Lecture 08

ADC Software Filters

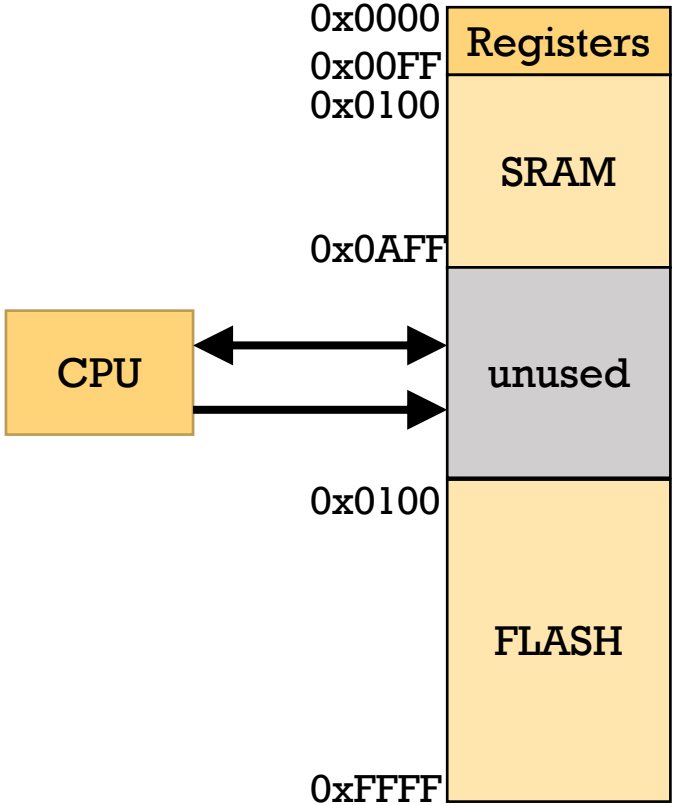
Agenda

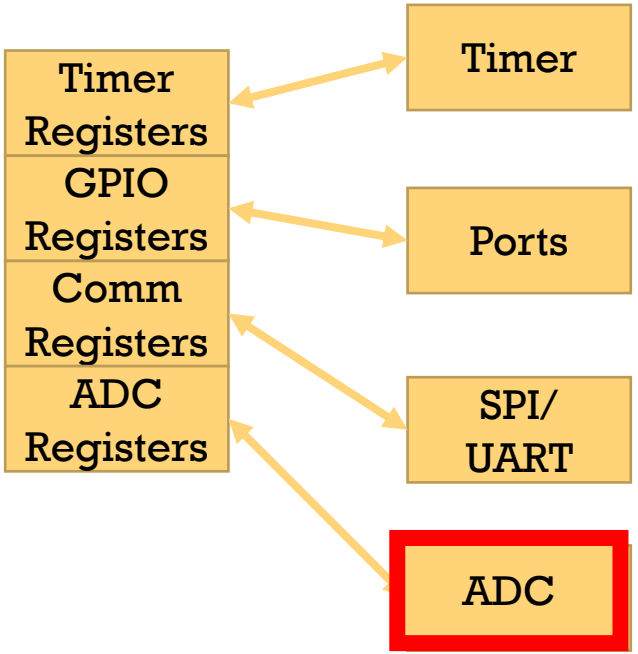
01. Analog to Digital Converter (ADC)
02. ADC on ATmega 32U4 (Teensy)
03. Software filters
04. Analog rotational position sensors (if time)

01

Analog to Digital Converter (ADC)

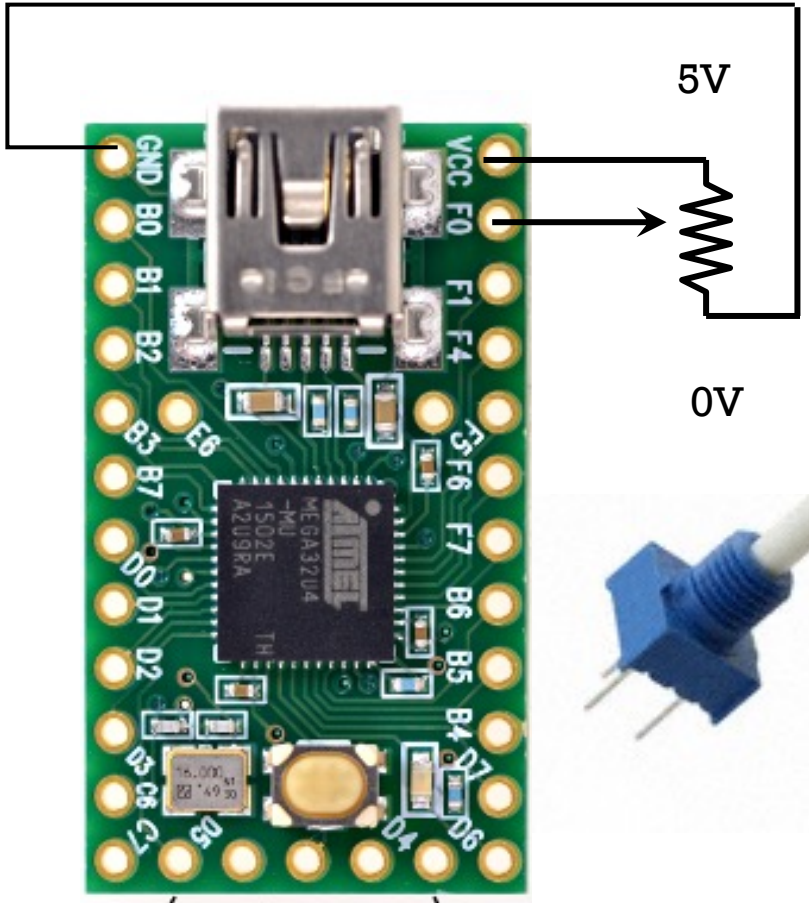
Peripherals run concurrent with main code





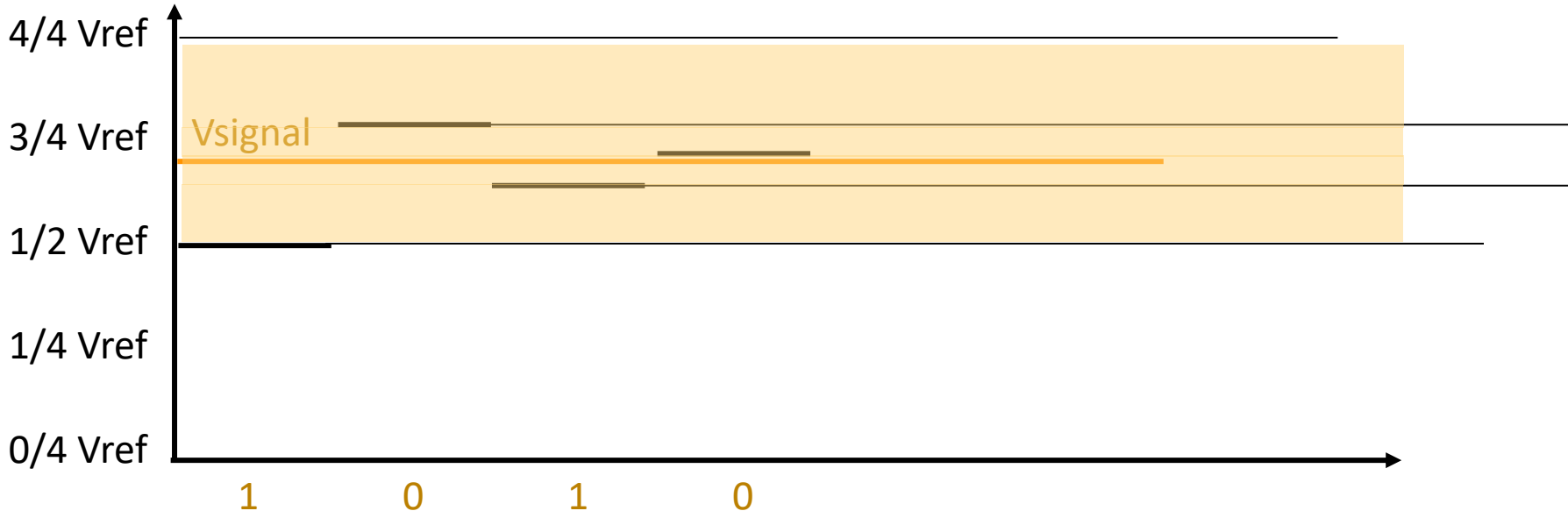
What and why?

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



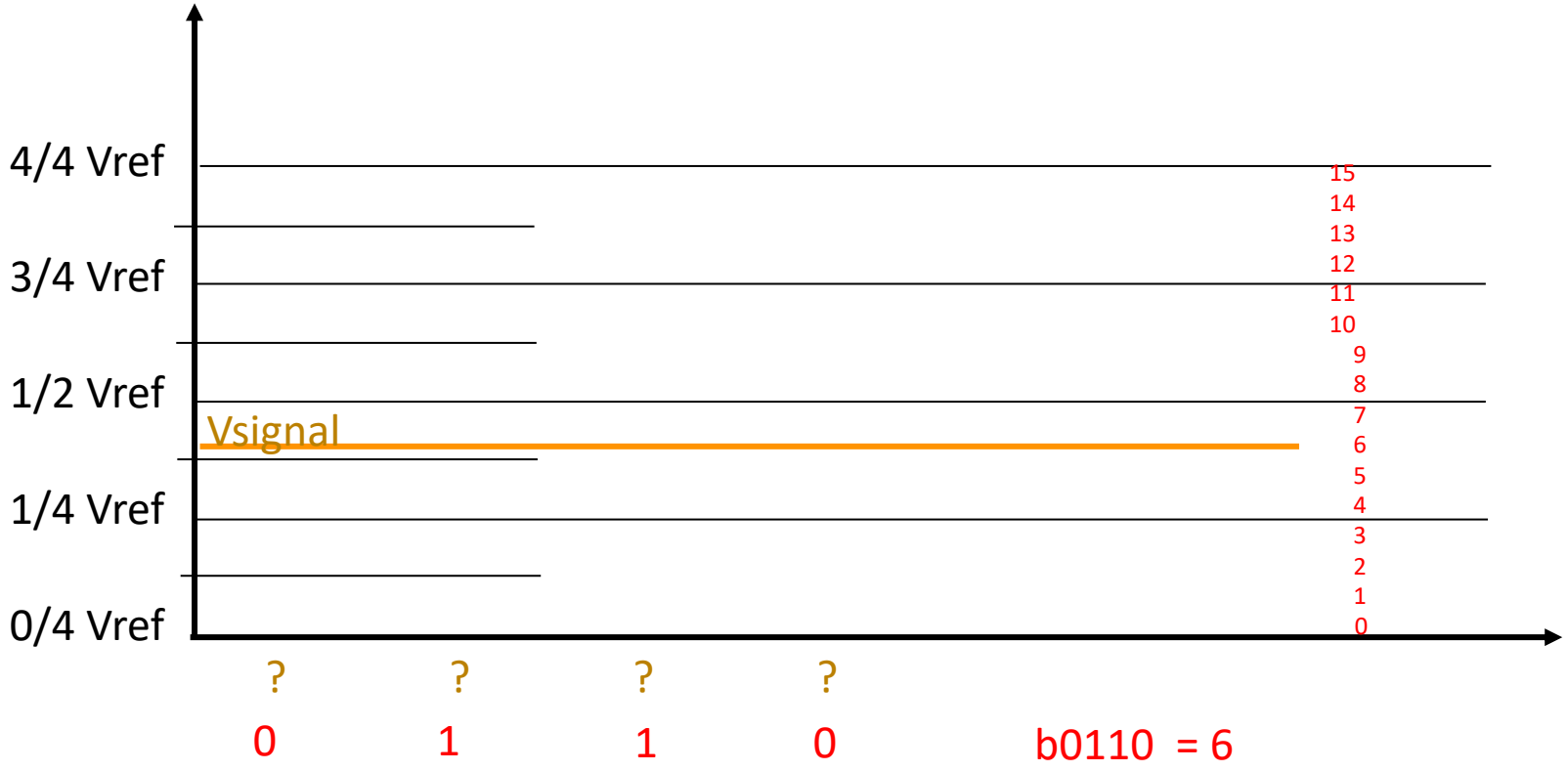
Ex: Successive Approximation (SAR)

Ask successively: "Which half is the signal in?"



