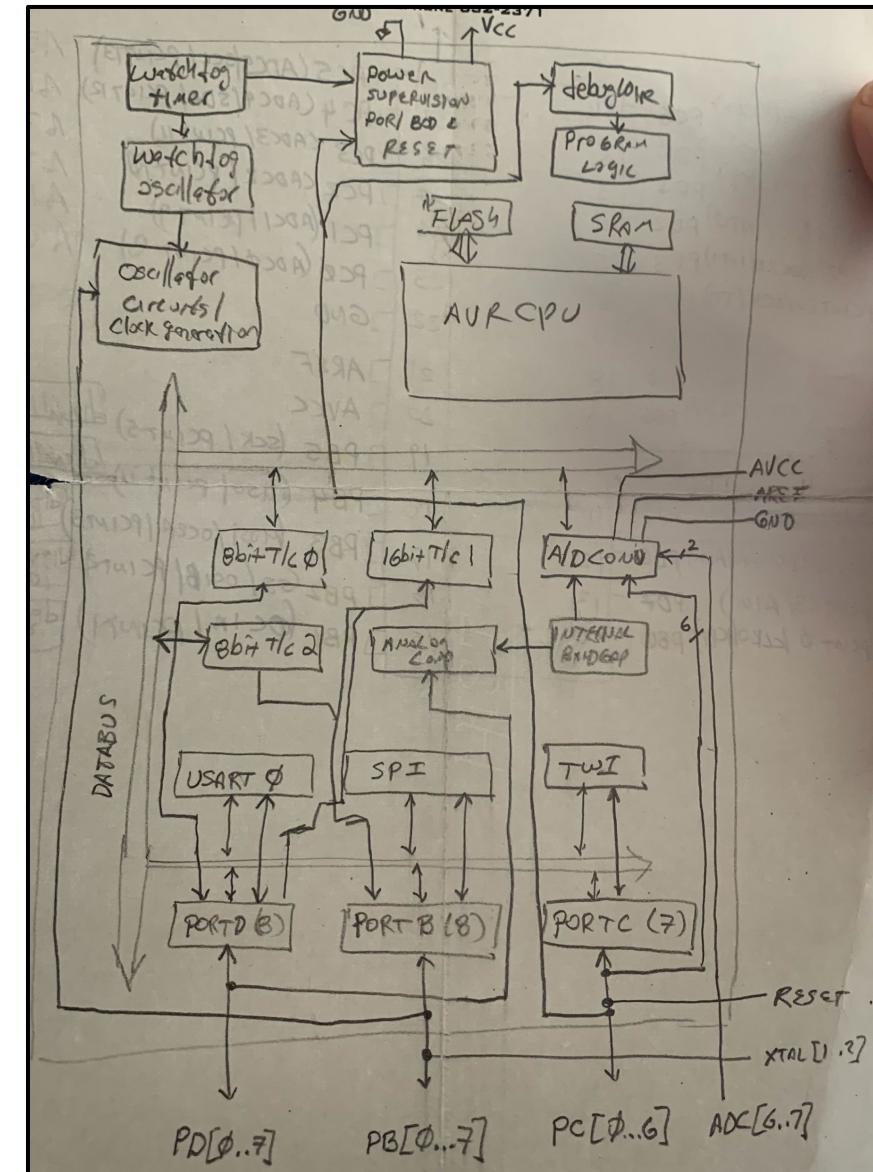


# 8.7 & 8.8

## Timers/Counters

Prof. David McLaughlin  
ECE Dept  
UMass Amherst



# Timers/Counters on the ATmega328P

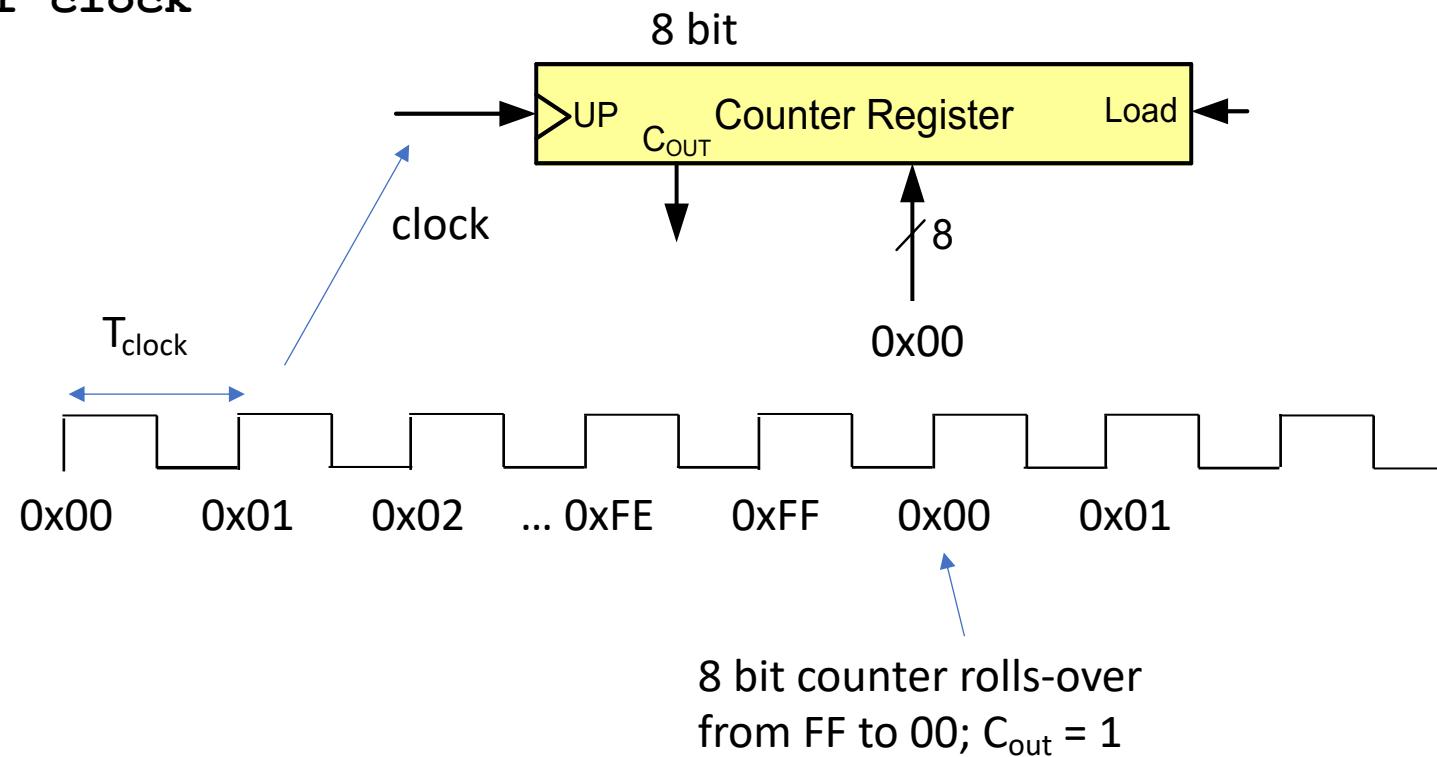
- *They function as timers:* counting specified numbers of cycles of known clock frequency to create delays & waveforms
- *They function as counters:* counting external triggers that cause transitions on external pins to capture & measure events
- 3 Timer/Counters on our ATmega328P MCU:
  - Timer/Counter0 -- 8 bit timer
    - TCNT0 - 8 bit register that counts 0 - 255
  - Timer/Counter1 -- a 16 bit timer
    - TCNT1 – a 16 bit register, TCNT1H, TCNT1L (counts 0 – 65,535)
  - Timer/Counter2 – an 8 bit timer
    - TCNT2 - 8 bit register that counts 0 - 255

we will use the terms timer & counter to each refer to the timers/counters in the MCU

# Let's build a delay timer (1 of 2)

pseudo-code:

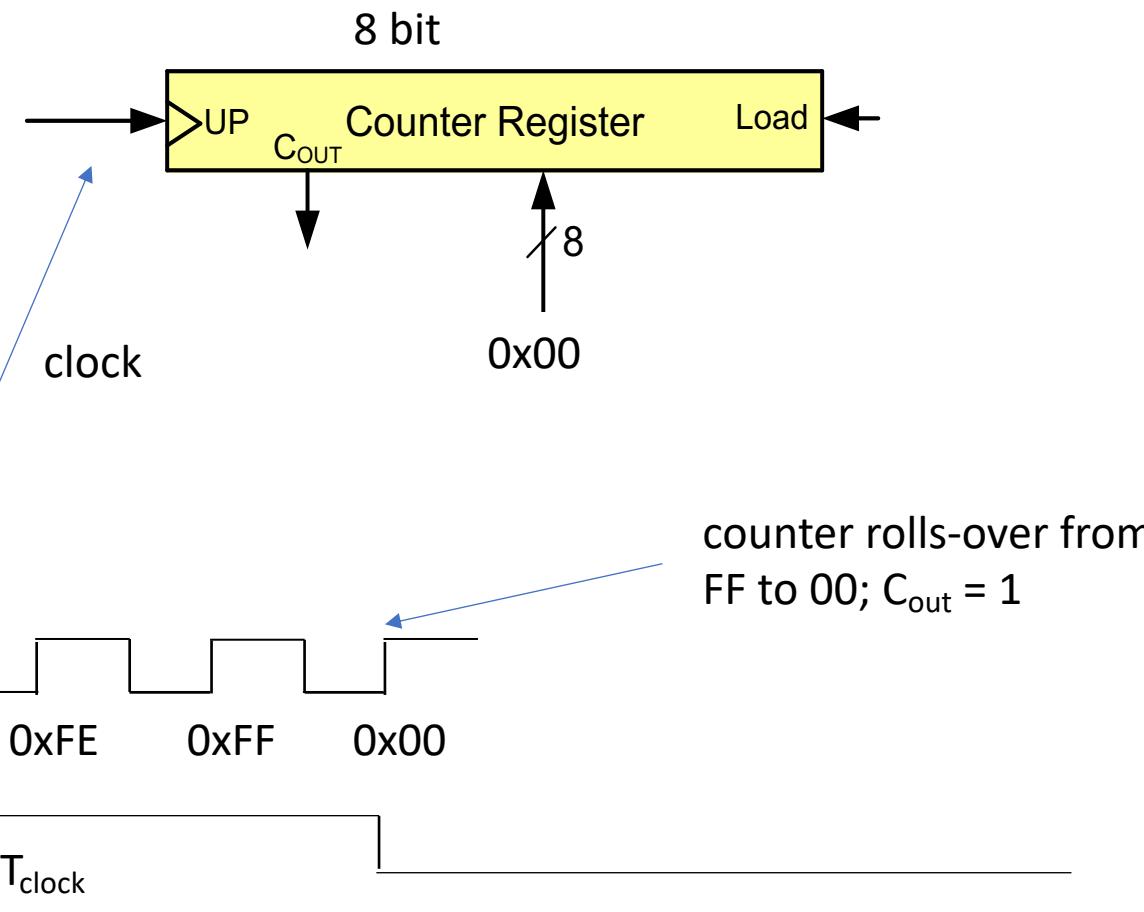
```
load 0x00 into counter  
start counter clock
```



# Let's build a delay timer (2 of 2)

pseudocode to turn on an led  
for an amount of time

```
make ledpin high
load 0x00 into counter
start counter clock
wait till Cout=1
stop counter clock
make ledpin low
```

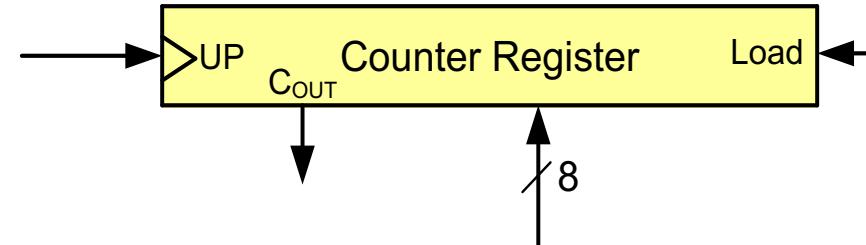


led remains on for  $256 \times T_{clock}$  seconds

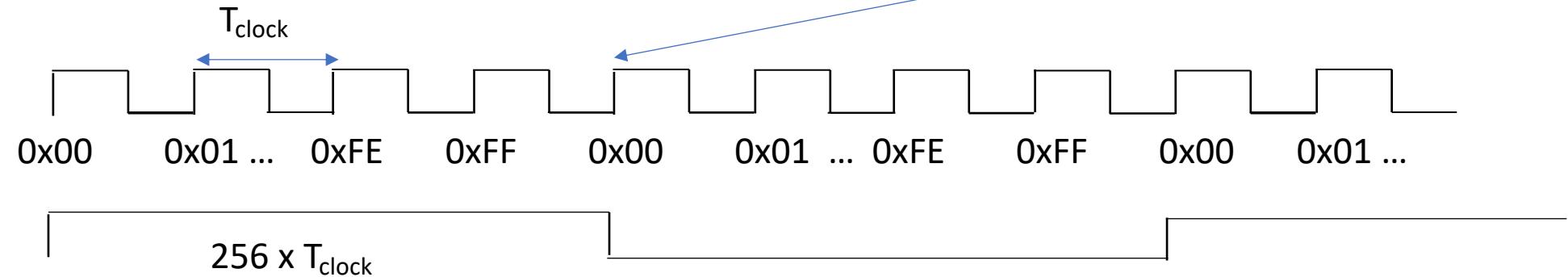
# Let's build a square wave

pseudocode to blink an led on and off at a certain rate

```
make ledPin high
loop{
    load 0x00 into counter
    start counter clock
    wait till Cout=1
    set Cout back to 0
    toggle ledPin
}
```



