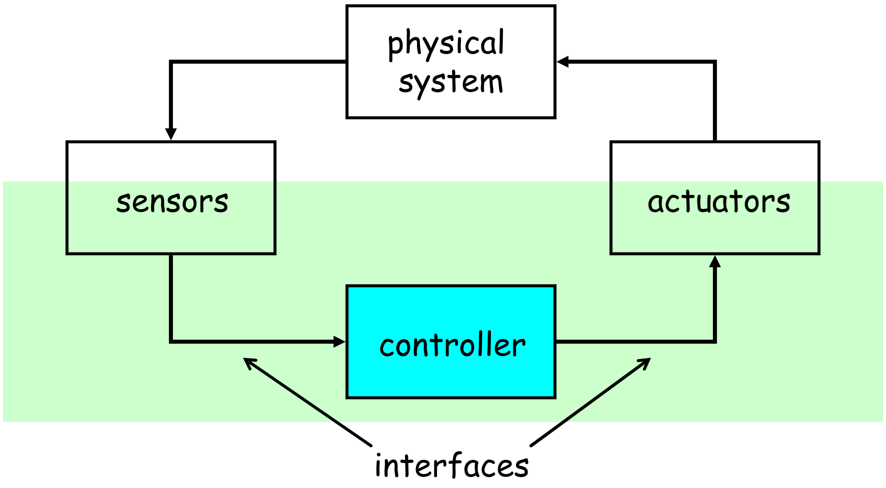
Interfacing

- Connecting the computational capabilities of a microcontroller to external signals
  - Transforming variable values into voltages and vice-versa
  - Digital and analog
- Issues
  - How many signals can be controlled?
  - How can digital and/or analog inputs be used to measure different physical phenomena?
  - How can digital and/or analog inputs be used to control different physical phenomena?

Controlling and reacting to the environment

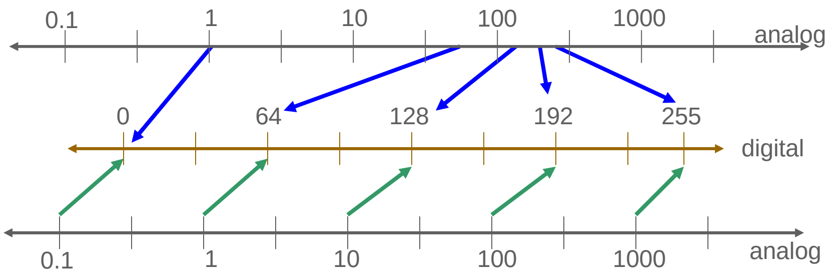
- To control or react to the environment we need to interface the microcontroller to peripheral devices
  - Microcontroller may contain specialized interfaces to sensors and actuators
- Things we want to measure or control
  - light, temperature, sound, pressure, velocity, position
- Sensors
  - e.g., switches, photoresistors, accelerometers, compass, sonar
- Actuators
  - e.g., motors, relays, LEDs, sonar, displays, buzzers

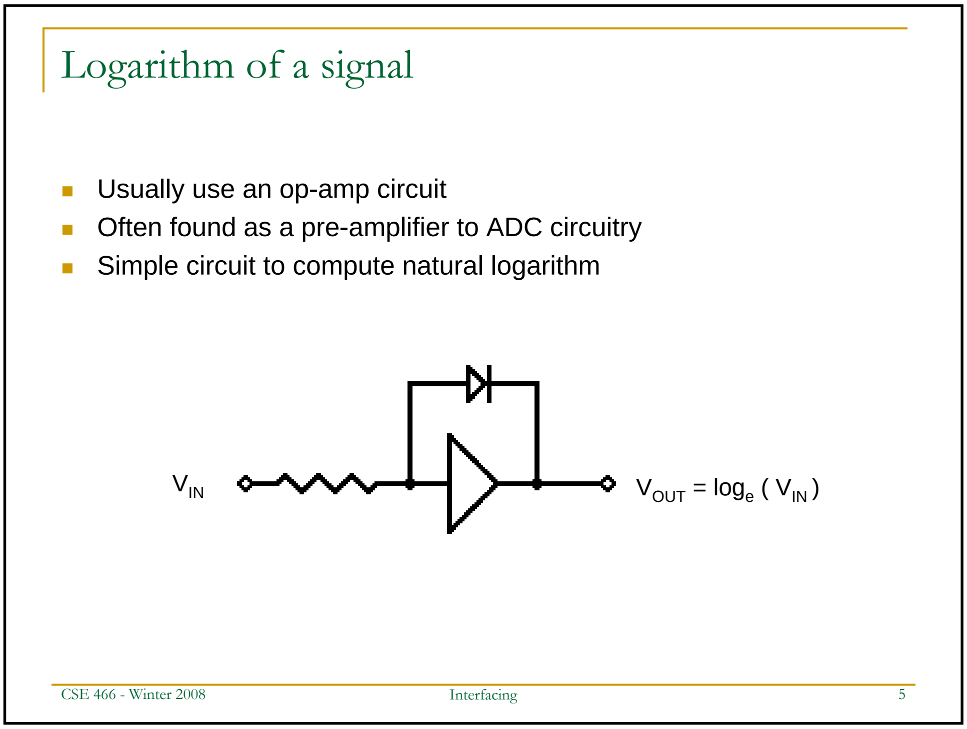
Typical control system



Analog to digital conversion

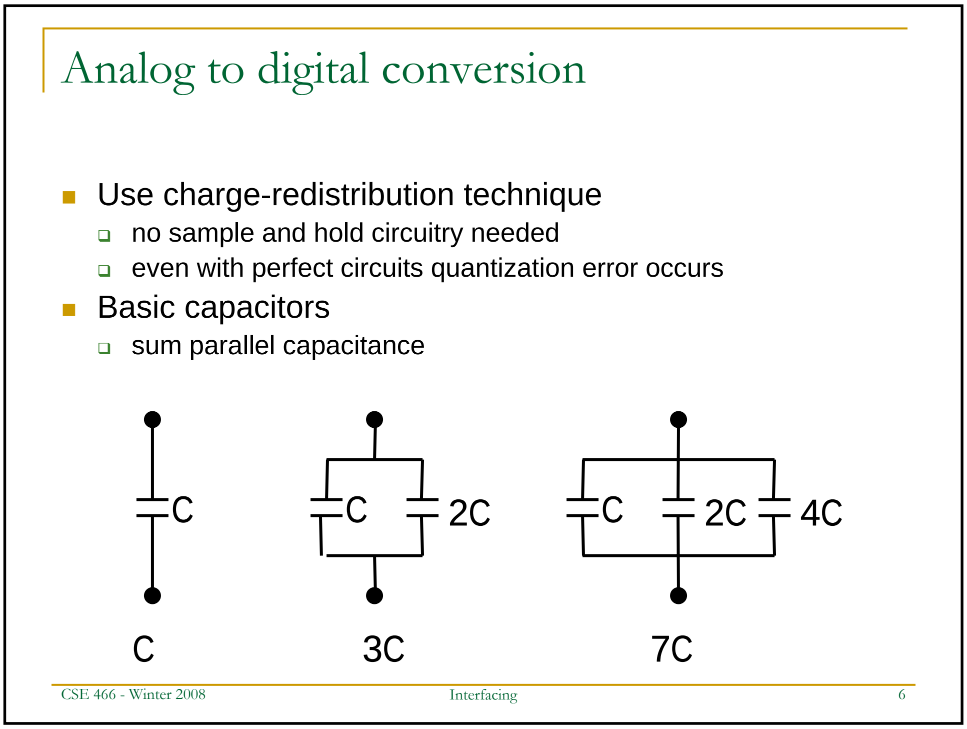
- Map analog inputs to a range of binary values
  - 8-bit A/D has outputs in range 0-255
- What if we need more information?
  - linear vs. logarithmic mappings
  - larger range of outputs (16-bit a/d)





Logarithm of a signal

- Usually use an op-amp circuit
- Often found as a pre-amplifier to ADC circuitry
- Simple circuit to compute natural logarithm

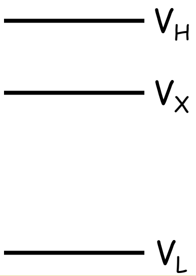


Analog to digital conversion

- Use charge-redistribution technique
  - no sample and hold circuitry needed
  - even with perfect circuits quantization error occurs
- Basic capacitors
  - sum parallel capacitance

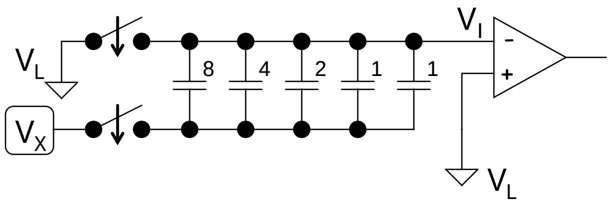
Analog to digital conversion (cont'd)

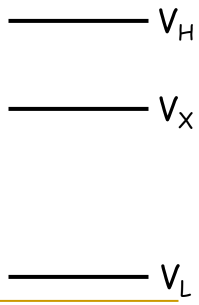
- Two reference voltage
  - mark bottom and top end of range of analog values that can be converted (V_L and V_H)
  - voltage to convert must be within these bounds (V_X)
- Successive approximation
  - most approaches to A/D conversion are based on this
  - 8 to 16 bits of accuracy
- Approach
  - sample value
  - hold it so it doesn't change
  - successively approximate
  - report closest match

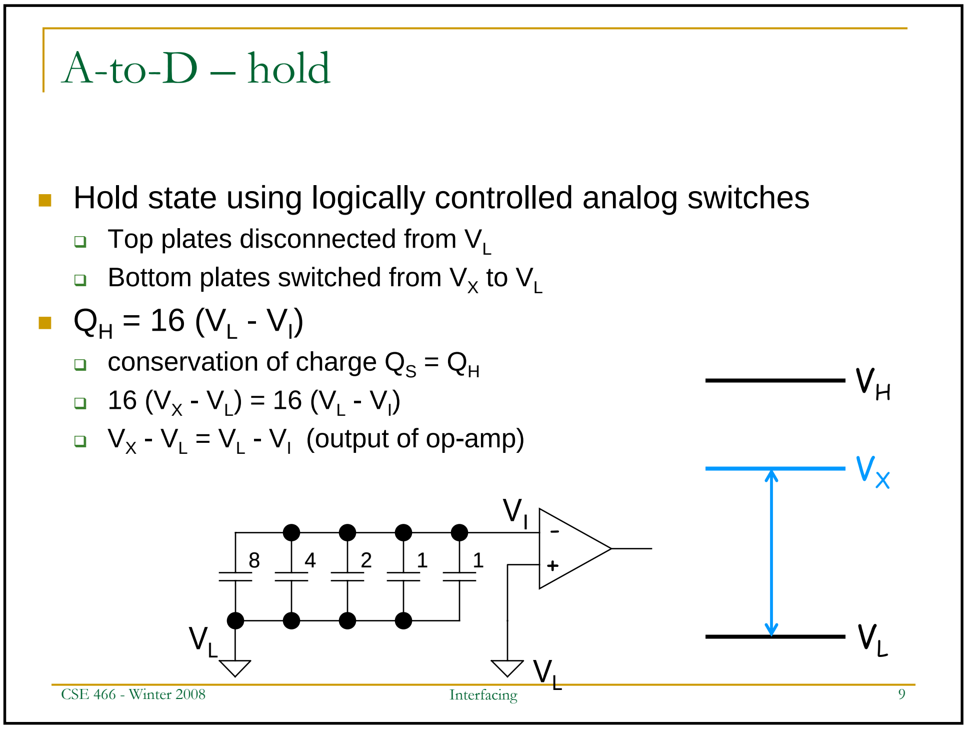


A-to-D – sample

- During the sample time the top plate of all capacitors is switched to reference low V_L
- Bottom plate is set to unknown analog input V_X
- Q = CV
- $Q_S = 16 (V_X - V_L)$

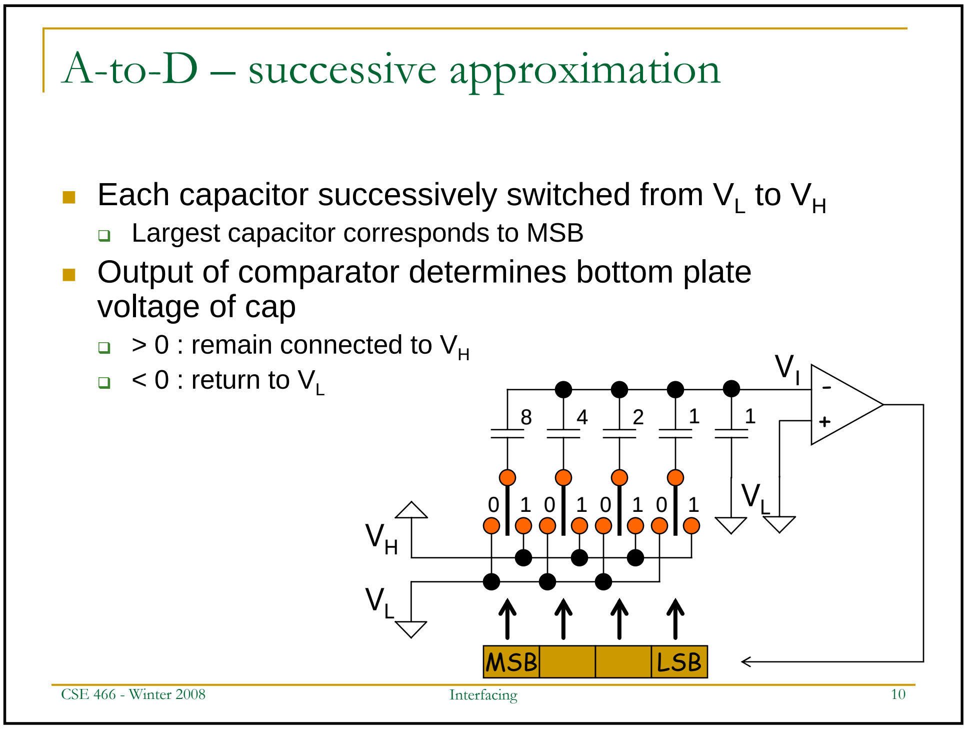






A-to-D – hold

- Hold state using logically controlled analog switches
  - Top plates disconnected from V_L
  - Bottom plates switched from V_X to V_L
- $Q_H = 16 (V_L - V_I)$
  - conservation of charge $Q_S = Q_H$
  - $16 (V_X - V_L) = 16 (V_L - V_I)$
  - $V_X - V_L = V_L - V_I$ (output of op-amp)

