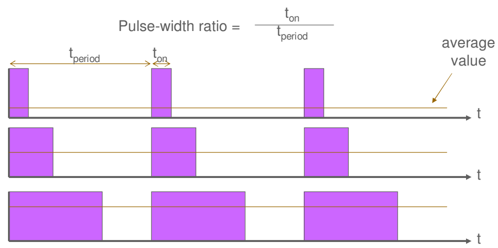
Pulse-width modulation

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



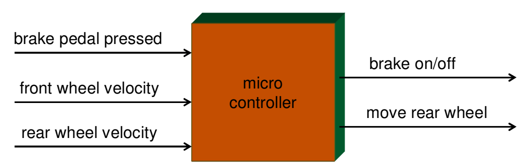
Why pulse-width modulation works

- 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

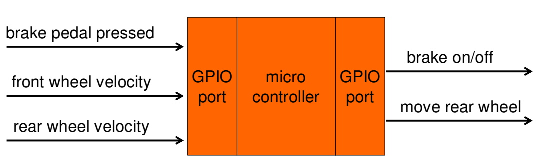
- Rear wheel controller/anti-lock brake system
  - Normal operation
    - Regulate velocity of rear wheel
  - Brake pressed
    - Gradually increase amount of breaking
    - If skidding (front wheel is moving much faster than rear wheel) then temporarily reduce amount of breaking
- Inputs
  - Brake pedal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



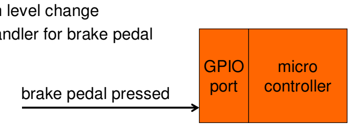
Basic I/O ports (brakes)

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



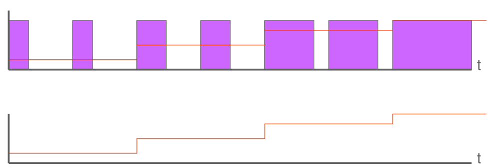
Polling vs. interrupts

- 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



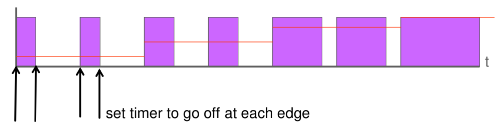
Pulse-width modulation for brakes

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



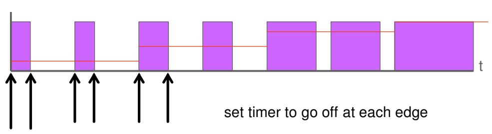
Brake pump setup

- 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?



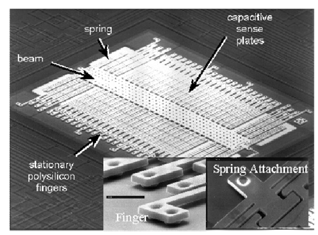
Brake pump setup (cont’d)

- Change value of “on” time to change analog average
  - average output = (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



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.

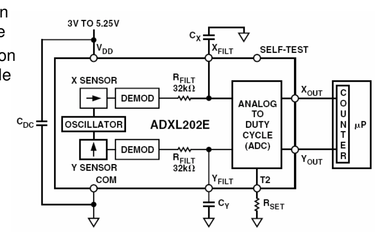


Accelerometer output

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

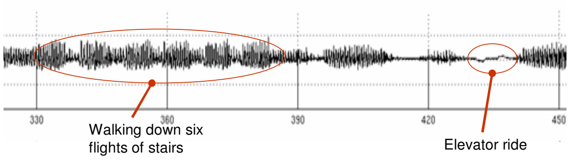
Analog Devices ADXL202

- 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



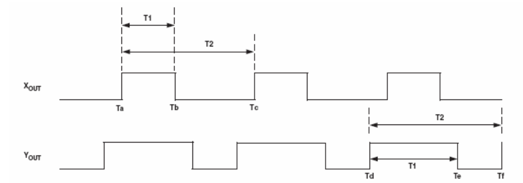
Typical measurement for ADXL202

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



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)

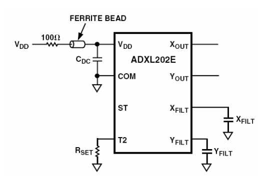


What to do about noise/jitter?

