James Cook University
Electrical and Computer Engineering
EE4306 VHDL Assignment 2008

Task

Design hardware to convert a fixed point integer into a floating point integer number. The input is a 16 bit integer, connected to pins 2 to 9 and 14 to 21 of the Lattice IC used in the ispM4A5 development boards, with pin 2 corresponding to the most significant bit and pin 21 being the least significant bit. The output is to have an 8 bit mantissa and a 4 bit exponent. As an example the 16 bit integer X”A346” is to be converted to the binary number 10100011 x 2^7, where “10100011” = X”A3” is the mantissa, being the most significant part of the integer and 8 corresponding to 2^8 is the exponent. Similarly X”0046” would result in a mantissa of X”46” and an exponent of X“0”. Further examples are shown in the table below:

Integer	Mantissa	Exponent
X”A346”	X”A3”= 1010 0011	8
X”5346”	X”A6”= 1010 0110	7
X”0346”	X”D1”= 1101 0001	2
X”0046”	X”46”= 0100 0110	0
X”0006”	X”06”= 0000 0110	0

The exponent is to be connected to pins 43 to 40, with 43 being the most significant pin. The 4 most significant bits of the mantissa are to be connected to pins 39 to 36, with pin 39 being the most significant digit. The 4 least significant bits of the mantissa are to be displayed as a Hexadecimal number using the seven segment display connected to pins 31 to 24 of the IC.

Testing:

The main testing should be done using waveforms. You may consider using Matlab, C or other ways to generate test vectors for all the possible 65536 input combinations and using the same software to generate the desired output. That will ensure that your circuit operates perfectly.

In addition you will need to test the hardware realisation. The input signals can be generated in two ways. Firstly one can use a test board with jumpers to set the input pins to the required level. Secondly another ispM4A5 development board can be used to operate a counter and generate all the input signals. (A VHDL program is required to do this.) This test board is then connected to the board with the code for the assignment using a ribbon cable. The ribbon cable must be twisted to ensure that Vcc and Gnd line up on both ends of the cable. Failure to do this will result in damage to the boards. Also ensure that the clock on the board with the integer to floating point converter is disabled by not having any jumpers connecting the clock. (Park the jumper on one pin only to prevent its loss.) The output can be tested in a similar way and the assignments will be marked using this technique. Since we only have a limited number of ispM4A5 development boards, students will need to share boards. Make sure that you complete the assignment early, so that you will be able to access sufficient boards to fully test your realisation.

Submission

Submit the complete design before 10am Monday 6 October 2008. The assignment is worth 20 marks being 10% of the total course. Zip the complete directory for the VHDL code and all the resulting ispLever project files and submit this using the digital drop-box submission in LearnJCU, by the due date. In addition submit a copy of the VHDL code in electronic form through Safe-assign in LearnJCU. That will then automatically check for any plagiarism.

In addition submit a hardcopy of the user documents, outlining full details of the principles of operation, results of the simulation of the code, including a written copy of the code and with the attached mark sheet with your name included as the front page, to A/Prof Kikkert by the due date. The hardcopy will be marked for the quality of and the explanations given in the report as well as marks for the code. (The hardcopy will allow me to write comments, for feedback to the student. The electronic files allow me to program the CPLD with your code to verify it's operation.)

To enable me to assess the code, you must specify which compiler (Synplify or Precision) you have used.

Marking Scheme:

5 marks will be given for documentation outlining the operation of the circuit.

1 mark will be given for the comments contained in the code

7 marks will be given for the VHDL code, it's organisation and function.

3 marks will be given for simulation of the code. (Note you are encouraged to test your code using all the 65536 input combinations. However your submission should have less than 1024 test vectors, to ensure that the assignment can be marked in a reasonable time.). No simulation or more than 1024 test vectors, no 3 marks.

4 marks will be given for functionality of the circuit when fitted into the CPLD. To evaluate this I will check that the JED file is produced by your code and I will program it into the CPLD and check the operation of the sinewave generator. To get this mark you must have a JED and all the ispLever project files as part of your electronic submission and those files must contain compiler ticks, indicating that you have properly compiled the code without errors. Ensure that your submission contains your JED file. No JED file no 4 marks.

Minus 3 marks will given if the Documentation and VHDL code are not clearly labelled with your name and student number. Also please ensure that your submission can easily be identified. Do not submit the file as Assignment. Zip, rather use YourName.zip. This ensures that no submissions are overwritten during the downloading. Ensure that the zipped directory containing your code is named appropriately to be able to identify you, do not name the directory assignment, as I will then end up with 30 directories called assignment during marking, making it very difficult to correctly identify your work.

It is important that each student does this work themselves, to ensure that they learn how to write VHDL code. Safe-assign will automatically check for any plagiarism and will flag to the lecturer any assignments that are similar.

C. J. Kikkert

8 August 2008, V3 30 Aug.

VHDL Assignment Marksheet

Name

Student Number

	Max	Award	Comment
Documentation	5		
Comments	1		
VHDL Code	7		
Simulation	3		
Functionality	4		
Penalty	-3		
Total			

Solution

Introduction

The conversion process outlined can be used as an instantaneous compander for a 16 bit ADC circuit. It can also be used as a logarithmic type converter for DSP applications.

