ROUNDING
Rounding

- Rounding operations eliminate lower-weight bits; in other words, bits on the LSB end of the word
- It is commonly necessary to reduce the number of bits due to word growth
  - For example, if we multiply two 5-bit words, the product will have 10 bits
    \[
    \text{xxxxx} \times \text{yyyyy} = \text{zzzzzzzzzzz}
    \]
    and we likely can not handle or do not want or need all that precision
  - In some cases such as when processed data must be written back into its source memory, a reduction in word size is clearly required
Rounding

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

• Matlab rounding
  1) round(⋅): towards nearest integer
     • Pos. and neg. numbers are rounded symmetrically about zero
     • Generally the best possible rounding algorithm
  2) fix(⋅): truncates towards zero
     • Pos. and neg. numbers are rounded symmetrically about zero
  3) floor(⋅): rounds towards negative infinity
  4) ceil(⋅): rounds towards positive infinity
1) matlab round()

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

- Truncates toward zero
- Numerical performance poor
- Symmetric rounding for positive and negative numbers
- Max error 1 LSB
3) Truncation, or matlab `floor()`

- Numbers rounded down towards \(-\infty\)
- Numerical performance poor
- Very simple hardware
- \(\ldots\) in \(\ldots\) out
- Max error 1 LSB
4) matlab ceil()

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

A. The easiest is truncation

- $xxx.xxxxx$
  
  $xxx.xx---$

- Maximum rounding error $\sim 1$ post-rounded LSB

- Signed magnitude
  - Positive and negative numbers both truncate towards zero
  - Matlab $\text{fix(·)}$

- 2’s complement and unsigned
  - All numbers truncate towards negative infinity
  - Matlab $\text{floor(·)}$
Hardware Rounding:  
B) Add ½ LSB and Truncate

B. Better rounding numerically is to add ½ LSB (that is, one half of the LSB of the output) and then truncate

\[
\begin{array}{c}
1 \\
+ \ xxx \ xxxxx \\
\hline
yyy \ yyyyxx \\
yyy \ yy---
\end{array}
\]

- Our 5th rounding method
- Maximum rounding error ½ post-rounded LSB
- Two cases:
  a. When the input is xxxx.5000 (base 10) (or xxx.xx100 (base 2) in the example above)
    - Rounding is towards positive infinity (for both positive and negative numbers)
    - matlab ceil(·)
  b. Otherwise
    - Performs best rounding: matlab round(·)
Hardware Rounding:
B) Add $\frac{1}{2}$ LSB and Truncate

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

![Diagram showing keep and truncate bits for rounding]
Hardware Rounding:
B) Add $\frac{1}{2}$ LSB and Truncate

- But there is a biased rounding of the $\text{xxx}.1000$ cases
  - Is fine in many cases, especially when many bits are being rounded off (then the $\text{xxxx}.1000$ case is less frequent)
  - The exact behavior depends on the number format being used:
    - Signed magnitude
      - Both positive and negative $\text{xxxx}.1000$ cases round away from zero
    - 2’s complement and unsigned
      - Both positive and negative $\text{xxxx}.1000$ cases round towards positive infinity
Add $\frac{1}{2}$ LSB and Truncate 2’s Complement

- matlab floor($x+1/2$)
- The numerical performance is often sufficient
- $1$
  $+ \overline{xxxxxx}$
  $\underline{yyyyxx}$
  $yyyy---$
- Biased rounding for 2’s complement
- Max error $\frac{1}{2}$ LSB
Add \( \frac{1}{2} \) LSB and Truncate Signed Magnitude

- \texttt{matlab floor}(x+1/2)
- Functions same as \texttt{matlab round()}
- Unbiased rounding for signed magnitude
- Max error \( \frac{1}{2} \) LSB
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
Hardware Rounding: C) Unbiased

– 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)
Hardware Rounding: C) Unbiased

- 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 # A negative integer
     ii. of the form xxxx.0000) # If input was −0.5
     iii. Or zero
  3) Otherwise, remove the ½ LSB rounding from the input and recalculate the output
  4) Truncate as with method (B)

\[
1 + xxx.xxxxx \\
\hline
yyy.yyyxx \\
yyy.yy---
\]
Hardware Rounding: C) Unbiased

- 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
- This is likely faster than the other two approaches however it does require about double the amount of hardware