Workshop 3

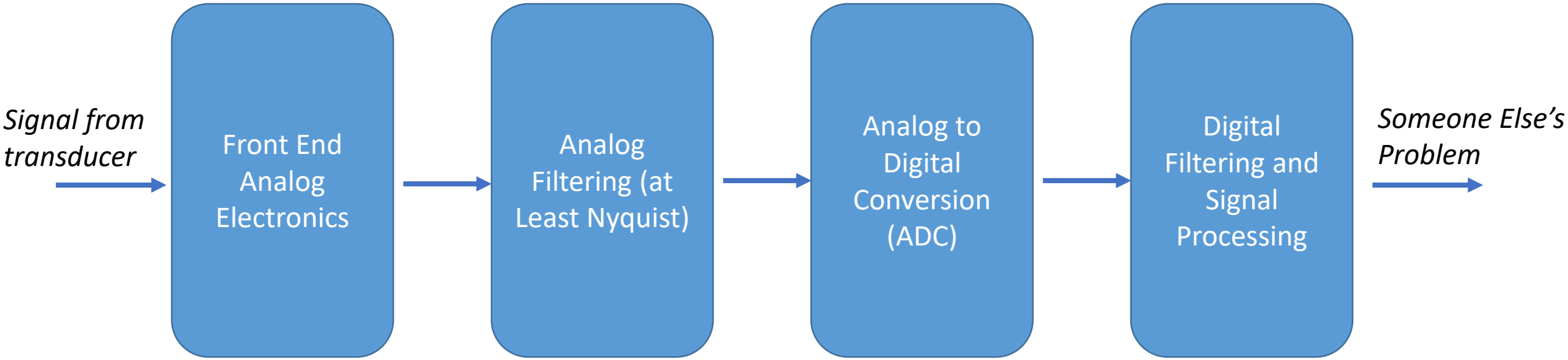
Sensor Interfaces – filtering and ADCs

Overview

- Many, many sensors out there:
  - Mechanical – gyroscope, accelerometer, piezoelectric, electromechanical, flow sensors…
  - Magnetic – hall effect, magnetoresistor, RF coil…
  - Light – photodiode/transistor, photoresistor, solar cell, SPAD
  - Temperature – RTD, thermistor, thermocouple…
  - Bioelectric – EMG, EEG, ECG, etc
  - Biochemical/Chemical – potentiostat, ISFET (and other functionalized transistors), DNA nanopore…
  - The list goes on

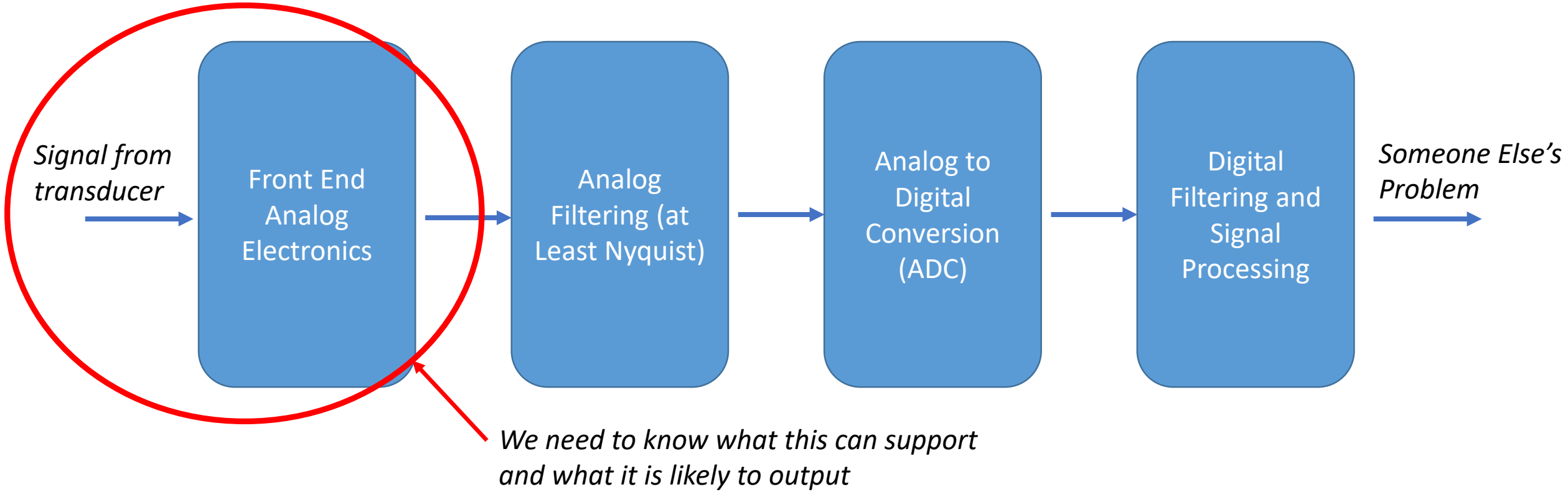
Overview

- Converting the signal from transducer to useful digital output often looks like this:



