CprE 288 – Introduction to Embedded Systems
(Analog-to-Digital Converter)

Instructors:
Dr. Zhao Zhang (Sections A, B, C, D, E)
Dr. Phillip Jones (Sections F, G, J)



## Overview of Today's Lecture

- Announcements
- ADC
  - Successive Approximation
- ADC on the ATmega128



## Announcements

- This Thursday
  - Homework 5 due



## ANALOG TO DIGITAL CONVERTERS



## ADC and DAC

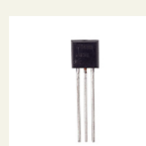
Analog-to-Digital Conversion (ADC) and
Digital-to-Analog Conversion (DAC)

Why do we need ADC and DAC?
- Analog sensors: Temperature, pressure, light, humidity, compass, and so on
- Audio applications: Microphone and speakers
- Image and video applications: Digital cameras, camcorder

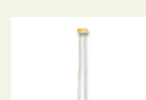


## ADC and DAC

Analog sensor: Convert a physical signal into an ***analog*** electrical signal



Temperature Sensor $3.95



Photoresistor $1.95 (Light Sensor)

Passive IR sensor $9.95 for motion detection

Sensor pictures and prices from http://www.parallax.com

## ADC and DAC

ADCs in microcontroller

- Most ***ADC***s also work as ***DAC***
- The ADC circuitry is located inside the microcontroller, with dozens of ADC pins going outside to support ***multiple input channels***
- Mostly 10-bit ***resolution*** (the higher the better, e.g. 14-bit in high-end digital cameras)
- Supports ***continuous mode*** and ***single conversion mode***
- ***Sampling rate*** in continuous mode ranges from tens of KHz to hundreds of KHz (the faster the better)



## ADC Terminology

**analog**: continuously valued signal, such as temperature, speed, or voltage with infinite possible values in between

**digital**: discretely valued signal, such as integers, encoded in binary

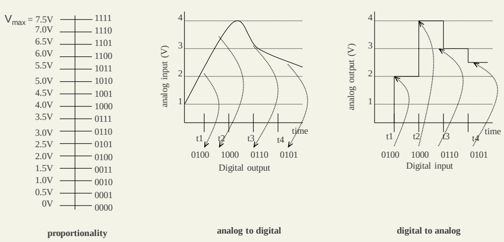
**analog-to-digital converter**: ADC, A/D, A2D; converts an analog signal to a digital signal

**digital-to-analog converter**: DAC, D/A, D2A; converts a digital number to an analog signal

An embedded system's surroundings typically involve many analog signals.



## Analog-to-digital converters




*Embedded Systems Design: A Unified Hardware/Software Introduction*, (c) 2000 Vahid/Givargis

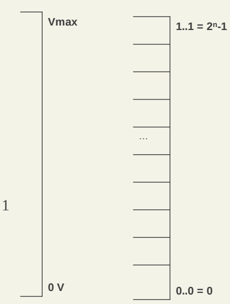


## Proportional Signals

<u>Simple Equation</u>

Assume minimum voltage of 0 V.
**Vmax** = maximum voltage of the analog signal
**a** = analog value
**n** = number of bits for digital encoding
  $2^n$ = number of digital codes
  **M** = number of steps, either $2^n$ or $2^n - 1$
**d** = digital encoding

**a / Vmax = d / M**






## Resolution

Let n = 2

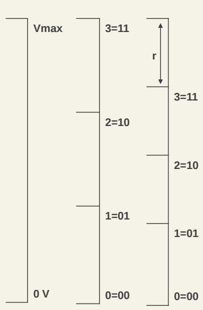
<u>$M = 2^n - 1$</u>

3 steps on the digital scale
$d_0 = 0 = 0b00$
$d_{Vmax} = 3 = 0b11$

<u>$M = 2^n$</u>

4 steps on the digital scale
$d_0 = 0 = 0b00$
$d_{Vmax - r} = 3 = 0b11$ (no $d_{Vmax}$ )

**r**, **resolution**: smallest analog change resulting from changing one bit






## DAC vs. ADC

- DAC
  - n digital inputs for digital encoding **d**
  - analog input for **Vmax** and **Vmin**
  - analog output **a**
- ADC
  - n digital outputs for digital encoding **d**
  - analog input for **Vmax** and **Vmin**
  - analog input **a**

## DAC vs. ADC

ADC:

Given a Vmax analog input and an analog input a, how does the converter know what binary value to assign to d in order to satisfy the ratio?

