



Analog to Digital and Back

Dealing with Analog Signals in a
Digital Environment

Motivation

- Embedded systems are frequently used to take measurements from analog sensors
- They can be used to generate sound as well, either for simple musical instruments or for warnings, etc.
- A basic understanding of both the theory of analog sampling is required in order to correctly design these systems

Example: Seismic data acquisition

- Thumper trucks are used to generate shock waves in the ground
- The waves echo off formations underground. The speed and amplitude of the returning waves can be correlated with the shape and composition of underground formations.
- This information is used to determine the location and extent of natural resources (oil, gas, etc.)



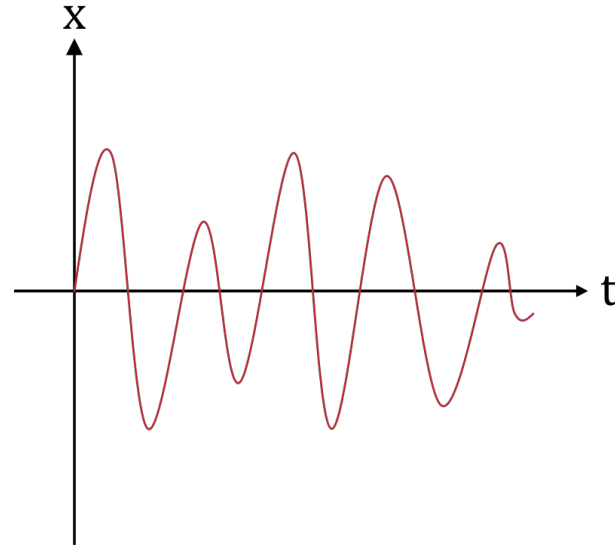
Example: Seismic data acquisition

- Questions:
 - What sampling frequency would be needed to capture the returning wave?
 - How many bits would be required per sample? (the sample word length)



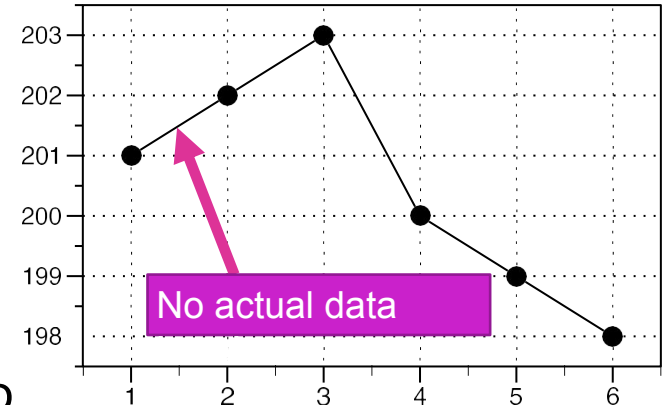
Analog Signals


- Analog signals are *continuous time* signals
- Sources include
 - Microphones
 - Temperature sensors
 - Seismic activity detectors
 - Heart monitors



Digital Signals

- Digital signals are *discrete-time*
 - They consist of individual values
- They *only* exist at those points – there is no information between the points
- The location on the x-axis of a point has no inherent meaning
 - You must know more about how the sequence was created





From Analog to Digital

- Process of conversion is called *sampling or digitization*
- Analog values are measured (sampled) at a predetermined rate
- The rate used is critical, as information is lost if the rate is too low

Nyquist-Shannon Sampling Theorem

Nyquist principle:

If f_h is the highest frequency component of a signal in its *bandwidth of interest*, then the sampling frequency f_s must satisfy

