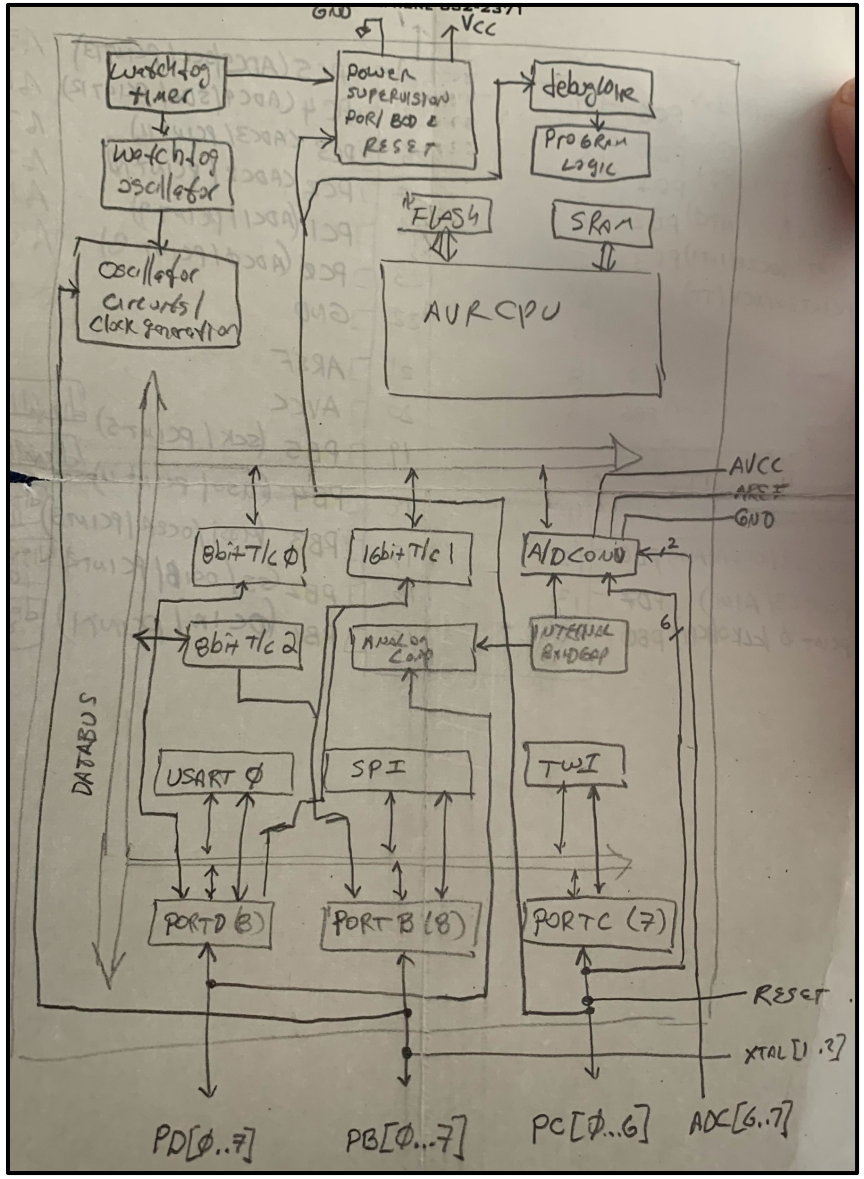
8.7 & 8.8 Timers/Counters

Prof. David McLaughlin

ECE Dept

UMass Amherst

Spring 2024



Timers/Counters on the ATmega328P

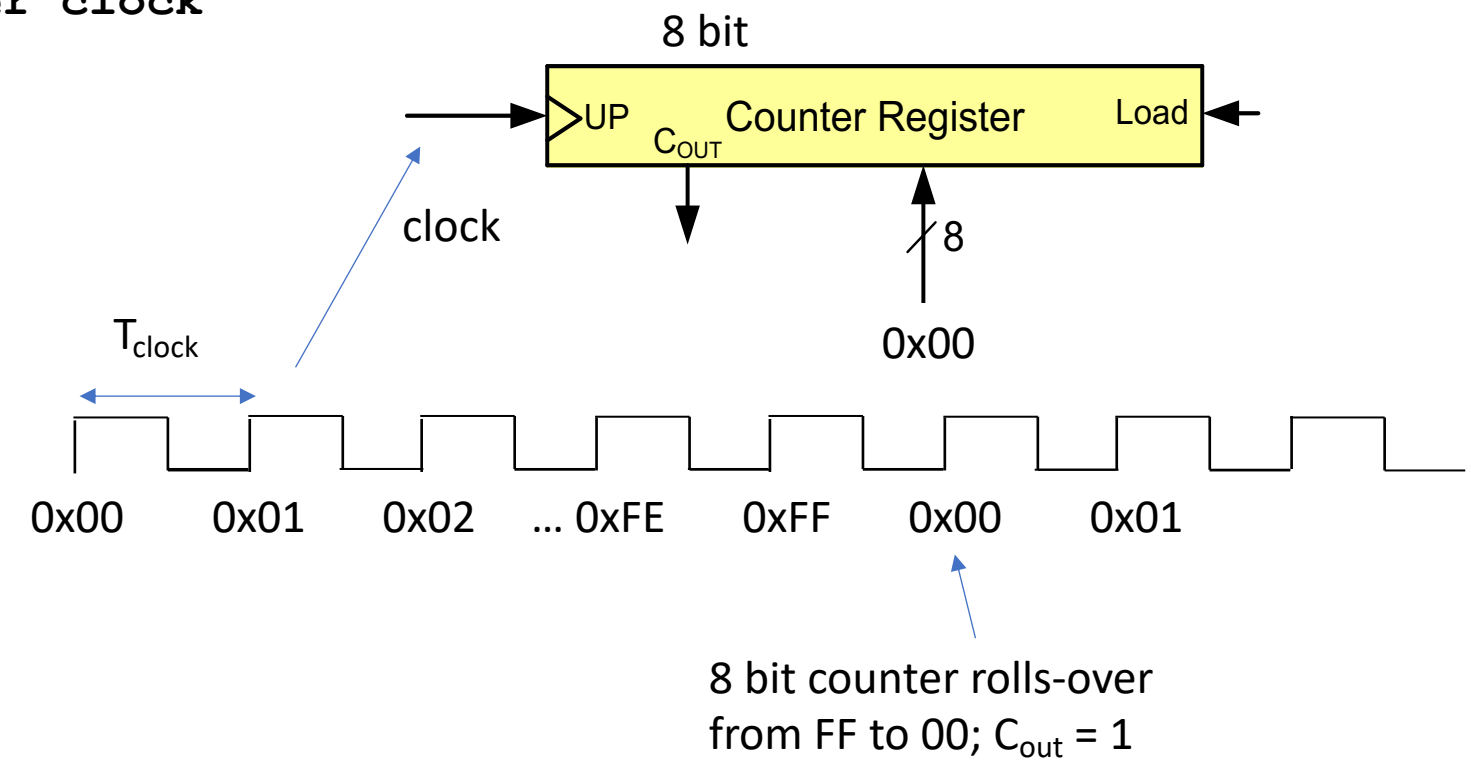
- *They function as timers*: counting specified numbers of cycles of known clock frequency to create delays & waveforms
- *They function as counters:* counting external triggers that cause transitions on external pins to capture & measure events
- 3 Timer/Counters on our ATmega328P MCU:
  - Timer/Counter0 -- 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:

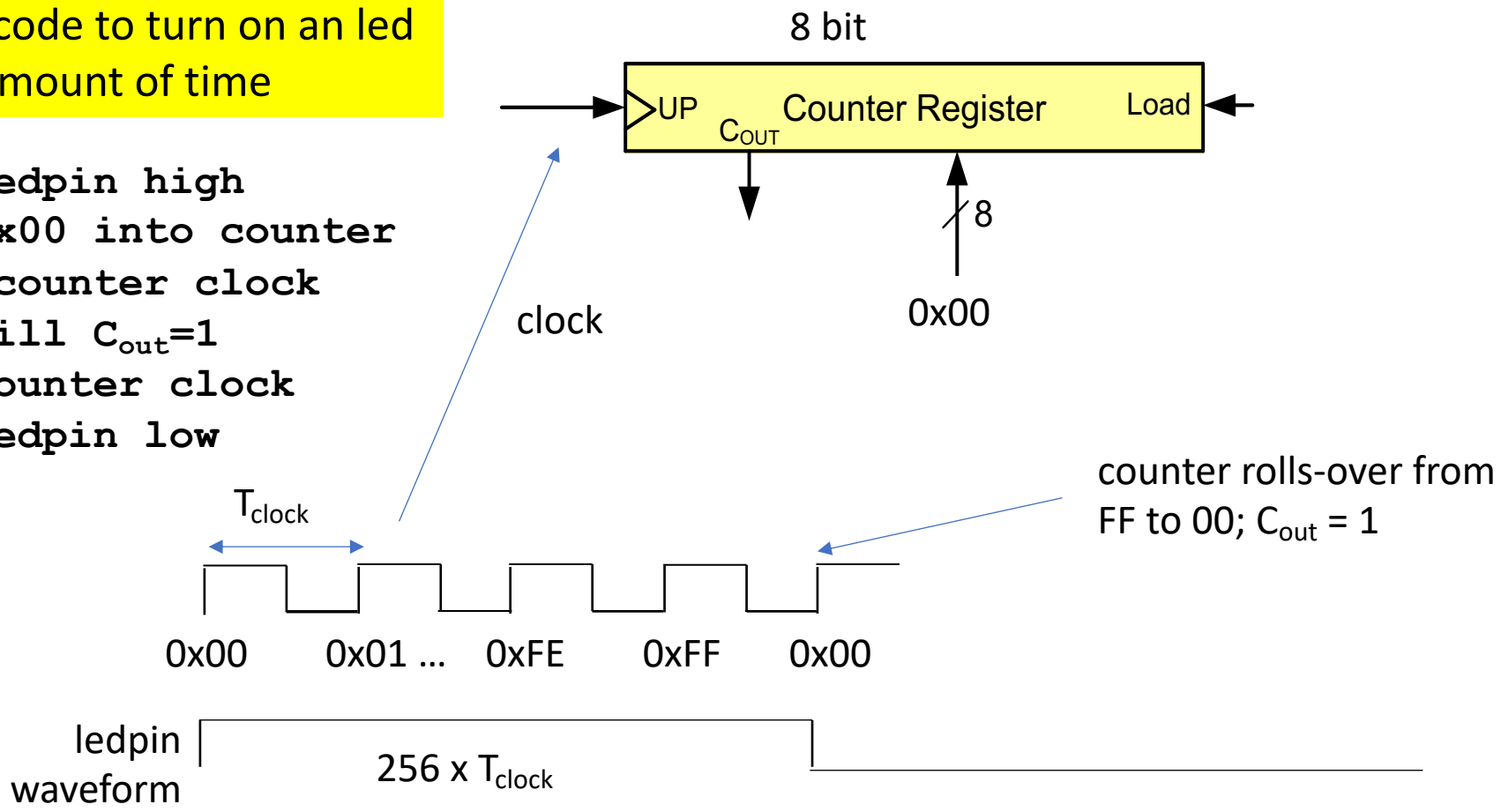
```
load 0x00 into counter
start counter clock
```



