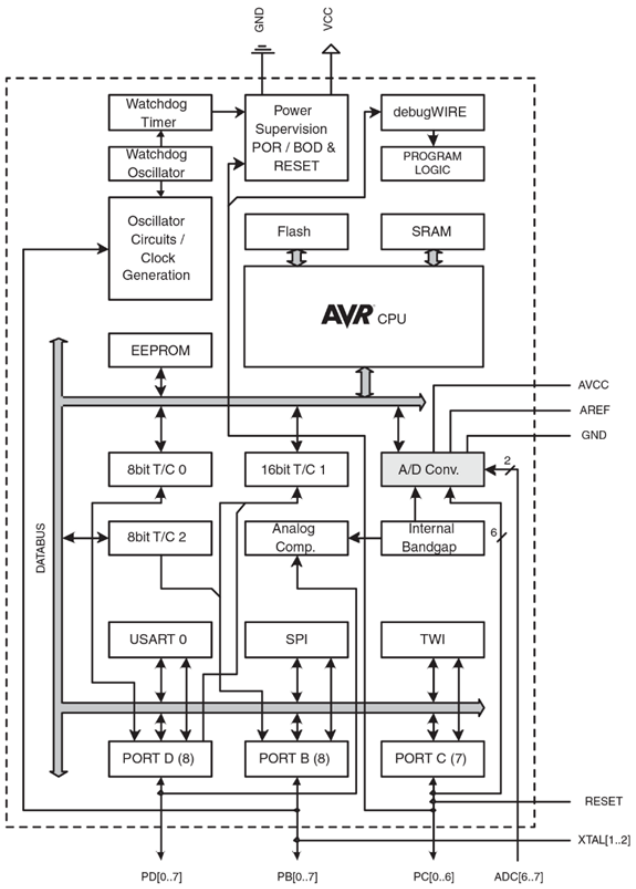
Hardware Datapath Components Lab

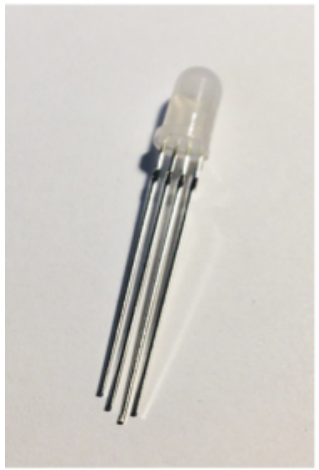
Hardware Lab

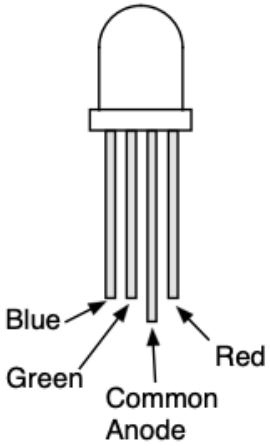
- We will use combinational and sequential components to:
  - Illustrate what is happening in the HW modules integrated on the Arduino
  - Offload software tasks from the Arduino.
    - In many embedded systems there will be too much processing and control for software to keep up with
  - Illustrate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

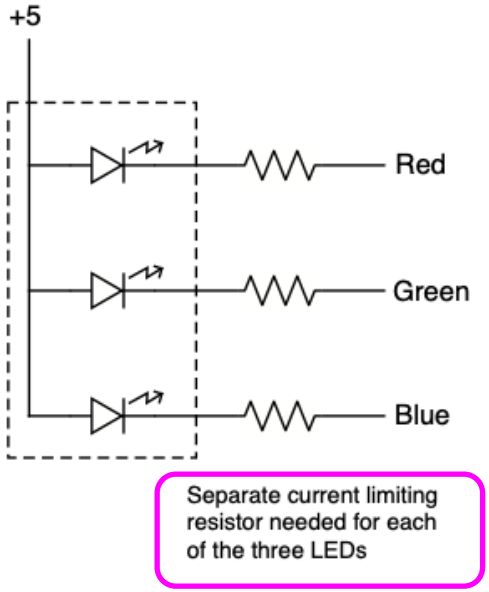


RGB LED

- You will build your own controllable PWM system to produce varying 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 and leave the other two colors off



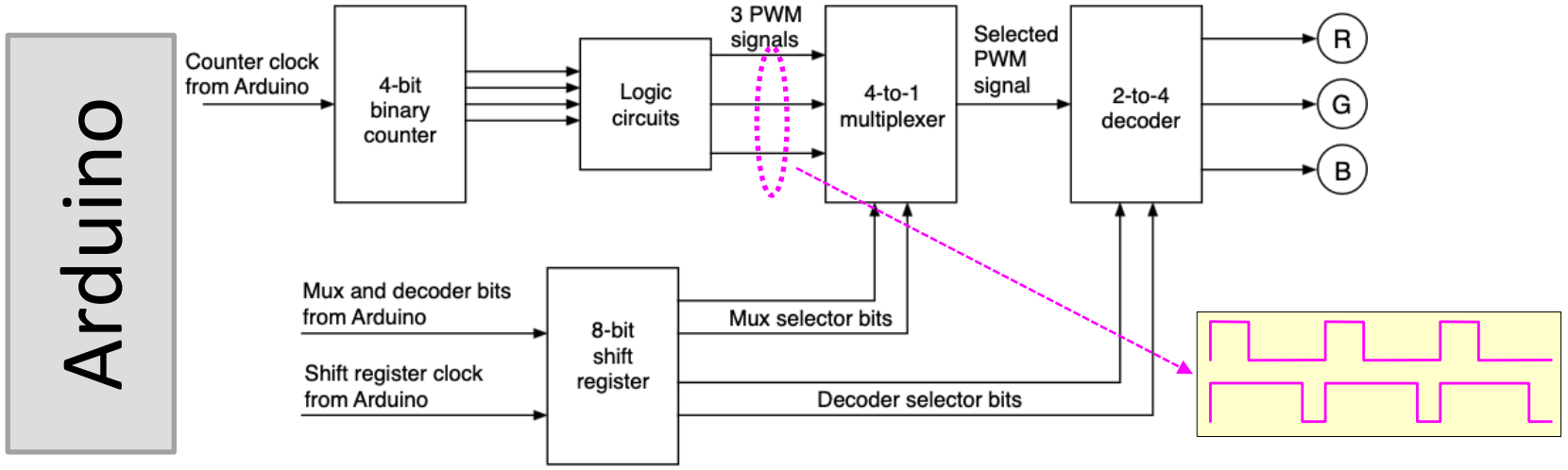




The RGB LEDs consist of three separate LEDs in one package. Each color has a separate cathode (–) lead, and all the the anode (+) leads are connected together.

Overall System Description

- You will produce various duty cycles (PWM) signals with a counter and custom logic (NAND gates), then choose the desired duty cycle with a mux and route it to the R, G, or B signal of the RGB LED with a demux (decoder)
- Which signal and color are selected is under control of software and output by a serial (1-bit) data output and clock.