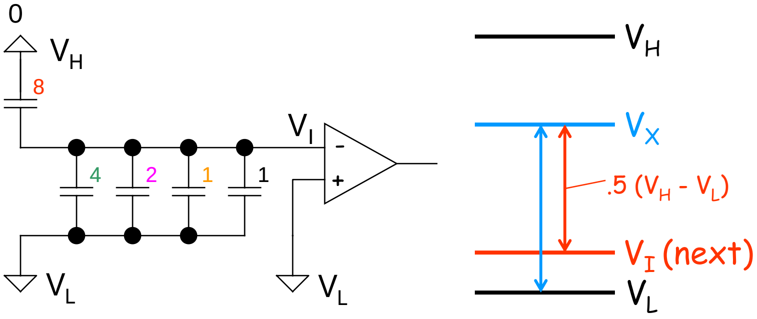


A-to-D – successive approximation

- Each capacitor successively switched from V_L to V_H
  - Largest capacitor corresponds to MSB
- Output of comparator determines bottom plate voltage of cap
  - > 0 : remain connected to V_H
  - < 0 : return to V_L

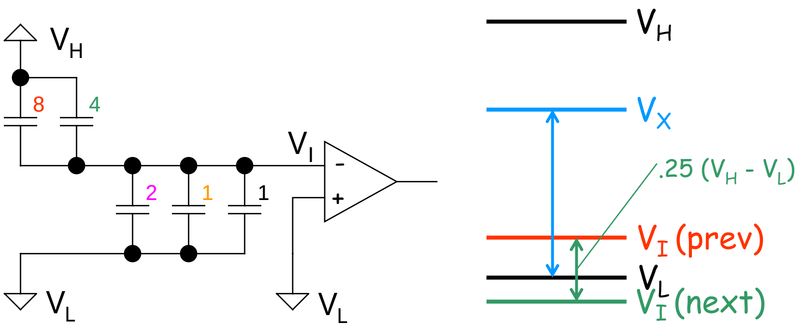
A-to-D example - MSB

- Suppose V_X = 21/32 (V_H - V_L) and already sampled
- Compare after shifting half of capacitance to V_H
  - V_I goes up by + 8/16 (V_H-V_I) - 8/16 (V_L-V_I) = + 8/16 (V_H - V_L)
  - original V_L - V_I goes down and becomes
  - V_L - (V_I + .5 (V_H - V_L)) = V_L - V_I - .5 (V_H - V_L)
- Output > 0



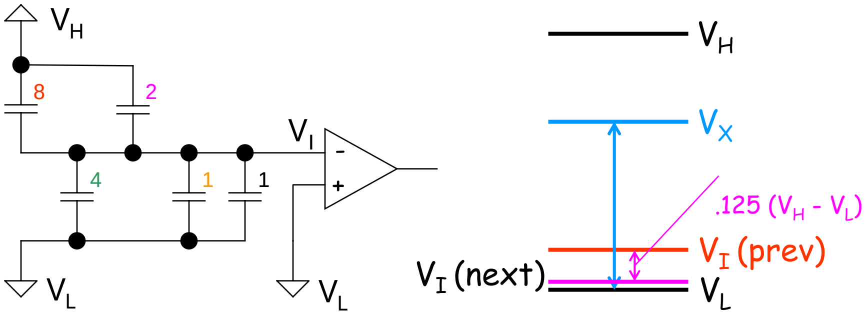
A-to-D example - (MSB-1)

- Compare after shifting another part of cap. to V_H
  - V_I goes up by + 4/16 (V_H-V_I) - 4/16 (V_L-V_I) = + 4/16 (V_H - V_L)
  - original V_L - V_I goes down and becomes
  - V_L - (V_I + .25 (V_H - V_L)) = V_L - V_I - .25 (V_H - V_L)
- Output < 0 (went too far)



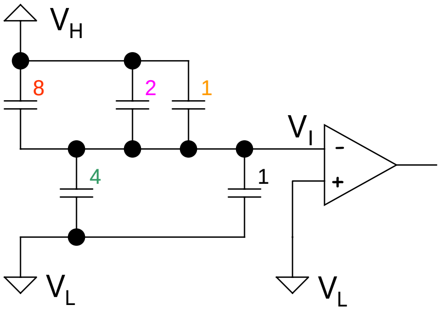
A-to-D example - (MSB-2)

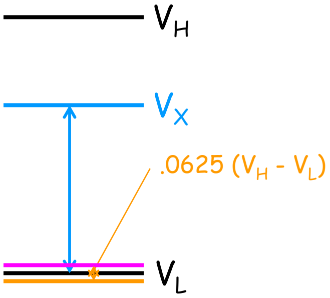
- Compare after shifting another part of cap. to V_H
  - V_I goes up by + 2/16 $(V_H - V_I)$ - 2/16 $(V_L - V_I)$ = + 2/16 $(V_H - V_L)$
  - original $V_L - V_I$ goes down and becomes
  - $V_L - (V_I + .125 (V_H - V_L)) = V_L - V_I - .125 (V_H - V_L)$
- Output > 0



A-to-D example - LSB

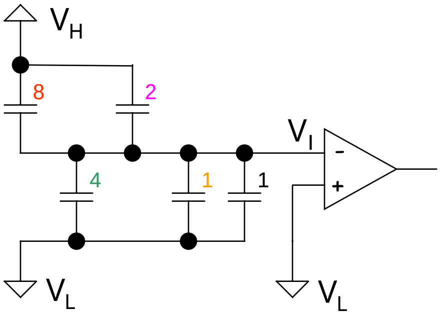
- Compare after shifting another part of cap. to V_H
  - V_I goes up by + 1/16 $(V_H - V_I)$ - 1/16 $(V_L - V_I)$ = + 1/16 $(V_H - V_L)$
  - original $V_L - V_I$ goes down and becomes
  - $V_L - (V_I + .0625 (V_H - V_L)) = V_L - V_I - .0625 (V_H - V_L)$
- Output < 0 (went too far again)



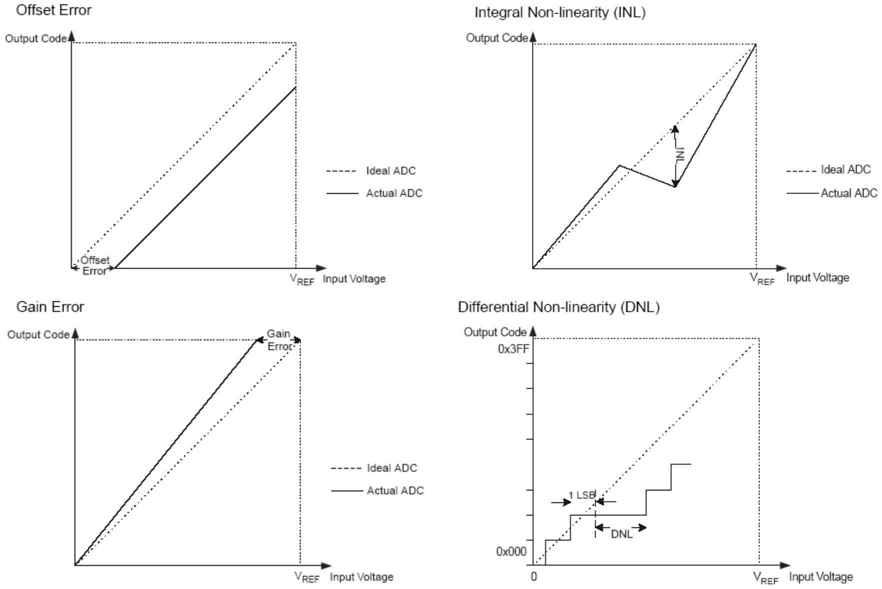


A-to-D example final result

- Input sample of 21/32
- Gives result of <u>1010</u> or 10/16 = 20/32
- 3% error



A-to-D Conversion Errors


Offset Error

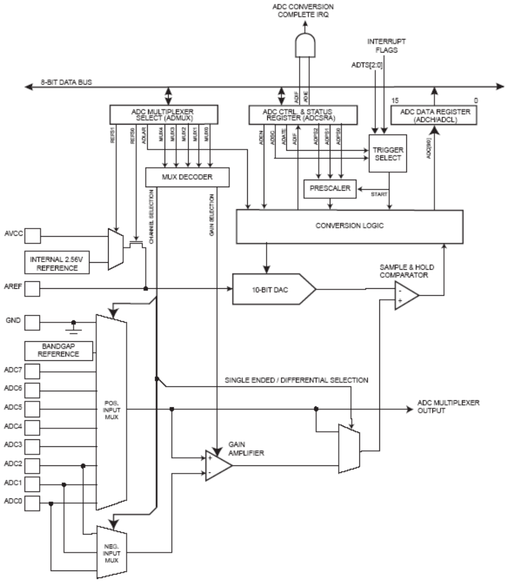
Integral Non-linearity (INL)

Gain Error

Differential Non-linearity (DNL)

Closer Look at A-to-D Conversion

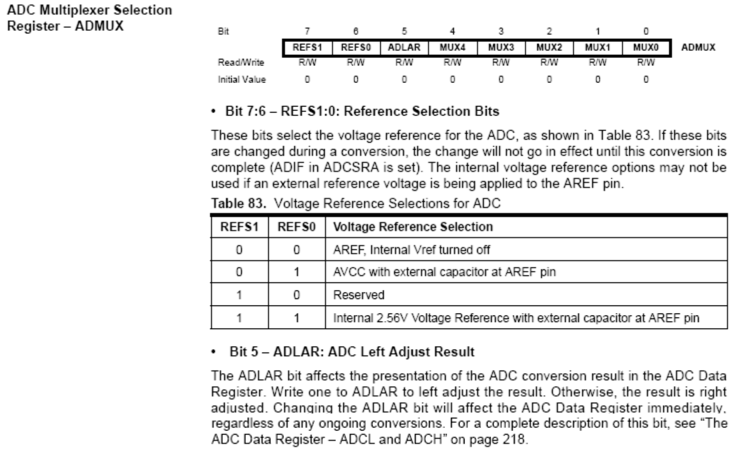
- Needs a comparator and a D-to-A converter
- Takes time to do successive approximation
- Interrupt generated when conversion is completed



A-to-D Conversion on the ATmega16

- 10-bit resolution (adjusted to 8 bits as needed)
- 65-260 usec conversion time
- 8 multiplexed input channels
- Capability to do differential conversion
  - Difference of two pins
  - Optional gain on differential signal (amplifies difference)
- Interrupt on completion of A-to-D conversion
- 0-V_{CC} input range
- 2*LSB accuracy (2 * 1/1024 = ~0.2%)
  - Susceptible to noise – special analog supply pin (AVCC) and capacitor connection for reference voltage (AREF)

A-to-D Conversion (cont’d)



ADC Multiplexer Selection Register – ADMUX

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

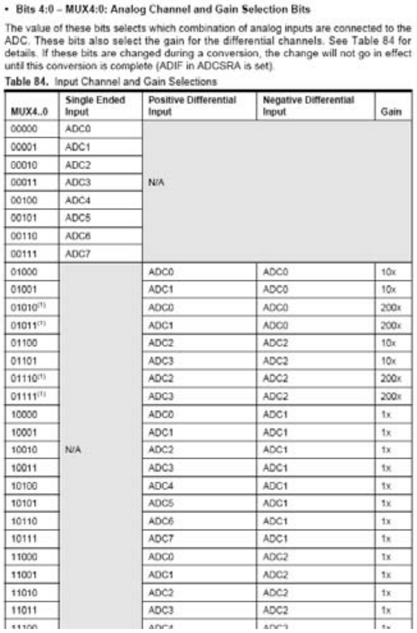
REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

- **Bit 5 – ADLAR: ADC Left Adjust Result**

The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately, regardless of any ongoing conversions. For a complete description of this bit, see “The ADC Data Register – ADCL and ADCH” on page 218.

A-to-D Conversion (cont’d)

- Single-ended or differential
  - 1 of 8 single-ended
  - ADCx – ADC1 at 1x gain
  - ADC{0,1} – ADC0 at 10x
  - ADC{0,1} – ADC0 at 200x
  - ADC{2,3} – ADC2 at 10x
  - ADC{2,3} – ADC3 at 200x
  - ADC{0,1,2,3,4,5} – ADC2 at 1x

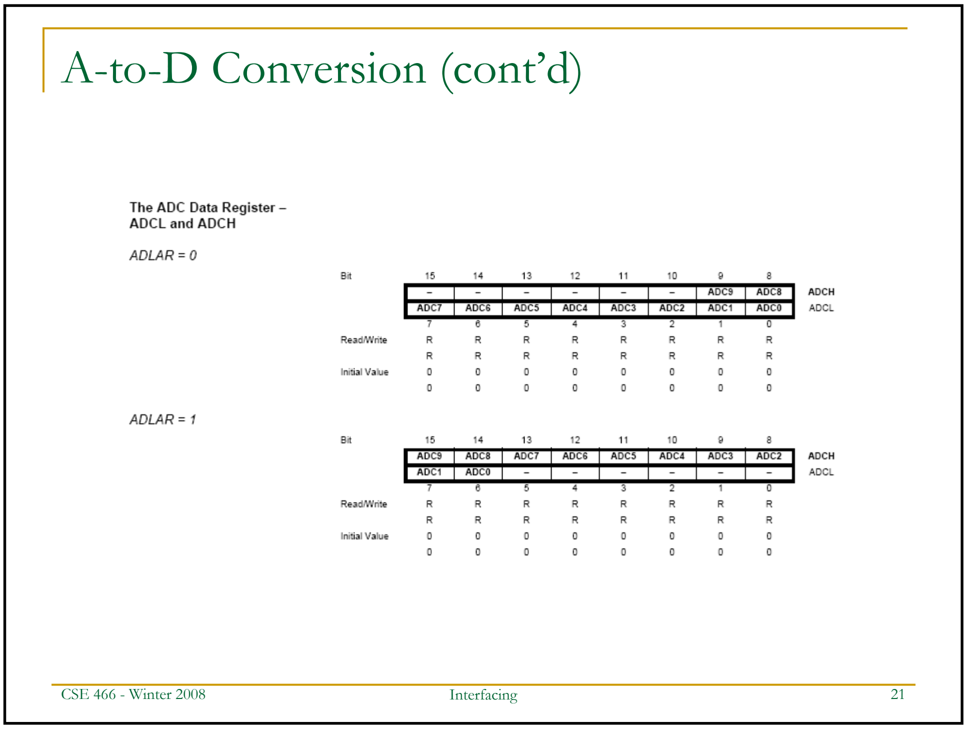


- **Bits 4:0 – MUX4:0: Analog Channel and Gain Selection Bits**

The value of these bits selects which combination of analog inputs are connected to the ADC. These bits also select the gain for the differential channels. See Table 84 for details. 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).

