Digital Signal processing

Lecture notes • SysNum-2

Robin Gerzaguet
Associate Professor @ENSSAT
robin.gerzaguet@enssat.fr

November 9, 2018
Objectives

From signal processing specifications

- Digital filter analysis
- Digital Signal processing transform
- Infinite Impulse Response (IIR) filters
- Finite Impulse Response (FIR) filters
- Spectral analysis
- Multirate processing

To practical implementation

- Fixed point design
- Digital Signal Processor (DSP)

Digital Signal processing versus Processing of digital signal!
Objectives

DSP in a nutshell
From Signal processing tools
  ■ Filters, transforms, analysis
To DSP applications
  ■ Fixed point for real time efficient implementations
  ■ Signal processing architectures (DS)

Operating methods
  ■ CM (20h), TD (16h)
  ■ TP (9h)
  ■ And DSP module (TP+project)
Gratitude

These slides and the lecture structure have been massively inherited from the work of several (awesome) persons:

- Olivier Senteys: people.rennes.inria.fr/Olivier.Senteys/
- Daniel Menard: http://dmenard.perso.insa-rennes.fr/
- Olivier Berder: https://www-granit.irisa.fr/

Sources can be found at my personnel page:
https://perso.univ-rennes1.fr/robin.gerzaguet/
Objectives

1. Introduction
   - Digital signal processing in a nutshell
   - Main applications
   - Analog Digital conversion
   - DSP algorithm implementation

2. Digital filtering analysis
   - Z transform
   - Parallel and successive structures
   - Filtering specification
   - FIR filtering
   - IIR filtering
Objectives

3 Fixed point arithmetic

- Introduction
- Numbers representation: coding family
- Fixed and floating points
  - Fixed point
  - Floating point
- Fixed point arithmetic
  - Addition
  - Multiplication
- Fixed point design
- Dynamic
  - Principle
  - Simulation methods
  - Analytical methods
- Point positionning
  - Dynamic
  - Rules
  - Conclusion
- Data width
  - Format
  - Word charac.
  - Quantization noise
- Conclusion
Objectives

4 IIR synthesis
- Introduction
- Filter prototype functions
  - Butterworth filters
  - Chebyshev filters
  - Elliptic filters
  - Bessel filters
- Transformation functions
  - Impulse Invariance method
  - Euler transformation
  - Bilinear transform
- Conclusion

5 FIR synthesis
- Introduction
- Linear phase filters
- Window based synthesis
  - Principle
  - Apodisation functions
  - Windows comparison
  - Window selection
- Frequency sampling method
- Conclusion
Objectives

6 Fast Fourier Transform
- Introduction
- Principle of the FFT
- Time split FFT
- Frequency split FFT
- Conclusion

7 Spectral analysis
- Introduction and principle
- Truncation and FFT
- Time-frequency grid
- Zero padding
- Influence of apodisation function
Objectives

8 Multirate processing
- Introduction
- Downsampling
- Upsampling
- Fractional interpolation

9 Digital Signal processor
- Introduction
- DSP core units
  - Processing units
  - Memory units
  - Control unit
- Some DSPs
- Conclusion on DSP
Introduction
Introduction

1. Introduction
   - Digital signal processing in a nutshell
   - Main applications
   - Analog Digital conversion
   - DSP algorithm implementation

2. Digital filtering analysis

3. Fixed point arithmetic

4. IIR synthesis

5. FIR synthesis

6. Fast Fourier Transform

7. Spectral analysis
Digital Signal processing?

Figure: Digital and analog domains
DSP in a global chain

From sensors to actuators [1]

DSP in a global chain

Example with a smooth filtering of sensor capture

1. Raw sensor measure (high variations)
2. After blocker in ADC chain
3. Digital samples
4. Treated digital samples can be used for additional post-processing
5. After converted back in analog domain
6. Smooth output from sensors

Possible to do the processing directly in analog domain.

- More flexibility to do in digital domain
- Additional processing and re-use
- More complex to design
Digital versus analog processing

Advantages

- **No drift**: processing independent from temperature, aging
- **Precision**: depending on bit range
- **Softness**: several possible tasks and reconfiguration
- **Prediction**: same behaviour as in simulation
- **Prototyping**: Software/firmware updates
- **Performances**
- **Integration**: in systems (SoC, DSP, VLSI)

Drawbacks

- **High cost** for simple systems
- **Required computational speed** is function of system bandwidth (can be very high!)
- **Complexity**: Hardware, software, interfaces
- Need a specific methodology to design a DSP system
Digital versus analog processing

Advantages

- **No drift**: processing independent from temperature, aging
- **Precision**: depending on bit range
- **Softness**: several possible tasks and reconfiguration
- **Prediction**: same behaviour as in simulation
- **Prototyping**: Software/firmware updates
- **Performances**
- **Integration** in systems (SoC, DSP, VLSI)

Drawbacks

- **High cost** for simple systems
- Required computational **speed** is function of system bandwidth (can be very high!)
- **Complexity**: Hardware, software, interfaces

Need a specific methodology to design a DSP system
DSP as a specific methodology

From a need (and spec) to a design (and product)

- Specification
- Operating spec
- Technological spec
- Fonctionnal spec
- Real time processing constraints
- Energy consumption
- Flexibility
- Cost

Need

Algorithm definition
Fonctionnal conception

Simulation
Algorithm description

Performances

Partionning

Hardware implementation
Software implementation

Implantation
DSP as a specific methodology

From a need to specification

- What are the requirements?
- What are the key performance indicators (what should I do and how should I know what to do?)
- Ensure that the system is OK (simulation)
DSP as a specific methodology

From a need to specification

- What are the requirements?
- What are the key performance indicators (what should I do and how should I know what to do?)
- Ensure that the system is OK (simulation)

To a real implementation

- Taking into account all specifications (technological, functional and operating)
  - Operating: linked to how the system shall be functional (size, temperature, ...)
  - Functional: Linked to expected behaviour of the system (checked with simulations)
- Choosing the architecture and its consequences in terms of design flow
- Taking into account the constraints
For what applications?

Digital Signal processing is everywhere

At home
- Broadcast TV, Set top Box, video games, VR, networks
- DVD, HDTV, CD, DAB, DVD, WiFi, ...

At work
- Videoconference (Webex), VoIP, phones, internet
- High speed networks (Fiber, ADSL, ...), smart buildings, IA, ...

In mobility
- Mobile connectivity, smartphones, smartwatch, ...
- GPS, Radar, cellular coverage (3G, 4G-LTE, 5G-NR)

Digital signal processing is everywhere, leveraging new usages and markets
Some DSP applications

Different processing on different signals

- Signal from sensors
  - Associated to physical measure (temperature, pressure, ...)

- Biomedical signals
  - EEG, IRM, scanner, ...

- Data signals
  - Financial data (trading), statistics

- Speech processing

- Image processing

- ...

Processing must be *adapted* to the signal.
Some DSP applications

Example of speech processing

- From physics, deduce a parametric model
  - Power is generated with lung and trachea
  - Which leads to vibrations of vocal cords
  - Combined and modulated through vocal pipe and tongue (articulatory apparatus)

Figure: Converting physics into diagram flow

- Nasal cavity
- Vocal pipe
- Larynx
- Tongue
Example of speech processing

Figure: Time and frequency component. Up: Vowel. Down: consonant
Example of speech processing

Signal in time domain

- Sparsity and statistical properties
- Intra and Inter user variability (speech signature)
- Speech masking

Signal in frequency domain

- Harmonic analysis
- Frequency masking
Example of phone strike detection

- Each key as a specific frequency
- A filter bank is used to extract desired frequency

![Diagram of frequency detection system]

Frequencies of the digital keyboard:
- 697 Hz
- 770 Hz
- 852 Hz
- 941 Hz
- 1209 Hz
- 1336 Hz
- 1477 Hz

Digital filter mask:
- |G| (dB) values:
  - -30 dB
  - -3 dB
  - 0 dB

Changes in frequency detection due to different filters and detectors:
- BP Filter
- HP Filter
- Limiter
- Detector

Frequencies:
- 697 Hz
- 770 Hz
- 852 Hz
- 941 Hz
- 1209 Hz
- 1336 Hz
- 1477 Hz
Example of echo cancelling

Cancelling of unwanted echo

- Crosstalk in some (old) communications systems
- Webex and conference call with echo
- Unwanted mother signal in echography

Signal processing and intelligence (adaptive processing) to cancel the unwanted contribution [2]

Figure: Noise cancelling principle apply to jet engine. [3]

Example of echo cancelling

Signal processing and intelligence (adaptive processing) to cancel the unwanted contribution [2]

Figure: Noise cancelling principle apply to earphones. [4]

Example of MP3 coding

MP3 coding

- A code that uses various rate
  - Non uniform coding parametrized by output control

- Based on psycho-acoustic linked to ear physic behaviour
  - Perceptual model
  - Take into account masking phenomena

- Processing done in frequency domain
  - Acoustic bandwidth is separated into sub-bandwidth
  - Processing (and threshold) depends on frequencies
Example of MP3 coding

Figure: Synoptic of MP3 coding
Example of MP3 coding

\[ X(m) = \sum_{k=0}^{n-1} f(k)x(k)\cos\left[\frac{\pi}{2n}(2k + 1 + \frac{n}{2})(2m + 1)\right], \quad m = 0, \ldots, \frac{n}{2} - 1 \]

**Modified Discrete Cosine Transform (MDCT)**

**Figure:** Synoptic of MP3 coding - Discrete Cosine Transform stage
Example of MP3 coding

Frequency processing

Data is converted in frequency domain (FFT) in order to calculate energy in subdband

- Psycho-acoustic model in frequency domain
- If loud noise in one band, other band can be ignored
Analog to Digital conversion
Analog to Digital conversion

First stage that model analog to digital transformation
- Analog input
- Digital output
We call this a mixed signal device

Quantization
Input signal is continuous and output signal is discrete
- Only $N$ level as the output
- Lead to a continuous to discrete conversion error: quantization error
- Can also be modeled as a additive noise (quantization noise)
ADC architecture

Basic ADC architecture

- Input is between 0V and $V_{\text{ref}}$
- Conversion on 3 bits: 8 levels

<table>
<thead>
<tr>
<th>Observed voltage</th>
<th>Output level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 1 V</td>
<td>0 0 0</td>
</tr>
<tr>
<td>1 - 2 V</td>
<td>0 0 1</td>
</tr>
<tr>
<td>2 - 3 V</td>
<td>0 1 0</td>
</tr>
<tr>
<td>3 - 4 V</td>
<td>0 1 1</td>
</tr>
<tr>
<td>4 - 5 V</td>
<td>1 0 0</td>
</tr>
<tr>
<td>5 - 6 V</td>
<td>1 0 1</td>
</tr>
<tr>
<td>6 - 7 V</td>
<td>1 1 0</td>
</tr>
<tr>
<td>7 - 8 V</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>
ADC architecture

Output is a digital word

- Composed of \( n \) bits associated to an analog range (uniform quantization)
- Some bits carry more “information” than other

**Least Significant bit (LSB) and Most Significant bit (MSB)**

We have \( \{b_0, b_1, b_2\} \)

- An error in \( b_0 \) leads to an error of 1 V
- An error in \( b_2 \) leads to an error of 4 V

See quantization noise in next lectures.
Samples and hold ADC

Most famous (and easy to understand) topology, architecture relies on:
- Sampling time $T_c$
- ADC resolution (i.e., numbers of outputs bits)

ADC samples (captures, takes) the voltage of an analog signal and holds its value at a constant level for a specified minimum period of time ($T_c$)

![Sample and hold architecture](image)

**Figure:** Sample and hold architecture [5]

- Control signal pilot the holding during $T_c$

Samples and hold ADC

Figure: Sampling and Hold behaviour

- Rate at a given sampling frequency (dash lines)
- Signal at this time is hold (red point)
- An output is given for each interval (purple curve)
Sigma-Delta ADC

- In ADC mode, convert a high sampling frequency low bit resolution to a low sampling frequency high bit rate.
- Based on 1-Bit ADC quantization stage.
- Topology is half Digital // half analog.
- Advantage: the digital filter reduces noise impact.

**Figure:** Sigma-Delta ADC topology [6]

Sigma-Delta ADC

Sigma-Delta modulator topology

A comparator (analog)
An integrator (analog)
A 1-bit ADC (output is filter input)
A 1-bit DAC retro-action

Application: Input signal is 1V
Sigma-Delta ADC

If input is a constant 1V signal

- $x(t) = 1, \forall \ t$
- Order I integrator
- 1 bit ADC/DAC piloted by $V_{cc}$

**Figure:** Scheme of a sigma-delta modulator

**Figure:** Output of sigma-delta ADC for a constant 1V input
Sigma-Delta ADC

- Retro-action aims to keep the ADC stage to a 1-bit
- Filtering stage convert the raw sampled data to the expected one

**FIGURE 1.12** Illustration of the PDM output streams of a 1st-order \( \Sigma \Delta M \): (a) Output for a ramp input using 1-bit internal quantizer, (b) Output for a sinusoidal input using 3-bit internal quantizer.

**Figure:** Output of A-bit ADC stage when a sine wave is at the input [7]

Sigma-Delta ADC

Integrator aims to modify noise distribution

- Noise is pushed to high frequency
- Noise distribution is non equivariant
- Impact of digital filter afterwards (to reject noise)

**Figure:** Impact of integrator in noise distribution
Key design space is the digital filter

- Often sinc or sinc$^3$ shape filters
- Filter built to allow flat response in useful band
- Care design due to notch frequencies

---

**Figure:** Digital filter used in sigma-delta ADC [8]

---

SAR - Successive Approximation Register ADC

- The input voltage is successively identified with a register that moves its binary values.
- For $N$ bits identification, SAR should be $N$ times faster.
- DAC for analog comparison between input and output.
- After convergence, output binary sequence is sent to digital stage.
  - End of Conversion (EOC).

**Figure:** SAR scheme
SAR - ADC ● Algorithm

- for i = 1 : N
  - $b_{N-i} = 1$;
  - $W = \text{DAC}\{ b(\cdot) \}$;
  - $e = V - W$;
  - if $e > 0$
    - $b_{N-i} = 0$;
  - else
    - $b_{N-i} = 1$;
- end
- end

Figure: SAR scheme
SAR ADC

If input is a constant 1V signal

- \( x(t) = 1, \forall t \)
- 8 bits
- EOC computed at each iteration

Figure: Scheme of a sigma-delta modulator

Figure: Output of SAR ADC for a constant 1V output
ADC comparison

Different technology for different purpose

Sample and Hold

✓ Simple architecture
ADC comparison

Different technology for different purpose

Sample and Hold

- Simple architecture
- Analog architecture that takes into account good properties of MOS technology

Sampling moment distortion (due to clock raising failing time issue)

Errors due to MOS transistor switches (MOS switch charge injection)
ADC comparison

Different technology for different purpose

Sample and Hold

- Simple architecture
- Analog architecture that takes into account good properties of MOS technology

× Sampling moment distortion (due to clock raising failing time issue)
ADC comparison

Different technology for different purpose

Sample and Hold

- ✔ Simple architecture
- ✔ Analog architecture that takes into account good properties of MOS technology
- ✗ Sampling moment distortion (due to clock raising failing time issue)
- ✗ Errors due to MOS transistor switches (MOS switch charge injection)
ADC comparison

Different technology for different purpose

**Sigma-Delta architecture**

- Massive digital architecture: low power, easy to scale, not temperature dependent
ADC comparison

Different technology for different purpose

**Sigma-Delta architecture**

- Massive digital architecture: low power, easy to scale, not temperature dependent
- Possible very good precision only dependent on speed and processing
ADC comparison

Different technology for different purpose

**Sigma-Delta architecture**

- ✔ Massive digital architecture: low power, easy to scale, not temperature dependent
- ✔ Possible very good precision only dependent on speed and processing
- ✔ Flexibility ensured with digital post-filtering (no need strong anti aliasing filters)
Different technology for different purpose

**Sigma-Delta architecture**

- ✔ Massive digital architecture: low power, easy to scale, not temperature dependent
- ✔ Possible very good precision only dependent on speed and processing
- ✔ Flexibility ensured with digital post-filtering (no need strong anti aliasing filters)
- ✗ Required speed for high bit precision: in practice limited bandwidth
ADC comparison

Different technology for different purpose

**Sigma-Delta architecture**

- ✔ Massive digital architecture: low power, easy to scale, not temperature dependent
- ✔ Possible very good precision only dependent on speed and processing
- ✔ Flexibility ensured with digital post-filtering (no need strong anti aliasing filters)
- ✗ Required speed for high bit precision: in practice limited bandwidth
- ✗ Latency induced by digital filtering stage
Different technology for different purpose

**SAR architecture**

- Massive digital architecture: low power, easy to scale
ADC comparison

Different technology for different purpose

SAR architecture

✅ Massive digital architecture: low power, easy to scale
✅ One clock to rule them all (versus pipeline architectures)
ADC comparison

Different technology for different purpose

SAR architecture

- Massive digital architecture: low power, easy to scale
- One clock to rule them all (versus pipeline architectures)
- More bandwidth than Sigma-delta
ADC comparison

Different technology for different purpose

**SAR architecture**

- ✔ Massive digital architecture: low power, easy to scale
- ✔ One clock to rule them all (versus pipeline architectures)
- ✔ More bandwidth than Sigma-delta
- ✗ Calibration and DAC architecture
ADC comparison

ADC Technologies - $\Delta \Sigma$

**Advantages**
- High Resolution
- High Stability (averages and filters out noise)
- Low Power
- Low cost

**Disadvantages**
- Cycle-Latency
- Low Speed

$\Delta \Sigma$ – Delta Sigma
Or Sigma Delta
(Oversampling)

SAR
Successive Approximation

Pipeline

Figure: ADC architecture-use cases adequation from Texas Instrument [9]
ADC comparison

ADC Technologies - SAR

Figure: ADC architecture-use cases adequation from Texas Instrument [9]

ADC comparison

ADC Technologies - pipeline

Figure: ADC architecture-use cases adequation from Texas Instrument [9]

### Which ADC Architecture to Use??

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Pipelined</th>
<th>SAR</th>
<th>Delta Sigma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput (samples/sec)</td>
<td>++</td>
<td>+</td>
<td>0/+</td>
</tr>
<tr>
<td>Resolution (ENOB)</td>
<td>0</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>Latency (Sample-to-Output)</td>
<td>+</td>
<td>++</td>
<td>0</td>
</tr>
<tr>
<td>Suitability for converting Multiple Signals per ADC</td>
<td>+</td>
<td>++</td>
<td>0</td>
</tr>
<tr>
<td>Capability to convert non-periodic multiplexed signals</td>
<td>+</td>
<td>++</td>
<td>-</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>Scales with Sample Rate or Constant</td>
<td>Scales with Sample Rate</td>
<td>Constant</td>
</tr>
</tbody>
</table>

**Figure:** ADC architecture-use cases adequation from Texas Instrument [9]

---

DSP algorithm implementation

Signal is now digital
- Assume that digitalization went OK
- Consequence of digital conversion in next sections
  - Fixed point implementation
  - Quantization noise impact

DSP implementation
- Data is ready to be processed
- Processing depends on chosen architecture
- But common rules on how we can do the processing
DSP algorithm implementation

FIR filtering

- Theory in next lecture
- Consider that output is based on input processing (convolution)

Difference equation

\[ y[n] = \sum_{k=0}^{L-1} h[k]x[n - k] \]

- \( L \) is the number of taps (depth of the filter)
- \( n \) is the sampled time index \( (x[n] = x(nT_c)) \)
- \( h[k] \) the filter tap at delay \( k \)
  - Causal filter \( (k \geq 0) \)
  - Finite duration filter \( h[k] = 0 \) if \( k \geq L \)
DSP algorithm implementation

From difference equation to flowgraph

\[ y[n] = \sum_{k=0}^{L-1} h[k]x[n - k] \]

**Figure:** FIR flowgraph

- Equivalence between equation and flowgraph
- Implementation description
DSP algorithm implementation

Some typical examples

Filtering

\[ y[n] = y[n] + x[n - k] \times h[k] \quad \text{MAC: Multiplication and accumulation} \]
DSP algorithm implementation

Some typical examples

Filtering

\[ y[n] = y[n] + x[n - k] \times h[k] \quad \text{MAC: Multiplication and accumulation} \]

Adaptive form

\[ y[n] = y[n - 1] + x[n - k] \times h[k] \quad \text{MAD: Multiplication and addition} \]
DSP algorithm implementation

Some typical examples

Filtering

\[ y[n] = y[n] + x[n - k] \times h[k] \quad \text{MAC: Multiplication and accumulation} \]

Adaptive form

\[ y[n] = y[n - 1] + x[n - k] \times h[k] \quad \text{MAD: Multiplication and addition} \]

Complex arithmetic

\[ x[n - k] = \{x_i[n - k], x_q[n - k]\}, \ h[k] = \{h_i[k], h_q[k]\} \]
\[ y_i[n] = x_i[n - k] \times h_i[k] - x_q[n - k] \times h_q[k] \]
\[ y_i[n] = x_i[n - k] \times h_q[k] + x_q[n - k] \times h_i[k] \]
Some properties to keep in mind

- Signal processing needs heavy calculation
  - lot of MAC/MAD
  - Complex multiplications in many fields (FFT, ...)
DSP algorithm implementation

Some properties to keep in mind

- Signal processing needs heavy calculation
  - lot of MAC/MAD
  - Complex multiplications in many fields (FFT, ...)

- Quantization and digital issues
  - Full scale and fixed point analysis
  - Noise and distortions
Some properties to keep in mind

- Signal processing needs heavy calculation
  - lot of MAC/MAD
  - Complex multiplications in many fields (FFT,...)

- Quantization and digital issues
  - Full scale and fixed point analysis
  - Noise and distortions

- Data storage and manipulation
  - Linear or indexed access
  - LUT
  - Data collection and ageing
DSP algorithm implementation

Some properties to keep in mind

Digital signal processing core is **real time** processing

- Execution time linked to input sampling rate \( T_I \) and output sampling rates \( T_O \)
  - Same rate between I and O: \( T_I = T_O \)
  - Dowsampling rates \( T_I > T_O \)
  - Upsampling rates \( T_I < T_O \)

- Effective processing done in \( T_{ex} \)
  - \( T_{ex} < T_O \)

Digital signal processing can means **lot** of data

- High computational load
- Extensive use of memory and I/O (DMA)
Real time processing

Online processing

- One output per input
- Processing time must be lower than output sampling time
- 3 steps
  1. Acquisition
  2. Processing
  3. Restitution

\[
\begin{align*}
x[n] & \quad \text{Input, output()} \\
\text{DSP} & \quad \text{idle} \\
y[n] & \quad \text{Restitution} \\
\end{align*}
\]
Online processing: C code

```c
int main() {
    float x[N], y, acc;
    int i;
    x[0] = Read.Input.Sample();
    acc = x[0]*h[0];
    for (i=N-1; i>0; i--) {
        acc = acc + x[i]*h[i];
        x[i] = x[i-1];
    }
    y = acc;
    Write.Output.Sample(y);
}
```
Real time processing

Block processing

- processing done on several consecutive inputs
- 3 steps
  1. acquisition and storing
  2. processing
  3. restitution

\[ N.T_e > t_{exec} + N.t_{acq} \]
**Real time processing**

**Block processing**

Block is associated to samples selected by a window

1. Window can be consecutive (IFFT processing in digital communications)
2. Window can overlap (Welch spectral analysis)
3. Window can be disjointed
int main() { /* Filter with L taps, block size M */
    float x[L+M], y[M], acc;
    int i, j; x[0] = x[1];
    /* Getting samples and store it */
    for (j=M-1; j>=0; j--) {
        x[j] = Read.Input.Sample();
    }
    /* Apply filter after filling buffer */
    for (j=0; j<M; j++) {
        acc = 0;
        for (i=0; i<L-1; i++) {
            acc = acc + x[i+j]*h[i];
        }
        y[j] = acc;
    }
}
Processing rate

Device and application parameters:
- Number of parallel path that can be processed $N_p$
- Elementary number of DSP units $N_e$
- Sampling period $T_e$

Processing rate of the device

$$P_p = \frac{N_p \times N_e}{T_e}$$
Digital filtering
Digital filtering

What is digital filtering?