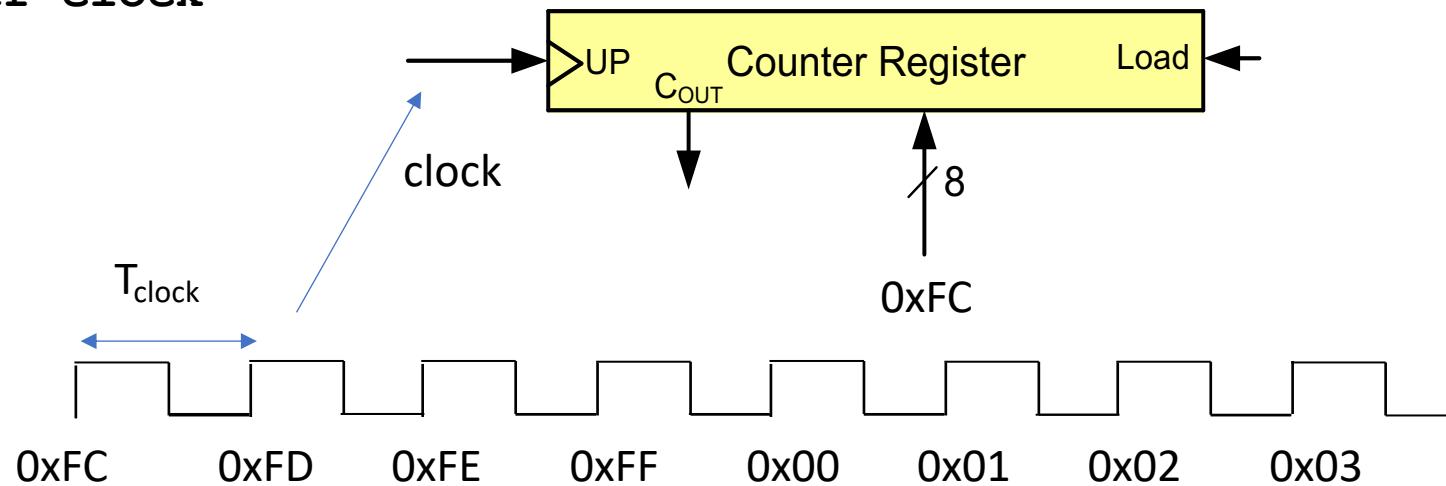
counter rolls-over from FF to 00; C<sub>out</sub> = 1  
it gets reset back to 0



square wave toggles the led every  $256 \times T_{clock}$  seconds

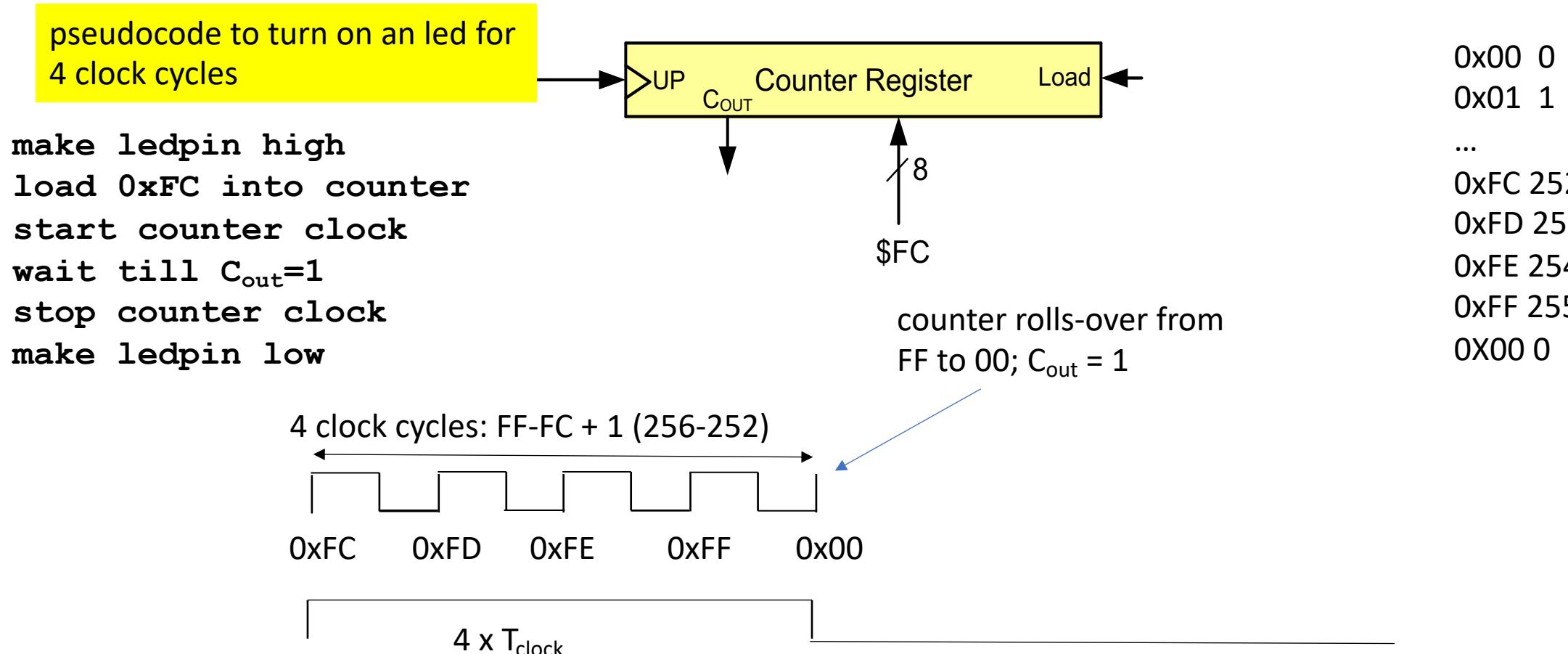
# Let's build a 4 cycle (or 4 count) delay (1 of 2)

load 0xFC into counter  
start counter clock



counter rolls-over from  
FF to 00; C<sub>out</sub> = 1

# Let's build a 4 cycle (or 4 count) delay (2 of 2)



led remains on for  $4 \times T_{clock}$  seconds

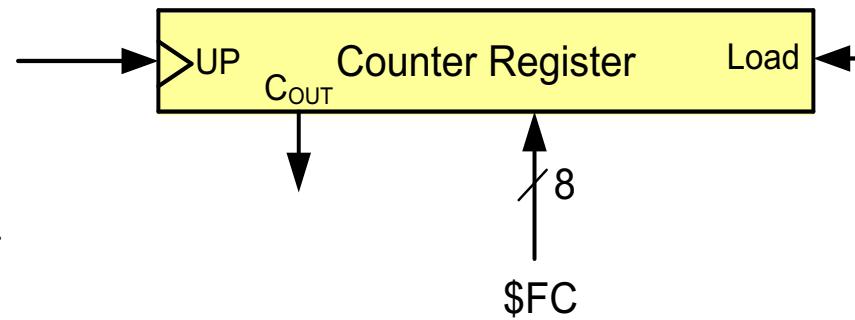
# Let's build a 4 cycle (or 4 count) square wave

pseudocode to blink an led on and off every 8 clock cycles

```

make ledPin high
loop{
    load 0xFC into counter
    start counter clock
    wait till Cout=1
    set Cout back to 0
    toggle ledPin
}

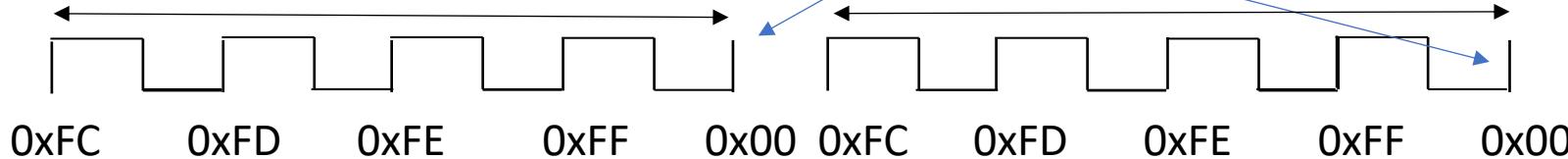
```



0x00	0
0x01	1
...	
0xFC	252
0xFD	253
0xFE	254
0xFF	255

counter rolls-over from FF to 00; C<sub>out</sub> = 1

4 clock cycles: FF-FC + 1 (256-212)



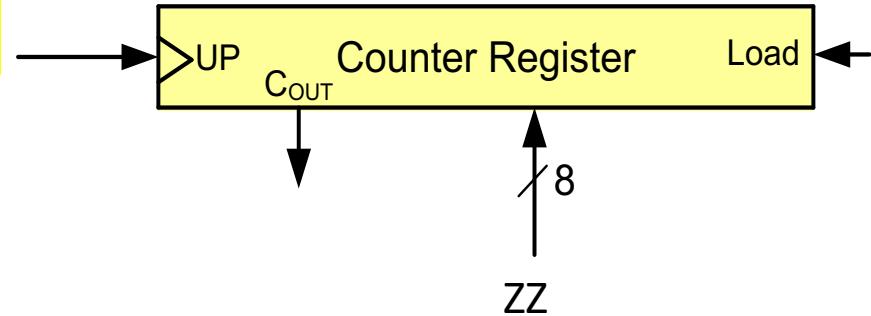
4 x T<sub>clock</sub>

square wave toggles the led every 4 x T<sub>clock</sub> seconds

# Here is an N cycle (or N count) oscillator

pseudocode to blink an led on and off every  $2N$  clock cycles

```
make ledPin high
loop{
    load ZZ into counter
    start counter clock
    wait till Coutt=1
    set Cout back to 0
    toggle ledPin
}
```

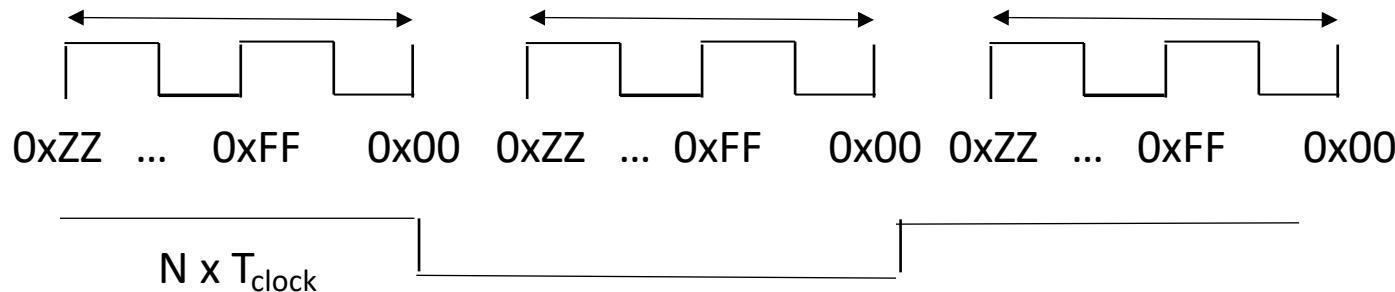


counter rolls-over from FF to 00;  $C_{out} = 1$

Hexadecimal:  
 $N = FF - ZZ + 1$   
 $ZZ = FF - N + 1$

Decimal:  
 $N_{10} = 255 - ZZ_{10} + 1$   
 $ZZ_{10} = 255 - N_{10} + 1$

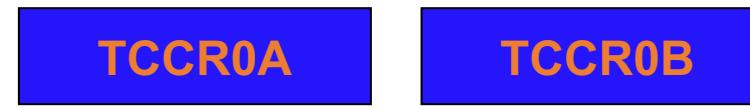
$N$  clock cycles:  $N = FF - ZZ + 1$  ( $N_{10} = 255 - ZZ_{10} + 1$ )



square wave toggles the led every  $N \times T_{clock}$  seconds

# Timer/Counter 0 (8-bit; simplified)

- TCNT0 (Timer/Counter register)
- TOV0 (Timer Overflow flag)
- TCCR0A (Timer Counter control register A)
- TCCR0B (Timer Counter control register B)



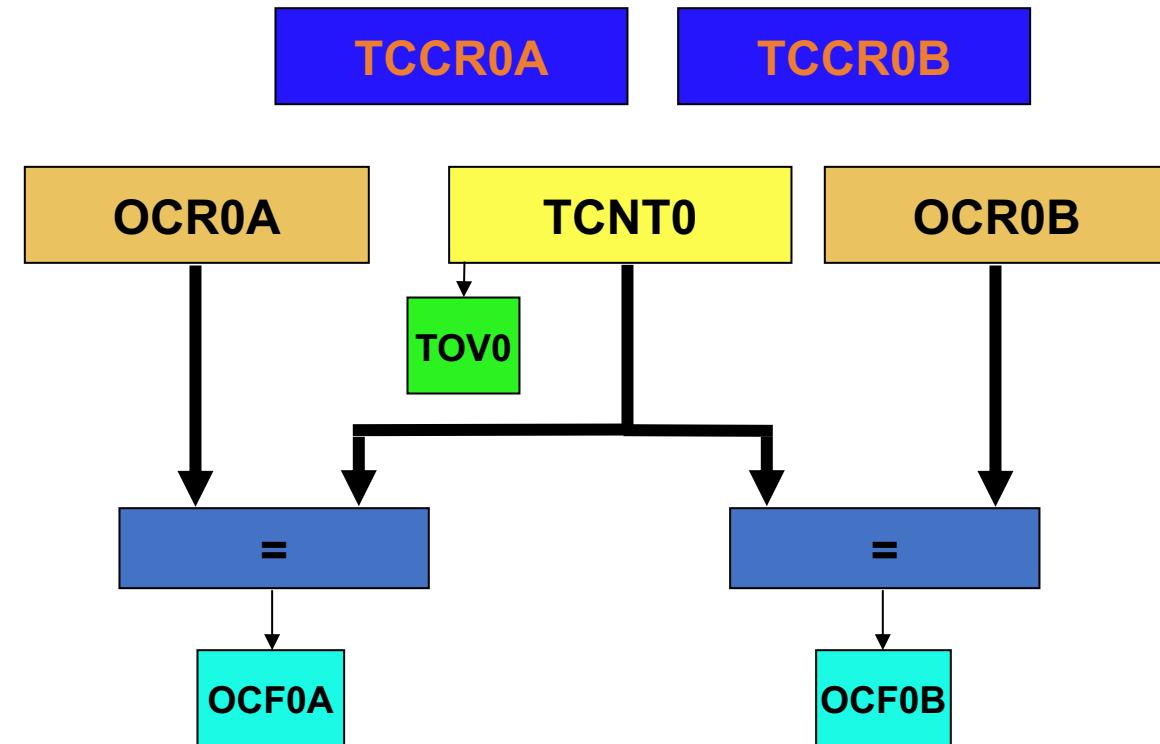
<b>TCCR0A</b>	<b>Timer Counter Control Register 0 A</b>
<b>TCCR0B</b>	<b>Timer Counter Control Register 0 B</b>
<b>TIFR0</b>	<b>Timer Interrupt Flag Register 0</b>

The table details the bit fields for each register:

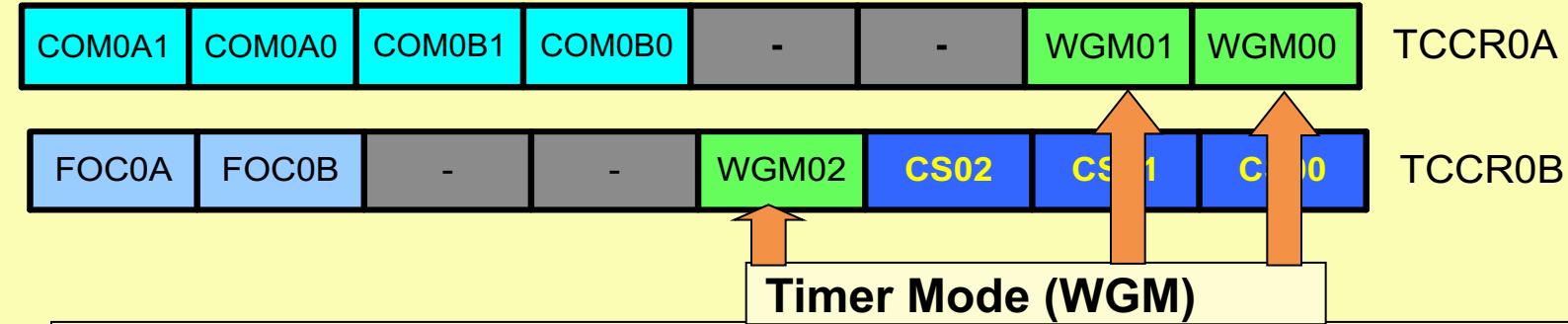
- TCCR0A:** Bits include COM0A1, COM0A0, COM0B1, COM0B0, -, -, WGM01, and WGM00.
- TCCR0B:** Bits include FOC0A, FOC0B, -, -, WGM02, CS02, CS01, CS00.
- TIFR0:** Bits include -, -, -, -, OCF0B, OCF0A, and TOV0.

