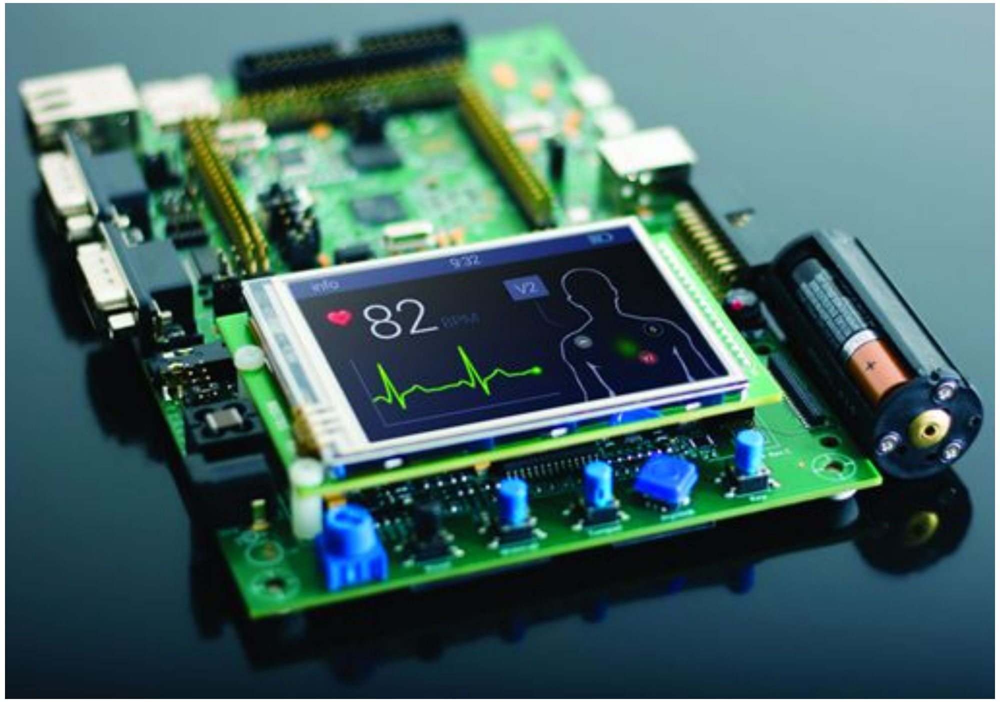
Embedded Software

CS 145/145L



Caio Batista de Melo

Announcements (2022-05-05)

- Project 3 is due tomorrow
  - Check the rubric!

- Homework 3 is also due tomorrow

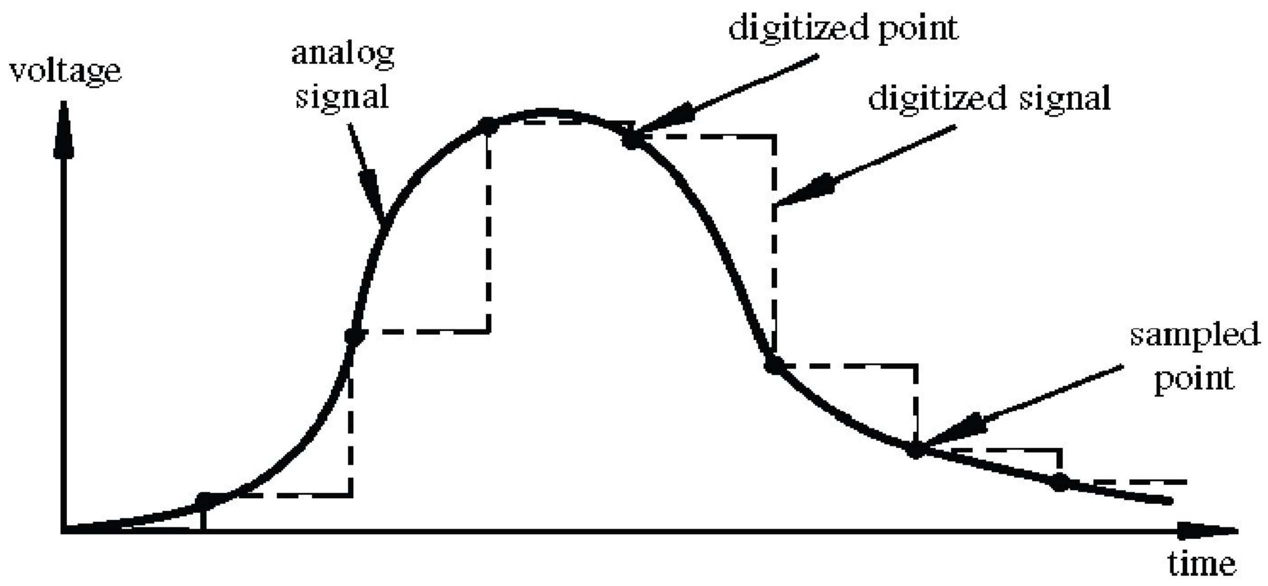
Agenda

- Analog-digital conversion
  - Rushed last class, so let's revisit!
- ATmega32's ADC
- Project 4

Analog-Digital Conversion (ADC)

Analog-Digital Conversion (ADC)

- Digital has **two** values: **on** and **off**
- Analog has many (**infinite**) values
- Computers don't really do analog, they ***quantize***