Table 84. Input Channel and Gain Selections

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010[1]		ADC0	ADC0	200x
01011[1]		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110[1]		ADC2	ADC2	200x
01111[1]		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x



A-to-D Conversion (cont’d)

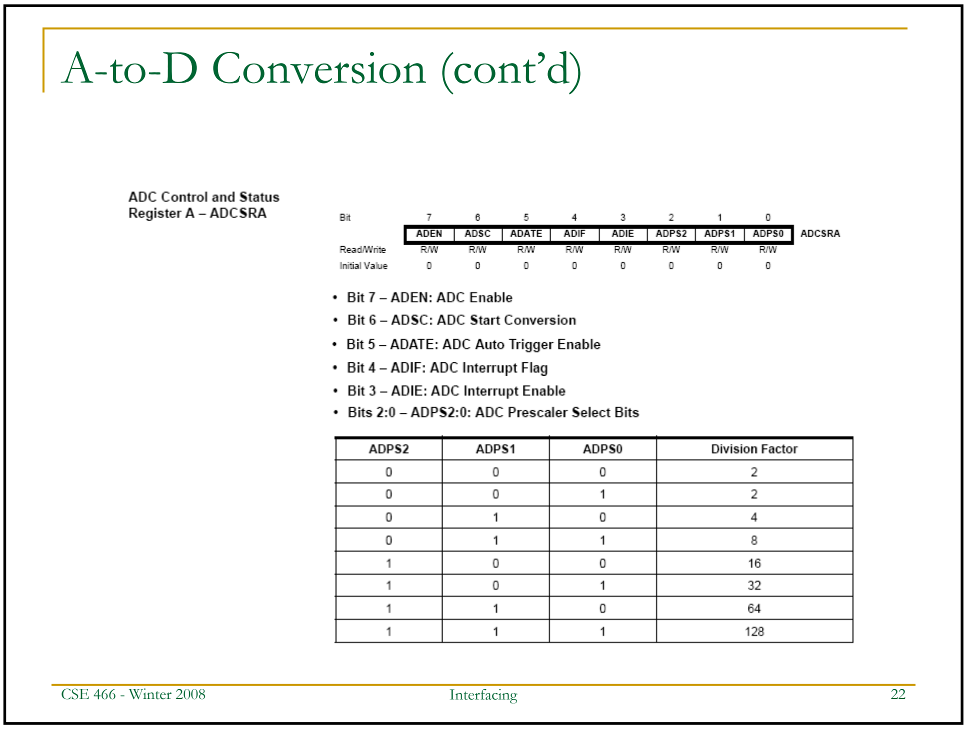
The ADC Data Register – ADCL and ADCH

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	



A-to-D Conversion (cont’d)

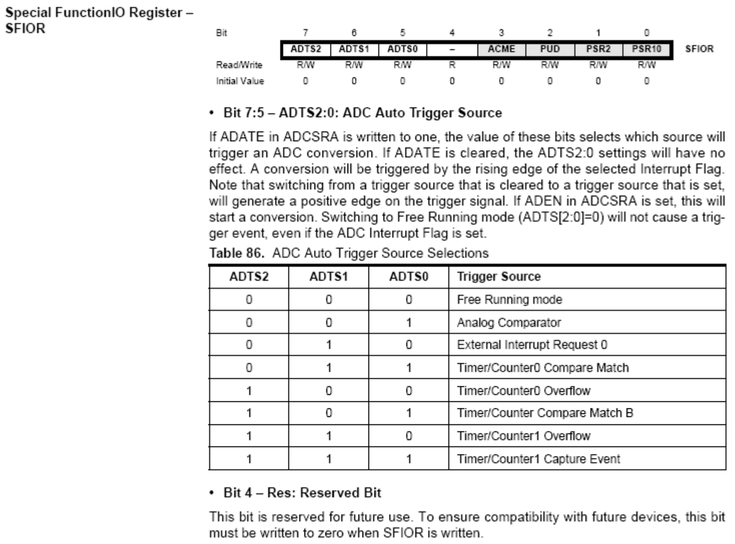
ADC Control and Status Register A – ADCSRA

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**
- **Bit 6 – ADSC: ADC Start Conversion**
- **Bit 5 – ADATE: ADC Auto Trigger Enable**
- **Bit 4 – ADIF: ADC Interrupt Flag**
- **Bit 3 – ADIE: ADC Interrupt Enable**
- **Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits**

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

A-to-D Conversion (cont'd)



Special FunctionIO Register – SFIOR

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:5 – ADTS2:0: ADC Auto Trigger Source**

If ADATE in ADCSRA is written to one, the value of these bits selects which source will trigger an ADC conversion. If ADATE is cleared, the ADTS2:0 settings will have no effect. A conversion will be triggered by the rising edge of the selected Interrupt Flag. Note that switching from a trigger source that is cleared to a trigger source that is set, will generate a positive edge on the trigger signal. If ADEN in ADCSRA is set, this will start a conversion. Switching to Free Running mode (ADTS[2:0]=0) will not cause a trigger event, even if the ADC Interrupt Flag is set.

Table 86. ADC Auto Trigger Source Selections

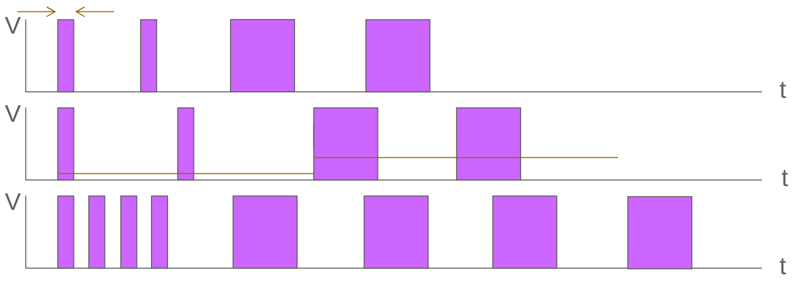
ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

- **Bit 4 – Res: Reserved Bit**

This bit is reserved for future use. To ensure compatibility with future devices, this bit must be written to zero when SFIOR is written.

Digital to analog conversion

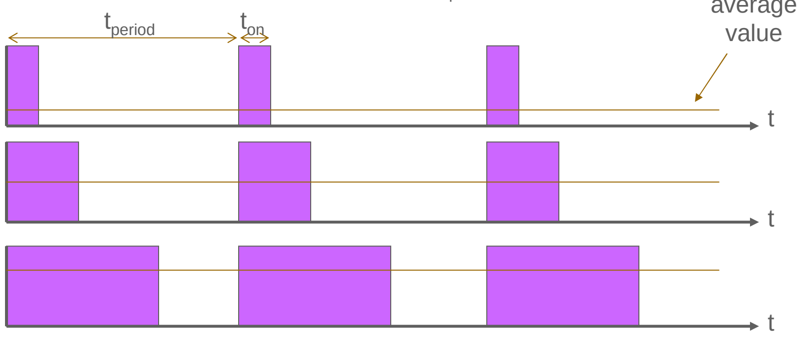
- Map binary values to analog outputs (voltages)
- Most devices have a digital interface – use time to encode value
- Time-varying digital signals – almost arbitrary resolution
  - pulse-code modulation (data = number or width of pulses)
  - pulse-width modulation (data = duty-cycle of pulses)
  - frequency modulation (data = rate at which pulses occur)



Pulse-width modulation

- Pulse a digital signal to get an average “analog” value
- The longer the pulse width, the higher the voltage

Pulse-width ratio = $\frac{t_{on}}{t_{period}}$



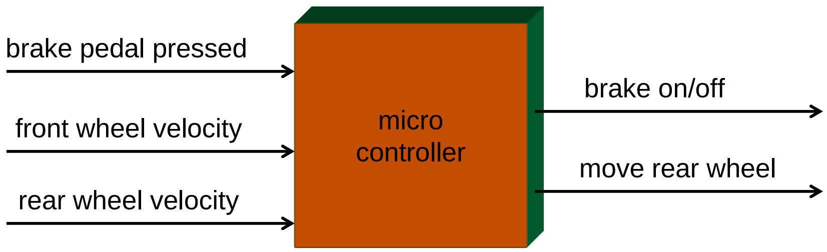
Why pulse-width modulation works

- Most mechanical systems are low-pass filters
  - Consider frequency components of pulse-width modulated signal
  - Low frequency components affect components
    - They pass through
  - High frequency components are too fast to fight inertia
    - They are “filtered out”
- Electrical RC-networks are low-pass filters
  - Time constant (τ = RC) sets “cutoff” frequency that separates low and high frequencies

Anti-lock brake system

- Rear wheel controller/anti-lock brake system
  - Normal operation
    - Regulate velocity of rear wheel
  - Brake pressed
    - Gradually increase amount of breaking
    - If skidding (front wheel is moving much faster than rear wheel) then temporarily reduce amount of breaking
- Inputs
  - Brake pedal
  - Front wheel speed
  - Rear wheel speed
- Outputs
  - Pulse-width modulation rear wheel velocity
  - Pulse-width modulation brake on/off

Rear wheel controller/anti-lock brake system



brake pedal pressed

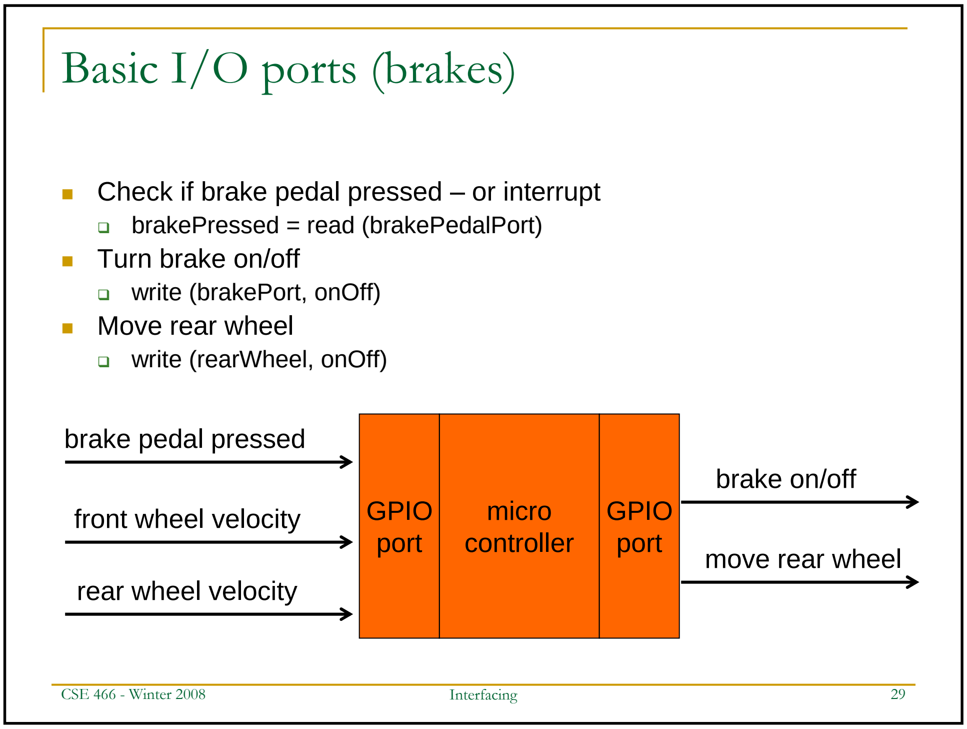
front wheel velocity

rear wheel velocity

micro controller

brake on/off

move rear wheel



Basic I/O ports (brakes)

- Check if brake pedal pressed – or interrupt
  - brakePressed = read (brakePedalPort)
- Turn brake on/off
  - write (brakePort, onOff)
- Move rear wheel
  - write (rearWheel, onOff)

