

Lecture 08

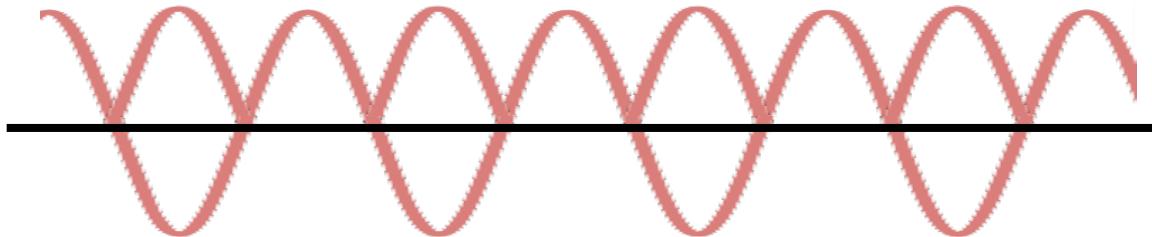
ADC and Lab 3

Agenda

- 00. Lab 2 Stuff
- 01. Analog to Digital Converter
- 02. ADC on ATmega 32U4 (Teensy)
- 03. Advanced ADC functions
- 04. Rotational position sensors
- 06. Software filters (if time)

Lab 2

- 60Hz or 120Hz
- Lab 2.4 Asks for 200Hz in the presence of 60Hz ambient light, 2000Hz is probably easier, so, lets do 2000Hz.
- Dealing with analog values on Teensy (LVTTL logic levels)
 - Above 1.9V is considered logic high $V_{IH} = 1.9V$
 - Below 0.9V is considered logic low $V_{IL} = 0.9V$



ESP32 Woes

- ESP32 sometimes gets very hot and fails... alot
- I suspect that it is electrostatic discharge.
- Try to stay grounded
 - Metal frame of plugged in laptop

Okay to touch





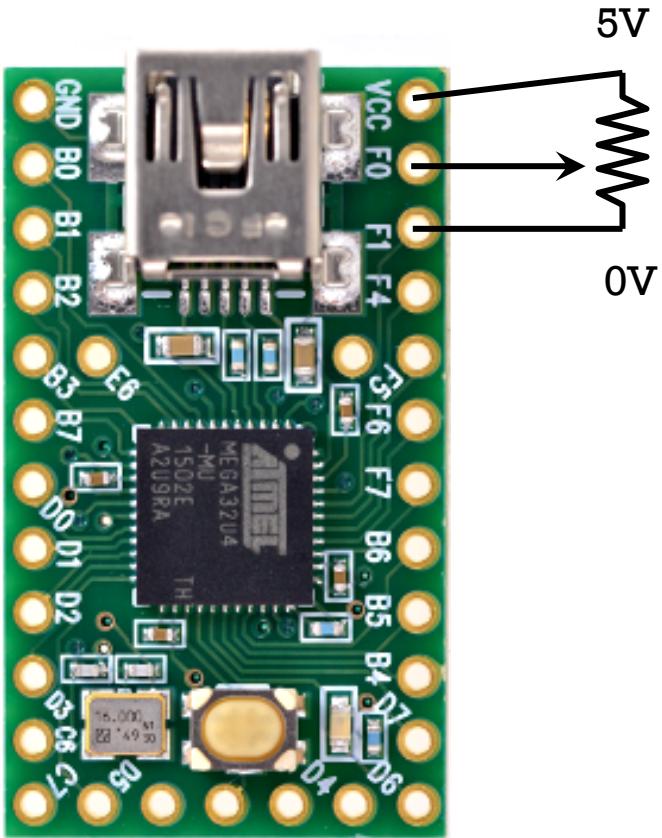
Lab 3 Teaser

01

Analog to Digital Converter (ADC)

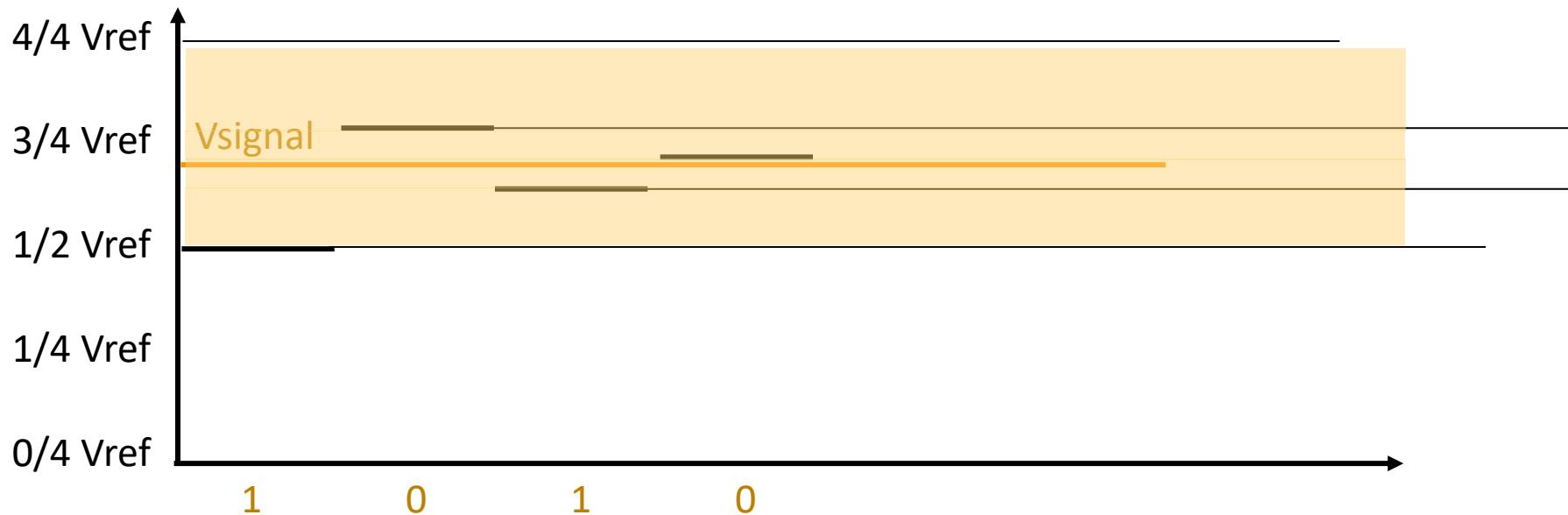
What and why?

- What is an analog to digital converter?
- What is the typical output?
- Where would you use one?

