ROUNDING
Rounding

- Rounding is a fundamental method to reduce the size of a word, such as after arithmetic operations
  - For example to maintain the word width for memory storage

- Bits are removed from the LSB end of the word

XXX

YYYYY
• Another example: if we multiply two 5-bit words, the product will have 10 bits
  \[ xxxxx \times yyyyy = zzzzzzzzzzz \]
  and we likely can not handle or do not want or need all that precision
• More issues are present with signed data
• Issues vary for different formats:
  – unsigned
  – 2’s complement
  – sign magnitude
  – etc.
Rounding

• Rounding modes in IEEE 754 are much more complex than what is commonly needed in digital signal processing systems

• There are four fundamental rounding modes whose matlab function names are:
  1) round(·): towards nearest integer
     • Generally the best rounding algorithm
  2) fix(·): truncates towards zero
  3) floor(·): rounds towards negative infinity
  4) ceil(·): rounds towards positive infinity
1) matlab round()

- Often the best general-purpose rounding mode
- “Unbiased” rounding
- Symmetric rounding for positive and negative numbers
- Max error $\frac{1}{2}$ LSB
2) matlab fix()

- Truncates toward zero
- Numerical performance is poor
- Symmetric rounding for positive and negative numbers
- Very simple hardware for the magnitude of sign magnitude (simple truncation)
  -xxxxxx in
  -xxxx-- out
- Max error 1 LSB
3) matlab floor()

- Numbers rounded down towards $-\infty$
- Numerical performance is poor
- Very simple hardware for 2’s complement (simple truncation)
  - $\cdots\cdots\cdots\text{in}$
  - $\cdots\cdots\cdots\text{out}$
- Max error 1 LSB
4) `matlab ceil()`

- Numbers rounded up toward +infinity
- Numerical performance is poor
- Max error 1 LSB
Hardware Rounding: 
A) Truncation

A. The easiest hardware method is truncation

- \(xxx.xxxxx\)  
  \(xxx.xx--\)

- Simply neglect the truncated bits and remove all hardware which calculates only those bits

- Maximum rounding error \(\sim 1\) post-rounded LSB

- Sign magnitude format numbers (obviously the magnitude portion)
  - Positive and negative numbers both truncate towards zero
  - Same as matlab \texttt{fix}(\texttt{•})

- 2’s complement format numbers
  - All numbers truncate towards negative infinity
  - Same as matlab \texttt{floor}(\texttt{•})

- Unsigned format numbers
  - All numbers truncate towards zero (negative infinity)
  - Same as matlab \texttt{fix}(\texttt{•}) or \texttt{floor}(\texttt{•})
B. **Method #5.** Add $\frac{1}{2}$ LSB (that is, one half of the LSB of the output) and then truncate

- This does not correspond to any of the matlab rounding functions for all binary formats
- Maximum rounding error $\frac{1}{2}$ of the post-rounded LSB

\[
\begin{array}{c}
\text{input} \\
\text{intermediate sum} \\
\text{rounded output}
\end{array}
\]

\[
\begin{array}{c}
1 \\
+ \text{xxxxx.xxx} \\
\text{yyyyy.yxx} \\
\text{yyyyy.--} \\
\end{array}
\]

the post-rounded LSB position

\[\text{rounding bit added here}\]
Hardware Rounding: B) Add $\frac{1}{2}$ LSB and Truncate

- It is often not difficult to find a place to add the extra "1" in a complex datapath if you plan ahead.

\[ \text{keep these bits} \quad \text{truncate these bits after adding everything (to get the correct carry bits)} \]

"1" rounding bit has a weight of $\frac{1}{2}$ of the post-rounded LSB.
Hardware Rounding: B) Add $\frac{1}{2}$ LSB and Truncate

- It is often not difficult to find a place to add the extra “1” in a complex datapath if you plan ahead.
Hardware Rounding: B) Add \( \frac{1}{2} \) LSB and Truncate

- The exact behavior depends on the number format being used:
  - Unsigned
    - Unbiased rounding
  - Magnitude portion of Sign magnitude
    - Unbiased rounding
  - 2’s complement
    - Both positive and negative \( xxxx.1000 \) cases round towards positive infinity as explained previously
    - The behavior requires a little more analysis
B) Add $\frac{1}{2}$ LSB and Truncate

