EECS 452 – Lecture 2

Today:

Sampling and reconstruction review
FIR and IIR filters
C5515 eZDSP
Direct digital synthesis

Reminders:

HW 1 is due on tuesday.
PPI is due on Thurs (email to hero by 5PM)
Lab starts next week.

Last one out should close the lab door!!!!

Please keep the lab clean and organized.

The numbers may be said to rule the whole world of quantity, and the four rules of arithmetic may be regarded as the complete equipment of the mathematician.

— James C. Maxwell

Sampling and reconstruction

Here, as last time, F denotes Herzian freq. and f denotes Digital freq.

Sampling is the part of the Analog-to-Digital Converter (ADC) that converts cts time signal $x(t)$ into discrete time signal $x[n] = X(nT_s)$. $F_s = 1/T_s$ is the *sampling rate* (samples/sec).

Reconstruction is the part of the Digital-to-Analog Converter (DAC) that converts discrete time signal $x[n] = x(nT_s)$ to cts time signal $x(t)$. $F_s = 1/T_s$ is the *conversion rate*.

Sampling and reconstruction are commonly combined into a ADC/DAC device called the CODEC (Coder-Decoder). The C5515 and DE2 have audio CODECs on board.

CODEC's work well as long as $x(t)$ does not have significant energy at frequencies above $F_s/2$ Hz
Can verify this condition using FT: $X(F) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi Ft}dt$

Simply bandlimited waveforms

Lowpass signal: Negligible energy ($X(F) = 0$) for all $|F| > B$. Single sided bandwidth is B Hz.

If sample $x(t)$ at $F_s > 2B$ samples/sec can exactly reconstruct. (Nyquist sampling theorem)

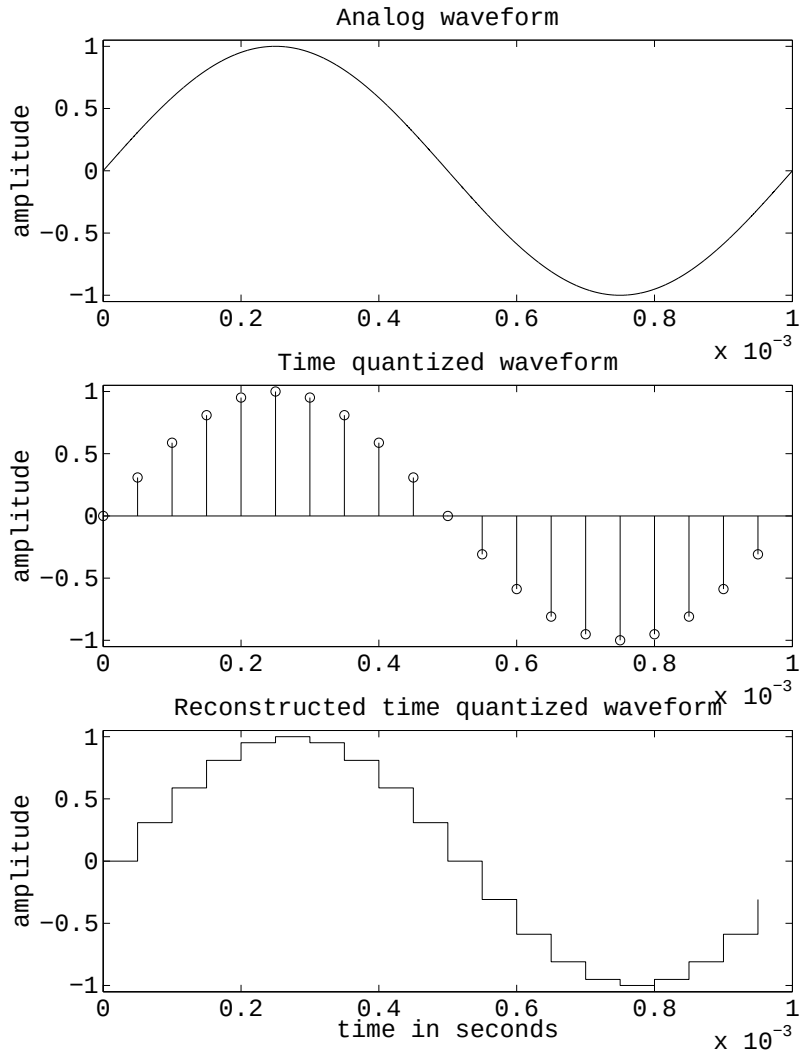
Bandpass signal: Negligible energy outside of a band, $B = F_2 - F_1$ not containing 0 Hz.

If sample at $F_s > 2B$ can exactly reconstruct. (bandpass sampling theorem[a])

Note that for bandpass waveforms do not need $F_s > 2F_2$!

[a] see http: www.eiscat.se:8080/usersguide/BPsampling.html

Sampling & reconstruction for a sinusoid

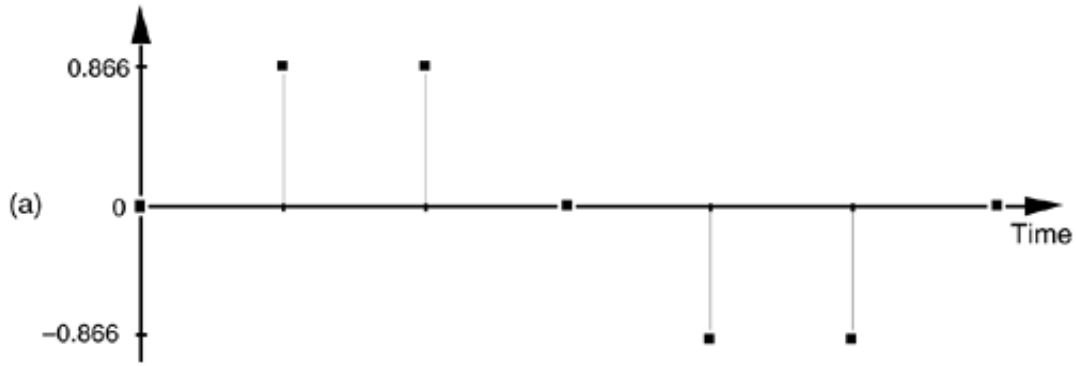


Analog waveform.

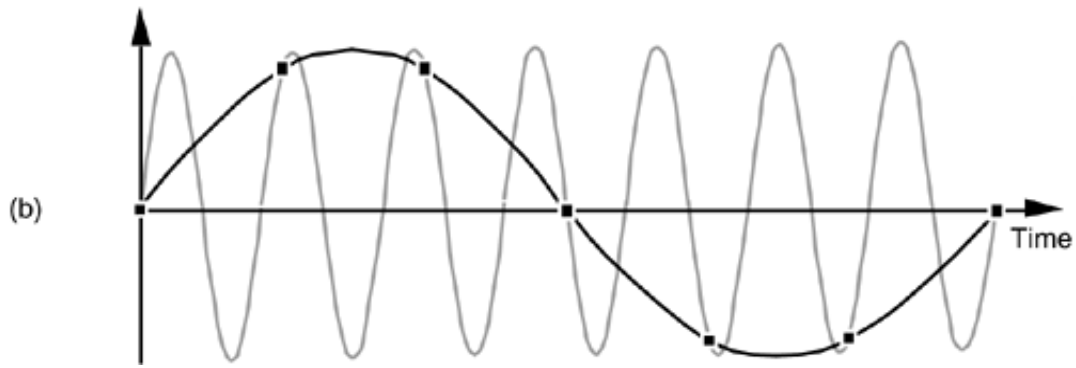
Uniformly time sampled.

Reconstructed (zero-order hold).

Cannot reliably reconstruct without knowing input frequency range



Samples from single period of sinusoid



There are many higher frequency sinusoids that could fit samples.

$\Rightarrow$ For unambiguous reconstruction need at least 2 samples per cycle

What happens when we sample?

Performing ideal sampling on an analog signal $x(t)$ means the following:

$$x_s(t) \quad = \quad x(t)p(t) = \sum_{n=-\infty}^{\infty} x(nT_s)\delta(t - nT_s)$$

where $p(t)$ is the pulse train $\sum_{n=-\infty}^{\infty} \delta(t - nT_s)$ with sample spacing T_s.