brake pedal pressed

front wheel velocity

rear wheel velocity

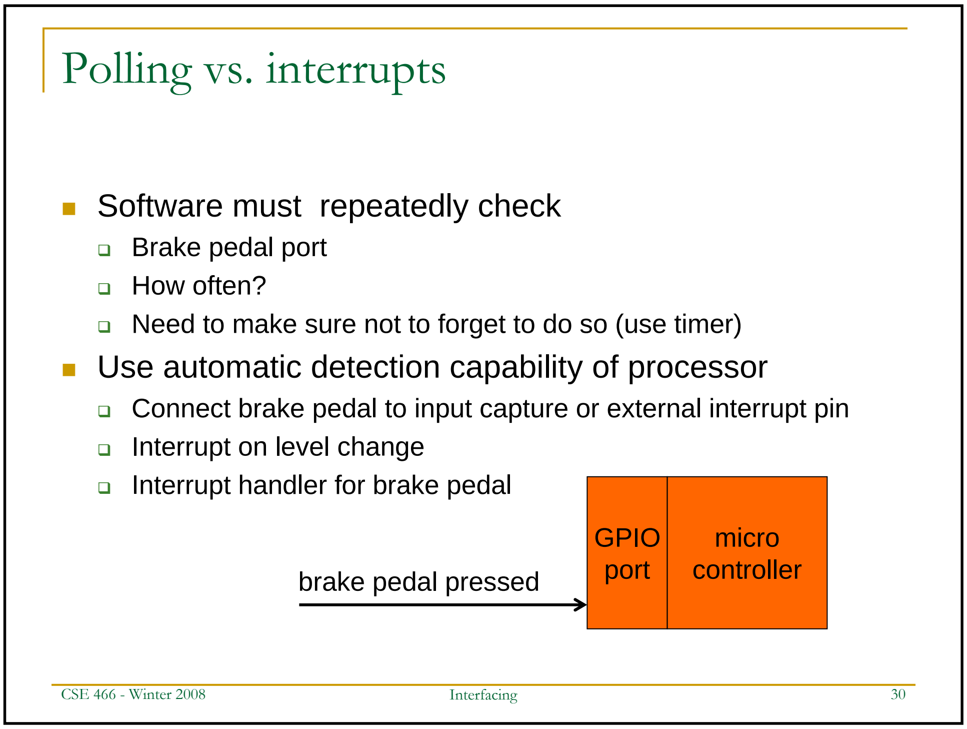
GPIO port

micro controller

GPIO port

brake on/off

move rear wheel



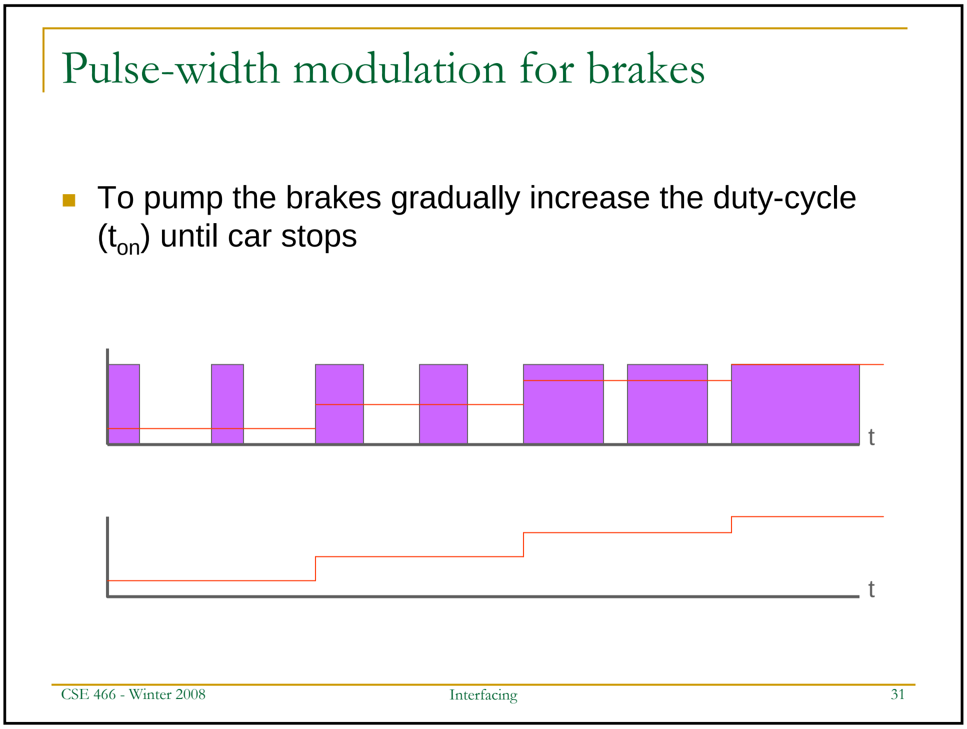
Polling vs. interrupts

- Software must repeatedly check
  - Brake pedal port
  - How often?
  - Need to make sure not to forget to do so (use timer)
- Use automatic detection capability of processor
  - Connect brake pedal to input capture or external interrupt pin
  - Interrupt on level change
  - Interrupt handler for brake pedal

brake pedal pressed

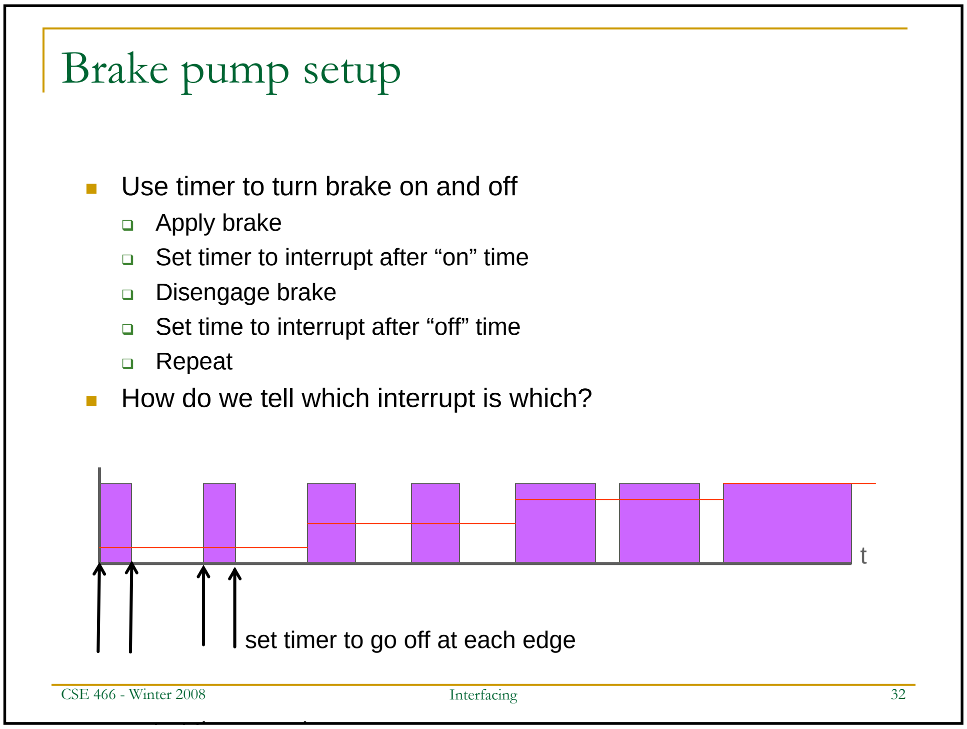
GPIO port

micro controller



Pulse-width modulation for brakes

- To pump the brakes gradually increase the duty-cycle (t_{on}) until car stops

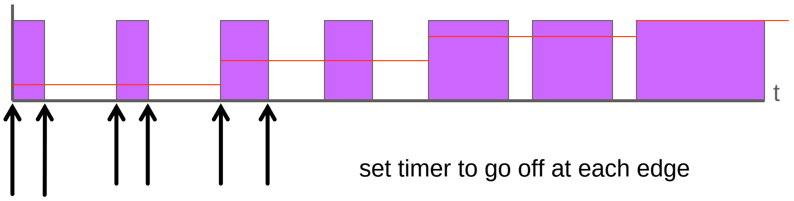


Brake pump setup

- Use timer to turn brake on and off
  - Apply brake
  - Set timer to interrupt after “on” time
  - Disengage brake
  - Set time to interrupt after “off” time
  - Repeat
- How do we tell which interrupt is which?

Brake pump setup (cont'd)

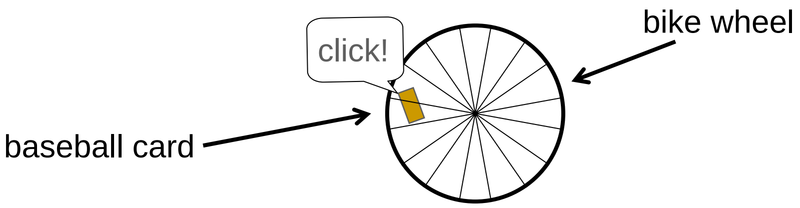
- Change value of "on" time to change analog average
  - average output = (time on) / (period)
- How do we decide on the period of the pulses?
- Using two timers
  - One to set period (auto-reload)
  - One to turn it off at the right duty cycle

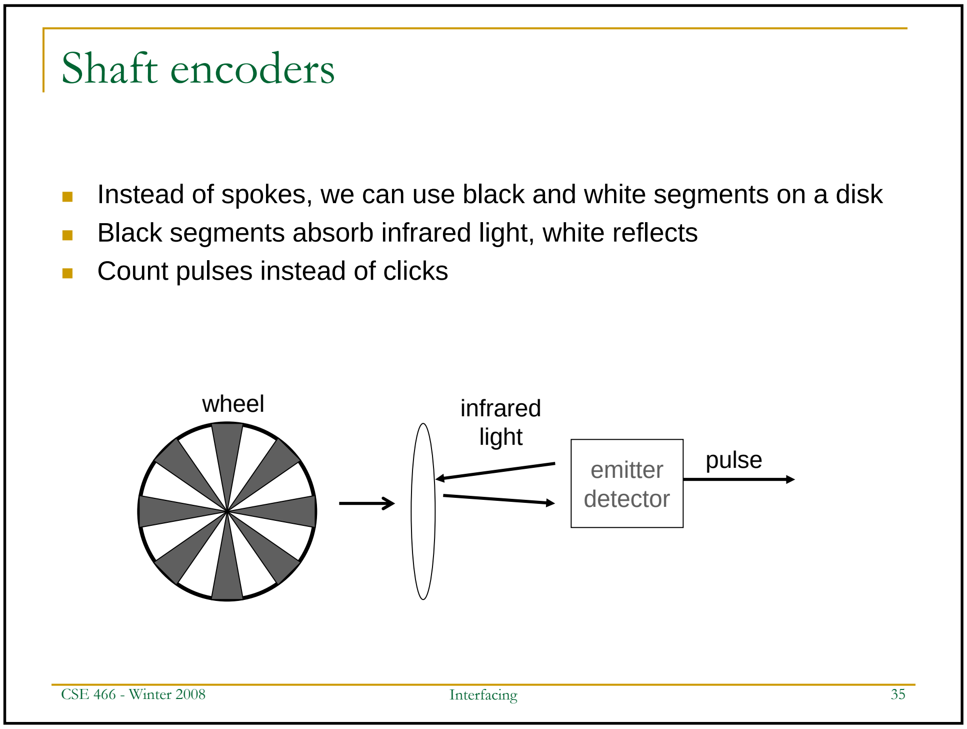


set timer to go off at each edge

Shaft encoders

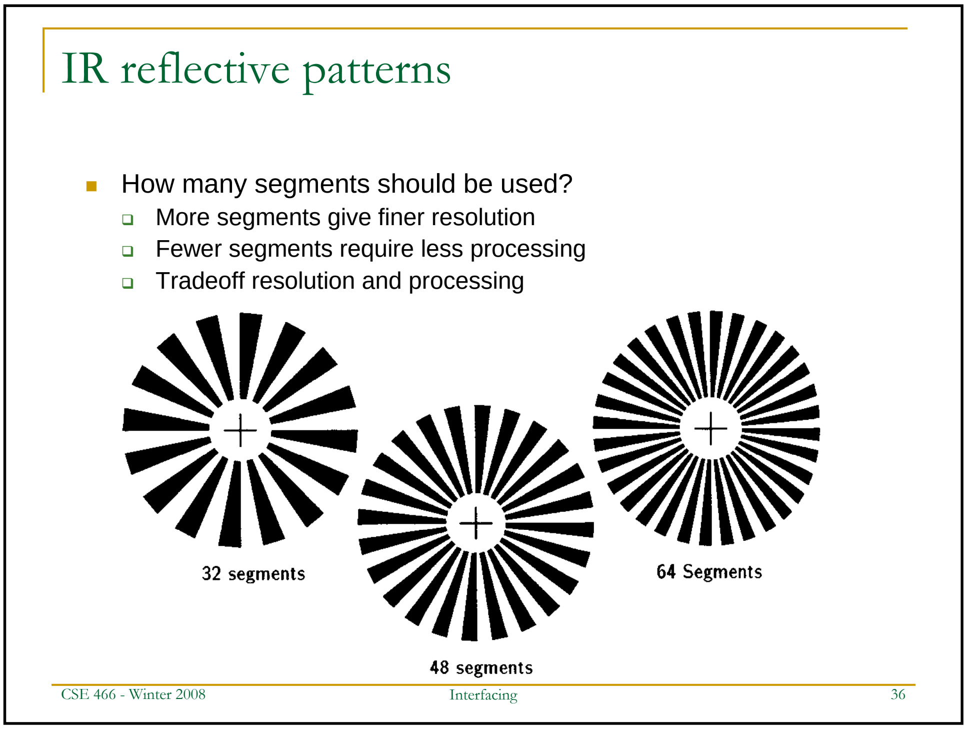
- Need to determine the rear wheel velocity
  - Use sensor to detect wheel moving
- Determine speed of a bicycle
  - Attach baseball card so it pokes through spokes
  - Number of spokes is known
  - Count clicks per unit time to get velocity
- Baseball card sensor is a shaft encoder





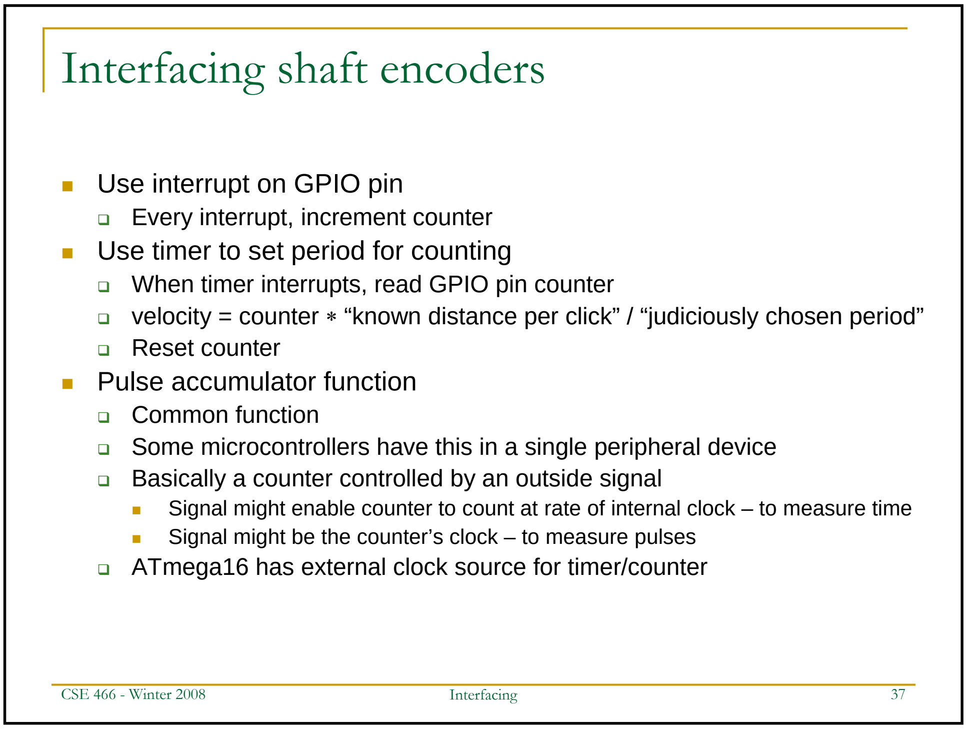
Shaft encoders

- Instead of spokes, we can use black and white segments on a disk
- Black segments absorb infrared light, white reflects
- Count pulses instead of clicks



