EE 121 October 10, 2002
Digital Design Laboratory Handout #10

Laboratory Assignment #3

FLOATING POINT CONVERSION

Due date: Friday, October 18. Prelab due: Tuesday, October 15

For this laboratory assignment, you will use Xilinx Foundation software to design and test a combinational circuit that converts a 12-bit *linear encoding* of an analog signal into a *companded* 8-bit floating-point (FP) representation. You will use the Xilinx Foundation schematic editor to enter a hierarchical logic design of the circuit, which you will test by simulating it under Xilinx Foundation.

The logic symbol for this floating-point encoder is shown below.

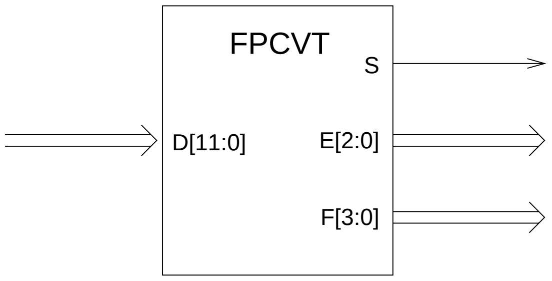

Figure 1: FPCVT Logic Symbol

The pins of the FPCVT logic block have the following uses:

FPCVT Pin Descriptions	
D[11:0]	Input data in twos'-complement representation: D0 is the least significant bit, D11 is the sign bit
S	Sign bit of floating-point representation
E[2:0]	3-bit exponent of floating-point representation
F[3:0]	4-bit significand of floating-point representation

BACKGROUND

Analog signals are often converted to digital form for storage or transmission. A linear encoding using 8 bits can represent the unsigned number range $\{0, \ldots, 255\}$ or (using twos'-complement representation) the signed range $\{-128, \ldots, +127\}$. Seven or eight bits of precision is adequate for intelligible speech or almost listenable music, but does not provide sufficient dynamic range to capture both loud and soft sounds. Therefore nonlinear encodings are used in most commercial systems. These encodings represent signals by numbers that approximate the logarithms of the analog values. Two standard systems, μ-law PCM and A-law PCM, are described briefly at the end of this handout.

For this laboratory assignment, we will use a simplified floating-point representation consisting of one sign bit, a 3-bit *exponent*, and a 4-bit *significand* (also called the *fraction* or, somewhat inaccurately, the *mantissa*):

7	6	5	4	3	2	1	0
S		E			M		

The value represented by an 8-bit byte in this format is

$$V = (1 - 2S) \cdot M \cdot 2^E \,.$$

The factor $(1 - 2S)$ is a mathematical trick for stating that the value is negative (or zero) when the sign bit is 1 and positive (or zero) when the sign bit is 0. The 4-bit significand M ranges from $0000_2 = 0$ to $1111_2 = 15$, and the exponent ranges from $000_2 = 0$ to $111_2 = 7$.

The following table shows the values corresponding to several FP representations.

Floating Point Representation Examples		
FP representation	Formula	Value
$\boxed{0}\boxed{000}\boxed{0000}$	$+0 \times 2^0$	0
$\boxed{1}\boxed{010}\boxed{1010}$	-10×2^2	-40
$\boxed{0}\boxed{011}\boxed{0111}$	$+7 \times 2^3$	56
$\boxed{0}\boxed{010}\boxed{1110}$	14×2^2	56

The last two rows of the above table demonstrate that some numbers have multiple FP representations. The preferred representation is the one in which the most significant bit of the significand is 1; this representation is said to be *normalized.*

It is quite straightforward to produce the linear encoding corresponding to a floating-point representation; this operation is called *expansion.* This laboratory assignment is to build a combinational circuit for the inverse operation, called *compression.* A device that performs both expansion and compression is called a *compander.* The compression half of a compander is more challenging because there are more input bits than output bits and therefore many different linear encodings must be mapped to same floating-point representation. Values that do not have FP representations should be mapped to the closest FP encoding; this process is called *rounding.*

To simplify the conversion procedure, first consider the 12-bit linear encoding of a non-negative number. The sign bit `D11` is 0, and the remaining bits `D[10:0]` encode an integer in the range $0 \ldots 2047$. Large numbers have fewer leading zeroes than small numbers. The FP exponent is determined by counting the number of leading zeroes (including the sign bit `D11`), as shown in the following table.

Leading Zeroes	Exponent
1	7
2	6
3	5
4	4
5	3
6	2
7	1
≥ 8	0

The significand consists of the 4 bits immediately following the last leading 0. (When the exponent is 0, the significand is the least significant 4 bits; these representations are called *denormalized*.) For example, $422_{10} = 000110100110_2$ has three leading zeroes (including the sign bit), so its exponent is $5 = 101_2$, and its significand is 1101, yielding the FP representation $\boxed{0}\boxed{101}\boxed{1101}$. This FP representation expands to $+13 \times 2^5 = 416$; the number 422 cannot be represented exactly but with an error of about 1.5%.

