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

- physical system
  - sensors
  - actuators
  - controller
  - interfaces

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)

![Diagram of analog to digital conversion](image-url)
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

![Logarithm Circuit Diagram]

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

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

![Capacitor Diagrams]

- C, 3C, 7C
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 $Q_S = Q_H$
  - $16 (V_X - V_I) = 16 (V_L - V_I)$
  - $V_X - V_L = V_L - V_I$ (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 \( + \frac{2}{16} (V_H - V_L) - \frac{2}{16} (V_L - V_I) = \frac{2}{16} (V_H - V_L) \)
  - 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 \( + \frac{1}{16} (V_H - V_L) - \frac{1}{16} (V_L - V_I) = \frac{1}{16} (V_H - V_L) \)
  - 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 1010 or 10/16 = 20/32
- 3% error

A-to-D Conversion Errors
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<sub>CC</sub> input range
- 2*LSB accuracy (2 * 1/1024 = ~0.2%)
A-to-D Conversion (cont’d)

- Bit 7:6 - REFS1:0: Reference Selection Bits
  These bits select the voltage reference for the ADC, as shown in Table B3. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (AUIF 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>
<thead>
<tr>
<th>REFS1</th>
<th>REFS0</th>
<th>Voltage Reference Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td></td>
<td>AVCC, internal Vref turned off</td>
</tr>
<tr>
<td>0 1</td>
<td></td>
<td>A/CC with external capacitor at AREF pin</td>
</tr>
<tr>
<td>1 0</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>1 1</td>
<td></td>
<td>Internal 2.56V voltage Reference with external capacitor at AREF pin</td>
</tr>
</tbody>
</table>

- 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. Whatever the ADLAR bit will affect the ADCL/ADCH 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[2,3] – ADC2 at 200x
  - ADC[2,3] – ADC3 at 200x
  - ADC[0,1,2,3,4,5] – ADC2 at 1x

<table>
<thead>
<tr>
<th>Mode</th>
<th>Single-Ended Input</th>
<th>Positive Differential Input</th>
<th>Negative Differential Input</th>
<th>Show</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>ADCH</td>
<td>ADCL</td>
<td>1x</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>ADCH</td>
<td>ADCL</td>
<td>10x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ADCH</td>
<td>ADCL</td>
<td>200x</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>ADCH</td>
<td>ADCL</td>
<td>1x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ADCH</td>
<td>ADCL</td>
<td>10x</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>ADCH</td>
<td>ADCL</td>
<td>200x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ADCH</td>
<td>ADCL</td>
<td>1x</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>ADCH</td>
<td>ADCL</td>
<td>10x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ADCH</td>
<td>ADCL</td>
<td>200x</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>ADCH</td>
<td>ADCL</td>
<td>1x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ADCH</td>
<td>ADCL</td>
<td>10x</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>ADCH</td>
<td>ADCL</td>
<td>200x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ADCH</td>
<td>ADCL</td>
<td>1x</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>ADCH</td>
<td>ADCL</td>
<td>10x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ADCH</td>
<td>ADCL</td>
<td>200x</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>ADCH</td>
<td>ADCL</td>
<td>1x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ADCH</td>
<td>ADCL</td>
<td>10x</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>ADCH</td>
<td>ADCL</td>
<td>200x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ADCH</td>
<td>ADCL</td>
<td>1x</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>ADCH</td>
<td>ADCL</td>
<td>10x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>ADCH</td>
<td>ADCL</td>
<td>200x</td>
</tr>
</tbody>
</table>
A-to-D Conversion (cont’d)

The ADC Data Register – ADC0 and ADC1

ADC0 = 0

<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

Read/Write: 0 0 0 0 0 0 1 0
Initial Value: 0 0 0 0 0 0 0 0

ADC0 = 1

<table>
<thead>
<tr>
<th>Bit</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>D</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>

Read/Write: 0 0 0 0 0 0 1 0
Initial Value: 0 0 0 0 0 0 0 0

A-to-D Conversion (cont’d)

ADC Control and Status Register A – ADC0RA

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>ADE</td>
<td>ADC</td>
<td>ADIE</td>
<td>AFC</td>
<td>ADE</td>
<td>ADIE</td>
<td>ADIE</td>
<td>ADIE</td>
</tr>
</tbody>
</table>

Read/Write: 0 0 0 0 0 0 0 0

- Bit 7 – ADE: ADC Enable
- Bit 6 – ADIE: ADC Start Conversion
- Bit 5 – ADE: ADC Auto Trigger Enable
- Bit 4 – ADIE: ADC Interrupt Flag
- Bit 3 – ADIE: ADC Interrupt Enable
- Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits

<table>
<thead>
<tr>
<th>ADPS2</th>
<th>ADPS1</th>
<th>ADPS0</th>
<th>Division Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>128</td>
</tr>
</tbody>
</table>
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

\[ \text{Pulse-width ratio} = \frac{t_{\text{on}}}{t_{\text{period}}} \]

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 \((\tau = RC)\) sets “cutoff” frequency that separates low and high frequencies
Antilock 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)

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?

set timer to go off at each edge
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

