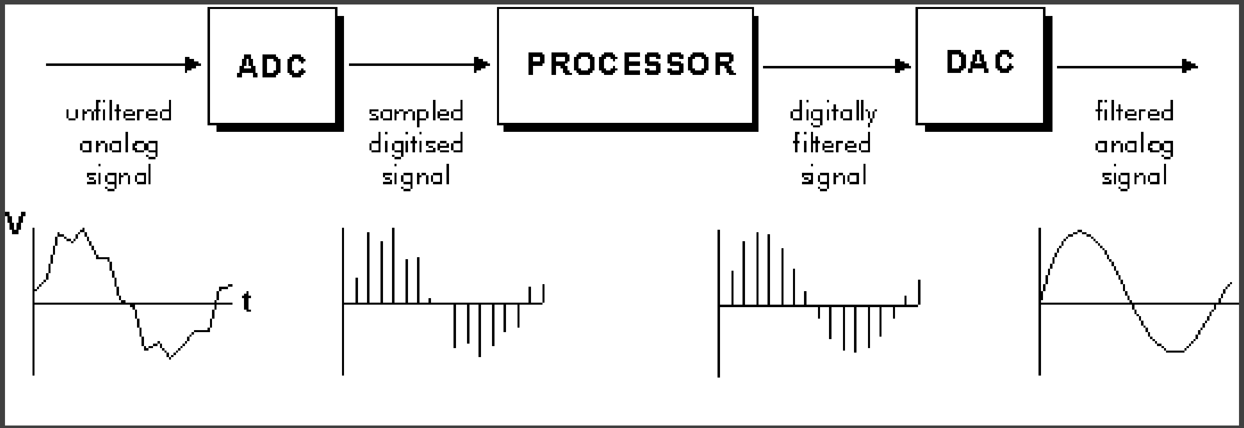
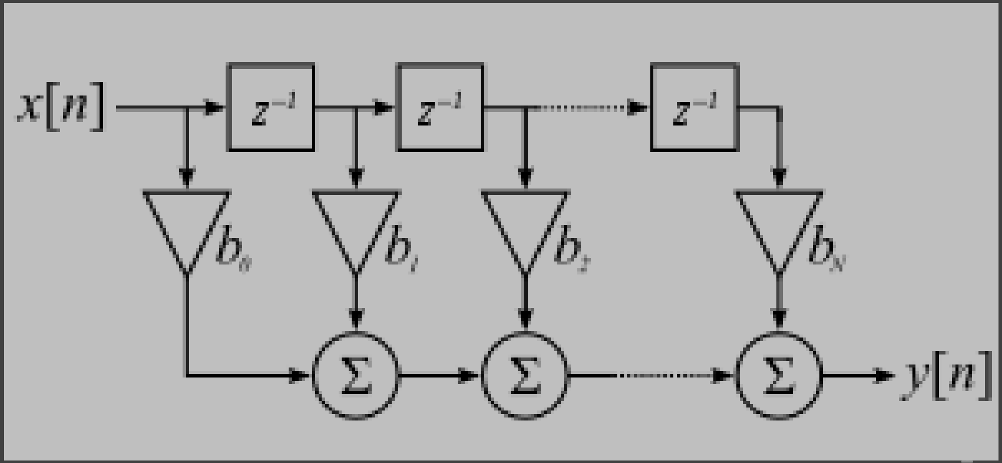
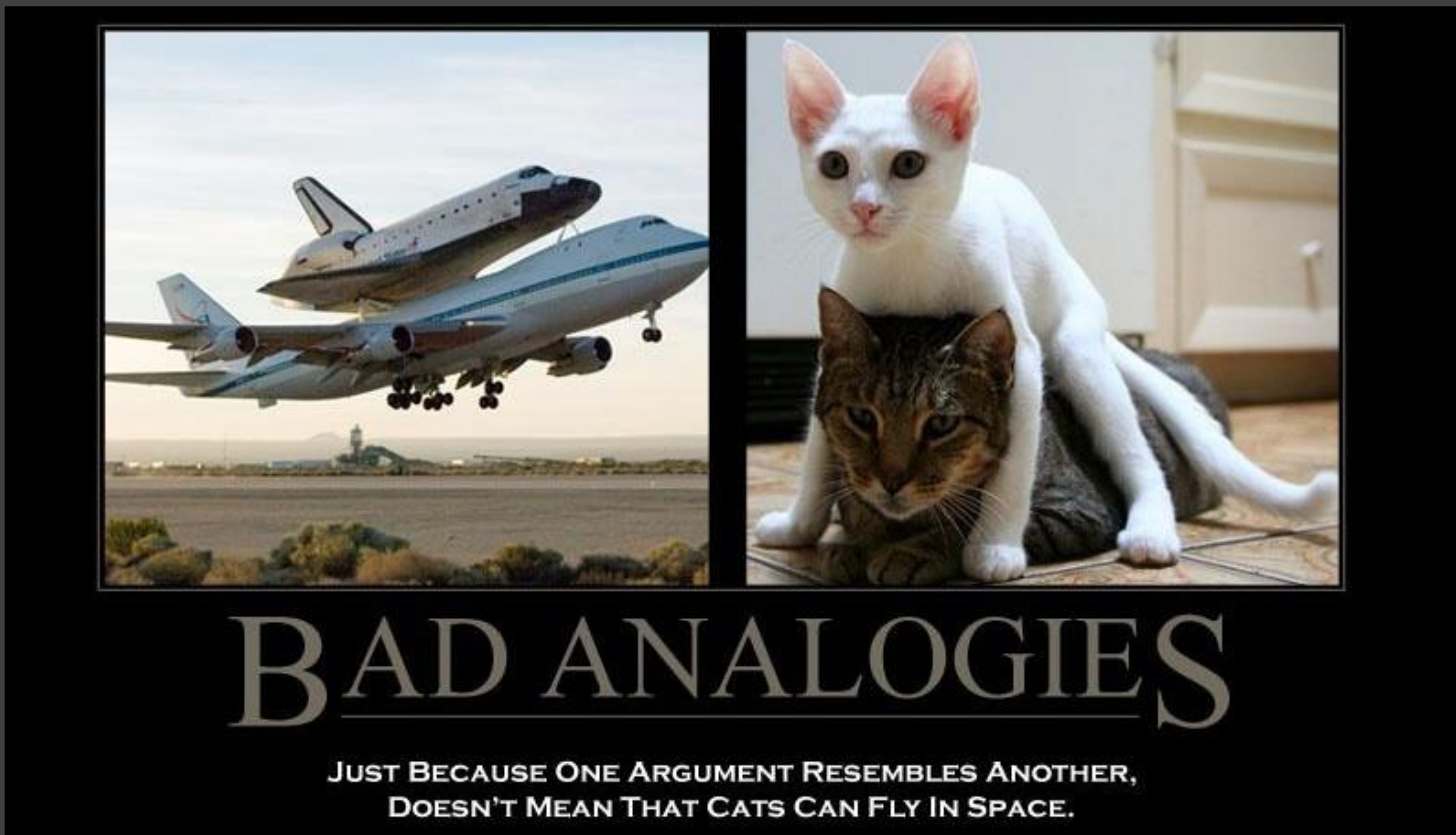
Intro to DSP Filters

June 2020





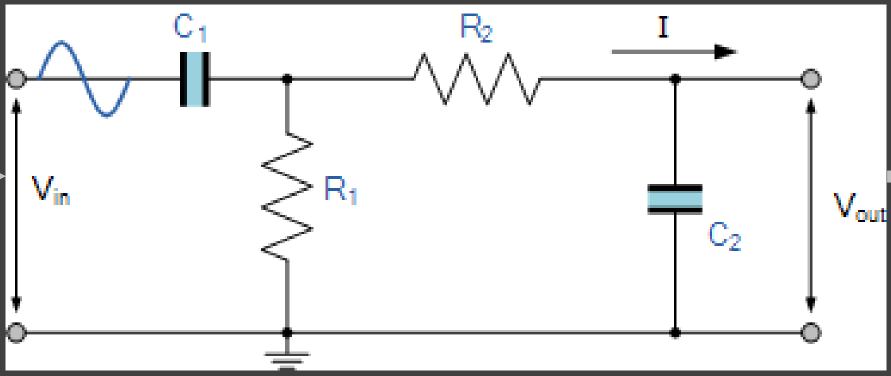
Rules of thumb, assumptions and mixed-quality analogies to come!



What is a DSP Filter?

≠ Analog Signal Processing

Input



Output

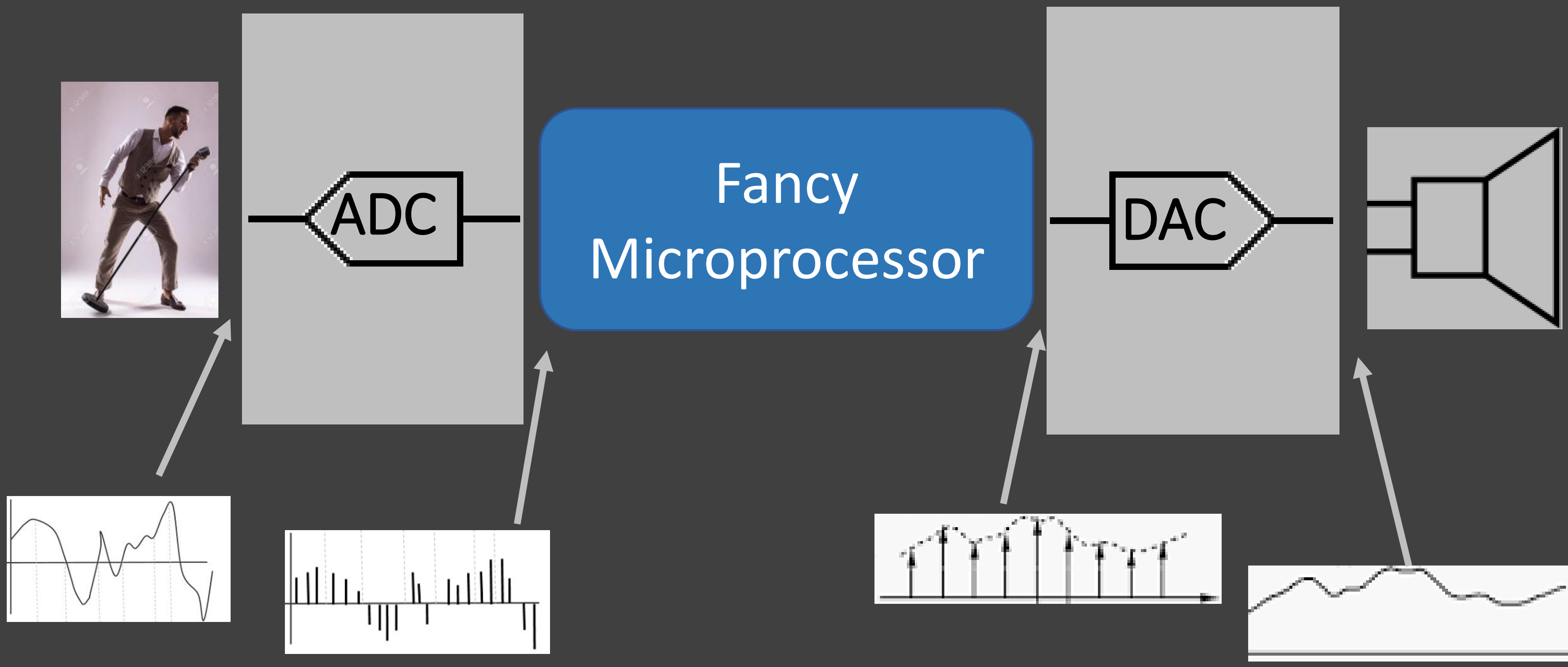
DSP = Digital Signal Processing

Input

Fancy Microprocessor

Output

Audio Implementation



Why use Analog vs Digital Processing?