# Timer/Counter 0 (8-bit)

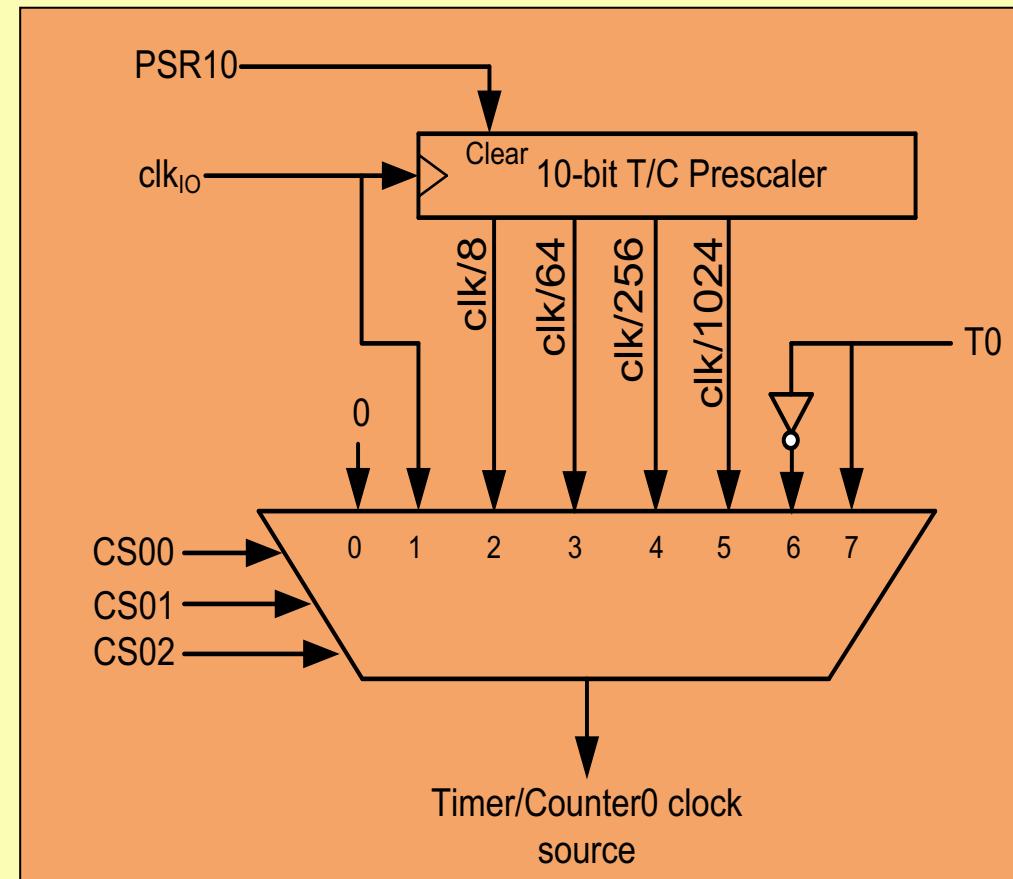
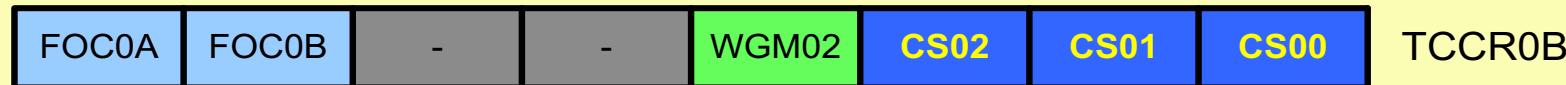
- TCNT0 (Timer/Counter register)
- TOV0 (Timer Overflow flag)
- TCCR0 (Timer Counter control register)
- OCR0 (output compare register)
- OCF0 (output compare match flag)

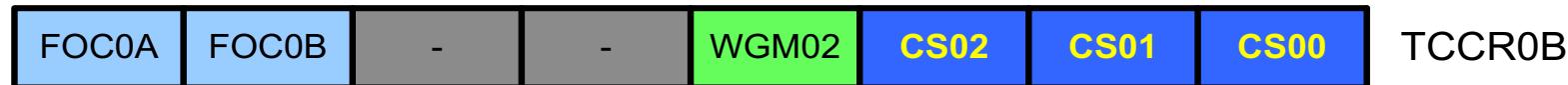


COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00	TCCR0A	Timer Counter Control Register 0 A
FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00	TCCR0B	Timer Counter Control Register 0 B
-	-	-	-	-	OCF0B	OCF0A	TOV0	TIFR0	Timer Interrupt Flag Register 0



WGM02	WGM01	WGM00	Comment
0	0	0	Normal
0	0	1	Phase correct PWM
0	1	0	CTC (Clear Timer on Compare Match)
0	1	1	Fast PWM
1	0	0	Reserved
1	0	1	Phase correct PWM
1	1	0	Reserved
1	1	1	Fast PWM





### Clock Selector (CS)

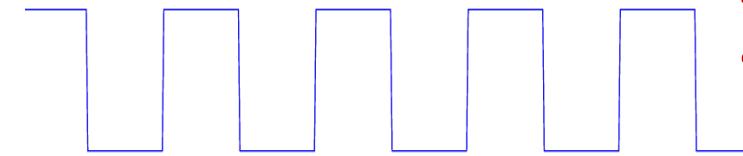
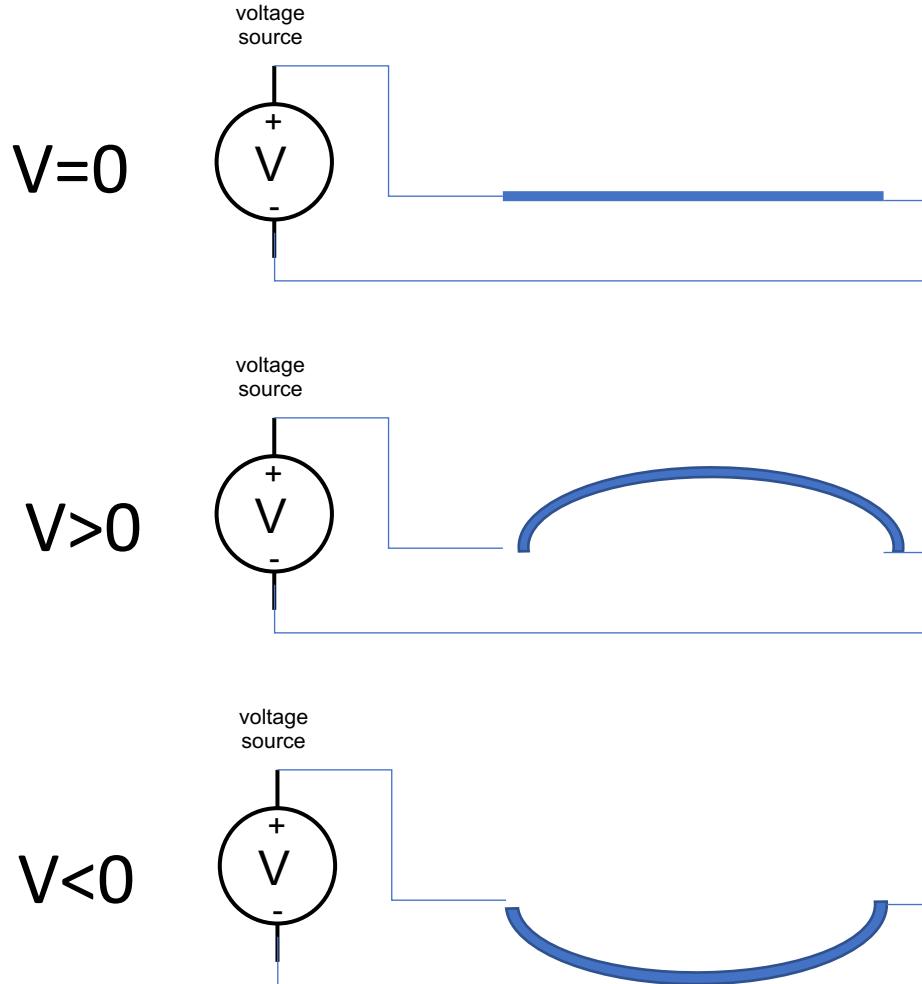
CS02	CS01	CS00	Comment
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk (No Prescaling)
0	1	0	clk / 8
0	1	1	clk / 64
1	0	0	clk / 256
1	0	1	clk / 1024
1	1	0	External clock source on T0 pin. Clock on falling edge
1	1	1	External clock source on T0 pin. Clock on rising edge

**Write a program to create a 1 KHz square wave applied to a piezo-electric speaker**

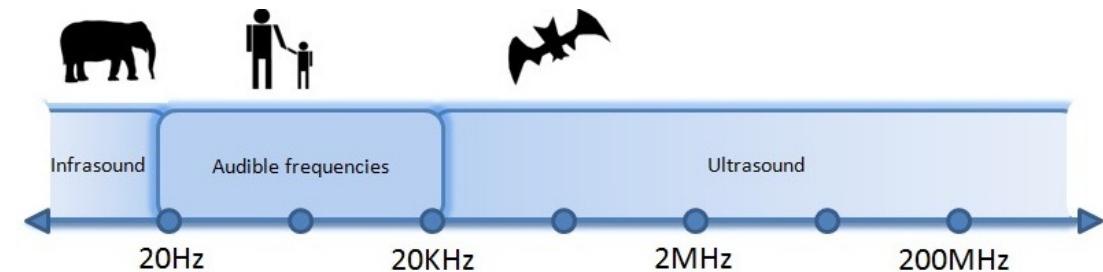
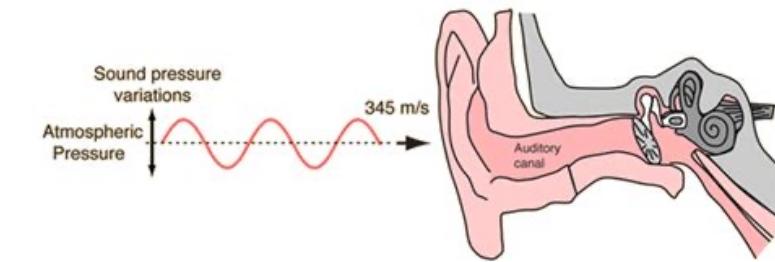


# piezoelectric speaker

16



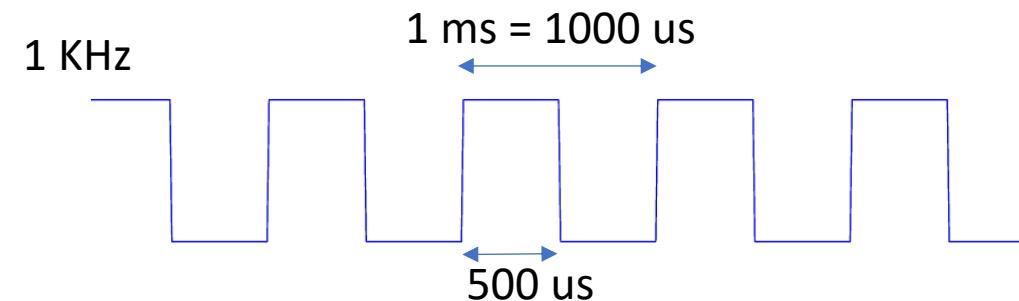
oscillating voltage waveform



# Write a program to create a 1 KHz square wave applied to a piezo-electric speaker

17

using Timer0 in **normal mode** and the **1:64 pre-scaler** to create the delay



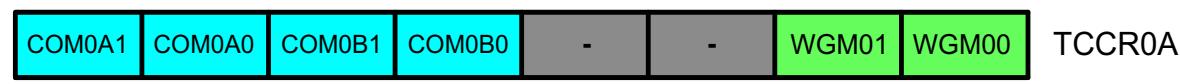
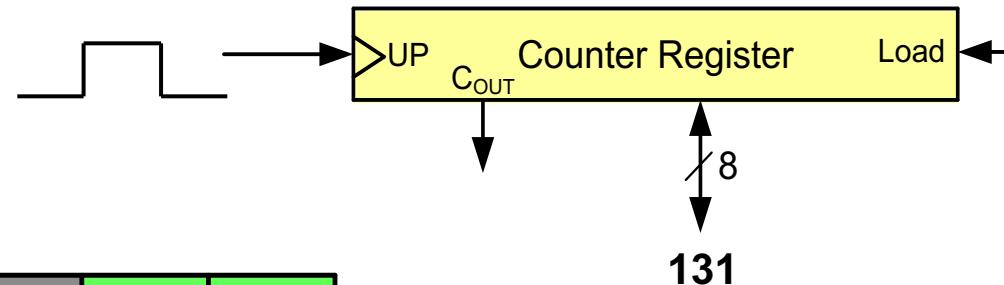
1. set up the GPIO bit
2. Pre-load counter  $TCNT0 = xxxx$
3. Set operating mode (normal)
4. Start 1:64 clock
5. Toggle the GPIO bit

1 KHz wave = 1 msec period, Tdelay = 0.5 ms = 500 us  
328P clock on Arduino Uno:  $F_{osc} = 16MHz$ ;  $T_{osc} = 1/F_{osc} = 0.0625 \mu s$   
Prescaler 1:64  $\rightarrow T_{clock} = 64 * 0.0625 \mu s = 4 \mu s$   
# cycles needed per count-up:  $500 \mu s / 4 \mu s = 125$   
preload:  $256 - 125 = 131$

