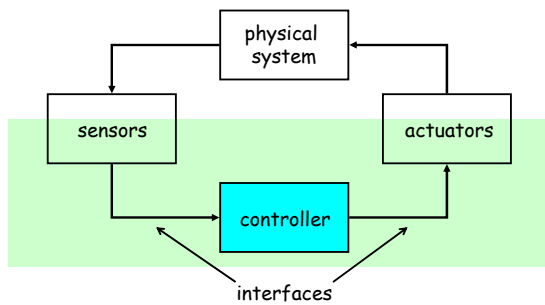## Interfacing

- Connecting the computation 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?

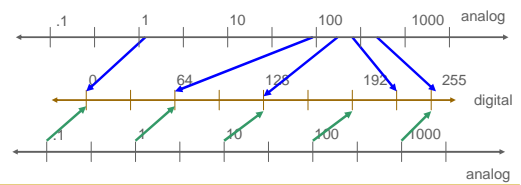## Controlling and reacting to the environment

- 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
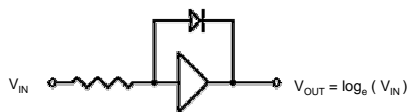
Typical control system

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
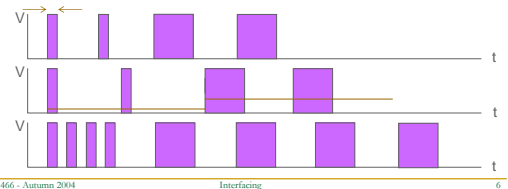 - larger range of outputs (16-bit a/d)

Logarithm of a signal

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

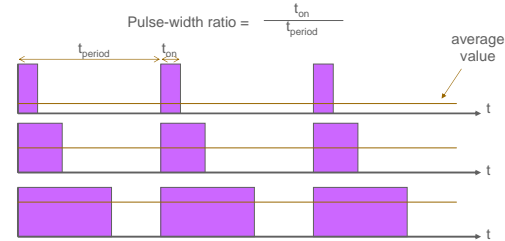$$V_{IN} \qquad V_{OUT} = \log_e (V_{IN})$$

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)

1

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 = $\dfrac{t_{on}}{t_{period}}$

t_{period} t_{on}

average value

t

t

t

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
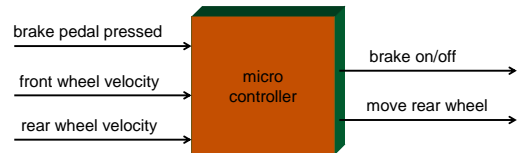
Anti-lock brake system

- 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
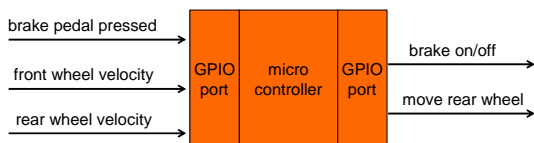
Basic I/O ports (brakes)

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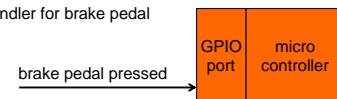- Move rear wheel
  - write (rearWheel, onOff)

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
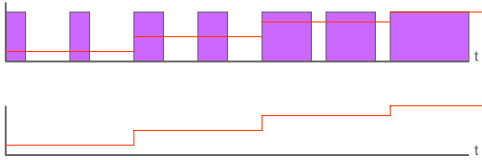 - Interrupt handler for brake pedal

brake pedal pressed

GPIO port   micro controller

2

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
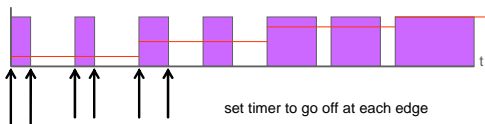- How do we tell which interrupt is which?

set timer to go off at each edge

Brake pump setup (cont'd)

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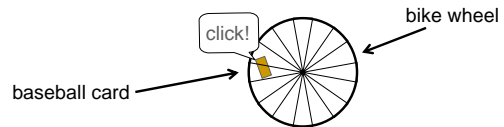 - One to turn it off at the right duty cycle