Analog:

- Infinite bandwidth
  - Good for high frequency applications
- Doesn’t require ADC, microprocessor
- Can offload complexity from processor

Digital:

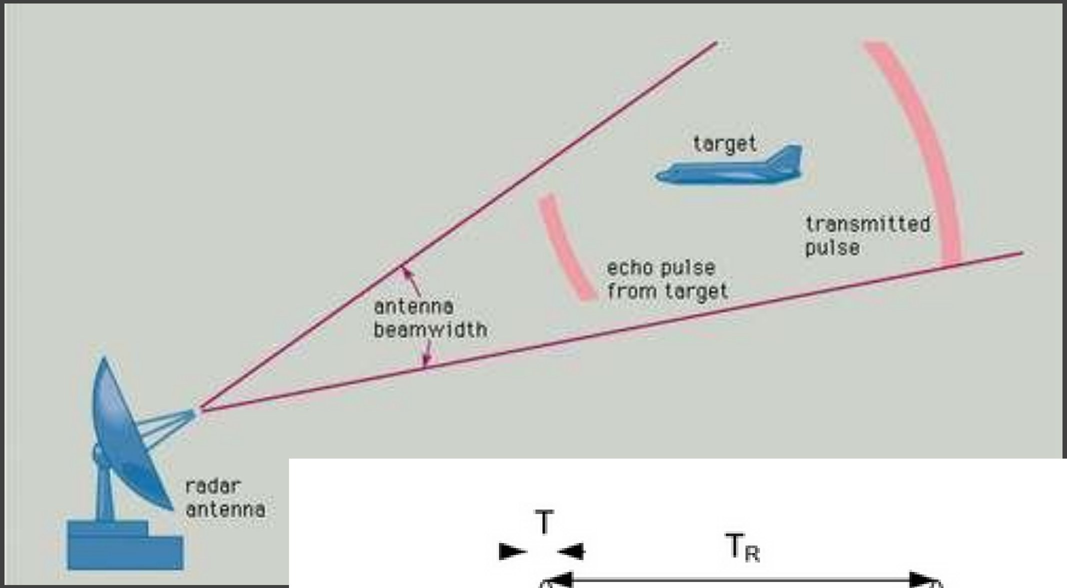
- Can change filter characteristics easily
- More ideal filters are possible
- Can do types of processing that analog can’t
  - Data compression
  - Math operations like cos(), atan()
  - Pattern recognition, red-eye reduction
- Lots of specific knowledge (Nyquist sample rates, quantization noise, etc.)

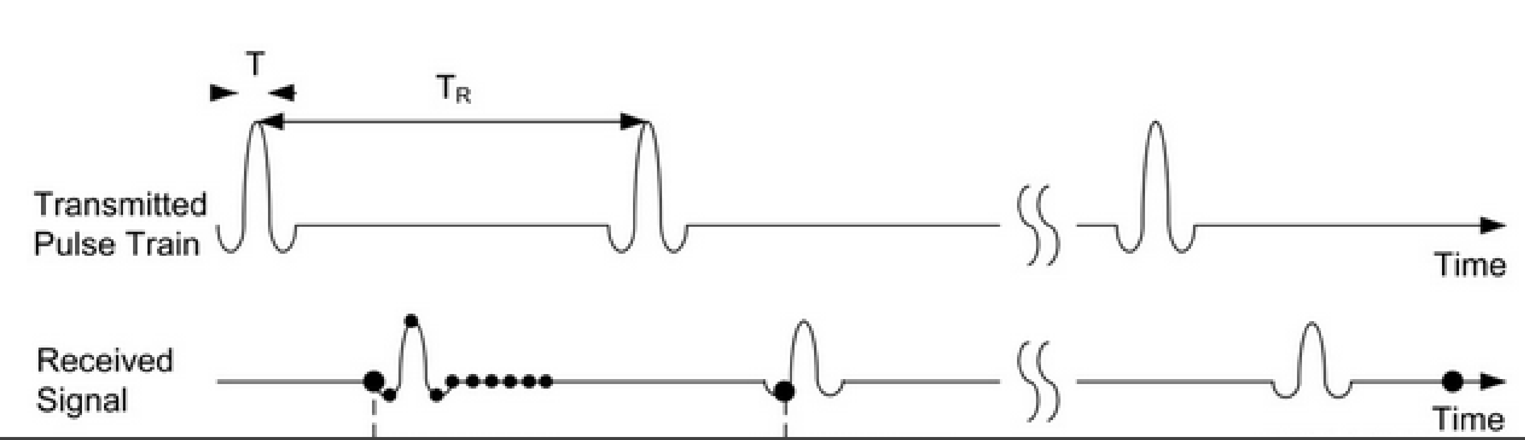
Where is DSP Used

Radars
Sonars
Speech Recognition
Image Processing
Channel Equalization
Medical Imaging
Speaker Crossovers
Mobile Phones (Speech Coding, Noise Reduction)

Ex: Radar Pulse Compression[1]

Radar: emits a pulse of a specific shape and looks for a reflected pulse that matches that shape

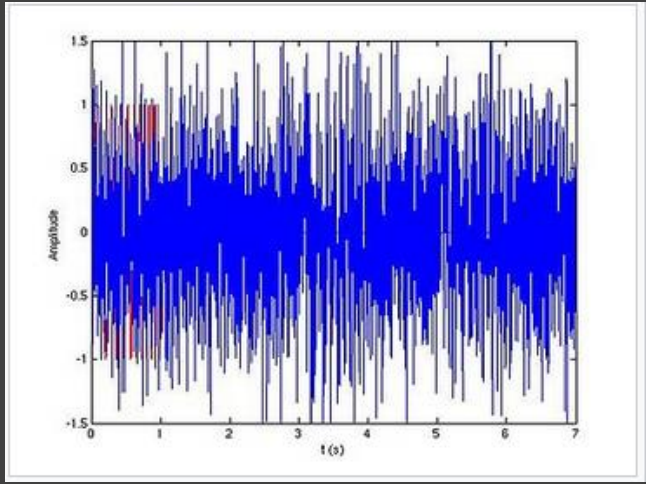




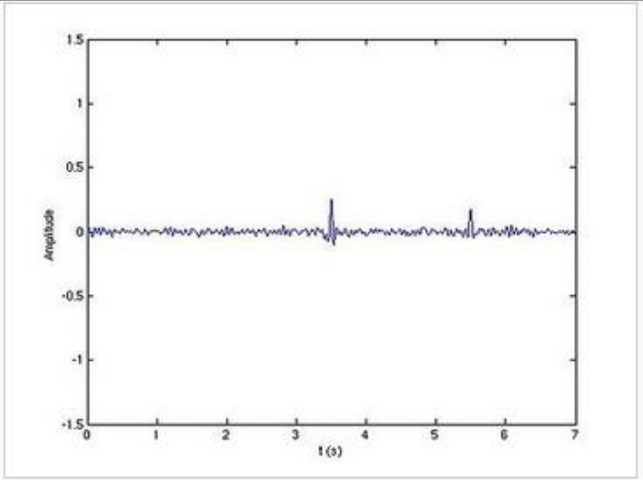
Ex: Radar Pulse Compression[2]

<u>"Matching Filter":</u>
Looks for correlation between received signals and the emitted pulse

Received Signal

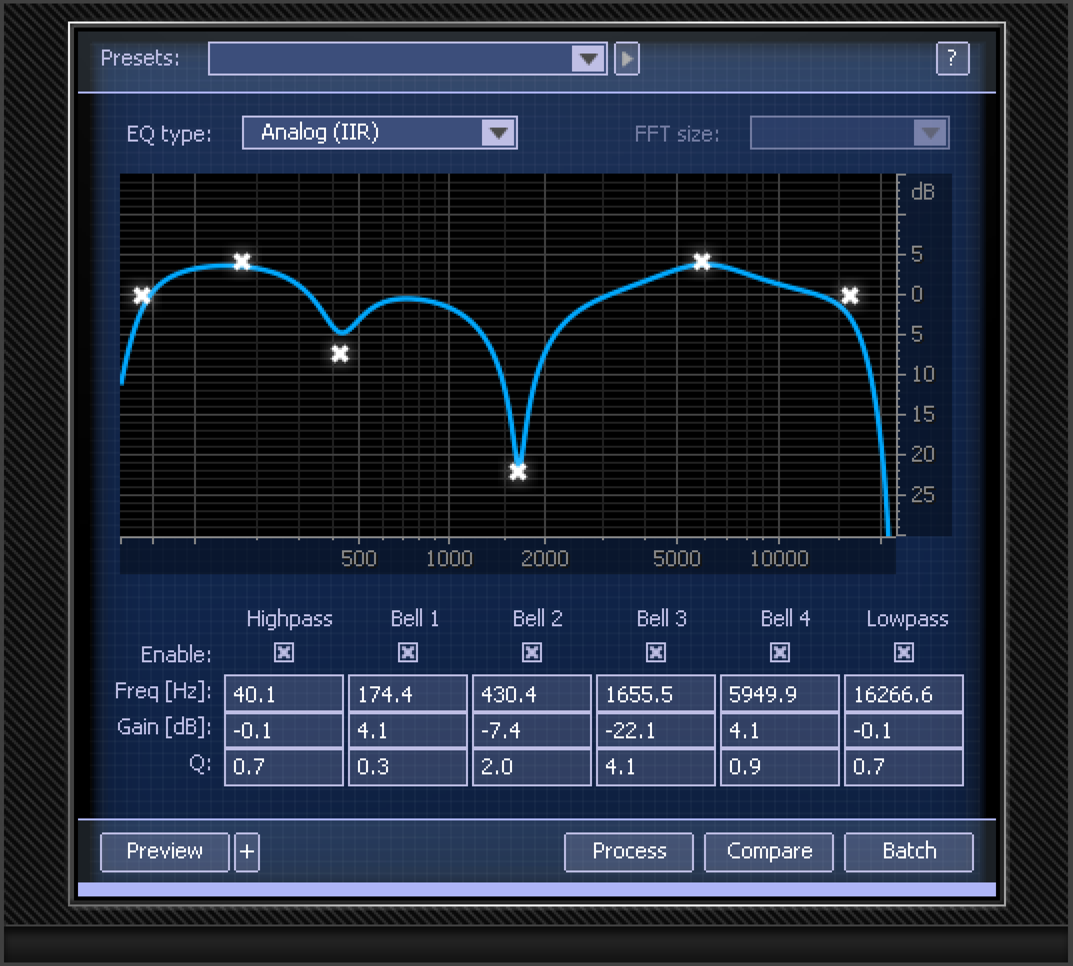


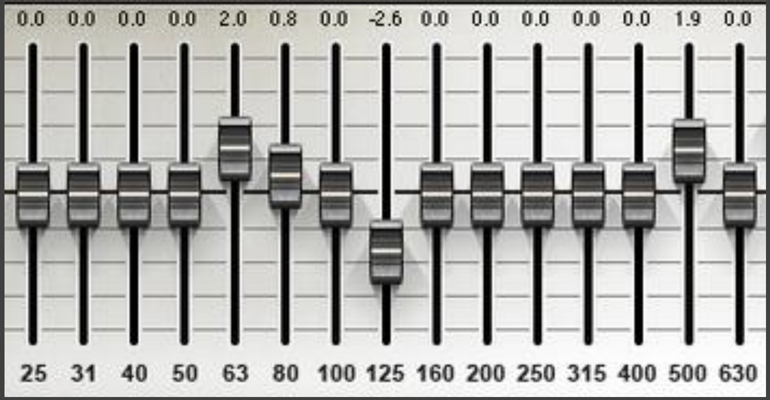
Post-Processed Signal



Ex: Parametric EQ [3]

DSP equalizer can allow the user to tailor frequency response nearly infinitely





How are DSP filters implemented?