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
```

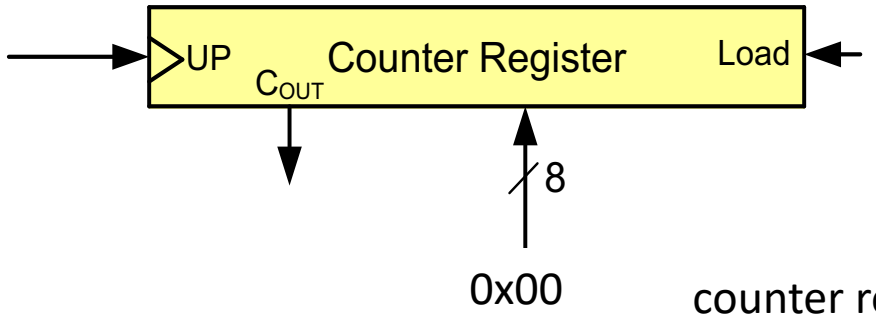


led remains on for 256 x T_{clock} seconds

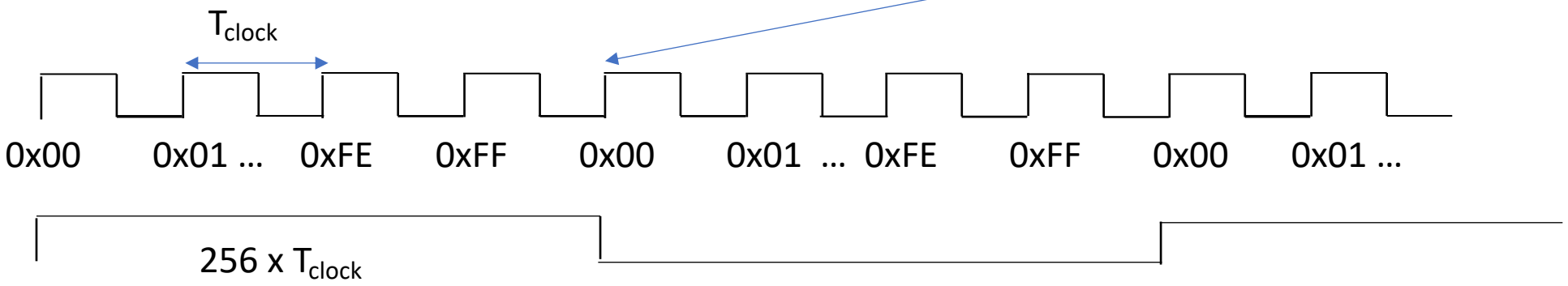
Let's build a square wave

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
}
```



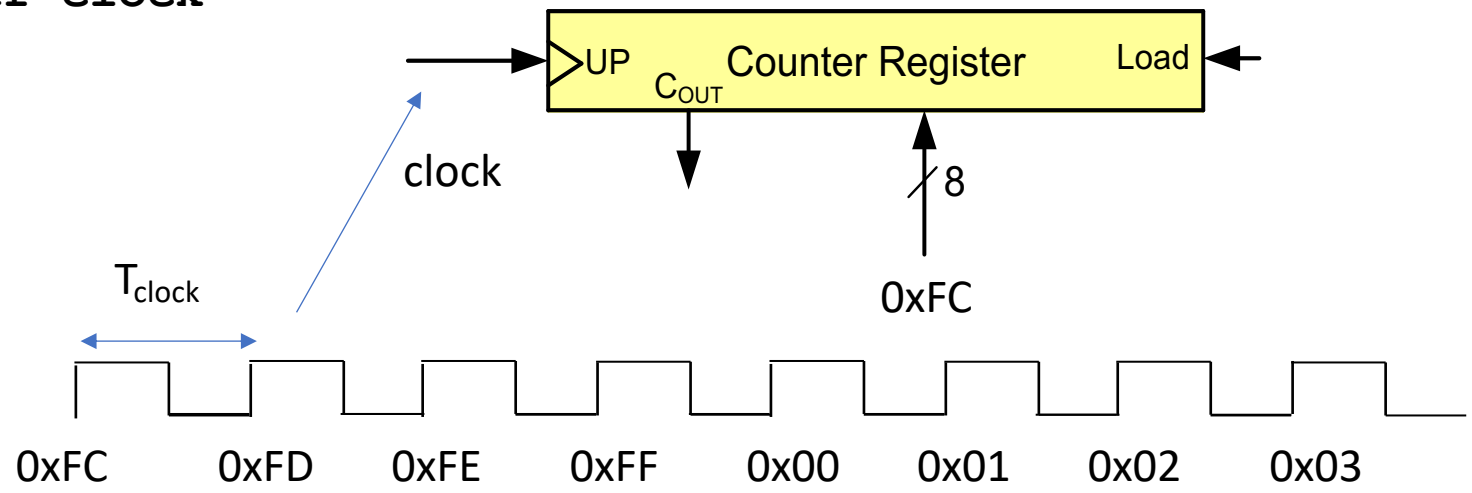
counter rolls-over from FF to 00; C_{out} = 1
it gets reset back to 0



square wave toggles the led every 256 x T_{clock} seconds

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


```
load 0xFC into counter
start counter clock
```

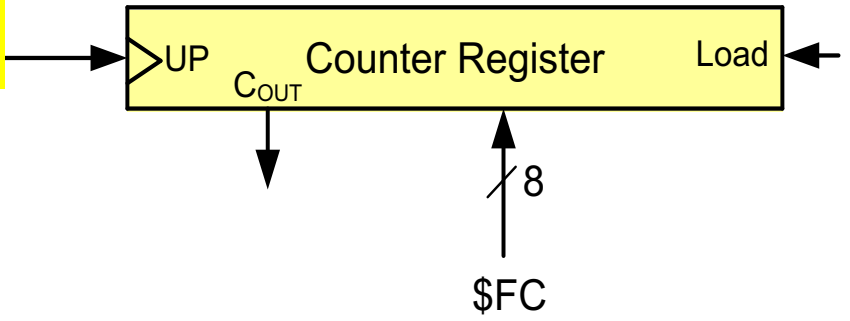


counter rolls-over from FF to 00; C_{out} = 1

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

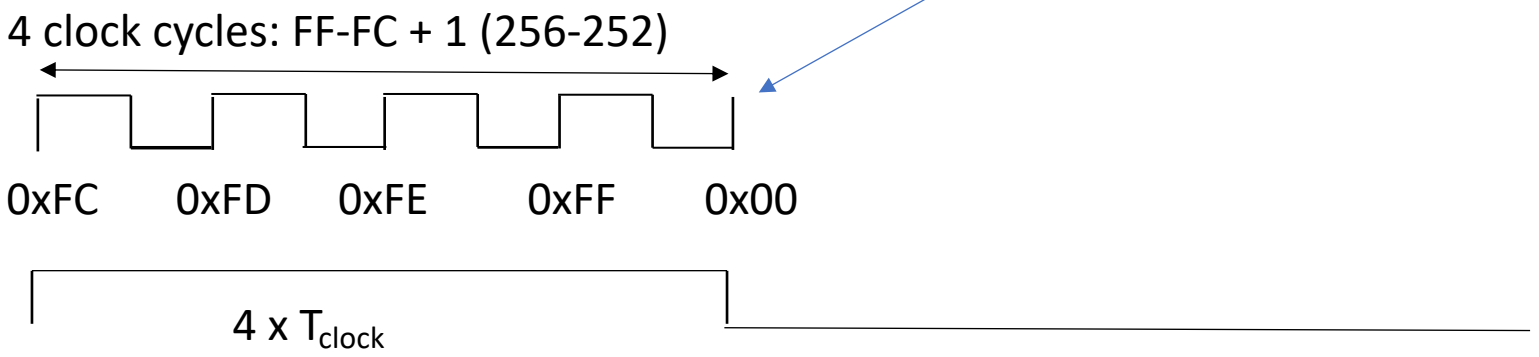
pseudocode to turn on an led for 4 clock cycles