The above procedure produces the correct FP representation for about half the linear encodings, but it does not guarantee the most accurate representation. The circuit that you will design is required to *round* the linear encoding to the nearest FP encoding. You should use the simple rounding rule that depends only on the fifth most significant bit. If that bit is 0, the nearest number is obtaining by truncation—simply use the four most significant bits. But if the fifth bit is 1, the representation is obtained by rounding up—that is, adding 1. The following table gives examples of rounding.

Rounding Examples		
Linear Encoding	FP Encoding	Round
000000101100	$\boxed{0}\boxed{010}\boxed{1011}$	Down
000000101101	$\boxed{0}\boxed{010}\boxed{1011}$	Down
000000101110	$\boxed{0}\boxed{010}\boxed{1100}$	Up
000000101111	$\boxed{0}\boxed{010}\boxed{1100}$	Up

The rounding stage of the FP conversion can leading to an unpleasant complication. When the maximum significand 1111 is rounded up, adding one causes an *overflow*; the result, 10000, does not fit in the 4-bit significand field. This problem is solved by dividing the significand by 2, or right shifting, to obtain 1000, and increasing the exponent by 1 to compensate. For example,

$$000001111101 \rightarrow \boxed{0}\boxed{3}\boxed{10000} \text{ (Oops!)} \rightarrow \boxed{0}\boxed{4}\boxed{1000}$$

In this example, 125 is converted to $8 \times 2^4 = 128$, which is indeed the closest FP number. Note that the overflow possibility can be detected either before or after the addition of the rounding bit. (Which method is easier?)

When rounding very large linear encodings, such as $2047 = 01111111111_2$, the exponent may be incremented beyond 7 to 8, which cannot be stored in the exponent field. One solution to this problem is to use the largest possible FP representation $\boxed{1 \mid 111 \mid 1111}$. For this lab assignment, you can employ an easier simpler solution: ignore the problem.

An overall block diagram for the floating-point conversion circuit is shown below.

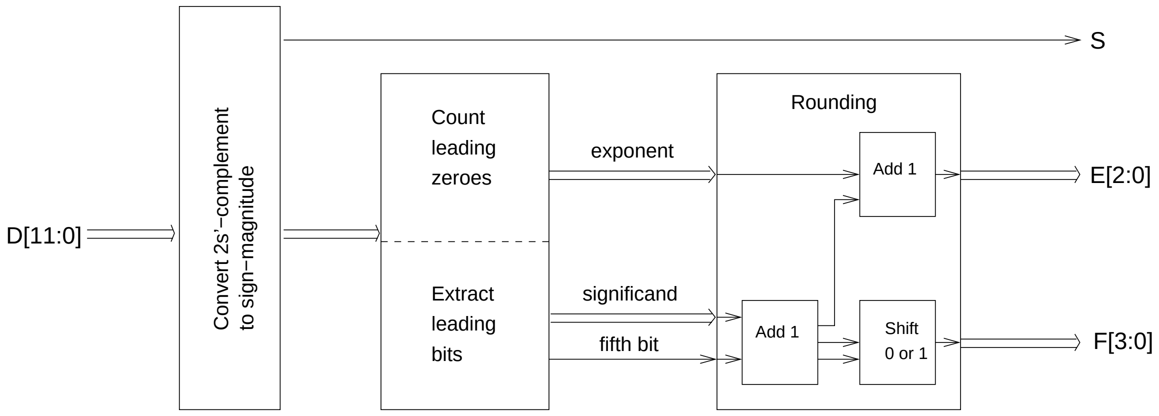

Figure 2: FPCVT block diagram

The first block converts the 12-bit twos'-complement input to sign-magnitude representation. Nonnegative numbers (sign bit 0) are unchanged, while negative numbers are replaced by their absolute value. As explained in section 2.5 of *DDPP*, the negative of a number in twos'-complement representation can be found by complementing (inverting) all bits, then incrementing (adding 1) to this intermediate result. One problem case is the most negative number, $-2048 = 100000000000$; when complement-increment is applied, the result is 100000000000, which looks like -2048 instead of $+2048$. Ignore this case; it requires special case handling by the rounding block as well.

The second block performs the basic linear to floating-point conversion. The exponent output encodes the number of leading zeroes of the linear encoding, as shown in the table above. The significand output is obtained by right shifting the most significant input bits from 0 to 7 bit positions. What this means is that each bit of significand comes from one of 8 possible magnitude bits, and therefore the suggested implementation of the basic conversion block uses 8-to-1 multiplexers.

The third block performs rounding of the floating-point representation. If the fifth most significant bit of the intermediate FP representation is 1, the significand is incremented by 1; if the significand overflows, it is right shifted one bit and the exponent is increased by 1.