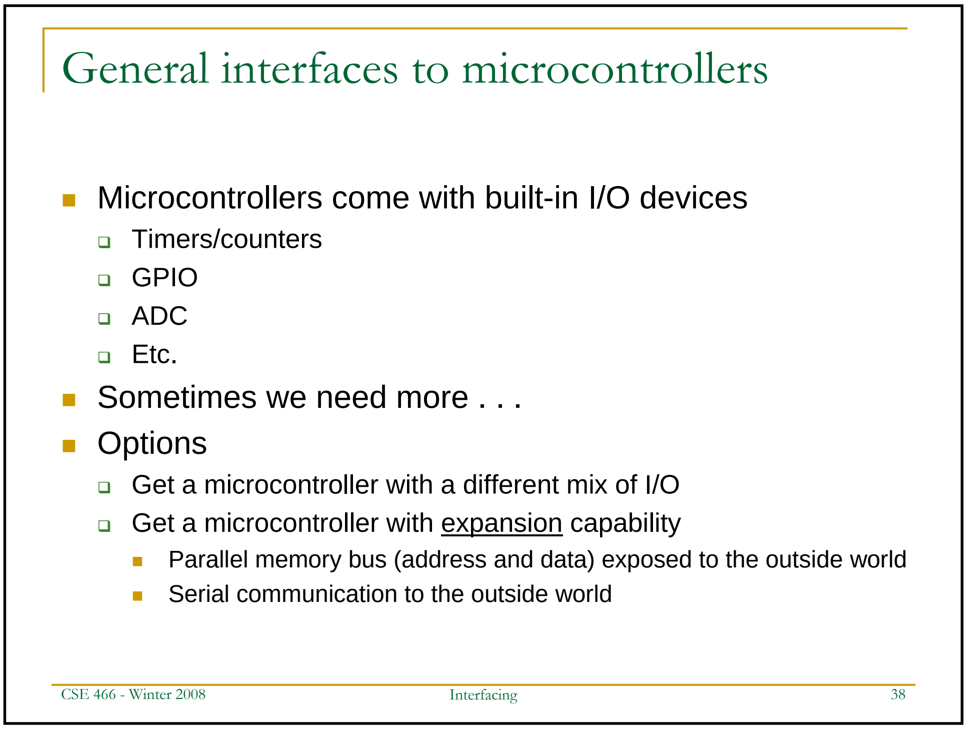
IR reflective patterns

- How many segments should be used?
  - More segments give finer resolution
  - Fewer segments require less processing
  - Tradeoff resolution and processing



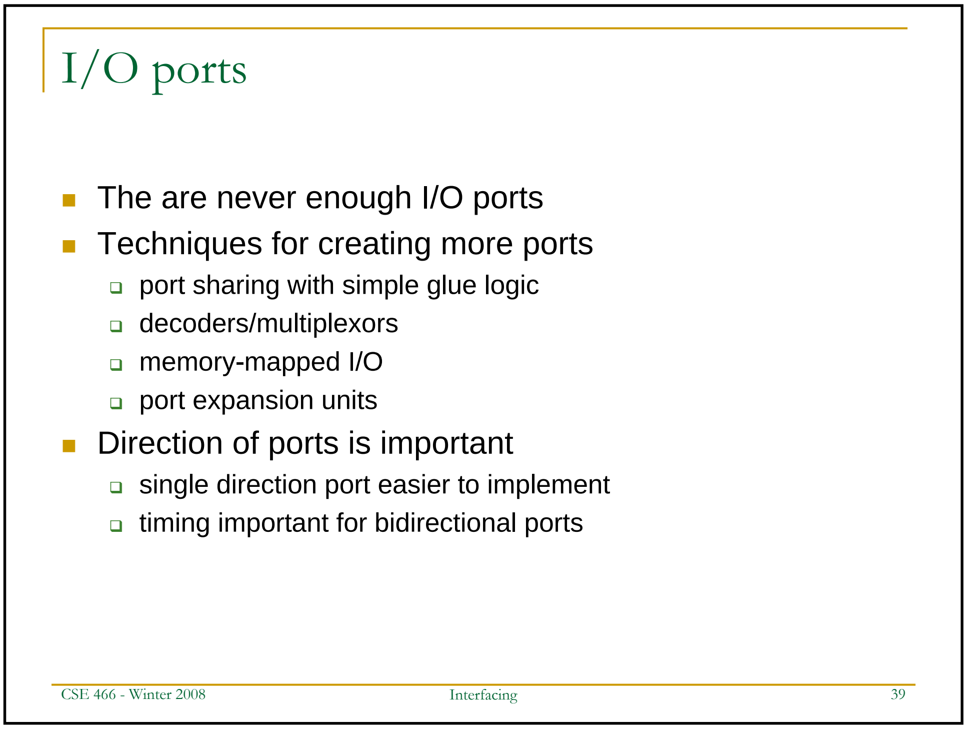
Interfacing shaft encoders

- Use interrupt on GPIO pin
  - Every interrupt, increment counter
- Use timer to set period for counting
  - When timer interrupts, read GPIO pin counter
  - velocity = counter ∗ “known distance per click” / “judiciously chosen period”
  - Reset counter
- Pulse accumulator function
  - Common function
  - Some microcontrollers have this in a single peripheral device
  - Basically a counter controlled by an outside signal
    - Signal might enable counter to count at rate of internal clock – to measure time
    - Signal might be the counter’s clock – to measure pulses
  - ATmega16 has external clock source for timer/counter



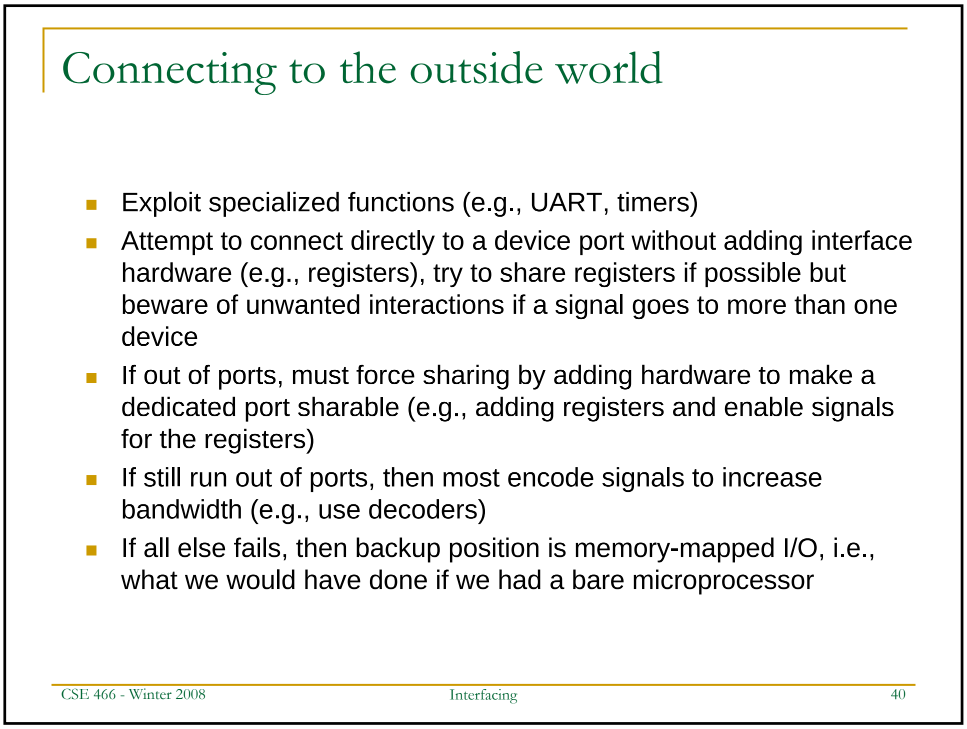
General interfaces to microcontrollers

- Microcontrollers come with built-in I/O devices
  - Timers/counters
  - GPIO
  - ADC
  - Etc.
- Sometimes we need more . . .
- Options
  - Get a microcontroller with a different mix of I/O
  - Get a microcontroller with expansion capability
    - Parallel memory bus (address and data) exposed to the outside world
    - Serial communication to the outside world



I/O ports

- The are never enough I/O ports
- Techniques for creating more ports
  - port sharing with simple glue logic
  - decoders/multiplexors
  - memory-mapped I/O
  - port expansion units
- Direction of ports is important
  - single direction port easier to implement
  - timing important for bidirectional ports

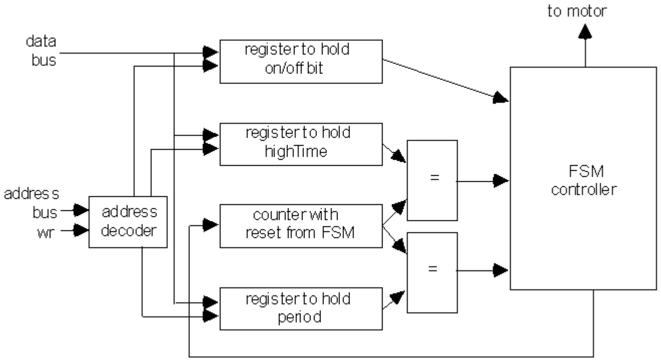


Connecting to the outside world

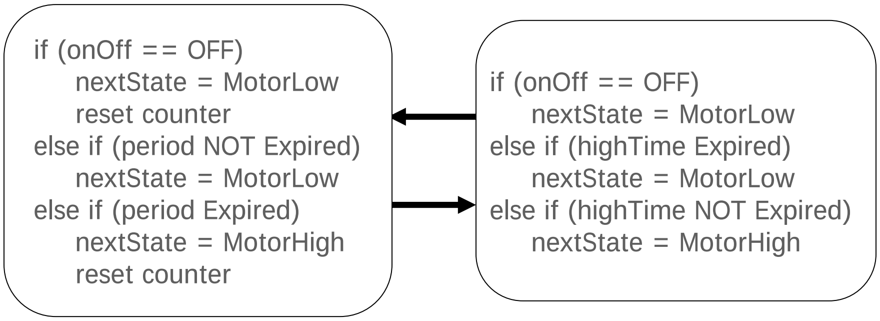
- Exploit specialized functions (e.g., UART, timers)
- Attempt to connect directly to a device port without adding interface hardware (e.g., registers), try to share registers if possible but beware of unwanted interactions if a signal goes to more than one device
- If out of ports, must force sharing by adding hardware to make a dedicated port sharable (e.g., adding registers and enable signals for the registers)
- If still run out of ports, then most encode signals to increase bandwidth (e.g., use decoders)
- If all else fails, then backup position is memory-mapped I/O, i.e., what we would have done if we had a bare microprocessor

External PWM Unit

- Design a system to control the speed of a motor with a digital value
- Solution: design a PWM unit



External PWM FSM Controller

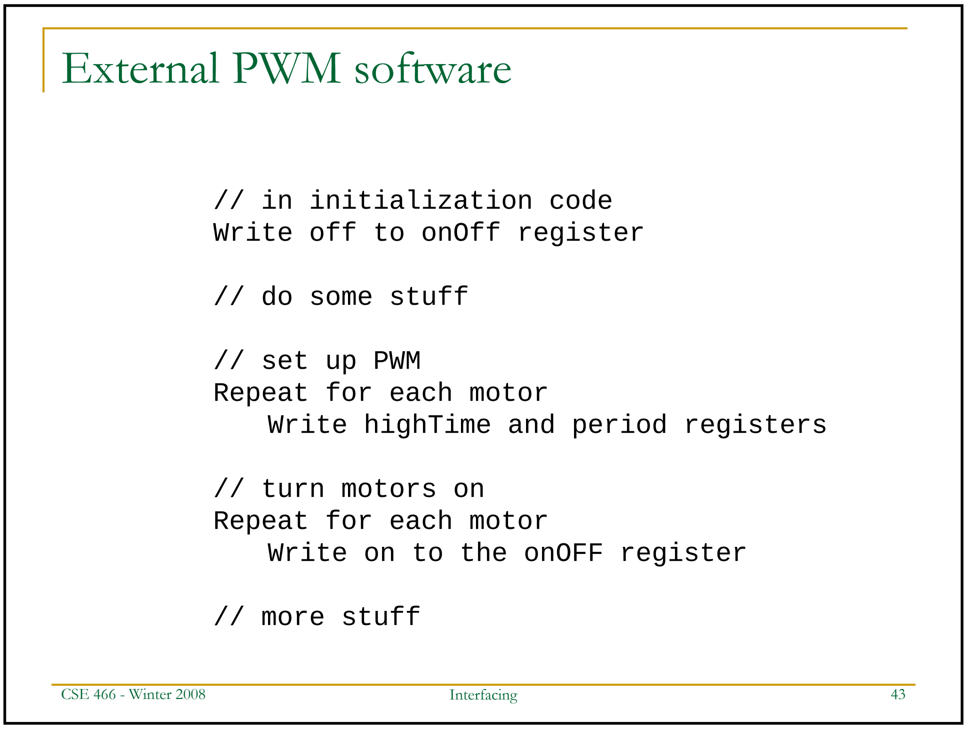


```
if (onOff == OFF)
    nextState = MotorLow
    reset counter
else if (period NOT Expired)
    nextState = MotorLow
else if (period Expired)
    nextState = MotorHigh
    reset counter
```

```
if (onOff == OFF)
    nextState = MotorLow
else if (highTime Expired)
    nextState = MotorLow
else if (highTime NOT Expired)
    nextState = MotorHigh
```

Motor Low State

Motor High State



External PWM software

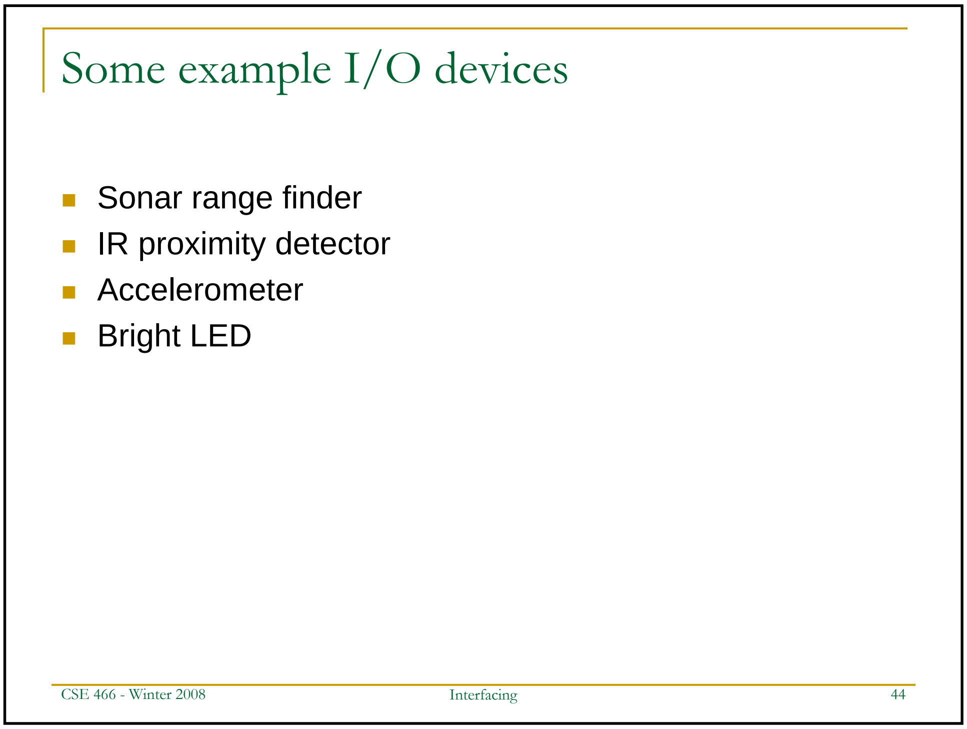
```
// in initialization code
Write off to onOff register

// do some stuff

// set up PWM
Repeat for each motor
   Write highTime and period registers

// turn motors on
Repeat for each motor
   Write on to the onOFF register

// more stuff
```