It is easiest to do this task in 4 steps:

1. Develop and fully test the conversion module as a component. To fully test this a full 16 bit test vector, (65536 values) is required. This can most easily be generated using an Excel spreadsheet. Test and verify this operation.
2. Add a seven segment display component module and write a simple main module containing a counter and linking the conversion and display components, to count through all the input values and display the resulting Exponent and Mantissa on the seven segment displays. This will allow the conversion and display components to be verified. Test and verify this operation.
3. Modify the main program to provide the Exponent and Mantissa on the output pins 24 to 43 as required. Write a simple “Test Display” vhdl program to take the output values from the conversion board and display them as required on a second board, connected to the first board via a ribbon cable. The “Test Display” board should now display the same counting sequence as steps 2. Test and verify this operation.
4. Modify the main program to use the input pins 2 to 21 to obtain the 16 bit input vector for the fixed point integer into a floating point integer number converter as required in the specifications. Write a simple vhdl counter program to fit on a third board and count through all the input values. The “Test Display” board should now display the same counting sequence as steps 2 and 3.

Step 1

The VHDL code for the fixed point integer into a floating point integer number converter is as follows:

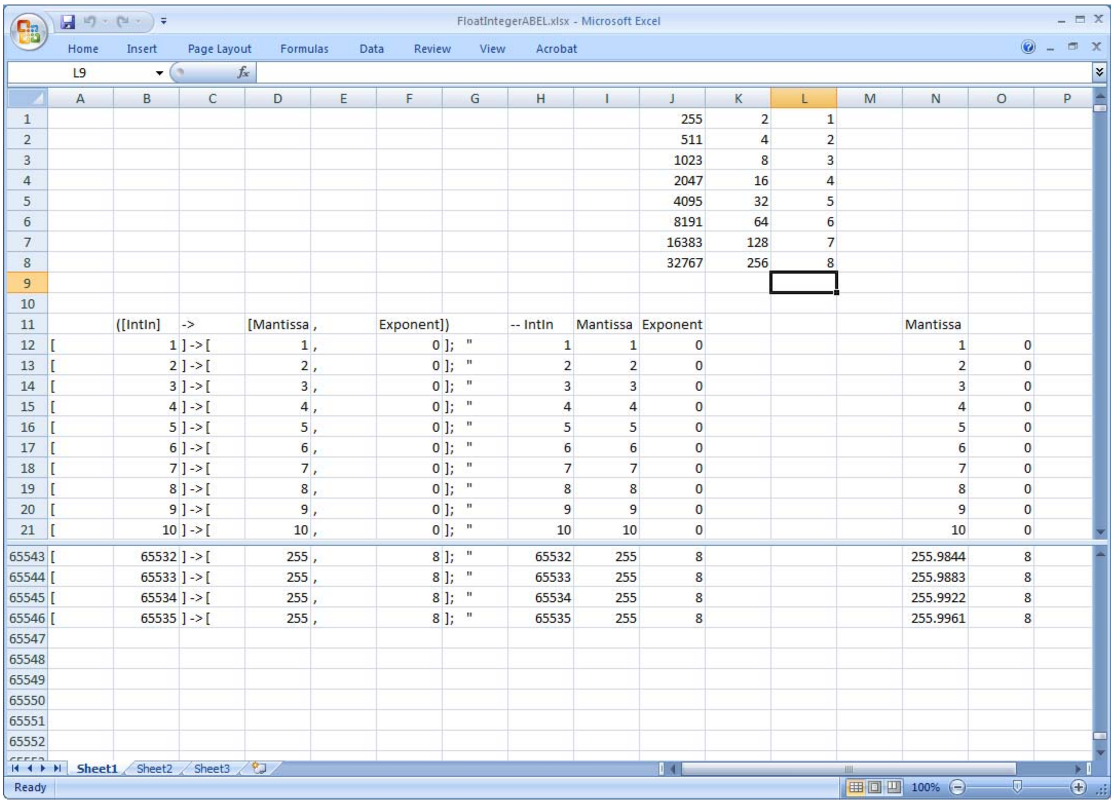
```
--floating point conversion component module
-- for the 2008 VHDL assignment for EE4306 and EE3300
--author C. J. Kikkert, 1 September 2008
--note having this as a component allows for easy progressive testing
--This module is the main program here and is fully tested using ABEL test vectors
--this same module is used as a component in subsequent code

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity Convert is
port(
	IntIn: in std_logic_vector ( 15 downto 0 );		-- 16 bit interger input
	Mantissa: out std_logic_vector ( 7 downto 0 );		--mantissa put to
	Exponent: out std_logic_vector ( 3 downto 0 ) );
end;
```

```
architecture Convert_A of Convert is
begin
Convert_P: Process(IntIn)
        begin
                if (IntIn(15 downto 8) = "00000000") then
--8 MSB's all 0, IntIn 8 LSB's = Mantissa
                        Mantissa <= IntIn(7 downto 0);
                        Exponent <= X"0";
                elsif (IntIn(15 downto 9) = "0000000") then
--7 MSB's all 0, 8th a 1, IntIn slides along one for Mantissa
                        Mantissa <= IntIn(8 downto 1);
                        Exponent <= X"1";
                elsif (IntIn(15 downto 10) = "000000") then
--6 MSB's all 0, 7th a 1, IntIn slides along one for Mantissa
                        Mantissa <= IntIn(9 downto 2);
                        Exponent <= X"2";
                elsif (IntIn(15 downto 11) = "00000") then
--5 MSB's all 0, 6th a 1, IntIn slides along one for Mantissa
                        Mantissa <= IntIn(10 downto 3);
                        Exponent <= X"3";
                elsif (IntIn(15 downto 12) = "0000") then
--4 MSB's all 0, 5th a 1, IntIn slides along one for Mantissa
                        Mantissa <= IntIn(11 downto 4);
                        Exponent <= X"4";
                elsif (IntIn(15 downto 13) = "000") then
--3 MSB's all 0, 4th a 1, IntIn slides along one for Mantissa
                        Mantissa <= IntIn(12 downto 5);
                        Exponent <= X"5";
                elsif (IntIn(15 downto 14) = "00") then
--2 MSB's all 0,3rd a 1, IntIn slides along one for Mantissa
                        Mantissa <= IntIn(13 downto 6);
                        Exponent <= X"6";
                elsif (IntIn(15) = '0') then
--1 MSB's all 0, 2nd a 1, IntIn slides along one for Mantissa
                        Mantissa <= IntIn(14 downto 7);
                        Exponent <= X"7";
                else
        -- MSB's a 1, IntIn 8 MSB's = Mantissa
                        Mantissa <= IntIn(15 downto 8);
                        Exponent <= X"8";
                end if;
        end process Convert_P;
end Convert_A;
```