Pre-load counter:

$$256 - 125 = 131$$

**TCNT0 = 131;**



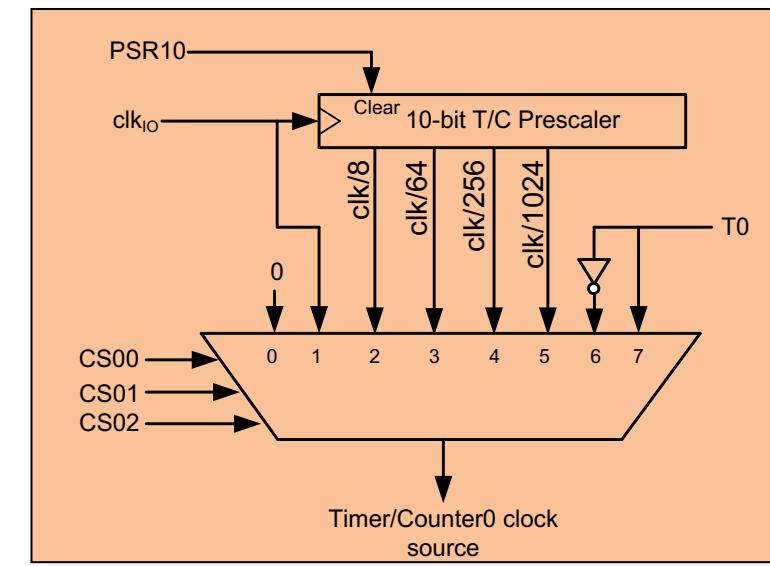
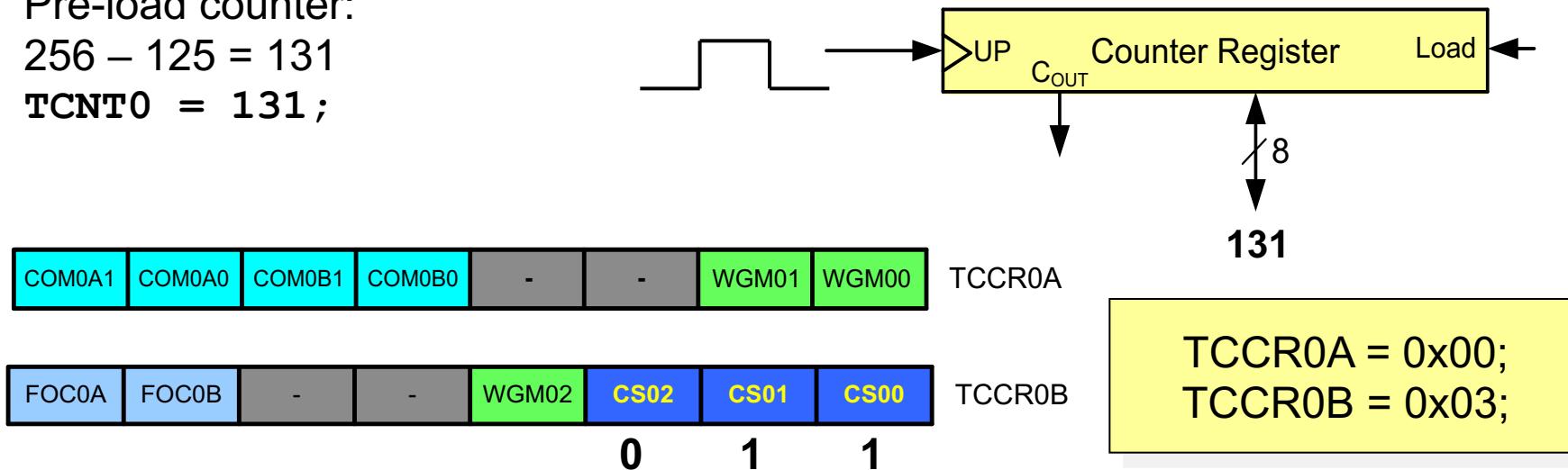
TCCR0A = 0x00;  
TCCR0B = 0x03;

WGM02	WGM01	WGM00	Comment
0	0	0	Normal
0	0	1	Phase correct PWM
0	1	0	CTC
0	1	1	Fast PWM
1	0	0	Reserved
1	0	1	Phase correct PWM
1	1	0	Reserved
1	1	1	Fast PWM

Pre-load counter:

$$256 - 125 = 131$$

**TCNT0 = 131;**

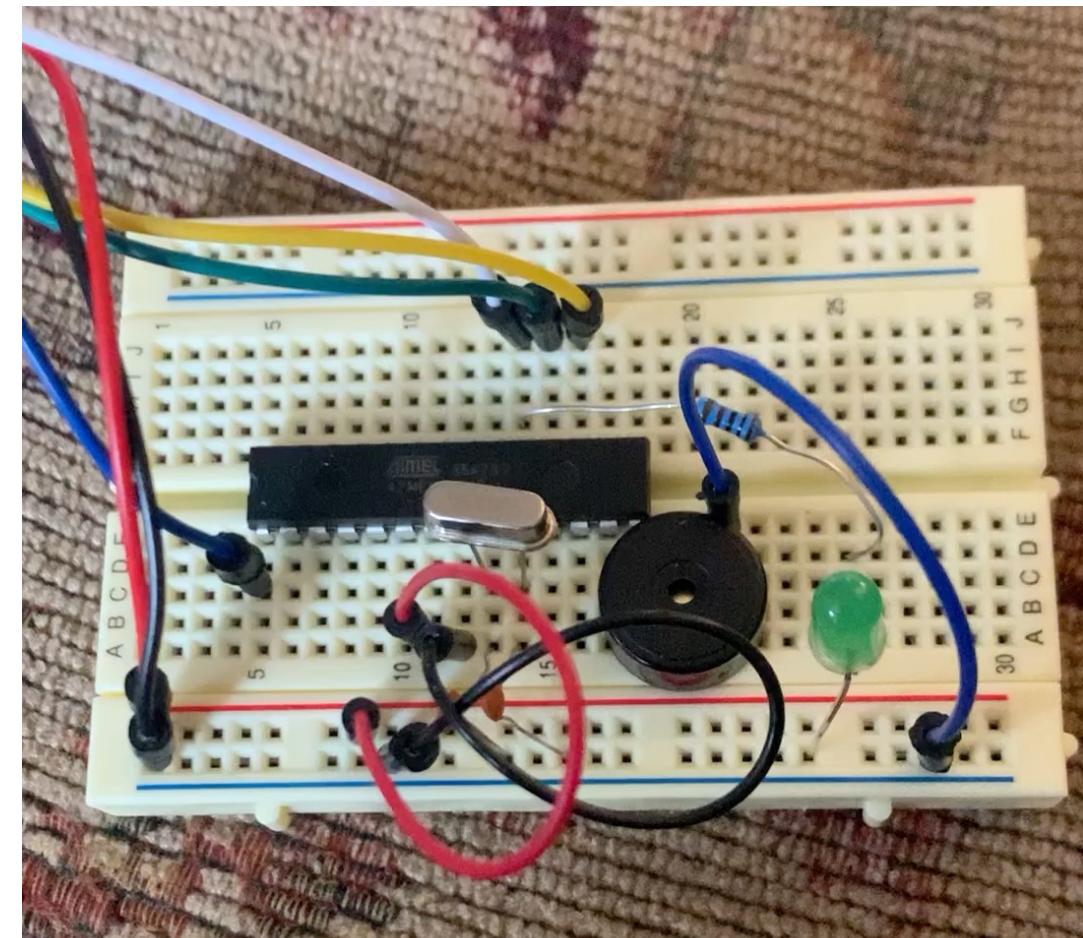


# Write a program to create a 1 KHz square wave using Timer 0 in normal mode and the 1:64 pre-scaler to create the delay

C one\_kHz.c > ...

```

1  /* one_khz.c This code creates a 1 KHz square wave for a piezo speaker on PB0
2  using timer 0. Timer arithmetic: 1 KHz square wave has a period of 1 ms.
3  Need to toggle the speaker every 0.5 ms or 500 us. 16 MHz clock has T=1/16 uSEC.
4  Need to divide the clock by 500*16=8000. Since the time can only count to 256,
5  we need to pre-divide the clock. Pre-divide options are 8, 64, 256, and 1024.
6  This code uses 64, then counts to 8000/64=125. Clock is pre-loaded with 256-125=131.
7  D. McLaughlin ECE-231 demo Spring 2022. Version 1.*/
8
9 #include <avr/io.h>
10
11 int main(void){
12     // Set up the GPIO bit
13     DDRB = 1<<DDB0;
14
15     while(1){
16         // Pre-load counter
17         TCNT0 = 131;
18         // Set operating mode to normal, clk/64
19         TCCR0A = 0;
20         TCCR0B = 1<<CS01|1<<CS00;
21         // Wait for timer to roll over
22         while ((TIFR0 & (1<<TOV0))==0);
23         // Stop the clock
24         TCCR0B = 0;
25         // Clear the roll over flag
26         TIFR0 |= 1<<TOV0;
27         // Toggle the Speaker
28         PORTB ^= 1<<PORTB0;
29     }
30 }
```



write a program to sequence through 4 tones at 62.5 Hz, 250 Hz, 1 KHz and 4 KHz, each played for 3 seconds

Use the following prescalars:

62.5 Hz, 1:1024

250 Hz, 1:256

1 KHz, 1:64

4 KHz, 1:8

We need to find the timer preload values for each tone.

$$F_{osc} = 16 \text{ MHz} \rightarrow T_{\text{machine cycle}} = 1/16 \text{ MHz} = 0.0625 \text{ us}$$

(a) for  $f=62.5 \text{ Hz}$ ,  $T=1/62.5 = 16 \text{ ms}$ ;  $T_{\text{delay}} = T/2 = 8 \text{ ms} = 8000 \text{ us}$

Prescaler = 1:1024  $\rightarrow T_{\text{clock}} = 1024 * 0.0625 \text{ us} = 64 \text{ us}$

$8000 \text{ us}/64 \text{ us} = 125$ ; preload:  $256 - 125 = 131$ ; prescaler: 5

(b) for  $f=250 \text{ Hz}$ ,  $T=1/250 = 4 \text{ ms}$ ;  $T_{\text{delay}} = T/2 = 2 \text{ ms} = 2000 \text{ us}$

Prescaler = 1:256  $\rightarrow T_{\text{clock}} = 256 * 0.0625 \text{ us} = 16 \text{ us}$

$2000 \text{ us}/16 \text{ us} = 125$ ; preload:  $256 - 125 = 131$ ; prescaler: 4

(c) for  $f=1 \text{ KHz} = 1000 \text{ Hz}$ ,  $T=1/1000 = 1 \text{ ms}$ ;  $T_{\text{delay}} = T/2 = 0.5 \text{ ms} = 500 \text{ us}$

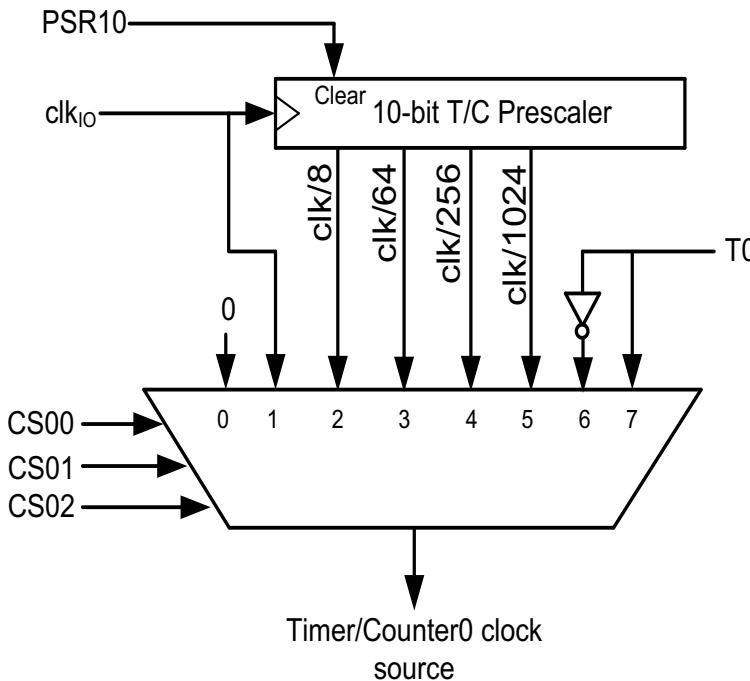
Prescaler = 1:64  $\rightarrow T_{\text{clock}} = 64 * 0.0625 \text{ us} = 4 \text{ us}$

$500 \text{ us}/4 \text{ us} = 125$ ; preload:  $256 - 125 = 131$ ; prescaler: 3

(d) for  $f=4 \text{ kHz} = 4000 \text{ Hz}$ ,  $T=1/4000 = 0.25 \text{ ms}$ ;  $T_{\text{delay}} = T/2 = 0.125 \text{ ms} = 125 \text{ us}$

Prescaler = 1:8  $\rightarrow T_{\text{clock}} = 8 * 0.0625 \text{ us} = 0.5 \text{ us}$