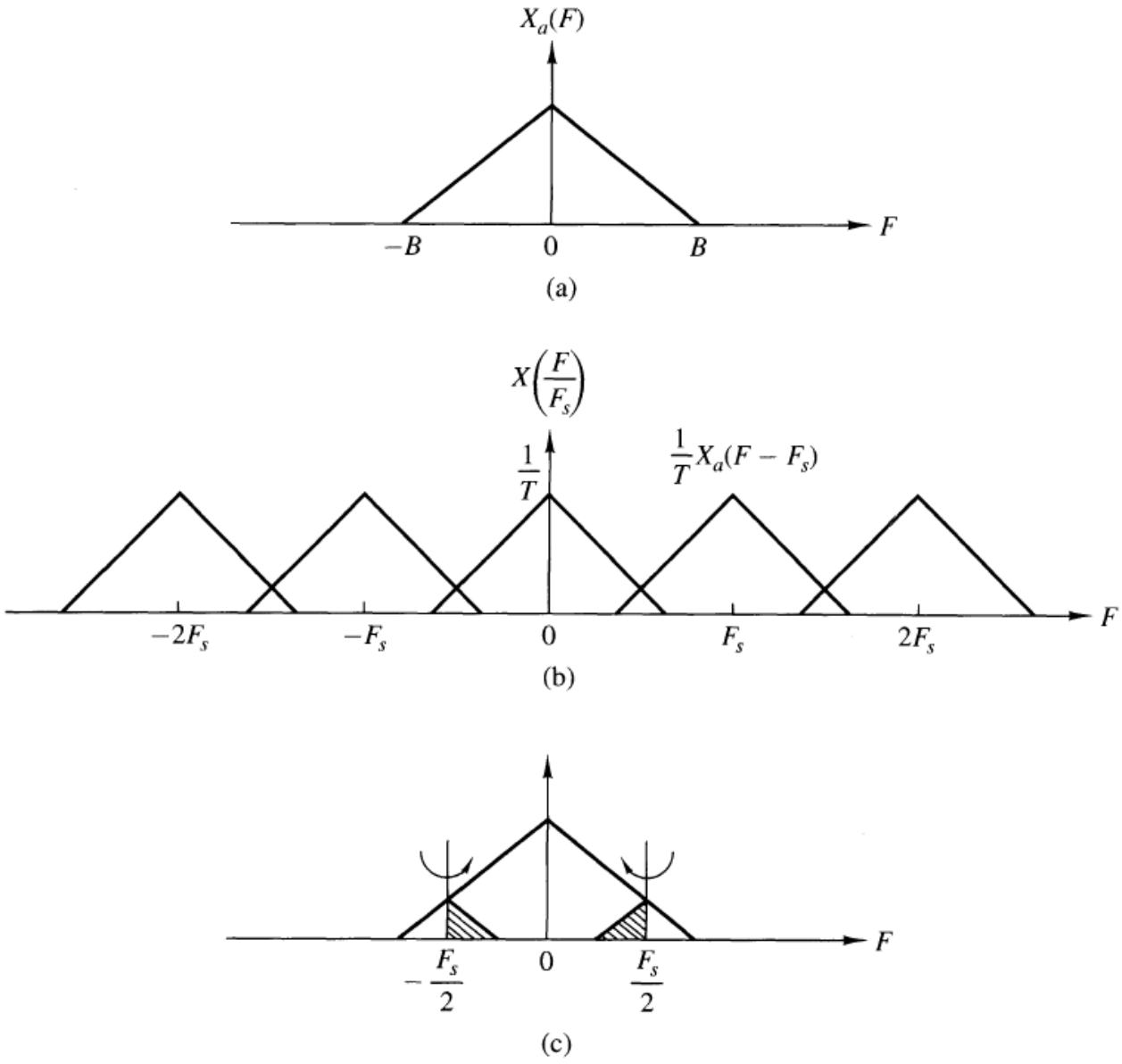
$$P(F) = \mathcal{F}\{p(t)\} = T_s^{-1} \sum_{k=-\infty}^{\infty} \delta(F - \frac{k}{T_s})$$

Taking Fourier transform of $x_s(t)$ ("*" denotes convolution)

$$X_s(F) \quad = \quad X(F) * P(F) = X(F) * \sum_{k=-\infty}^{\infty} \delta(F - \frac{k}{T_s})\frac{1}{T_s}$$

$$= \quad \frac{1}{T_s} \sum_{k=-\infty}^{\infty} X(F - \frac{k}{T_s})$$

Frequency domain view of aliasing



Where does "the" alias land?

What frequencies contain aliased components of a sampled signal?

Consider a sinusoidal signal $x(t)$ at F_c Hz

$$x(t) = \cos(2\pi F_c t) = \frac{e^{j2\pi F_c t} + e^{-j2\pi F_c t}}{2}$$

with continuous Fourier transform (CTF)

$$X(F) = \frac{1}{2}\delta(F + F_c) + \frac{1}{2}\delta(F - F_c)$$

If sample $x(t)$ at frequency F_s, sampled signal has CTF

$$X_s(F) = \sum_{k=-\infty}^{\infty} X(F - kF_s) = \frac{1}{2}\sum_{k=-\infty}^{\infty} \delta(F + F_c - kF_s) + \delta(F - F_c - kF_s)$$

Conclude: frequency F_c will alias to the frequencies $\{F_c \pm kF_s\}_{k\neq 0}$.

Relation between FT, DTFT, and DFT of x_s

Consider the Fourier transform $X_{WFT}(F)$ of $x_s(t)$ over the window $t \in [0, (N-1)T_s]$ (contains N samples)

$$X_{WFT}(F) = \int_0^{(N-1)T_s} x_s(t) e^{-j2\pi Ft} dt = \sum_{n=0}^{N-1} x[n] e^{-j2\pi FnT_s}$$

$$X_{DTFT}(f) = \text{DTFT}(x[n]) = \sum_{n=0}^{N-1} x[n] e^{-j2\pi fn}$$

$$X_{DFT}(k) = \text{DFT}(x[n]) = \sum_{n=0}^{N-1} x[n] e^{-j2\pi \frac{k}{N} n}$$

$\rightarrow X_{DTFT}(f) = X_{WFT}(fF_s)$ by identifying $F/F_s = f$ $(F_s = 1/T_s)$.

$\rightarrow X_{DFT}(k) = X_{DTFT}\left(\frac{k}{N}\right) = X_{WFT}\left(\frac{k}{N}F_s\right)$.

Units for Herzian, digital, and normalized digital frequency

Units *typically* used to describe baseband frequency range:

	units	range	limits
F	Hz	F_s	$-F_s/2 \leq F < F_s/2$
f	normalized Hz	1	$-1/2 \leq f < 1/2$
ω	normalized radians	2π	$-\pi \leq \omega < \pi$

Comments on sampling

The frequency $F_s/2$ (Hz) is called the *Nyquist* frequency.

Given a real valued lowpass spectrum with bandwidth, B the sample frequency equal to $2B$ is often called the *Nyquist* sample rate.

In practice one should sample at a rate of *at least* two or three times the Nyquist *rate*.

Common sample rates:

standard telephone system	8 kHz
wideband telecommunications	16 kHz
home music CDs	44.1 kHz
professional audio	48 kHz
DVD-Audio	192 kHz
instrumentation, RF, video	extremely fast

The anti-alias filter

Anti-alias filter H is an analog LPF with bandwidth $F_s/2$ applied to $x(t)$ before sampling.

Anti-alias filter eliminates frequencies that would otherwise be aliased into the baseband $F_s/2 \leq F \leq F_s/2$.

Anti-alias filters need to have sharp transition band at their cutoff frequency $F_s/2$.

The samplers of the CODECs on the DE2 and C5515 boards have sophisticated built-in anti-alias filters. The cutoff frequencies change with the selected sample rate.

Reconstruction: using interpolation

Assume that bandwidth B signal $x(t)$ has been sampled at Nyquist ($F_s = 2B$) giving samples $x[n]$.

