CONCORDIA UNIVERSITY
Department of Electrical and Computer Engineering
COEN 313 - Digital Systems Design II - Winter 2017

Lab 4: A combinational integer to floating-point converter

Objectives

•To become familiar with using VHDL combinational processes to design combinational logic.
•To become acquainted with some useful command line options to the vcom command.
•To become better acquainted with the subtle differences between signals and variables.

Introduction

In this lab, a combinational VHDL process will be used to design a combinational integer to floating-point converter. The converter will have a single input which represents an unsigned 4-bit binary integer and two outputs - an 8-bit output representing the integer value in a specific floating-point format and an output to indicate underflow situations (the input is a number less than 1, i.e. 0).

Floating-point representation

Recall that in floating-point representation, a real number is stored using a **sign** bit, an **exponent** and a **mantissa**. The simple format used in this lab is shown in Figure 1.

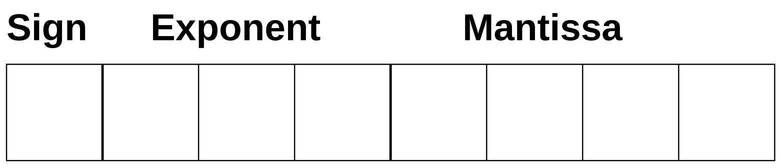

Figure 1: A simple 8-bit floating-point format.

In this format, a single bit is used to represent the sign of the number (0 = a positive number, 1 = a negative number). Three bits are used to store the exponent value in binary using a special notation known as *excess-3* notation. The remaining 4 bits are used to store the mantissa in normalized form (there are no leading zeros in the mantissa). Excess notation is a manner of representing both positive and negative values using only unsigned binary numbers. It is commonly used for the exponent field in many different floating point formats such as the IEEE 754 floating point standard. Table 1 lists the decimal value of an exponent and how it would be internally stored within the 3 bit exponent field of the floating point format shown in Figure 1.

Table 1: Excess-3 notation

Actual decimal value of exponent	Stored value of exponent in 3-bit unsigned binary (E_{stored}= E_{actual} + 3)
-3	000
-2	001
-1	010
0	011
1	100
2	101
3	110
4	111

To obtain the stored value of the exponent, we simply add 3 (hence the name excess-3) to the actual value of the exponent. For example, if the actual value of the exponent is -2:

$$\begin{aligned} E_{stored} &= E_{actual} + 3 \\ &= -2 + 3 \\ &= 1 \\ &= 001 \text{ (in 3-bit binary)} \end{aligned}$$

To obtain the actual value of the exponent, we simply remove the excess 3 from the stored value:

$$\begin{aligned} E_{actual} &= E_{stored} - 3 \\ &= 1 - 3 \\ &= -2 \end{aligned}$$

The mantissa is said to be *normalized* which means that there are no leading zeros in the mantissa. With this understanding, the mantissa represents a number in the range:

$$0.5 <= \text{mantissa} < 1$$

The possible values of the mantissa in this simple format are: .1XXX , where X can either be a 0 bit or a 1 bit. The value represented by a number in this format is given by the following expression:

Value = -1^{sign} x 0.mantissa x $2^{\text{exponent stored - 3}}$

Some examples will help to clarify the above:

Example 1:

Suppose the 8-bit floating-point format number is 0 110 1111 . We begin by extracting the sign, exponent, and mantissa portions of the number to obtain:

sign = 0
exponent = 110 = 6
mantissa = 1111

Value = -1^{0} x 0.1111 x 2^{6-3}
= (1) x 0.1111 x 2^{3}
= 111.1 (7.5 in decimal) (recall that mulitplication by powers of 2 in binary correspond the shifting the binary point a corresponding number of times in the appropriate direction)

Example 2:

To convert an integer value into its floating point representation, we perform the following:

3 = 11 (convert from decimal to binary)
= 0.1100 x 2^{2} (express as 0.mantissa x $2^{\text{exponent actual}}$)
= 0.1100 x 2^{5} (add the excess 3 to the actual exponent to obtain the stored exponent)

sign = 0
exponent = 5 = 101
mantissa = 1100

Assembling the three fields together yields 0 101 1100 as the value in the floating point representation.