$125 \text{ us}/0.5 \text{ us} = 250$ ; preload:  $256 - 250 = 6$ ; prescaler: 2

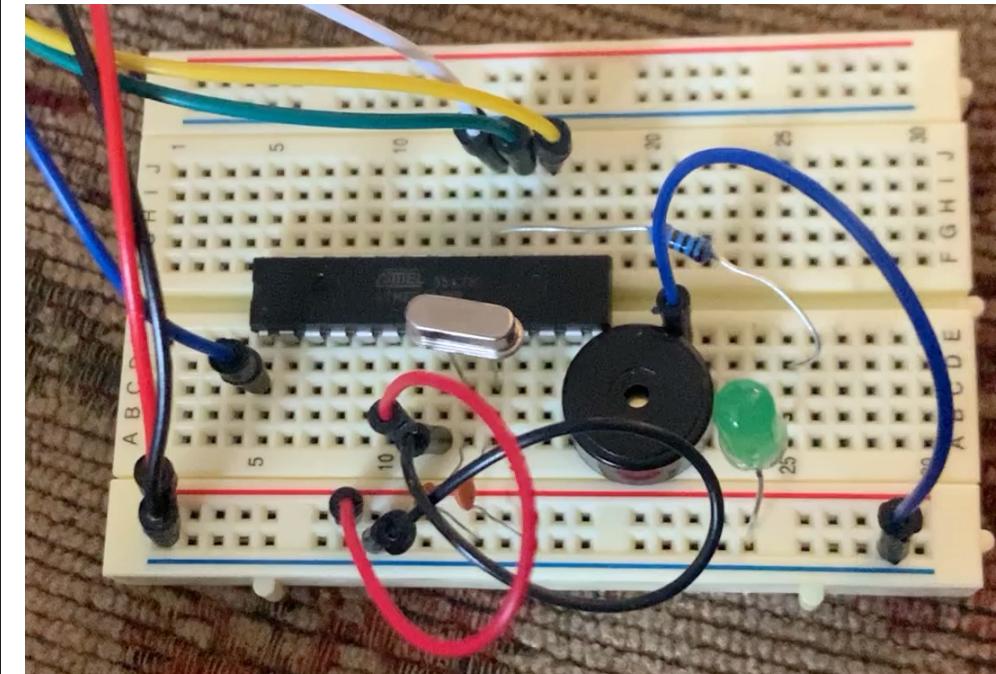


## C four\_tones.c &gt; ...

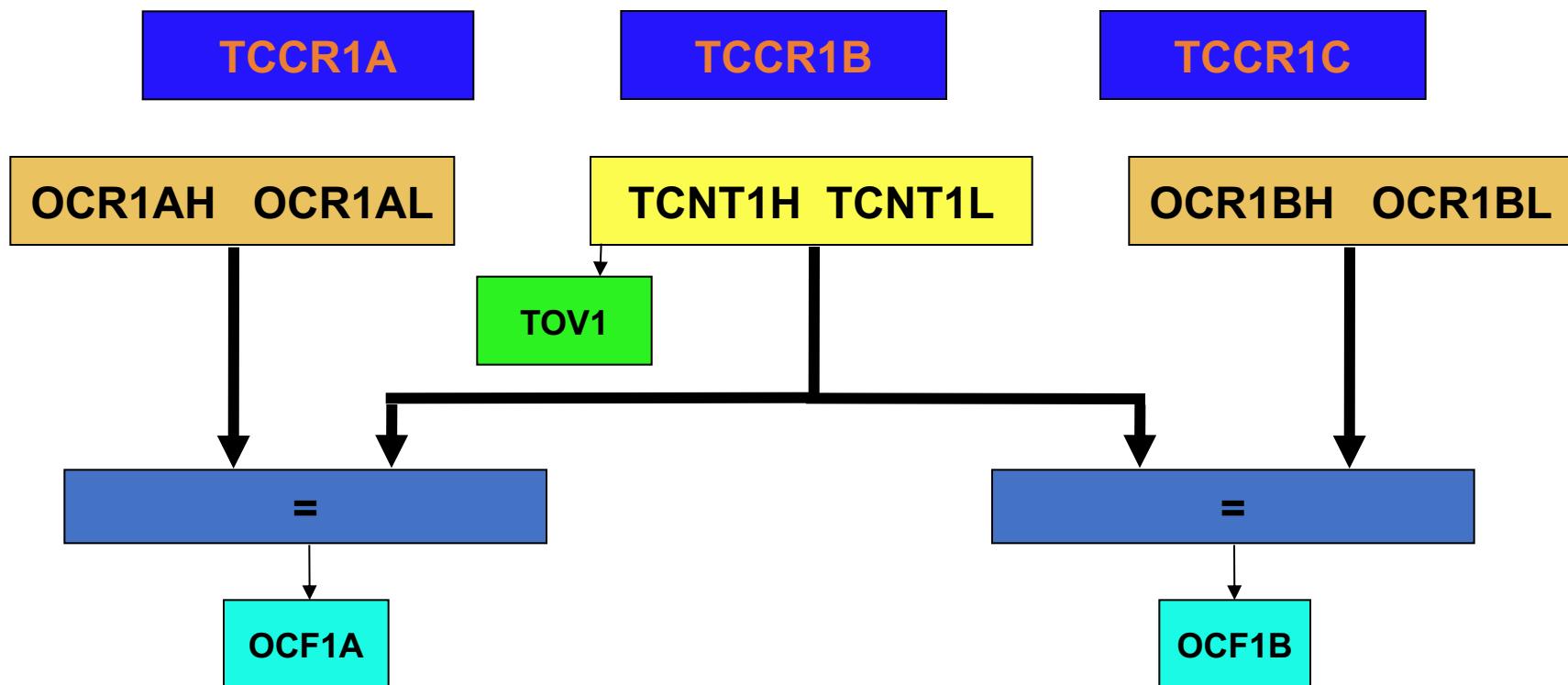
```

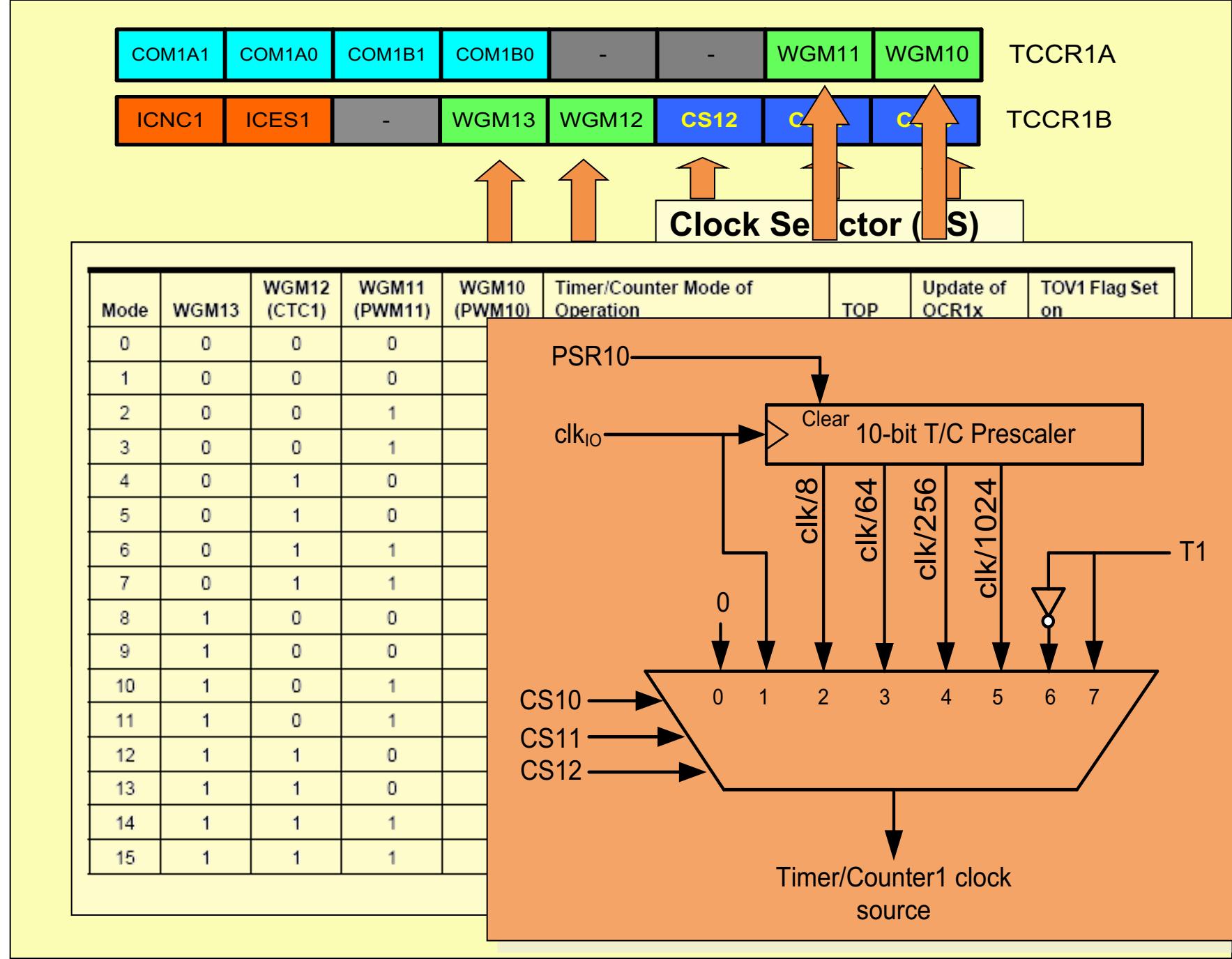
1  /* four_tones.c Plays a sequence of tones: 62.5 Hz, 250 Hz,
2  1 KHz, 4 KHz, repeatedly. Piezo Speaker on PB0.
3  D. McLaughlin ECE-231 Spring 2022 */
4
5  #include <avr/io.h>
6  int main(void)
7  {
8      unsigned char preloads[4] = {131, 131, 131, 6};
9      unsigned char prescaler[4] = {5, 4, 3, 2};
10     float frequency[4] = {62.5, 250, 1000, 4000};
11     unsigned int i, j;
12
13     DDRB = 1 << DDB0; //PB0 is output pin
14     while (1) {
15         for (i=0; i<4; i++)
16             for (j=0; j<3*frequency[i]; j++) {
17                 TCNT0 = preloads[i]; //preload value for tone
18                 TCCR0A = 0x00; //normal mode
19                 TCCR0B = prescaler[i]; //prescaler value for tone
20                 while ((TIFR0 & (1 << TOV0)) == 0); //wait till done
21                 TCCR0B = 0; //stop the clock
22                 TIFR0 = (1 << TOV0); //clear the overflow flag
23                 PORTB ^= 1 << PORTB0; //toggle the output pin
24
25     }
26 }
27
28

```



# Timer 1



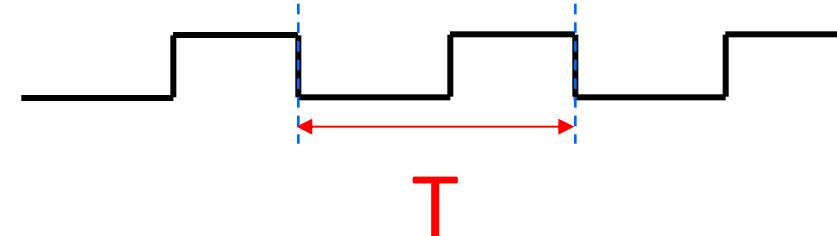


# Waveform Characteristics

25

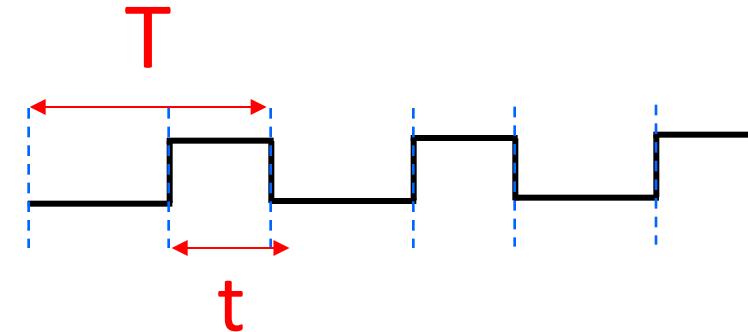
- Period  $T$

- Frequency  $f = \frac{1}{T}$

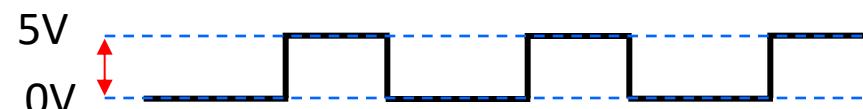


- Duty cycle

$$\text{duty cycle} = \frac{t}{T} \times 100$$



- Amplitude



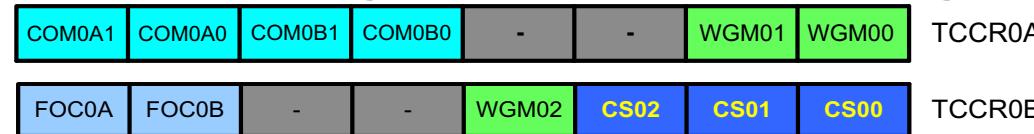
# 3 Timers/Counters on the ATmega328P

timer0 – 8 bits (counts 0-255)

timer1 – 16 bits (counts 0-65,536)

timer2 - 8 bits (counts 0-255)

Configure timers using TCCRxA & TCCRxB registers



WGM02, WGM01, WGM00 (shorthand: WGM02:00) – configure the timer mode. There are multiple modes.

CS02:00 – select the clock divider (/1, /8, /64, /1024)

normal mode, timer0:

configure the timer

loop:

pre-load TCNT0

wait for overflow flag TOV1

clear overflow flag

## Demo: square wave of increasing frequency, applied to LED, piezo speaker, and oscilloscope

Timer1; 16 MHz system clock; pre-scaler = 64

Timer clock frequency:  $16\text{MHz}/64 = 250\text{ KHz}$

Timer clock period:  $1/250\text{KHz} = 4\text{ usec}$

*\*\*\* each count of the clock is 4 usec long \*\*\**

variable “pitch” controls the frequency of the output.

Timer is pre-loaded with  $\text{TCNT1} = 65636 - 65636/\text{pitch}$ , so total timer delay is:

$4\text{ uSec} * (65,536 - \text{TCNT1})$

$4\text{ usec} * (65,536 - (65,536 - 65,536/\text{pitch}))$

**4 usec \* 65,536/pitch.**

Square wave period is 2x timer delay

=  $8\text{ usec} * 65,536/\text{pitch}$

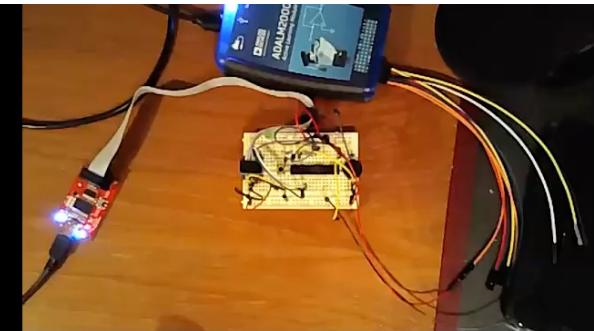
Square wave frequency is 1/ square wave period

= **pitch/1.9 Hz**

Every time button is pressed, pitch ~ doubles, so result is this set of frequencies: 1.9, 3.8, 7.6, 15.2, etc... \*/

**C** rampfreq.c >  main(void)

```
1  /* rampfreq.c  Ramps (increases) the frequency on PD6
2   every time button on PB0 is pressed.
3   D. McLaughlin 4/6/22 for ECE-231 Demo Spring 2022 */
4
5  #include <avr/io.h>
6  #include <util/delay.h>
7
8  int main(void){
9      unsigned int pitch = 1;
10     DDRD = 1<<DDD6;          // LED, speaker, oscope on PD6
11     DDRB = 0;                // pushbutton switch on PB0
12     PORTB = 1<<PB0;         // Set pullup on PB0
13
14     // Configure the timer
15     TCCR1A = 0;              // Normal mode (count up)
16     TCCR1B = 3;              // Divide clock by 64
17
18     while(1{
19         // Change the frequency variable when switch is pressed
20         if ( (PINB & (1<<PB0)) == 0){
21             pitch*=2;
22             // This delay prevents code from repeatedly changing the pitch
23             // When finger is on the button
24             _delay_ms(200);
25         }
26         // Counter delay
27         TCNT1=65536-65536/pitch;        // Load the timer
28         while ((TIFR1 & (1<<TOV1))==0); // Wait for rollover flag
29         TIFR1 |= 1<<TOV1;               // Clear the rollover flat
30
31         PORTD ^= 1<<PORTD6;           // Toggle the speaker & LED
32     }
33 }
```



```
C rampfreqfunction.c > ...
1  /* rampfreqfunction.c Increases the frequency of square wave on PD6
2  every time a button on PB0 is pressed.
3  D. McLaughlin 4/6/22 for ECE-231 Demo Spring 2022 */
4
5  #include <avr/io.h>
6  #include <util/delay.h>
7  void timer1_init(void);
8  void delay(unsigned int);
9
10 int main(void){
11     unsigned int pitch = 1; // This variable controls the pitch (freq)
12     DDRD = 1<<DDD6;        // LED, speaker, oscope on PD6
13     DDRB = 0;              // Push-button switch on PB0
14     PORTB = 1<<PB0;        // Set pullup on PB0
15     timer1_init();         // Initialize timer1
16
17     while(1){
18         if ( (PINB & (1<<PB0)) == 0){ // Change each time button is pressed
19             pitch*=2;
20             _delay_ms(200); // Delay avoids multiple changes per button press
21         }
22         delay(pitch);
23         PORTD ^= 1<<PORTD6;           // Toggle the speaker & LED
24     }
25 }
26
27 // Initialize timer1: normal mode (count up), divide clock by 64
28 void timer1_init(){
29     TCCR1A = 0;            // Timer 1 Normal mode (count up)
30     TCCR1B = 3;            // Divide clock by 64
31 }
32 // This function causes a delay of 0.26/pitch seconds.
33 void delay(unsigned int pitch){
34     TCNT1=65536-65536/pitch;
35     while ((TIFR1 & (1<<TOV1))==0); // Wait for rollover flag
36     TIFR1 |= 1<<TOV1;           // Clear the rollover flat
37 }
38 /** end of file***/
```

Let's re-examine  
the code from the  
previous demo.