Interpolate the samples $x[n] = x(nT_s)$ using the cardinal series (FT of ideal low-pass filter (LPF) with BW B):

$$x(t) = \sum_{n=-\infty}^{\infty} x(nT_s) \cdot \ \mathrm{sinc}(2\pi B(t - nT_s))$$

- This is called the cardinal series expansion of $x(t)$
- This perfectly recovers the input signal $x(t)$, per Nyquist sampling theorem
- Cardinal series reconstruction is not causal: output $x(t)$ depends on all past and future samples $x(nT_s)$.
- A simpler sample and hold reconstruction is used in practice - but requires anti-imaging filter (will study this later)

Discrete time LTI filters

The output $y[n]$ of discrete time LTI filter with input $x[n]$ is

$$y[n] = h[n] * x[n] = \sum_{k=-\infty}^{\infty} h[n-k]x[k] = \sum_{k=-\infty}^{\infty} h[k]x[n-k]\,.$$

Take DTFT to obtain equivalent frequency domain relation:

$$Y(f) = H(f)X(f), \quad f \in [-1/2, 1/2]$$

where $Y(f), X(f), H(f)$ are DTFTs of $y[n], x[n], h[n]$

- $h[n]$ is the *impulse response*: $h[n] = y[n]$ when $x[n] = \delta[n]$

$$\delta[n] = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$$

- $H(f)$ is the *transfer function* of LTI
- Often LTI I/O relation is expressed in z-transform domain

$$Y(z) = H(z)X(z), \quad z \in \mathbb{C}$$

The z-transform

The z-transform of a discrete set of values, $x[n]$, $-\infty < n < \infty$, is defined as

$$X_Z(z) = \mathcal{Z}(x[n]) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}$$

where z is complex valued. The z transform only exists for those values of z where the series converges. z can be written in polar form as $z = re^{j\theta}$.

r is the magnitude of z and θ is the angle of z. When $r = 1$, $|z| = 1$ is the unit circle in the z-plane.

When $x[n] = 0$ for $n < 0$, $X(z)$ reduces to single-sided z-transform

$$X_Z(z) = \mathcal{Z}(x[n]) = \sum_{n=0}^{\infty} x[n]z^{-n}$$

Note: often simply written without the subscript as $X(z)$

Elementary properties of z-transform

$$X_Z(z) = \mathcal{Z}\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n]z^{-n}$$

- If $X(z), Y(z)$ are z-transforms of $x[n], y[n]$

$$\mathcal{Z}\{ax[n] + by[n]\} = aX(z) + bY(z)$$

- If $X(z)$ is z-transform of $x[n]$ then

$$\mathcal{Z}\{x[n-k]\} = z^{-k}\mathcal{Z}\{x[n]\}$$

- If $X_Z(z)$ and $X_{DTFT}(f)$ the z-transform and DTFT of $x[n]$

$$X_{DTFT}(f) = X_Z(e^{j2\pi f})$$

FIR digital filters

Finite impulse response (FIR) digital filters produce output $y[n]$ samples as linear combination of the most recent input samples $x[n]$

$$y[n] = b_0x[n]+b_1x[n-1]+b_2x[n-2]+\cdots+b_Mx[n-M] = \sum_{k=0}^{M} b_kx[n-k]$$

M is called the order of the FIR filter.

Note that output depends on the $M+1$ most recent input samples

FIR filters are sometimes called "moving window summation filters"

Impulse response of FIR filter

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

Recall: impulse response $h[n]$ is filter output when input $x[n] = \delta[n]$

$$\delta[n] = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$$

From (1) we obtain

$$h[n] = \begin{cases} b_n, & n = 0, \ldots, M \\ 0, & n < 0, \;\; n > M \end{cases}$$

as the impulse response of the FIR filter.

FIR (Direct Form) block diagram

Time domain input-output relation:

$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_M x[n-M]$$

in z-domain

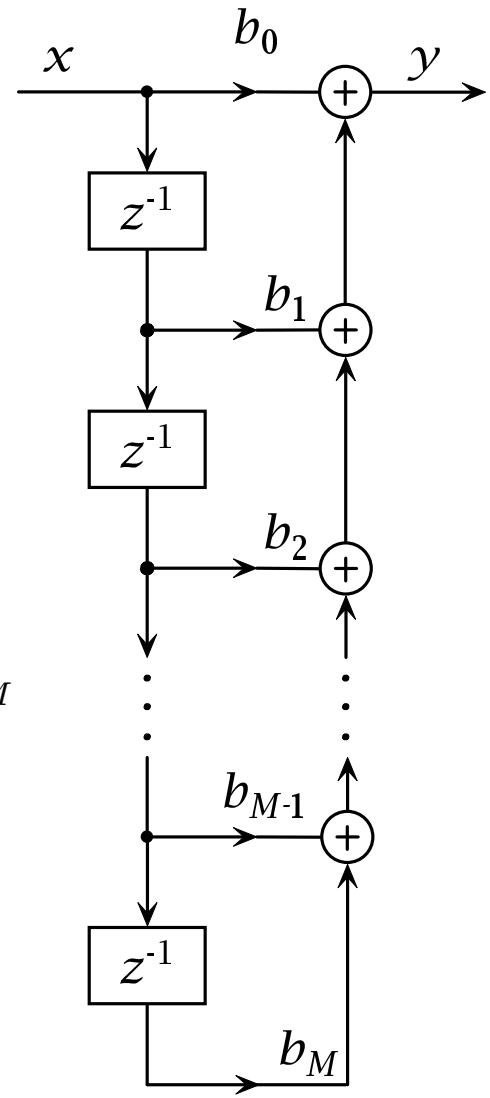
$$Y(z) \quad = \quad (b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}) X(z)$$

Transfer function (z-domain)

$$H(z) = \frac{Y(z)}{X(z)} \quad = \quad b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_M z^{-M}$$

Polynomial over $z^{-1} \in \mathbb{C}$: has M zeros

z^{-1} corresponds to the "unit delay operator", $X(z)z^{-1}$ is the z-transform of $x[n-1]$.



FIR frequency transfer function

Z-domain transfer function of M-th order FIR filter with coefficients $\{b_n\}_{n=0}^{M}$

$$H_z(z) = \sum_{n=0}^{M} h[n] z^{-n} = \sum_{n=0}^{M} b_n z^{-n} = \frac{Y(z)}{X(z)}$$

To get (digital) frequency domain transfer function you evaluate $H_z(z)$ on the unit circle $z = e^{j2\pi f}$, $f \in [-1/2, 1/2]$

$$H(f) = H_z\left(e^{j2\pi f}\right) = \sum_{n=0}^{M} h[n] e^{-j2\pi f n}$$

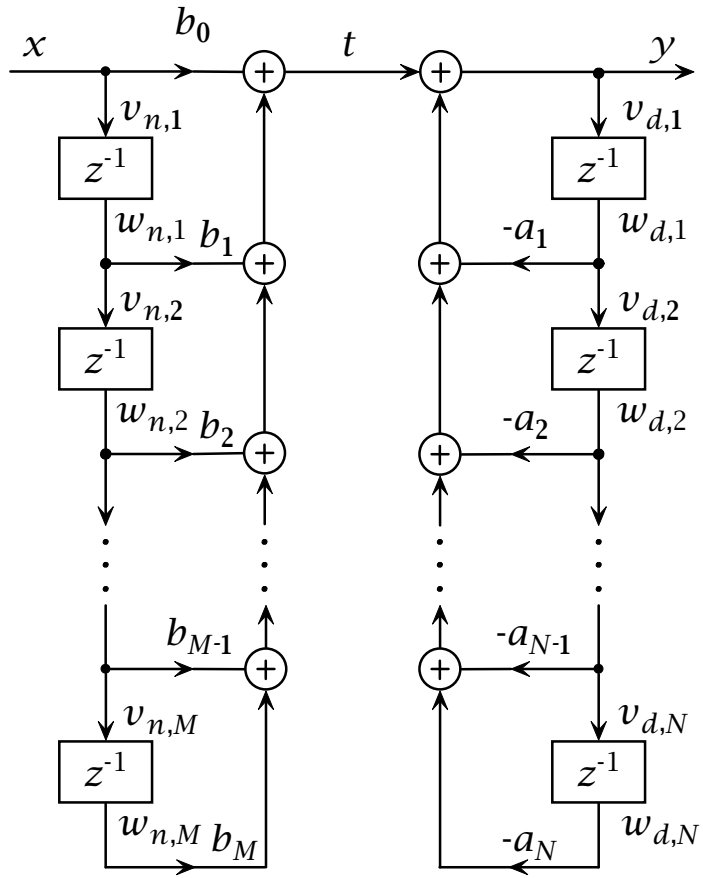
or, less compactly,

$$H(f) = h[0] + h[1]e^{-j2\pi f} + \cdots + h[M]e^{-j2\pi f M}.$$

IIR (Infinite Impulse Response) Filter

By contrast, in an infinite impulse response (IIR) filter, output depends on not only current and previous M input samples, but also the previous N filter outputs.

