#### 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



### Rounding

• Another example: if we multiply two 5-bit words, the product will have 10 bits

xxxxx × yyyyy = zzzzzzzz 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( $\cdot$ ): towards nearest integer
    - Generally the best rounding algorithm
  - 2) fix( $\cdot$ ): truncates towards zero
  - 3) floor( $\cdot$ ): rounds towards negative infinity
  - 4) ceil( $\cdot$ ): 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 <sup>1</sup>/<sub>2</sub> 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 –infinity
- Numerical performance is poor
- Very simple hardware for 2's complement (simple truncation)
  - xxxxxx in
     xxxx-- 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 ~1 post-rounded LSB
  - Sign magnitude format numbers (obviously the magnitude portion)
    - Positive and negative numbers both truncate towards zero
    - Same as matlab fix (•)
  - 2's complement format numbers
    - All numbers truncate towards negative infinity
    - Same as matlab floor (•)
  - Unsigned format numbers
    - All numbers truncate towards zero (negative infinity)
    - Same as matlab fix (•) and floor (•)

- **B. Rounding Algorithm #5.** Add ½ 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 <sup>1</sup>/<sub>2</sub> of the post-rounded LSB



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



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



- The exact behavior depends on the number format being used:
  - Unsigned
    - Unbiased rounding, same as round()
  - Sign magnitude, magnitude portion
    - Unbiased rounding, same as round()
  - 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 <sup>1</sup>/<sub>2</sub> 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 ½ LSB



#### B) Add <sup>1</sup>/<sub>2</sub> LSB and Truncate 2's Complement

- matlab floor (x+1/2)
- The numerical performance is often sufficient
  - 1 + xxxxxx yyyyxx yyy---
- Biased rounding for 2's complement
- Max error <sup>1</sup>/<sub>2</sub> LSB



#### B) Add ½ LSB and Truncate 2's Complement

- There are three key cases to consider for 2's complement: (examples assume output is rounded to an integer)
  - a. When the input is of the form xxxxx.100 (base 2) in the example above, and positive
    - Rounding is towards positive infinity which **is the same** as **round(•)**
  - b. When the input is of the form xxxxx.100 (base 2) in the example above, and negative
    - Rounding is towards positive infinity which is NOT the same as round (•)
  - c. Otherwise
    - It performs the same as matlab **round(•)**

#### B) Add ½ LSB and Truncate 2's Complement

- The biased rounding in the xxx.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: xxxxxx.x rounded to yyyyyy

Pre-rounded value		Rounding action	Net effect
(+)	xxxxxx.0	No change in value	Same as round()
(+)	xxxxxx.1	Rounds to integer +0.5	Same as round()
( - )	xxxxxx.0	No change in value	Same as round()
( - )	xxxxxx.1	Rounds to integer +0.5	Same as round() +1

#### B) Add <sup>1</sup>/<sub>2</sub> LSB and Truncate 2's Complement

• Example: 64 positive values of xxxxxx rounded to yyyyyy

Pre-rounded value	Rounding action	Net effect
(+) xxxxxx.000000	No change in value	Same as round()
(+) xxxxxx.000001	Rounds down to integer	Same as round()
(+) xxxxxx.000010	Rounds down to integer	Same as round()
(+) xxxxxx.100000	Rounds up to integer	Same as round()
(+) xxxxxx.111101	Rounds up to integer	Same as round()
(+) xxxxxx.111110	Rounds up to integer	Same as round()
(+) xxxxxx.111111	Rounds up to integer	Same as round()

#### B) Add <sup>1</sup>/<sub>2</sub> LSB and Truncate 2's Complement

• Example: 64 negative values of xxxxxx rounded to yyyyyy

Pre-rounded value	<b>Rounding action</b>	Net effect
(-) xxxxxx.000000	No change in value	Same as round()
(-) xxxxxx.000001	Rounds down to integer	Same as round()
(-) xxxxxx.000010	Rounds down to integer	Same as round()
(-) xxxxxx.100000	Rounds up to integer	Same as round() +1 🦂
•••	• • •	
(-) xxxxxx.111101	Rounds up to integer	Same as round()
(-) xxxxxx.111110	Rounds up to integer	Same as round()
(-) xxxxxx.111111	Rounds up to integer	Same as round()

- C. Unbiased Rounding: the same as matlab round( $\cdot$ )
  - 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.