Note: Each resolution bit takes an evaluation cycle. Higher resolution takes more time

Q1: What are the 4 bits from successive approximation?



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
- ±2LSB Absolute Accuracy
- 65 - 260µs 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.

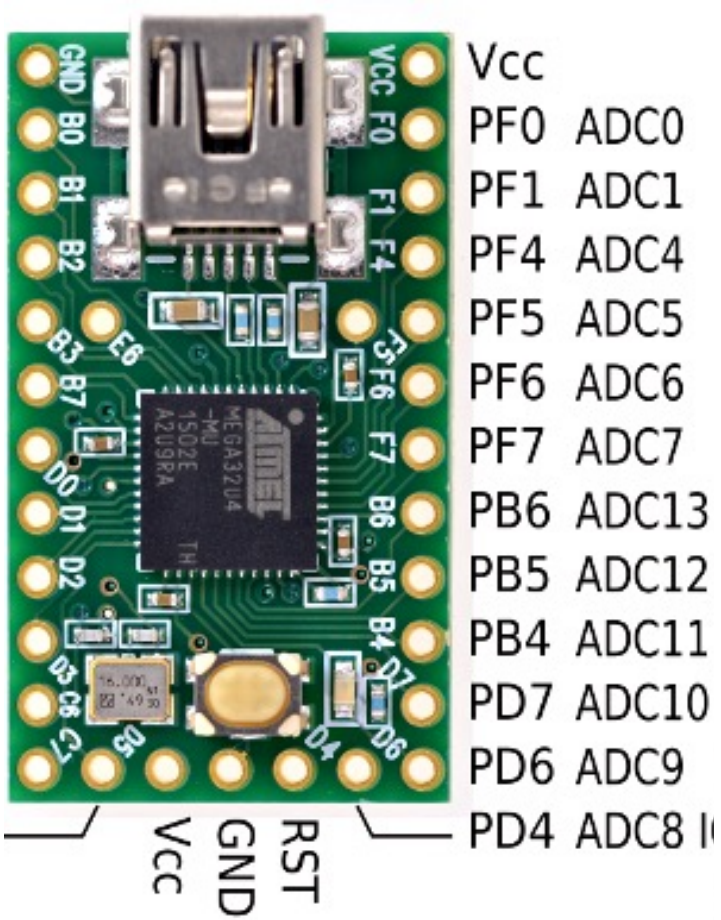
The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown in Figure 24-1.

The ADC has a separate analog supply voltage pin, AV_{CC}. AV_{CC} must not differ more than ± 0.3V 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

Setup registers

Get reading from one pin

repeat

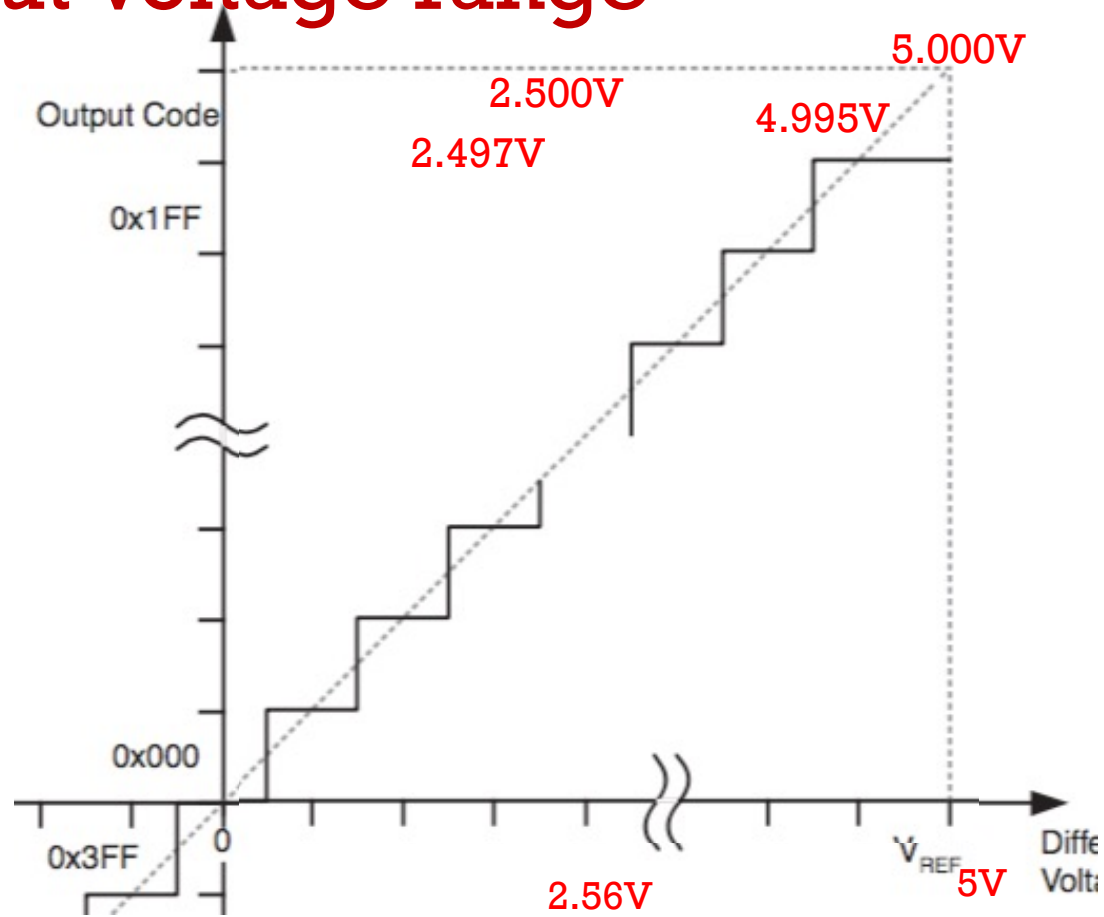
All of this using registers.

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



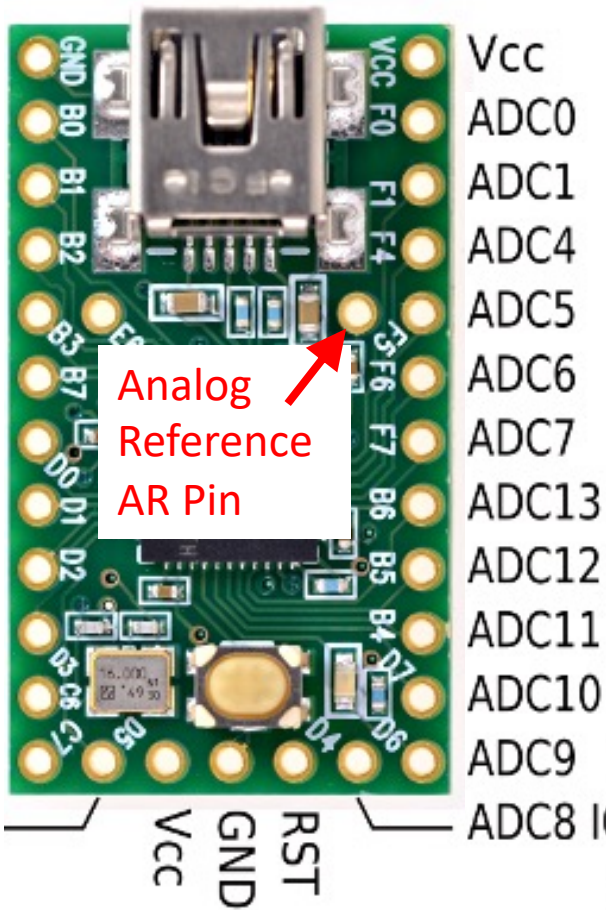
ADC Reference

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 for 16MHz system

Notes:

Analog read is slow…
Takes 25 ADC clock 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	
	ADC7D	ADC6D	ADC5D	ADC4D	-	-	ADC1D	ADC0D	DIDR0
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	

- **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	
	-	-	ADC13D	ADC12D	ADC11D	ADC10D	ADC9D	ADC8D	DIDR2
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	

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

Q2: setup ADC to use ADC9

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

7	6	5	4	3	2	1	0	
-	-	ADC13D	ADC12D	ADC11D	ADC10D	ADC9D	ADC8D	DIDR2
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	
ADC7D	ADC6D	ADC5D	ADC4D	-	-	ADC1D	ADC0D	DIDR0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Q2: setup ADC to use ADC9

1. Use VCC for Vref. `set(ADMUX, REFS0);`
2. Set prescaler /128

```
set(ADCSRA, ADPS0);
set(ADCSRA, ADPS1);
set(ADCSRA, ADPS2);
```

1. Disable digital input for ADC0 (the pin we decide to use)

```
set(DIDR2, ADC9D);
```

Step 5: Choose ADC channel

(skip step 4 for now)