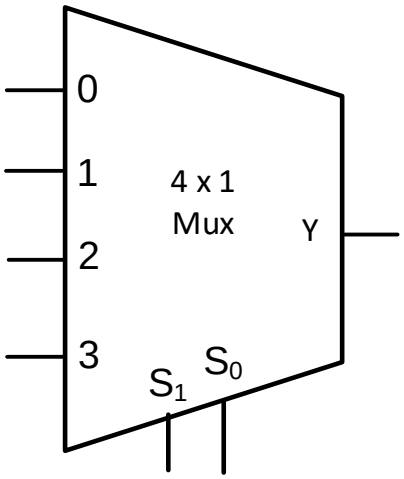


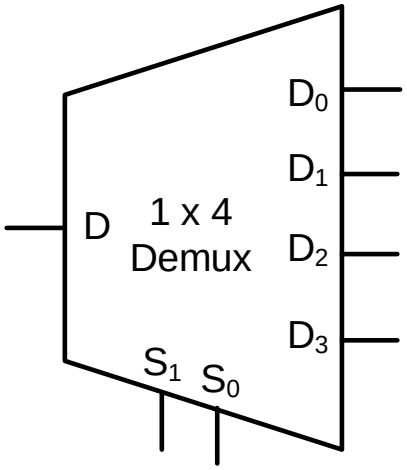
Block diagram of circuit to control 3 LEDs

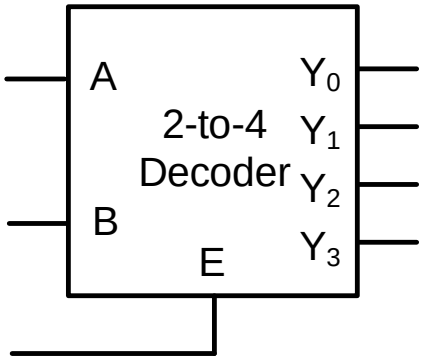
DEMULTIPLEXERS AND DECODERS

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



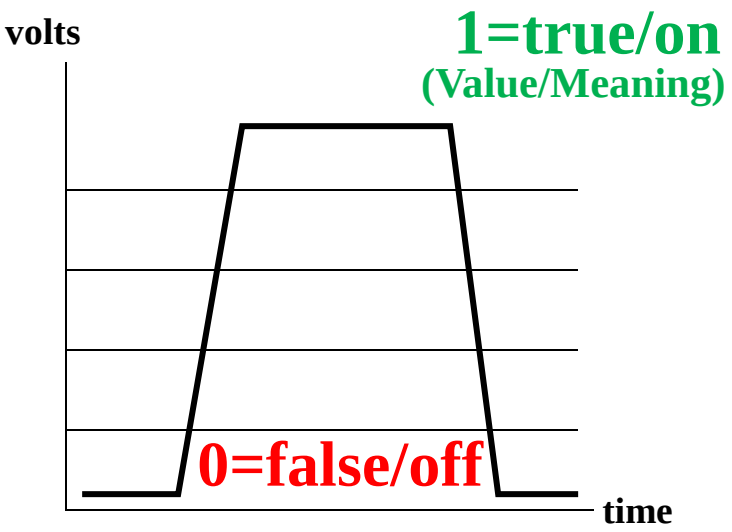




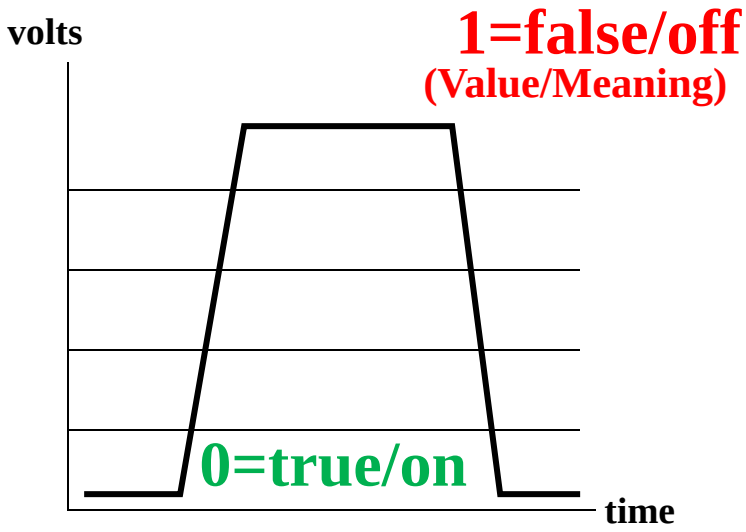
NEGATIVE (ACTIVE-LO) LOGIC

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

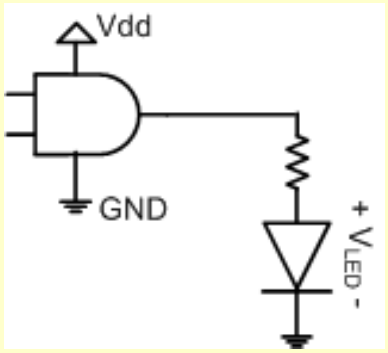


Negative Logic (Active-Lo) Convention

Why Active-low

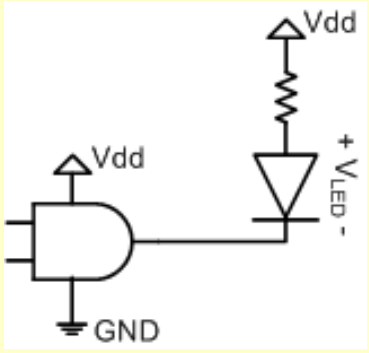
- Some digital circuits are better at “sinking” (draining/sucking) electric current than “sourcing” (producing) current

Active-hi output



LED is on when gate outputs '1'

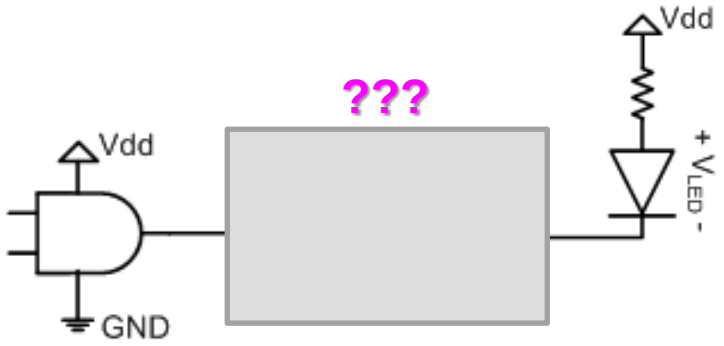
Active-low output



LED is on when gate outputs '0'

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



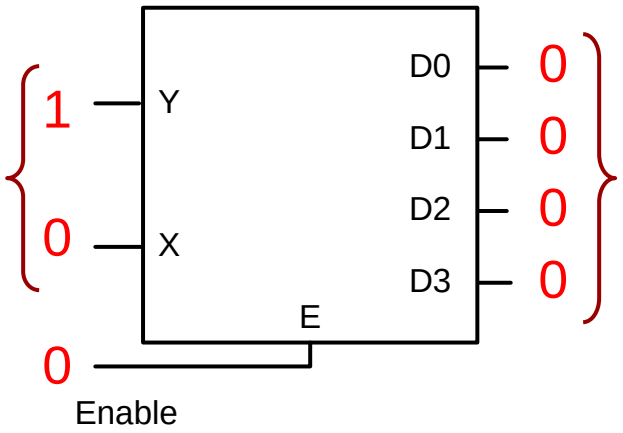
Active-high (Produces '1' when we want to turn the LED on)

