

L12 & L13 Topics

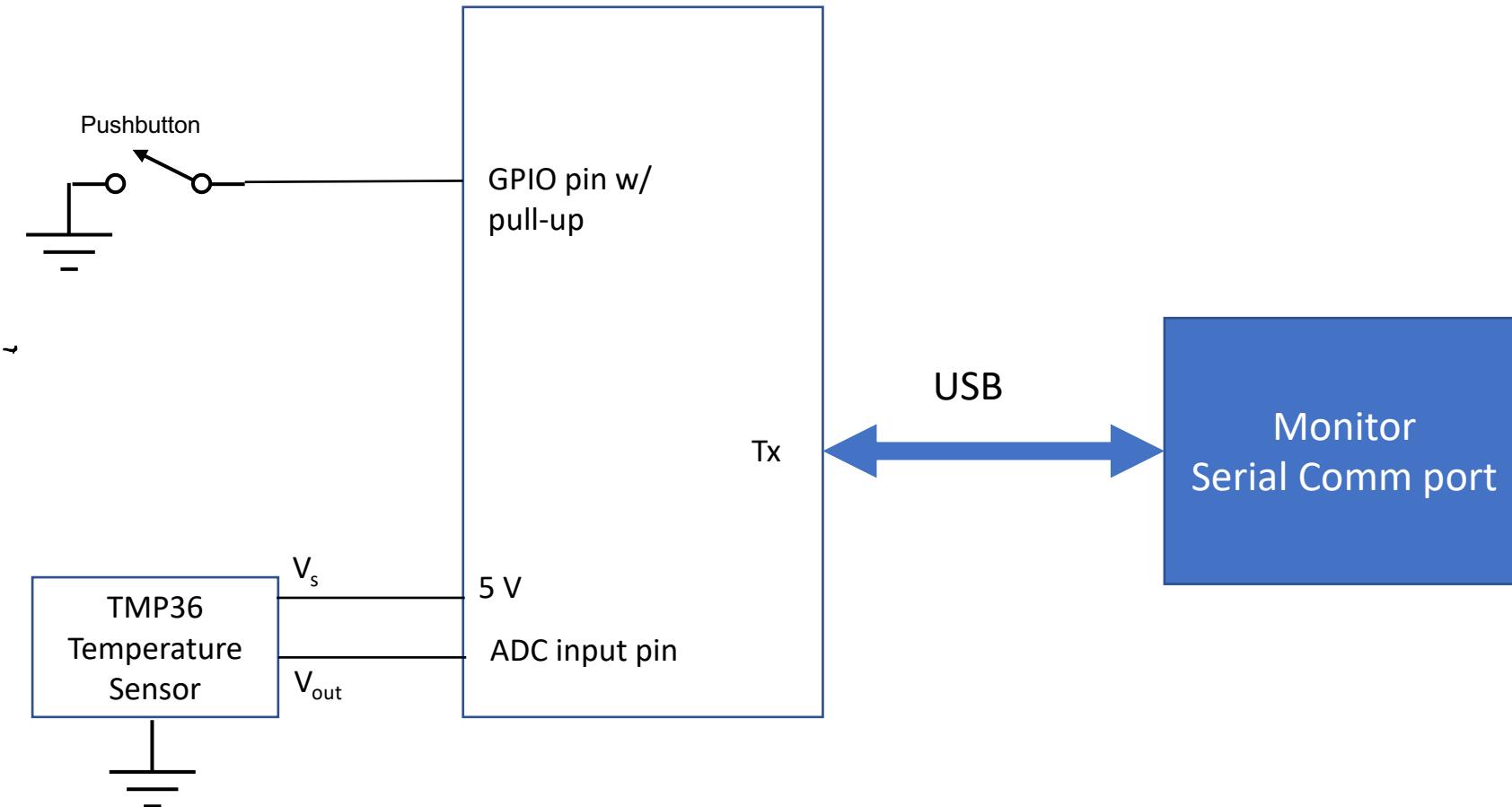
- Second half of the semester schedule
- Lab assignment 3
- UART & Comm port, revisited
- LED interfacing
- 4 digit 7 segment LED

14	Arduino Uno	3/21/22	Mon	L12	UART & LEDs	Lab 3 Assigned	
15	Arduino Uno	3/23/22	Wed	L13	UART & LEDs, continued		
16	Arduino Uno	3/28/22	Mon	L14	Analog to digital converter (ADC)		
17	Arduino Uno	3/30/22	Wed	L15	Lab3 Q&A		
18	ATmega328p breadboard	4/4/22	Mon	L16	ATmega328 on breadboard	Lab 4 Assigned	Lab 3 Due Sunday 4/3/22 11:59 pm
19	ATmega328p breadboard	4/6/22	Wed	L17	Timers & counters		
20	ATmega328p breadboard	4/11/22	Mon	L18	Motor Interfacing & PWM		
21	ATmega328p breadboard	4/13/22	Wed	L19	Lab4 Q&A		
		4/18/22	Mon		Holiday - no class		
22	ATmega328p breadboard	4/20/22	Wed	L20	External event triggers	Lab 5 Assigned	Lab 4 Due Monday 4/18/22 11:59 pm
23	ATmega328p breadboard	4/25/22	Mon	L21	Ultrasound Sensor		
24	ATmega328p breadboard	4/27/22	Wed	L22	OLED & I2C Interfacing		
25	ATmega328p breadboard	5/2/22	Mon	L23	Lab5 Q&A		
26	ATmega328p breadboard	5/4/22	Wed		Class wrap-up, review		Lab 5 due Wednesday 5/4/22 11:59 pm

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	L12 UART & LED Lab 3 Assigned 20	21	22	L13 UART & LED continued 23	24	25
	L14 Analog to Digital Converter 27	28	29	L15 Lab 3 Q&A 30	31	1
Lab 3 Due 11:59 pm	L16 bare 328P Lab 4 Assigned 3	4	5	L17 Timers & Counters 6	7	8
	L18 Motors & PWM 10	11	12	L19 Lab 4 Q&A 13	14	15
	No Class Lab 4 Due 11:59 pm 17	18	19	L20 External Events Lab 5 Assigned 20	21	22
	L21 Ultrasound Sensor 24	25	26	OLED & I2C Interfacing 27	28	29
	L23 Lab 5 Q&A 1	2	3	Last Class Lab 5 due 11:59 pm 4		

Lab Assignment #3 – Build a digital thermometer that displays ambient temp in C or F. One version displays temp (eg 98.6F or 37.0C) to serial comm port.

use to switch
between C and
F temperature
scale display



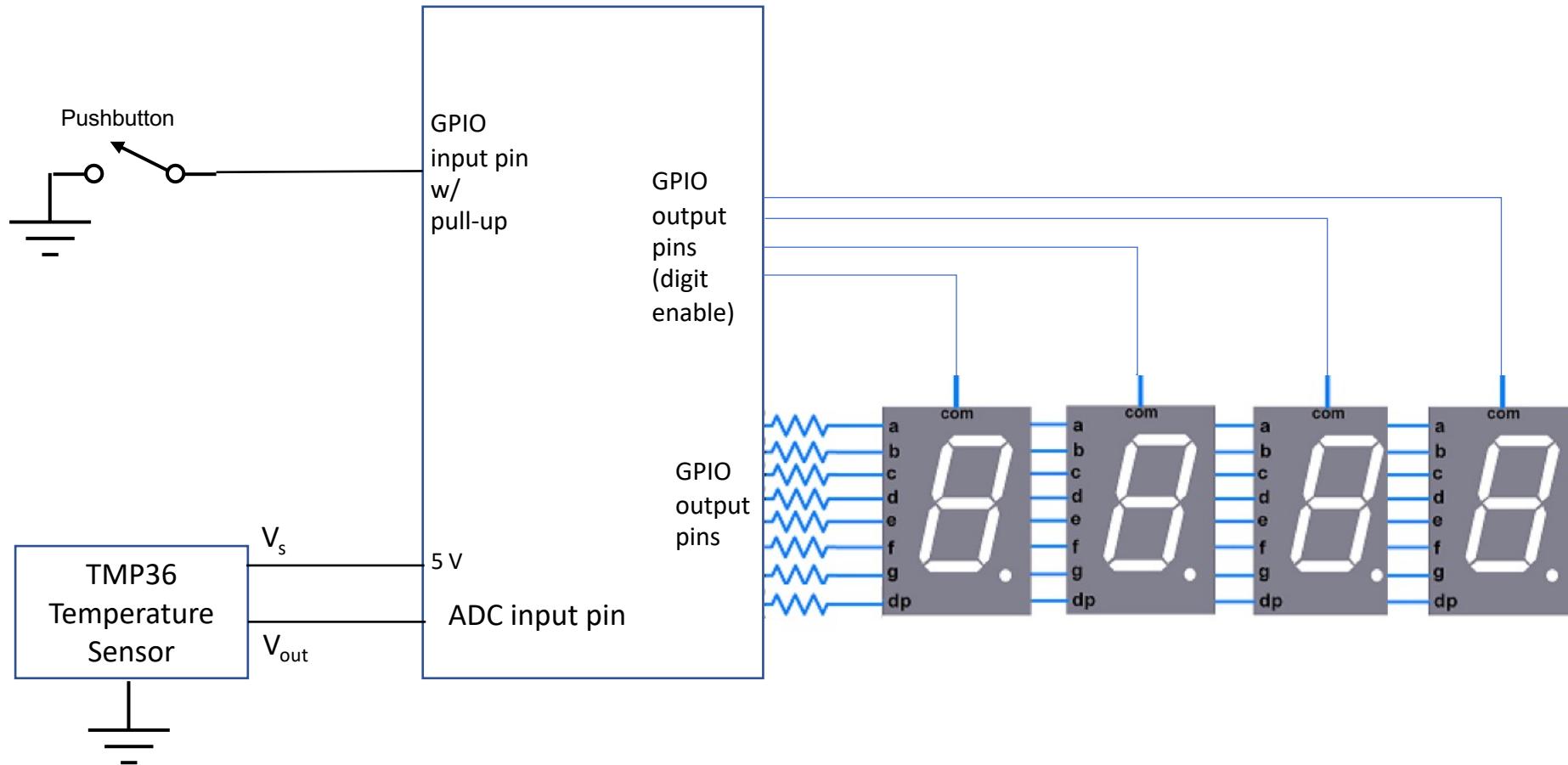
measure
ambient room
temperature
or temp of
your finger

See the Assignment posted to Moodle for details.

Lab Assignment #3, cont. – Build a digital thermometer that displays ambient temp in C or F...another version to display temp on the 4-digit 7-segment display

use to switch
between C and
F temperature
scale display

measure
ambient room
temperature
or temp of
your finger

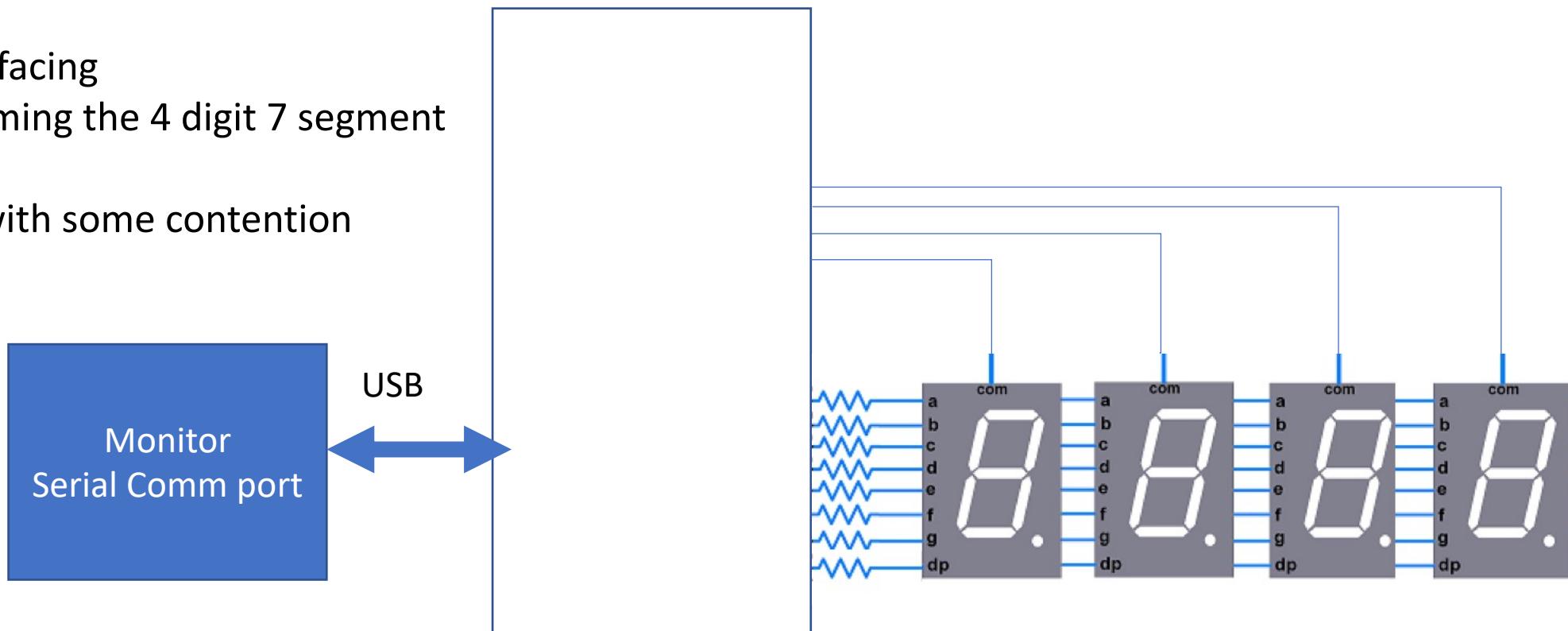


See the Assignment posted to Moodle for details.

Let's build a four digit ring counter (0000 – 9999) that displays digits on a serial monitor and on the 4-digit 7-segment display.

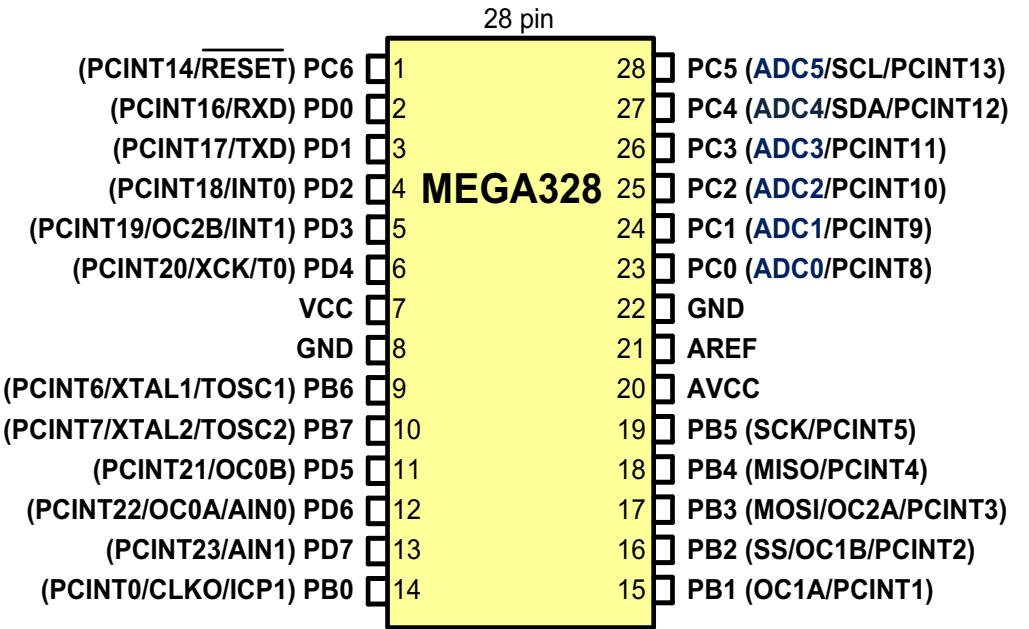
Topics:

- sending digits to serial monitor via UART
- LED interfacing
- programming the 4 digit 7 segment LED
- dealing with some contention issues



Sending digits to serial monitor
via UART:

write a program that
implements a 4 digit ring
counter (0-9999, then repeat),
sending the count via UART to
the serial monitor.



serialwow.c > ...

```
*****
* serialcomm_main.c - this program continuously sends
* the letter 'WOW!' to the serial monitor via the UART on the ATmega328P MCU.
* Date      Author      Revision
* 3/1/22    D. McLaughlin initial release. based on serialcomm_main.c
*****
```

```
#include <avr/io.h>          // Defines constants USCR0B, USCR0C, etc...

void uart_init(void);         // Function prototype (declaration)
void send_char(char);        // Function prototype (declaration)

int main(void){               // Main function definition
    uart_init();              // Initialize the UART

    while (1) {                // Send repeatedly
        send_char('W');
        send_char('o');
        send_char('W');
        send_char(10);           // Carriage Return
        send_char(13);           // Line feed
    }

    // This function initializes the UART peripheral
    // enable; 8 data bits; 1 stop bit, no parity
    // 9600 baud from 16 MHz clock
    void uart_init(void) {
        UCSR0B = (1<<TXEN0);
        UCSR0C = (1<< UCSZ01)|(1<<UCSZ00);
        UBRR0L = 103; // Gives us 9600 baud from 16 MHz clock
    }

    // This function sends a single character to the serial comm port
    void send_char(char letter){
        while (! (UCSR0A&(1<<UDRE0))); // Wait until Tx buffer is empty
        UDR0=letter;
    }
}

***** End of File *****
```

C uart_string.c > ...

```
*****
* uart_string.c - This code sends strings to com port via uart.
* Date      Author      Revision
* 12/16/21  D. McLaughlin Initial writing of the code
* 1/15/22   D. McLaughlin Tested on Arduino Uno w/ Apple M1 pro
*                      host running Monterey
* 2/27/22   D. McLaughlin tested on Windows 11 on Parallels VM
*****
```

```
#include <avr/io.h>
#include <util/delay.h>
#include <string.h> //so that we can use the strlen() function

void uart_init(void);
void uart_send(unsigned char);
void send_string(char *stringAddress);

int main(void){
    char mystring[] = "Shall I compare thee to a summer's day?";
    uart_init(); // initialize the USART
    while (1) {
        send_string(mystring);
        uart_send(13); // Carriage return (goto beginning of line)
        uart_send(10); //line feed (new line)
        _delay_ms(500);
    }
}

// Send a string, char by char, to uart via uart_send()
// Input is pointer to the string to be sent
void send_string(char *stringAddress){
    for (unsigned char i = 0; i < strlen(stringAddress); i++)
        uart_send(stringAddress[i]);
}

> void uart_init(void){...
40
41 > void uart_send(unsigned char ch){...
45
46 **** End of file *****
```

we are making use of 3 user-defined functions:

`uart_init()` `uart_send()` `send_string()`

ASCII Codes:

Dec	Hex	Ch
32	20	
33	21	!
34	22	"
35	23	#
36	24	\$
37	25	%
38	26	&
39	27	,
40	28	<
41	29	>
42	2A	*
43	2B	+
44	2C	-
45	2D	/
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	:
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Ch
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D]
94	5E	^
95	5F	_

Send (transmit) the string: "Wow!"