Q3: Use set and clear to choose ADC9

ADCSRB: 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

Bit	7	6	5	4	3	2	1	0	
	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	

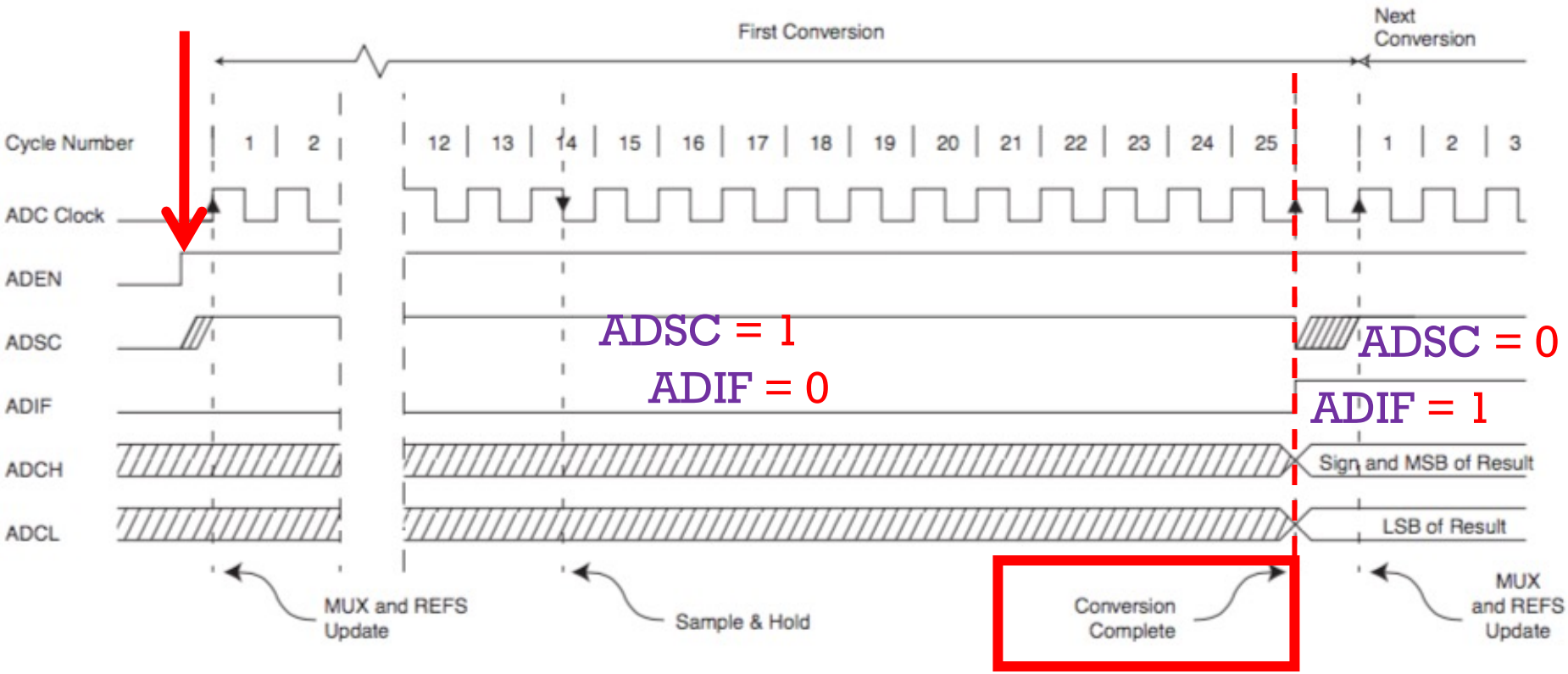
Step 6 to 10: Read ADC values

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 to print

ADC single conversion timing diagram

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



ADC demo

```
int main(void){
  ADC_setup(ADC0); // My subroutine to initialize the ADC registers
  m_usb_init(); // Subroutine to initialize USB communication

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

      m_usb_tx_string("\rADC = ");  // \r = carriage return,
      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 */
}
```

ADC free-running

- Setup stuff:
  - Prescaler and/or Voltage Reference
  - choose which channels to read
  - set DIDR bits
  - set Autotrigger (ADCSRA:ADATE) and/or enable interrupts (ADIE)
  - set first channel
  - If using interrupts enable global (sei();)
- Interrupt based calls when multiplexing channels
  - Read / store value in appropriate channel variable
  - select next channel to read

ADC Multiplexing (multiple channels)

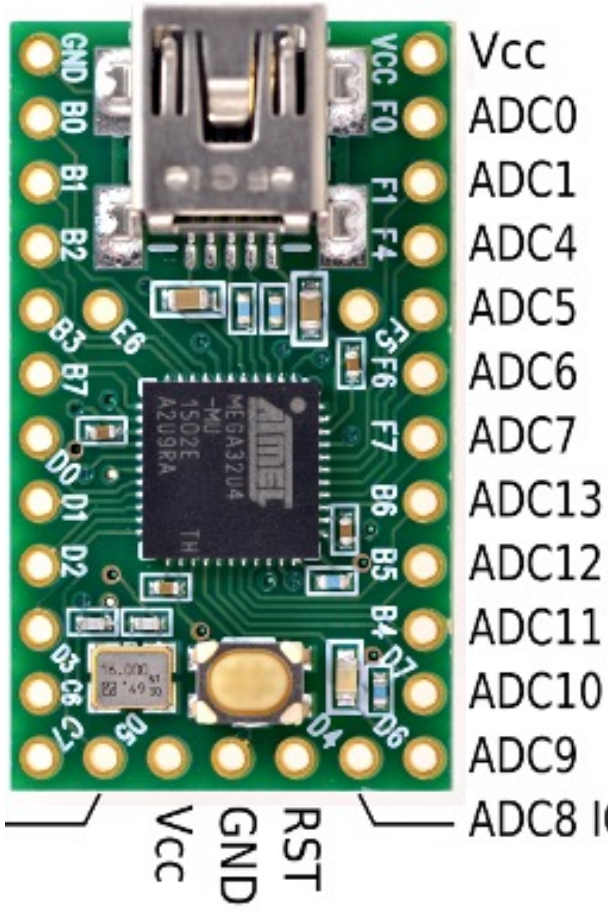
- Simplest using free run mode

  Setup

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

  Loop at ~5Khz

  4. disable conversion (clear ADEN)
  5. select the analog channel (change to new one if desired ADMUX)
  6. enable conversions (set ADEN)
  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

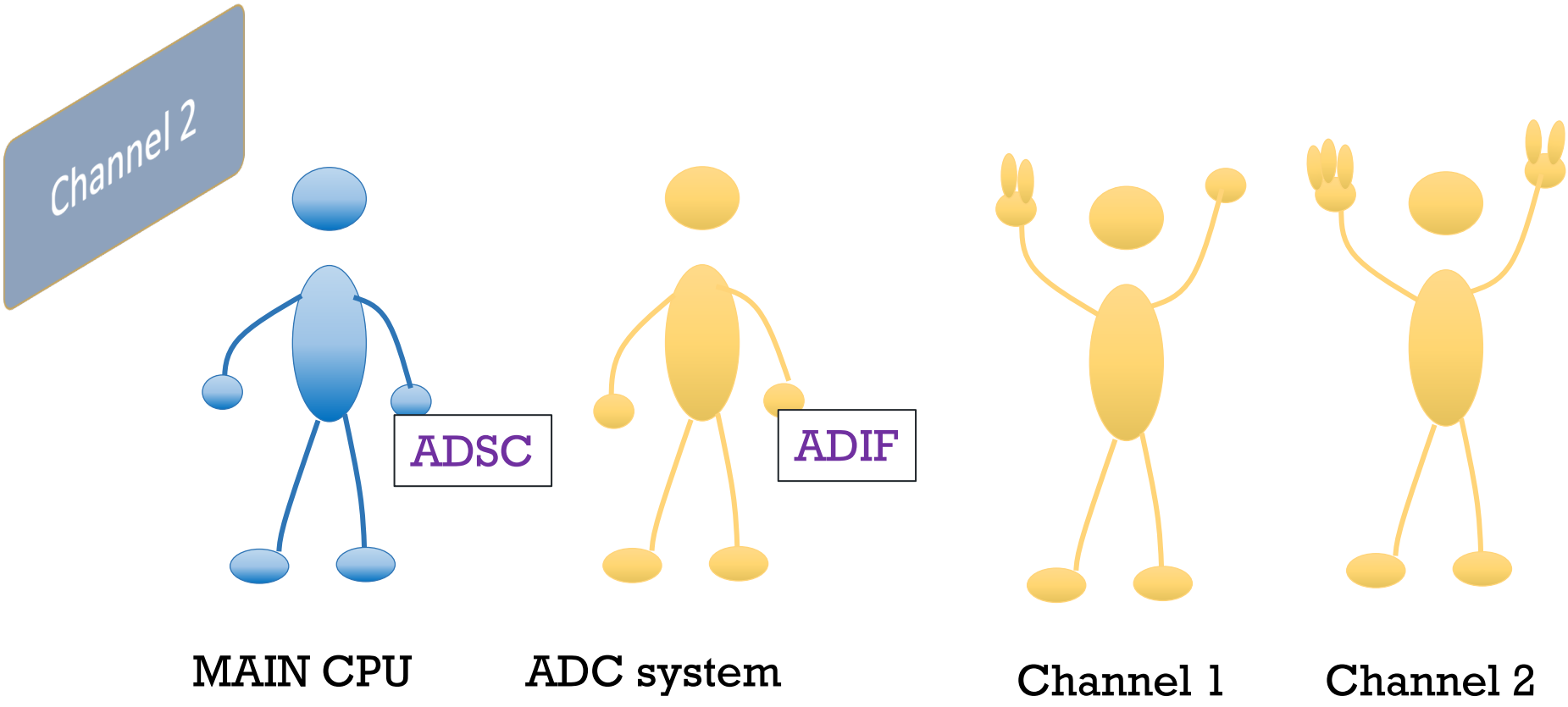
  1. set the voltage reference
  2. set the ADC clock prescaler
  3. disable some digital inputs
  4. select the analog input for 1st read

  Loop @ ~10Khz

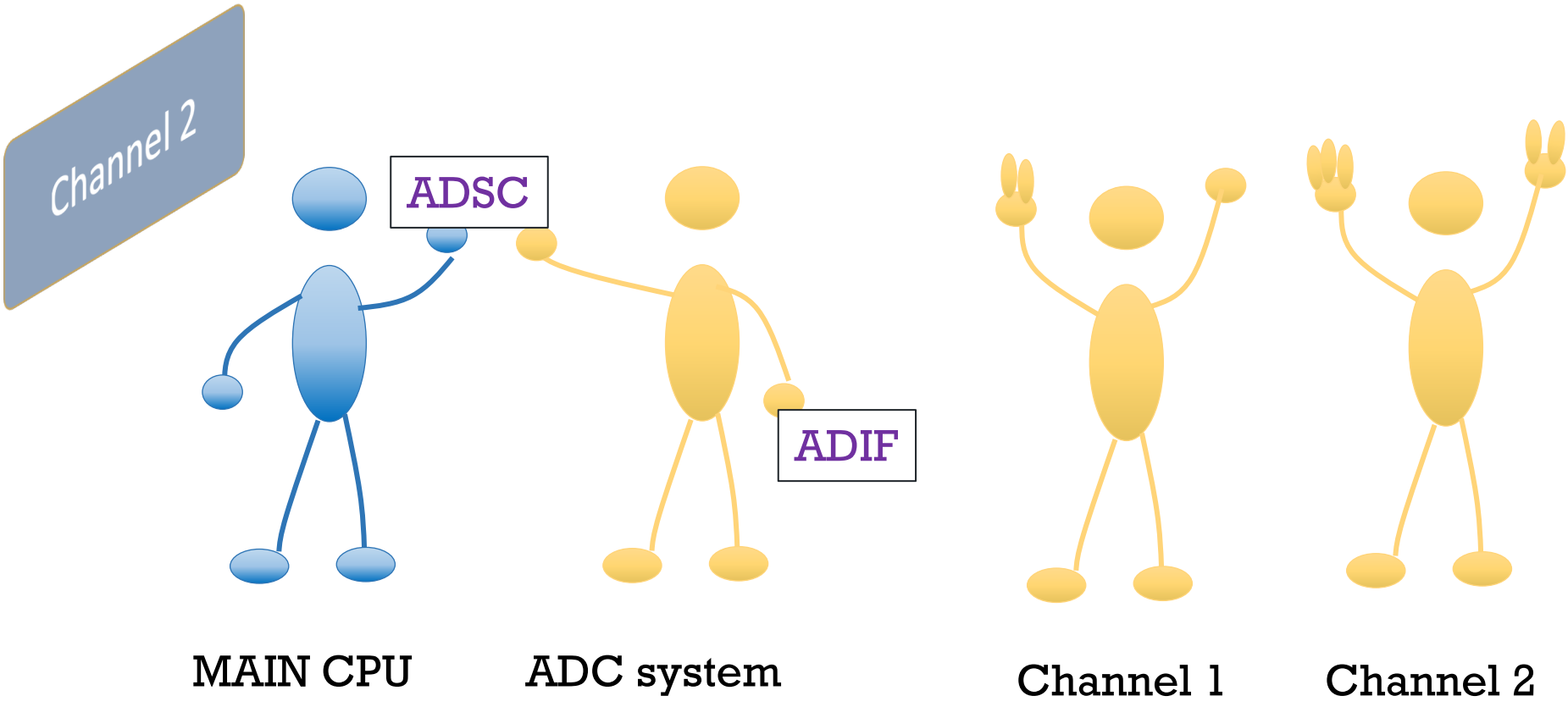
  5. start conversion for nth read (set ADSC)
  4. select/change analog input for (nth+1) read
  6. wait for conversion to finish
  7. read nth result

see Atmel docs starting at p 300 for more info.