- may use DAC to generate analog values for comparison with a
- ADC "guesses" an encoding d, then checks its guess by inputting d into the DAC and comparing the generated analog output a' with original analog input a
- How does the ADC guess the correct encoding?



## ADC: Digital Encoding

Guessing the encoding is similar to finding an item in a list.

1. Sequential search – counting up: start with an encoding of 0, then 1, then 2, etc. until find a match.
   - $2^n$ comparisons: Slow!
2. Binary search – successive approximation: start with an encoding for half of maximum; then compare analog result with original analog input; if result is greater (less) than the original, set the new encoding to halfway between this one and the minimum (maximum); continue dividing encoding range in half until the compared voltages are equal
   - n comparisons: Faster, but more complex converter

→ Takes time to guess the encoding: start conversion input, conversion complete output



## ADC Using Successive Approximation

- Given an analog input signal whose voltage should range from 0 to 15 volts, and an 8-bit digital encoding, calculate the correct encoding for 5 volts. Then trace the successive-approximation approach to find the correct encoding.
- Assume $M = 2^n - 1$

$$a / Vmax = d / M$$
$$5 / 15 = d / (256 - 1)$$
$$d = 85 \text{ or binary } 01010101$$



## ADC Using Successive Approximation

Step 1-4: Determine bits 0-3

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ½($V_{max}$ – $V_{min}$) = 7.5 volts<br>$V_{max}$ = 7.5 volts. | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ½(7.5 + 0) = 3.75 volts<br>$V_{min}$ = 3.75 volts. | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| ½(7.5 + 3.75) = 5.63 volts<br>$V_{max}$ = 5.63 volts | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| ½(5.63 + 3.75) = 4.69 volts<br>$V_{min}$ = 4.69 volts. | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |

*Embedded Systems Design: A Unified Hardware/Software Introduction,* (c) 2000 Vahid/Givargis



## ADC Using Successive Approximation

Step 5-8: Determine bits 4-7

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ½(5.63 + 4.69) = 5.16 volts<br>$V_{max}$ = 5.16 volts. | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| ½(5.16 + 4.69) = 4.93 volts<br>$V_{min}$ = 4.93 volts. | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| ½(5.16 + 4.93) = 5.05 volts<br>$V_{max}$ = 5.05 volts. | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| ½(5.05 + 4.93) =<br>4.99 volts | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

*Embedded Systems Design: A Unified Hardware/Software Introduction,* (c) 2000 Vahid/Givargis



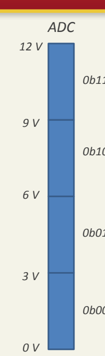
## Resolution

- Example
  - 2-bits of resolution
  - Vmin = 0 Volts
  - Vmax = 12 Volts

$M = 2^n$

$2^2 = 4$ buckets (or ranges) for the analog signal to fall

**r, resolution**: smallest analog change resulting from changing one bit



## Practice Problem 1

- Question:
  - Given:
    - n = 2 bit resolution
    - **Vmin** = 0 volts
    - **Vmax** = 12 volts
    - **a** = 5 volts
  - What is **d**?

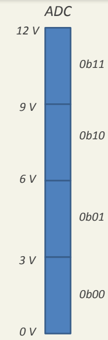


## Practice Problem 1

- Question:
  - Given:
    - n = 2 bit resolution
    - **Vmin** = 0 volts
    - **Vmax** = 12 volts
    - **a** = 5 volts
  - What is **d**?
- Answer:
  - Analog and Digital signals are proportional

$$\frac{a - Vmin}{Vmax - Vmin} = \frac{d}{2^n}$$

  - **d** is 0b01





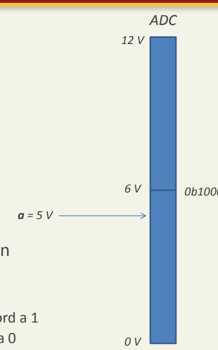

## Practice Problem 2

- Question:
  - Given:
    - n = 4 bit resolution
    - **Vmin** = 0 volts
    - **Vmax** = 12 volts
    - **a** = 5 volts
  - What is **d**?