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



0x00 0
0x01 1
…
0xFC 252
0xFD 253
0xFE 254
0xFF 255
0X00 0

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

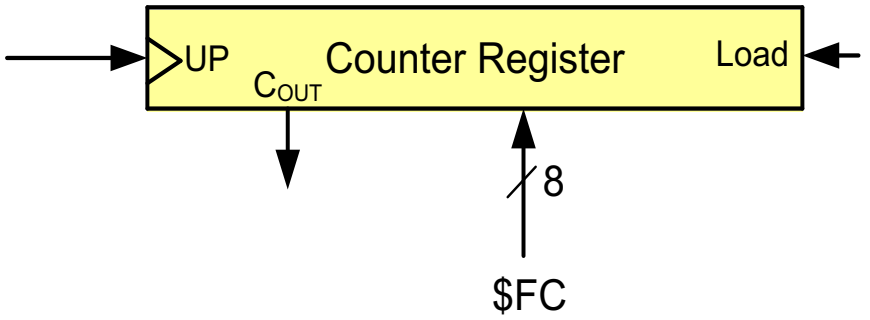


led remains on for 4 x T_{clock} seconds

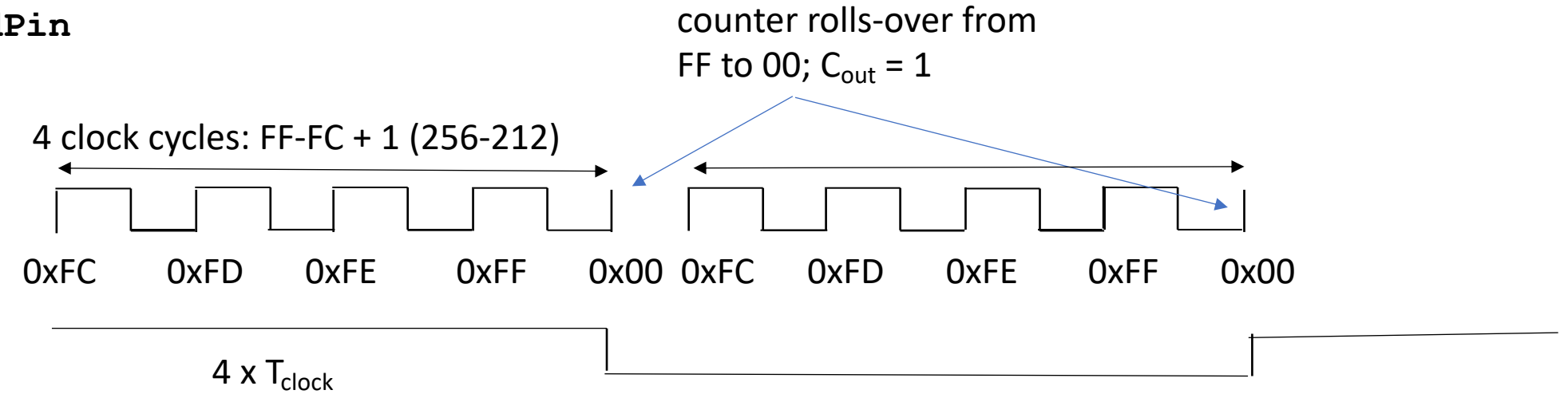
Let's build a 4 cycle (or 4 count) square wave

