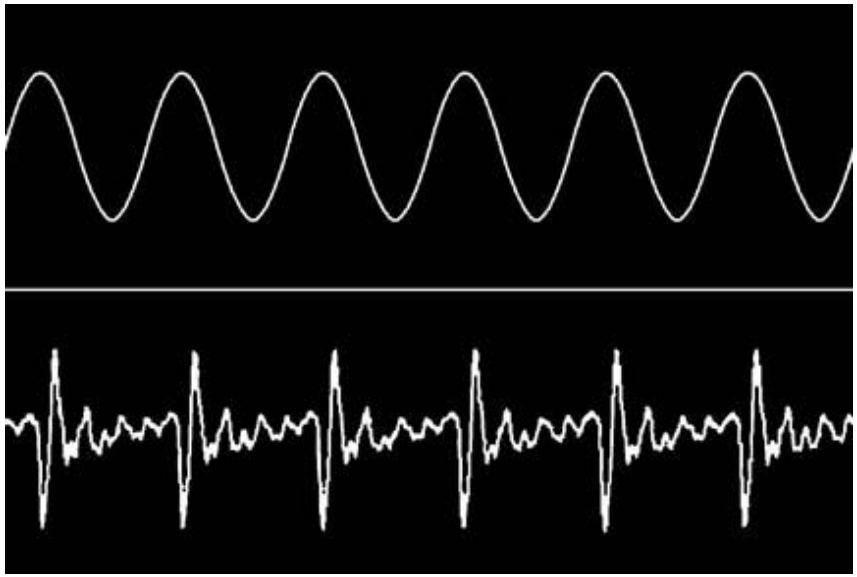
EE 109 Appendix A

Analog-to-Digital Conversion

ANALOG TO DIGITAL CONVERSION

Electric Signals

- Information is represented electronically as a time-varying voltage
  - Each voltage level may represent a unique value
  - Frequencies may represent unique values (e.g. sound)



Sound converted to electronic signal (voltage vs. time)

Electronic Information

- Digital Camera
  - CCD's (Charge-Coupled Devices) output a voltage proportional to the intensity of light hitting it
  - 3 CCD's filtered for measuring Red, Green, and Blue light produce 1 color pixel

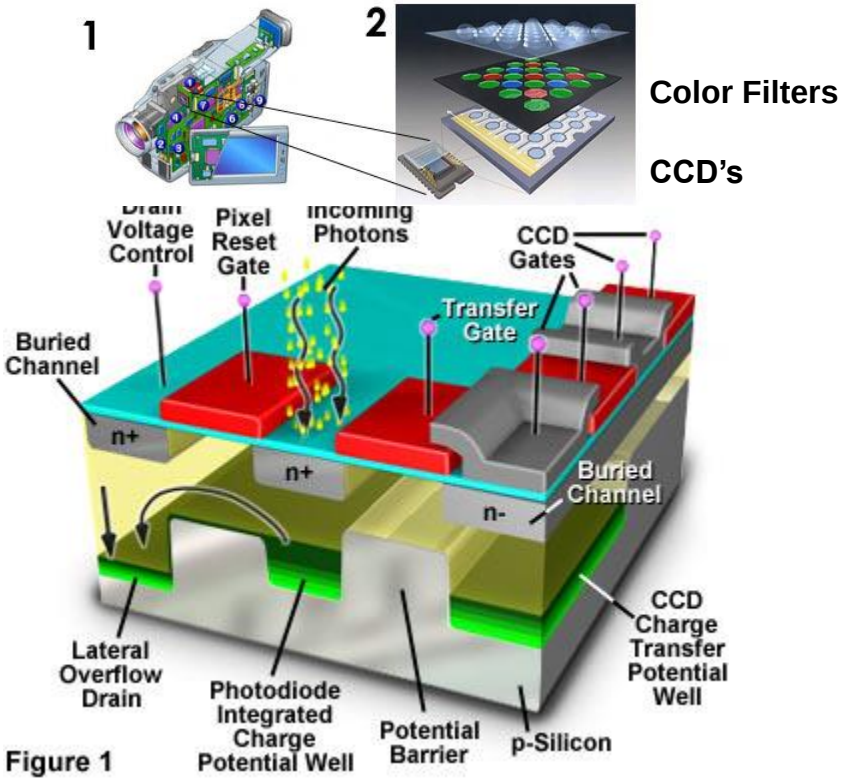
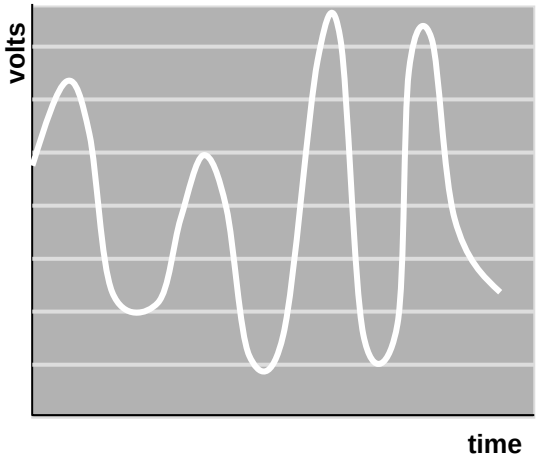