The principles of operation are quite simple. The position of the most significant 1 in the IntIn integer input signal is determined and the Mantissa is then the 8 bits of the IntIn input starting with the most significant 1 as the most significant bit of the Mantissa. The exponent is the number of bits of least significant bits of the IntIn input that are not mapped in the mantissa.

This vhdl code is run as a vhdl module, but since we only test this module for the code, a .jed file is not required and only the waveforms are used to verify it’s function.

For the test vector file, an Excel spreadsheet is generated containing all the 65536 test vectors. The relevant part of the Excel spreadsheet is shown in figure 1. Alternately Matlab. C++ or Java programs can be used to generate this test vector data.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1										255	2	1				
2										511	4	2				
3										1023	8	3				
4										2047	16	4				
5										4095	32	5				
6										8191	64	6				
7										16383	128	7				
8										32767	256	8				
9																
10																
11		([IntIn]	->	[Mantissa ,		Exponent])		-- IntIn	Mantissa	Exponent				Mantissa		
12	[	1	] -> [	1 ,		0];	"	1	1	0				1	0	
13	[	2	] -> [	2 ,		0];	"	2	2	0				2	0	
14	[	3	] -> [	3 ,		0];	"	3	3	0				3	0	
15	[	4	] -> [	4 ,		0];	"	4	4	0				4	0	
16	[	5	] -> [	5 ,		0];	"	5	5	0				5	0	
17	[	6	] -> [	6 ,		0];	"	6	6	0				6	0	
18	[	7	] -> [	7 ,		0];	"	7	7	0				7	0	
19	[	8	] -> [	8 ,		0];	"	8	8	0				8	0	
20	[	9	] -> [	9 ,		0];	"	9	9	0				9	0	
21	[	10	] -> [	10 ,		0];	"	10	10	0				10	0	
65543	[	65532	] -> [	255 ,		8];	"	65532	255	8				255.9844	8	
65544	[	65533	] -> [	255 ,		8];	"	65533	255	8				255.9883	8	
65545	[	65534	] -> [	255 ,		8];	"	65534	255	8				255.9922	8	
65546	[	65535	] -> [	255 ,		8];	"	65535	255	8				255.9961	8	
65547																
65548																
65549																
65550																
65551																
65552																

