1674 AUD

Creating Audio and DSP Applications with 16/32 bit Microcontrollers

Objectives

- Learn how Digital Filtering works
- Understand and use speech compression techniques
- Review PIC32 external codec interface
- Construct a MP3 player using Helix MP3 decoder algorithm

MICROCHIP
MASTERs 2012

Agenda

- Digitization of Signals
- Digital Filtering
- Lab 1 – Digital Filtering
- Speech Coding
- Speech Compression
- Lab 2 – Speex Algorithm
- External Codec Interfacing
- Audio Coding
- Helix MP3 Algorithm API
- Lab 3 – Helix MP3 Algorithm

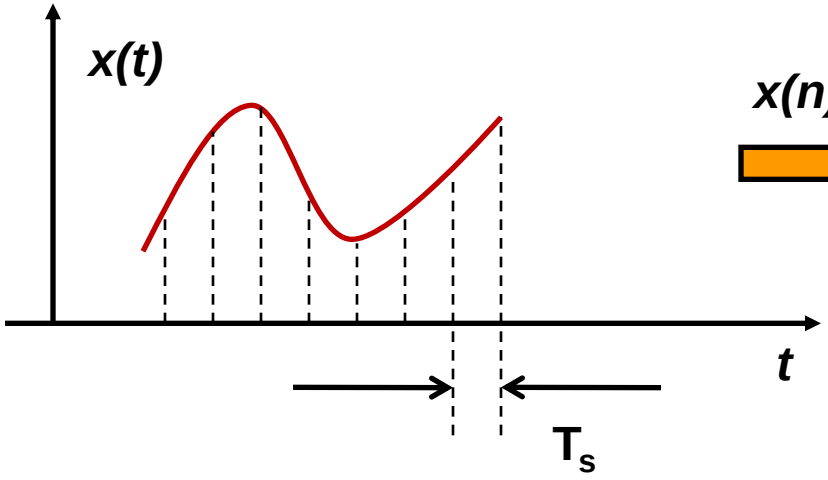
Labs

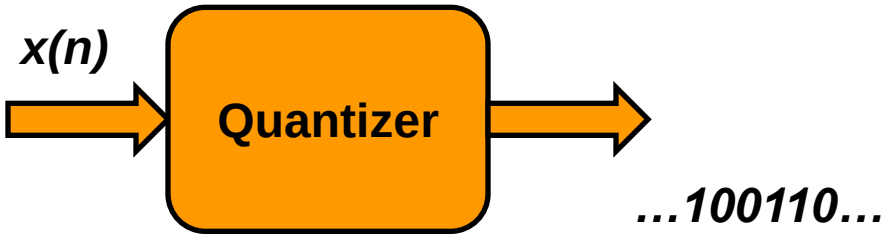
- **Lab 1 – Remove unwanted tone from speech signal**
- **Lab 2 – Develop a voice recorder**
- **Lab 3 – Develop a USB thumb drive based MP3 player**

Digitization of Signals

Digitization

- **What is digitization**
  - Sampling (Discrete time)
  - Quantization (Discrete amplitude)
  - Analog to Digital Conversion



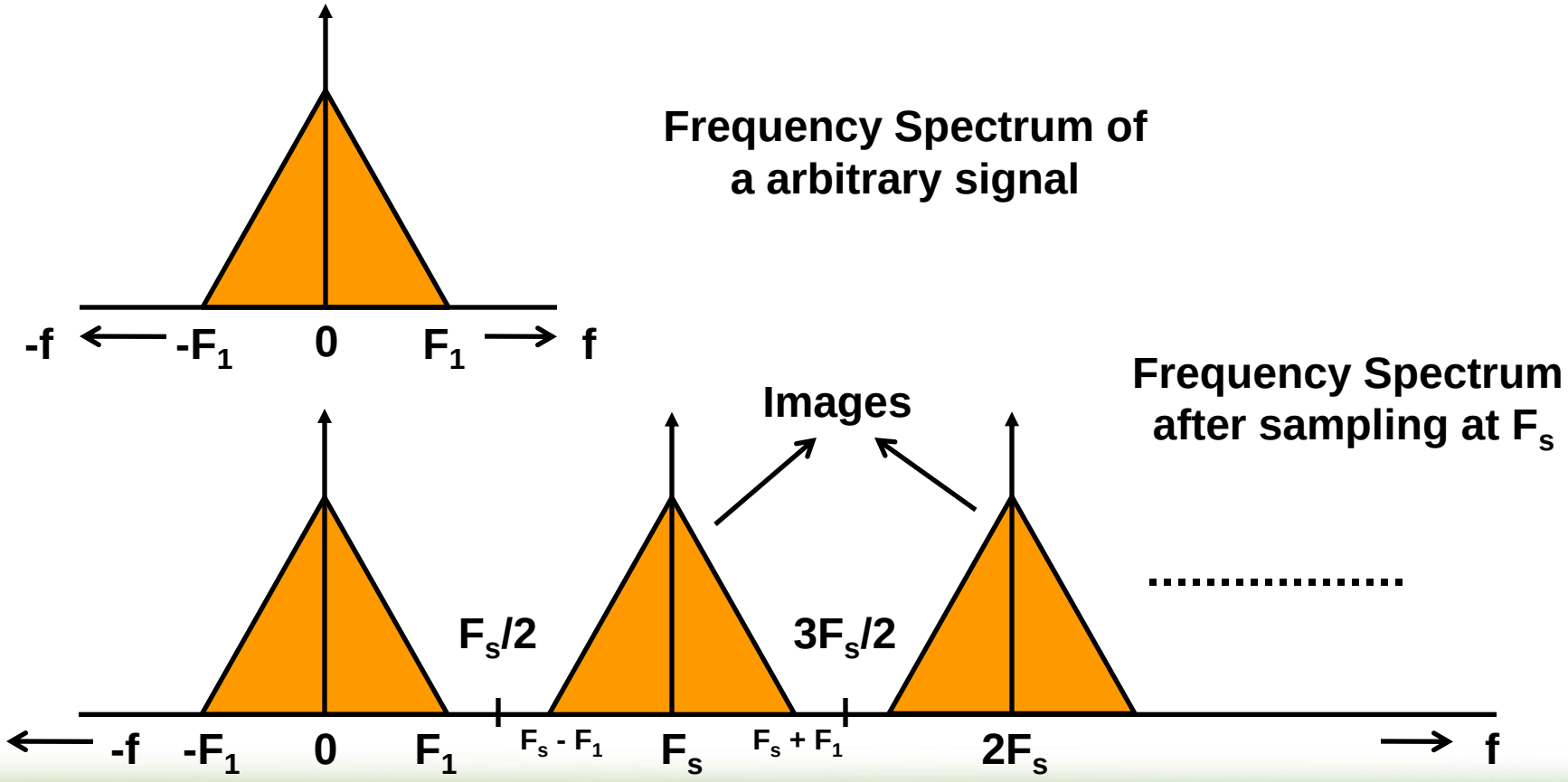


Digitization

- **Some comments**
  - Sampling frequency $F_s = 1/T_s$
  - Bit resolution (how much)
    - **Application dependant**
    - **Cost sensitivity**
  - '*t*' is the continuous time index
  - '*n*' is the discrete time index

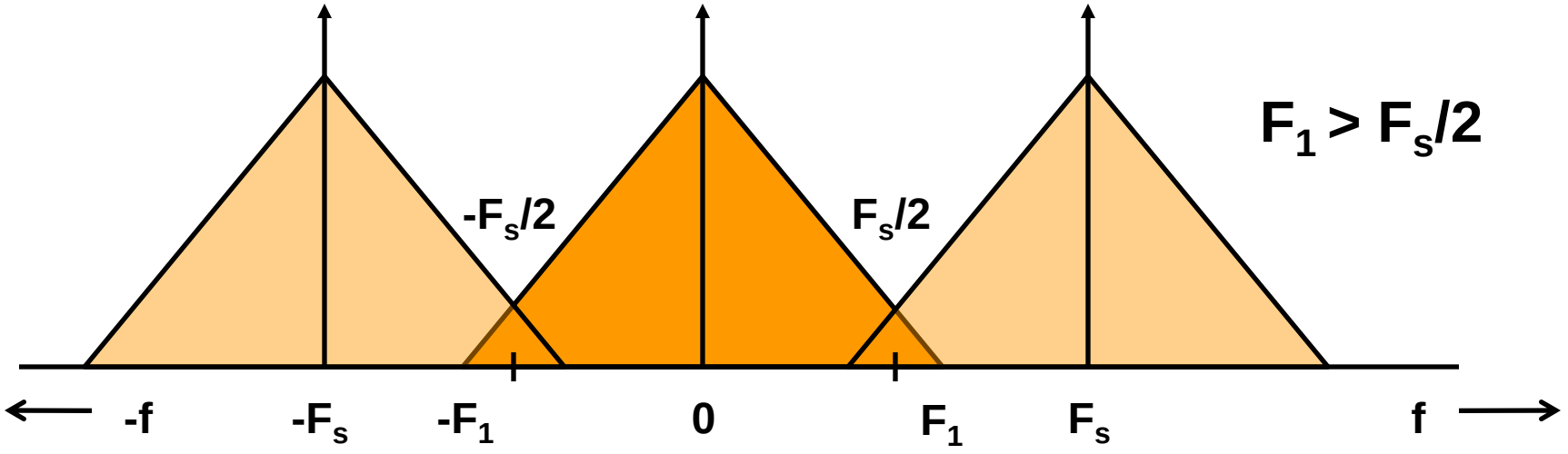
Sampling Operation

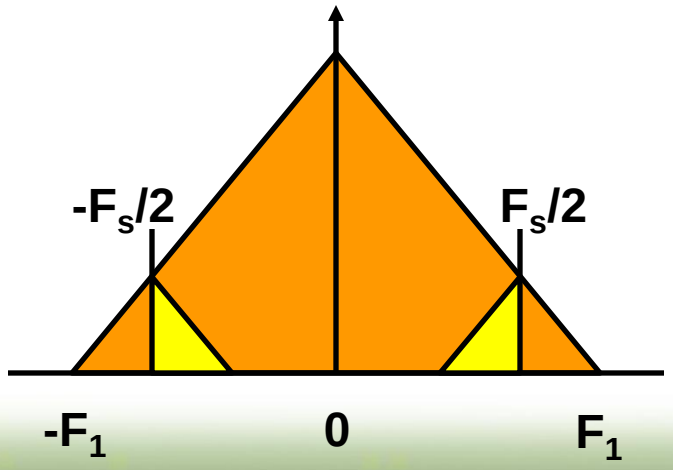
- What happens when you sample a signal



Aliasing

- What is aliasing

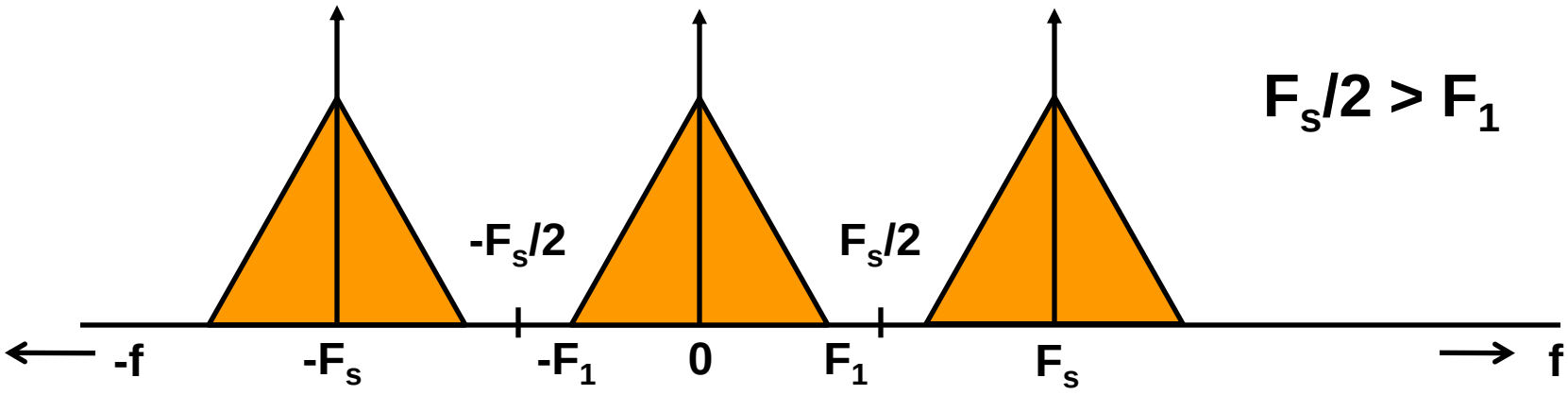




- Signal is not sampled adequately
- Ambiguity in reconstruction
- High frequencies fold over into lower frequencies
- $F_s/2$ is called the folding frequency
- What happens when you sample 7600Hz signal at 8000Hz?

Aliasing

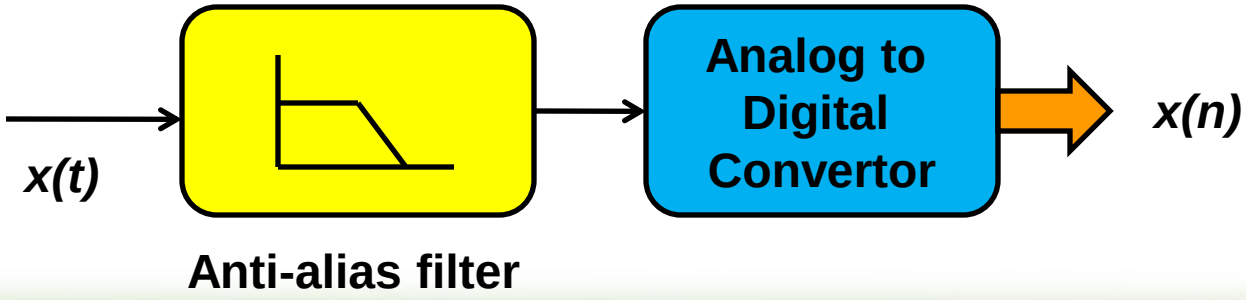
- **How to prevent it**
  - Ensure that Fs > $2F_1$ (Sampling theorem)



- **Sample at > twice maximum frequency**
- **Signal can be reconstructed**
- **F_s is the Nyquist rate**

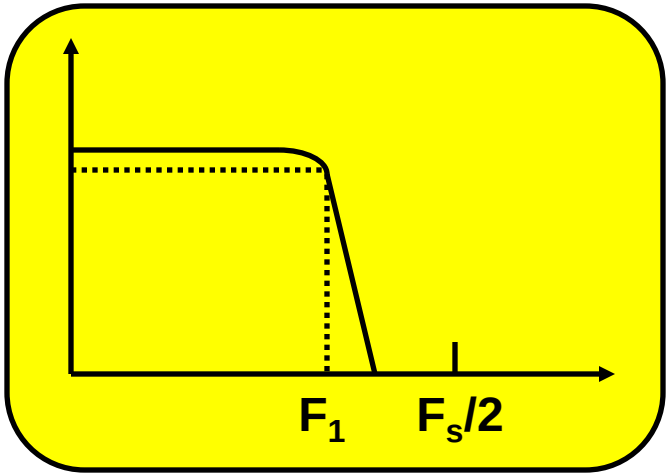
Practical Digitization

- **Limit the bandwidth of the input signal**
  - Anti-Alias low pass filter
  - Must be a good quality filter (op-amps)
  - Sharp roll-off
  - Typically has higher order
  - Stop band attenuation greater than 60dB
  - Tight specifications
- **Analog to Digital convertor (ADC)**

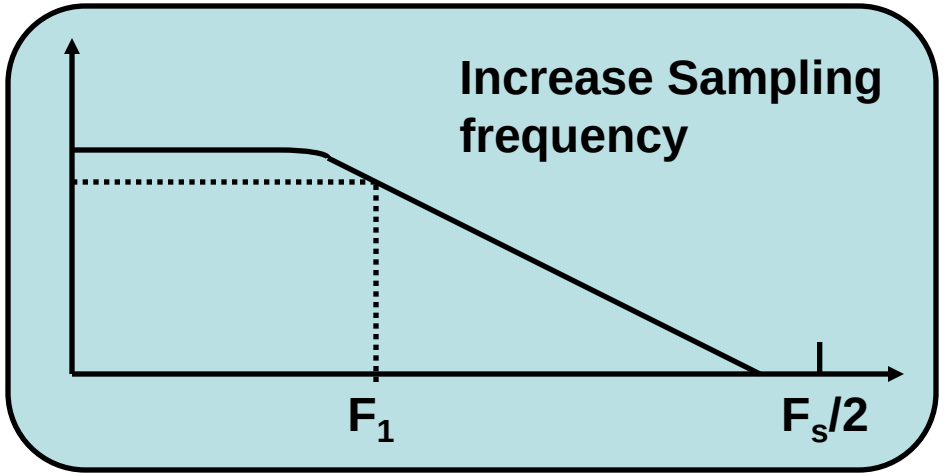


Oversampling

- Anti-aliasing filter is critical to system performance
- How to ease the filter specification?
- Use Oversampling technique