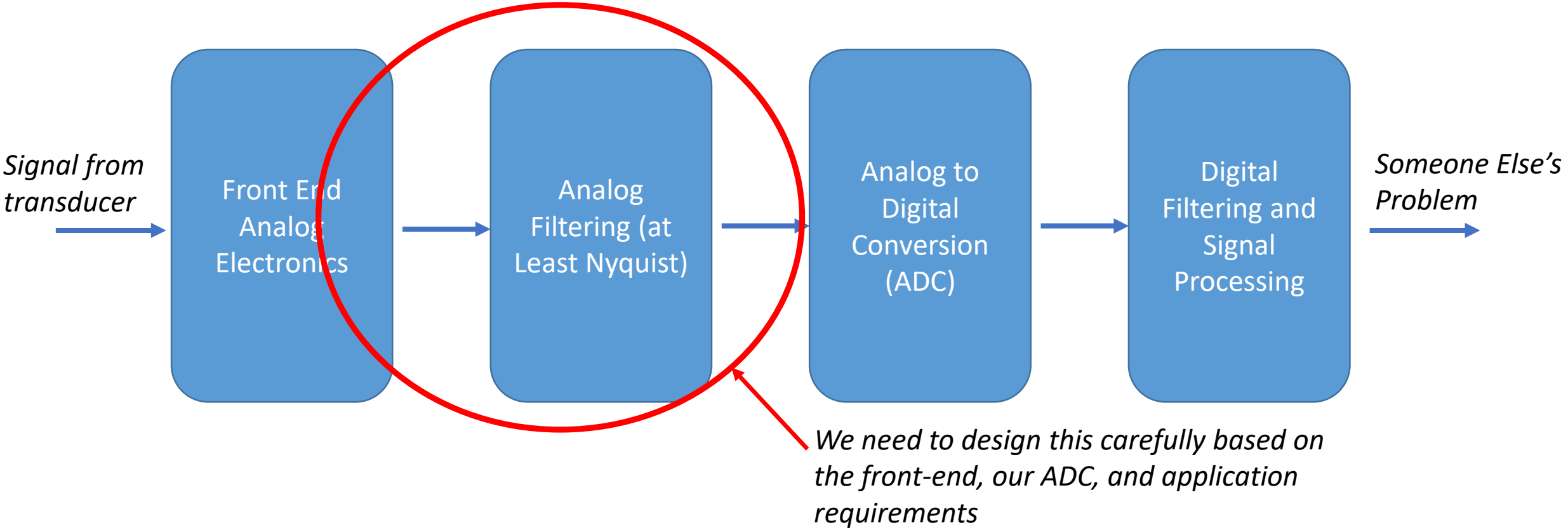
Overview

- Converting the signal from transducer to useful digital output often looks like this:



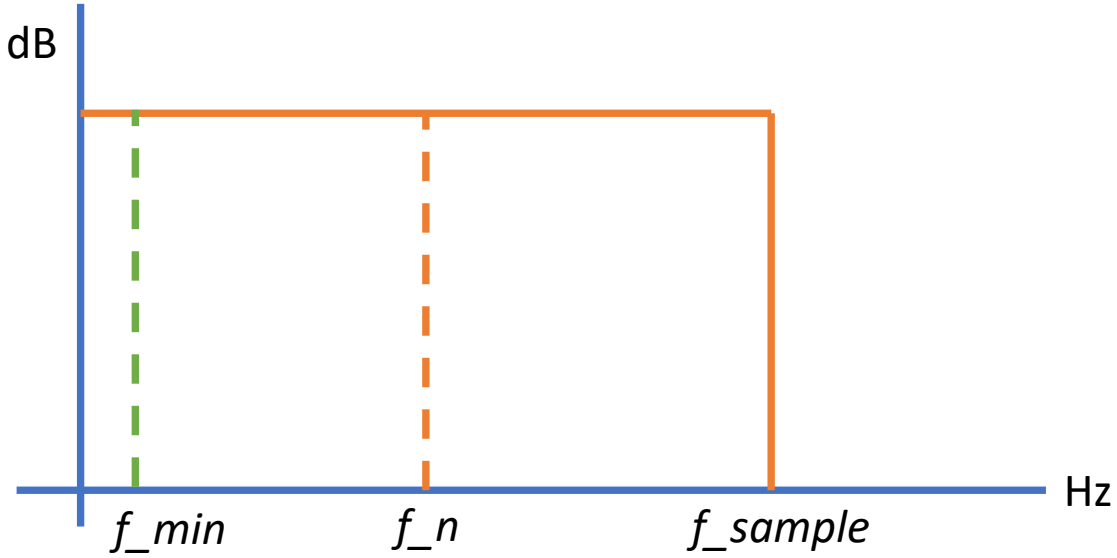
Overview

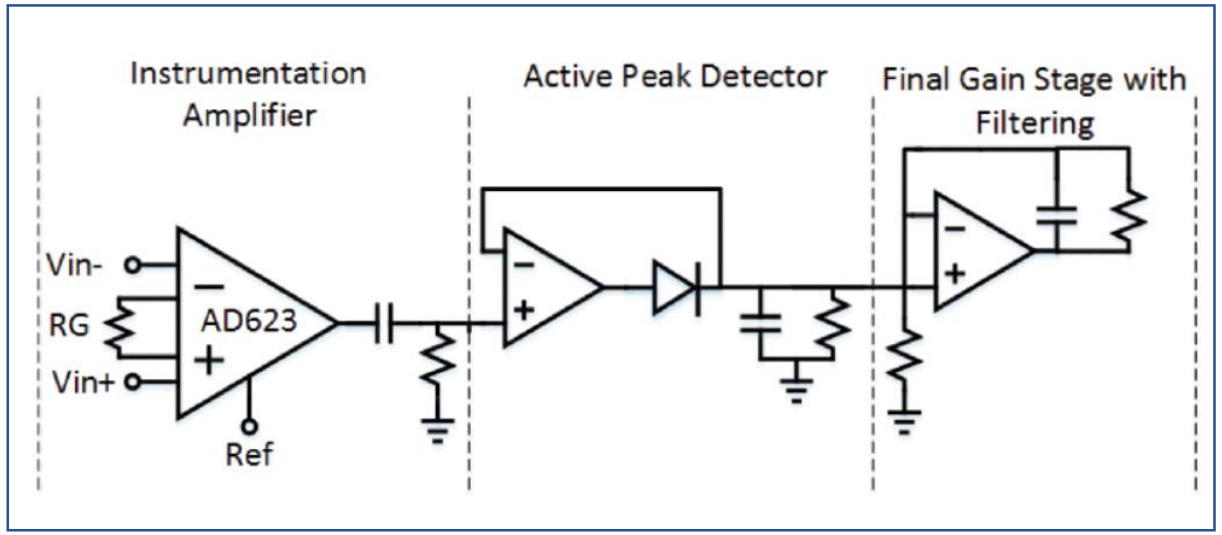
- Converting the signal from transducer to useful digital output often looks like this:



Analog Filtering

- Need to sample at least twice the frequency of the signal being measured to be useful
  - If max sampling frequency = X samples/second, need to filter at ½X Hz or lower
- It may also be appropriate to make this a band pass filter