This code has been  
re-arranged such  
that timer1 details  
are included in

**timer1\_init()**

and

**delay()**

## Review what we've just done:

### Configure Timer1

- "normal mode"
- divide system clock by 64 (pre-scale)

```
// Configure the timer
TCCR1A = 0;           // Normal mode (count up)
TCCR1B = 3;           // Divide clock by 64
```

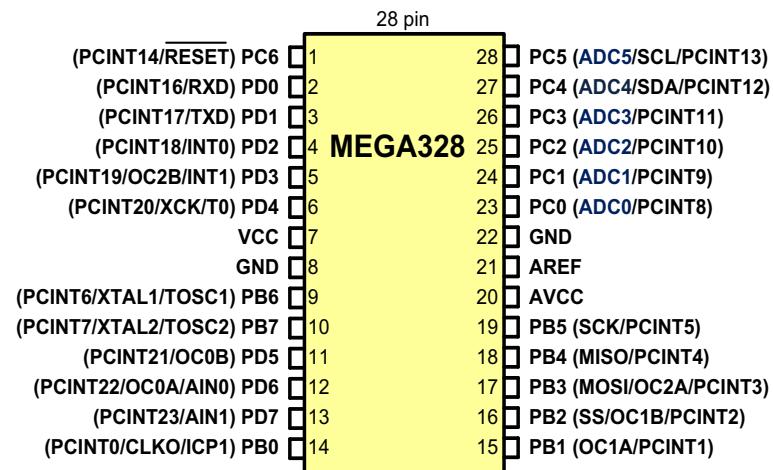
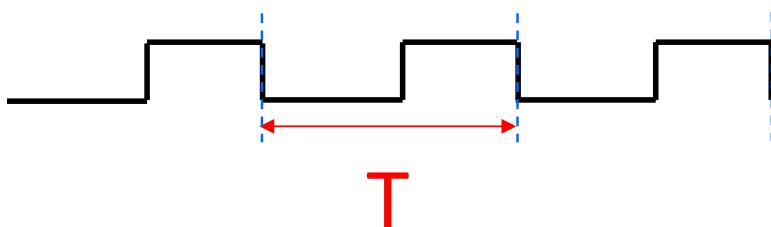
### Timer delay loop

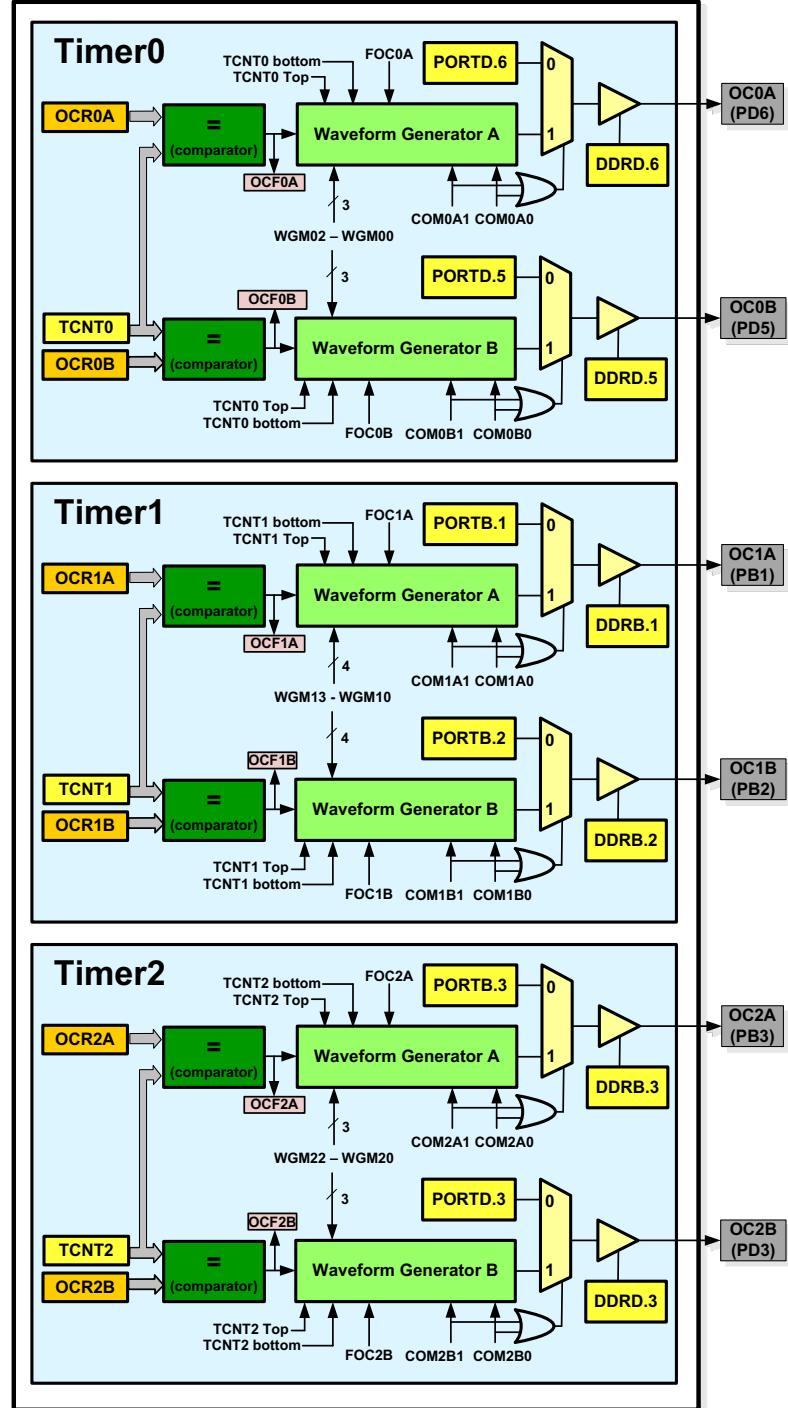
- pre-load value into timer
- timer counts from pre-load value to TOP
- 16 bit timer, so TOP =  $2^{16}-1 = 65,536$
- wait for overflow to occur
- clear overflow flag (a quirky feature of AVR microcontrollers)
- toggle output pin (we happened to use PD6)

```
// Counter delay
TCNT1=65536-65536/pitch;
while ((TIFR1 & (1<<TOV1))==0);
TIFR1 |= 1<<TOV1;
PORTD ^= 1<<PORTD6;
```

### LED, speaker, oscilloscope on PD6

$$f = \frac{1}{T}$$





Each timer/counter has 2 waveform generators

6 waveform outputs share 6 GPIO pins:

Timer0: OC0A/PD6 & OC0B/PD5

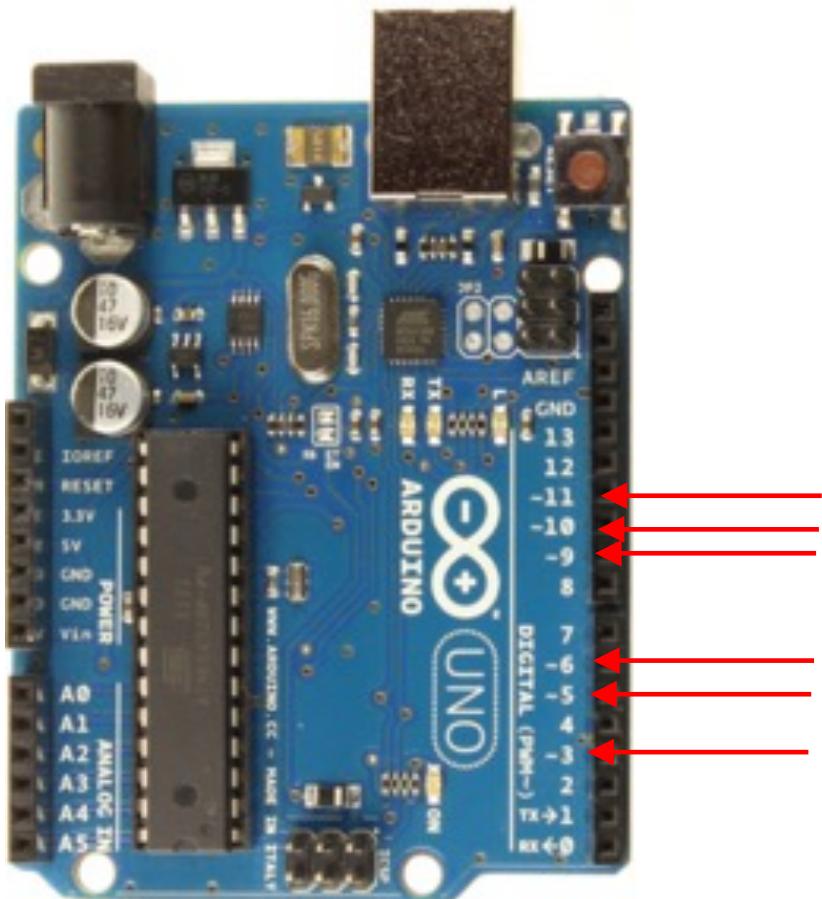
Timer1: OC1A/PB1 & OC1B/PB2

Timer2: OC2A/PB3 & OC2B/PD3

28 pin	
(PCINT14/RESET) PC6	28 PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD) PD0	27 PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD) PD1	26 PC3 (ADC3/PCINT11)
(PCINT18/INT0) PD2	25 PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1) PD3	24 PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0) PD4	23 PC0 (ADC0/PCINT8)
VCC	22 GND
GND	21 AREF
(PCINT6/XTAL1/TOSC1) PB6	20 AVCC
(PCINT7/XTAL2/TOSC2) PB7	19 PB5 (SCK/PCINT5)
(PCINT21/OC0B) PD5	18 PB4 (MISO/PCINT4)
(PCINT22/OC0A/AIN0) PD6	17 PB3 (MOSI/OC2A/PCINT3)
(PCINT23/AIN1) PD7	16 PB2 (SS/OC1B/PCINT2)
(PCINT0/CLKO/ICP1) PB0	15 PB1 (OC1A/PCINT1)

**MEGA328**

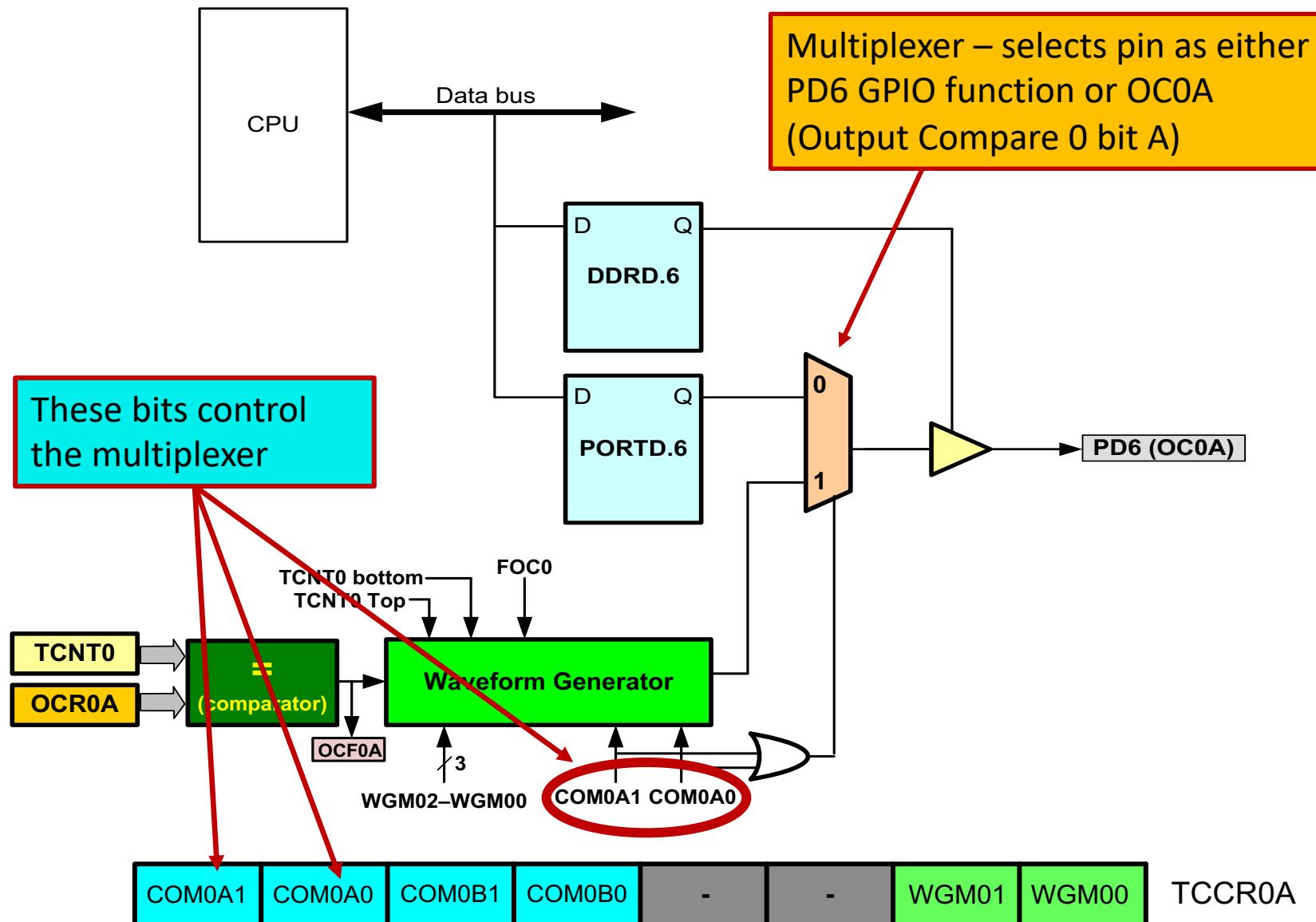
FYI: these 6 pins correspond to Arduino Uno pins 3, 5, 6, 9, 10, 11 (~PWM pins)



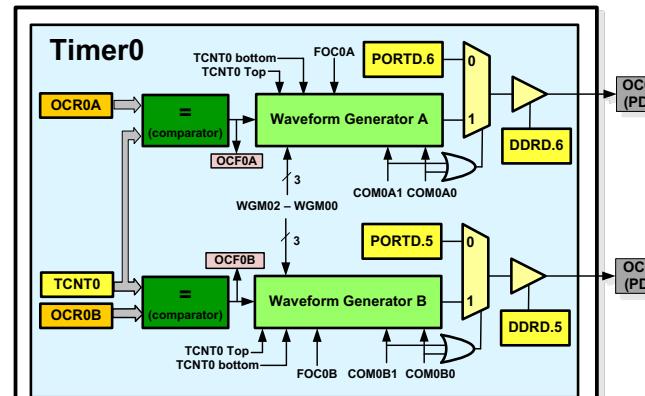
28 pin	
(PCINT14/RESET) PC6	1
(PCINT16/RXD) PD0	2
(PCINT17/TXD) PD1	3
(PCINT18/INT0) PD2	4
(PCINT19/OC2B/INT1) PD3	5
(PCINT20/XCK/T0) PD4	6
VCC	7
GND	8
(PCINT6/XTAL1/TOSC1) PB6	9
(PCINT7/XTAL2/TOSC2) PB7	10
(PCINT21/OC0B) PD5	11
(PCINT22/OC0A/AIN0) PD6	12
(PCINT23/AIN1) PD7	13
(PCINT0/CLK0/ICP1) PB0	14
28	PC5 (ADC5/SCL/PCINT13)
27	PC4 (ADC4/SDA/PCINT12)
26	PC3 (ADC3/PCINT11)
25	PC2 (ADC2/PCINT10)
24	PC1 (ADC1/PCINT9)
23	PC0 (ADC0/PCINT8)
22	GND
21	AREF
20	AVCC
19	PB5 (SCK/PCINT5)
18	PB4 (MISO/PCINT4)
17	PB3 (MOSI/OC2A/PCINT3)
16	PB2 (SS/OC1B/PCINT2)
15	PB1 (OC1A/PCINT1)

# Waveform Generator 0, output A

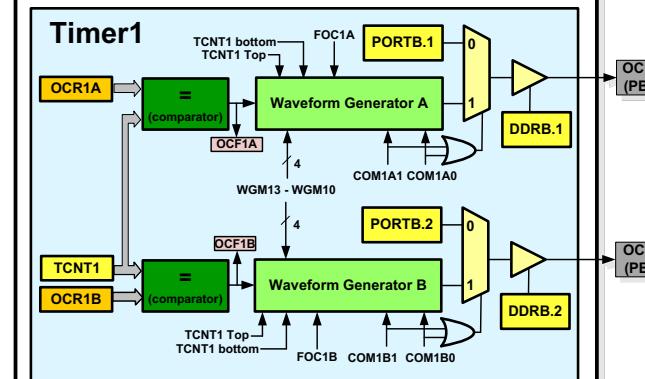
34



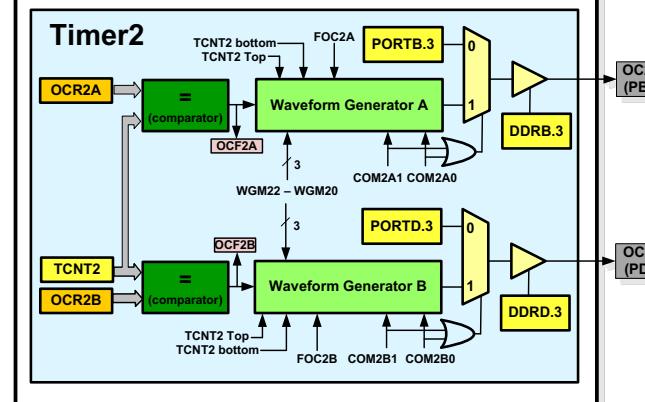
# Circuitry from previous slide is repeated for all 6 waveform generators...