- Steep transition band
- Tight filter specifications
- No software overhead



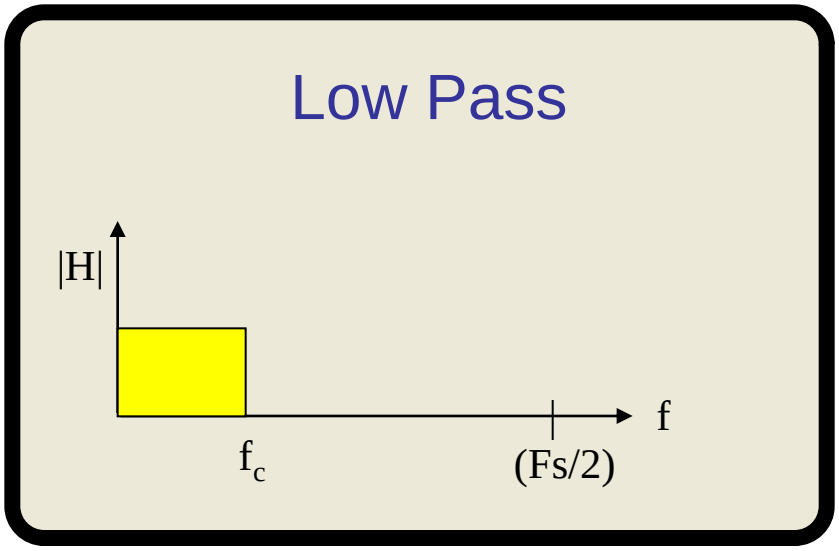
- Broader transition band
- Relaxed filter specifications
- Tighter filter implement in software
- Sample rate conversion

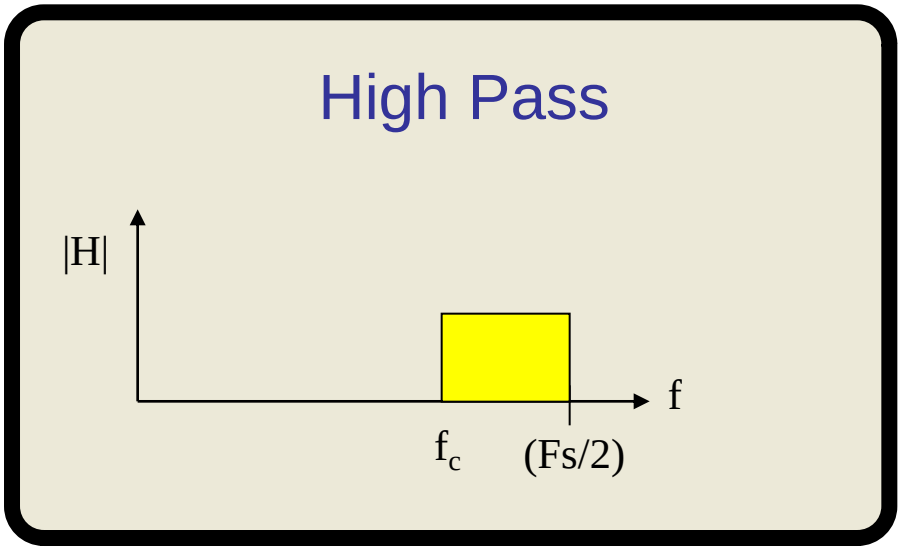
Digital Filtering

Digital Filters

- Important class of DSP applications
- Finite Impulse Response (FIR) Filter
- Infinite Impulse Response (IIR) Filters
- Linear Filtering using DFT

Filter Types

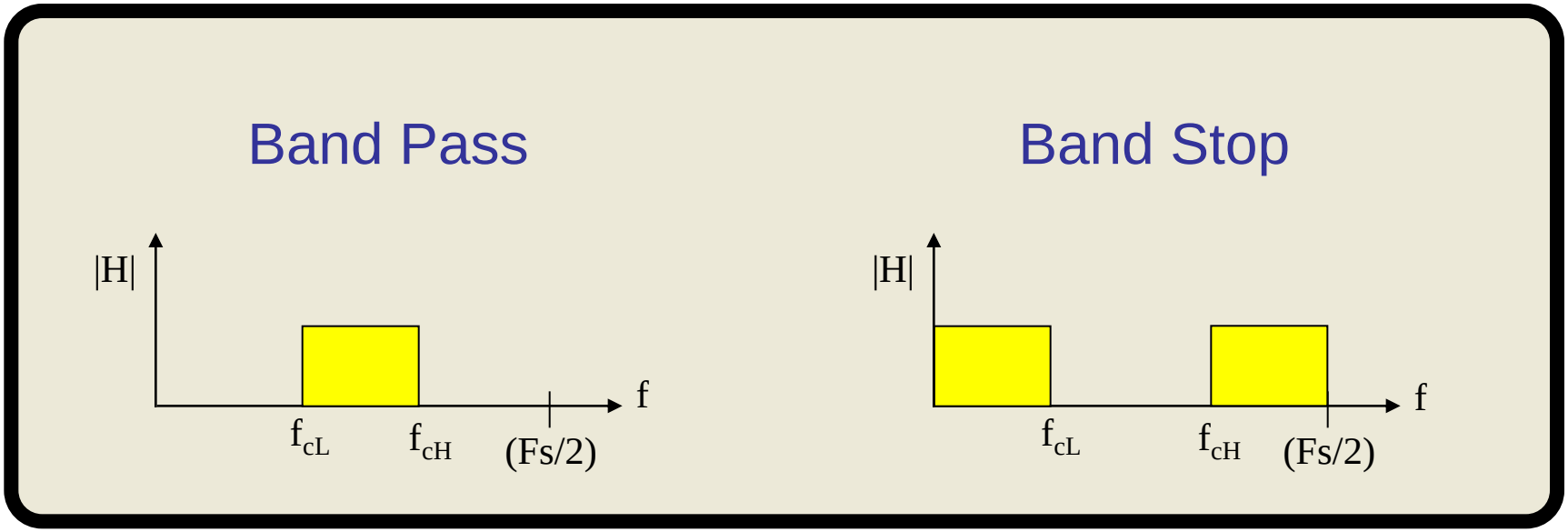




- **Applications**
  - Anti-aliasing
  - Smoothing
  - Noise Reduction
  - Treble Cut

- **Applications**
  - DC Removal
  - Power Line Hum
  - Bass Reduction

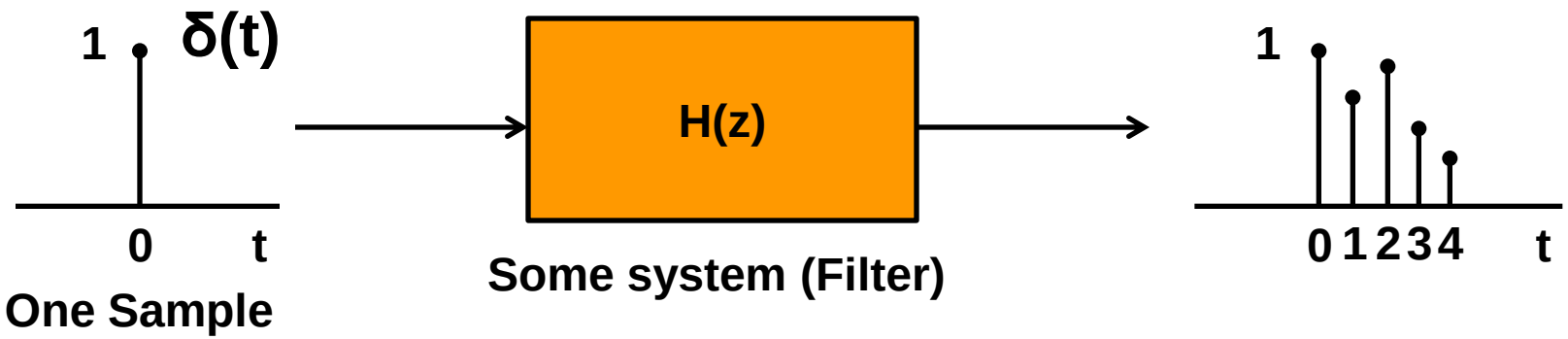
Filter Types



- **Applications**
  - Equalizer
  - Wireless transmitter and receivers
  - Filter banks in MP3 Compression

Impulse Response

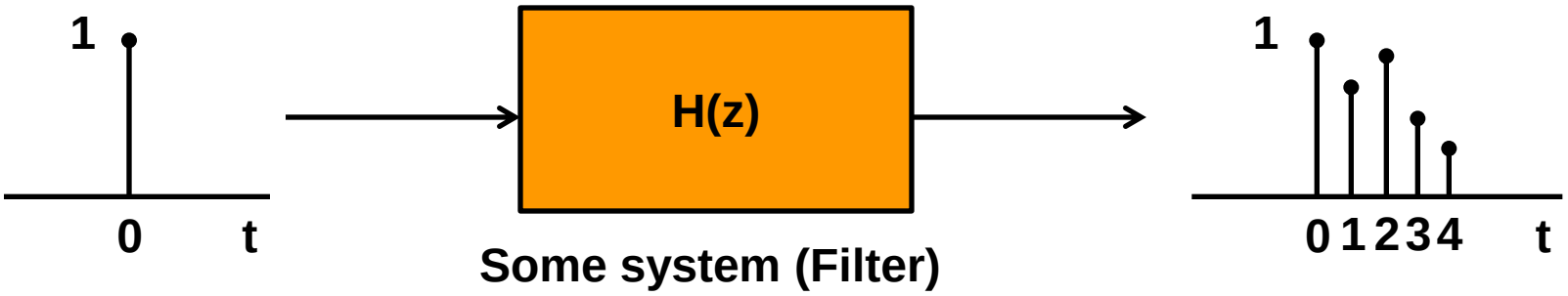
- What is an Impulse Response



- Response of a system to an impulse
- Response to the finest signal (temporal sense)
- Typically computed mathematically or through modeling
- Any system can be treated as a filter

What is a LTI system

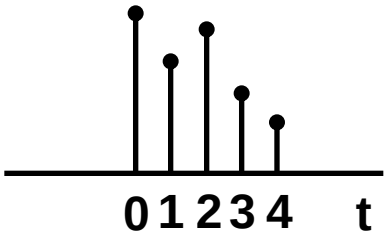
- What does Linear Time Invariant (LTI) mean?



- $f(x_1) + f(x_2) + f(x_3) = f(x_1 + x_2 + x_3)$ …superposition
- Impulse response does not change with time
- Any system (even non linear time variant) can be treated LTI for sake of analysis

Why Impulse Response

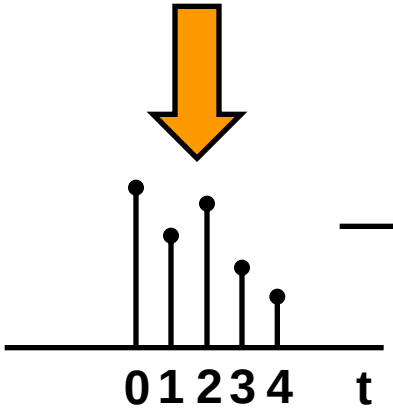
- Why is Impulse response important

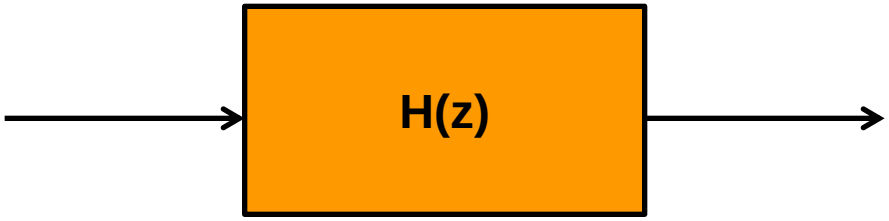


Arbitrary Digital Signal

$K_0\,\delta(t) + K_1\,\delta(t-1) + K_2\,\delta(t-2) + K_3\,\delta(t-3) + K_4\,\delta(t-4)$

Sum of Weighted and translated Impulses



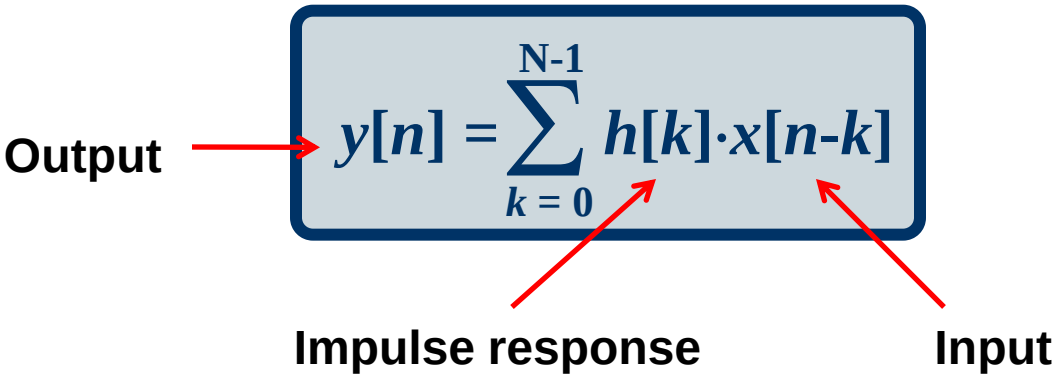


Some system (Filter)
Impulse Response Known

Output = Sum of weighted and translated impulse responses

Digital Filtering Basics

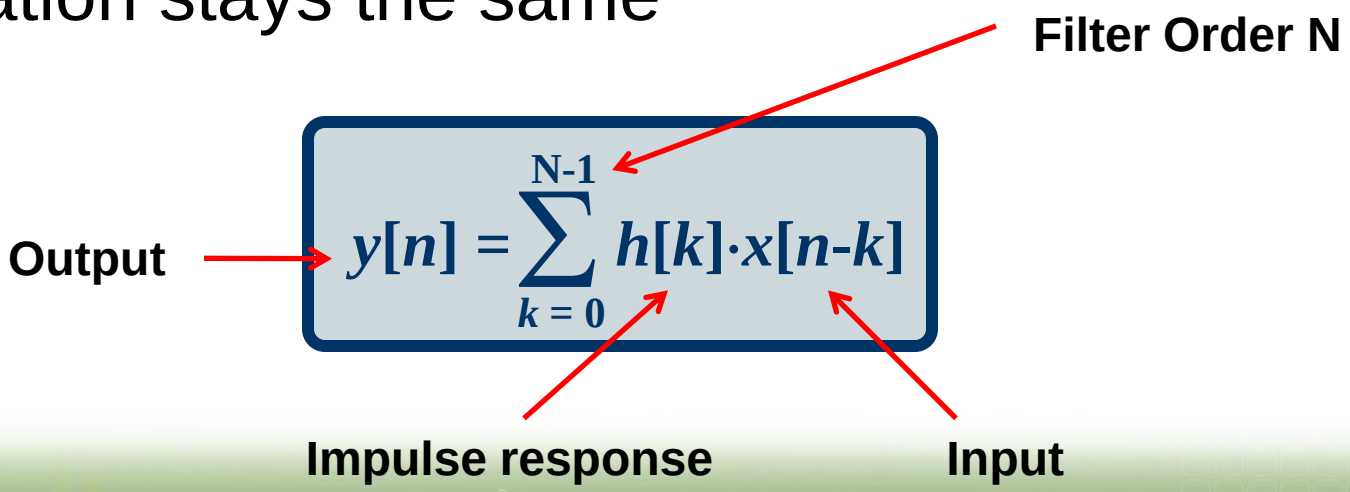
- **Impulse response helps compute output of a system**
- **Works only for LTI**
  - Impulse response can be scaled
  - Impulse response can be translated



- **Output computed by convolution sum**

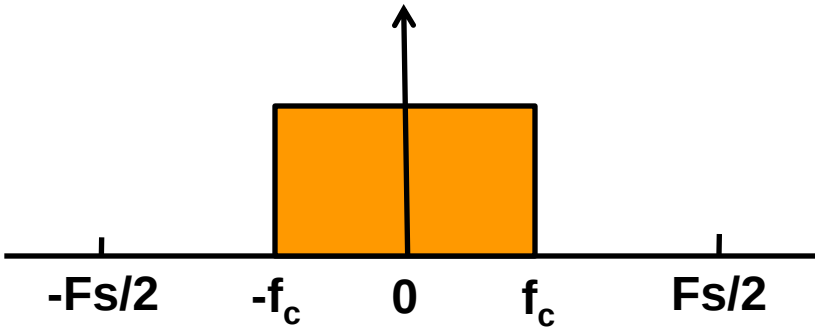
Finite Impulse Response (FIR) Digital Filters

- **Output equals sum of**
  - Weighted present input
  - Weighted past finite number of inputs
- **Weights are filter coefficients**
  - Coefficients define filter response
  - Equation stays the same