$$\begin{aligned} y[n] &= b_0 x[n] + b_1 x[n-1] + \ldots \\ &+ b_M x[n-M] \\ &- a_1 y[n-1] - \cdots - a_N y[n-N] \end{aligned}$$

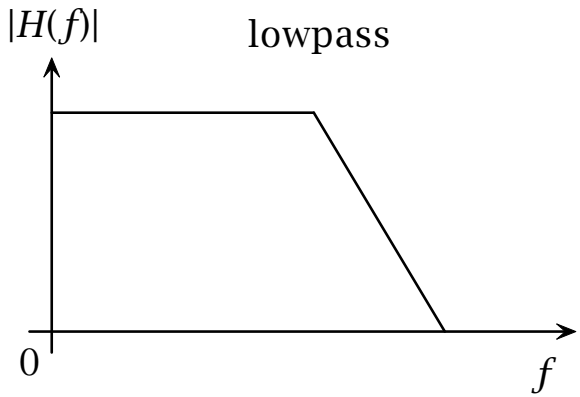


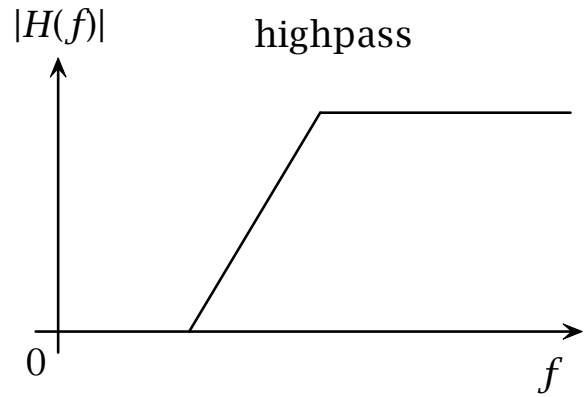
Transfer function

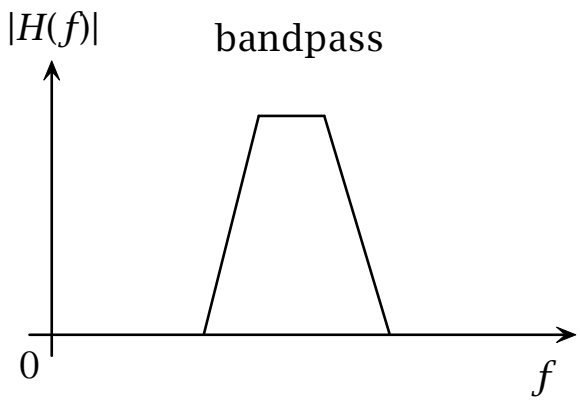
$$\begin{aligned} H(z) &= \frac{Y(z)}{X(z)} \\ &= \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}} \end{aligned}$$

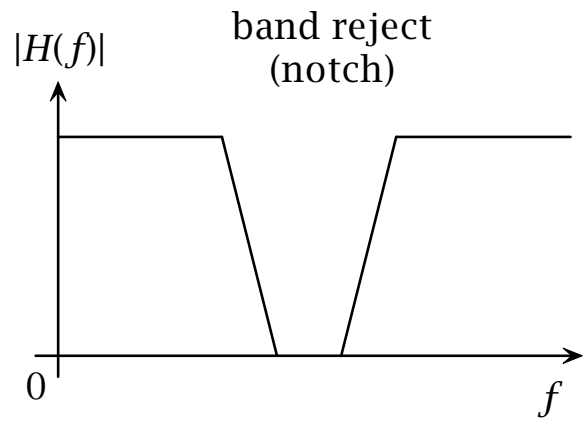
A ratio of polynomials: has N poles and M zeros as function of $z^{-1} \in \mathbb{C}$.