## Practice Problem 2

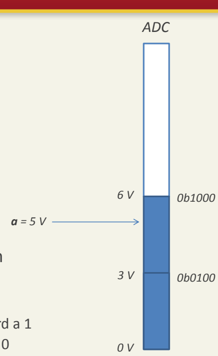
- Question:
  - Given:
    - n = 4 bit resolution
    - **Vmin** = 0 volts
    - **Vmax** = 12 volts
    - **a** = 5 volts
  - What is **d**?
- Answer:
  - Use successive approximation
  - **d** = 0b**?**???
    - Midpoint is 0b1000 at 6 volts
    - If **a** is greater than 6 volts, record a 1
    - If **a** is less than 6 volts, record a 0






## Practice Problem 2

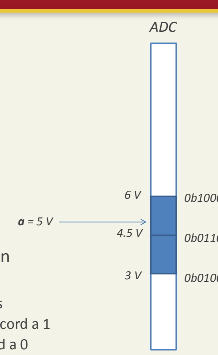
- Question:
  - Given:
    - n = 4 bit resolution
    - **Vmin** = 0 volts
    - **Vmax** = 12 volts
    - **a** = 5 volts
  - What is **d**?
- Answer:
  - Use successive approximation
  - **d** = 0b0**?**??
    - Midpoint is 0b0100 at 3 volts
    - If **a** is greater than 3 volts, record a 1
    - If **a** is less than 3 volts, record a 0





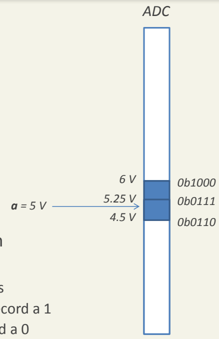

## Practice Problem 2

- Question:
  - Given:
    - n = 4 bit resolution
    - **Vmin** = 0 volts
    - **Vmax** = 12 volts
    - **a** = 5 volts
  - What is **d**?
- Answer:
  - Use successive approximation
  - **d** = 0b01**?**?
    - Midpoint is 0b0110 at 4.5 volts
    - If **a** is greater than 4.5 volts, record a 1
    - If **a** is less than 4.5 volts, record a 0



## Practice Problem 2

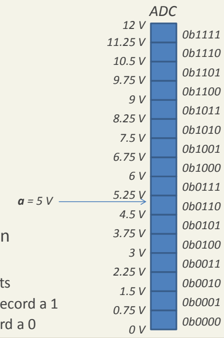
- Question:
  - Given:
    - n = 4 bit resolution
    - **Vmin** = 0 volts
    - **Vmax** = 12 volts
    - **a** = 5 volts
  - What is **d**?
- Answer:
  - Use successive approximation
  - **d** = 0b011?
    - Midpoint is 0b0111 at 5.25 volts
    - If **a** is greater than 5.25 volts, record a 1
    - If **a** is less than 5.25 volts, record a 0






## Practice Problem 2

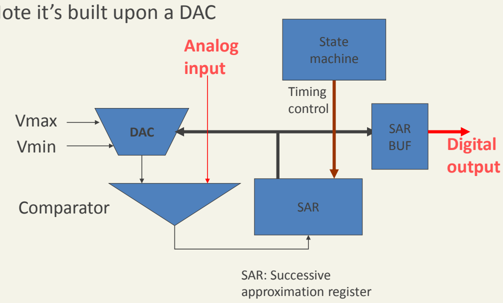
- Question:
  - Given:
    - n = 4 bit resolution
    - **Vmin** = 0 volts
    - **Vmax** = 12 volts
    - **a** = 5 volts
  - What is **d**?
- Answer:
  - Use successive approximation
  - **d** = 0b0110
    - Midpoint is 0b0111 at 5.25 volts
    - If **a** is greater than 5.25 volts, record a 1
    - If **a** is less than 5.25 volts, record a 0