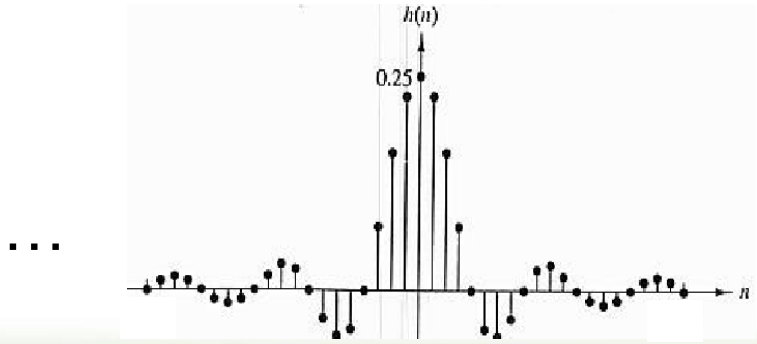


FIR Filter Design

- **Start with ideal frequency response of a low pass filter**



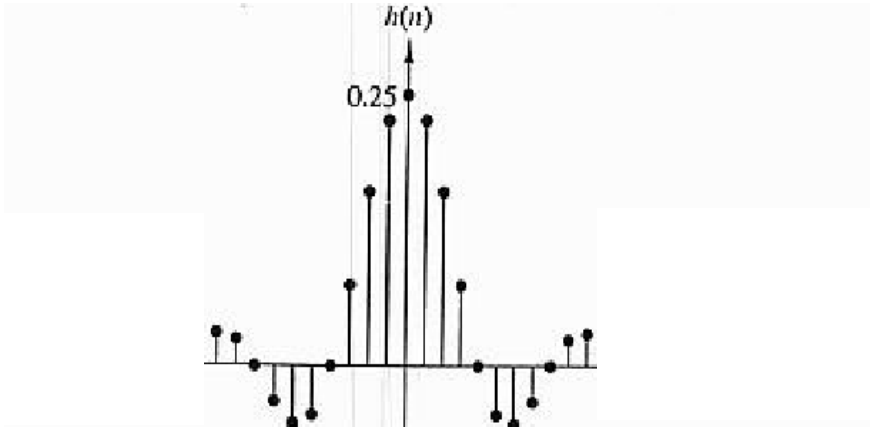
- **Take Inverse Fourier Transform to get Impulse Response**



dsPIC® DSC Filter Design Tool

FIR Filter Design

- **Impulse of Ideal Low pass filter extends to infinity**
  - Cannot deal with infinite sequences in the real world



- **Truncate Impulse Response around axis of symmetry**
- **Shift the sequence to make it causal**
- **Shifted sequence is h(n) or filter coefficients**
- **Process of truncating is called windowing**

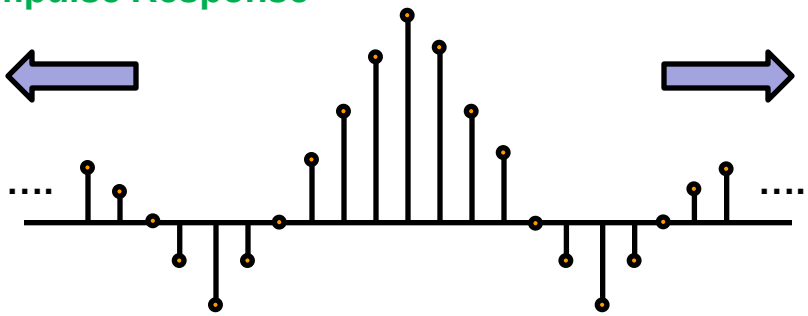
FIR Filter Design

- **What is a window**
  - Used to truncate Infinite Impulse Response (hence FIR)
  - Multiply window function by the impulse response
  - Choice of window affects
    - **Filter Transition band**
    - **Stop band attenuation**
  - Direct Truncation is same as using rectangular window of unit amplitude

Windowing

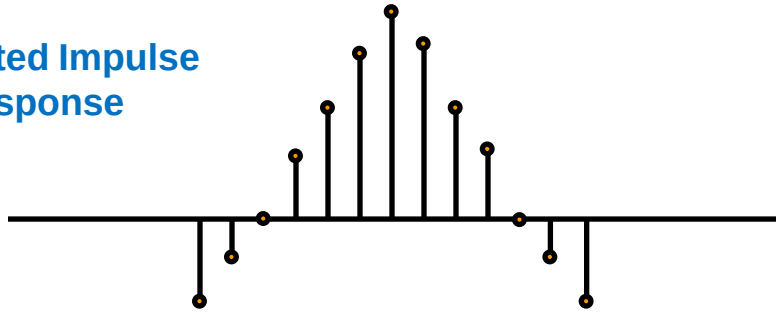
Rectangular Window

Ideal Impulse Response

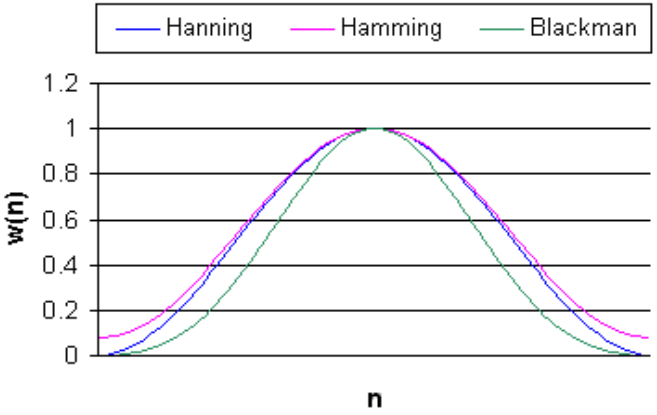


Multiplication (Windowing)

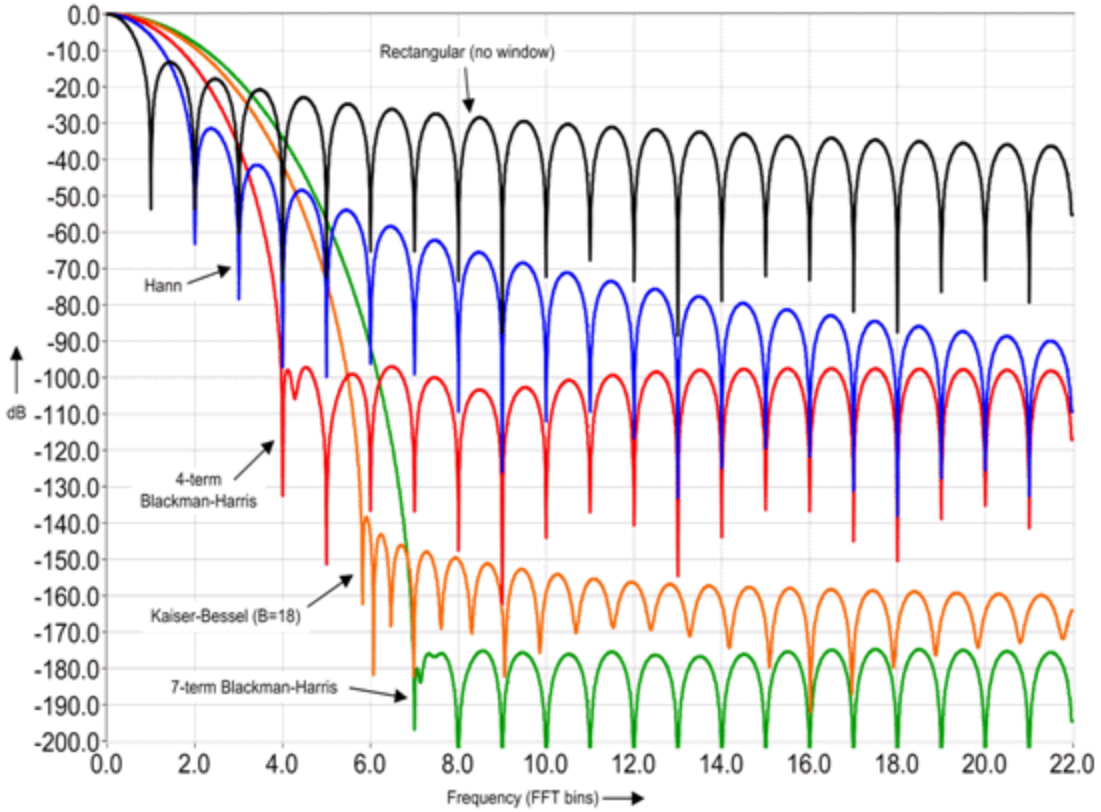
Truncated Impulse Response



Window functions



Window Frequency Response



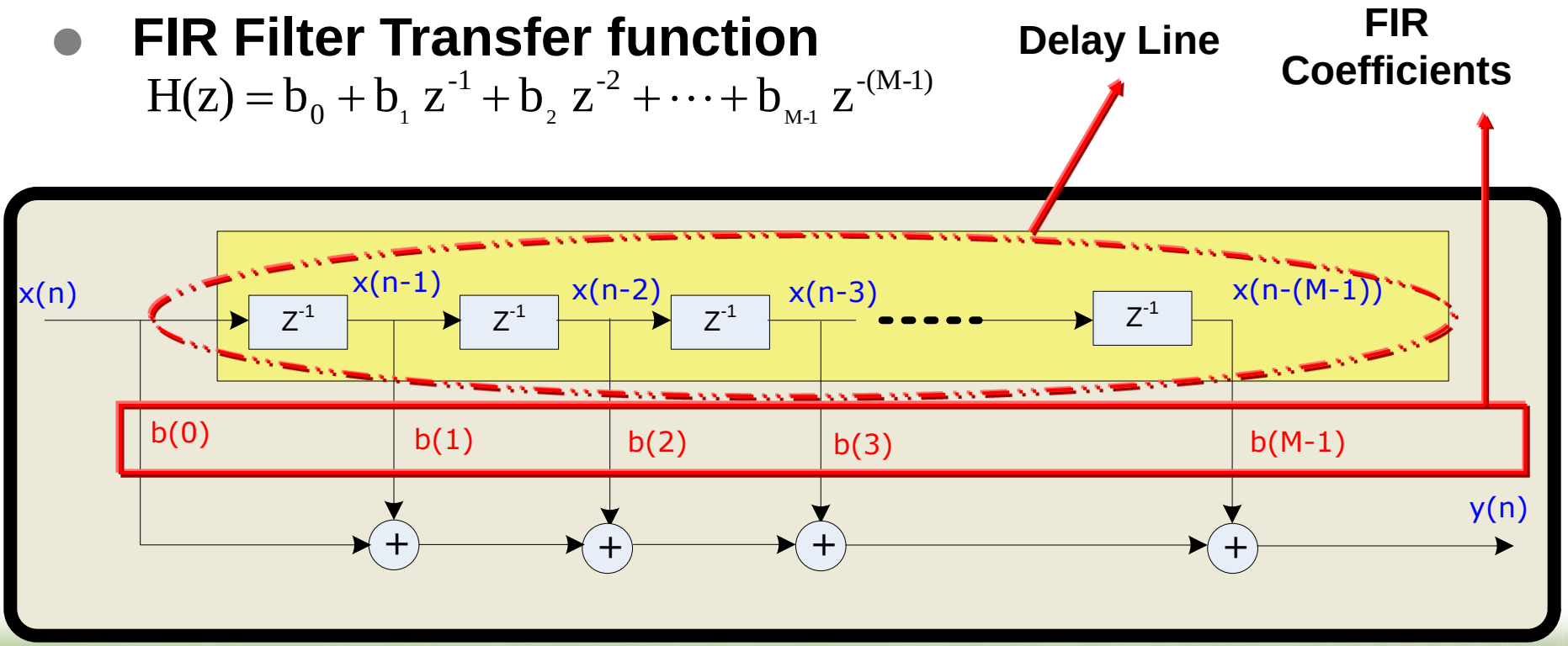
FIR Filter Implementation

- **FIR Filter difference equation**

$$y(n) = \sum_{k=0}^{M-1} b_k x(n-k)$$

- **FIR Filter Transfer function**

$$H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_{M-1} z^{-(M-1)}$$



Infinite Impulse Response (IIR) Digital Filters

- **Designed by considering an equivalent analog filter**
- **Get an expression in s-domain**
- **Convert analog filter to digital**
  - Bilinear Transformation
- **Get an expression in z-domain**
- **Implement z-domain expression**
- **Typically IIR filters use both poles and zeros**

Butterworth Filter

- **Pass Band**

  Maximum flat magnitude response in pass-band

- **Transition Region**

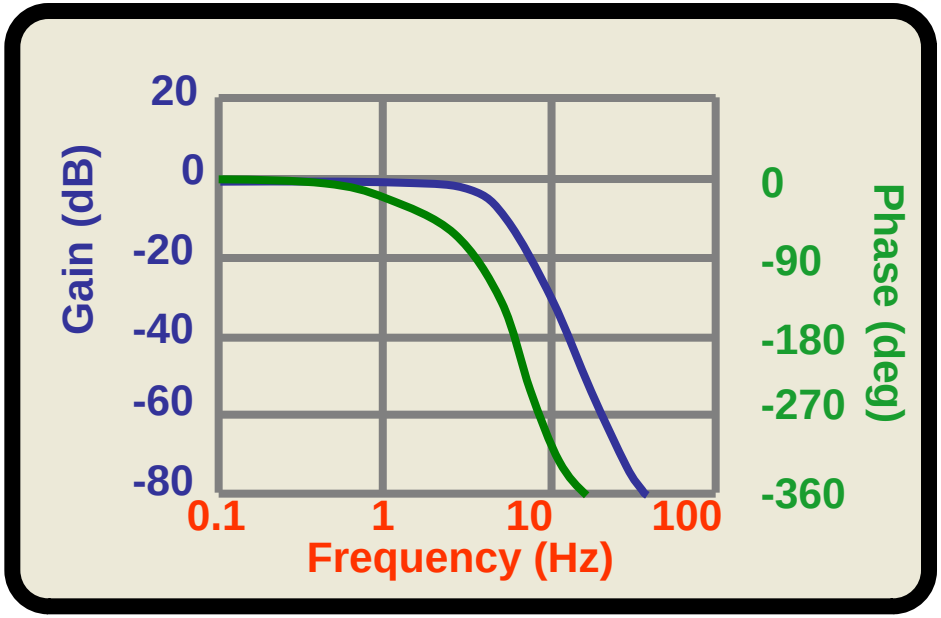
  Steeper than Bessel, not as good as Chebyshev filter

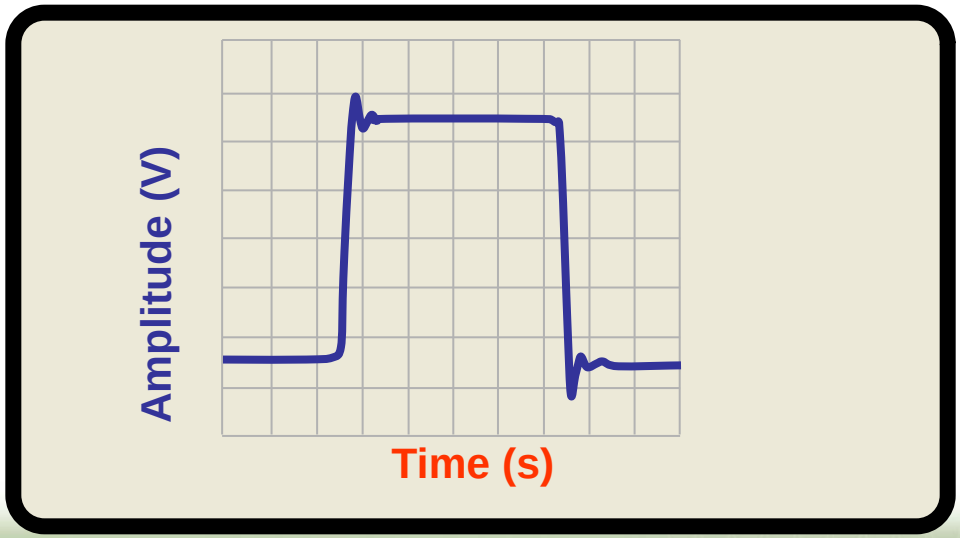
- **Stop Band**

  No ringing

- **Pulse Response**

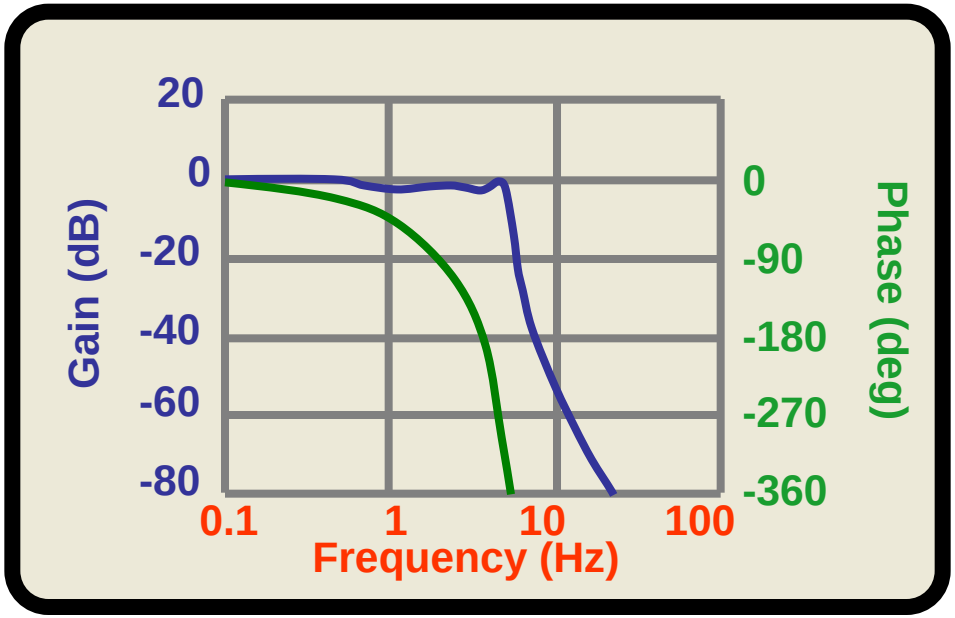
  Some overshoot and ringing, but less than the Chebyshev filter