- A modification of the time frequency components of the input signal
- Time and frequency are linked (dual), so define transformation in one domain is enough

We consider a digital signal sampled at sampling time $T_c$ defined as

$$x[n] = x(nTe)$$

- Filtering affects all the samples $x[n]$ (but not only $x$ !)
Filtering

Introducing Z transform

- Filter transformation is expressed with the Z transform
- Filter and signal can be derived with the Z transform

\[ x[n] \rightarrow h \rightarrow y[n] \]

Filtre numérique

\[ x(n) \leftrightarrow X(z) \quad h(n) \leftrightarrow H(z) \quad y(n) \leftrightarrow Y(z) \]
Filtering

General framework of digital filtering

Filters affects input and output of signal

- 2 parts in the digital filtering: one affects $x$ and one $y$
- We will focus here on causal filters with depth $L$
- Filtering is a bilinear form
Filtering

General framework of digital filtering

Filters affect the input and output of signal

- 2 parts in the digital filtering: one affects $x$ and one $y$
- We will focus here on causal filters with depth $L$
- Filtering is a bilinear form

$$y[n] = c + \sum_{k=0}^{L-1} b_k x[n-k] + \sum_{k=1}^{L-1} a_k y[n-k]$$

- Coefficients $b_k$ affect the input
- Coefficients $a_k$ affect the output
- $c$ is a constant offset ($c = 0$ in most cases, dropped for the rest)
Filtering

Filtering and Z transform

\[ y[n] = \sum_{k=0}^{L-1} b_k x[n - k] + \sum_{k=1}^{L-1} a_k y[n - k] \]

By rearranging the previous equation, we can link Z transforms

\[
\begin{cases}
  x[n] \rightarrow X(z) \\
  y[n] \rightarrow Y(z) \\
  h[n] \rightarrow H(z)
\end{cases}
\]

We define the filtering operation in Z domain as

\[ Y(z) = X(z) \times H(z) \]
Filtering and Z transform

\[ y[n] = \sum_{k=0}^{L-1} b_k x[n - k] + \sum_{k=1}^{L-1} a_k y[n - k] \]
Filtering and Z transform

\[ y[n] = \sum_{k=0}^{L-1} b_k x[n-k] + \sum_{k=1}^{L-1} a_k y[n-k] \]

We have

\[ Y(z) = \left( \sum_{k=0}^{L-1} b_k z^{-k} \right) X(z) + \left( \sum_{k=1}^{L-1} a_k z^{-k} \right) Y(z) \]  \hspace{1cm} (1)
Filtering and Z transform

\[ y[n] = \sum_{k=0}^{L-1} b_k x[n - k] + \sum_{k=1}^{L-1} a_k y[n - k] \]

We have

\[ Y(z) = \left( \sum_{k=0}^{L-1} b_k z^{-k} \right) X(z) + \left( \sum_{k=1}^{L-1} a_k z^{-k} \right) Y(z) \]  \hspace{1cm} (1)

\[ \left( \sum_{k=0}^{L-1} b_k z^{-k} \right) X(z) = \left( 1 - \sum_{k=1}^{L-1} a_k z^{-k} \right) Y(z) \]  \hspace{1cm} (2)

\[ N(z) \quad D(z) \]
Filtering and Z transform

\[
\left( \sum_{k=0}^{L-1} b_k z^{-k} \right) X(z) = \left( 1 - \sum_{k=1}^{L-1} a_k z^{-k} \right) Y(z)
\]

(3)
Filtering

Filtering and Z transform

\[
\left( \sum_{k=0}^{L-1} b_k z^{-k} \right) X(z) = \left( \sum_{k=1}^{L-1} a_k z^{-k} \right) Y(z)
\]

By recalling \( H(z) = \frac{Y(z)}{X(z)} \)

\[
\left( \sum_{k=0}^{L-1} b_k z^{-k} \right) X(z) = \left( 1 - \sum_{k=1}^{L-1} a_k z^{-k} \right) Y(z)
\]

\[
H(z) = \frac{\sum_{k=0}^{L-1} b_k z^{-k}}{1 - \sum_{k=1}^{L-1} a_k z^{-k}} = \frac{N(z)}{D(z)}
\]
Poles and Zeros

Filter can be expressed with a numerator and a denominator

\[
\begin{align*}
N(z) &= \sum_{k=0}^{L-1} b_k z^{-k} \\
D(z) &= 1 - \sum_{k=1}^{L-1} a_k z^{-k}
\end{align*}
\]

Note that we can have \( D(z) = \sum_{k=0}^{L-1} a_k z^{-k} \) with \( a_0 = 1 \).

We can express the 2 polynomials as

\[
\begin{align*}
N(z) &= b_0 \times \prod_{k=1}^{N_z} (z - z_k) \\
D(z) &= \prod_{k=1}^{N_p} (z - p_k)
\end{align*}
\]

- \( N(z) \) is the numerator
  - It has \( N_z \) zeros
- \( D(z) \) is the denominator
  - It has \( N_p \) poles
Poles and Zeros

Filter can be expressed with a numerator and a denominator

\[
\begin{align*}
N(z) &= \sum_{k=0}^{L-1} b_k z^{-k} \\
D(z) &= 1 - \sum_{k=1}^{L-1} a_k z^{-k}
\end{align*}
\]

The filter can be expressed in Z transform as

\[
H(z) = \frac{\sum_{k=0}^{L-1} b_k z^{-k}}{1 - \sum_{k=1}^{L-1} a_k z^{-k}}
\]

\[
H(z) = b_0 \times \frac{\prod_{k=1}^{N_z} (z - z_i)}{\prod_{k=1}^{N_p} (z - p_i)}
\]

Reminder

- Stability is ensured with \(|p_i| < 1\)
- Minimal phase filter\(^1\): zeros belongs to the unit circle

---

\(^1\) A filter is said to be minimum-phase if the system and its inverse are causal and stable.
Some filters can be split into parallel architecture or cascade architecture.

- The global $Z$ transform is function of subindexed filters $H_i$.

\[
H_{\text{parallel}} = \sum_{i=1}^{M} H_i(z) = \sum_{i=1}^{M} \frac{\sum_{k=0}^{L-1} b^i_k z^{-k}}{1 - \sum_{k=1}^{L-1} a^i_k z^{-k}}
\]

\[
H_{\text{cascade}} = \prod_{i=1}^{M} H_i(z) = \prod_{i=1}^{M} \frac{\sum_{k=0}^{L-1} b^i_k z^{-k}}{1 - \sum_{k=1}^{L-1} a^i_k z^{-k}}
\]
Parallel and cascade form

Some filters can be split into parallel architecture or cascade architecture.

- The global $Z$ transform is function of subindexed filters $H_i$.

$$H_{\text{parallel}} = \sum_{i=1}^{M} H_i(z) = \sum_{i=1}^{M} \frac{\sum_{k=0}^{L-1} b_i^k z^{-k}}{1 - \sum_{k=1}^{L-1} a_i^k z^{-k}}$$

$$H_{\text{cascade}} = \prod_{i=1}^{M} H_i(z) = \prod_{i=1}^{M} \frac{\sum_{k=0}^{L-1} b_i^k z^{-k}}{1 - \sum_{k=1}^{L-1} a_i^k z^{-k}}$$
Parallel and cascade form

Example of cascade form

\[ y[n] = 4x[n] + 3x[n - 1] - x[n - 2] \]

Polynomial form
We have \( H(z) = 4 + 3z^{-1} - z^{-2} \)

\[ -x^2 + 3x + 4 = 0 \]

That leads to

\[ \begin{cases} x_1 = \frac{-3 - \sqrt{25}}{2 \times -1} = 4 \\ x_2 = \frac{-3 + \sqrt{25}}{2 \times -1} = -1 \end{cases} \]
Example of cascade form

\[ y[n] = 4x[n] + 3x[n - 1] - x[n - 2] \]

We have \( H(z) = 4 + 3z^{-1} - z^{-2} \), equivalent to \( H(z) = (z^{-1} + 1)(z^{-1} - 4) \)
Filtering specification

Before being implemented filters should be specified

- What do they do in time and frequency domains
- How are they characterized

More convenient to look at the frequency domain behaviour

- Some parts are cut and some are allowed

1. Low pass filtering (a)
2. High pass filtering (b)
3. Bandpass filtering (c)
4. Stopband filtering (d)

Figure: Frequency response (ideal)
Low pass filtering

We define 3 areas:

1. **Bandwidth**: \(0 < \Omega < \Omega_p\)
2. **Transition band**: \(\Omega_p < \Omega < \Omega_a\)
3. **Attenuated band**: \(\Omega_a < \Omega < \pi\)

\[
|H(e^{j\Omega})| = \begin{cases} 
1, & \Omega < \Omega_p \\
1 - \delta_1, & \Omega_p < \Omega < \Omega_a \\
1 + \delta_1, & \Omega_a < \Omega < \pi
\end{cases}
\]

\[
|H(e^{j\Omega})| (dB) = \begin{cases} 
0 dB, & \Omega < \Omega_p \\
20\log(1-\delta_1), & \Omega_p < \Omega < \Omega_a \\
20\log(1+\delta_1), & \Omega_a < \Omega < \pi
\end{cases}
\]
Bandpass filtering

\[ |H(e^{j\Omega})| \]

Figure: Frequency mask of bandpass
Low pass filtering

All filters are LPF

- HPF as 1-LPF
- BP as translated LPF
- SB as 1-BP

Summary of filter parameters

<table>
<thead>
<tr>
<th></th>
<th>LPF</th>
<th>HPF</th>
<th>Bandpass</th>
<th>Stopband</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selectivity $s$</td>
<td>$\frac{\Omega_p}{\Omega_a}$</td>
<td>$\frac{\Omega_a}{\Omega_p}$</td>
<td>$\frac{\Omega_p+\Omega_a}{\Omega_p-\Omega_a}$</td>
<td>$\Omega_p-\Omega_a$</td>
</tr>
<tr>
<td>Ripple</td>
<td>$\delta_1$</td>
<td>$\delta_1$</td>
<td>$\delta_1$</td>
<td>$\delta_1$</td>
</tr>
<tr>
<td>Attenuation</td>
<td>$\delta_2$</td>
<td>$\delta_2$</td>
<td>$\delta_2$</td>
<td>$\delta_2$</td>
</tr>
<tr>
<td>Centering frequency $\Omega_0$</td>
<td>-</td>
<td>-</td>
<td>$\sqrt{\frac{\Omega_p+\Omega_a}{\Omega_p-\Omega_a}}$</td>
<td>$\sqrt{\frac{\Omega_p+\Omega_a}{\Omega_p-\Omega_a}}$</td>
</tr>
<tr>
<td>Bandwidth broadness $B$</td>
<td>-</td>
<td>-</td>
<td>$\Omega_0$</td>
<td>$\Omega_0$</td>
</tr>
</tbody>
</table>
Filter classification

Several criteria for classification

Impulse response classification

Depending on the Z transform structure, two kind of filters
- Finite Impulse Response filter (FIR)
- Infinite Impulse Response (IIR)

As the name suggests, function of impulse response nature

Structure classification

Function of recursive classification

- Non recursive filters → FIR
- Recursive filters → IIR

---

2Equivalence with one exception, see later
Finite Impulse Response

Cannot be derived from analog filters

**FIR filters**

- Impulse response of output signal is finite
- Particular case of IIR filters when there is no denominator
Finite Impulse Response

Cannot be derived from analog filters

**FIR filters**

- Impulse response of output signal is finite
- Particular case of IIR filters when there is no denominator
- Output is only function of input signal

\[
y[n] = \sum_{k=0}^{L-1} b_k x[n-k]
\]

\[
H(z) = \sum_{k=0}^{L-1} b_k z^{-k}
\]

Limited response in time domain

- \( \forall k > L - 1, \ b_k = 0. \)
Finite Impulse Response

Some key FIR properties

- ✓ Synthesis methods can lead to arbitrary design of any desired frequency response
- ✓ Inherent stability as output only depends on input
- ✓ Design can ensure linear phase, equivalent to constant group delay (no harmonic distortion in signal)
- ✓ Easier implementation in digital architecture
- ✗ Transition bandwidth larger than the one of IIR filter with same size (depth of numerator)
Finite Impulse Response

FIR implementation

- Direct structure (one part of the previously introduced IIR filter)
- Alternative method: transposed structure

\[ x[n] x b_0 z^{-1} x b_1 z^{-1} x b_2 z^{-1} \ldots x b_{L-1} y[n] \]
Finite Impulse Response

FIR implementation

- Direct structure (one part of the previously introduced IIR filter)
- Alternative method: transposed structure
FIR complexity

Lets assess the number of operation to perform a $L$ taps filtering

- $L$ multiplications
- $L - 1$ additions
- $2L$ elements in memory (storing $b_0 \cdots b_{L-1}$) and $L$ elements for the input signal $x[n - k]$

Equivalent to $L$ Multiplication and addition (MAC)
FIR complexity

Lets assess the number of operation to perform a $L$ taps filtering

- $L$ multiplications
- $L - 1$ additions
- $2L$ elements in memory (storing $b_0 \cdots b_{L-1}$) and $L$ elements for the input signal $x[n - k]$ 

Equivalent to $L$ Multiplication and addition (MAC)

A DSP can do one MAC per clock cycle

- To filter a data sampled at $F_c$ what is the minimal DSP ?
- $P_c$ expressed in Million Instruction per Second would require a power calculation at least greater than

$$ P_c > \frac{L \times F_c}{10^6} $$

(4)
Infinite Impulse Response

These filters are directly derived from **analog** filters.

- Same effects for digital filters
- **But** additional effects due to digitalisation (and quantization)

**IIR formulation**

Definition with

- Z transformation
- Difference equation

\[
y[n] = \sum_{k=0}^{L-1} b_k x[n - k] + \sum_{k=1}^{L-1} a_k y[n - k]
\]

\[
H(z) = \frac{\sum_{k=0}^{L-1} b_k z^{-k}}{1 - \sum_{k=1}^{L-1} a_k z^{-k}} = \frac{N(z)}{D(z)}
\]

General bilinear form that links output to input
Infinite Impulse Response

We have defined a numerator and a denominator

\[
\begin{align*}
N(z) &= \sum_{k=0}^{L-1} b_k z^{-k} \\
D(z) &= 1 - \sum_{k=1}^{L-1} a_k z^{-k}
\end{align*}
\]

- If \( D(z) \) does not divide \( N(z) \) we have an infinite number of terms:

\[
H(z) = \sum_{k=0}^{\infty} c_k z^{-k}
\]

- Otherwise, FIR filtering
Some key IIR properties

- Small transition bandwidth
  - Best template can be achieved with IIR filtering

- Synthesis methods directly inherited from analog methods

- Risk of instability due to pole positioning

- Risk of digital instability (feedback loop as output depends on previous outputs)

Efficient filtering architecture but use with caution
Exercise

We consider $H(z) = \frac{1}{M} \frac{1-z^{-M}}{1-z^{-1}}$

Questions

1. Show that we have $y[n] = y[n-1] + \frac{1}{M} [e[n] - e[n-M]]$

2. Show that we have $y[n] = \frac{1}{M} \sum_{i=0}^{M-1} e[n-i]$
Exercise

We consider $H(z) = \frac{1}{M} \frac{1-z^{-M}}{1-z^{-1}}$

Questions

1. Show that we have $y[n] = y[n-1] + \frac{1}{M} [e[n] - e[n-M]]$
2. Show that we have $y[n] = \frac{1}{M} \sum_{i=0}^{M-1} e[n-i]$

Correction

1. From rational expression directly as $H(z) = \frac{Y(z)}{X(z)}$
2. We have $(1 - z^{-M}) = (1 - z^{-1})(1 - z^{-M+1})$
   - Geometric progression with scale factor $z^{-1}$
   - FIR expression
IIR structures

Different structures can be used for IIR filtering

- Two based on recursive and non recursive decomposition
  - One straightforward structure (independent stages)
  - One that group same delay operation

- Structure that minimize memory footprint
  - Introduction of \( w[n - k] \) linked to what happens at delay \( k \)
  - 2 structures can be defined (direct and transposed)

Non canonic structures

- \( H(z) = \frac{N(z)}{D(z)} = N(z) \times \frac{1}{D(z)} \)
- IIR is the cascade of a FIR and a pure recursive filter (also called all pole filter)
IIR structure - Non Canonic Cascade

In cascade form
- One stage dedicated to Numerator and one to Denominator

Figure: IIR cascade structure
IIR structure - Non Canonic direct

In direct form
One stage per delay
IIR filter - Canonic structures

**Direct** versus **transpose** structures

- Due to multiplication commutativity
- IIR is also the cascade of an **all pole** filter and a FIR filter

However these structures require a $\times 2$ in terms of memory (as $D(z)$ and $N(z)$ are treated independently)

→ Minimization of number of memories: **Canonic structures**
IIR filter - Canonic structures

**Direct versus transpose structures**
- Due to multiplication commutativity
- IIR is also the cascade of an **all pole** filter and a FIR filter

However these structures require a $\times 2$ in terms of memory (as $D(z)$ and $N(z)$ are treated independently)

Minimization of number of memories: **Canonic structures**

**Canonic structures**
We introduce $W(z)$ and we express the filter as a set of equations

\[
\begin{align*}
W(z) &= \frac{1}{D(z)} \cdot X(z) \\
Y(z) &= N(z) \cdot W(z)
\end{align*}
\]

\[
\begin{align*}
W[n] &= x[n] + \sum_{k=1}^{L-1} a_k \cdot w[n - k] \\
y[n] &= \sum_{k=0}^{L-1} b_k \cdot w[n - k]
\end{align*}
\]  

(5)

- Same memory register $w[n - k]$ in the two structures
- A canonic transpose can be straightforwardly deduced from (5) due to multiplication commutativity.
IIR Filter - Canonic direct

\[ x[n] + b_0 \times y[n] - z^{-1} \times (x[n] + b_1 \times y[n-1] + b_2 \times y[n-2] + \ldots + b_{L-1} \times y[n-L+1]) = a_1 \times w[n] + a_2 \times w[n-1] + a_{L_1} \times w[n-L+1] \]

\[ y[n] = \frac{1}{1 + a_1 + a_2 + a_{L_1}} \cdot (x[n] - z^{-1} \times (x[n] + b_1 \times y[n-1] + b_2 \times y[n-2] + \ldots + b_{L-1} \times y[n-L+1])) 
\]

**Figure**: Direct Canonic structure
IIR Filter - Canonic transpose

\[
x[n] \times b_0 + y[n] = x[n]\times b_1 x[a_1] z^{-1} + x[b_2] x[a_2] z^{-1} + \ldots + x[b_{L-1}] x[a_{L-1}] z^{-1}
\]

**Figure:** Transpose Canonic structure
IIR complexity evaluation

Let's assess the number of operation to perform a $L$ taps IIR filtering

- $2L$ multiplications
- $2(L - 1)$ additions
- Memory depends on structure
  - $4L$ elements for direct structure ($2L$ coefficients, $L$ inputs, $L$ outputs)
  - $3L + 2$ for canonic structure ($x[n], y[n], 2L$ coefficients, and $L w[n - k]$ coefficients)

Equivalent to $2L$ Multiplication and addition (MAC)
IIR complexity evaluation

Let's assess the number of operations to perform a $L$ taps IIR filtering:

- $2L$ multiplications
- $2(L-1)$ additions
- Memory depends on structure:
  - $4L$ elements for direct structure ($2L$ coefficients, $L$ inputs, $L$ outputs)
  - $3L + 2$ for canonic structure ($x[n], y[n], 2L$ coefficients, and $L w[n-k]$ coefficients)

Equivalent to $2L$ Multiplication and addition (MAC)
IIR complexity evaluation

Let's assess the number of operation to perform a $L$ taps IIR filtering

- $2L$ multiplications
- $2(L - 1)$ additions
- Memory depends on structure
  - $4L$ elements for direct structure ($2L$ coefficients, $L$ inputs, $L$ outputs)
  - $3L + 2$ for canonic structure ($x[n], y[n], 2L$ coefficients, and $L \ w[n - k]$ coefficients)

Equivalent to $2L$ Multiplication and addition (MAC)

A DSP can do one MAC per clock cycle

- To filter a data sampled at $F_c$ what is the minimal DSP?
- $P_c$ expressed in Million Instruction per Second would require a power calculation at least greater than

$$P_c > \frac{2L \times F_c}{10^6}$$

(6)
Fixed point arithmetic
A simple example to begin

Picsou wants to have an interesting investment for his cash.

The banker proposes the following rule:

- You place $e - 1$ euros
- The first year, we multiply by 1 and take over 1 euro
- The second year, we multiply by 2 and take over 1 euro
- The $n$ year, we multiply by $n$ and take over 1 euro
- You get your cash back in 25 years
A simple example to begin

So the banker do the simulation on his computer

- After 25 years, Picsou would earn $+4645987753\text{€}$

So picsou is happy

After going back to his home, he calculates the expected benefits in his calculator

- He obtains $-140 \times 10^{12}\text{€}$

How we use (and represent) numbers matter!
that can have tremendous impact...

Ariane V explosion on June 4\textsuperscript{th} 1996 (flight 501).

- Explosion just after launch (30 second)
- Loss of launcher and cargo (500 million dollars)

After investigation

- This was due to the inertial measurement unit (IMU)
- The horizontal speed was calculated on 64 bits and then converted to 16 bits without overflow checking
- System was inherited from Ariane 4, with lower ground speed
What happens?

Digital signal is discrete in time domain
- But it also have a finite number for its representation
- As there is a time-frequency grid for digital signal processing, there is an amplitude grid

Some (weird) phenomena can occur
- Absorption
- Truncation
- Elimination

The way we do the calculation also matters!

First let’s have a look on number representation
Numbers representation

Unsigned integer coding

We have a number $x$, represented on a basis $B$
- Element $B$: base (can be 10, 2, whatever)
- Decomposition on $n$ elements
- Decomposition on the elementary power of the base element

$$x = \sum_{k=0}^{n-1} a_k B^k$$

- $B^k$ are the weights
- $a_k$ basis elements (coefficients) such as $a_k \in [0; B - 1]$
Numbers representation

Example of binary coding
The coefficient $a_k$ are also called the **bits** (or binary digits)

Bit origin
First use in a 1948 paper entitled “A Mathematical Theory of Communication” by Claude Shannon [10]

- He attributed its origin to John W. Tukey, who had written a Bell Labs memo on 9 January 1947 in which he contracted "binary information digit" to simply "bit"

$$
 x = \sum_{k=0}^{n-1} a_k B^k = a_0 a_1 a_2 \ldots a_{n-1}
$$

- $a_0$ is the **Less significant bit (LSB)**
- $a_{N_1}$ is the **Most significant bit (MSB)**

# Numbers representation

<table>
<thead>
<tr>
<th>Decimal value $A_{10}$</th>
<th>Binary code $a_3a_2a_1a_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
</tr>
</tbody>
</table>
## Numbers representation

<table>
<thead>
<tr>
<th>Decimal value $A_{10}$</th>
<th>Binary code $a_3a_2a_1a_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
</tr>
</tbody>
</table>

For binary representation

$$x = \sum_{k=0}^{b-1} b_k2^k$$

- Coding on $b$ bits
- Can represent any number between 0 and $2^b - 1$
Sign encoding

Signed integer coding (Sign absolute value)
We focus now on binary representation

- Still $b$ bits

Unsigned bit coding allow to code between 0 and $2^b - 1$

How to represent negative numbers?

One bit is now allocated to the sign

- 1 bit for sign (often called $S$)
- $b - 1$ bits for binary representation
Sign encoding

Signed integer coding (Sign absolute value)
We focus now on binary representation

- Still $b$ bits

Unsigned bit coding allow to code between 0 and $2^b - 1$

How to represent negative numbers?