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

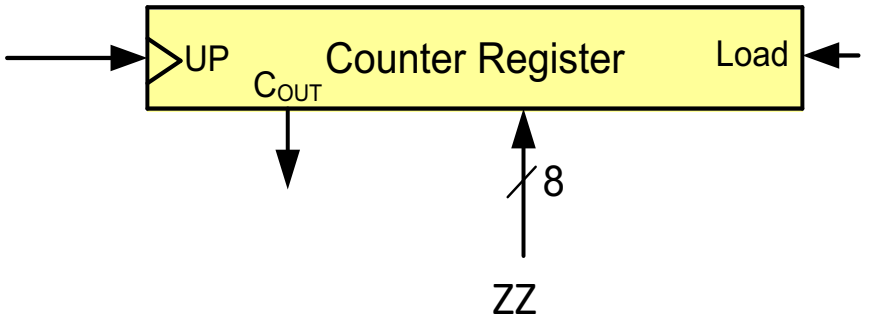


square wave toggles the led every 4 x T_{clock} seconds

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 Cout=1
    set Cout back to 0
    toggle ledPin
}
```



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

Hexadecimal:

N = FF-ZZ+1

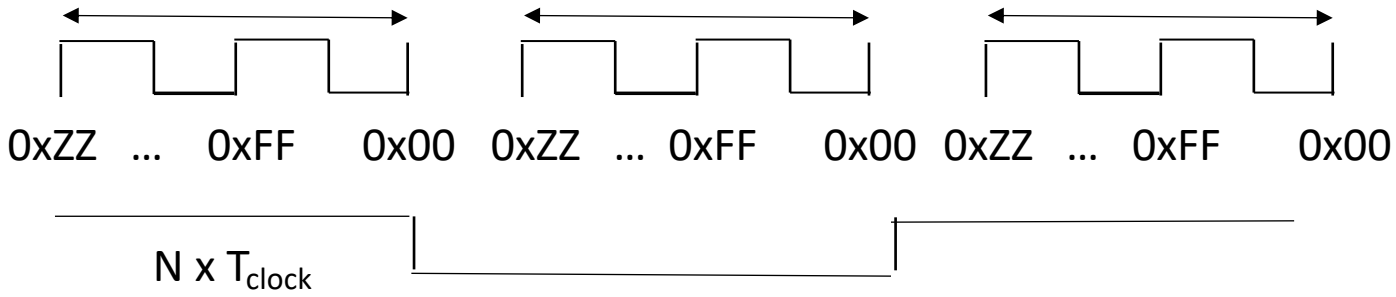
ZZ = FF – N + 1

Decimal:

N_{10}=255- ZZ_{10} +1

ZZ_{10}=255- N_{10} + 1

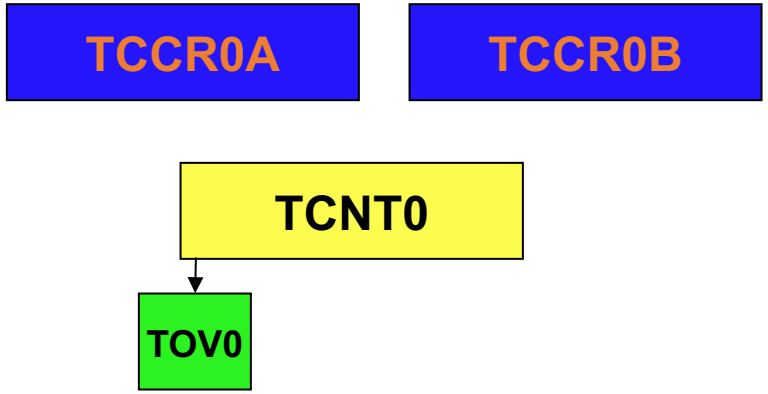
N clock cycles: N=FF-ZZ + 1 (N_{10}=255-ZZ_{10}+1)



square wave toggles the led every N x T_{clock} seconds

Timer/Counter 0 (8-bit; simplified)

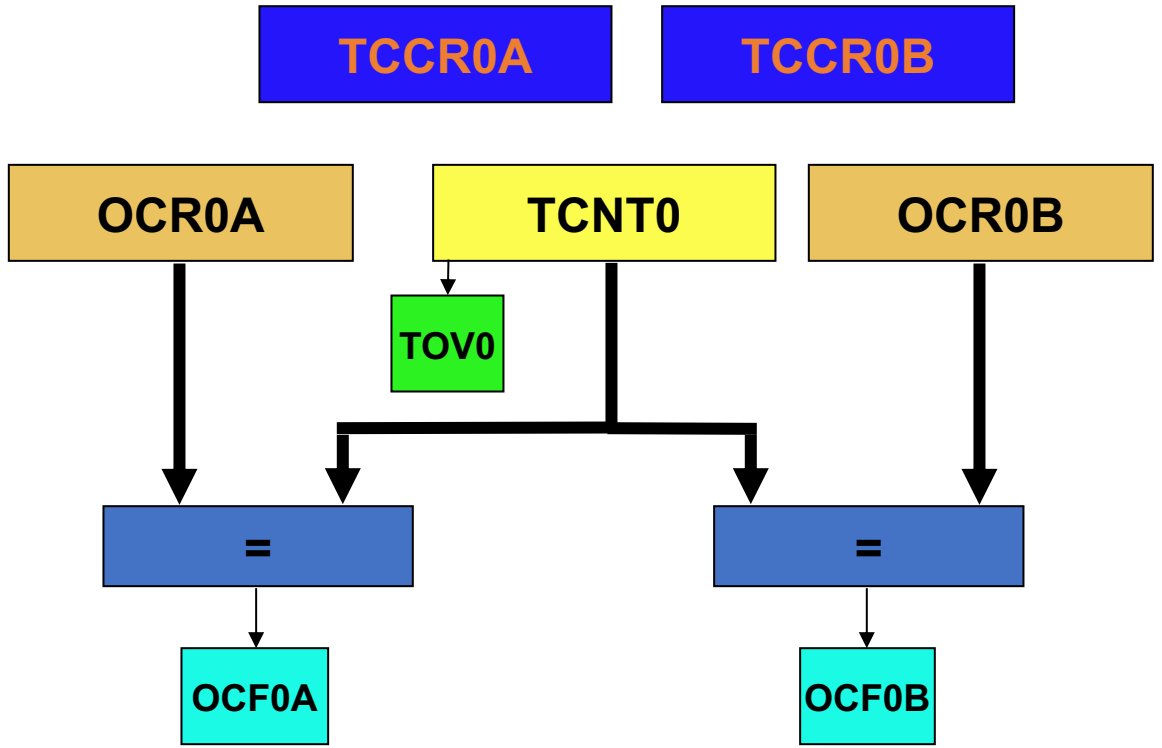
- TCNT0 (Timer/Counter register)
- TOV0 (Timer Overflow flag)
- TCCR0A (Timer Counter control register A)
- TCCR0B (Timer Counter control register B)



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**

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)

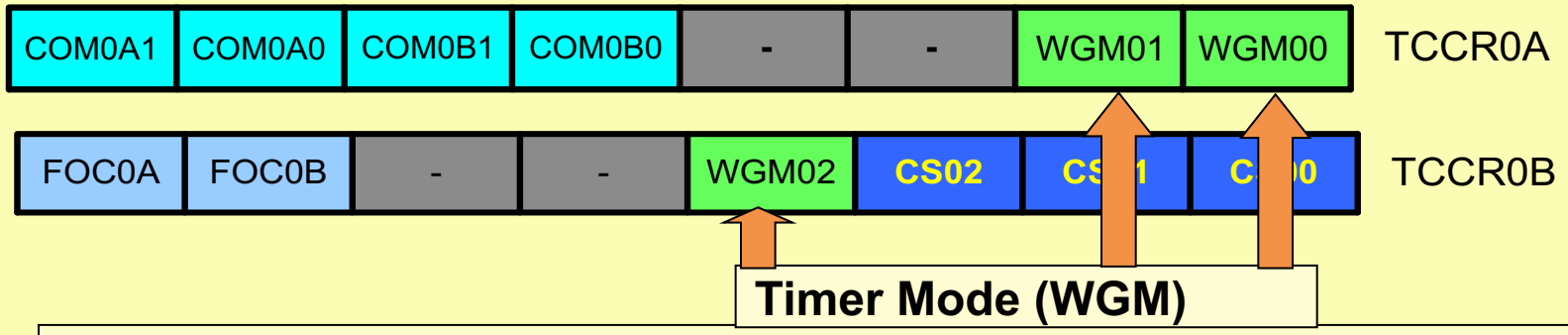


COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00	TCCR0A
FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00	TCCR0B
-	-	-	-	-	OCF0B	OCF0A	TOV0	TIFR0

Timer Counter Control Register 0 A

Timer Counter Control Register 0 B

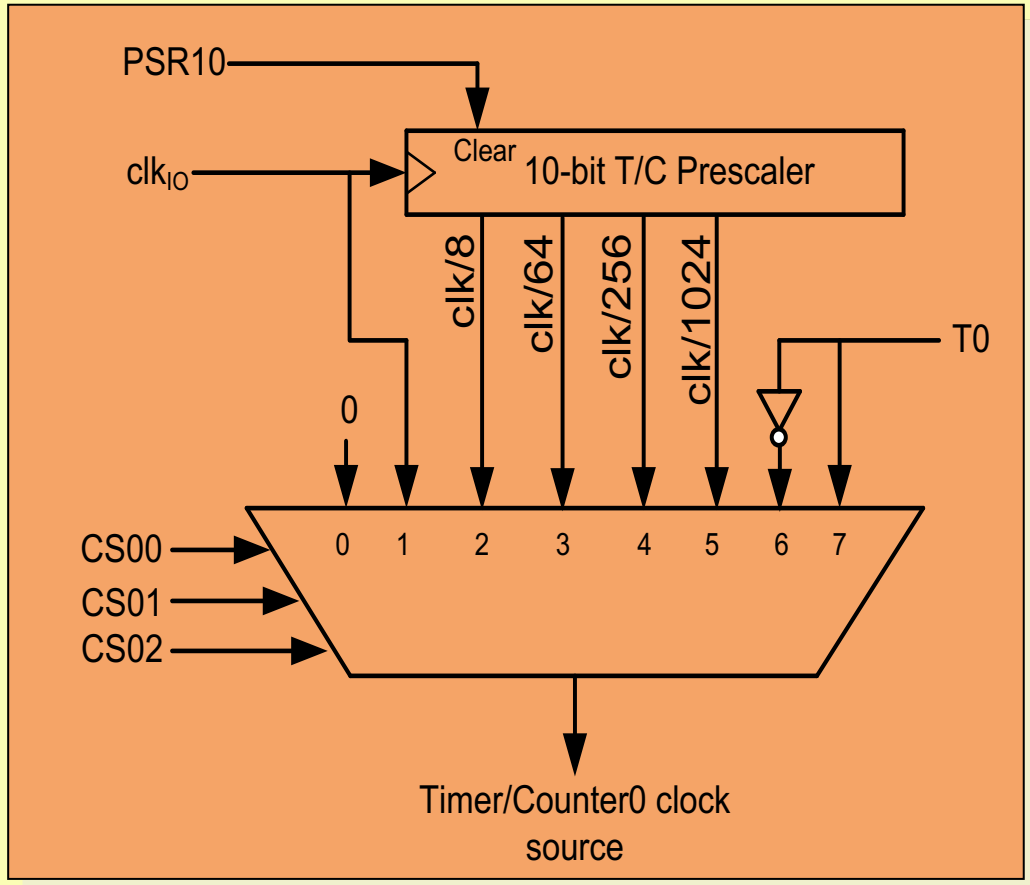
Timer Interrupt Flag Register 0

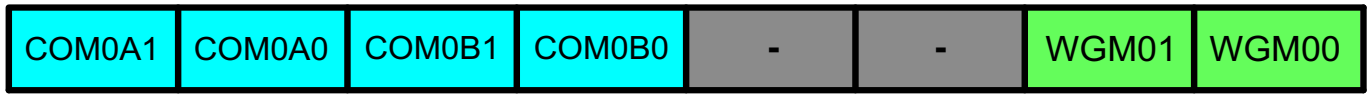


WGM02	WGM01	WGM00	Comment
0	0	0	Normal
0	0	1	Phase correct PWM
0	1	0	CTC (Clear Timer on Compare Match)
0	1	1	Fast PWM
1	0	0	Reserved
1	0	1	Phase correct PWM
1	1	0	Reserved
1	1	1	Fast PWM

COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00	TCCR0A

FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00	TCCR0B