Different types of filter transfer functions

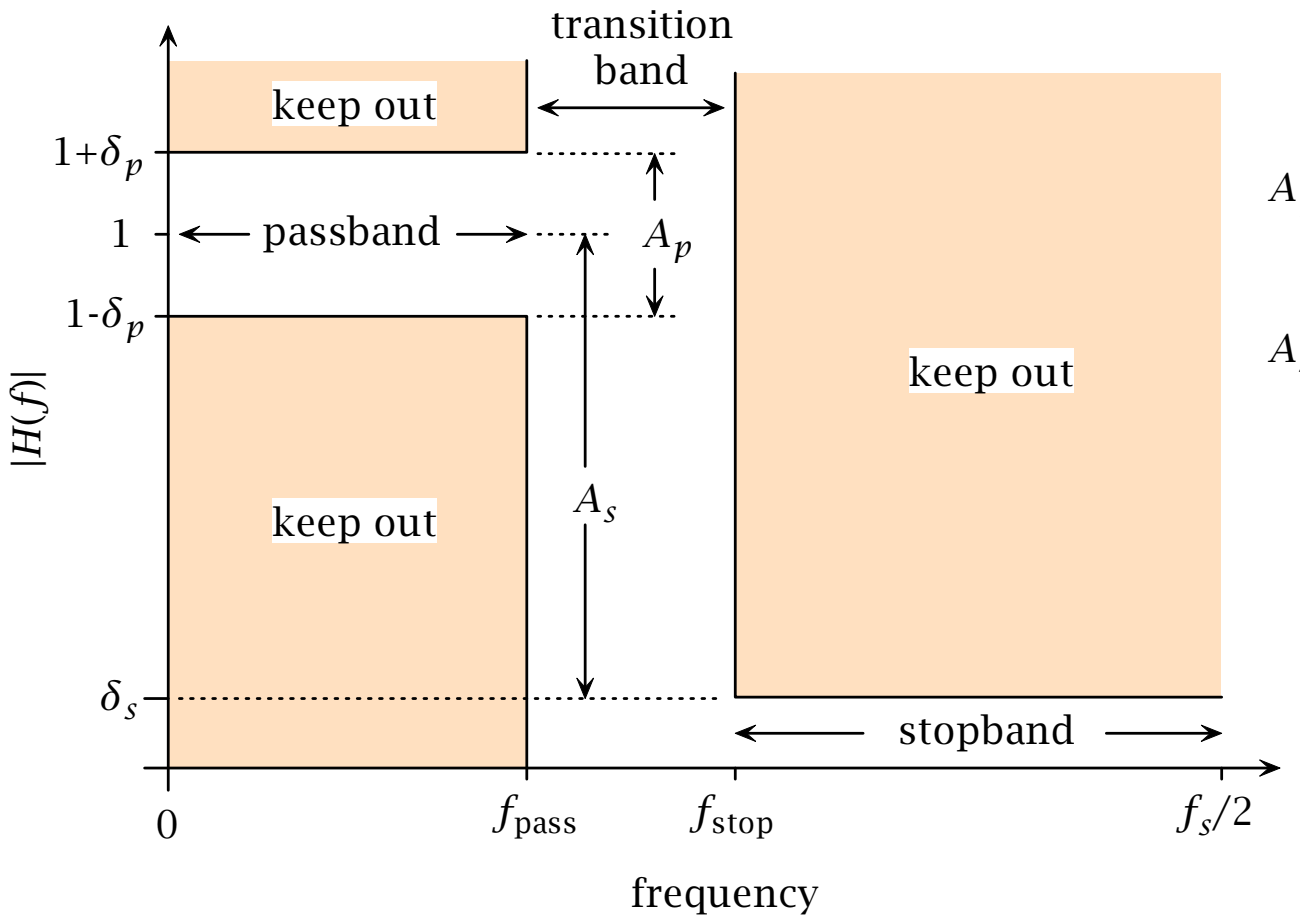








Lowpass filter design template



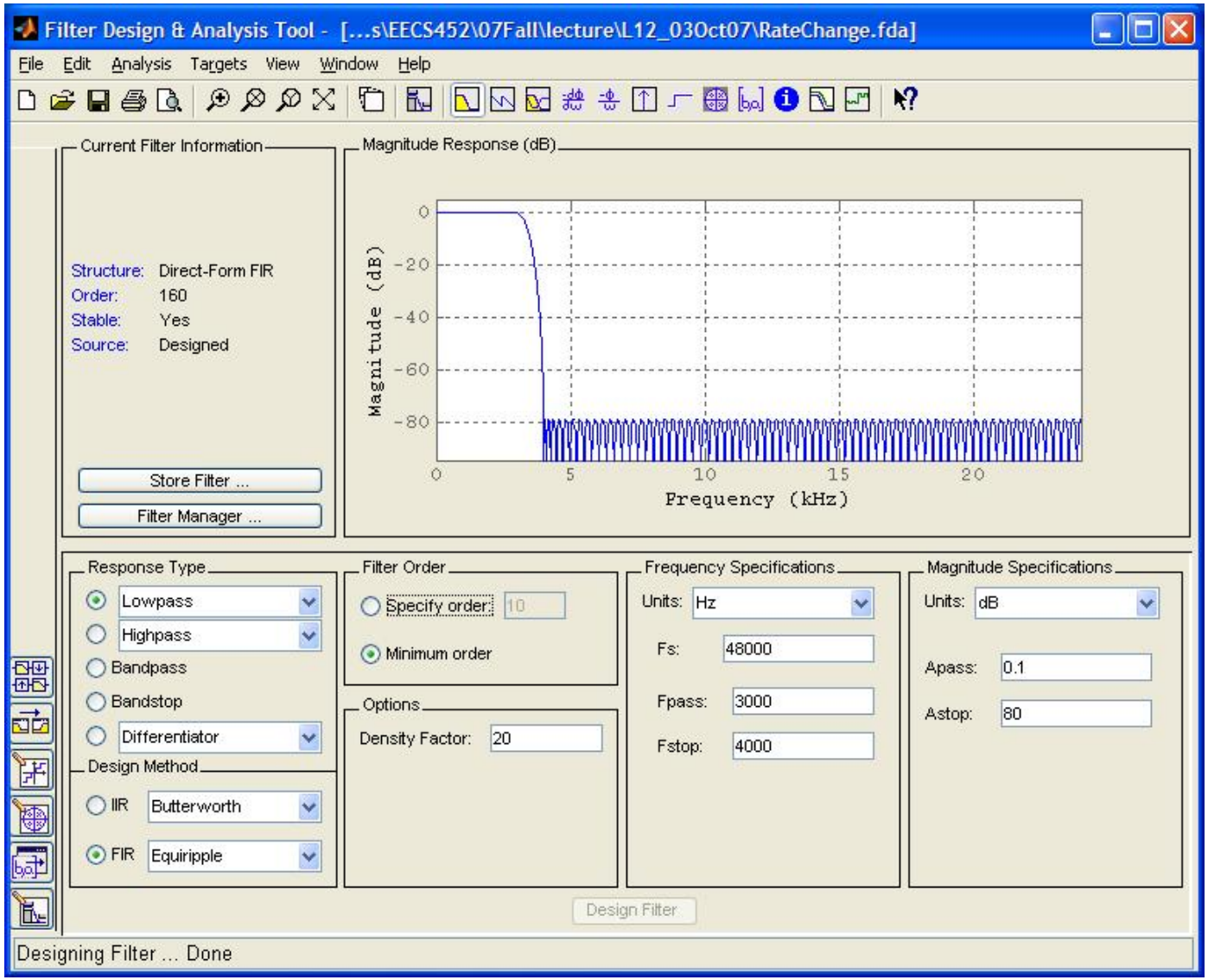
$$A_p = 20\log_{10}\left(\frac{1+\delta_p}{1-\delta_p}\right) \text{ dB}$$

$$A_s = -20\log_{10}(\delta_s) \text{ dB}$$

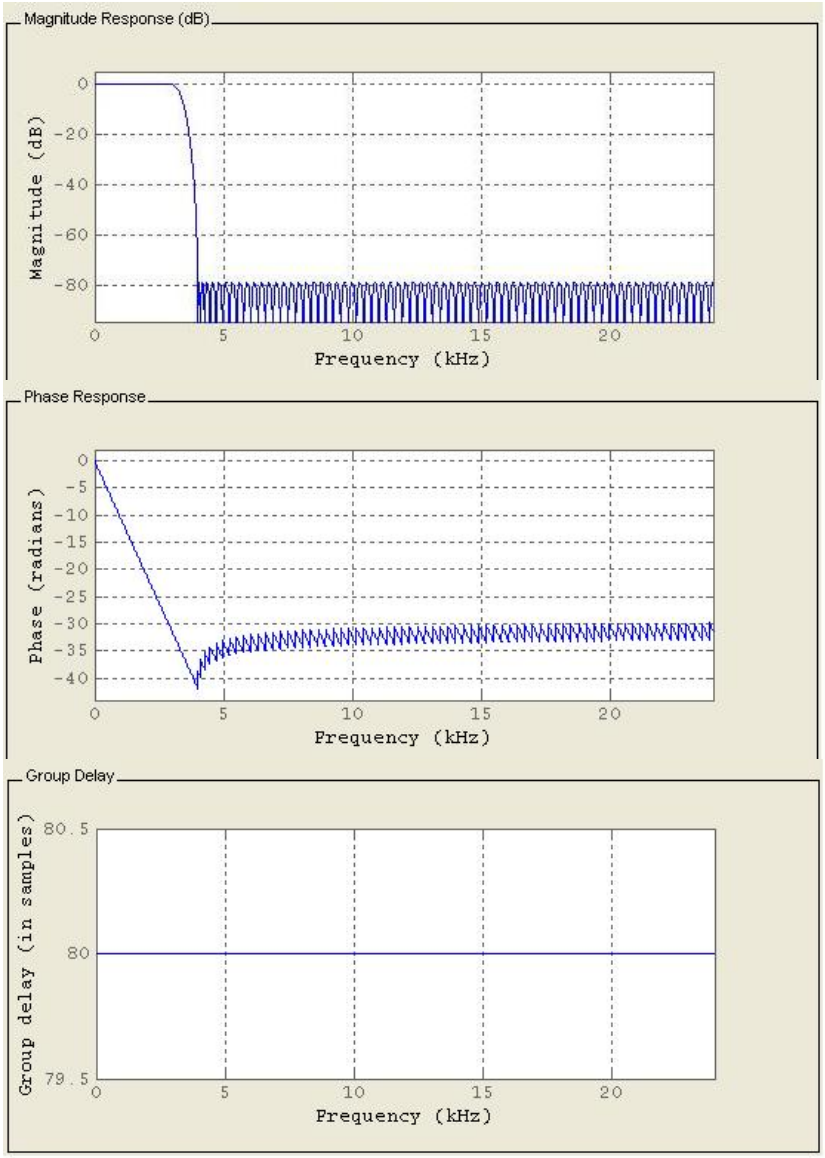
Equiripple LPF FIR filter design example

- Low pass filter.
- F_s=48000 Hz.
- Bandpass ripple: ± 0.1 dB.
- Transition region 3000 Hz to 4000 Hz.
- Minimum stop band attenuation: 80 dB.

Matlab's fdatool's solution



fdatool's magnitude, phase and group delay



What is group delay?

A digital filter transfer function has a magnitude and a phase

$$H(f) = |H(f)|\, e^{j\theta(f)}\,.$$

The filter's group delay at frequency f is defined as

$$\tau(f) = -\frac{1}{2\pi}\frac{d\theta(f)}{df}$$

Linear phase filters: $\theta(f) = -2\pi f\tau$, where τ is independent of frequency

- have group delay is the same at all frequencies
- shift each frequency component of input by same amount of delay
- have group delay proportional to the negative slope of the phase $\theta(f)$

The group delay of a digital filter is often expressed in seconds

$$\tau = \left(-\frac{1}{2\pi}\frac{d\theta(f)}{df}\right) T_s$$

(Recall: $f = F/F_s = FT_s$).

Why is group delay important?

Constant group delay is important in digital communications.

A system not having constant group delay distorts digital pulse waveforms. This smears them together and makes it difficult to make bit decisions.

Many communication systems have a special circuit that can adaptively equalize channel phase response to obtain a constant group delay. To do so it must measure it. This leads to the use of a training waveform.

