02 – Numerical Representations

Oscar Gustafsson

LINKÖPING UNIVERSITY

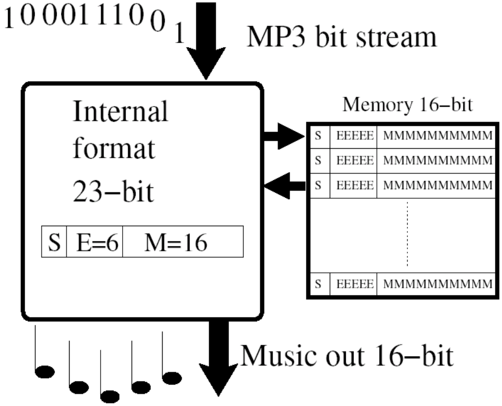
Todays lecture

- Finite length effects, continued from Lecture 1
  - Floating-point (continued from Lecture 1)
  - Rounding
  - Overflow handling

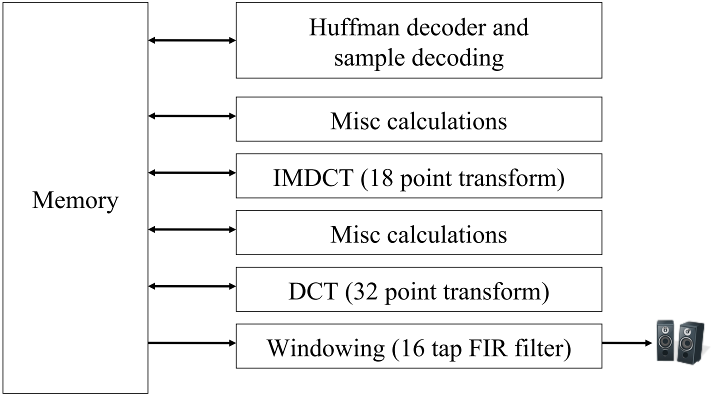
Discussion break

- *If you are designing an application specific processor where you need floating-point numbers, can you reduce the hardware cost by using a custom FP format?*
- For reference: Single precision IEEE-754:
  - 23 bit mantissa (with implicit one)
  - 8 bit exponent
  - 1 sign bit
  - Other features:
    - Rounding (Round to $+\infty$, $-\infty$, 0, and round to nearest even)
    - Subnormal numbers (sometimes called denormalized numbers)

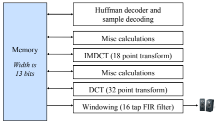
Example: Floating-point Audio Processing



Example: MPEG-1 Layer III (also known as MP3) decoding


Memory
Huffman decoder and sample decoding
Misc calculations
IMDCT (18 point transform)
Misc calculations
DCT (32 point transform)
Windowing (16 tap FIR filter)

Demo (from last lecture): MP3 decoding with 13 bits wide memory


Memory
Width is 13 bits
Huffman decoder and sample decoding
Misc calculations
IMDCT (18 point transform)
Misc calculations
DCT (32 point transform)
Windowing (16 tap FIR filter)

- 13-bit wide memory for intermediate results (16 was used in the actual research, but we are exaggarating the effects using 13 bits)
- Two alternatives
  - Fixed-point data for all intermediate results in memory
  - Floating-point data for all intermediate results in

ASIP floating-point

- May not adhere to IEEE754 features:
  - Number of bits for exponent/mantissa
  - Rounding modes
  - Exception handling
  - Denormalized numbers
  - May use different base (e.g. $(-1)^s \times m \times 16^e$ instead of $(-1)^s \times m \times 2^e$ (A good choice for FPGA based floating-point adders due to the high cost of shifters)

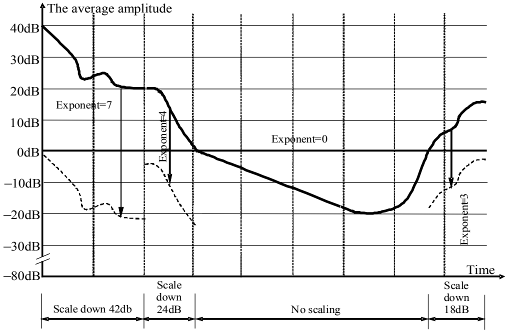
Block floating-point

- A fixed-point processor does not have a floating-point hardware unit
- Floating-point computations can be emulated by fixed-point DSP processors when larger dynamic range is required
    - (But this is slow!)