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
  - _____

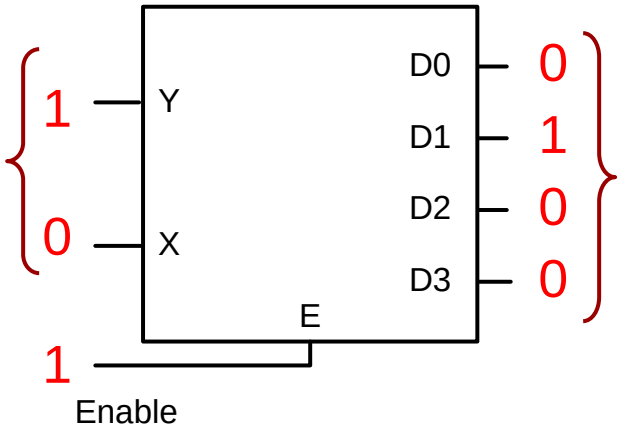
Enables

When E=0, inputs is ignored



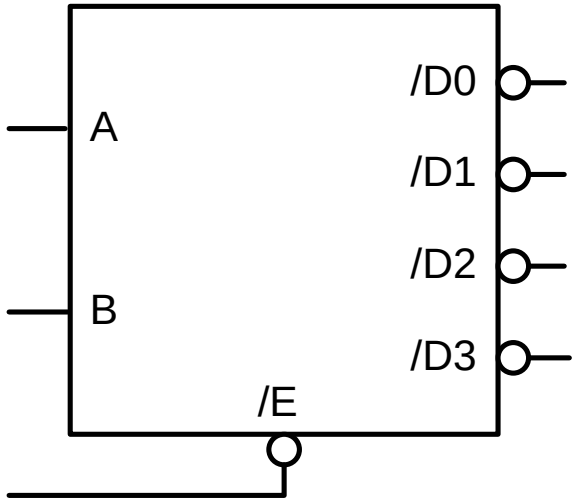
Since E=0, all outputs = 0

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



Since E=1, outputs will function normally

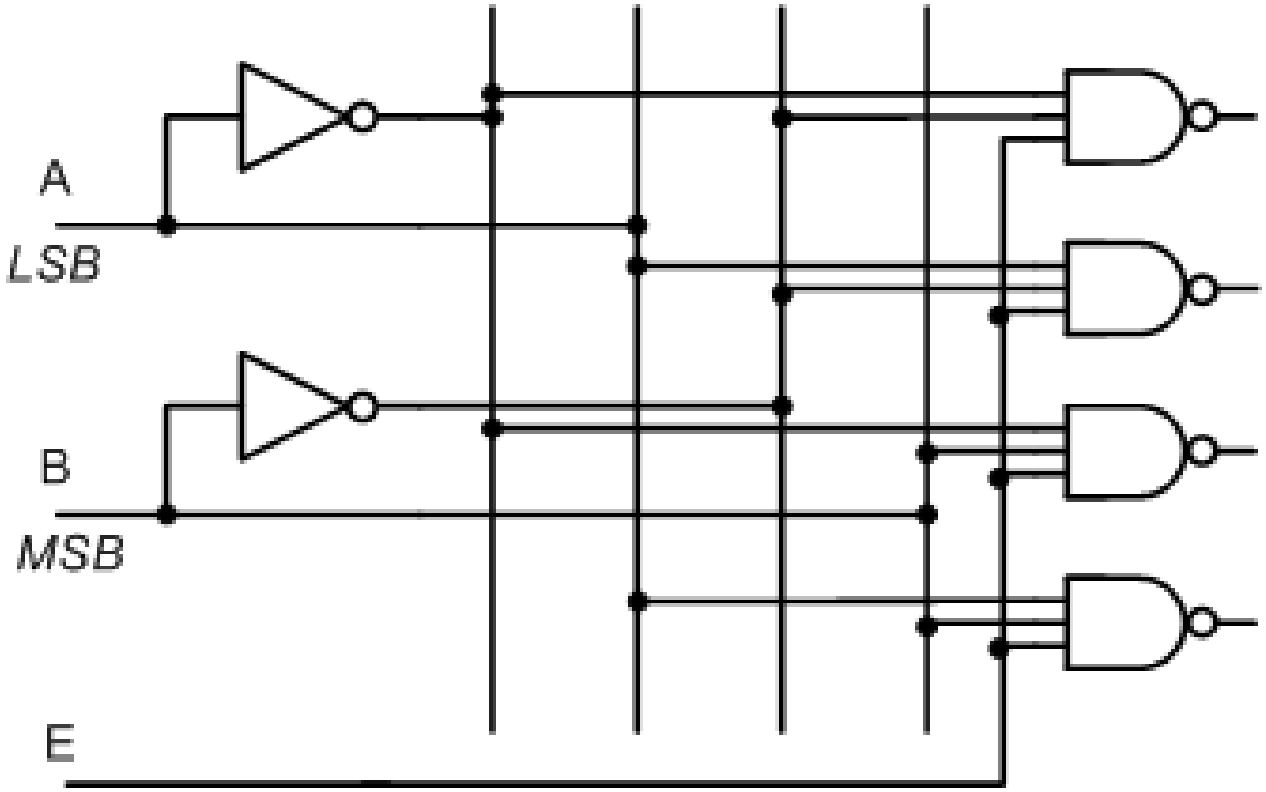
Decoder w/ Active Low Enable and Outputs



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…

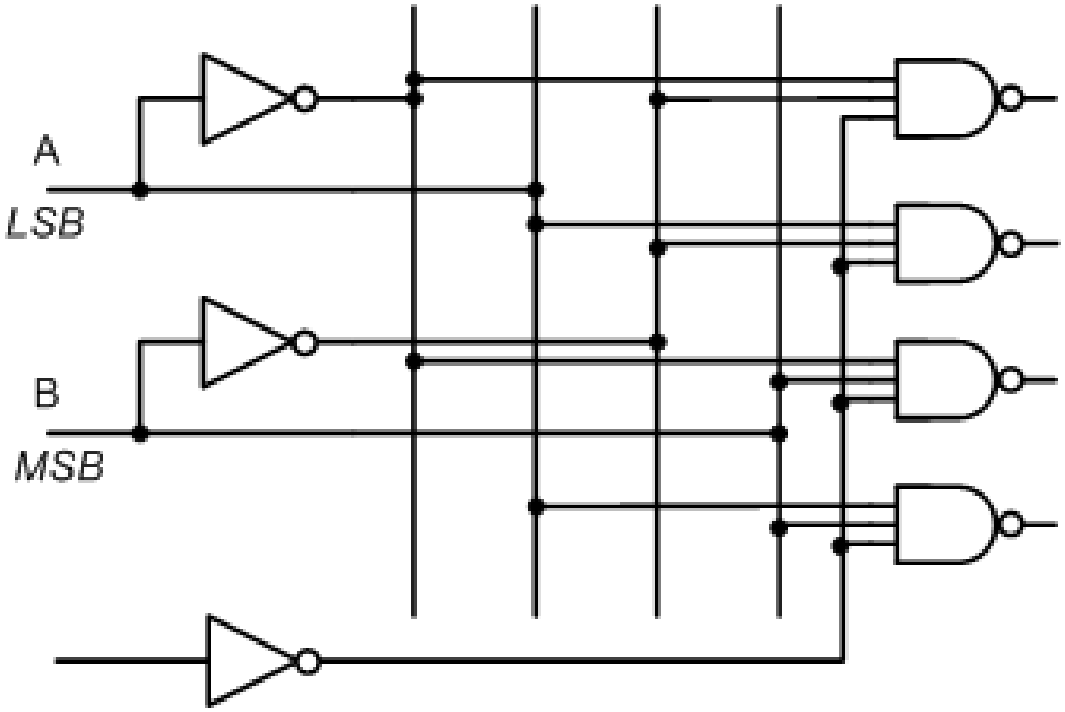
Active-Lo Outputs



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)