One bit is now allocated to the sign

- 1 bit for sign (often called $S$)
- $b - 1$ bits for binary representation

$$x = (-1)^S \sum_{k=0}^{b-2} b_k 2^k$$
Sign encoding

Signed integer coding (Sign absolute value)

Drawbacks

- 0 can be represented in 2 manners: +0 and -0.
- Basic operation cannot be straightforwardly obtained from one bit operations

Example of signed absolute coding drawbacks: additions

- Depending on the sign, two operations must be performed
  - Same sign: adding value and keep the sign
  - Different sign: subtraction is required
Sign encoding

Signed integer coding (2’s complement)
Special case of binary representation

- Avoid the two representation of 0
- Representation in order to ease arithmetic operation
- From sign product to sign addition

Complement to two

\[ x = (-2)^{b-1} \times S + \sum_{k=0}^{b-2} b_k 2^k \]

- MSB is the sign bit
- Positive number have a legacy unsigned representation on \( b - 1 \) bits
- Negative numbers starts from \( 2^{b-1} - 1 \)
Sign encoding

Signed integer coding (2’s complement)

Properties

- Definition domain is not symmetric: more negative values than positive ones

\[ D = [-2^{b-1}, \ldots, 2^{b-1} - 1] \]

- Transition from positive numbers to negative numbers: Bit flip +1

\[
0 = \{00\ldots0\} \\
-0 = \{11\ldots1\} + 1 = \{00\ldots0\}
\]
Sign encoding

Signed integer coding (2’s complement)
Example: 0000 0101

- 8 bits, signed, two’s complement
- First bit is 0: positive number
- Positive part: 000 0101 ➔ 5

Example: -5

From initial example
- Bit flip of 5 + 1
  - 1111 1011

From scratch
- First bit is equal to 1 as negative number: 1xxx xxxx
- 8 bits (signed), -5 is coded from inverse on 7 bits: 2\( \text{x} \) 128 \(-\) 5 = 123
- Representation of 123 is 111 1011
Sign encoding

Signed integer coding (2’s complement)
Example: 0000 0101
- 8 bits, signed, two’s complement
- First bit is 0: positive number
- Positive part: 000 0101 ➞ 5

Example: -5 ?
- What is the binary representation of -5 ?
- From initial example
  - Bit flip of 5 + 1
  - 1111 1011
- From scratch
  - First bit is equal to 1 as negative number: 1xxx xxxx
  - 8 bits (signed), -5 is coded from inverse on 7 bits: \(2^7 = 128 - 5 = 123\) ➞ x111 1011
  - Representation of 123 is 111 1011
Fixed point

Number coding

- We have considered integer (unsigned and signed) numbers.
- In a DSP acquisition, any number can be considered.
- How can we use real numbers?

We have a certain number of bits:

- Any number cannot be represented, and precision depends on how we represent the number.
- A number can be split as an integer part and a fractional part.

Fixed point

- Integer part is coded on $m$ bits.
- Fractional part is on $n$ bits.
Fixed point representation

- $2^m$ to $2^{m-1}$
- $2^l$ to $2^0$
- $2^{-1}$
- $2^{-n}$

- $S$
- $b_{m-1}$ to $b_{m-2}$
- $b_1$ to $b_0$
- $b_{-1}$
- $b_{-n+2}$ to $b_{-n+1}$
- $b_{-n}$

$m$ bits for integer part

$n$ bits for fractional part

$b$ bits = $m + n + 1$

$(b, m, n)$ bits or $Q(m, n)$

Still depends on the representation

Figure: Fixed point representation
Fixed point parameters

Some useful metrics can be defined for fixed point arithmetic

**Range**

- Range of the number coding

\[ D = X_{\text{max}} - X_{\text{min}} = 2 \times (2^m - 2^{-n}) \]  

(7)

**Number cardinal**

Number of effective numbers that can be mapped by the representation \( N_c \)

\[ N_c = 2^b \]
Fixed point parameters

Quantization step

- Distance (euclidean) between 2 successive values (uniform quantization)

\[ q = \frac{D}{N_c - 1} \]  

(8)

Dynamic in dB

Comparison between maximal and minimal coded value, in log scale

\[ N_D \ dB = 20 \log \left( \frac{\max(|x|)}{\min(|x|)} \right) \approx 20 \cdot b \cdot \log(2) = 6.02 \cdot b \]  

(9)
Fixed point

Two’s Complement FP

Focus here on 2’s complement approach [11].
There are three ways for decimal point positioning:

- Right justified: it means that the data is an integer as no fractional bits are used

- Left justified: it means that the data is completely fractional: normalized input

- $Q(m,n)$ with $m \neq 0$ and $n \neq 0$: data is composed of an integer and a fractional parts.

## Fixed point

<table>
<thead>
<tr>
<th>Representation</th>
<th>Framing Left</th>
<th>Framing Right</th>
<th>Framing $n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>$m = 0$</td>
<td>$n = 0$</td>
<td>$n + m = b - 1$</td>
</tr>
<tr>
<td>$q$</td>
<td>$2^{-(b-1)}$</td>
<td>1</td>
<td>$2^{-(n)}$</td>
</tr>
<tr>
<td>$\mathcal{D}$</td>
<td>$[-1; 1 - q]$</td>
<td>$[-2^{b-1}; 2^{b-1} - q]$</td>
<td>$[-2^{m}; 2^{m} - q]$</td>
</tr>
</tbody>
</table>

**Table:** FP parameters
Example

<table>
<thead>
<tr>
<th>Left framing</th>
<th>Right framing</th>
<th>Value</th>
<th>( m = 3 )</th>
<th>( n = 2 )</th>
<th>C.A.2</th>
<th>S.V.A.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.96875</td>
<td>31</td>
<td>7.75</td>
<td></td>
<td></td>
<td>011111</td>
<td>011111</td>
</tr>
<tr>
<td>0.9375</td>
<td>30</td>
<td>7.5</td>
<td></td>
<td></td>
<td>011110</td>
<td>011110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0.3125</td>
<td>1</td>
<td>0.25</td>
<td></td>
<td></td>
<td>000001</td>
<td>000001</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>000000</td>
<td>100000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>-0.3125</td>
<td>-1</td>
<td>-0.25</td>
<td></td>
<td></td>
<td>111111</td>
<td>100001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>-0.9375</td>
<td>-30</td>
<td>-7.5</td>
<td></td>
<td></td>
<td>100010</td>
<td>111110</td>
</tr>
<tr>
<td>-0.96875</td>
<td>-31</td>
<td>-7.75</td>
<td></td>
<td></td>
<td>100001</td>
<td>111111</td>
</tr>
<tr>
<td>-1</td>
<td>-32</td>
<td>-8</td>
<td></td>
<td></td>
<td>100000</td>
<td>...</td>
</tr>
</tbody>
</table>
We consider a Q(6,3,2) system

<table>
<thead>
<tr>
<th>Binary</th>
<th>Binary</th>
<th>Decimal value</th>
</tr>
</thead>
<tbody>
<tr>
<td>011111</td>
<td>0 111.11</td>
<td>7.75</td>
</tr>
<tr>
<td>011110</td>
<td>0 111.10</td>
<td>7.5</td>
</tr>
<tr>
<td>000011</td>
<td>0 000.11</td>
<td>0.75</td>
</tr>
<tr>
<td>000010</td>
<td>0 000.10</td>
<td>0.5</td>
</tr>
<tr>
<td>000001</td>
<td>0 000.01</td>
<td>0.25</td>
</tr>
<tr>
<td>000000</td>
<td>0 000.00</td>
<td>0</td>
</tr>
<tr>
<td>111111</td>
<td>1 111.11</td>
<td>-0.25</td>
</tr>
<tr>
<td>111111</td>
<td>1 111.10</td>
<td>-0.5</td>
</tr>
<tr>
<td>100001</td>
<td>1 000.01</td>
<td>-7.75</td>
</tr>
<tr>
<td>100000</td>
<td>1 000.00</td>
<td>-8.00</td>
</tr>
</tbody>
</table>
Floating point

Fixed point coding

- One part is dedicated to integer coding
- One part is dedicated to fractional part coding

The position of the point is fixed

Alternative way to represent number: floating point

- Equivalent to scientific notation
- The number of bits $b$ is fixed, but the position of the point varies and adapt to the encoded data
- Part of the number coded is devoted to point location information
How can we use a floating point system

- Standardized by IEEE (IEEE 754)
- We still have the sign bit $S$
- One part is called exponent
- One part is called significand/mantissa/fraction

![Diagram of floating point representation](image)
Floating point

Exponent

Denoted as \( u \)
- Corresponds to a scaling factor that can be adapted
- Exponent can be positive or negative, and coded with 2’s complement

Significand/mantissa/fraction

- Corresponds to the value of the data divided by the scaling factor
- Significand belongs to \([1; 2]\)

Expression of data \( x \) is

\[
x = (-1)^S \cdot 2^u \left( 1 + \sum_{i=0}^{M-1} C_i 2^{i-M} \right)
\]

with

\[
u = \sum_{i=0}^{E-1} d_i 2^i - (2^{E-1} - 1)
\]

(10)
### Floating point

**Table: IEEE 754 number representation.** WL, M, and E denote respectively bit width, fractional size, and exponent size.

<table>
<thead>
<tr>
<th>Representation</th>
<th>WL (bits)</th>
<th>F (bits)</th>
<th>E (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple precision (float)</td>
<td>32</td>
<td>23</td>
<td>8</td>
</tr>
<tr>
<td>double precision (double)</td>
<td>64</td>
<td>52</td>
<td>11</td>
</tr>
<tr>
<td>Extended double precision (long double)</td>
<td>80</td>
<td>64</td>
<td>15</td>
</tr>
</tbody>
</table>

- 2 codes corresponds to 0 (+0 and -0)
- Special sequence corresponds to $\pm\infty$, NaN ...
- More (and precious) information in [12]

---

Fixed point vs Floating point

What is the dynamic for fixed point and floating point?

- For fixed point, dynamic is 6.02b
- Depends on bit size and exponent allocation for floating point (assume 1/4 exponent alloc)

---

Figure: Dynamic comparison between fixed point implementation and floating point implementation
Fixed point vs Floating point

Floating point takes advantage when many bits are used

- If number of bits is greater than 16, better use a Floating point system

However having a lot of bits can be costly

- In many DSP applications, few bits are considered
- High number of bits affects the DAC architecture (and the targeted bandwidth)
- Modern architectures tend to be floating point but many DSP applications are still based on fixed point arithmetic (and will be)

Let’s focus on Fixed point approach
Conclusion on Fixed/Floating point

Fixed point

- One sign, on integer part, on fractional part
- Two’s complement implementation
- \((b, m, n)\) or \(Q(m, n)\) with \(b = m + n + 1\)

Floating point

- One sign, exponent and fraction
- Resolution depends on coded word
- IEEE 754 norm
Fixed point arithmetic

Simple rules for bit manipulation

- Arithmetic can be done with logical gates

---

**Figure: Logical gates**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**NOT**

**AND**

**NAND**

**OR**

**NOR**

**EXCLUSIVE OR**

**EXCLUSIVE NOR**
Fixed Point arithmetic

Addition and subtraction for unsigned numbers

As shown before, operation corresponds to exclusive OR and AND for carry

- The carry is iteratively shifted to the left
- If inputs on $n$ bits and output on $n$ bits: overflow can occur
- If the carry of the last stage is 1, result is greater than $2^n - 1$
Fixed Point arithmetic

Addition and subtraction for Two’s complement

- The carry is iteratively shifted to the left
- If inputs on $n$ bits and output on $n$ bits: overflow can occur
- Subtraction and additions are the same (carry of the sign bit is not shifted though).
- Fixed point addition: same!

\[
\begin{array}{cccc}
14 & 0000 & 1110 & -14 & 1111 & 0010 & -14 & 1111 & 0010 \\
+13 & 0000 & 1101 & +13 & 0000 & 1101 & -13 & 1111 & 0011 \\
+27 & 0001 & 1011 & -1 & 1111 & 1111 & -27 & 1110 & 0101 \\
\end{array}
\]
Fixed point arithmetic

Addition

- Exclusive OR for bit addition
- The carry is a AND

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Exclusive OR &amp; carry</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>(1, 0)</td>
</tr>
<tr>
<td>Propag-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>carry-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Res-1</td>
<td>0</td>
<td>1</td>
<td>(1, 0)</td>
<td>0</td>
</tr>
<tr>
<td>Propag-2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>carry-2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Res-2</td>
<td>0</td>
<td>(1, 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Propag-3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>carry-3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Res-3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Res</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Example

\[
\begin{array}{c|cccccccc}
\times & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
y & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\(x+y\) ? Check with decimal that the result is correct
Sign encoding

Example with 8 bits coding

- Binary coding of 3
- Binary coding of -4
- Check that 3 + 3 leads to 6
- Check that 3 - 4 leads to -1
Addition in Fixed point

Rules

- $x$ and $y$ must have the same format $(b, m, n)$
- If they have not the same format, need to recast the data

\[ m_c = \max(m_a, m_b) \]
\[ n_c = \max(n_a, n_b) \]
\[ b_c = m_c + n_c + 1 \]  

And the recast is done as follows

- Fractional bits added $(n_c - n_a)$ (right) are set to 0
- Additional integer bits $(m_c - m_a)$ are set to sign bit value (sign bit extension)
Addition in Fixed point

Result of addition is on

\[ z \rightarrow (b_c, m_c, n_c) \]

- Result may not belong to interval of interest
- One bit can be necessary to avoid the overflow
- If result does not belong to \( D_c = \left[ -2^{m_c}; 2^{m_c} \right] \)

\[ z \rightarrow (b_c+1, m_c+1, n_c) \]
Fixed point arithmetic

Multiplication

- Elementary bit operation is a AND
- Output is extended on more bits than input

![Sign number multiplication](image)

**Figure:** Sign number multiplication
Fixed point arithmetic

Multiplication

- In fixed point output is on more bit than inputs
- Sign bit extension at each stage

\[
\begin{align*}
m_{Mult} &= m_a + m_b + 1 \\
n_{Mult} &= n_a + n_b \\
b_{Mult} &= b_a + b_b
\end{align*}
\] (12)
**Fixed point arithmetic**

**Example** Consider format \((4,0,3)\)

- \(x = -0.5\)
- \(y = 0.75\)

Calculate \(x \times y\)

\[
\begin{array}{c}
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
\end{array}
\]

- \(x = (-0.5)\) format: \((4,0,3)\)

\[
\begin{array}{c}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]

- \(y = (0.75)\) format: \((4,0,3)\)

Multiply the two numbers:

\[
\begin{array}{cccccccc}
& & & & & & & \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
\times & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\hline
1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

- \(z = (-0.375)\) format: \((8,1,6)\)

**Sign bit**

**Redundant sign bit** --

\(2^{-1}\)

\(2^6\)
Conclusion

Addition

- Same format (sign bit extension)
- Output is $m_c + 1$ (truncation if definition domain is maintained)

Multiplication

- Bit sign extension
- Output format is different

\begin{align*}
m_{\text{Mult}} &= m_a + m_b + 1 \\
n_{\text{Mult}} &= n_a + n_b \\
b_{\text{Mult}} &= b_a + b_b
\end{align*}

(13)
Fixed point design

We now how to do bit operation in Fixed point

How to manage and built a complete system?

Fixed point design

3 aspects

1. What is the dynamic?
2. How to split integer and fractional parts
3. What is the quantization noise?
Fixed point design

- Requires a graph flow of desired applications
- Output is noisy: Precision is evaluated with Signal to Noise Ratio (SNR)
- Floating point baseline for input (and SNR estimation)
- Design is done **before** implementation
Fixed point design

**Dynamic**
Dynamic is used in order to minimize number of bits
- Done to ensure that no overflow occur

\[ x[n] \xrightarrow{} F \xrightarrow{} y[n] \]

- \( x[n] \) in floating point
- \( y[n] \) in floating point

How to calculate dynamic of \( y[n] \)?

**Dynamic estimation**
1. Simulation method
2. Analytical methods
Fixed point design

Dynamic

Simulation method
First way to estimate dynamic

- Simpler but risky method
- From various $x[n]$ calculate the floating output and determine its dynamic
- Does not ensure no overflow! Depends on how simulated $x[n]$ differs from experienced $x[n]$
- Sophisticated methods based on statistical approach
Fixed point design

Example with a FIR filter

Figure: FIR example impulse response
Fixed point design

Example with a FIR filter

![FIR example frequency response](image)

**Figure:** FIR example frequency response
Fixed point design

- **White Gaussian noise**:
  - $x(n)$
  - $y(n)$

- **Chirp ($0 \rightarrow F_s/2$)**: 
  - $x(n)$
  - $y(n)$

- **Uniform white noise**: 
  - $x(n)$
  - $y(n)$

- **Maxima**:
  - $\max_{n}(|y(n)|) = 3.4$
  - $\max_{n}(|y(n)|) = 2.4$
  - $\max_{n}(|y(n)|) = 9.3$
Fixed point design

Dynamic

Interval arithmetic

Other way to estimate dynamic

- Deduce definition domain from all elementary operations
- Use definition domain of input of elementary operations

\[ D_x = [x, \bar{x}] \]
\[ D_y = [y, \bar{y}] \]

- Output definition domain depends on input and operations

\[
\begin{align*}
z = x + y & \quad [z, \bar{z}] = [x + y, \bar{x} + \bar{y}] \\
z = x - y & \quad [z, \bar{z}] = [x - \bar{y}, \bar{x} - y] \\
z = x \times y & \quad [z, \bar{z}] = [\min(xy, x\bar{y}, \bar{x}y, \bar{x}\bar{y}), \max(xy, x\bar{y}, \bar{x}y, \bar{x}\bar{y})]
\end{align*}
\]
Fixed point design

Dynamic

Interval arithmetic
- Complete graphflow can be obtained from all interval calculation
- But can only be done with non-recursive systems
- Can be done for FIR and not for IIR
Consider the previous FIR filter.

\[ x(n) \rightarrow z^{-1} \rightarrow c_0 \rightarrow c_1 \rightarrow \sum \rightarrow y(n) \]

\[ x(n) \rightarrow z^{-1} \rightarrow c_{N-2} \rightarrow c_{N-1} \rightarrow \sum \rightarrow y(n) \]

Dynamic for FIR filter

Filter expression

\[ h_k = \frac{1}{5} + \frac{3. k}{160} \quad \forall k \in \left[0; \frac{L}{2} - 1\right] \]  

(15)

\[ h_k = h_{L-1-k} \quad \forall k \in \left[\frac{L}{2}; L-1\right] \]  

(16)

Dynamic evaluation

\[ D_y = 10.90 \]  

(17)
Norm

- Norm is an operand that calculate output dynamic based on input dynamic
- Assume that the system is
  - Linear
  - Invariant in time domain
- For Single input
  \[ y[n] = h[n] \ast x[n] \Leftrightarrow Y(z) = H(z).X(z) \]  
  (18)

- and for Multiple inputs
  \[ y[n] = \sum_{i=0}^{N_e-1} h_i[n] \ast x_i[n] \]  
  (19)

Objective: Determine maximal output based on maximal input
Considering single input system, we have

\[ y[n] = h[n] \ast x[n] \iff Y(z) = H(z)X(z) \]

And absolute value of output is

\[ |y(n)| = \left| \sum_{m=-\infty}^{+\infty} h(m)x(n - m) \right| \]

(20)
Considering single input system, we have
\[ y[n] = h[n] \star x[n] \Leftrightarrow Y(z) = H(z) \cdot X(z) \]

And absolute value of output is
\[ |y(n)| = \left| \sum_{m=-\infty}^{+\infty} h(m)x(n - m) \right| \quad (20) \]

This can be bounded by:
\[ y_{max} = \max_n (|y(n)|) \leq \max_n (|x(n)|) \sum_{m=-\infty}^{+\infty} |h(m)| \quad (21) \]
Fixed point design

L1 norm

Multiple inputs
Extension to multiple inputs

\[ y_{\text{max}} = \max_n \left( |y(n)| \right) \leq \sum_{i=0}^{N_e-1} \max_n \left( |x_i(n)| \right) \sum_{m=-\infty}^{+\infty} |h_i(m)| \]  \hspace{1cm} (22)

L1 Norm summary

■ Based on maximal value bound
■ L1 norm does not rely on x statistical hypothesis
■ No overflow, but may be overestimated
Example with FIR filtering

Consider the previous FIR filter.

![FIR filter diagram]

Dynamic for FIR filter

Filter expression

\[ h_k = \frac{1}{5} + \frac{3. k}{160} \quad \forall \ k \in [0; \frac{L}{2} - 1] \]

\[ h_k = h_{L-1-k} \quad \forall \ k \in \left[\frac{L}{2}; L - 1\right] \]

L1 Norm:

\[ D_y = 10.90 \]
Fixed point design

Chebyshev norm

Single input

- L1 norm can be overestimated, particularly if input signal is narrowband
- We consider a pure tone signal

\[ x(n) = A \cos(\Omega \cdot n) \quad \text{with} \quad 0 \leq \Omega < \pi \]  \hspace{1cm} (23)

- \( H \) is a filter, and the output is

\[ |y(n)| = |A| \cdot |H(e^{j\Omega})| \cdot |\cos(\Omega \cdot n + \arg(H(e^{j\Omega})))| \quad \text{with} \quad 0 \leq \Omega < \pi \]  \hspace{1cm} (24)

- Chebyshev norm is defined as

\[ y_{\text{max}} = \max_{n} (|y(n)|) \leq \max_{n} (|x(n)|) \max_{\Omega} (|H(e^{j\Omega})|) \]  \hspace{1cm} (25)
Fixed point design

Chebyshev norm

Multiple inputs

- Extension to multiple inputs

\[ y_{\text{max}} = \max_{n} (|y(n)|) \leq \sum_{i=0}^{N_{e}-1} \max_{n} (|x_{i}(n)|) \max_{\Omega} (|H_{i}(e^{j\Omega})|) \]  \hspace{1cm} (26)

Chebyshev Norm summary

- System excited by single tone
- Less pessimistic than L1 norm
- Requires condition on input narrowness
Chebychev norm for L1 norm

\[ y_{\text{max}} = \left| H(e^{j0}) \right| = \left| \sum_{m=0}^{N-1} c_m z^{-m} \right|_{z=1} = 10.90 \]
Application to cascaded system

We consider a cascaded system with 2 filters

\[ H_1(z) \cdot H_2(z) \]

\[ x(n) \rightarrow H_1(z) \rightarrow y_1(n) \rightarrow H_2(z) \rightarrow y_2(n) \]

\[ |H_1(e^{j\omega})| \]
\[ |H_2(e^{j\omega})| \]
\[ |H_1(e^{j\omega})| = |H_2(e^{j\omega})| \]
Application to cascaded system

When a cascaded system is considered, need to use the **equivalent transfer function**

$$H(z) = H_1(z) \times H_2(z) \times H_3(z)$$

**Cascaded system**

$$y_{2\text{--max}} = \max_n \left( |x(n)| \right) \max_\Omega \left( |H_{12}(e^{j\Omega})| \right)$$

$$= \max_\Omega \left( |H_1.H_2(e^{j\Omega})| \right) = 0,356$$
Application to cascaded system

When a cascaded system is considered, need to use the **equivalent transfer function**

\[
x(n) \rightarrow H_1(z) \rightarrow y_1(n) \rightarrow H_2(z) \rightarrow y_2(n)
\]

**Sucesive system**

\[
Y_2(z) = H_2(z) \times Y_1[z]
\]

**Dynamic:**

\[
y_{2\text{--max}} = \max_n (|x(n)|) \max_\Omega (|H_1(e^{j\Omega})|) \max_\Omega (|H_2(e^{j\Omega})|) = 1 \times 1 = 1
\]
Conclusion on dynamic

- Dynamic to calculate variation of output (wrt input): number of bits calculation

Simulated methods
- Based on maximal encountered value based on inputs
- Precise estimation. Does not avoid overflow

Analytic methods
- Interval arithmetic
  - Based on flowgraph between input and output
  - Elementary operations are propagated

- Norms
  - L1 Norm
    - No overflow but (can be) overestimated
  - Chebyshev norm
    - Based on pure tone excitation
  - For norms, cascaded systems as single output/input
Point positioning

We know what the dynamic is

- As we are in fixed point, number of bits is not enough to characterize data
- Need to position the point (limit between integer and fractional part)

Point positioning

- Each stage of the flowgraph (elementary operation) will have its own triplets
- Ensure correct specification with respect to arithmetic
- Ensure no overflow

From dynamic and interval definition

\[ 2^{m_x-1} \leq x_{\text{max}} < 2^{m_x} \]  \hspace{1cm} (28)

Position is thus:

\[ m_x = \left\lfloor \log_2 \left( \max_n (|x(n)|) \right) \right\rfloor + 1 \]  \hspace{1cm} (29)
Example of point positioning

Data is on 8 bits

1. Position of point if $x \in [0, 1]$ ?
2. Position of point if $x \in [0, 1[ ?
3. Position of point if $x \in [0, 20] ?$
4. Position of point if $x \in [0, 512] ?$
5. Position of point if $x \in [0, 0.004] ?$
Point positioning

Rules

Where the point is after elementary operation?

Multiplication

- Fixed point multiplication leads to redundant sign bit
- This bit is integrated into integer part of result

\[
m_z = m_x + m_y + 1
\]

- In other words

\[
b_z = b_x + b_y, \quad n_z = n_x + n_y
\]

\[
\rightarrow m_z = m_x + m_y + 1
\]
Point positioning

Rules

Where the point is after elementary operation?