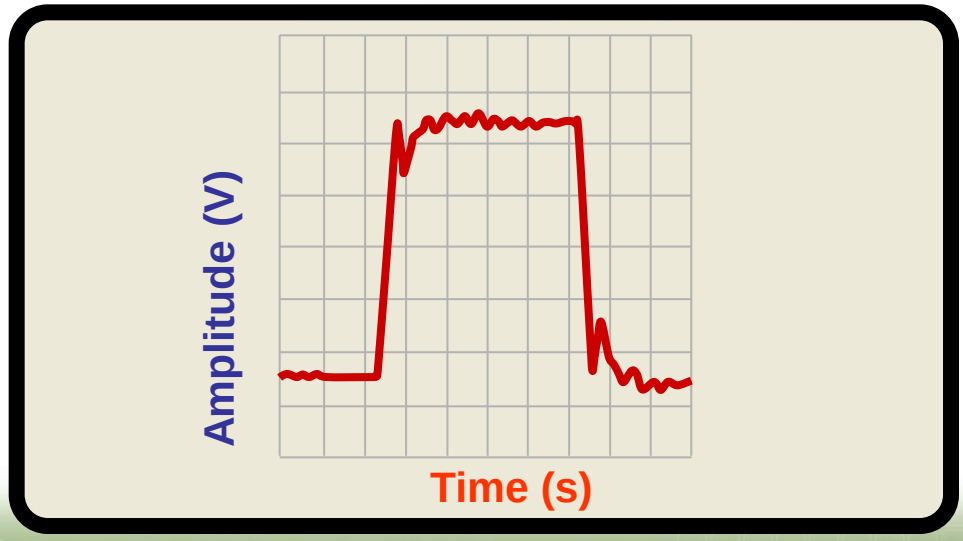




Chebyshev Filter

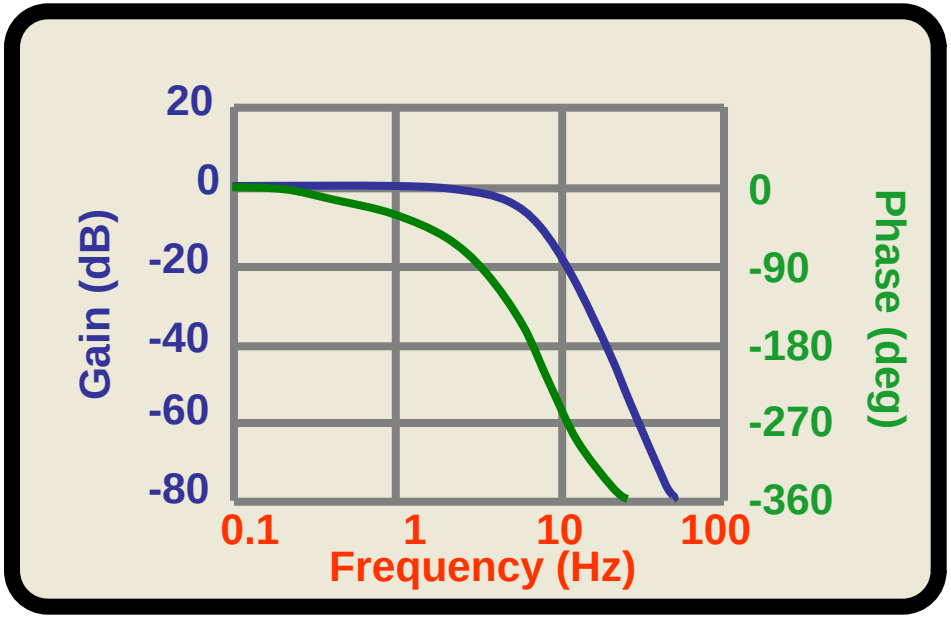
- **Pass Band**
  Ripple in the pass-band
- **Transition Region**
  Steeper than Butterworth and Bessel filters
- **Stop Band**
  No ringing
- **Pulse Response**
  High degree of overshoot and ringing

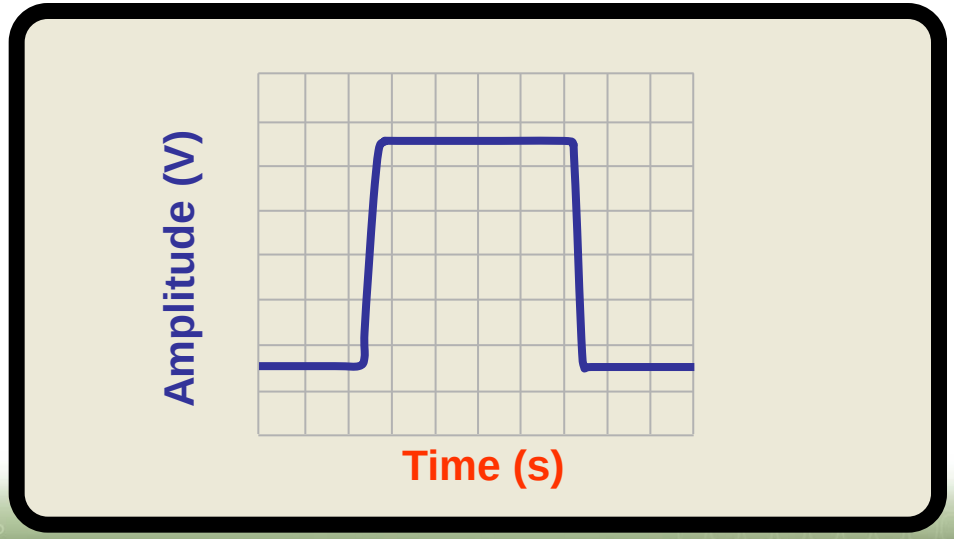




Bessel Filter

- **Pass Band**
  Flat magnitude response in pass-band
- **Transition Region**
  Slower than the Butterworth or Chebyshev filters
- **Stop Band**
  No ringing
- **Pulse Response**
  No overshoot or ringing





IIR Filters

- **Filter types**
  - Butterworth, Chebyshev, Elliptic, Bessel…
- **How to choose**
  - Depends on application requirements
  - Consider phase response
  - Consider frequency response
  - Consider time domain step response
- **Also review the pole locations**
  - Indicate transition band
  - Indicate stability

Implementation

- **IIR Filter difference equation**

$$y(n) = -\sum_{k=1}^{N} a_k \times y(n-k) + \sum_{k=0}^{M} b_k x(n-k)$$

- **IIR Filter Transfer function**

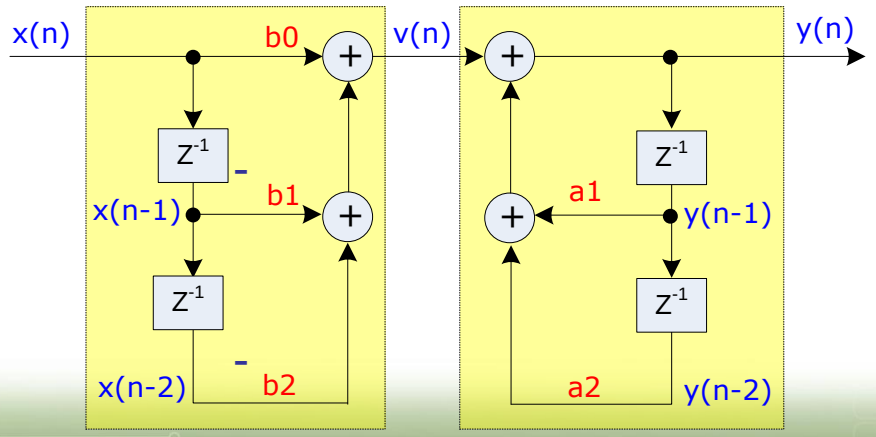
$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_M z^{-M}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_N z^{-N}} = \left(\frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}}\right) \bullet \left(\frac{b'_0 + b'_1 z^{-1} + b'_2 z^{-2}}{a'_0 + a'_1 z^{-1} + a'_2 z^{-2}}\right) \cdots$$

Biquad Section

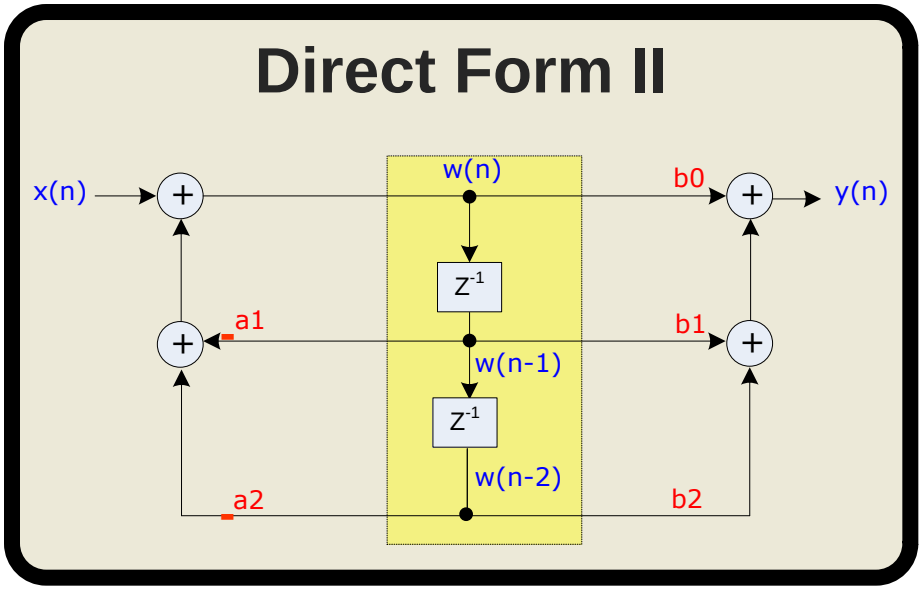
Direct Form I

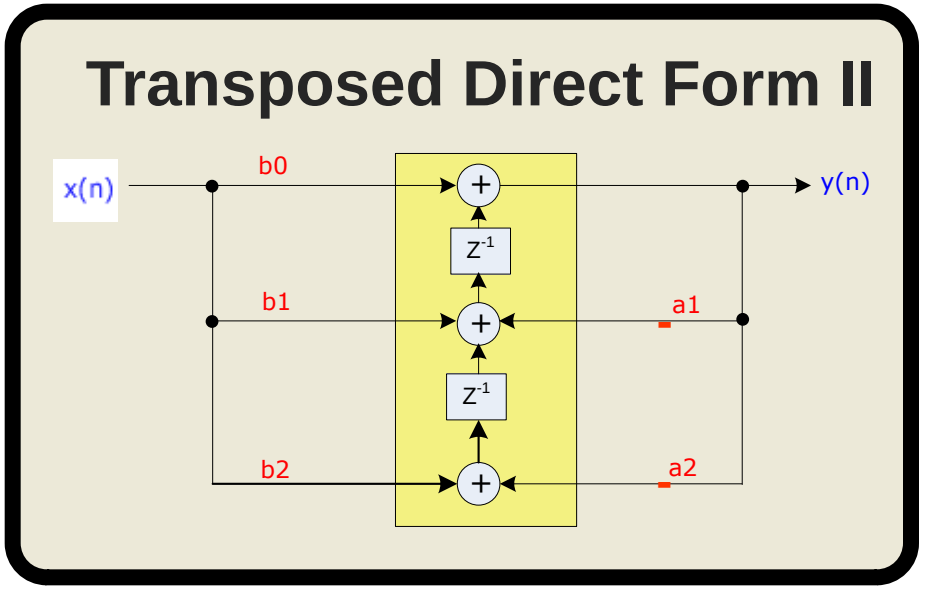
Denominator – Poles

Numerator - Zeroes



Implementation





- **Theoretically, Direct Form I and II and Transposed Direct Form II are not different**
- **From implementation perspective Direct Form II:**
  - Requires less memory space
  - Require input scaling

FIR vs. IIR filter

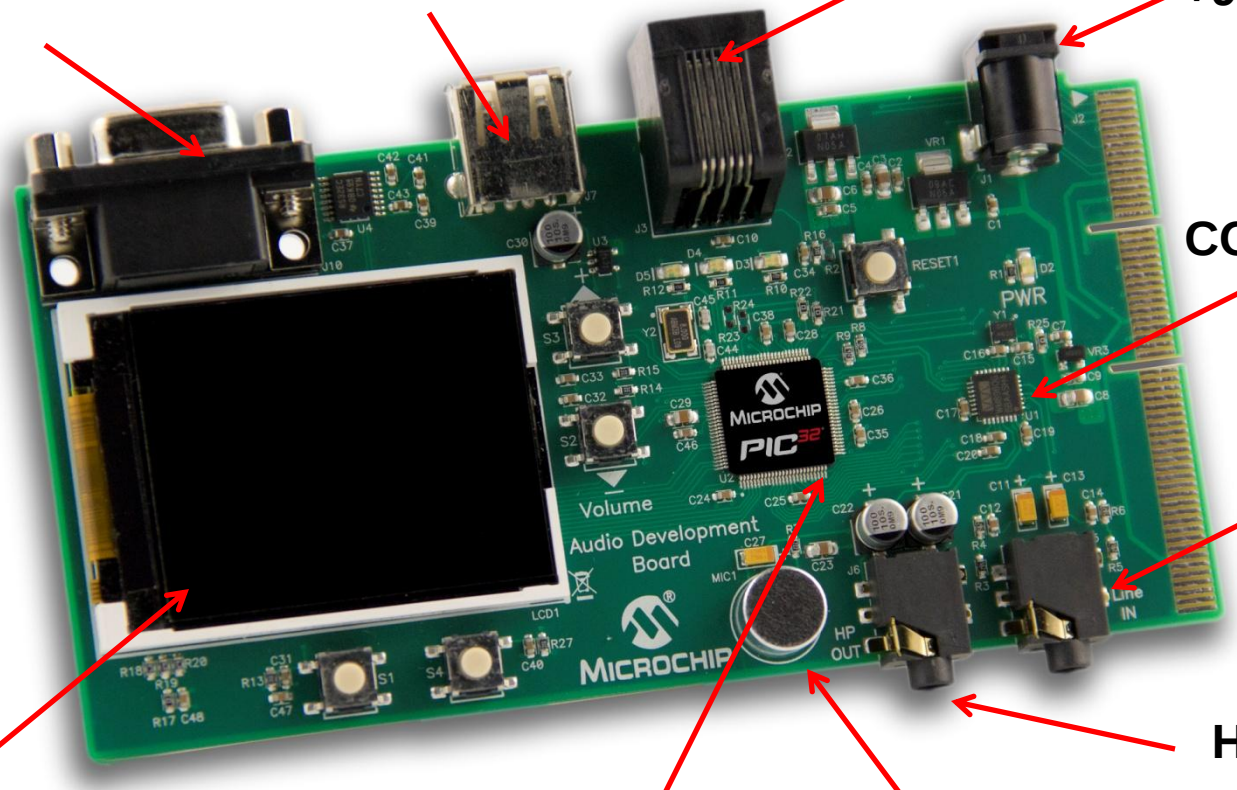
Finite Impulse Response Filter

- ☒ All zeros filter
- ☑ Linear phase easy to obtain
- ☑ FIR filters always stable
- ☒ Large number of filter taps may be required for sharp responses

Infinite Impulse Response Filter

- ☑ Typically uses poles and zeros
- ☒ Phase is difficult to control
- ☒ Possibility for instability
- ☑ Fewer numerical operations than FIR filters for sharp responses

Audio Development Board (ADB)



Advanced Digital Audio Development Board for PIC32®
(Part # DM320011)