Figure 1

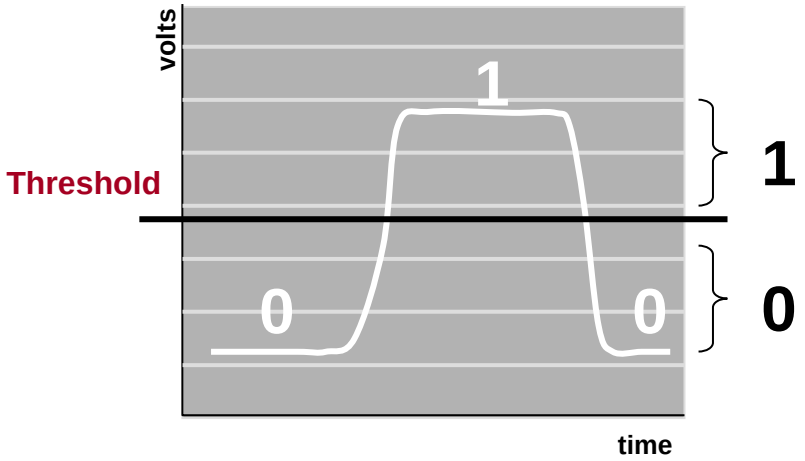
More info: http://www.science.ca/scientists/scientistprofile.php?pID=129
http://www.microscopy.fsu.edu/primer/digitalimaging/concepts/ccdanatomy.html

Signal Types

- Analog signal
  - Continuous time signal where each voltage level has a unique meaning
  - Most information types are inherently analog
- Digital signal
  - Continuous signal where voltage levels are mapped into 2 ranges meaning 0 or 1
  - Possible to convert a single analog signal to a set of digital signals



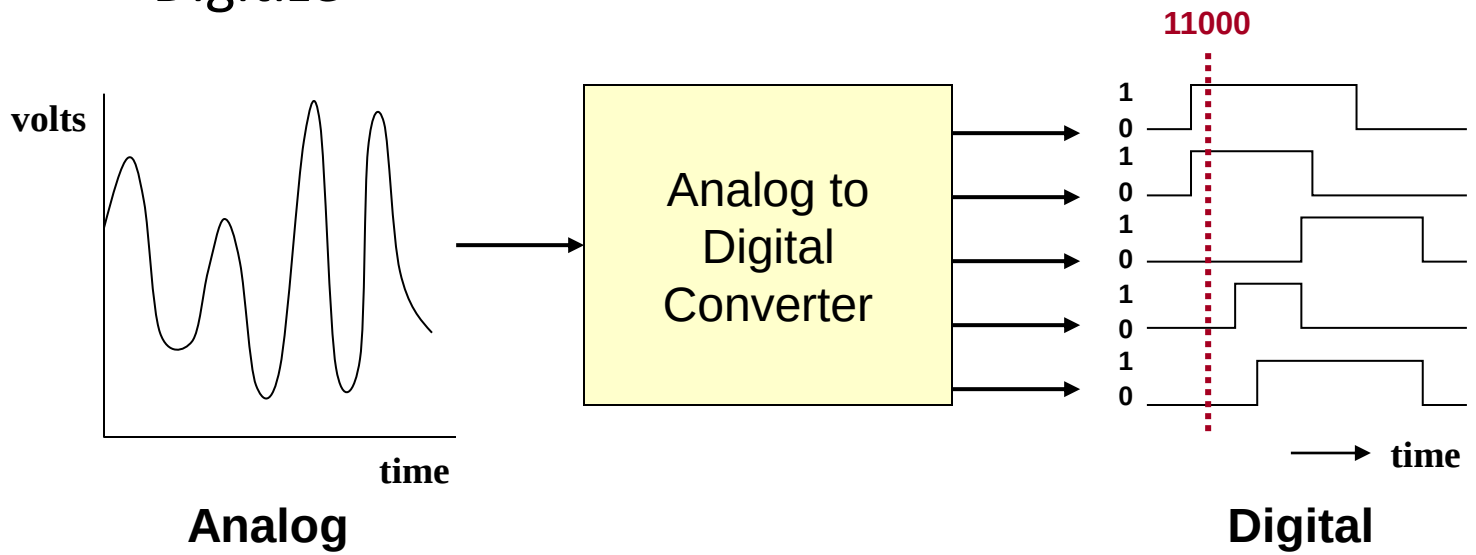
Analog



Digital

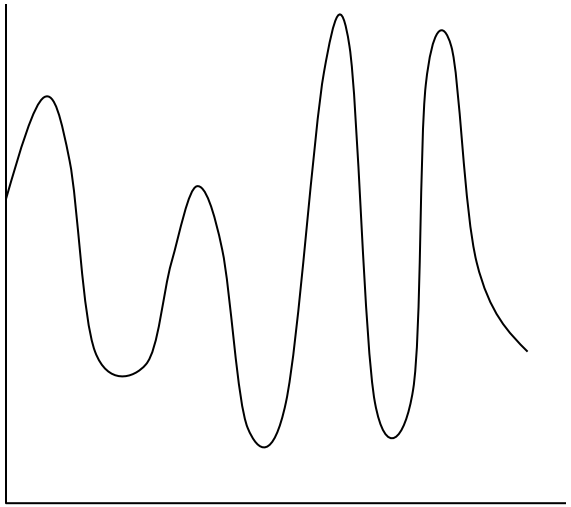
Analog to Digital Conversion

- 1 Analog signal can be converted to a *set* of digital signals (0's and 1's)
- 3 Step Process
  - Sample
  - Quantize (Measure)
  - Digitize

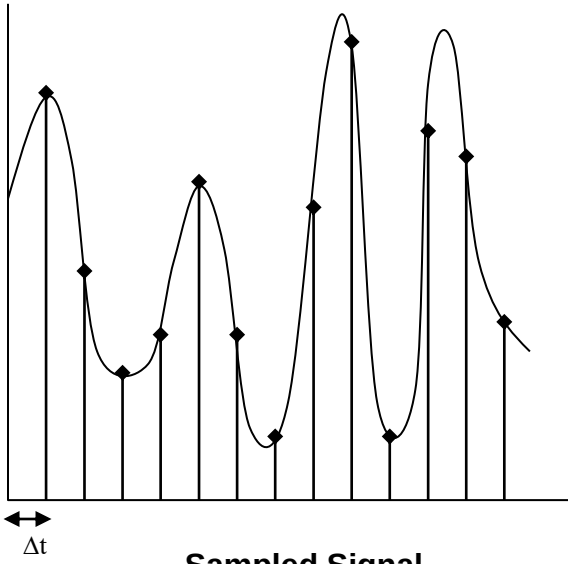


Sampling

- Measure (take samples) of the signals voltage at a regular time interval
- Sampling converts the continuous time scale into discrete time samples