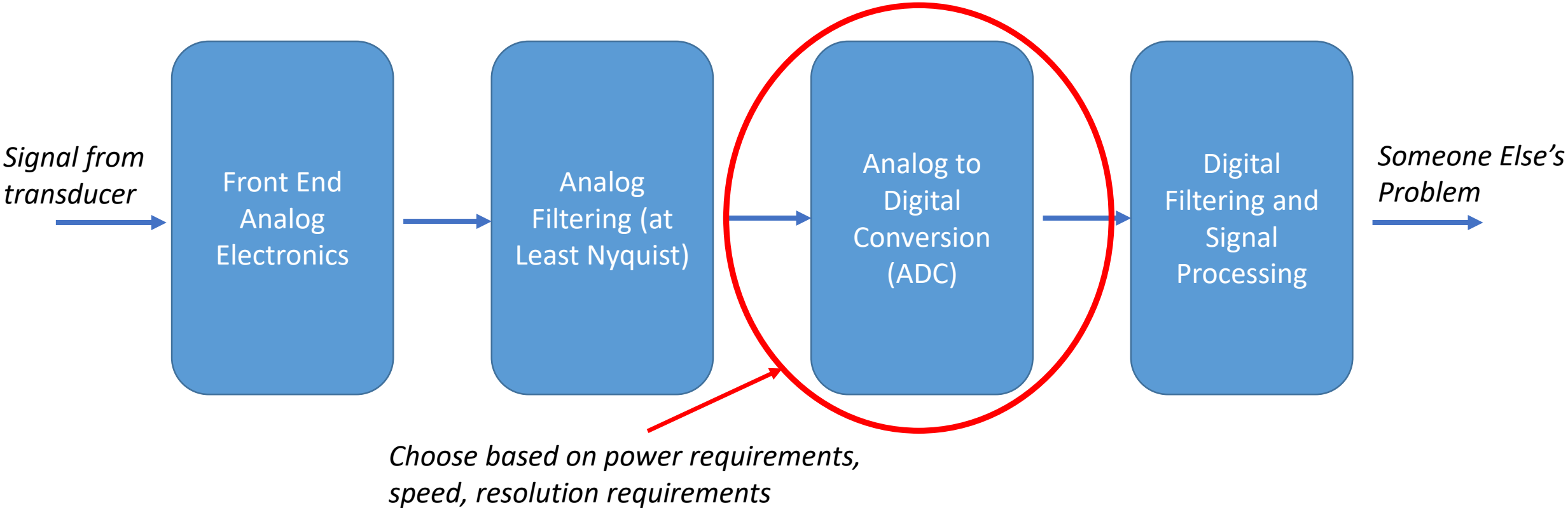




Example – Electromyographic recording

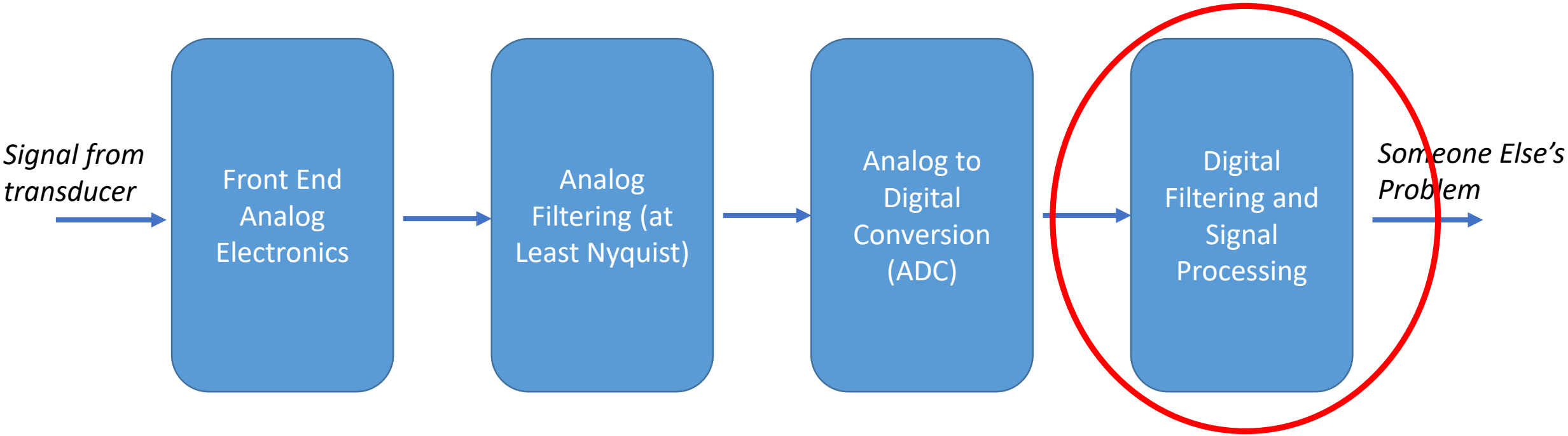
Overview

- Converting the signal from transducer to useful digital output often looks like this:



Overview

- Converting the signal from transducer to useful digital output often looks like this:



Example Project – Collision Detection

- Detecting birds hitting a wind turbine
- Detecting an object hitting your robot
- Use accelerometer to detect small/fast impacts
  - Note – a piezoelectric sensor might be better for this but the design flow is the same

Signal Flow

Front End Analog Electronics → Analog Filtering (at Least Nyquist) → Analog to Digital Conversion (ADC) → Digital Filtering and Signal Processing → *Binary – collision or no collision*

Signal Flow

Front End Analog Electronics

Analog Filtering (at Least Nyquist)

Analog to Digital Conversion (ADC)