TCCR0A

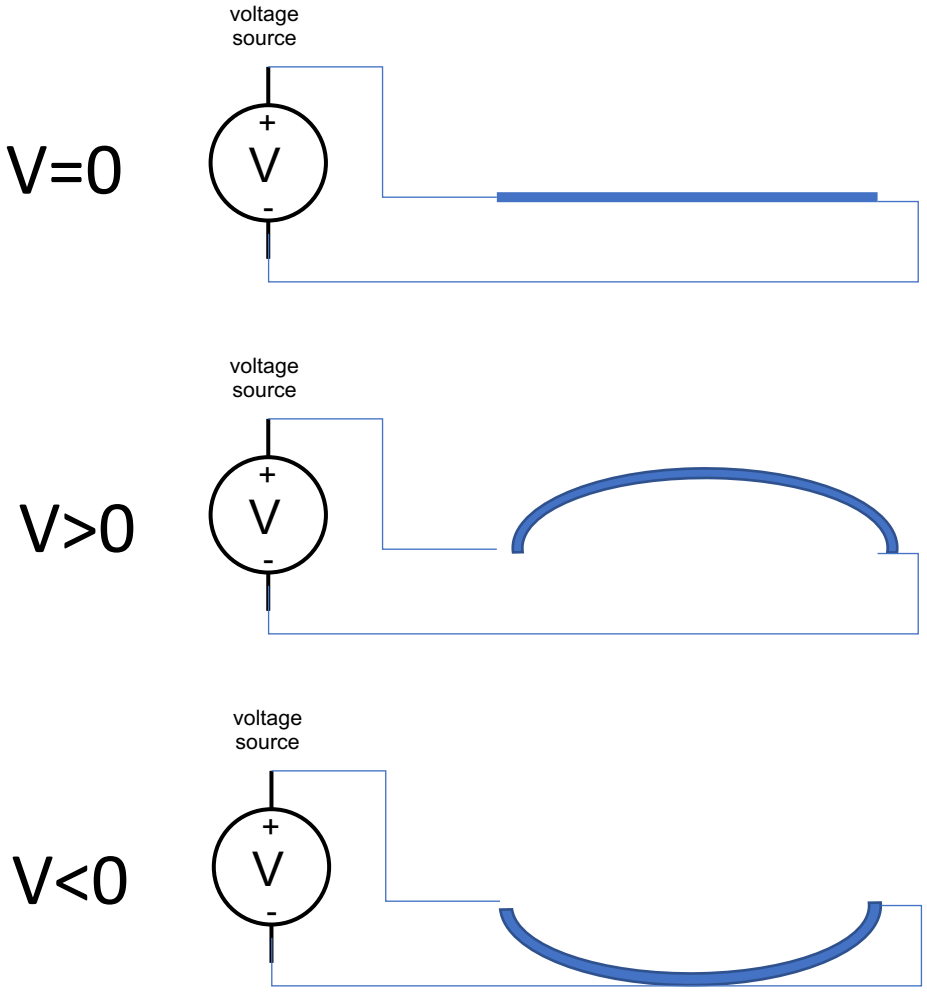
TCCR0B

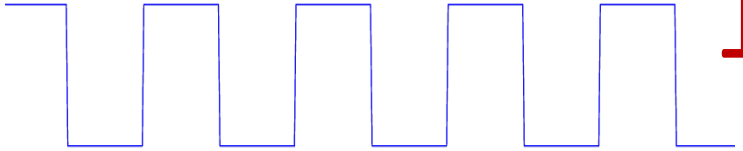
Clock Selector (CS)

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

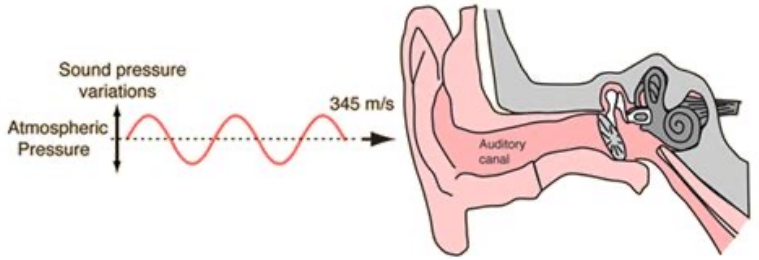
Write a program to create a 1 KHz square wave applied to a piezo-electric speaker

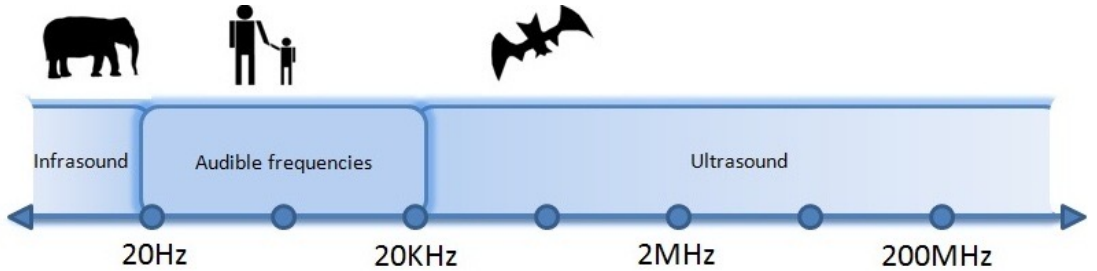
piezoelectric speaker





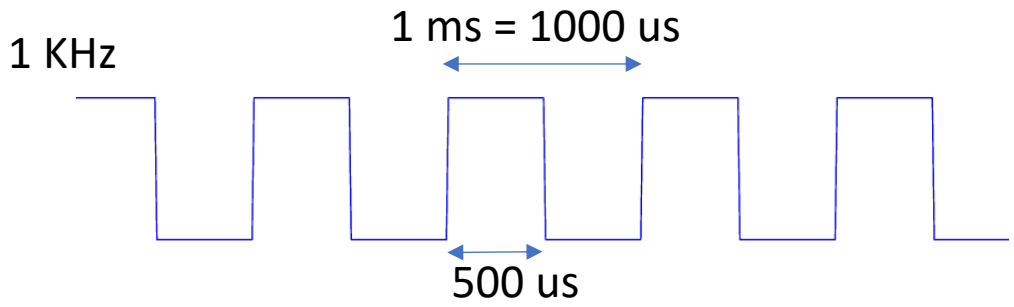
oscillating voltage waveform





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

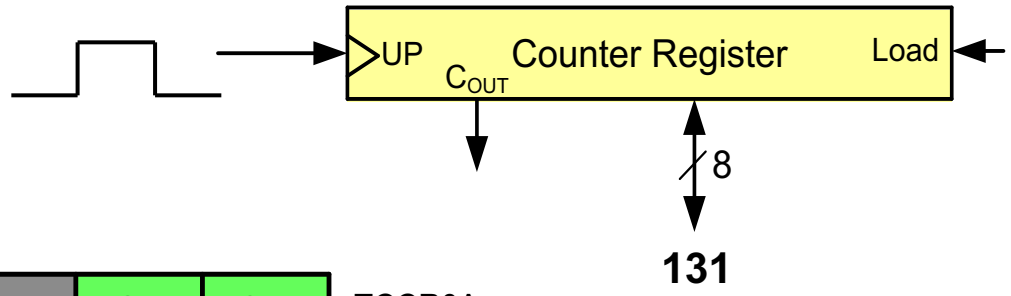
using Timer0 in normal mode and the 1:64 pre-scaler to create the delay

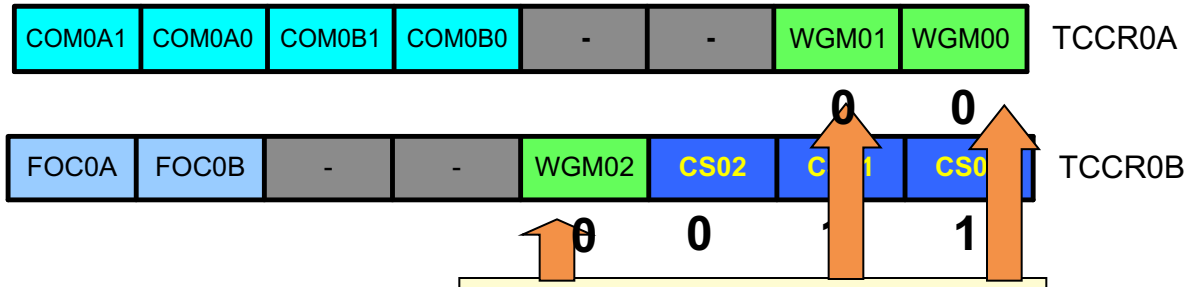


1. set up the GPIO bit
2. Pre-load counter TCNT0 = xxxx
2. Set operating mode (normal)
3. Start 1:64 clock
4. Wait till counter overflows
5. Toggle the GPIO bit

1 KHz wave = 1 msec period, Tdelay = 0.5 ms = 500 us
328P clock on Arduino Uno: Fosc = 16MHz; Tosc = 1/Fosc=0.0625 us
Prescaler 1:64 → Tclock = 64 * 0.0625 us = 4 us
cycles needed per count-up: 500 us/4 us = 125
preload: 256 – 125 = 131

Pre-load counter:
256 – 125 = 131
`TCNT0 = 131;`



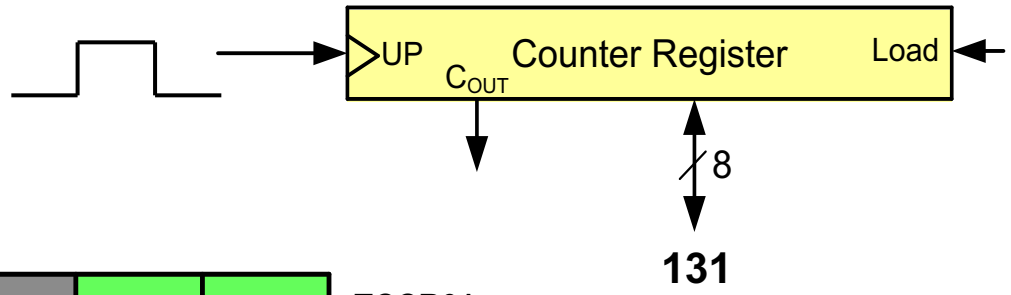


TCCR0A = 0x00;
TCCR0B = 0x03;

Timer Mode (WGM)

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

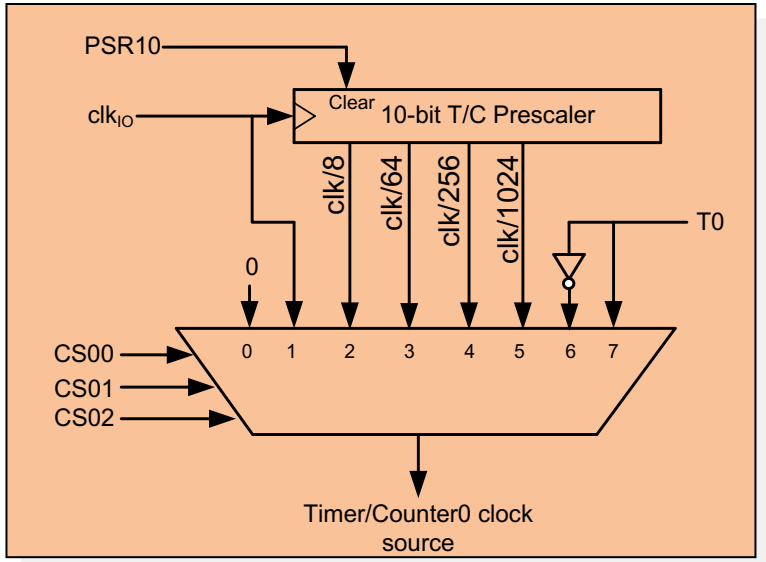
Pre-load counter:
256 – 125 = 131
`TCNT0 = 131;`



COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00	TCCR0A

FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00	TCCR0B
					0	**1**	**1**	

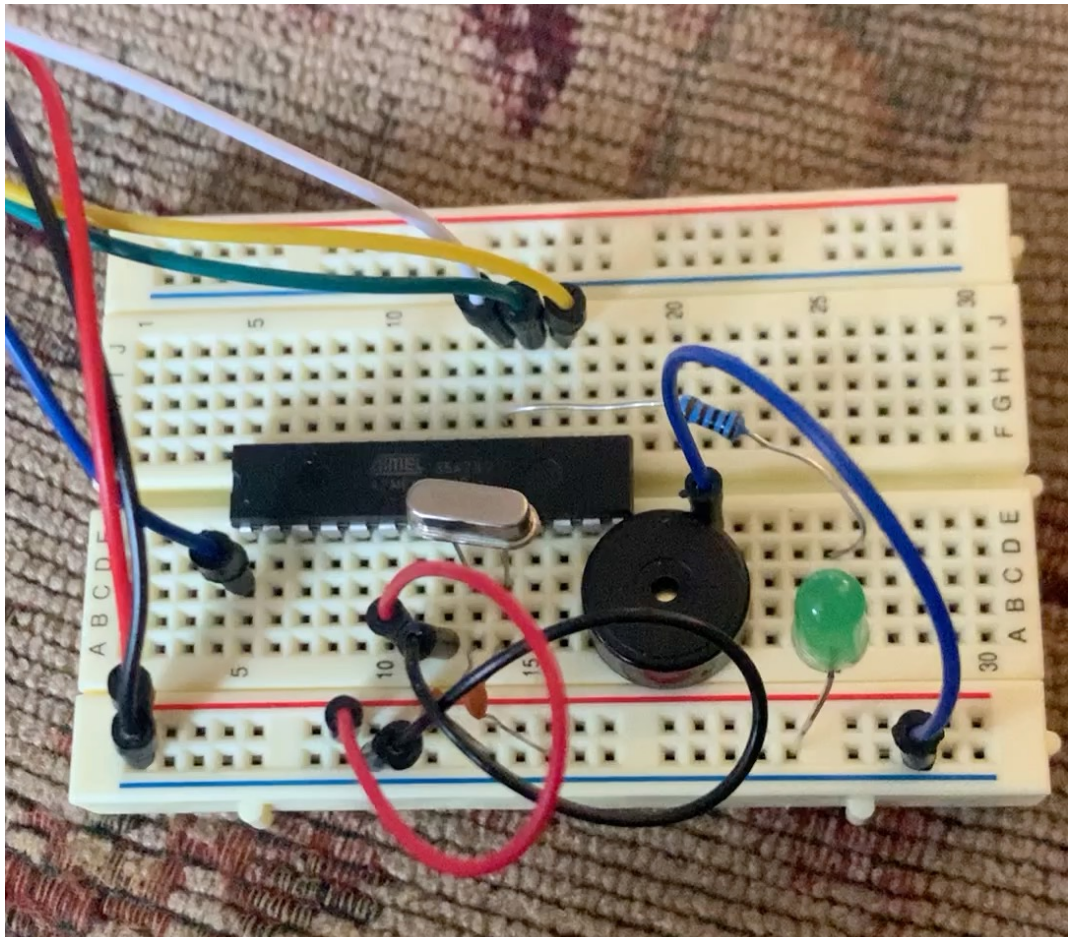
```
TCCR0A = 0x00;
TCCR0B = 0x03;
```



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

one_kHz.c > ...