FIR filters can be designed to give yield constant group delay as measured from input $x(t)$ to output $y(t)$ of the DSP+CODEC system.

C55xx implementation using TI's DSP Library

`dsplib`: TI's implementations of DSP functions for the C55xx

"These routines are typically used in computationally intensive real-time applications where optimal execution speed is critical. By using these routines you can achieve execution speeds considerable faster than equivalent code written in standard ANSI C language."

Functional categories of dsplib routines

Fast-Fourier Transforms (FFT)
Filtering and convolution
Adaptive filtering
Correlation
Math
Trigonometric
Miscellaneous
Matrix

(TMS320C55xx DSP Library Programmers Reference (spru422))

TI's DSPlib FIR

fir *FIR Filter*

Function ushort oflag = fir (DATA *x, DATA *h, DATA *r, DATA *dbuffer, ushort nx, ushort nh)

Arguments

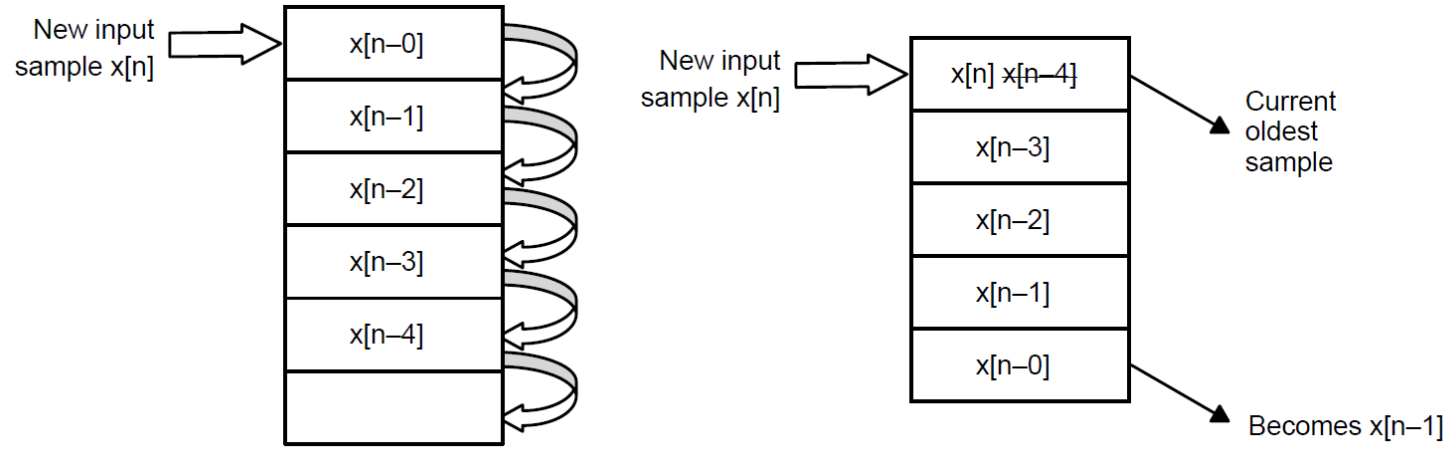
x[nx]	Pointer to input vector of nx real elements.
h[nh]	❑ Pointer to coefficient vector of size nh in normal order. For example, if nh=6, then h[nh] = {h0, h1, h2, h3, h4, h5} where h0 resides at the lowest memory address in the array.
r[nx]	Pointer to output vector of nx real elements. In-place computation (r = x) is allowed.
dbuffer[nh+2]	Pointer to delay buffer of length nh = nh + 2 ❑ In the case of multiple-buffering schemes, this array should be initialized to 0 for the first filter block only. Between consecutive blocks, the delay buffer preserves the previous elements needed. ❑ The first element in this array is special in that it contains the array index−1 of the oldest input entry in the delay buffer. This is needed for multiple-buffering schemes, and should be initialized to 0 (like all the other array entries) for the first block only.
nx	Number of input samples
nh	The number of coefficients of the filter. For example, if the filter coefficients are {h0, h1, h2, h3, h4, h5}, then nh = 6. Must be a minimum value of 3. For smaller filters, zero pad the coefficients to meet the minimum value.
oflag	Overflow error flag (returned value) ❑ If oflag = 1, a 32-bit data overflow occurred in an intermediate or final result. ❑ If oflag = 0, a 32-bit overflow has not occurred.

TI's DSPlib conventions

Argument	Description
x,y	argument reflecting input data vector
r	argument reflecting output data vector
nx,ny,nr	arguments reflecting the size of vectors x,y, and r respectively. In functions where nx = nr = nr, only nx has been used.
h	Argument reflecting filter coefficient vector (filter routines only)
nh	Argument reflecting the size of vector h
DATA	data type definition equating a short, a 16-bit value representing a Q15 number. Usage of DATA instead of short is recommended to increase future portability across devices.
LDATA	data type definition equating a long, a 32-bit value representing a Q31 number. Usage of LDATA instead of long is recommended to increase future portability across devices.
ushort	Unsigned short (16 bit). You can use this data type directly, because it has been defined in dsplib.h

`dbuffer` is a circular buffer

http://www.ti.com/lit/an/spra645a/spra645a.pdf



Conventional buffer (shift old samples) Circular buffer (shift pointer)

Circular buffer is more power and computation efficient for FIR filtering

$$y[n] = \sum_{k=0}^{M} b_n x[n-k]$$

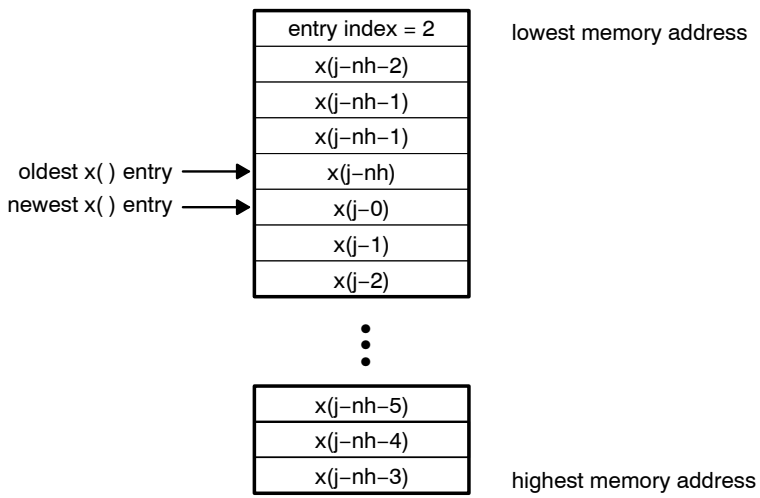
TI's FIR notes

Implementation Notes The first element in the dbuffer array (index = 0) is the entry index for the input history. It is treated as an unsigned 16-bit value by the function even though it has been declared as signed in C. The value of the entry index is equal to the index – 1 of the oldest input entry in the array. The remaining elements make up the input history. Figure 4–16 shows the array in memory with an entry index of 2. The newest entry in the dbuffer is denoted by x(j–0), which in this case would occupy index = 3 in the array. The next newest entry is x(j–1), and so on. It is assumed that all x() entries were placed into the array by the previous invocation of the function in a multiple-buffering scheme.

The dbuffer array actually contains one more history value than is needed to implement this filter. The value x(j–nh) does not enter into the calculations for for the output r(j). However, this value is required in other DSPLIB filter functions that utilize the dual-MAC units on the C55x, such as FIR2. Including this extra location ensures compatibility across all filter functions in the C55x DSPLIB.

Figure 4–16, Figure 4–17, and Figure 4–18 show the dbuffer, x, and r arrays as they appear in memory.

Figure 4–16. dbuffer Array in Memory at Time j



TI's FIR notes (cont.)