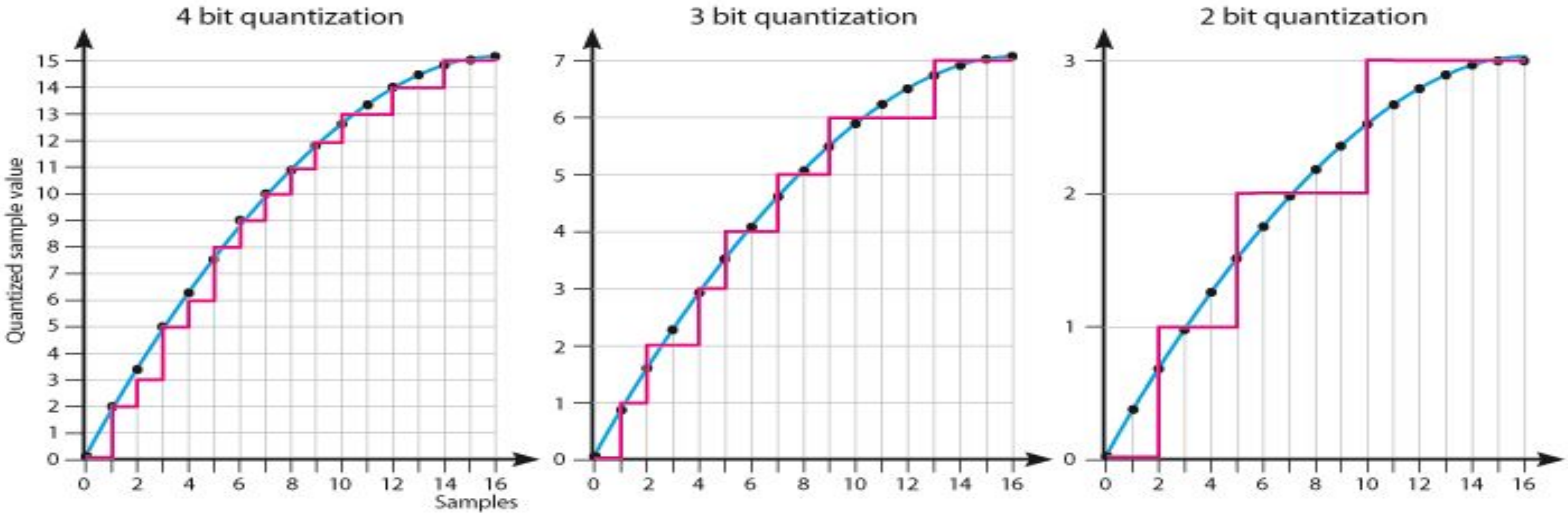


ADC Parameters

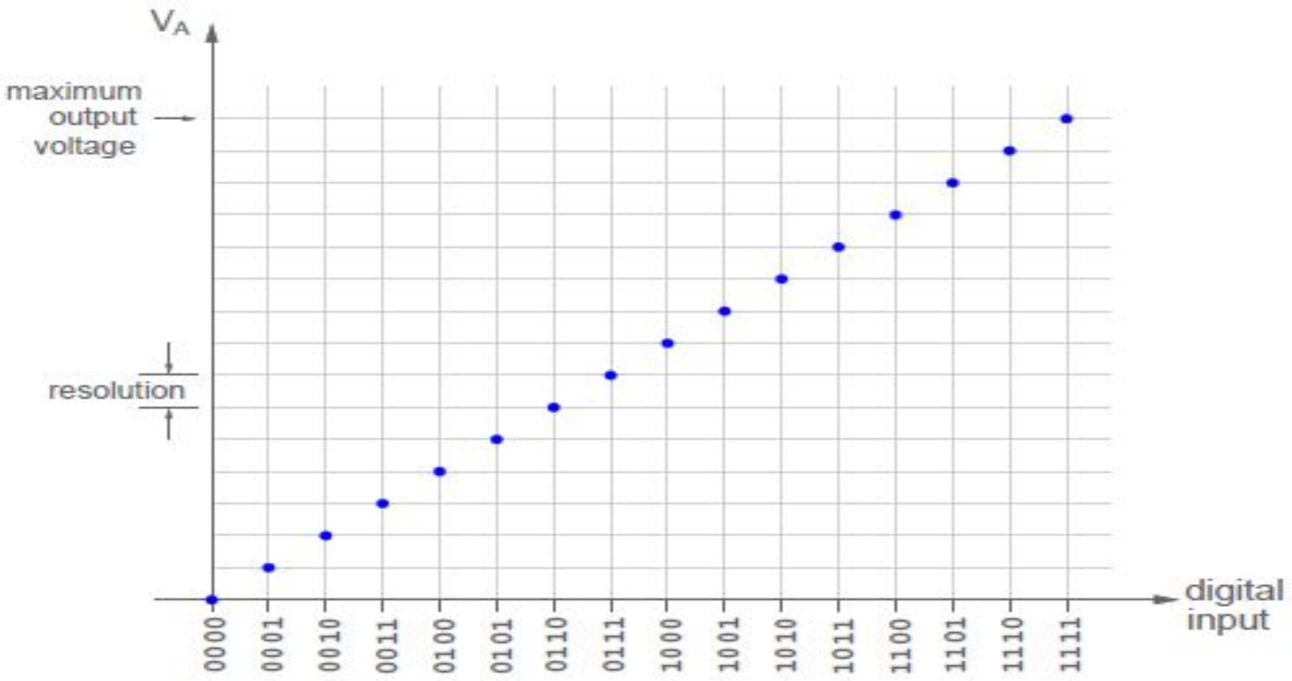
- Range
  - What are the minimum/maximum possible values
- Sampling Rate
  - How often we get a new data point
- Precision
  - How many bits we can use to represent the values

ADC 0~5V Example

Reads the voltage applied to an analog input pin and returns a number between **0 and 2^N-1** that represents the voltages between 0 and 5 V.



4-bit Quantization



b3	b2	b1	b0	Expected Decimal	Vout (V)
0	0	0	0	0	0
0	0	0	1	1	0.315
0	0	1	0	2	0.630
0	0	1	1	3	0.937
0	1	0	0	4	1.268
0	1	0	1	5	1.577
0	1	1	0	6	1.892
0	1	1	1	7	2.2
1	0	0	0	8	2.51
1	0	0	1	9	2.83
1	0	1	0	10	3.14
1	0	1	1	11	3.45
1	1	0	0	12	3.78
1	1	0	1	13	4.09
1	1	1	0	14	4.4
1	1	1	1	15	4.72

Intuitive Conversion

b3	b2	b1	b0	Expected Decimal	Vout (V)
0	0	0	0	0	0
0	0	0	1	1	0.315
0	0	1	0	2	0.630
0	0	1	1	3	0.937
0	1	0	0	4	1.268
0	1	0	1	5	1.577
0	1	1	0	6	1.892
0	1	1	1	7	2.2
1	0	0	0	8	2.51
1	0	0	1	9	2.83
1	0	1	0	10	3.14
1	0	1	1	11	3.45
1	1	0	0	12	3.78
1	1	0	1	13	4.09
1	1	1	0	14	4.4
1	1	1	1	15	4.72