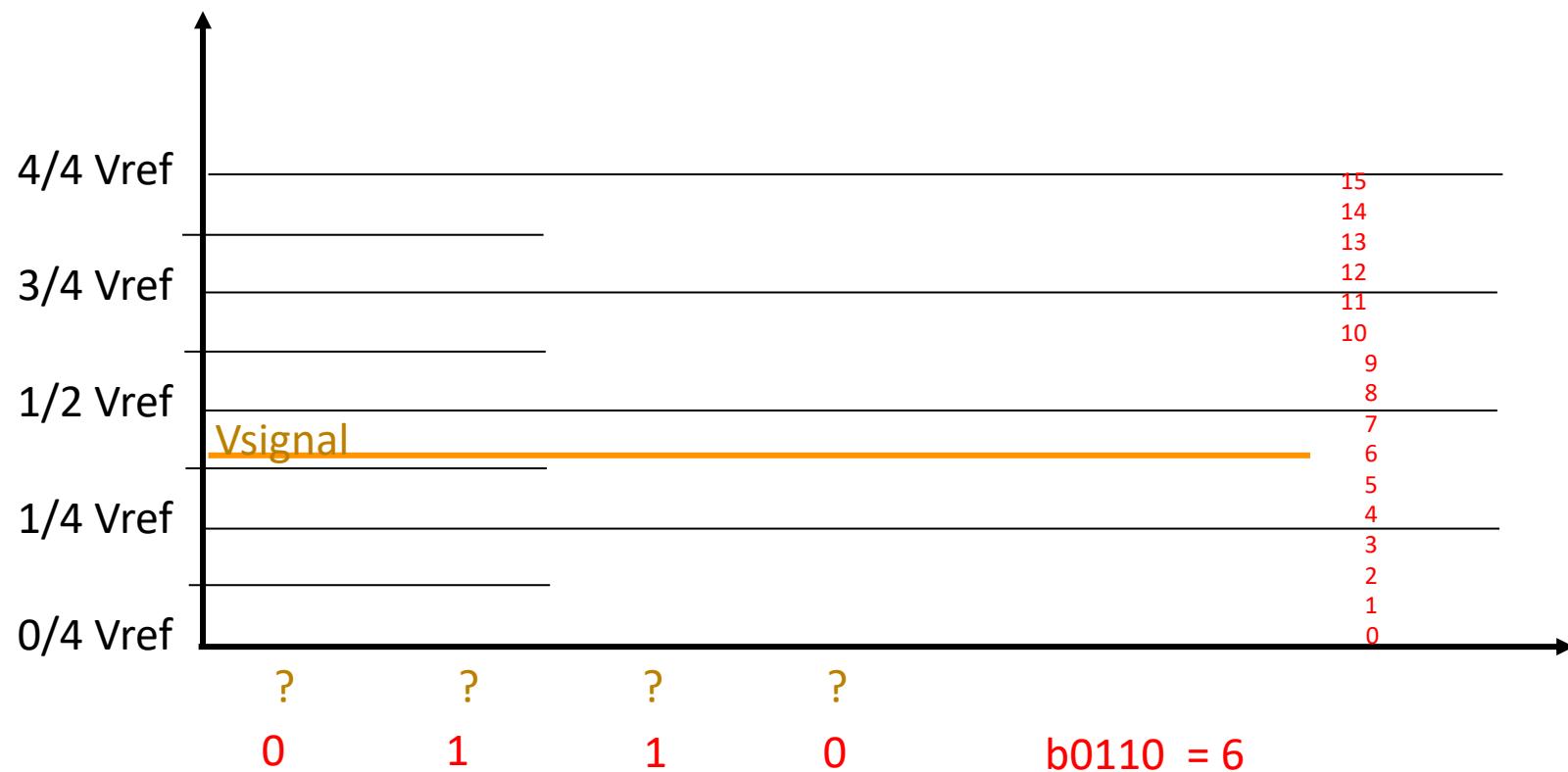


Ex: Successive Approximation (SAR)

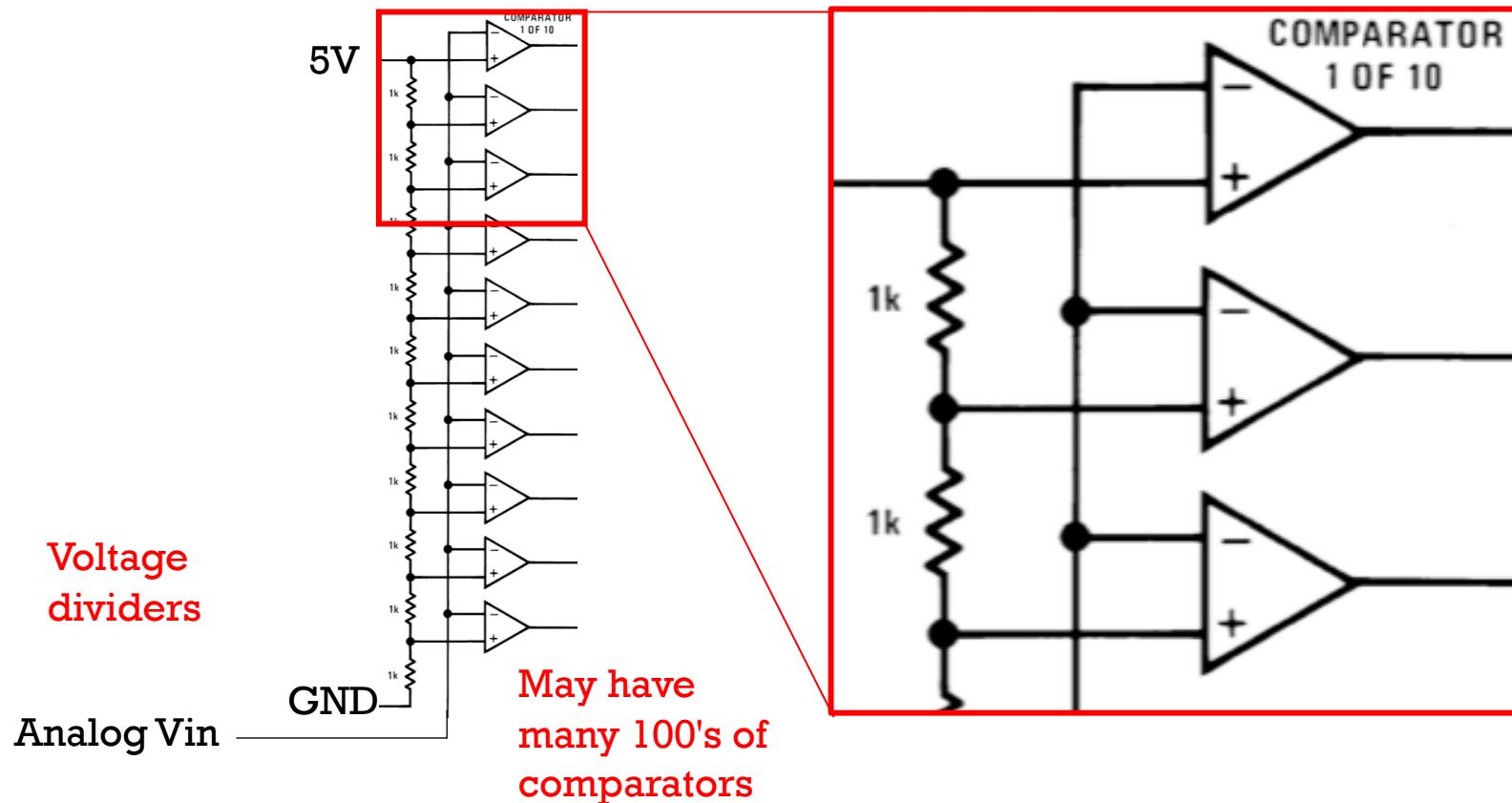
Note: resolution and sampling rate are coupled here. More resolution takes more time



Q1: Successive Approximation



Example Flash Conversion



ADC Architecture

Type of ADC	Resolution (Max Bits)	Sample Rate Typ (Max)	
Successive Approximation	8 - 18	10-20 kS/s	(10 MS/s)
Flash	4 - 12	400 kS/s	(10 GS/s)
Sigma Delta	8 - 32	1 kS/s	(1MS/s)
Dual Slope	12 - 20	30 S/s	(100 S/s)

Most Popular:

- Successive Approximation (SAR) – Low cost (e.g. orders of magnitude less the cost of flash for 10bit resolution). This is what Teensy uses (10bit)
- Sigma Delta – generally higher resolution but slower.
- DAC combo with comparator – very low cost, not really an ADC.

02

ADC on ATmega 32U4

Analog to Digital Converter - ADC

- Page 297 of datasheet (on Canvas) (Chapter 24)

[Atmel-7766-8-bit-avr-ATmega16U4-32U4_Datasheet.pdf](#)

24. Analog to Digital Converter - ADC

24.1 Features

- 10/8-bit Resolution
- 0.5LSB Integral Non-linearity
- ± 2 LSB Absolute Accuracy
- 65 - 260 μ s Conversion Time
- Up to 15kSPS at Maximum Resolution
- Twelve Multiplexed Single-Ended Input Channels
- One Differential amplifier providing gain of 1x - 10x - 40x - 200x
- Temperature sensor
- Optional Left Adjustment for ADC Result Readout
- 0 - V_{CC} ADC Input Voltage Range
- Selectable 2.56V ADC Reference Voltage
- Free Running or Single Conversion Mode
- ADC Start Conversion by Auto Triggering on Interrupt Sources
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler

The ATmega16U4/ATmega32U4 features a 10-bit successive approximation ADC. The ADC is connected to an 12-channel Analog Multiplexer which allows six single-ended voltage inputs constructed from several pins of Port B, D, and F. The single-ended voltage inputs refer to 0V (GND).

The device also supports 32 differential voltage input combinations, thanks to a differential amplifier equipped with a programmable gain stage, providing amplification steps of 0 dB (1x), 10 dB (10x), 16dB (40x), or 23dB (200x) on the differential input voltage before the A/D conversion. Two differential analog input channels share a common negative terminal (ADC0/ADC1), while any other ADC input can be selected as the positive input terminal. If 1x, 10x, or 40x gain is used, 8-bit resolution can be expected. If 200x gain is used, 7-bit resolution can be expected.