Figure 4–17. x Array in Memory

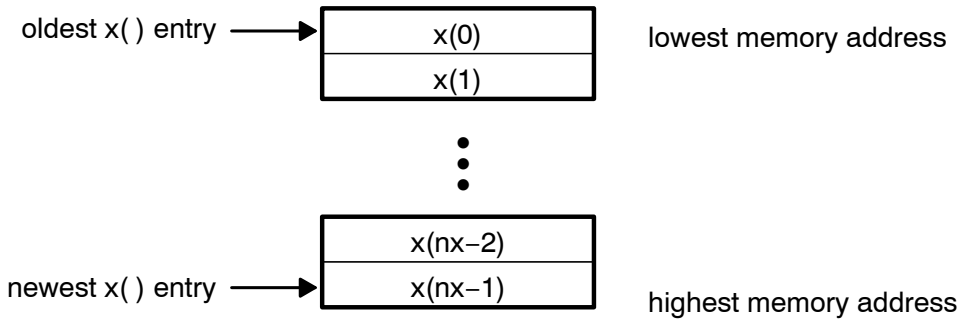
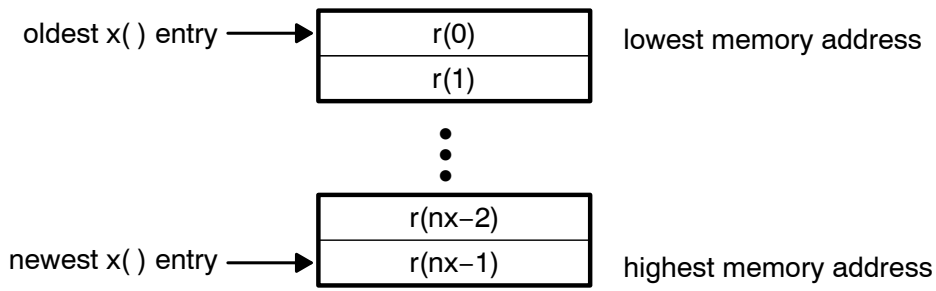


Figure 4–18. r Array in Memory



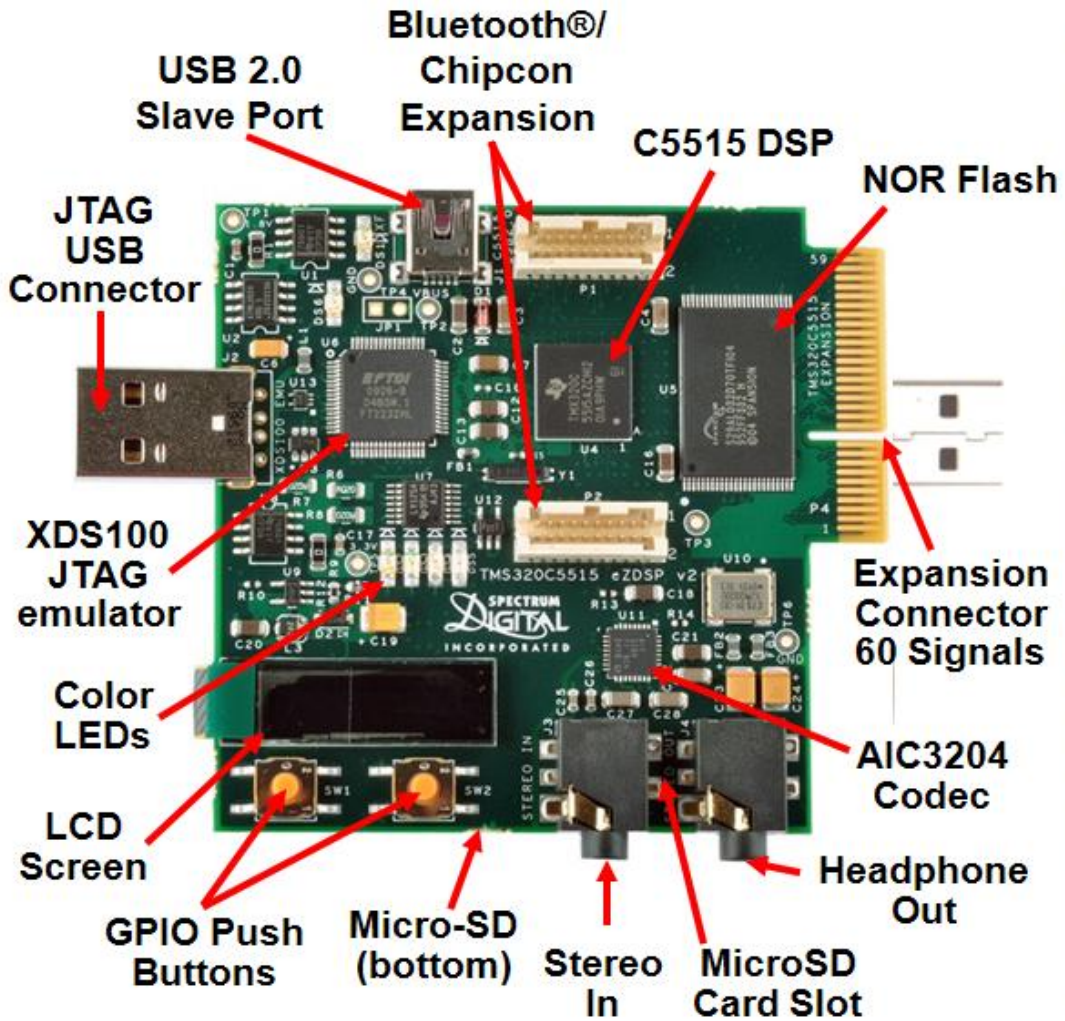
Example See examples/fir subdirectory

Benchmarks (preliminary)

Cycles†	Core:	nx * (2 + nh)
	Overhead:	25
Code size (in bytes)	107	

C5515 eZDSP

C5515 eZdsp USB Stick Development Tool



- On-board emulation
- Integrated audio codec
- Audio line out/line in connectors
- USB 2.0 Slave Port
- Micro-SD Slot
- 4 MByte NOR Flash
- Extension connectors (UART/SPI/I2S/I2C/GPIO/SD)
- Simple form factor plugs into any USB host port
- Simplifies development tools setup by eliminating power and interface cables
- $79

C5515 eZDSP Description

The C5515 is a member of TI's TMS320C5000 fixed-point Digital Signal Processor (DSP) product family and is designed for *low-power applications*. It is based on the TMS320C55x DSP generation CPU processor core.

TI's list of C5515 DSP applications include:

Wireless Audio Devices
Echo Cancellation Headphones
Portable Medical Devices
Voice Applications
Industrial Controls
Fingerprint Biometrics
Software Defined Radio
`http://processors.wiki.ti.com/index.php/C5515`

In this lecture we focus on the CODEC.

Direct Digital Synthesis (DDS) basic idea

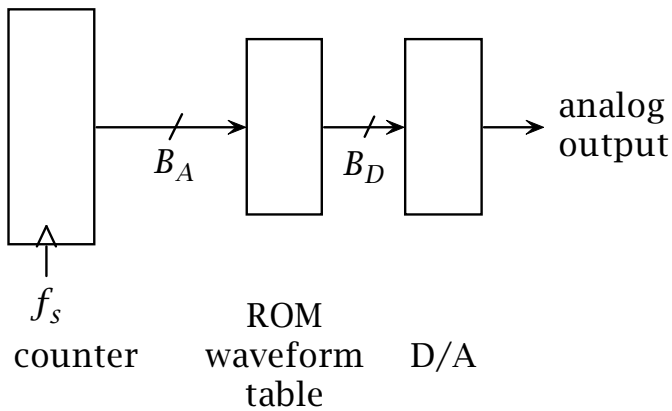
A method for digitally creating sine waves of arbitrary frequency by reading samples out of memory.

- We store a sine table in ROM (read-only memory).
  - These values are samples from a single period.
  - The number of values we can store/access is determined by the size of the address we use.
  - Say we use a B-bit address, so we can store 2^B values.
  - The number of values determines the *resolution* of the table.
  - These values are *frequency-less*.
- Now let's read out these values at a certain *speed*.
  - Say x values per second.
  - The output (after D/A processing) now form a waveform of frequency ???
- DDS idea: as long as we can arbitrarily control the speed at which we read/drive out these values, we can generate waveforms of arbitrary frequency.

DDS: read out samples using a counter

Let's use a binary counter, say B_A bits.

- Driven by a f_s Hz clock: counter increments once per tick.
- Use the counter value to address the sine table, also B_A bits.
- So what is the output frequency?



- When the counter wraps around, we start reading from the beginning of the sine table, i.e., the next period.
- The time it takes to finish one period is simply the time it takes to count to maximum: $2^{B_A}/f_s$ seconds.
- Output frequency: $f_s/2^{B_A}$.

