Overview

- Floating and Fixed Point Arithmetic
- System Design Flow
  - Requirements and Specifications (R&S)
  - Algorithmic Development in Matlab and Coding Guidelines
- 2’s Complement Arithmetic
- Floating Point Format
- Qn.m format for Fixed Point Arithmetic
- Addition, Multiplication and Scaling in Qn.m
- LTI systems and implementation in Qn.m format
Floating & Fixed Point Arithmetic

- Two Types of arithmetic
  - Floating Point Arithmetic
    - After each arithmetic operation numbers are normalized
    - Used where precision and dynamic range are important
    - Most algorithms are developed in FP
      - Ease of coding
    - More Cost (Area, Speed, Power)
  - Fixed Point Arithmetic
    - Place of decimal is fixed
    - Simpler HW, low power, less silicon
    - Converting FP simulation to Fixed-point simulation is time consuming
    - Multiplication doubles the number of bits
      - \( N \times N \) multiplier produces \( 2N \) bits
    - The code is less readable, need to worry about overflow and scaling issues

System Design Flow and Fixed

![Diagram showing the system design flow and fixed point implementation process.](image)
System Design Flow

• The requirements and specifications of the application are captured
• The algorithms are then developed in double precision floating point format
  – Matlab or C/C++
• A signal processing system general consists of hybrid target technologies
  – DSPs, FPGAs, ASICs
• For mapping application developed in double precision is partitioned into
  – hardware & software
• Most of signal processing applications are mapped on Fixed-point Digital Signal Processors or HW in ASICs or FPGAs
• The HW and SW components of the application are converted into Fixed Point format for this mapping

Requirements & Specifications

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Power</td>
<td>2W</td>
</tr>
<tr>
<td>Spurious Emission</td>
<td>&lt;60 dB</td>
</tr>
<tr>
<td>Harmonic Suppression</td>
<td>&gt;55 dB</td>
</tr>
<tr>
<td>Frequency Stability</td>
<td>2ppm or better</td>
</tr>
<tr>
<td>Reliability</td>
<td>&gt;10,000 hours MTBF</td>
</tr>
<tr>
<td></td>
<td>&lt;30 min MTTR</td>
</tr>
<tr>
<td>Handheld</td>
<td>12V DC nickel metal hybrid, nickel cadmium or lithium-ion battery pack</td>
</tr>
</tbody>
</table>
R&S of a UHF Radio (cont)

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency Range</td>
<td>420 MHz to 512 MHz</td>
</tr>
<tr>
<td>Data rate</td>
<td>Up to 512 kbps multi-channel non-line of sight</td>
</tr>
<tr>
<td>Channel</td>
<td>Multi-path with 15 μs delay spread and 220 km/h relative speed between transmitter and receiver</td>
</tr>
<tr>
<td>Modulation</td>
<td>OFDM supporting BPSK, QPSK and QAM</td>
</tr>
<tr>
<td>FEC</td>
<td>Turbo codes, convolution, Reed–Solomon</td>
</tr>
<tr>
<td>Frequency hopping</td>
<td>&gt; 600 hops/s, frequency hopping on full hopping band</td>
</tr>
<tr>
<td>Waveforms</td>
<td>Radio works as SDR and should be capable of accepting additional waveforms</td>
</tr>
</tbody>
</table>

Algorithm Development and Mapping

- The R&S related to digital design are forwarded to algorithm developers and system designers
- Algorithms are coded in behavioral modeling tools like Matlab
- The Matlab code is then translated into a high level language, for example, C/C++
- System is designed based on R&S
  - System usually consists of hybrid technologies consisting of ASICs, DSPs, GPP, and FPGAs
- Partitioning of the application into HW/SW parts is performed
- The SW is then developed for the SW part and architectures are designed and implemented for the HW parts
- Integration and testing is performed throughout the design cycle
Guidelines for Matlab Coding

- Signal processing applications are mostly developed in Matlab
- As the Matlab code is to be mapped in HW and SW so adhering to coding guidelines is critical
  - The code must be designed to work for processing of data in chunks
  - The code should be structured in distinct components
    - Well defined interfaces in terms of input and output arguments and internal data storages
  - All variables and constants should be defined in data structures
    - User defined configurations in one structure
    - System design constants in another structure
    - Internal states for each block in another structure
  - Initialization in the start of simulation

Processing in Chunks

```matlab
% BPSK = 1, QPSK = 2, 8PSK = 3, 16QAM = 4
% All user defined parameters are set in structure USER_PARAMS
USER_PARAMS.MOD_SCH = 2; % select QPSK for current simulation
USER_PARAMS.CHUNK_SZ = 256; % set buffer size
USER_PARAMS.NO_CHUNKS = 100; % set no of chunks for simulation

raw_data = randint(1, USER_PARAMS.NO_CHUNKS*USER_PARAMS.CHUNK_SZ)

% generate raw data for simulation
PARAMS = MOD_Params_Init(USER_PARAMS);
STATES = MOD_States_Init(PARAMS);
mod_out = [];
% Code should be structured to process data on chunk-by-chunk basis
for iter = 0:USER_PARAMS.NO_CHUNKS-1
    in_data = raw_data
    [out_sig,STATES] = Modulator(in_data,PARAMS,STATES);
    mod_out = [mod_out out_sig];
end
```
Fixed Point vs. Floating Point HW

- Algorithms are developed in floating point format using tools like Matlab
- Floating point processors and HW are expensive
- Fixed-point processors and HW are used in embedded systems
- After algorithms are designed and tested then they are converted into fixed-point implementation
- The algorithms are ported on Fixed-point processor or application specific hardware

Digital Signal Processors

- Floating Point DSP
- Fixed Point DSP

Precision is critical. These DSPs are more expensive
C67x, C3X, C4X, Tiger Shark
ADI 21000 series.