Digital Filtering and Signal Processing

Binary – collision or no collision

ADXL335 Accelerometer

ADXL335

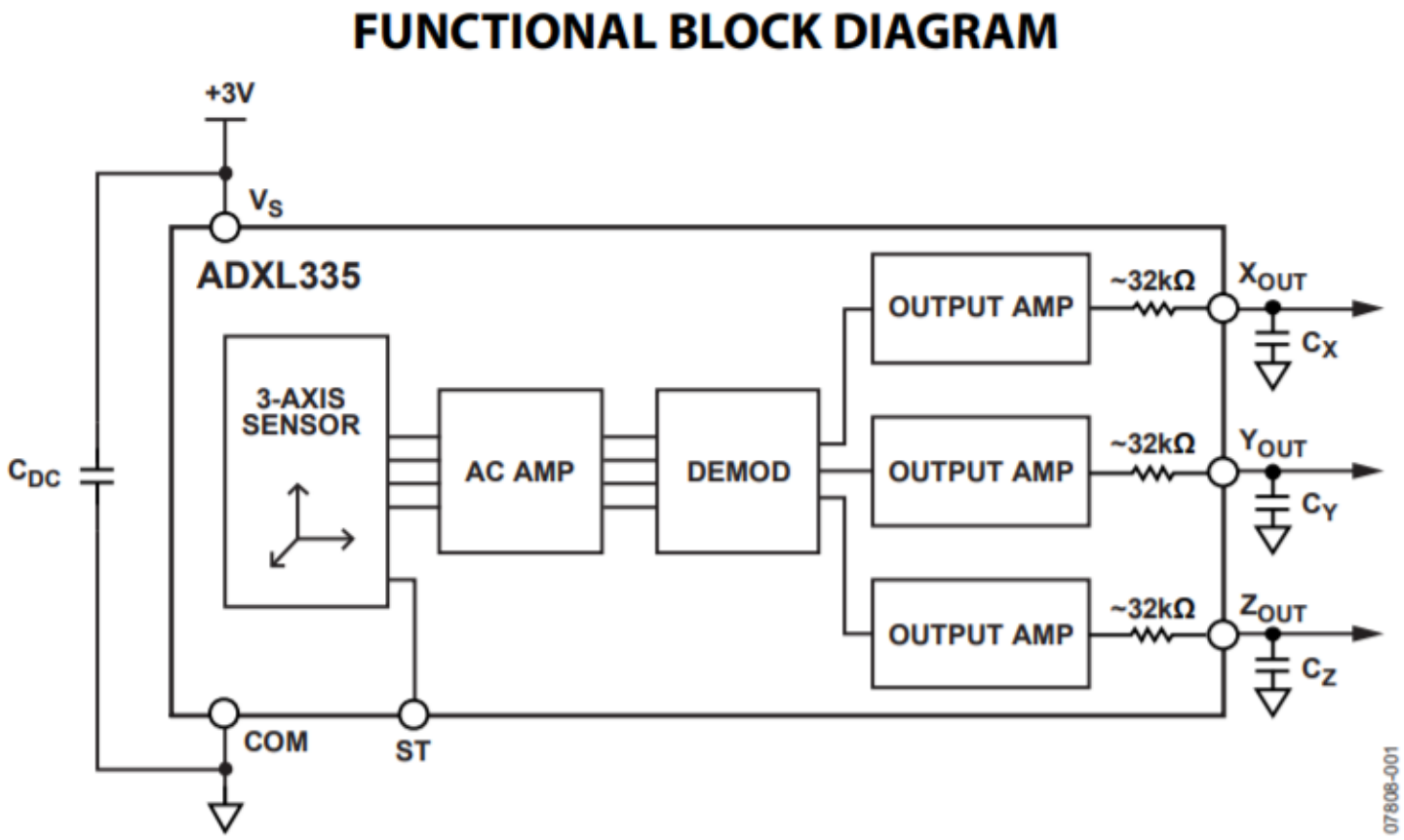


Figure 1.