Addition without guard bits

- **Common point positioning** for the two inputs
- Position to ensure no overflow

If output has no guard bit, point positioning as follows:

\[ m_c = \max (m_x, m_y, m_z) \]

Fixed point format update before mapping to addition module:

\[
\begin{align*}
  d_x &= m_c - m_x \quad \text{Rightshift } d_x \text{ bits (} \gg \text{)} \\
  d_y &= m_c - m_y \quad \text{Rightshift } d_y \text{ bits (} \gg \text{)} \\
  d_z &= m_c - m_z \quad \text{Leftshift } d_z \text{ bits (} \ll \text{)}
\end{align*}
\] (30)
Addition with guard bits

- If inputs have not same format (MSB not aligned)
- Data should be **realigned** before doing addition

\[
\begin{align*}
    m_x' &= m_x - g_x \\
    m_y' &= m_y - g_y \\
    m_z' &= m_z - g_z
\end{align*}
\]  \hspace{1cm} (31)

- With \( g \) number of bits, \( m \) initial dot positioning and \( m' \) common dot positioning
Point positioning

Addition with guard bits

\[ m_c = \max (m_{x'}, m_{y'}, m_{z'}) = \max (m_x - g_x, m_y - g_y, m_z - B_g) \]
Guard bit set to maximal value from adder

Common point positioning

Adding guard bit (sign replication)

Result is then right shifting if not all guard bits are effective used
Addition with guard bits

Guard bits can be used at the output of the adder

\[
\begin{align*}
g_z &= m_z - m_c \quad \text{if} \quad m_z > m_c \\
g_z &= 0 \quad \text{if} \quad m_z \leq m_c
\end{align*}
\] (32)

Input point positioning

\[
\begin{align*}
m_x &= m_c + g_x \\
m_y &= m_c + g_y
\end{align*}
\] (33)
Point positioning

Addition with guard bits

- Input point positioning
  \[
  \begin{align*}
  m_x &= m_c + g_x \\
  m_y &= m_c + g_y
  \end{align*}
  \tag{35}
  \]

- Output point positioning
  \[
  m_z = m_c + g_z
  \tag{36}
  \]
FIR filter example

Indecent consideration of each filter unit:

For each cell, Fixed point format depends on previous cell
FIR filter example
Conclusion on Point positioning

Conclusion

- Dynamic gives position of point

\[ m_x = \left\lfloor \log_2 \left( \max_n (|x(n)|) \right) \right\rfloor + 1 \]

- Rules for Fixed point arithmetic
  - Multiplication
    \[ m_z = m_x + m_y + 1 \] (37)
  - Adder w/o guard bit
    - Common format
    - Shift for alignment and no overflow
    \[
    \begin{align*}
    d_x &= m_c - m_x & \text{Rightshift } d_x \text{ bits } (\gg) \\
    d_y &= m_c - m_y & \text{Rightshift } d_y \text{ bits } (\gg) \\
    d_z &= m_c - m_z & \text{Leftshift } d_z \text{ bits } (\ll)
    \end{align*}
    \]
Conclusion on Point positioning

Conclusion

- Adder with guard bits
  - Need to shift to common format
    \[
    \begin{align*}
    m'_{x} &= m_{x} - g_{x} \\
    m'_{y} &= m_{y} - g_{y} \\
    m'_{z} &= m_{z} - g_{z}
    \end{align*}
    \]
  - Guard bits can be used at the output of the adder
    \[
    \begin{align*}
    g_{z} &= m_{z} - m_{c} & \text{if } m_{z} > m_{c} \\
    g_{z} &= 0 & \text{if } m_{z} \leq m_{c}
    \end{align*}
    \]
  - Input point positioning
    \[
    \begin{align*}
    m_{x} &= m_{c} + g_{x} \\
    m_{y} &= m_{c} + g_{y}
    \end{align*}
    \]
- Output point positioning
  \[
  m_{z} = m_{c} + g_{z}
  \]
Data width

From previous section

- know the dynamic
- We know how to locate the point

In some cases, enough as data width is fixed (for example, 8 or 16 bits)
- Just ensure that data format is acceptable

In many cases, we want to use only bits that bear effective data
- What is the fractional part length?
Data width

Data width depends on instruction set used

- Instruction set depends on chosen target
- For $\mu C$, DSP, 3 sets

Classic operations

- 1 operation per cycle, with data width $b_{op}$
- $b_{op}$ depends on chosen architecture

Arbitrary precision operations

- Data has a larger width
- Data is obtained by word concatenation

Sub word parallelism operations

- Data has a narrow width
- Parallel processing with $k$ branch
- Can process word with data length $b_{op}/k$. 
Example of MAC operation

Double precision (with guard bit)

Double precision (w/o guard bit)

Single precision
Example of FIR filtering

Figure: Generic system for FIR
Example of FIR filtering

**Input:** (16,0,15)
**Coefficient:** (16,-1,16)
**Output of multiplication:** (16,0,15)
**Common pre-adder format:** (16,4,11)
**Output:** (16,4,11)

**Figure:** Fixed point FIR - Single precision
Example of FIR filtering

![FIR Filter Diagram]

**Figure**: Fixed point FIR - Double precision

- **Input**: (16,0,15)
- **Coefficient**: (16,-1,16)
- **Output of multiplication**: (32,0,31)
- **Common pre-adder format**: (32,4,27)
- **Output**: (16,4,11)
Example of FIR filtering

![Diagram of FIR filter with fixed point representation](image)

**Figure:** Fixed point FIR - Double precision with 8 guard bits

- **Input:** \((16,0,15)\)
- **Coefficient:** \((16,-1,16)\)
- **Output of multiplication:** \((32,0,31)\)
- **Common pre-adder format:** \((36,4,31)\)
- **Output:** \((16,4,11)\)
Data width

In some components (ASIC or FPGA), data width is customizable: want to reduce data footprint

- Lower the energy consumption
- Enhance the available space

The lower the data width the better

What is the limit?

- Ensure that processing is done as we want
- Limitation of the quantization noise
- We define the Signal to Quantization noise

$$SNR_q = 10 \log_{10} \left( \frac{E[|y_{\text{floating}}|^2]}{E[|y_{\text{fixed}} - y_{\text{floating}}|^2]} \right)$$

- Purpose: maximizing SNR while reducing data width
Quantization noise

Data is quantified to a discrete level. Two consequences

- **Overflow**
- **Quantization noise**

**Overflow**

- From definition domain, output is

\[
f_D(x) = \begin{cases} 
  x & \forall x \in D_D \\
  D(x) & \forall x \notin D_D 
\end{cases}
\]  

(38)

- with \( D \) the overflow rule

**Quantization noise**

- Surjective mapping between input and output

\[
y_i = \frac{u_{i+1} - u_i}{2} = u_i + \frac{q}{2} \quad \forall x \in \Delta_i = [u_i; u_{i+1}]
\]

(39)

- Output noise defined as

\[
e = x - x_i
\]

(40)
Saturation:

\[
D(x) = \begin{cases} 
X_{\text{min}} & \forall \ x < X_{\text{min}} \\
X_{\text{max}} & \forall \ x > X_{\text{max}} 
\end{cases}
\]  

(41)

Modular rule

\[
D(x) = x \mod [X_{\text{max}} - X_{\text{min}}]
\]

(42)
Quantization noise

Rounding versus truncation

Figure: Rounding

Maximal error between 2 quantization steps

\[ y_i = \frac{u_{i+1} - u_i}{2} = u_i + \frac{q}{2} \quad \forall \ x \in \Delta_i = [u_i; u_{i+1}] \]  (43)
Quantization noise

Rounding versus truncation

Figure: Truncation (Absolute sign value)

- Maximal error before quantization steps
- Formulation with and without 2's complement
- Without 2's complement:

\[
y_i = \begin{cases} 
    u_i & \forall \ x > 0 \\
    u_{i+1} & \forall \ x < 0 
\end{cases}
\]  

(44)
Quantization noise

Rounding versus truncation

Figure: Truncation (2’s complement)

- Maximal error before quantization steps
- Formulation with and without 2’s complement
- With 2’s complement:

\[ y_i = u_i \] (45)
Noise model

- Quantization leads to noise
- Input is random, so is noise

How to model it?

\[ e[n] = x[n] - y[n] \]
Noise model

Noise quantization is added to the signal

\[ e[n] = x[n] - y[n] \]

Equivalent noise model:

![Diagram showing noise quantization and equivalent noise model](image)

Probability density function

![Probability density functions for rounding and truncation](image)
Noise model

Noise properties (assuming uncorrelated zero mean full scale stationary input)

1. Stationarity
2. Uncorrelation between signal and noise
3. Quantization noise is white
4. Quantization is bounded by quantization step
5. Probability density function is uniform
6. Noise is ergodic
Noise model

Probability density function

Rounding

- Mean: 0
- Variance: $\frac{q^2}{12}$

Truncation

- Mean: $\frac{q}{2}$
- Variance: $\frac{q^2}{3}$
Noise model

Format change: truncation

What happens if data format change?

- Some LSB bits are removed (assume truncation)
- Noise impact is more important (SNR decreases)

If we remove $k$ bits of fractional part

$$ e = \sum_{i=n-k}^{n} b_i 2^{-i} $$

(46)
Noise model

Format change: truncation

What happens if data format change?

- Some LSB bits are removed (assume truncation)
- Noise impact is more important (SNR decreases)

![Diagram showing the impact of truncation](image)

**Figure:** Impact of truncation

Equivalent to add a noise with the following properties

- **Mean:** $\frac{q}{2} \times (1 - 2^{-k})$
- **Variance:** $\frac{q^2}{12} \times (1 - 2^{-2k})$
Conclusion on data width

Need to calculate number of fractional part bits

- For some applications, number of bits is fixed
  - Data can be split or group to modulate data width
  - Arithmetic rules for propagation

- Other applications: number of bits is a freedom degree
  - Maximizing SNR while reducing number of bits

- Data is quantized in a digital application
  - Quantization noise can be modeled as a random process
  - Uniform white process
  - Different digitalization methods leads to different noise behaviour
Fixed point conclusion

- Fixed point design: sign, integer part and fractional part \((b,m,n)\)
- Baseline: floating point model

Sign

- Unsigned versus signed version
- Different way to encode sign data: Sign absolute value versus 2’s complement
- 2’s complement allows straightforward use of arithmetic and only value encodes 0
Fixed point conclusion

Exponent

- Encodes integer part
- Linked to dynamic of data
- Sign extension bit in many arithmetic operations to avoid overflow

Fractional part

- Encodes fractional
- Linked to output Signal to quantized Noise Ratio (SNR or SQNR)

Arithmetic rules

- Multiplication and additions have rules for FP design
- In practical systems, fixed point design corresponds to format calculation (Point positioning, data width)
IIR Synthesis
Create a arbitrary filter that respects the target specifications
Different methods can be used in order to built the filter

Synthesis methods

- Analog synthesis methods. From analog function $H(p)$, transformation $f$ is done to switch from $p$ plan to frequency plan in order to obtain $H(z)$. IIR structure is straightforward from $z$ structure.
  - See next point for different $f$ functions

- Direct synthesis in $z$ domain can also be done (but similar to first point)

- Digital optimisation methods can be used in order to find a $H(z)$. Cost functions are introduced to minimize the error (distortion between desired and obtained)
Analog filter specifications

Figure: Reminder of analog specification (and introduction of digital filtering)
Filter normalisation

First objective: obtain a normalized specifications: normalized template

- The cut-off pulsation is set to 1
- Attenuated bandwidth start after $1/s$
- Ripple and attenuation remains unchanged

$$|H_{Norm}(p)|$$

$$1-\delta_1$$

$$1+\delta_1$$

$$1$$

$$1-\delta_1$$

$$\delta_2$$

$$1$$

$$1/s$$

$$\omega$$

$$|H_{Norm}(p)|$$ (dB)

$$\Delta_1$$

$$0$$

$$-\Delta_1$$

$$\Delta_2$$

$$\omega$$

$$1$$

$$1/s$$

a) Normalized template (linear)  
b) Normalized template (log)

Figure: Prototype template (normalized)
From normalized template, deduce a $p$ transform with approximation functions

Filter prototype functions

1. Butterworth filters
2. Chebyshev filters
3. Elliptic filters
4. Bessel filters
Butterworth filters

Introduced in 1930 by Stephen Butterworth [13]
- Defined with a target length $N$
- The filter is defined from the square of its frequency response

\[ |H_B(\omega)|^2 = \frac{1}{1 + \left(\frac{\omega}{\omega_c}\right)^{2N}} \]  \hspace{1cm} (47)

Butterworth filters

Introduced in 1930 by Stephen Butterworth [13]

- Defined with a target length $N$
- The filter is defined from the square of its frequency response

\[ |H^B(\omega)|^2 = \frac{1}{1 + \left(\frac{\omega}{\omega_c}\right)^{2N}} \]  

(47)

Butterworth filters properties

- All pole filter
- Amplitude is a decreasing monolitic function of $\omega$
- Frequency response if as constant as possible in bandpass and attenuated band
- Maximal amplitude for $\omega = 0$ (with amplitude of 1)
- $\omega_c$ is the 3dB cut-off frequency
  - As $|H^B(\omega_c)| = \frac{1}{\sqrt{2}}$

Butterworth filters

Introduced in 1930 by Stephen Butterworth [13]

- Defined with a target length $N$
- The filter is defined from the square of its frequency response

\[ |H^B(\omega)|^2 = \frac{1}{1 + \left(\frac{\omega}{\omega_c}\right)^{2N}} \] (48)

Butterworth filters properties

- The attenuation in high frequency (asymptotic) is equal to $20 \times N$ per decades
- The filter order is given by the following relationship

\[ N \geq \frac{\log \left(\frac{1}{\delta_2 \sqrt{\delta_1}}\right)}{\log \left(\frac{1}{s}\right)} \] (49)

Butterworth filters

Figure: Butterworth filter in $s$ domain
Butterworth filters

Figure: Butterworth filter in s domain (zoom in bandpass)
Butterworth filter in analog domain

Well known filter used in analog domain [14]
- Various possible implementations
- Passive implementation with Cauer structure

![Figure: Passive implementation of Butterworth filter using Cauer structure](image)

- Corresponds to a cascade structure
- Value of capacitance and inductance depending on desired order

Butterworth filter in analog domain

Well known filter used in analog domain [14]

- Various possible implementations
- Active implementation with Sallen Key topology

![Active implementation of Butterworth filter using Sallen Key structure](image)

**Figure:** Active implementation of Butterworth filter using Sallen Key structure

- Each stage implement a conjugate pair of poles
- If the order is odd, need to add a RC circuits separately

Exercise: Butterworth specification

We consider the following LPF specification

- \( H_p(3kHz) \geq -0.2dB \)
- \( H_p(10kHz) \leq -40dB \)

Questions

1. Calculate the order of the Butterworth LPF filter
2. Deduce the cut-off frequency of the filter
Exercise: Butterworth specification

We recall the butterworth filter expression:

\[ |H^B(\omega)|^2 = \frac{1}{1 + \left(\frac{\omega}{\omega_c}\right)^{2N}} \]
Exercise: Butterworth specification

We recall the butterworth filter expression:

\[ |H^B(\omega)|^2 = \frac{1}{1 + \left(\frac{\omega}{\omega_c}\right)^{2N}} \]

Applied to the two specifications, we have

\[ |H_p(\omega)|^2 = \frac{1}{1 + \left(\frac{\omega_p}{\omega_c}\right)^{2N}} \]
\[ |H_a(\omega)|^2 = \frac{1}{1 + \left(\frac{\omega_a}{\omega_c}\right)^{2N}} \]
Exercise: Butterworth specification

We recall the butterworth filter expression:

\[ |H^B(\omega)|^2 = \frac{1}{1 + \left(\frac{\omega}{\omega_c}\right)^{2N}} \]

Applied to the two specifications, we have

\[ |H_p(\omega)|^2 = \frac{1}{1 + \left(\frac{\omega_p}{\omega_c}\right)^{2N}} \]
\[ |H_a(\omega)|^2 = \frac{1}{1 + \left(\frac{\omega_a}{\omega_c}\right)^{2N}} \]

We have then

\[ \frac{1}{|H_p(\omega)|^2} = 1 + \left(\frac{\omega_p}{\omega_c}\right)^{2N} \]
\[ \frac{1}{|H_a(\omega)|^2} = 1 + \left(\frac{\omega_a}{\omega_c}\right)^{2N} \]
Exercise: Butterworth specification

Equivalent to

\[
\left(\frac{\omega_p}{\omega_c}\right)^{2N} = \frac{1}{|H_p(\omega)|^2} - 1
\]

\[
\left(\frac{\omega_a}{\omega_c}\right)^{2N} = \frac{1}{|H_a(\omega)|^2} - 1
\]
Exercise: Butterworth specification

Equivalent to

\[
\left( \frac{\omega_p}{\omega_c} \right)^{2N} = \frac{1}{|H_p(\omega)|^2 - 1}
\]

\[
\left( \frac{\omega_a}{\omega_c} \right)^{2N} = \frac{1}{|H_a(\omega)|^2 - 1}
\]

Then,

\[
\left( \frac{\omega_p}{\omega_c} \right)^{2N} = \left( \frac{\omega_a}{\omega_c} \right)^{2N} = \frac{1}{|H_a(\omega)|^2 - 1} - 1
\]

\[
\left( \frac{\omega_p}{\omega_c} \right)^{2N} = \frac{1}{|H_p(\omega)|^2 - 1} - 1
\]

\[
\left( \frac{\omega_a}{\omega_c} \right)^{2N} = \frac{1}{|H_p(\omega)|^2 - 1} - 1
\]

\[
\left( \frac{\omega_p}{\omega_a} \right)^{2N} = \frac{1}{|H_a(\omega)|^2 - 1} - 1
\]
Exercise: Butterworth specification

We take the log to extract $n$

$$2n \log \left( \frac{\omega_p}{\omega_a} \right) = \log \left( \frac{1}{|H_p(\omega)|^2 - 1} \right) - \log \left( \frac{1}{|H_a(\omega)|^2 - 1} \right)$$
Exercise: Butterworth specification

We take the log to extract $n$

$$2n \log \left( \frac{\omega_p}{\omega_a} \right) = \log \left( \frac{1}{|H_p(\omega)|^2 - 1} \right) - \log \left( \frac{1}{|H_a(\omega)|^2 - 1} \right)$$

And finally $n$ is equal to

$$n = \frac{\log \left( \frac{1}{|H_p(\omega)|^2 - 1} \right) - \log \left( \frac{1}{|H_a(\omega)|^2 - 1} \right)}{2 \log \left( \frac{\omega_p}{\omega_a} \right)}$$
Exercise: Butterworth specification

We take the log to extract \( n \)

\[
2n \log \left( \frac{\omega_p}{\omega_a} \right) = \log \left( \frac{1}{|H_p(\omega)|^2 - 1} \right) - \log \left( \frac{1}{|H_a(\omega)|^2 - 1} \right)
\]

And finally \( n \) is equal to

\[
n = \frac{\log \left( \frac{1}{|H_p(\omega)|^2 - 1} \right) - \log \left( \frac{1}{|H_a(\omega)|^2 - 1} \right)}{2 \log \left( \frac{\omega_p}{\omega_a} \right)}
\]

Numeric application

- We find \( n = 5.093 \)
- We choose \( n = 6 \)
Exercise: Butterworth specification

Calculating the cut-off frequency

- We have 2 specifications, which will give 2 different cut-off frequencies
- The cut-off frequency will be calculated as the geometric mean of the 2 cut-off frequencies

\[ \omega_c = \frac{\omega}{\left( \frac{1}{|H|^2} - 1 \right)^{\frac{1}{2n}}} \]

\( \omega_{fa} = 4.64 \text{ kHz} \)
\( \omega_{fp} = 3.87 \text{ kHz} \)

The cut-off frequency can be finally calculated as

\[ \omega_c = \frac{\omega_{fa} + \omega_{fp}}{2} \]

\[ \omega_c = 4.24 \text{ kHz} \]
Exercise: Butterworth specification

Calculating the cut-off frequency

- We have 2 specifications, which will give 2 different cut-off frequencies
- The cut-off frequency will be calculated as the geometric mean of the 2 cut-off frequencies

\[ \omega_c = \frac{\omega}{\left(\frac{1}{|H|^2} - 1\right)^{\frac{1}{2n}}} \]

Which leads to

\[
\begin{align*}
&f_c^a = 4.64\,kHz \\
&f_c^p = 3.87\,kHz
\end{align*}
\]
Exercise: Butterworth specification

Calculating the cut-off frequency

- We have 2 specifications, which will give 2 different cut-off frequencies
- The cut-off frequency will be calculated as the geometric mean of the 2 cut-off frequencies

\[ \omega_c = \frac{\omega}{\left( \frac{1}{|H|^2} - 1 \right)^{\frac{1}{2n}}} \]

Which leads to

\[
\begin{cases}
  f_c^a &= 4.64\, kHz \\
  f_c^p &= 3.87\, kHz
\end{cases}
\]

The cut-off frequency can be finally calculated as

\[ f_c = \sqrt{f_c^a f_c^p} = 4.24\, kHz \]
Chebyshev filters - Type I

- Filters named from Pafnuty Chebyshev as the filter is derived from Chebyshev polynomials [15]
- Also defined by the square of the frequency response

$$|H^C(\omega)|^2 = \frac{1}{1 + \epsilon^2 T_N^2 \left( \frac{\omega}{\omega_c} \right)}$$

(50)

- $\omega_c$ is the cut-off frequency
- $\epsilon$ is a parameter that controls ripple amplitude
- $T_N(\cdot)$ is the $N$ order Chebyshev polynomial

Chebyshev filters minimize the error between the idealized and the actual filter characteristic over the range of the filter but with ripples in the passband.