The range of this format can be determined with a little thought and analysis. The smallest representable value is when we have the smallest value for the exponent (-3) with the smallest mantissa (0.1000) to give:

0.1000 x 2^{-3} = 0.0625

The largest value which can be represented in this format is:

$0.1111 \times 2^4 = 1111 = 15$

On the negative side of the number line, we would have the corresponding negative values. Figure 2 illustrates the range of numbers representable with this simplified format.

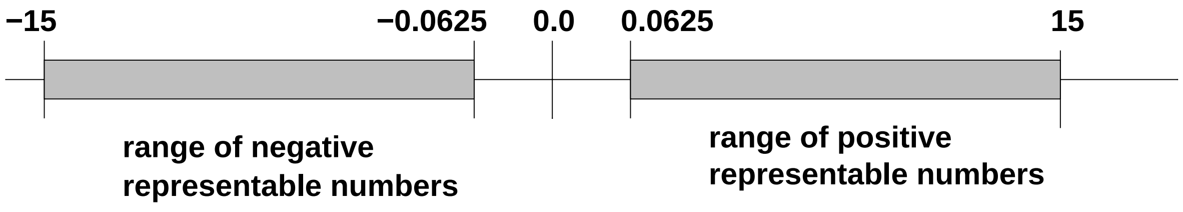
−15
−0.0625
0.0
0.0625
15
range of negative representable numbers
range of positive representable numbers

Figure 2: Range of representable numbers.

Note that there is no representation for exact 0 in this simple floating point format. Underflow is the term used to refer to numbers which are smaller than the smallest representable value and overflow is the term to signify when a number has a value beyond the largest representable.
In the case when the input is “0000”, the underflow output (`uf_out`) will be set to ‘1’, and the 8 bits of the floating point value (`fp`) will be set to “00000000”.

Design using a combinational VHDL process an integer converter to floating-point converter for the floating point format described above. Use the following VHDL entity specification:

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

entity  int_to_float   is
port( fp : out std_logic_vector(7 downto 0);
      uf_out : out std_logic ; -- underflow output
      int : in std_logic_vector(3 downto 0)); -- input value
end;
```

Your design should make use of a single combinational process (along with any other combinational VHDL statements which may be required such as CSAs to perform the negation of any top-level outputs to account for the active-low LEDs on the FPGA board).

Some Hints

• Make use of the `case` statement together with vector slices (array manipulation) and the concatenation operator in the process which converts the integer input into its floating-point representation.

• It may be helpful to complete Table 2 which lists the 16 possible inputs together with their representation in the floating-point format used in this lab. Then, use a “brute-force” case statement.

Table 2: All 16 inputs and their floating-point representation

Decimal input value	input value in binary	equivalent value with true exponent	representation in given floating-point format
0	0000	not applicable	0 000 0000 and uf = 1
1	0001	0.1×2^1	0 100 1000 and uf = 0
2	0010	0.10×2^2	0 101 1000 and uf = 0
3	0111	0.11×2^2	0 101 1100
8	1000	0.1000×2^4	0 111 1000
15	1111	0.1111×2^4	0 111 1111

Options to the `vcom` command

Most software tools used in engineering design are fairly complex and may be controlled by the user through the use of command line options. For example, the Modelsim simulator command vcom contains many such options (the first two labs did not make use of any of them). To obtain a list of all the available options, one may use the -help option as in:

```
ted@deadflowers Code 12:26pm >vcom -help
Model Technology ModelSim SE-64 vcom 6.6g Compiler 2012.05 May 23
2012
```