Can be 16, 24 or 32-bit
C64x, C64X, C55x
ADI 2100 series.
Number Representation

- In a digital design fixed or floating point numbers are represented in binary format
- Types of Representation
  - one's complement
  - sign magnitude
  - canonic sign digit (CSD)
  - two's complement
- In digital system design for fixed point implementation the canonic sign digit (CSD), and two's complement are normally used

2’s Complement

- MSB is the sign bit (has a negative weight)

<table>
<thead>
<tr>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

-8  +0  +2  +1 = -5

<table>
<thead>
<tr>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

0  +4  +2  +0 = 6
Floating Point Format

- Floating point arithmetic is appropriate for high precision applications
- Applications that deals with number with wider dynamic range
- A floating point number is represented as
  \[ x = (-1)^s \times 1 \times m \times 2^{e-b} \]
  - \( s \) represents sign of the number
  - \( m \) is a fraction number >1 and < 2
  - \( e \) is a biased exponent, always positive
  - The bias \( b \) is subtracted to get the actual exponent

IEEE 754 FP Format (single precision)

- Sign: 8 bit
  - 0= +
  - 1= -
- True exponent = e - 127
- Bias is 127
- Mantissa is normalized (MSB=1). There is an implied 1 to make it >1 and <2
- \[ x = (-1)^s \times 1.m \times 2^{e-127} \]
IEEE 754 FP Format

- SEEEEEEEEMMMMMMMMMMMMMMMMMMMM
- If E=255 and M is nonzero, then V=NaN ("Not a number")
- If E=255 and M is zero and S is 1, then V=-∞
- If E=255 and M is zero and S is 0, then V=∞
- If 0<E<255 then V=(-1)^S * 2^(E-127) * (1.M) where "1.M" is intended to represent the binary number created by prefixing M with an implicit leading 1 and a binary point.
- If E=0 and M is nonzero, then V=(-1)^S * 2 ** (-126) * (0.M) These are "unnormalized" values.
- If E=0 and M is zero and S is 1, then V=-0
- If E=0 and M is zero and S is 0, then V=0

Examples

- 0 00000000 000000000000000000000000 = 0
- 1 00000000 000000000000000000000000 = -0
- 0 11111111 000000000000000000000000 = Infinity
- 1 11111111 000000000000000000000000 = -Infinity
- 0 11111111 000001000000000000000000 = NaN
- 1 11111111 001000100010010101010101 = NaN
- 0 10000000 000000000000000000000000 = +1 * 2**(128-127) * 1.0 = 2
- 0 10000001 101000000000000000000000 = +1 * 2**(129-127) * 1.101 = 6.5
- 1 10000001 101000000000000000000000 = -1 * 2**(129-127) * 1.101 = -6.5
- 0 00000001 000000000000000000000000 = +1 * 2**(1-127) * 1.0 = 2**(-126)
- 0 00000000 100000000000000000000000 = +1 * 2**(-126) * 0.1 = 2**(-127)
- 0 00000000 0000000000000000000000001 = +1 * 2**(-126) * 0.000000000000000000000001 = 2**(-149) (Smallest positive value)
IEEE 754 Double Precision

- 64 bit number
- 1 bit for sign
- 11 bits for exponent
- 52 bits for mantissa
- Bias is 1023

FP Addition

- **S0**: Append the implied 1 of the mantissa
- **S1**: Shift the mantissa from S0 with smaller exponent \(e_s\) to the right by \(e_l - e_s\), where \(e_l\) is the larger of the two exponents
- **S2**: For negative operand take two’s complement and then add the two mantissas.
  - If the result is negative, again takes two’s complement of the result for storing in IEEE format
- **S3**: Normalize the sum back to IEEE format by adjusting the mantissa and appropriately changing the value of the exponent \(e_l\)
- **S4**: Round or truncate the resultant mantissa to fit in IEEE format
FP Addition -- Example

• Add these 2 numbers (e=4, m=5, bias=7)

\[0_{1010_00101} = 2^{10-7} \times 1.00101 = 1001.01 = 9.25\]
\[0_{1001_00101} = 2^{9-7} \times 1.00101 = 1.00101 = 4.625\]

Align \[1.00101 \times 2^3\] => \[1.00101 \times 2^3\]
\[1.00101 \times 2^2 \Rightarrow 0.100101 \times 2^3\]
\[1.101111 \times 2^3 = 1.734375 \times 8 = 13.875\]