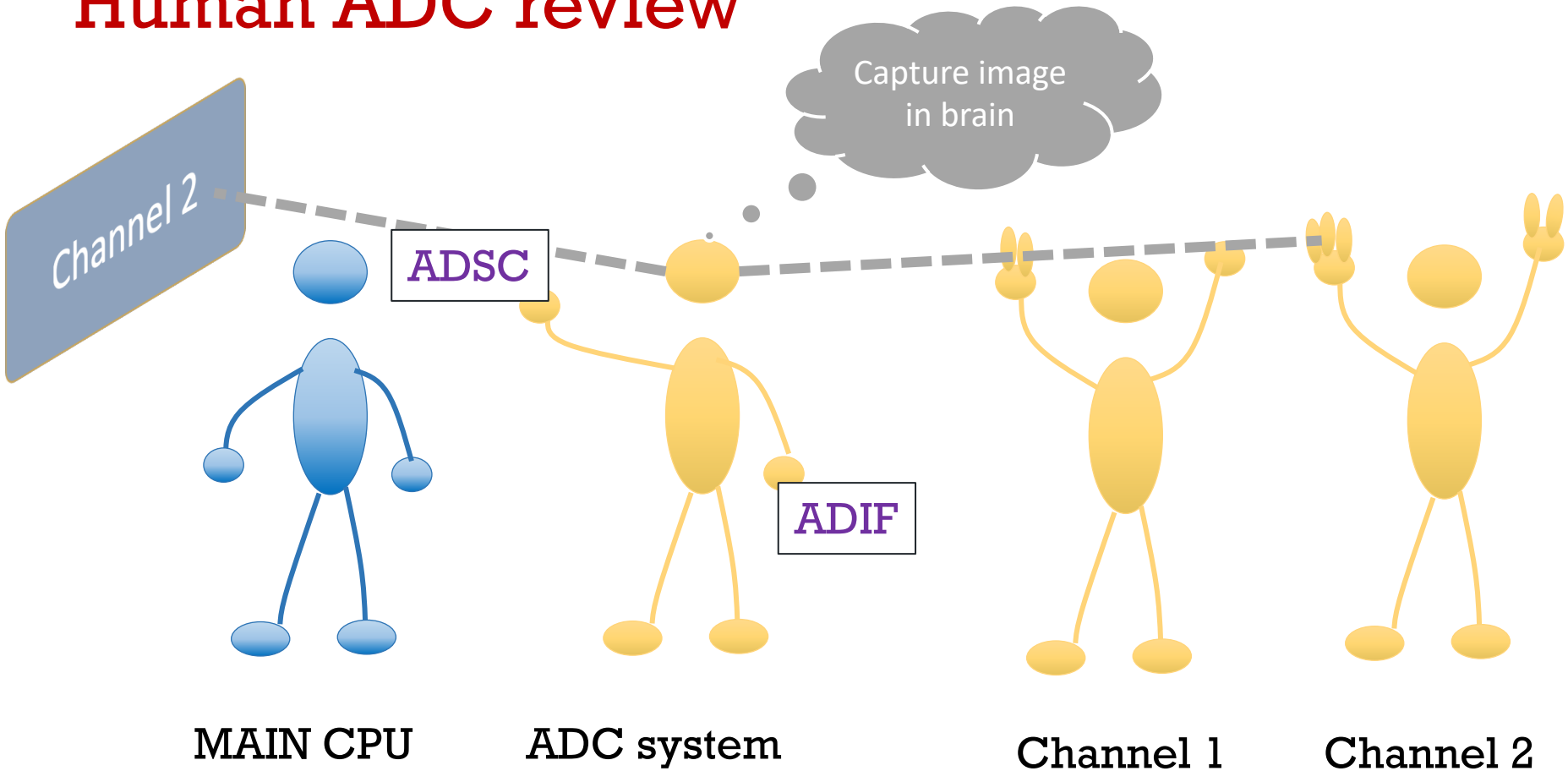
Human ADC review



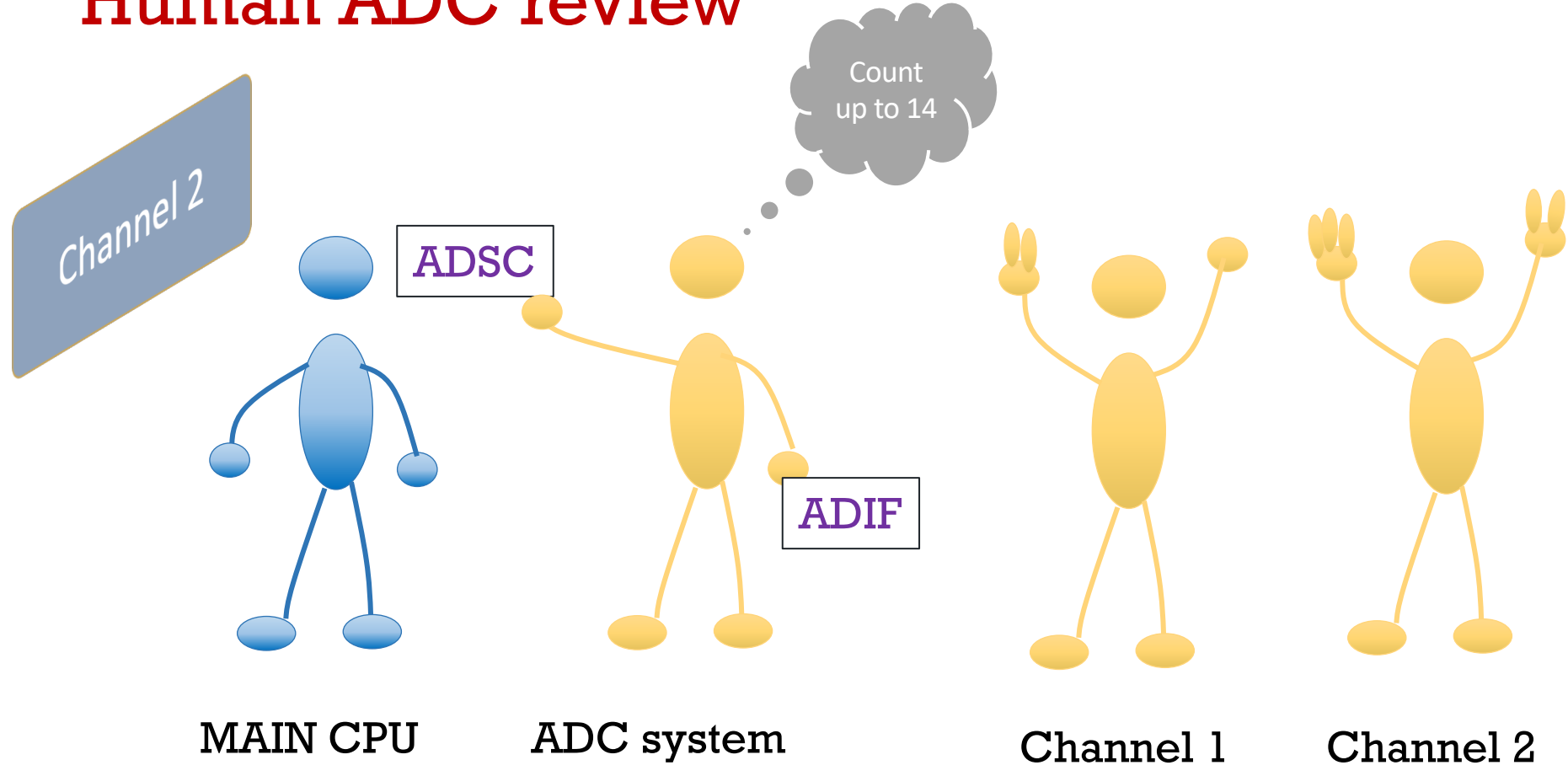
Human ADC review



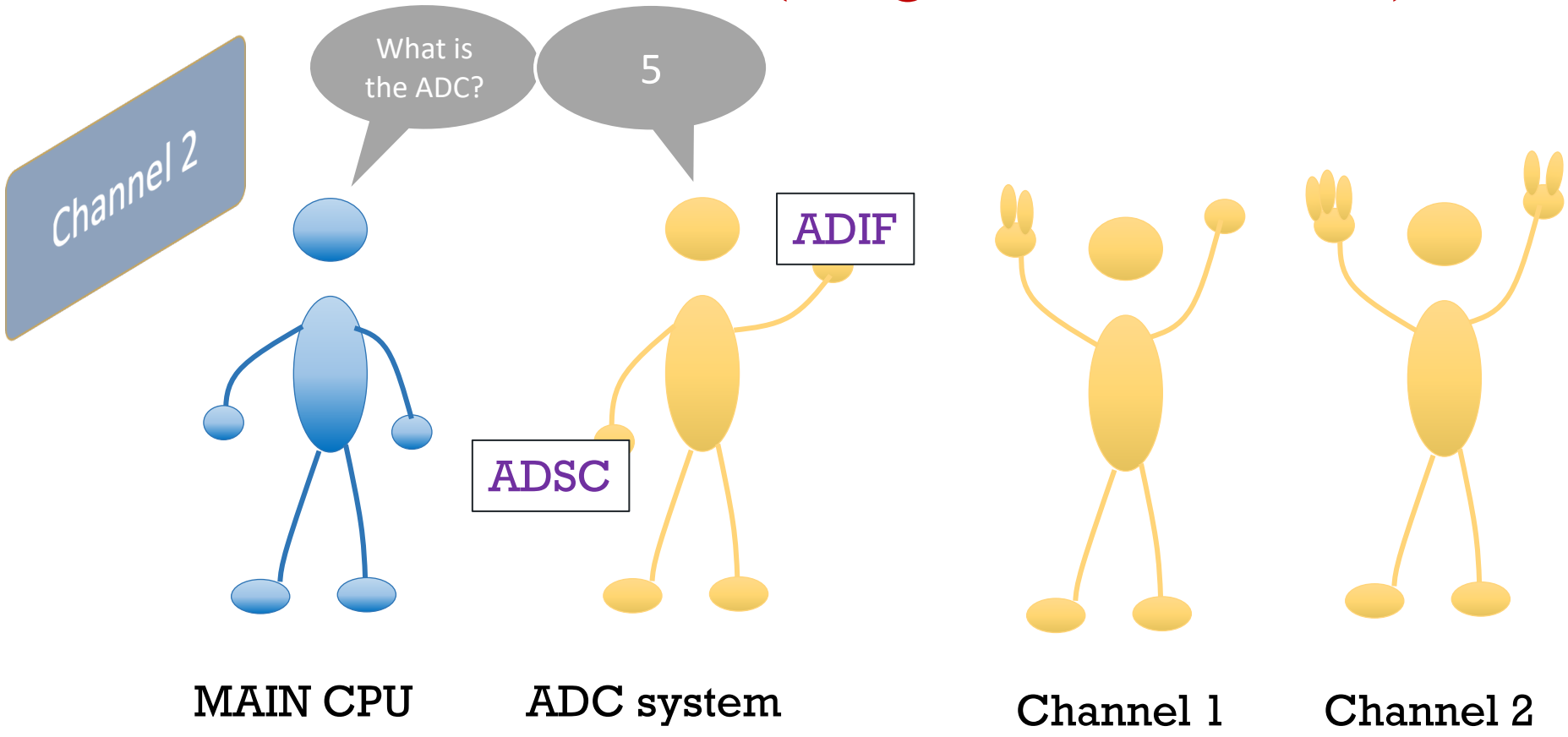
Human ADC review



Human ADC review

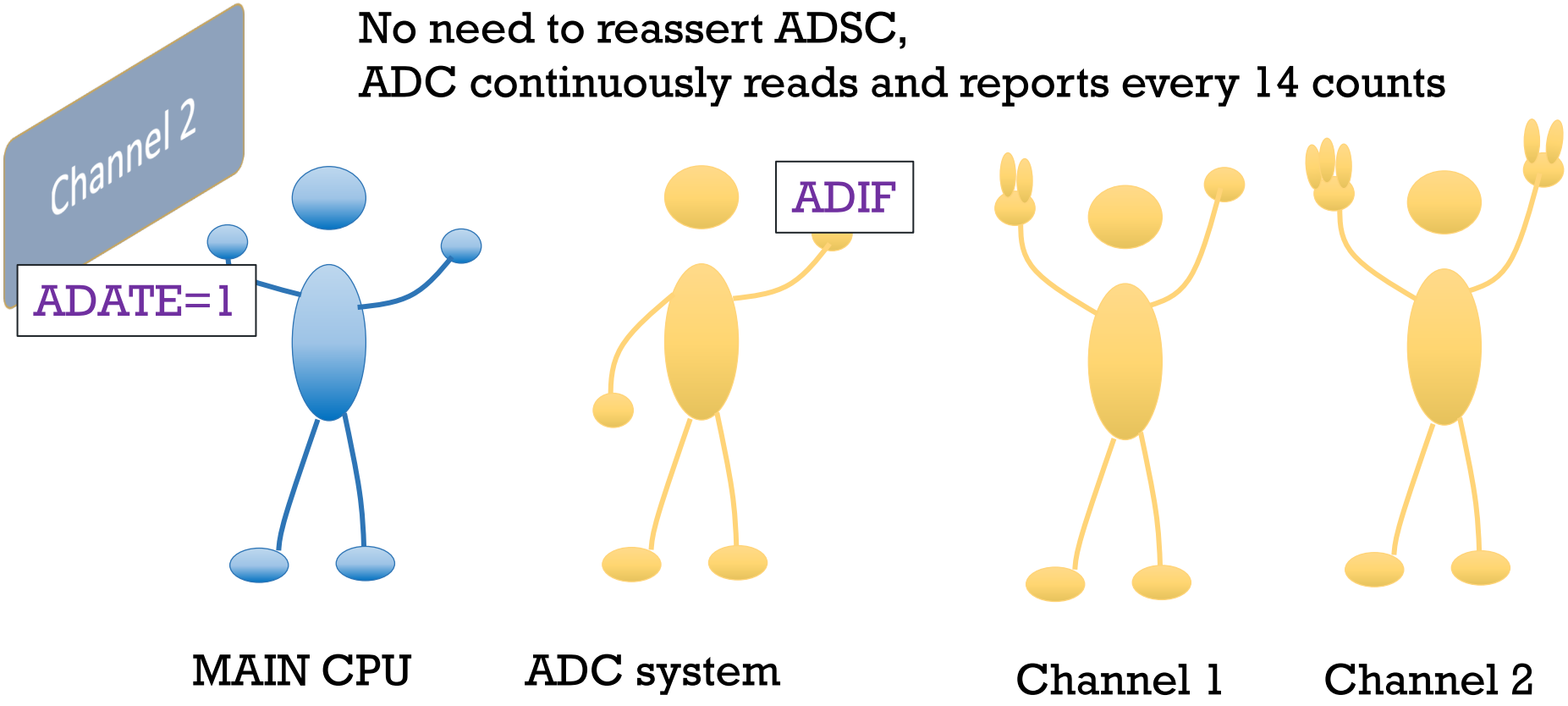


Human ADC review (single conversion)



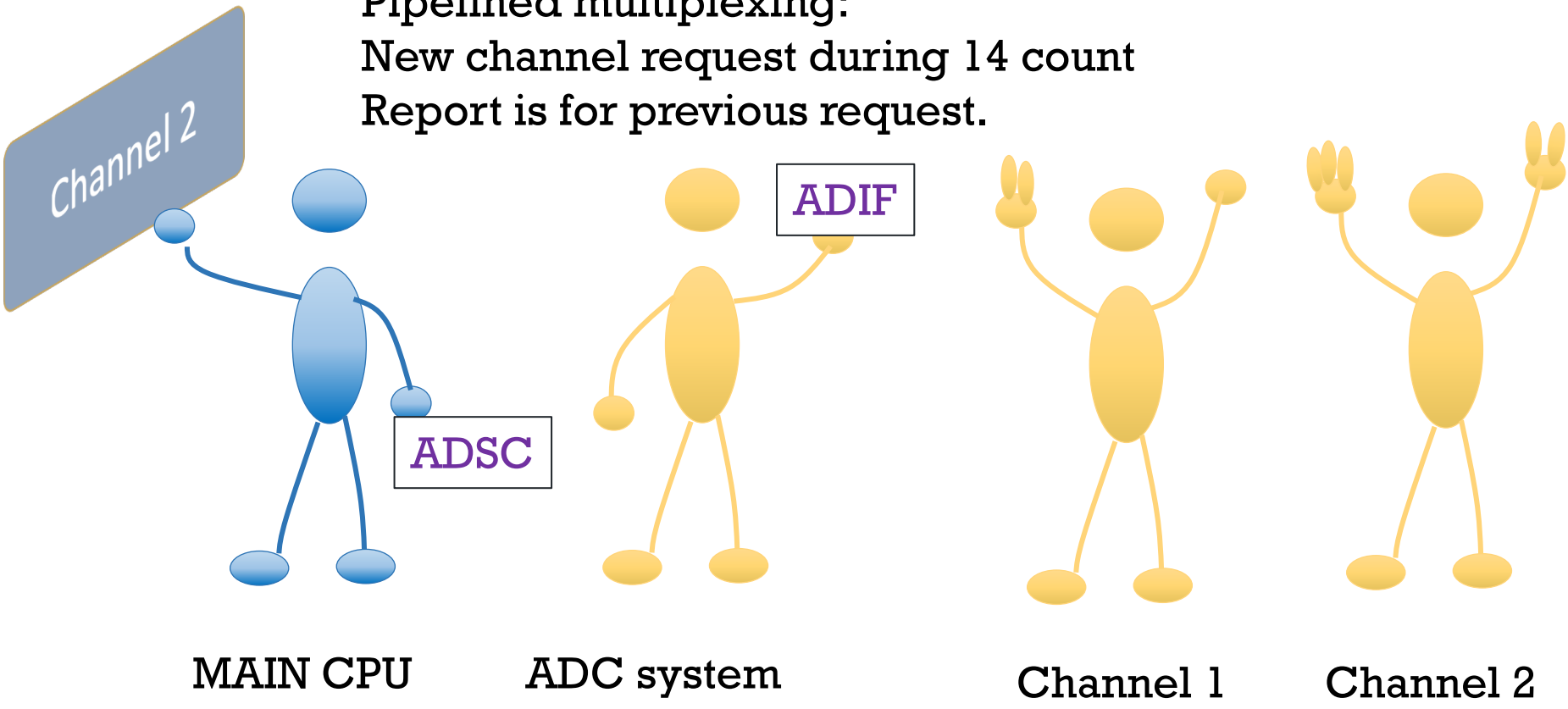
Human ADC review (free-running)

No need to reassert ADSC,
ADC continuously reads and reports every 14 counts



Human ADC review (multiplexing)

Pipelined multiplexing:
New channel request during 14 count
Report is for previous request.



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(DlDRO, ADC1D);  // disable digital input
    set(ADMUX, MUXO);   // pin f1
    clear(ADMUX, MUX1);
    clear(ADMUX, MUX2);
    clear(ADCSR8, MUX5);
  }
  if(ADCchannel == 4)    {
    set(DIDRO, ADC4D);  // disable digital input
    clear(ADMUX, MUXO); // pin f4
    clear(ADMUX, MUXl);
    set(ADMUX, MUX2)
    clear(ADCSR8, MUX5);
  }
...   if(ADCchannel == 5) ...
...   if(ADCchannel == 6) ...
...   // up to ADCchannel 13
...   // do other things like enable ADEN, ADATE if free running, prescaler etc.

  }
```

THIS CODE HAS ERRORS
DO NOT COPY AND PASTE
FOR LAB 3!!

Code simplicity (maybe elegance)