Figure 1. Excel spreadsheet for generating ABEL test vectors

The ABEL test vector file can then be produced by having ispLever generate the ABEL test vector file and replacing the Exponent = [Exponent_3_,Exponent_2_,Exponent_1_,Exponent_0_];

```
MODULE FloatCon_E
" TOOL:     vhdl2tf
" DATE:     08/13/08  06:41:53
" TITLE:    Lattice Semiconductor Corporation
" MODULE:   FloatCon_E
" DESIGN:   FloatCon_E
" FILENAME: FloatCon_E.abt
" PROJECT:  floatcon
" VERSION:  1.0
" NOTE: DO NOT EDIT THIS FILE DIRECTLY.
" This file is auto generated by ispLEVER System.
" It can be removed or overwritten automatically.
" If you want to edit this file, copy it to another file or
" rename it with different file extension first.
" Inputs
    IntIn_15_ pin;
    IntIn_14_ pin;
    IntIn_13_ pin;
    IntIn_12_ pin;
    IntIn_11_ pin;
    IntIn_10_ pin;
    IntIn_9_ pin;
    IntIn_8_ pin;
    IntIn_7_ pin;
    IntIn_6_ pin;
    IntIn_5_ pin;
    IntIn_4_ pin;
    IntIn_3_ pin;
    IntIn_2_ pin;
    IntIn_1_ pin;
    IntIn_0_ pin;

" Outputs
```

```
  Mantissa_7_ pin;
  Mantissa_6_ pin;
  Mantissa_5_ pin;
  Mantissa_4_ pin;
  Mantissa_3_ pin;
  Mantissa_2_ pin;
  Mantissa_1_ pin;
  Mantissa_0_ pin;
  Exponent_3_ pin;
  Exponent_2_ pin;
  Exponent_1_ pin;
  Exponent_0_ pin;


" Bidirs

IntIn
=[IntIn_15_,IntIn_14_,IntIn_13_,IntIn_12_,IntIn_11_,IntIn_10_,IntIn_9_,IntIn_8_,IntIn_7_,IntIn_6_,IntIn
_5_,IntIn_4_,IntIn_3_,IntIn_2_,IntIn_1_,IntIn_0_ ];
Mantissa                                                                                          =
[Mantissa_7_,Mantissa_6_,Mantissa_5_,Mantissa_4_,Mantissa_3_,Mantissa_2_,Mantissa_1_,Mantissa_0_];
Exponent = [Exponent_3_,Exponent_2_,Exponent_1_,Exponent_0_];

Test_vectors   ([IntIn] ->    [Mantissa    ,    Exponent])    " IntIn    Mantissa  Exponent

[     1     ] -> [  1     ,    0     ];   "  1     1     0
[     2     ] -> [  2     ,    0     ];   "  2     2     0
[     3     ] -> [  3     ,    0     ];   "  3     3     0
[     4     ] -> [  4     ,    0     ];   "  4     4     0
[     5     ] -> [  5     ,    0     ];   "  5     5     0
[     6     ] -> [  6     ,    0     ];   "  6     6     0
[     7     ] -> [  7     ,    0     ];   "  7     7     0
[     8     ] -> [  8     ,    0     ];   "  8     8     0
[     9     ] -> [  9     ,    0     ];   "  9     9     0
[     10    ] -> [  10    ,    0     ];   "  10    10    0
[     11    ] -> [  11    ,    0     ];   "  11    11    0
[     12    ] -> [  12    ,    0     ];   "  12    12    0
[     13    ] -> [  13    ,    0     ];   "  13    13    0
[     14    ] -> [  14    ,    0     ];   "  14    14    0
[     15    ] -> [  15    ,    0     ];   "  15    15    0
[     16    ] -> [  16    ,    0     ];   "  16    16    0
[     17    ] -> [  17    ,    0     ];   "  17    17    0
[     18    ] -> [  18    ,    0     ];   "  18    18    0
[     19    ] -> [  19    ,    0     ];   "  19    19    0
[     20    ] -> [  20    ,    0     ];   "  20    20    0
[     21    ] -> [  21    ,    0     ];   "  21    21    0
[     22    ] -> [  22    ,    0     ];   "  22    22    0
[     23    ] -> [  23    ,    0     ];   "  23    23    0
[     24    ] -> [  24    ,    0     ];   "  24    24    0
[     25    ] -> [  25    ,    0     ];   "  25    25    0
[     26    ] -> [  26    ,    0     ];   "  26    26    0
[     27    ] -> [  27    ,    0     ];   "  27    27    0
[     28    ] -> [  28    ,    0     ];   "  28    28    0
[     29    ] -> [  29    ,    0     ];   "  29    29    0
[     30    ] -> [  30    ,    0     ];   "  30    30    0
[     31    ] -> [  31    ,    0     ];   "  31    31    0
[     32    ] -> [  32    ,    0     ];   "  32    32    0
[     33    ] -> [  33    ,    0     ];   "  33    33    0
[     34    ] -> [  34    ,    0     ];   "  34    34    0
[     35    ] -> [  35    ,    0     ];   "  35    35    0
[     36    ] -> [  36    ,    0     ];   "  36    36    0
[     37    ] -> [  37    ,    0     ];   "  37    37    0
[     38    ] -> [  38    ,    0     ];   "  38    38    0
[     39    ] -> [  39    ,    0     ];   "  39    39    0
[     40    ] -> [  40    ,    0     ];   "  40    40    0
[     41    ] -> [  41    ,    0     ];   "  41    41    0
[     42    ] -> [  42    ,    0     ];   "  42    42    0
[     43    ] -> [  43    ,    0     ];   "  43    43    0
[     44    ] -> [  44    ,    0     ];   "  44    44    0
[     45    ] -> [  45    ,    0     ];   "  45    45    0
[     46    ] -> [  46    ,    0     ];   "  46    46    0
[     47    ] -> [  47    ,    0     ];   "  47    47    0
```

