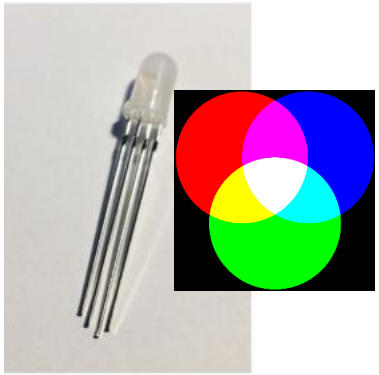
PWM and Hardware components Lab

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Lab Overview

- This lab will introduce use of :
  - An RGB LED that can produce many different colors when combined with the hardware PWM of your Arduino
    - Can use a **pulse-width-modulation (PWM)** style output to control ***BRIGHTNESS***.



  - A servo motor
    - Requires a **pulse-width-modulation (PWM)** style output to control it



  - Combinational and sequential logic chips to offload some processing to hardware (and not software)



© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

This Photo by Unknown Author is licensed under CC BY-SA

PULSE WIDTH MODULATION (PWM)

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

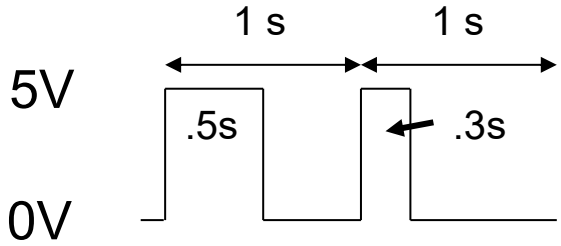
Power

- Recall (or learn) that Power is a measure of:
  - Energy per unit time
- In an electronic circuit, P = I * V
  - Power = Current & Voltage (each may be varying w/ time)
- A circuit that draws a constant 2 mA of current at a constant 5V would consume 10 mW
- Since voltage and current may change rapidly, it is often helpful to calculate the average power

$$P = \frac{1}{T}\int_0^T P(t)dt$$

- Just sum the total power and divide by the total time

I = 1A



Average Power
= (1*5*.8)/2 = 2W

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Output Devices

- What do the following have in common?
  - Servo motor that can rotate to any angle w/in 180 degrees
  - Light dimmer
  - Oven or microwave with various power levels
- They are controlled by Pulse Width Modulation (PWM)
  - Usually a 3-pin interface: Power (Vcc), GND, PWM Signal



© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

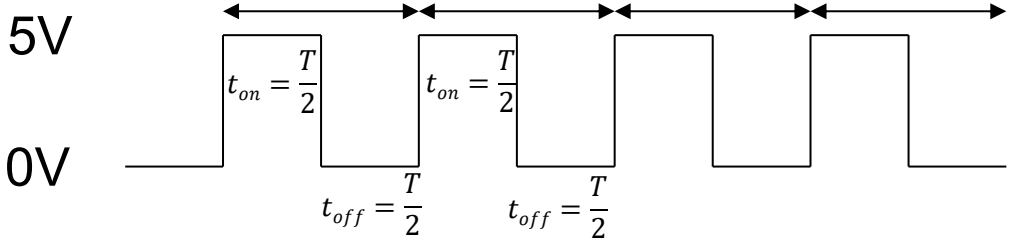
Duty Cycle

- A pulse is just a short window of time when a signal is ON (t_{on})
- We could repeat the pulse at some regular period, **T**
- We define the duty cycle as

$$Duty\ Cycle\ \% = \left(\frac{t_{on}}{T}\right) * 100 = \left(\frac{t_{on}}{t_{on} + t_{off}}\right) * 100$$

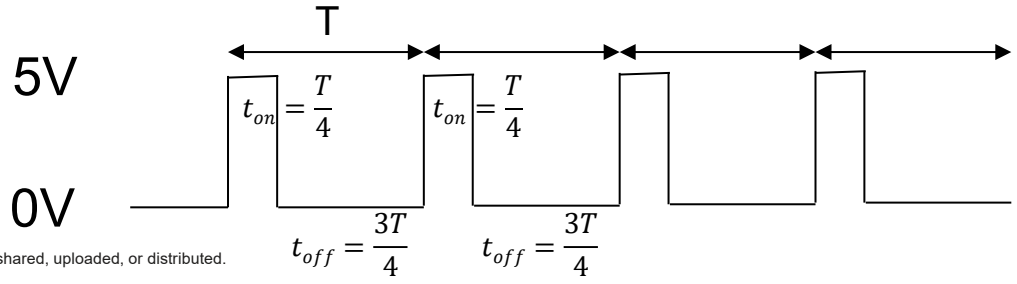
Duty Cycle = 50%

$\left(\frac{t_{on}}{T} = \frac{T/2}{T} = \frac{1}{2}\right)$



Duty Cycle = 25%

$\left(\frac{t_{on}}{T} = \frac{T/4}{T} = \frac{1}{4}\right)$



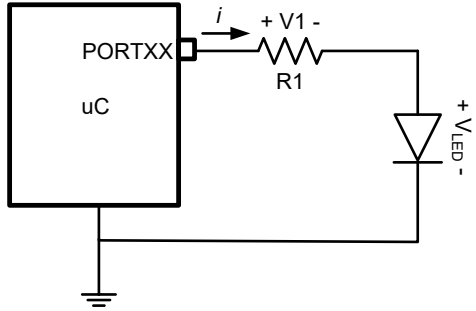
© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

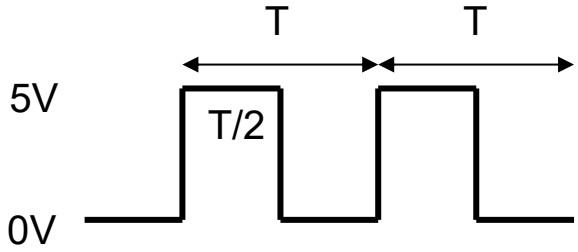
Power & Duty Cycle

- When we light up an LED we often just turn a PORTxx output 'on' and leave it 'on'
  - This supplies the maximum power possible to the LED and it glows at a "constant" brightness
- We could pulse the output at some duty cycle (say 50%) at a fast rate
  - Faster than the human eye can't detect it flashing
  - Average power would be ½ the original always 'on' power
  - Result would be a 'dimmer' LED glow



PORTxx 'on' constantly



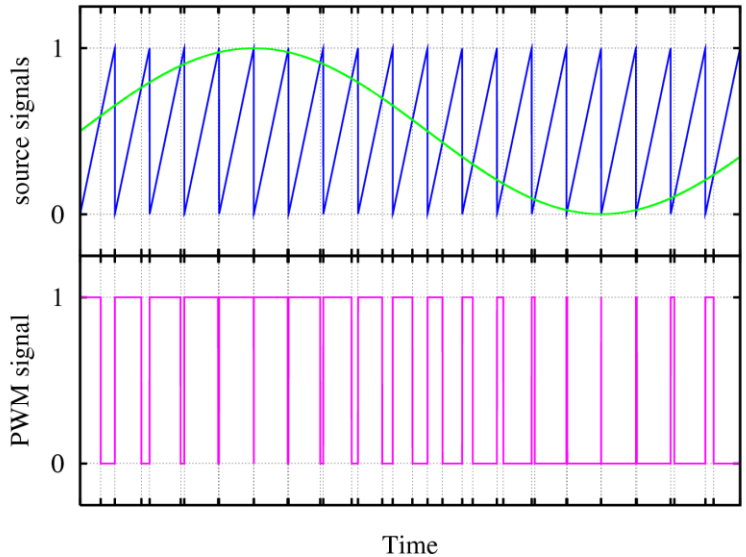


PORTxx 'on' 50% of time

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

PWM

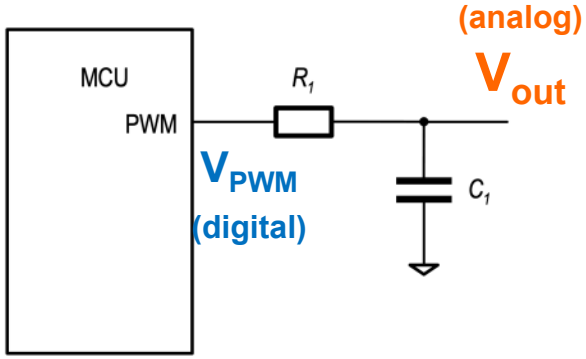
- Modulation refers to changing a value based on some signal (i.e. changing one signal based on another)
- Pulse width modulation refers to modifying the width of a pulse based on another signal
- It can be used to transform one signal into another
  - Example below of sine wave represented as pulses w/ different widths
- Or it can just be used to alter average power as in the last activity