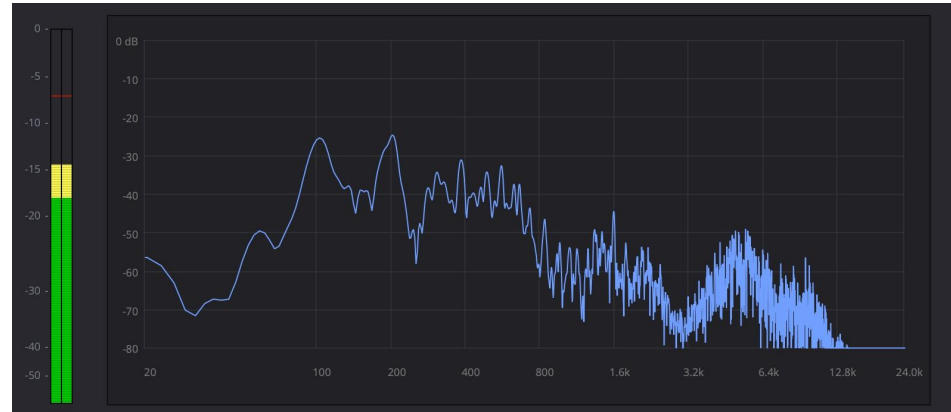
$$f_s \geq 2 \times f_h$$

Nyquist Frequency $f_N = 2 \times f_h$

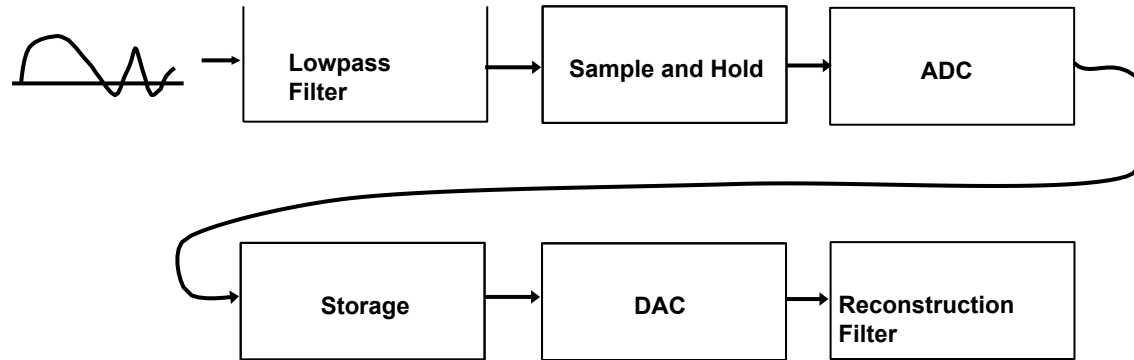
Nyquist Rate $T_N = \frac{1}{f_N}$

Bandwidth

- Bandwidth of a signal is the range of frequencies found in a signal
- Any complex signal (sound, for example) consists of sin waves at many frequencies and phases (Fourier theorem)
- Spectrum analyzers show the relative amplitude of each frequency

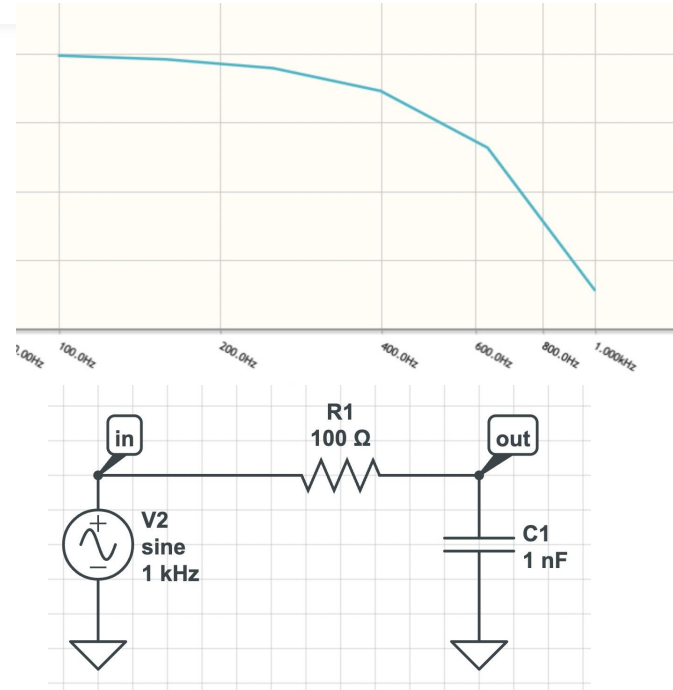


Signal Processing Chain



Low Pass Filter

- Attempts to remove frequencies above the Nyquist limit
- Never perfect
 - Sharper the curve, the more phase distortion at high frequencies



Sample and Hold (S/H)

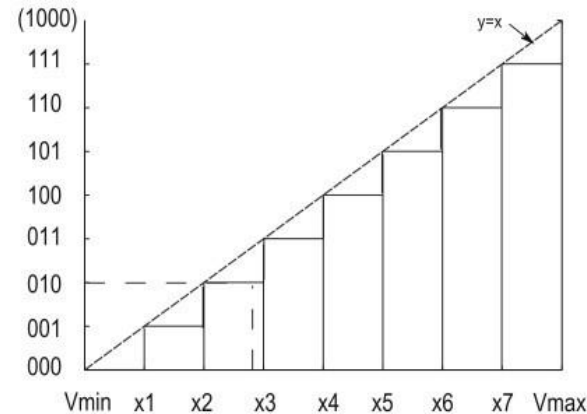
- The incoming signal varies in amplitude
- The ADC takes a certain amount of time to sample the signal
- If the signal changes before the conversion is complete, an inaccurate reading would result
- The S/H holds the signal at a steady level so that the ADC can complete the conversion
- The S/H is an *analog* circuit!