Chebyshev filters - Type I

- Filters named from Pafnuty Chebyshev as the filter is derived from Chebyshev polynomials [15]
- Also defined by the square of the frequency response

\[ |H^C(\omega)|^2 = \frac{1}{1 + \epsilon^2 T_N^2 \left( \frac{\omega}{\omega_c} \right)}, \]  

(50)

- \( \omega_c \) is the cut-off frequency
- \( \epsilon \) is a parameter that controls ripple amplitude
- \( T_n(\cdot) \) is the \( N \) order Chebyshev polynom

- Chebyshev filters minimize the error between the idealized and the actual filter characteristic over the range of the filter but with ripples in the passband

Chebyshev filters - Type I

Chebyshev properties

- Chebyshev polynom definition (first kind) [16]

\[ T_0(x) = 1, \ T_1(x) = x, \ T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x). \]

- For a type I, the amplitude in the bandpass as a ripple expressed as

\[ 0 \leq \omega \leq \omega_c \implies \frac{1}{1 + \epsilon^2} \leq |H^C(\omega)|^2 \leq 1 \]

Chebyshev filters - Type I

Chebyshev properties

- Chebyshev polynom definition (first kind) [16]

\[ T_0(x) = 1, \quad T_1(x) = x, \quad T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x). \]

- For a type I, the amplitude in the bandpass as a ripple expressed as

\[ 0 \leq \omega \leq \omega_c \implies \frac{1}{1 + \epsilon^2} \leq |H^C(\omega)|^2 \leq 1 \]

- \( \epsilon \) is thus closely related to the effective ripple as

\[ \epsilon = \sqrt{10^{\frac{\delta}{10}}} - 1 \]

- Zeros of the filter can be analytically expressed

\[ s_{pm}^\pm = \pm \sinh \left( \frac{1}{n} \text{arsinh} \left( \frac{1}{\epsilon} \right) \right) \sin(\theta_m) + j \cosh \left( \frac{1}{n} \text{arsinh} \left( \frac{1}{\epsilon} \right) \right) \cos(\theta_m) \]
Chebyshev filters - Type I

Chebyshev Type I filters (1dB)

Gain [dB]

1/s
Chebyshev filters - Type I

Chebyshev Type I filters (0.1 dB)
Chebyshev filters - Type I

Chebyshev Type I filters (1dB)

Gain [dB] vs. 1/s
Chebyshev filters - Type I

Chebyshev Type I filters (0.1 dB)
Chebyshev filters - Type II

- Filters named from Pafnuty Chebyshev as the filter is derived from Chebyshev polynomials [15]
- Type II are also called inverse Chebyshev filters
- Also defined by the square of the frequency response

\[ |H_C(\omega)|^2 = \frac{1}{1 + \frac{1}{\epsilon^2 T_N^2(\frac{\omega}{\omega_c})}}, \tag{51} \]

\( \omega_c \) is the cutoff frequency
\( \epsilon \) is a parameter that controls ripple amplitude
\( T_n(\cdot) \) is the \( N \) order Chebyshev polynomial

Chebyshev filters - Type II

Filters named from Pafnuty Chebyshev as the filter is derived from Chebyshev polynomials [15]

Type II are also called inverse Chebyshev filters

Also defined by the square of the frequency response

\[
|H^C(\omega)|^2 = \frac{1}{1 + \frac{1}{\epsilon^2 T^2_N(\omega/\omega_c)}}, \quad (51)
\]

- \(\omega_c\) is the cut-off frequency
- \(\epsilon\) is a parameter that controls ripple amplitude
- \(Tn(\cdot)\) is the \(N\) order Chebyshev polynom

Chebyshev filters minimize the error between the idealized and the actual filter characteristic over the range of the filter but with ripples in the stopband

Elliptic filters

- Butterworth filters: No ripple
- Chebyshev filters have ripple in bandpass
- Elliptic filters have ripples in both bandpass and attenuated bandwidth [17]

Elliptic filters

- Ripple in bandpass and attenuated bandwidth
- Optimal in terms of selectivity (minimal order for a given spec)
- Also defined by the square of the frequency response

\[ |H^E(\omega)|^2 = \frac{1}{1 + \epsilon^2 R_N^2 \left(\frac{\omega}{\omega_c}\right)}, \]

Elliptic filters

Elliptic filter expression

\[ |H^E(\omega)|^2 = \frac{1}{1 + \epsilon^2 R_N^2(\xi, \frac{\omega}{\omega_c})}, \]

- \( R_N(x, y) \) is a rational Chebyshev function with order \( N \)
- \( \epsilon \) and \( \varepsilon \) are parameters that control ripple amplitudes
- Ripple and bandpass and attenuated bandwidth can be independently monitored
- If \( \varepsilon \) tends to 0, elliptic tends to be a order I Chebyshev
- The gain in the bandpass varies between 1 and \( \frac{1}{\sqrt{1 + \epsilon}} \)
- The attenuation in the attenuated bandwidth varies between \( -\infty \) and \( \frac{1}{\sqrt{1 + \epsilon^2 L_n^2}} \) with \( L_n \) (discrimination factor) defined as

\[ L_n = R_N(\varepsilon, \xi) \]
Elliptic filters

Figure: Elliptic profile and ripple parameters

\[ G = \frac{1}{\sqrt{1 + \varepsilon^2}} \]

\[ G = \frac{1}{\sqrt{1 + \varepsilon^2 L_n^2}} \]
Rational Chebyshev function

\[ R_n(\xi, x) \equiv \text{cd} \left( n \frac{K(1/L_n)}{K(1/\xi)} \text{cd}^{-1}(x, 1/\xi), 1/L_n \right) \]

- With \( \text{cd} \) defined as the **Jacobi elliptic cosine** function
- \( K() \) is a **complete elliptic integral of the first kind**

\[ K(k) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1 - k^2 \sin^2 \theta}} = \int_0^1 \frac{dt}{\sqrt{(1 - t^2)(1 - k^2 t^2)}} \]
**Figure:** Elliptic filter with 1dB and 50dB rejection
Elliptic filters

Figure: Elliptic filter with 0.1dB and 30dB rejection
Elliptic filters

Elliptic filter bandpass (0.1dB/30dB)

Figure: Elliptic filter with 1dB and 50dB rejection in bandpass
Elliptic filters

Figure: Elliptic filter with 0.1dB and 30dB rejection in bandpass
Bessel filters

- Introduced with Thomson (sometimes called Thomson filters) [18]
- Linear filter with a maximally flat group/phase delay
- It preserves the signals in the passband.
- Often used in audio processing
- But not a good selectivity!
- Defined directly with the transfer function

\[ H(s) = \frac{\theta_n(0)}{\theta_n(s/\omega_0)} \]  (53)

- \( \theta_n(x) \) is a reverse Bessel polynomial
- Again, all pole filter

Bessel filters

First Bessel polynomials

1. $s + 1$
2. $s^2 + 3s + 3$
3. $s^3 + 6s^2 + 15s + 15$
4. $s^4 + 10s^3 + 45s^2 + 105s + 105$
5. $s^5 + 15s^4 + 105s^3 + 420s^2 + 945s + 945$
Bessel filters

First Bessel polynomials

1. \( s + 1 \)
2. \( s^2 + 3s + 3 \)
3. \( s^3 + 6s^2 + 15s + 15 \)
4. \( s^4 + 10s^3 + 45s^2 + 105s + 105 \)
5. \( s^5 + 15s^4 + 105s^3 + 420s^2 + 945s + 945 \)

In general case, we have

\[
\theta_n(s) = \sum_{k=0}^{n} a_k s^k
\]

where

\[
a_k = \frac{(2n - k)!}{2^{n-k} k! (n - k)!} \quad k = 0, 1, \ldots, n
\]
Bessel filters

Figure: Bessel filter in fullband
Bessel filters

Figure: Bessel filter in bandpass
Bessel filters

![Bessel filter phase in bandpass](image)

**Figure:** Bessel filter phase in bandpass
Bessel filters

Figure: Bessel filter phase in fullband
Un-normalization

The obtained transfer function is normalized

- Need to switch from normalized $H_{\text{norm}}(p_N)$ to $H(p)$

Need to introduce a function for variable switching

- Depends on filter nature

<table>
<thead>
<tr>
<th>LPF</th>
<th>HPF</th>
<th>BPF</th>
<th>SBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_N = \frac{p}{\omega_c}$</td>
<td>$p_N = \frac{\omega_c}{p}$</td>
<td>$p_N = \frac{1}{B} \left( \frac{p}{\omega_0} + \frac{\omega_0}{p} \right)$</td>
<td>$p_N = \left[ \frac{1}{B} \left( \frac{p}{\omega_0} + \frac{\omega_0}{p} \right) \right]^{-1}$</td>
</tr>
</tbody>
</table>

Table: Unormalization function
Figure: Reminder of analog specification (and introduction of digital filtering)
Impulse invariance method

Filter we have defined are **analog** filters.

**Impulse invariance method**

Filter in digital domains corresponds to analog filter sampled at the desired sampling frequency

\[ h[n] = h(nTc) = h(t)_{t=nTc} \]  \hspace{1cm} (54)

![Diagram of analog and digital filters](image)
Impulse invariance method

Assuming that the filter is an all pole one, we can deduce the digital poles from the analog poles.

**Analog poles**

\[ H_a(p) = \frac{N(p)}{D(p)} = \sum_{k=0}^{L-1} \frac{h_k}{p + p_k}, \]

\[ h_a(t) = \sum_{k=0}^{L-1} h_k e^{-p_k t} \quad \text{with inverse LT} \]

**Digital poles**

\[ h[nT] = \sum_{k=0}^{L-1} h_k e^{-p_k k T}, \quad H(z) = \sum_{k=0}^{L-1} h_k \frac{z}{z - e^{-p_k T}} = \sum_{k=0}^{L-1} h_k \frac{1}{1 - e^{-p_k T} z^{-1}} \]
Example: simple LPF filter

If we consider a simple LPF filter expressed as

\[ H(p) = \frac{1}{a + p} \]
Example: simple LPF filter

If we consider a simple LPF filter expressed as

\[ H(p) = \frac{1}{a + p} \]

We obtain in time domain (inverse Laplace transform)

\[ h(t) = u(t) \times e^{-at} \]
Example: simple LPF filter

If we consider a simple LPF filter expressed as

$$H(p) = \frac{1}{a + p}$$

We obtain in time domain (inverse Laplace transform)

$$h(t) = u(t) \times e^{-at}$$

Which leads to the analog response (function of sampled data)

$$h_e(t) = T_c \sum_{k=0}^{\infty} u(kT_c) \times e^{-akT_c} \delta(T - kT_c)$$
Example: simple LPF filter

If we consider a simple LPF filter expressed as

\[ H(p) = \frac{1}{a + p} \]

We obtain in time domain (inverse Laplace transform)

\[ h(t) = u(t) \times e^{-at} \]

Which leads to the analog response (function of sampled data)

\[ h_e(t) = T_c \sum_{k=0}^{\infty} u(kT_c) \times e^{-akT_c} \delta(T - kT_c) \]

Which leads to IIR expression (Z transform)

\[ H_e(z) = T_c \sum_{k=0}^{\infty} e^{-akT_c} z^{-k} = T_c \sum_{k=0}^{\infty} (e^{-aT_c} z)^{-k} \]

\[ H_e(z) = T_c \frac{z}{z - e^{-aT_c}} \]
Impulse invariance method

Frequency response

\[ H(e^{j\omega T}) = \left. H(z) \right|_{z=e^{j\omega T}} = \frac{1}{T} \sum_{k} H_a(j\omega + j\frac{2\pi k}{T}) \]  

(55)

- **Aliasing in frequency domain**
  - Spectrum is folded for frequency not in \([\frac{-1}{2T_c}; \frac{1}{2T_c}]\)
  - Only valid if analog filter spectrum is null in this interval
Impulse invariance method

Frequency response

\[ H(e^{j\omega T}) = H(z)\bigg|_{z=e^{j\omega T}} = \frac{1}{T} \sum_{k} H_a(j\omega + j\frac{2\pi k}{T}) \]  

(55)

- Aliasing in frequency domain
  - Spectrum is folded for frequency not in \([\frac{-1}{2T_c}; \frac{1}{2T_c}]\)
  - Only valid if analog filter spectrum is null in this interval

Requirement:

\[ |H_a(j\omega)| \simeq 0 \text{ for } |\omega| > \omega_B \text{ avec } \omega_B < \frac{\pi}{T_c} \]

- Scaling factor \(\frac{1}{T}\) in PSD expression: need a normalisation factor to recover analog response
Impulse invariance method

Figure: Aliasing consequence in frequency domain if the filter is not bandlimited (impulse invariance method)
Euler transformation

Impulse invariance method is a simple efficient method for IIR synthesis

- Ensure stability and direct sampling method
- Risk of frequency domain aliasing
Euler transformation

Impulse invariance method is a simple efficient method for IIR synthesis

✔ Ensure stability and direct sampling method

❌ Risk of frequency domain aliasing

Reason

- The link between Laplace (continuous) and Z transform (discrete) is not a bijection.
- \( p \to z = e^{pT} \) is a surjection of the Laplace transform convergence area
Euler transformation

Impulse invariance method is a simple efficient method for IIR synthesis

- Ensure stability and direct sampling method
- Risk of frequency domain aliasing

Reason

- The link between Laplace (continuous) and Z transform (discrete) is not a bijection.
- \( p \rightarrow z = e^{pT} \) is a surjection of the Laplace transform convergence area

Euler transform principle

Use a algebraic transformation between \( p \) and \( z \) with derivation approximation

\[
\frac{de(t)}{dt} \bigg|_{t=nT} = \lim_{\Delta t \to 0} \frac{\Delta e(nT)}{\Delta t} \simeq \frac{e[nT] - e[(n-1)T]}{T}
\]  

(56)
Euler transformation

We assume that an output $s$ is obtained from $e$ with this approximation

$$s(nT) = \frac{e[nT] - e[(n - 1)T]}{T}$$
Euler transformation

We assume that an output $s$ is obtained from $e$ with this approximation

$$s(nT) = \frac{e[nT] - e[(n - 1)T]}{T}$$

With $z$ transform, it leads to:

$$S(z) = \frac{1 - z^{-1}}{T} E(z)$$
Euler transformation

We assume that an output $s$ is obtained from $e$ with this approximation

$$s(nT) = \frac{e[nT] - e[(n-1)T]}{T}$$

With $z$ transform, it leads to:

$$S(z) = \frac{1 - z^{-1}}{T} E(z)$$

This *derivation* approximation is the filter $H(z)$ defined as

$$H(z) = \frac{1 - z^{-1}}{T}$$
Euler transformation

We assume that an output $s$ is obtained from $e$ with this approximation

$$s(nT) = \frac{e[nT] - e[(n-1)T]}{T}$$

With z transform, it leads to:

$$S(z) = \frac{1 - z^{-1}}{T} E(z)$$

This derivation approximation is the filter $H(z)$ defined as

$$H(z) = \frac{1 - z^{-1}}{T}$$

Besides, in $p$ plan, derivation is defined as

$$H_a(p) = p$$
Euler transformation

We assume that an output $s$ is obtained from $e$ with this approximation

$$ s(nT) = \frac{e[nT] - e[(n-1)T]}{T} $$

With $z$ transform, it leads to:

$$ S(z) = \frac{1 - z^{-1}}{T} E(z) $$

This derivation approximation is the filter $H(z)$ defined as

$$ H(z) = \frac{1 - z^{-1}}{T} $$

Besides, in $p$ plan, derivation is defined as

$$ H_a(p) = p $$

Se we get

$$ z = \frac{1}{1 - pT} \quad (57) $$
Euler transformation

From this link, we can deduce link between digital pulsation $\omega$ and analog pulsations $\omega_a$

$$z = \frac{1}{1 - j\omega_a T} = e^{j\omega T}$$

$$z = \frac{1}{2} \left[ 1 + \frac{1 + j\omega_a T}{1 - j\omega_a T} \right] = \frac{1}{2} \left[ 1 + e^{j2\arctan(\omega_a T)} \right]$$

- Euler transformation changes $z$ trajectory (radius=$\frac{1}{2}$ and center at $\frac{1}{2}$)
- Changes the frequency response of the filter
- Still ensures stability
- No spectrum aliasing

Better transformation (shape in frequency domain) is possible: bilinear transform
Bilinear transform

Same approach with a different kernel function: Integer approach

- Integer approximated with rectangles
- Bilinear transform is special case of Möbius transformation [19],

\[ s(nT) = s((n - 1)T) + T \frac{e(nT) + e((n - 1)T)}{2} \]  

(58)


Figure: Integer approximation with rectangle method
Bilinear transform

With same approach as previously, we can map the integer with a filter defined in $z$ domain (digital filter)

$$S(z) = z^{-1}S(z) + T \frac{E(z) + z^{-1}E(z)}{2}$$

$$H(z) = \frac{S(z)}{E(z)} = \frac{T}{2} \frac{1 + z^{-1}}{1 - z^{-1}}$$

Besides, in $p$-plan, integration is defined as

$$H_a(p) = \frac{1}{p}$$

So we have a $p$ to $z$ transformation defined as

$$p = \frac{1}{1 - z^{-1}} \left(1 + z^{-1}\right)$$

And a $z$ to $p$ transformation

$$z = \frac{2}{T} + p \frac{2}{T} - p = \frac{2}{T} + j\omega_a$$

$$H(z) = \frac{S(z)}{E(z)} = \frac{T}{2} \frac{1 + z^{-1}}{1 - z^{-1}}$$
Bilinear transform

With same approach as previously, we can map the integer with a filter defined in $z$ domain (digital filter)

\[ S(z) = z^{-1}S(z) + T\frac{E(z) + z^{-1}E(z)}{2} \]

\[ H(z) = \frac{S(z)}{E(z)} = \frac{T}{2} \frac{1 + z^{-1}}{1 - z^{-1}} \]

Besides, in $p$ plan, integration is defined as

\[ H_a(p) = \frac{1}{p} \]
Bilinear transform

With same approach as previously, we can map the integer with a filter defined in $z$ domain (digital filter)

$$S(z) = z^{-1}S(z) + T \frac{E(z) + z^{-1}E(z)}{2}$$

$$H(z) = \frac{S(z)}{E(z)} = \frac{T}{2} \frac{1 + z^{-1}}{1 - z^{-1}}$$

Besides, in $p$ plan, **integration** is defined as

$$H_a(p) = \frac{1}{p}$$

So we have a $p$ to $z$ transformation defined as

$$p = \frac{1 - z^{-1}}{1 + z^{-1}} \quad (59)$$
Bilinear transform

With same approach as previously, we can map the integer with a filter defined in \( z \) domain (digital filter)

\[
S(z) = z^{-1}S(z) + T \frac{E(z) + z^{-1}E(z)}{2}
\]

\[
H(z) = \frac{S(z)}{E(z)} = \frac{T}{2} \frac{1 + z^{-1}}{1 - z^{-1}}
\]

Besides, in \( p \) plan, **integration** is defined as

\[
H_a(p) = \frac{1}{p}
\]

So we have a \( p \) to \( z \) transformation defined as

\[
p = \frac{1 - z^{-1}}{1 + z^{-1}}
\] (59)

And a \( z \) to \( p \) transformation

\[
z = \frac{2/T + p}{2/T - p} = \frac{2/T + j\omega_a}{2/T - j\omega_a}
\] (60)
Bilinear transform

Link between analog and digital sequences can be derived

\[
\begin{align*}
  z &= \frac{2 + j\omega a T}{2 - j\omega a T} = e^{j\omega T} \\
  |z| &= 1 \\
  \text{arg}(z) &= \arctan\left(\frac{\omega a T}{2}\right) + \arctan\left(\frac{\omega a T}{2}\right) = 2 \arctan\left(\frac{\omega a T}{2}\right) \\
  z &= e^{j\omega T} \Rightarrow \text{arg}(z) = \omega T
\end{align*}
\]
Bilinear transform

Link between analog and digital sequences can be derived

\[ z = \frac{2 + j\omega_a T}{2 - j\omega_a T} = e^{j\omega T} \]

\[ |z| = 1 \]

\[ \arg(z) = \arctan\left(\frac{\omega_a T}{2}\right) + \arctan\left(\frac{\omega_a T}{2}\right) = 2 \arctan\left(\frac{\omega_a T}{2}\right) \]

\[ z = e^{j\omega T} \Rightarrow \arg(z) = \omega T \]

And we finally have

\[ \frac{\omega_a T}{2} = \tan\left(\frac{\omega_d T}{2}\right) \] (61)

Every point on the unit circle in the discrete-time filter z-plane, \( z = e^{j\omega_d T} \) is mapped to a point on the \( j\omega \) axis on the continuous-time filter s-plane, \( s = j\omega_a \)

Equivalently, continuous frequency range \(-\infty < \omega_a < \infty \) is mapped onto the fundamental frequency interval \(-\frac{\pi}{T} < \omega_d < \frac{\pi}{T} \)
Transformation is bijective between $\Re\{p\} < 0$ and unity circle: maintain stability

- Frequency is distorted with the transformation but this ensures non aliasing
This distortion can be critical to maintain filter specifications

In this case, we compensate (pre-distortion) this critical frequencies to maintain the filter specifications

Figure: Analog to digital pulsations link
Bilinear transform

Figure: Pre-distortion principle
Bilinear transform

Bilinear transform method for filter creation

1. Have an initial filter specification in analog domain
Bilinear transform

Bilinear transform method for filter creation

1. Have an initial filter specification in analog domain

2. From filter specifications, apply the pre-distortion on critical frequencies (i.e. frequencies in bandpass)

\[
\omega_{a p,a} = \frac{2}{T} \tan \left( \frac{\Omega_{p,a}}{2} \right),
\]  \hspace{1cm} (62)

Digital Signal processing
IIR synthesis
Transformation functions
Bilinear transform method for filter creation

1. Have an initial filter specification in analog domain

2. From filter specifications, apply the pre-distortion on critical frequencies (i.e., frequencies in bandpass)

\[ \omega_{a,p,a} = \frac{2}{T} \tan \left( \frac{\Omega_{p,a}}{2} \right), \quad (62) \]

3. Getting analog filter from appropriate method (see before)
Bilinear transform method for filter creation

1. Have an initial filter specification in analog domain

2. From filter specifications, apply the pre-distortion on critical frequencies (i.e. frequencies in bandpass)

\[ \omega_{a\,p,a} = \frac{2}{T} \tan \left( \frac{\Omega_{p,a}}{2} \right), \]  

(62)

3. Getting analog filter from appropriate method (see before)

4. using the bilinear transform with \( p \) to \( z \) transform

\[ H(z) \triangleq H(p) \bigg|_{p=\frac{2}{T} \frac{1-z^{-1}}{1+z^{-1}}} \]  

(63)
Conclusion on IIR synthesis

Designing an IIR filter is difficult as the filter should be function of input and output.
- But IIR filter is closed to analog filter.
- Well known methods to design a specific filter based on different approaches.

Filter prototype functions:

1. Butterworth filters
2. Chebyshev filters
3. Elliptic filters
4. Bessel filters
Conclusion on IIR synthesis

From analog filter specification, we need to derive a digital filter. Two main methods for filter transformation:

1. Impulse invariance
2. Bilinear transform
Conclusion on IIR synthesis

From analog filter specification, we need to derive a digital filter. Two main methods for filter transformation:

1. Impulse invariance
2. Bilinear transform

**Impulse invariance**

- Maintain response in time domain, stability preserved
- Creates aliasing in frequency domain (no $p$ to $z$ bijection)
- Straightforward transformation
Conclusion on IIR synthesis

From analog filter specification, we need to derive a **digital filter**. Two main methods for filter transformation:

1. **Impulse invariance**
2. **Bilinear transform**

**Impulse invariance**

- Maintain response in time domain, stability preserved
- Creates aliasing in frequency domain (no $p$ to $z$ bijection)
- Straightforward transformation

**Bilinear transformation**

- Transformation of analog filter with $p = \frac{2}{T} \frac{1-z^{-1}}{1+z^{-1}}$
- No aliasing and stability preserved
- Designing the filter with **pre-distortion** to maintain specifications
FIR Synthesis
We recall first what a FIR filtering is

- Finite response in time domain
  - Applied on a digital input \( x[n] \)
  - With a depth \( L \)

- Equivalent formulation: output only depends on input

\[
y[n] = \sum_{k=0}^{L-1} h_k x[n - k]
\]

\[
H(z) = \sum_{k=0}^{L-1} h_k z^{-k}
\]

- These filters can only exists in digital domain

- Synthesis methods can not be inherited from analog ones
Introduction

FIR synthesis methods
- Based on signal processing approaches
- Different methods to enable a specific design

FIR Filtering synthesis
1. Windowing method
2. Frequency sampling
3. Optimisation methods
Linear phase filters

Phase is important to monitor a filter
- Phase can not be equal to zero for causal systems
- We may want to have minimal impact of the phase terms

Linear phase
- Constant group delay versus frequency
- Each frequency is delayed with the same amount (coefficient of the linear phase)
- No frequency distortion
Linear phase filters

Phase is important to monitor a filter

- Phase can not be equal to zero for causal systems
- We may want to have minimal impact of the phase terms

Linear phase

- Constant group delay versus frequency
- Each frequency is delayed with the same amount (coefficient of the linear phase)
- No frequency distortion

Impact on filter:

\[ H(e^{j\Omega}) = A(e^{j\Omega})e^{-j\alpha\Omega} + j\beta, \]  (64)
Linear phase filters

Linear phase filter

\[ H(e^{j\Omega}) = A(e^{j\Omega})e^{-j\alpha\Omega + j\beta}, \]

- \( A(e^{j\Omega}) \) is called pseudo-module (module without taking off the sign)
- The phase is \( \varphi(\Omega) = -\alpha\Omega + \beta \)
  - The phase is linear
- The group delay is calculated as \( \tau g(\Omega) = -\frac{d\varphi(\Omega)}{d\Omega} \) is a constant equal to \( \alpha \)
Linear phase filters

Linear phase filter

\[ H(e^{j\Omega}) = A(e^{j\Omega})e^{-j\alpha\Omega + j\beta}, \]

- \( A(e^{j\Omega}) \) is called pseudo-module (module without taking off the sign)
- The phase is \( \varphi(\Omega) = -\alpha\Omega + \beta \)
  - The phase is linear
- The group delay is calculated as \( \tau_g(\Omega) = -\frac{d\varphi(\Omega)}{d\Omega} \) is a constant equal to \( \alpha \)

Linear phase properties

A linear phase filter is defined with

- group delay \( \alpha \)
- Offset phase \( \beta \)
- The depth \( L \) (number of filter taps)

How to built a linear phase filter ?
Linear phase filters

Calculating influence of linear phase filter

\[ H(z) = \sum_{k=0}^{L-1} h_k z^{-k} \]
Linear phase filters

Calculating influence of linear phase filter

\[ H(z) = \sum_{k=0}^{L-1} h_k z^{-k} \]

\[ H(e^{j\Omega}) = H(z)|_{z=e^{j\Omega}} = \sum_{k=0}^{L-1} h_k e^{-jk\Omega} \]
Calculating influence of linear phase filter

\[ H(z) = \sum_{k=0}^{L-1} h_k z^{-k} \]

\[ H(e^{j\Omega}) = H(z) \big|_{z=e^{j\Omega}} = \sum_{k=0}^{L-1} h_k e^{-jk\Omega} \]

\[ H(e^{j\Omega}) = \Re[H(e^{j\Omega})] + j\Im[H(e^{j\Omega})] \]
Linear phase filters

Calculating influence of linear phase filter

\[ H(z) = \sum_{k=0}^{L-1} h_k z^{-k} \]

\[ H(e^{j\Omega}) = H(z)|_{z=e^{j\Omega}} = \sum_{k=0}^{L-1} h_k e^{-jk\Omega} \]

\[ H(e^{j\Omega}) = \Re[H(e^{j\Omega})] + j\Im[H(e^{j\Omega})] \]

\[ H(e^{j\Omega}) = |H(e^{j\Omega})| e^{j\text{arg}(\Omega)} = A(e^{j\Omega}) e^{j\varphi(\Omega)} \]

with \( \varphi(\Omega) = -\alpha\Omega + \beta \)
Linear phase filters

$z$ transform and linear phase filter:

$$H(e^{j\Omega}) = |H(e^{j\Omega})|e^{j\text{arg}(\Omega)} = A(e^{j\Omega})e^{j\varphi(\Omega)}$$

with $\varphi(\Omega) = -\alpha\Omega + \beta$

First case: $\varphi(\Omega) = -\alpha\Omega$

- $\beta = 0$
- $-\pi \leq \Omega \leq \pi$

Second case: $\varphi(\Omega) = \beta - \alpha\Omega$

- $\beta \neq 0$
- $-\pi \leq \Omega \leq \pi$
Linear phase filters

First case: \( \varphi(\Omega) = -\alpha \Omega \)

We express the system output in z domain

\[
H(e^{j\Omega}) = |H(e^{j\Omega})|e^{j \arg(\Omega)} = A(e^{j\Omega})e^{j\varphi(\Omega)}
\]

With Euler formula, it comes

\[
H(e^{j\Omega}) = A(e^{j\Omega})e^{-j\alpha \Omega} = A(e^{j\Omega})[\cos \alpha \Omega - j \sin \alpha \Omega]
\]

(65)

\[
H(e^{j\Omega}) = \sum_{k=0}^{L-1} h_k e^{-jk\Omega} = \sum_{k=0}^{L-1} h_k [\cos k\Omega - j \sin k\Omega]
\]

(66)

By identifying (65) with (66) we obtain the 2 equations that link time domain response and linear phase impact:

\[
\begin{cases}
A(e^{j\Omega}) \cos \alpha \Omega = \sum_{k=0}^{L-1} h_k \cos k\Omega \\
A(e^{j\Omega}) \sin \alpha \Omega = \sum_{k=0}^{L-1} h_k \sin k\Omega
\end{cases}
\]
First case: $\varphi(\Omega) = -\alpha \Omega$

We have linked $z$ transform and time domain response:

\[
\begin{align*}
A(e^{j\Omega}) \cos \alpha \Omega &= \sum_{k=0}^{L-1} h_k \cos k\Omega \\
A(e^{j\Omega}) \sin \alpha \Omega &= \sum_{k=0}^{L-1} h_k \sin k\Omega 
\end{align*}
\] (67)

And we do (67).a $\times \sin \alpha \Omega$ - (67).b $\times \cos \alpha \Omega$ .

