Spiral 3-2

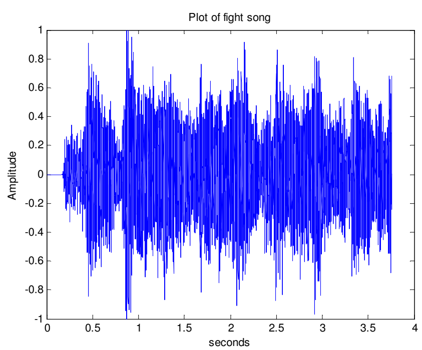
Signal & Image Processing

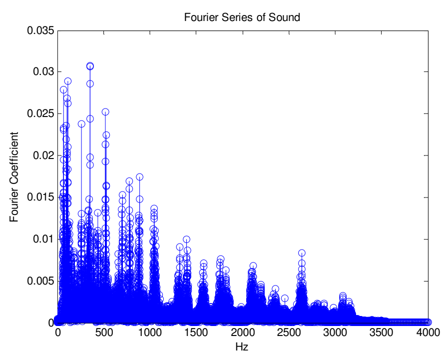
Finding and exploiting patterns in raw data

SIGNAL AND IMAGE PROCESSING

Example

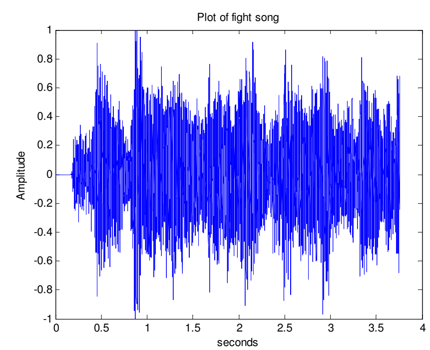
- Take USC fight song and remove high frequency audio from the song (i.e. lower the “treble”)
- We can view the song as samples over time or by taking the Fourier transform, we can see the component frequencies (i.e. the frequency domain representation)

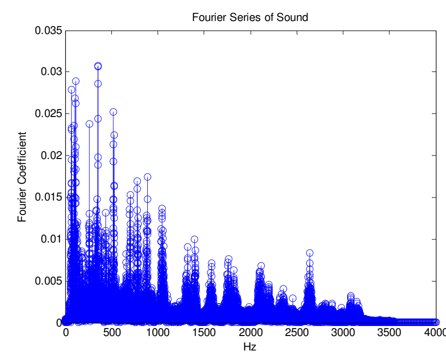




Low Pass Filter

- We would like to remove the high frequency components

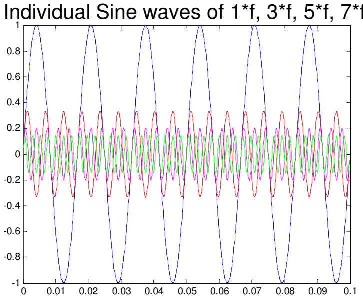




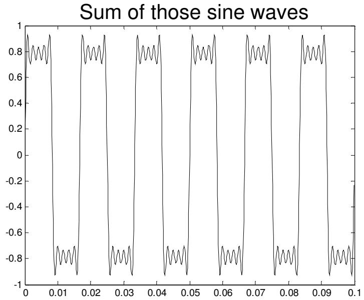
Tangent – Frequency Domain

- Fourier theory says any signal can be represented as sum of different frequency sine waves

Individual Sine waves of 1*f, 3*f, 5*f, 7*f



Sum of those sine waves

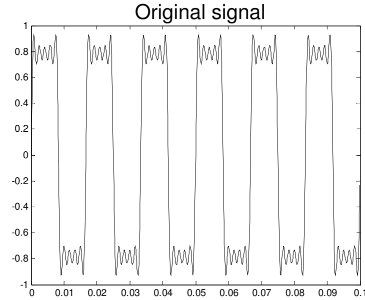


Fourier Composition – By summing different sine waves we can form a square wave or any other signal