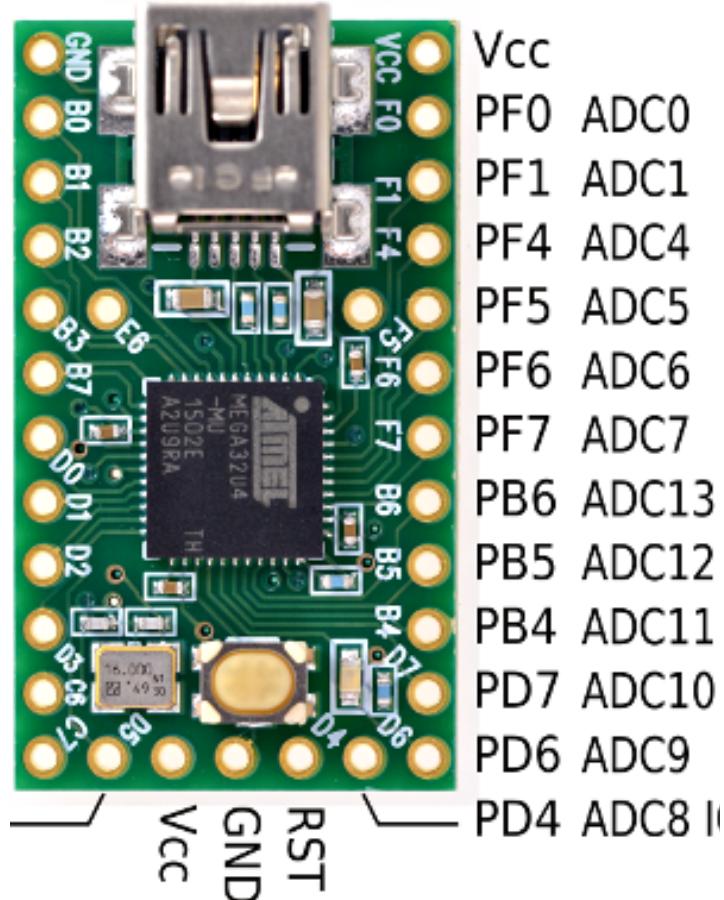
The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown in [Figure 24-1](#).

The ADC has a separate analog supply voltage pin, AV_{CC}. AV_{CC} must not differ more than ± 0.3 V from V_{CC}. See the paragraph "[ADC Noise Canceler](#)" on page 305 on how to connect this pin.

Internal reference voltages of nominally 2.56V or AV_{CC} are provided On-chip. The voltage reference may be externally decoupled at the AREF pin by a capacitor for better noise performance.

Important ADC registers

- **DIDR0/DIDR2** – Digital Input Disable #
- **ADMUX** – Analog to Digital Multiplexer
- **ADCSRA** – Analog to Digital Converter Status Register A
- **ADC** – Analog to Digital Converter registers (2byte) – holds 10bit value for all channels.



Note: no ADC2 nor ADC3

Steps to use ADC

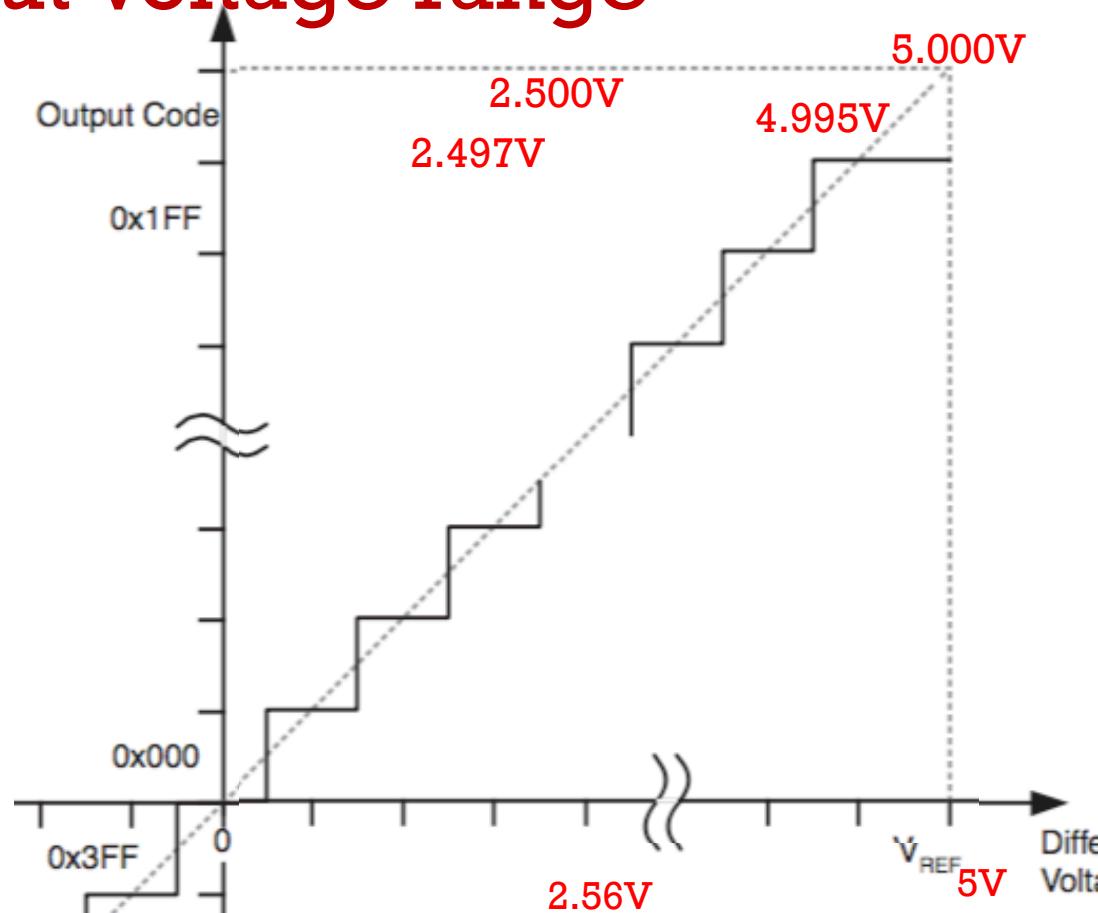
1. set the voltage reference
 2. set the ADC clock prescaler
 3. disable ADC digital input
 - ~~4. set up interrupts and triggering, if desired~~
 5. select the desired analog input
 6. enable conversions
 7. start the conversion process
 8. wait for conversion to finish
 9. read the result
 10. clear the conversion flag
-
- The diagram features two red curly braces. One brace groups the first three steps (1-3) and is labeled "Setup registers". Another brace groups the last seven steps (5-10) and is labeled "Get reading from one pin".

All of this using registers.

See <http://medesign.seas.upenn.edu/index.php/Guides/MaEvArM-adc>

Step 1: ADC input voltage range

- Sensitivity determined by
 - input range and
 - resolution (#of bits)
- Vref can be Vcc, 2.56V or AREF pin (internal on Teensy)
- If signal is 0-2.5V, then using 2.56V for Vref is better



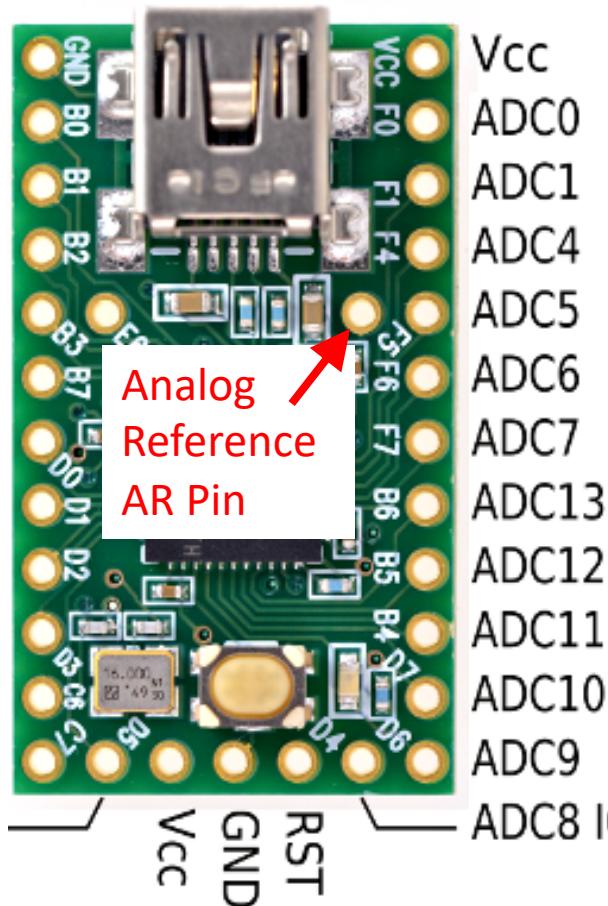
ADC Reference

Voltage Reference

The reference voltage for the ADC is set by the **REFS1** and **REFS0** bits in **ADMUX** as:

ADMUX: REFS1	ADMUX: REFS0	
0	0	external AREF (AR pin) - DEFAULT
0	1	Vcc
1	1	internal 2.56V

Note - Do not set this to **Vcc** or **2.56V** if there is a voltage applied to the **AR** pin, as they will be shorted together!