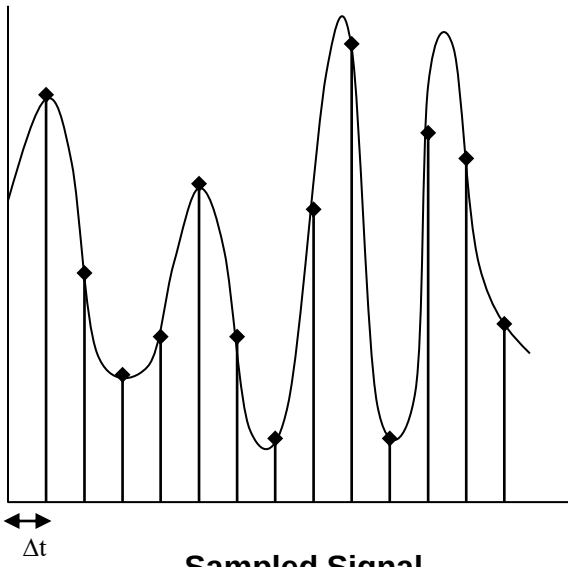
Original Analog Signal



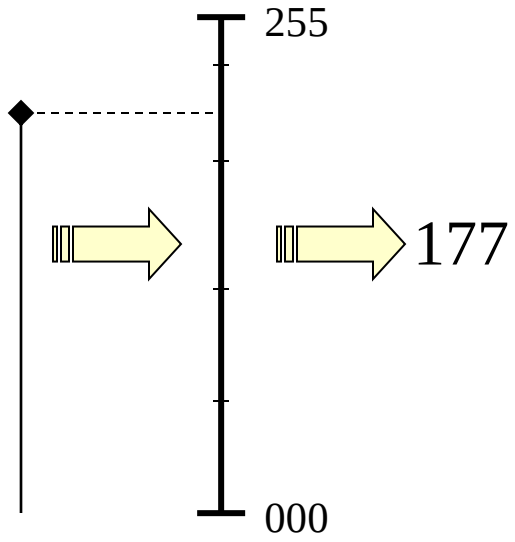
Sampled Signal

Quantization

- Voltage scale is divided into a set of finite numbers (e.g. 256 values: 0 – 255)
- Each sample is rounded to the nearest number on the scale
- Quantization converts continuous voltage scale to a discrete (finite) set of numbers