```
void setDIDR(unsigned char ADCchannel)
{

}
```

Byte ordering is wrong :(

ADC0D = 0
ADC1D = 1
ADC4D = 4
ADC5D = 5
ADC6D = 6
ADC7D = 8

Is there one line that can set one DIDR bit for a corresponding ADC channel (ADC0 to ADC7) using bit operations?

(0x7E)	DIDR0	ADC7D	ADC6D	ADC5D	ADC4D	-	-	ADC1D	ADC0D
Addr	NAME	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

Low byte
High byte

DIDR

Register map

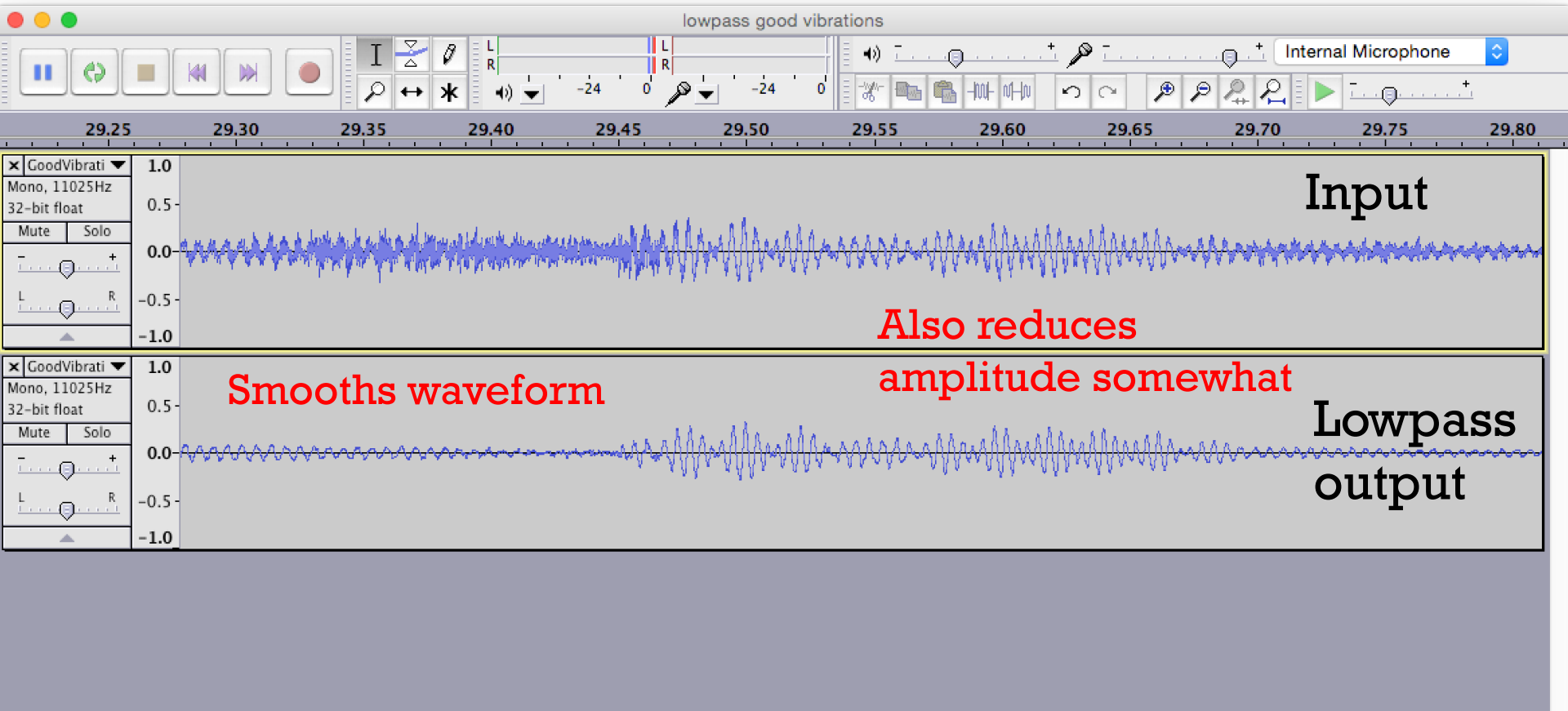
ACD Summary

- ADC Modes: single conversion, free-running, multiplexing, interrupt-driven
- Set which channel in ADMUX (plus ADCSRB)
- Control in ADCSRA register
  - ADEN enables ADC
  - ADSC starts conversion
  - ADIF indicates conversion complete
- Read 10-bit value in ADC register

03

Software Filters

Audio filters work same



Simple software low pass software filter

Q5: 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

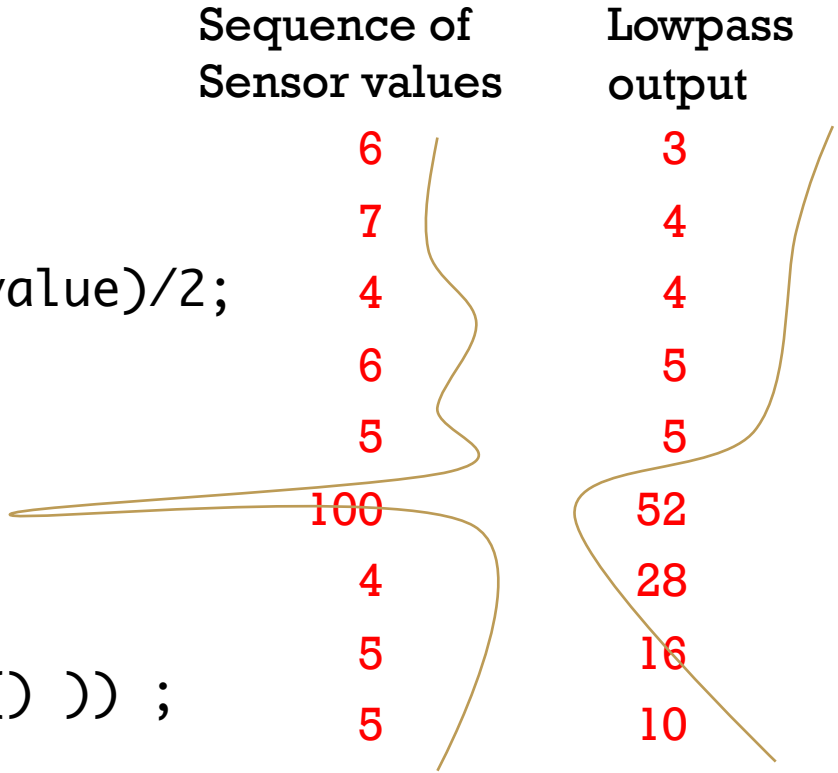
Use subroutines

```
int lowpass(int newvalue) {
  static int lastvalue;

  lastvalue = (newvalue + lastvalue)/2;

  return lastvalue;
}

int main() {
  for(;;){
    PRINT (lowpass( readSensor() )) ;
  }
}
```



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);
```

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

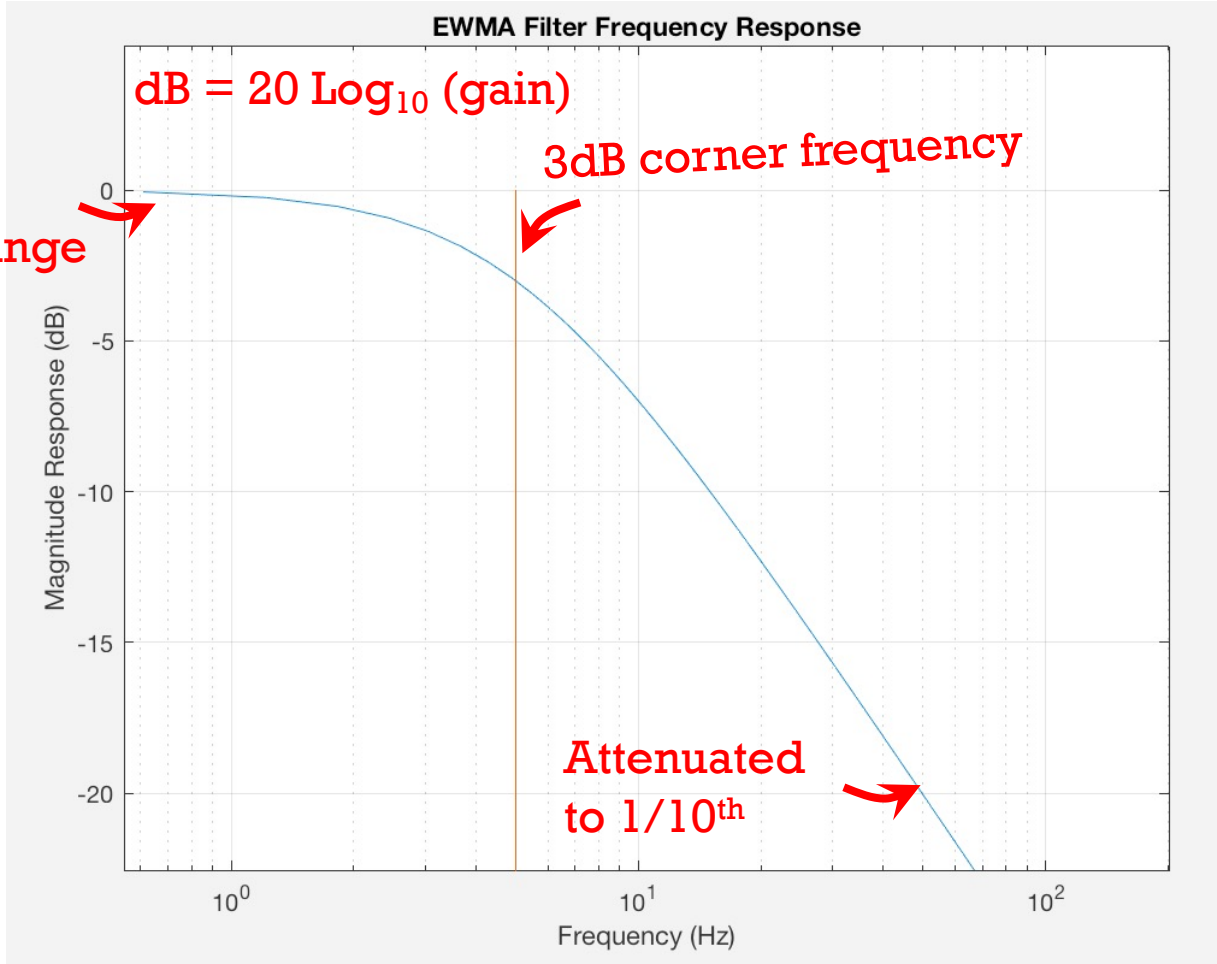
f_{3dB} **is the corner (cutoff) frequency**
f_s **is the sampling frequency**
α **is** `1-(WEIGHT/(WEIGHT+1))` **in the previous code**
`WEIGHT` $= (1 - \alpha)/\alpha$

https://dsp.stackexchange.com/questions/40462/exponential-moving-average-cut-off-frequency

EMA Example

Fs = 40000Hz
F3db = 5Hz
WEIGHT = 1273
(α = 7.85 e-4)

Q6: in chat: What kind of filter is this?