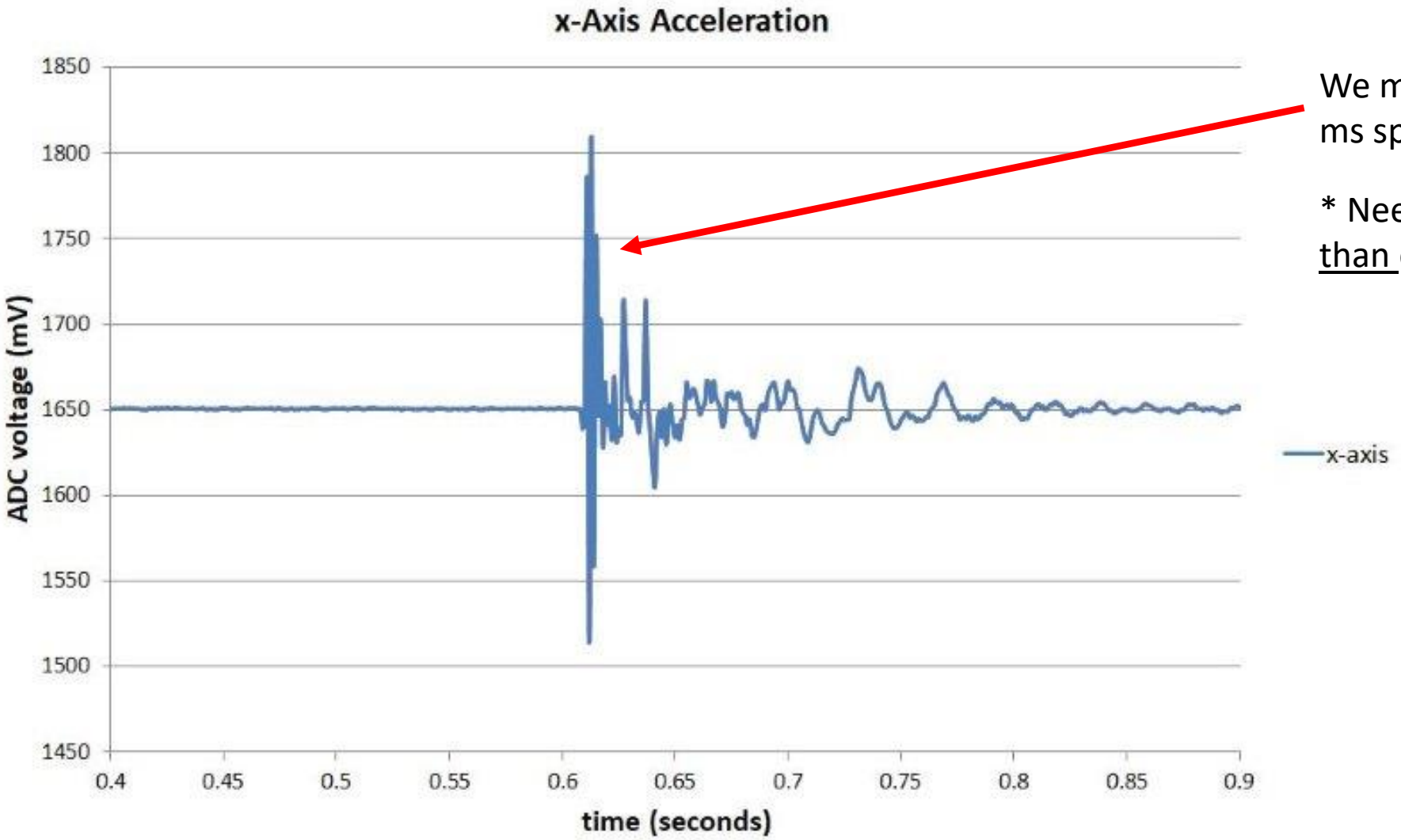
Table 4. Filter Capacitor Selection, C_X, C_Y, and C_Z

Bandwidth (Hz)	Capacitor (µF)
1	4.7
10	0.47
50	0.10
100	0.05
200	0.027
500	0.01

The user selects the bandwidth of the accelerometer using the C_X, C_Y, and C_Z capacitors at the X_{OUT}, Y_{OUT}, and Z_{OUT} pins. Bandwidths can be selected to suit the application, with a range of 0.5 Hz to 1600 Hz for the X and Y axes, and a range of 0.5 Hz to 550 Hz for the Z axis.

Image Credit: https://www.sparkfun.com/datasheets/Components/SMD/adxl335.pdf

How Fast?



We might want to capture 5 ms spikes (200 Hz)

* Need to sample faster than every 2.5 ms (400 sps)

Image Credit: https://www.allaboutcircuits.com/projects/how-to-protect-your-robot-with-automatic-collision-detection/

How small?

- Determined by noise, accelerometer sensitivity, ADC resolution

- 3.3 V supply, 8 bit ADC, 300 mV/g sensitivity → ~43x10-3 g/bit
- 3.3 V supply, 10 bit ADC, 300 mV/g sensitivity → ~11x10-3 g/bit

- With noise and other ADC non-idealities, we can expect to do much worse than this

Signal Flow

Front End Analog Electronics

Analog Filtering (at Least Nyquist)

Analog to Digital Conversion (ADC)

Digital Filtering and Signal Processing

Binary – collision or no collision

ADXL335 Accelerometer

ATtiny85

Will our ADC Work?