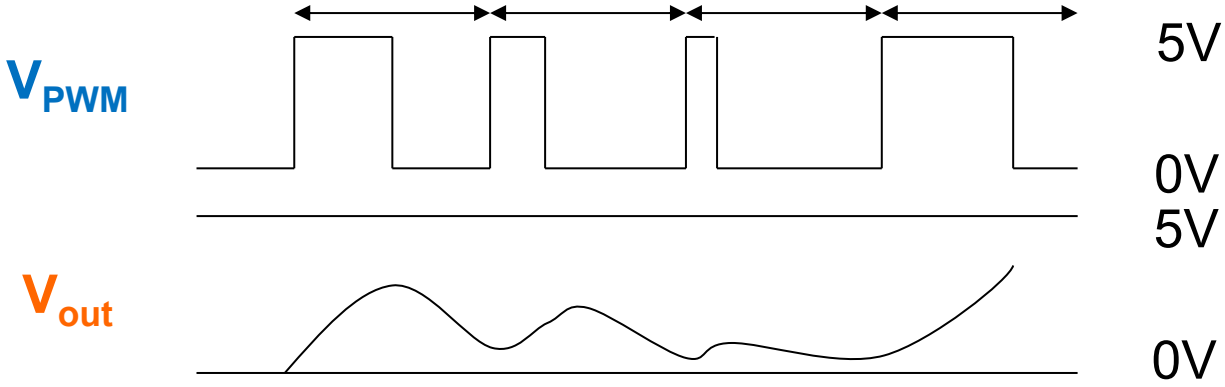


© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Simple Digital-To-Analog

- We can create a simple form of Digital to Analog Conversion by connecting a PWM output to a simple resistor-capacitor circuit as shown
- The voltage at Vout (charge on the capacitor) "integrates" the digital PWM signal
  - Analogy: Imagine you have a leaky bucket (i.e. capacitor) and you want to produce a variable level (i.e. analog voltage, Vc) of water by only turning the hose (digital output) on or off





© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Servo Motors

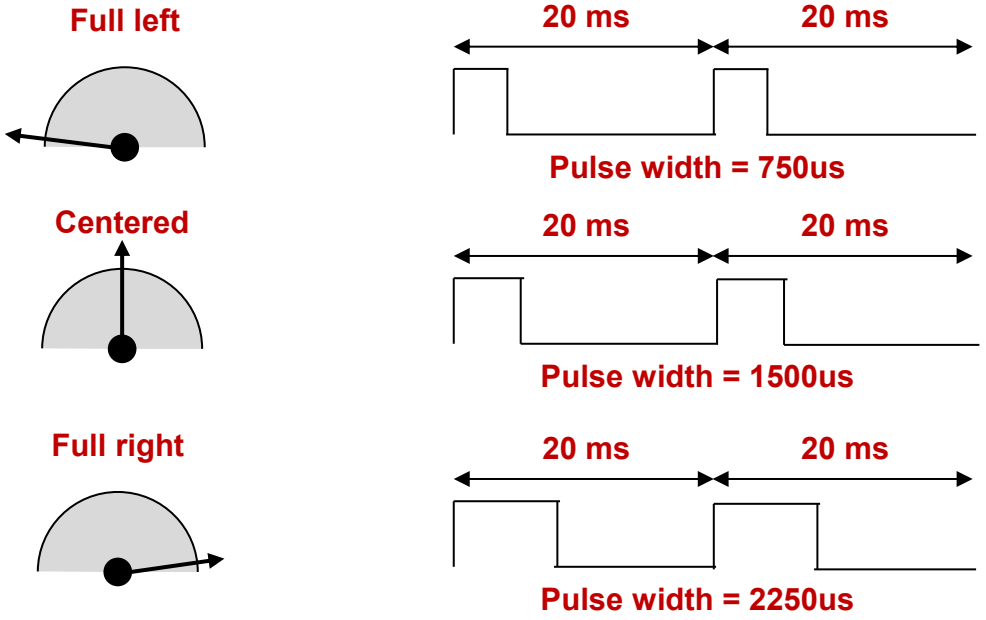
- Many embedded systems use servo motors to move or rotate mechanical devices
- Most servo motors use some form of pulse width modulation to control the direction and speed of their rotation
- 2 Kinds
  - Standard servo motors: can only rotate through a certain arc (usually 180 degrees)
  - Continuous: can keep spinning round and round while pulses are provided



© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Standard Servo Motor

- Pulse width determines angle (position) of servo motor
- Must continue to give pulses for the duration of time it takes to rotate to the desired position
- No pulses = no movement (rotor stays at current position)

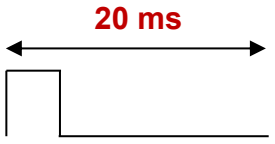


Do an Internet search for Standard Servo Motors & try to find the appropriate pulse width for each position

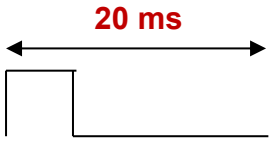
© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Continuous Servo Motors

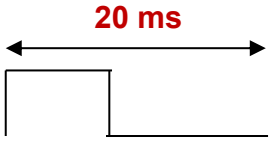
- Pulse width determines speed & direction of rotation
- Controlled via PWM (Pulse Width Modulation)
  - Short pulse = Rotate one direction
  - Medium pulse = Stop
  - Long pulse = Rotate other direction



Pulse Width = 1000 us = Full Speed Clockwise



Pulse Width = 1500 us = Stopped



Pulse Width = 2000 us = Full Speed Counter-Clockwise

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

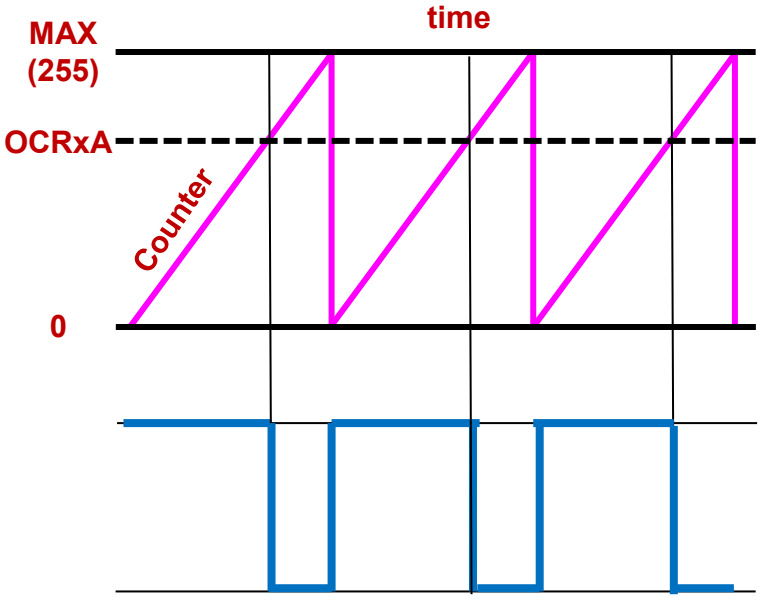
Implementing PWM

- Option 1: Can use delays or timers to **make your own** pulse signals
- Option 2: But most microcontrollers have **hardware to automatically generate PWM signals** based on the contents of some control registers
  - The hardware for PWM is the **SAME hardware as the timers**
  - Recall the **timer** module gave us a **counter** that would increment until it hit some 'modulus' (MAX) count which would cause it to restart and also generate an interrupt…
  - What if we don't restart at 0 but keep counting to a second value.

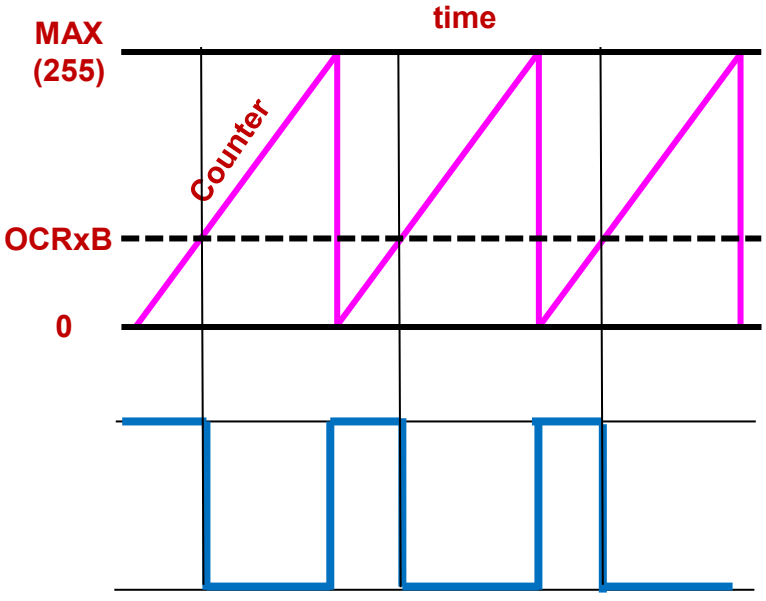
© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Using Timers for PWM

- Can implement PWM by having the timer count 0 to some MAX, making the:
  - PWM output = '1' while the count < threshold (OCRxx) and
  - PWM output = '0' when the count >= threshold (OCRxx)