The ADC: Quantizing the Signal

- Each sample must be turned into a binary number
- The number of bits combined with the voltage range determines precision
- Sampling rate determines the highest frequency that can be represented
- Number of bits in quantization affects the signal to noise ratio of the conversion
 - Higher number of bits \square more accurate measurement

Quantization (2)

- Each voltage measured is rounded to the nearest available binary number
- The difference between the measured value and the actual value is called quantization error
- Quantization error is referred to as quantization noise, and can be audible in sound recordings



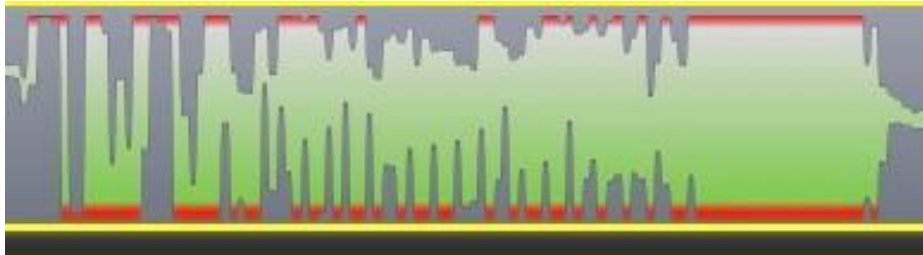
Error is up
to 1 LSB

$$V_{FS} = V_{\max} - V_{\min}$$

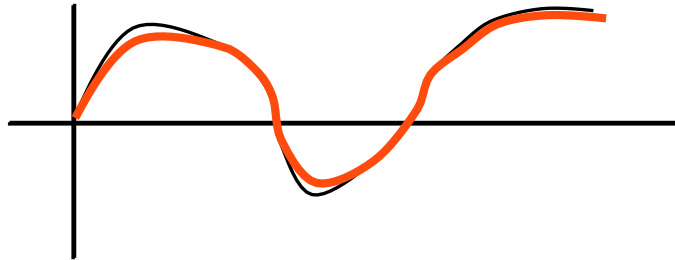
$$\Delta = \frac{V_{\max} - V_{\min}}{2^n} = \frac{FS}{2^n} = \text{LSB}$$

Issues with Signal Levels

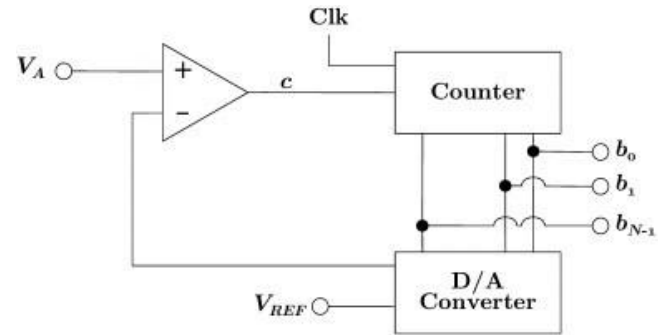
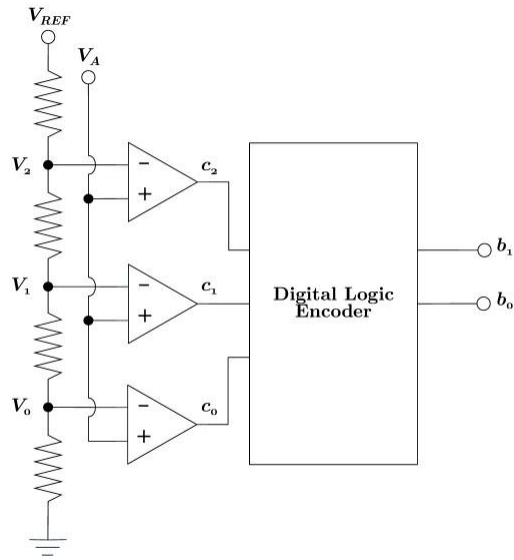
- Clipping: the input signal is higher than V_{max}
 - Leads to distorted signal with high frequency artifacts