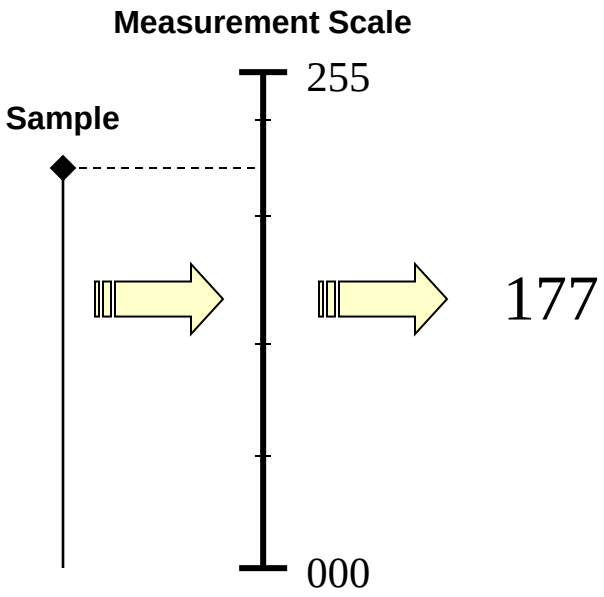
Sampled Signal



Each sample is quantized

Digitization

- The measured number from each sample is converted to a set of 1's and 0's



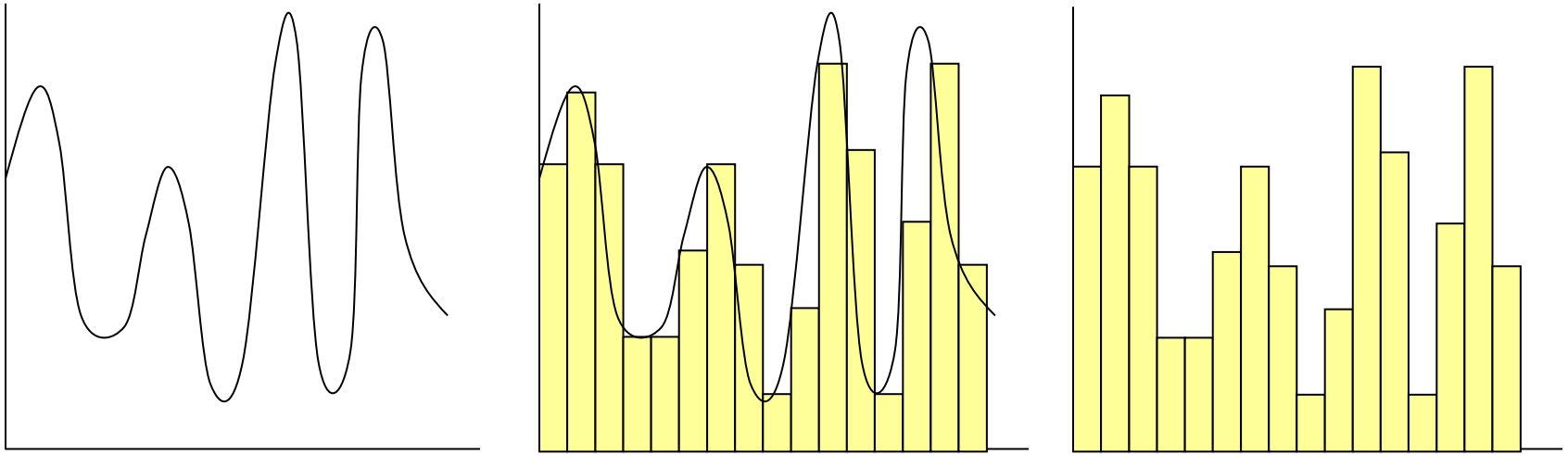
Each sample is quantized

177 = 10110001

Quantized value is converted to bits

Error

- Error is introduced because the discrete time and quantized samples only approximate the original analog signal

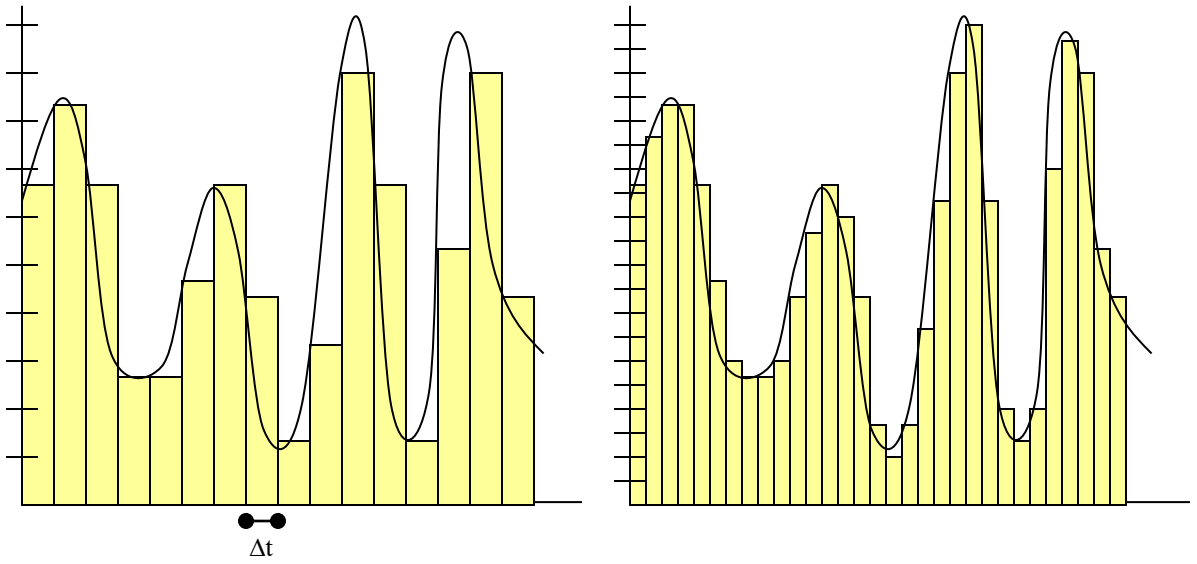


Original Analog Signal

Sampled Signal

Sampling Rates and Quantization Levels

- Higher sampling rates and quantization levels produce more accurate digital representations

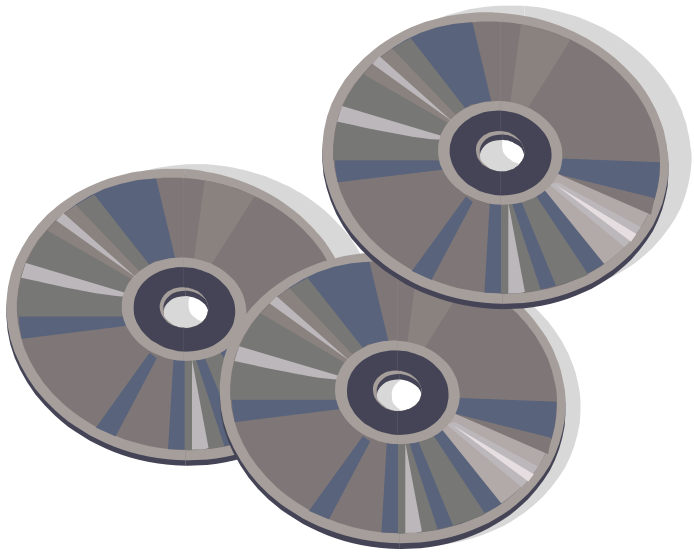


Lower sampling rate and quantization levels

Higher sampling rate and more quantization levels

Digital Sound

- CD Quality Sound
  - 44.1 Kilo-samples per second
  - 65,536 quantization levels (16-bits per sample)
  - 44.1KSamples * 16-bits/sample = 705 Kbps
- MP3 files compress that information to 128Kbps – 320 Kbps