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

- There 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 the 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
Memory-mapped I/O

- Address bus selects device
- Data bus contains data

![Diagram of memory-mapped I/O with address bus, data bus, and device addr]

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

<table>
<thead>
<tr>
<th>Address</th>
<th>device select</th>
<th>can be used as inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>msb</td>
<td>lsb</td>
<td></td>
</tr>
</tbody>
</table>

![Table for address mapping]

---

CSE 466 - Winter 2007

Interfacing
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

```
<table>
<thead>
<tr>
<th>uc</th>
<th>addr</th>
<th>port expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ctrl</td>
<td></td>
</tr>
</tbody>
</table>
```

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

![Diagram of PWM FSM Controller]

**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
- Pulse sensor
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 sensitized 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

IR proximity detector

- Sends out IR pulses at a set frequency
- Detector drives LED (guaranteed to match frequency)
IR frequency modulation

Proximity code

- turn on emitter
- sleep for 600us
- val_on = read detector
- turn off emitter
- sleep for 600us
- val_off = read detector
- return (val_on & ~val_off)

Mostly in main

- Using interrupt handlers
- turn on emitter
- set timer
- sleep
- timer goes off
- val_on = read detector
- turn off emitter
- reset timer
- sleep
- timer goes off (again)
- val_off = read detector
- wake
- return (val_on & ~val_off)
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
  - 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)

![Typical signal from ADXL202](image)

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 = \( \frac{7}{8} \) * current average + \( \frac{1}{8} \) * 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

<table>
<thead>
<tr>
<th>Bandwidth</th>
<th>Capacitor Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Hz</td>
<td>0.47 μF</td>
</tr>
<tr>
<td>50 Hz</td>
<td>0.10 μF</td>
</tr>
<tr>
<td>100 Hz</td>
<td>0.05 μF</td>
</tr>
<tr>
<td>200 Hz</td>
<td>0.027 μF</td>
</tr>
<tr>
<td>500 Hz</td>
<td>0.01 μF</td>
</tr>
<tr>
<td>5 kHz</td>
<td>0.001 μF</td>
</tr>
</tbody>
</table>

### 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%)

\[
A(g) = \frac{(T1/T2 - 0.5)}{0.5}\% \\
0g = 50\% \text{ DUTY CYCLE} \\
T2(s) = \frac{R_{SET}(\Omega)}{125M\Omega}
\]

<table>
<thead>
<tr>
<th>T2</th>
<th>( R_{SET} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ms</td>
<td>125 kΩ</td>
</tr>
<tr>
<td>2 ms</td>
<td>250 kΩ</td>
</tr>
<tr>
<td>5 ms</td>
<td>625 kΩ</td>
</tr>
<tr>
<td>10 ms</td>
<td>1.25 MΩ</td>
</tr>
</tbody>
</table>
ADXL202 Orientation

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

<table>
<thead>
<tr>
<th>X Axis Orientation to Horizontal (°)</th>
<th>X Output (p)</th>
<th>2 per Degree of Tilt (p)</th>
<th>V Output (p)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-90</td>
<td>-1.060</td>
<td>-0.2</td>
<td>0.030</td>
</tr>
<tr>
<td>-75</td>
<td>-0.805</td>
<td>0.11.9</td>
<td>0.230</td>
</tr>
<tr>
<td>-60</td>
<td>-0.636</td>
<td>0.12.2</td>
<td>0.270</td>
</tr>
<tr>
<td>-45</td>
<td>-0.509</td>
<td>0.12.4</td>
<td>0.310</td>
</tr>
<tr>
<td>-30</td>
<td>-0.389</td>
<td>0.12.6</td>
<td>0.350</td>
</tr>
<tr>
<td>-15</td>
<td>-0.255</td>
<td>0.12.6</td>
<td>0.395</td>
</tr>
<tr>
<td>0</td>
<td>0.000</td>
<td>12.5</td>
<td>0.945</td>
</tr>
<tr>
<td>15</td>
<td>0.225</td>
<td>12.9</td>
<td>0.985</td>
</tr>
<tr>
<td>30</td>
<td>0.500</td>
<td>15.2</td>
<td>0.965</td>
</tr>
<tr>
<td>45</td>
<td>0.707</td>
<td>15.5</td>
<td>0.945</td>
</tr>
<tr>
<td>60</td>
<td>0.800</td>
<td>20.0</td>
<td>0.930</td>
</tr>
<tr>
<td>75</td>
<td>0.866</td>
<td>8.7</td>
<td>0.230</td>
</tr>
<tr>
<td>90</td>
<td>1.060</td>
<td>0.2</td>
<td>0.090</td>
</tr>
</tbody>
</table>

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

```c
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

```c
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);
    TMR2 = _BV(T0IF2);
    /* setup initial conditions */
    delay = 0;
    /* enable interrupts */
    sei();
    for (;;) {
        // LOOP FOREVER as the interrupt will make necessary adjustment
        return (0);
    }
}
```
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
**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

**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 $2\pi$ 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)

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

HSV to RGB Conversion

```cpp
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                   { 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 \text{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