PWM Output 1



PWM Output 2

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

PWM Control Registers

- The Arduino has 3 timers that can be used for PWM:
  - **Two 8-bit timers: Timer/Counter0 and Timer/Counter2**
  - **One 16-bit timer: Timer/Counter1**
- Refer to Timer slides with following **modifications**
  - Set WGM0[2:0] bits for Fast PWM mode as opposed to CTC

Timer + Channel A/B	Associated Output
Timer 0, Ch. A	Port D, bit 6
Timer 0, Ch. B	Port D, bit 5
Timer 1, Ch. A	Port B, bit 1
Timer 1, Ch. B	Port B, bit 2
Timer 2, Ch. A	Port B, bit 3
Timer 2, Ch. B.	Port D, bit 3

See datasheet, textbook or other documentation for further explanation

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

8-bit TIMER0 PWM Control Registers

- Set WGM bits for PWM mode
  - **Usually, Fast PWM mode as opposed to CTC**
- Pick COM0?[1:0] for desired waveform
- Still need to pick a prescalar to slow down the clock
- Set OCR0A or OCR0B to the desired threshold which will effectively control the duty cycle of the PWM output

COM 0A1	COM 0A0	COM 0B1	COM 0B0	-	-	WGM 01	WGM 00

TCCR0A Reg.
Timer/Counter0 Control Register

FOC 0A	FOC 0B	-	-	WGM 02	CS02	CS01	CS00

TCCR0B Reg.
Timer/Counter0 Control Register

WGM01, WGM00	WGM02=0	WGM02=1
00	Normal (Counter)	Unused
01	Phase Correct PWM	Phase Correct PWM (Top=OCR0A)
10	CTC (Timer)	Unused
11	**Fast PWM (Top=255, Thresh=OCRx)**	Fast PWM (Top=OCR0A, Thresh = OCR0B)

CS0[2:0]	Prescalar
000	Timer off
010	Clk / 8
011	Clk / 64
100	Clk / 256
101	Clk / 1024

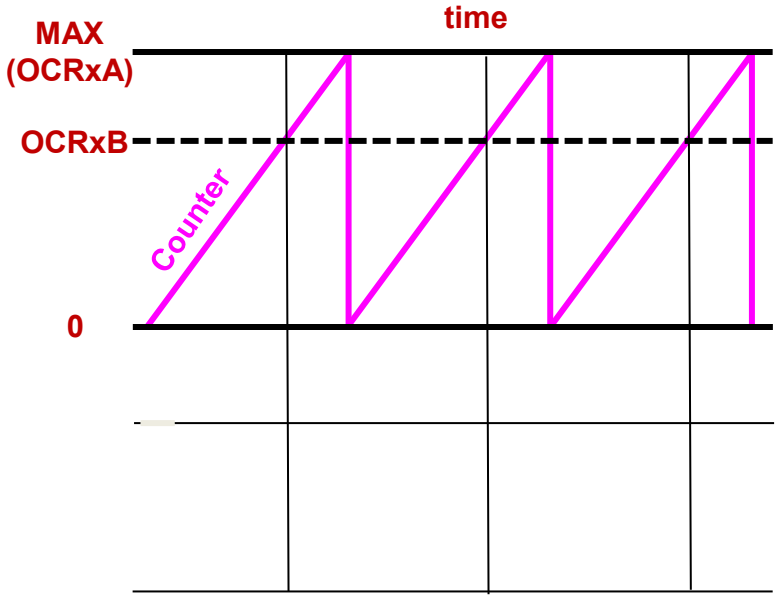
COM0?1, COM0?0	Output Compare pin (assume WGM02=0)
00	Don't use Pin
01	Don't use Pin
10	**Set Pin on CTR=0x00, Clear pin on match=OCR0A/B**
11	**Clear Pin on CTR=0x00, Set pin on match=OCR0A/B**

8-bit Timers can only count to 255. Be sure to select a prescalar such that your OCR value will fit in 8-bits.

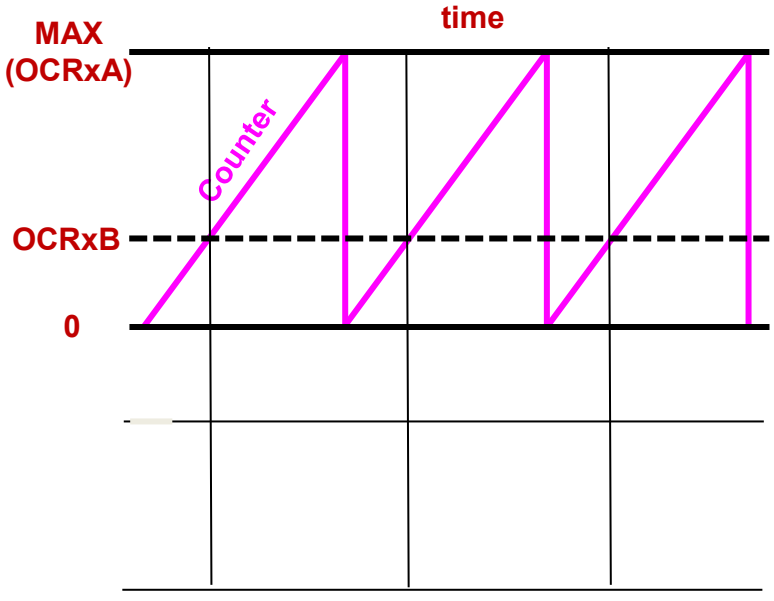
© 2025 by Mark Redekopp

Using Timers for PWM (TOP=255)

- In the examples we've show (with WGMx2 = 0), the MAX is 2^n-1 (where n=8 or 16) based on which timer you use



PWM Output 1

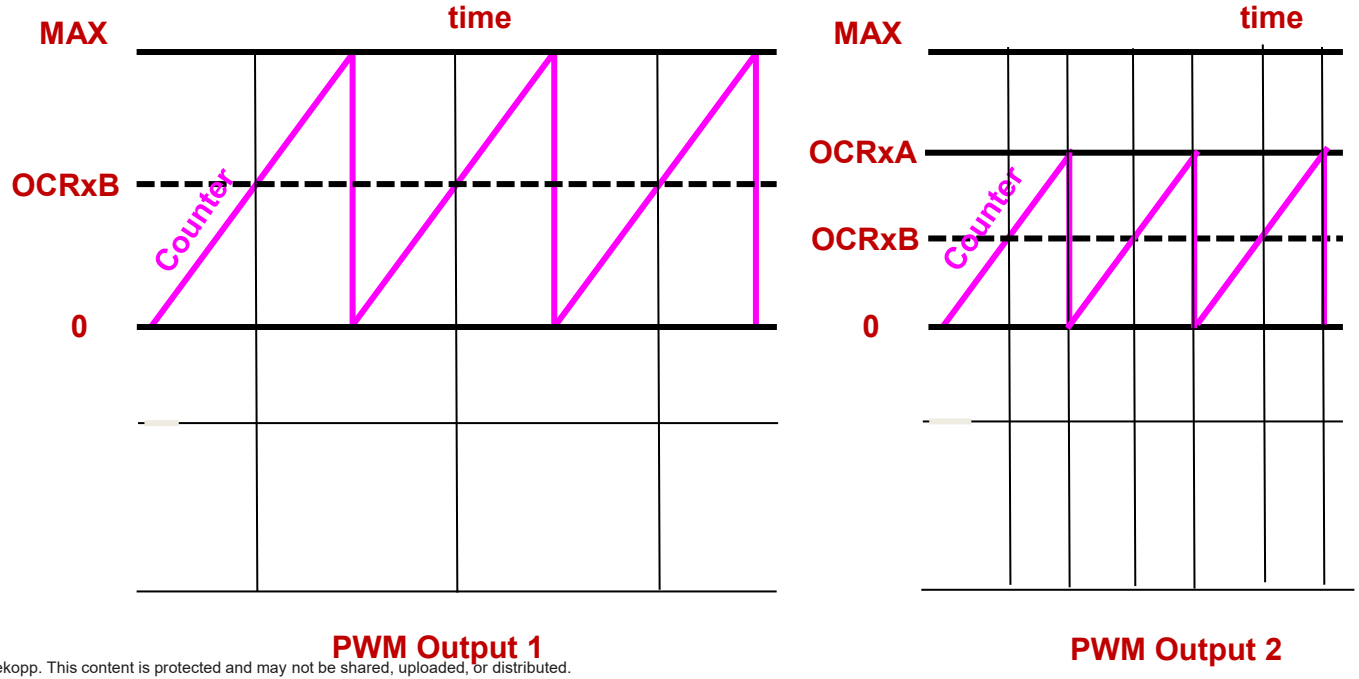


PWM Output 2

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Using Timers for PWM (TOP=OCRxA)

- In a different mode (WGMx2 = 1) we can have a user-specified MAX by placing any number in OCRA



© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

16-bit TIMER1 PWM Control Registers

- Set WGM bits for PWM mode [usually Fast PWM mode] as opposed to CTC
- Pick COM1?[1:0] for desired waveform
- Still need to pick a prescaler to slow down the clock
- Set OCR1A or OCR1B to the desired threshold which will effectively control the duty cycle of the PWM output

COM 1A1	COM 1A0	COM 1B1	COM 1B0	-	-	WGM 11	WGM 10

TCCR1A Reg.
Timer/Counter1 Control Register

FOC 1A	FOC 1B	-	WGM 13	WGM 12	CS12	CS11	CS10

TCCR1B Reg.
Timer/Counter1 Control Register

WGM11, WGM10	WGM12=0	WGM12=1
00	Normal (Counter)	Unused
01	Phase Correct PWM	Phase Correct PWM (Top=OCR1A)
10	CTC (Timer)	Unused
11	**Fast PWM (Top=2^{16}-1, Thresh=OCRx)**	Fast PWM (Top=OCR1A, Thresh = OCR1B)

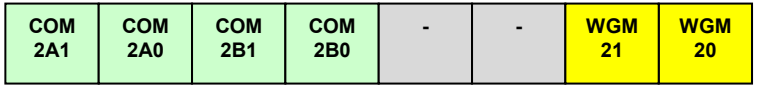
CS1[2:0]	Prescalar
000	Timer off
010	Clk / 8
011	Clk / 64
100	Clk / 256
101	Clk / 1024

COM1?1, COM1?0	Output Compare pin (assume WGM12=0)
00	Don't use Pin
01	Don't use Pin
10	**Set Pin on CTR=0x0000, Clear pin on match=OCR1A/B**
11	**Clear Pin on CTR=0x0000, Set pin on match=OCR1A/B**

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

8-bit TIMER2 PWM Control Registers

- Set WGM bits for PWM mode
  - **Usually Fast PWM mode as opposed to CTC**
- Pick COM2?[1:0] for desired waveform
- Still need to pick a prescalar to slow down the clock
- Set OCR2A or OCR2B to the desired threshold which will effectively control the duty cycle of the PWM output



TCCR2A Reg.
Timer/Counter2 Control Register



TCCR2B Reg.
Timer/Counter2 Control Register

WGM21, WGM20	WGM22=0	WGM22=1
00	Normal (Counter)	Unused
01	Phase Correct PWM	Phase Correct PWM (Top=OCR2A)
10	CTC (Timer)	Unused
11	**Fast PWM (Top=255, Thresh=OCRx)**	Fast PWM (Top=OCR2A, Thresh = OCR2B)

CS2[2:0]	Prescalar
000	Timer off
010	Clk / 8
011	Clk / 32
100	Clk / 64
101	Clk / 128
110	Clk / 256
111	Clk / 1024

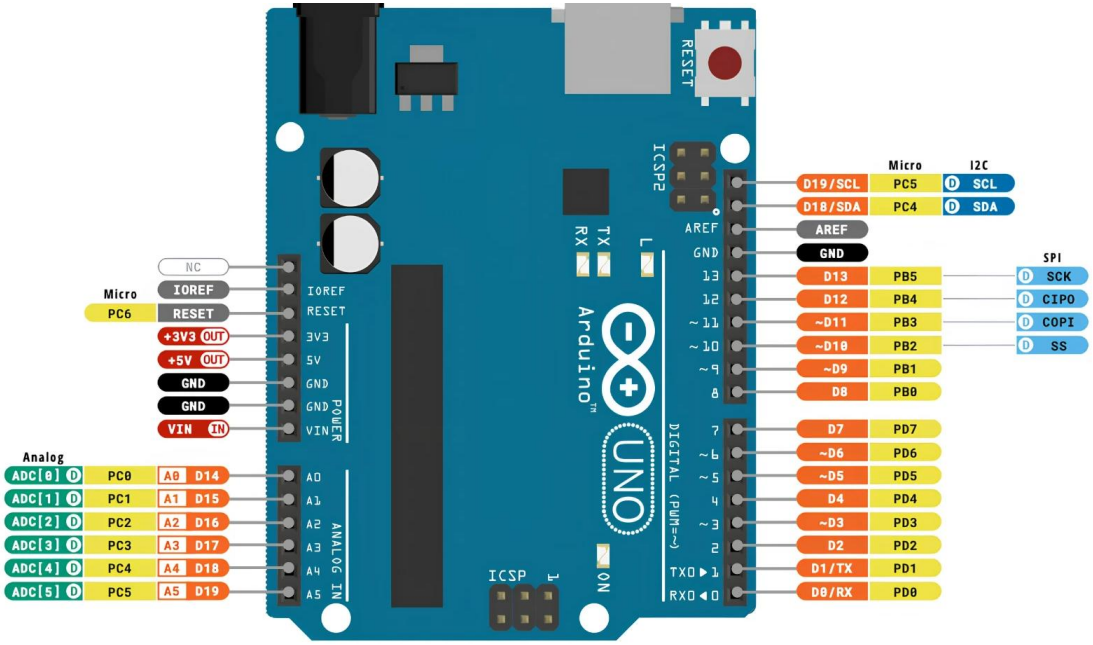
COM2?1, COM2?0	Output Compare pin (assume WGM22=0)
00	Don't use Pin
01	Don't use Pin
10	**Set Pin on CTR=0x00, Clear pin on match=OCR2A/B**
11	**Clear Pin on CTR=0x00, Set pin on match=OCR2A/B**

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Problem

- The LCD shield utilizes most of the pins associated with the internal PWM
- Only Timer 2's PWM signals are available.
  - **We will use Port B, bit 3 (i.e. D11)**



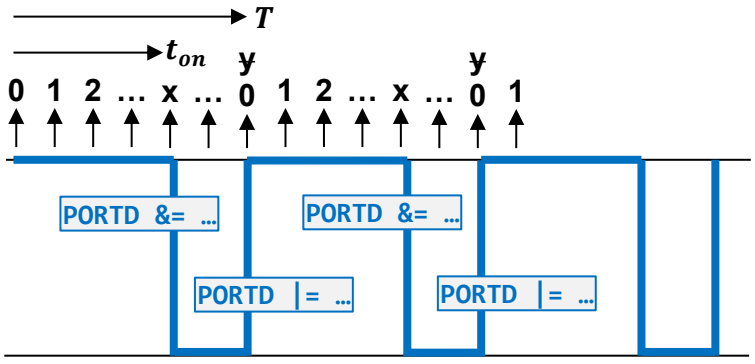


Timer + Channel A/B	Associated Output
Timer 0, Ch. A	Port D, bit 6
Timer 0, Ch. B	Port D, bit 5
Timer 1, Ch. A	Port B, bit 1
Timer 1, Ch. B	Port B, bit 2
Timer 2, Ch. A	Port B, bit 3
Timer 2, Ch. B.	Port D, bit 3

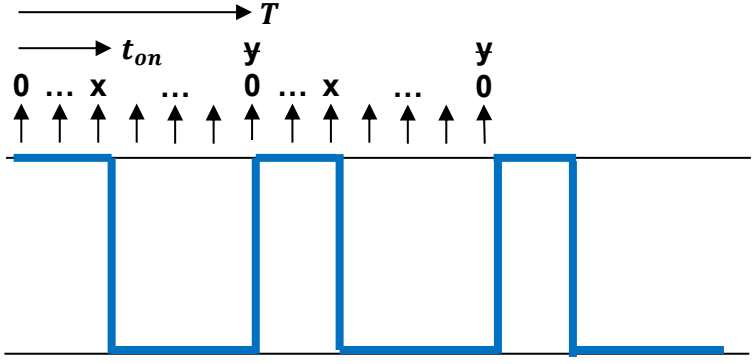
© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

"Timer + Software" PWM

- Rather than using the timers in **PWM** mode to automatically flip the output at desired times…
- …we could implement PWM in "software" by using a timer to generate a periodic interrupt (as we have in many earlier labs) and "count" and "threshold" variables to determine when to manually flip an output that should act as a PWM signal



PWM Output 1

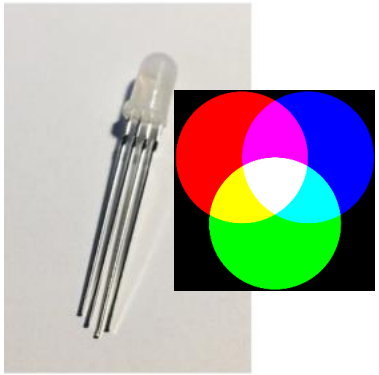


PWM Output 2

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Lab Overview

- This lab will introduce use of :
  - An RGB LED that can produce many different colors when combined with the hardware PWM of your Arduino



  - A servo motor



  - Combinational and sequential logic chips to offload some processing to hardware (and not software)