`uart_send('W');`

`uart_send('o');`

`uart_send('w');`

`uart_send('!');`

or

`uart_send(87);`

`uart_send(111);`

`uart_send(119);`

`uart_send(33);`

or

`send_string("Wow!");`

we are making use of 3 user-defined functions:

`uart_init()` `uart_send()` `send_string()`

ASCII Codes:

Dec	Hex	Ch
32	20	
33	21	!
34	22	"
35	23	#
36	24	\$
37	25	%
38	26	&
39	27	,
40	28	<
41	29	>
42	2A	*
43	2B	+
44	2C	-
45	2D	.
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Ch
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D]
94	5E	^
95	5F	_

6 ways to transmit the letter G

`uart_send('G');`
`uart_send(71);`
`uart_send(0x47);`
`uart_send(6+65);`
`uart_send(6+'A');`
`send_string("G"); // strings in " "`

6 ways to transmit the number 4

`uart_send('4');`
`uart_send(52);`
`uart_send(0x34);`
`uart_send(4+48);`
`uart_send(4+'0');`
`send_string("4");`

counter_uart.c — seven_segment_LED

_main.c C counter_uart.c X C counter_uart2.c M makefile ⌂ ...

counter_uart.c > main(void)

```
/* counter_uart.c This code implements a 0- 9999 ring counter
and displays the count value on the monitor using the UART.
Uses itoa() function to convert count to a string.
This is a demo for ECE-231 Spring 2022
D. McLaughlin 3/18/22 */

#include <avr/io.h>      // #defines all the port pins
#include <util/delay.h> // Declares _delay_ms() function
#include <stdlib.h>       // Declares itoa() function
#include <string.h>        // Declares strlen() function

void uart_init(void);
void uart_send(unsigned char);
void send_string(char *stringAddress);

int main(void){
    char mystring[10];
    uart_init(); // initialize the USART
    while (1) {
        for (unsigned int i=0; i<10000; i++){
            itoa(i, mystring, 10); // Convert i to a string, base 10
            send_string(mystring);
            uart_send(13); // Carriage return (goto beginning of line)
            uart_send(10); //line feed (new line)
            _delay_ms(1000);
        }
    }
}

// Send a string, char by char, to uart via uart_send() ...
void send_string(char *stringAddress){...}

// Initialize the uart, 8,1,0, 9600 baud
void uart_init(void){...

    // Send a single character to the UART transmitter
    void uart_send(unsigned char ch){
        while (!(UCSR0A & (1 << UDRE0))); //wait til tx data buffer empty
        UDR0 = ch; //write the character to the USART data register
    }
}
```

```
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24 TOKENS AND KEYWORDS
25
26 CONSTANT
27
28 VARIABLE
29
30 OPERATORS AND
31 EXPRESSIONS
32
33 DECISION CONTROL
34
35 STATEMENT
36
37 LOGIC CONTROL STATEMENTS
38
39 CASE CONTROL STATEMENTS
40
41
42
43
44
45
```

commenting out the
uart_send(10); line

```
lude 9 reading on-chip flash data  
ing  
lude verifying  
lude 418 bytes of flash verified  
lude safemode: Fuses OK (E:00, H:00, L:00)  
lude Done. Thank you.  
mcl taught me wine seven_segment_LED %  
- PLAIN TYPES  
- TOKENS AND KEYWORDS  
- CONSTANT  
- VARIABLE  
- OPERATORS AND  
EXPRESSIONS  
- DECISION CONTROL  
STATEMENTS  
- LOOP CONTROL STATEMENTS  
- CASE CONTROL STATEMENTS
```

```
/* counter_uart2.c This code implements a 0- 9999 ring counter
and displays the count value on the monitor using the UART.
Sends each count digit individually
This is a demo for ECE-231 Spring 2022
D. McLaughlin 3/18/22 */

#include <avr/io.h>      // #defines all the port pins
#include <util/delay.h> // Declares _delay_ms() function
#include <string.h>      // Declares strlen() function

void uart_init(void);
void uart_send(unsigned char);
void send_string(char *stringAddress);

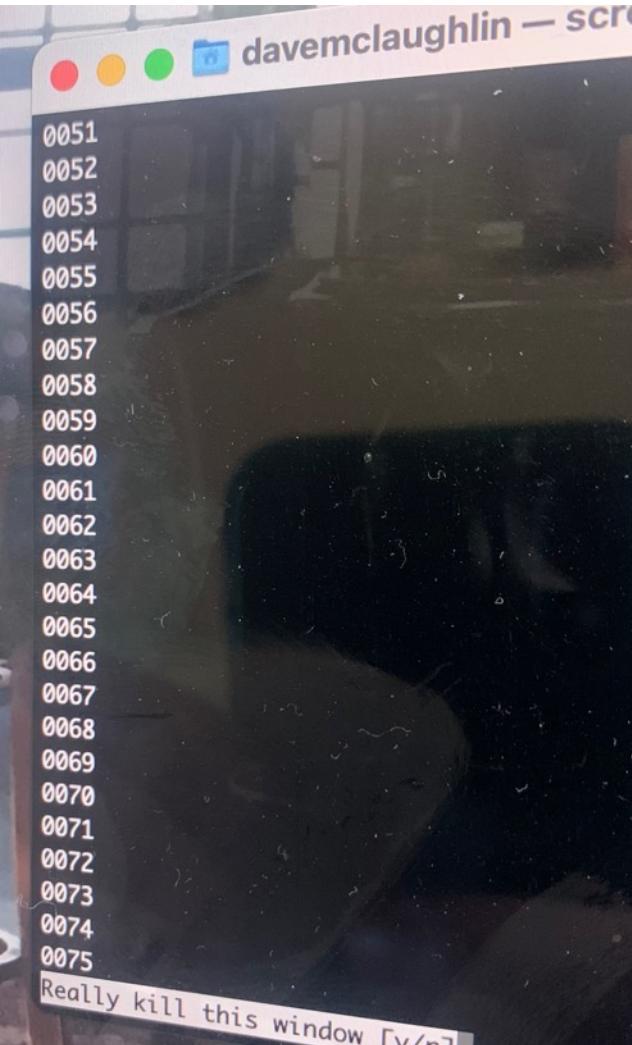
int main(void){
    char digit;
    uart_init(); // initialize the USART
    while (1) {
        for (unsigned int i=0; i<10000; i++){
            digit = i/1000;          // 1000's place (most signif digit)
            uart_send(digit+'0');
            digit = (i/100) %10;    // 100's place
            uart_send(digit+'0');
            digit = (i/10) %10;     // 10's place
            uart_send(digit+'0');
            digit = i%10;           // 1's place or least significant digit
            uart_send(digit+'0');
            uart_send(13); // Carriage return (goto beginning of line)
            uart_send(10); //line feed (new line)
            _delay_ms(10);
        }
    }
}

> // Send a string, char by char, to uart via uart_send()...
> void send_string(char *stringAddress){...

> void uart_init(void){...

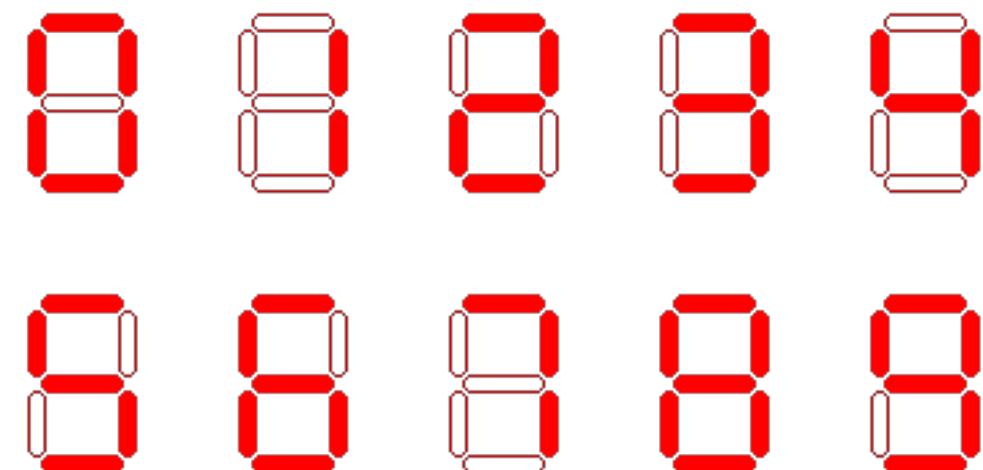
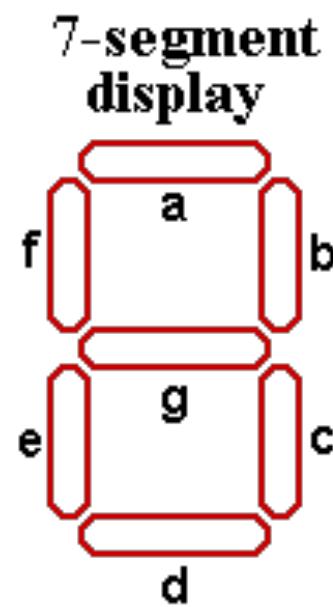
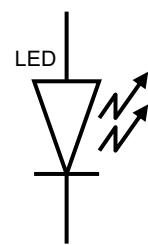
> void uart_send(unsigned char ch){...

***** End of file *****
```

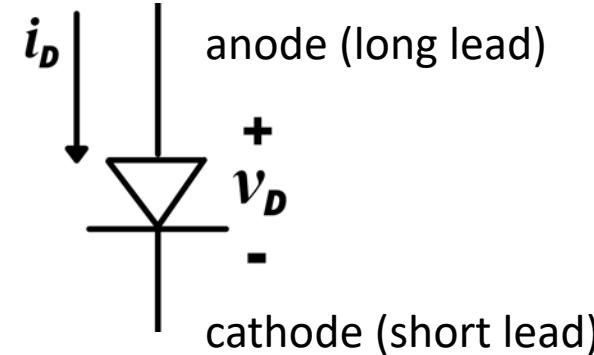
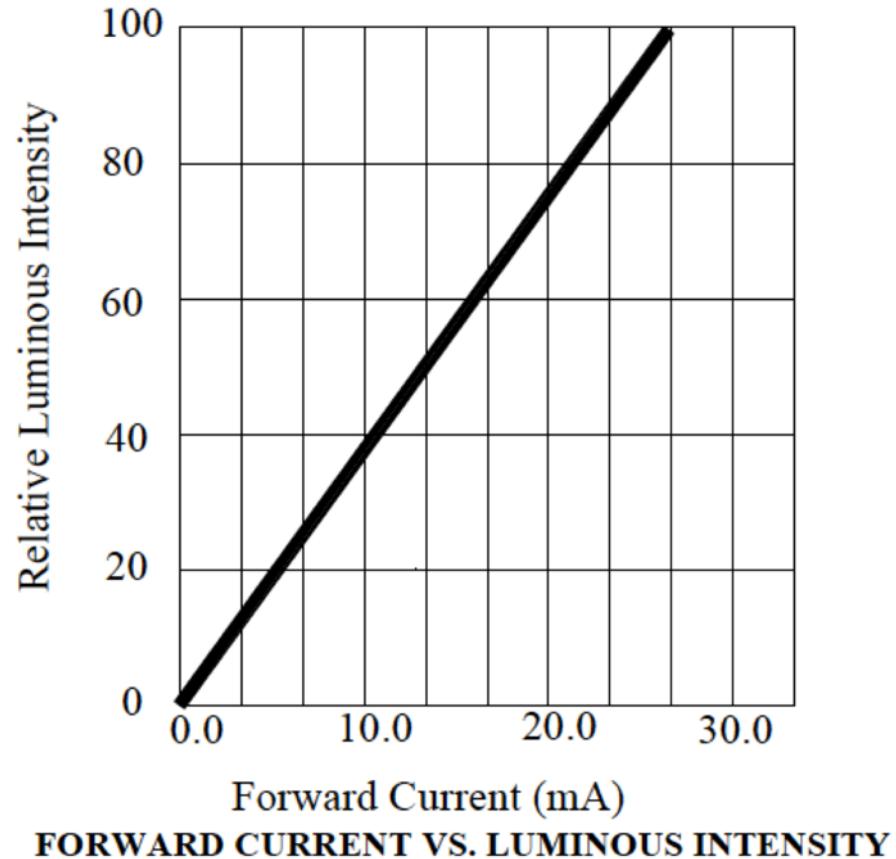


LED Interfacing & Programming

- Interfacing LEDs (current-limiting resistor)
- Retinal persistence
- 4 digit LED display
 - interfacing (hardware)
 - programming (software)



Light Emitting Diode (LED): brightness proportional to forward current
-- we consider here the LEDs in our kits --



forward voltage: $v_D = 2V$ (depends on diode color)

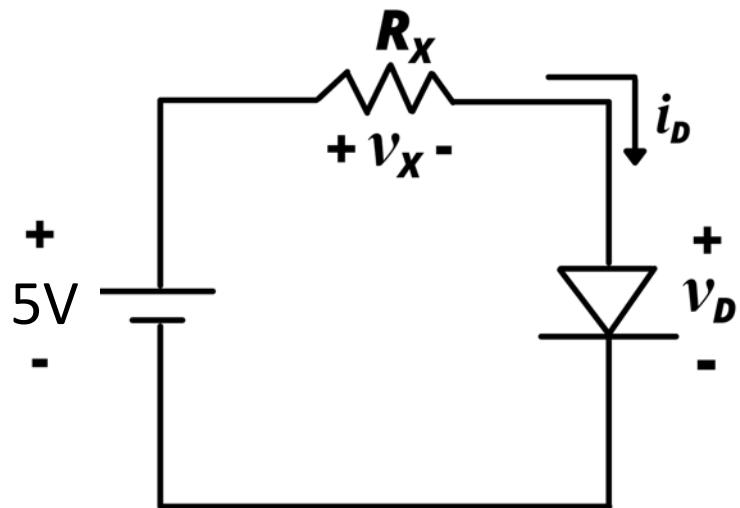
forward current: $i_d = 30 \text{ mA , max}$



anode cathode

Want to know more about diodes? See
<http://openbooks.library.umass.edu/funee/chapter/4-1/>

Use a current-limiting resistor to drive an LED



$-5 + v_x + v_D = 0$ (via Kirchoff's Voltage Law)

$-5 + i_D R_x + v_D = 0$ (via Ohm's Law)

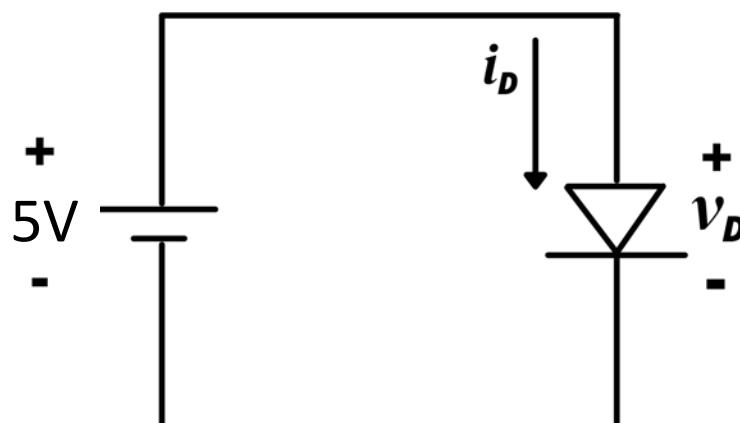
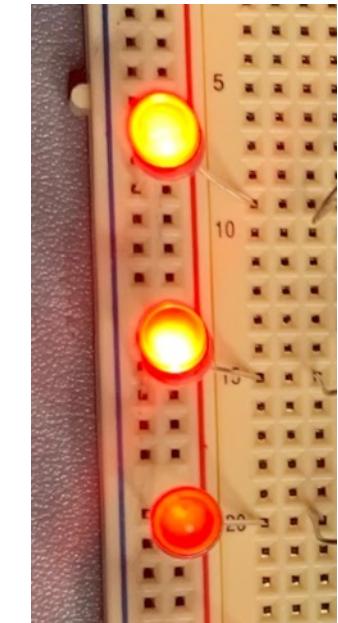
$-5 + i_D R_x + 2 = 0$ (using $v_D = 2$ volts)

$i_d = 3/R_x$ (current is limited by resistance R_x)

$$R_x = 100 \Omega \rightarrow i_d = 0.03 \text{ A} = 30 \text{ mA}$$

$$R_x = 1000 \Omega = 1\text{k} \Omega \rightarrow i_d = 0.003 \text{ A} = 3 \text{ mA}$$