Active-Lo Enable

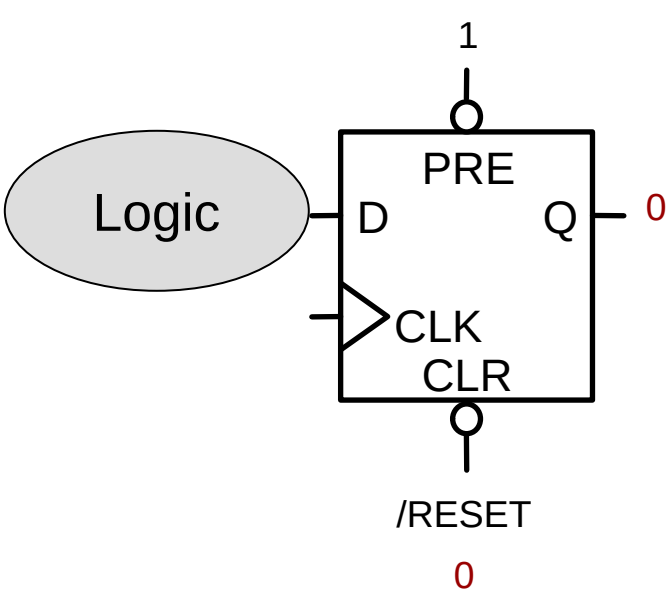


When E=inactive (inactive means 1), Outputs turn off (off means 1)

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

Active Low CLR and PRESET

- The reset signal might also be active low (0 = Reset, 1 = Normal operations)
- FFs can be made with active low /CLR & /PRE

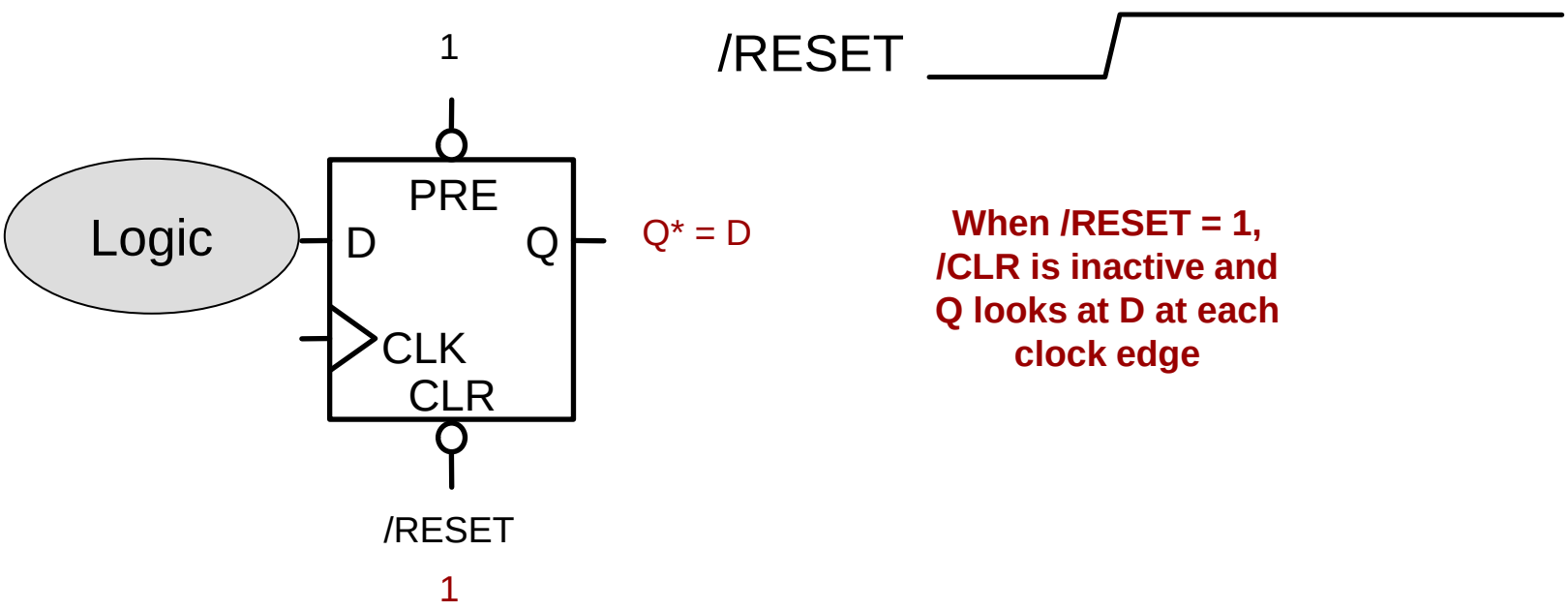


/RESET _____

When /RESET = 0, /CLR is activated and Q is forced to 0

Active Low CLR and PRESET

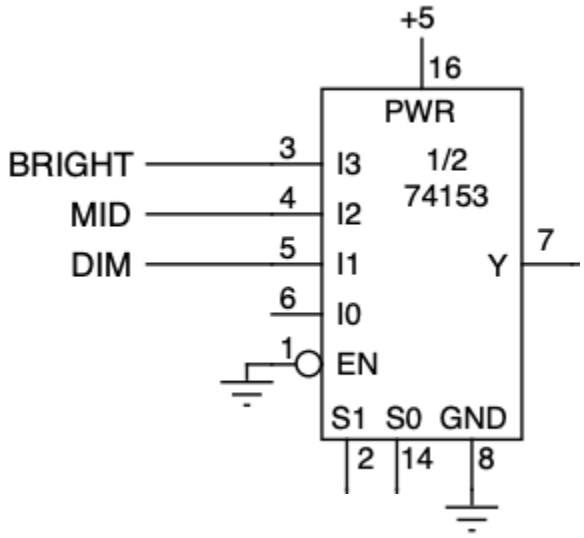
- Need to be able to initialize Q to a known value (0 or 1)