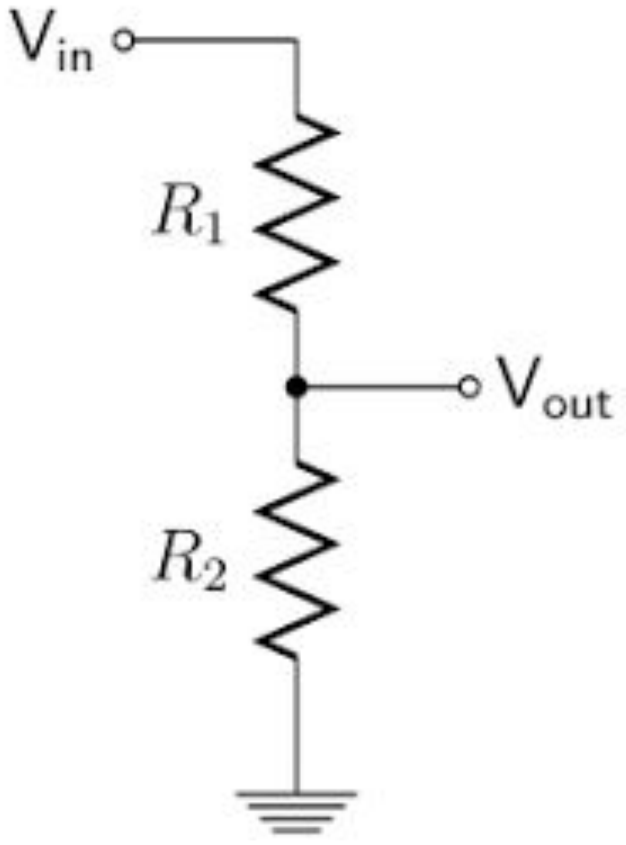
The pattern is repeated

- The MSB is contributing to half of the voltage.

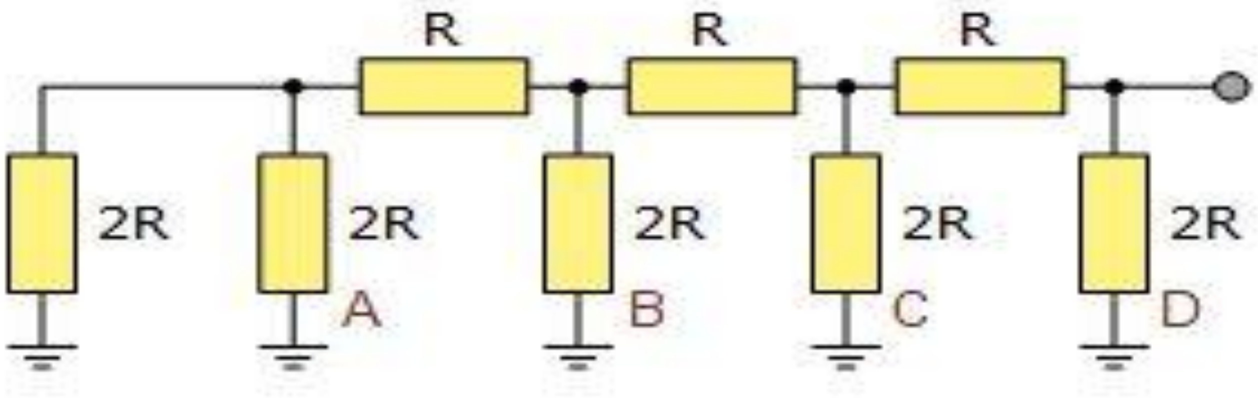
$$V_{out} = b3 * 2^3 + b4 * 2^2 + b1 * 2^1 + b0 * 2^0$$

b3 is contributing to 50% of the voltage
b2 to 25%
b1 to 12.5%
b0 to 6.25%

Intuitive Hardware



Voltage divider
(single bit)