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]	x[9]

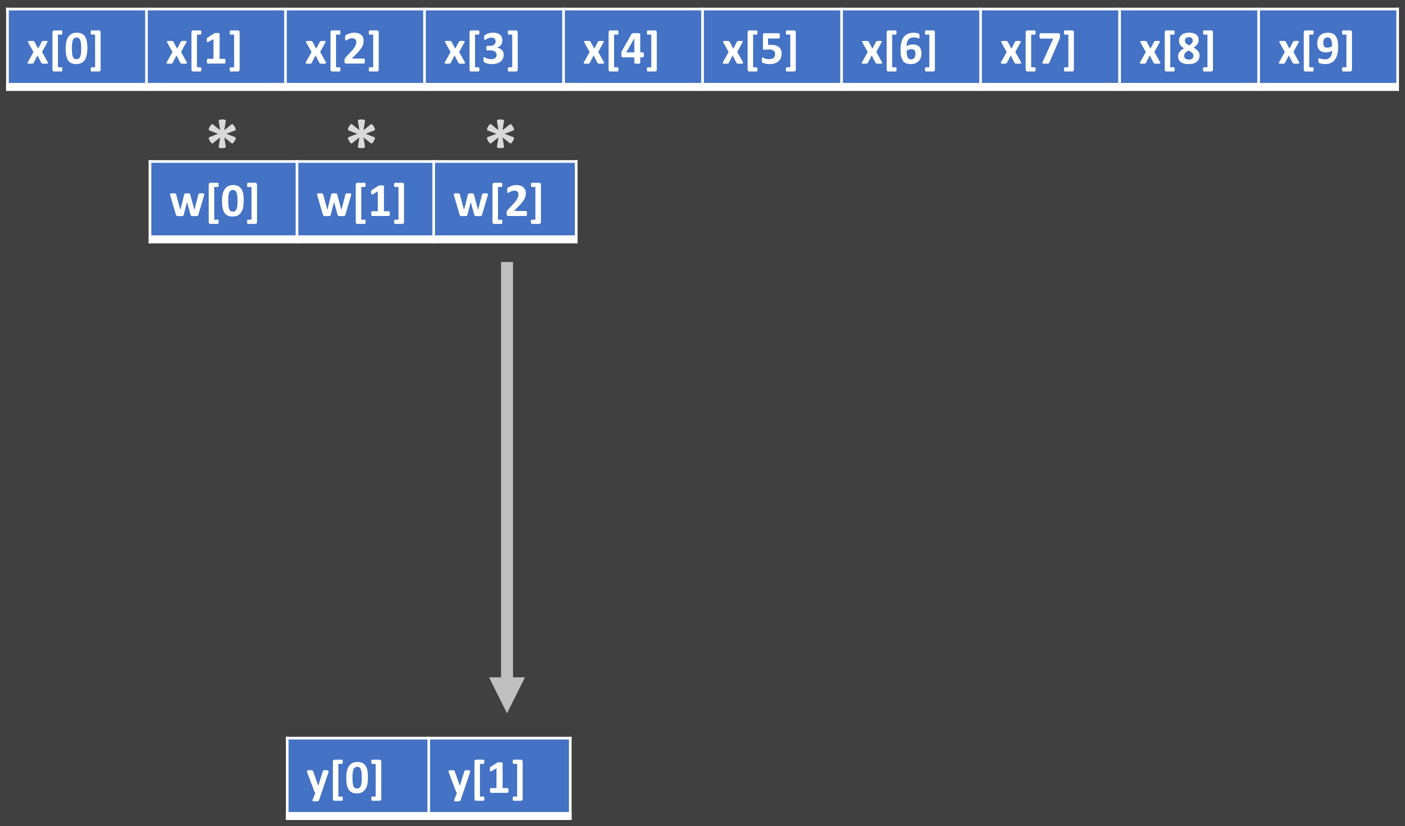
* * *

w[0]	w[1]	w[2]

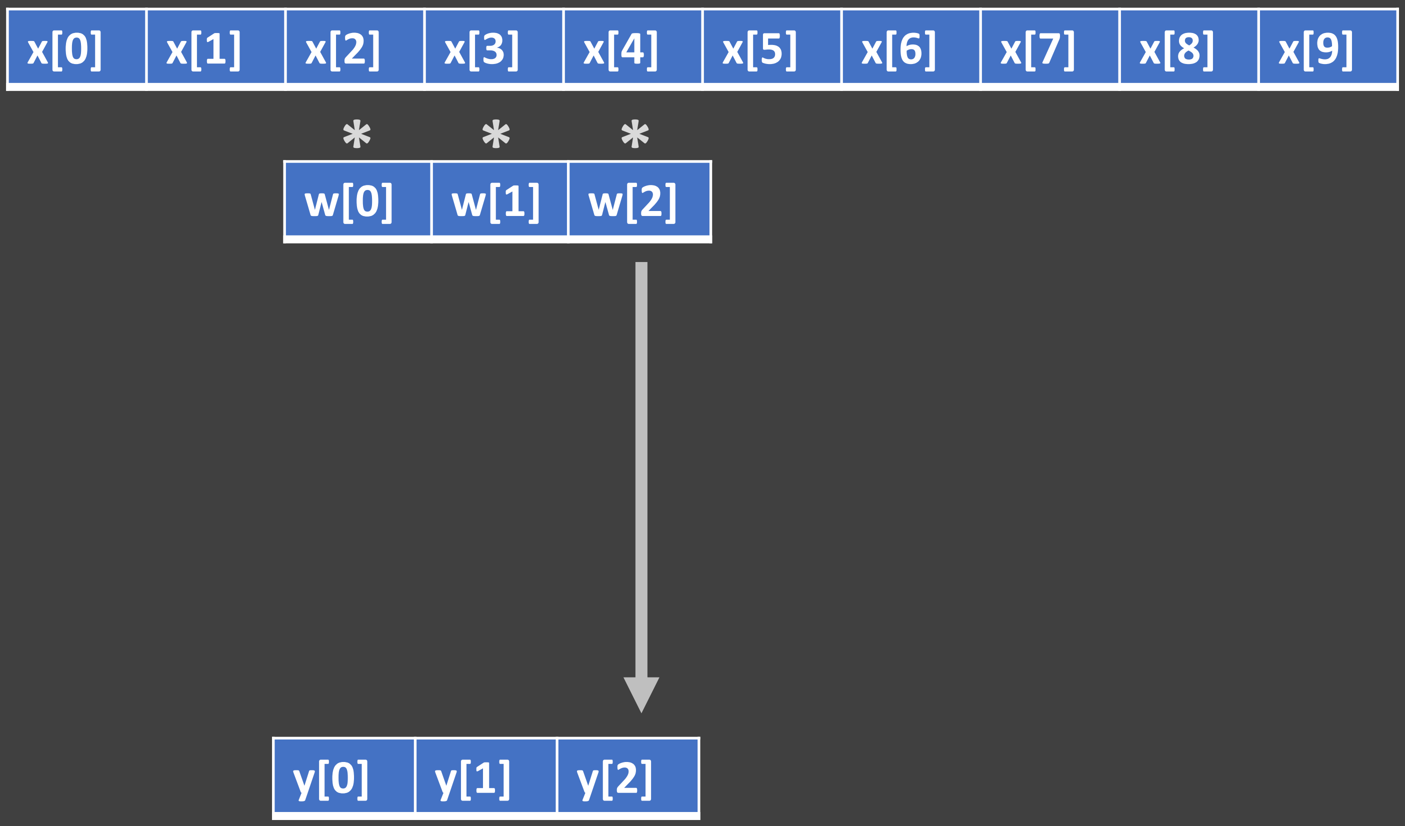
y[0]= x[0]*w[0] + x[1]*w[1] + x[2]*w[2]

y[0]

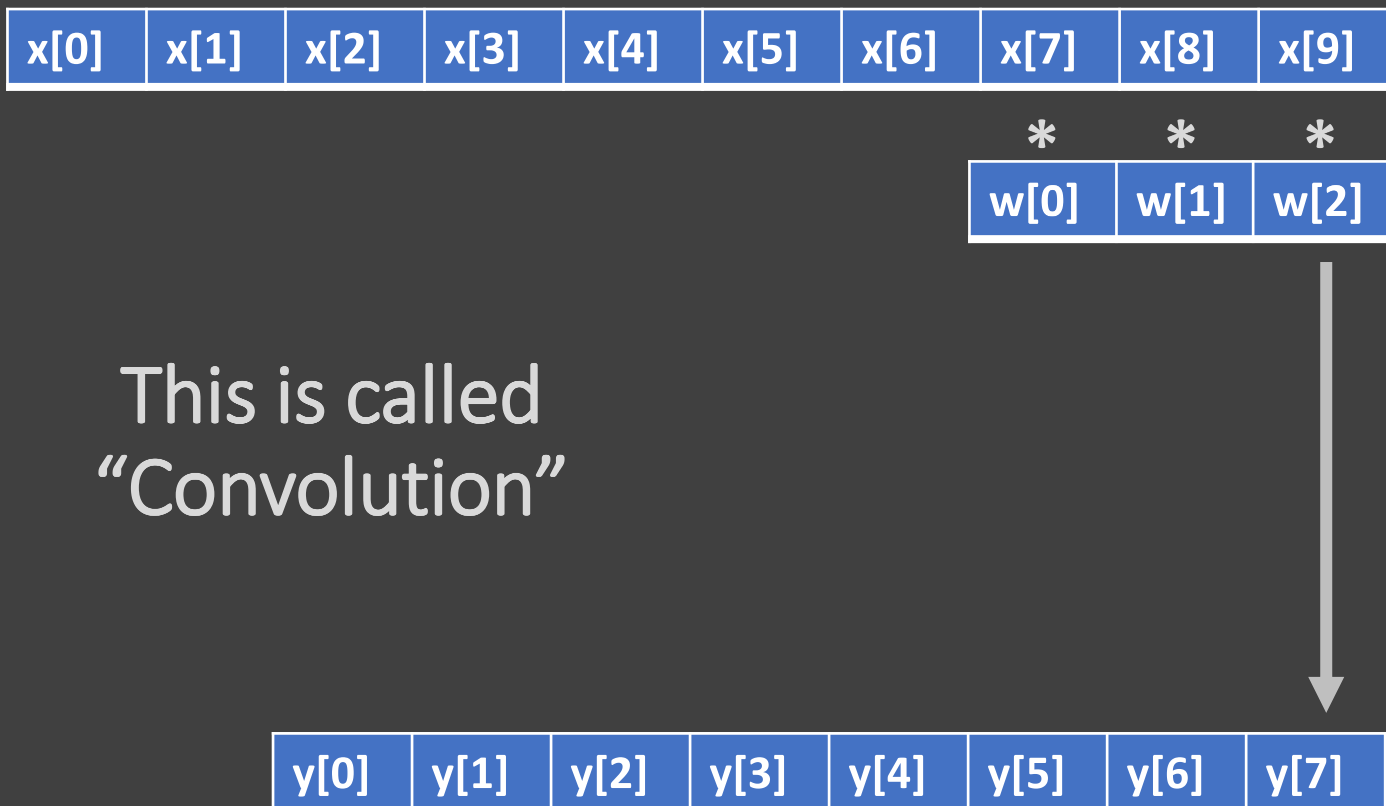
How are DSP filters implemented?



How are DSP filters implemented?