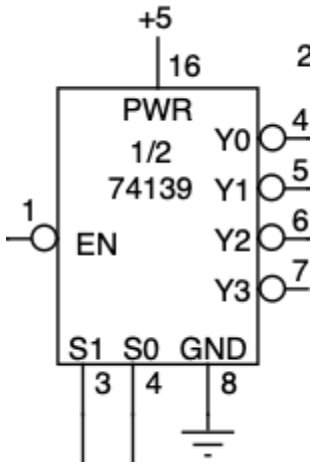
COMPONENTS USED

Mux and Decoder (Demux) Components

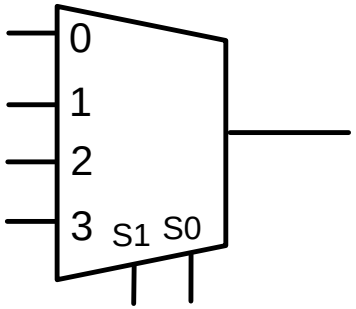
74LS153 Mux Component



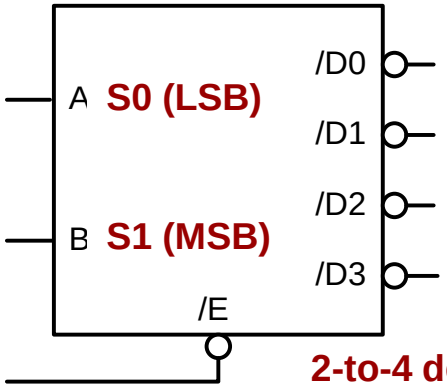
74LS139 Decoder/DeMux Component



Components As Presented in Lecture



4-to-1 mux



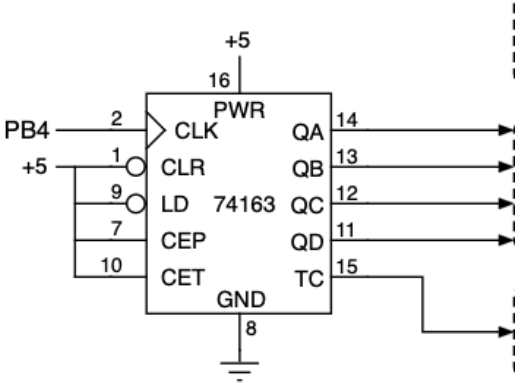
2-to-4 decoder acting as a demux

Counter and Shift Register Components

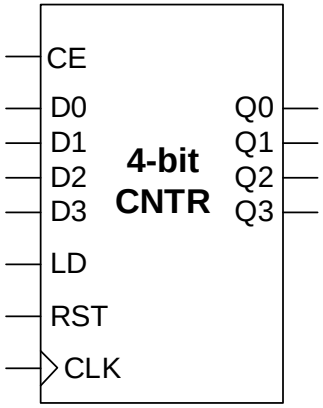
4-bit Counter

- CEP/CET = CE
- Active-low Load and Clear
- TC output
  - True only when QD-QA = 1111 (i.e. the max count before it overflows back to 0000)

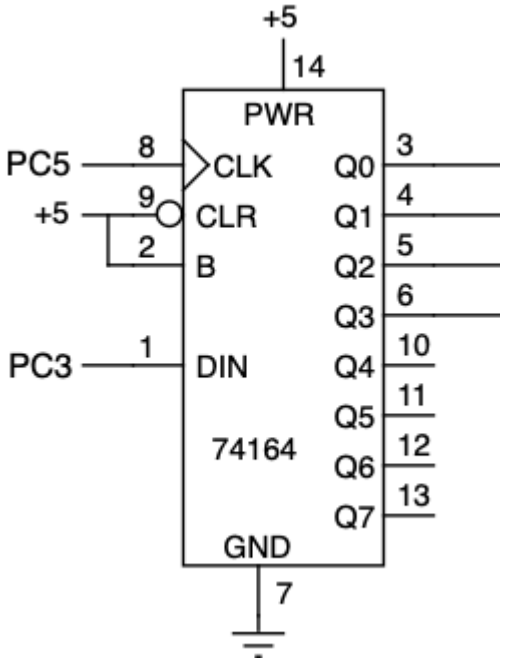
CLK	RST	LD	CE	Q*
0,1	X	X	X	Q
↑↑	1	X	X	0
↑↑	0	1	X	D[3:0]
↑↑	0	0	1	Q+1
↑↑	0	0	0	Q