Timer0 Output Comparator 0 Waveform A (OC0A)



Timer1 Output Comparator 1 Waveform A (OC1A)

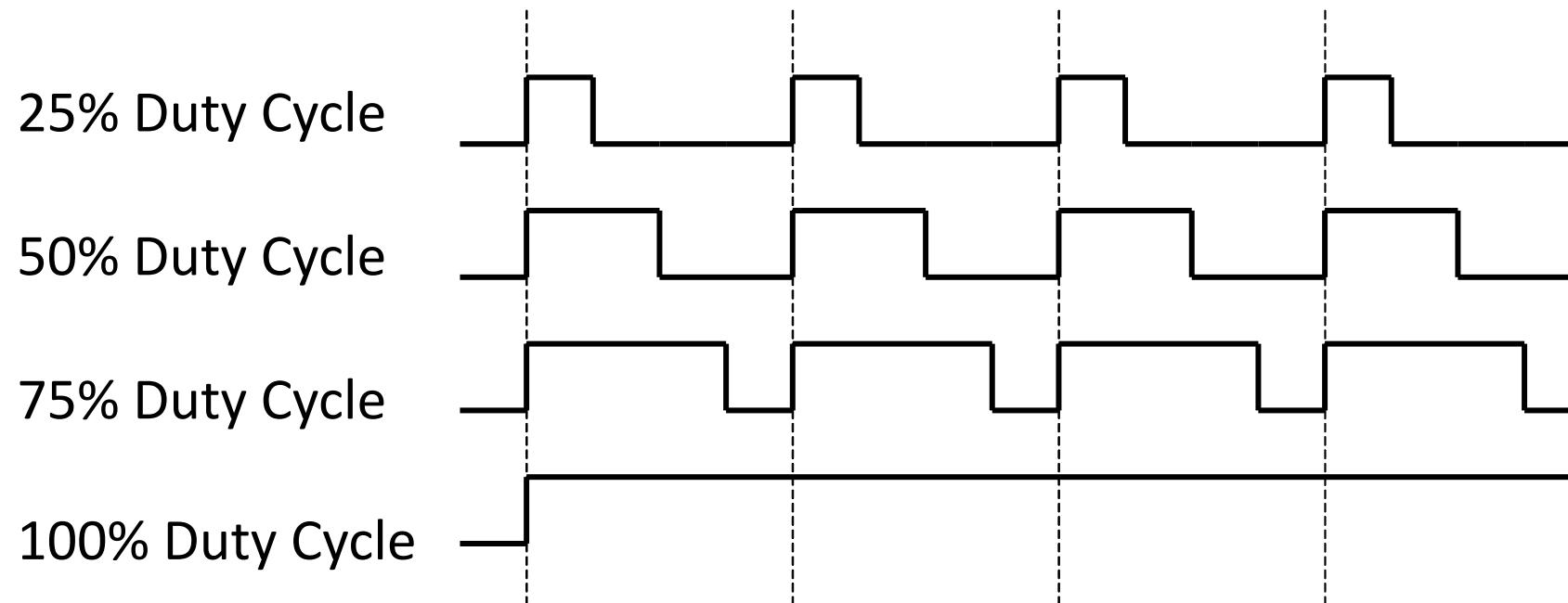


Timer2 Output Comparator 2 Waveform A (OC2A)

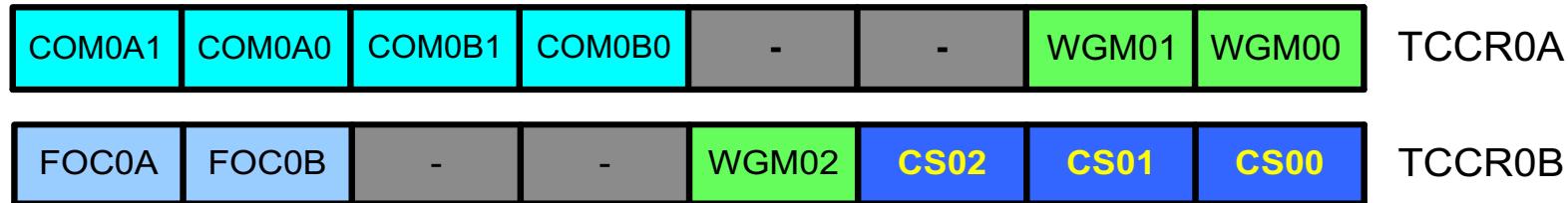
Timer2 Output Comparator 2 Waveform B (OC2B)

# We will use the waveform generators for Pulse Width Modulation (variable Duty Cycle)

Can be used to dim LEDs, vary motor speed, & other applications



# Fast PWM Mode (PWM = pulse width modulation)



COM0A1:COM0A0 = 10 (clear OC0A on compare match; set on top)

WGM02:WGM00 = 011 (fast PWM mode)

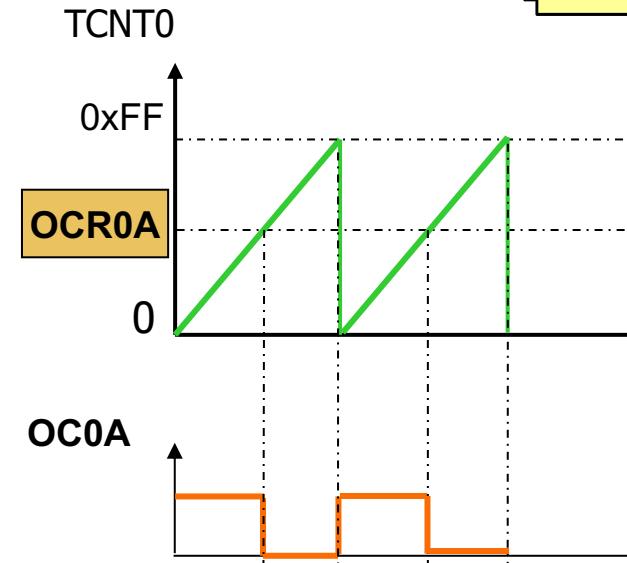
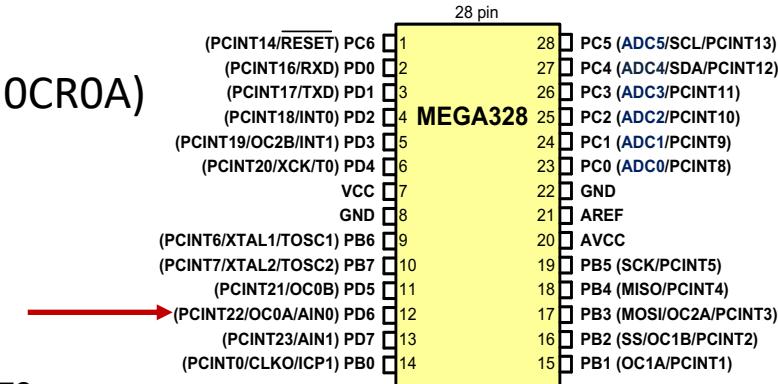
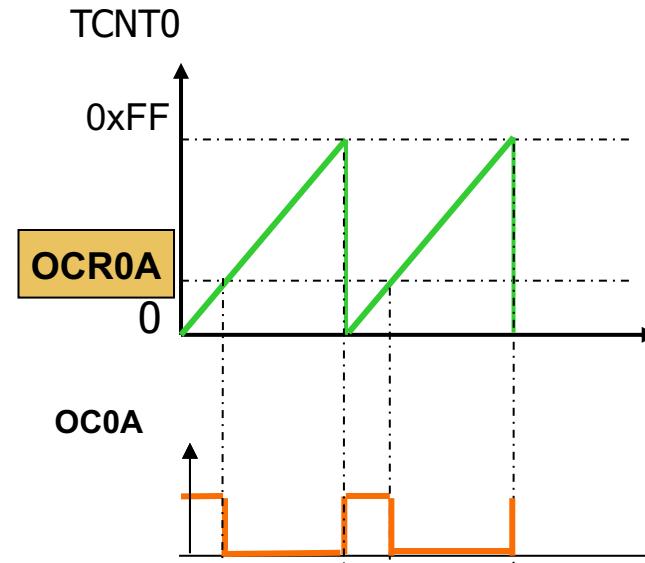
CS02:CS00 = 011 ( 64 pre-scaler)

$$\text{TCCR0A} = 10000011 = 0x83$$

$$\text{TCCR0B} = 00000011 = 0x03$$

# Fast PWM Mode (Timer0, output OC0A)

- configure timer0 for Fast PWM Mode
- load value into OCR0A (Output Compare Register OCR0A)
- timer continuously counts from 0x00 to 0xFF
- **Output pin OC0A/PD6:**
  - set when TCNT0 = 0xFF (top)
  - cleared when TCNT0=OCR0A

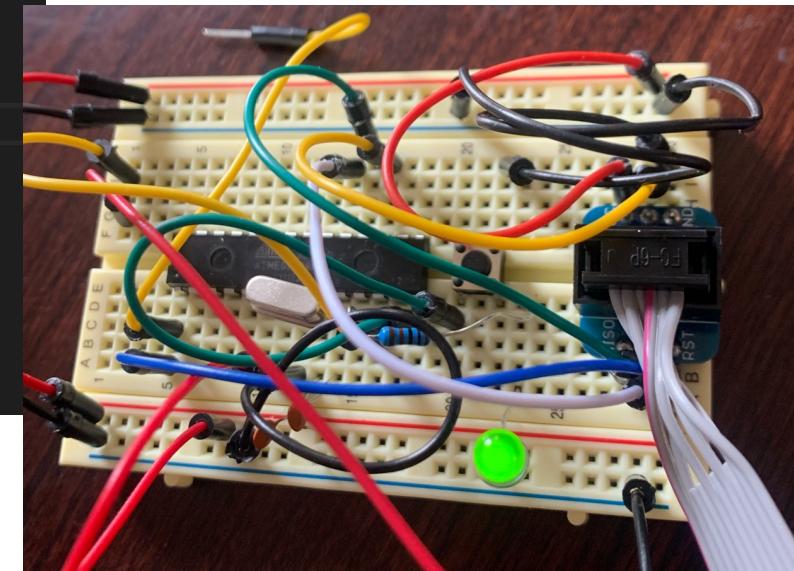


Increase & decrease duty cycle by increasing & decreasing OCR0A  
OCR0A=0 → 0% duty cycle; OCR0A = 255 → 0% duty cycle

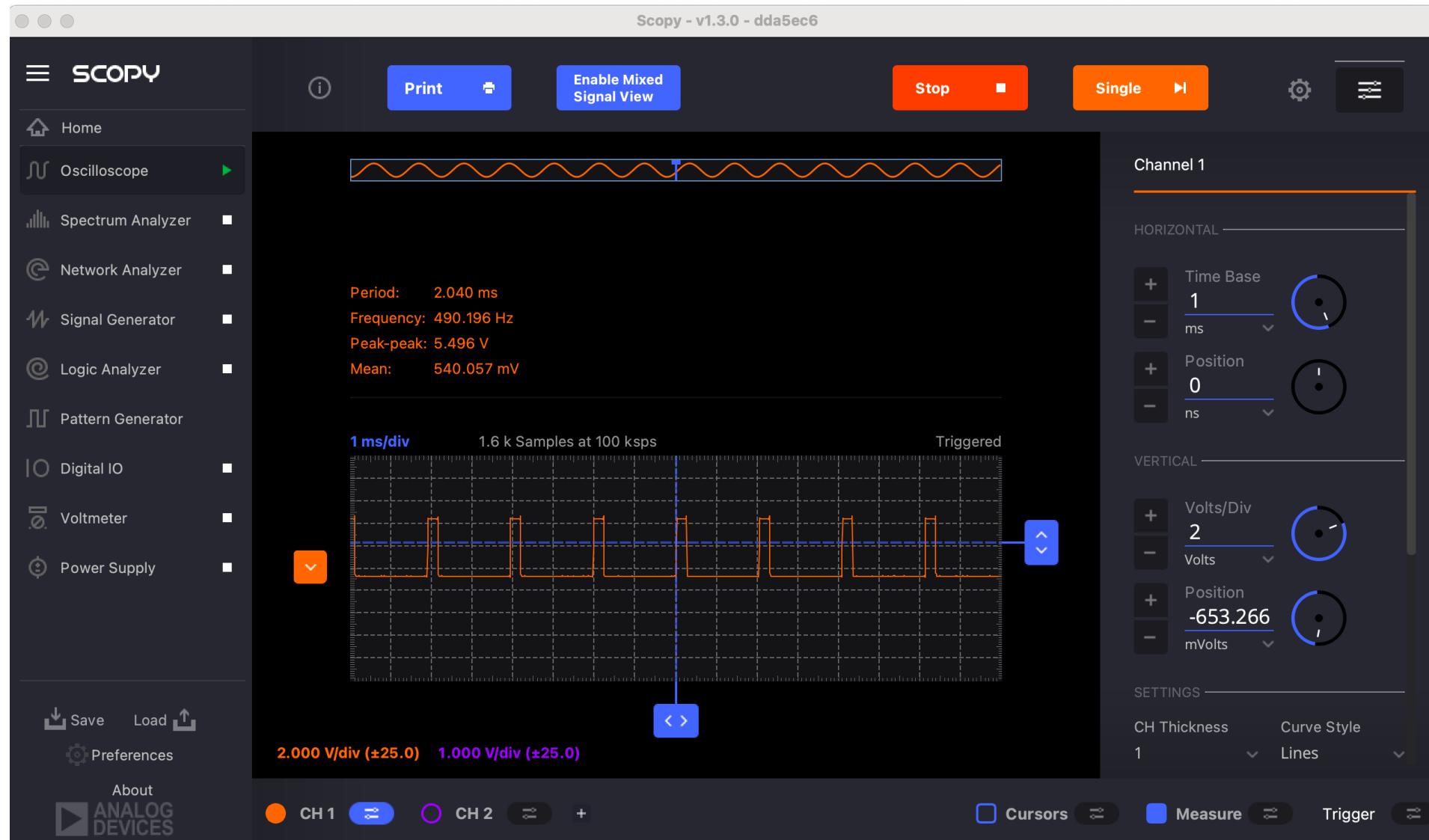
stpw.c >  main(void)

```
/* fastpwm.c Creates 12.5% duty cycle waveform  
on OC0A (PB6). Uses timer0 waveform generator A  
with /64 pre-scaler.  
Waveform frequency: 16MHz/64/256 = 976 Hz  
Duty cycle: 12.5% corresponds to OCR0A = 12.5% of 256 = 32  
D. McLaughlin 4/6/22 for ECE-231 Demo Spring 2022 */
```

```
#include <avr/io.h>  
#include <util/delay.h>  
  
int main(void){  
    DDRD = 1<<DDD6;      // Output must be PD6 for this code  
  
    // Initialize timer0 for FASTPWM, /64 prescaler  
    TCCR0A = 0x81;  
    TCCR0B = 0x03;  
  
    // Load the value into OCR0A:  
    OCR0A = 32;        // 32 is 12.5% of 256 so 12.5% duty cycle  
  
    while(1);          // Just sit there and do nothing  
}
```