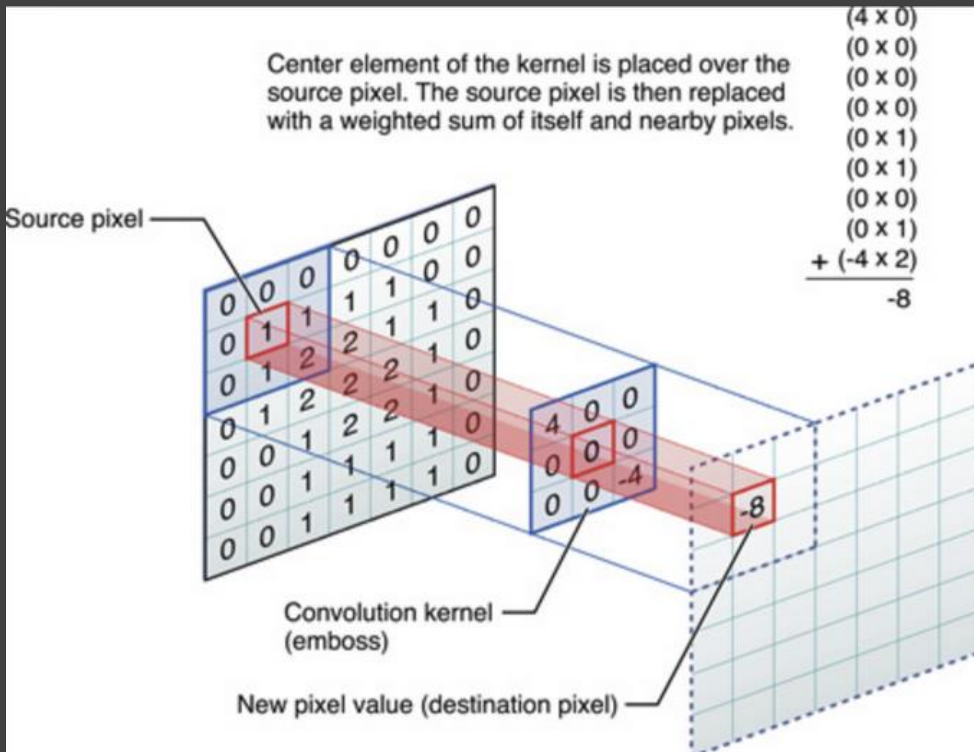


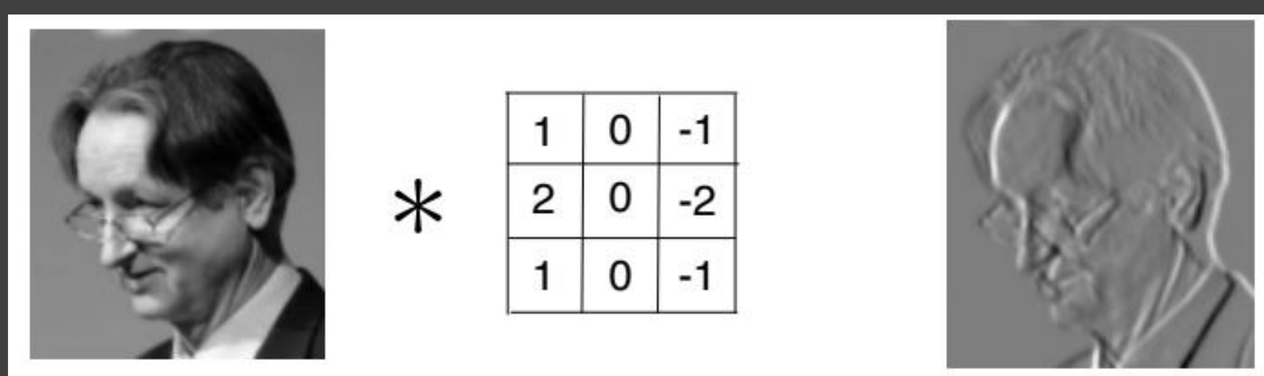
How are DSP filters implemented?



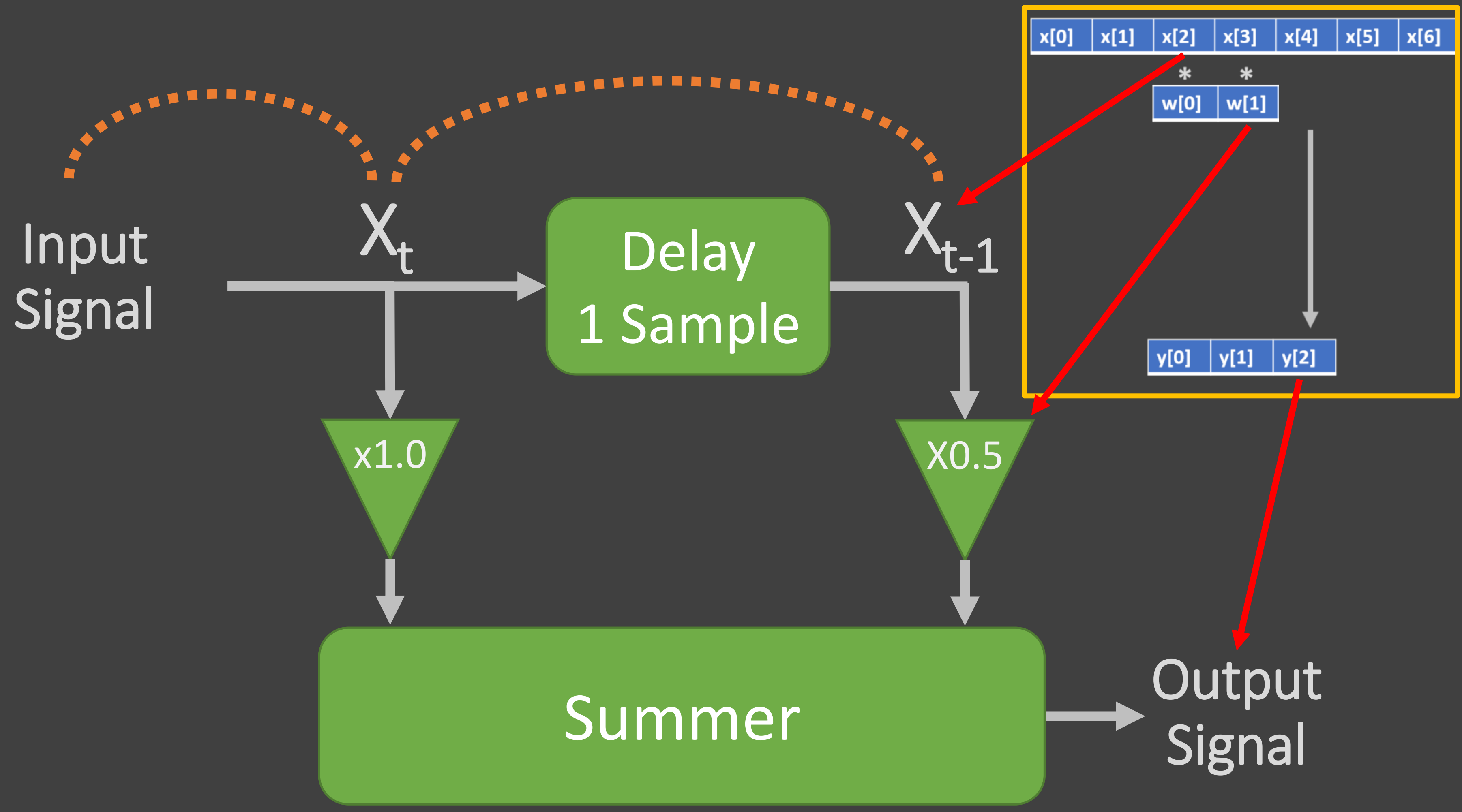
This is called "Convolution"

2D Convoluton Example[4][5]