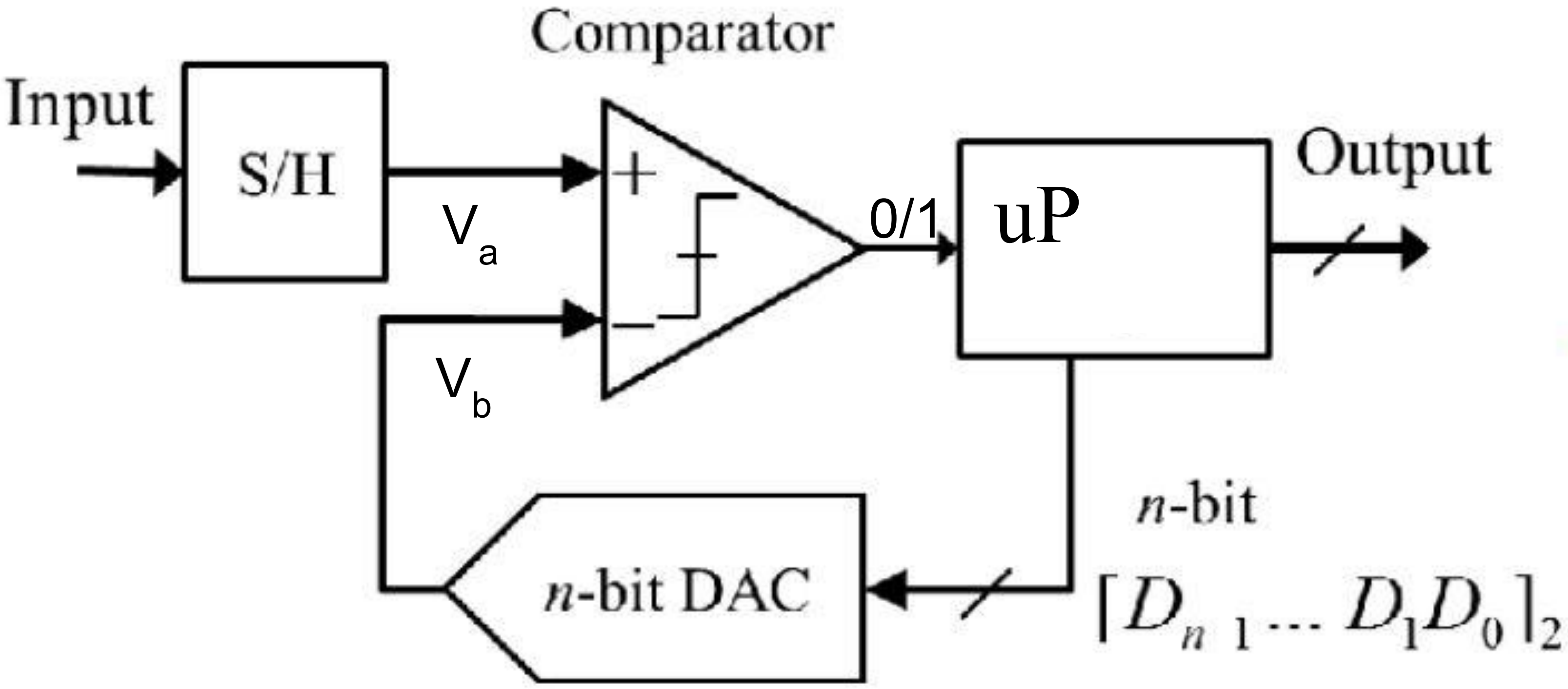


High quality signal without noise but expensive because resistances should have precise tolerances