We have

\[
\sum_{k=0}^{L-1} h_k \cos k\Omega \times \sin \alpha \Omega - \sum_{k=0}^{L-1} h_k \sin k\Omega \times \cos \alpha \Omega = 0
\]
Linear phase filters

First case: \( \varphi(\Omega) = -\alpha \omega \)

We have linked \( z \) transform and time domain response:

\[
\begin{align*}
A(e^{j\Omega}) \cos \alpha \Omega &= \sum_{k=0}^{L-1} h_k \cos k\Omega \\
A(e^{j\Omega}) \sin \alpha \Omega &= \sum_{k=0}^{L-1} h_k \sin k\Omega
\end{align*}
\]

(67)

And we do (67).a \times \sin \alpha \Omega - (67).b \times \cos \alpha \Omega.

We have

\[
\sum_{k=0}^{L-1} h_k \cos k\Omega \times \sin \alpha \Omega - \sum_{k=0}^{L-1} h_k \sin k\Omega \times \cos \alpha \Omega = 0
\]

Which leads to

\[
\sum_{k=0}^{L-1} h_k \sin ((\alpha - k)\Omega) = 0
\]

(68)
Linear phase filters

First case: $\varphi(\Omega) = -\alpha \Omega$

To obtain a linear phase filter in the first case, the impulse response should be verified

$$\sum_{k=0}^{L-1} h_k \sin ((\alpha - k) \Omega) = 0$$

(69)

If $\alpha \neq 0$ it is equivalent to

$$\begin{cases} 
\alpha = \frac{L-1}{2} \\
h_k = h_{L-1-k} \text{ for } 0 \leq k \leq \alpha 
\end{cases}$$

(70)

The impulse response is symmetric with respect to half the depth of the filter

- But depends on the parity of $L$
Linear phase filters

Second case: \( \varphi(\Omega) = \beta - \alpha \Omega \)

To obtain a linear phase filter in the first case, the impulse response should verify

\[
\sum_{k=0}^{L-1} h_k \sin ((\alpha - k)\Omega + \beta) = 0
\]

(71)

If \( \alpha \neq 0 \) it is equivalent to

\[
\begin{cases}
\beta = \pm \frac{\pi}{2} \\
\alpha = \frac{L-1}{2} \\
h_k = h_{L-1-k} \text{ for } 0 \leq k \leq \alpha
\end{cases}
\]

(72)

The impulse response is **antisymmetric** with respect to half the depth of the filter

- But depends on the parity of \( L \)
Linear phase filters

We have 4 cases

1. $\beta = 0$ and $L$ is odd: Type I
2. $\beta = 0$ and $L$ is even: Type II
3. $\beta = \pm \frac{\pi}{2}$ and $L$ is odd: Type III
4. $\beta = \pm \frac{\pi}{2}$ and $L$ is even: Type IV
Linear phase filters

Type I linear phase filter

- $\beta = 0 \quad \varphi(\Omega) = -\alpha \Omega$
- $L$ is odd
- Symetric impulse response $h_k = h_{L-1-k}$
- Group delay $\alpha = \frac{L-1}{2}$ is an integer
Linear phase filters

Type II linear phase filter

- \( \beta = 0 \Rightarrow \varphi(\Omega) = -\alpha \Omega \)
- \( L \) is even
- Symmetric impulse response
  \[ h_k = h_{L-1-k} \]
- Group delay \( \alpha = \frac{L-1}{2} \) is not an integer

Remark: If we have \( \Omega = \pi \), we shall have \( H(e^{j\pi}) = 0 \) which means that type II filter shall validate this constraint. As a consequence, HPF cannot be type II linear phase filters
Linear phase filters

Type III linear phase filter

- \( \beta \pm \frac{\pi}{2} \rightarrow \varphi(\Omega) = \beta - \alpha \Omega \)
- \( L \) is odd
- Antisymmetric impulse response
  \( h_k = -h_{L-1-k} \)
- Group delay \( \alpha = \frac{L-1}{2} \) is an integer

Remark: We shall have \( H(e^{j\pi}) = H(e^{j\pi}) = 0 \) which means that type II filter shall validate this constraint. Example: bandpass filters
Linear phase filters

Type III linear phase filter

- $\beta \pm \frac{\pi}{2} \Rightarrow \varphi(\Omega) = \beta - \alpha\Omega$
- $L$ is even
- Antisymmetric impulse response
  
  \[ h_k = -h_{L-1-k} \]
- Group delay $\alpha = \frac{L-1}{2}$ is not an integer

Remark: We shall have $H(e^{j\pi}) \neq 0$ and $H(e^{j0}) = 0$ which means that type II filter shall validate this constraint. Example: HPF
Conclusion on linear phase filters

Importance of linear phase filters

- Each frequency is delayed with the same amount
- Ensure no distortion in frequency domain
  - Particularly suitable for audio applications
  - Also used in digital communications
- Linear phase in frequency domain is linked to time domain symmetry
  - $\varphi(\Omega) = \beta - \alpha\Omega$
- Depending on phase characteristics and filter depth $L$, 4 classes

Linear phase types

1. $\beta = 0$ and $L$ is odd: Type I
2. $\beta = 0$ and $L$ is even: Type II
3. $\beta = \pm \frac{\pi}{2}$ and $L$ is odd: Type III
4. $\beta = \pm \frac{\pi}{2}$ and $L$ is even: Type IV
Conclusion on linear phase filters

a) Type I filter  b) Type II filter

c) Type III filter  d) Type IV filter
Window based synthesis

Window basic idea
Any filter defined with a finite length corresponds to an infinite filter that is observed

- Example of simplest observation window: the gate
- Any window have consequence both in time and frequency domains

Mathematical model
We consider a ideal filter $H(e^{j\Omega})$, with $2\pi$ period. We can express is Fourier transform as

$$ H(e^{j\Omega}) = \sum_{k=-\infty}^{\infty} h_k e^{-jk\Omega} \quad (73) $$
Window based synthesis

Window basic idea
Any filter defined with a finite length corresponds to an infinite filter that is observed

- Example of simplest observation window: the gate
- Any window have consequence both in time and frequency domains

Mathematical model
We consider a ideal filter $H(e^{j\Omega})$, with $2\pi$ period. We can express it Fourier transform as

$$H(e^{j\Omega}) = \sum_{k=-\infty}^{\infty} h_k e^{-jk\Omega}$$

Which leads to the expression of each tap as

$$h_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} H(e^{j\Omega}) e^{jk\Omega} d\Omega$$
The previous definition is valid for each $h_k$ from $-\infty$ to $+\infty$

- The desired FIR can be deduced by truncation

$$\hat{h}_k = \begin{cases} h_k, & 0 \leq k < L \\ 0, & k < 0 \text{ and } k \geq L \end{cases}$$

This corresponds to a windowing that affects both the time and frequency domain

$$\hat{h}_k = h_k w_k (75)$$

The rectangular shape $r(n)$ is defined as

$$r_k = \begin{cases} 1, & 0 \leq k < L \\ 0, & k < 0 \text{ and } k \geq L \end{cases}$$

For rectangular shape in time domain, we have a Gibbs effect [20]: Ripple in frequency domain due to hard truncation in time domain

$$\hat{H}(e^{j\Omega}) = H(e^{j\Omega}) \ast W(e^{j\Omega})$$

Window based synthesis

The previous definition is valid for each $h_k$ from $-\infty$ to $+\infty$

- The desired FIR can be deduced by truncation

$$\hat{h}_k = \begin{cases} h_k, & 0 \leq k < L \\ 0, & k < 0 \text{ and } k \geq L \end{cases}$$

- This corresponds to a windowing that affects both the time and frequency domain

$$\hat{h}_k = h_k \cdot w_k$$ (75)

The rectangular shape $r(n)$ is defined as

$$r_k = \begin{cases} 1, & 0 \leq k < L \\ 0, & k < 0 \text{ and } k \geq L - 1 \end{cases}$$

Window based synthesis

The previous definition is valid for each $h_k$ from $-\infty$ to $+\infty$

- The desired FIR can be deduced by truncation

$$\hat{h}_k = \begin{cases} h_k, & 0 \leq k < L \\ 0, & k < 0 \text{ and } k \geq L \end{cases}$$

- This corresponds to a windowing that affects both the time and frequency domain

$$\hat{h}_k = h_k \cdot w_k \quad (75)$$

The rectangular shape $r(n)$ is defined as

$$r_k = \begin{cases} 1, & 0 \leq k < L \\ 0, & k < 0 \text{ and } k \geq L - 1 \end{cases}$$

- For rectangular shape in time domain, we have a Gibbs effect [20]: Ripple in frequency domain due to hard truncation in time domain

$$\hat{H}(e^{j\Omega}) = H(e^{j\Omega}) \ast W(e^{j\Omega})$$

Window based synthesis

\[ \hat{H}(\Omega) = H(\Omega) \star W(\Omega) \]

**Figure:** Impact of apodisation on filter frequency template
Apodisation functions (windows)

We can use different window shapes

- Having a smooth transition in time domain to limits the ripple in frequency domain
- Often used in power spectral density estimation

Main apodisation functions

1. Rectangular shape
2. Triangular shape (Bartlett window)
3. Hanning window
4. Hamming window
5. Blackman window
Apodisation functions (windows)

Rectangular shape

\[ w_k = \begin{cases} 
1, & 0 \leq k < L \\
0, & k < 0 \text{ and } k \geq L
\end{cases} \]  \hspace{1cm} (76)

Figure: Rectangular window (time and frequency responses)
Apodisation functions (windows)

Triangular (Bartlett) shape

\[ w_k = \begin{cases} 
\frac{2k}{L-1}, & 0 \leq k \leq \frac{L-1}{2} \\
2 - \frac{2k}{L-1}, & \frac{L-1}{2} \leq k \leq L - 1 \\
0, & k < 0 \text{ and } k \geq L 
\end{cases} \tag{77} \]

Figure: Bartlett window (time and frequency responses)
Apodisation functions (windows)

Hanning window

\[ w_k = \begin{cases} 
\frac{1}{2} - \frac{1}{2} \cos \left( \frac{2\pi k}{L-1} \right), & 0 \leq k \leq L - 1 \\
0, & k < 0 \text{ and } k \geq L 
\end{cases} \]  

Figure: Hanning window (time and frequency responses)
Apodisation functions (windows)

Hamming window

\[ w_k = \begin{cases} 
0.54 - 0.46 \cos \left( \frac{2\pi k}{L-1} \right), & 0 \leq k \leq L - 1 \\
0, & k < 0 \text{ and } k \geq L 
\end{cases} \]  

(79)

Figure: Hamming window (time and frequency responses)
Apodisation functions (windows)

Blackman window

\[ w_k = \begin{cases} 
0.42 - 0.5 \cos \left( \frac{2\pi k}{L-1} \right) + 0.08 \cos \left( \frac{4\pi k}{L-1} \right), & 0 \leq k \leq L - 1 \\
0, & k < 0 \text{ et } k \geq L 
\end{cases} \]

(80)

Figure: Blackman window (time and frequency responses)
Apodisation functions (windows)

**Hanning window**
- Introduced by Julius von Han
- Based on harvesine function $\sin^2()$ [21]
- Exponentially attenuated sinc in frequency domain

Apodisation functions (windows)

Hanning window
- Introduced by Julius von Han
- Based on harvesine function $\sin^2()$ [21]
- Exponentially attenuated sinc in frequency domain

Hamming window
- Direct derivation from Hanning function with different numeric parameters
- Numeric choices (0.42 and 0.08) places a zero-crossing at frequency $\frac{5\pi}{(L - 1)}$ which cancels the first sidelobe of the Hann window [22]

Apodisation functions (windows)

Hanning window
- Introduced by Julius von Han
- Based on harvesine function $\sin^2()$ [21]
- Exponentially attenuated sinc in frequency domain

Hamming window
- Direct derivation from Hanning function with different numeric parameters
- Numeric choices (0.42 and 0.08) places a zero-crossing at frequency $5\pi/(L - 1)$ which cancels the first sidelobe of the Hann window [22]

Blackman window
- Also derivative of Hanning window [23]
- 0.42, 0.5 and 0.08 place zeros at the third and fourth sidelobes (discontinuity at the edges and a 6 dB/oct fall-off).

Apodisation functions (windows)

Comparison between windows

Figure: Comparison between apodisation windows (time domain)
Apodisation functions (windows)

Comparison between windows

Figure: Comparison between apodisation windows (frequency domain)
### Apodisation functions (windows)

<table>
<thead>
<tr>
<th>Window</th>
<th>Amplitude ratio between main lobe and secondary lobe</th>
<th>Mainlobe width $\Delta\Omega_m$</th>
<th>Minimal attenuation in attenuated bandwidth $\Delta A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rectangular</td>
<td>$-13$ dB</td>
<td>$4\pi / L$</td>
<td>$-21$ dB</td>
</tr>
<tr>
<td>Bartlett</td>
<td>$-25$ dB</td>
<td>$8\pi / L$</td>
<td>$-25$ dB</td>
</tr>
<tr>
<td>Hanning</td>
<td>$-31$ dB</td>
<td>$8\pi / L$</td>
<td>$-44$ dB</td>
</tr>
<tr>
<td>Hamming</td>
<td>$-41$ dB</td>
<td>$8\pi / L$</td>
<td>$-53$ dB</td>
</tr>
<tr>
<td>Blackman</td>
<td>$-57$ dB</td>
<td>$12\pi / L$</td>
<td>$-74$ dB</td>
</tr>
</tbody>
</table>

Table: Window characterization
Other windows

There are many (many many many) windows that can be designed and used

- Some are parameter based window that can lead to proper design

  Adaptive windows that can be tuned for a proper uses in time and/or frequency domain

Adjustable windows

1. Gaussian window
2. Dolph Chebyshev window

For more information (and many examples) see [21]

Other windows

Gaussian window

- Eigenvector: Corresponds to the eigenvector of the Fourier transform (FT of Gaussian window is a Gaussian window)
- Smoothness: Smooth transition both in time and frequency domain
- Windowception: Gaussian function is infinite, so window is truncated (or itself windowed)
- Parametrized: $\sigma$ also linked to bandwidth time product (BT) $< 0.5$

$$w_k = e^{-\frac{1}{2} \left( \frac{k-(L-1)/2}{\sigma(L-1)/2} \right)^2}$$  

(81)
Other windows

Gaussian window

\[ w_k = e^{-\frac{1}{2} \left( \frac{k-(L-1)/2}{\sigma(L-1)/2} \right)^2} \]

Figure: Gaussian window (time and frequency responses)
Other windows

Plenty of different windows with different uses

- Maximization of energy in main lobe
- Side lobe reduction
- Trade-off
- ...

Other windows

1. Generalized normal window
2. Tukey window
3. Discrete prolate spheroidal sequence (or Slepian window)
4. Kaiser window
5. Dolph–Chebyshev window
Window selection

We focus on the non-parametric windows

1. Rectangular shape
2. Triangular shape (Bartlett window)
3. Hanning window
4. Hamming window
5. Blackman window

Strong influence of window in both time and frequency domains

How to we choose the correct window?

We must define appropriate metrics
Window selection

Transition bandwidth

- Defined as $\Delta \Omega = |\Omega_p - \Omega_a|$.
- Linked to main lobe width $\Delta \Omega_m$.
- Approximation: $\Delta \Omega \approx \frac{\Delta \Omega_m}{2}$.
- The larger $L$; the lower $\Delta \Omega$.

Ripples

- Equal in bandpass and attenuated bandwidth: **Equiripple** filters.
- $\Delta A = \delta_1 = \delta_2$.
- Window dependant but independant from $L$. 
Window selection

Figure: Windowing impact and important metrics
Window selection

FIR synthesis with windowing

- Recall: we have an infinite impulse filter from direct synthesis
- To transform it into FIR, apodisation (windowing) is necessary

\[ \hat{h}_k = h_n - \alpha w_k \]

With \( \alpha \) the added delay to ensure linear phase filter
Window selection

FIR synthesis with windowing

- Recall: we have an infinite impulse filter from direct synthesis
- To transform it into FIR, apodisation (windowing) is necessary

Window selection

How do we select the window

1. Choose the window nature with $\Delta_A$ selection
2. From $\Delta_\Omega$ and from the window choice, deduce the targeted length $L$
Window selection

FIR synthesis with windowing

- Recall: we have an infinite impulse filter from direct synthesis
- To transform it into FIR, apodisation (windowing) is necessary

Window selection

How do we select the window

1. Choose the window nature with $\Delta_A$ selection
2. From $\Delta_\Omega$ and from the window choice, deduce the targeted length $L$

The FIR is finally obtained by apodisation i.e

$$\hat{h}_k = h_{n-\alpha}w_k$$

- With $\alpha$ the added delay to ensure linear phase filter
Window selection

Exercise

Window selection
We consider the following requirements

1. Transition bandwidth $\Delta \Omega = 0.1\pi$,
2. Targeted attenuation greater than 30dB
Window selection

Exercise

Window selection
We consider the following requirements

1. Transition bandwidth $\Delta \Omega = 0.1\pi$, 
2. Targeted attenuation greater than 30dB

- From attenuation, we choose Hanning, Hamming or Blackman
- As transition is short, we prefer Hamming (shorter window with better $\lambda$)
- We have a window linked to $\Delta \Omega_m = \frac{8\pi}{L} = 2\Delta \Omega$; $L = 40$. 

Digital Signal processing
FIR synthesis
Window based synthesis
The filter specifications are often in **frequency domain**

- Basic idea: use the specification in frequency domain to obtain the filter in time domain
- We need a tool to convert a sampled signal in frequency domain to a sampled signal in time domain

---

Fourier transform
Frequency sampling method

The filter specifications are often in **frequency domain**

- Basic idea: use the specification in frequency domain to obtain the filter in time domain
- We need a tool to convert a sampled signal in frequency domain to a sampled signal in time domain

  Fourier transform

**Frequency sampling method**

The desired filter is obtained as the inverse Fourier transform of the filter specifications defined in frequency domain

- We start from the ideal continuous frequency response $H(e^{j\omega})$
- We choose to sample this response with $L$ points
Frequency sampling method

The desired filter is obtained as the inverse Fourier transform of the filter specifications defined in frequency domain

- We start from the ideal continuous frequency response \( H(e^{j\omega}) \)
- We choose to sample this response with \( L \) points
- The sampling frequency grid (or sampling space in frequency domain) is \( \Omega_e = \frac{2\pi}{L} \) and the digital filter (in frequency domain) is

\[
\hat{H}(e^{jk\Omega_e}) \triangleq H(e^{j\Omega})|_{\Omega=\frac{2k\pi}{L}}
\]  

\( (82) \)
Frequency sampling method

The desired filter is obtained as the inverse Fourier transform of the filter specifications defined in frequency domain

- We start from the ideal continuous frequency response \( H(e^{j\omega}) \)
- We choose to sample this response with \( L \) points
- The sampling frequency grid (or sampling space in frequency domain) is \( \Omega_e = \frac{2\pi}{L} \) and the digital filter (in frequency domain) is

\[
\hat{H}(e^{j k\Omega_e}) \triangleq H(e^{j\Omega})|_{\Omega=\frac{2k\pi}{L}}
\]  

(82)

- We obtain the FIR impulse response by means of inverse Fourier transform

\[
\hat{h}_k = \frac{1}{L} \sum_{n=0}^{L-1} \hat{H}(e^{jk\Omega_e})e^{2j\pi \frac{k}{L}n}, \quad k = 0, 1 \ldots L - 1
\]

(83)

\[
\hat{h}_k = 0, \text{ otherwise}
\]

(84)
Frequency sampling method

Filter in time domain

\[ \hat{h}_k = \frac{1}{L} \sum_{n=0}^{L-1} \hat{H}(e^{jk\Omega_e}) e^{2j\pi \frac{k}{L} n}, \quad k = 0, 1 \ldots L - 1 \]

\[ \hat{h}_k = 0 \text{ otherwise} \]

Why 2 equations?

- Sampling in frequency \[\rightarrow\] Periodisation in time domain
- Equivalent to apply a rectangular window to the filter

\[ \hat{h}_k = \frac{1}{L} \sum_{n=0}^{L-1} \hat{H}(e^{jk\Omega_e}) e^{2j\pi \frac{k}{L} n} \times w_k \]

\[ w_k = \begin{cases} 1, & 0 \leq k < L \\ 0, & k < 0 \text{ and } k \geq L \end{cases} \]

- Any window can be used!
Frequency sampling method

Example of derivative filter

- We took $L$ points with uniform frequency sampling between 0 and $2\pi$
- Last point is not taken!
- Important: Ensure a linear phase

Figure: Example of frequency sampling of derivative filter
Let’s assume that we want to design a LPF filter with a cut-off frequency of $1/4 \times 2\pi$.