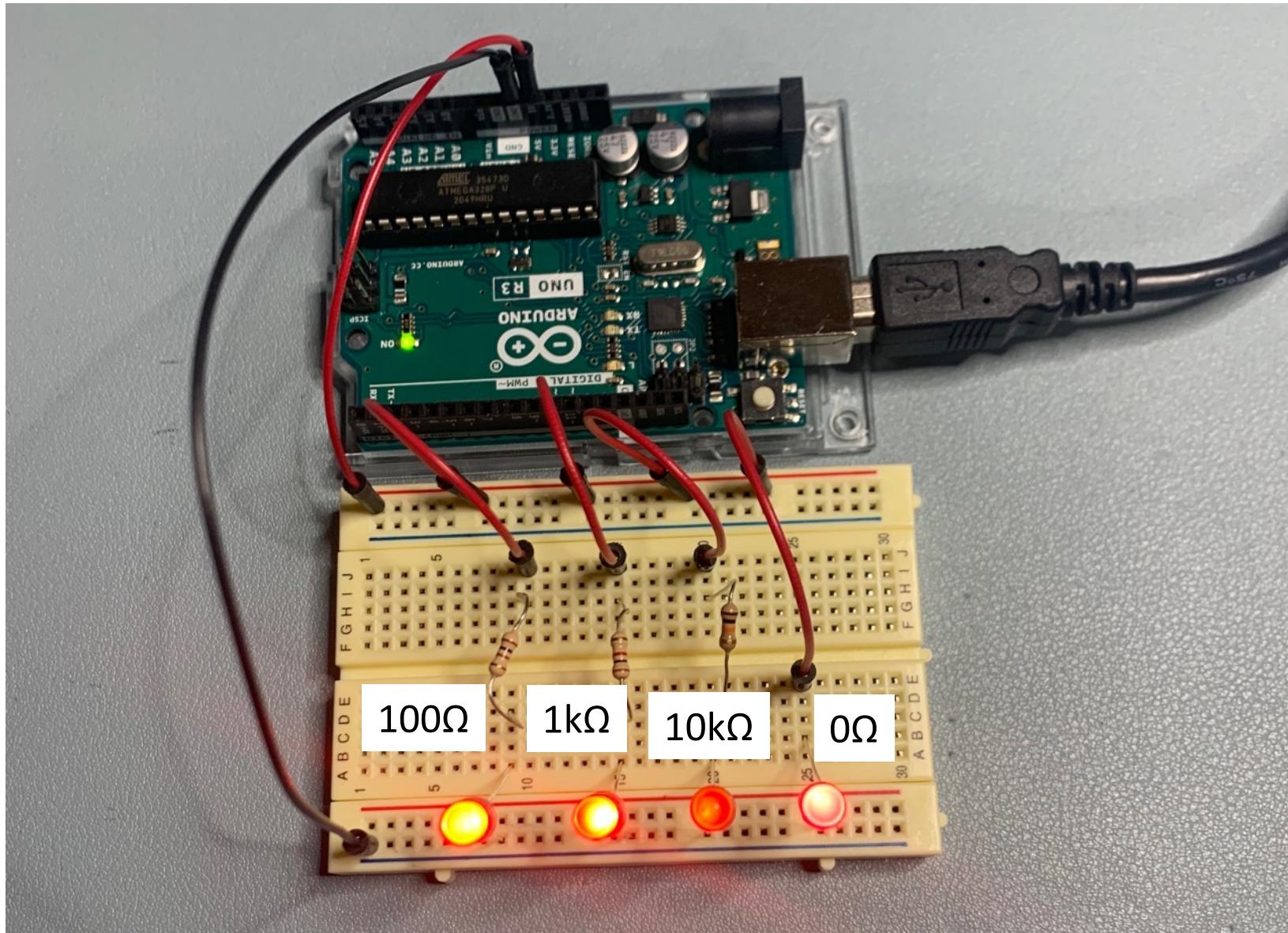
$$R_x = 10,000 \Omega = 10\text{k} \Omega \rightarrow i_d = 0.0003 \text{ A} = 0.3 \text{ mA}$$



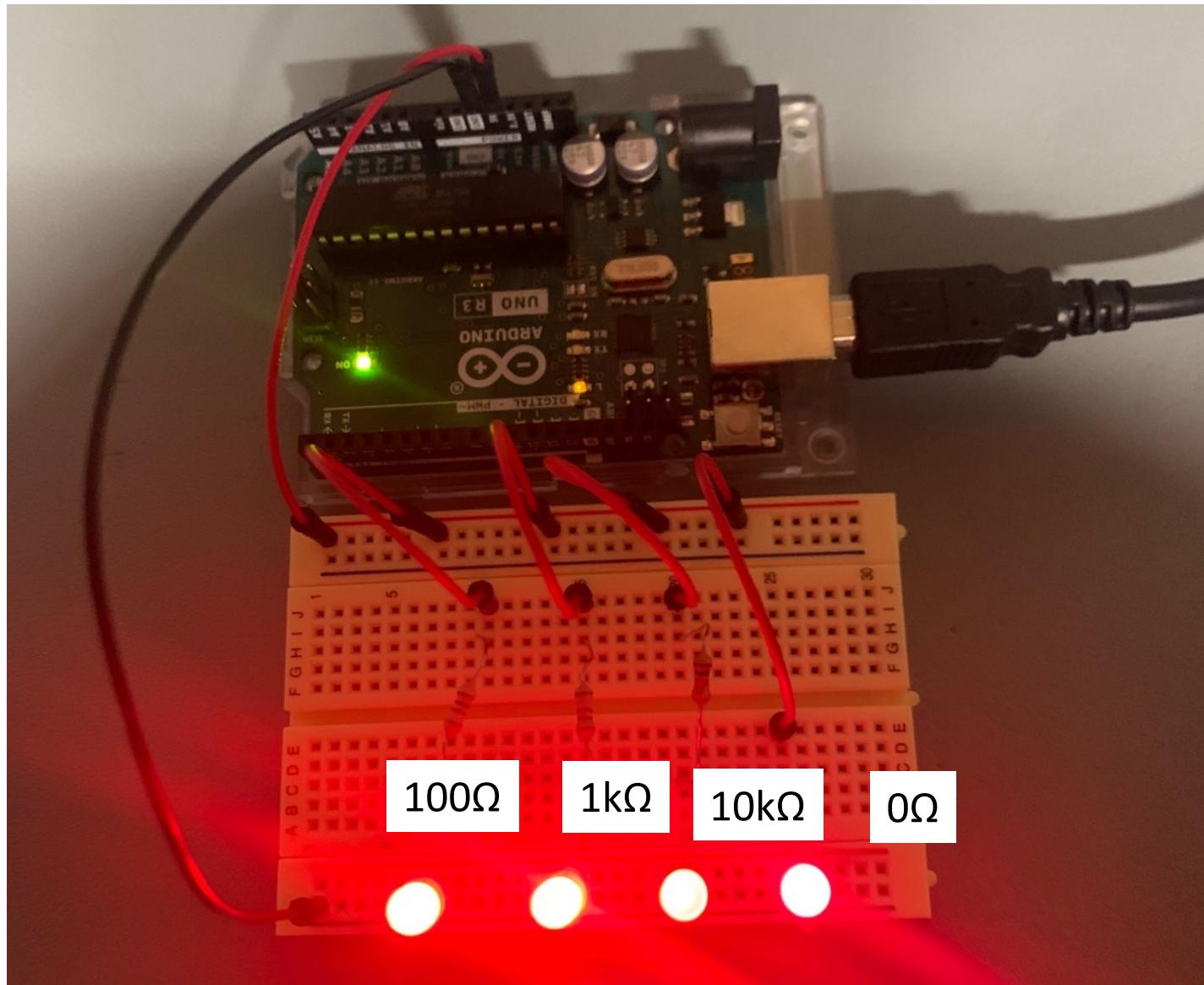
$$R_x = 0 \Omega \rightarrow i_d = \infty \text{ A}$$

(limited by the max current from the power supply or by the resistance of the power supply & wires. In either case, the LED is damaged)





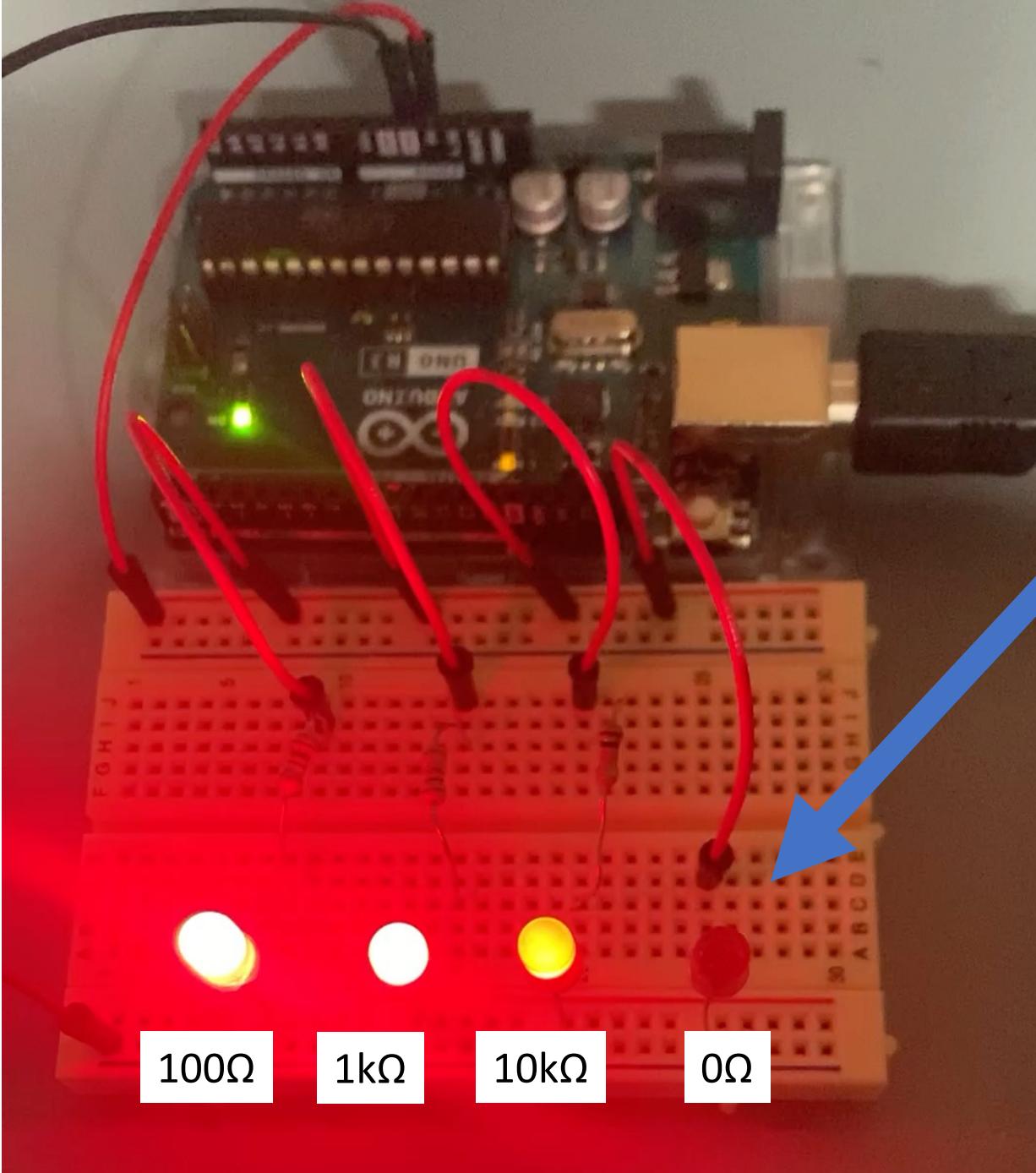
LED's powered by the 5V output pin from Arduino Uno (200 mA max)



LED's powered by the 5V output pin from Arduino Uno (200 mA max)

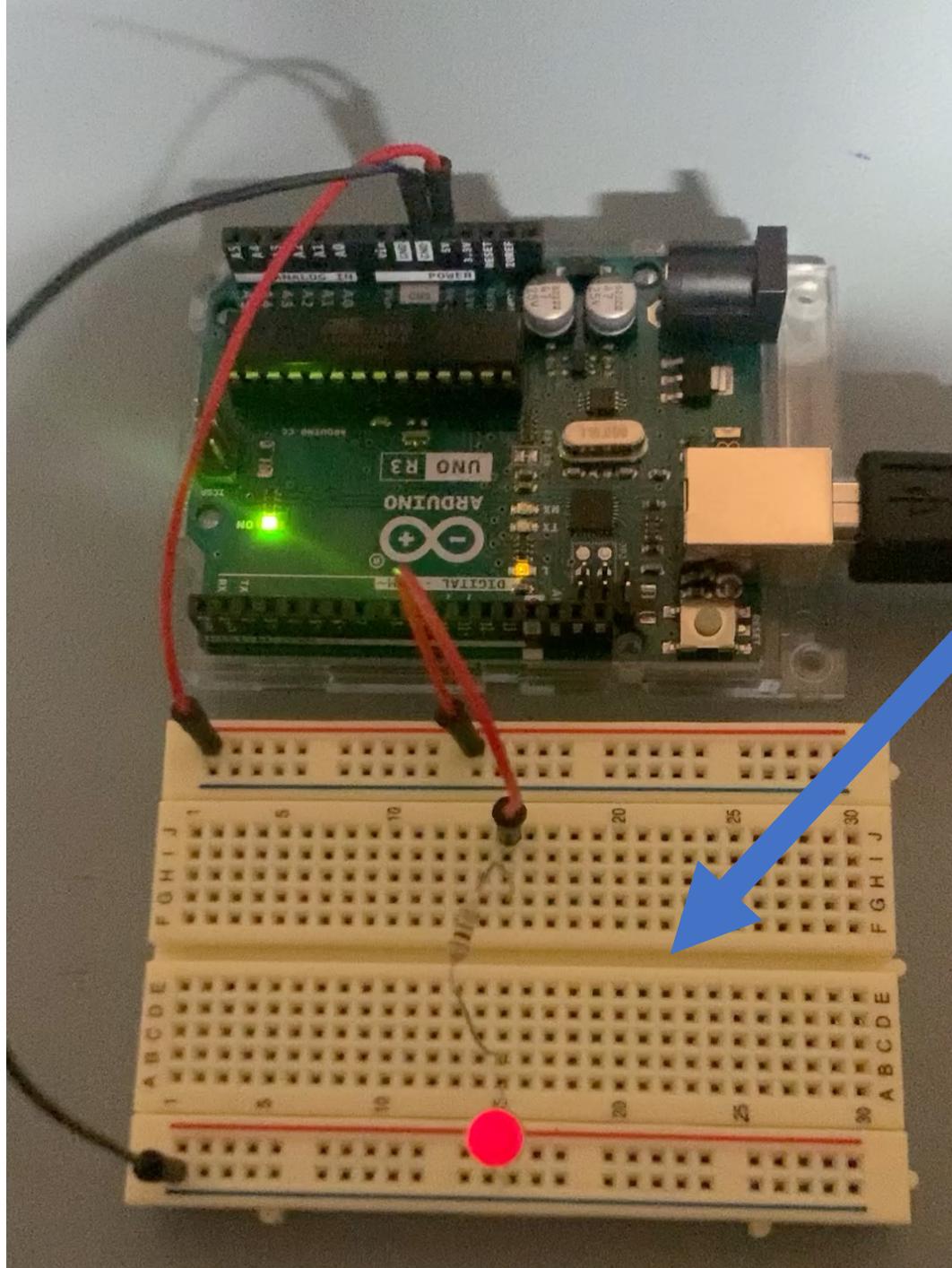
Room lights darkened;
LEDS look much
brighter.

Engineering Malpractice



This LED is damaged. It glows intermittently.

Suppose we return this damaged LED to our parts drawer and re-use it....



Re-using a damaged LED.

It should be on.
But it blinks intermittently.
Imagine using this damaged
LED to diagnose an
embedded system

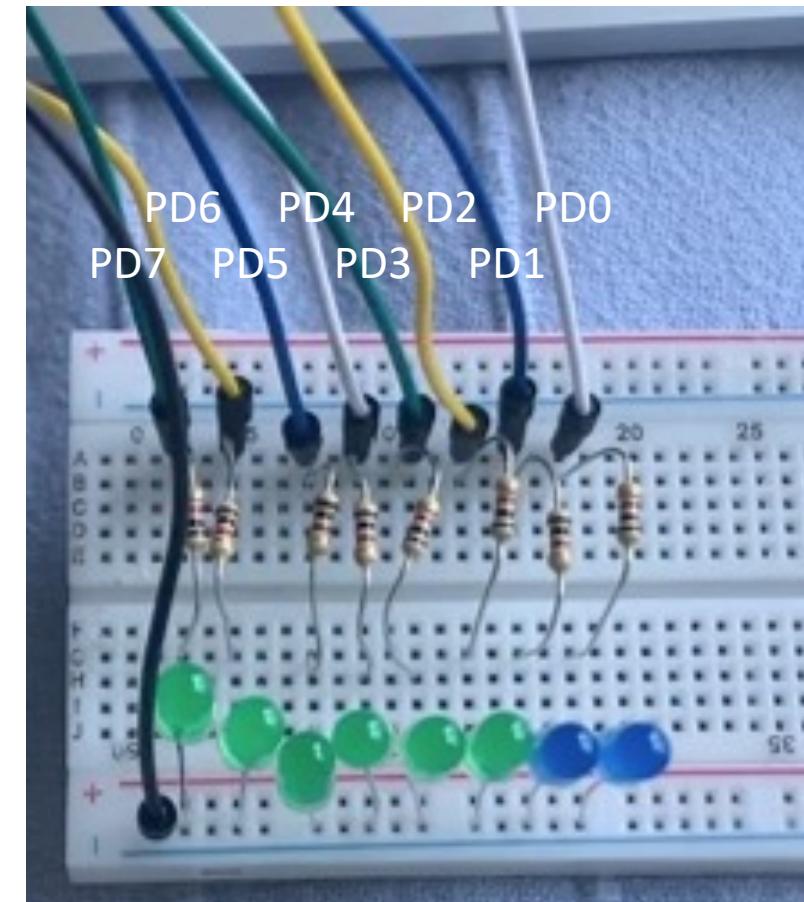
8 bit binary counter

Note: your textbook typically has this line at the top of the code. This tells the compiler the frequency of the clock driving the ATmega328P. (The ATmega328P doesn't know time; it just counts CPU cycles.) We have the line -DF_CPU 16000000 in makefile that defines this.