Analog
Distortion

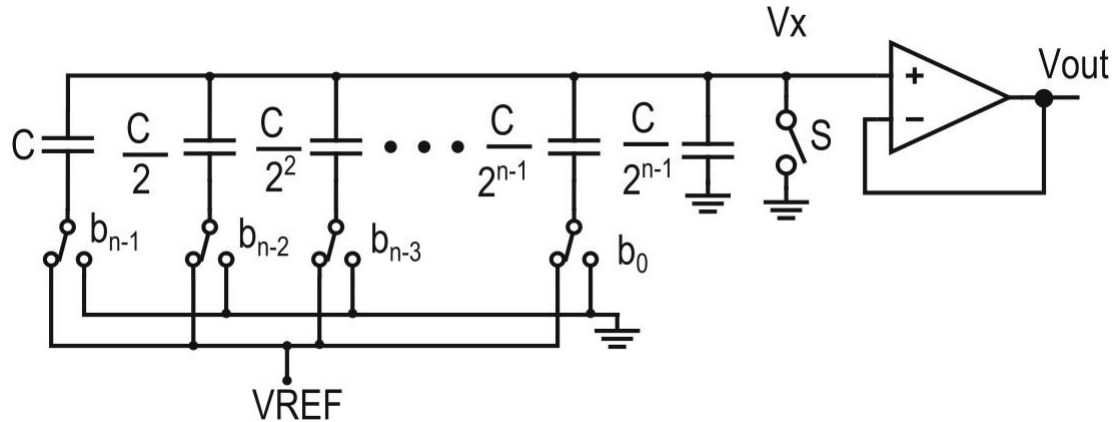


ADC Circuits



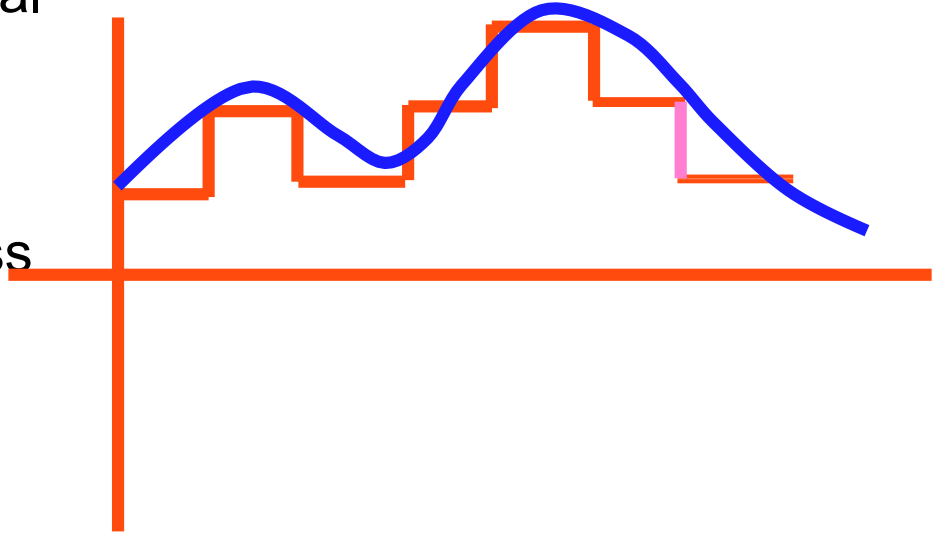
Digital to Analog (DAC) Converters

- Ladder DAC using capacitors
- Better suited to CMOS ICs



Reconstruction Filter

- Output of DAC is stepped signal
- Contains high frequency components
- Reconstruction filter is low-pass filter that smooths the output



Signal Processing

- Common applications for digital signal processing
 - Amplitude manipulation
 - Compression, expansion, amplification
 - Noise / artifact reduction
 - Gating, notch filtering
 - Frequency analysis
 - Tone identification
 - Equalization
 - Time-domain analysis
 - Identification of arrhythmias, etc.
 - Effects processing
 - Echo, reverb, flanging
 - Signal synthesis



Analogue to Digital Conversion on the AVR

AVR ADC Programming

I Before Sampling

- Determine the bandwidth of the signal
 - This dictates the required sample rate
- Determine the required resolution for the measurement and the voltage (dynamic) range of the input signal
 - This dictates the quantization and reference voltage requirements and indicates the possible need for external normalization of the input signal

Steps in Programming the ADC

- Enable the ADC functionality
- Select the desired ADC channel (there are 16)
 - Make the pin for the channel an input
- Select the conversion speed
- Select the voltage reference
- Start the conversion
- Wait for the conversion to complete
- Read the value