- Here is a straightforward algorithm (1. calc w/o 1/2 LSB, 2. check if the result *is* the special case, 3. add 1/2 LSB and recalculate if the result is *not* the special case)
  - 1) *Remove* the normal <sup>1</sup>/<sub>2</sub> LSB rounding bit
  - 2) Keep the output when the **result**(!) is:
    - i. Negative and
    - ii. Of the form xxxxx.1000

# Unrounded result is
# -xxx.5000 (base 10)

- Equivalently, we could also not add the ½ LSB when the result is in the range: xxxxx.0000 to xxxxx.1000
- 3) Otherwise, *add* the ½ LSB rounding bit back into the input and *recalculate the output*
- 4) Truncate as with method (B)





- Here is a second basic algorithm
  - 1) Add the normal <sup>1</sup>/<sub>2</sub> LSB rounding bit
  - 2) Keep the output when the **result**(!) is *not*:
    - i. (Negative and
    - ii. of the form xxxxx.0000)
    - iii. Or zero

# A negative integer

# If input was -0.5

- 3) Otherwise, *remove* the ½ LSB rounding from the input and *recalculate the output*
- 4) Truncate as with method (B)





369

- A third option is to calculate the result two times in parallel:
  - 1) with <sup>1</sup>/<sub>2</sub> LSB added in
  - 2) without <sup>1</sup>/<sub>2</sub> 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

### matlab for previous plots

• copy, paste, and try it out

% rounding.m

% Look at various rounding modes

```
% 2005/02/03 Written
% 2018/03/05 Updated colors and a number of details. Added PrintOn on line 40
% to print plots in .tif files.
```

% The only reason to declare the main body as a function is because matlab % chokes if a function is declared inside a non-function. function rounding

```
%----- Definitions
```

```
= -2.5 : 0.01 : 2.5;
                                  % "continuous" samples
х
              = 0 : 0.01 : 2.5; % "continuous" non-negative samples
xpos
             xcirc5
                                  % circles at x.500 points
                                   % circles at x.000 points
xcircint
                                  % circles at x.500 points (positive)
xcircpos
axis limits
              = [-2.5 2.5 -2.5 2.5]; % for axis limits on plots
                                   % points on axis to draw lines
ticks
              = [-2 -1 0 1 2];
```

```
%----- Main
```

```
plot_one(1, x, round(x), xcirc5, round(xcirc5), ...
axis_limits, ticks, 'round(\cdt)');
plot_one(2, x, fix(x), xcircint, fix(xcircint), ...
axis_limits, ticks, 'fix(\cdot)');
plot_one(3, x, floor(x), xcircint, floor(xcircint), ...
axis_limits, ticks, 'floor(\cdot)');
plot_one(4, x, ceil(x), xcircint, ceil(xcircint), ...
axis_limits, ticks, 'ceil(\cdot)');
plot_one(5, x, floor(x+0.5), xcirc5, floor(xcirc5+0.5), ...
axis_limits, ticks, 'floor(x+0.5)');
plot_one(6, xpos, floor(x+0.5)');
plot_one(6, xpos, floor(x+0.5), xcircpos, floor(xcircpos+0.5), ...
axis_limits, ticks, 'floor(x+0.5), positive inputs only');
return;
%----- Function to plot one figure
```

function plot\_one(fignum, x, y, xi, yi, axis\_limits, ticks, nametitle)

```
PrintOn = 0; % select whether to print plots to .tif and .eps files
```

```
figure(fignum); clf
% plot diagonal dashed line
plot([axis_limits(1) axis_limits(4)], [axis_limits(1) axis_limits(4)], ...
'--g', 'linewidth', 1.5);
hold on; % don't erase prior plots
% plot black vertical y-axis line
plot([0 0], [axis_limits(1) axis_limits(4)], '-k', 'linewidth', 3);
% print horizontal lines
plot(x, y, '.b');
% print large circles
plot(xi, yi, '.r', 'markersize', 30, 'linewidth', 2);
```

```
% touch up the plot
axis(axis_limits);
set(gca, 'XTick', ticks);
set(gca, 'YTick', ticks);
grid on;
grid minor;
xlabel('Input number');
ylabel('Post-rounded number');
title(nametitle);
```

```
% print out .eps and .tiff figures
filename = strcat('rounding', num2str(fignum));
if (PrintOn)
%print('-deps', filename);
print('-dtiff', filename);
end
```