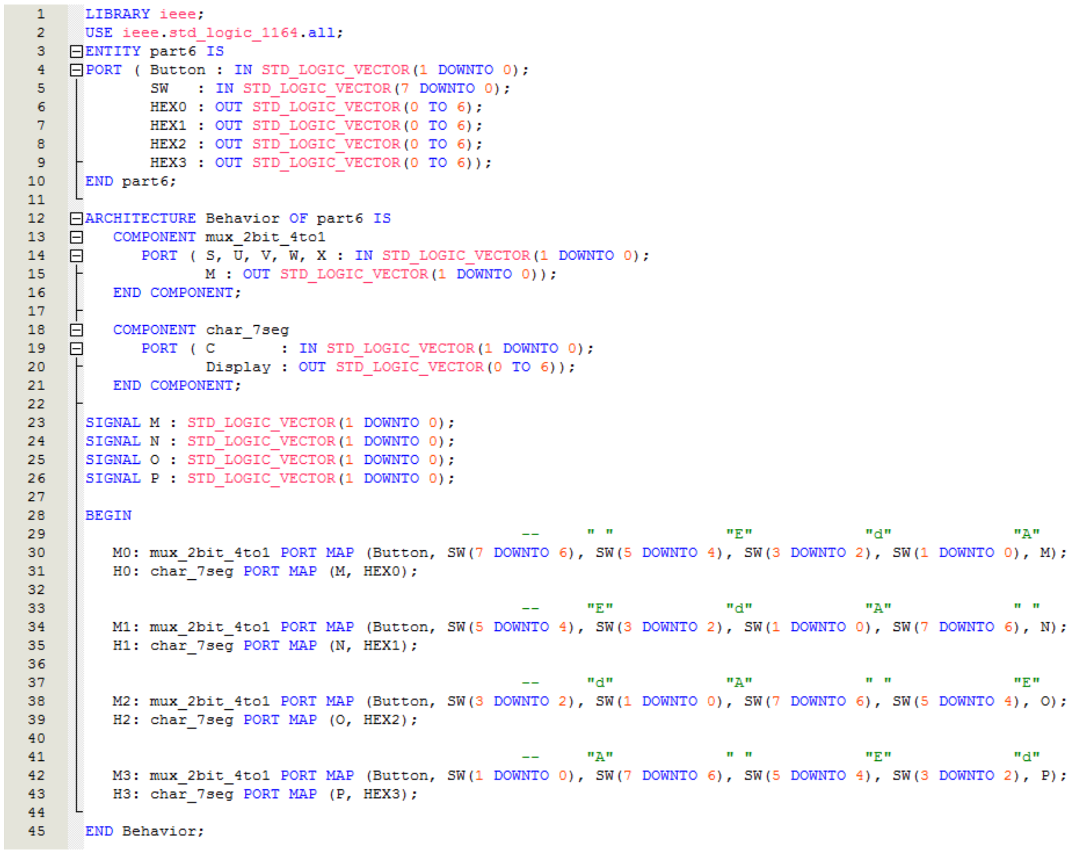
Of course, there is a subtle change to the mux (3-to-1 to 4-to-1) (Figure 59).



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- implements a 2-bit wide 4-to-1 multiplexer
ENTITY mux_2bit_4to1 IS
   PORT ( S, U, V, W, X : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
   M : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
END mux_2bit_4to1;