```c
/* one_khz.c This code creates a 1 KHz square wave for a piezo speaker on PB0
using timer 0. Timer arithmetic:  1 KHz square wave has a period of 1 ms.
Need to toggle the speaker every 0.5 ms or 500 us. 16 MHz clock has T=1/16 uSEC.
Need to divide the clock by 500*16=8000.Since the time can only count to 256,
we need to pre-divide the clock. Pre-divide options are 8, 64, 256, and 1024.
This code uses 64, then counts to 8000/64=125. Clock is pre-loaded with 256-125=131.
D. McLaughlin ECE-231 demo Spring 2022. Version 1.*/

#include <avr/io.h>

int main(void){
    // Set up the GPIO bit
    DDRB = 1<<DDB0;

    while(1){
        // Pre-load counter
        TCNT0 = 131;
        // Set operating mode to normal, clk/64
        TCCR0A = 0;
        TCCR0B = 1<<CS01|1<<CS00;
        // Wait for timer to roll over
        while ((TIFR0 & (1<<TOV0))==0);
        // Stop the clock
        TCCR0B = 0;
        // Clear the roll over flag
        TIFR0 |= 1<<TOV0;
        // Toggle the Speaker
        PORTB ^= 1<<PORTB0;
    }
}

```



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

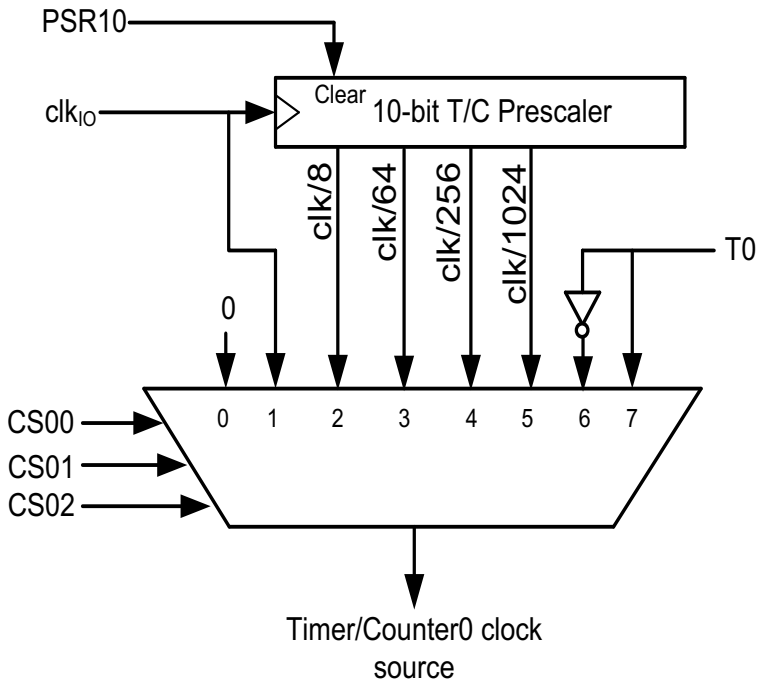
Use the following prescalars:

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 MHz → $T_{machine\ cycle}$ = 1/16 MHz = 0.0625 us

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

Prescaler = 1:1024 → T_{clock} = 1024 * 0.0625 us = 64 us

8000 us/64 us = 125; preload: 256 – 125 = 131; prescaler: 5

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

Prescaler = 1:256 → T_{clock} = 256 * 0.0625 us = 16 us

2000 us/16 us = 125; preload: 256 - 125 = 131; prescaler: 4

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

Prescaler = 1:64 → T_{clock} = 64 * 0.0625 us = 4 us

500 us/4 us = 125; preload: 256 – 125 = 131; prescaler: 3

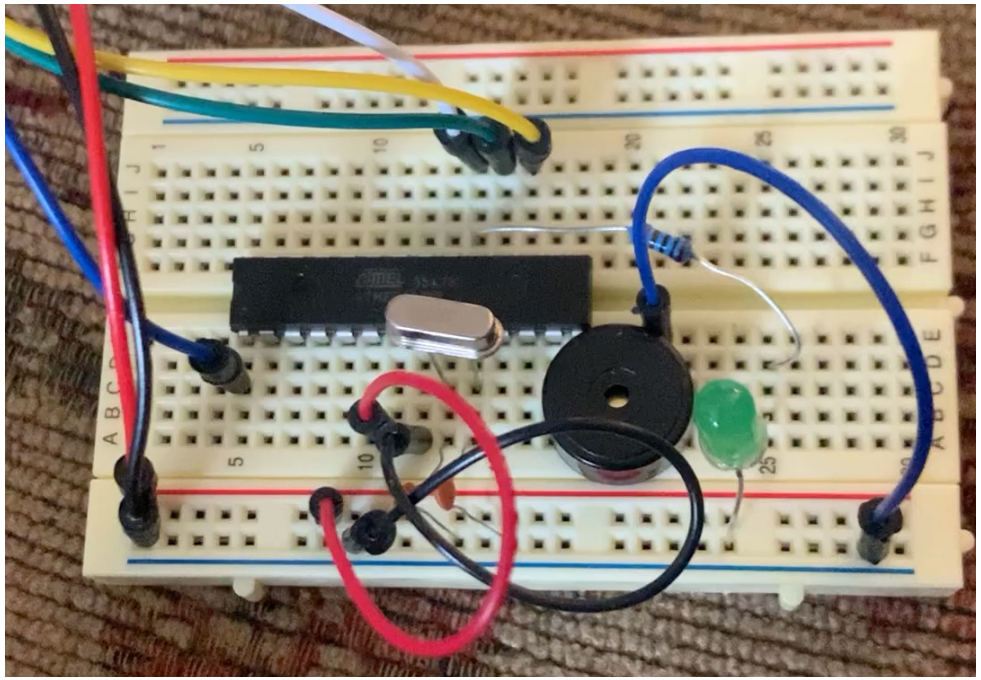
(d) for f=4 kHz= 4000 Hz, T=1/4000 = 0.25 ms; T_{delay} = T/2 = 0.125 ms = 125 us

Prescaler = 1:8 → T_{clock} = 8 * 0.0625 us = 0.5 us

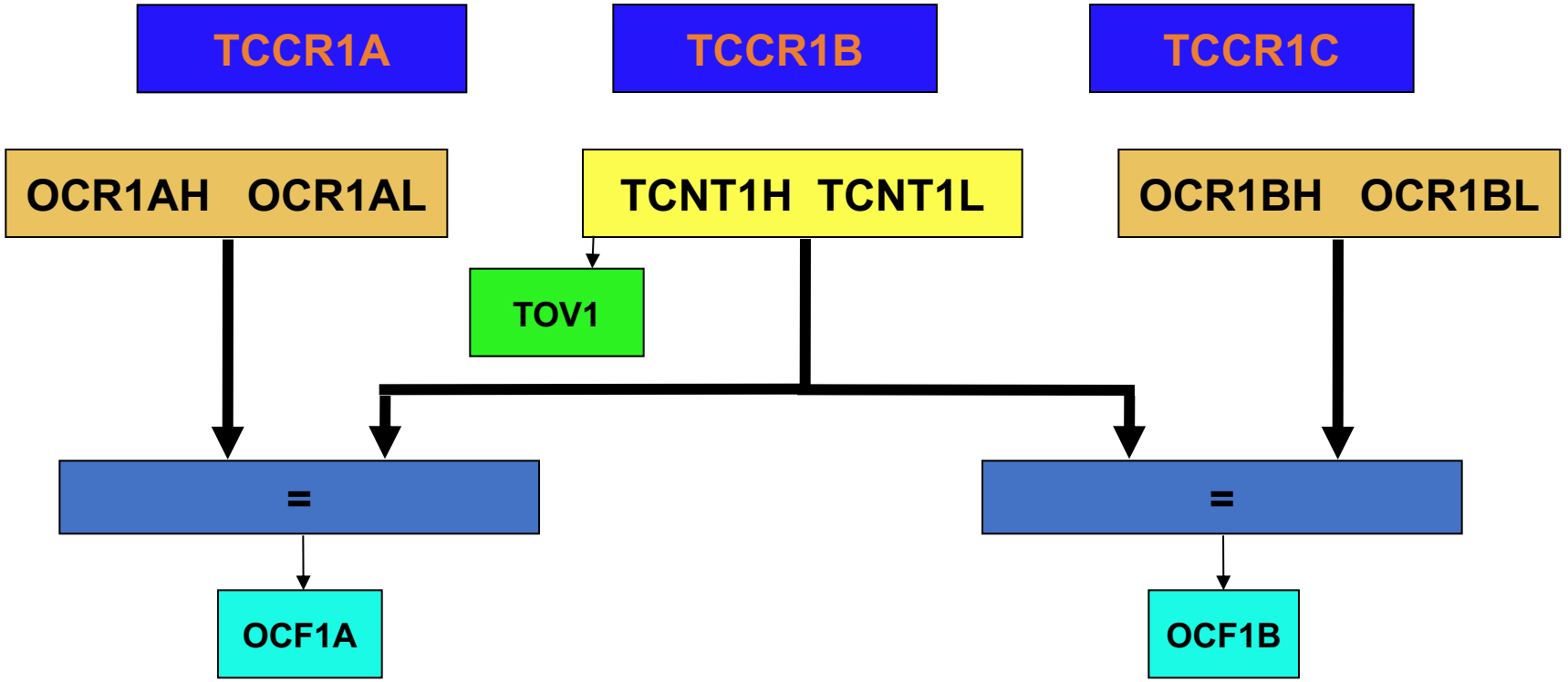
125 us/0.5 us = 250; preload 256 – 250 = 6; prescaler: 2

four_tones.c > ...

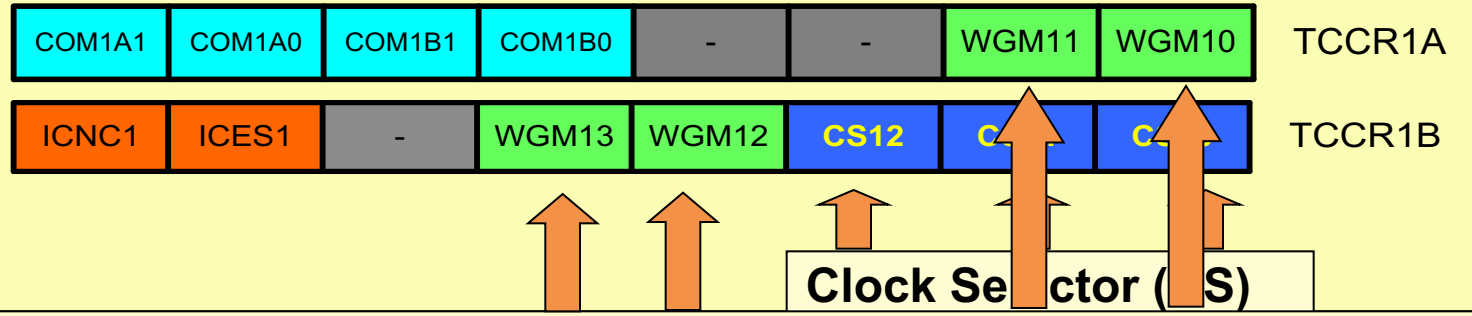
```c
/* four_tones.c  Plays a sequence of tones: 62.5 Hz, 250 Hz,
1 KHz, 4 KHz, repeatedly. Piezo Speaker on PB0.
D. McLaughlin ECE-231 Spring 2022 */