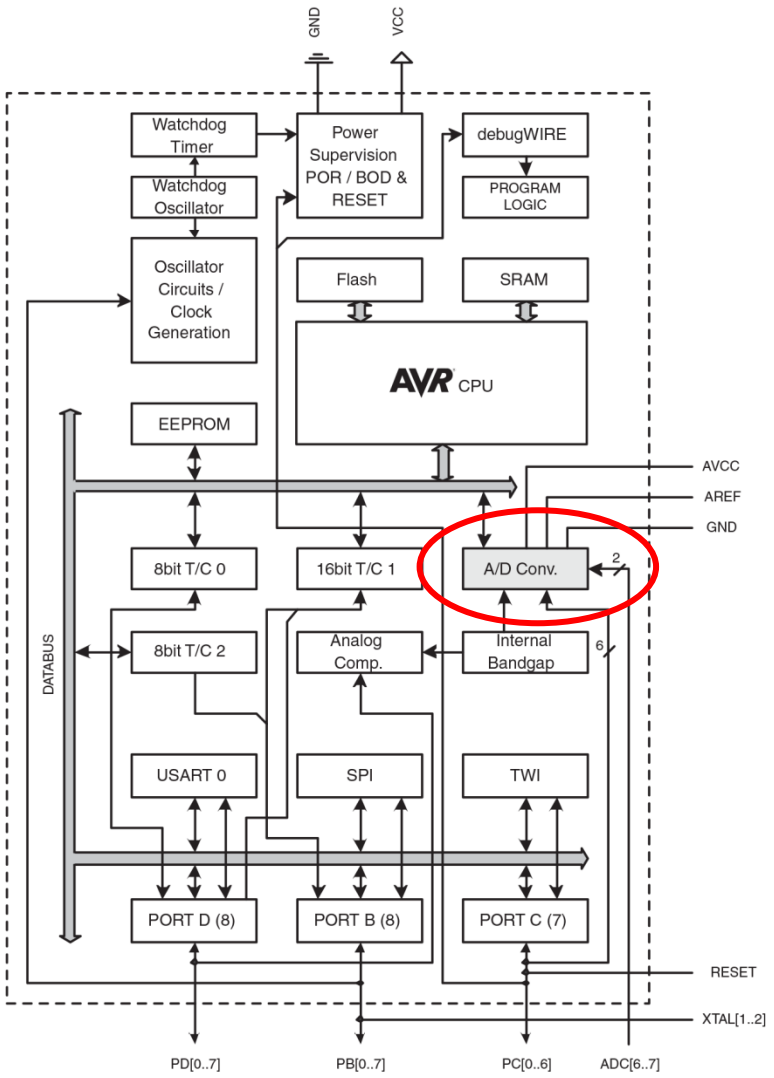


Converting voltages to digital numbers

ADC MODULE

ADC Module

- Your Atmel micro has an A-to-D Converter (ADC) built in
- The ADC module can be used to convert an analog voltage signal into 10 bit digital numbers.
- Not fast enough for video or audio.
- Controlled by a set of six registers which you must program appropriately



Note

- Microcontroller modules often come with many adjustable features and settings to make it useful to a wide variety of applications
- In EE 109 we may not want to use all that functionality so we have to enable or disable those features or alter certain settings
- How do we do this? By setting bits in specific registers
  - The values we program into the registers control how the hardware works!

ADC Registers

- ADC is primarily controlled by two registers whose bits control various aspects of the ADC
  - ADMUX – ADC Multiplexor Selection Register
  - ADCSRA – ADC Control and Status Register A
- We will see what these bits means as we continue through our slides…

ADMUX

7	6	5	4	3	2	1	0
REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0

ADCSRA

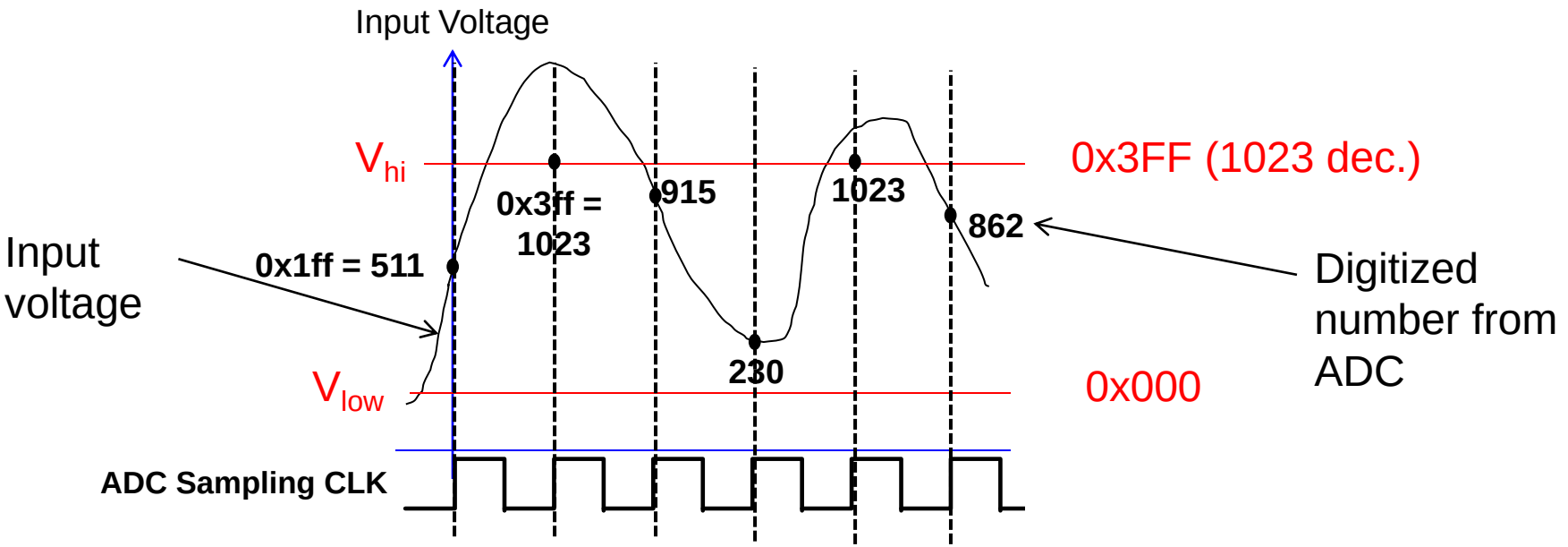
7	6	5	4	3	2	1	0
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