--Many lines have been removed here for the sake of brevity

```
[       65517  ] -> [   255     ,       8       ];   "   65517  255     8
[       65518  ] -> [   255     ,       8       ];   "   65518  255     8
[       65519  ] -> [   255     ,       8       ];   "   65519  255     8
[       65520  ] -> [   255     ,       8       ];   "   65520  255     8
[       65521  ] -> [   255     ,       8       ];   "   65521  255     8
[       65522  ] -> [   255     ,       8       ];   "   65522  255     8
[       65523  ] -> [   255     ,       8       ];   "   65523  255     8
[       65524  ] -> [   255     ,       8       ];   "   65524  255     8
[       65525  ] -> [   255     ,       8       ];   "   65525  255     8
[       65526  ] -> [   255     ,       8       ];   "   65526  255     8
[       65527  ] -> [   255     ,       8       ];   "   65527  255     8
[       65528  ] -> [   255     ,       8       ];   "   65528  255     8
[       65529  ] -> [   255     ,       8       ];   "   65529  255     8
[       65530  ] -> [   255     ,       8       ];   "   65530  255     8
[       65531  ] -> [   255     ,       8       ];   "   65531  255     8
[       65532  ] -> [   255     ,       8       ];   "   65532  255     8
[       65533  ] -> [   255     ,       8       ];   "   65533  255     8
[       65534  ] -> [   255     ,       8       ];   "   65534  255     8
[       65535  ] -> [   255     ,       8       ];   "   65535  255     8
End
```

Once this has been done, then the ABEL test vectors can be used to check the accuracy of the conversion module. Running the code and checking the waveforms gives the following result:

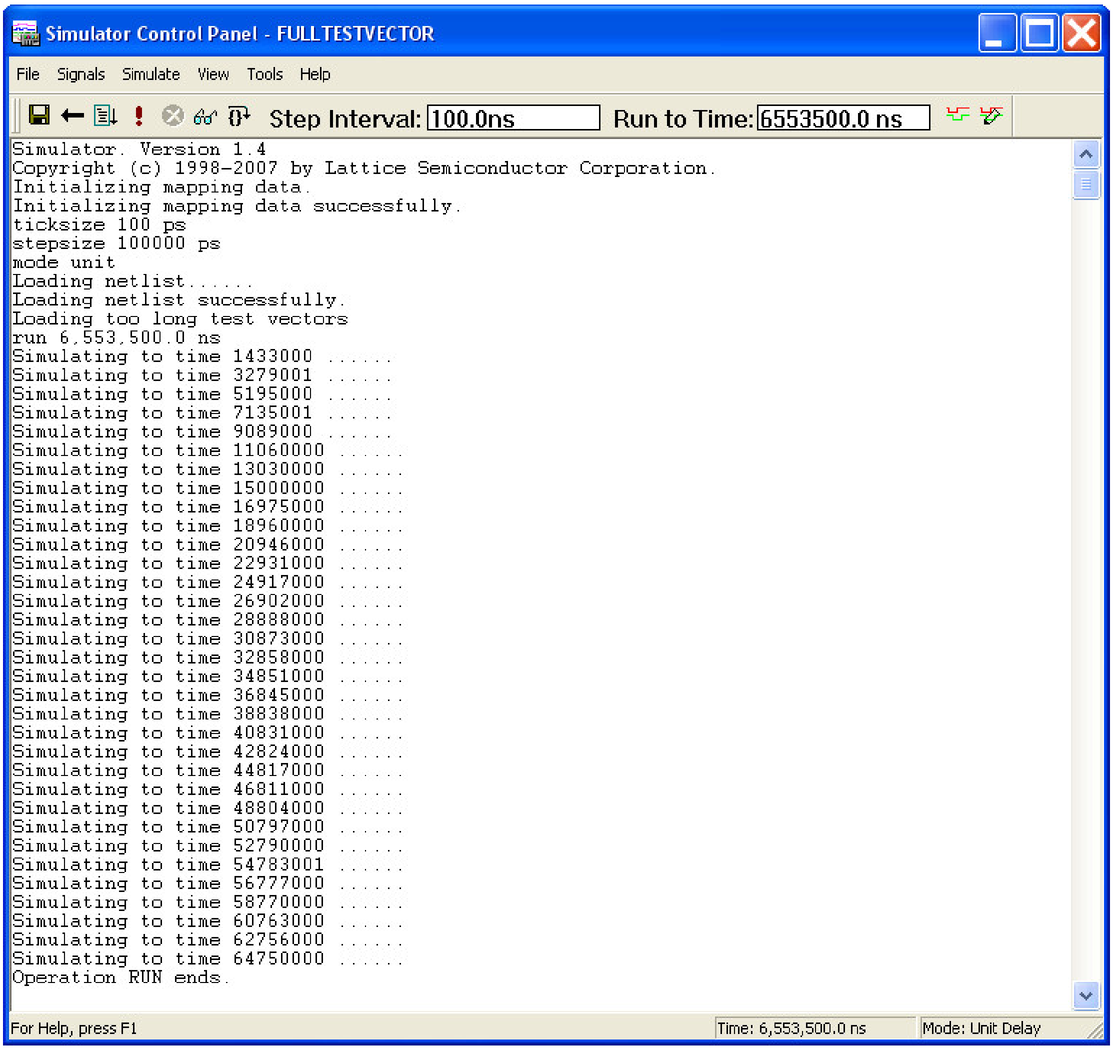


Figure 2: Simulation Control Panel, showing no errors.

In addition the waveforms can be observed, as shown in figures 3 and 4. It can be seen that a uniform count of the input results in a change of frequency of the lower bits of the exponent, as the exponent only changes every doubling of the input data.