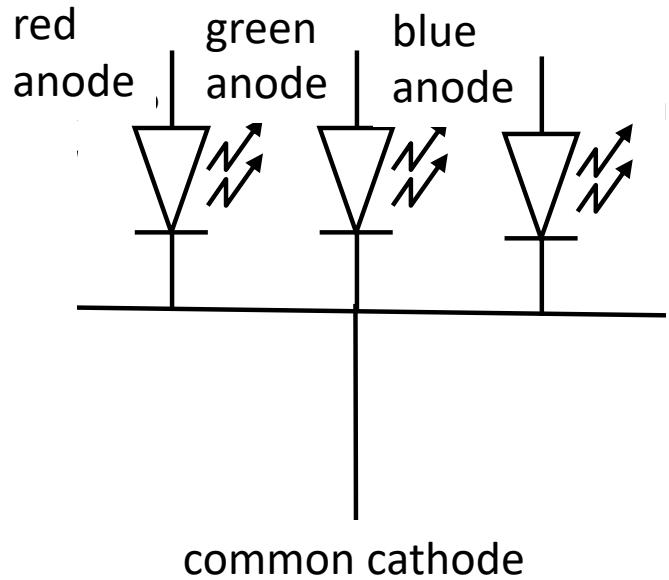
```
/*
#define F_CPU 16000000UL
#include <util/delay.h>
#include <avr/io.h>

int main(void)
{
    unsigned char i=0;
    DDRD = 0xFF;
    while (1) {
        i++;
        PORTD=i;
        _delay_ms(100);
    }
}
```

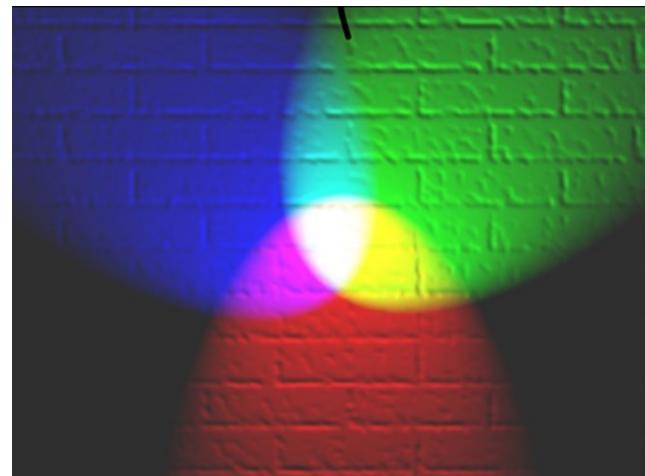
8 LEDs connected to the pins of port D



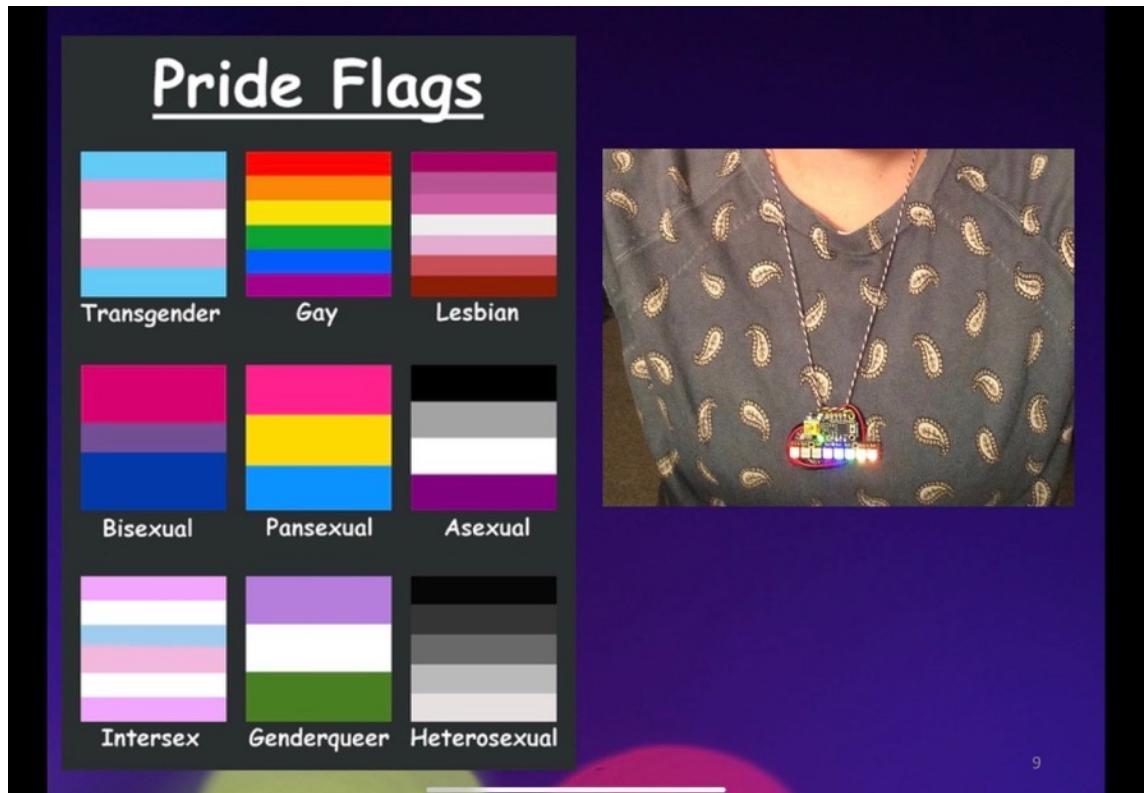
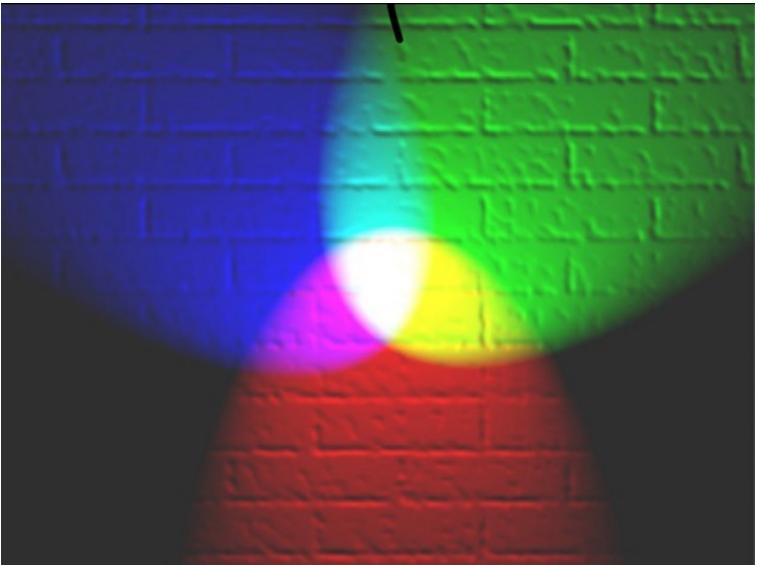
Common Cathode RGB LED



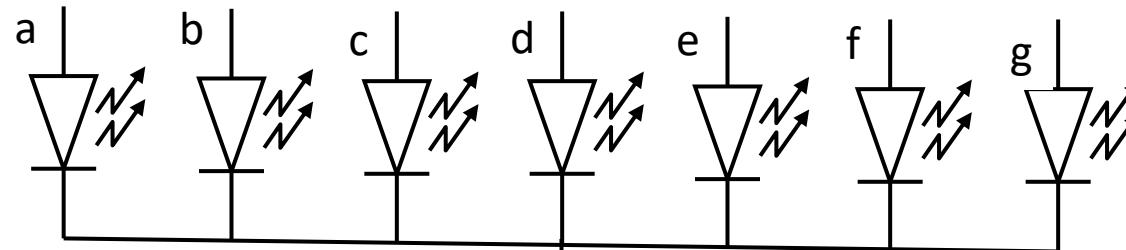
red + green = yellow
red + blue = magenta
green + blue = cyan
red + green + blue = white



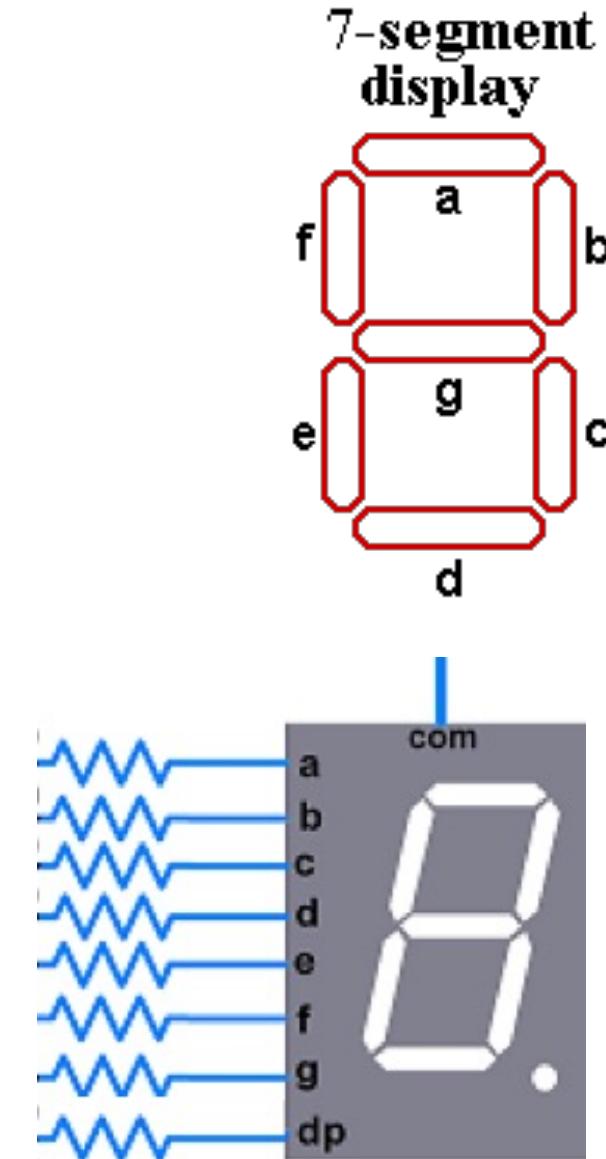
red, green, blue
anodes (+)
common
cathode (-)



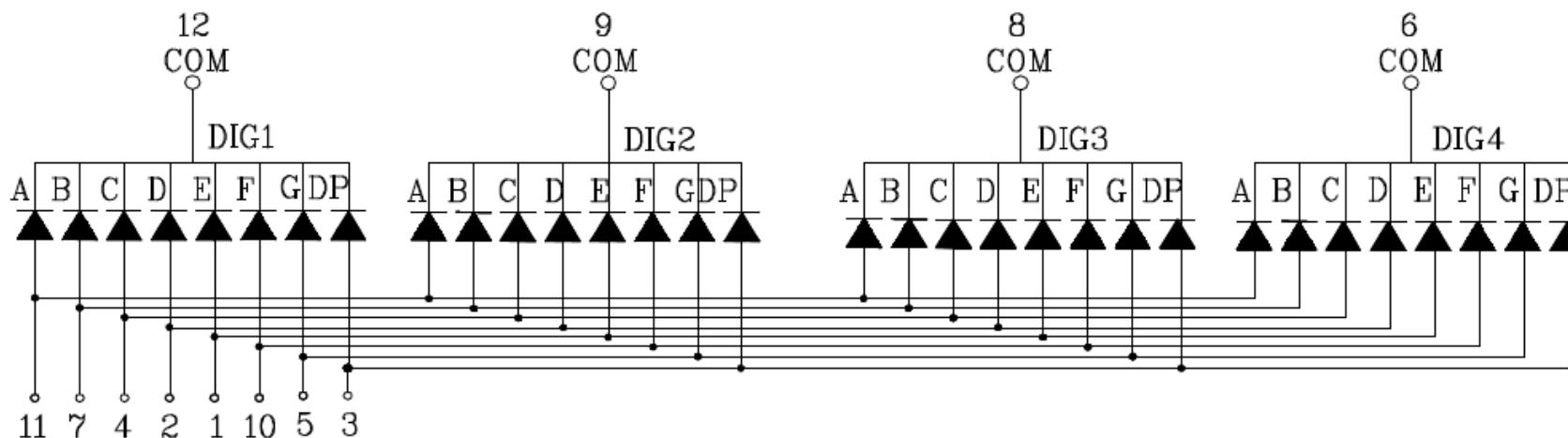
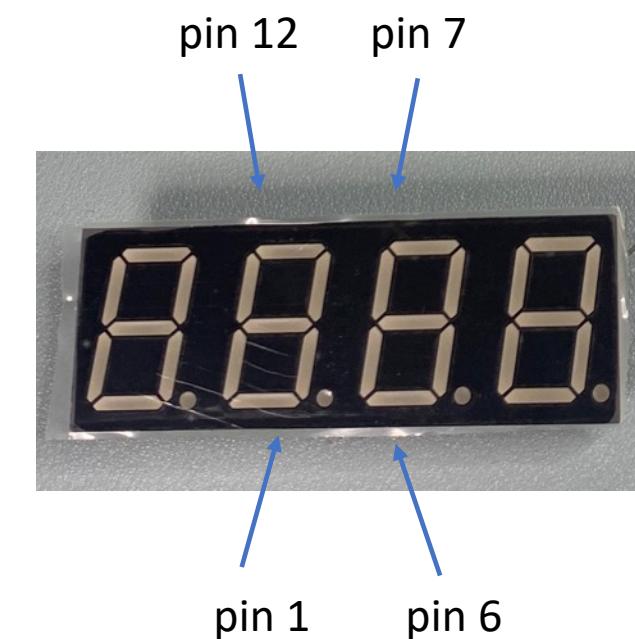
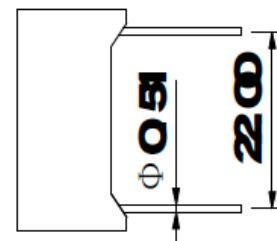
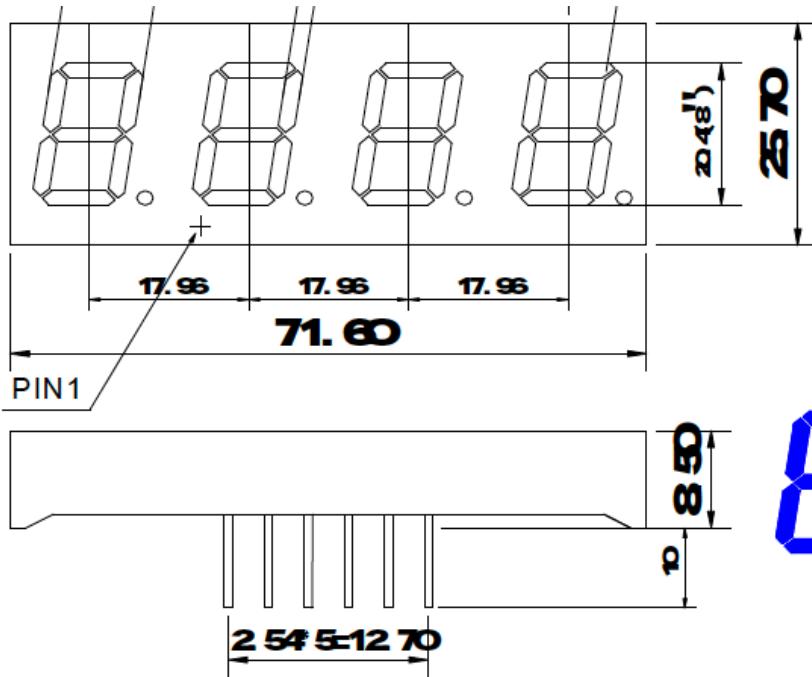
Common Cathode 7 Segment LED



common cathode

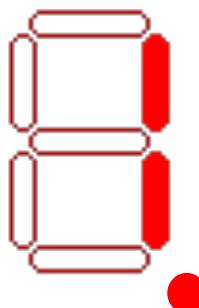
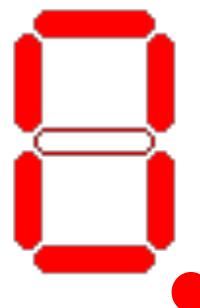
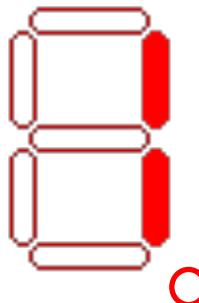
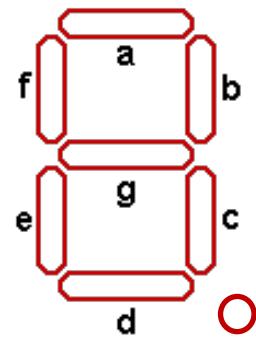


some also have a decimal point as an 8th segment



Datasheet: <https://www.sparkfun.com/products/11409>

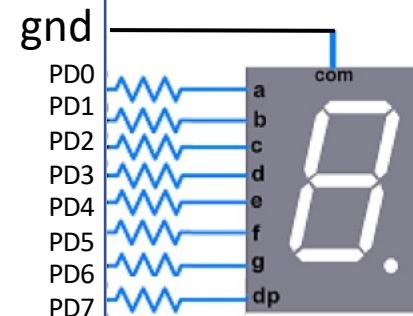
7-segment display



	dp	g	f	e	d	c	b	a	Binary	Hex
	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0		
0	0	0	1	1	1	1	1	1	0011 1111	0x3F
1	0	0	0	0	0	1	1	0	0000 0110	0x06

	dp	g	f	e	d	c	b	a	Binary	Hex
	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0		
0.	1	0	1	1	1	1	1	1	1011 1111	0xBF
1.	1	0	0	0	0	1	1	0	1000 0110	0x86