Only need to perform once before you start using the ADC

ADC INITIALIZATION PROCESS

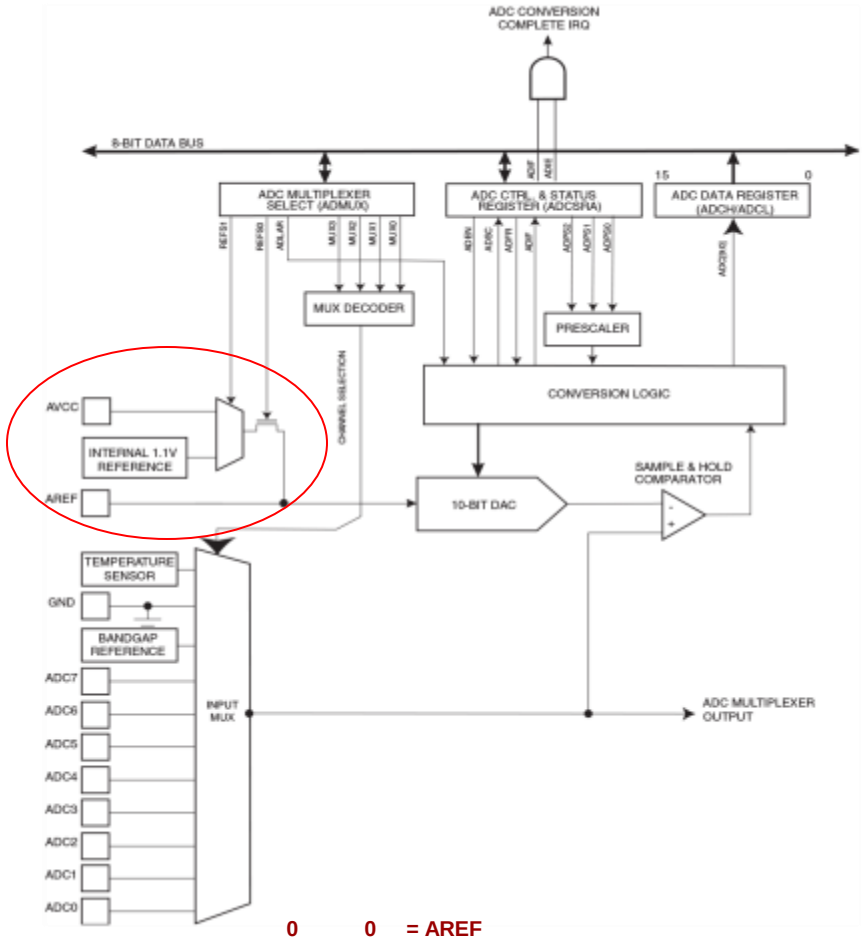
ADC Voltage Reference

- The ADC can only measure voltages in the range of V_{hi} to V_{low}
  - If the voltage is higher than Vhi it just converts to 1023=0x3ff
  - If the voltage is lower than Vlow it just converts to 0
  - Voltages between the limits are converted linearly to digital values.
- Samples will be taken either at regular intervals or just when you tell it to take a sample



ADC Voltage Reference

- The low reference is fixed at ground = 0V.
- High reference is selectable
  - AVCC (connected to VCC)
    - Usually the one we want!
  - AREF
  - Internal 1.1V reference
- Reference selection controlled by bits in a register
- **ADC Init Step 1**: Set REF bits to choose AVCC to give analog range of 0-5V
  - Set ADMUX register bit
    - REFS1 to a 0
    - REFS0 to a 1