Figure 3: Full waveform of the ABEL test vectors

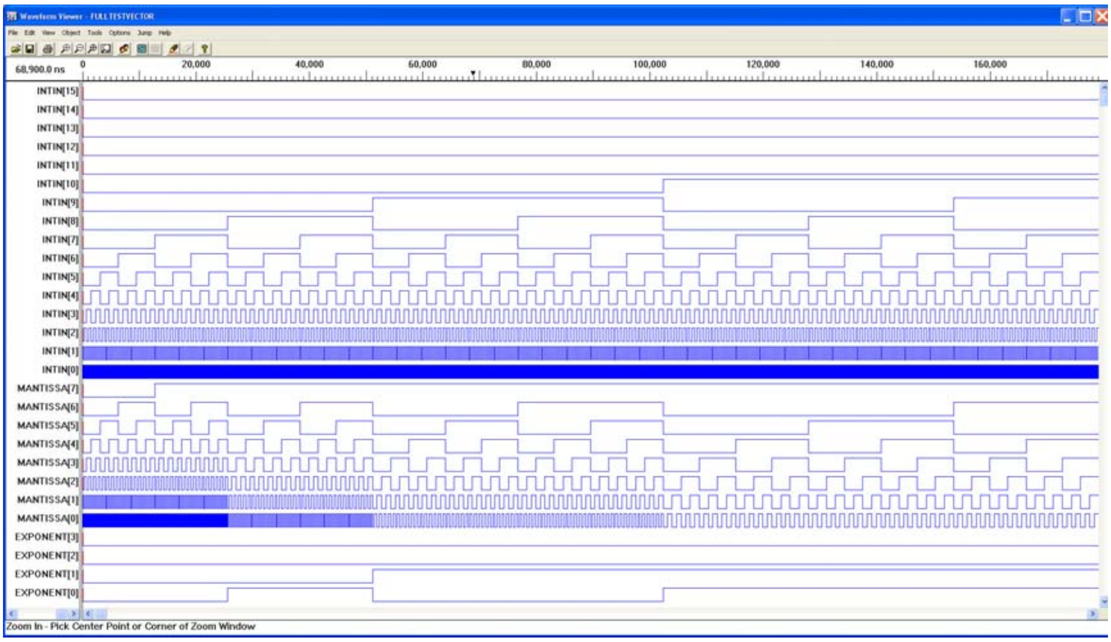


Figure 3: Zoomed in waveform of the ABEL test vectors

Step 2

For the second step, the conversion module which is simply a combinational logic process is connected to a counter and a display, all contained a single Lattice Board. The counter counts through all the input steps and the display shows the exponent and mantissa. The following main program, uses the conversion module described above and the seven segment display module described in the lecture notes.

```
--Main Program that uses the conversion and display components on one board
--this then allows the code to be debugged before rearranging it such that
--only the conversion code is on the main board as required for the assignment
--Author C. J. Kikkert, October 2008.
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity FloatCon is
port(Clk : in std_logic ;          --clock for counter
                    SevSeg2: out std_logic_vector (6 downto 0);       -- Seven segment display Digit 3
                    SevSeg1: out std_logic_vector (6 downto 0);       -- Seven segment display Digit 2
                    SevSeg0: out std_logic_vector (6 downto 0)        -- Seven segment display LS Digit
          );        --end of port
          attribute loc : string;
          attribute loc of Clk:signal is "P11" ;
--        attribute loc of SevSeg3:signal is "P2, P3, P4, P5, P6, P7, P8" ;
          --pin allocation of MS display
          attribute loc of SevSeg2:signal is "P14, P15, P16, P17, P18, P19, P20" ;
          attribute loc of SevSeg1:signal is "P24, P25, P26, P27, P28, P29, P30" ;
          attribute loc of SevSeg0:signal is "P36, P37, P38, P39, P40, P41, P42" ;--pin allocation of LS
display
end FloatCon;

architecture FloatCon_A of FloatCon is
component Convert
          port(Intin: in std_logic_vector ;                    -- 16 bit interger input
          Mantissa: out std_logic_vector ;                     --mantissa put to
          Exponent: out std_logic_vector );
end component;

component SevenSeg
          port (Hexin: in std_logic_vector;
          SevSegOut : out std_logic_vector);
end component;

Signal IntIn: std_logic_vector ( 15 downto 0 );                -- 16 bit interger input
Signal Mantissa: std_logic_vector(7 downto 0);
Signal Exponent: std_logic_vector ( 3 downto 0 );

begin
          L1: Convert port map (IntIn, Mantissa, Exponent);
--        D0: SevenSeg port map (Count(3 downto 0), SevSeg0);      -- Seven Segment Display MS Digit
          D1: SevenSeg port map (Exponent, SevSeg2);      -- Seven Segment Display Digit 3
          D2: SevenSeg port map (Mantissa(7 downto 4), SevSeg1);   -- Seven Segment Display Digit 2
          D3: SevenSeg port map (Mantissa(3 downto 0), SevSeg0);   -- Seven Segment Display LS Digit

          P_Clock: process
          begin
          wait until rising_edge(Clk);
                              IntIn <= IntIn + X"0001";     -- Incement the Integer value for the
converter input
          end process;
end FloatCon_A;
```

The output from this program is then observed as the input is incremented with the counter.