#include <avr/io.h>
int main(void)
{
    unsigned char preloads[4]= {131,131,131,6};
    unsigned char prescaler[4]={5, 4, 3, 2};
    float frequency[4]={62.5, 250, 1000, 4000};
    unsigned int i,j;

    DDRB = 1 << DDB0; //PB0 is output pin
    while (1) {
        for (i=0; i<4; i++)
            for (j=0; j<3*frequency[i]; j++){
                TCNT0 = preloads[i]; //preload value for tone
                TCCR0A = 0x00; //normal mode
                TCCR0B = prescaler[i]; //prescaler value for tone
                while ((TIFR0 & (1 << TOV0)) == 0); //wait till done
                TCCR0B = 0; //stop the clock
                TIFR0 = (1 << TOV0); //clear the overflow flag
                PORTB ^= 1 << PORTB0; //toggle the output pin
            }
    }
}
```



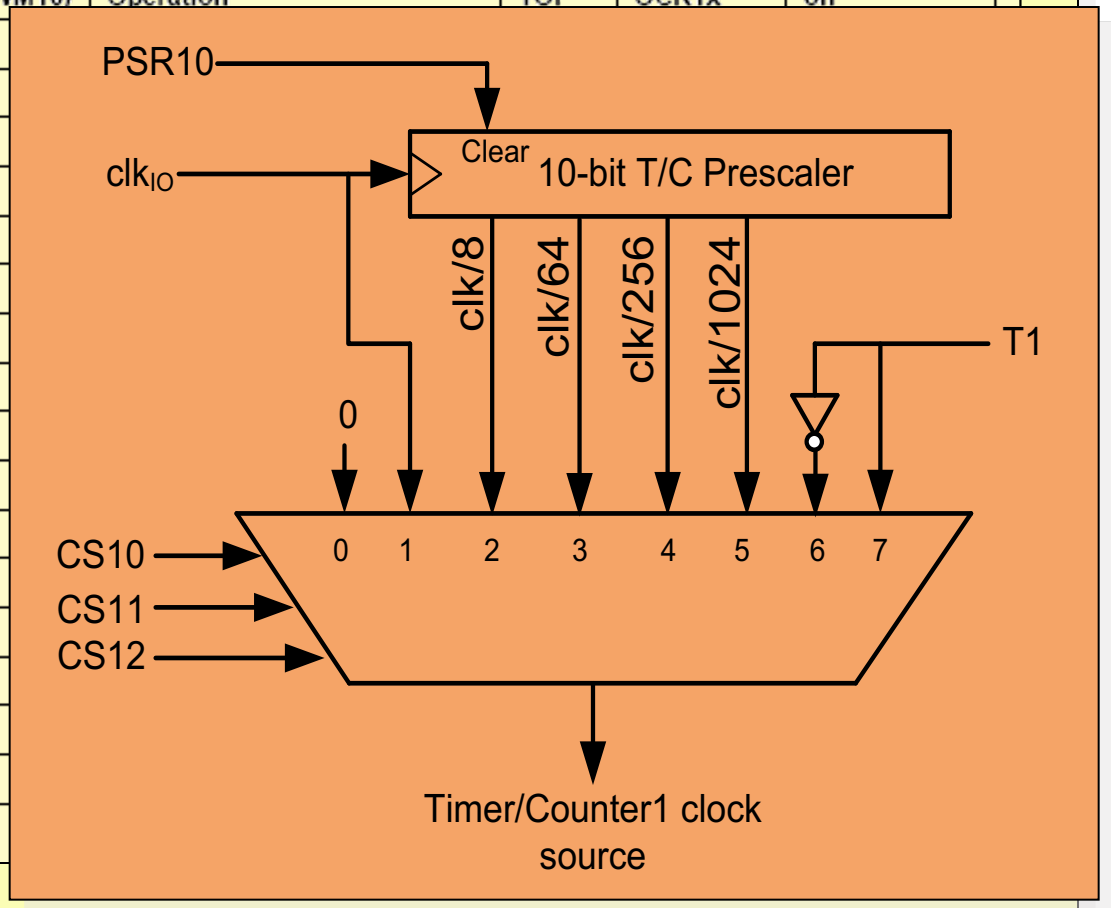
Timer 1



TIFR1

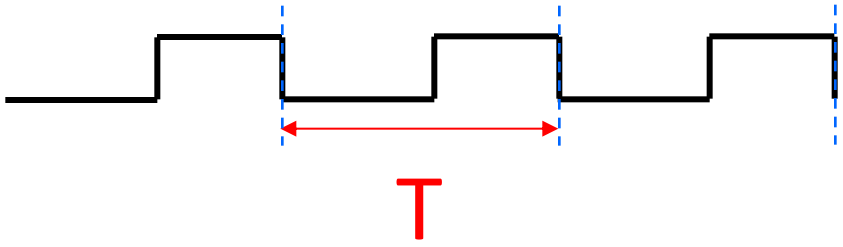


Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x	TOV1 Flag Set on
0	0	0	0					
1	0	0	0					
2	0	0	1					
3	0	0	1					
4	0	1	0					
5	0	1	0					
6	0	1	1					
7	0	1	1					
8	1	0	0					
9	1	0	0					
10	1	0	1					
11	1	0	1					
12	1	1	0					
13	1	1	0					
14	1	1	1					
15	1	1	1					



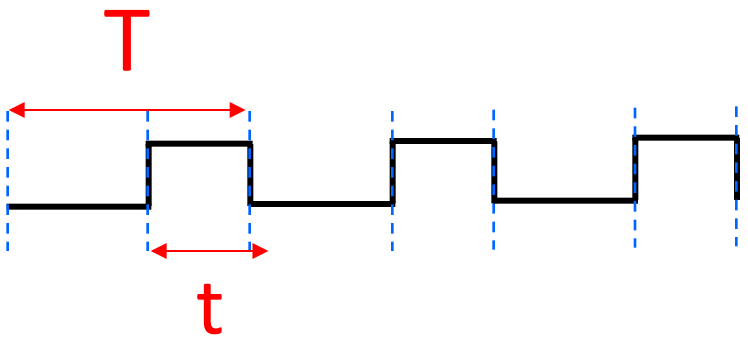
Waveform Characteristics

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

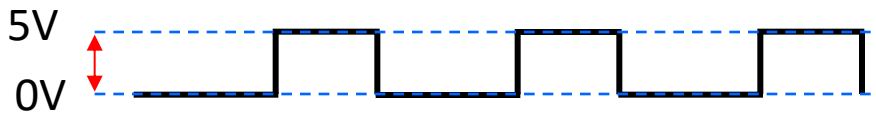


- Duty cycle

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



- Amplitude

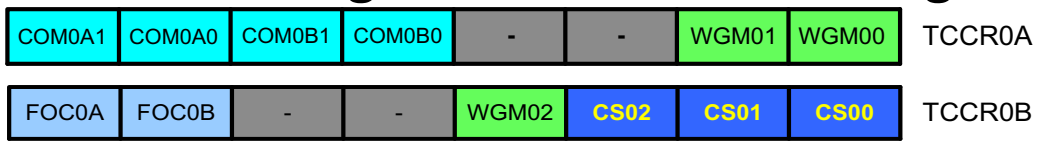


26

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: 16MHZ/64 = 250 KHz
Timer clock period: 1/250KHz = 4 usec
*** *each count of the clock is 4 usec long* ***

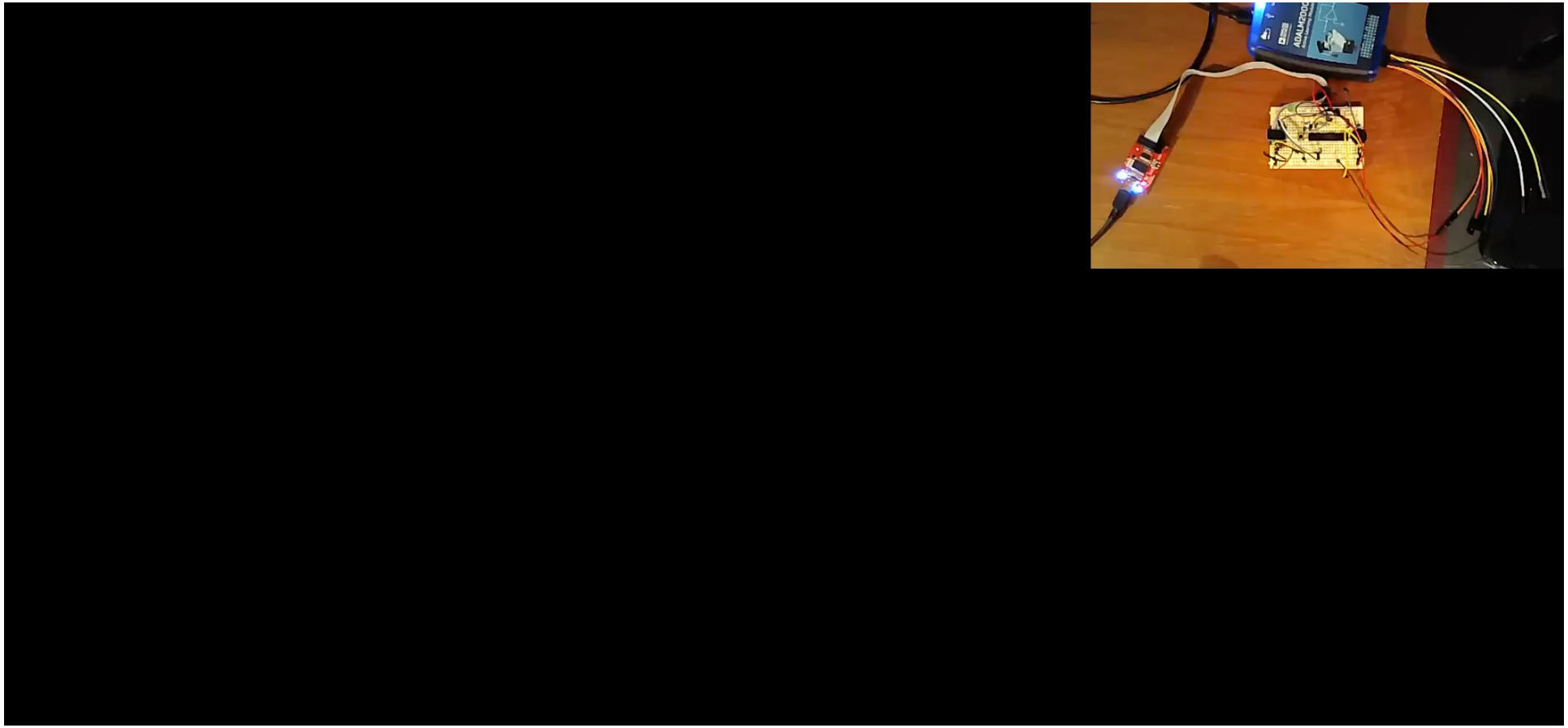
variable “pitch” controls the frequency of the output.

Timer is pre-loaded with TCNT1 = 65636-65636/pitch, so total timer delay is:
4 uSec * (65,536-TCNT1)
4 usec * (65,536 – (65,536 – 65,536/pitch))
4 usec * 65,536/pitch.

Square wave period is 2x timer delay
= 8 usec * 65,536/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)