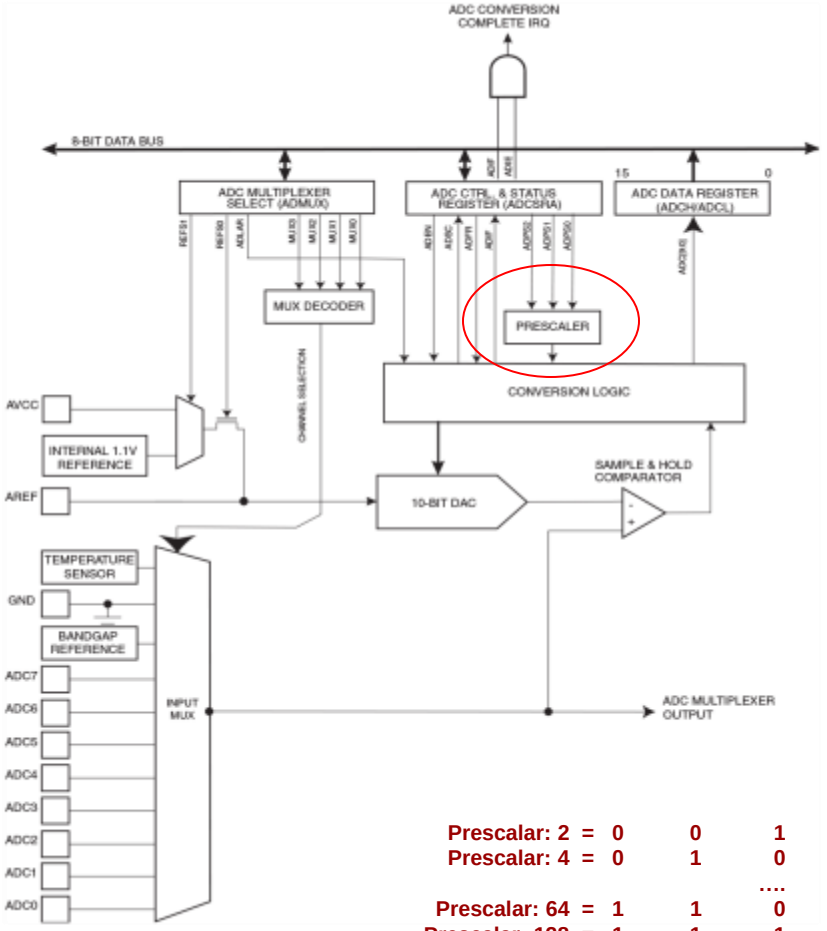
0 0 = AREF
0 1 = AVCC
1 1 = Int 1.1V

REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0

ADMUX Register

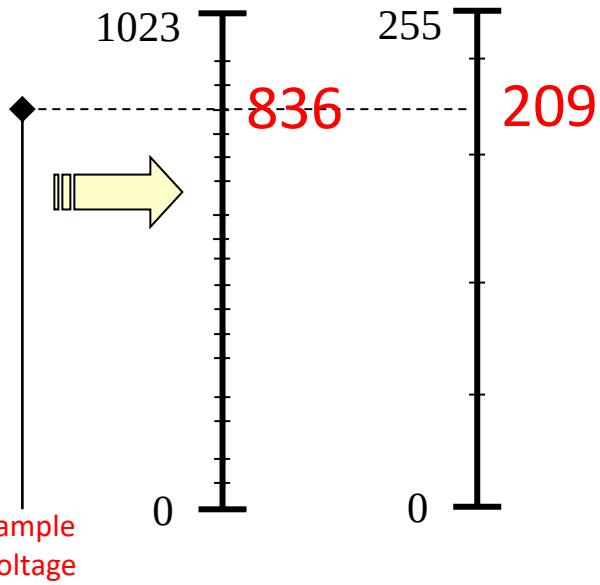
ADC Clock Generation

- **Documentation requirement**: The ADC needs a clock in the range 50kHz to 200kHz in order to operate.
- Clock generated for the Arduino's processor is 16Mhz
- Prescalar (a.k.a. divider) reduces the clock to a lower frequency by dividing its frequency
- Divide by 2, 4, 8, 16, 32, 64, or 128
  - $ADC\ Freq = \frac{CPU\ Clock\ Freq}{Prescalar}$
  - If Precalar=64 then ADC Freq = 16MHz / 64 = 250KHz (still too fast)
- **ADC Init Step 3**: Set prescalar to 128 by turning on (setting) ADPS2, ADPS1, ADPS0 bits in ADCSRA register



Scale

- Analogy: Some scales give your weight to the nearest pound (137) while others are accurate to the tenth of pound (137.6)
  - It's nice to have accuracy but for most of us we are content with the accuracy just at the nearest pound
- Our ADC can provide readings up to 10-bits accuracy (on a scale from 1023)…
- …but it can also drop the lower 2 bits to provide readings of 8-bit accuracy (on a scale from 256)
- The question is simply do we need 10-bit accuracy or is 8-bit accuracy sufficient
- **In EE109 we'll always use 8-bit readings**
- **ADC Init Step 4:** Set ADLAR bit to 1 in the ADMUX register (1 = 8-bit results, 0 = 10-bit results)



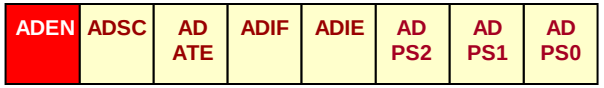
REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0

ADMUX Register

Enable the ADC

- The ADC module has an 'enable' bit which effectively acts as an on/off switch (turn off to save power)
- **ADC Init Step 5:** Set ADEN bit to 1

1 = Enable
0 = Disable



ADCSRA Register

ADC Register Review

- ADMUX – ADC Multiplexor Selection Register
  - REFS - Voltage reference selection (bits 7-6)
    - 01 to select AVCC, connected to VCC (+5V) on µC
  - ADLAR - Left adjust results (bit 5)
    - 0 = "right adjust" for 10-bit result
    - 1 = "left adjust" for 8-bit result
  - MUX - Input channel selection (bits 3-0)
    - Use values 0000 to 0101 to select pins A0 to A5

ADMUX

7	6	5	4	3	2	1	0
REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0

ADC Register Review

- ADCSRA – ADC Control and Status Register A
  - ADPS - Prescalar selection (bits 2-0)
    - Selects the clock divisor used in the prescaler
  - ADEN – ADC Enable (bit 7)
    - Set to 1 to turn on the ADC (must do)
  - ADSC – ADC Start Conversion (bit 6) **[More on this in a few slides]**
    - Set to 1 to start a conversion
    - When goes to a zero, conversion is complete
  - Other bits for generating interrupts (to be discussed in future labs)