```
Usage: vcom [options] files
Options:
  -help              Print this message
  -time              Print the compilation wall clock time
  -version           Print the version of the compiler
  -32                Run in 32-bit mode
  -64                Run in 64-bit mode
  -work <path>       Specify library WORK
  -error <msgNumber>[,<msgNumber>...]
                     Report the listed messages as errors
  -warning <msgNumber>[,<msgNumber>...]
                     Report the listed messages as warnings
  -note <msgNumber>[,<msgNumber>...]
                     Report the listed message as notes
  -suppress <msgNumber>[,<msgNumber>...]
                     Suppress the listed messages
  -87                Enable support for VHDL 1076-1987 only
  -93                Enable support for VHDL 1076-1993
  -ams               Enable AMS wreal extensions
  -2002              Enable support for VHDL 1076-2002
  -check_synthesis   Check for compliance to some synthesis rules
```

(not all options are shown above as the list is long)

A helpful option is the -check_synthesis which checks that the VHDL code complies with synthesis rules. Writing VHDL meant for simulation purposes is different from writing VHDL code which is crafted to produce working hardware by a synthesis tool. Being aware of the differences between the two is the hallmark of a good digital designer. To paraphrase a well known song, "you can't always synthesize what you simulate". For example, in a combinational process, all signals which are being read from within the process should be listed in the sensitivity list. The -check_synthesis option will detect the existence of any signals which are read within a process but do not appear in the process sensitivity list.

In addition to simulation tool warnings, a designer should also be aware of any messages/warnings/errors generated by the logic synthesis tool. The Precision RTL synthesis tool creates a log file called "precision.log" (in the working project directory). One should always refer to this log file and check for any informative messages, warnings, or errors. Typical informative messages include ones relating to operator inferencing such as the following:

```
# Info: [44841]: Counter Inferencing === Detected : 2, Inferred
(Modgen/Selcoun
ter/AddSub)
```

Other messages may be due to design errors and are more serious and will cause synthesis to fail:

```
# Error: Net is driven by multiple primitive gates --  NET:
int_value(4)  DRIVING GATE: lat_int_value(4)
```

Some messages may be related to warnings which appear to be innocuous, but may lead to bizarre circuit behaviour if left ignored:

```
# Warning: [9033]: Combinatorial feedback loop on a 2 input AND
gate. Behavior is ’X’ to stuck at ’0’. Input net:keith of
instance:ix1 is connected to the output.
```

One should always be attentive of any warning/error messages and investigate the reason for their appearances.

Requirements

1. Modelsim simulation results for the design. Test your design for all possible input values.

2. RTL schematic diagram of the synthesized circuit.

3. The precision.log file highlighting any messages/warnings. Discuss the importance of any ge nerated message or warning.

4. Download to the FPGA board and demonstrate to your TA the operation of your circuit.

Questions

1. Although, the “brute-force” case statement method is manageable for a 4-bit input value, it becomes unwieldy if the input is larger (such as a 32 bit vector). An alternative method is to count the number of leading 0’s present in the input, and based on the number of leading zeros, determine the value of the mantissa and exponent portions of the floating point representation accordingly. For example, consider input numbers which have two leading zeros:

$00\underline{10} = .\underline{10}\mathbf{00} \times 2^{2}$
$00\underline{11} = .\underline{11}\mathbf{00} \times 2^{2}$

In both of these cases, the *first* two bits of the mantissa of the floating point format are the *last* two bits of the input value and we add two 0s at the rightmost side. The value of the (true) exponent in these cases is 2.

Now, consider an input which has one leading zero:

$0\underline{110} = .\underline{110}\mathbf{0} \times 2^{3}$

The *first three* bits of the mantissa are the same as the *last three* of the input and we pad the right-most most side of the mantissa with a single 0, and the true value of the exponent is 3.

Rewrite your VHDL code using a process which counts the number of leading zeros present in the input and based on this number, determines the values of the mantissa and exponent fields of the floating point format accordingly.

Hint: Should you use a signal or a variable to keep track of the number of leading zeros?

Another hint: Use a boolean to determine when you have found the first 1 in the input and when the first 1 is found, stop counting 0s.

Synthesize your design and compare the RTL schematics as produced by Precision RTL.
You do not have to download this modified version of the code to the FPGA board.

2. What will result (during synthesis) if a signal appears on both sides of the signal assignment operator (<=) within a combinational VHDL process such as:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity is_this_good_ou_mauvaise is
port( mick : in std_logic;
      keith : out std_logic;
end;

architecture rtl of is_this_good_ou_mauvaise is

begin

signal stone: std_logic;

process(mick)
begin
  stone <= mick and stone;
end ;

keith <= stone ; tout le monde sais que Keith == stone
end ;
```

T. Obuchowicz
Revised: March 10, 2017