Lab 1:
Design and use a digital filter

Lab 1 Objectives

- Design a Digital FIR Notch filter
- Invoke XC32 DSP LIB filter function
- Remove unwanted tone from a speech signal

Lab 1 Summary

- FIR filter equation stays the same across the filter type
- XC32 DSP LIB filter functions are available
- Notch filters can be used in simple noise reduction applications

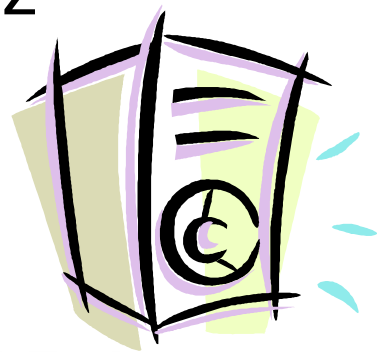
Section 1 - Summary

- Reviewed Digitization and Sampling
- Reviewed how a digital filter works
- Reviewed how a FIR Filter is designed
- Understood the effects of window choices
- Understood how a IIR filter is designed
- Developed and implemented a FIR notch filter in LAB 1

Speech Coding

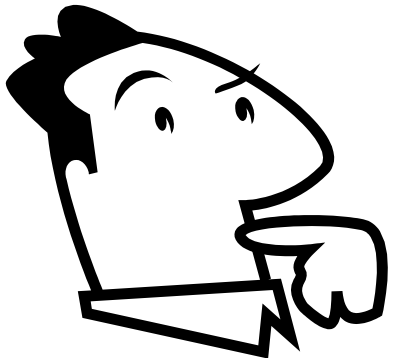
Audio Signals

- **Section of frequency spectrum that is audible: 20Hz – 20KHz**
- **Speech signals 300 to 6000Hz**
- **Music signals have a wider range**
- **Sampling Rates**
  - 8, 16, 32, 44.1, 48, 96 and 192KHz
- **Classification (for our class)**
  - Speech Signals
  - Audio Signals



Speech coding

- What is Speech?
- What is speech coding?
- Why would you want to encode (decode) speech?
  - Compress it. Reduce size
  - Reduce memory
  - Reduce bandwidth
  - Analyze/Synthesize
  - Security (Encryption)



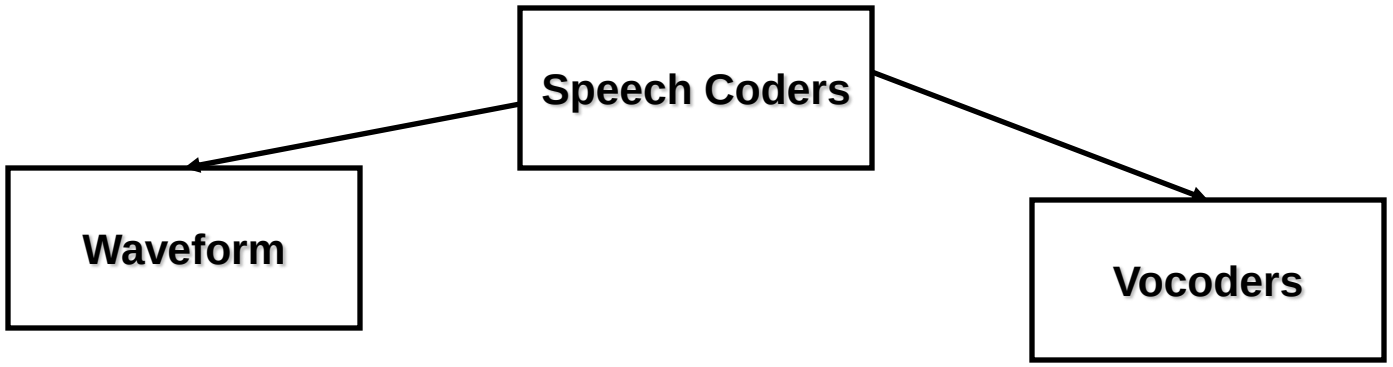
Speech Signal Specs.

- **Typically sampled at 8KHz**
  - Wideband application use 16KHz
- **Bit resolution**
  - 12 to 16 bits typical
- **ADC/DAC (CODECS)**
  - Voice ADC/DAC
  - Audio ADC/DAC
- **Analysis frame size 10-20msec**
- **Signals exhibit strong correlation**

What is correlation

- Correlation measures similarity
- Can measure presence of information
- Autocorrelation indicates presence of pattern
- Speech signals exhibit strong correlation
- Correlation can be exploited for speech coding

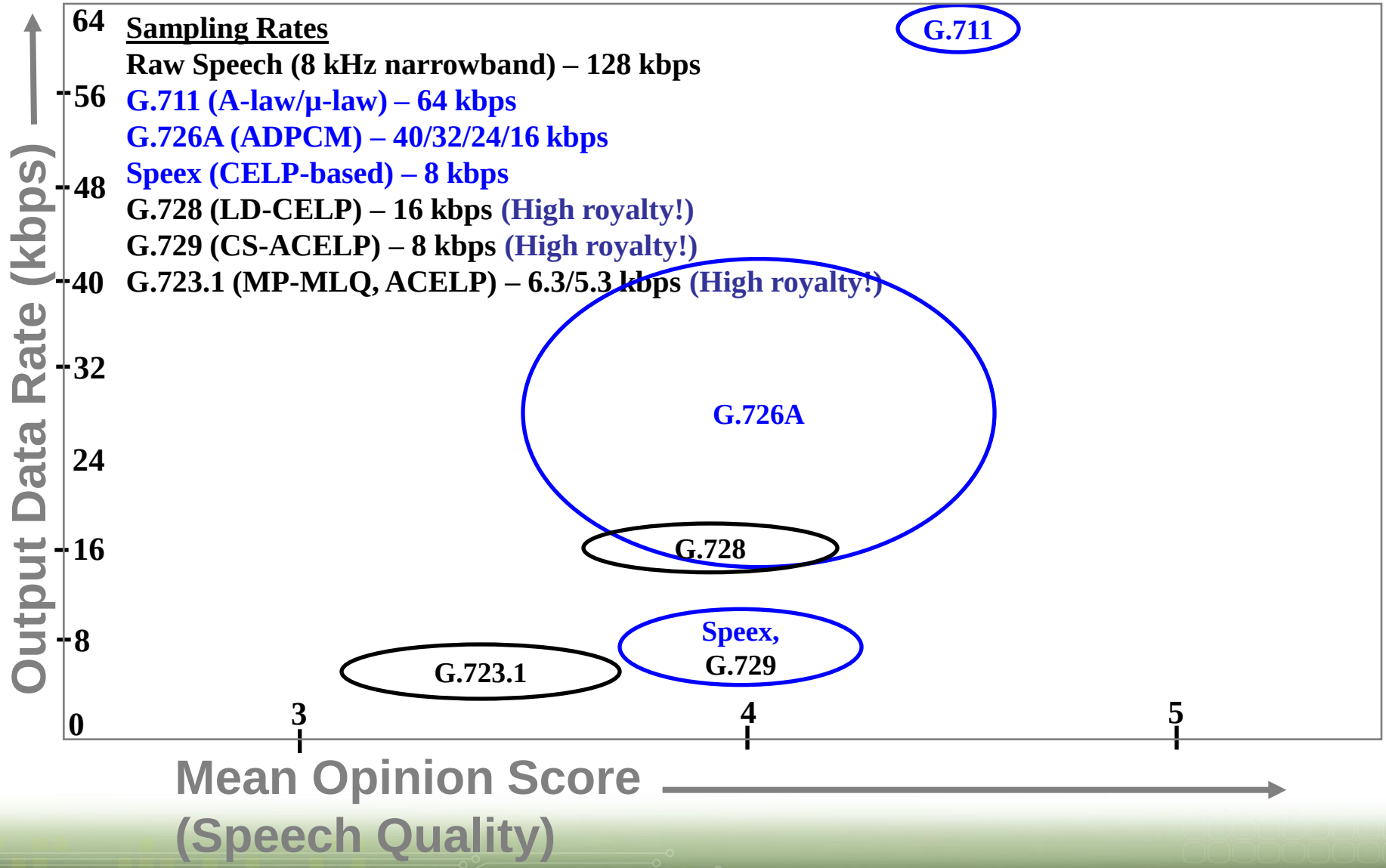
Types of Speech Coders



- Encode the wave shape
- Reconstruct the time domain waveform
- Operate at medium compression rates
- Moderate computational complexity
- DPCM, ADPCM, a/u law
- G.711, G.726A (ADPCM)

- Parametric coding
- Model the human speech system
- Operate at medium to high compression rates
- High computational complexity
- CELP, ACELP
- Speex

Speech Coding - Comparison



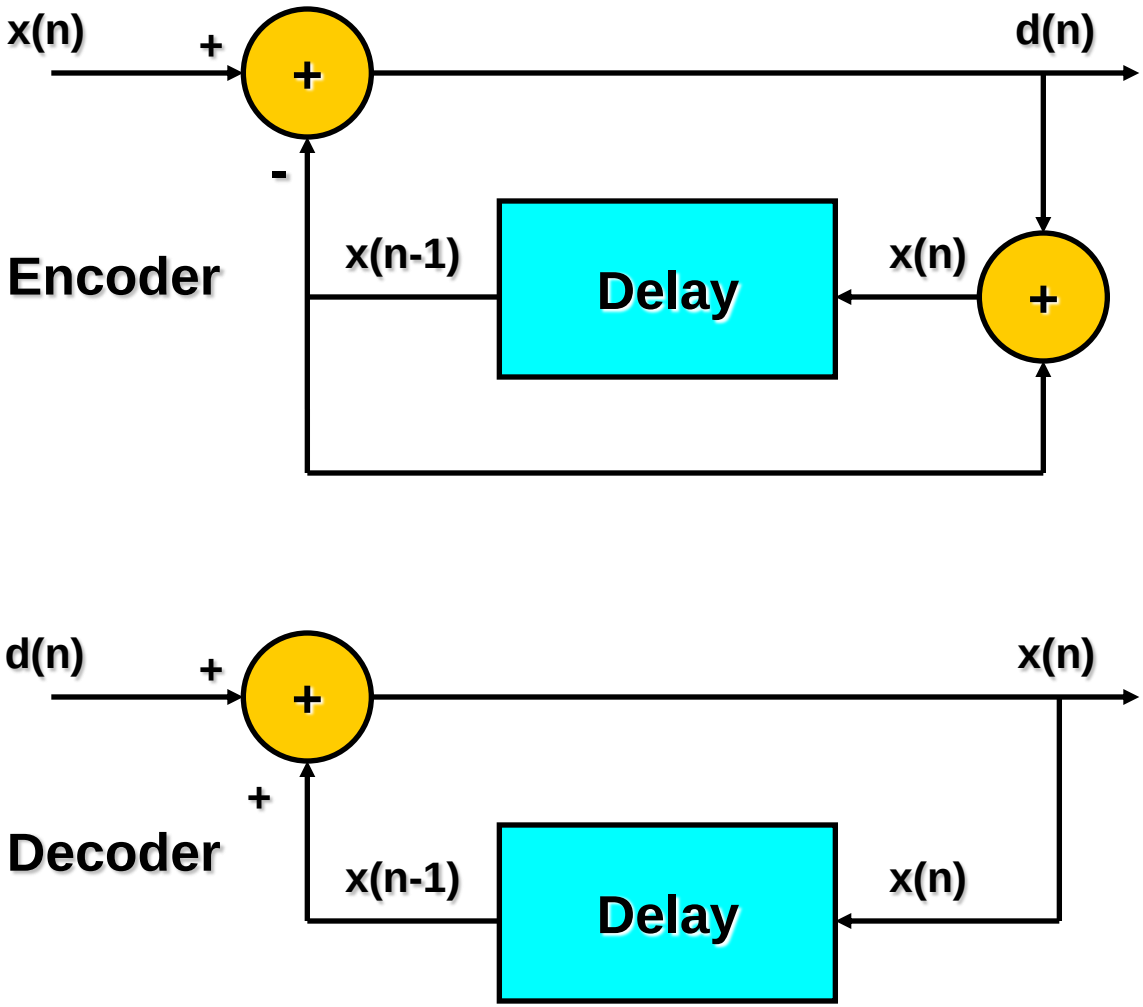
G.711

- **ITU-T recommendation**
- **8KHz sampling rate**
- **2:1 compression (128Kbps to 64Kbps)**
- **How does it work?**
  - Signal is broken up into amplitude intervals
  - Intervals are log spaced
  - The interval number, some amplitude bits and sign is coded
  - u-law and a-law

G.726A

- **ITU-T recommendation**
- **8KHz sampling rate**
- **Four bit rates (16, 24, 32 and 40kbps)**
- **How does it work**
  - Difference between adjacent samples is less
  - Encode and quantize difference
  - Use adaptive techniques to maximize SNR and reduce variance

G.726A

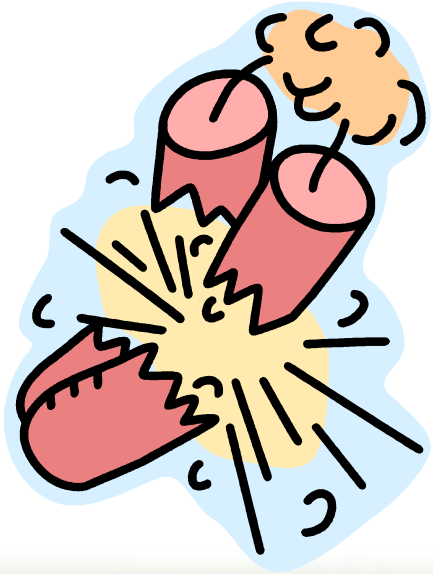


Parametric Coding

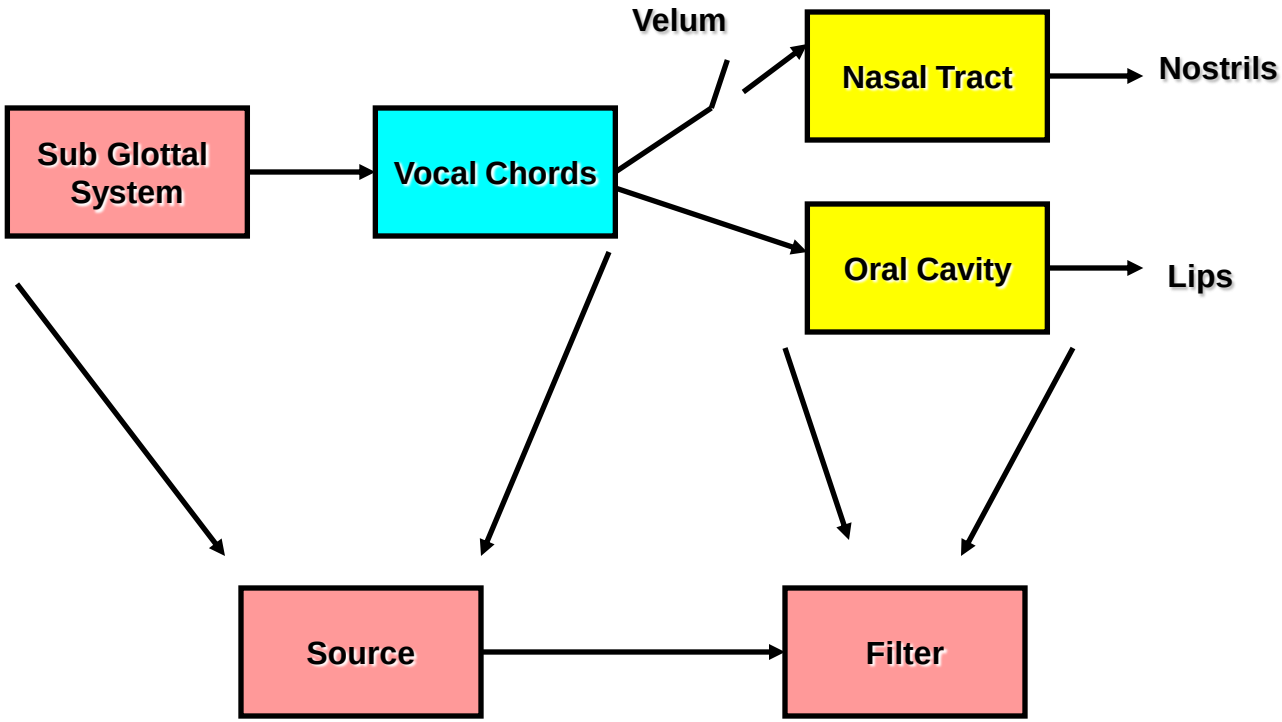
- Uses the source filter voice model
- Extract vocal tract feature parameters within a frame
- Frame size is 10-20 msec
- More complicated
- Better compression
- Used widely in today's communication devices