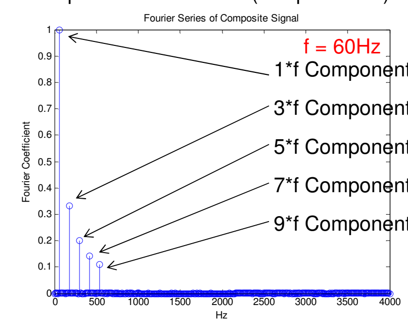
Fourier Decomposition

- Fourier theory says we can also find the sine wave components given the original signal

Original signal



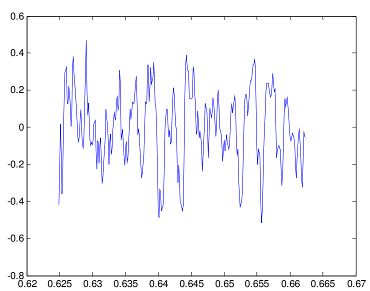
Component Sine Waves (Freq. Domain)



Fourier Composition – By summing different sine waves we can form a square wave or any other signal

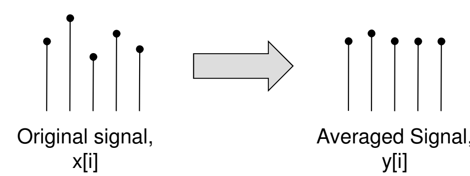
Designing a Low Pass Filter

- Below is a zoomed view
- Removing high frequency components (parts of the signal that change rapidly) means smoothing the signal or finding its basic curve and not the bumpiness
- To do this, for each sample, make it equal to the average of neighboring pixels.



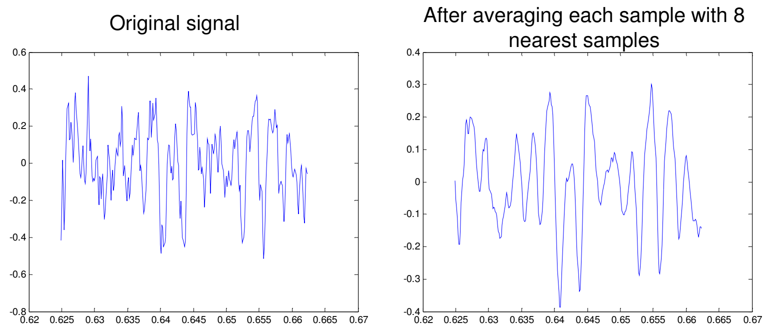
Moving Average

- By making each sample equal to the average of itself plus neighboring samples we tend to smooth the signal



Averaged Signal

- Averaging smoothes the waveform and effectively filters out high-frequency components

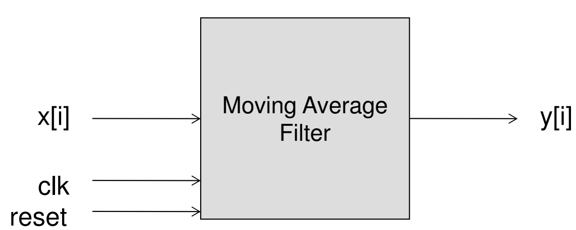


8-tap Moving Average Filter

- Assume each sample is the average of 8 surrounding samples, we can describe the output as:
  $y[i] = \Sigma_{k=0 \text{ to } 7} (1/8) * x[i-k]$
- Example:
  - $y[7] = 1/8*x[7] + 1/8*x[6] + … + 1/8*x[0]$
  - $y[8] = 1/8*x[8] + 1/8*x[7] + … + 1/8*x[1]$
- If we want a weighted average rather than pure average we can generalize from 1/8 to some weight coefficient: w_k
  $y[i] = \Sigma_{k=0 \text{ to } 7} w_k*x[i-k]$

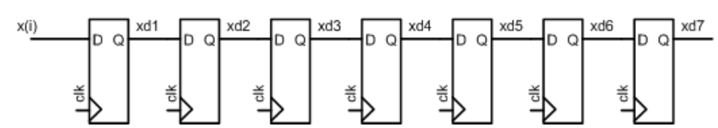
Digital Implementation

- The system we want to design gets one sample per clock and produces one output sample per clock