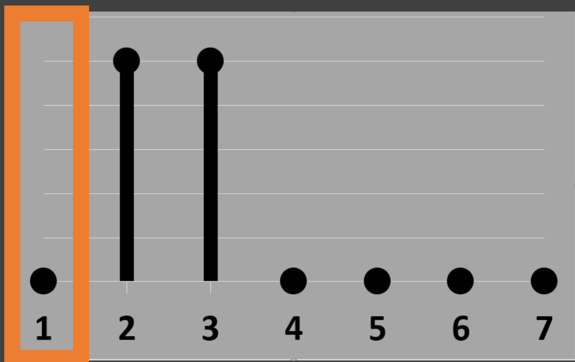


DSP Filter Block Diagram

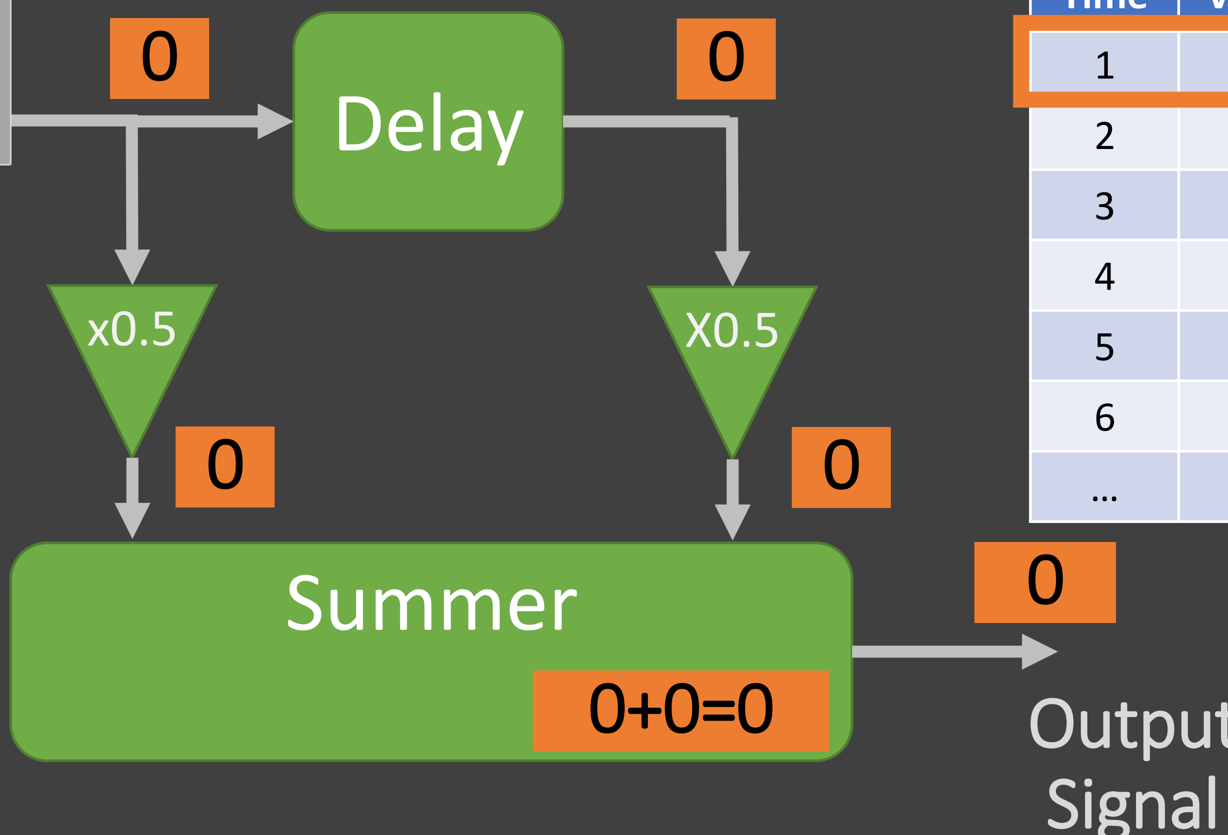


DSP Example

t=1

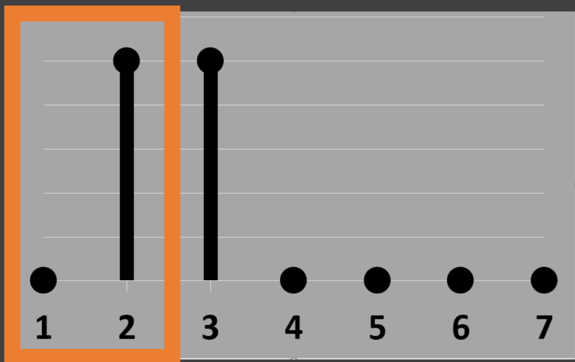


Time	Value
1	0
2	1
3	1
4	0
5	0
6	0
…	…



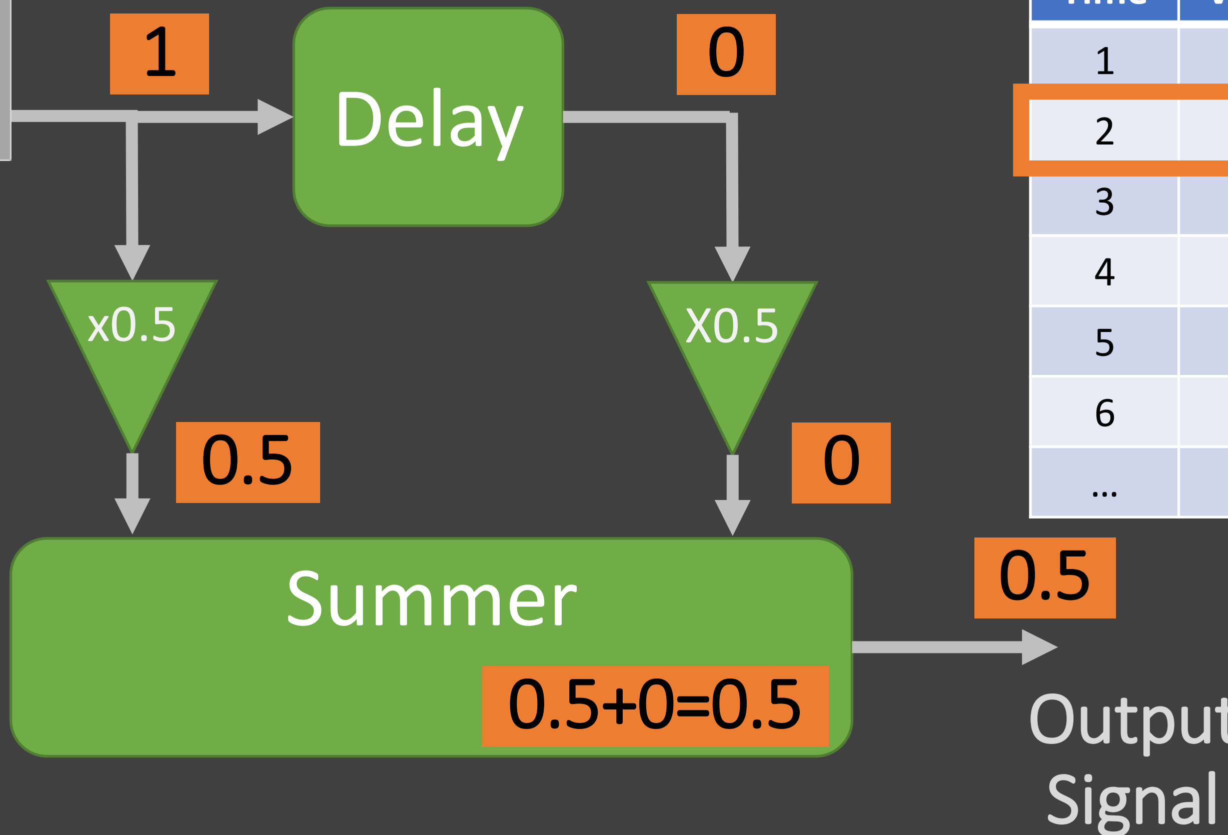
Time	Value
1	0
2	
3	
4	
5	
6	
…	…

DSP Example



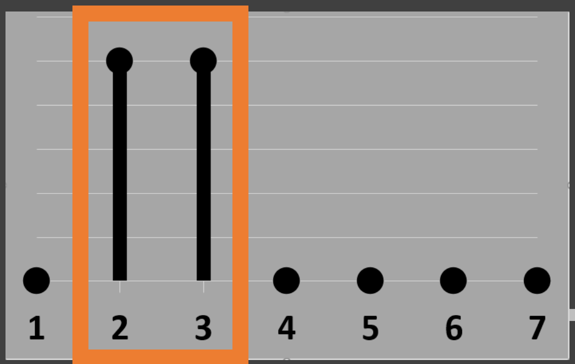
Time	Value
1	0
2	1
3	1
4	0
5	0
6	0
…	…

t=2

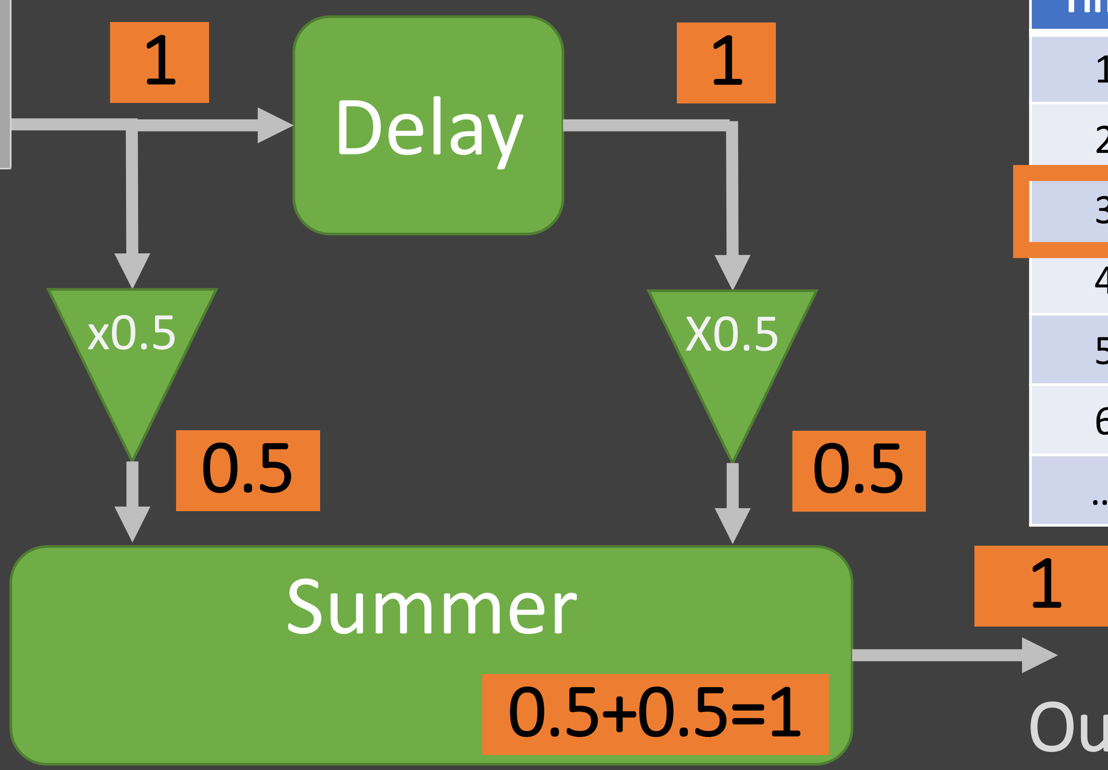


Time	Value
1	0
2	0.5
3	
4	
5	
6	
…	…

DSP Example



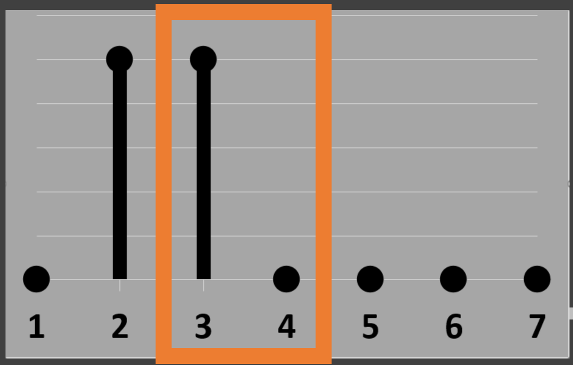
t=3



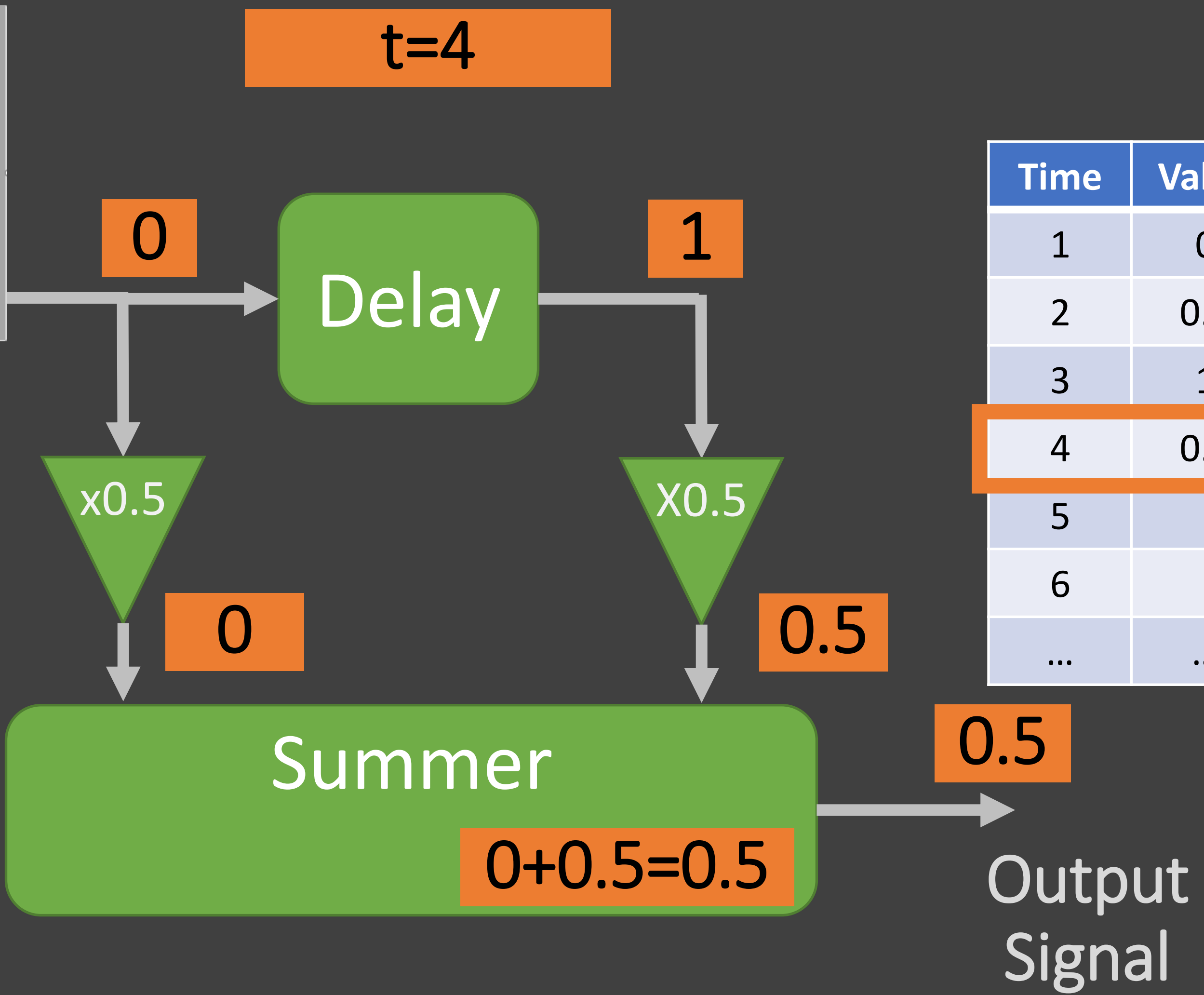
Time	Value
1	0
2	1
3	1
4	0
5	0
6	0
…	…

Time	Value
1	0