Enabling the ADC

- We use the ADCSRA and ADCSRB registers to configure the ADC for
 - Write 1 to ADEN to enable the ADC
 - Write 0 to ADIE to disable interrupts (for now)
 - Write 0 to ADATE to disable auto trigger

ADCSRA

Bit	7	6	5	4	3	2	1	0
0x7A	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Enabling the ADC

```
// setup the A register
*my_ADCSRA |= 0b10000000;
*my_ADCSRA &= 0b11011111;
*my_ADCSRA &= 0b11110111;
*my_ADCSRA &= 0b11111000;
```

- We use the ADCSRA and ADCSRB registers to configure the ADC for
 - Write 1 to ADEN to enable the ADC
 - Write 0 to ADIE to disable interrupts (for now)
 - Write 0 to ADATE to disable auto trigger
 - ADPS2-0: ADC Pre-scaler Select Bits.

ADCSRA

Bit	7	6	5	4	3	2	1	0
0x7A	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Sampling Modes

```
*my_ADCSRB &= 0b11110111;  
*my_ADCSRB &= 0b11111000;
```

- The ADC can be one of three modes:
 - free-running, so that a new sample is taken when a conversion is completed
 - single conversion mode
 - each sample is started explicitly in the software
 - auto-triggered mode
 - sampling starts based on a timer compare match, a timer overflow, the analog comparator, or an external interrupt request
 - These are enabled through setting ADATE in ADCSRA to 1

Table 26-6. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match A
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

ADCSRB

Preparing the Channel

NOTE: The book shows Port C for ADC0, but that is for the 328

- Example: use ADC0
- On the 2560, the input pin corresponds to PORTF pin 0 (pin 97)
 - Write 0 to DDRF to set as input
 - Write 1 to DIDR0 to turn off digital input
- Set ADC0 as input to ADC converter using ADMUX register

MUX5:0	Single Ended Input
000000	ADC0
000001	ADC1
000010	ADC2
000011	ADC3
000100	ADC4
000101	ADC5
000110	ADC6
000111	ADC7

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

NOTE: MUX5 is in the ADCSRB register

Preparing the Channel (2)

- Samples are 10 bits wide
- Data can be left or right justified
- Set the ADLAR bit in ADMUX to select the justification

// setup the MUX Register

```
*my_ADMUX &= 0b11011111;
```

```
*my_ADMUX &= 0b11100000;
```

8.3.3.1 ADLAR = 0

Bit	7	6	5	4	3	2	1	0
0x79	-	-	-	-	-	-	ADC9	ADC8
0x78	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
Read/Write	R	R	R	R	R	R	R	R
Default	0	0	0	0	0	0	0	0

8.3.3.2 ADLAR = 1

Bit	7	6	5	4	3	2	1	0
0x79	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
0x78	ADC1	ADC0	-	-	-	-	-	-
Read/Write	R	R	R	R	R	R	R	R
Default	0	0	0	0	0	0	0	0

8.3.1 ADMUX - ADC MULTIPLEXER SELECTION REGISTER

Bit	7	6	5	4	3	2	1	0
0x7C	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Setting V_{ref}

```
// setup the MUX Register  
*my_ADMUX &= 0b01111111;  
*my_ADMUX |= 0b01000000;
```

ADMUX register used to set V_{ref}

Bit	7	6	5	4	3	2	1	0
0x7C	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

REFS1-0	Description
00	AREF, internal V_{ref} turned off
01	AV_{CC} with external capacitor at AREF pin Note: Arduino already has capacitor placed on line.
10	Reserved
11	Internal 1.1V reference with external capacitor at AREF pin Note: Arduino already has capacitor placed on line.

Start the Conversion and Wait

- Start the conversion by writing 1 to ADSC in the ADCSRA register (single conversion mode)
- Wait for completion by checking the ADIF flag:
 - `while((*ADCSRA & 0x40) != 0);`

Bit	7	6	5	4	3	2	1	0
0x7A	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Read the Value

- The data is in two registers: ADCL (low) and ADCH (high)

```
// read only high byte for 8 bit accuracy
// assumes left justified
portADCDataRegisterHigh = ( unsigned char *) 0x79;
value = * portADCDataRegisterHigh;
```

```
// read entire 10 bits, with left justification
portADCDataRegister = ( unsigned short *) 0x78;
value = (* portADCDataRegister & 0 xFFC0) >> 6;
```

```
// read entire 10 bits with right justification
portADCDataRegister = (unsigned short *) 0x78;
value = (* portADCDataRegister & 0 x03FF);
```