Step 3

The third step involves sending the required output data to another board for display. The main program still produces the input data with a counter. This step allows the testing of the Exponent and Mantissa Display. Since the seven segment display on the main board is connected directly to the pins, when a ribbon cable is used to connect the main board output to pins 24 to 43 of the display board, the least significant bit of the mantissa will then be shown on the third from the left display element on both the main and display board.

The main program below already has the final code to connect the input to the pins. In the code below, that is commented out.

```
--Main Program that has a counter and does the conversion and the MS hex bit of the mantissa
--and displays that in the 2nd seven segment display. The exponent and mantissa are connected to
--the correct pins as required for the assignment.
--this then allows the interfacing and operation of the TestDisplay board to be tested and debugged.
--the display board runs the "TestDisplay" code
--Author C. J. Kikkert, October 2008.
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity FloatCon is
port( Clk: in std_logic;
--          IntIn: in std_logic_vector ( 15 downto 0 );              -- 16 bit interger input
            MantissaMSB: out std_logic_vector ( 3 downto 0 ); --4 msb of mantissa put to
            Exponent: out std_logic_vector ( 3 downto 0 );
            SevSeg1: out std_logic_vector (6 downto 0)     -- Seven segment display Digit 2
            );          --end of port
            attribute loc : string;
--          attribute loc of IntIn:signal is "P2, P3, P4, P5, P6, P7, P8, P9, P14, P15, P16, P17, P18, P19, P20,
P21" ;
            attribute loc of Exponent: signal is "P43, P42, P41, P40" ;          --pin allocation of
Exponent
            attribute loc of MantissaMSB:signal is "P39, P38, P37, P36";  --pin allocation of MSB of Mantissa
            attribute loc of SevSeg1:signal is "P24, P25, P26, P27, P28, P29, P30" ;

end FloatCon;

architecture FloatCon_A of FloatCon is
component Convert
            port(Intin: in std_logic_vector ;                -- 16 bit interger input
            Mantissa: out std_logic_vector ;                 --mantissa put to
            Exponent: out std_logic_vector );
end component;

component SevenSeg
            port (Hexin: in std_logic_vector;
            SevSegOut : out std_logic_vector);
end component;

Signal IntIn: std_logic_vector ( 15 downto 0 );              -- 16 bit interger input
Signal Mantissa: std_logic_vector(7 downto 0);
--Signal Exponent: std_logic_vector ( 3 downto 0 );

begin
            L1: Convert port map (IntIn, Mantissa, Exponent);
            D3: SevenSeg port map (Mantissa(3 downto 0), SevSeg1);   -- Seven Segment Display 2nd Digit

            MantissaMSB<= Mantissa(7 downto 4);

            P_Clock: process
            begin
            wait until rising_edge(Clk);
                        IntIn <= IntIn + X"0001";    -- Incement Integer value of converter input
            end process;

end FloatCon_A;
```

The code for the display board is shown below. This code again uses the standard seven segment display module.

```
--main program for the display module for the 2008 VHDL assignment
--Author C. J. Kikkert, October 2008
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity TestDisplay_E is

port(
        Exp: in std_logic_vector ( 3 downto 0 );          --Exponent
        MantM: in std_logic_vector ( 3 downto 0 );        --4 MSB of Mantissa
        SevSeg1: out std_logic_vector ( 6 downto 0 );
        SevSeg0: out std_logic_vector ( 6 downto 0 ) );
        attribute loc : string;
        --cable pin mapping exponent convert board 43 to 40 with same pins on display board
        attribute loc of Exp: signal is "P43, P42, P41, P40" ;                 --pin allocation of
Exponent
        attribute loc of MantM:signal is "P39, P38, P37, P36";       --pin allocation of MSB of Mantissa
--      attribute loc of Exp:signal is "P21, P20, P19, P18" ;                          --pin allocation
of Exponent Input from other board
--      attribute loc of MantM:signal is "P17, P16, P15, P14 " ; --pin alloc Mantissa 4 MSB's
        attribute loc of SevSeg0:signal is "P2, P3, P4, P5, P6, P7, P8" ;
        --pin allocation of MS display
        attribute loc of SevSeg1:signal is "P14, P15, P16, P17, P18, P19, P20" ;
--      attribute loc of SevSeg1:signal is "P24, P25, P26, P27, P28, P29, P30" ;
--      attribute loc of SevSeg0:signal is "P36, P37, P38, P39, P40, P41, P42" ;--pin allocation of LS
display
end;

architecture TestDisplay_A of TestDisplay_E is
        component SevenSeg
                port (Hexin: in std_logic_vector;
                SevSegOut : out std_logic_vector);
        end component;

begin
        D0: SevenSeg port map (Exp, SevSeg0);-- Seven Segment Display LS Digit
        D1: SevenSeg port map (MantM, SevSeg1);       -- Seven Segment Display Digit 2
end TestDisplay_A;
```

Step 4

The fourth and final step in the design and testing is for the converter module to be connected to an external input. For this a separate counter program is developed that is coded into a second Lattice Board. The code for the main board is changed to remove the P_Clock process counter above and use the input pins for the input. In addition the IntIn port declaration and pin allocation is enabled in the code on the previous page.