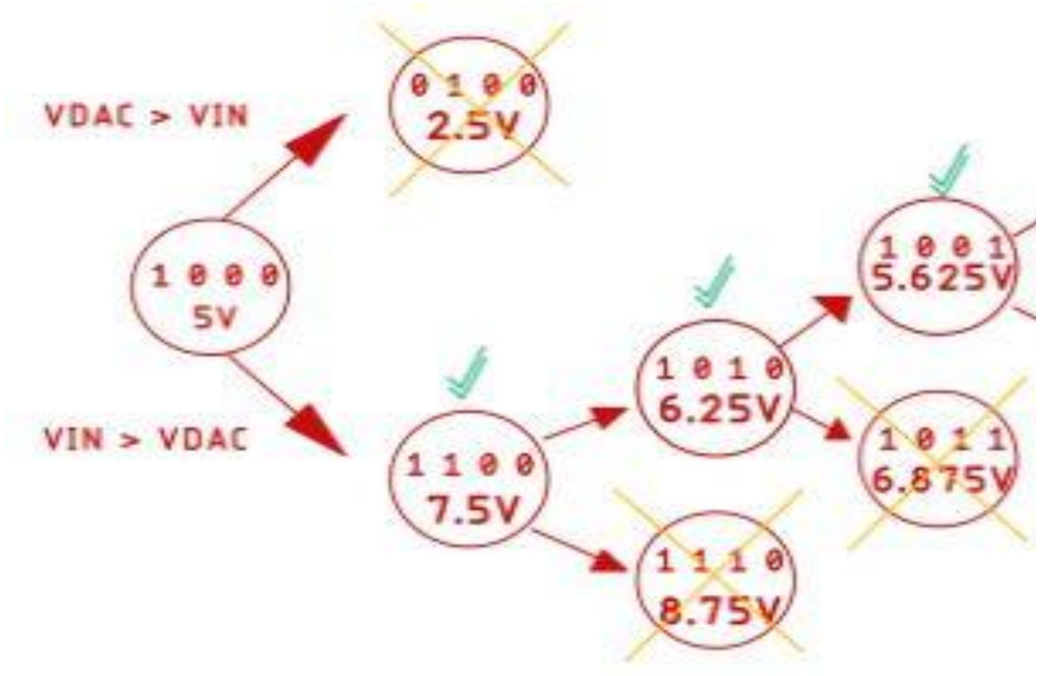
* e.g., audio systems

ADC Layout

Output	
0	Va < Vb
1	Va > Vb



ADC Result

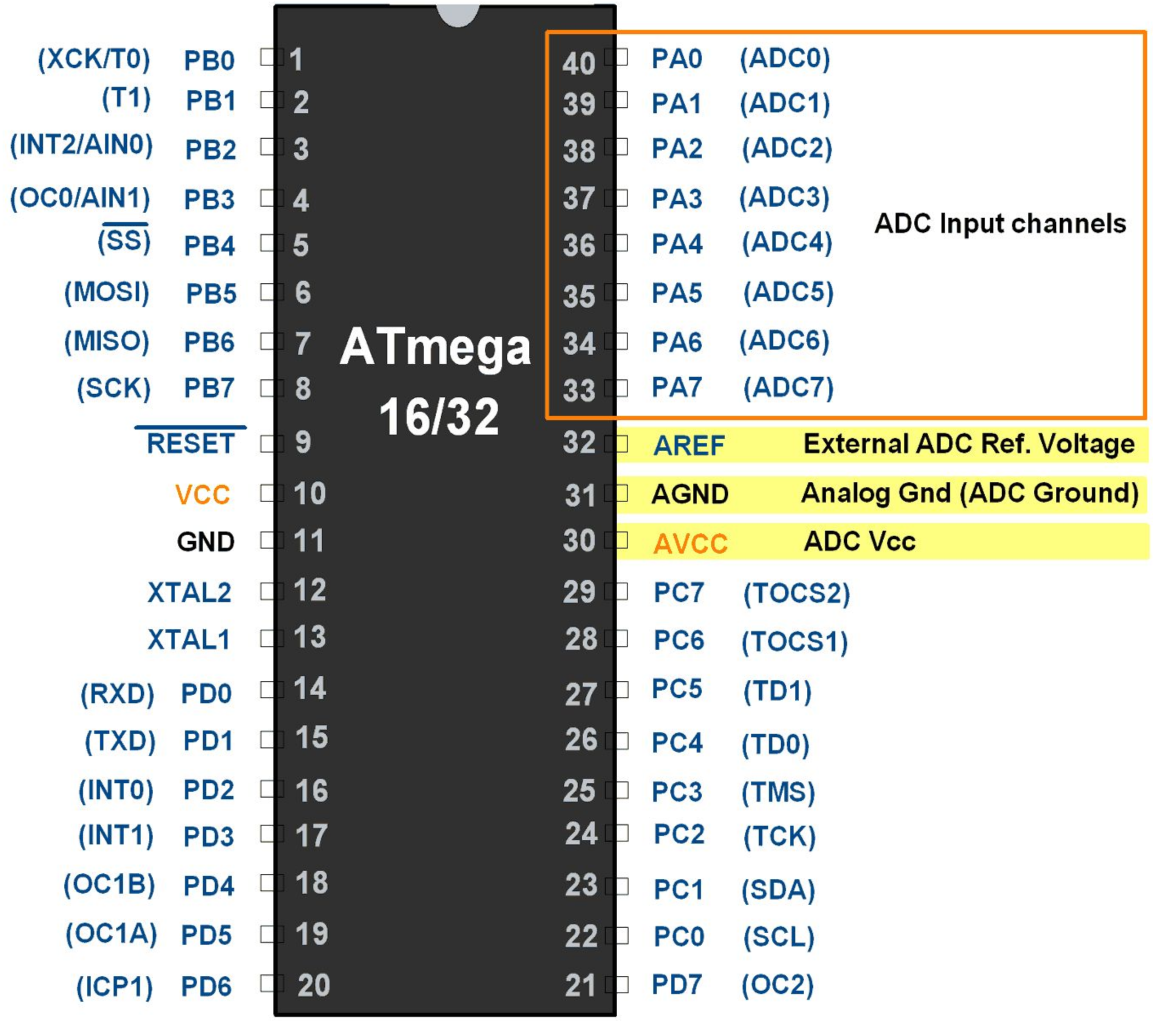


Successive Approximation
(Binary search)

- Let's say, the sampled input is 5.8V
- The reference of the ADC is 10V. When conversion starts, we guess it is 0b1000, which means half of the 10V reference.
- Now this voltage will be compared to the input voltage and based on the comparator output, the output of the register will be changed.

ADC on the ATmega32

ATmega32 Layout



https://www.electronicwings.com/avr-atmega/atmega1632-adc

ADC Registers

- ADMUX: ADC Multiplexer selection register
- ADC Data Registers
  - ADCH: Holds digital converted data higher byte
  - ADCL: Holds digital converted data lower byte
- ADCSRA: ADC Control and Status Register

ADC Multiplexer Selection Register (ADMUX)
Page 214 on manual

ADC Multiplexer Selection Register – ADMUX

Right or left shift

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	

- **Bit 7:6 – REFS1:0: Reference Selection Bits**

These bits select the voltage reference for the ADC, as shown in Table 83. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

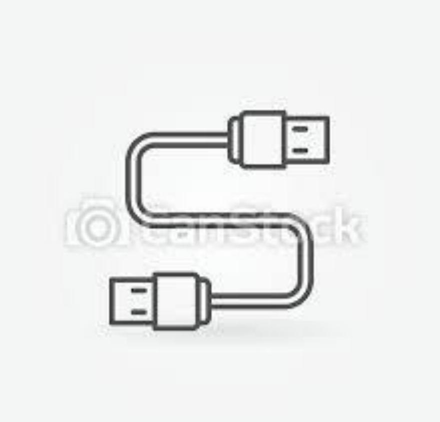
Table 83. Voltage Reference Selections for ADC

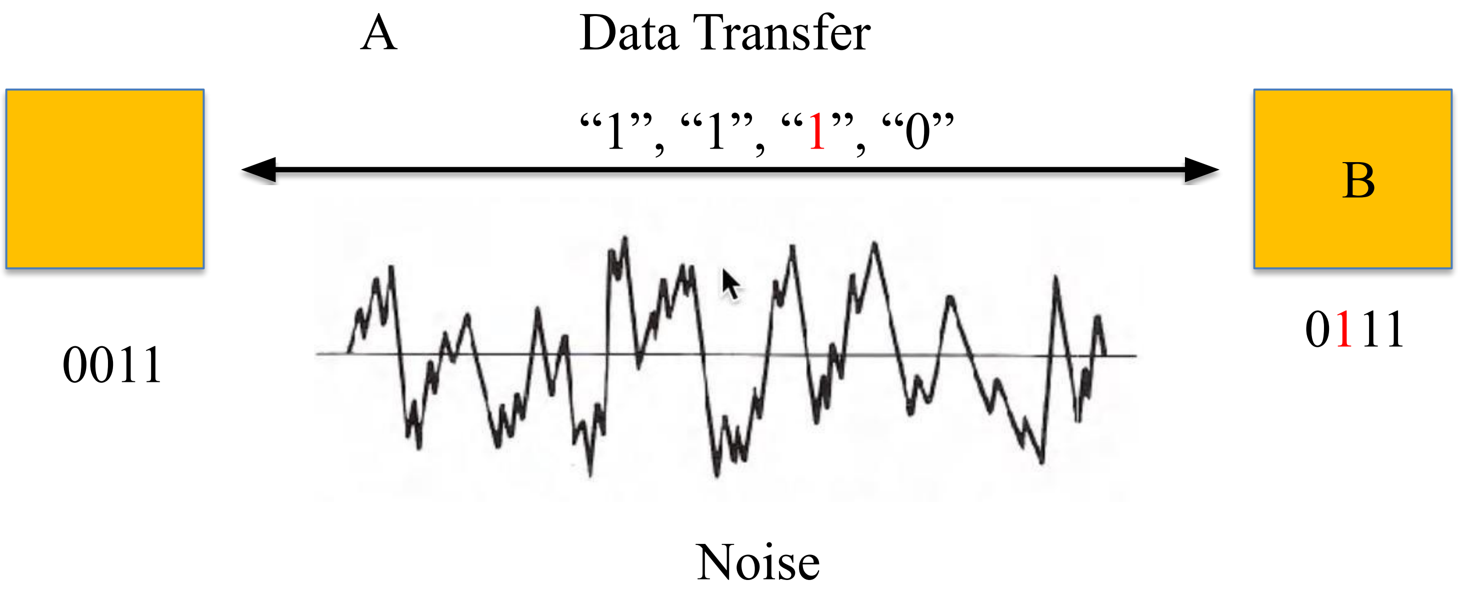
REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

ADMUX[0:4]
Page 215 on manual

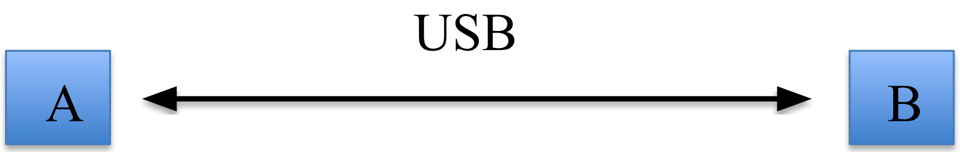
- Decides on multiplexing of the ports
- Let’s check the manual
- Why would you use differentials?