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
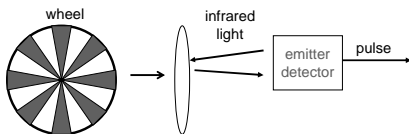- Baseball card sensor is a shaft encoder

click!

bike wheel

baseball card

Shaft encoders

- Instead of spokes we'll use black and white segments
- Black segments absorb infrared light, white reflects
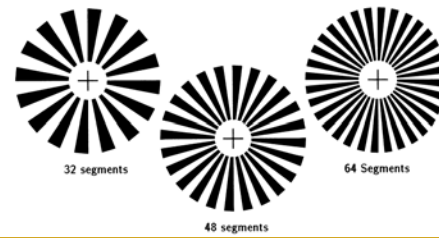- Count pulses instead of clicks

wheel

infrared light

emitter detector

pulse

IR reflective patterns

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

32 segments

64 Segments

48 segments

3

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

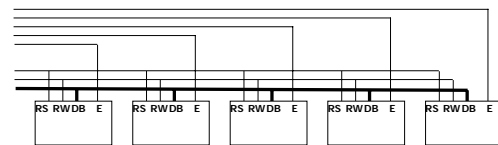## General interfaces to microcontrollers

- 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 <u>expansion</u> 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
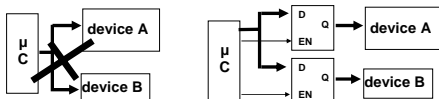 - timing important for bidirectional ports

Port sharing

- If signals all in same direction and have a separate guard signal, then able to share without glue logic
- Example: connect 5 LCD displays to microcontroller
  - can share connections to RS, RW, and DB but not E
  - changes on E affect display – must guarantee only one is active
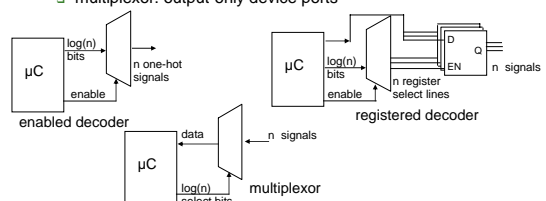
Forced sharing

- Conflict on device signals (e.g., one signal affects both)
  - solution is to insert intervening registers that keep signals stable
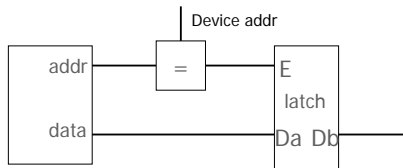 - registers require enable signals which now need ports as well

Decoders and multiplexors

- Encode n single-bit device ports using log n bits of a controller port
  - enabled decoder: one-hot, input-only device ports
  - registered decoder: input-only (but not one-hot) device ports
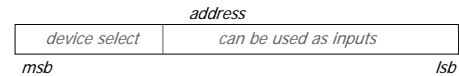 - multiplexor: output-only device ports

4

Memory-mapped I/O

- Address bus selects device
- Data bus contains data

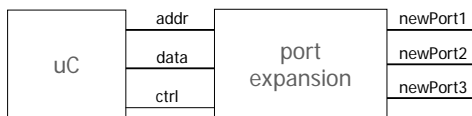Device addr

addr

=

E
latch

data

Da Db

Memory-mapped I/O

- Partition the address space
- Assign memory-mapped locations
- Software
  - loads read from the device
  - stores write to the device
- Can exploit unused bits for device input-only ports

address

device select	can be used as inputs
msb	lsb

Port expansion units

- Problem of port shortage so common port expansion chips exist
- Easily connect to the microprocessor
- Timing on ports may be slightly different
- May not support interrupts

uC

addr

data

ctrl

port
expansion

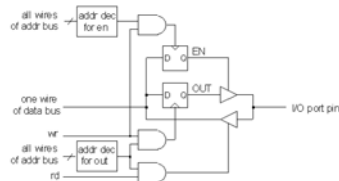newPort1

newPort2

newPort3

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

64-bit General-purpose I/O port

- Suppose we wanted a 64-bit I/O port
- If EN is true, then we have an output pin
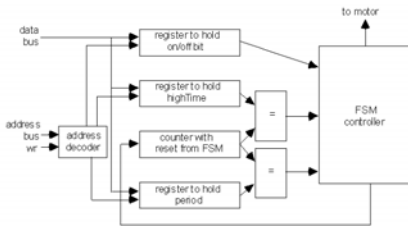- If EN is false, then we have an input pin