ARCHITECTURE Behavior OF mux_2bit_4to1 IS
BEGIN -- Behavior
   WITH S SELECT
      M <= U WHEN "00",
           V WHEN "01",
           W WHEN "10",
           X WHEN "11";
END Behavior;
```

Figure 59: Modified mux

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

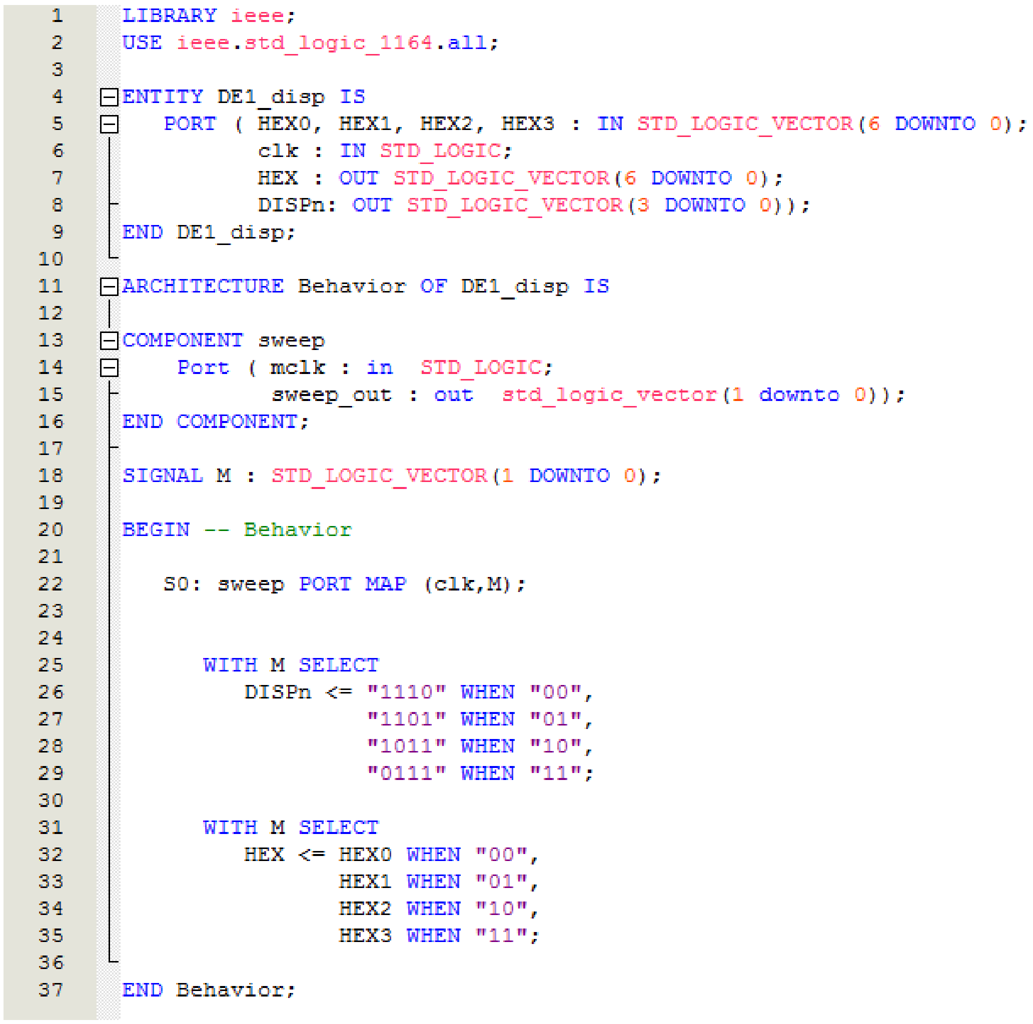