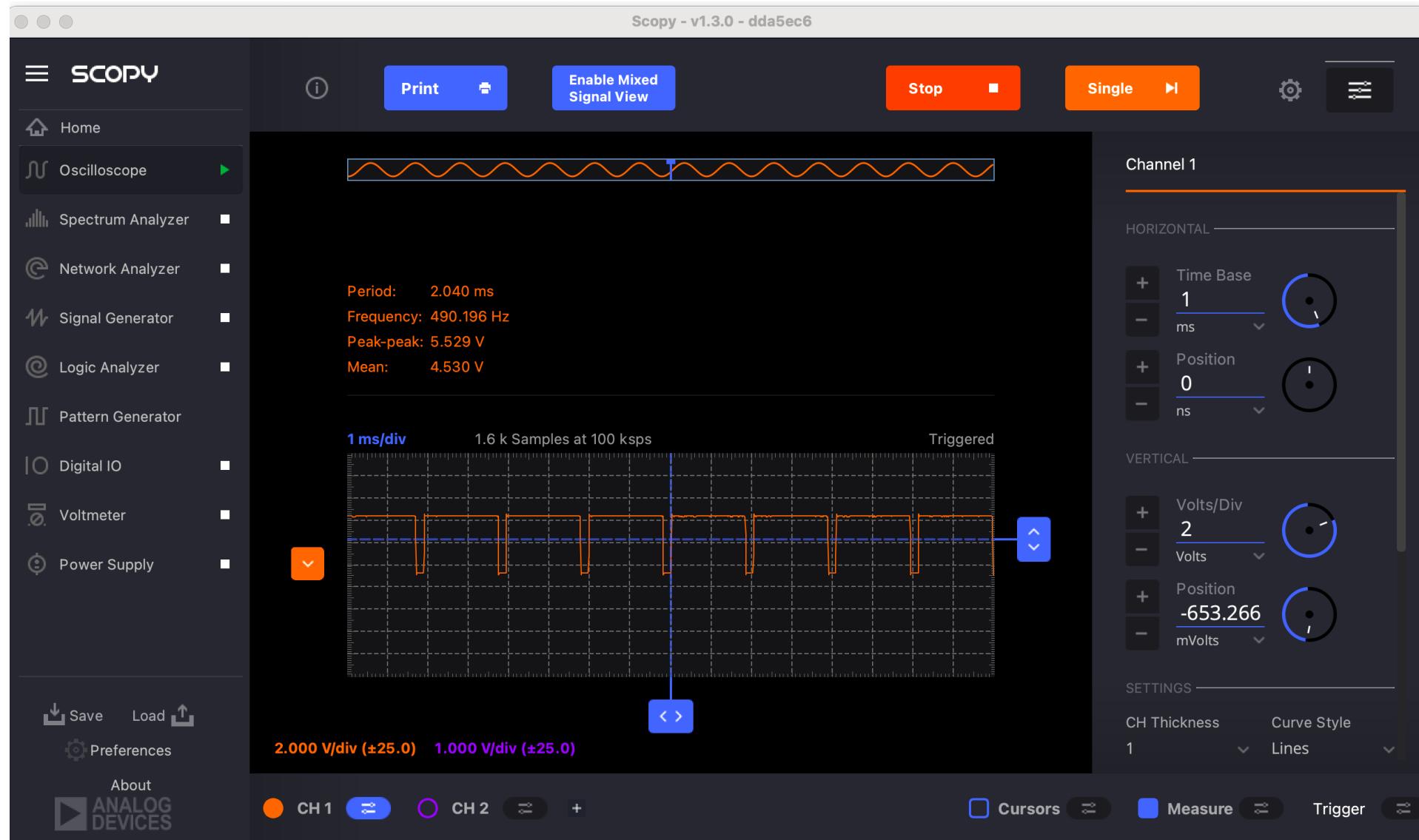
# 490 Hz pulse train, 12% duty cycle



**C** fastpwm.c >  main(void)

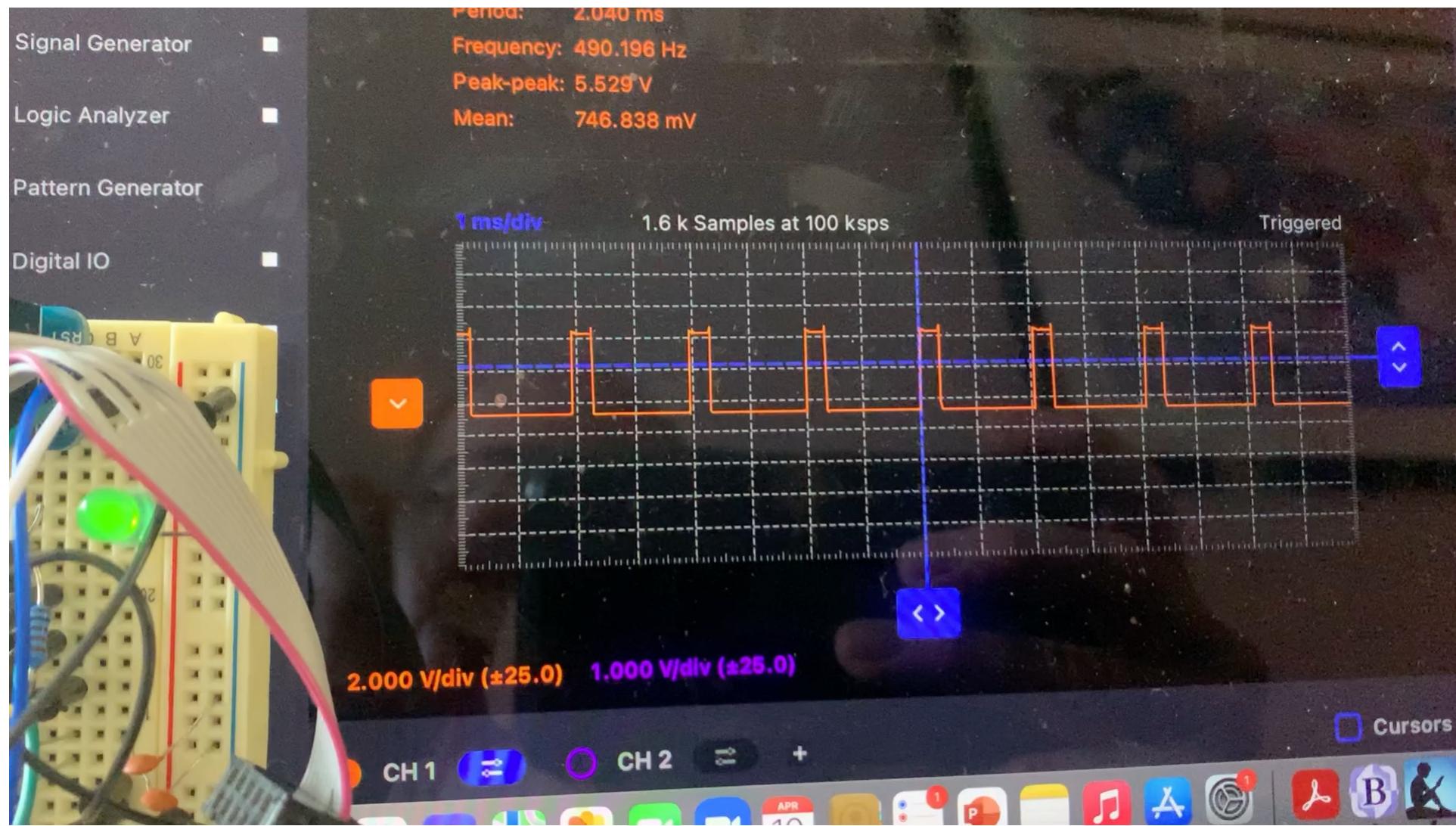
```
1  /* fastpwm.c Creates 12.5% duty cycle waveform
2   on OC0A (PB6). Uses timer0 waveform generator A
3   with /64 pre-scaler.
4   Waveform frequency: 16MHz/64/256/2 = 488 Hz
5   Duty cycle: 90% corresponds to OCR0A = 90% of 256 = 230
6   D. McLaughlin 4/6/22 for ECE-231 Demo Spring 2022 */
7
8  #include <avr/io.h>
9  #include <util/delay.h>
10
11 int main(void){
12     DDRD = 1<<DDD6;      // Output must be PD6 for this code
13
14     // Initialize timer0 for FASTPWM, /64 prescaler
15     TCCR0A = 0x81;
16     TCCR0B = 0x03;
17
18     // Load the value into OCR0A:
19     OCR0A = 230;        // 230 is 90% of 256 so 90% duty cycle
20
21     while(1);           // Just sit there and do nothing
22 }
23
24
```

# 490 Hz pulse train, 90% duty cycle



**C fastpwmramp.c > ⊞ WAIT**

```
1  /* fastpwmramp.c Ramps up duty cycle on OC0A (PB6)
2   between 0 and 100%. Uses timer0 waveform generator
3   with /64 pre-scaler
4   D. McLaughlin 4/6/22 for ECE-231 Demo Spring 2022 */
5
6  #include <avr/io.h>
7  #include <util/delay.h>
8
9  #define WAIT 250
10 int main(void){
11     unsigned char i=0;
12     DDRD = 1<<DDD6;      // Output must be PD6 for this code
13
14     // Initialize timer0 for FASTPWM, /64 prescaler
15     TCCR0A = 0x81;
16     TCCR0B = 0x03;
17
18
19     while(1){
20         i=i+10;
21         OCR0A=i;          // pulse turns off when TCNT1 = OCROA
22         _delay_ms(WAIT);
23     }
24 }
```

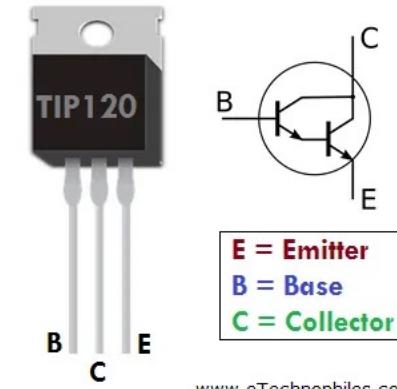
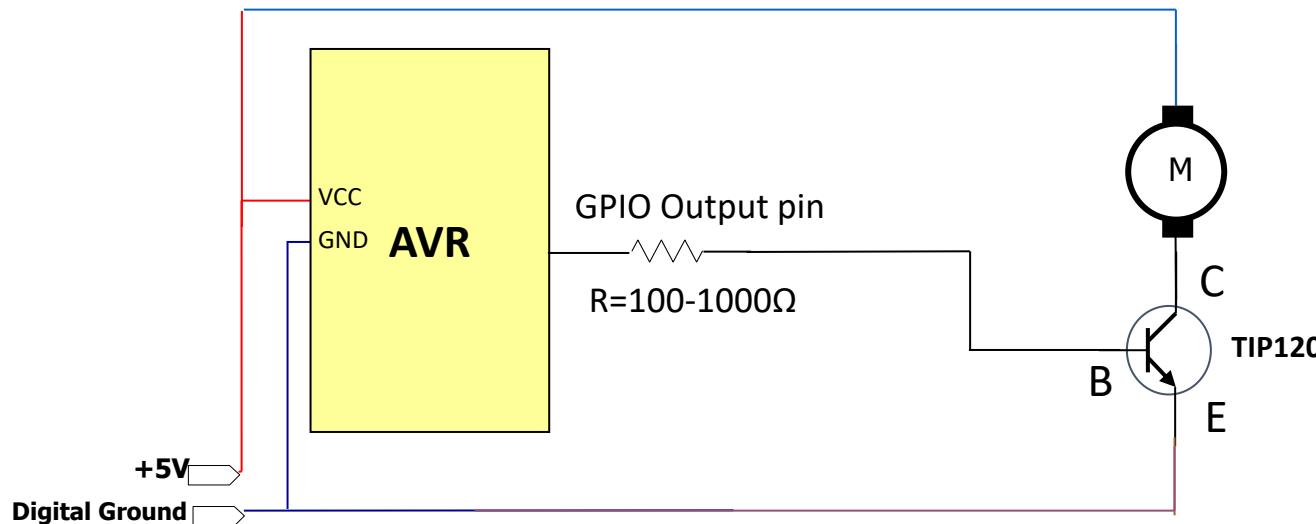


# Drive a motor using a GPIO pin

GPIO pins can only source up to 40 mA

Motors draw hundreds of mA

Darlington transistor serves as motor driver: when GPIO pin is high, the transistor is on, making a connection between C & E.  
Motor draws its current directly from the +5V power supply rather than the GPIO pin



- The meter is showing the DC voltage (average value) of the waveform. The motor speed is proportional to DC voltage.
- Separate bench-top power supply for the motor.
- This experiment reset my 328p, ADALM2000 scope, and USB port repeatedly due to electrical noise from the motor.



- Motor connected directly to my bench-top variable DC power supply.
- The meter is showing the power supply voltage
- Note the impact of the motor on the power supply voltage (spikes, fluctuations)
- Adding a capacitor across the power supply (VCC & GND) suppresses some of the noise.



Opto-isolator, fly-back diode, separate power supplies for motor & 328P  
(This is a useful interface circuit)