Some example I/O devices

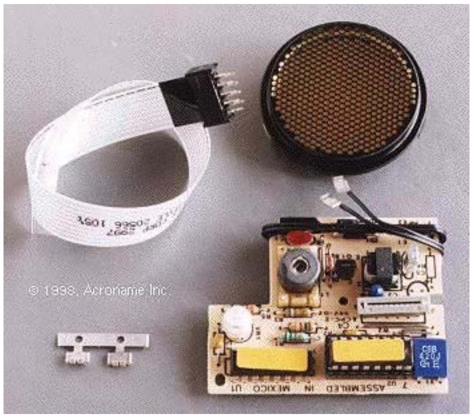
- Sonar range finder
- IR proximity detector
- Accelerometer
- Bright LED

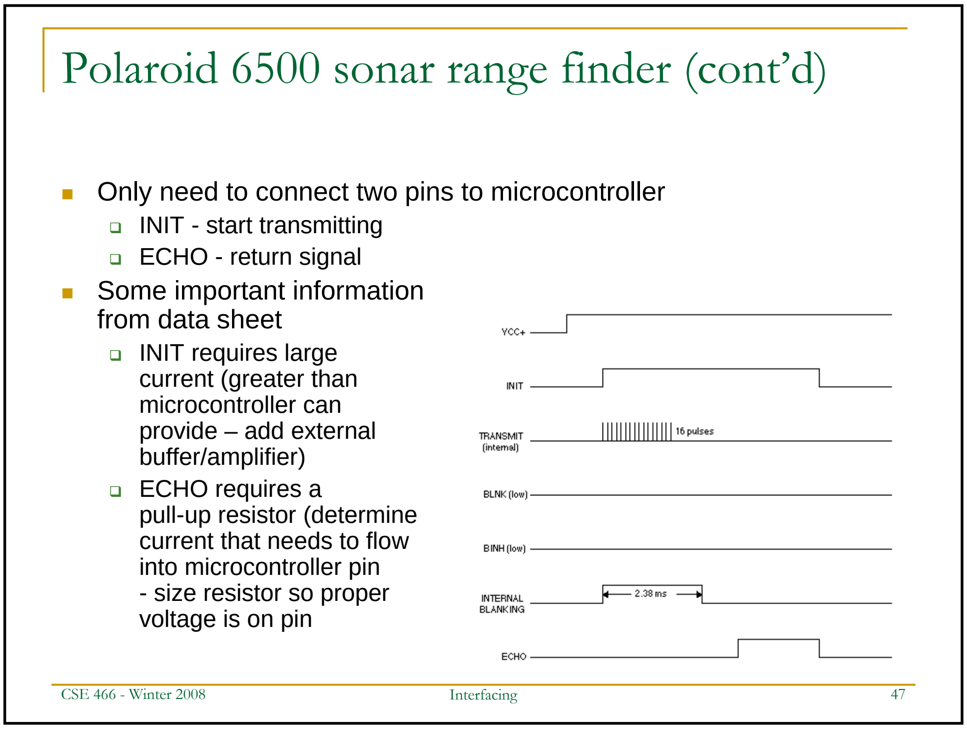
Sonar range finder

- Uses ultra-sound (not audible) to measure distance
- Time echo return
- Sound travels at approximately 343m/sec
  - need at least a 34.3kHz timer for cm resolution
- One simple echo not enough
  - many possible reflections
  - want to take multiple readings for high accuracy

Polaroid 6500 sonar range finder

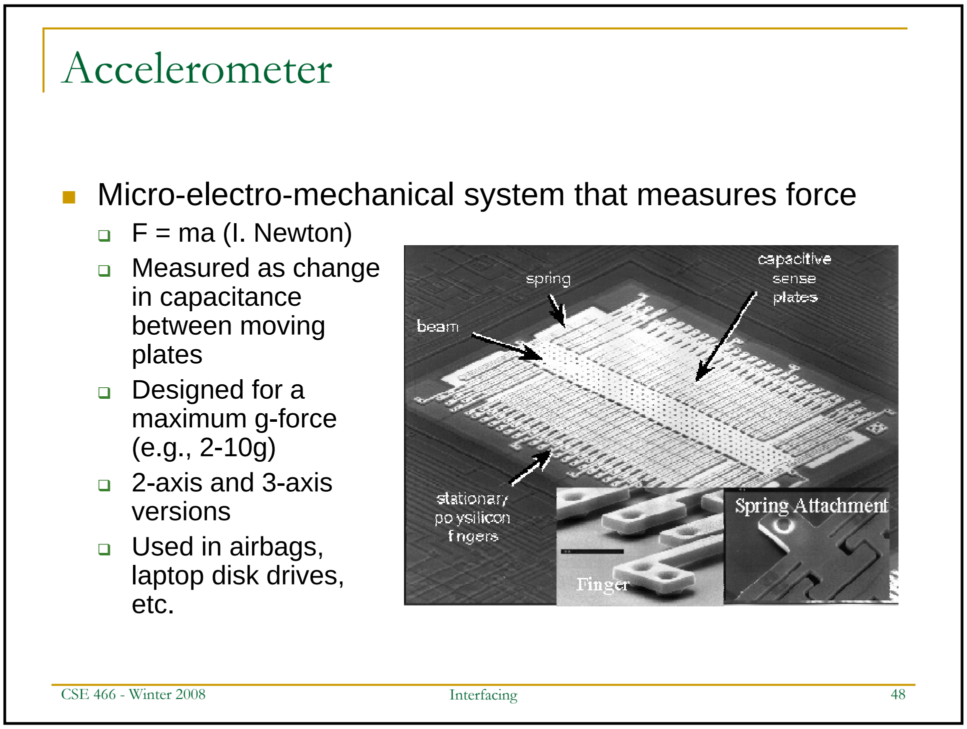
- Commonly found on old Polaroid cameras, now a frequently used part in mobile robots
- Transducer (gold disc)
  - charged up to high voltage and “snapped”
  - disc stays sentisized so it can detect echo (acts as microphone)
- Controller board
  - high-voltage circuitry to prepare disc for transmitting and then receiving





Polaroid 6500 sonar range finder (cont'd)

- Only need to connect two pins to microcontroller
  - INIT - start transmitting
  - ECHO - return signal
- Some important information from data sheet
  - INIT requires large current (greater than microcontroller can provide – add external buffer/amplifier)
  - ECHO requires a pull-up resistor (determine current that needs to flow into microcontroller pin - size resistor so proper voltage is on pin



Accelerometer

- Micro-electro-mechanical system that measures force
  - F = ma (I. Newton)
  - Measured as change in capacitance between moving plates
  - Designed for a maximum g-force (e.g., 2-10g)
  - 2-axis and 3-axis versions
  - Used in airbags, laptop disk drives, etc.

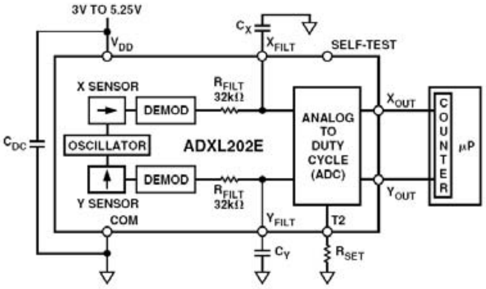
spring
capacitive sense plates
beam
stationary polysilicon fingers
Spring Attachment
Finger

Accelerometer output

- Analog output too susceptible to noise
- Digital output requires many pins for precision
  - Could use serial interface
- Use pulse-width modulation
- What about gravity?

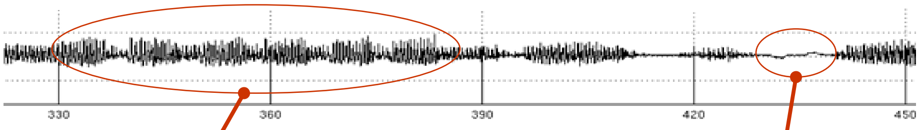
Analog Devices ADXL202

- 2-axis accelerometer
  - Set 0g at 50% duty-cycle
  - Positive acceleration increases duty cycle
  - Negative acceleration decreases duty cycle
  - 12.5% per g in either direction



Typical measurement for ADXL202

- Noisy data – all forces are aggregated by accelerometer
- Sample trace at 250Hz

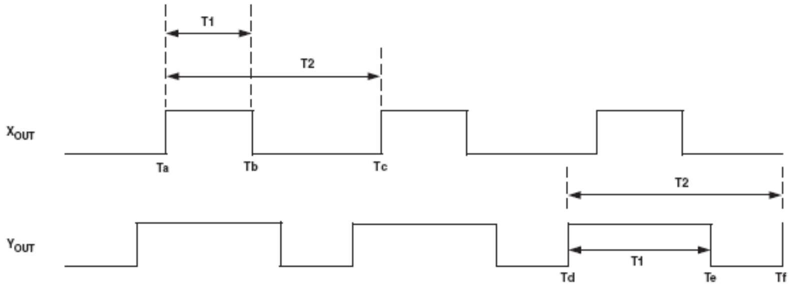


Walking down six
flights of stairs

Elevator ride

Typical signal from ADXL202

- Cause interrupts at Ta, Tb, and Tc from X-axis output
- 1. Look for rising edge, reset counter: Ta = 0
- 2. Look for falling edge, record timer: Tb = positive duty cycle
- 3. Look for rising edge, record timer, reset counter: Tc = period
- Repeat from 2
- Same for Y-axis output (T2 is the same for both axes)

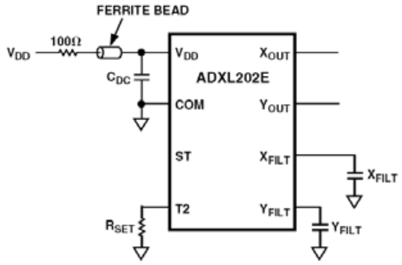


What to do about noise/jitter?

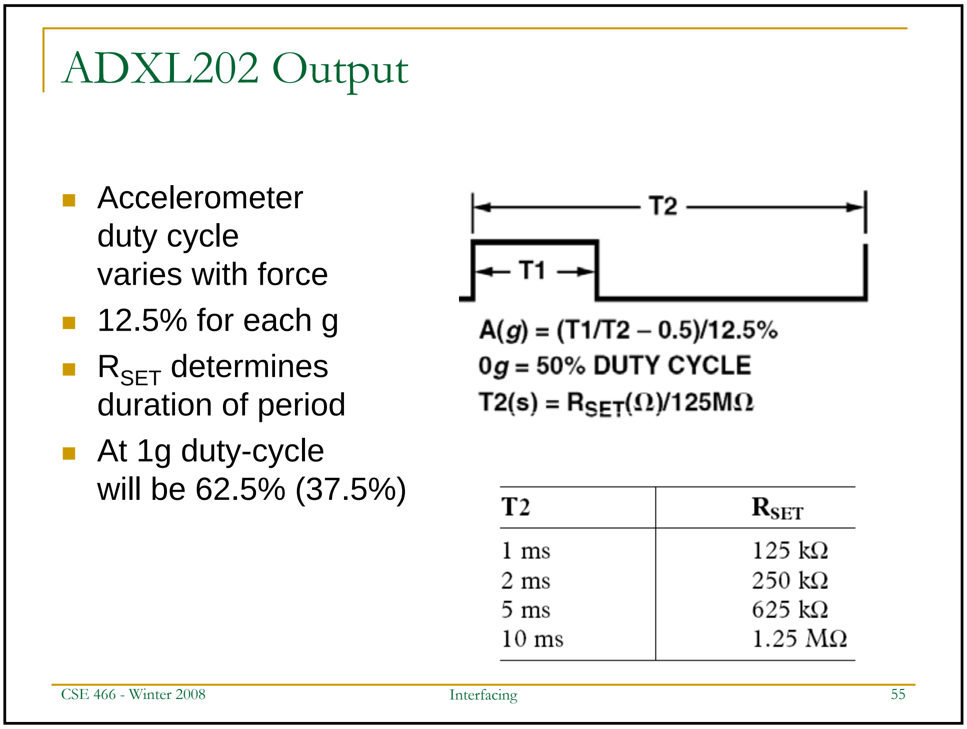
- Average over time – smoothing
  - Software filter – like switch debouncing
- Take several readings
  - use average for Tb and Tc or their ratio
- Running average so that a reading is available at all times
  - e.g., update running average of 8 readings
    current average = ⅞ * current average + ⅛ * new reading
- Take readings of both Tb and Tc to be extra careful
  - Tc changes with temperature
  - Usually can do Tc just once

Built-in filter

- Filter capacitors limit noise
  - bandwidth limiting – eliminate high-frequency noise



Bandwidth	Capacitor Value
10 Hz	0.47 µF
50 Hz	0.10 µF
100 Hz	0.05 µF
200 Hz	0.027 µF
500 Hz	0.01 µF
5 kHz	0.001 µF