all wires
of addr bus

addr dec
for en

EN

D Q

one wire
of data bus

D Q

OUT

I/O port pin

wr

all wires
of addr bus

addr dec
for out

rd

64-bit I/O port software

- We need 8 8-bit registers to store/write the 64 bits
  - Select the EN addresses to be $...000 to $...007
  - Select OUT addresses to be $...010 to $...017
- Read 15th bit
  - load value at address $...011 (2nd set of OUT regs)
  - logical AND with 0x80
  - bit position 7 of result is 15th bit
- Write the 47th bit
  - read OUT register at $...015
  - set bit position 7 to desired value (or with 0x80)
  - store in $...015
  - load EN register at $...005
  - set bit to output
  - store value back to $...005

External PWM Unit

- Design a system to control a digital
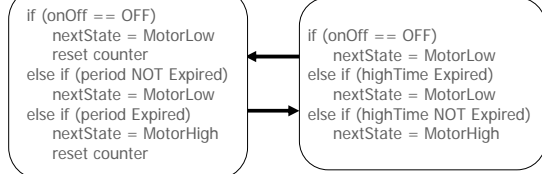- Solution: design a PWM unit

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
- Compass
- 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
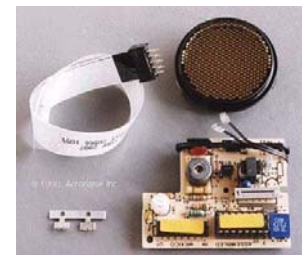 - want to take multiple readings for high accuracy

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
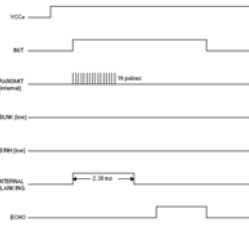 - high-voltage circuitry to prepare disc for transmitting and then receiving

6

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
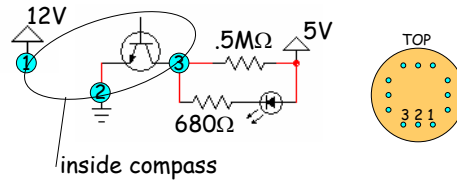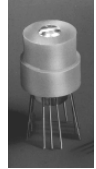

Compass

- Four compass directions (each has three pins)
- One-hot/two-hot encoding
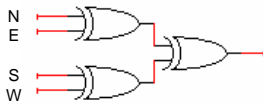 - one-hot for N, E, S, W
  - two-hot for NE, SE, SW, NW

inside compass

Compass (cont'd)

- Detecting a change in compass direction
  - 4 bits change from 0001 to 0011 to 0010 to 0110 to 0100 …
  - Always alternating between one bit on and two bits on
- Parity tree can detect difference between one and two bits being asserted
  - XOR tree of four bits (one TTL SSI package)
  - Output must change at least once for every change in orientation
  - Use interrupts to detect changes

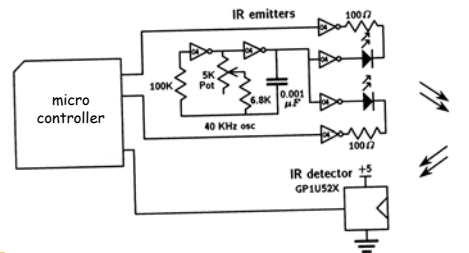e.g.,
NE → E → SE
1100 → 0100 → 0110
0 → 1 → 0

IR proximity detector

- Oscillator must be set to match detector
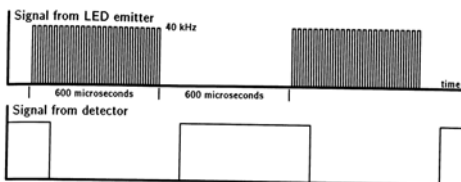