2	0.5
3	1
4	
5	
6	
…	…

DSP Example

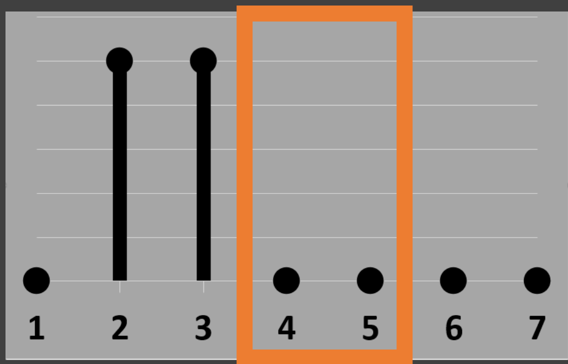


Time	Value
1	0
2	1
3	1
4	0
5	0
6	0
…	…

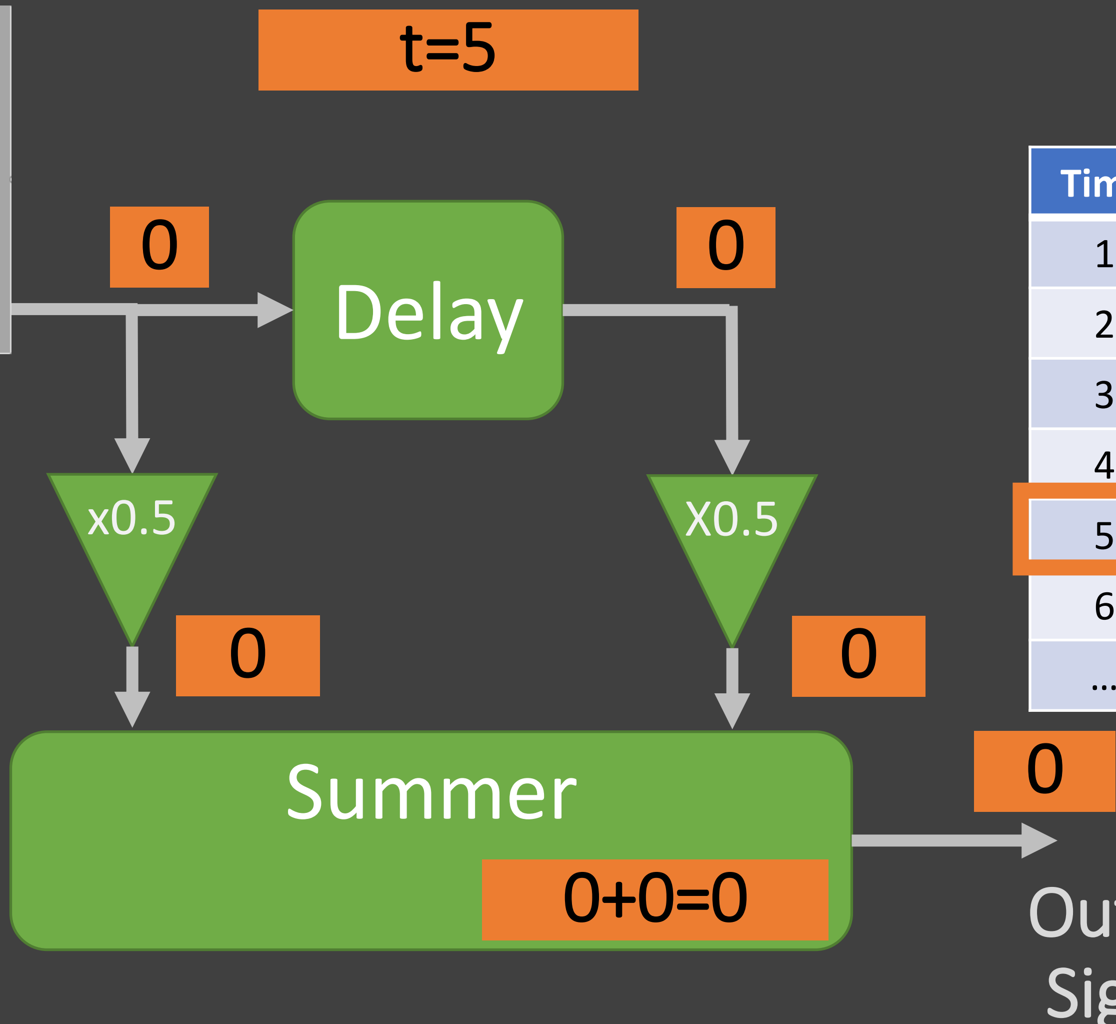


Time	Value
1	0
2	0.5
3	1
4	0.5
5	
6	
…	…

DSP Example

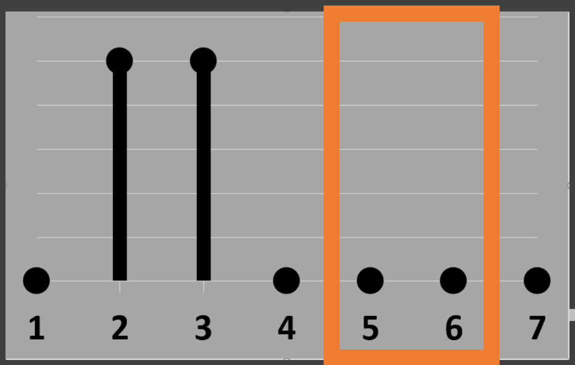


Time	Value
1	0
2	1
3	1
4	0
5	0
6	0
…	…



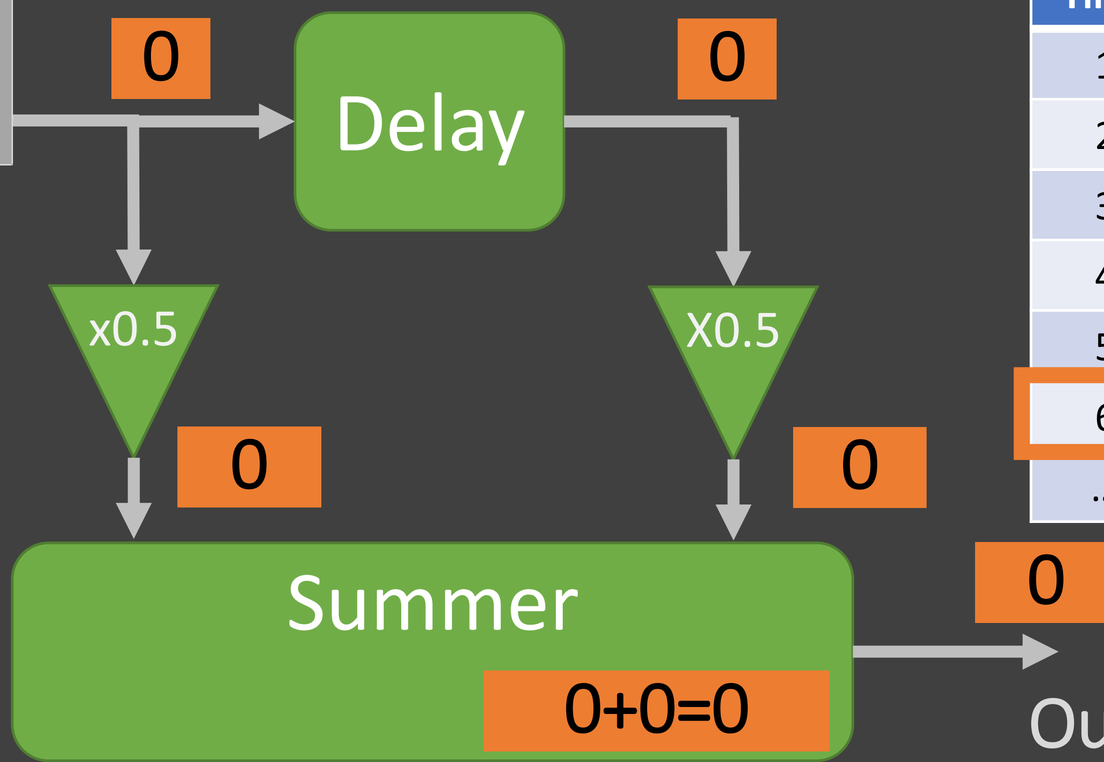
Time	Value
1	0
2	0.5
3	1
4	0.5
5	0
6	
…	…

DSP Example



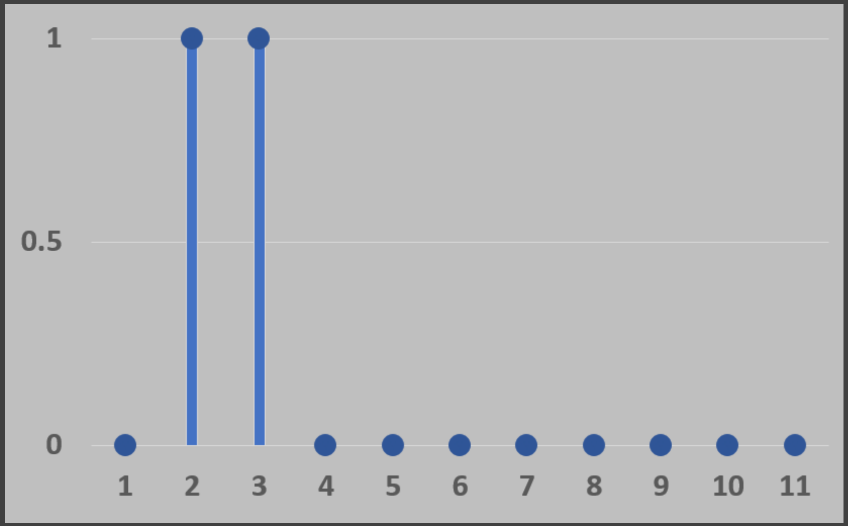
Time	Value
1	0
2	1
3	1
4	0
5	0
6	0
…	…

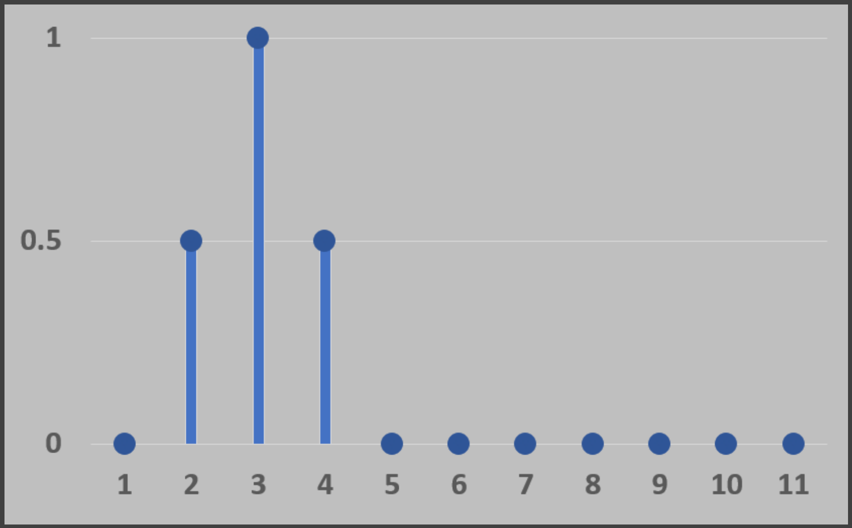
t=5



Time	Value
1	0
2	0.5
3	1
4	0.5
5	0
6	0
…	…

DSP Example Result





Fancy Maths – Please Ignore!