- 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 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

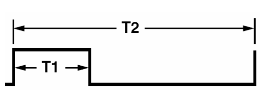
- Filter capacitors limited noise frequency
  - bandwidth limiting



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%)

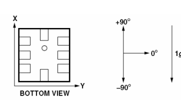


$A(g) = (T1/T2 – 0.5)/12.5\%$
$0g$ = 50% DUTY CYCLE
$T2(s) = R_{SET}(\Omega)/125M\Omega$

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



X Axis Orientation to Horizon (°)	X Output: X Output (g)	X Output: Δ per Degree of Tilt (mg)	Y Output (g): Y Output (g)	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;
```


Bright LED

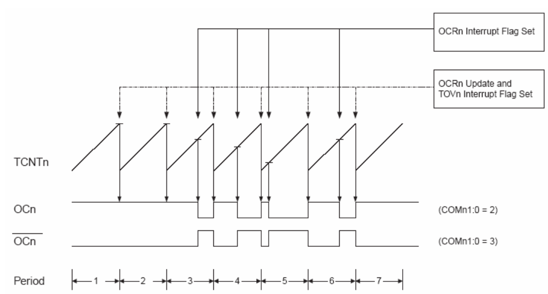
- Easy to control intensity of light through pulse-width modulation
- Duty-cycle is averaged by human eye
  - Light is really turning on and off each period
  - Too quickly for human retina (or most video cameras)
  - Period must be short enough (< 1ms is a sure bet)
- LED output is low to turn on light,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

Sample code for LED

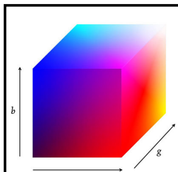
- Varying PWM output

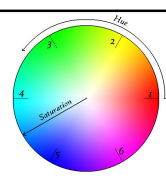
```
volatile uint8_t width; /* positive pusle width */
volatile uint8_t delay; /* used to slow the pulse width changing */

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



Color

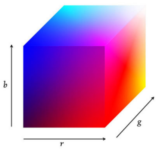




- Color perception usually involves three quantities:
  - *Hue*: Distinguishes between colors like red, green, blue, etc
  - *Saturation*: How far the color is from a gray of equal intensity
  - *Lightness*: The perceived intensity of a reflecting object
- Sometimes lightness is called *brightness* if 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.

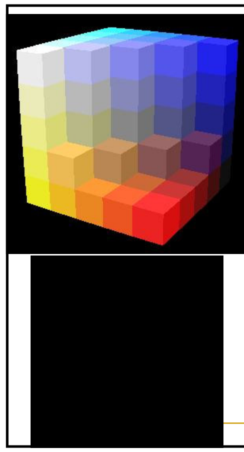
Color Spaces

- Definition: A mapping of color components onto a Cartesian coordinate system in three or more dimensions.
- 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

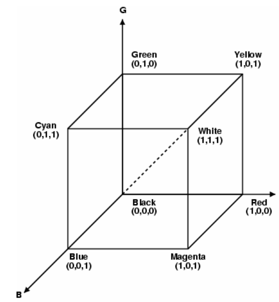


Additive Model: (RGB System)

- 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 in 1931: B: 435.8 nm, G: 546.1 nm, R: 700 nm.
- 3 fixed components acting alone can't generate all spectrum colors.



RGB Color space

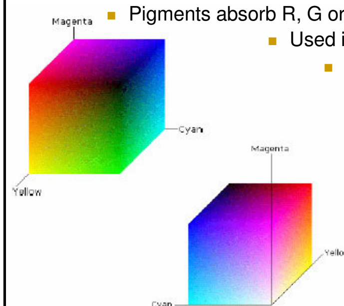


Problems with RGB

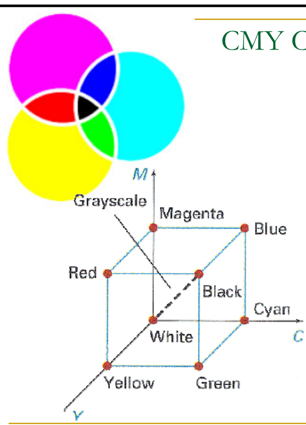
- Only a small range of potential perceivable colors (particularly for monitor RGB)
- It isn't easy for humans to say how much of RGB to use to get a given color
  - How much R, G and B is there in "brown"?
- Perceptually non-linear
  - Two points, a certain distance apart, may be perceptually different in one part of the space, but could be same in another part of the space.