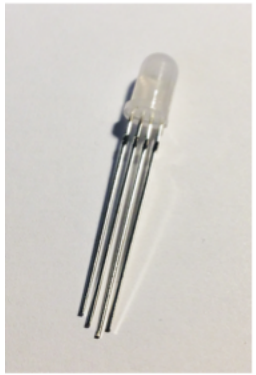


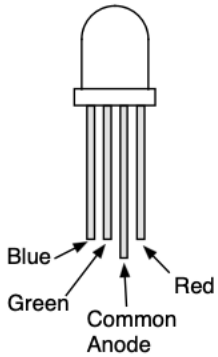
© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

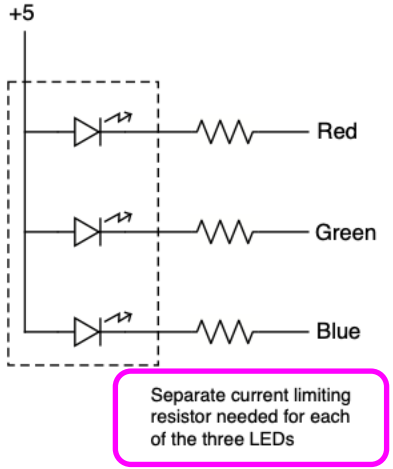
This Photo by Unknown Author is licensed under CC BY-SA

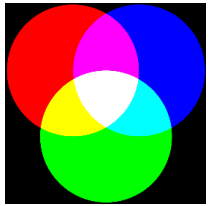
RGB LED

- You will build your own controllable PWM system to produce varying combinations of intensities of red, green, or blue light from an RGB LED
  - While normally we would control all 3 colors at the same time, our system will produce varying intensities on only 1 color at a time but allow you to turn the other LED's on or off to produce many combinations









The RGB LEDs consist of three separate LEDs in one package. Each color has a separate cathode (–) lead, and all the the anode (+) leads are connected together.

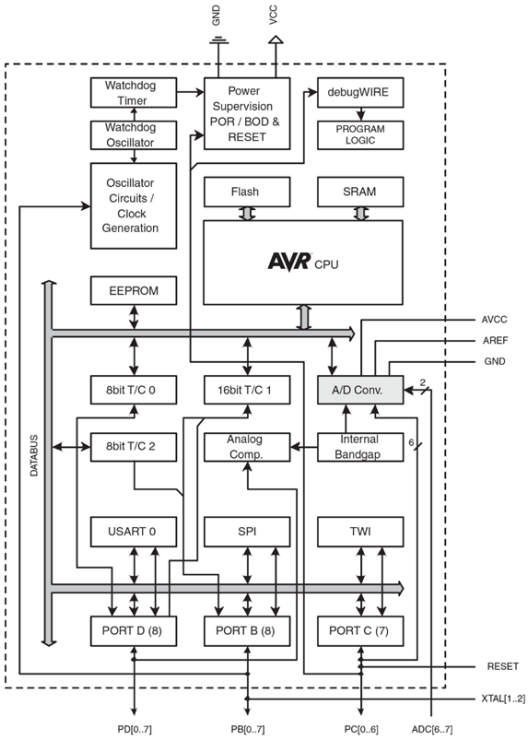
This Photo by Unknown Author is licensed under CC BY-SA

© 2025 by Mark Redekopp. This content is protected

Hardware/Software Lab

- We will use combinational and sequential components to:
  - Offload software tasks from the Arduino.
    - In many embedded systems there will be too much processing and control for software to keep up with
  - Illustrate how to save I/O pins by using external hardware
  - Illustrate how the internal Arduino hardware operates

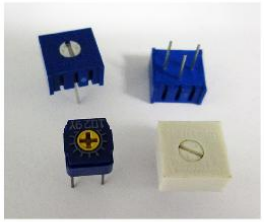




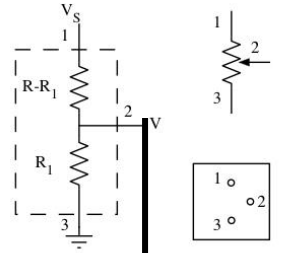
© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Analog Inputs

- Continue to use the potentiometer (variable resistor) that produces a changing voltage as you rotate it
- The LCD shield has 5 buttons



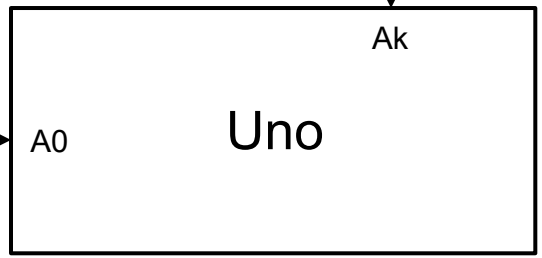
Potentiomers or "pots"



Functional diagram of a potentiometer, schematic symbol, and diagram of pin layout.



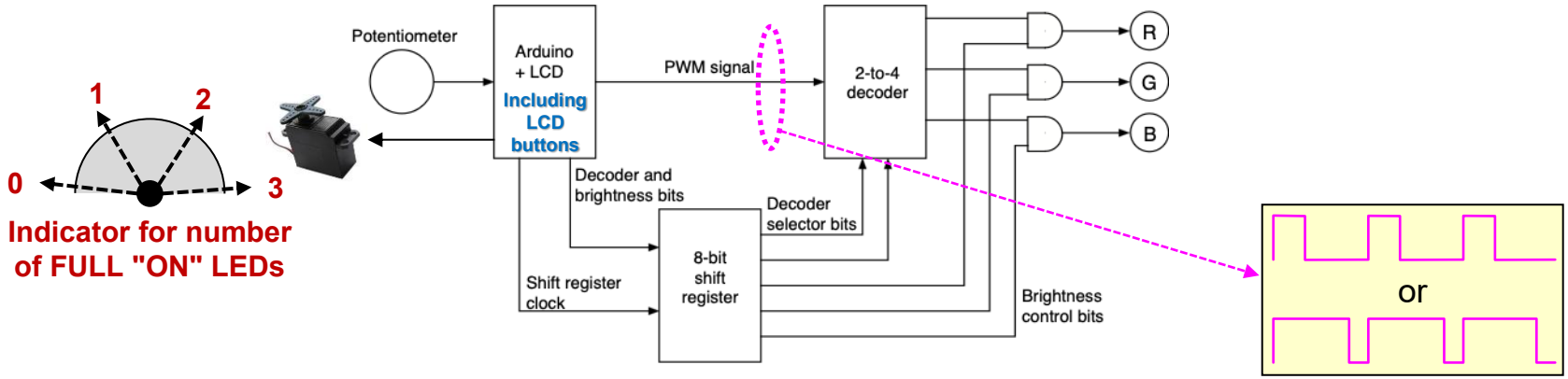
5 Button Inputs => 1 analog voltage



© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Overall System Description

- You will use a "potentiometer" dial to produce a continuous range of voltages that you will then digitize to the range 0-255.
- That digitized input will control the duty cycle (on-time) of a PWM signal that your Arduino will generate.
- Using buttons on the LCD and external hardware, you will then be able to route (demultiplex) that PWM signal to any of the 3 light segments (R, G, B) while also being able to turn segments FULL ON or FULL OFF to produce a variety of colors.
- A servo motor to "point" to a specific angle that indicates how many LEDs are FULL ON [0 to 3]
- You'll save pins by using only 2 outputs to control the various selections of light segments and which color uses the PWM signal.

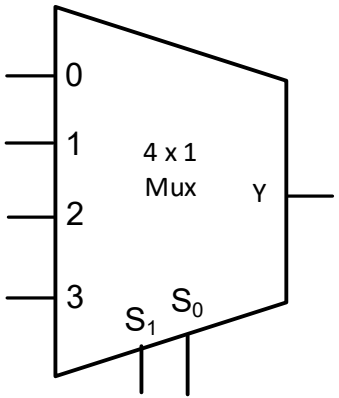


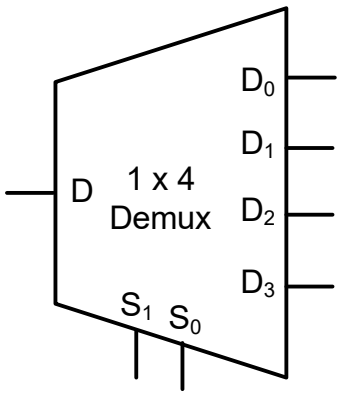
Block diagram of circuit to control 3 LEDs

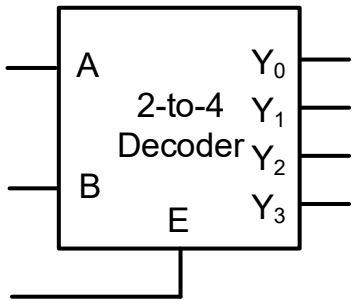
© 2025 by Mark Redekopp. This content is protected and n

Demuxes = Decoders

- A demultiplexer does the opposite job as a mux: passes the 1 input to the selected output
- It turns out a demux is EQUIVALENT to a decoder