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

$$H(z) = \frac{1}{2}\left(\frac{z+1}{z}\right)$$

$$H(e^{j\omega}) = \frac{1}{2} + \frac{1}{2} e^{-j\omega}$$

$$H(e^{j\omega}) = \frac{1}{2} e^{-\frac{1}{2}j\omega}\left(e^{\frac{1}{2}j\omega} + e^{-\frac{1}{2}j\omega}\right)$$

$$H(e^{j\omega}) = \frac{1}{2} e^{-\frac{1}{2}j\omega} \cos\left(\frac{1}{2}\omega\right)$$

$$\left|H\left(e^{j\omega}\right)\right| = \cos\left(\frac{1}{2}\omega\right)$$

Frequency Domain Response

$$\Phi = e^{-\frac{1}{2}j\omega}$$

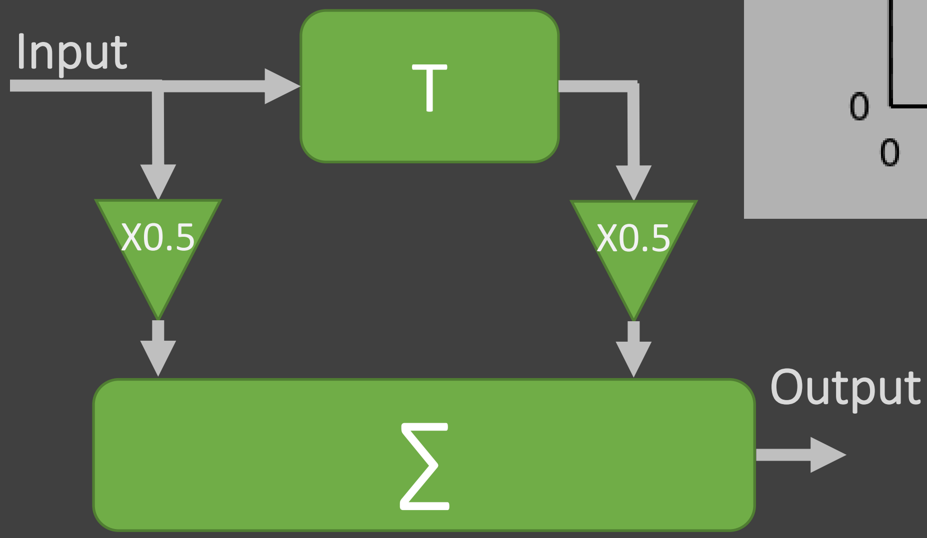
Phase Response

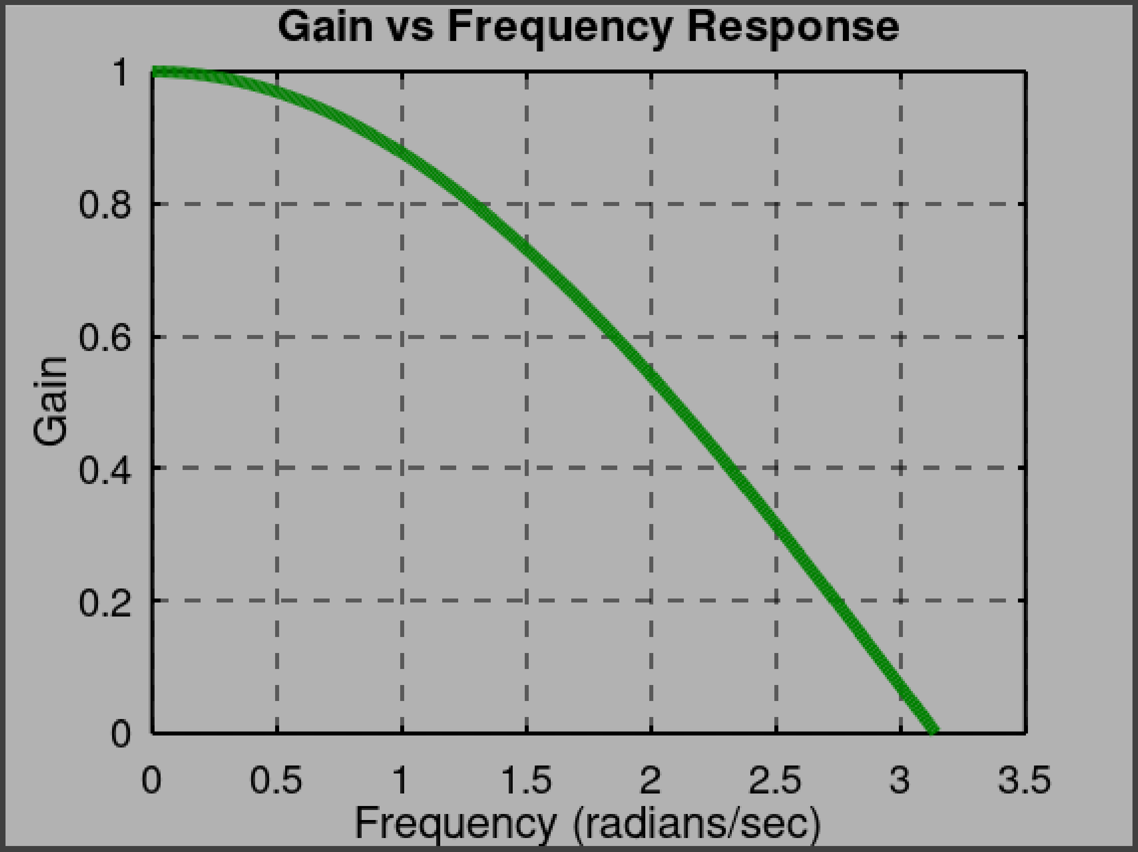
$$\frac{d\Phi(\omega)}{\omega} = \tau g, \ \tau = \frac{1}{2}$$

Group Delay Response

Frequency Response Plot

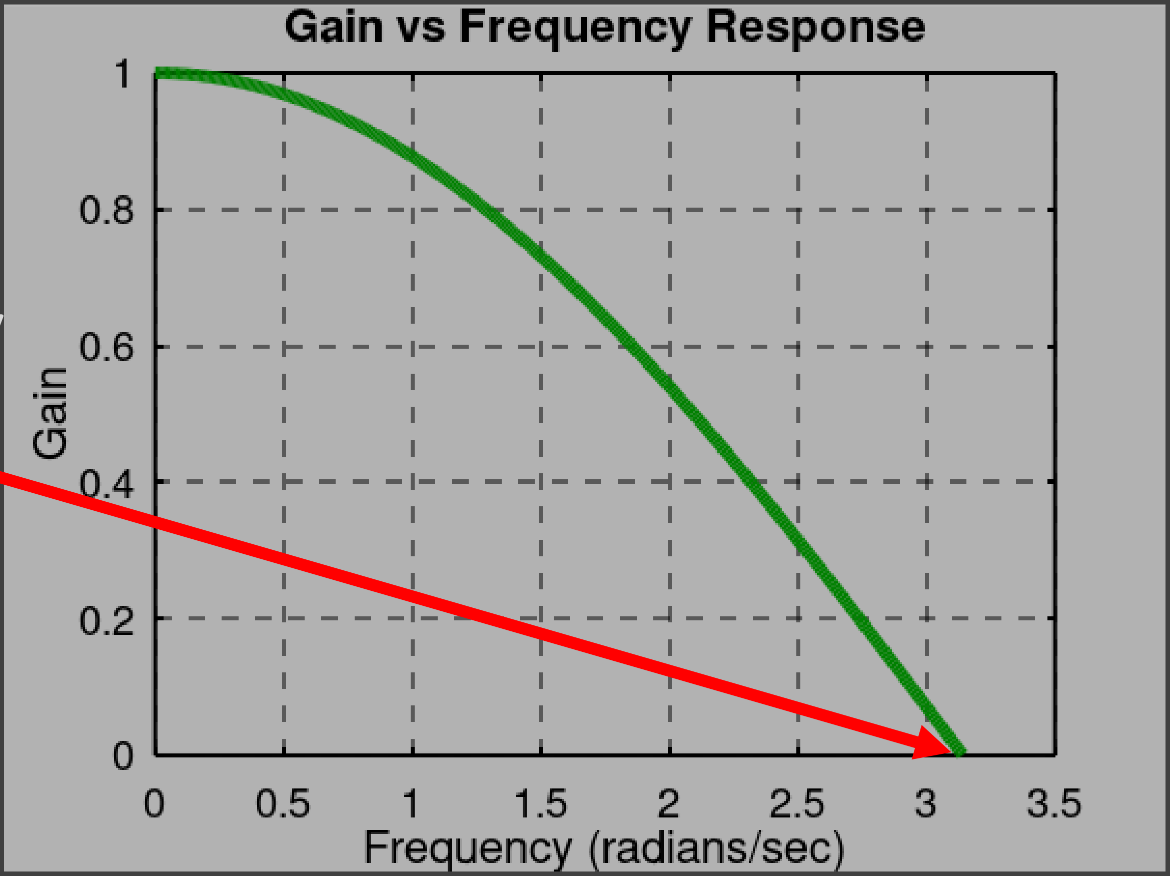
$$\left|H\left(e^{j\omega}\right)\right| = cos\left(\frac{1}{2}\omega\right)$$