Step 2: ADC frequency

- Valid Clock is 50kHz to 200khz – you set by **ADCSRA:ADPS[012]**

ADCSRA: ADPS2	ADCSRA: ADPS1	ADCSRA: ADPS0		Speed w/16Mhz
0	0	0 or 1	/2	8Mhz
0	1	0	/4	4Mhz
0	1	1	/8	2Mhz
1	0	0	/16	1Mhz
1	0	1	/32	500khz
1	1	0	/64	250khz
1	1	1	/128	125khz

Only valid one

Notes:

Analog read is slow...

Takes 25 cycles first read (5Khz)

13 cycles every other (~10Khz)

Compare to digital read (~4Mhz)

Timer Prescalers DO NOT effect
ADC prescaler

There is a system clock
prescaler, CLKPR which scales
everything including ADC
clock.

Step 3 Disable Digital Input

DIDR0/DIDR2 Digital Input Disable Register

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	DIDR0							
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7:4, 1:0 – ADC7D..4D - ADC1D..0D: ADC7:4 - ADC1:0 Digital Input Disable

When this bit is written logic one, the digital input buffer on the corresponding ADC pin is disabled.

Bit	7	6	5	4	3	2	1	0	
Read/Write	-	-	ADC13D	ADC12D	ADC11D	ADC10D	ADC9D	ADC8D	DIDR2
Initial Value	0	0	0	0	0	0	0	0	

- Bit 5:0 – ADC13D..ADC8D: ADC13:8 Digital Input Disable

Q2: setup ADC to use ADC0

1. Use VCC for Vref.
 2. Set prescaler /128 -> ~100kHz (more info @ atmel doc p.315)
 3. Disable digital input for ADC0

ADMUX: REFS1	ADMUX: REFS0	
0	0	external AREF (AR pin) - DEFAULT
0	1	Vcc
1	1	internal 2.56V

ADCSRA: ADPS2	ADCSRA: ADPS1	ADCSRA: ADPS0	
1	0	0	/16
1	0	1	/32
1	1	0	/64
1	1	1	/128

Q2: setup ADC to use ADC0

Step 5: Choose ADC channel

(skip step 4 for now)

Q3: Use set and clear to choose ADC9

ADCSR: MUX5	ADMUX: MUX2	ADMUX: MUX1	ADMUX: MUX0		
0	0	0	0	ADC0	F0
0	0	0	1	ADC1	F1
0	1	0	0	ADC4	F4
0	1	0	1	ADC5	F5
0	1	1	0	ADC6	F6
0	1	1	1	ADC7	F7
1	0	0	0	ADC8	D4
1	0	0	1	ADC9	D6
1	0	1	0	ADC10	D7
1	0	1	1	ADC11	B4
1	1	0	0	ADC12	B5
1	1	0	1	ADC13	B6

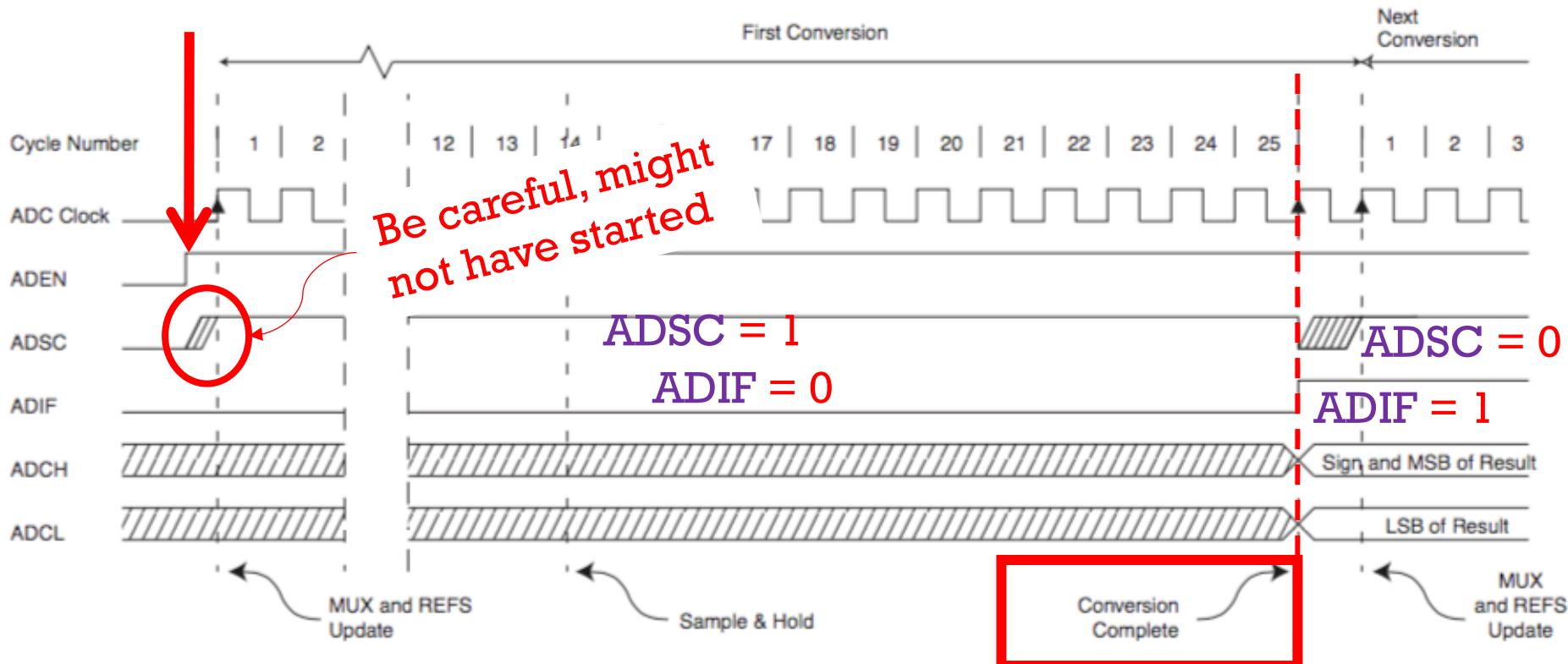
Step 6 to 10: Read ADC values

6. Enable ADC – set **ADCSRA:ADEN**
7. Start conversion – set **ADCSRA**,
ADSC
8. Wait for conversion to finish (check **ADCSRA:ADIF**)
9. Use value e.g. print (**ADC**) to USB
10. Clear the conversion complete flag (if using **ADCSRA:ADIF**)

Q4: write commands to do steps 6 - 10

ADC single conversion timing diagram

Difference between (ADCSRA:ADSC) and (ADCSRA:ADIF) ?



ADC demo

```
int main(void){
    initADC();      // My subroutine to initialize the ADC registers
    m_usb_init();  // Subroutine to initialize USB communication
    set(DDRF,1);   // set PF1 to be ground for voltage divider pot
    clear(PORTF,1); // PF1 pin output low

    for(;;){
        if (bit_is_set(ADCSRA, ADIF)) { // if flag is set (conversion complete) update
            set(ADCSRA, ADIF);        // reset the flag, see page 316

            m_usb_tx_string("\rADC = "); // \r is for a carriage return, ascii 13
            m_usb_tx_uint(ADC);        // print the ADC value from the ADC register
            asteriskmeter(ADC/20);    // my subroutine to print ***'s

            set(ADCSRA, ADSC);        //start converting again
        }
    }
    return 0; /* never reached */
}
```

03

ADC Advanced Functions and Subroutine Coding

ADC Multiplexing (multiple channels)

- Simplest using free run mode

Setup

1. set the voltage reference
2. set the ADC clock prescaler
3. disable some digital inputs

Loop at ~5Khz

4. disable conversion
5. select the analog channel (change to new one if desired)
6. enable conversions
7. wait for conversion to finish
8. read the result for the channel selected

ADC Multiplexing (multiple channels)

- Faster than simple (about double speed)

Setup

1. set the voltage reference
2. set the ADC clock prescaler
3. disable some digital inputs
4. select the analog input for 1st read

Loop @ ~10Khz

5. enable conversion for nth read
4. select/change analog input for (**nth+1**) read
6. wait for conversion to finish
7. read **nth** result

see Atmel docs starting at p 300 for more info.