**Unsigned, Sign Magnitude**

- matlab `floor(x+1/2)`
- matlab `fix(x+1/2)`
- Both positive and negative xxxx.1000 cases round away from zero just like `round()`
- Functions the same as matlab `round()` which is the best of our four matlab rounding functions
- Max error $\frac{1}{2}$ LSB
B) Add $\frac{1}{2}$ LSB and Truncate

2’s Complement

- matlab $\text{floor}(x+1/2)$
- The numerical performance is often sufficient
- $\begin{array}{c}
1 \\
+ \overline{x x x x x x} \\
\overline{y y y y x x} \\
\overline{y y y - -} \\
\end{array}$
- Biased rounding for 2’s complement
- Max error $\frac{1}{2}$ LSB
B) Add ½ LSB and Truncate

**2’s Complement**

- There are three key cases to consider for 2’s complement:
  a. When the input is of the form \texttt{xxxxx.100} (base 2) in the example above, and positive
     - Rounding is towards positive infinity which is the same as \texttt{round(•)}
  b. When the input is of the form \texttt{xxxxx.100} (base 2) in the example above, and negative
     - Rounding is towards positive infinity which is NOT the same as \texttt{round(•)}
  c. Otherwise
     - It performs the same as matlab \texttt{round(•)}
B) Add ½ LSB and Truncate

2’s Complement

- The biased rounding in the $\text{xxxx.1000}$ cases when using 2’s complement may be fine in many cases, especially when many bits are being rounded off, but if only a few bits are being rounded off, the case that differs from round() occurs more often.

- **Example:** $\text{xxxxxx.x}$ rounded to $\text{yyyyyy}$

<table>
<thead>
<tr>
<th>Pre-rounded value</th>
<th>Rounding action</th>
<th>Net effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(+)$ $\text{xxxxxx.0}$</td>
<td>No change in value</td>
<td>Same as round()</td>
</tr>
<tr>
<td>$(+)$ $\text{xxxxxx.1}$</td>
<td>Rounds to integer +0.5</td>
<td>Same as round()</td>
</tr>
<tr>
<td>$(-)$ $\text{xxxxxx.0}$</td>
<td>No change in value</td>
<td>Same as round()</td>
</tr>
<tr>
<td>$(-)$ $\text{xxxxxx.1}$</td>
<td>Rounds to integer +0.5</td>
<td>Same as round() +1</td>
</tr>
</tbody>
</table>
B) Add $\frac{1}{2}$ LSB and Truncate

2’s Complement

- Example: positive values of $\ldots .xxxxxx$ rounded to $yyyyyy$

<table>
<thead>
<tr>
<th>Pre-rounded value</th>
<th>Rounding action</th>
<th>Net effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>(+) $xxxxxx\ldots00000$</td>
<td>No change in value</td>
<td>Same as round()</td>
</tr>
<tr>
<td>(+) $xxxxxx\ldots00001$</td>
<td>Rounds down to integer</td>
<td>Same as round()</td>
</tr>
<tr>
<td>(+) $xxxxxx\ldots00010$</td>
<td>Rounds down to integer</td>
<td>Same as round()</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(+) $xxxxxx\ldots10000$</td>
<td>Rounds up to integer</td>
<td>Same as round()</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(+) $xxxxxx\ldots11101$</td>
<td>Rounds up to integer</td>
<td>Same as round()</td>
</tr>
<tr>
<td>(+) $xxxxxx\ldots11110$</td>
<td>Rounds up to integer</td>
<td>Same as round()</td>
</tr>
<tr>
<td>(+) $xxxxxx\ldots11111$</td>
<td>Rounds up to integer</td>
<td>Same as round()</td>
</tr>
</tbody>
</table>
B) Add ½ LSB and Truncate

2’s Complement

- Example: negative values of \( \text{xxxxxx.xxxxxx} \) rounded to \( \text{yyyyyy} \)