- One way to do this is to force the frequency to 0 when frequency is greater than the desired cut-off frequency.
- The higher the filter size, the lower the space between two consecutive frequency points: the filter becomes more frequency selective.
- Also, impact of the window used (no window is a rectangular window).
Frequency sampling method

Figure: Influence of frequency spacing in Frequency sampling method
**Frequency sampling method**

![Graph showing Frequency sampling method](image)

**Figure:** Influence of frequency spacing in Frequency sampling method (zoom on bandpass)

- **Frequency:** 0 to 1.4
- **Magnitude:** -6 to 0
- **Windows:** L = 16 with rect window, L = 32 with rect window, L = 64 with rect window, L = 128 with rect window
- **Filter spec**
Frequency sampling method

Figure: Influence of window apodisation in Frequency sampling method
Frequency sampling method

![Graph showing Frequencysampling method](image)

**Figure**: Influence of window apodisation in Frequency sampling method (zoom on bandpass)
Conclusion in frequency sampling

Simple method for FIR synthesis

- Based on what we want to have in frequency domain
- FIR impulse is obtained with inverse Fourier transform
- 2 freedom degrees in synthesis
  - Filter length in samples (linked to frequency spacing)
  - Choice of apodisation window

But can be tricky if used without caution
Forcing the grid leads to too many constraints for proper design
Ripples in bandpass
Gain in bandpass to monitor
Conclusion in frequency sampling

Simple method for FIR synthesis

- Based on what we want to have in frequency domain
- FIR impulse is obtained with inverse Fourier transform
- 2 freedom degrees in synthesis
  - Filter length in samples (linked to frequency spacing)
  - Choice of apodisation window

But can be tricky if used without caution

- Forcing the grid leads to too many constraints for proper design
- Ripples in bandpass
- Gain in bandpass to monitor
Conclusion on FIR synthesis

Dedicated methods for FIR synthesis
- Cannot be based on analog filter identification

FIR properties
FIR implementation are simpler
- Stable by construction
- Only dependant on signal input
- Linear phase FIR filters are often used
Conclusion on FIR synthesis

Dedicated methods for FIR synthesis

- Cannot be based on analog filter identification

FIR properties

FIR implementation are simpler

- Stable by construction
- Only dependant on signal input
- Linear phase FIR filters are often used

Synthesis methods

Different synthesis methods can be used

1. Window based FIR
2. Frequency sampling FIR
3. Adaptive FIR
Window based filters

- FIR can be seen as IIR with observation window (truncation)
- Different window can be used with different properties
- Trade-off between main lobe width and other lobes level
- Main windows
  1. Rectangular window
  2. Bartlett window
  3. Hamming, Hanning, Blackman windows
  4. Parametrable windows (Gaussian, ... )
Conclusion on FIR synthesis

Frequency sampling method
Based on sampling of the continuous analog frequency response

- FIR is obtained with inverse Fourier transform
- Jointly used with apodisation (windowing)
- Arbitrary frequency response can be obtained

Adaptive filters
Sometimes FIR cannot be pre-computed

- Set up a cost function to minimize
- The filter taps are iteratively computed with a minimization method (LMS, RLS, APA, ...)

Application example: noise cancellation
Fast Fourier Transform
Many processing in Digital signal processing requires to shift from **time domain** to **frequency domain**. Done with Fourier transform.

But how Fourier transform can be (efficiently) done?
Fourier transform formulation

Let’s go back to the initial Fourier transform

- The signal $x(n)$ is sampled at its sampling frequency $F_c$
- If we consider a FT of size $N$ we have

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-2\pi j n \frac{k}{N}}, \quad k = 0, 1 \ldots N - 1$$  \hspace{1cm} (85)
Fourier transform formulation

Let’s go back to the initial Fourier transform

- The signal $x(n)$ is sampled at its sampling frequency $F_c$
- If we consider a FT of size $N$ we have

$$X(k) = \sum_{n=0}^{N-1} x(n).e^{(-2j\pi \frac{n.k}{N})}, \quad k = 0, 1 \ldots N - 1$$  \hspace{1cm} (85)

- And its inverse Fourier transform is defined as

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k).e^{(2j\pi \frac{n.k}{N})}, \quad n = 0, 1 \ldots N - 1$$  \hspace{1cm} (86)

Note that in practise both time domain and frequency domain signal are complex.

Computational complexity?
Fourier transform formulation

Equation 85 can be formulated with the use of matrixes

\[ X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-2\pi \frac{n \cdot k}{N}}, \quad k = 0, 1 \ldots N - 1 \]  

Discrete Fourier Transform (DFT) with matrix

\[
\begin{pmatrix}
X(0) \\
X(1) \\
\vdots \\
X(N - 1)
\end{pmatrix}
= \begin{pmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & W_N^1 & W_N^2 & \cdots & W_N^{N-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)^2}
\end{pmatrix}
\times
\begin{pmatrix}
x(0) \\
x(1) \\
\vdots \\
x(N - 1)
\end{pmatrix}
\]

with \( W_N \) is called **twiddle factor** and is equal to \( e^{-2\pi \frac{j}{N}} \)
We introduce the Discrete Fourier Transform (DFT) matrix as

\[
F_N = \frac{1}{\sqrt{N}} \begin{pmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & W_N^1 & W_N^2 & \cdots & W_N^{N-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\
1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)^2}
\end{pmatrix}
\]

with \( W_N \) is called \textbf{twiddle factor} and is equal to \( e^{-2j\frac{\pi}{N}} \)
We introduce the Discrete Fourier Transform (DFT) matrix as

\[
F_N = \frac{1}{\sqrt{N}} \begin{pmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & W_N^1 & W_N^2 & \cdots & W_N^{N-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\
1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)^2}
\end{pmatrix}
\]

with \( W_N \) is called twiddle factor and is equal to \( e^{-2j\frac{\pi}{N}} \)

**DFT matrix properties**

- The twiddle factor is a primitive Nth root of unity
- The matrix is a **Vandermonde** matrix
  - Matrix with the terms of a geometric progression
- Normalisation factor definition can sometimes be different
  - Normalisation product (between \( F \) and \( F^{-1} \)) should be equal to 1
  - With \( \frac{1}{\sqrt{N}} \) the matrix becomes unitary (i.e. it preserves energy)
Fourier transform formulation

\[
\begin{pmatrix}
X(0) \\
X(1) \\
\vdots \\
X(N-1)
\end{pmatrix} =
\begin{pmatrix}
1 & 1 & \cdots & 1 \\
1 & W_1^1 & \cdots & W_1^{N-1} \\
\vdots & \vdots & \ddots & \vdots \\
1 & W_{N-1}^1 & \cdots & W_{N-1}^{N-1}
\end{pmatrix}
\times
\begin{pmatrix}
x(0) \\
x(1) \\
\vdots \\
x(N-1)
\end{pmatrix}
\]

Complexity (complex)

- \( N^2 \) complex multiplications
- \( N(N - 1) \) complex additions

Complexity (real)

- \( 4N^2 \) real multiplications
- \( N(4N - 2) \) real additions

Complexity in \( O(N^2) \)
Principle of the FFT

Fourier transform is too complex

- $4N^2$ real multiplications
- $N(4N - 2)$ real additions
- And lot of memory ($N$ samples lot of basis elements)

But possibility to use Fourier transform properties

\[
W_N^{k(N-n)} = (W_N^{kn})^* \\
W_N^{k(n+N)} = W_N^{n(k+N)} = W_N^{kn} \\
W_N^{n+N/2} = -W_N^n \\
W_N^{2kn} = W_N^{kn}
\]

Complexity is reduced by still in $O(N^2)$ [24]

Principle of the FFT

Complexity can be reduced as Fourier transform can be clustered (or regrouped)

Principle of the Fast Fourier Transform (FFT)

FFT algorithm principle

- Separation can be done in time domain
  - 2 sequences depending on index parity
  - $x(n)$ is divided into 2 sub-sequences
- Separation can be done in frequency domain
  - 2 sequences depending on index parity
  - $X(k)$ is divided into 2 sub-sequences

Initial FFT proposition in 1965 in [25]
Lot of research still today to these methods
Similar methods for other decomposition (Hadamard . . .)

Time split FFT

Let’s express the DFT in time domain

- Separate the odd from the even indexes

\[ X(k) = \sum_{n \text{ even}} x(n)W_{N}^{nk} + \sum_{n \text{ odd}} x(n)W_{N}^{nk} \]
Let’s express the DFT in time domain

- Separate the odd from the even indexes

\[ X(k) = \sum_{n \text{ even}} x(n) \cdot W_N^{nk} + \sum_{n \text{ odd}} x(n) \cdot W_N^{nk} \]

\[ X(k) = \sum_{n=0}^{N/2-1} x(2n) \cdot W_N^{2nk} + \sum_{n=0}^{N/2-1} x(2n+1) \cdot W_N^{(2n+1)k} \]
Time split FFT

Let’s express the DFT in time domain

- Separate the odd from the even indexes

\[ X(k) = \sum_{n \text{ even}} x(n) \cdot W_N^{nk} + \sum_{n \text{ odd}} x(n) \cdot W_N^{nk} \]

\[ X(k) = \sum_{n=0}^{N/2-1} x(2n) \cdot W_N^{2nk} + \sum_{n=0}^{N/2-1} x(2n + 1) \cdot W_N^{(2n+1)k} \]

\[ X(k) = \sum_{n=0}^{N/2-1} x(2n) \cdot W_{N/2}^{nk} + W_N^k \sum_{n=0}^{N/2-1} x(2n + 1) \cdot W_{N/2}^{nk} \]
Time split FFT

Let's express the DFT in time domain

- Separate the odd from the even indexes

\[
X(k) = \sum_{n \text{ even}} x(n)W_N^{nk} + \sum_{n \text{ odd}} x(n)W_N^{nk}
\]

\[
X(k) = \sum_{n=0}^{N/2-1} x(2n)W_N^{2nk} + \sum_{n=0}^{N/2-1} x(2n+1)W_N^{(2n+1)k}
\]

\[
X(k) = \sum_{n=0}^{N/2-1} x(2n)W_{N/2}^{nk} + W_N^k \sum_{n=0}^{N/2-1} x(2n+1)W_N^{nk}
\]

\[
X(k) = G(k) + W_N^kH(k)
\]

With

\[
\begin{cases} 
G(k) \text{ is the DFT on } N/2 \text{ even points} \\
H(k) \text{ is the DFT on } N/2 \text{ odd points}
\end{cases}
\]
Time split FFT

DFT of size $N$ is now divided into 2 sub-DFT of size $N/2$

- Not a complexity reduction though!

We can now use the DFT properties (linked to twiddle factor)

$$W_{N/2}^{k+N/2} = -W_N^k$$

And the DFT can be expressed as

$$X(k + \frac{N}{2}) = \sum_{n=0}^{N/2-1} x(2n).W_{N/2}^{n(k+N/2)} + W_N^{k+N/2} \sum_{n=0}^{N/2-1} x(2n + 1).W_{N/2}^{n(k+N/2)}$$

$$X(k + \frac{N}{2}) = G(k) - W_N^k. H(k)$$

This is called a butterfly
To FFT

If \( N \) is a power of 2, \( N/2 \) can also be divided by 2.

- In a Divide and conquer approach
- Can be done iteratively until a DFT on two points

**Input - output link**

\[
X(k) = G(k) + W_N^k \cdot H(k)
\]

\[
X(k + \frac{N}{2}) = G(k) - W_N^k \cdot H(k)
\]

**2 points DFT**

Minimal graph flow

- Minimal complexity, as complexity is on \( \mathcal{O}(n) \)
- Then, recombination with \( \log_2(n) \) recombinations

\[
\text{FFT complexity is } \mathcal{O}(n) \times \log_2(n)
\]
Time split FFT

Separation of the DFT into two main streams
- One dedicated to odd points
- One dedicated to even points

With twiddle factor, recombination can be done

\[
\begin{align*}
  x(0) & \quad x(1) \\
  x(4) & \quad x(2) \\
  x(6) & \quad x(4) \\
  x(2) & \quad x(3) \\
  x(3) & \quad x(5) \\
  x(7) & \quad x(7) \\
\end{align*}
\]

Figure: Index division principle
Minimal FFT butterfly

A FFT is a flow based on a minimal butterfly

- This minimal flow is present at each $\log_2(N)$ stage of the FFT
- Let’s have a glance at stage $m$

\[ X_m(p) \xrightarrow{\mathbf{1}} X_{m+1}(p) \]
\[ X_m(q) \xrightarrow{W_N^r} X_{m+1}(q) \]

**Figure:** Butterfly flow

Link between input and output (depends on index, generic pattern here) linked to the equations

\[
\begin{align*}
X_{m+1}(p) &= X_m(p) + W_N^r X_m(q) \\
X_{m+1}(q) &= X_m(p) - W_N^r X_m(q)
\end{align*}
\] (89)
Minimal FFT butterfly

For each unit
- One complex multiplication (based on twiddle factor)
- One addition
- One subtraction

Now, this unit has to be used in each \( \log_2(N) \) stages of the FFT
- Divide and conquer: go from last stage to first stage
- Indexes can be rearranged (index are in bit reverse mode)

Figure: Butterfly flow
Example with 16 points FFT

**Figure:** Example of last stage for a 16 points FFT with butterfly approach
Example with 16 points FFT

Figure: Example of a 16 points FFT with butterfly approach
Example with 16 points FFT

Same without index shifting
Frequency split FFT

This method split the FFT in time domain

- Possibility to split the FFT in frequency domain
- *i.e* successive instead of interleaved separation

\[
X(k) = \sum_{n=0}^{N/2-1} x(n) \cdot W_N^{nk} + \sum_{n=N/2}^{N-1} x(n) \cdot W_N^{nk}
\]
Frequency split FFT

This method split the FFT in time domain

- Possibility to split the FFT in frequency domain
- i.e successive instead of interleaved separation

\[
X(k) = \sum_{n=0}^{N/2-1} x(n) \cdot W_N^{nk} + \sum_{n=N/2}^{N-1} x(n) \cdot W_N^{nk}
\]

\[
X(k) = \sum_{n=0}^{N/2-1} x(n) \cdot W_N^{nk} + W_N^{kN/2} \sum_{n=0}^{N/2-1} x(n + N/2) \cdot W_N^{nk}
\]
Frequency split FFT

This method split the FFT in time domain

- Possibility to split the FFT in frequency domain
- i.e successive instead of interleaved separation

\[
X(k) = \sum_{n=0}^{N/2-1} x(n) \cdot W_N^{nk} + \sum_{n=N/2}^{N-1} x(n) \cdot W_N^{nk}
\]

\[
X(k) = \sum_{n=0}^{N/2-1} x(n) \cdot W_N^{nk} + W_N^{kN/2} \sum_{n=0}^{N/2-1} x(n + N/2) \cdot W_N^{nk}
\]

\[
X(k) = \sum_{n=0}^{N/2-1} [x(n) + (-1)^k \cdot x(n + N/2)] W_N^{nk}
\]
Frequency split FFT

We can now separate odd and even indexes

\[
X(2p) = \sum_{n=0}^{N/2-1} [x(n) + x(n + N/2)] W_N^{2pk}
\]

\[
X(2p + 1) = \sum_{n=0}^{N/2-1} [x(n) - x(n + N/2)] W_N^{2pk} \cdot W_N^n
\]
Frequency split FFT

We can now separate odd and even indexes

\[ X(2p) = \sum_{n=0}^{N/2-1} [x(n) + x(n + N/2)] W_N^{2^p k} \]

\[ X(2p + 1) = \sum_{n=0}^{N/2-1} [x(n) - x(n + N/2)] W_N^{2^p k}.W_N^n \]

And same divide and conquer method

\[ X(2p) = \sum_{n=0}^{N/2-1} [x(n) + x(n + N/2)] W_{N/2}^{p k} \]

\[ X(2p + 1) = \sum_{n=0}^{N/2-1} \left( [x(n) - x(n + N/2)] W_{N/2}^{p k} \right).W_N^n \]
Frequency split FFT

Figure: FFT with decimation in frequency
Conclusion on FFT

Discrete Fourier Transform is a basis operation in signal processing

- Frequency processing widely used in many fields
- Base of modern transmission systems (OFDM)

Classic DFT requires lot of computational load

- $4N^2$ real multiplications
- $N(4N - 2)$ real additions
- And lot of memory ($N$ samples lot of basis elements)

Fourier transform has a lot of embedded symmetry that can be exploited

- Based on roots of unity (twiddle factor)
- Complexity is highly reduced when $N$ is a power of 2
Conclusion FFT

Time split FFT

Divide and conquer method

- Each stage is divided into 2 subflows with \( n/2 \) DFT stages
- Based on DFT properties

\[
X(k) = G(k) + W_N^k.H(k)
\]
\[
X(k + \frac{N}{2}) = G(k) - W_N^k.H(k)
\]

- Minimal system with a butterfly with 2 coefficients
- Index can be rearranged (index in bit reverse mode) to better show the butterfly
- \( N/2 \log_2(N) \) complex multiplications and \( N \log_2(N) \) additions

Remark: this approach is called a **Radix-2** butterfly

- Possibility to use larger number of radix (4, 16, \ldots)
- In practice, radix-2 is not best solution (performance/calculation, memory \ldots)
Spectral analysis
Introduction and principle

FFT is a baseline in digital signal processing techniques

- Analysis (and processing) can be done in frequency domain

Time to frequency transformation with Discrete Fourier Transform

- and its efficient FFT architecture

Still questions remains to perform a frequency analysis

Frequency analysis

- How to select a signal ($N$ points)
- What are the influence of the observation?
- How to monitor the frequency resolution from an acquisition

Frequency analysis is a wide and complex area: see lecture from C. Carriou
Deterministic frequency analysis

We now how to perform a deterministic frequency analysis

A kind remainder on how it is done on a deterministic analog signal

As we already saw, observation is equivalent to apodisation

- This is a truncation
- No active window: rectangular window

This window affects the spectral analysis: how?
Impact of truncation

The signal is observed on a finite length
- The Fourier analysis is done on $N$ points with $N < \infty$
- Conversely, $N$ can be a power a 2

Characterization
- $F_c$ the sampling frequency
- $x(t)$ the analog signal
- $x[n] = x(nT_e)$ the digital signal of infinite length
  - We assume perfect quantization here
- $w_N[n]$ the window (apodisation function)
  - Causal function, $N$ points
- $x_N[n]$ the observed signal
  - Causal, $N$ points, sampling time $T_c$
Impact of truncation

The observed signal $x_T[n]$ can be expressed as

$$x_N[n] = x[n] \times w_N[n]$$

Legacy expression of windowing (see FIR synthesis part)

Rectangular apodisation

Example with “no window” (i.e rectangular)

$$w_N[n] = \begin{cases} 1, & 0 \leq n < N \\ 0, & n < 0 \text{ and } n \geq N \end{cases}$$

The signal is

$$x_T[n] = x[n] \cdot w_N[n] = \sum_{k=0}^{N-1} x[k] \cdot \delta(n - k) \quad (90)$$
Impact of truncation

Consequence in frequency domain

Multiplication in time domain \[\rightarrow\] Convolution in frequency domain

If we apply the Discrete Fourier Transform, we obtain

\[
X_N(e^{j\Omega}) = X(e^{j\Omega}) \ast W(e^{j\Omega})
\]

\[
= \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\Theta}) \cdot W(e^{j(\Omega-\Theta)}) \, d\Theta
\]

With

- \(X(e^{j\Omega})\) Fourier transform of signal w/o apodisation
- \(W(e^{j\Omega})\) Fourier transform of window
Impact of truncation

Rectangular window

If we calculate the Fourier transform of the rectangular window

- Sampled at $T_c$
- On $N$ points

$$W(e^{j\Omega}) = W_r(e^{j\Omega}) = e^{-j\Omega N - \frac{1}{2}} \sin\left(\frac{N\Omega}{2}\right) \sin\left(\frac{\Omega}{2}\right)$$
Impact of truncation

Rectangular window

If we calculate the Fourier transform of the rectangular window

- Sampled at $T_c$
- On $N$ points

We have

$$\mathcal{W}(e^{j\Omega}) = \mathcal{W}_r(e^{j\Omega}) = e^{-j\Omega \frac{N-1}{2}} \frac{\sin(N\Omega/2)}{\sin(\Omega/2)}$$  \hspace{1cm} (92)

- The main lobe can melt two distinct frequencies
- Frequency ripple due to secondary lobes (noise and masking)
Impact of truncation

Rectangular shape

\[ w_N[n] = \begin{cases} 
1, & 0 \leq n < N \\
0, & k < 0 \text{ and } n \geq N 
\end{cases} \]  

(93)

Figure: Rectangular window (time and frequency responses)
Discrete Fourier transform and frequency mapping

Fourier transform is discrete

**Sampling time**

- Signal in time domain as $N$ points
- Each index is separated by $T_c$ with $T_c$ the sampling time

**Sampling frequency**

- Signal is converted in frequency domain with $N$ points
- We have thus a **frequency grid** with discrete elements
- How can be link this indexes to frequencies?
We want to transmit a pure tone A (440 Hz) during 10 seconds

Signal in time domain?

Signal in frequency domain?

Physical interpretation of x axis
Example of A tone

Figure: Signal in time domain (in samples)
Example of A tone

Figure: Signal in time domain (in s)

- Transition from samples to second: Multiplication by $T_c$
Example of A tone

**Figure:** Signal in frequency domain (in samples)

- Window is large (10s): huge amount of data in frequency domain
Example of A tone

Figure: Signal in frequency domain (in samples)

- Peak depends on FFT size!
Example of A tone

Signal frequency is between $-1/2$ and $1/2$

**Figure**: Signal in frequency domain (normalized frequency)
Example of A tone

Signal frequency is between $-F_c/2$ and $F_c/2$

- We start from frequency bins

$$k \in [0 : N - 1]$$

- We convert this into normalized frequency, or pulsations

$$\omega_k = \frac{k}{N} \text{ or } \omega_k = \frac{2k\pi}{N}$$

- To physical frequencies

$$f_k = \omega_k \times F_c$$

**Figure:** Signal in frequency domain (normalized frequencies)
Example of A tone

**Figure:** Signal in frequency domain (in Hz)
We perform a **Discrete Fourier Transform**

- Discrete points in frequency domain
  - These bins correspond to a frequency resolution
  - Resolution increases with number of points

- Frequency is arbitrary
  - As bins depends on FFT size
  - Possibility to express in normalized frequency (or pulsation)
  - Conversion to Hz at the end of the methodology

- Hz conversion is the last stage:
  - Frequency can be in any time-linked arbitrary unit
  - Example: one acquisition per hour, per day, per minutes, ...
  - Frequency is expressed as “cycles”
Example: analysis of RTE signal

Electric consumption (typical use case) with several year acquisition

- One acquisition per hour
- 24 acquisitions per day
- 3 year acquisitions

Number of samples: 26280 samples

- Some signal processing can be done here to find pattern
- Data from research project on electrical consumption prediction
Example: analysis of RTE signal

Figure: Complete dataset
Example: analysis of RTE signal

![Graph showing the evolution of consumption over time](image)

**Figure:** Week analysis
Example: analysis of RTE signal

Evolution of consommation: Day Cycle

Figure: Day analysis
Frequency analysis is done with 26280 samples.

- Cycle 1 for year harmonic
- Cycle 365 for day harmonic
- Cycle 8760 for hour harmonic

Information is in harmonic

- If a peak is at cycle 52 what does it means?
Frequency analysis

**Figure:** RTE signal in Fourier domain

- Harmonics at multiple of some specific frequencies
Frequency analysis

Figure: RTE signal in Fourier domain
Frequency analysis

Figure: RTE signal in Fourier domain

- Left side: annual elements
- Peak around 4: seasons!
Frequency analysis

Figure: RTE signal in Fourier domain
Frequency analysis

**Figure:** RTE signal in Fourier domain

- Left side: Harmonic of 52 (weeks)
Frequency analysis

Figure: RTE signal in Fourier domain

- 365 and its multiples (days)
Frequency analysis

Figure: RTE signal in Fourier domain

- We can extract the components
Frequency analysis

Figure: RTE signal in Fourier domain

We can extract the components
Zero padding

The acquisition (or observation) gives a signal with length $N$
- The frequency grid is directly inherited from $N$
- The frequency resolution is proportional to the FFT size
- Complexity is also reduced is $N$ is a power of 2

How can we have a better resolution in frequency domain?

How can we gain more flexibility in the FFT size (independent from observation window size)?
A starting example

We still consider the A tone at 440 Hz.