ATMEGA programmable gain functions

- The ATMEGA32U4 has some fancy ways of amplifying values between two ADC ports.
- Can set 10x 40x 200x gains.
- It's a bit complicated and not very common among ADC's in other processors – not recommended, but it is there.
- More info:
 - Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf
 - Page 297 to 315

Example: ADC Setup Subroutine

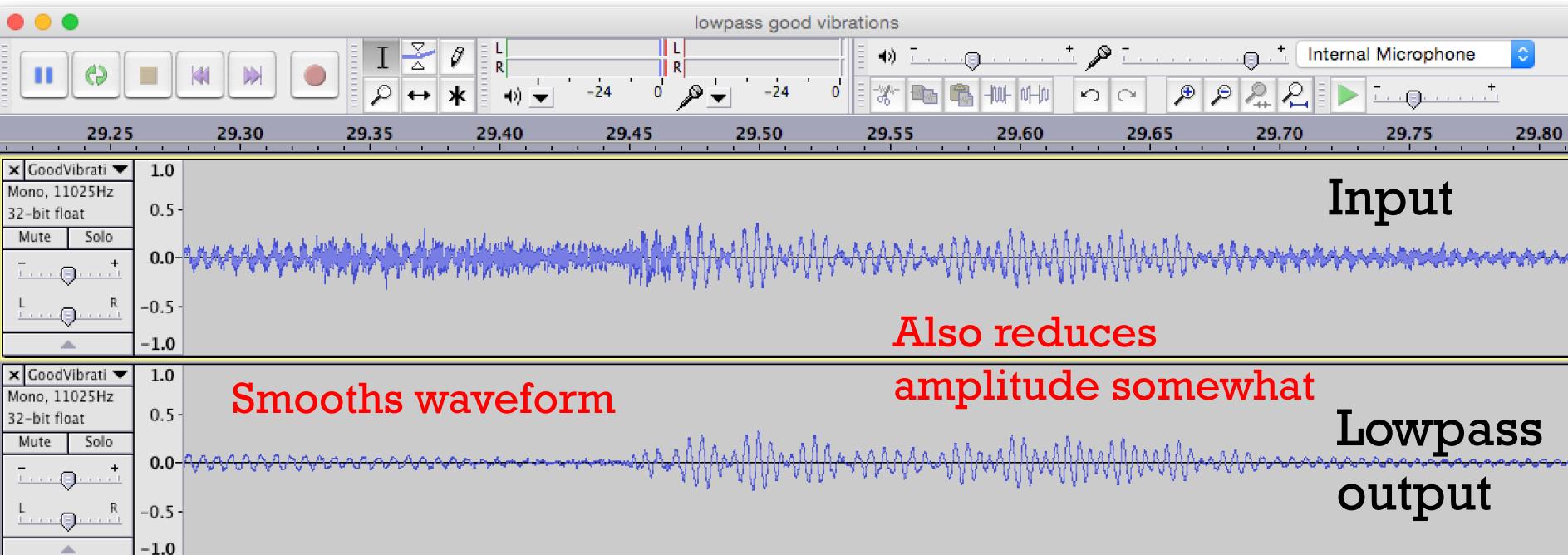
```
void adc_setup(int ADCchannel){  
    if(ADCchannel == 1) {  
        set(DIDRO, ADC1D); // disable digital input  
        set(ADCSRA, ADATE); // triggering  
        set(ADMUX, MUX0); // pin f1  
        clear(ADMUX, MUX1);  
        clear(ADMUX, MUX2);  
        clear(ADCSR8, MUX5);  
    }  
    if(ADCchannel == 4) {  
        set(DIDRO, ADC4D); // disable digital input  
        set(ADCSRA, ADATE); // triggering  
        clear(ADMUX, MUX0); // pin f4  
        clear(ADMUX, MUX1);  
        set(ADMUX, MUX2);  
        clear(ADCSR8, MUX5);  
    }  
    ... if(ADCchannel == 5) ...  
    ... if(ADCchannel == 6) ...  
    ... // up to ADCchannel 13  
}
```

THIS CODE HAS ERRORS
DO NOT COPY AND PASTE!!

04

Software Filters

Audio filters work same



Simple software low pass software filter

Q4: What does this code do?

Use subroutines

```
int avgtwo(int data1, data2) {  
    return (data1 + data2)/2;  
}
```

```
int main() {  
    for(;;){  
        PRINT (avgtwo(readSensor(),readSensor()));  
    }  
}
```

Sequence of Sensor values	Lowpass output
------------------------------	-------------------

6	
7	6
4	

6	5
5	

100	52
4	

5	5
5	

Simple software low pass software filter

Use subroutines

```
int lowpass(int newvalue) {  
    static int lastvalue;  
  
    lastvalue = (newvalue + lastvalue)/2;  
  
    return lastvalue;  
}  
  
int main() {  
    for(;;){  
        PRINT (lowpass( readSensor() )) ;  
    }  
}
```

Sequence of
Sensor values

6	3
7	4
4	4
6	5
5	5
100	100
4	4
5	5
5	5

Lowpass
output

3
4
4
5
5
52
28
16
10

Moving Average Software Filter

```
int movingavg(int newvalue,int WEIGHT) {  
    int update;  
    static int lastvalue;  
    update = (WEIGHT*lastvalue + newvalue)/(WEIGHT+1);  
    lastvalue = update;  
    return update;  
}
```

Example input	6	7	4	6	5	100	4	5	5
Running avg (weight = 1)	3	5	4	5	5	52	28	16	10
Running avg (weight = 3)	2	3	3	3	3	27	21	17	14

Exponential Moving Average (EMA) [Infinite Impulse Response (IIR) filter]

$$y[n] = (1 - \alpha)y[n - 1] + \alpha x[n]$$

Example: update = (1 - 0.3) lastvalue + 0.3 * newvalue;
update = (WEIGHT*lastvalue + newvalue)/(WEIGHT+1);

$$f_{3dB} = \frac{f_s}{2\pi} \cos^{-1} \left[1 - \frac{\alpha^2}{2(1 - \alpha)} \right]$$