DDS: increasing the sinusoidal frequency

What if we want to increase the sinusoidal frequency without increasing f_s?

- We can try to make the counter increment by n at a time, instead of 1. (We will see this can be done in a minute.)
- This way it wraps around in $\frac{2^{B_A}}{n \cdot f_s}$ seconds, an n-fold increase in output frequency!
- However, if we are using a B_A-bit sine table then we are skipping a lot of samples!
- We only address 1 in every n samples of the sine table: lower resolution and less D/A quality.
- But if this is what we do we can store fewer samples using a smaller table.

DDS: decreasing the sinusoidal frequency

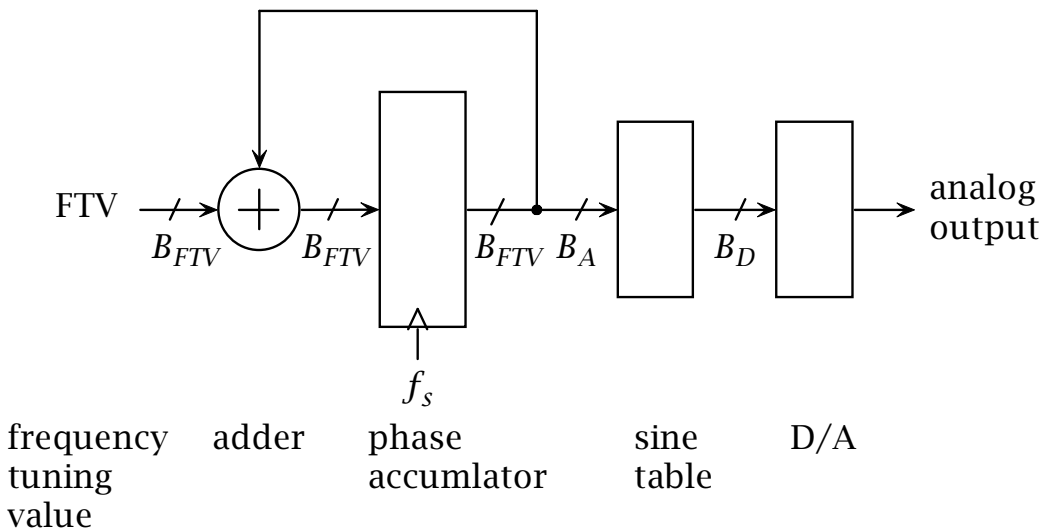
What if we want to increase the sinusoidal period without decreasing f_s?
Q. Can you make the counter count slower without changing f_s?
A. Yes

- Keep the sine table B_A-bit.
- Increase the counter to $B_{FTV} > B_A$ bit.
- With the same clock, the counter now counts slower: it takes $2^{B_{FTV}}/f_s$ seconds to wrap around.
- Use the highest B_A bits of the counter value to address the sine table.
- So it takes $2^{B_{FTV}-B_A}$ ticks to move to the next sample, if the counter increments by 1 per tick.
  - Example: $B_{FTV} = 8$, $B_A = 4$. We have 16 samples in the table. The counter counts to 255 before wrapping around. It takes 16 counter increments to increase the table address by 1.
- The output frequency: $f_s/2^{B_{FTV}}$, a $2^{B_{FTV}-B_A}$-fold decrease!

The DDS design that achieves both

Replace the counter with an accumulator and adder. Now can use steps larger than 1 in the increment.
Frequency tuning value (FTV) is the step size of increment.

Use more bits in the accumulator than in the ROM address. This gives finer frequency resolution since output frequency can only be integer multiples of $\frac{f_s}{2^{B_{FTV}}}$.

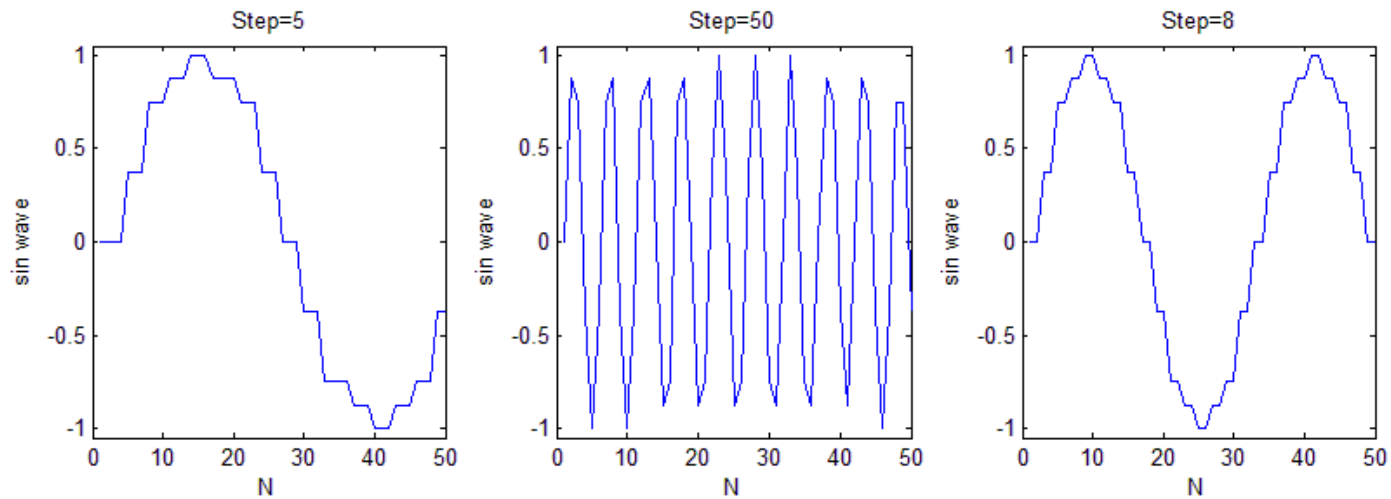


Filter at D/A output not shown.
What is the output frequency? $f_o = FTV \frac{f_s}{2^{B_{FTV}}}$.

Results illustrated

Difference under different FTV values.

With $B_{FTV} = 8, B_A = B_D = 4$, and $f_s = 2^{14}$Hz. Assume output ranges from $[-1, 1]$V.



DDS discussion

Synthesized waveform is an approximation to an analog one. Need to balance step size, clock rate, ROM size and number of bits.

- Output frequency:
  - It has nothing to do with B_A, as long as $B_A \leq B_{FTV}$.
  - It is only determined by how fast we accumulate/count the integers, and how fast we wrap around.
  - B_A does determine the resolution of the table, and the quality of D/A output.
- Hypothetically, what if $B_A > B_{FTV}$?
  - Then we have not reached the end of the table (a single period) when counting wraps around.
  - We will always be missing a segment of the period.
  - Output waveform distorted, though frequency as desired.
  - One solution is to use the higher B_{FTV} bits of B_A.
- The lowest frequency you can generate: $\frac{f_s}{2^{B_{FTV}}}$.

DDS example

We can implement a direct digital sinewave synthesizer on the C5515 using the codec's sample clock. A number of values are possible, let's use $f_s = 48$ kHz. An unsigned long (32 bits) can be used as the accumulator, `ac0`. A table of 256 samples of single period of a sinewave will be used in place of the ROM. The output frequency will be

$$f_o = \text{FTV}\,\frac{48000}{2^{32}} \quad \text{Hz.}$$

For a desired f_o the value of FTV can be found

$$\text{FTV} = \frac{2^{32} f_o}{48000}.$$

For $f_o = 1000$ Hz we have FTV$= 89,478,485.333\cdots$. If we round FTV to the closest integer, then the error in f_o will be

$$\frac{1}{3}\,\frac{48000}{2^{32}} \approx 3.7 \times 10^{-6} \quad \text{Hz.}$$

About 4 parts in 10^9. Good enough for most applications and probably much better than the crystal being used to generate the 48 kHz.

Summary of what we covered today

- Sampling and reconstruction
  - Fourier spectrum of sampled cts time signals
  - Relation between windowed FT, DTFT, DFT
  - Anti-aliasing filter
- FIR and IIR digital filters
  - z-transform, frequency response, transfer function
  - matlab's fdatool for filter design, phase shift and group delay
  - TI's DSPlib FIR filter implementation
- Direct Digital Synthesis (DDS)
- Next: Finite precision arithmetic