Speech Signals

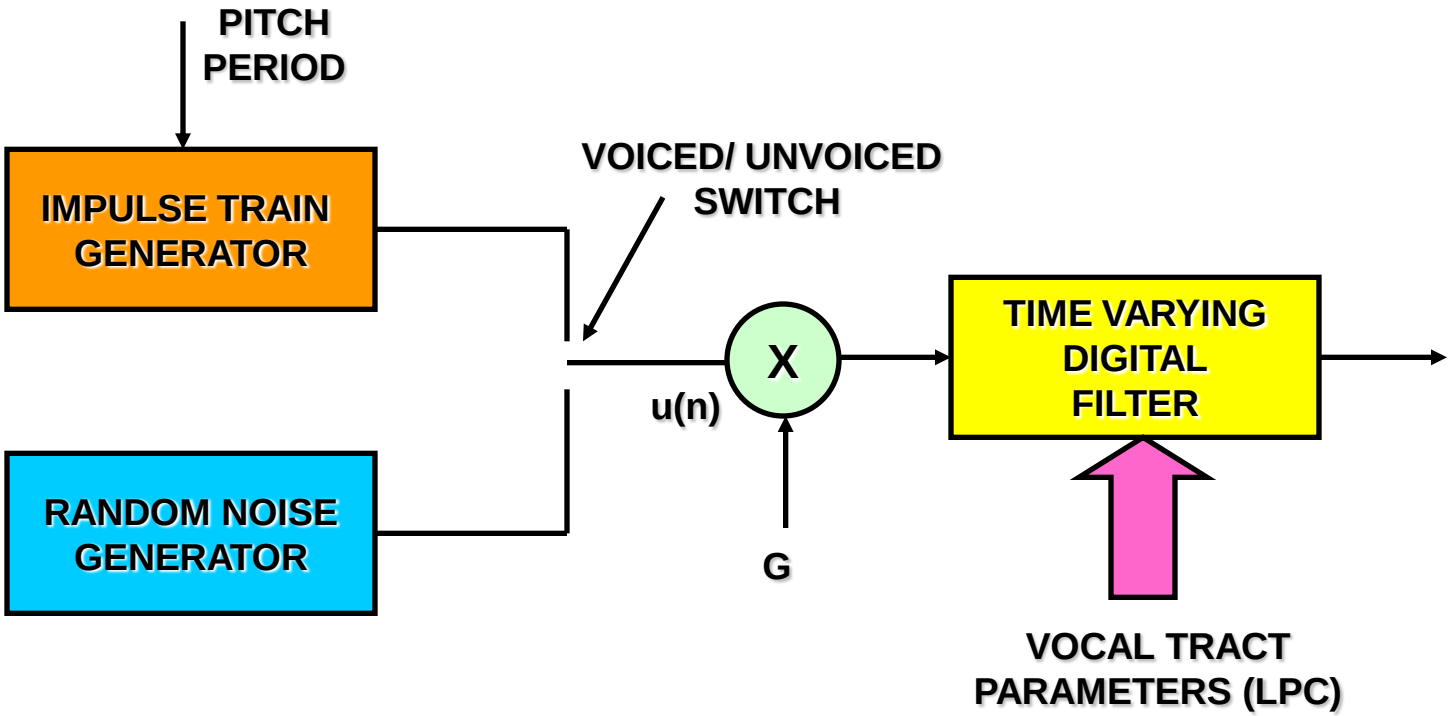
- **Classification of speech sounds**
  - Voiced
  - Unvoiced
  - Plosives



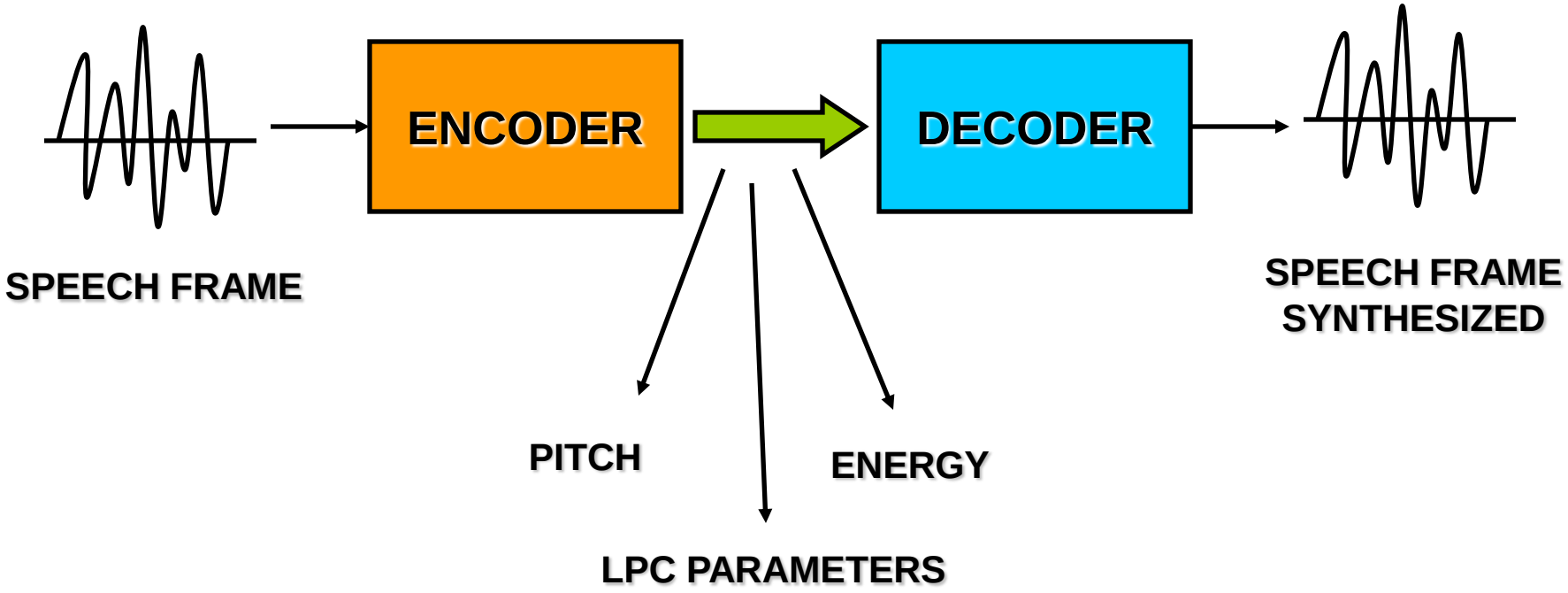
Speech Modeling



Source System Model

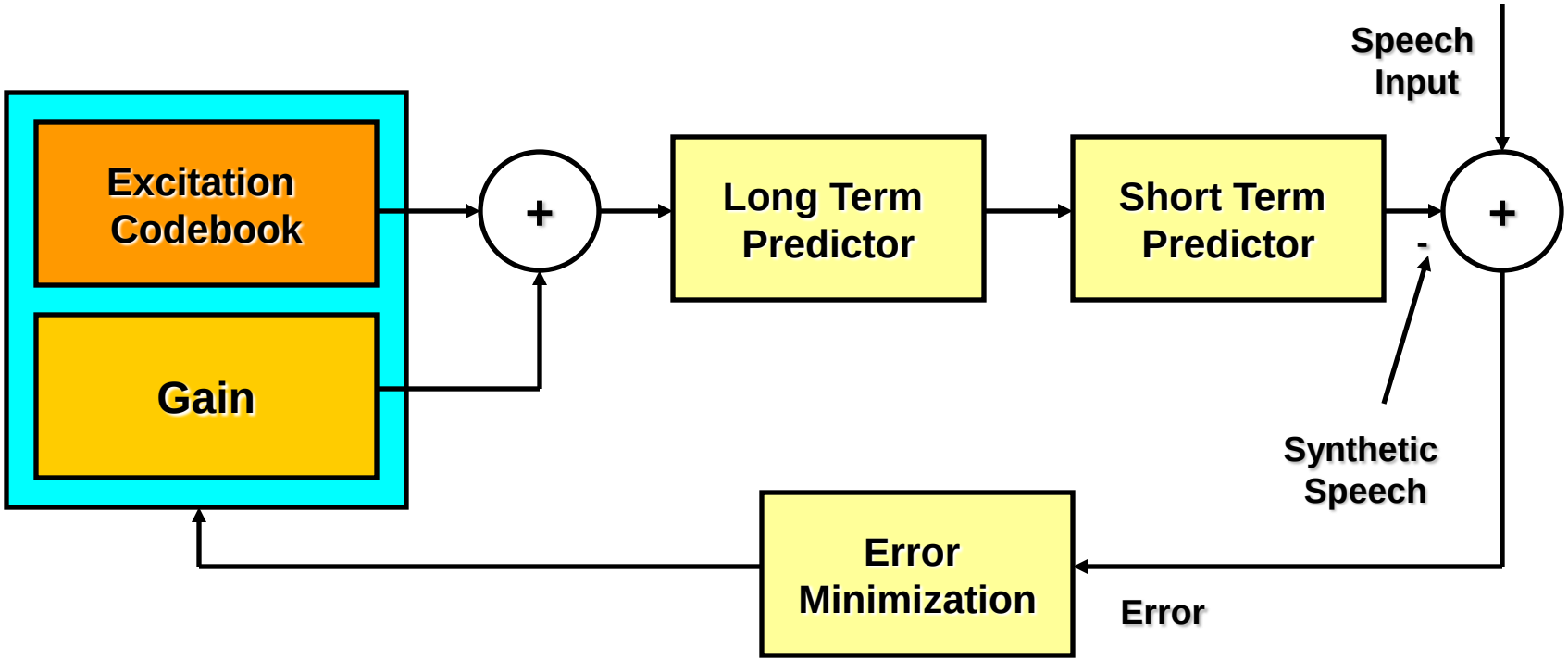


Vocoders



- Encoder sends the parameters
- Decoder reconstructs using the parameters

Code Excited Linear Prediction (CELP)



Speex Algorithm

- 8Khz, 16Khz and 32Khz sampling modes
- Fixed point port available
- Bit rates from 2 to 44kbps
- Open source (www.speex.org)
- Uses CELP
- Ports available for dsPIC33F/33E and PIC32
  - dsPIC® DSC port is assembly optimized
  - PIC32 port uses public API

How to use Speex Encoder

- **Speex Encoder**
  1. Initialize the encoder
  2. Set encoder quality level
  3. Initialize the bits structure
  4. Reset the bits structure
  5. Encode the input
  6. Do steps 4 and 5 as needed
  7. De-allocate bits and encoder once done

Speex Encoder API

- **Initialize the encoder**

```
#include<speex/speex.h>
SpeexBits bits;
void *encoder;
Speex_bits_init(&bits);
encoder = speex_encoder_init(&speex_nb_mode);
```

- **Set the encoder quality level**

```
speex_encoder_ctl(enc_state,
            SPEEX_SET_QUALITY,&quality);
```

Speex Encoder API

- Reset the bits structure

```
speex_bits_reset(&bits);
```

- Encode the input

```
speex_encode_int(encoder,input,&bits)
nbBytes = speex_bits_write(&bits,output,
MAX_NB_BYTES);
```

- Destroy allocated memory when done

```
speex_bits_destroy(&bits) //if bits obtained dynamically
speex_encoder_destroy(enc_state);
```

How to use Speex Decoder

- **Speex Decoder**
  1. Initialize the decoder and the bits structure
  2. Load the bits structure with input frame
  3. Decode the input frame
  4. Perform steps 3 and 4 as needed.
  5. De-allocate bits and decoder once done

Speex Decoder API

- **Initialize the decoder and bits structure**

```
#include<speex/speex.h>
SpeexBits bits;
void *decoder;
speex_bits_init(&bits);
decoder = speex_decoder_init(&speex_nb_mode);
```

- **Load bits structure with input frame**

```
speex_bits_read_from(&bits, cbits,nbBytes);
```

Speex Decoder API

- Decode the input frame

```
speex_decode(decoder, &bits, output)
```

- Destroy decoder and bits structure

```
speex_bits_destroy(&bits) //if bits obtained dynamically
speex_decoder_destroy(decoder);
```

Lab 2: Build a Voice Recorder Application

Lab 2 Objectives

- Use speex vocoder in a voice recorder application
- Use speex encoder to encode speech
- Store encoded file on thumb drive
- Use speex decoder to decode speech and playback

Lab 2 Summary

- Speex Library can reduce memory requirements
- Processing requirements must be considered
- Encoder and decoder can be supported in one application

Section 2 Summary

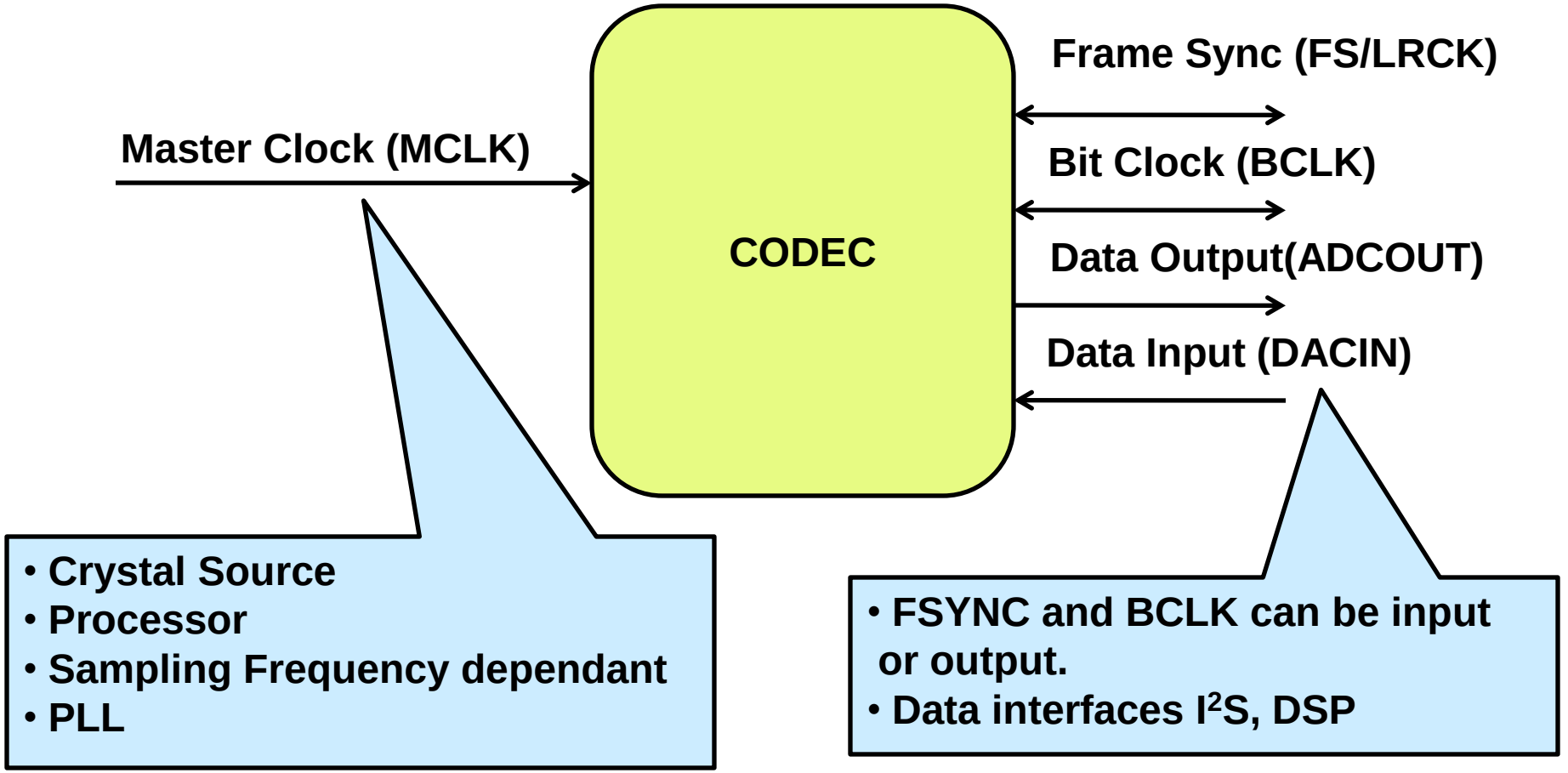
- Reviewed basics of speech signals and Speech coding
- Reviewed waveform and parametric coders
- Understood speech modeling
- Reviewed G.711, G.726A and Speex coders
- Used the Speex codec in LAB 2