ADCSRA

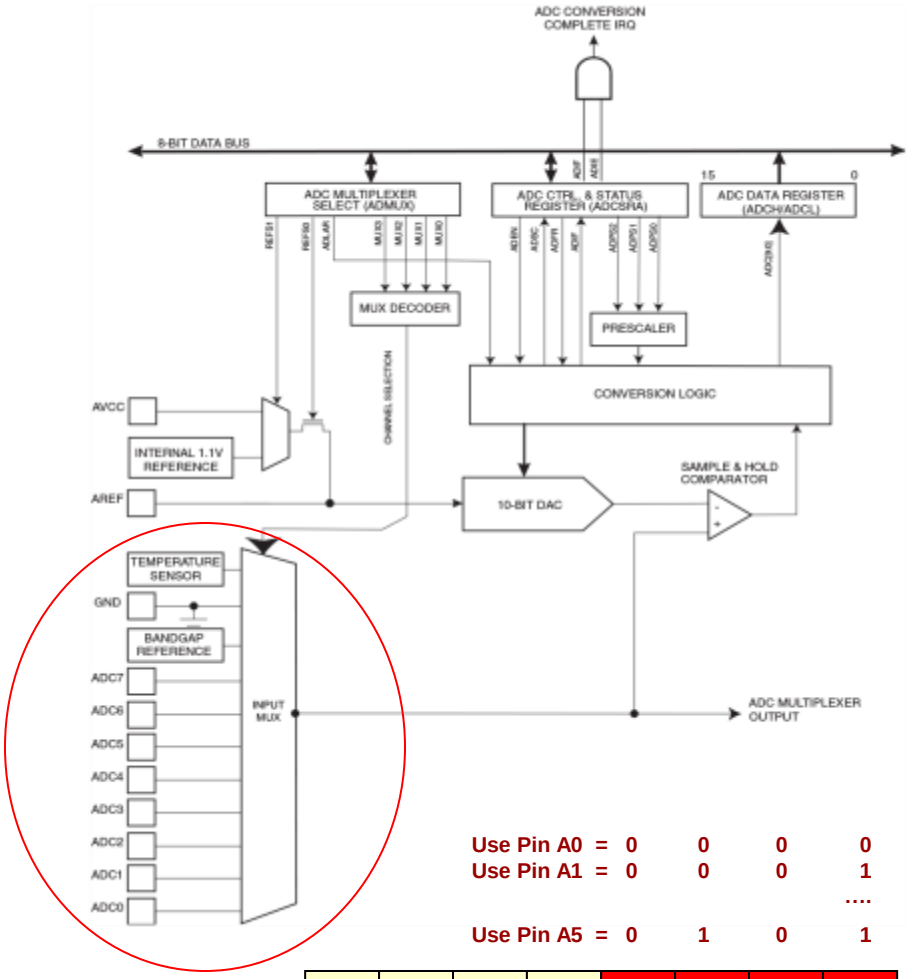
7	6	5	4	3	2	1	0
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

Perform each time you want to take a new sample

ADC SAMPLING PROCESS

ADC Input Selection

- The ADC has six input channels/pins that can be connected to the one built-in converter
- Only one channel can be converted at any one time (i.e. is internally muxed)
- Channel selection controlled by bits in a register
- **ADC Sample Step 0:** Set MUX bits in ADMUX register to desired channel number
  - If we want channel A3, set mux bits to 0011



Selecting a Channel

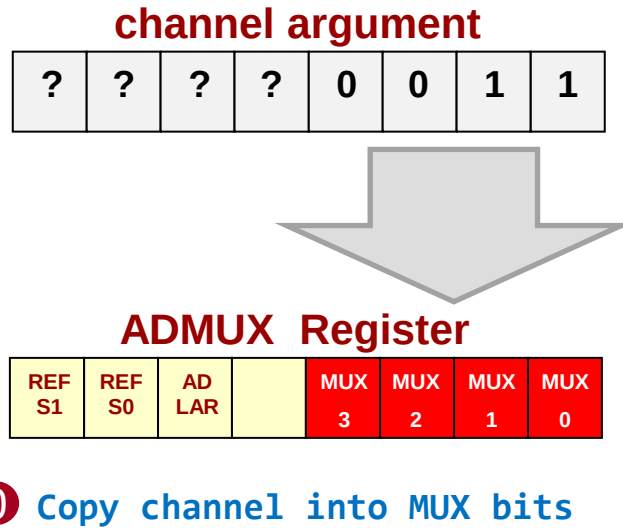
- **ADC Sampling Step 0**: Copy the 4-bit channel argument to the ADMUX register

```
unsigned char adc_sample(char channel)
{
  // Step 0: copy channel bits into ADMUX

  // Step 1: Start a sample

  // Step 2: Wait for ADC to indicate
  //         the sample is ready

  // Step 3: Retrieve and return the sample
}
```



❶ Copy channel into MUX bits

Starting a Sample

- The ADC does not continuously sample
- We must tell it when to take a sample by setting the 'start' bit (ADSC)
- **ADC Sampling Step 1**: Set the ADSC bit in the ADCSRA register
- Some time will elapse while the ADC takes the sample. During this time the ADSC bit will remain at 1
- When the ADC is done it will AUTOMATICALLY clear the ADSC bit to 0
- **ADC Sampling Step 2**
  - Need to continuously check whether the ADSC bit has turned back to 0 (i.e. loop *while* the ADSC is still a 1)

ADMUX Register

REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0

0 `Copy channel into MUX bits`

1 = Start/(ed)
0 = Done

ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADCSRA Register

1 `ADCSRA |= _____;`

ADEN	ADSC = 1	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

2

3 `while((ADCSRA & ___) != 0 )`

… `{}`

ADEN	ADSC = 1	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

t

ADEN	ADSC = 0	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

Retrieving a Sample

- [**From last slide**] Need to continuously check whether the ADSC bit has turned back to 0 (i.e. loop *while* the ADSC is still a 1)
  - Once the loop finishes we know the sample is ready!
- **ADC Sampling Step 3:** Read (retrieve) the 8-bit sample result from the **ADCH** register
  - Just read the value from ADCH (i.e. `unsigned char result = ADCH;`) and then use that value in your application
- You can repeat the process as many times as you like
  - Set the start (ADSC) bit
  - Loop until the start (ADSC) bit goes to 0
  - Retrieve the sample from ADCH

2
3
…

```
while((ADCSRA & ___) != 0 )
{}
```

ADEN	ADSC	AD ATE	ADIF	ADIE	AD PS2	AD PS1	AD PS0
	1						

t

ADEN	ADSC	AD ATE	ADIF	ADIE	AD PS2	AD PS1	AD PS0
	0						

```
unsigned char result = ADCH;
```

7	6	5	4	3	2	1	0
	8-bit ADC Sample						

ADCH Register

Named Bit Constants

- In `<avr/io.h>` there are constants defined for each bit name and position
  - `REFS1 = 7, REFS0 = 6, ADLAR = 5, …`
  - `ADEN = 7, ADSC = 6, …`
- Using these we can write shift expressions with more clarity
  - `ADCSRA |= (1 << ADSC);`
  - `ADMUX &= ~(1 << ADLAR)`