328P

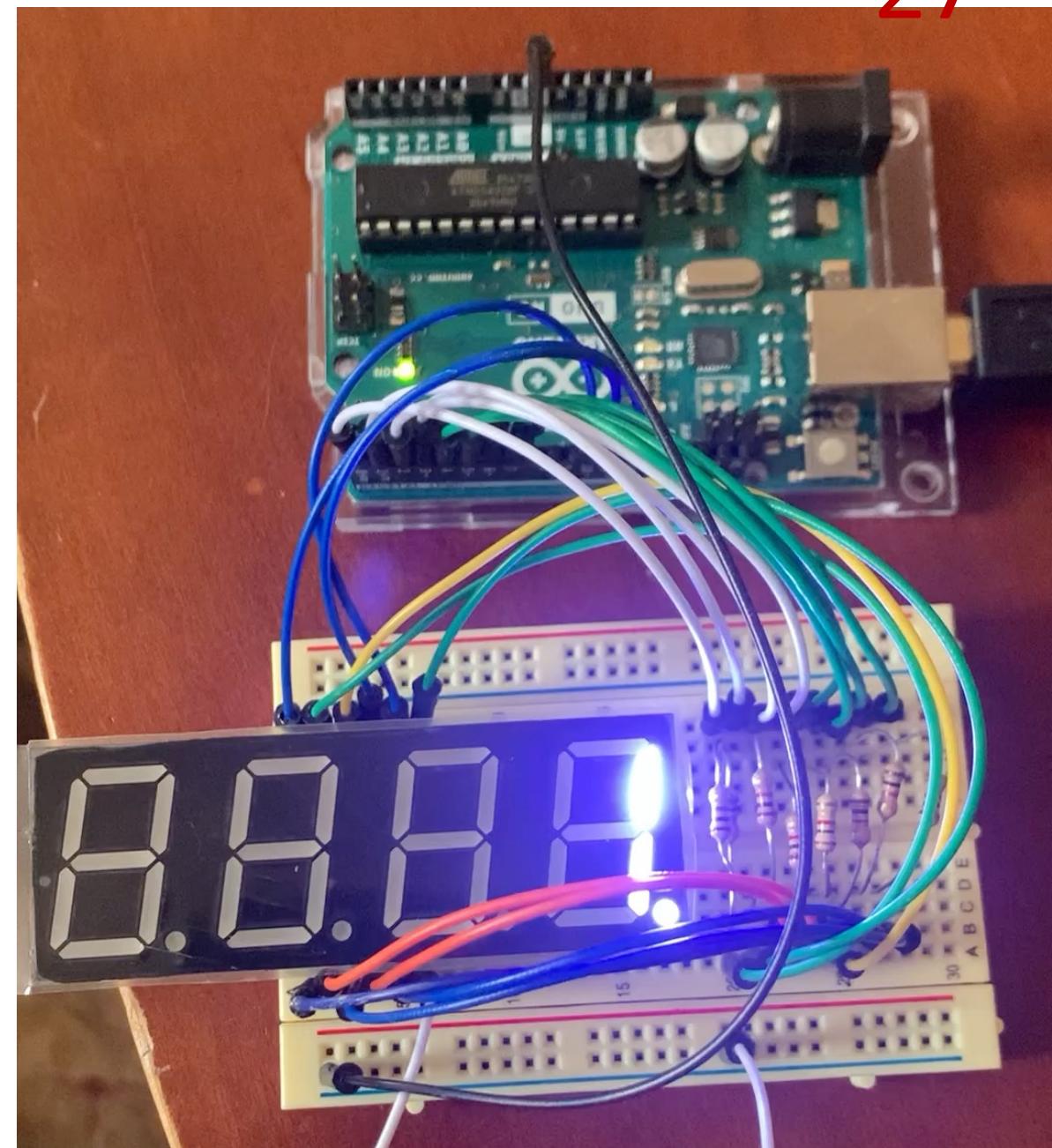


Common Cathode 7
Segment LED

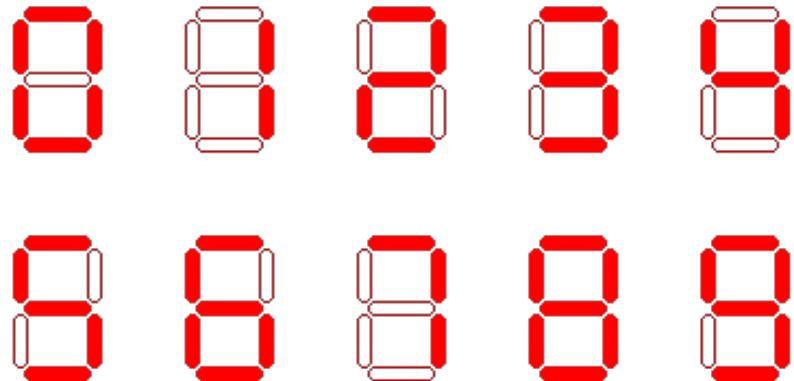
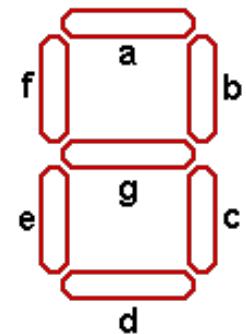
note: these slides use ABCDEFG and abcdefg interchangeably

```
/* seven_0101.c This code displays the digits 0, 1, 0., 1.  
in sequence on DIG4 of the 4 digit, 7 segment (+ dp) LED  
display. This is a demo for ECE-231 Spring 2022  
D. McLaughlin 3/20/22 */
```

```
#include <avr/io.h>  
#include <util/delay.h>  
#define MYDELAY 500  
  
int main(void){  
    DDRD = 0xFF;          // Set all 8 pins as output  
    PORTB = 0;            // Set all pins low  
    while(1){  
        PORTD = 0x3f;    // Illuminate 0  
        _delay_ms(MYDELAY);  
        PORTD = 0x06;    // Illuminate 1  
        _delay_ms(MYDELAY);  
        PORTD = 0xBF;    // Illuminate 0.  
        _delay_ms(MYDELAY);  
        PORTD = 0x86;    // Illuminate 1.  
        _delay_ms(4*MYDELAY); // Longer delay before repeating  
    }  
}
```

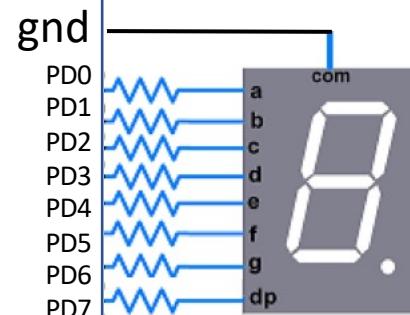


7-segment
display



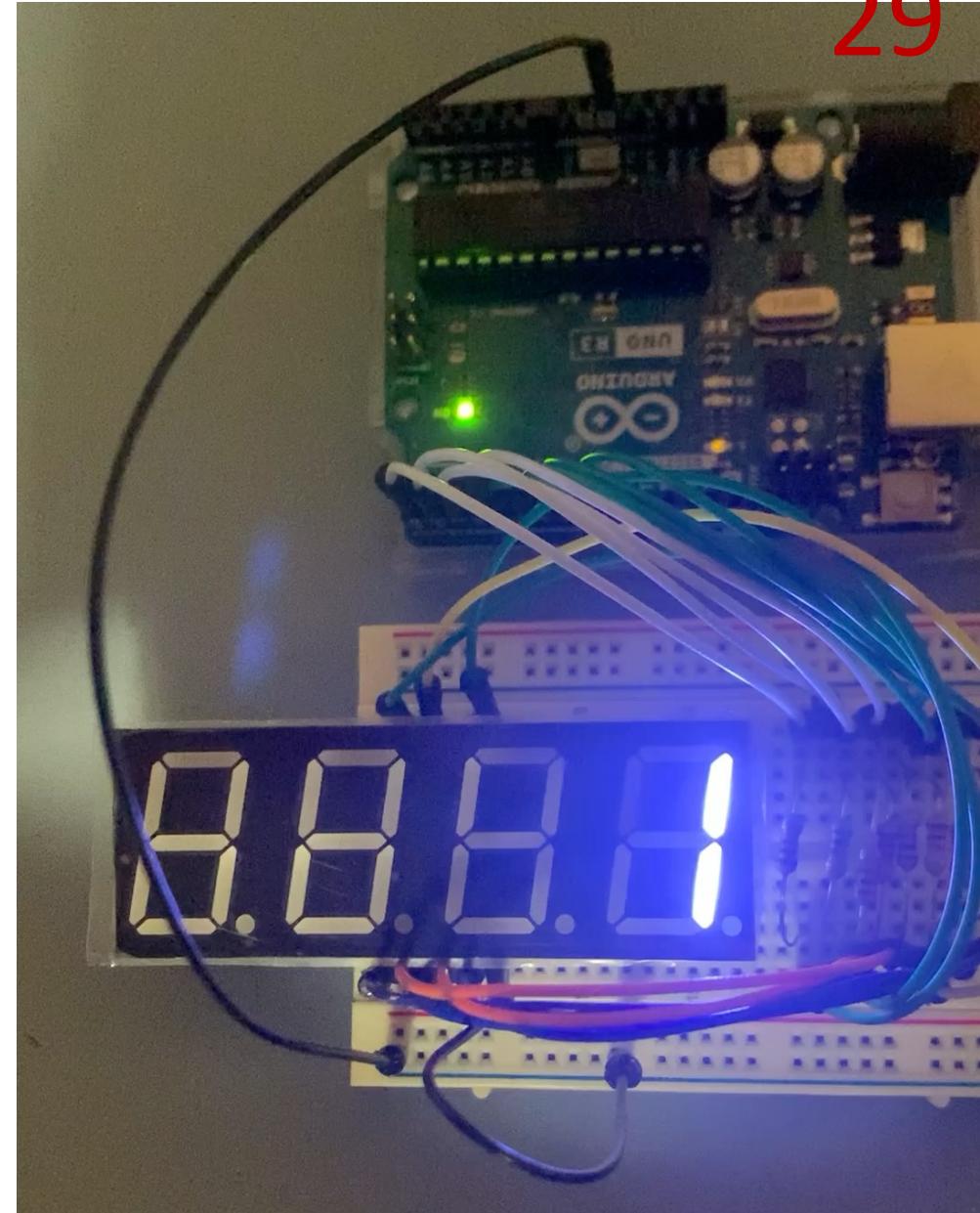
	dp	g	f	e	d	c	b	a	Binary	Hex
	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0		
0	0	0	1	1	1	1	1	1	0011 1111	0x3F
1	0	0	0	0	0	1	1	0	0000 0110	0x06
2	0	1	0	1	1	0	1	1	0101 1011	0x5B
3	0	1	0	0	1	1	1	1	0100 1111	0x4F
4										
5	0	1	1	0	1	1	0	1	0110 1101	0x6D
6										
7										
8										
9	0	1	1	0	1	1	1	1	0110 1111	0x6F

328P



Common Cathode 7
Segment LED

```
/* seven_main.c This code demonstrates the use of a 4 digit  
7 segment LED.  
D. McLaughlin 3/16/22 ECE-231 Demo */  
  
#include "avr/io.h"  
#include "util/delay.h"  
  
int main(void)  
{  
    unsigned char ledDigits[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D,  
        0x07, 0x7F, 0x67};  
    unsigned char i=0;  
    DDRD = 0xFF; //7segment pins  
  
    while (1) {  
        i++;  
        if(i>9)  
            i=0;  
        PORTD = ledDigits[i]; //digit  
        _delay_ms(10); ← for this movie,  
        _delay_ms(1000);  
    }  
}
```

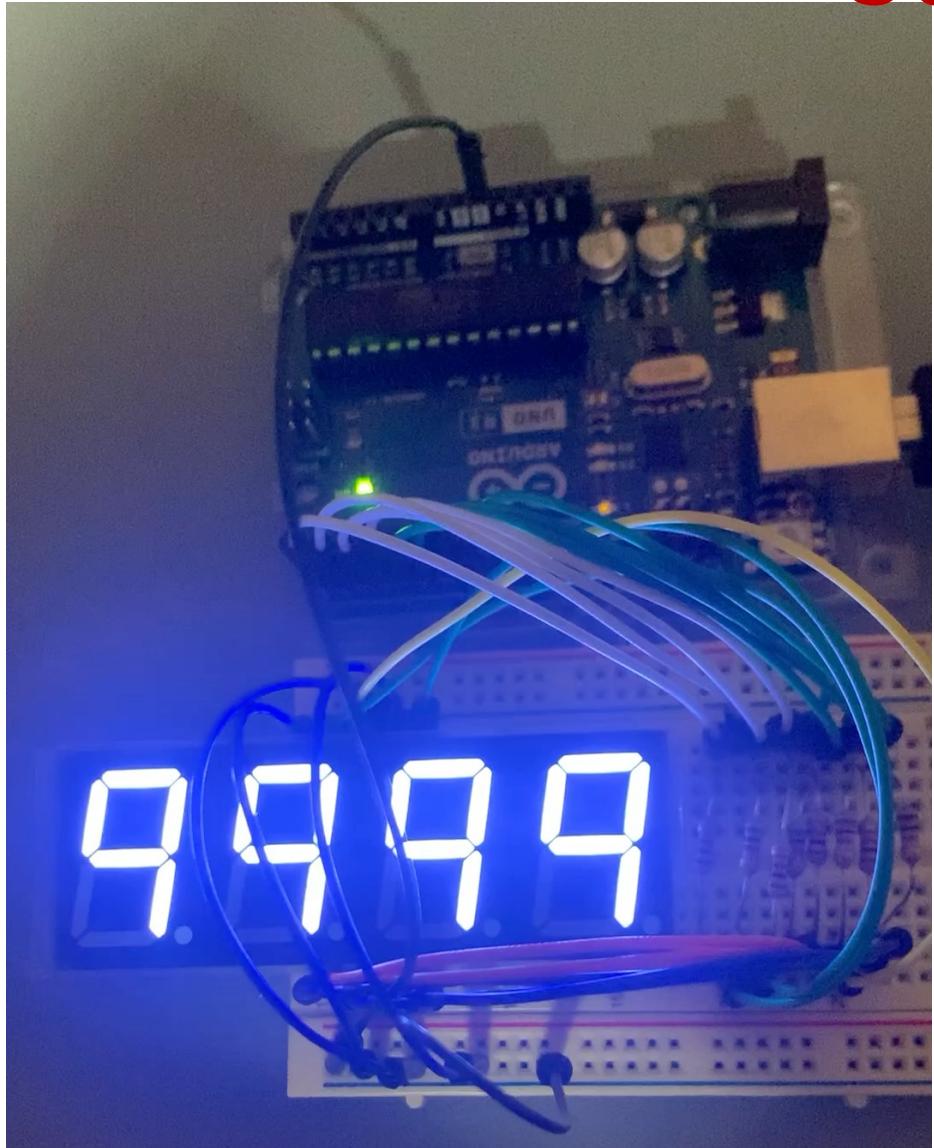
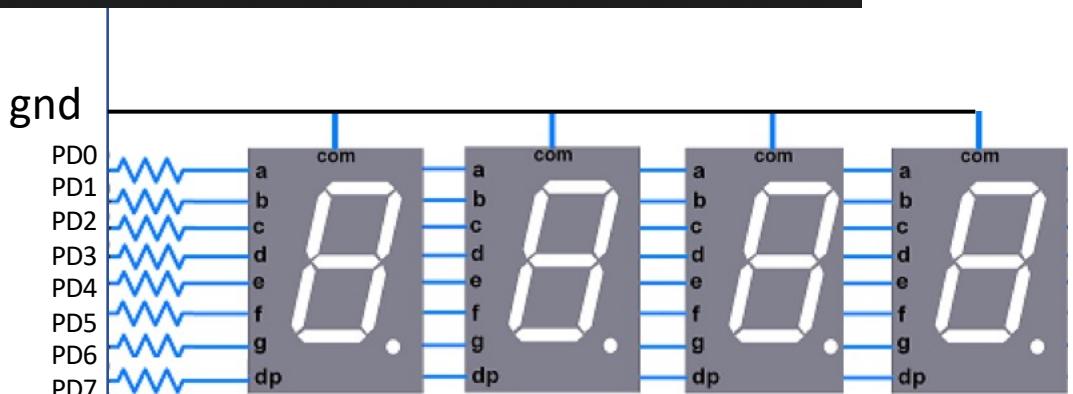


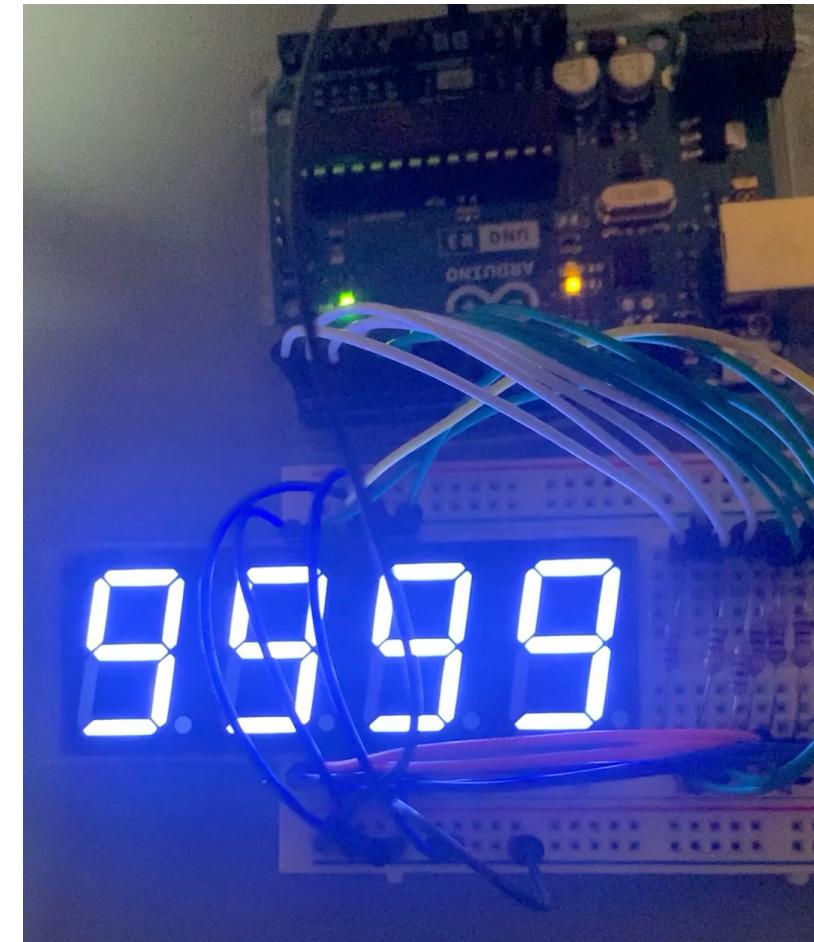
```
/* seven_main.c This code demonstrates the use of a 4 digit
7 segment LED.
D. McLaughlin 3/16/22 ECE-231 Demo */

#include "avr/io.h"
#include "util/delay.h"

int main(void)
{
    unsigned char ledDigits[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D,
        0x07, 0x7F, 0x67};
    unsigned char i=0;
    DDRD = 0xFF; //7segment pins

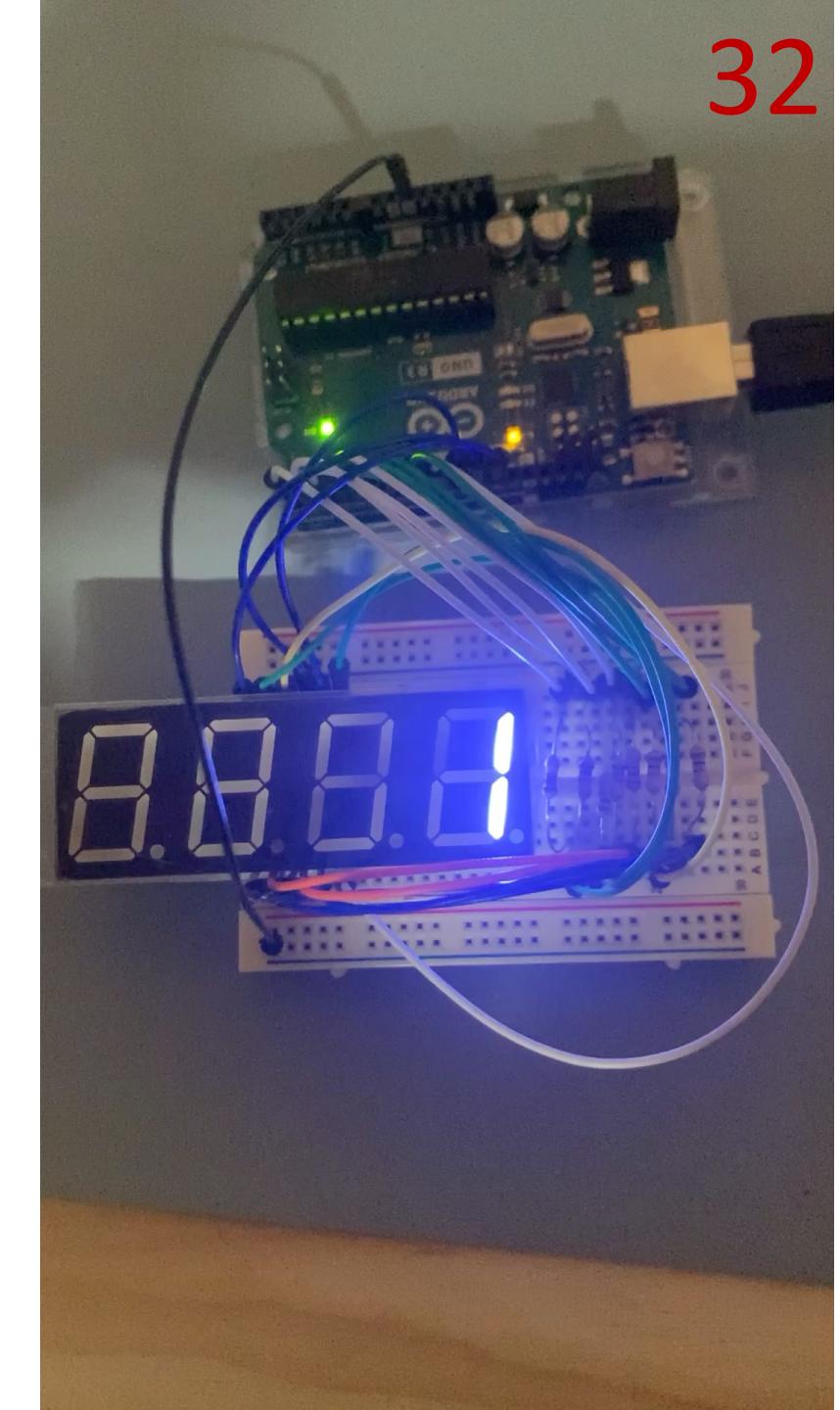
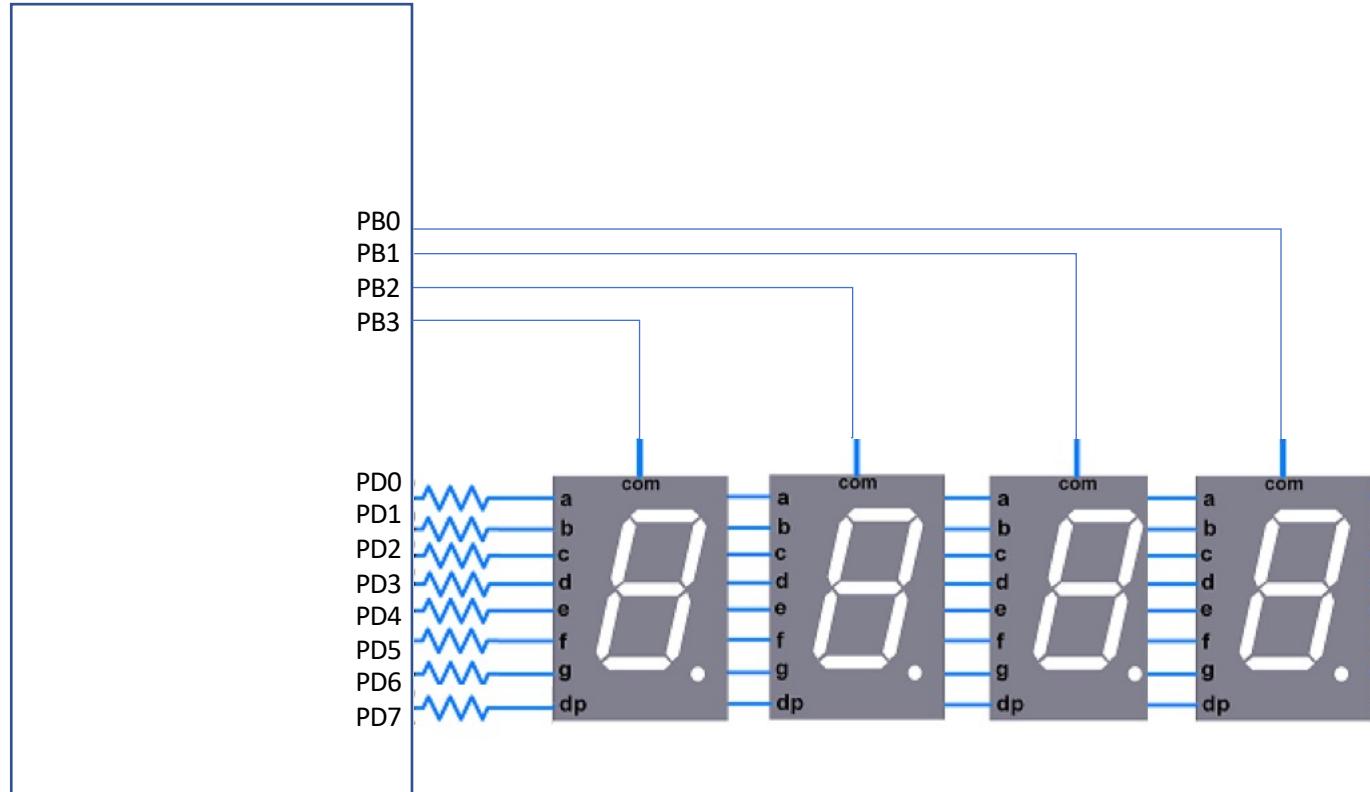
    while (1) {
        i++;
        if(i>9)
            i=0;
        PORTD = ledDigits[i]; //digit
        _delay_ms(10); ← for this movie,
        _delay_ms(1000); ←
    }
}
```