External Codec Interfacing with PIC32

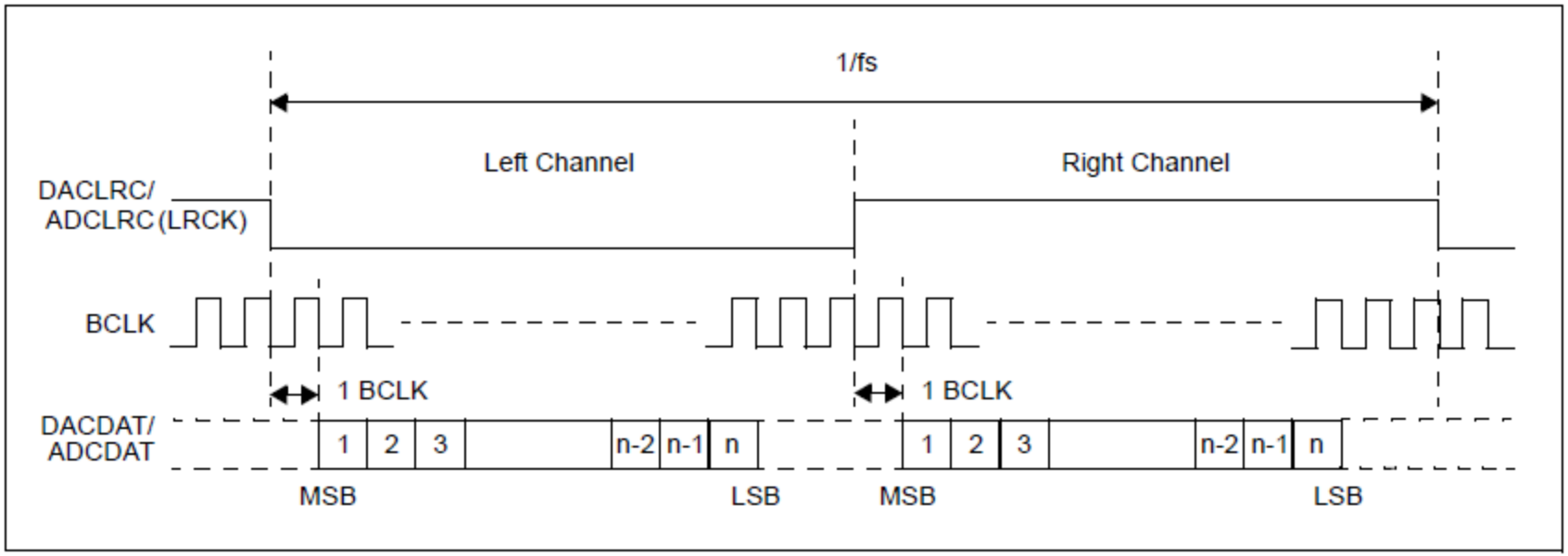
External Codec

- **ADC only, DAC only or combination**
- **Typical codecs also contain**
  - PLL for generating clock (expensive)
  - Headphone Amplifiers
  - Equalizer and ALC
  - Multiple input and output options
  - Support for multiple data formats

Interface Essentials

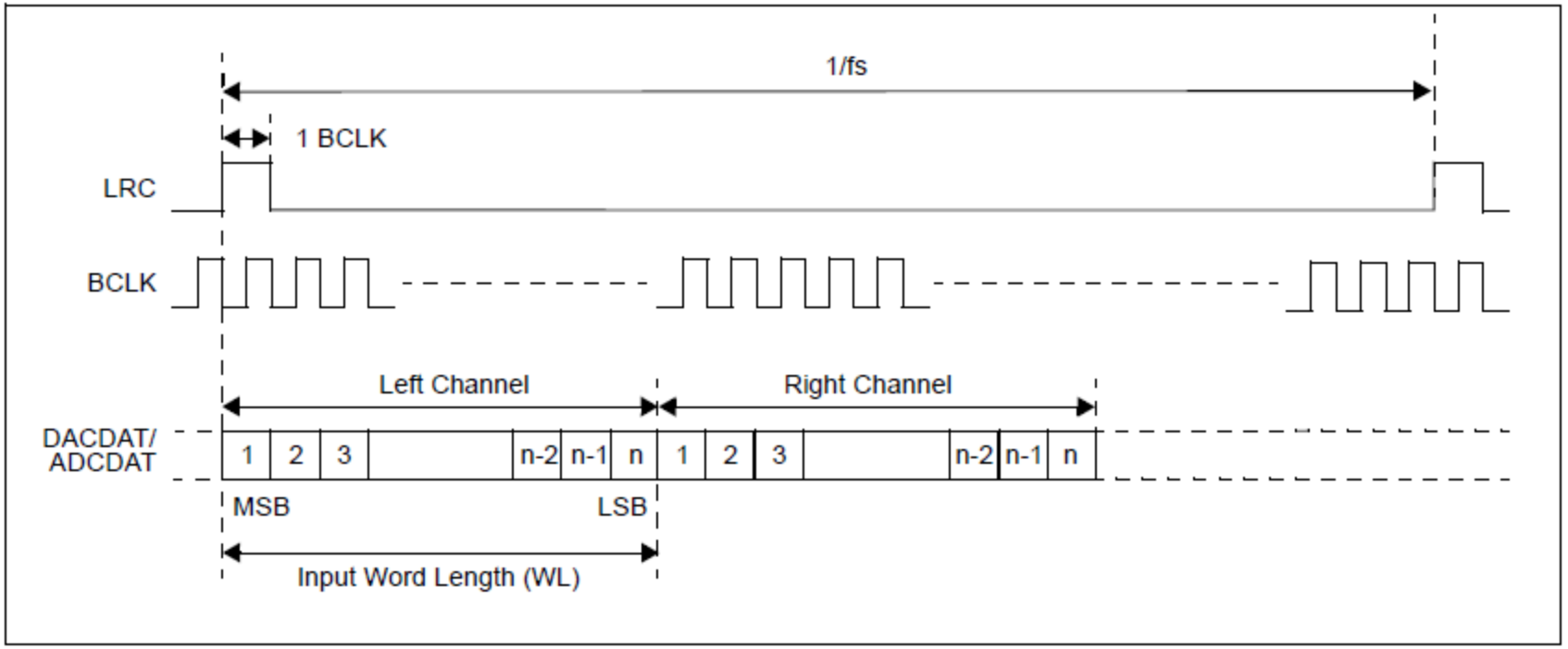


I²S Audio Interface



- Common in stereo applications
- Left Justified format available
- Standard does not dictate clock rates
- There may be unused BCLK cycles

DSP/PCM Interface



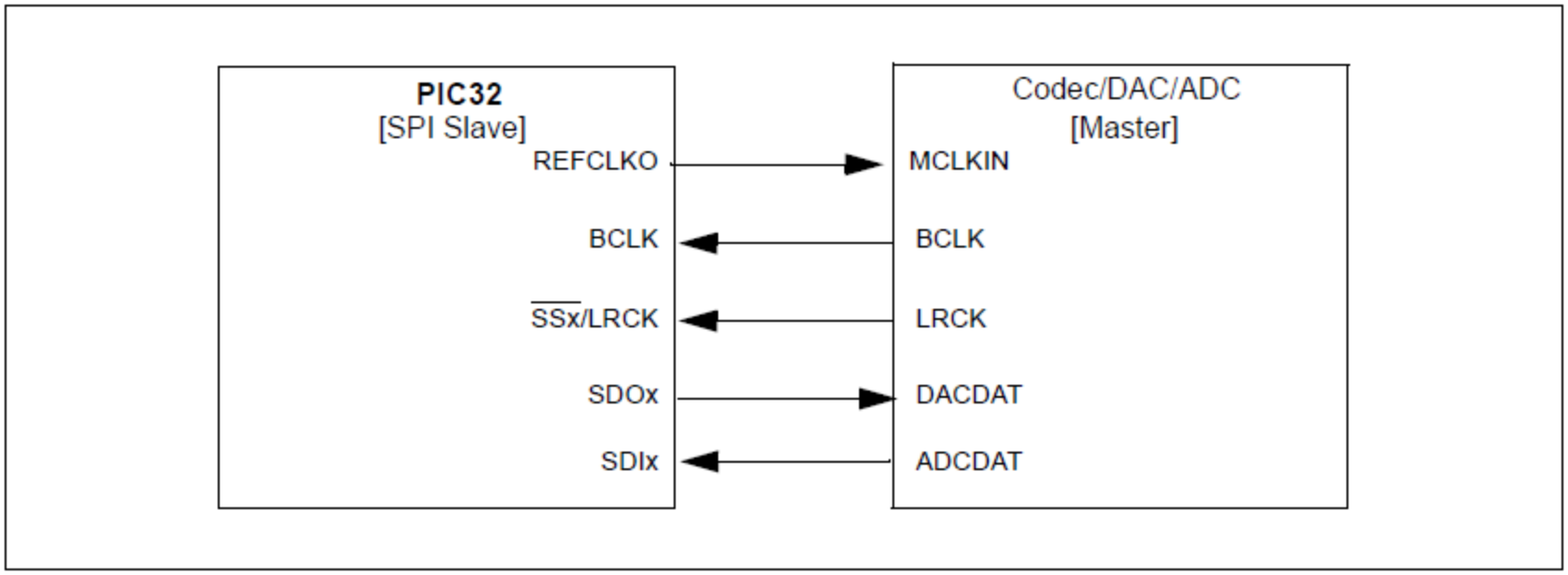
- **Common in multichannel audio**

Notes on Digital Audio Interfacing

- **Master generates BCLK and FSYNC**
- **Codecs with PLL can be Masters**
  - Usually expensive
- **MCLK is typically a multiple of sampling frequency**
  - Used to determine sampling frequency of non PLL codecs
- **Codec Interface solutions**
  - PIC32 SPI Module (supports I^2S, DSP/PCM)
  - dsPIC® DSC DCI Module (supports I^2S, DSP/PCM)

PIC32 REFCLKO

- **Use PIC32 SPI module to interface to codec**
- **PIC32 can generate clock (REFCLKO output)**
  - Can be used for MCLK
  - Don't have to use codec with PLL
  - Clock speed can be tuned for USB Audio Applications



Audio Coding

Audio Coding

- **Non Compression Type**
  - PCM, WAV, AIFF, AU, RAW..
- **Lossless Codecs**
  - FLAC, ALAC, Dolby TrueHD…
- **Lossy Codecs**
  - MP3, WMA, AAC, Vorbis

MPEG LAYER 3

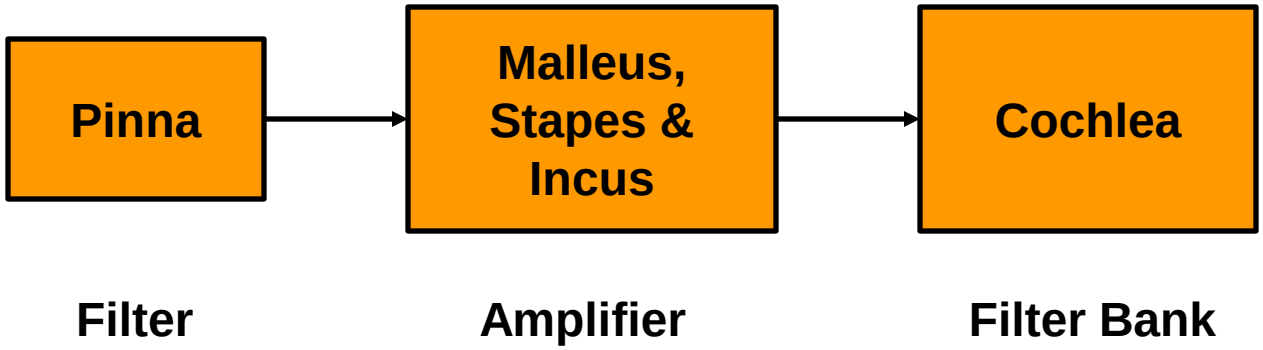
- **Layer 3 processing for MPEG-1, 2 and 2.5**
  - Layer 3 is the audio layer
  - More popularly known as MP3
- **Uses Perceptual Coding**
- **Uses psychoacoustic principles**
- **Patented by Fraunhofer Society and Thomson**
- **Supports both CBR and VBR**
- **Supports multiple bit rates and sample rates**
- **Open source fixed point ports available (Helix)**

Audio Coding Principles

- Frame Analysis
- Sub-band Processing
- Psychoacoustic processing
- Bit Allocation
- Huffman coding

Sub-band Processing

- How does the human ear work



- Filter bank organized as a set of critical bands
- Frequency to place transform in cochlea.
- This filter bank is modeled as a basis for analysis and synthesis

Psychoacoustic Processing

- **Loudness curves**
- **Auditory masking**
- **Temporal masking**
- **Other Phenomena**
  - Cochlea echoes
  - Phase Locking
  - Temporal Integration
  - Post stimulation auditory fatigue
  - Auditory adaptation
  - Haas Effect
  - Binaural Masking

Loudness curves

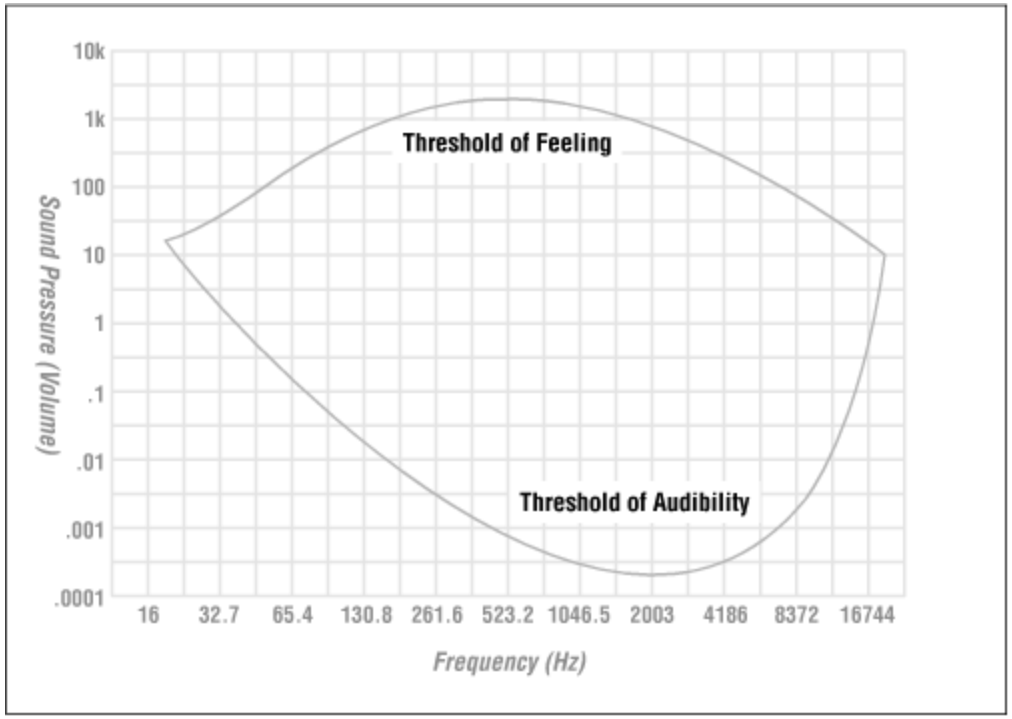
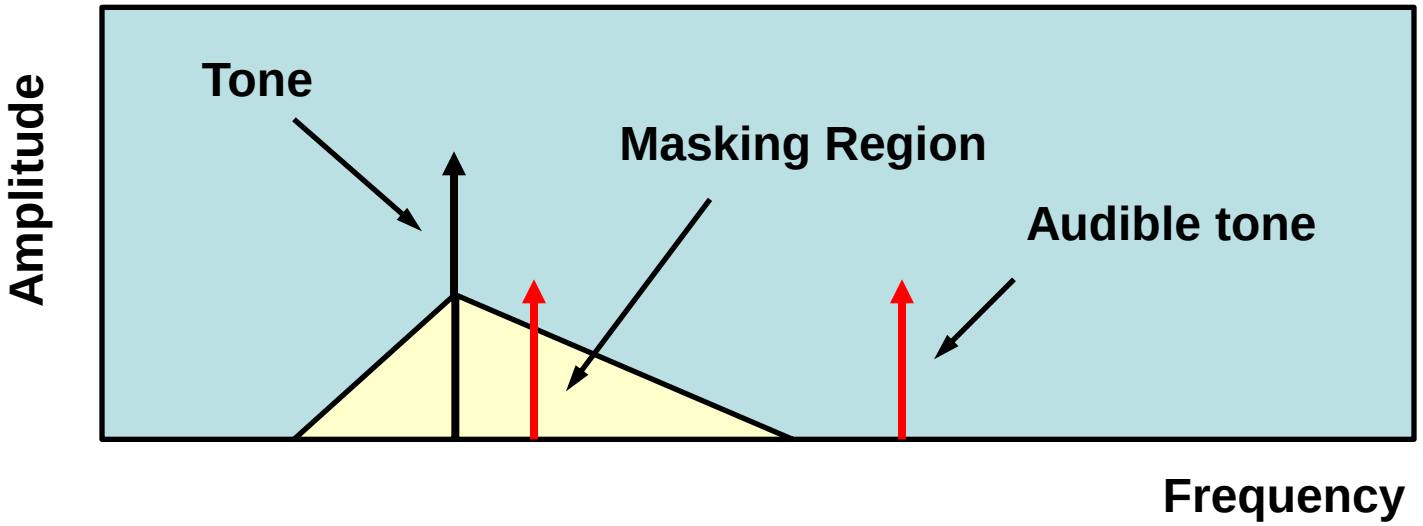