ADXL202 Output

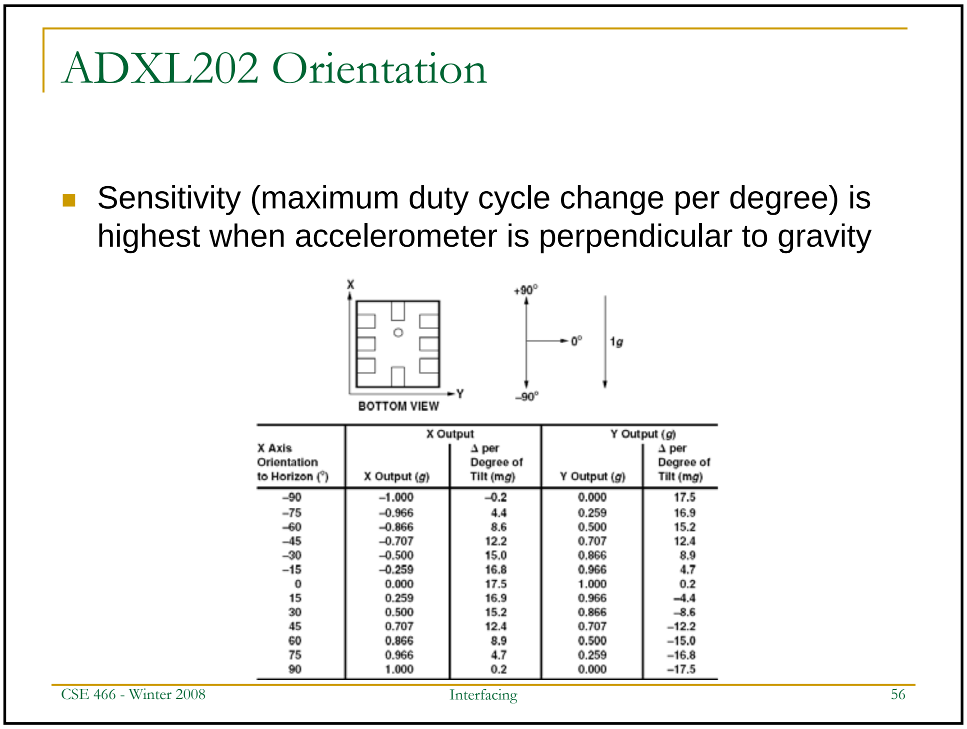
- Accelerometer duty cycle varies with force
- 12.5% for each g
- R_{SET} determines duration of period
- At 1g duty-cycle will be 62.5% (37.5%)

$A(g) = (T1/T2 - 0.5)/12.5\%$

$0g$ = 50% DUTY CYCLE

$T2(s) = R_{SET}(\Omega)/125M\Omega$

T2	R_{SET}
1 ms	125 kΩ
2 ms	250 kΩ
5 ms	625 kΩ
10 ms	1.25 MΩ

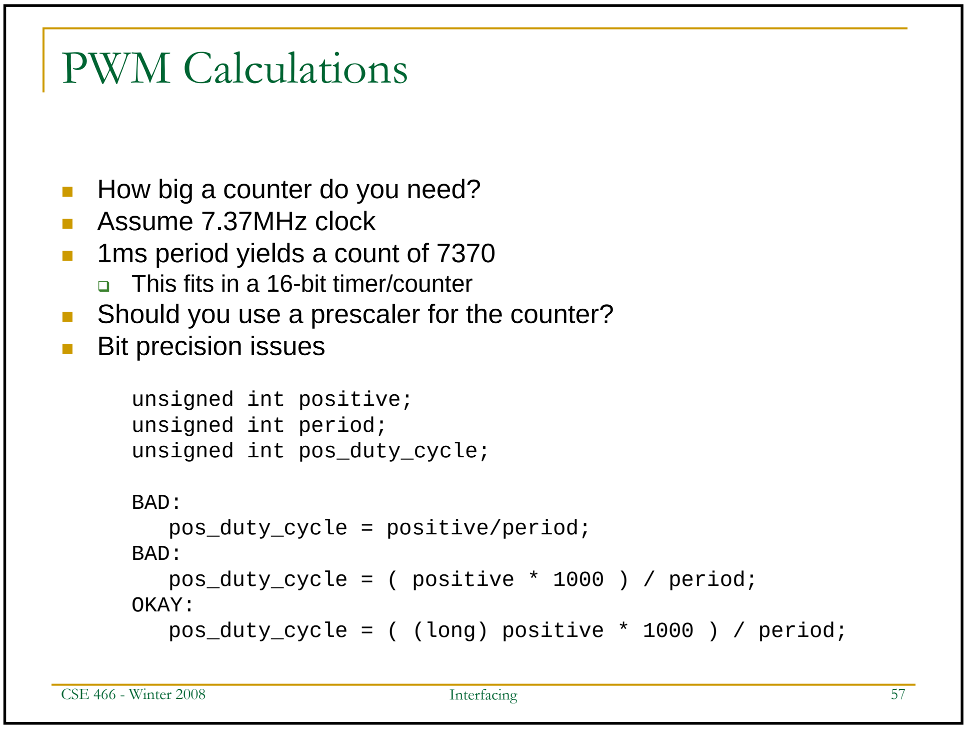


ADXL202 Orientation

- Sensitivity (maximum duty cycle change per degree) is highest when accelerometer is perpendicular to gravity

X
Y
BOTTOM VIEW
+90°
0°
–90°
1g

X Axis Orientation to Horizon (°)	X Output		Y Output (g)	
	X Output (g)	Δ per Degree of Tilt (mg)	Y Output (g)	Δ per Degree of Tilt (mg)
–90	–1.000	–0.2	0.000	17.5
–75	–0.966	4.4	0.259	16.9
–60	–0.866	8.6	0.500	15.2
–45	–0.707	12.2	0.707	12.4
–30	–0.500	15.0	0.866	8.9
–15	–0.259	16.8	0.966	4.7
0	0.000	17.5	1.000	0.2
15	0.259	16.9	0.966	–4.4
30	0.500	15.2	0.866	–8.6
45	0.707	12.4	0.707	–12.2
60	0.866	8.9	0.500	–15.0
75	0.966	4.7	0.259	–16.8
90	1.000	0.2	0.000	–17.5

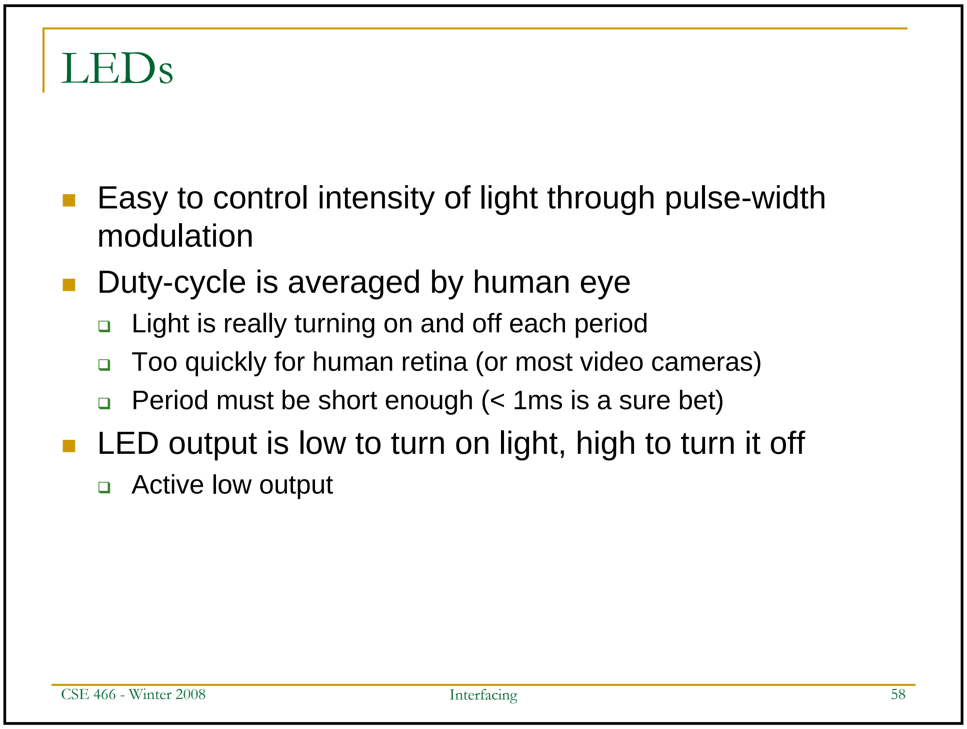


PWM Calculations

- How big a counter do you need?
- Assume 7.37MHz clock
- 1ms period yields a count of 7370
  - This fits in a 16-bit timer/counter
- Should you use a prescaler for the counter?
- Bit precision issues

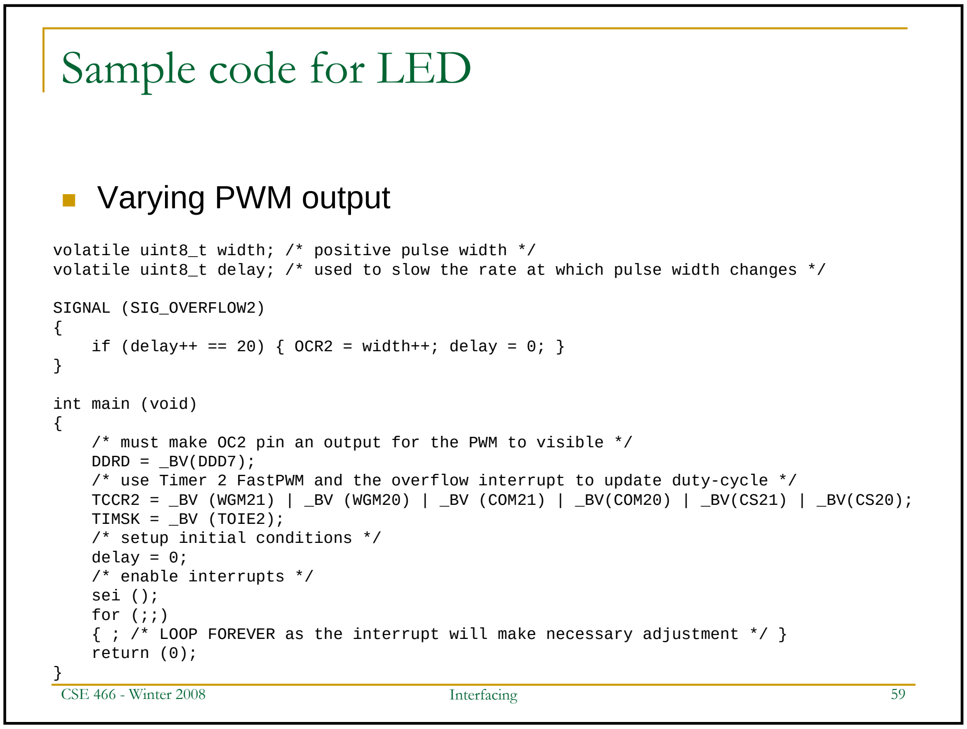
```
unsigned int positive;
unsigned int period;
unsigned int pos_duty_cycle;

BAD:
   pos_duty_cycle = positive/period;
BAD:
   pos_duty_cycle = ( positive * 1000 ) / period;
OKAY:
   pos_duty_cycle = ( (long) positive * 1000 ) / period;
```



LEDs

- Easy to control intensity of light through pulse-width modulation
- Duty-cycle is averaged by human eye
  - Light is really turning on and off each period
  - Too quickly for human retina (or most video cameras)
  - Period must be short enough (< 1ms is a sure bet)
- LED output is low to turn on light, high to turn it off
  - Active low output



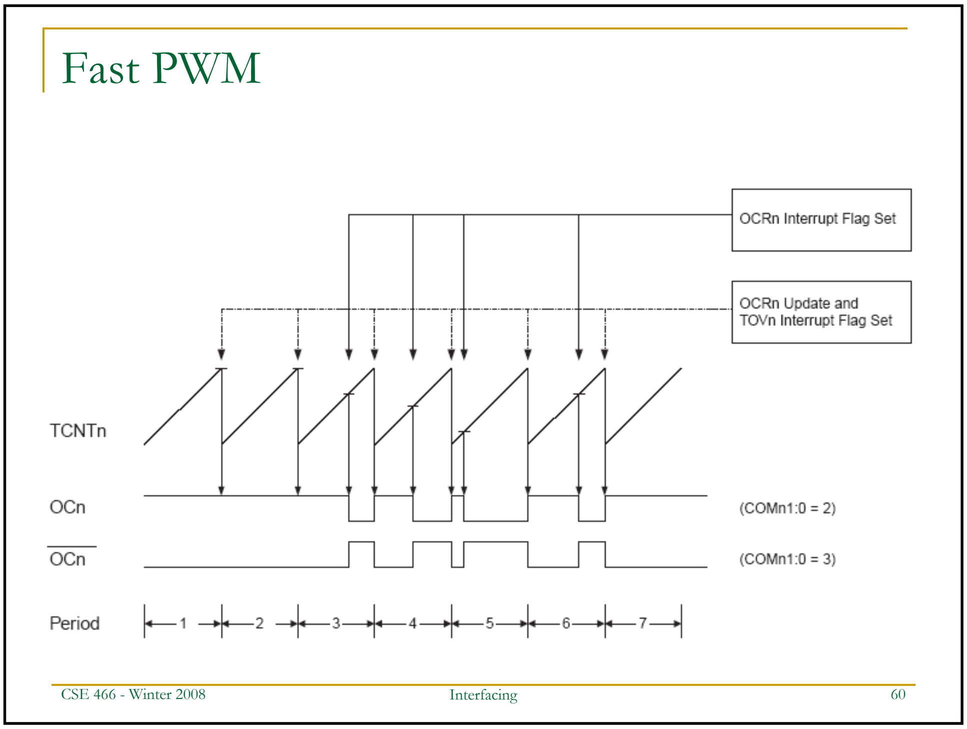
Sample code for LED

- Varying PWM output

```
volatile uint8_t width; /* positive pulse width */
volatile uint8_t delay; /* used to slow the rate at which pulse width changes */

SIGNAL (SIG_OVERFLOW2)
{
    if (delay++ == 20) { OCR2 = width++; delay = 0; }
}

int main (void)
{
    /* must make OC2 pin an output for the PWM to visible */
    DDRD = _BV(DDD7);
    /* use Timer 2 FastPWM and the overflow interrupt to update duty-cycle */
    TCCR2 = _BV (WGM21) | _BV (WGM20) | _BV (COM21) | _BV(COM20) | _BV(CS21) | _BV(CS20);
    TIMSK = _BV (TOIE2);
    /* setup initial conditions */
    delay = 0;
    /* enable interrupts */
    sei ();
    for (;;)
    { ; /* LOOP FOREVER as the interrupt will make necessary adjustment */ }
    return (0);
}
```



Fast PWM

Lab 3

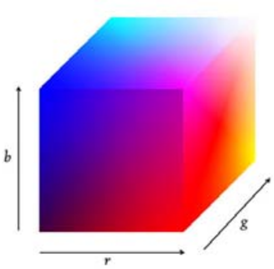
- Use accelerometer to set RGB-LED to a color
- Vary intensity using a potentiometer
- Think of it as a mouse with an enabling button
  - Tilt the mouse to move in color space – color in X, Y
  - Turn potentiometer (pot) to adjust brightness

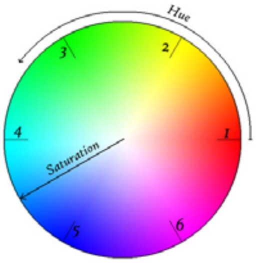
Color

- Color perception usually involves three quantities:
  - Hue: Distinguishes between colors like red, green, blue, etc
  - Saturation: How far the color is from a gray of equal intensity
  - Lightness: The perceived intensity of a reflecting object
- Sometimes lightness is called brightness if the object is emitting light instead of reflecting it.
- In order to use color precisely in computer graphics, we need to be able to specify and measure colors.