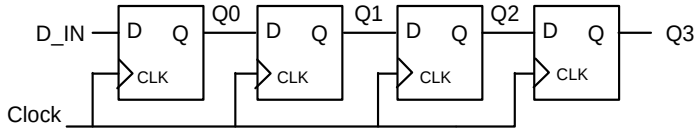
74LS163 Counter Component



Component As Presented in Lecture



74LS164 Shift Register Component



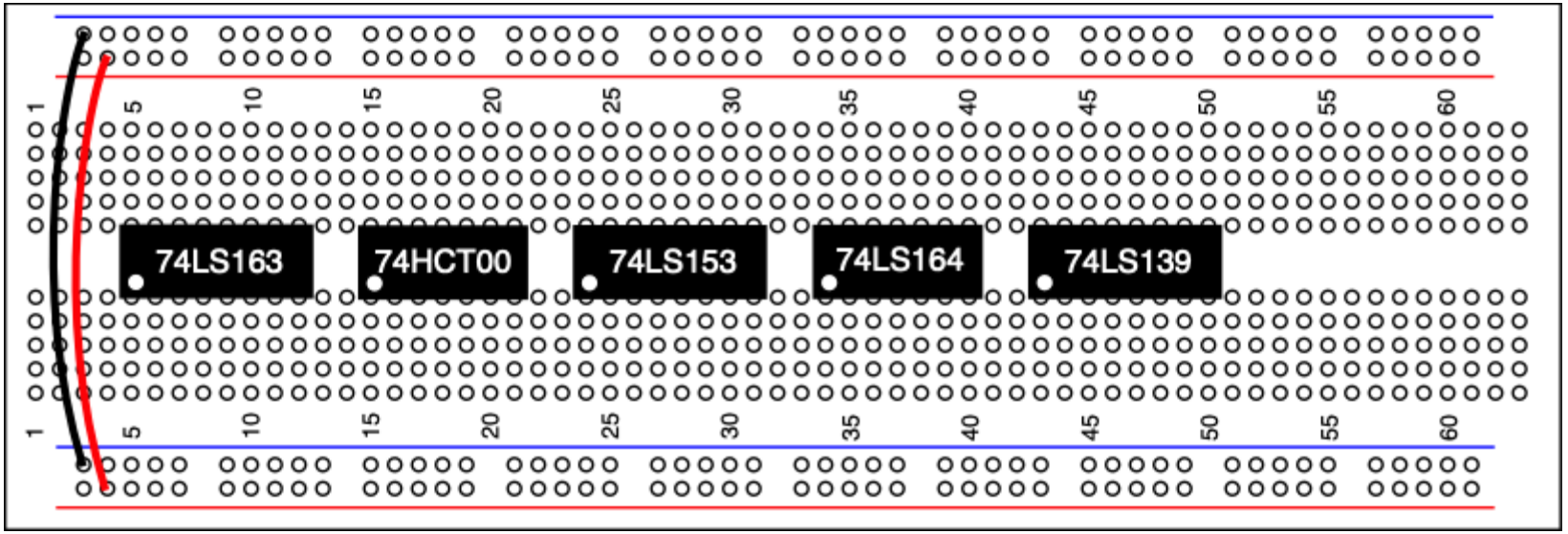
Shift Register w/ FF's

Component As Presented in Lecture

TASKS

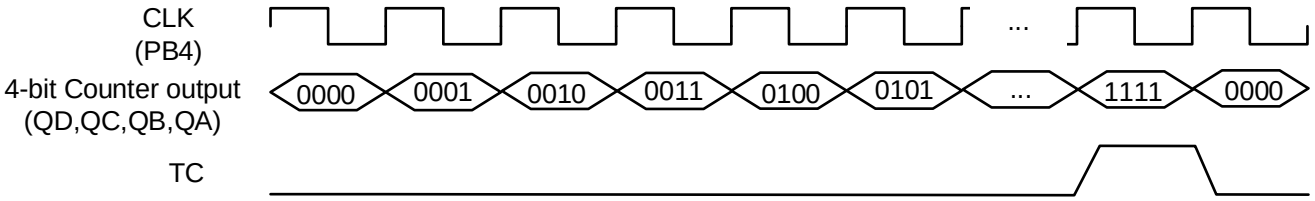
Laying Out your Circuit Board

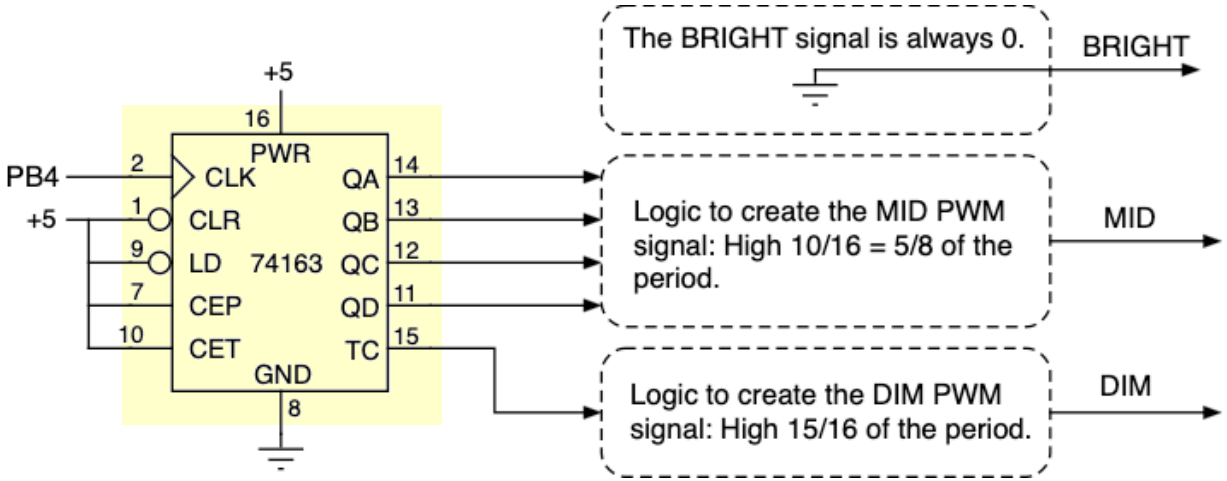
- You have the 74HCT00 in your kit. All others can be collected when you come to lab.
- Layout your chips as we show below to make your wiring task easier and so our staff can help you more quickly via common placement.



Circuit 1 – Counter Outputs

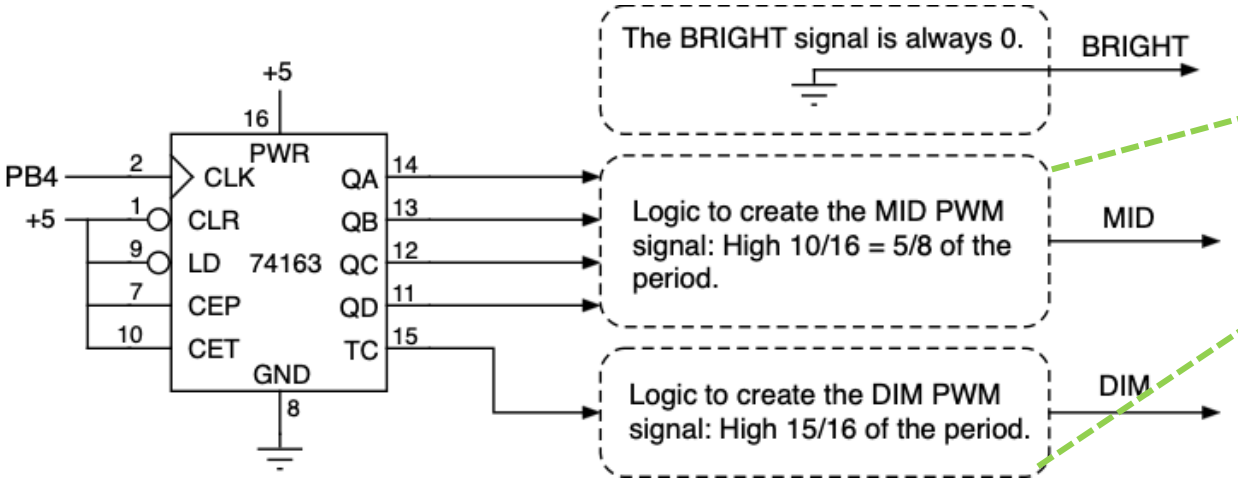
- Counter can generate all combinations of 4 bits which we can use as inputs to produce our desired PWM signal(s)

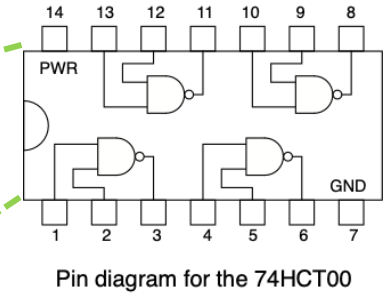




QD	QC	QB	QA	MID	TC
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

Circuit 1 – PWM Generation

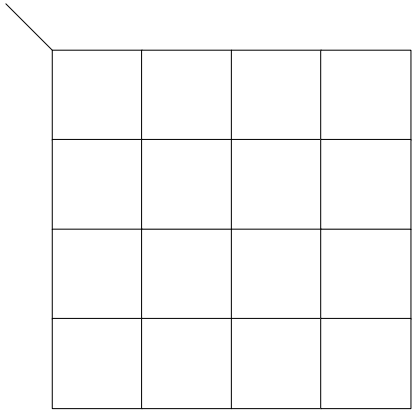




Pin diagram for the 74HCT00

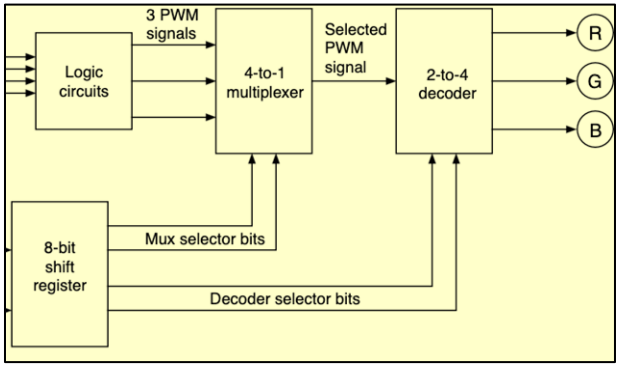
- We only have 4 2-input NAND gates to work with (no separate inverters, OR gates, etc.). DeMorgan's Theorem to the rescue!
- 1 NAND gate should be used to produce DIM from TC
- We can produce our MID circuit by choosing 10 consecutive input combinations to produce a 1
- But given our limitation of NO separate inverters and ONLY 4-input NAND gates we have to choose the 10 combinations that will produce the least inversions