## Constructing the ADC

Note it's built upon a DAC






## ADC Using Successive Approximation

- Example: 0V-16V range, 9.5V input, 4-bit resolution.
- Essentially a form of binary search
- Each step works out the value of one bit.
- Instructor will fill in the table

| Step | Range | Mid (digital) | Mid (voltage) | Is a > Mid (voltage)? |
|---|---|---|---|---|
| 0 | 0bxxxx | 0b1000 | 8 Volts | Yes |
| 1 | 0b1xxx | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |



## ADC Using Successive Approximation

- Example: 0V-16V range, 9.5V input, 4-bit resolution.
- Essentially a form of binary search
- Each step works out the value of one bit.
- Instructor will fill in the table

| Step | Range | Mid (digital) | Mid (voltage) | Is a > Mid (voltage)? |
|---|---|---|---|---|
| 0 | 0bxxxx | 0b1000 | 8 Volts | Yes |
| 1 | 0b1xxx | 0b1100 | 12 Volts | No |
| 2 | 0b10xx | 0b1010 | 10 Volts | No |
| 3 | 0b100x | 0b1001 | 9 Volts | Yes |
| 4 | 0b1001 | - | - | - |



## ADC Using Successive Approximation

- Example: 0V-16V range, 9.5V input, 4-bit resolution.

$$\frac{a - Vmin}{Vmax - Vmin} = \frac{d}{2^n}$$

$$\frac{9.5}{16 - 0} = \frac{d}{16}$$

$$d = \frac{9.5 * 16}{16} = 9.5$$

*d* = 0b1001 = 9

## ADC Bit Weight

Notice the concept of bit weight in the last example:
bit 7 = 7.5 V = 15/2
bit 6 = 3.75 V = 15/4

Each bit is weighted with an analog value, such that a 1 in that bit position adds its analog value to the total analog value represented by the digital encoding.

**Example:** -5 V to +5 V analog range, n=8

| Digital Bit | Bit Weight (V) |
|---|---|
| 7 | 10/2 = 5 |
| 6 | 10/4 = 2.5 |
| 5 | 10/8 = 1.25 |
| 4 | 10/16 = 0.625 |
| 3 | 10/32 = 0.313 |
| 2 | 10/64 = 0.157 |
| 1 | 10/128 = 0.078 |
| 0 | 10/256 = 0.039 |



## ADC Bit Weight

**Example (continued):** -5 V to +5 V analog range, n=8

Digital numbers for a few analog values
- Values shown increment by 6 bits (weight for bit position 5 is 1.25 V)
- Maximum digital number only approximates the maximum analog value in the range
  - Try (-5) + sum of all bit weights

| Analog (V) | Digital (hex) |
|---|---|
| -5 | 00 |
| -3.75 | 20 |
| -2.5 | 40 |
| -1.25 | 60 |
| 0 | 80 |
| 1.25 | A0 |
| 2.5 | C0 |
| 3.75 | E0 |
| 5-0.039 = 4.961 | FF |



## ADC Terms & Equations

**Offset**: minimum analog value

**Span** (or **Range**): difference between maximum and minimum analog values
Max - Min
**n**: number of bits in digital code (sometimes referred to as n-bit resolution)

**Bit Weight**: analog value corresponding to a bit in the digital number

**Step Size** (or **Resolution**): smallest analog change resulting from changing one bit in the digital number, or the analog difference between two consecutive digital numbers; also the bit weight of the
Span / $2^n$ (Assume M = $2^n$)



## ADC ON ATMEGA128



## ATMega128 ADC

- 10-bit ADC conversion
- Eight input channels through a MUX
- Analog input on one of ADC0-ADC7 pins
- Up to 15K samples per second
- 0 – Vcc or 0 – 2.56V ADC input voltage range



## Programming Interface: ADCSRA

Three registers
- **ADCSRA**: ADC control and status register A
- **ADMUX**: ADC input selection
- **ADCW**: ADC result register, 16-bit

## ATMega128 I/O Ports

ATMega128 I/O Ports and **Alternative Functions**

Ports A-G, each pin can be configured as **General Purpose Digital I/O Pin**

- DDR*x* decides if a pin is for input or output *
- PORT*x* is for writing to output pins
- PIN*x* is for reading from input pins

(* There is a special tri-state configuration)

**Alternatively**, those pins can be used as I/O pins for internal I/O devices

- Activate a pin's alternative function by enabling the corresponding I/O devicce
- Then, the pin MAY NOT be used as GP I/O pin