Features

- 10-bit Resolution
- 1 LSB Integral Non-linearity
- ± 2 LSB Absolute Accuracy

← Probably fine, but we may want to oversample or disregard a few LSB depending on application

- 65 - 260 µs Conversion Time
- Up to 15 kSPS at Maximum Resolution

← Overkill

- Four Multiplexed Single Ended Input Channels
- Two differential input channels with selectable gain
- Temperature sensor input channel
- Optional Left Adjustment for ADC Result Readout
- 0 - V_{CC} ADC Input Voltage Range
- Selectable 1.1V / 2.56V ADC Voltage Reference
- Free Running or Single Conversion Mode
- ADC Start Conversion by Auto Triggering on Interrupt Sources
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Cancele
- Unipolar / Bibilar Input Mode
- Input Polarity Reversal Mode

The Hardware

Firmware - ADC

ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
0x07	REFS1	REFS0	ADLAR	REFS2	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	

Voltage Reference

Left Shift

Mux selects ADC channel

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
0x06	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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	

Enable ADC!

Start Conversion

Interrupt Flag

Enable Interrupts

Clock Prescaler

Firmware - ADC

17.13.3.1 ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
0x05	–	–	–	–	–	–	ADC9	ADC8	ADCH
0x04	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

17.13.3.2 ADLAR = 1

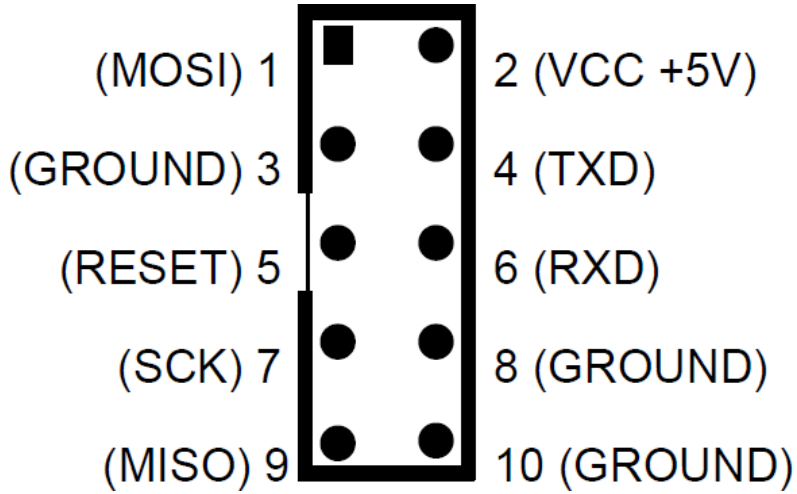
Bit	15	14	13	12	11	10	9	8	
0x05	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
0x04	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

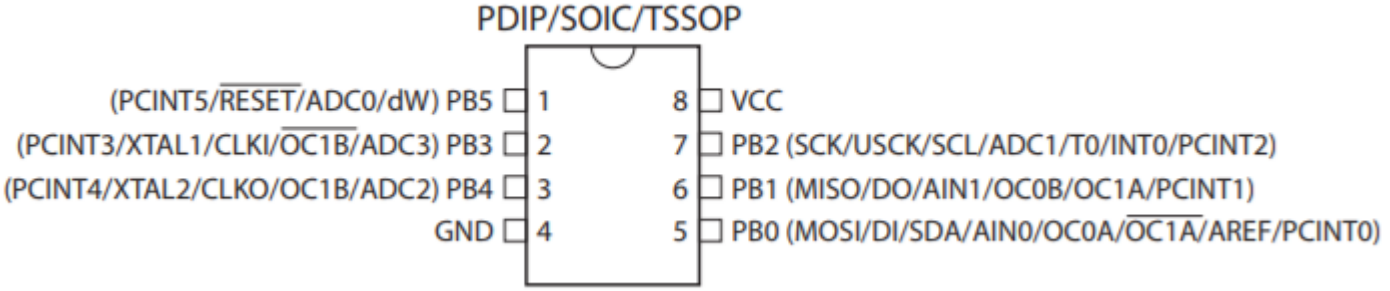
Data → Computer via Software UART

- We don’t have hardware UART on the ATtiny85 so we ‘bit-bang’
- Helpful tutorial on this: https://marcelmg.github.io/software_uart/
- Basically just use timer counter registers and an IO pin to recreate a UART TX signal (we don’t need RX for this project)

Set up hardware

- Plug ATTINY85 into breadboard
- Wire programmer
  - VCC → VCC
  - GND → GND
  - MOSI → MOSI
  - MISO → MISO
  - RESET → RESET
  - SCK → SCK