PRELAB

1. What number has the FP representation $\boxed{1 \mid 111 \mid 1111}$?
2. How many different FP representations correspond to the number 0 ?
3. Which numbers have FP representations of the form $\boxed{0 \mid 010 \mid \text{xxxx}}$?

4. How many different FP representations correspond to the number 80 ?

5. How many bits are needed for the twos'-complement representation (linear encoding) of numbers whose FP representations have a sign bit, 4 exponent bits, and 6 significand bits?

6. Our 8-bit FP format can represent at most 256 different numbers. But many numbers have multiple representations. Exactly how many distinct numbers have FP representations?

7. (Bonus) Our FP representation is inefficient because the most significant bit of the significand is 1 except for the denormalized numbers. The IEEE 754 standard 32-bit and 64-bit floating point representations use a *hidden bit*, just to the left of the leftmost significand bit. Suppose we use the same trick to increase the precision of our FP representation.

   a. Find the modified FP representation of 256.

   b. What number has the modified FP representation of $\boxed{0 \mid 000 \mid 0000}$?

   c. Trick question: How would you represent the number 0 in the modified representaton?

REQUIREMENTS

As penance for all the paper we wasted on Lab 1, we will use electronic submission for this and all future labs in this course. You do not need to print anything out. To submit your lab electronically:

1. Write a README file that contains your name, your partner's name, the name of the project, and any information that the TAs should know when evaluating your design.

   You should say whether or not you think your design completely meets the assignment's requirements. If it does, tell us which script file(s) you used to prove to yourself that your design worked.

   If your design doesn't work, explain your approach and what problems you've encountered. List each major component, whether or not you think it works, and what script file(s) you used to verify them. Identify what component(s) or what stage of integration does not work.

   Designs without a README file will lose points. But if your design passes both your tests and ours, then we will not expect any analysis of your design in the README file. A simple file with the basic information (names, project name, whether you think it works, and the name of a script file) is all that is necessary if your design is fully functional. If it is not, however, the README file is very important. Nonfunctional designs without README files will almost certainly receive very poor grades; nonfunctional designs that clearly identify what works and what does not may receive substantial partial credit, depending, of course, on exactly how much is working.

2. Choose `Project->Clear` Implementation Data from the Project Manager to remove all the implementation-related files that we don't need to see.

3. Choose `File->Archive Project`. Set the Compression Factor to Max. Do not give your archive a password. Then click Next.

4. Accept the default choice of archiving the entire project. Click Next.
5. Add any outside your project directory that you want to add to your archive; usually, you won't need to add any. Then click Finish. A Zip file with your project's name will appear in the directory one level above your project directory. This is your archived project.
6. Submit your archived project in one of the following ways:
   a. Drop it in the "submit" directory of your Z drive on the EE 121 server. You must use one of the computers in Packard 128 to access your submit directory. You will be able to write to the submit directory but not modify or delete any of the files in it. If you need to re-submit your project, you must copy a new archive into the submit directory.
   b. Email your archived project as an attachment to the TA who heads your lab section.

Your FPCVT design must use at least two levels of hierarchy, and you will probably find that three levels are most useful. Your lowest-level schematics may use any of the gates from the Xilinx Foundation XCS2S100 library, including basic gates, such as AND2 and INV, and more complex parts, such as the MX8_1E 8-input multiplexer and the X74_148 8-input priority encoder. (The X74_148 has active low inputs, so modifications or additions will be needed it if is to be used for detecting the first 1.)

For this laboratory assignment, partners should work independently. Only normal homework collaboration is permitted.

You must submit your project electronically, whether or not it works correctly, before 5:00pm on the due date or you will receive no credit.

Appendix: Standard Companded Representations

The floating-point representation used in this assignment is not actually used in standard companded encodings. Its main drawback is that many numbers have multiple representations, so there are fewer than 256 representable numbers.

Two standard encodings are the μ-law PCM ("mu-law" pulse-coded modulation) used in the North American telephone network and the A-law PCM used in European telephone networks. Both μ-law and A-law PCM use 8-bit representations with a sign bit, 3 exponent bits, and 4 mantissa bits.

The conversion formula for μ-law PCM, described in section 10.1.6 of *DDPP*,is

$$(1 - 2S) \cdot (2^E \cdot (2M + 33) - 33)\,,$$

which results in a range of $\{-8159, \ldots, +8159\}$ and minimum step size 2.

For A-law PCM, the minimum step size is also 2 but the range is $\{-4032, \ldots, +4032\}$. Exponent values (called segment codes) of 0 and 1 are used to represent the 32 numbers $\{1, 3, \ldots, 63\}$. Thus compared to μ-law PCM, A-law PCM has finer resolution for small values but less dynamic range.

It should be noted that a common method for performing expansion to linear encoding is by a lookup table stored in read-only memory (ROM). The combinational circuit approach uses far fewer gates but requires more design effort.