## ATMega128 I/O Pins

Most pins have Alternative Functions: USART, ADC, input capture, output compare, and others

USART0 uses port E

- PE2: External Clock (PE – Port E)
- PE1: Transmit Pin
- PE0: Receive Pin

USART1 uses port D

- PD5: External Clock
- PD3: Transmit Pin
- PD2: Receive Pin



## ADC I/O Pins

The ADC uses Port F, all eight pins

- There are eight input channels
- PF0 – PF7 for channels 0-7

If ADC is enabled, then avoid using port F for external I/O device

- No need to configure Port F as input, just enable the ADC



## Program Interface: ADCSRA

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| ADEN | ADSC | ADFR | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 | ADCSRA |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

**ADEN: ADC Enable**

Write 1 to this bit to enable ADC

**ADSC: ADC Start Conversion**

Write 1 to this bit to start ADC conversion. It turns to 0 after conversion is done



## Coding Examples

Enable ADC

```
ADCSRA |= 0x80;
```

Disable ADC

```
ADCSRA &= 0x7F;
```

Read an ADC word, assuming it's ready

```
unsigned reading = ADCW;
```



## Coding Examples

**Using a different coding style**

Enable ADC

```
ADCSRA |= (1 << ADEN);
```

Disable ADC

```
ADCSRA &= ~(1 << ADEN);
```

Notes

- <avr/io.h> declares ADEN and other names as macros
- ADEN is defined as 7: (1 << 7) == 0x80

## Coding Example

Assume ADC has been configured appropriately and in one-shot mode with interrupts disabled.

Write code to (1) start ADC, (2) wait for the conversion to complete, and (3) read the output.

```
ADCSRA |= (1<<ADSC);
while (ADCSRA & (1<<ADSC))
  {}
unsigned ADC_reading = ADCW;
```



## Program Interface: ADCSRA

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| ADEN | ADSC | ADFR | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 | ADCSRA |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

**ADFR: ADC Free Running Select**
- 1: continues sampling, 0: one shot

**ADIF: ADC Interrupt Flag**
- 1: interrupt raised, 0: otherwise

**ADIE: ADC Interrupt Enable**



## Coding Examples

Enable ADC interrupt

```
ADCSRA |= (1 << ADIE);
```

Disable ADC interrupt

```
ADCSRA &= ~(1 << ADIE);
```

Check ADC interrupt flag manually

```
if (ADCSRA & (1 << ADIF))
    … // do something
```



## Coding Examples

**Use a different coding style**

Enable ADC interrupt

```
ADCSRA |= _BV(ADIE);
```

Disable ADC interrupt

```
ADCSRA &= ~_BV(ADIE);
```

**_BV** (**Bit Vector**) is declared in <arv/io.h> like follows:

```
#define _BV(x)    (1<<(x))
```



## Program Interface: ADCSRA

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| ADEN | ADSC | ADFR | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 | ADCSRA |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

**ADPS2:0: ADC Prescaler Select Bits**

Select one of seven division factors

000: 2, 001: 2, 010: 4, 011: 8,
100: 16, 101: 32, 110: 64, 111: 128

Typical conversion times:
- 25 ADC clock cycles for the first conversion
- Faster time (13 or 14 ADC cycles) for second conversion and thereafter in continuous sampling mode

**Required ADC clock frequency**: 50KHz – 200KHz
- Higher frequency possible with less than 10-bit accuracy



## Coding Example

System clock is 16MHZ. What prescalar value is valid?

| Pres. Bits | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| Div. Factor | 2 | 2 | 4 | 8 | 16 | 32 | 64 | 128 |
| Freq. (Hz) | 8M | 8M | 4M | 2M | 1M | 500K | 250K | 125K |

What happens with higher frequency?