Retinal Persistence

The com pin for each digit connected to PB0-PB4.
When these pins are low, the digit will glow.
Code (not shown) sequences through PB0 – PB1 –
PB2 – PB3 to sequence through the digits.



retinal persistence: repeatedly illuminate digits with update faster than $1/30$ sec = 0.033 sec = 33 msec

pseudocode:

increment counter (0-9999)

illuminate 1's digit

wait 5 ms

illuminate 10's digit

wait 5 ms

illuminate 100's digit

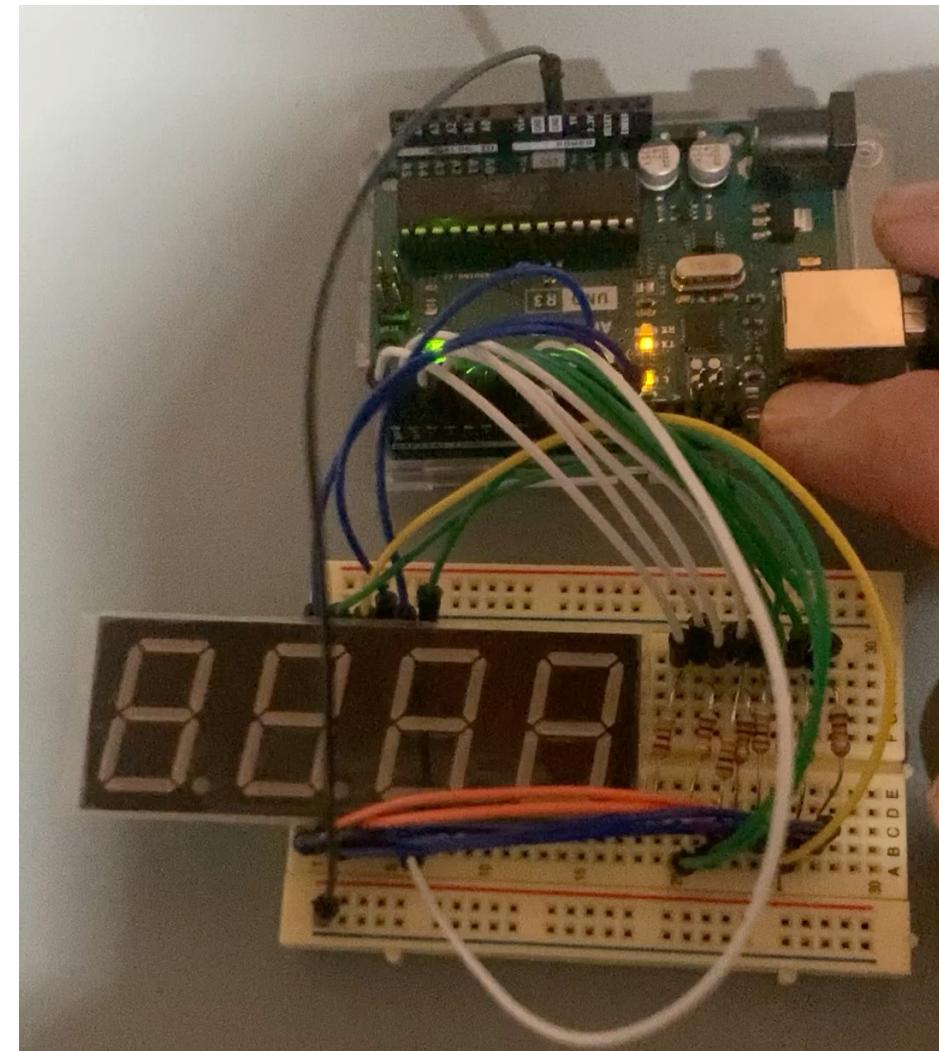
wait 5 ms

illuminate 1000's digit

wait 5 ms

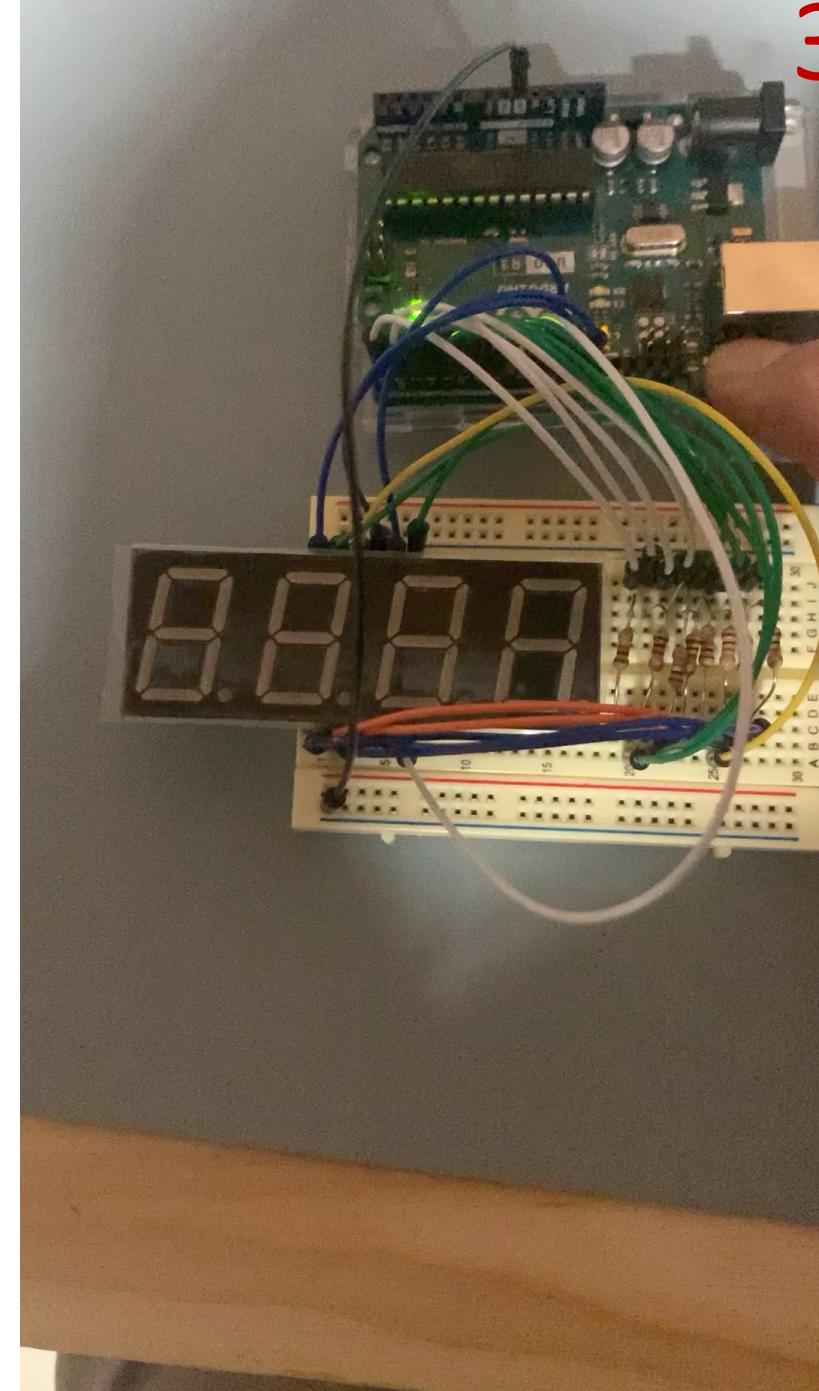
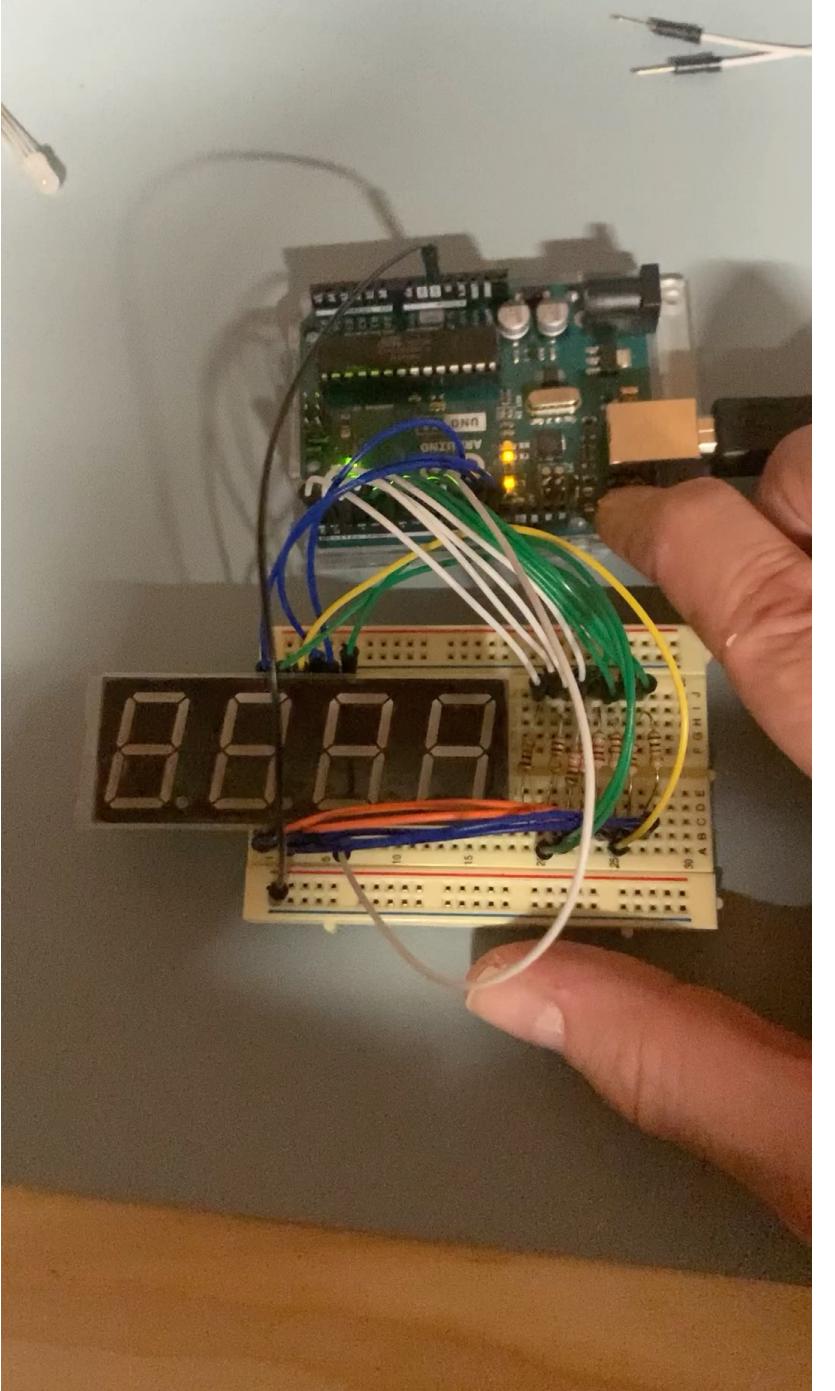
repeat

this loop
repeatedly
illuminate
each digit
every 20 msec



```
/* seven_counter_main.c This code demonstrates the use of a 4 digit  
7 segment LED. This version counts 0-1000 repeatedly and uses retinal  
persistence to create an always-on effect in the digits  
D. McLaughlin 3/16/22 ECE-231 Demo */
```

```
#include "avr/io.h"  
#include "util/delay.h"  
#define PERSISTENCE 5  
  
int main(void){  
  
    unsigned char ledDigits[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D,  
                                0x07, 0x7F, 0x67};  
    unsigned int i=0;  
    unsigned char DIG1, DIG2, DIG3, DIG4;  
  
    DDRD = 0xFF;      // 7segment pins  
    DDRB = 0xFF;      // Digit enable pins  
    PORTB = 0xFF;     // Disable all the digits initially  
  
    while (1) {  
        i++;  
        if(i>9999) i=0;  
  
        DIG4 = i%10;           // 1's digit (Least Significant Digit)  
        PORTD = ledDigits[DIG4];  
        PORTB = ~ (1<<1);      // enable 1's digit (DIG4)  
        _delay_ms(PERSISTENCE); // stay on for small amount of time  
  
        DIG3= (i/10)%10;       // 10's digit  
        PORTD = ledDigits[DIG3];  
        PORTB = ~ (1<<2);      // Enable 10's digit (DIG3)  
        _delay_ms(PERSISTENCE);  
  
        DIG2 = (i/100)%10;      // 100's digit  
        PORTD = ledDigits[DIG2];  
        PORTB = ~ (1<<3);      // Enable 100's digit (DIG2)  
        _delay_ms(PERSISTENCE);  
  
        DIG1 = (i/1000);        // 1000's digit (Most signif digit)  
        PORTD = ledDigits[DIG1];  
        PORTB = ~ (1<<4);      // Enable 1000's digit (DIG1)  
        _delay_ms(PERSISTENCE);  
    }  
}
```



```
/* seven_enable_main.c This code demonstrates the use of a 4 digit  
7 segment LED. This version counts 0-1000 repeatedly and uses  
persistence to create an always-on effect in the digits  
D. McLaughlin 3/16/22 ECE-231 Demo */
```

```
#include "avr/io.h"  
#include "util/delay.h"  
  
#define PERSISTENCE 5  
#define COUNTTIME 200 // This is the # of ms between counts  
  
int main(void)  
{  
    unsigned char ledDigits[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D,  
        0x07, 0x7F, 0x67};  
    unsigned int i=0;  
    unsigned char DIG1, DIG2, DIG3, DIG4;  
  
    DDRD = 0xFF; // 7segment pins  
    DDRB = 0xFF; // Digit enable pins  
    PORTB = 0xFF; // Initially disable the digits  
    while (1) {  
        i++;  
        if(i>9999) i=0;  
  
        DIG4 = i%10;  
        DIG3= (i/10)%10;  
        DIG2 = (i/100)%10;  
        DIG1 = (i/1000);
```

```
        for (int j=0; j<COUNTTIME/PERSISTENCE/4; j++){  
  
            PORTD = ledDigits[DIG4];  
            PORTB = ~ (1<<1);  
            _delay_ms(PERSISTENCE);  
  
            PORTD = ledDigits[DIG3];  
            PORTB = ~ (1<<2);  
            _delay_ms(PERSISTENCE);  
  
            PORTD = ledDigits[DIG2];  
            PORTB = ~ (1<<3);  
            _delay_ms(PERSISTENCE);  
  
            PORTD = ledDigits[DIG1];  
            PORTB = ~ (1<<4);  
            _delay_ms(PERSISTENCE);  
        }  
    }
```