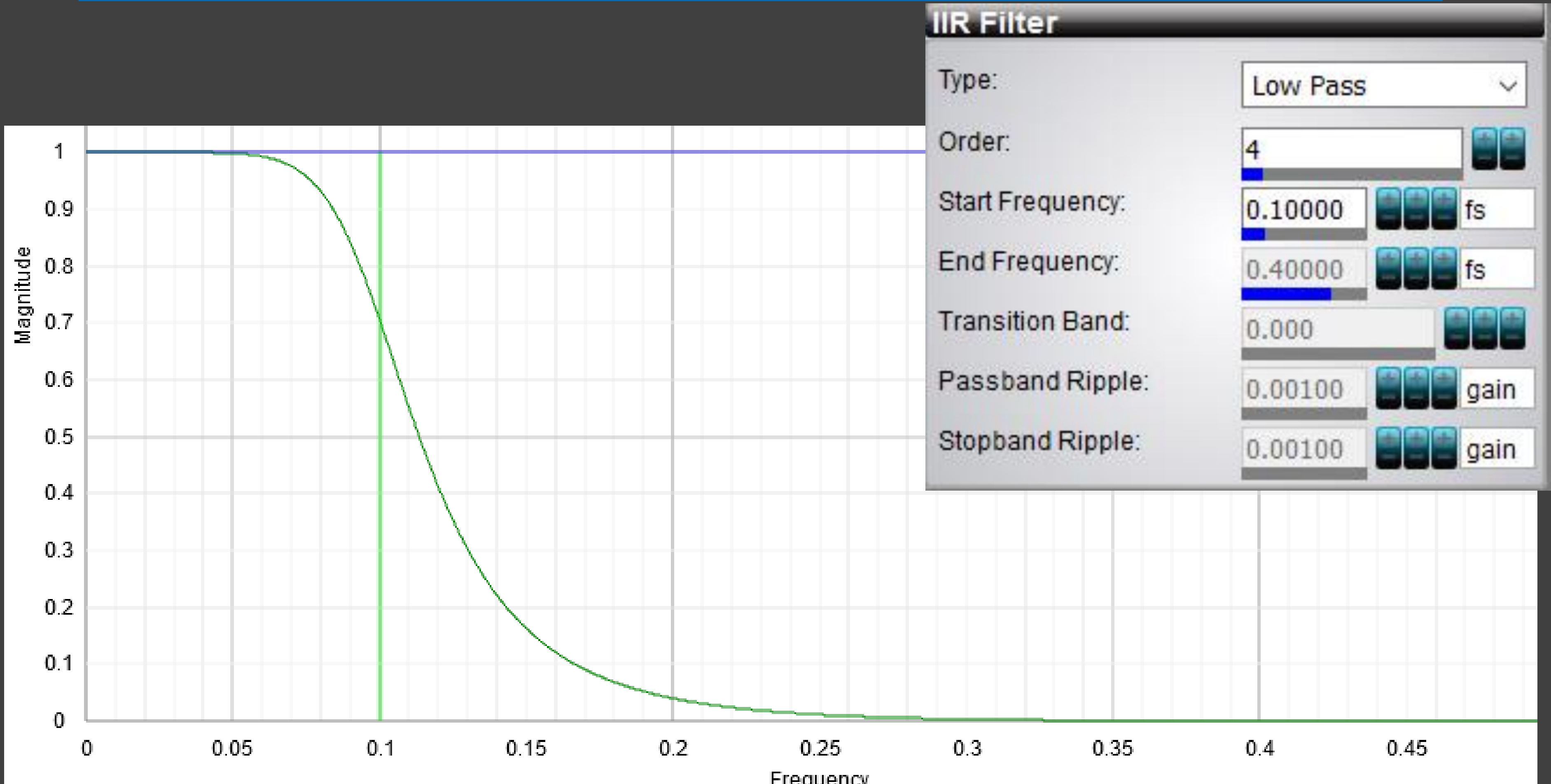
Normalized Frequency

π = Nyquist Frequency
= ½ Sample Rate



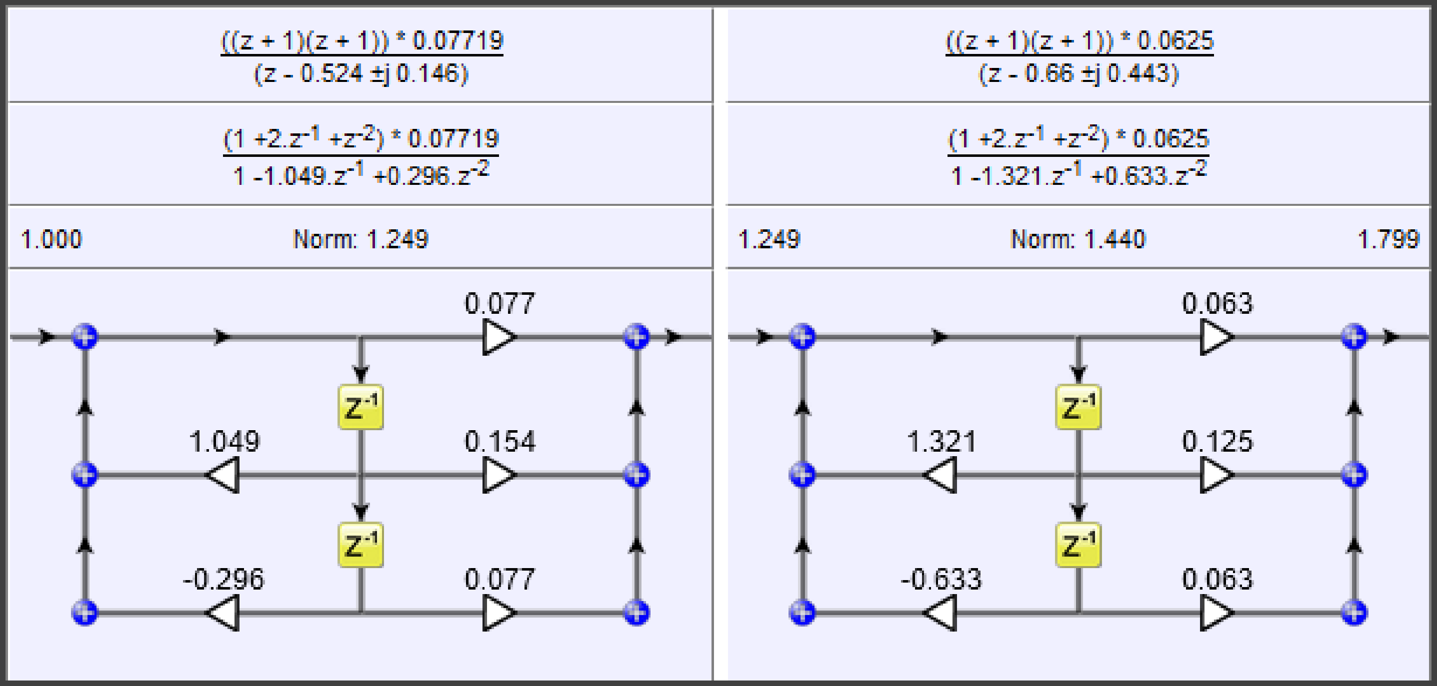
If we sampled at 10kHz
Then π = 5kHz

Practical – Use Calculator[6]



Set filter properties

Practical – Use Calculator[6]



Get block diagram

Practical – Use Calculator[6]

```
#ifndef FILTER1_H_ // Include guards
#define FILTER1_H_

static const int filter1_numStages = 2;
static const int filter1_coefficientLength = 10;
extern float filter1_coefficients[10];

typedef struct
{
	float state[8];
	float output;
} filter1Type;

typedef struct
{
	float *pInput;
	float *pOutput;
	float *pState;
	float *pCoefficients;
	short count;
} filter1_executionState;


filter1Type *filter1_create( void );
void filter1_destroy( filter1Type *pObject );
 void filter1_init( filter1Type * pThis );
 void filter1_reset( filter1Type * pThis );
#define filter1_writeInput( pThis, input )  \
	filter1_filterBlock( pThis, &(input), &(pThis)->output, 1 );

#define filter1_readOutput( pThis )  \
	(pThis)->output

 int filter1_filterBlock( filter1Type * pThis, float * pInput, float * pOutput, unsigned int count );
```

Copy generated code

DSP Microprocessors[7]

Fast Multiply-Accumulate

Oriented towards fast execution of multiply/add of a series of filter coefficients and sample data points

SHARC DSP:

- One clock cycle: multiply, add, 2 data moves, update 2 circular buffer pointers, loop control
- A 100 tap FIR filter is executed in 105-110 cycles

Traditional CPU, several thousand cycles

DSP Microprocessors[7]

Circular Buffers

Circular Buffer
- Holds sampled data
- New data point rolls in
- Oldest point rolls out
- Extremely fast updates

Address	Sample Value	Time
10000	0.0544	x[n-4]
10001	0.1827	x[n-3]
10002	0.3488	x[n-2]
10003	0.8447	x[n-1]
10004	0.3712	x[n]
10005	0.3947	x[n-7]
10006	0.7640	x[n-6]
10007	0.6671	x[n-5]

Newest

Oldest

DSP Microprocessors[7]

Extended Precision Accumulator

Typical 16 processor -> 16 bit accumulator

A 10 tap filter means 10 adds. To prevent overflow we scale the numbers by 1/10.

Now the signal strength has been reduced by 90%

Worsens signal to noise ratio dramatically for filters with large operations

A DSP processor would have a 32 – 40 bit accumulator

DSP Microprocessors

Floating vs Fixed Point

ADSP-21161: Ways to multiply 2 numbers

Fixed	Float
Rn = Rx * Ry	Fn = Fx * Fy
MRF = Rx * Ry	
MRB = Rx * Ry	
Rn = MRF + Rx * Ry	
Rn = MRB + Rx * Ry	
MRF = MRF + Rx * Ry	
MRB = MRB + Rx * Ry	
Rn = MRF – Rx * Ry	
Rn = MRB – Rx * Ry	
MRF = MRF – Rx * Ry	
MRB = MRB – Rx * Ry	
Rn = SAT MRF	
Rn = SAT MRB	
MRF = SAT MRF	
MRB = SAT MRB	
Rn = RND MRF	
Rn = RND MRB	
MRF = RND MRF	
MRB = RND MRB	
MRF = 0	
MRB = 0	
MRxF = Rn	
MRxB = Rn	
Rn = MRxF	
Rn = MRxB	

Fixed -> cheaper
Floating -> better and easier

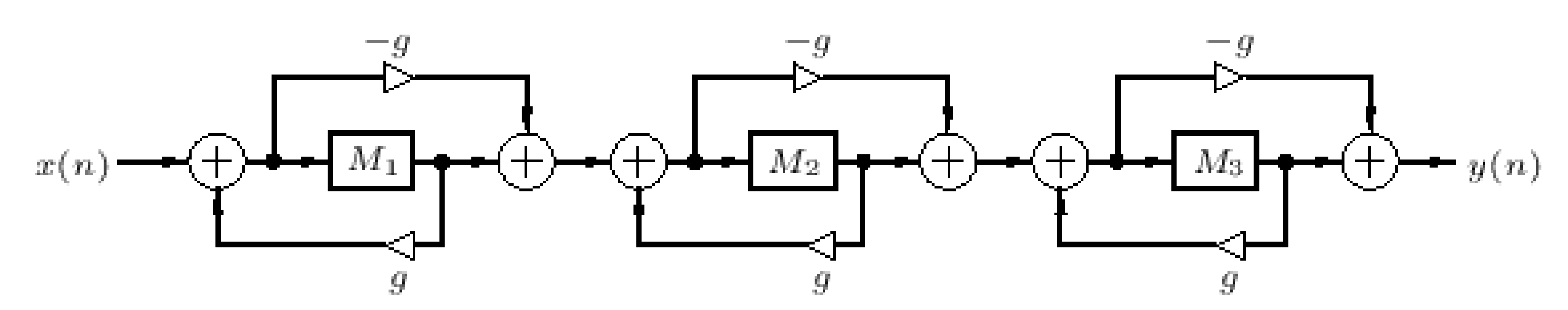
Guitar Effects Processor

Strymon BigSky effects processor

- Analog Devices SHARC running at 366MHz
- Samples at 96,000 samples/second
- 3,800 operations per sample (e.g. multiple/add)

Artificial Reverb[8]

Typically use “Schroeder Allpass Sections”



g is less than 1 (typically ~0.7)

Computational Loads[9]

5G Application:

- CEVA-XC Gen 4: 1.8GHz, 7nm, 1.6 teraflops/sec

Audio: hi-fidelity, looking for 44kHz:

		Max Sample Rate (Hz) for FFT+IFFT with 50% Overlap (Bigger is Better)					
Inputs		Generic C FFT		CMSIS FFT			
N	Data	Arduino Uno	Arduino M0	Teensy 3.2	Teensy 3.5	Teensy 3.6	FRDM-K66F
128	Int16	*	20,447	228,997	297,785	517,464	457,143
128	Int32	*	7,545	105,367	138,528	234,432	198,758
128	Float32	*	2,599	21,042	218,669	336,984	304,762
* Insufficient RAM							

References

[1] https://www.intechopen.com/books/radar-technology/radar-performance-of-ultra-wideband-waveforms
[2] https://en.wikipedia.org/wiki/Pulse_compression
[3] http://help.izotope.com/docs/rx/pages/reference_parametricequalizer.htm
[4] https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/convolution.html
[5] https://medium.com/@james.chain1990/study-3d-convolutional-neural-networks-for-human-action-recognition-7eaeb9a0ec00
[6] https://www.micromodeler.com/dsp/
[7] http://www.dspguide.com/ch28/3.htm
[8] https://ccrma.stanford.edu/~jos/pasp/Schroeder_Allpass_Sections.html
[9] http://openaudio.blogspot.com/2016/10/benchmarking-teensy-36-is-fast.html