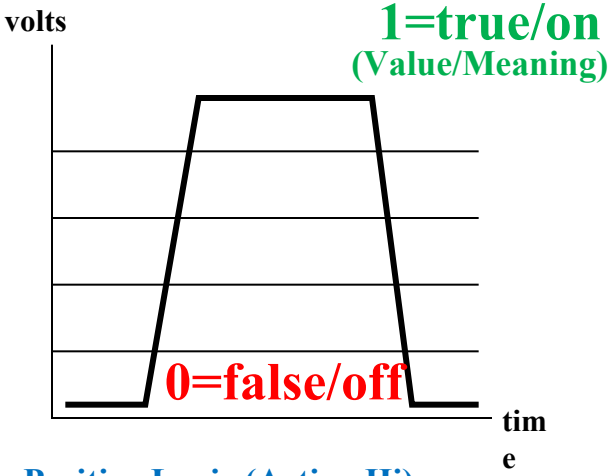
© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

NEGATIVE (ACTIVE-LO) LOGIC

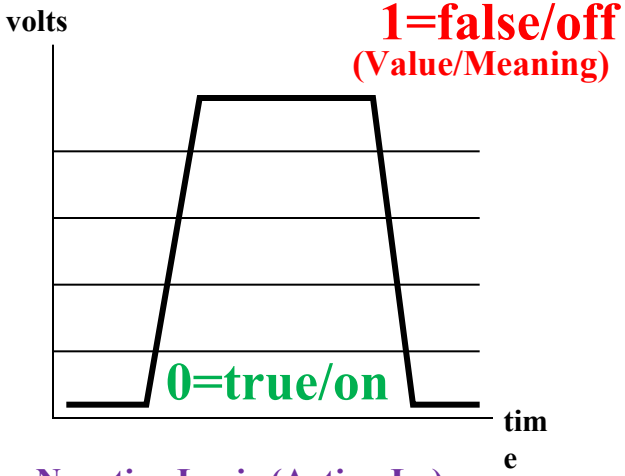
© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Negative (Active-Lo) Logic

- Recall it is up to us humans to assign **MEANING** to the **TWO** voltage levels our digital circuits produce and process
  - Thus, far we've (unknowingly) used the positive logic convention where:
    1 means **true** and 0 means **false**
  - In negative logic,
    0 means **true** and 1 means **false**



Positive Logic (Active-Hi) Convention



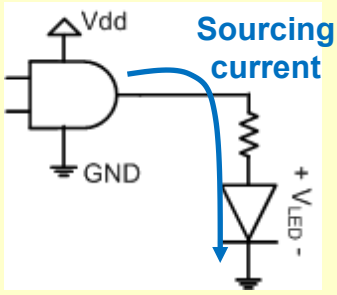
Negative Logic (Active-Lo) Convention

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Why Active-low

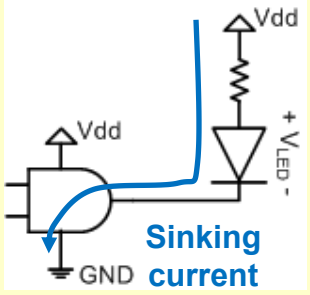
- Some digital circuits are better at either
  - "Sinking" (draining/sucking) electric current
  - Or, "sourcing" (producing) current

Active-hi output



LED is on when gate outputs '1'

Active-low output

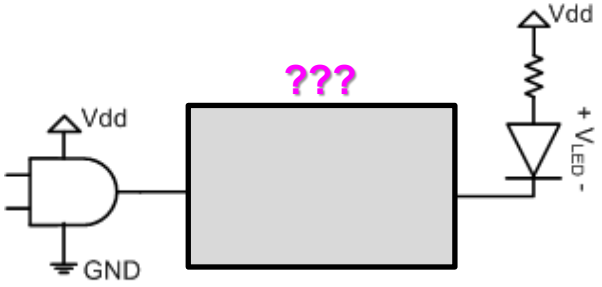


LED is on when gate outputs '0'

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Converting Between Active-hi and low

- Active-hi convention
  - 1 = on/true/active
  - 0 = off/false/inactive
- Active-low convention
  - 0 = on/true/active
  - 1 = off/false/inactive



Active-high (Produces '1' when we want to turn the LED on)

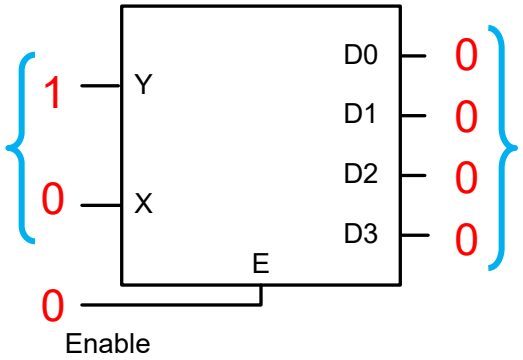
Active-low (Needs '0' to turn the LED on)

- As shown above, what if I had an active-high output and wanted to connect it to an active-low input?
- To convert between conventions
  - _____

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

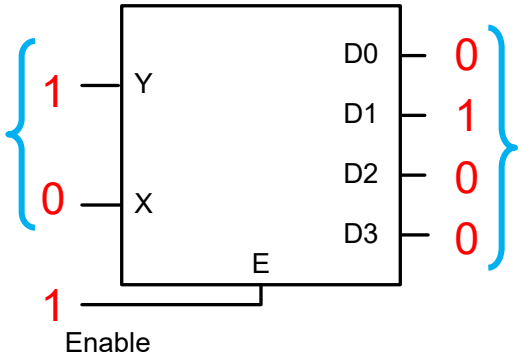
Enables

When E=0, inputs is ignored



Since E=0, all outputs = 0

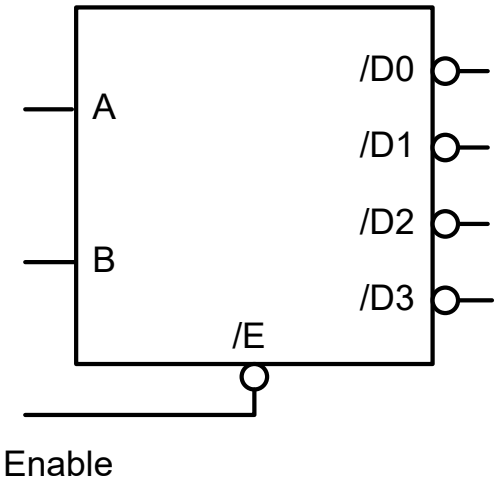
When E=1, inputs will cause the appropriate output to go active



Since E=1, outputs will function normally

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Decoder w/ Active Low Enable and Outputs

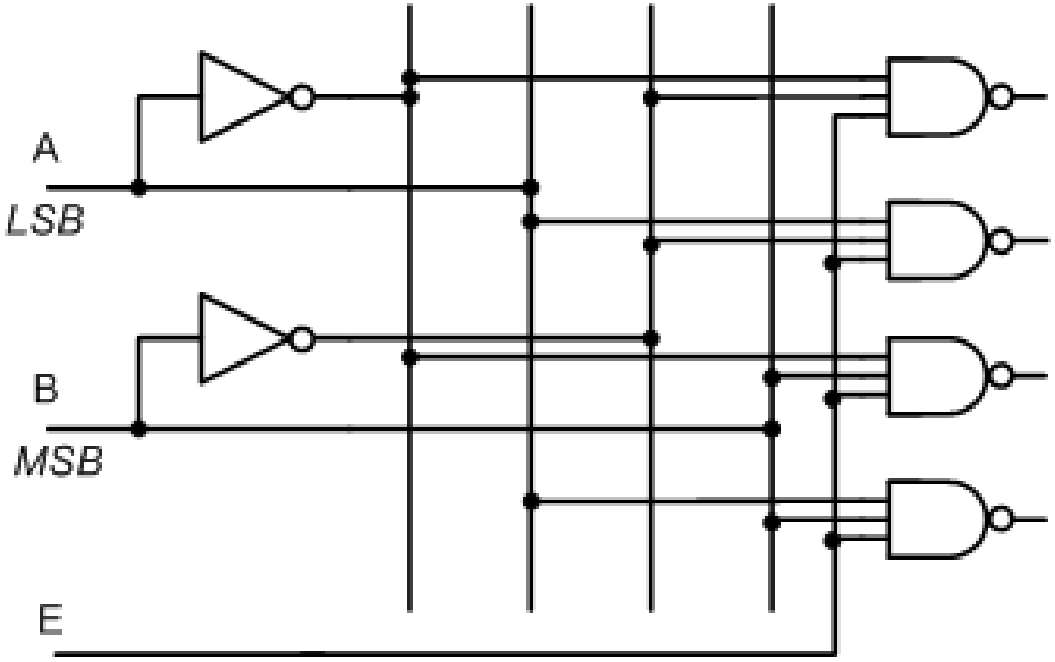


Inputs and outputs that have a "true/false" or "yes/no" meaning (e.g. enable or decoder outputs) are often candidates to be active-low.