```
 1  LIBRARY ieee;
 2  USE ieee.std_logic_1164.all;
 3  ENTITY part6 IS
 4  PORT ( Button : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
 5         SW     : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
 6         HEX0 : OUT STD_LOGIC_VECTOR(0 TO 6);
 7         HEX1 : OUT STD_LOGIC_VECTOR(0 TO 6);
 8         HEX2 : OUT STD_LOGIC_VECTOR(0 TO 6);
 9         HEX3 : OUT STD_LOGIC_VECTOR(0 TO 6));
10  END part6;
11
12  ARCHITECTURE Behavior OF part6 IS
13     COMPONENT mux_2bit_4to1
14        PORT ( S, U, V, W, X : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
15               M : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
16     END COMPONENT;
17
18     COMPONENT char_7seg
19        PORT ( C       : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
20               Display : OUT STD_LOGIC_VECTOR(0 TO 6));
21     END COMPONENT;
22
23  SIGNAL M : STD_LOGIC_VECTOR(1 DOWNTO 0);
24  SIGNAL N : STD_LOGIC_VECTOR(1 DOWNTO 0);
25  SIGNAL O : STD_LOGIC_VECTOR(1 DOWNTO 0);
26  SIGNAL P : STD_LOGIC_VECTOR(1 DOWNTO 0);
27
28  BEGIN
29                                                 --     " "            "E"            "d"             "A"
30     M0: mux_2bit_4to1 PORT MAP (Button, SW(7 DOWNTO 6), SW(5 DOWNTO 4), SW(3 DOWNTO 2), SW(1 DOWNTO 0), M);
31     H0: char_7seg PORT MAP (M, HEX0);
32
33                                                 --     "E"            "d"            "A"             " "
34     M1: mux_2bit_4to1 PORT MAP (Button, SW(5 DOWNTO 4), SW(3 DOWNTO 2), SW(1 DOWNTO 0), SW(7 DOWNTO 6), N);
35     H1: char_7seg PORT MAP (N, HEX1);
36
37                                                 --     "d"            "A"            " "             "E"
38     M2: mux_2bit_4to1 PORT MAP (Button, SW(3 DOWNTO 2), SW(1 DOWNTO 0), SW(7 DOWNTO 6), SW(5 DOWNTO 4), O);
39     H2: char_7seg PORT MAP (O, HEX2);
40
41                                                 --     "A"            " "            "E"             "d"
42     M3: mux_2bit_4to1 PORT MAP (Button, SW(1 DOWNTO 0), SW(7 DOWNTO 6), SW(5 DOWNTO 4), SW(3 DOWNTO 2), P);
43     H3: char_7seg PORT MAP (P, HEX3);
44
45  END Behavior;
```

Figure 60: 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).

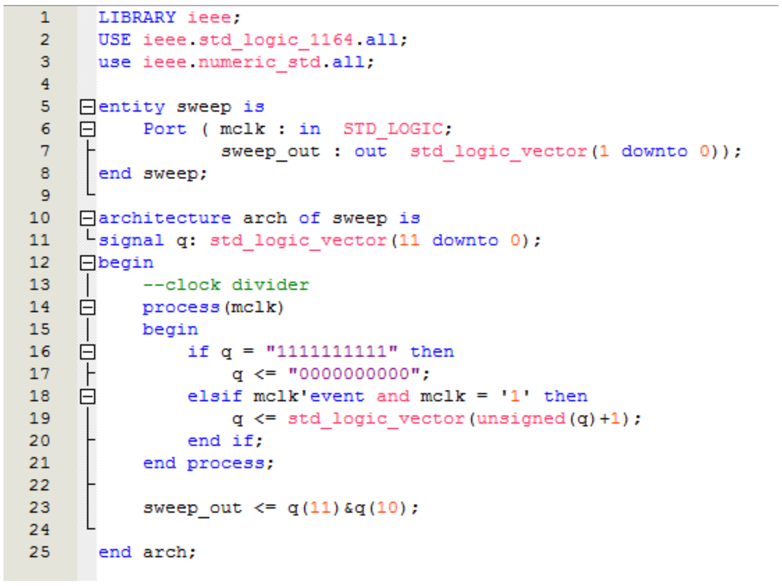
Of course, in Part V we finished with the idea that we might make the M21EDA displays behave more like the DE1 so we can read labs across more readily - which leads to the following code (Figure 61).



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY DE1_disp IS
   PORT ( HEX0, HEX1, HEX2, HEX3 : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
          clk : IN STD_LOGIC;
          HEX : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
          DISPn: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END DE1_disp;

ARCHITECTURE Behavior OF DE1_disp IS

COMPONENT sweep
    Port ( mclk : in  STD_LOGIC;
           sweep_out : out  std_logic_vector(1 downto 0));
END COMPONENT;

SIGNAL M : STD_LOGIC_VECTOR(1 DOWNTO 0);

BEGIN -- Behavior

   S0: sweep PORT MAP (clk,M);


      WITH M SELECT
         DISPn <= "1110" WHEN "00",
                  "1101" WHEN "01",
                  "1011" WHEN "10",
                  "0111" WHEN "11";

      WITH M SELECT
         HEX <= HEX0 WHEN "00",
                HEX1 WHEN "01",
                HEX2 WHEN "10",
                HEX3 WHEN "11";

END Behavior;
```