f_{3dB} is the corner (cutoff) frequency

f_s is the sampling frequency

α is $1 - (\text{WEIGHT} / (\text{WEIGHT} + 1))$ in the previous code

$$\text{WEIGHT} = (1 - \alpha)/\alpha$$

EMA Low Pass Example

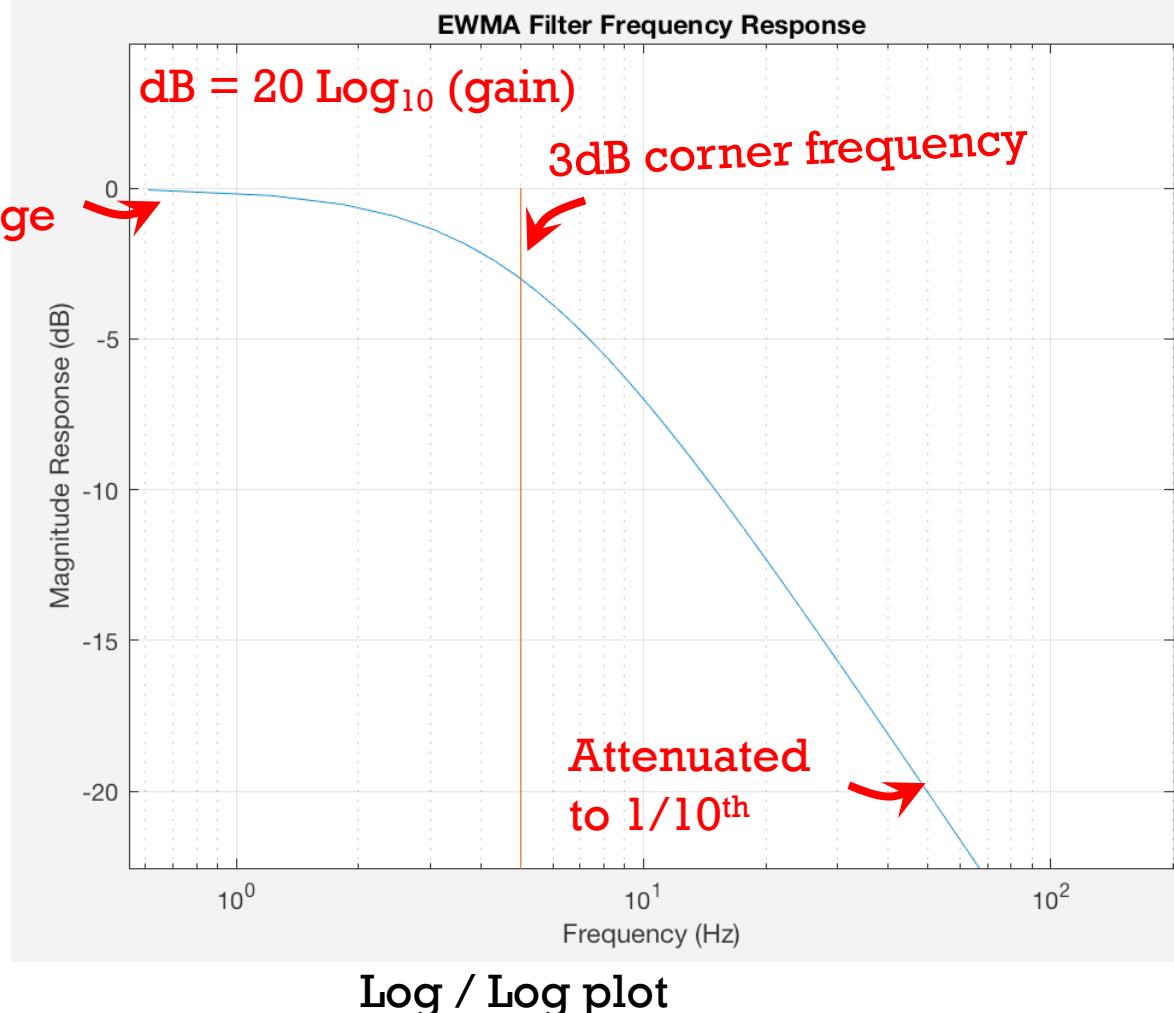
$F_s = 40000\text{Hz}$

$F_{3\text{db}} = 5\text{Hz}$

$\text{WEIGHT} = 1273$

$(\alpha = 7.85 \times 10^{-4})$

No change



High Pass?

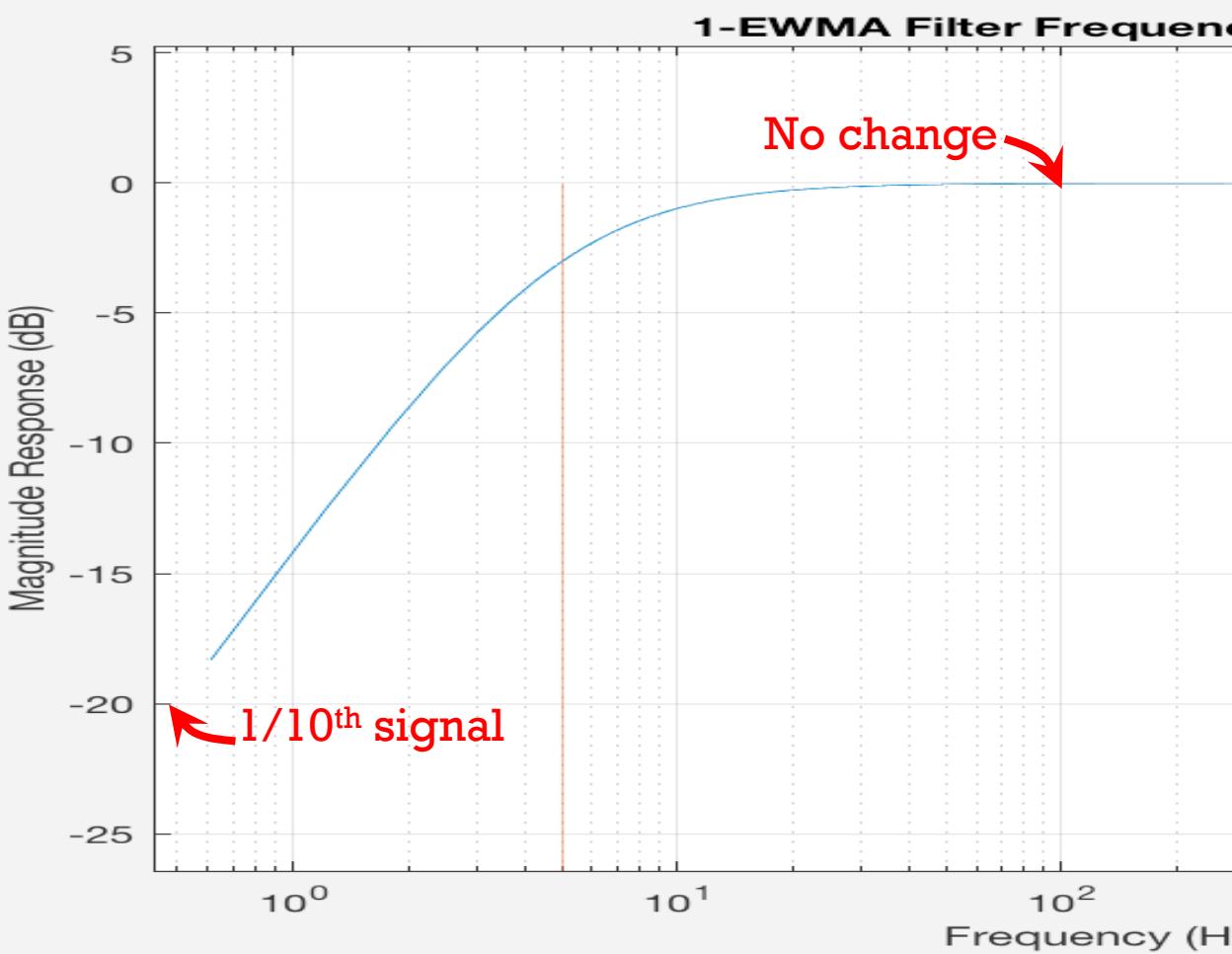
$F_s = 40000\text{Hz}$

$F_{3\text{db}} = 5\text{Hz}$

$\text{WEIGHT} = 1273$

$(\alpha = 7.85 \times 10^{-4})$

Plot of 1 – EMA response



High Pass Filter (just subtract lowpass)

```
int highpass(int newvalue,int WEIGHT){  
    static int lastvalue;  
  
    lastvalue = WEIGHT*lastvalue + newvalue)/(WEIGHT+1);  
    return newvalue - lastvalue;  
}
```

Example input	6	7	4	6	5	100	4	5	5
Running avg (weight = 1)	3	5	4	5	5	52	26	15	10
Highpass	3	2	0	1	0	48	-22	-10	-5

Band Stop AKA Notch Filter

$F_s = 40000\text{Hz}$

$F3db1 = 5\text{Hz}$

$\text{WEIGHT} = 1273$

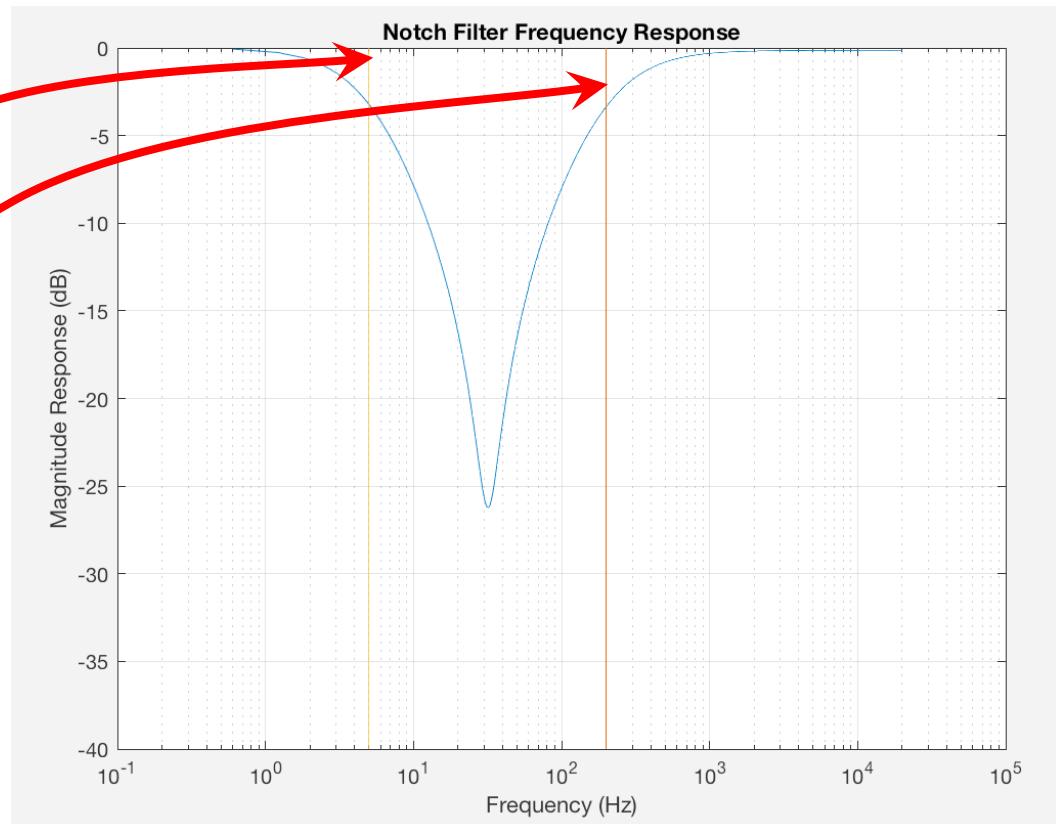
$(\alpha = 7.85 \text{ e-}4)$

$F3db1 = 200\text{Hz}$

$\text{WEIGHT} = 31$

$(\alpha = 0.0309)$

Plot of $\text{EMA2} - \text{EMA1}$ response



Band Pass?