Bubbles and signals starting with a slash '/' indicate an active-low input or output…not an inverter…the inverters are actually in the logic diagram on the next pages…

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Active-Lo Outputs

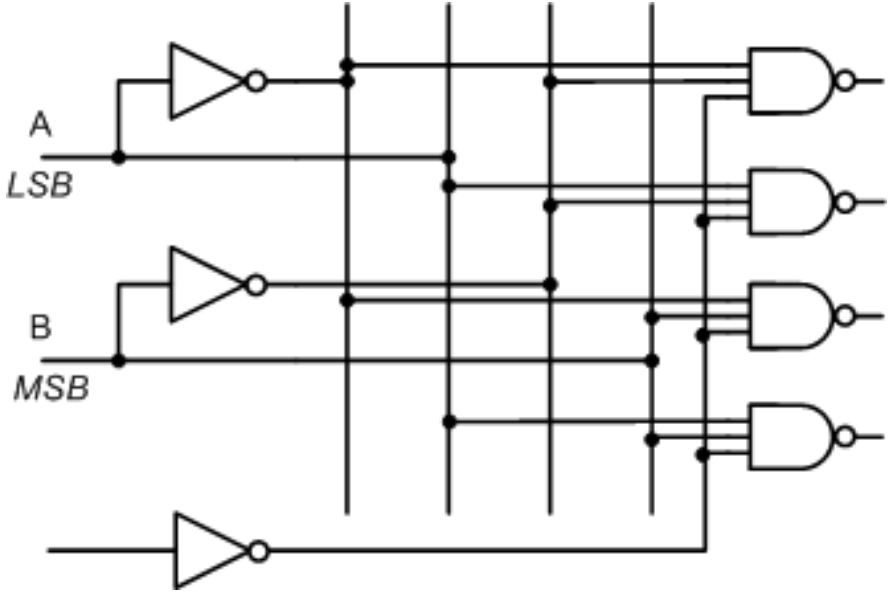


When E=inactive (inactive means 0), Outputs turn off (off means 1)

When E=active (active means 1), Selected outputs turn on (on means 0)

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Active-Lo Enable



When E=inactive (inactive means 1), Outputs turn off (off means 1)

When E=active (active means 0), Selected outputs turn on (on means 0)

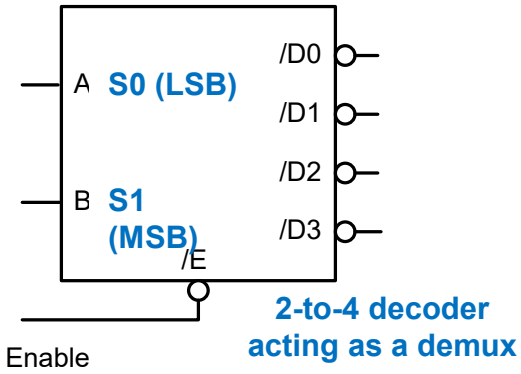
© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

COMPONENTS USED

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

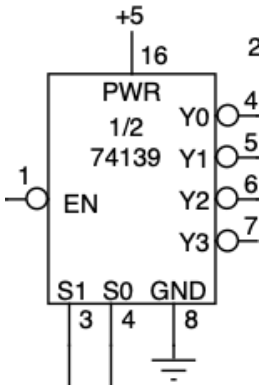
Decoder (Demux) Component

Components As Presented in Lecture



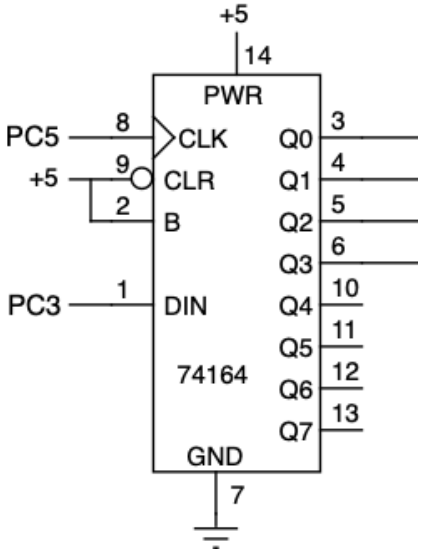
2-to-4 decoder acting as a demux

74LS139 Decoder/DeMux Component

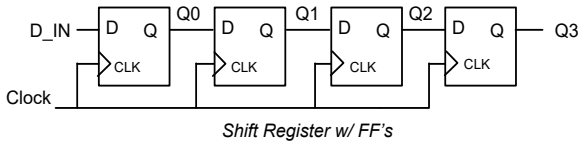


© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Shift Register Components



74LS164 Shift Register Component



Shift Register w/ FF's

Component As Presented in Lecture

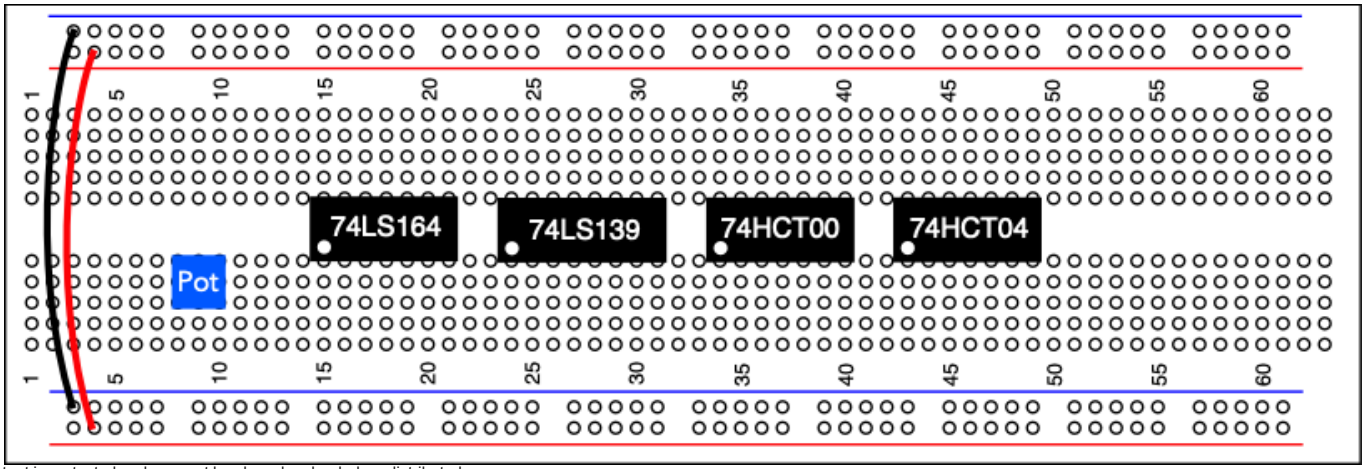
© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

TASKS

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Laying Out your Circuit Board

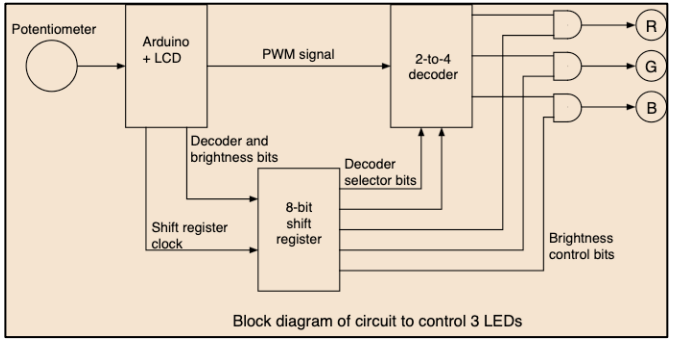
- You have the 74HCT00 (NAND gates) and 74HCT04 (inverters) in your kit. All others can be collected when you come to lab.
- Layout your chips as we show below to make your wiring task easier and so our staff can help you more quickly via common placement.