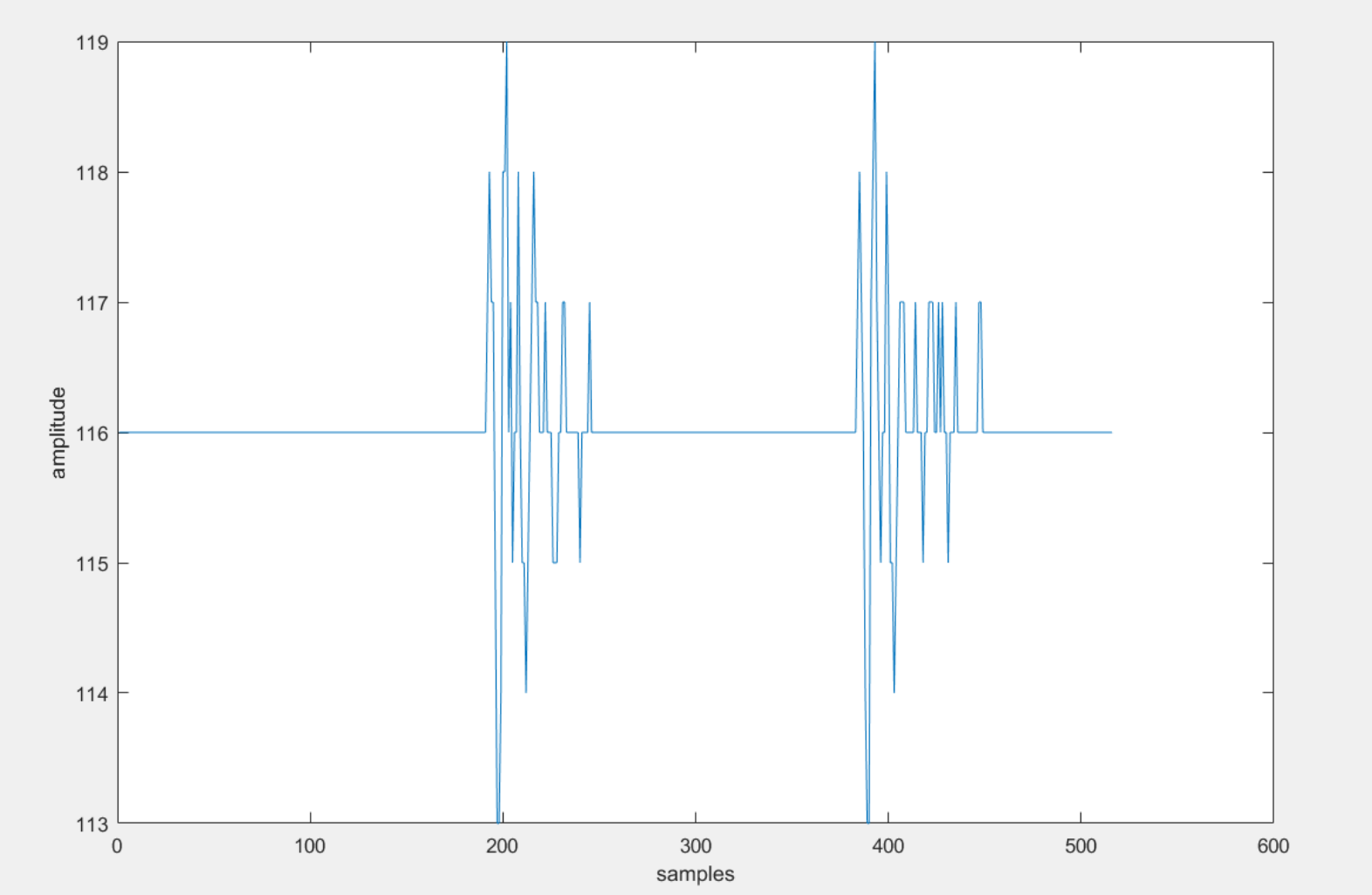


Set up Hardware

- X_out, Y_out, and Z_out connect to ADC1 (PB2), ADC2 (PB4), and ADC3 (PB3)
- PB0 acts as TX port – connect this to the RX of the FTDI along with 3.3V and GND

Initial Test

- Want some algorithm to interpret this if we are to detect an object



TODO

- How can we optimize this code? Do we need to?
- If we need to know frequency information, what do we need to do?
  - Currently timing isn’t easy to manipulate
- Write an algorithm to give binary yes/no collision information
  - Combine data from multiple axis?
- How sensitive can we get (try 10 bit resolution)

ADC Timing

Figure 17-5. ADC Timing Diagram, Single Conversion