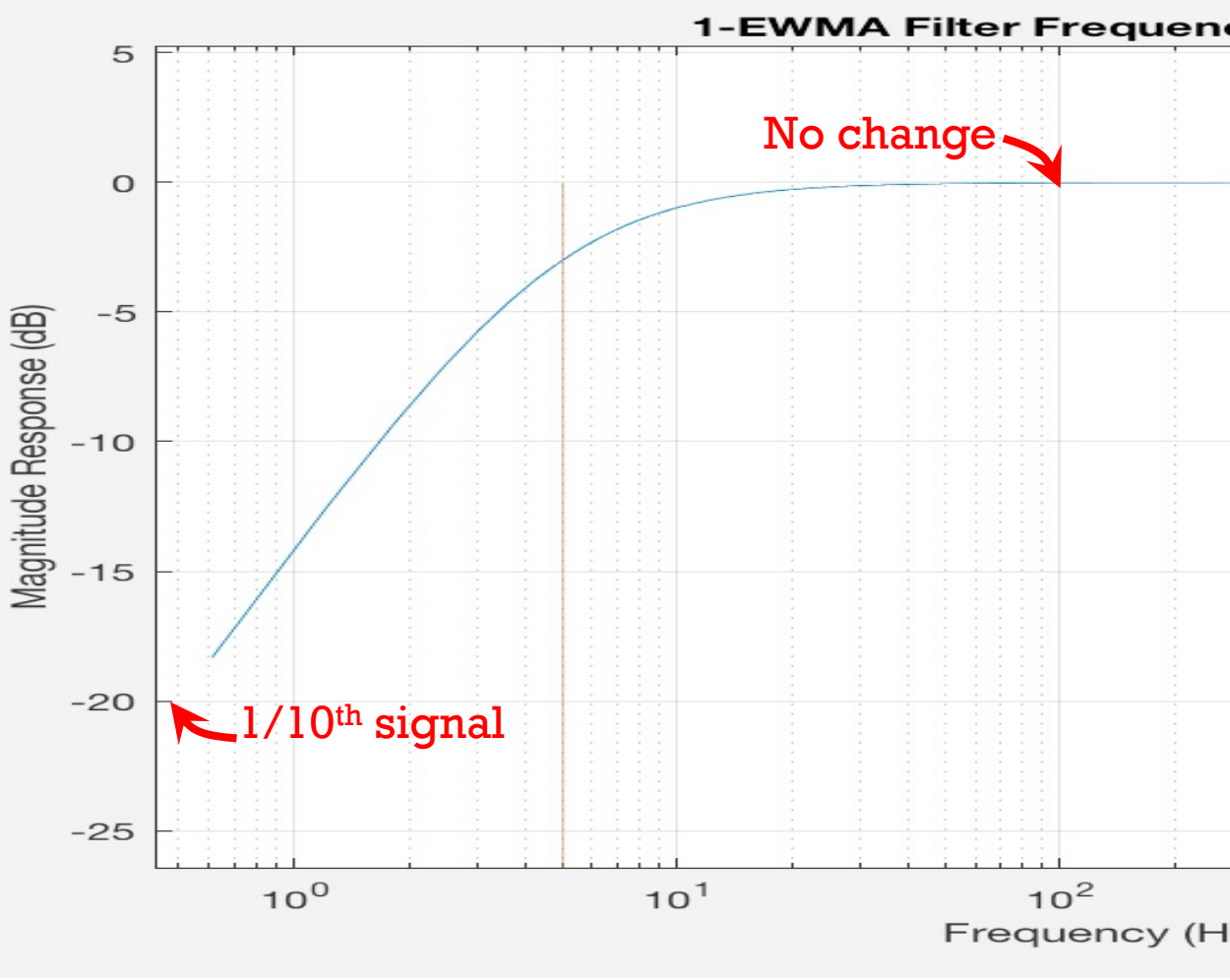


Log / Log plot

High Pass?

Fs = 40000Hz
F3db = 5Hz
WEIGHT = 1273
(α = 7.85 e-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

https://www.norwegiancreations.com/2016/03/arduino-tutorial-simple-high-pass-band-pass-and-band-stop-filtering/

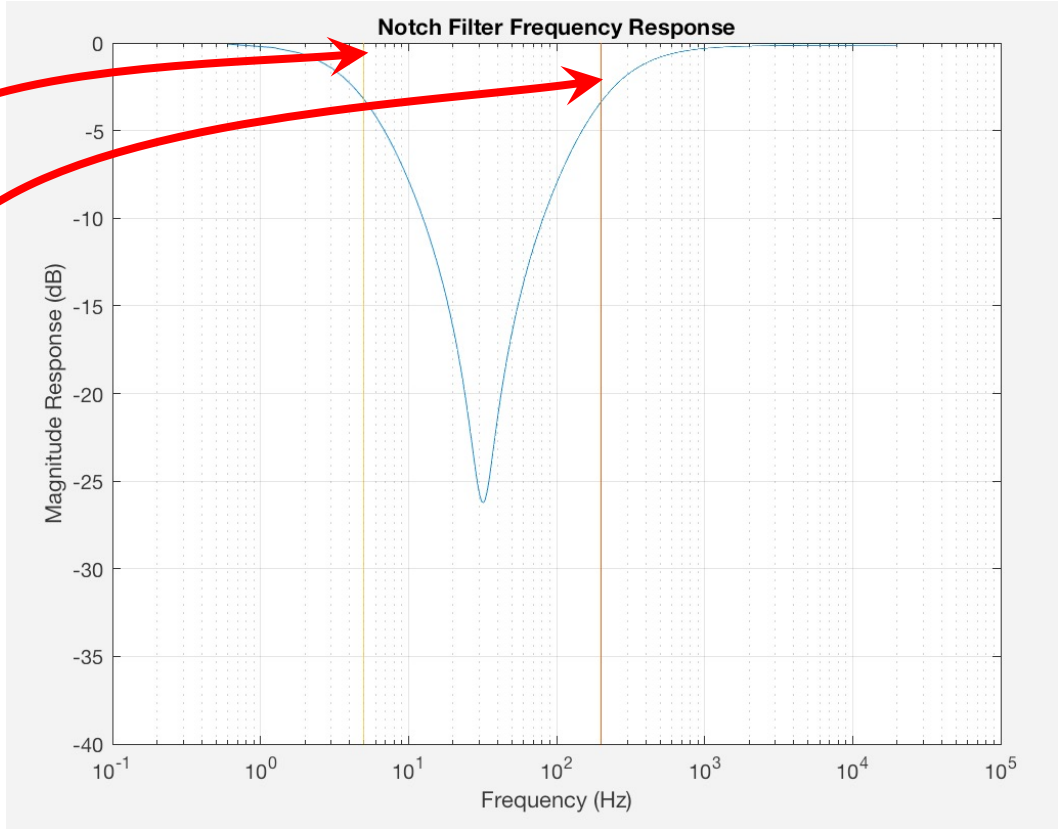
Band Stop AKA Notch Filter

Fs = 40000Hz

F3db1 = 5Hz
WEIGHT = 1273
(α = 7.85 e-4)

F3db1 = 200Hz
WEIGHT = 31
(α = 0.0309)

Plot of EMA2 – EMA1 response



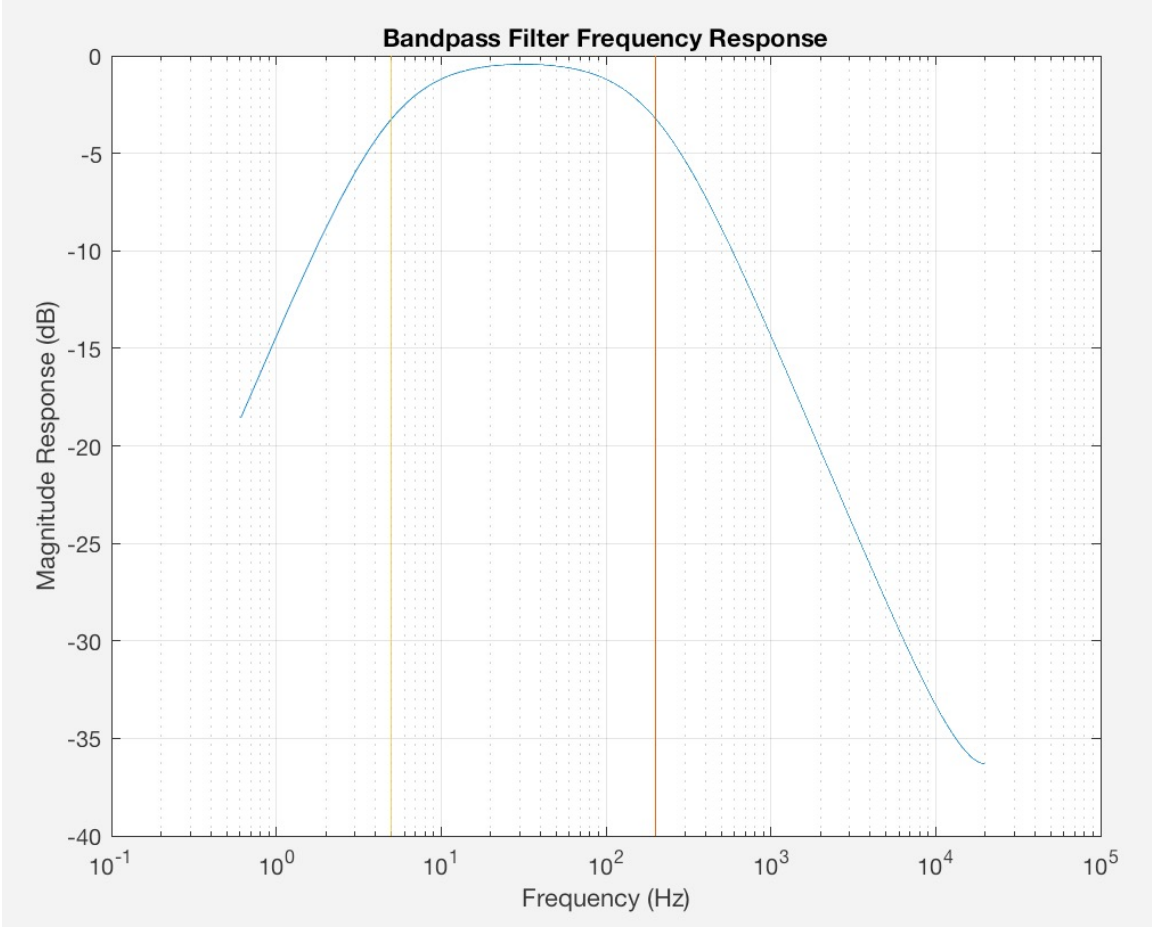
Band Pass?

Fs = 40000Hz

F3db1 = 5Hz
WEIGHT = 1273
(α = 7.85 e-4)

F3db1 = 200Hz
WEIGHT = 31
(α = 0.0309)

Plot of 1- Band stop



Designing for WEIGHT cutoff frequency

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

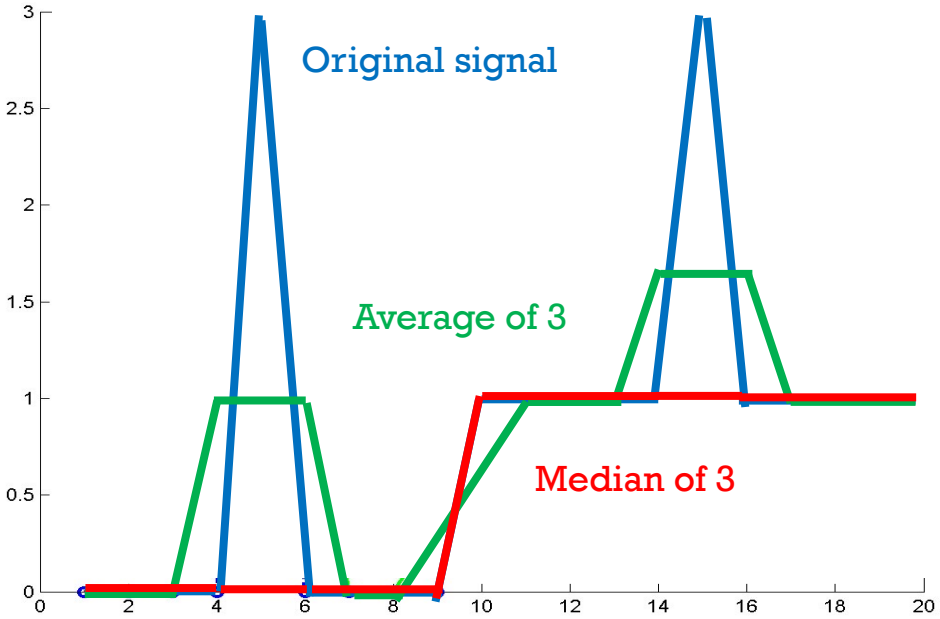
f_s is the sampling frequency

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

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

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

Median filter



Example of median of 3 values:

```
x = [2 80 6 3 ]
```

The median filtered output signal y will be:

```
y[1] =                        Median[2  2 80] = 2
y[2] = Median[2 80 6] = Median[2  6 80] = 6
y[3] = Median[80 6 3] = Median[3  6 80] = 6
y[4] = Median[6 3 3]  = Median[3  3  6] = 3
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;
}
```