Figure 61: Impersonate the DE1 displays

The "sweep" component, called by DE1_disp, is just re-casting of the "clkdiv" component from Part V (Figure 62).



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sweep is
    Port ( mclk : in  STD_LOGIC;
           sweep_out : out  std_logic_vector(1 downto 0));
end sweep;

architecture arch of sweep is
signal q: std_logic_vector(11 downto 0);
begin
    --clock divider
    process(mclk)
    begin
        if q = "1111111111" then
            q <= "0000000000";
        elsif mclk'event and mclk = '1' then
            q <= std_logic_vector(unsigned(q)+1);
        end if;
    end process;

    sweep_out <= q(11)&q(10);

end arch;
```

Figure 62: Reused and Renamed

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

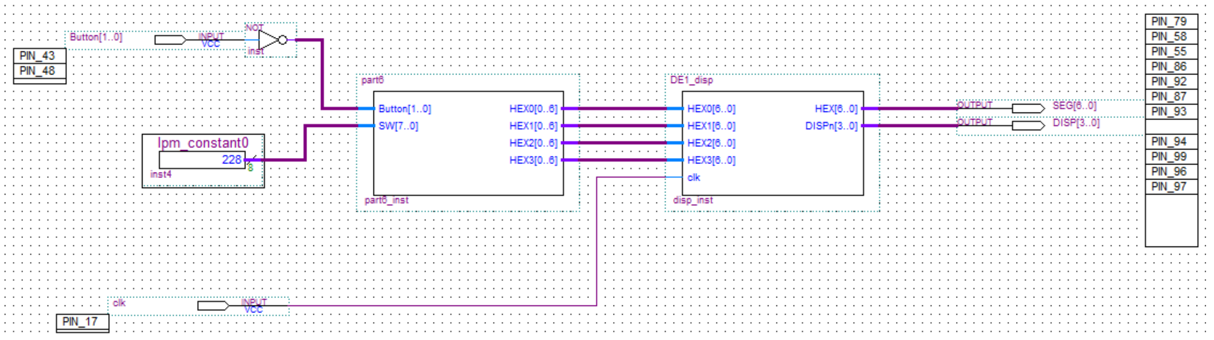


Figure 63: The final assembly

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).

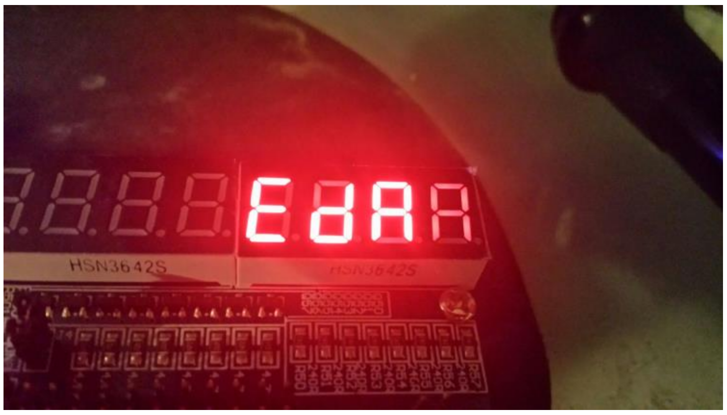


Figure 64: Final result (well almost)

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!!