Image obtained from www.mp3-converter.com

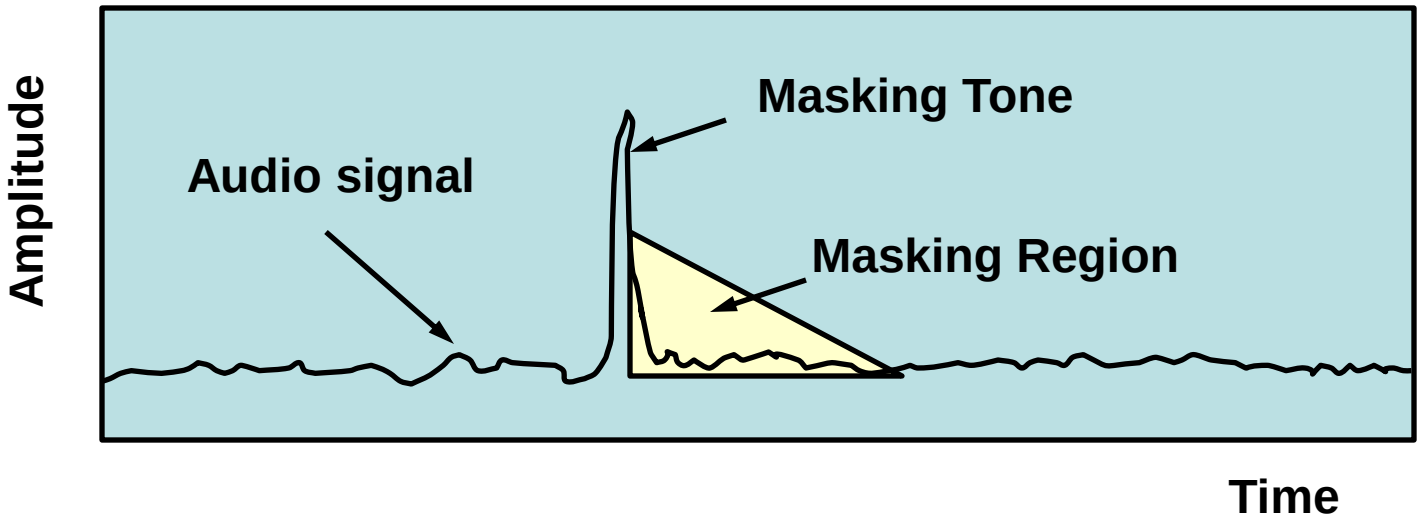
- **Loudness perception depends on frequency**
- **May vary between individuals**
- **Allocate less bits to what may not be heard**

Auditory Masking



- Sensitivity to similar tones within the critical band is reduced
- Tone of same amplitude outside critical band is audible

Temporal Masking



- Tones before or after masking tones are not audible

Post Psychoacoustic Processing

- **Bit Allocation**
  - Assign more bits to important information
  - Also manage overall bit rate
  - Bit reservoir
- **Huffman Coding**
  - Lossless Coding Technique
  - Common pattern coded using lesser bits
  - Typically implemented using a look up table

Helix MP3 Algorithm

- Pure 32 bit fixed point implementation
- Implemented in C
- Provides layer 3 support for MPEG -1, 2 and 2.5
- Supports CBR and VBR
- Support mono and stereo modes
- Fully re-entrant and statically linkable

How to use Helix MP3 Library

1. Initialize the decoder
2. Read a frame
3. Find MP3 Sync Word (the first time)
4. Get next frame info (optional)
5. Decode the MP3 frame
6. Get last frame info (optional)
7. Do 2 – 6 till end of file
8. Free the decoder (if needed)

Helix MP3 Lib API

- Initialize the decoder

```
#include "coder.h"
#include "mp3dec.h"

HMP3Decoder mp3Decoder;
mp3Decoder = MP3InitDecoder();

// If mp3Decoder equals NULL, then not enough memory
```

- Find MP3 Sync Word

```
offset = MP3FindSyncWord(input,MAX_FRAME_SIZE);

// If offset equals 0, the input frame does not
// contain the sync word
```

Helix MP3 Lib API

- Get next frame info

```
err = MP3GetNextFrameInfo(mp3Decoder, &frameInfo, input);

// If error equals MP3_INVALID_FRAME_HEADER, then
// get the next frame and try.
```

- Decode MP3 Frame

```
err = MP3Decode(mp3Decoder, &input, &bytesLeft, output,0);

// Check err value for any of the possible values.
```

Helix MP3 Lib API

- Get last frame Information

```
MP3GetLastFrameInfo(mp3Decoder, &mp3FrameInfo);

// mp3FrameInfo.outputSamps contains the number of
// output samples
```

- Free decoder

```
MP3FreeDecoder(mp3Decoder);
```

Helix MP3 Lib API

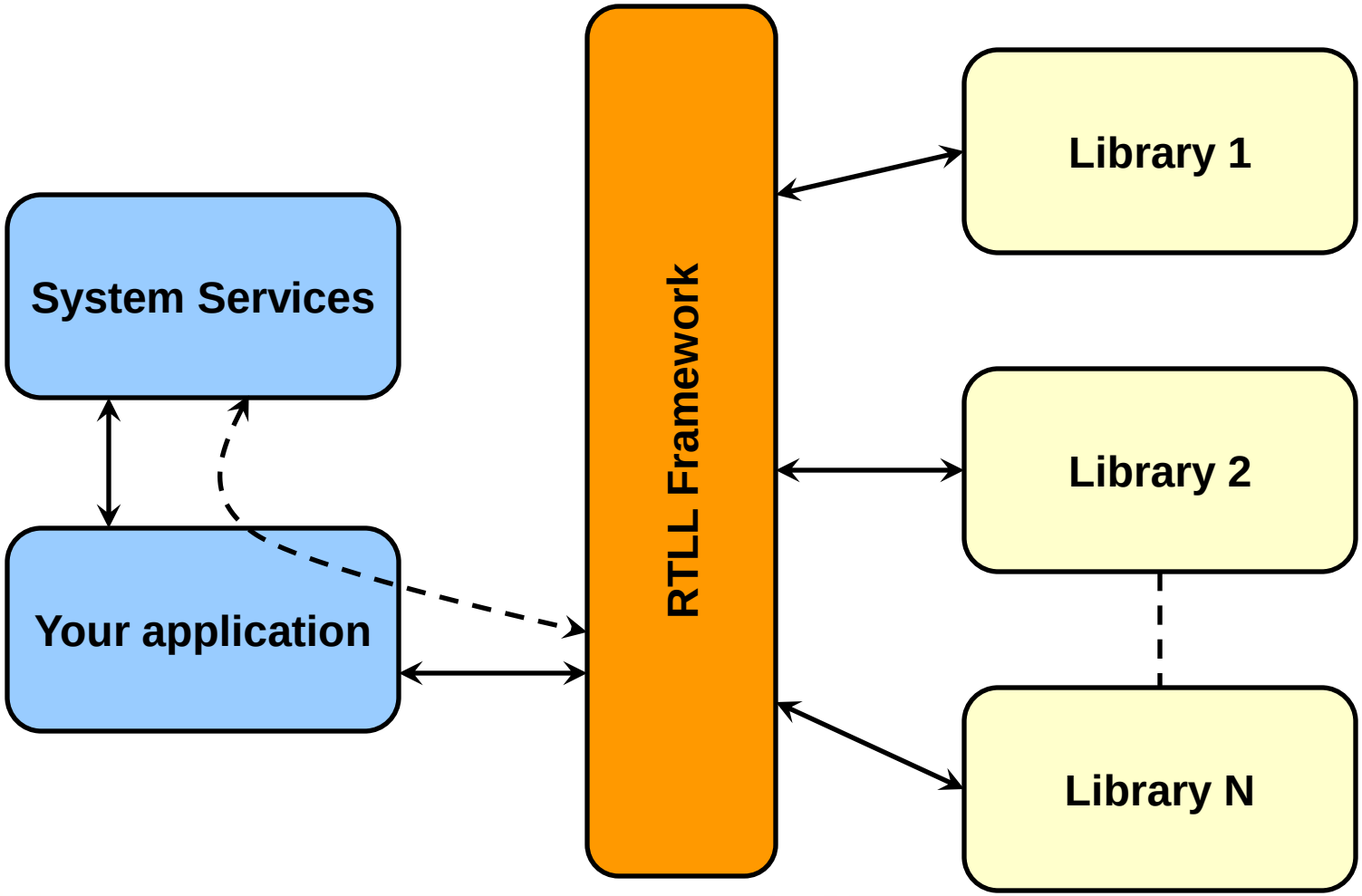
- About MP3 Frame Information

```
//MP3FrameInfo mp3FrameInfo;

// mp3FrameInfo.outputSamps
// mp3FrameInfo.bitrate
// mp3FrameInfo.samprate
// mp3FrameInfo.bitsPerSample
// mp3FrameInfo.nChans
```

Run Time Library Loading

- **Why RTLL**
  - Open source library covered by open source license
  - Static Linking, your code may get covered by the open source license
  - Open source code and your code compiled separately
  - Need a way to link dynamically
  - Multiple applications can use one instance of the library

RTLL Technique



RTLL API

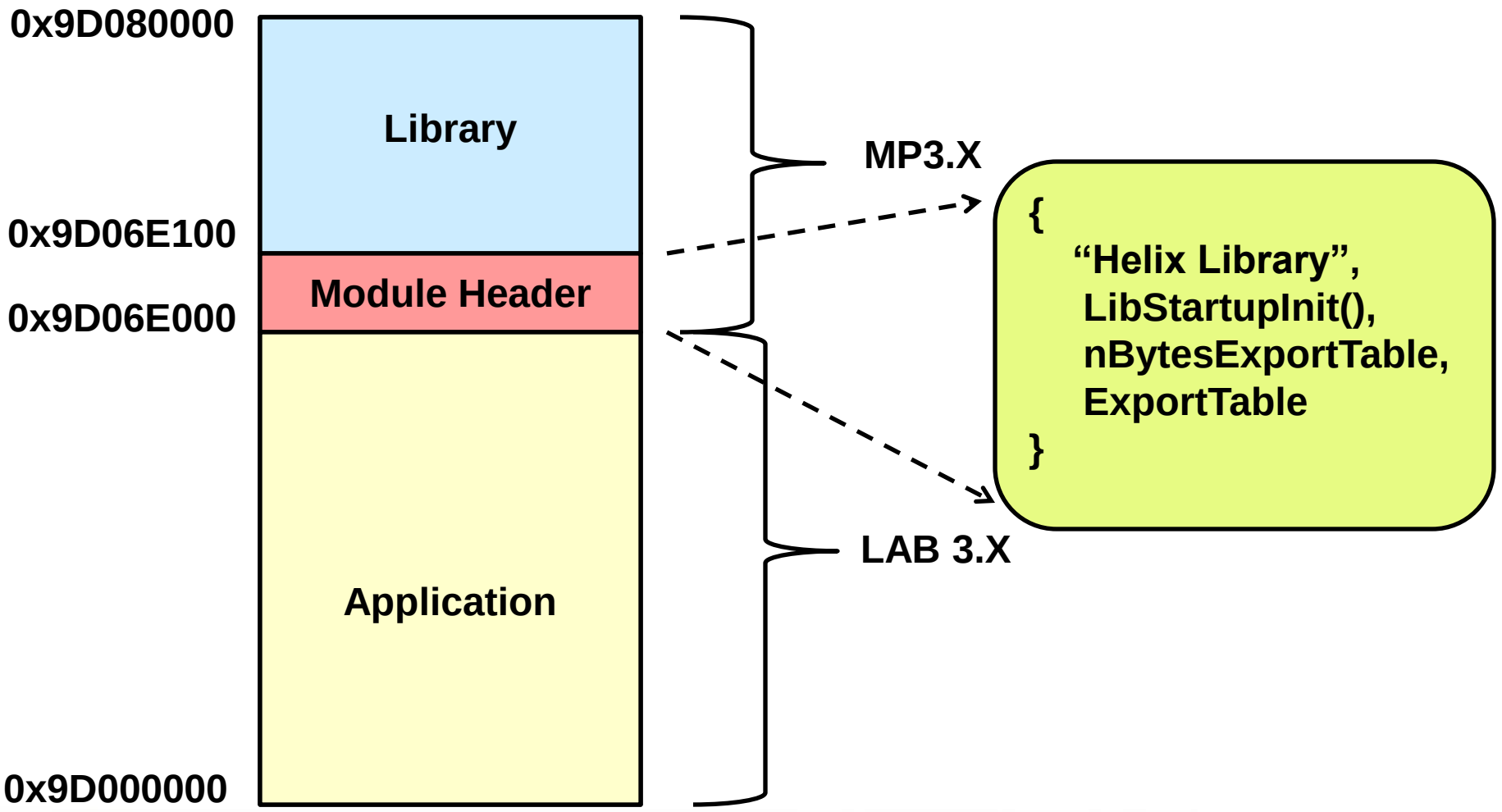
- Get Handle to Library

```
void * dlopen(char * libString, void * libLoadPoint);

// libString – string description of library
// libLoadPoint – address where the library starts
```

- Get handle to library functions

```
void * dlsym(void * libHandle, char * funcName);

// libHandle – handle to the library
// funcName – string description of the function.
```

RTLL – Helix MP3



RTLL - Export Table

- **`dlopen()` function**
  - Checks if library exists
  - Call the library startup code
- **`dlsym()` function**
  - Returns function pointer based on string key

```
const T_PROC_DCPT exportFunctionsTbl[] =
{
  {"HelixMP3InitDecoder",      (void *)MP3InitDecoder},
  {"HelixMP3FreeDecoder",      (void *)MP3FreeDecoder},
  {"HelixMP3Decode",           (void *)MP3Decode},
  {"HelixMP3GetLastFrameInfo",(void *)MP3GetLastFrameInfo },
  {"HelixMP3GetNextFrameInfo",(void *)MP3GetNextFrameInfo },
  {"HelixMP3FindSyncWord",     (void *)MP3FindSyncWord},
  {"HelixMP3RegMalloc",        (void *)RegisterAppMalloc},
  {"HelixMP3RegFree",          (void *)RegisterAppFree}
};
```

About RTLL

- **Export table published by library**
  - Contains string description
  - Contains pointer to exported function
  - Resides in the RTLL framework
- **Library can export system service register functions.**
- **Use –g0 option while compiling library code.**
- **Use –no-gc-sections option while linking library code**
- **Application should know**
  - Function description string
  - Function prototype
  - Library description string
  - Library Load Point

Disables global Pointer usage on PIC32 else this will have to saved and restored every time

Disables garbage collection. All function are included when library is compiled.

Lab 3:
MP3 Player

Lab 3 Objectives

- Use the Helix MP3 Algorithm API
- Build an MP3 player
- Experiment with MP3 files
- Study RTLL

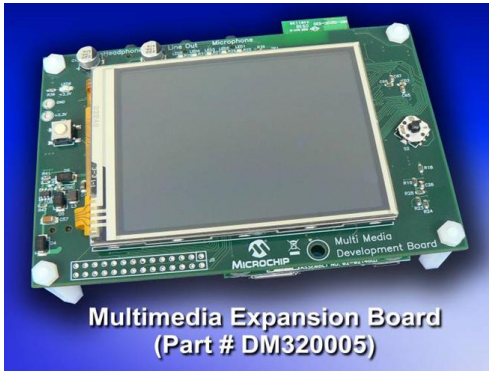
Lab 3 Summary

- We used RTLL to invoke Helix MP3 algorithm
- Multiple bit rates are supported
- Helix MP3 API is invoked through RTLL

Summary

- **Today we covered:**
  - Digital Filtering, Aliasing and Convolution
  - Speech and Audio Signal Basics
  - Speech And Audio encoding and decoding
  - External Codec Interfacing
  - Helix MP3 Decoder Algorithm
  - RTLL

Other tools



Multimedia Expansion Board

PIC32MX1XX/2XX Starter Kit
(Part # DM320013)

PIC32MX1XX/2XX Starter Kit

Additional Resources

- **Applied Speech and Audio Processing –** Ian McLoughlin, Cambridge University Press
- **Digital Signal Processing –** Proakis and Manolakis, Prentice Hall
- **Audio Signal Processing and Coding –** A. Spanias et al., Wiley International
- **The Scientist's and Engineer's Guide to DSP –** Steve Smith
- **Speex User Guide –** www.speex.org
- **Microchip's AN1367 "Porting the Helix MP3 Decoder onto Microchip's PIC32MX 32-bit MCUs"**
- **Microchip's AN1422 "High quality Audio Application using the PIC32"**

1674 AUD

Thank you

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, KeeLoq, KeeLoq logo, MPLAB, PIC, PICmicro, PICSTART, PIC^{32} logo, rfPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.
FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICkit, PICtail, REAL ICE, rfLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.
All other trademarks mentioned herein are property of their respective companies.