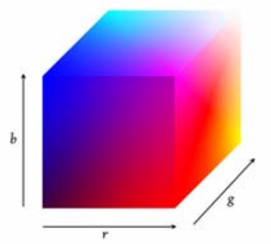
Numerous Color Spaces

- RGB, CMY, XYZ; HSV, HLS; Lab, UVW, YUV, YCrCb, Luv, L* u* v*, ..
- Different Purposes: display, editing, computation, compression, ..
- Equally distant colors may not be equally perceivable
- Separation of luminance and chromaticity (YIQ)

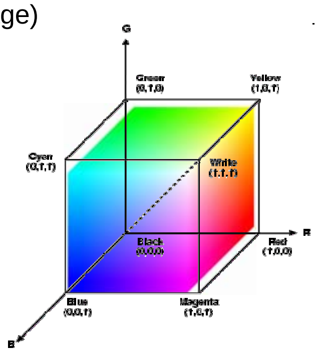




Additive Model (RGB System)



- R, G, B normalized on orthogonal axes
- All representable colors inside the unit cube
- Color monitors mix R, G and B
- Video cameras pick up R, G and B
- CIE (Commission Internationale de l'Eclairage) standardized this system in 1931
  - B: 435.8 nm, G: 546.1 nm, R: 700 nm.
- 3 fixed components acting alone can't generate all spectrum colors.

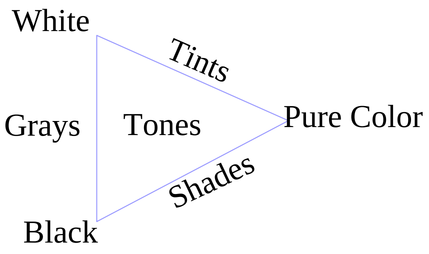


Problems with RGB

- Only a small range of potential perceivable colors (particularly for monitor RGB)
- It isn’t easy for humans to say how much of RGB to use to get a given color
  - How much R, G, and B is there in “brown”?
- Perceptually non-linear

How Do Artists Do It?

- Artists often specify color as tints, shades, and tones of saturated (pure) pigments
- Tint: determined by adding white to a pure pigment, thereby decreasing saturation
- Shade: determined by adding black to a pure pigment, thereby decreasing lightness
- Tone: determined by adding white and black to a pure pigment

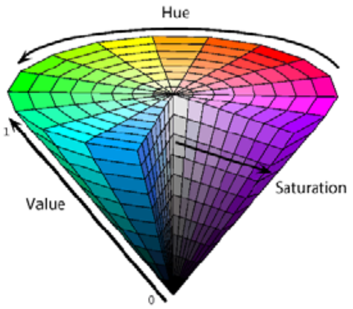


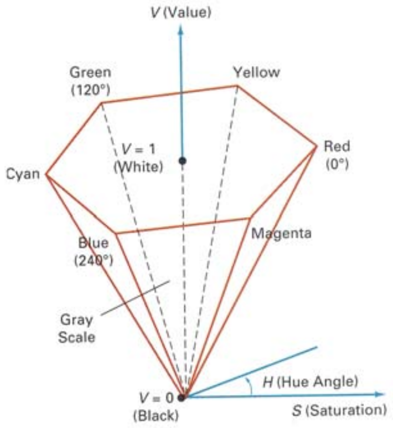
HSV Color Space

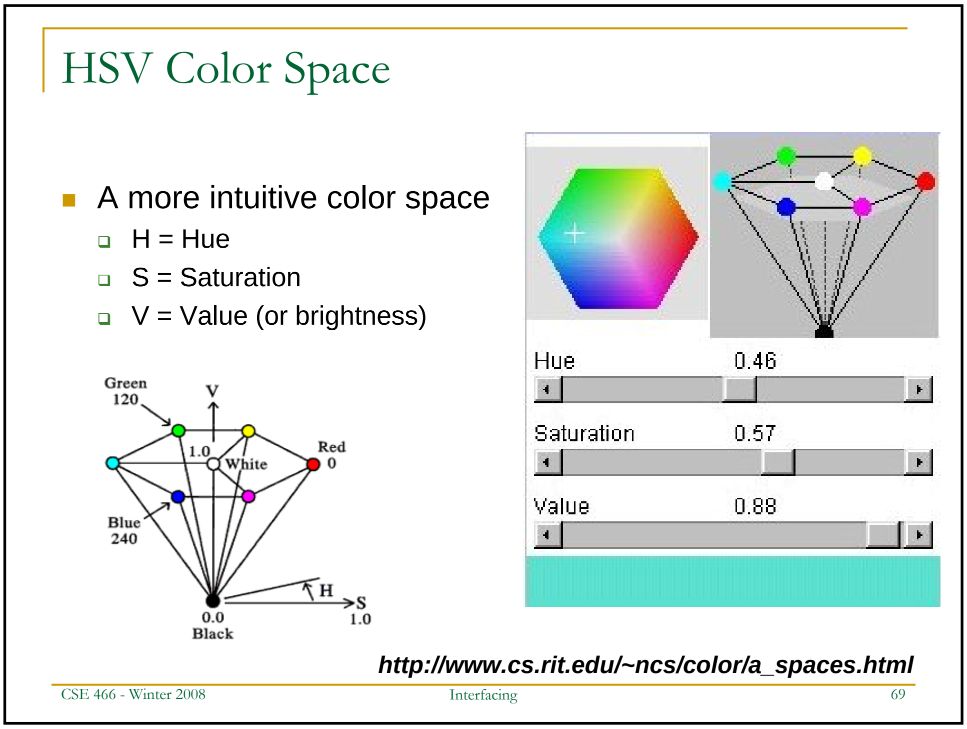
- Computer scientists frequently use an intuitive color space that corresponds to tint, shade, and tone:
  - Hue - The color we see (red, green, purple)
  - Saturation - How far is the color from gray (pink is less saturated than red, sky blue is less saturated than royal blue)
  - Brightness (Luminance) - How bright is the color (how bright are the lights illuminating the object?)

HSV Color space

- H and S are polar coordinates
  - H is angle (0 to 2π radians)
  - S is distance along radial (0 to 1)
- V is height (0 to 1)







HSV Color Space

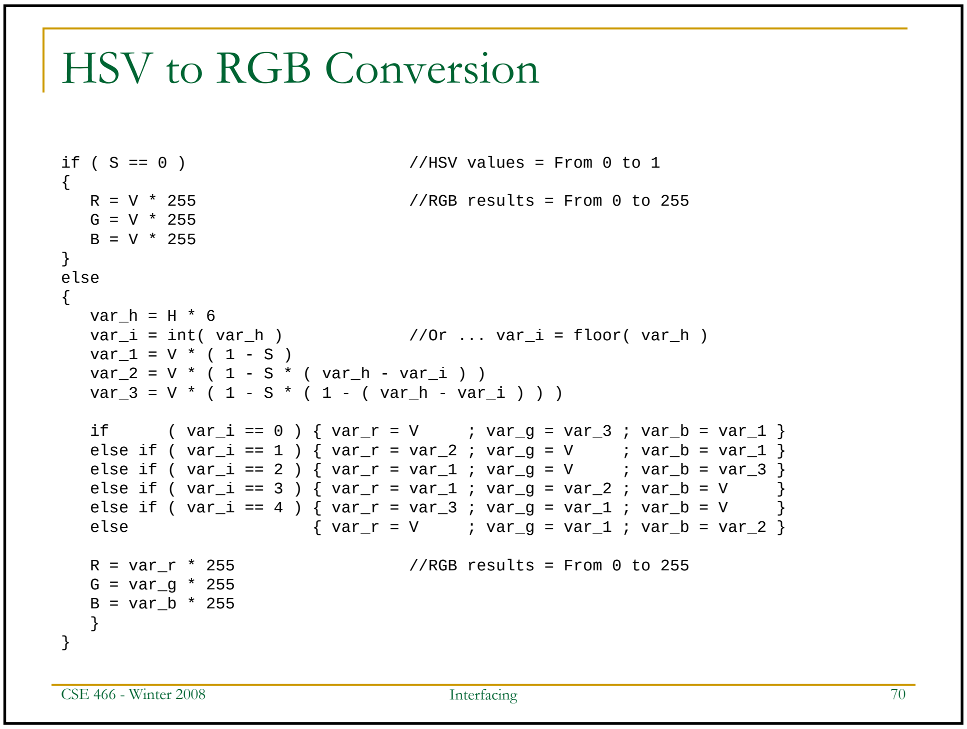
- A more intuitive color space
  - H = Hue
  - S = Saturation
  - V = Value (or brightness)

Hue 0.46

Saturation 0.57

Value 0.88

http://www.cs.rit.edu/~ncs/color/a_spaces.html



HSV to RGB Conversion

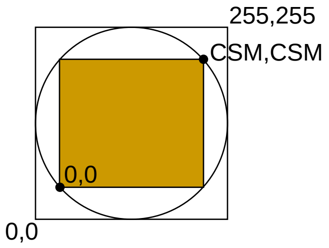
```
if ( S == 0 )                       //HSV values = From 0 to 1
{
   R = V * 255                      //RGB results = From 0 to 255
   G = V * 255
   B = V * 255
}
else
{
   var_h = H * 6
   var_i = int( var_h )             //Or ... var_i = floor( var_h )
   var_1 = V * ( 1 - S )
   var_2 = V * ( 1 - S * ( var_h - var_i ) )
   var_3 = V * ( 1 - S * ( 1 - ( var_h - var_i ) ) )

   if      ( var_i == 0 ) { var_r = V     ; var_g = var_3 ; var_b = var_1 }
   else if ( var_i == 1 ) { var_r = var_2 ; var_g = V     ; var_b = var_1 }
   else if ( var_i == 2 ) { var_r = var_1 ; var_g = V     ; var_b = var_3 }
   else if ( var_i == 3 ) { var_r = var_1 ; var_g = var_2 ; var_b = V     }
   else if ( var_i == 4 ) { var_r = var_3 ; var_g = var_1 ; var_b = V     }
   else                   { var_r = V     ; var_g = var_1 ; var_b = var_2 }

   R = var_r * 255                  //RGB results = From 0 to 255
   G = var_g * 255
   B = var_b * 255
   }
}
```

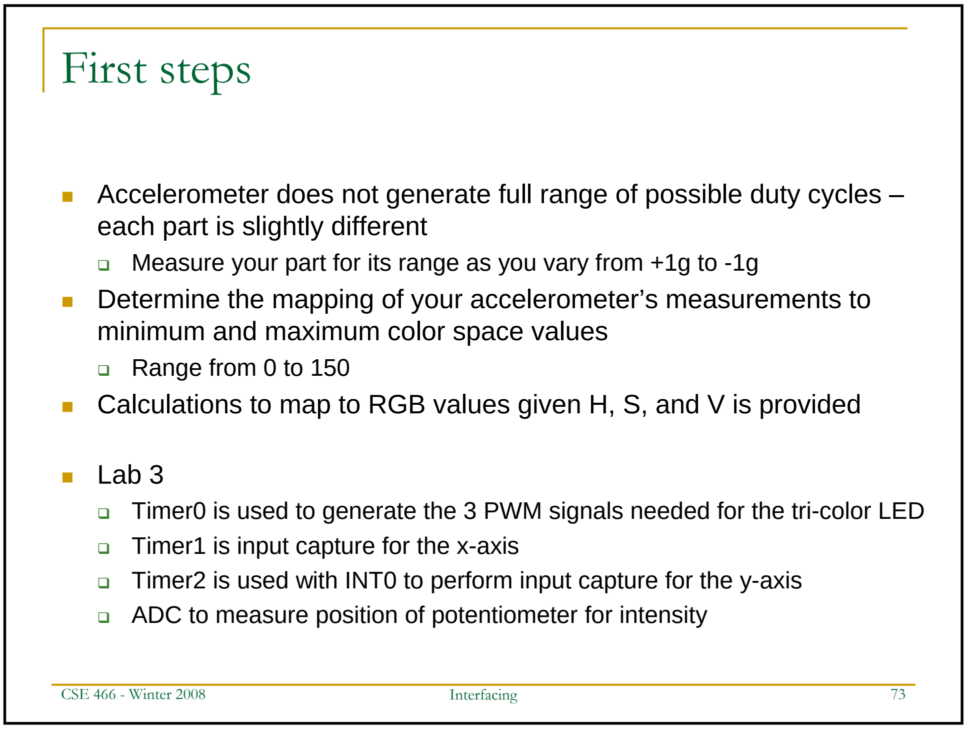
Our version

- HSV scale goes from 0 to COLOR_SPACE_MAX for H and S, and 0 to 255 for V
- Issue:
  - Full square of H, S doesn't translate to cone
    - Can't have 0,0 or 255,255
  - We use a smaller square
    - Clip some colors to that square



A Series of Translations

- Accelerometer
  - Provides PWM signal
- Measure duty-cycle using microcontroller
  - % of period PWM signal is high
- Map this to a color space
  - We'll use two dimensions of HSV space
    (H – hue) and (S – saturation) and
    leave the intensity (V – value)
    to be adjusted by a potentiometer
- Translate color values to PWM signals to control tricolor-LED
  - HSV becomes 3 separate duty-cycle %ages for RGB
- Generates these signals using timers of microcontroller
  - Translate to a period and counter value for corresponding duty-cycle
  - PWM tri-color LED reproduces color selected with accelerometer



First steps

- Accelerometer does not generate full range of possible duty cycles – each part is slightly different
  - Measure your part for its range as you vary from +1g to -1g
- Determine the mapping of your accelerometer’s measurements to minimum and maximum color space values
  - Range from 0 to 150
- Calculations to map to RGB values given H, S, and V is provided

- Lab 3
  - Timer0 is used to generate the 3 PWM signals needed for the tri-color LED
  - Timer1 is input capture for the x-axis
  - Timer2 is used with INT0 to perform input capture for the y-axis
  - ADC to measure position of potentiometer for intensity