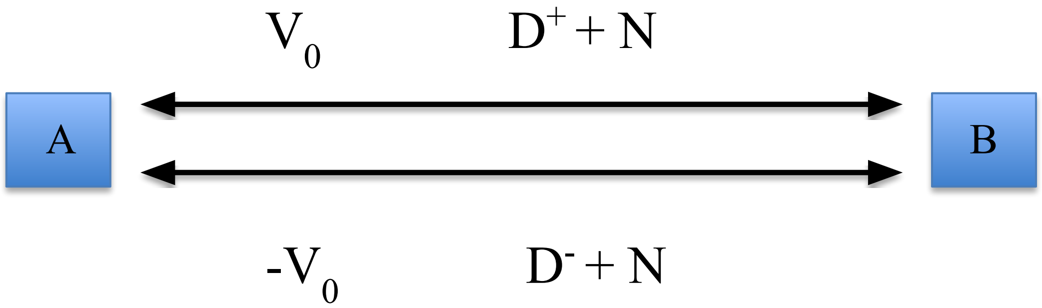
USB Example





USB Example (Two Ports)





0	V_0	$-V_0$	4.0, -4.0
1	V_1	$-V_1$	2.0, -2.0

$(D^+ - D^-)$

$(D^+ + N - (D^- + N))$

ADC Data Registers

The ADC Data Register – ADCL and ADCH

ADMUX[5]

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	**ADCH**
	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	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	**ADCH**
	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	

ADC Control and Status Register A (ADSCRA) Page 216

ADC Control and Status Register A – ADCSRA

Bit	7	6	5	4	3	2	1	0	
	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	

- **Bit 7 – ADEN: ADC Enable**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**

In Single Conversion mode, write this bit to one to start each conversion. In Free Running Mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

Project 4

Project 4

Design an embedded computer centered around the ATMega32 microcontroller.

For input: use a keypad and an analog-to-digital converter (ADC).

For output: use an LCD.

Write a C program that implements a simple voltmeter. Your voltmeter must take a sample every 500ms and update the display accordingly. Your system should:

- use the maximum ADC precision;
- show: (1) instantaneous, (2) max, (3) min, and (4) average voltages;
- always display instantaneous voltage when powered;
- reset minimum, maximum, and average voltages on a push of a button;
- start sampling those values after another push of a button;

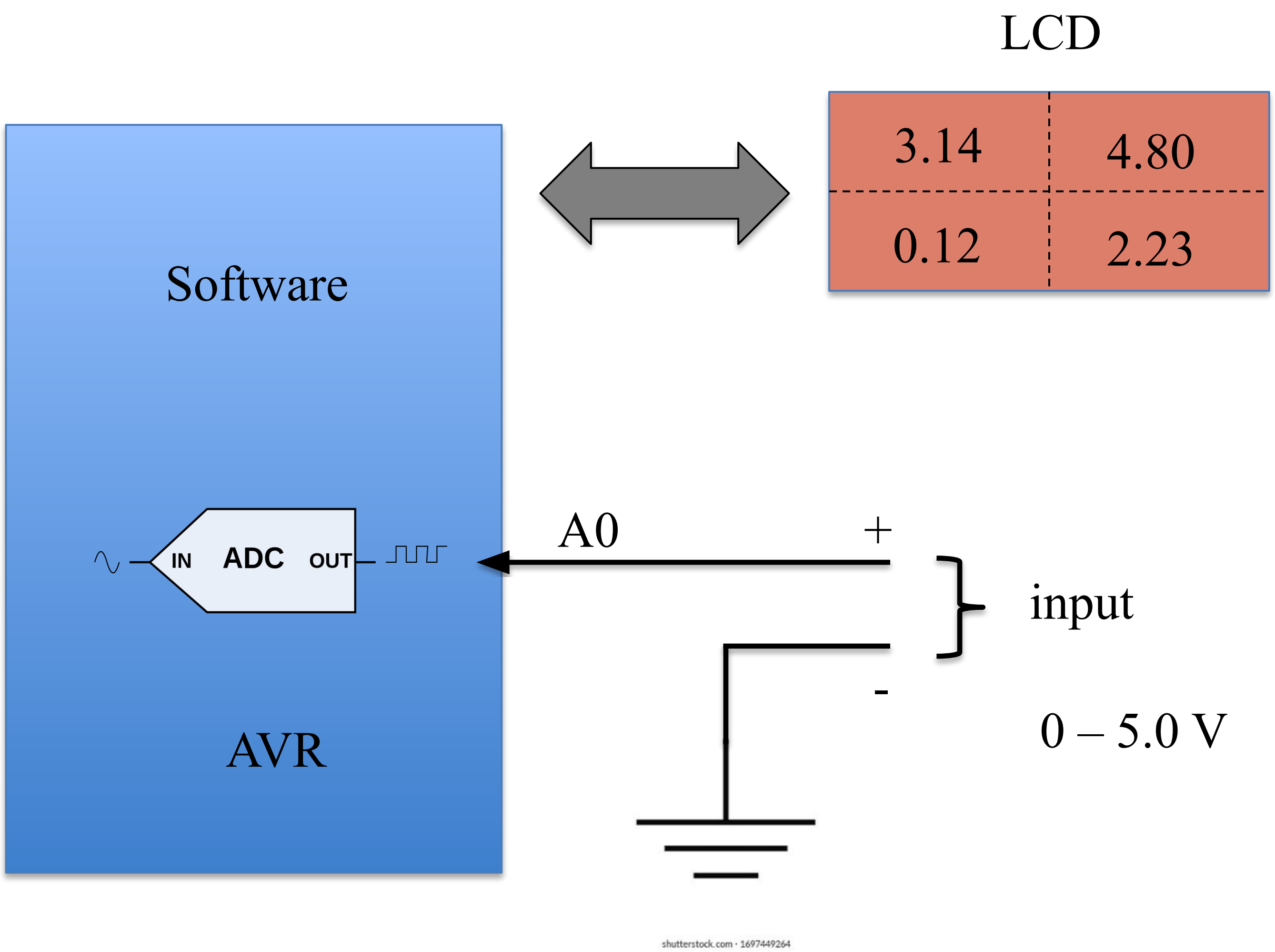
https://canvas.eee.uci.edu/courses/45047/assignments/929274