Storing Last 8 Samples

- Since we only get one sample a clock, but need to use the last 8 samples to do our average, we need to save the last 8 samples
  - To store values we use registers
  - Chain together several registers
    - xd1 = x[i] delayed by 1 clock
    - xd2 = x[i] delayed by 2 clocks



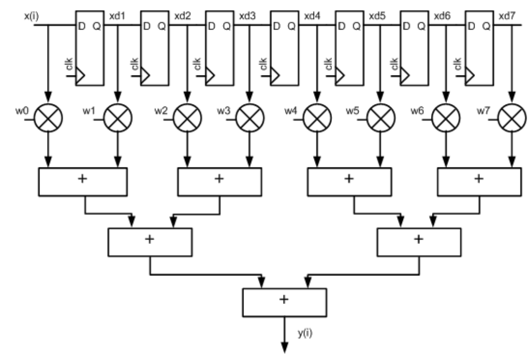
Time Space Diagram

Clock	X[i]	Xd1	Xd2	Xd3	Xd4	Xd5	Xd6	Xd7
0	X(0)	X(-1)=0	X(-2)=0	X(-3)=0	X(-4)=0	X(-5)=0	X(-6)=0	X(-7)=0
1	X(1)	X(0)	X(-1)=0	X(-2)=0	X(-3)=0	X(-4)=0	X(-5)=0	X(-6)=0
2	X(2)	X(1)	X(0)	X(-1)=0	X(-2)=0	X(-3)=0	X(-4)=0	X(-5)=0
3	X(3)	X(2)	X(1)	X(0)	X(-1)=0	X(-2)=0	X(-3)=0	X(-4)=0
4	X(4)	X(3)	X(2)	X(1)	X(0)	X(-1)=0	X(-2)=0	X(-3)=0
5	X(5)	X(4)	X(3)	X(2)	X(1)	X(0)	X(-1)=0	X(-2)=0
6	X(6)	X(5)	X(4)	X(3)	X(2)	X(1)	X(0)	X(-1)=0
7	X(7)	X(6)	X(5)	X(4)	X(3)	X(2)	X(1)	X(0)
8	X(8)	X(7)	X(6)	X(5)	X(4)	X(3)	X(2)	X(1)
9	X(9)	X(8)	X(7)	X(6)	X(5)	X(4)	X(3)	X(2)

Samples x[i] where i < 0 (negative indices) are equal to 0 since there register will be reset (cleared) at clock 0

Averaging the Samples

- Multiple each sample by the appropriate weight (in this case each $w_k = 1/8$)
- Add up all values



HW/SW Design (System on Chip)

CONCEPTS

Another Example: Image Compression

- Images are just 2-D arrays (matrices) of numbers
- Each number corresponds to the color or a pixel in that location
- Image store those numbers in some way

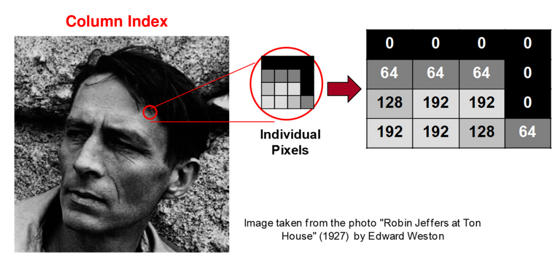


Image taken from the photo "Robin Jeffers at Ton House" (1927) by Edward Weston

Image Compression

129	131	130	133	132	132	130	129	128	130	131	129
130	130	131	129	131	132	131	133	130	129	129	131
132	131	130	132								
134	132	131	132								
133	131										
156	157										
153	155										
154	152										
207	204										
208	205										

Image Compression

129	131	130	133	132	132	130	129	128	130	131	129
130	130	131	129	131	132	131	133	130	129	129	131
132	131	130	132								
134	132	131	132								
133	131										
156	157										
153	155										
154	152										
207	204										
208	205										