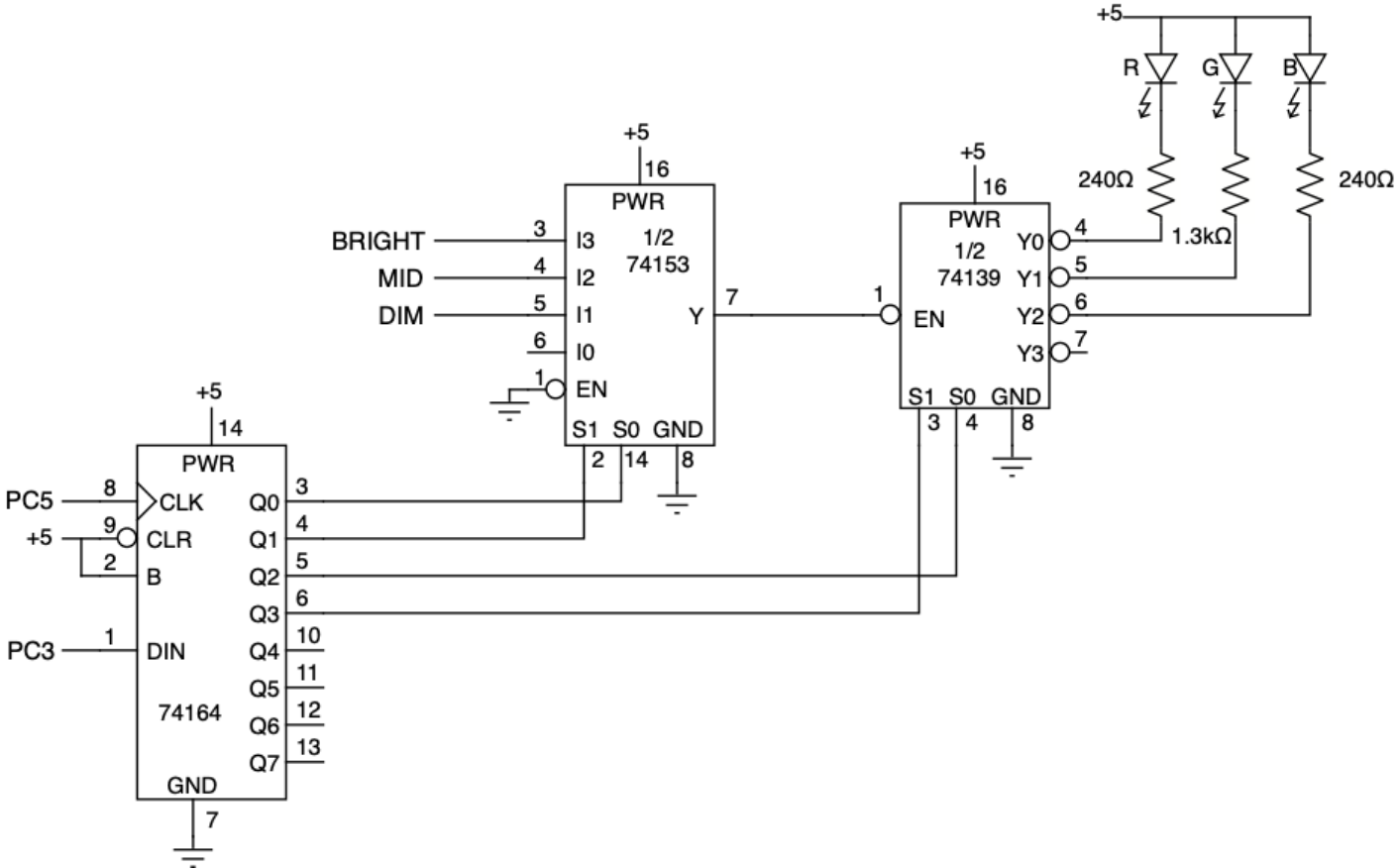
QD	QC	QB	QA	MID	TC
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		