<table>
<thead>
<tr>
<th>Pre-rounded value</th>
<th>Rounding action</th>
<th>Net effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>((-) \text{xxxxxx.000000} )</td>
<td>No change in value</td>
<td>Same as round()</td>
</tr>
<tr>
<td>((-) \text{xxxxxx.000001} )</td>
<td>Rounds down to integer</td>
<td>Same as round()</td>
</tr>
<tr>
<td>((-) \text{xxxxxx.000010} )</td>
<td>Rounds down to integer</td>
<td>Same as round()</td>
</tr>
<tr>
<td>(-\ldots) \text{xxxxxx.00010} )</td>
<td>(-\ldots) \text{xxxxxx.00010} )</td>
<td>(-\ldots) \text{xxxxxx.00010} )</td>
</tr>
<tr>
<td>((-) \text{xxxxxx.010000} )</td>
<td>Rounds up to integer</td>
<td>Same as round() +1</td>
</tr>
<tr>
<td>(-\ldots) \text{xxxxxx.010000} )</td>
<td>(-\ldots) \text{xxxxxx.010000} )</td>
<td>(-\ldots) \text{xxxxxx.010000} )</td>
</tr>
<tr>
<td>((-) \text{xxxxxx.101101} )</td>
<td>Rounds up to integer</td>
<td>Same as round()</td>
</tr>
<tr>
<td>(-\ldots) \text{xxxxxx.101101} )</td>
<td>(-\ldots) \text{xxxxxx.101101} )</td>
<td>(-\ldots) \text{xxxxxx.101101} )</td>
</tr>
<tr>
<td>((-) \text{xxxxxx.111101} )</td>
<td>Rounds up to integer</td>
<td>Same as round()</td>
</tr>
<tr>
<td>(-\ldots) \text{xxxxxx.111110} )</td>
<td>(-\ldots) \text{xxxxxx.111110} )</td>
<td>(-\ldots) \text{xxxxxx.111110} )</td>
</tr>
<tr>
<td>((-) \text{xxxxxx.111111} )</td>
<td>Rounds up to integer</td>
<td>Same as round()</td>
</tr>
</tbody>
</table>
C. Unbiased Rounding: the same as matlab round(·)

- For cases where a “DC” bias is unacceptable, positive and negative numbers must be rounded differently with 2’s complement
- Although logically simple, implementing an unbiased rounding with 2’s complement numbers can increase the critical path delay significantly
- The calculation is not so complex if the only operation is rounding, but this is uncommon. Things get interesting in the common case when rounding is the last step in a series of calculations.
Hardware Rounding:  
C) Unbiased for 2’s Complement

- Here is one basic algorithm (there are others)
  1) **Remove** the normal $\frac{1}{2}$ LSB rounding bit
  2) Keep the output when the result(!) is:
     i. Negative and
     ii. Of the form $xxxxx.1000$
     • Equivalently, we could also not add the $\frac{1}{2}$ LSB when the result is in the range:
       $xxxxx.0000$ to $xxxxx.1000$
       Do you see why?
  3) Otherwise, **add** the $\frac{1}{2}$ LSB rounding bit back into the input and **recalculate the output**
  4) Truncate as with method (B)

```
0
+ xxx.xxxxx
  yyy.yyyxx
  yyy. yy--
```
Hardware Rounding: C) Unbiased for 2’s Complement

Here is a second basic algorithm

1) *Add* the normal ½ LSB rounding bit
2) Keep the output when the result(!) is *not*:
   i. (Negative and
   ii. of the form \(xxxxx.0000\))
   iii. Or zero
3) Otherwise, *remove* the ½ LSB rounding from the input and *recalculat* the output
4) Truncate as with method (B)

\[
\begin{array}{c}
1 \\
+ \overline{xxx.xxxxx} \\
\underline{yyy.yyyxx} \\
\underline{yyy.yy---}
\end{array}
\]
Hardware Rounding:
C) Unbiased for 2’s Complement

- A third option is to calculate the result two times in parallel:
  1) with $\frac{1}{2}$ LSB added in
  2) without $\frac{1}{2}$ LSB added in
The correct answer is then selected with a mux when it is known which result is correct using one of the previously-described algorithms or another
  - This is faster than the other two approaches however it requires about twice as much hardware which could be unacceptably expensive in area and energy dissipation
• copy, paste, and try it out