This code combines UART & 4 digit 7 segment display

37

```
/* seven_uart_main.c This code demonstrates the use of a 4 digit
7 segment LED. This version counts 0-1000 repeatedly and uses
persistence to create an always-on effect in the digits. Digits
are also sent via UART. This code has D1 contention between UART &
the display. D. McLaughlin 3/20/22 ECE-231 Demo */

#include "avr/io.h"
#include "util/delay.h"
#define PERSISTENCE 5
#define COUNTTIME 1000           // This is the # of ms beteen counts
void uart_init(void);
void uart_send(unsigned char);

int main(void){
    unsigned char ledDigits[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D,
        0x07, 0x7F, 0x67};
    unsigned int i=0;
    unsigned char DIG1, DIG2, DIG3, DIG4;
    uart_init();
    DDRD = 0xFF;    // 7segment pins
    DDRB = 0xFF;    // Digit enable pins

    while (1) {
        i++;
        if(i>9999) i=0;

        DIG4 = i%10;          // Compute 1's digit (Least sig digit)
        DIG3= (i/10)%10;      // Compute 10's digit
        DIG2 = (i/100)%10;     // Compute 100's digit
        DIG1 = (i/1000);       // Compute 1000's digit (Most sig digit)
```

```
for (int j=0; j<COUNTTIME/PERSISTENCE/4; j++){
    PORTD = ledDigits[DIG1];           // 1000's digit (Most significant)
    uart_send(DIG1+'0');              // Tx 1000's digit
    PORTB = ~ (1<<4);               // Enable 1000's digit
    _delay_ms(PERSISTENCE);

    PORTD = ledDigits[DIG2];           // 100's digit
    uart_send(DIG2+'0');              // Tx 100's digit
    PORTB = ~ (1<<3);               // Enable 100's digit
    _delay_ms(PERSISTENCE);

    PORTD = ledDigits[DIG3];           // 10's digit
    uart_send(DIG3+'0');              // Tx 10's digit
    PORTB = ~ (1<<2);               // Enable 10's digit
    _delay_ms(PERSISTENCE);

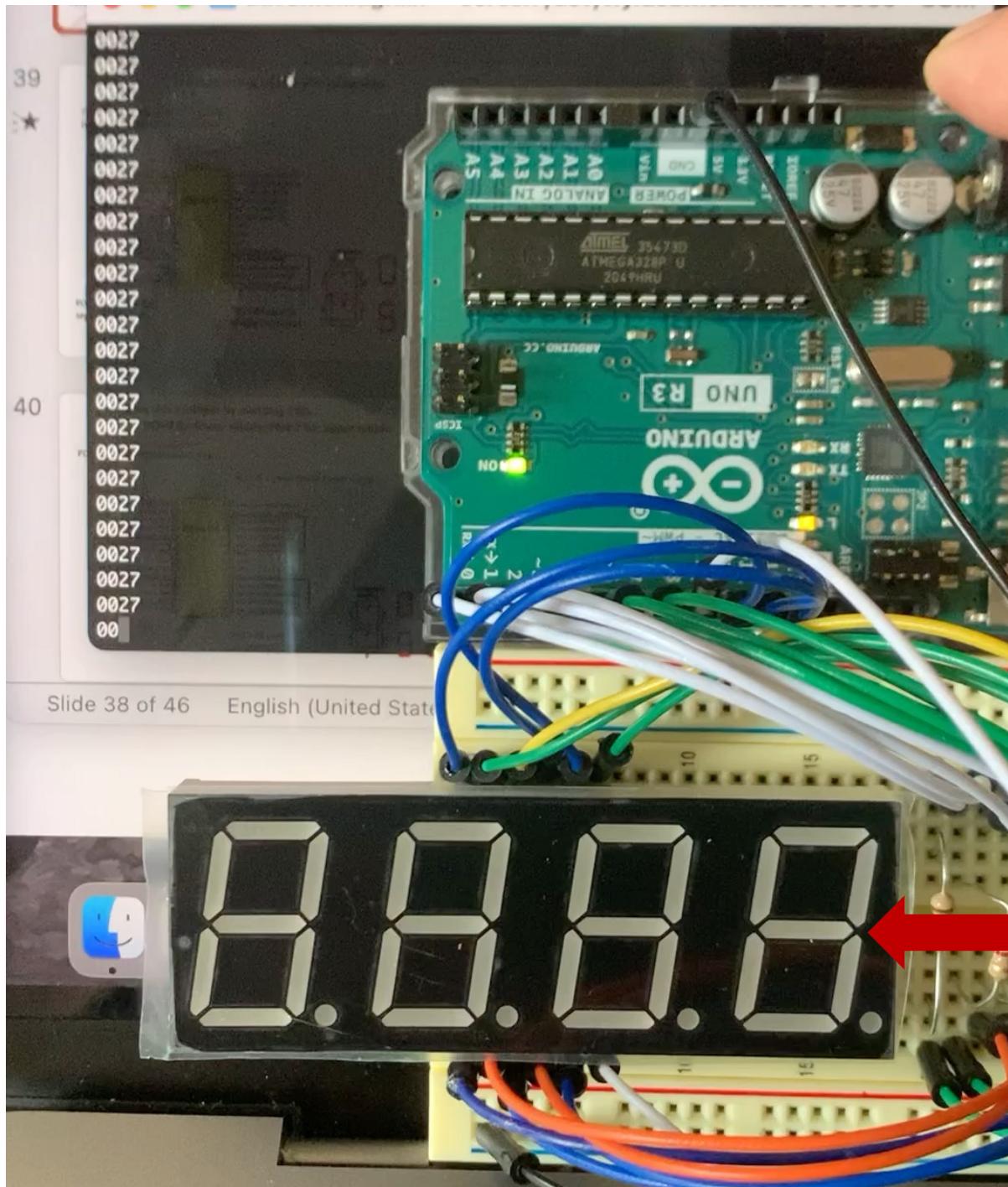
    PORTD = ledDigits[DIG4];           // 1's digit (Least sig digit)
    uart_send(DIG4+'0');              // Tx 1's digit
    PORTB = ~ (1<<1);               // Enable 1's digit
    _delay_ms(PERSISTENCE);

    PORTB = 0xFF;                     // Disable all digits

    uart_send(13); // Carriage return (goto beginning of line)
    uart_send(10); //line feed (new line)
}

> void uart_init(void){...}

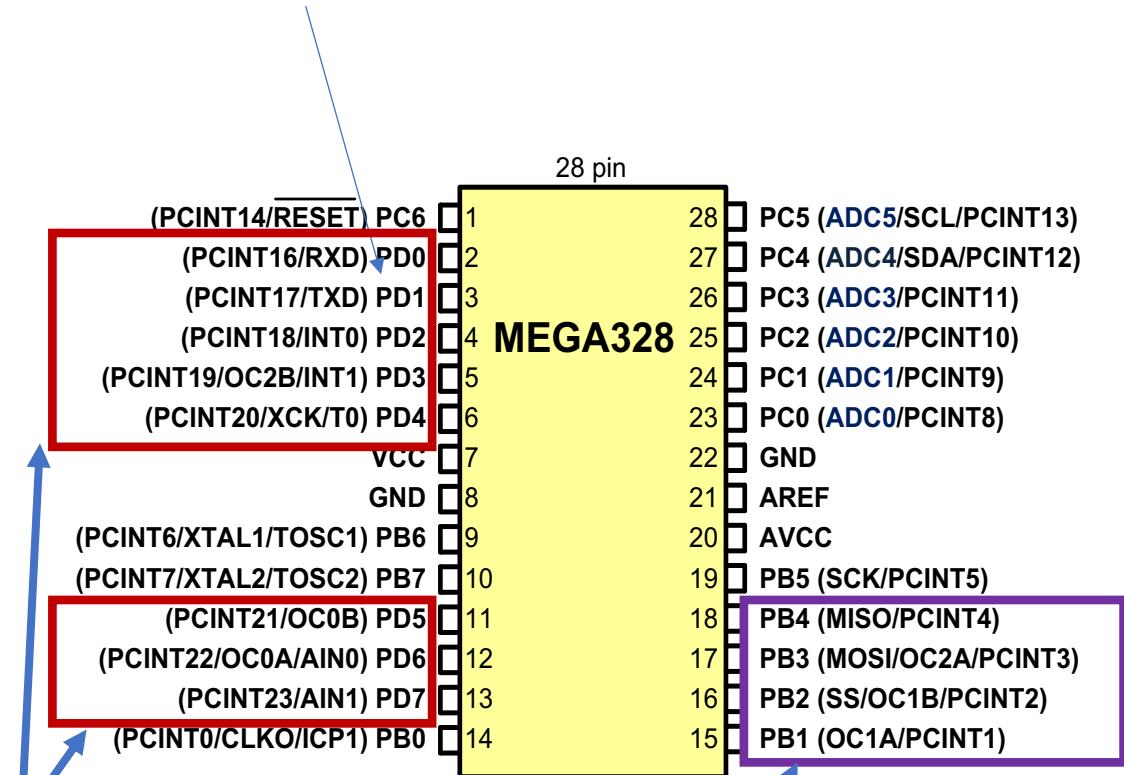
> void uart_send(unsigned char ch){...}
```



pay attention to
LED segment b

GPIO PORTD & USART wire contention

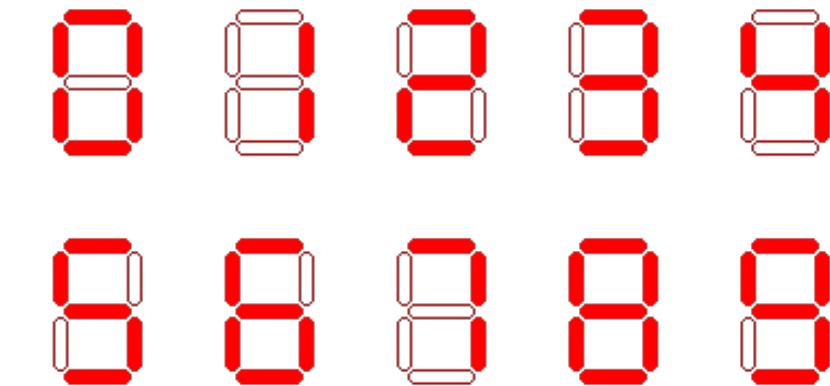
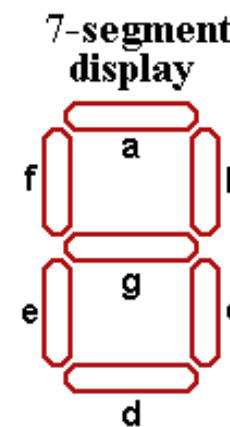
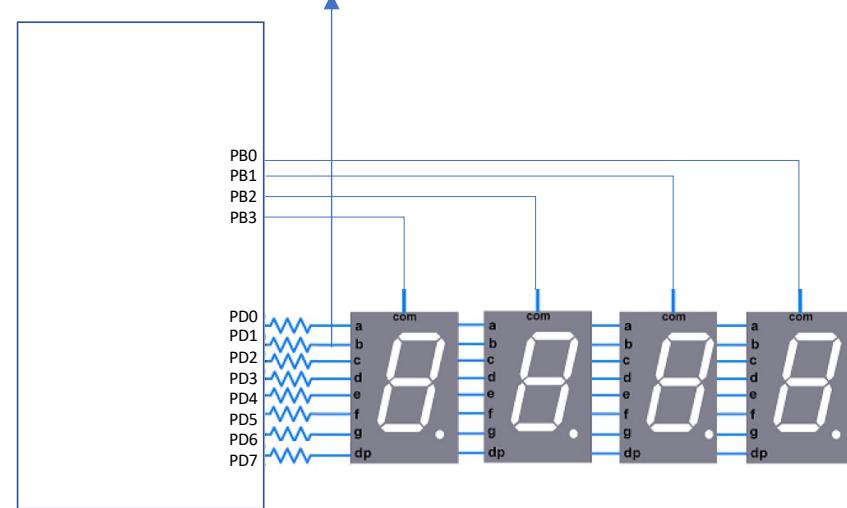
TXD (USART) uses same wire as PD1
(wired to segment b of the display)



PD0-PD7 used to for
segments a-g & dp

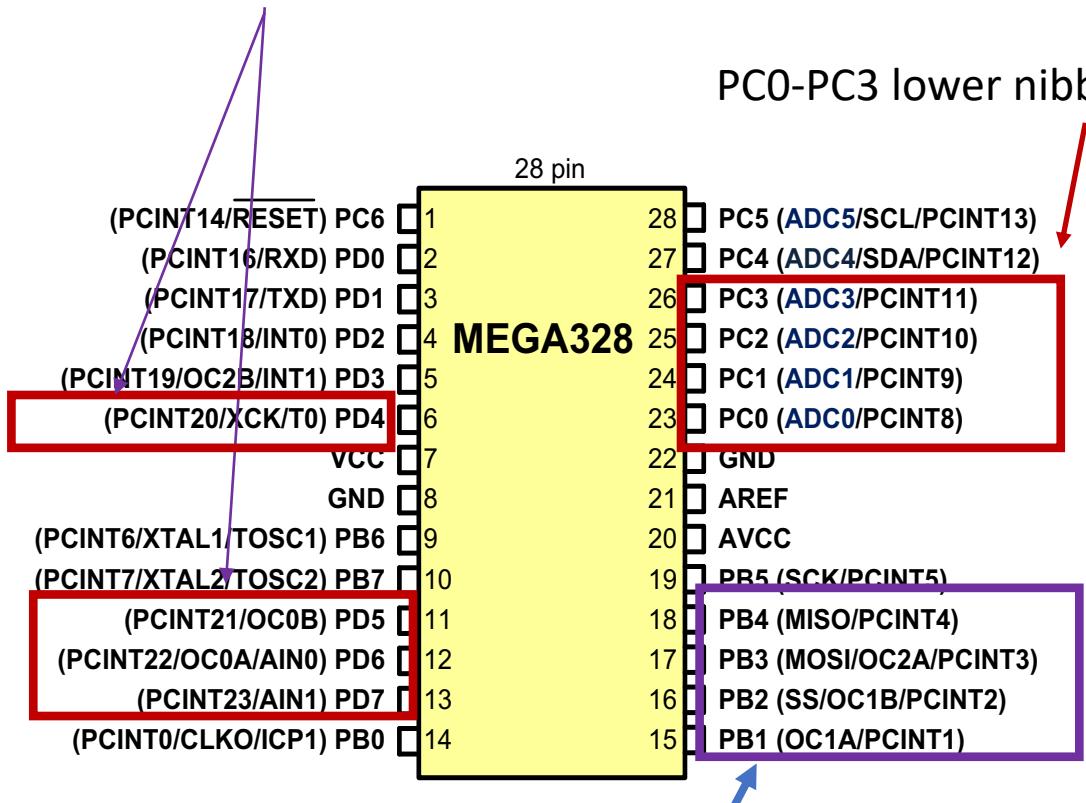
PB1-PB4 used to
enable DIG1-DIG4

TXD from USART to
UART/USB converter



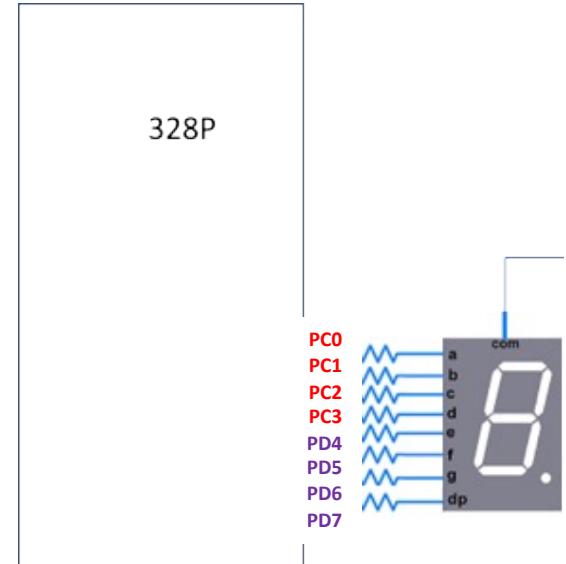
Solve this problem by avoiding PD0.
Use PC0-3 for lower nibble; PD4-7 for upper nibble

PD4-PD7 upper nibble (upper 4 bits)

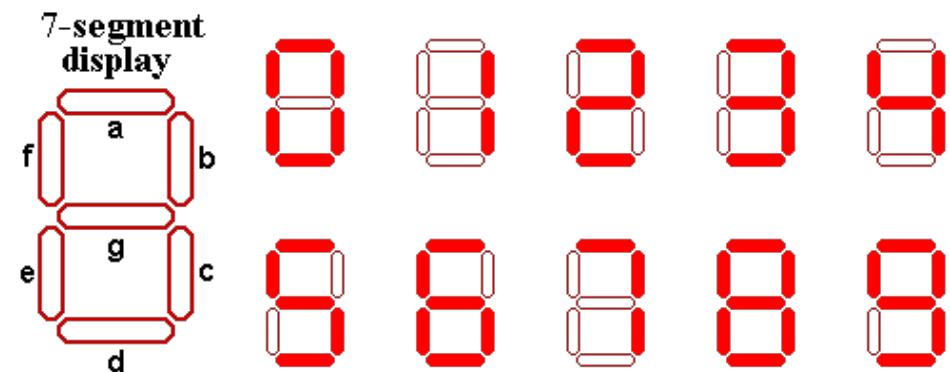


PC0-PC3 lower nibble (lower 4 bits)