Block floating-point

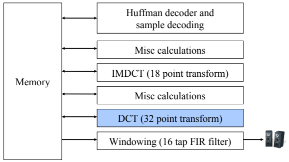
- Block floating-point (also called dynamic signal scaling technique) is an emulation method that is commonly used on fixed-point DSP processors to achieve some of the advantages of floating-point number formats
- Idea: Analyze dynamic range of a block of numbers (Before running an FFT or DCT for example)
  - Scale all values so that overflow is narrowly avoided
  - This will use the available bits as efficiently as possible

Block floating-point



[Liu2008, figure 2.9]

Block floating-point example of a DCT


Memory
Huffman decoder and sample decoding
Misc calculations
IMDCT (18 point transform)
Misc calculations
DCT (32 point transform)
Windowing (16 tap FIR filter)

- First version: 24-bit fixed-point
  - More or less perfect sound
- Second version: 14-bit fixed-point
- Third version: 14-bit fixed-point with dynamic scaling of input values

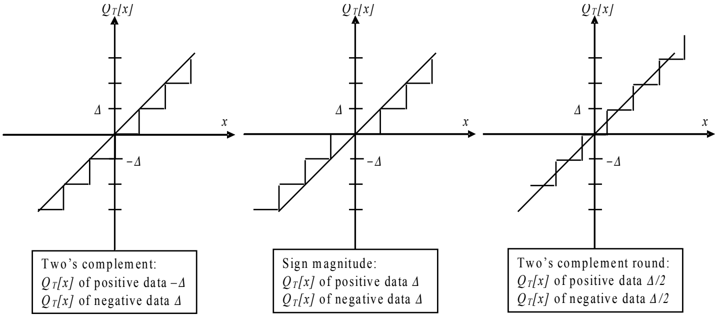
Finite length DSP: The problem

- A challenge to get the best available precision on the results under a precision-limited datapath and storage system – **Focus of the textbook and this course**
  - First problem: finite resolution of A/D converter
  - Second problem: DSP processors are typically fixed-point to minimize silicon cost
  - Third problem: extra quantization error introduced while scaling down signals within a finite data length system

A few definitions

- Definition 1: truncation
  - To convert a longer numerical format to a shorter one by simply cutting off bits at the LSB part
- Definition 2: quantization error
  - The numerical error introduced when a longer numeric format is converted (truncated) to a shorter one

Quantization errors


$Q_T[x]$
Δ
$-\Delta$
x
Two's complement:
$Q_T[x]$ of positive data $-\Delta$
$Q_T[x]$ of negative data Δ
Sign magnitude:
$Q_T[x]$ of positive data Δ
$Q_T[x]$ of negative data Δ
Two's complement round:
$Q_T[x]$ of positive data $\Delta/2$
$Q_T[x]$ of negative data $\Delta/2$

Liu2008 figure 2.11 (with bug in rightmost part corrected!)

Round operation

- Why: To eliminate bias errors after truncation. (Adds approx. $3dB$ to the Signal-to-Noise ratio)
- To round up the truncated part: Use the truncated MSB as the carry in of an add operation
- Hardware implementation 1: Mask the kept part and add to the original
- Hardware implementation 2: MSB of truncated part as the carry in

Round operation

- In certain scenarios you may be concerned with bias caused by rounding
    - Round to nearest even (IEEE-754)

A cautionary tale

- Saudi Arabia, february 25th, 1991
- An incoming Iraqi Scud missile impacts an army barracks killing 28 soldiers
- Cause: A software bug in the fixed-point math

Discussion break

- Why did things go wrong?
- System description:
  - Time is stored as a fixed-point value
  - Time is incremented every 100 ms by 1/10

Discussion break

- Why did things go wrong?
- System description:
  - Time is stored as a fixed-point value
  - Time is incremented every 100 ms by 1/10
  - **Hint:** How do you represent 1/10 in a fixed-point format?

Patriot missile bug

- 1/10 cannot be represented exactly using a binary fixed-point number (nor as a (binary) floating-point number)
  - $1/10$ is $0.0001100110011001100110011\ldots$ in base 2
- After the missile battery had been operational for over 100 hours the time has drifted by ≈ 0.34 s
- An incoming Scud travels at 1.7 km/s, thus the missile battery's tracking was off by over half a kilometer, causing the range gate of the radar to be missed
- For more information, see: `http://sydney.edu.au/engineering/it/~alum/patriot_bug.html`

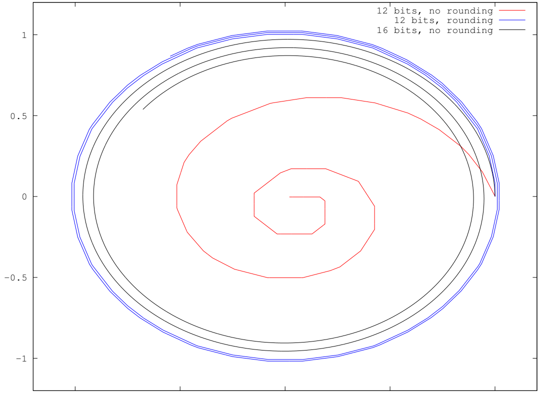
Demonstration of rounding effects

- Most of the time you will gain about half a bit when doing proper rounding
- However, in some cases you will get significantly better results
- Example: Iterated vector rotation