Subtractive model (CMY System)

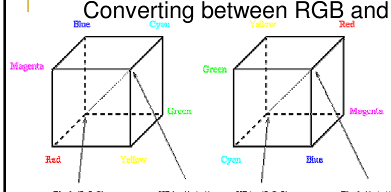
- Color results from removal of light from the illumination source
- Pigments absorb R, G or B and so give C, M or Y
- Used in deskjet/ inkjet printers.
- No ink (pigment) = white



CMY Color space



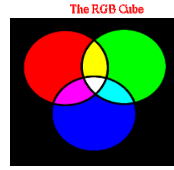
Converting between RGB and CMY



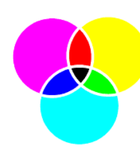
$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

The RGB Cube

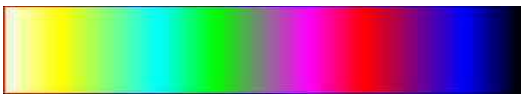


The CMY Cube



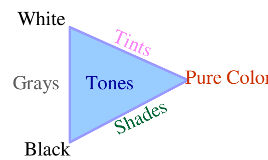
Specifying Color

- 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.



How Do Artists Do It?

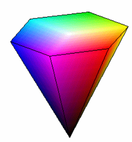
- Artists often specify color as tints, shades, and tones of saturated (pure) pigments
- *Tint*: Gotten by adding white to a pure pigment, decreasing saturation
- *Shade*: Gotten by adding black to a pure pigment, decreasing lightness
- *Tone*: Gotten by adding white and black to a pure pigment

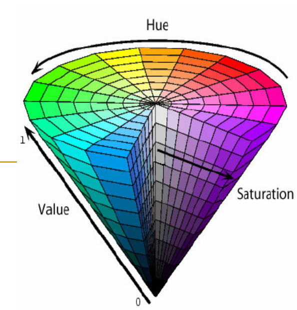


HSV Color Space

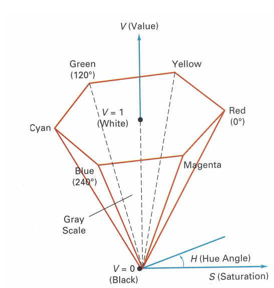
- Computer scientists frequently use an intuitive color space that corresponds to tint, shade, and tone:
  - Hue - The color we see (red, green, purple)
  - Saturation - How far is the color from gray (pink is less saturated than red, sky blue is less saturated than royal blue)
  - Brightness (Luminance)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

HSV Color space





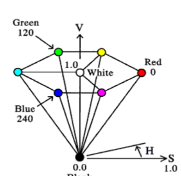
HSV Color Model

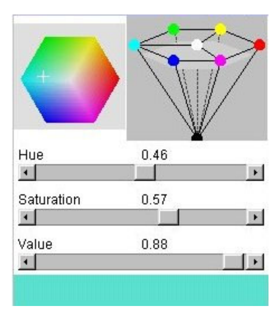


- Hue (H) is the angle around the vertical axis
- Saturation (S) is a value from 0 to 1 indicating how far from the vertical axis the color lies
- Value (V) is the height of the hexcone"

HSV Color Space

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



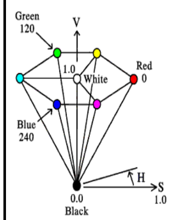


http://www.cs.rit.edu/~ncs/color/a_spaces.html

HSV System

- Normally represented as a cone or *hexcone*
- Hue is the angle around the circle or the regular hexagon; $0 \le H \le 360$
- Saturation is the distance from the center; $0 \le S \le 1$
- Value is the position along the axis of the cone or hexcone; $0 \le V \le 1$
- Value is not perceptually-based, so colors of the same value may have slightly different brightness
- Main axis is grey scale

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
}
```


Lab 3 Objectives

- The goal of this lab is to implement a virtual knob in HSV color space to generate the majority of colors using a tri-color LED in RGB color space.
- You will determine the movement of the virtual knob by measuring accelerometer readings through pulse width measurements.
- In addition, you will also use pulse width modulation to control the brightness of the LEDs.

Lab 3 actions

- how to read an accelerometer via pulse width measurement
- how to use the input capture on the 16-bit timer on the ATmega16 to do so
- how to adjust the intensity of a light using pulse width modulation

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