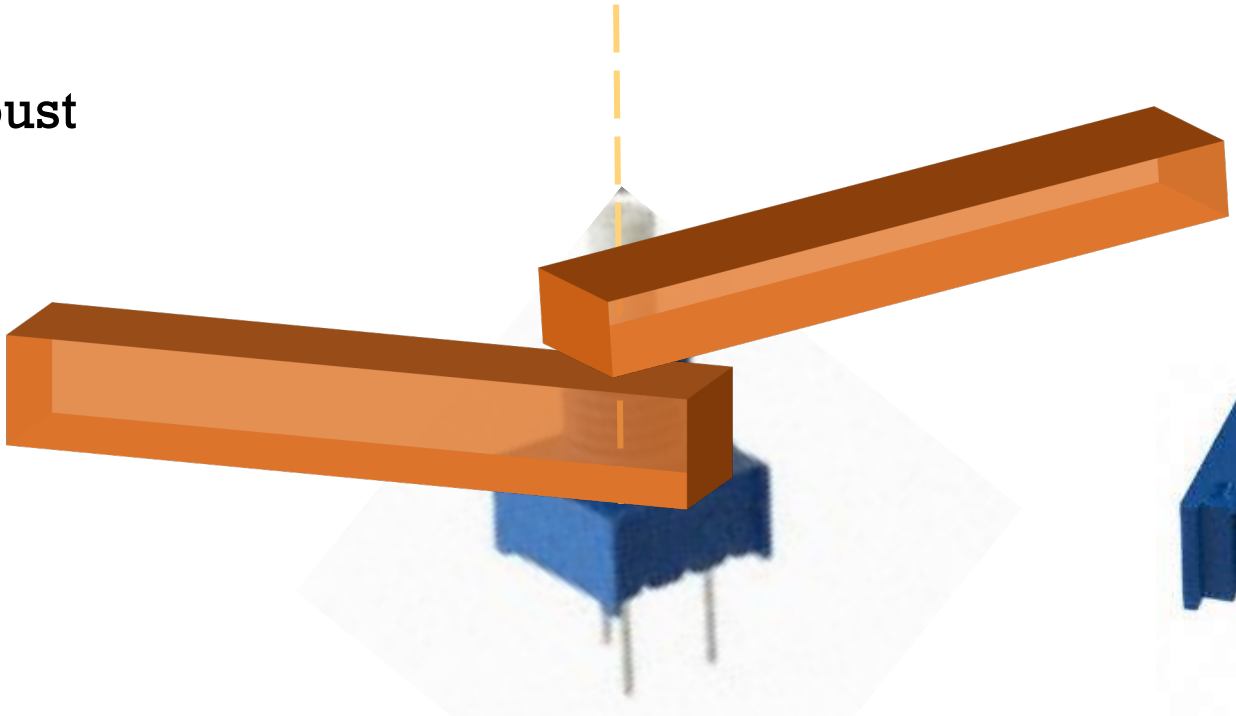
```
int movingavg(int newvalue, int WEIGHT){
  static int lastvalue;
  lastvalue = (WEIGHT*lastvalue + newvalue)/(WEIGHT+1);
  return lastvalue;
}
```

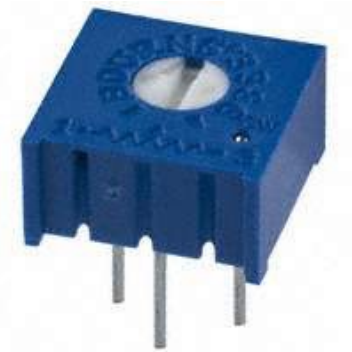
04

Rotational Position Sensors (options for LAB 3)

Potentiometer example

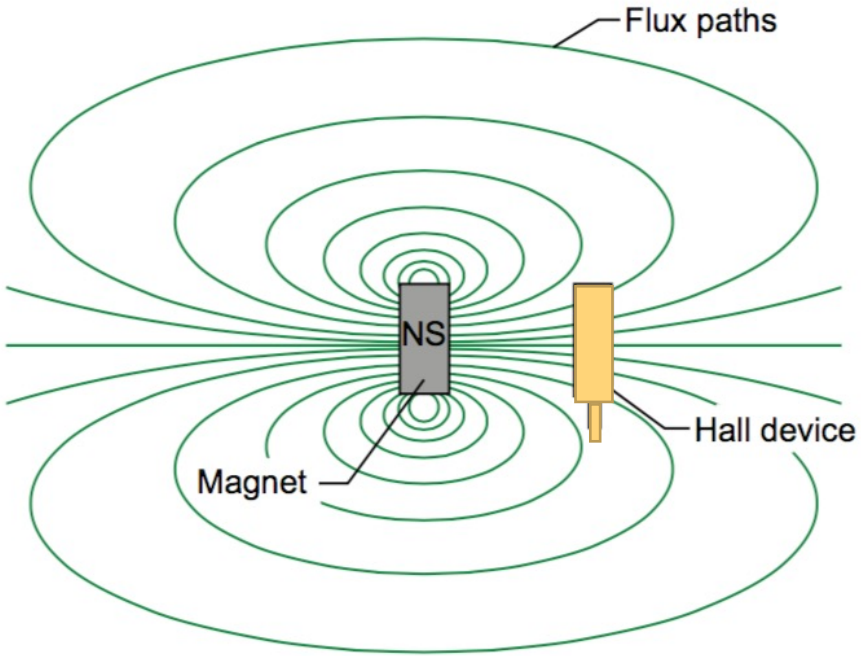
- Use potentiometer as voltage divider into an ADC
- Simple
- Not robust





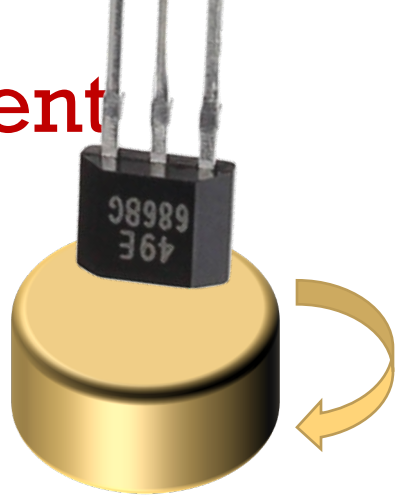
Hall Effect Sensor

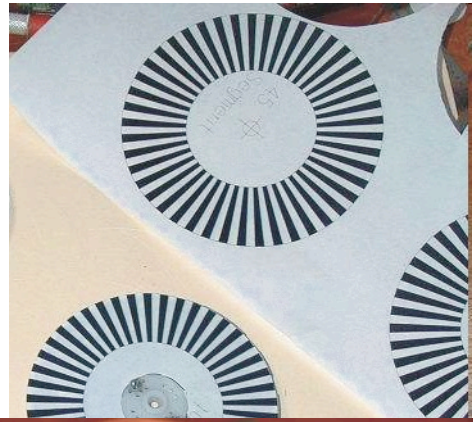
- Voltage proportional to applied magnetic field
- If the magnet rotates, the field will change correspondingly (non-linear, somewhat sinusoidal).

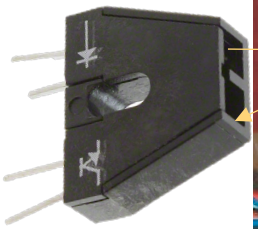


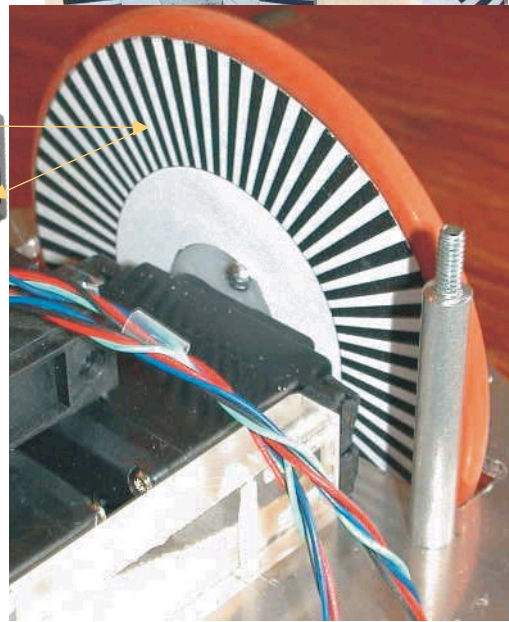
DIY Rotary Measurement

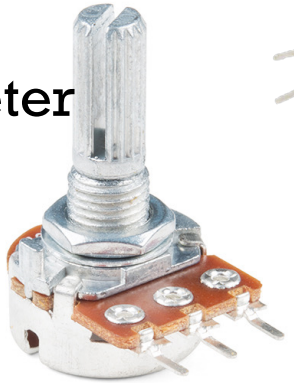
- Hall effect sensor (use ADC)
  - Absolute with ambiguity
  - Can't tell direction
  - Good for velocity sensing
- Make optical encoder disk
  - Need to be careful about ambient light
- Full rotation - hack potentiometer
  - Break limit stops
  - Deal with gap in wiper



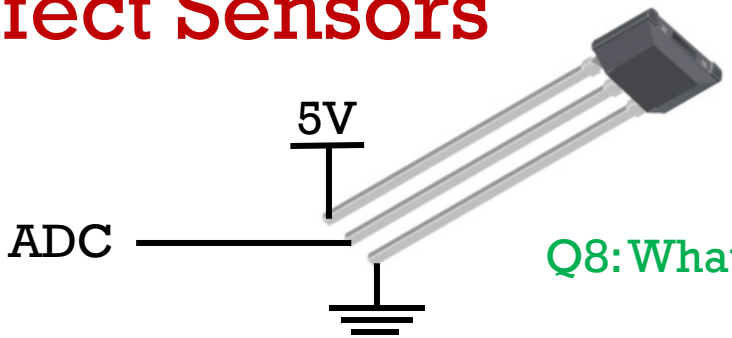


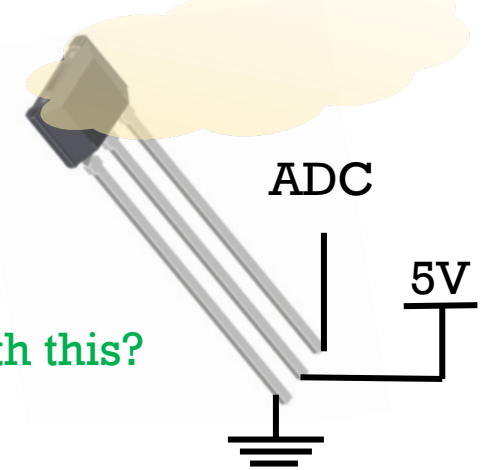






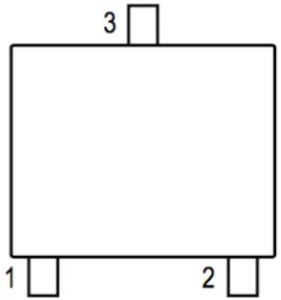
Hall Effect Sensors



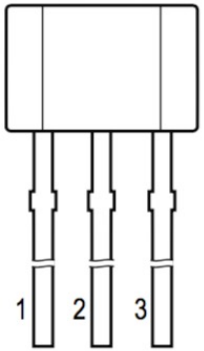


Q8: What's wrong with this?

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
- Simple software filters can act as high pass or low pass filters but add a delay
- Potentiometers make easy angle sensors if you can mount to them

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. Lab 2 material