Demonstration of rounding effects: Iterated vector rotation

- $X_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
- $a = 0.003$
- $X_{n+1} = \begin{pmatrix} \cos(a) & -\sin(a) \\ \sin(a) & \cos(a) \end{pmatrix} X_n$
- If you do not round the rotation matrix and X properly you'll get pretty bad results

Demonstration of rounding effects: Iterated vector rotation



Overflow, saturation, and guard bits

- Overflow: if the result of a (fractional) calculation (X) is not in the range $-1 \leq X < 1$
- Common reasons for overflow:
    - When the result is too large (or small)
    - Too many accumulations

Ways to deal with fixed-point overflow

- Ignore it
- Use a floating-point processor instead
- Redo calculation with scaled down input data
    - Tricky for real-time systems
- Exception $\rightarrow$ System restart
- Use guard bits and saturation arithmetic

Ways to deal with fixed-point overflow

- Ignore it
  - May or may not be a good idea. See Ariane 501 for an example where it was a bad idea…
- Use a floating-point processor instead
- Redo calculation with scaled down input data
  - Tricky for real-time systems
- Exception $\rightarrow$ System restart
- Use guard bits and saturation arithmetic

Managing overflow: Saturation/guard

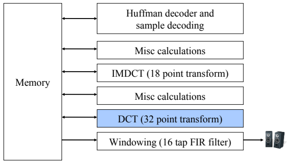
- Most popular way in DSP systems
- What is guard
  - Add more sign extension bits to operands
  - Increasing the range to: $-2^G \leq x < 2^G$

Managing overflow: Saturation/guard

```
if(result >= 1.0) {
    final_result = 0.99999;
} else if(result <= -1.0) {
    final_result = -1.0
} else {
    final_result = result;
}
```

- Performed after an iterative accumulation
  - Do not do it during a convolution
  - Often better than exception for hard-real time system

Managing overflow: Saturation/guard


Memory
Huffman decoder and sample decoding
Misc calculations
IMDCT (18 point transform)
Misc calculations
DCT (32 point transform)
Windowing (16 tap FIR filter)

- Example of overflow vs saturation
    - Same DCT example as for block floating-point

Discussion break

- What is the correct execution order for the following steps?
  - Truncation and saturation
  - Remove guard bits
  - DSP Kernel computations
  - Add guard bits
  - Round
- **Hint:** Think of a simple scalar product where the input is in fractional format and the output should be in fractional format
- $\sum_{i=1}^{N} x_i \times y_i$

Correct execution order

- Add guard bits
- DSP Kernel computations
- Round
- Truncation and saturation
- Remove guard bits

Corner cases

- When verifying a system it makes sense to concentrate on corner cases that exercises the system in unusual ways
- Example: Corner cases for a divider may be MAX_VAL/MAX_VAL, MAX_VAL/1, 1/MAX_VAL, 0/1, 0/MAX_VAL, 1/0, 0/0, MAX_VAL/0, and similar cases

Corner case, fractional multiplication

- Remember, a fractional number can be between -1 and $1 - 2^{n-1}$ (inclusive)
- Do you see any problems with a fractional multiplier which gives a fractional result?
- Expression for fractional multiplication:
  - `tmp[2*n-1:0] = $signed(a)*$signed(b)`
  - `result=tmp[2*n-2:n-1]`

Corner case, fractional multiplication

- $(-1) \times (-1) = 1$ (Answer cannot be represented in fractional!)
- (If not taken into account, the fractional multiplier will produce a -1 in this case.)
- How to handle? Probably best to saturate the result to $0.1111111111\ldots$
- (Do you think it is unlikely that you will get a -1? What about a broken sensor?)

Corner case, absolute operation

- Same problem, what about $|-1|$?
- Without guard/saturation:
  - ABS(1000) = INV(1000)+0001 = 0111+0001 = 1000 (?!)

Corner case, absolute operation

- With guard bits/saturation:
    - ABS(1000) calculated as TRUNC(SAT(ABS(GUARD(1000)))) = TRUNC(SAT(ABS(11000)))
    - TRUNC(SAT(ABS(11000))) = TRUNC(SAT(01000)) = TRUNC(00111) = 0111

Designing a minimal instruction set

- What is the smallest instruction set you can get away with while retaining the capability to execute all possible programs you can encounter?
- (Something to think about over the weekend.)

Oscar Gustafsson

www.liu.se

LINKÖPING
UNIVERSITY