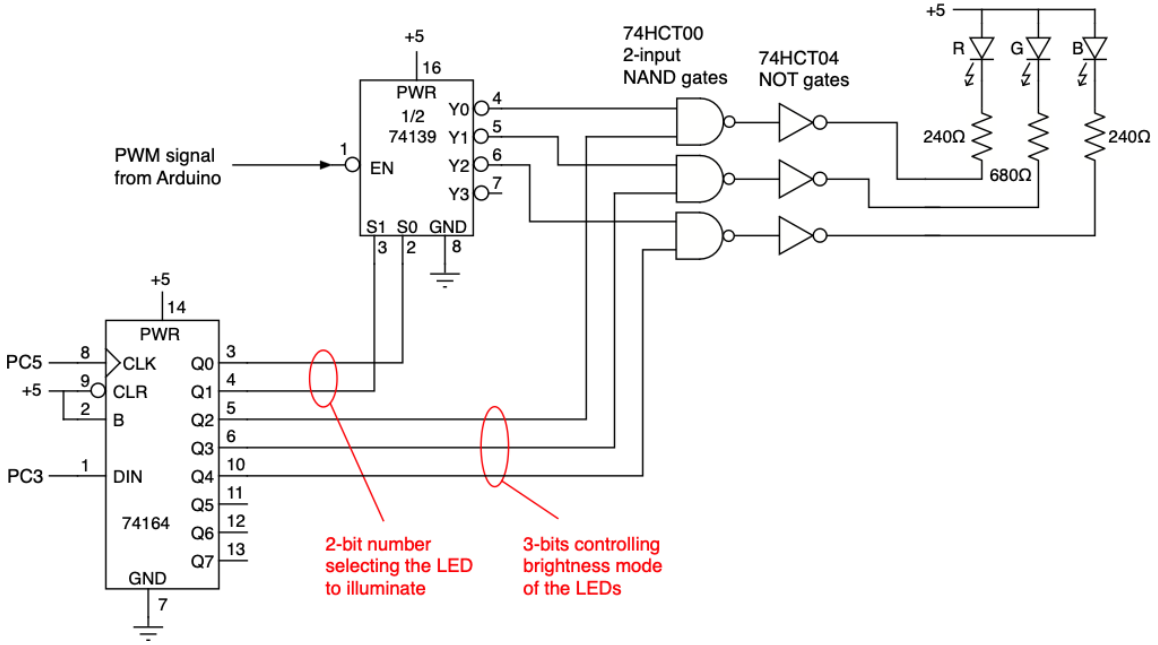
© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

LED Control Circuit

- Wire the remaining components

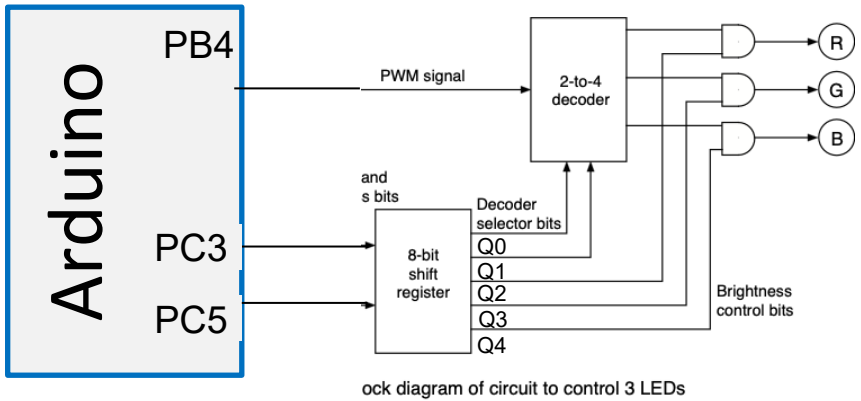


Block diagram of circuit to control 3 LEDs



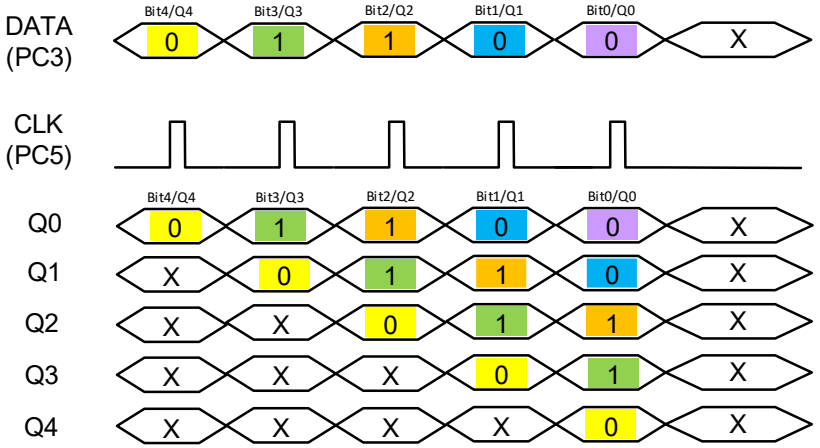
© 2025 by Mark Redekopp. This content is protected and

Software Task 1a



ock diagram of circuit to control 3 LEDs

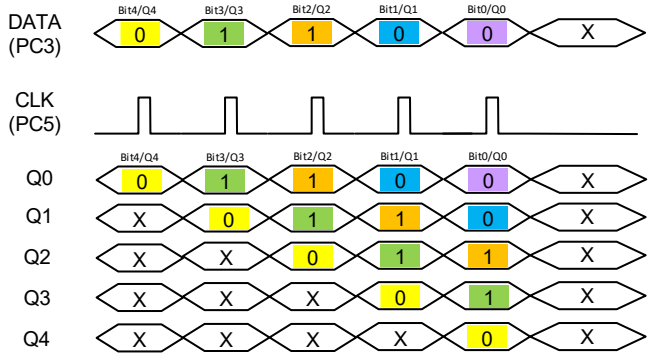
- Sending bits to the shift register is similar to your LCD interface
- Output 1 data bit at a time on PC3 (in the correct order) and then generate your own "clock" pulse on PC5 which will cause the shift register to capture the data bit and shift all the other outputs over by 1 location.
- Notice we can't change just 1 or 2 bits without affecting others (since they all shift). Thus, we must always send 4 bits, repeating any bits we don't want to change.



© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Software Task 1b

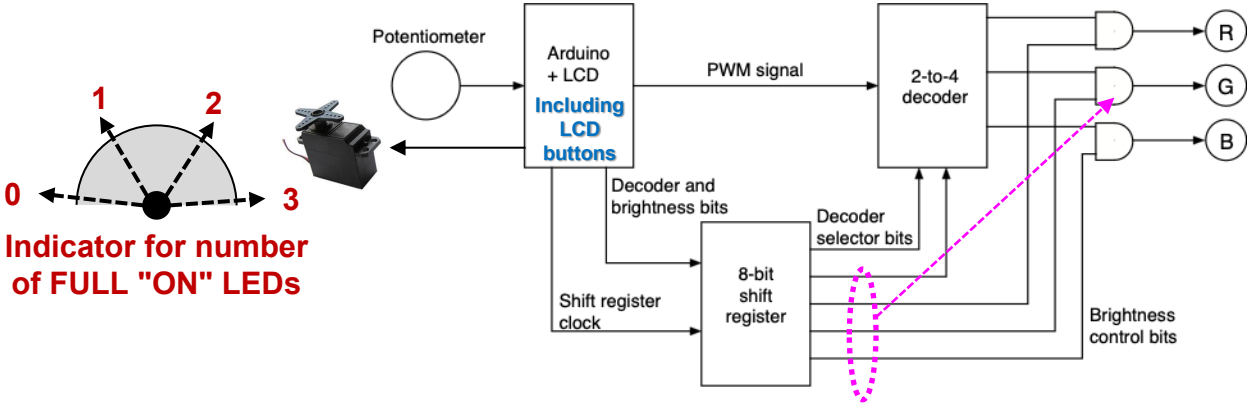
- Actual button interface and display software are written
- You need only write the functions to serially send the 4 mux and demux (decoder) select bits to the shift register
- You will write
  - `shift1bit(uint8_t bit);`
    - Sends the LSB of **bit** variable (as data on PC3) and generates a clock pulse on PC5
  - `void shift_load(uint8_t demux, uint8_t r, uint8_t g, uint8_t b);`
    - Call shift1bit() multiple times to send all the 5-bits; The demux's select bits are assumed to be in the **lower 2 bits** of the argument, while each of r, g, and b have the desired value in the **LSB** of the byte.
    - Take care to decide the order to send the 5 bits (refer to the schematic and look at how the shift register outputs are wired)



© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Servo Motor

- Recall, only one LED segment can be under control of the potentiometer
- But, we can turn any combination of the 3 LED segments FULL ON.
- Make the servo motor "point to" (rotate to) 1 of 4 possible angles to indicate how many LED segments are FULL ON.
- **Use software PWM to produce this signal**

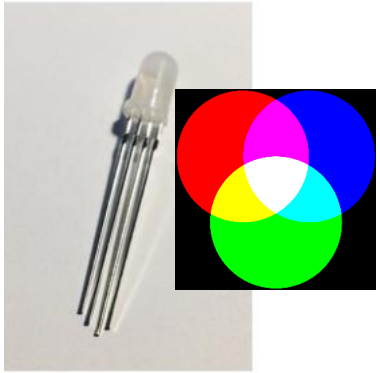


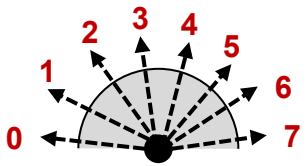
Block diagram of circuit to control 3 LEDs

© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

On Startup

- When embedded systems startup, it is common to perform some simple test sequence to ensure components are working (so that the user might recognize any failure)
- To that end, when your systems starts it should produce
  - Each of the 8 color combinations [black = all off, red, yellow (red + green), etc.]
  - Point the servo motor to an angle corresponding to 1/8 of the sector of rotation
  - Stay on each combination for 0.5 seconds





Indicator for number of FULL "ON" LEDs



© 2025 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.