$F_s = 40000\text{Hz}$

$F_{3\text{db}1} = 5\text{Hz}$

$\text{WEIGHT} = 1273$

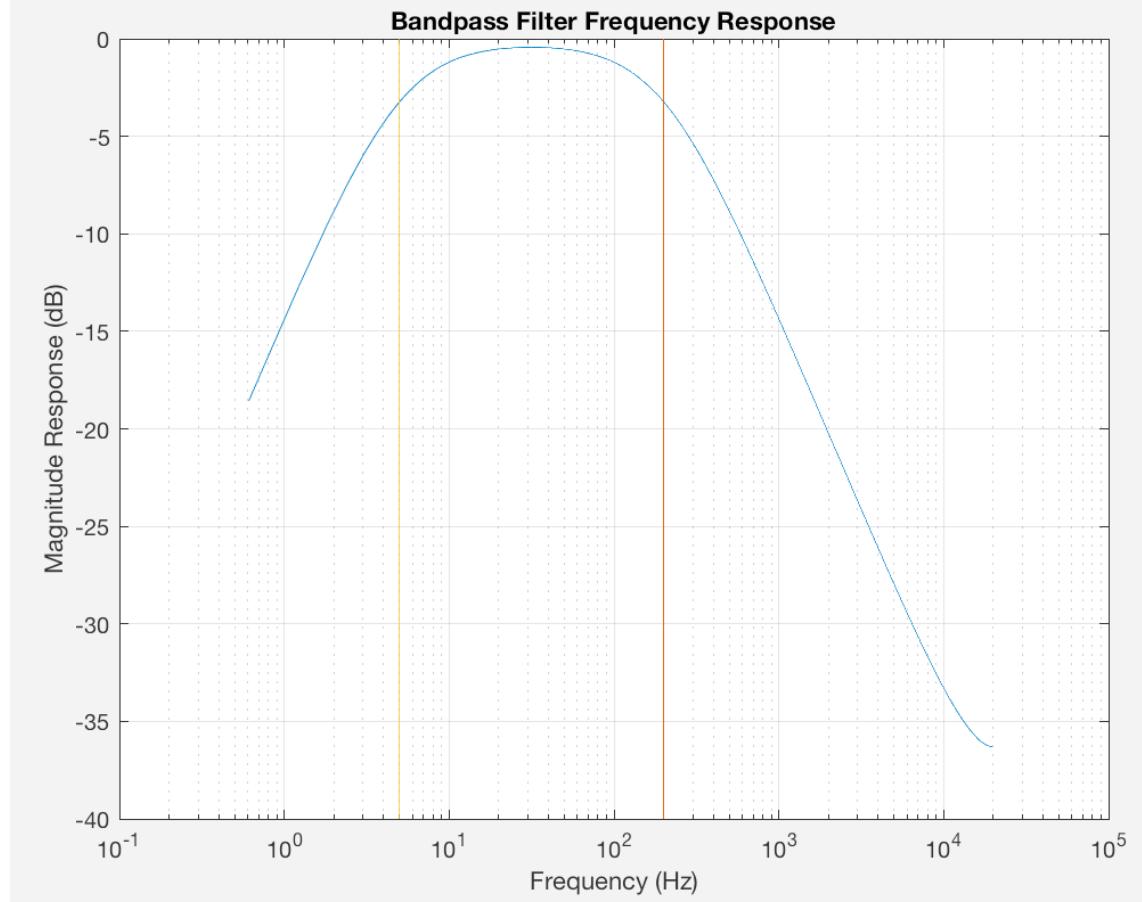
$(\alpha = 7.85 \text{ e-}4)$

$F_{3\text{db}1} = 200\text{Hz}$

$\text{WEIGHT} = 31$

$(\alpha = 0.0309)$

Plot of 1- Band stop



Designing for WEIGHT cutoff frequency

f_{3dB} is the corner (cutoff) frequency

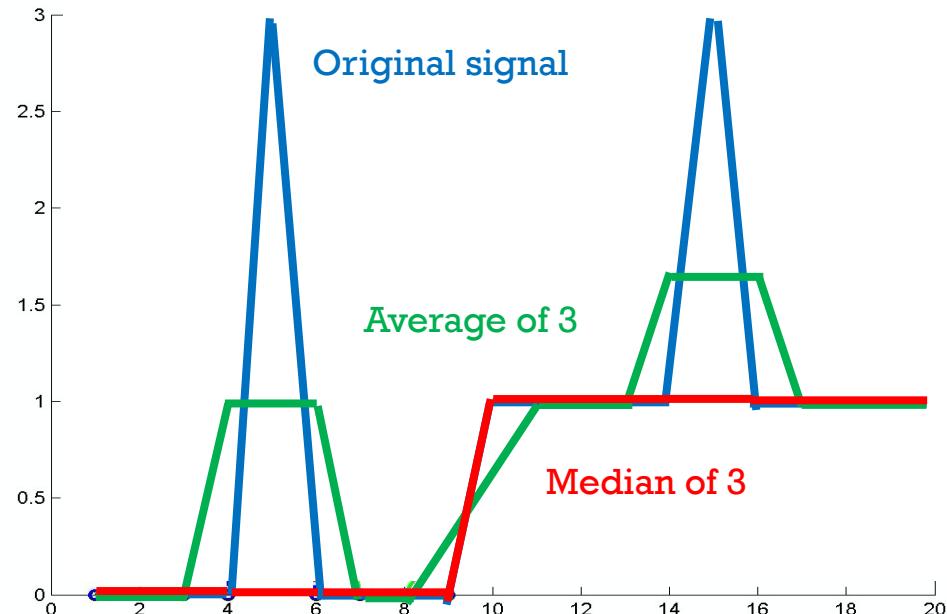
f_s is the sampling frequency

$$\Omega = \frac{2\pi}{f_s} f_{3dB} = \frac{2\pi}{40\text{KHz}} 60\text{Hz} = 0.0094$$

$$\alpha = \cos(\Omega) - 1 + \sqrt{\cos^2(\Omega) - 4 \cos(\Omega) + 3}$$

$$\text{WEIGHT} = (1 - \alpha)/\alpha = 105$$

Median filter



Example of median of 3 values:

$$x = [2 \ 80 \ 6 \ 3]$$

The median filtered output signal y will be:

$$y[1] = \text{Median}[2 \ 2 \ 80] = 2$$

$$y[2] = \text{Median}[2 \ 80 \ 6] = \text{Median}[2 \ 6 \ 80] = 6$$

$$y[3] = \text{Median}[80 \ 6 \ 3] = \text{Median}[3 \ 6 \ 80] = 6$$

$$y[4] = \text{Median}[6 \ 3 \ 3] = \text{Median}[3 \ 3 \ 6] = 3$$

$$\text{i.e. } y = [2 \ 6 \ 6 \ 3].$$

Simplest Code Examples

```
int med3filt(int a, int b, int c) {  
    int middle;  
    if ((a <= b) && (a <= c))  
        middle = (b <= c) ? b : c;  
    else if ((b <= a) && (b <= c))  
        middle = (a <= c) ? a : c;  
    else    middle = (a <= b) ? a : b;  
    return middle;  
}
```