In FP format \[0_{1010_10111} = 2^{10-7} \times 1.7185 = 13.75\]

FP Multiplication

• **S0**: Add the two exponents e1 and e2 and subtract the bias once

• **S1**: Multiply the mantissas as unsigned numbers to get the product, and XOR the two sign bits to get the sign of the product

• **S2**: Normalize the product if required

• **S3**: Round or truncate the mantissa
Qn.m Format for Fixed-Point Arithmetic

- Qn.m format is a fixed positional number system for representing floating point numbers
- A Qn.m format binary number assumes n bits to the left and m bits to the right of the binary point

<table>
<thead>
<tr>
<th>2⁻¹</th>
<th>2⁰</th>
<th>2⁻²</th>
<th>2⁻³</th>
<th>2⁻⁴</th>
<th>2⁻⁵</th>
<th>2⁻⁶</th>
<th>2⁻⁷</th>
<th>2⁻⁸</th>
</tr>
</thead>
</table>

### Example

- Q₂.₈ Format

<table>
<thead>
<tr>
<th>2⁻¹</th>
<th>2⁰</th>
<th>2⁻²</th>
<th>2⁻³</th>
<th>2⁻⁴</th>
<th>2⁻⁵</th>
<th>2⁻⁶</th>
<th>2⁻⁷</th>
<th>2⁻⁸</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>01 1101.0000</th>
</tr>
</thead>
</table>

1 + 1/2¹ + 1/2² + 1/2⁴ = 1.8125

The range of a Q₂.₈ is -2 (10_000_000) to +1.99609375 (01_1111_1111)

In Qn.m n determines the range of the integer, while m determines the precision of the fractional part
Fixed Point DSPs

- Commercially available off the shelf processors usually have 16 bits to represent a number
- In C, a programmer can define 8, 16 or 32 bit numbers as `char`, `short` and `int/long` respectively
- In case a variable requires different precision than what can be defined in C, the number is defined in higher precision
- For example an 18-bit number should be defined as a 32-bits integer
- High precision arithmetic is performed using low precision arithmetic operations
  - 32-bit addition requires two 16-bit addition
  - 32-bit multiplication requires four 16-bit multiplications

FP to Fixed Point Conversion

- Start with the algorithm (FP)
- Estimate the range
- Determine \( n \)
- The fractional part \( m \) requires detailed analysis of signal to quantization noise
Addition in Q Format

- Adding two numbers $Qn_1.m_1$ and $Qn_2.m_2$ results in $Qn.m$ where
- $N=\max(n_1,n_2)+1$, $m=\max(m_1,m_2)$
- Note that the decimal point does not exist in H/W, the designer must align the two number properly
Multiplication in Q Format

- Unsigned by unsigned
- Straightforward shift and add, no sign extension in general $Q(n_1+n_2).(m_1+m_2)$

\[
\begin{align*}
1 & 1 & 0 & 1 = 11.01 \text{ in } Q2.2 = 3.25 \\
1 & 0 & 1 & 1 = 10.11 \text{ in } Q2.2 = 2.75 \\

\end{align*}
\]

Sign extended number

- Signed by Unsigned

\[
\begin{align*}
1 & 1 & 0 & 1 = 11.01 \text{ in } Q2.2 = -0.75 \text{ (signed)} \\
0 & 1 & 0 & 1 = 01.01 \text{ in } Q2.2 = 1.25 \text{ (unsigned)} \\

\end{align*}
\]
Multiplication in Q Format

• Signed by Unsigned

\[
\begin{align*}
1101 &= 11.01 \text{ in Q2.2} = -0.75 \text{ (signed)} \\
1101 &= 01.01 \text{ in Q2.2} = 3.25 \text{ (unsigned)} \\
\quad \downarrow \\
11111101 \\
0000000 \\
111101 \\
11101 \\
11011001 = 1101.1001 \text{ in Q4.4 format} = \quad \text{Take 2's complement} \\
\quad \downarrow \text{Take 2's complement} \\
00100111 = -2.4375
\end{align*}
\]

• Unsigned by Signed

\[
\begin{align*}
1001 &= 11.01 \text{ in Q2.2} = 2.25 \text{ (unsigned)} \\
1101 &= 01.01 \text{ in Q2.2} = -0.75 \text{ (signed)} \\
\quad \downarrow \text{The 2's complement of the multiplicand 01001=10111} \\
00001001 \\
0000000 \\
001001 \\
10111 \\
11001101 = 1110.0101 \text{ in Q4.4 format take 2's complement it become} \\
0001.1011 = -1.6875
\end{align*}
\]
Multiplication in Q Format

- Signed by Signed

1 1 0 1 = 11.01 in Q2.2 = -0.75 (signed)
1 1 0 1 = 1.101 in Q1.3 = -0.375 (signed)

The 2's complement of 11.01 since we are multiplying it by the sign bit

10 0 0 0 0 1 0 0 1 = 00.01001 in Q2.5 format = 0.28125

Note that there is an extra sign bit so we need 2.5 not 3.5 format