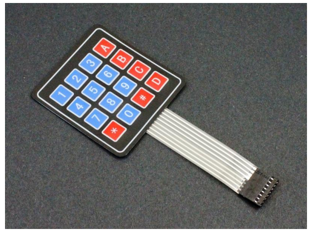
Project 4 - Grading

- Basic Project
  - Displays all 4 readouts (60%, 15% each)
  - Sampling rate should be at least 2 samples/second (i.e., every 500ms) (10%)
  - One button resets max/min/avg to blank (i.e., -----) (10%)
  - One button starts max/min/avg (10%)
  - Use all 10 bits precision (10%)
- Extra Credit
  - Early submission by 2022-05-14 (5 points)
  - Support differential inputs (10 points)
    - V1 and V2, display V1 - V2 (including negative values!)

General Voltmeter Layout

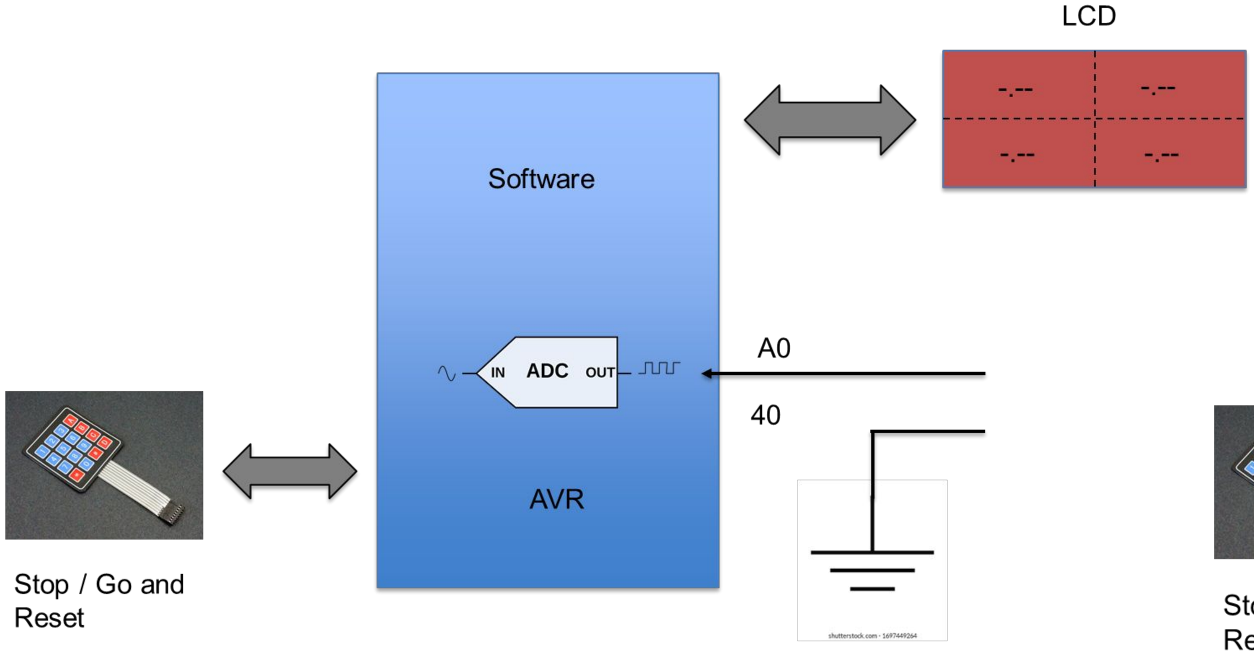
- [0, 5.0 V]
- 10 bit
- 0 – 1023 ⇔ 0.0 to 5.0V