PB1-PB4 used to enable DIG1-DIG4



Common Cathode 7 Segment LED



seven_uart_corrected.c □ COUNTTIME

```
/* seven_uart_corrected.c Demonstrates simultaneous display
of a 4 digit ring counter (0000-9999) to UART & 4 digit, 7
segment LED. PC0-3 & PD4-7 used for segments a-d, e-dp.
D. McLaughlin 3/19/22 ECE-231 Demo */

#include "avr/io.h"
#include "util/delay.h"
#define PERSISTENCE 5
#define COUNTTIME 50          // # ms between counts
void uart_init(void);
void uart_send(unsigned char);

int main(void){
    unsigned char ledDigits[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D,
        0x07, 0x7F, 0x67};
    unsigned int i=0;
    unsigned char DIG1, DIG2, DIG3, DIG4;
    uart_init();    // Initialize the UART
    DDRC = 0x0F;    // Segments a-d use PC0-PC3
    DDRD = 0xF0;    // Segments e-g & dp use PD4-PD7
    DDRB = 0xFF;    // Digit enable pins

    while (1) {
        i++;
        if(i>9999) i=0;

        DIG4 = i%10;      // 1's digit (least significant digit)
        DIG3= (i/10)%10; // 10's digit
        DIG2 = (i/100)%10; // 100's digit
        DIG1 = (i/1000); // 1000's digit (most significant digit)

        for (int j=0; j<COUNTTIME/PERSISTENCE/4; j++){

```

```
for (int j=0; j<COUNTTIME/PERSISTENCE/4; j++){

    // Show 1000's digit
    PORTC = ledDigits[DIG1];
    PORTD = ledDigits[DIG1];
    uart_send(DIG1+'0');           // Tx 1000's digit
    PORTB = ~ (1<<4);           // Enable DIG1
    _delay_ms(PERSISTENCE);

    // Show 100's digit
    PORTC = ledDigits[DIG2];
    PORTD = ledDigits[DIG2];
    uart_send(DIG2+'0');           // Tx 100's digit
    PORTB = ~ (1<<3);           // Enable DIG2
    _delay_ms(PERSISTENCE);

    // Show 10's digit
    PORTC = ledDigits[DIG3];
    PORTD = ledDigits[DIG3];
    uart_send(DIG3+'0');           // Tx 100's digit
    PORTB = ~ (1<<2);           // Enable DIG3
    _delay_ms(PERSISTENCE);

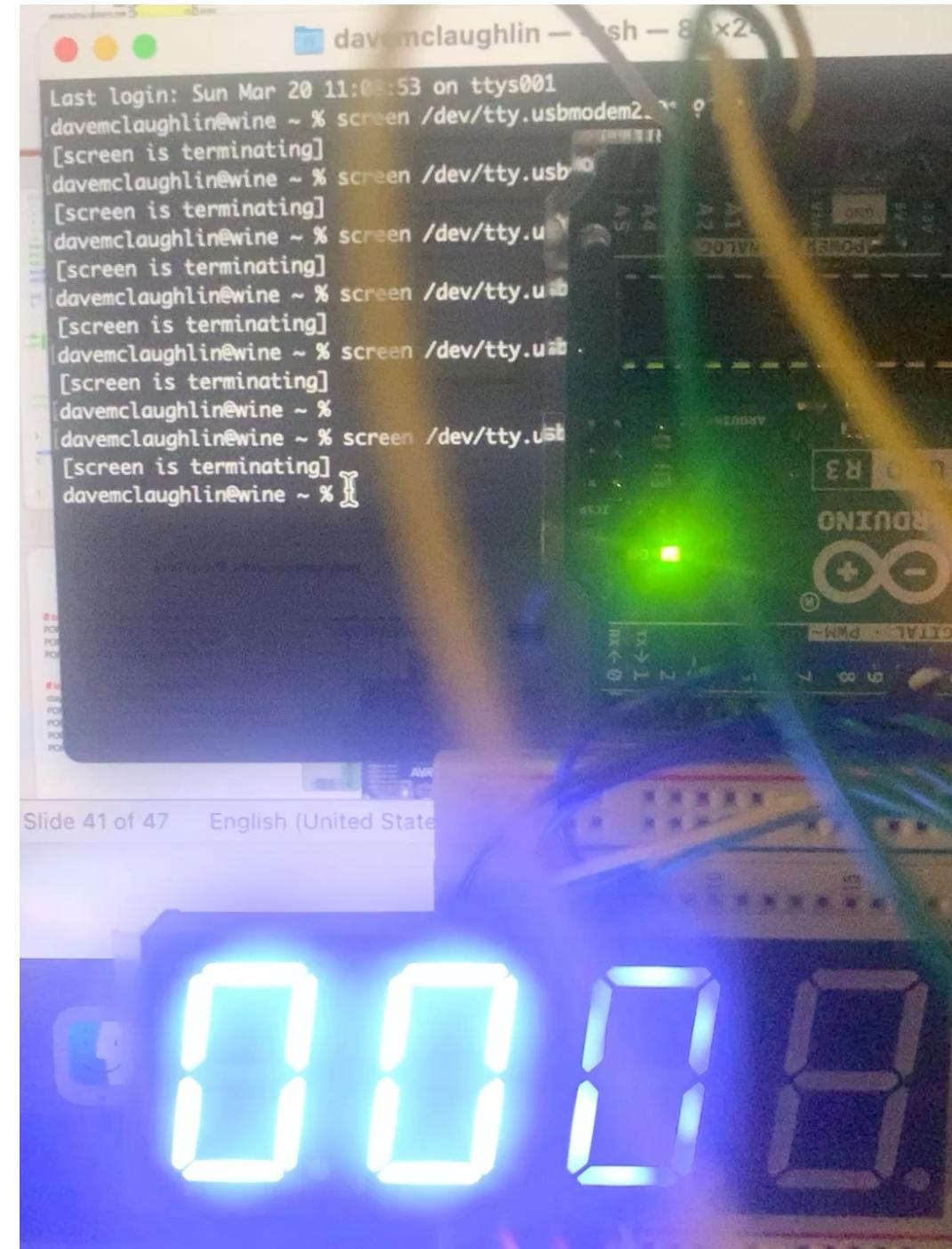
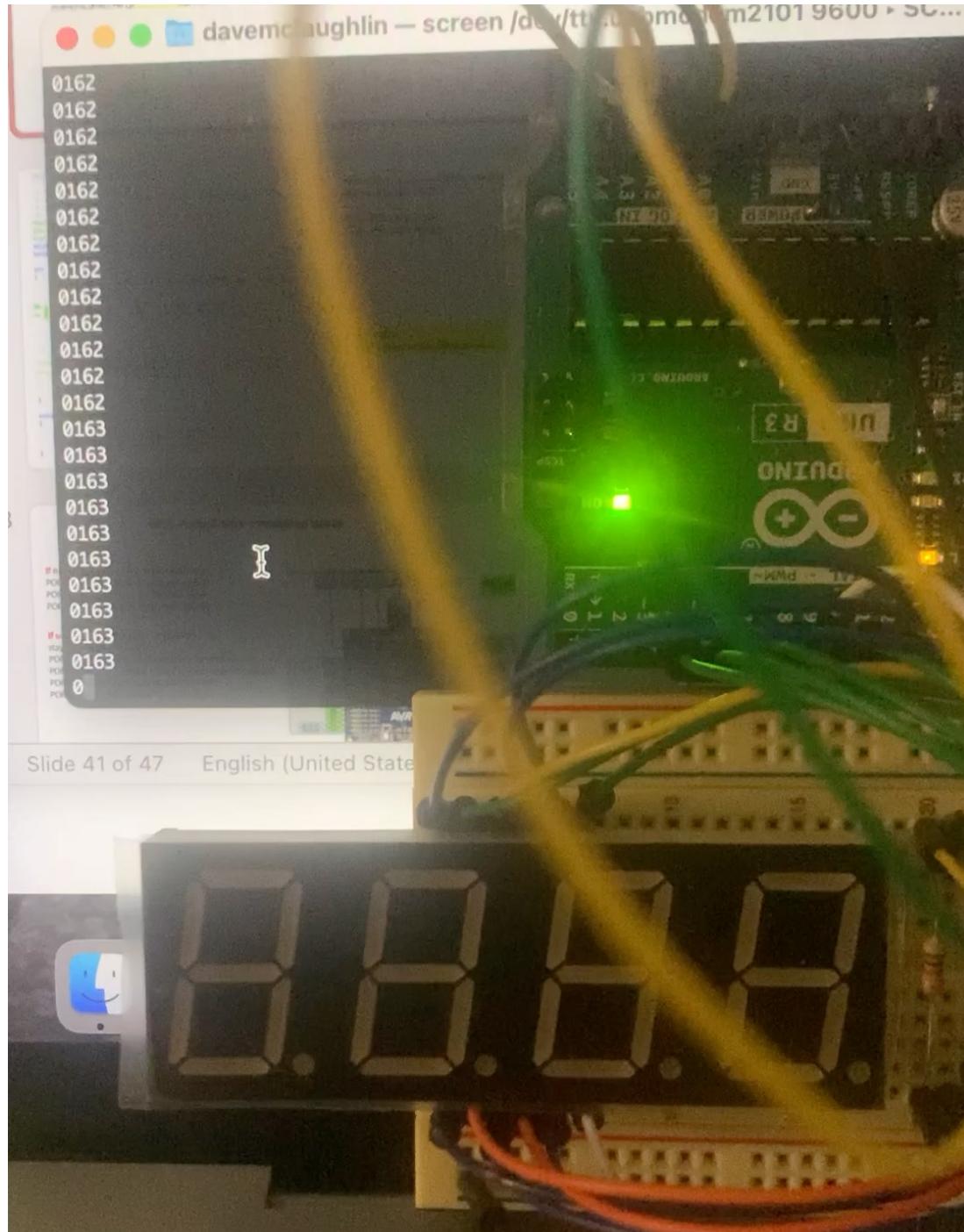
    // Show 1's digit
    PORTC = ledDigits[DIG4];
    PORTD = ledDigits[DIG4];
    uart_send(DIG4+'0');           // Tx 10's digit
    PORTB = ~ (1<<1);           // Enable DIG4
    _delay_ms(PERSISTENCE);

    PORTB = 0xFF;                  // Disable all digits
    uart_send(13);                 // Tx carriage return
    uart_send(10);                 // Tx line feed
}

> void uart_init(void){ ...
> void uart_send(unsigned char ch){ ...

```

42



GPIO PORTS

```
int main(void) {
    DDRB = 0xFF; //set PB-PB7 as outputs
    DDRD = 0x00; //set PD0-PD7 as inputs
    PORTB = 1<<5; // set PB5 (builtin LED) high
    PORTD = 1<<0|1<<1|1<<2; //set pullups on PD0, PD1, PD2
    while (1) {
        ...
        if ((PIND & (1<<1))==0) //check if button pressed
            PORTB ^= 1<<5; //turn on LED
        ...
    }
}
```

USART

```
int main(void) {
    UCSRB = (1 << TXEN0); //enable USART tx
    UCSRC = (1 << UCSZ01) | (1 << UCSZ00); //8 bit char size
    UBRR0L = 103; //9600 baud
    while (1) {
        ...
        while (!(UCSRA & (1 << UDRE0))); //wait till tx buffer empty
        UDR0 = ch; //tx ascii character
        ...
    }
}
```

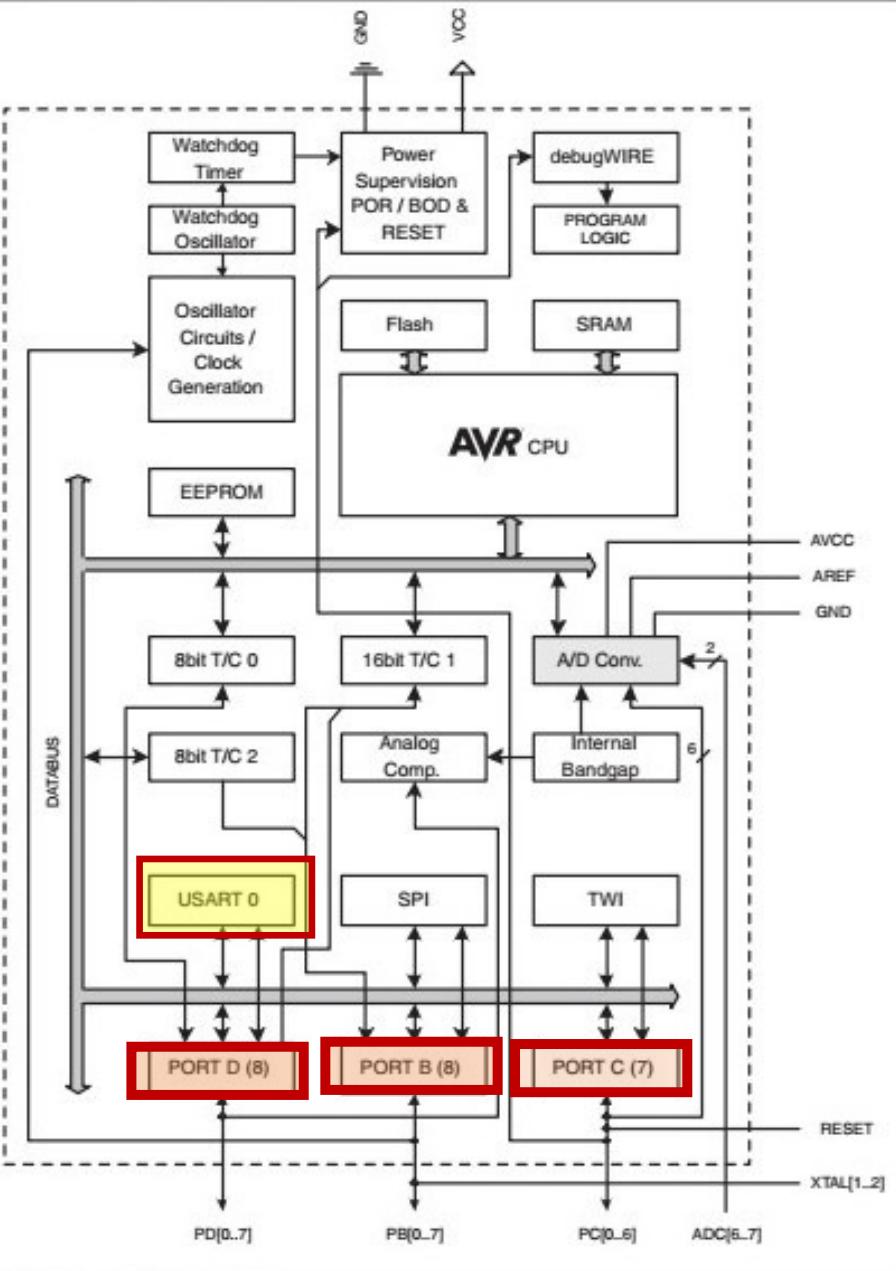


Figure 1-4. ATmega328 Block Diagram

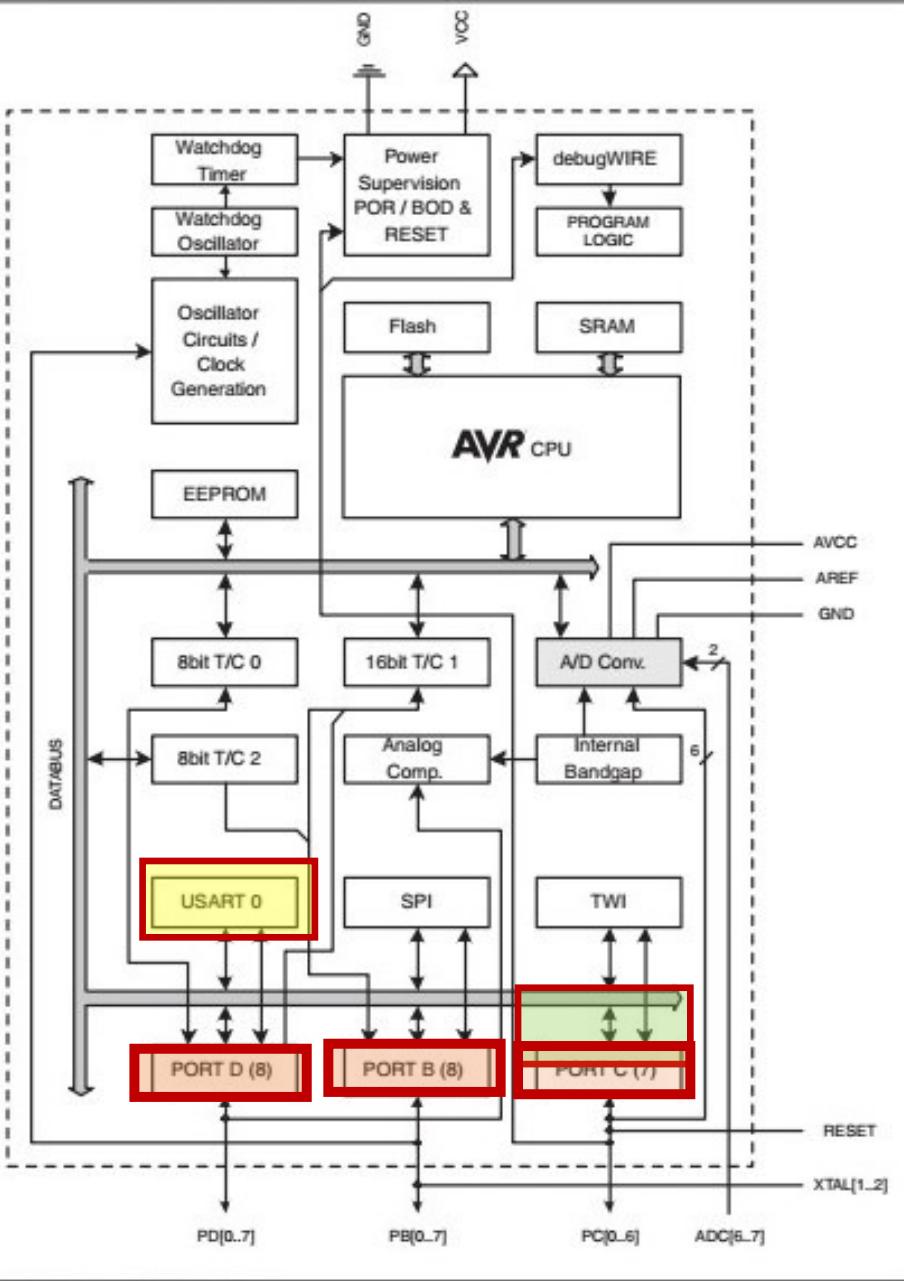


Figure 1-4. ATmega328 Block Diagram

What's next?

Analog to Digital Converter (ADC)

ADC

```

int main(void) {
    unsigned int digitalValue;
    DDRC = 0x00;
    ADMUX = 0xC0;
    ADCSRA = 0x87;
    while (1) {
        ...
        ADCSRA |= (1 << ADSC);
        while ((ADCSRA & (1 << ADIF)) == 0);
        digitalValue = ADCL | (ADCH << 8);
        ...
    }
}
//set PD0 = ADC0 as input
//select ADC0; Vref=1.1V
//enable ADC; speed 125 KHz
//start ADC conversion
//wait till finished converting
//read the ADC digital value

```