- We want to check that the tone is precisely at 440 Hz
- With a 10 second measurement the spectral analysis, this is OK

Figure: A tone power spectral density
A starting example

How the peak is estimated?
- As the maximum of the PSD
- With a frequency grid of $\delta_f$

$$\delta_f = \frac{1}{N} \times F_c$$

- In our case, 0.1 Hz of accuracy
- But if we had 0.12 second of signal (5040 points)? We have a grid with 8Hz width

To have at least a 1 Hz accuracy, we need 1 second of signal (i.e 42 000 points)
A starting example

**Figure:** Spectrum of signal
ZP principle

Observation window induces bias in spectrum analysis

- Complete framework in estimation-detection course

We need to have a better resolution in frequency grid

- Can be obtain with a larger FFT size
- Can be done with Zero padding

Zero padding

- Having a signal of length $L$
- Insert $N - L$ zero at the end of the signal
- Apply the FFT on $N$ points
ZP principle

Figure: Impact of ZP on rectangular window
ZP principle

**Figure:** Impact of ZP on spectral analysis
ZP principle

Figure: Impact of ZP on spectral analysis
Influence of apodisation function

Signal has a finite length
- Observation window

Influence of apodisation function

- Convolution in frequency domain between signal to be analyzed and apodisation window
- Same influence and parametrisation as described in FIR synthesis part

<table>
<thead>
<tr>
<th>Window</th>
<th>Amplitude ratio between main lobe and secondary lobe</th>
<th>Mainlobe width $\Delta \Omega_m$</th>
<th>Minimal attenuation in attenuated bandwidth $\Delta A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rectangular</td>
<td>$-13\text{dB}$</td>
<td>$4\pi/L$</td>
<td>$-21\text{dB}$</td>
</tr>
<tr>
<td>Bartlett</td>
<td>$-25\text{dB}$</td>
<td>$8\pi/L$</td>
<td>$-25\text{dB}$</td>
</tr>
<tr>
<td>Hanning</td>
<td>$-31\text{dB}$</td>
<td>$8\pi/L$</td>
<td>$-44\text{dB}$</td>
</tr>
<tr>
<td>Hamming</td>
<td>$-41\text{dB}$</td>
<td>$8\pi/L$</td>
<td>$-53\text{dB}$</td>
</tr>
<tr>
<td>Blackman</td>
<td>$-57\text{dB}$</td>
<td>$12\pi/L$</td>
<td>$-74\text{dB}$</td>
</tr>
</tbody>
</table>

Table: Window characterization
Apodisation functions (windows)

Comparison between windows

Figure: Comparison between apodisation windows (time domain)
Apodisation functions (windows)

Comparison between windows

Figure: Comparison between apodisation windows (frequency domain)
Figure: Apodisation function parameters
Resolution

Spectral analysis linked to 2 different resolution measures:

**Frequency resolution**
- Capacity to discriminate two spectral lines
  - Linked to frequency masking
- Apodisation function matters
  - Main lobe width of apodisation function
- Proportional to frequency analysis size
- $\Delta \Omega_m$

**Amplitude resolution**
- Capacity to distinguish low power lines (when closed to a large power one)
- Depends on spectral analysis size
- Depends on secondary lobes of apodisation function
- $\lambda$
Different steps for spectral analysis

- Signal with appropriate sampling and length
  - Spectral analysis on observation signal with length $N$.

- Spectral analysis is conditioned by apodisation window and size
  - Amplitude resolution $\lambda$
  - Frequency resolution with $N$ and $\Delta \Omega_m$

- Frequency resolution can be enhanced with the use of zero padding
  - Avoid reconstruction error during spectral analysis

Spectral analysis is an estimator: performance in terms of variance, bias, and convergence will be studied in a dedicated lecture.
Multirate processing
Introduction

Signal to be sampled must respect Shannon theorem [10].

- For the moment we have considered only one sampling frequency
- In some (many !) cases, frequency sampling can be changed during processing.

Example of low pass filtering and downsampling:

![Figure: Signal with low bandwidth (wrt sampling frequency)](image)

Modification of sampling frequency

- Reduction of sampling frequency: downsampling
- Increasing of sampling frequency: upsampling

Downsampling

- How ensure that signal is not distorted?
- How design the downsampling filter?

Upsampling

- How ensure that signal is still a low bandwidth signal
- How design the interpolation filter?

How downsampling and upsampling can be done with arbitrary factors?
Downsampling

Reduction of sampling frequency.

- Reduction is done with a downsampling factor $M$ (assumed to be integer)

We start from continuous signal

$$x(n) = x_c(nT_c)$$

with $T_c$ initial sampling time. Reduction of sampling frequency corresponds to increase of sampling time.

$$x_d(n) = x_c(nT_d)$$

with $T_d$ new sampling time, greater than $T_c$, with the relationship

$$T_d = M \times T_c$$

$$F_d = \frac{F_c}{M}$$
Downsampling

Downsampling condition

- A signal can be sampled if it respects the Shannon theorem

A signal can be downsampled at rate $F_d$ only if it respects Shannon theorem with respect to $F_d$.

Signal shall have limited bandwidth: We denote $\omega_0$ the maximal pulsation of $x_c$. It means

$$X_c(j\omega) = 0 \text{ for } |\omega| \leq \omega_0$$

Downsampling can be done (no aliasing) only if

$$\pi / T_d = \pi / (MT_c) \geq \omega_0$$

$$2\pi F_d = 2\pi F_c / M \geq \omega_0$$
Downsampling

Classic downsampling scheme

\[ x[nT_c] \rightarrow \downarrow M \rightarrow x[nT_d] \]

\[ T_d = M \times T_c \]

\[ F_d = \frac{F_c}{M} \]

**Figure:** Downsampling principle

- What is the spectral consequence of downsampling?
- Why an anti-aliasing filter?
Figure: Sampling impact on frequency domain
Downsampling

Real signals have leakages

- Aliasing if signal is not band limited
- In practise, leakages at high frequency (noise, replica, . . . )

Before downsampling, always apply a Low pass filter to avoid aliasing

Which filter?

- See IIR/FIR synthesis part

\[
x[nT_c] \xrightarrow{\text{Low pass filter}} x[nT_d]
\]

\[
\begin{align*}
\text{Gain} &= 1 \\
\Omega_c &= \frac{\pi}{M}
\end{align*}
\]
Upsampling

Increasing of sampling frequency.

- Increasing is done with a upsampling factor $L$ (assumed to be integer)

We start from continuous signal

$$x(n) = x_c(nT_c)$$

with $T_u$ initial sampling time. Increasing of sampling frequency corresponds to increase of sampling time.

$$x_u(n) = x_c(nT_u)$$

with $T_d$ new sampling time, lower than $T_c$, with the relationship

$$T_u = \frac{T_c}{L}$$

$$F_u = L \times F_c$$
Upsampling

How can we do an upsampling in practise?

- From signal at rate $F_c$, with insertion of zero

$$x_e(n) = \begin{cases} 
    x(n/L), & n = 0, \pm L, \pm 2L, \ldots \\
    0, & \text{otherwise}
\end{cases}$$  \hspace{1cm} (94)

or equivalently

$$x_e(n) = \sum_{k=-\infty}^{\infty} x(k) \cdot \delta(n - kL)$$  \hspace{1cm} (95)

$$x[nT_c] \xrightarrow{\uparrow L} x[nT_d]$$

$$T_d = \frac{T_c}{L} \times T_c$$

$$F_d = F_c \times L$$
Upsampling

What happens in frequency domain?

- Sampling in time at $T_d$ ➔ Periodisation in frequency domain at each $F_d$

**Figure:** Upsampling consequence (upsampling factor of 3)
Upsampling

Need to apply a low pass filtering to remove images

- Signal with a maximal frequency of $\omega_0$
- Cut-off frequency of LPF filter with value $\omega_0/L$ ($\pi/L$ in full band mode)

Figure: Upsampling consequence (upsampling factor of 3) followed by digital filtering
How can we do an upsampling in practise?

- From signal at rate $F_c$, with insertion of zero
- Followed by a low pass filtering

$$T_u = \frac{T_c}{L} \times T_c$$

$$F_u = F_c \times L$$
Fractional interpolation

Downsampling: Frequency reduction with integer factor
Upsampling: Frequency increase with integer factor

How can we manage fractional upsampling or fractional downsampling?

Output frequency $F_o = k \times F_c$ with $k \in \mathbb{Q}$.

$$F_o = \frac{L}{M} F_c$$

- Fractional interpolator can be seen as upsampling followed by a decimation
Fractional interpolation

$\begin{align*}
x[n_{T_c}] & \rightarrow \uparrow L \rightarrow \text{LPF filter} \rightarrow x[n_{T_d}] \rightarrow \text{LPF filter} \rightarrow \downarrow M \rightarrow x[n_{T_d}]
\end{align*}$

- Fractional interpolator can be seen as upsampling $L$ followed by a decimation $M$

**Fractional interpolation and filter design**

- Filter after upsampling stage
- Filter before downsampling stage

**Common interpolation filter**

- Only more selective filter has to be applied

$$\Omega_c = \min \left( \frac{\pi}{L}, \frac{\pi}{M} \right)$$
Conclusion on multirate processing

One functional frequency is not enough

- Need to decrease: downsampling
- Need to increase: upsampling

Downsampling

- Keep one sample per $M$
- Before dropping samples, LPF filter with $\frac{\pi}{M}$

\[
x[nT_c] \xrightarrow{\text{Low pass filter}} x[nT_d]
\]

\[
\text{Gain} = 1 \\
\Omega_c = \frac{\pi}{M}
\]

\[
\downarrow M
\]
Conclusion on multirate processing

Upsampling

- Insert $L$ zeros between samples
- Spectral images have to be removed with LPF filter of $\frac{\pi}{L}$

\[
x[nT_c] \xrightarrow{\uparrow L} \xrightarrow{\text{Low pass filter}} \xrightarrow{\Omega_c = \frac{\pi}{L}} x[nT_u]
\]

Fractional interpolator

- Changing rate with a $\frac{L}{M}$ rate
- Common filter (more selective applied) between DS and US

\[
x[nT_c] \xrightarrow{\uparrow L} \xrightarrow{\text{LPF filter}} \xrightarrow{\Omega_c = \min\left(\frac{\pi}{L}, \frac{\pi}{M}\right)} \xrightarrow{\downarrow M} x[nT_d]
\]
Digital Signal Processor
Architecture comparison

Energy efficiency

\[ E_E = \frac{P_c}{C} (MIPS / mW) \]

- **ASIC**: 100-1000 MOPS/mW
- **Embedded FPGA**: Flexibility
- **Embedded Processor**: SA110
  - 2 V DSP
  - 0.4 MIPS/mW
- **Reconfigurable Processor**: Pleiades
  - 10-50 MOPS/mW
- **ASIP**: Flexibility
- **DSP**: Pleiades
  - 3 MOPS/mW
- **Alpha**: Flexibility
  - 0.007 MIPS/mW

100-1000 MOPS/mW

**Digitally Signal processing**

Digital Signal processor

Introduction

382 / 420
Specific instructions dedicated to signal processing
Specific architecture
Architecture example

In practise, several components for one application

- Hardware block (analog)
- DSP based, $\mu$C based
- Processor based (e.g. arm, co-processor, ...)

Figure: Example of GSM architecture
DSP core units

Several unity dedicated to different purpose

Instruction processor

- Also called control unity
- Interpret instructions, and command other unities

Data processor

- Modify and update data

Memory unity

- Instruction memory that stores instructions
- Data memory that buffers data from data processor

Communication unit

- External interface unit
- Pilot access to data (or external instructions) and other processor links.
DSP core units

Control unit: I.P.

Data unit
D.P.

Memory unit
I.M. + D.M.

Communication unit EIU

Figure: Synoptic of DSP architecture
Processing units

Processing is done depending on topology

Fixed point

- One sign, on integer part, on fractional part
- Two’s complement implementation
- \((b,m,n)\) or \(Q(m,n)\) with \(b = m + n - 1\)

Floating point

- One sign, exponent and fraction
- Resolution depends on coded word
- IEEE 754 norm
Operation

Hardware multiplier
- Perform multiplication in one cycle
- Output is stored in register or given to ALU

Arithmetic and Logic Unit (ALU)
- Arithmetic operation (add/sub, sign shift, ...)
- Logical operation (or/not, ...)

Shift registers
- Cylinder shift: perform arbitrary shifting in one cycle

Specific blocks
- Depends on DSP (LMS, Viterbi, ...)

Storing register
Operation during a MAC

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input bits</th>
<th>Output bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplier</td>
<td>b</td>
<td>2b</td>
</tr>
<tr>
<td>UAL (add)</td>
<td>2b</td>
<td>2b</td>
</tr>
<tr>
<td></td>
<td>2b + bg</td>
<td>2b + bg</td>
</tr>
<tr>
<td>Saturation</td>
<td>2b</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>2b + bg</td>
<td>b</td>
</tr>
</tbody>
</table>

Registers used:

<table>
<thead>
<tr>
<th>Register</th>
<th>Bit number</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input operand</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>Accum</td>
<td>2b</td>
<td>2b + bg</td>
</tr>
</tbody>
</table>

Diagram of MAC operation with the following registers:
- A
- B
- Accumulateur
- Sat / Arr
- b_{nat}
- b_{mult}
- b_{add}

Digital Signal processing
Digital Signal processor
DSP core units
If we consider a classic Von Neumann architecture

```
loop:
    mov *r0,x0
    mov *r1,x1
    mpy x0,x1,a
    add a,b
    mov x1,*r2
    inc r0
    inc r1
    inc r2
    dec ctr
    tst ctr
    jnz loop
```

15 à 20 cycles for execution

**Figure:** FIR operation with classic architecture

- Memory pointer to input data
- Data processing (multiplication and addition)
- Aging data ($x[n] \rightarrow x[n-1]$)
- Update memory pointers
Classic processor architecture problem

Purpose: do these operation in one clock

1. Getting input data $x[n - k]$
2. Getting filter coefficient $h[k]$
3. Process data and accumulate
4. Aging data

Communications between memory and processing can struggle the effective rate.

Architecture

- Von Neuman: Unique memory unit for both program and data
- Harvard: Separation in two memory units (one for program and one for data). Each of them have a separate communication bus
DSP architectures

Based on Harvard architecture [26]
- Separation of memory unit and processing unit
- Faster processing compared to Von Neuman
- Pipeline: Instruction fetch → operand fetch

Figure: Classic Harvard architecture for DSP

Example: TMS320C10

DSP architectures

- Harvard: first variant [26]

- Data can be stored in IM

- Two operand can be accessed if IM supports two access in one cycle
  - Instruction fetch
  - 2 operand fetches
  - MAC execution
  - Output in I/O or in memory

![Diagram of Classic Harvard architecture for DSP]

**Figure:** Classic Harvard architecture for DSP

Example: AT&T DSP32 or DSP32C

---

DSP architectures

Harvard: Second variant [26]
- DM is a multiport memory
- Several access to memory per cycle

**Figure:** Classic Harvard architecture for DSP

Example: Fujitsu MB86232 (3 ports)

DSP architectures

Harvard: Third variant [26]
- 2 separated DM
- If time access to memory is the same, process 2 operand per cycle

![Diagram of Classic Harvard architecture for DSP]

Figure: Classic Harvard architecture for DSP

Example: Motorola DSP 56001 - 96002; TMS320C30 - C40

DSP architectures

Harvard: Fourth variant [26]

- IM can store operand
- Cache memory for frequently used instructions (ex. loop)
- Avoid memory access conflict (in version 1)

Figure: Classic Harvard architecture for DSP

Example: TMS320C25 (cache 1), ADSP-2100 (16-cache)

Conclusion on DSP architecture

Von Neuman architecture is not well adapted
- DSP is real time data oriented processing

Harvard architecture for DSP
- One Data Memory (DM), one Instruction Memory (IM)
- Several variant to speed up processing
  - Allow use of IM to store data
  - Multiport DM
  - Several DM
  - DM/IM combined with cache

How to choose? Depends on other constraints, and operand location: see [27]

Operand location

Register - memory model

- Use one data from memory and one data in register
- In one cycle, only one memory lecture is done
- Legacy Harvard architecture can be used
- Instruction execution time

\[ t_{\text{inst}} = t_{\text{acc}} + t_{\text{exec}} \]

- With \( t_{\text{acc}} \) duration of reading from memory
- \( t_{\text{exec}} \) execution of operation

- Minimization of \( t_{\text{acc}} \) with static memory (and pagination)

Figure: Register-memory model
Operand location

Memory-memory model (load-store)

- Avoid access to memory: operand are in registers
- Instruction and memory-to-register is done simultaneously
- Next operand are moved to register during current instruction
- Instruction execution time

\[ t_{\text{inst}} = \max (t_{\text{acc}}, t_{\text{exec}}) \]

- With \( t_{\text{acc}} \) duration of reading from memory
- \( t_{\text{exec}} \) execution of operation
- 2 operand are read: multi-port DM or several DM architecture

Figure: Register-register (load-store) model
Addressing modes

Data Memory is frequently used. How to map address in an efficient way?

Immediate addressing

- Instruction contains the operand value
  ADD #4, A
- Useful for initialization and for small data
- 2 words is needed to code the instruction for long word

Direct addressing mode

- Instruction contains the address of the operand
- If full address is coded, 2 words is needed
- Possibility to use only one word with paged memory
Addressing modes

Data Memory is frequently used → How to map address in an efficient way?

Direct addressing mode

- Paged memory is used to partially encode address
- Address is a page pointer (in register) and an offset
  - Data pointer and Stack pointer

\[
\begin{align*}
LD \ #x, \ DP & \quad \text{Loads MSB of } x\text{’s address in page pointer} \\
LD @x + 1, A & \\
ADD @x, A & \\
ADD @x + 2, A & 
\end{align*}
\]

In register (DP/SP)

9 bits ← TMS320C54x → 7 bits

In instruction
Addressing modes

Data Memory is frequently used. How to map address in an efficient way?

Indirect addressing mode

- Data in memory is pointed by an auxiliary register
- Instruction contains the register index
- Limited number of registers limits the bit use
- Particularly efficient in DSP architecture
- Natural array indexing (pointer) and BK to use circular buffer

Data Memory

Start_address = xxxxxxxxxxxxxxx00000
ARi
End_address = xxxxxxxxxxxxxxx11111
xxxxxxxxxxxx00010
ARi BK
N=30=1 1 1 1 0
Addressing modes

Data Memory is frequently used → How to map address in an efficient way?

Indirect addressing mode

- Possibility to update (calculate) the next used address
- Linear, indexed, modulo, circular buffer operations

• Register points to data
  • LD *AR1, A

• Post-Modification:
  • linear: AR := AR ± 1
  • indexed: AR := AR ± MR
    • MR: register d’index
  • modulo: (AR := AR ± 1)_N
  • bit-reverse: FFT

 addr = AR1 (A ◦ *AR1)

 addr = AR1
 AR1 = AR1 + 1

 addr = AR1
 AR1 = AR1 + AR0

 addr = AR1
 AR1 = (AR1 + 1) modulo BK

(BK) specifies the size of the circular buffer.

After access, AR0 is added to ARx with reverse carry (rc) propagation.
Addressing modes

Data Memory is frequency used How to map address in an efficient way?

Indirect addressing mode

• **Circular buffer**
  • Aging data is automatic (shifting address)

![Circular buffer diagram]

• **Bit-Reverse**:
  • Addressing data in FFT stages

<table>
<thead>
<tr>
<th>Index</th>
<th>bit</th>
<th>bit reverse</th>
<th>index bit reverse</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>011</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>111</td>
<td>7</td>
</tr>
</tbody>
</table>
Example of TSM320C4x

- 8 auxiliary registers (AR0...AR7)
- 2 processing unit ARAU
- 2 specific register
  - BK : circular buffer size
  - Index register AR0
Control unit

Classic pipeline used in several steps

1. Instruction search
2. Decoding instruction
3. Operand search
4. Execution
Stationnarity (time domain) from unit point of view

- **MULT R1, R2, T**
- **ADD T, A, A**
- **LD R1, AR1**
- **LD R2, AR2**

### Diagram

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>i+1</th>
<th>i+2</th>
<th>i+3</th>
<th>i+4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory transfert.</td>
<td>LOAD</td>
<td>LOAD</td>
<td>LOAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult Unit</td>
<td>MULT</td>
<td>MULT</td>
<td>MULT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACUM unit</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Instruction are coded sequentially
**Control unit**

Stationnarity (data domain) from data processing point of view

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>i+1</th>
<th>i+2</th>
<th>i+3</th>
<th>i+4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory transfert</td>
<td>LOAD</td>
<td>LOAD</td>
<td>LOAD</td>
<td>LOAD</td>
<td>LOAD</td>
</tr>
<tr>
<td>Mult unit</td>
<td>MULT</td>
<td>MULT</td>
<td>MULT</td>
<td>MULT</td>
<td>MULT</td>
</tr>
<tr>
<td>Accum unit</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
</tr>
</tbody>
</table>

- Simpler to code as closer to algorithm
Some DSPs

Different baselines

- Fixed point on 16-24 bits (accumulation 40-60 bits)
- 32 bits in floating point
- Instruction set: 16 - 32 bits
- One instruction per cycle (complex set)

leads to different architectures

- Strong architecture constraints
- External memory (on-chip)
- Hardware for loop management
Some DSPs

Texas instrument

- C2000: Fixed point 16 bits: motor control
- C5000: Fixed point 16 bits: Low consumption
- OMAP: C55x + µP ARM
- C6000: Fixed point 16 bits (C67 floating): High performance

Analog device

- ADSP21xx: Fixed point 16 bits
- SHARC: Floating point 32 bits
Specific instruction set

MAC operation

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Execution</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC[R] Smem, src</td>
<td>(Smem) x (TREG) + (src) --&gt; src</td>
<td>MAC *AR5, A</td>
</tr>
<tr>
<td>MAC[R] Xmem, Ymem, src [,dst]</td>
<td>(Xmem) x (Ymem) + (src) --&gt; src or [,dst] (Xmem) --&gt; TREG</td>
<td>MACR *AR5+, *AR6+, A, B</td>
</tr>
<tr>
<td>MAC #lk, src [,dst]</td>
<td>(TREG) x lk + (src) or [,dst]</td>
<td>MAC #345h, A, B</td>
</tr>
<tr>
<td>MAC Smem, #lk, src [,dst]</td>
<td>(Smem) x lk + (src) or [,dst] (Smem) --&gt; TREG</td>
<td>MAC *AR5+, #1234h, A</td>
</tr>
</tbody>
</table>

Description

The MAC[R] instruction multiplies and adds with or without rounding. The result is stored in the destination accumulator, if specified, or in the source accumulator. For syntaxes 2 and 3, the data-memory value after the instruction is stored in TREG. TREG is updated during the read phase. The MACR instruction rounds the result of the MAC operation by adding $2^{15}$ to the result and clearing the 16 LSBs (bits 15-0) to 0.

Operands

Smem: Single data-memory operand
Xmem, Ymem: Dual data-memory operands
src, dst: A (accumulator A), B (accumulator B)
### Description

FIRS is useful to implement symmetrical FIR filters. The FIRS instruction multiplies accumulator A(32-16) with a program memory value addressed by pmad (program memory address) and adds the result to the value in accumulator B. At the same time, it adds the memory operands Xmem and Ymem, shifts the result left 16 bits and loads this value into accumulator A. In the next iteration, pmad is incremented by 1. Once the repeat pipeline is started, the instruction becomes a single-cycle instruction.

### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Execution</th>
</tr>
</thead>
</table>
| FIRS Xmem, Ymem, pmad | pmad --> PAR  
While (RC) != 0  
(B) + (A(32-16)) x (Pmem addressed by PAR) --> B  
((Xmem) + (Ymem)) << 16 --> A  
(PAR) + 1 --> PAR  
(RC) - 1 --> RC |

### Example

FIRS *AR3+, *AR4+, COEFFS
Specific instruction set

LMS operation

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Execution</th>
<th>Example</th>
</tr>
</thead>
</table>
| LMS Xmem, Ymem | (A) + (Xmem) << 16 + 2^15 --> A  
(B) + (Xmem) x (Ymem) --> B | LMS *AR3+. *AR4+   |

Description

The LMS instruction is used to execute the least mean square algorithm (LMS). The dual data-memory operand Xmem is shifted left 16 bits and added to accumulator A. The result is rounded by adding $2^{15}$ to the high part of the accumulator (bits 31-16). The final result is stored in accumulator A. In parallel, Xmem and Ymem are multiplied and the result is added to accumulator B. Xmem does not overwrite TREG; therefore, TREG always contains the error value used to update coefficients.
Conclusion on DSP

- Digital signal Processor target DSP applications
- Many constraints linked to real time data oriented processing

Processing unit

- Fixed point or floating point based
- Processing unit are implemented in DSP
  - ALU
  - MAC
  - Additional blocks

Memory architecture

- Harvard based topology
- Several variant depending on operand location
- Direct or indirect addressing modes
Conclusion on DSP

Control unit

- Pipeline based architecture
- Instruction can be coded to be stationary in time domain, or in data domain
- Hardware loop to optimize resource management

Topology depends on target DSP

- Various architectures and units
- Workflow must be adapted to targeted DSP
References
References


References


References