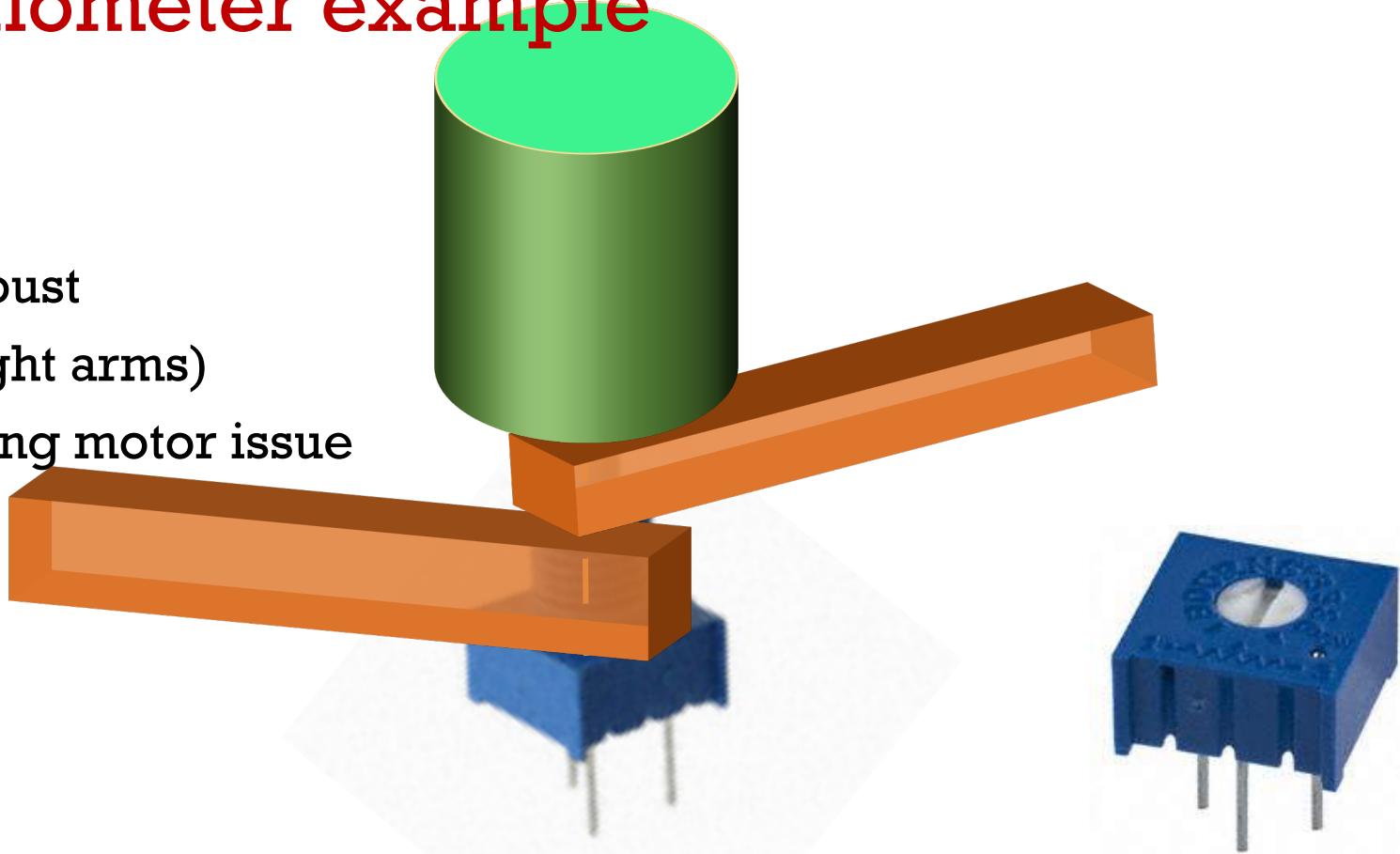
```
int movingavg(int newvalue, int WEIGHT){  
    static int lastvalue;  
    lastvalue = (WEIGHT*lastvalue + newvalue)/(WEIGHT+1);  
    return lastvalue;  
}
```

05

Rotational Position Sensors (options for LAB 3)

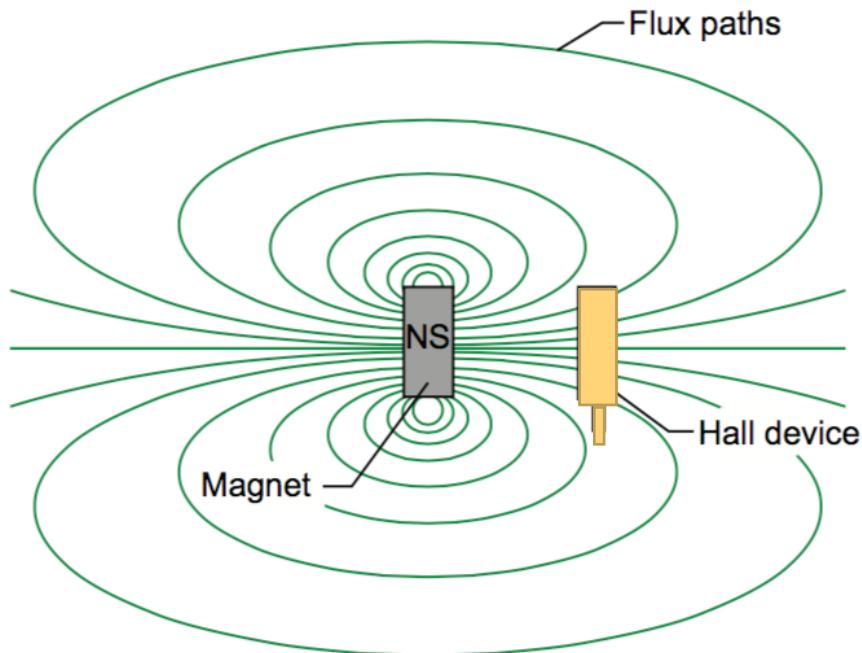
Potentiometer example

- Simple
- Not robust
- (use light arms)
- Mounting motor issue

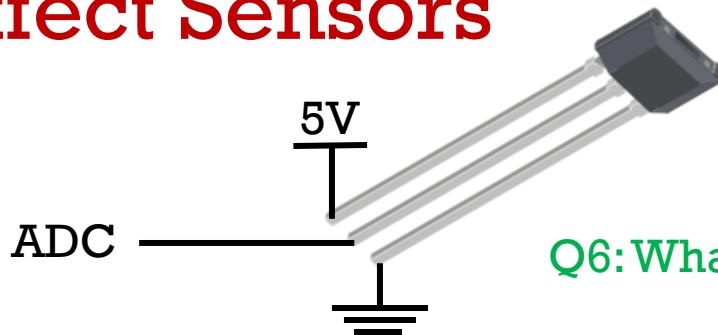


Hall Effect Sensor

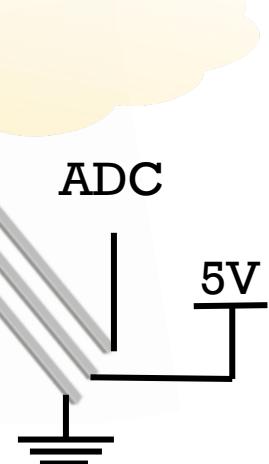
- Voltage proportional to applied magnetic field
- If the magnet rotates, the field will change correspondingly (non-linear, somewhat sinusoidal).



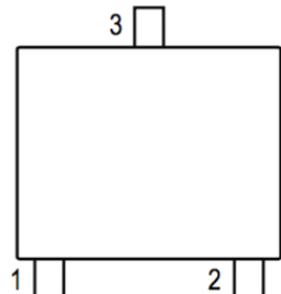
Hall Effect Sensors



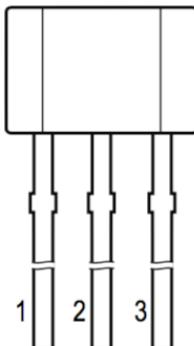
Q6: What's wrong with this?



Pin-out Diagrams



LH Package



UA Package

Terminal List Table

Name	Number		Function
	LH	UA	
VCC	1	1	Input power supply; tie to GND with bypass capacitor
VOUT	2	3	Output signal; also used for programming
GND	3	2	Ground

Summary

- ADCs setup with registers and are slow.
- Teensy ADC has 10 bit resolution
- ADC sensitivity can change by changing Vref
- Potentiometers make easy angle sensors if you can mount to them
- Simple software filters can act as high pass or low pass filters but add a delay

Answer in CHAT

Answer how you feel about each topic below with:

1. I don't understand this topic at all
2. I don't know now, but know what to do to get by
3. I understand some, but expect to get the rest later
4. I understand completely already

- A. Software filters
- B. ADC on Teensy
- C. Rotational sensors