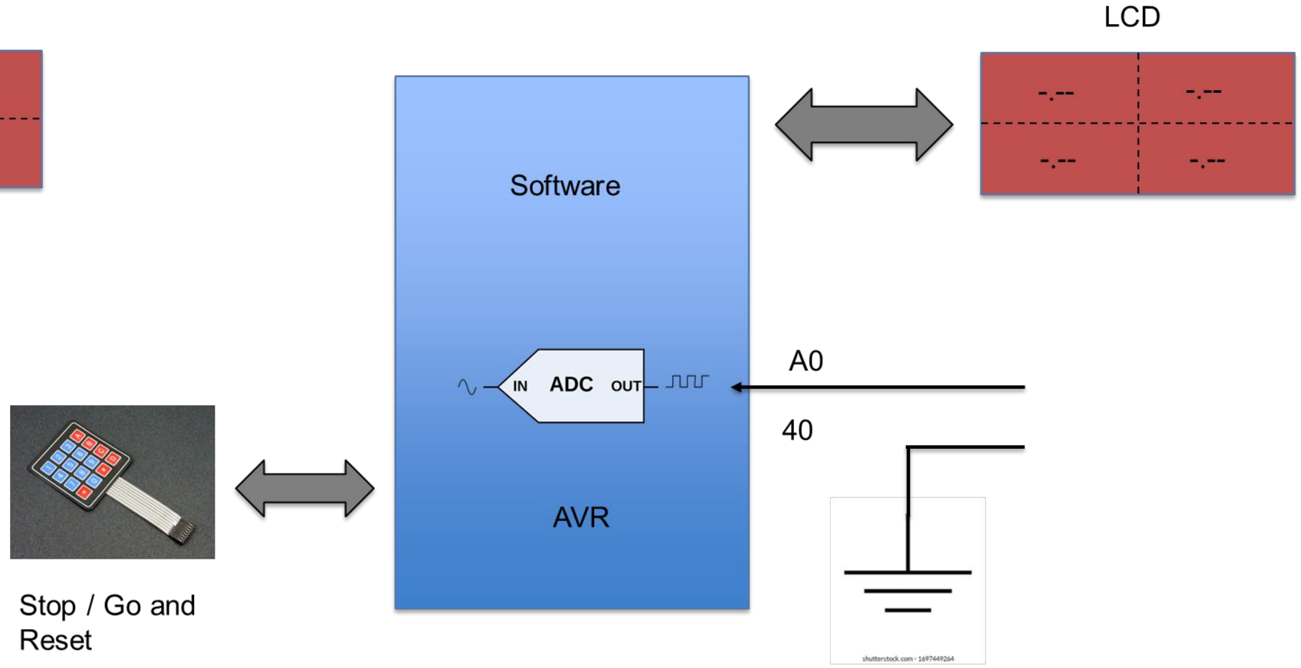


Stop / Go and Reset

Preliminary Tests



Connect to zero



Connect to 5V

Getting a Reading

```
int get_sample() {
    // configure the ADC
    // start conversion
    // wait for conversion result
    return result;
}
```

Gets a single sample, if you want many you'd call it many times.

Why int?

10 bits, so we need 2 bytes

ADMUX Reference Selection

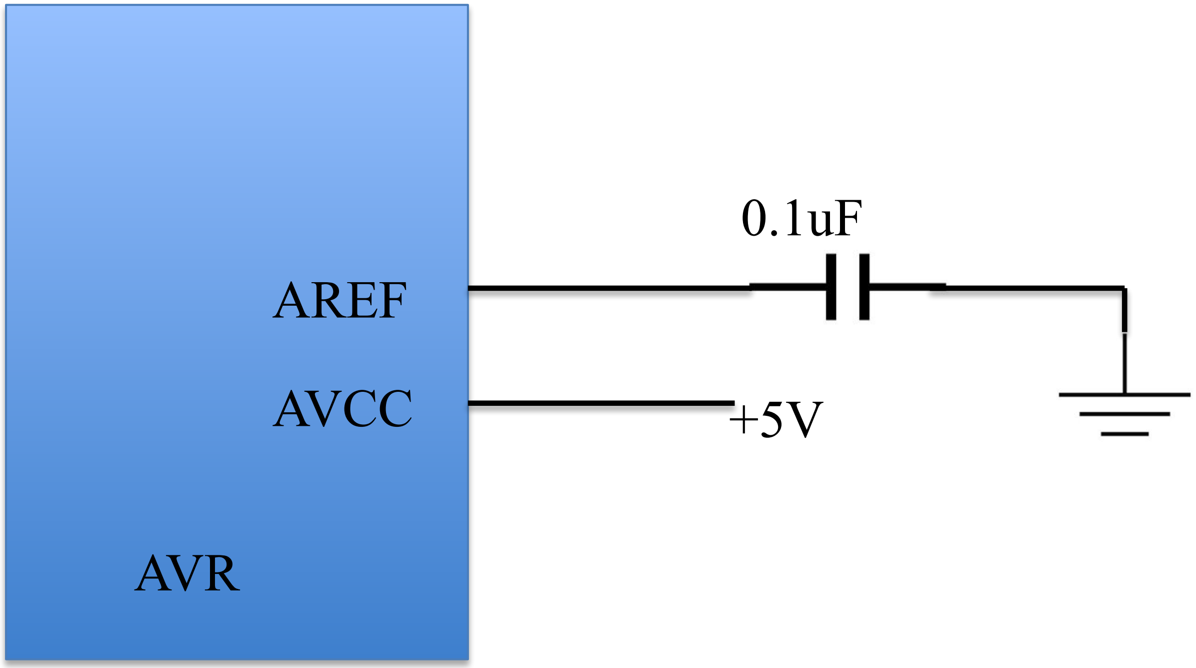
- **Bit 7:6 – REFS1:0: Reference Selection Bits**

These bits select the voltage reference for the ADC, as shown in Table 83. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

Table 83. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Analog Reference



Completing get_sample

```
int get_sample() {
    // configure the ADC
    // start conversion
    // wait for conversion result
    return result;
}
```

Depends how you're connecting things
To read from PA0, this should be 00000

```
int get_sample() {
    ADMUX = 0b010xxxxx;
    ADCSRA = 0b11yyyyyy;
    while (bit 6 of ADCSRA);
    return result;
}
```

Depends how you're converting things
To do single conversions, this could be 000000

Minimum Test Program

```
int main () {
    char buf[20];
    avr_init();
    lcd_init();
    while (1) {
        sprintf(buf, "%d", get_sample());
        lcd_clr();
        lcd_pos(0, 0);
        lcd_puts2(buf);
        avr_wait(500);
    }
}
```

Does this display volts?

No! It's a normalized value without unit!

How do you convert it?

"De-normalize" it :)

How do you keep track of min/max/avg?

Use variables!

Converting from normalized to volts

```
sprintf(buf, "%.2f", (get_sample() / 1023.0) * 5);
```

Store the value as an integer, only convert it at the very last moment.
This helps you keep precision!

You might need to change a setting in microchip studio so sprintf works with floats.
One of your classmates already figured this out, and they posted on EdStem!

Computing Min and Max

```
new_sample = get_sample();
if (new_sample > max) {
    max = new_sample;
}
```

New variable!

Remember to use int for these variables!

Computing Average

S0 → Avg = S0
S0, S1 → Avg = (S0 + S1) / 2
S0, S1, S2 → Avg = (S0 + S1 + S2) / 3

```
int sum = 0, count = 0;
while (1) {
    new_sample = get_sample();
    sum += new_sample;
    sprintf(buf, "%.2f", sum / ++count);
}
```

Any possible problems with this code?

Likely for sum to overflow!

Use unsigned (long) long

See you next time :)

Q & A