To set the frequency:

```
ADCSRA |= (7 << ADSP0);
```

Or

```
ADCSRA |= _BF(ADSP2) | _BF(ADSP1) | _BF(ADSP0);
```

## Programming Interface: ADMUX

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| REFS1 | REFS0 | ADLAR | MUX4 | MUX3 | MUX2 | MUX1 | MUX0 | ADMUX |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

**REFS1:0: Reference Selection Bits**

- Use code 01 to select Vcc as the reference voltage
- Can a 2.56V or external reference voltage; Use code 11 for 2.56V

**ADLAR: ADC Left Adjust Result**

- 1: 10-bit data is left adjusted in 16-bit reg
- 0: right adjusted



## Coding Example

Set the reference voltage to 2.56V:

```
ADMUX |= _BV(REFS1) | _BV(REFS0);
```

Make the result left adjusted:

```
ADMUX |= _BV(ADLAR);
```



## Programming Interface: ADMUX

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| REFS1 | REFS0 | ADLAR | MUX4 | MUX3 | MUX2 | MUX1 | MUX0 | ADMUX |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

**MUX4:0: Analog Channel and Gain Selection Bits**

There are eight pins connected to ADC through a MUX

Code 00<u>000</u> for ADC pin 0, 00<u>001</u> for pin 1, …, and 00<u>111</u> for pin 7.

01<u>xxx</u>, 10<u>xxx</u>, 11<u>xxx</u> are reserved for **differential inputs** (not to be discussed) with gain



## Coding Example

Get a reading from a given ADC channel

```
unsigned ADC_read(char channel)
{
  ADMUX |= (channel & 0x1F);
  ADCSRA |= _BV(ADSC);
  while (ADCSRA & _BV(ADCS))
    {}
  return ADCW;
}
```



## Code Example

Configure ADC as single shot, interrupt disabled, result right adjusted, precalar 128 and use 2.56 as the reference voltage

First, decide bits in ADCSR and ADMUX

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| ADEN | ADSC | ADFR | ADIF | ADIE | ADPS2 | ADPS1 | ADPS0 | ADCSRA |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| REFS1 | REFS0 | ADLAR | MUX4 | MUX3 | MUX2 | MUX1 | MUX0 | ADMUX |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |



## Programming Example

```
ADC_init()
{
  // REFS=11, ADLAR=0, MUX don't care
  ADMUX = _BV(REFS1) | _BV(REFS0);

  // ADEN=1, ADFR=0, ADIE=0, ADSP=111
  //   others don't care
  // See page 246 of user guide
  ADCSRA = _BV(ADEN) | (7<<ADPS0);
}
```

Note: It's a good idea to enable the device as the last step, so make change to ADCSRA as the last step

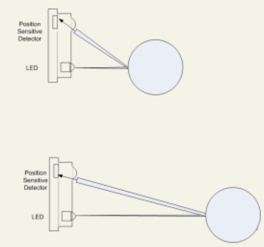
## Typical Conversion Times

| Condition | Sample & Hold | Conversion Time |
|---|---|---|
| First Conversion | 13.5 | 25 |
| Normal conversion, single ended | 1.5 | 13 |
| Normal conversion, differential | 1.5/2.5 | 13/14 |

55

## Measuring Distance with the IR Sensor

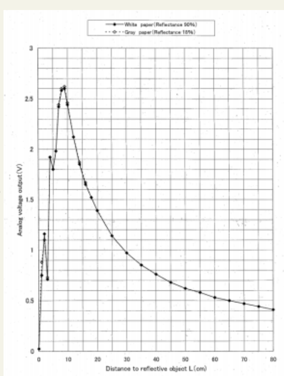
- The IR sensor emits an IR beam, and sets a voltage based on the distance of an object




56

**From the IR Sensor Datasheet**

- The voltage from the IR sensor depends on the distance
- As the distance increases, the voltage decreases (see graph)




57

## How To Measure Distance with the IR Sensor

Getting a distance from the IR sensor involves the following process:

1. The IR sensor measures a distance and sets the voltage on the wire leading to **ADC2** (**channel 2**)
2. The ADC converts this voltage into a digital value between 0 and 1023 and stores it in the registers ADCL and ADCH (ADCW)
3. Your program reads ADCW and converts the value into a distance… but how?!?

58

## How To Measure Distance with the IR Sensor

- Two methods to calibrate your distance
- Measure 50 points, create a table for comparing
  - Create a table that has the value of ADCW when an object is x centimeters away
  - Use this table to lookup the distance when a similar ADCW result is returned
- Measure 5 points, use Excel to get a trend line

59