1. Break Image into small blocks of pixels

129	131	130	133
130	130	131	129
132	131	130	132
134	132	131	132

→

129	2	1	4
2	1	2	0
3	2	1	3
5	3	2	3

2. Store the difference of each pixel and the upper left (or some other representative pixel)

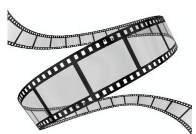
129	2	1	4
2	1	2	0
3	2	1	3
5	3	2	3

→

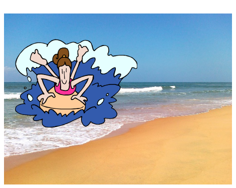
129	2	0	4
2	0	2	0
2	2	0	2
4	2	2	2

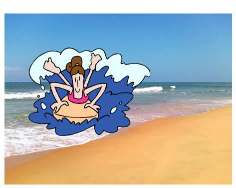
3. We can save more space by rounding numbers to a smaller set of options (i.e. only even # differences)

Video Compression



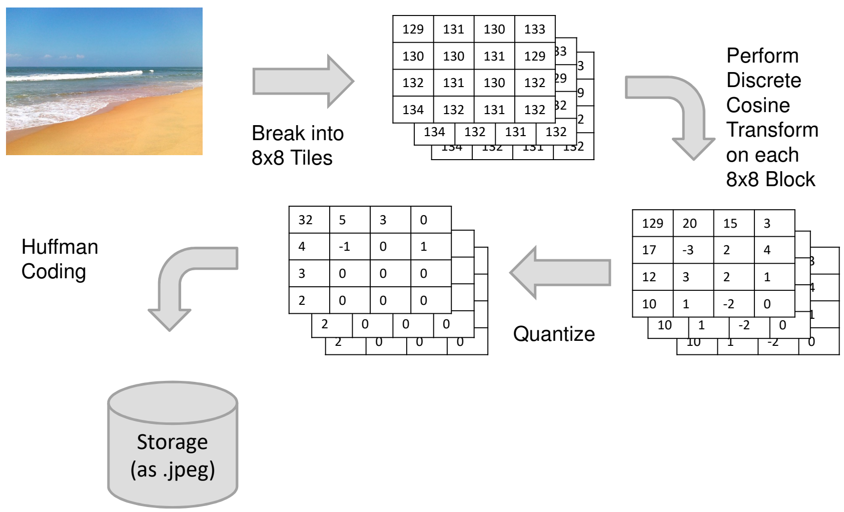
- Video is a sequence of still frames
  - 24-30 frames per second (fps)
- How much difference is expected between frames?
- Idea:
  - Store 1 of every N frames (aka key frame or I-frame), with other N-1 frames being differences from previous or next frame





JPEG

JPEG Conversion Process



Huffman Code

- Compression algorithm
- Variable-length code
  – Each character can be coded with a different number of bits
- Prefix code
  – No two codes start with the same prefix
- Assignment of codes to characters is based on frequency of the code in the message

Huffman Example 1

- "Mississippi"

Huffman Example 2

- "i boo big bruins"

DESIGN EXERCISE

Design Exercise

- Design a system that breaks a 8x8 image into 4 tiles of 4x4 and:
  - Leaves the upper-left pixel of each tile as is
  - Codes the remaining 15 pixels of the tile as relative values based on the upper-left
  - Computes the frequency of each pixel value to prepare for Huffman coding

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
32	33	34	35	36	37	38	39
40							
56	57	58	59	60	61	62	63

Computing Frequencies

- Given an array 16 numbers between 0-255 how could you compute their frequencies in software?

Block Diagram

Addressing

- Given the HW counter that counts 0000-1111 and tile counter 00-11, can you use those 6-bits to form the correct system address?

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
32	33	34	35	36	37	38	39
40							
56	57	58	59	60	61	62	63

System Address

Tile 0 / Tile 1 (top), Tile 2 / Tile 3 (bottom)

0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15
0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15

HW Counters

Verilog Description