The code for the counter for the third Lattice board is as follows:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity TestGen_E is

port( Clk: in std_logic;
        IntIn: inout std_logic_vector(15 downto 0));

        attribute loc : string;
        attribute loc of Clk: signal is "P11";
        attribute loc of IntIn:signal is "P2, P3, P4, P5, P6, P7, P8, P9, P14, P15, P16, P17, P18, P19, P20,
P21" ;
```

```
end;

architecture TestGen_A of TestGen_E is
begin
        process
        begin
                wait until rising_edge (Clk);
                        IntIn <= IntIn + X"0001";
        end process;
end TestGen_A;
```

The final testing involves observing the seven segment displays as the input counter increments. Figure 4 shows the resulting Lattice Boards connected to demonstrate the operation of the floating point converter. The top board contains the input counter, the middle board contains the floating point converter and the bottom board is the display board. The left display element on the bottom board shows the Exponent and the next two displays show the Mantissa. Note that the least significant 4 bits of the Mantissa (5) is shown on both the main board and the display board due to the ribbon cable connection.

Instead of a lattice board containing a counter for the input generator, it is also possible to use a simple board with jumper switches to set any input data. That has been used for the marking of all the assignments.

The full code is on the subject web site on the engineering domain.

http://eng.jcu.edu.au/subjects/ee4306/assignments/2008 in the file VHDLAssignSoln08.zip file

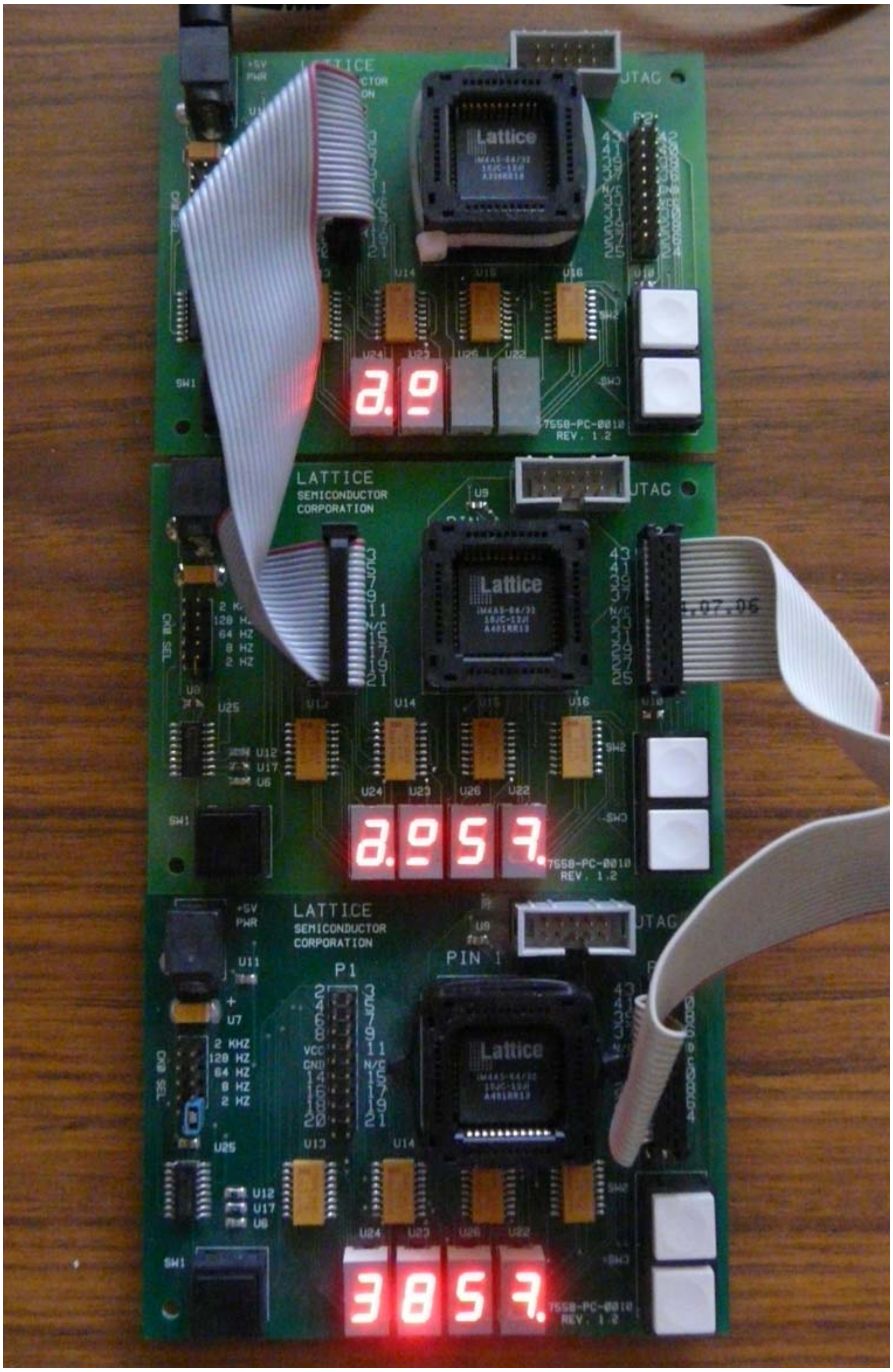


Figure 4: Integer Floating point Code Conversion in operation.