IR frequency modulation

Proximity code

```
turn on emitter
sleep for 600us                    timer goes off
val_on = read detector             wake
turn off emitter
sleep for 600us                    timer goes off
val_off = read detector            wake
return ( val_on & ~val_off )              Mostly in main
```
```
turn on emitter                  Using interrupt handlers
set timer
sleep         timer goes off
              val_on = read detector
              turn off emitter
              reset timer
              sleep        timer goes off (again)
                           val_off = read detector
return ( val_on & ~val_off )  wake
```
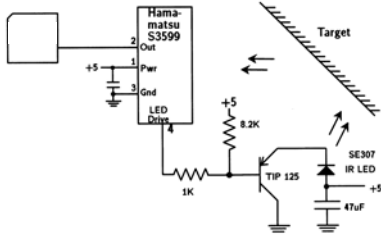
More integrated proximity detector

- Always sending out IR
- Detector drives LED (guaranteed to match frequency)
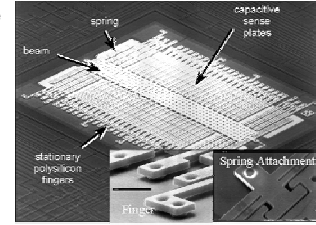
Accelerometer

- 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.

Accelerometer output

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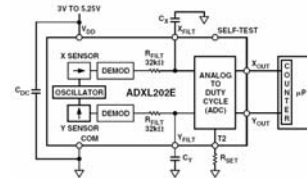- Use pulse-width modulation
- What about gravity?

Analog Devices ADXL202

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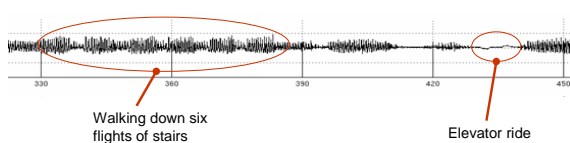 - 12.5% per g in either direction

Typical measurement for ADXL202

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

Walking down six flights of stairs
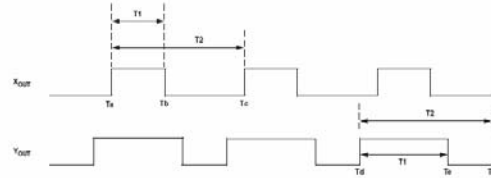
Elevator ride

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
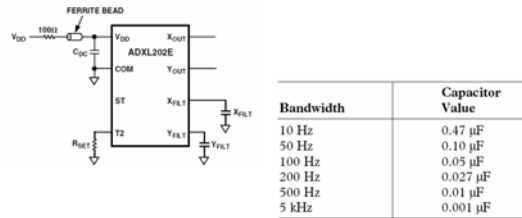- Same for Y-axis output (T2 is the same for both axes)

8

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 4 readings
    current average = ¾ * current average + ¼ * 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 limited noise frequency
  - bandwidth limiting

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

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;
```

Bright LED

- 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

9

Sample code for LED

- Varying PWM output

```
volatile uint8_t width; /* positive pusle width */
volatile uint8_t delay; /* used to slow the pulse width changing */

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);
}
```
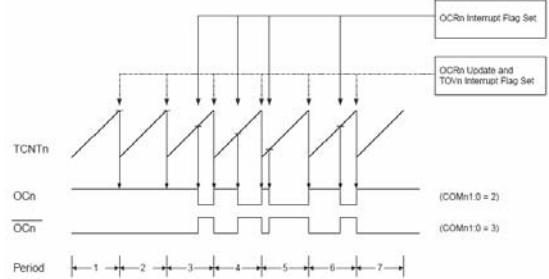
Fast PWM
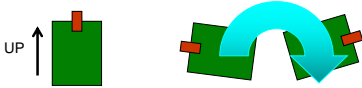
Ball Lightning

- Use accelerometer to brighten/dim LED
- Think of it as a virtual knob with enabling button
  - Turn clockwise – brighten, turn counter-clockwise – dim
- Speed of turn
  - 3 different speeds
    - Fast – about 180 degrees of turn goes from off to brightest
    - Medium – about 360 degrees of turn (2 180 degree turns)
    - Slow – about 720 degrees of turn (4 180-degree turns)

UP