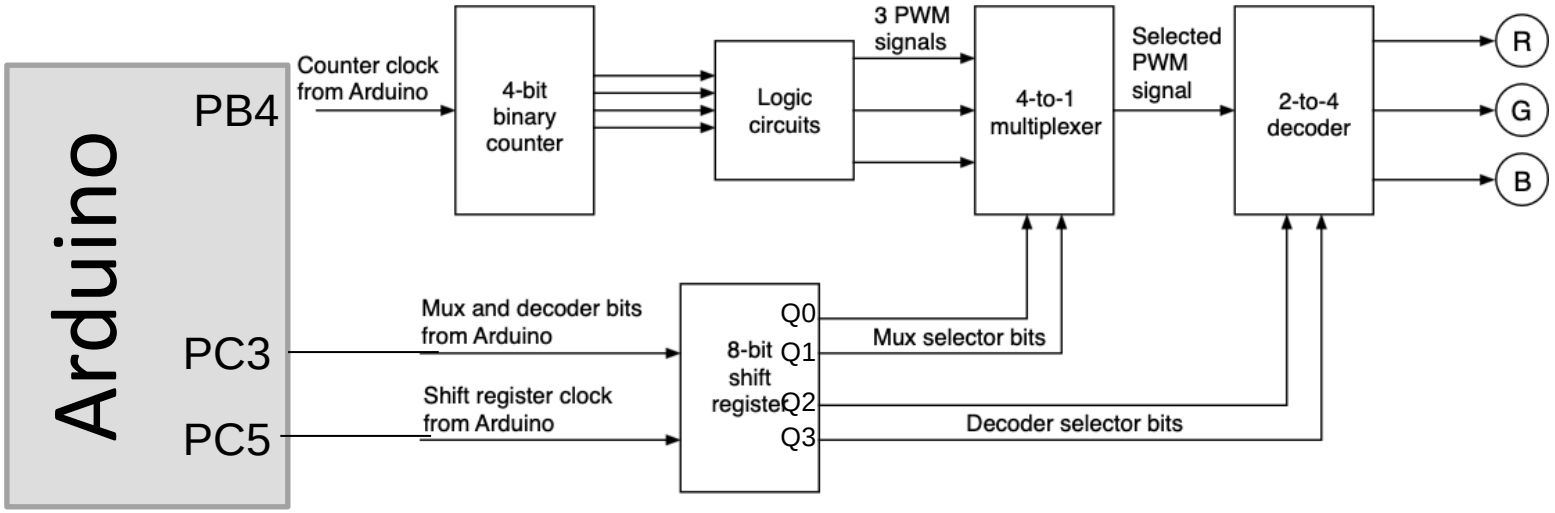
Circuit 2 – LED Control

- Wire the remaining components



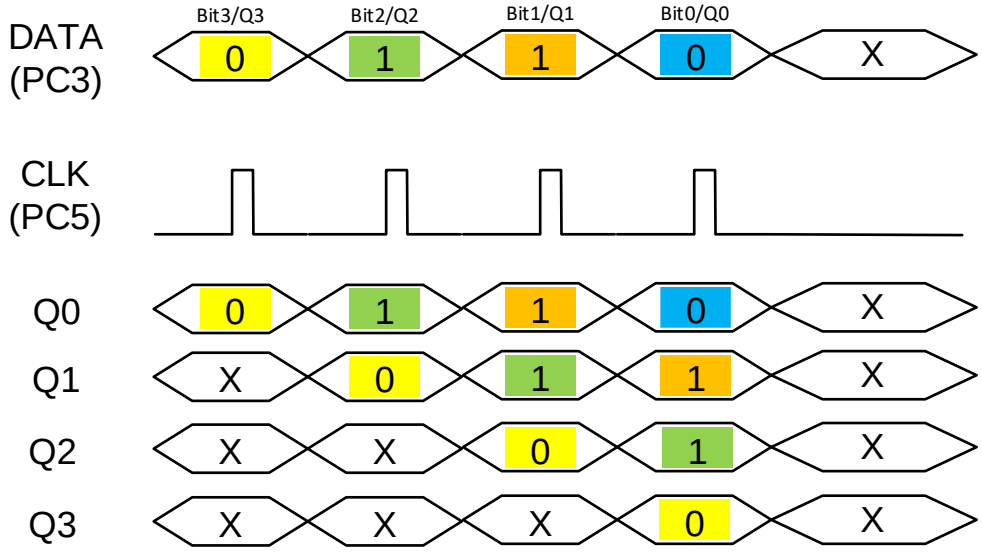


Software Task 1a



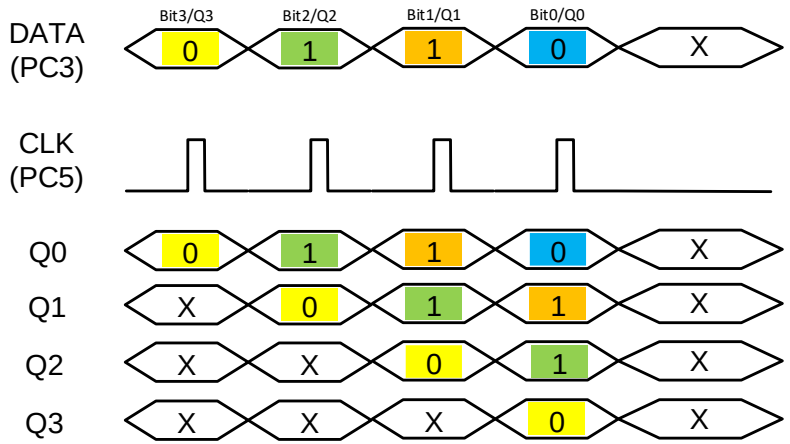
Block diagram of circuit to control 3 LEDs

- 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.



Software Task 1b

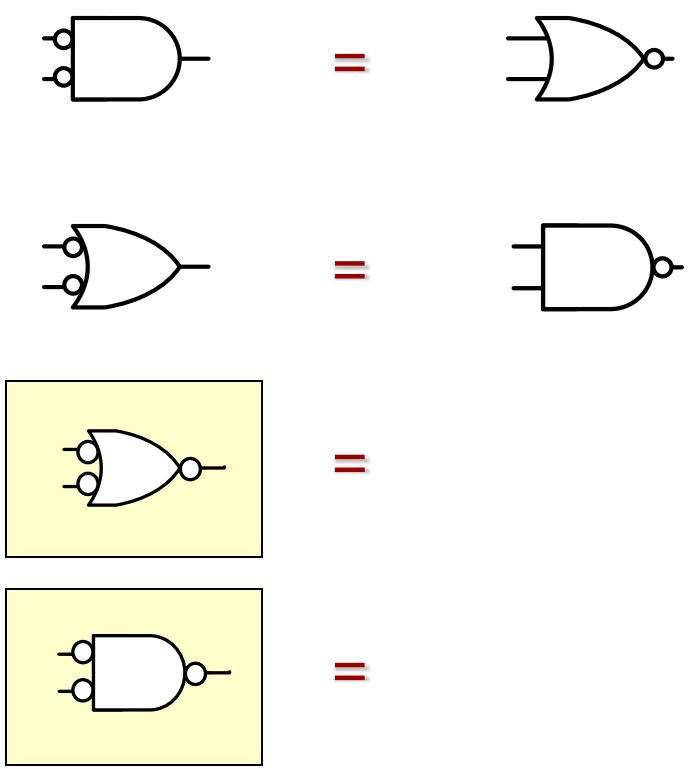
- Actual button interface and display software are written
- You need only write the functions to serially send the 4 mux and demux (decoder) select bits to the shift register
- You will write
  - `shift1bit(uint8_t bit);`
    - Sends the LSB of **bit** variable (as data on PC3) and generates a clock pulse on PC5
  - shift_load(uint8_t mux, uint8_t demux);
    - Sends the 4-bits of **mux** and **demux** select bits assuming the bits to send are in the **lower 2 bits** of each argument.
    - Take care to decide the order to send the 4 bits (refer to the schematic and look at how the shift register outputs are wired)



BACKUP

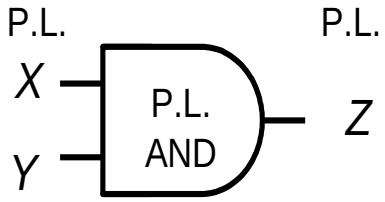
Duality and DeMorgan Equivalents

- If we used the active-lo convention, what would that do to our logic



Negative Logic 'AND' Function

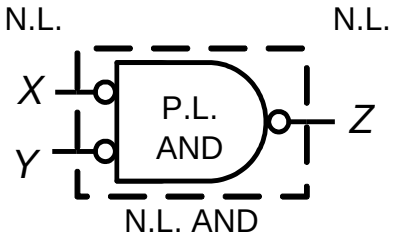
Traditional P.L. AND



X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Traditional AND gate functionality assumes positive logic convention

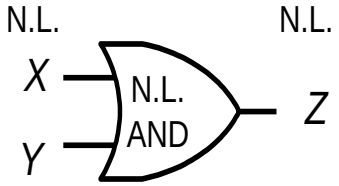
N.L. AND function



X	Y	Z
1	1	1
1	0	1
0	1	1
0	0	0

Given negative logic signals, we can invert to positive logic, perform the AND operation, then convert back to negative logic

N.L. AND = P.L. OR

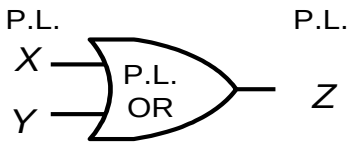


X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

However, we then see that an OR gate implements the negative logic 'AND' function

Negative Logic 'OR' Function

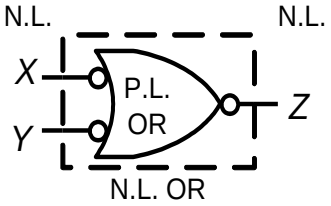
Traditional P.L. OR



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

Traditional OR gate functionality assumes positive logic convention

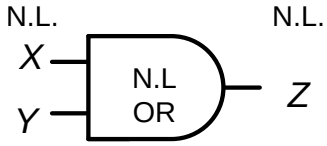
N.L. OR function



X	Y	Z
1	1	1
1	0	0
0	1	0
0	0	0

Given negative logic signals, we can invert to positive logic, perform the OR operation, then convert back to negative logic

N.L. OR = P.L. AND



X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

However, we then see that an AND gate implements the negative logic 'OR' function