```c
/* rampfreq.c  Ramps (increases) the frequency on PD6
every time button on PB0 is pressed.
D. McLaughlin 4/6/22 for ECE-231 Demo Spring 2022 */

#include <avr/io.h>
#include <util/delay.h>

int main(void){
    unsigned int pitch = 1;
    DDRD = 1<<DDD6;     // LED, speaker, oscope on PD6
    DDRB = 0;           // pushbutton switch on PB0
    PORTB = 1<<PB0;     // Set pullup on PB0

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

    while(1){
        // Change the frequency variable when switch is pressed
        if ( (PINB & (1<<PB0)) == 0){
            pitch*=2;
            // This delay prevents code from repeatedly changing the pitch
            // When finger is on the button
            _delay_ms(200);
        }
        // Counter delay
        TCNT1=65536-65536/pitch;        // Load the timer
        while ((TIFR1 & (1<<TOV1))==0); // Wait for rollover flag
        TIFR1 |= 1<<TOV1;               // Clear the rollover flat

        PORTD ^= 1<<PORTD6;             // Toggle the speaker & LED
    }
}
```



C rampfreqfunction.c > ...

```c
/* rampfreqfunction.c  Increases the frequency of square wave on PD6
every time a button on PB0 is pressed.
D. McLaughlin 4/6/22 for ECE-231 Demo Spring 2022 */

#include <avr/io.h>
#include <util/delay.h>
void timer1_init(void);
void delay(unsigned int);

int main(void){
    unsigned int pitch = 1; // This variable controls the pitch (freq)
    DDRD = 1<<DDD6;         // LED, speaker, oscope on PD6
    DDRB = 0;               // Push-button switch on PB0
    PORTB = 1<<PB0;         // Set pullup on PB0
    timer1_init();          // Initialize timer1

    while(1){
        if ( (PINB & (1<<PB0)) == 0){  // Change each time button is pressed
            pitch*=2;
            _delay_ms(200); // Delay avoids multiple changes per button press
        }
        delay(pitch);
        PORTD ^= 1<<PORTD6;             // Toggle the speaker & LED
    }
}

// Initialize timer1: normal mode (count up), divide clock by 64
void timer1_init(){
    TCCR1A = 0;         // Timer 1 Normal mode (count up)
    TCCR1B = 3;         // Divide clock by 64
}
// This function causes a delay of 0.26/pitch seconds.
void delay(unsigned int  pitch){
    TCNT1=65536-65536/pitch;
    while ((TIFR1 & (1<<TOV1))==0); // Wait for rollover flag
    TIFR1 |= 1<<TOV1;               // Clear the rollover flat
}
/*** 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
```

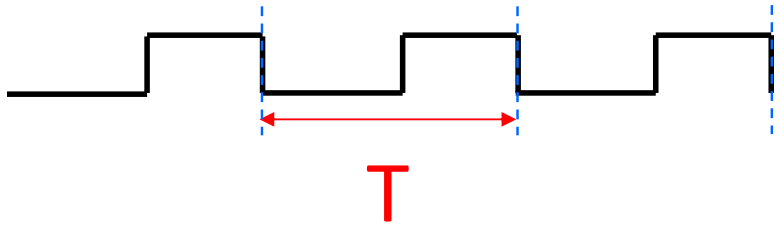
Timer delay loop

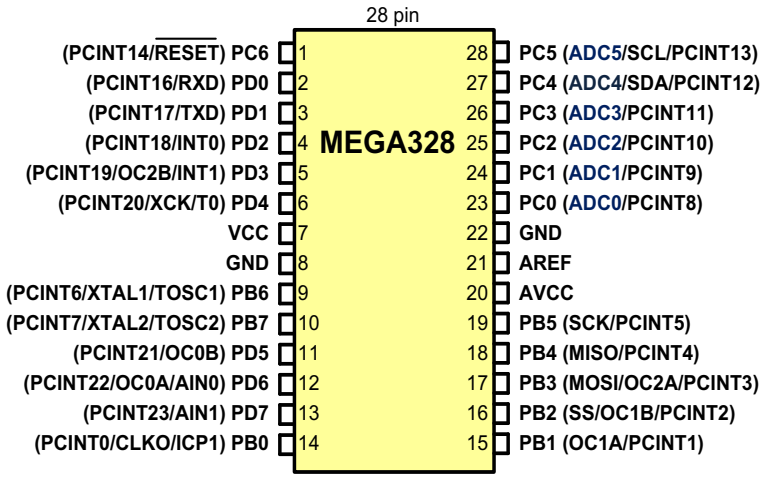
- pre-load value into timer
- timer counts from pre-load value to TOP
- 16 bit timer, so TOP = 2^{16}-1 = 65,536
- wait for overflow to occur
- clear overflow flag (a quirky feature of AVR mc
- toggle output pin (we happened to use PD6)

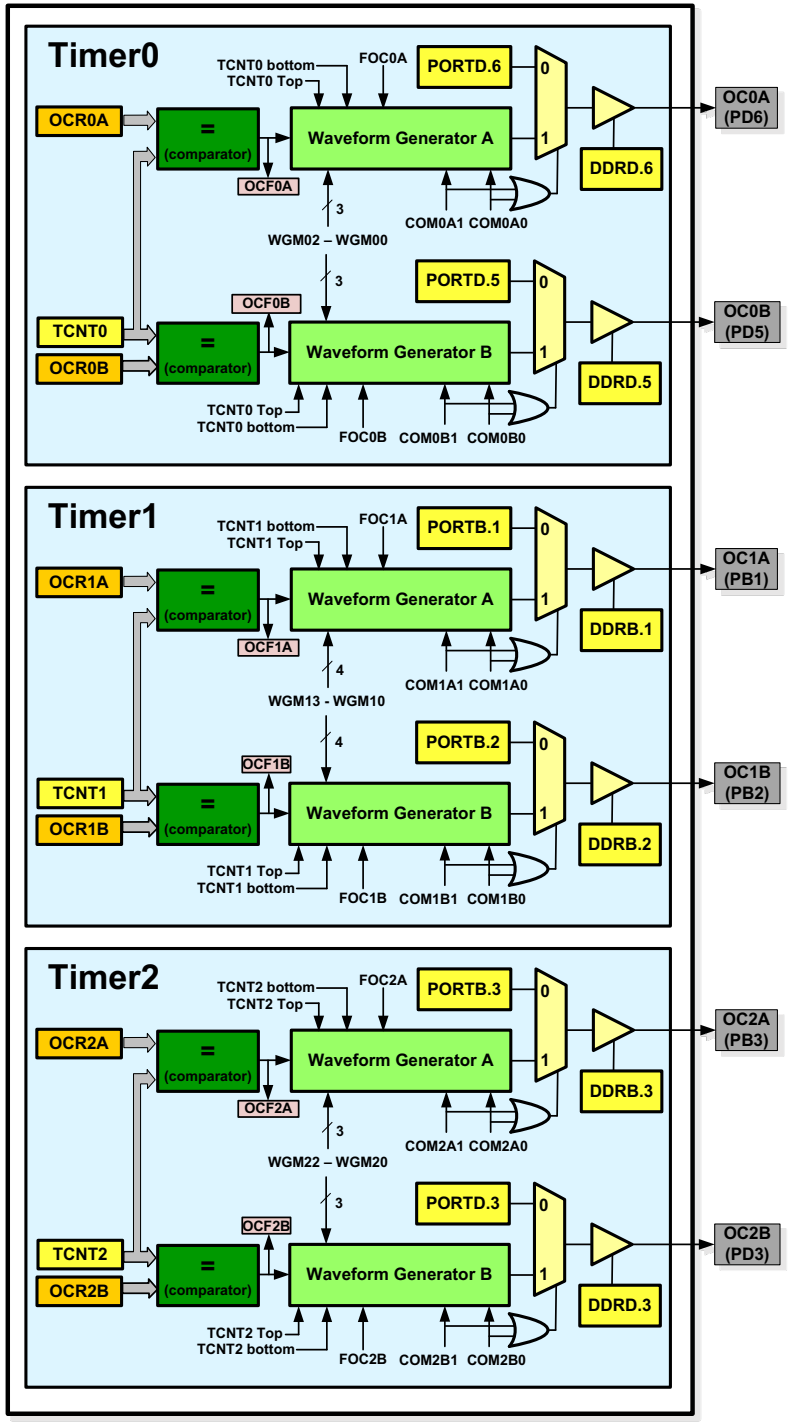
```
// Counter delay
TCNT1=65536-65536/pitch;
while ((TIFR1 & (1<<TOV1))==0);
TIFR1 |= 1<<TOV1;

PORTD ^= 1<<PORTD6;
```

LED, speaker, oscilloscope on PD6

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

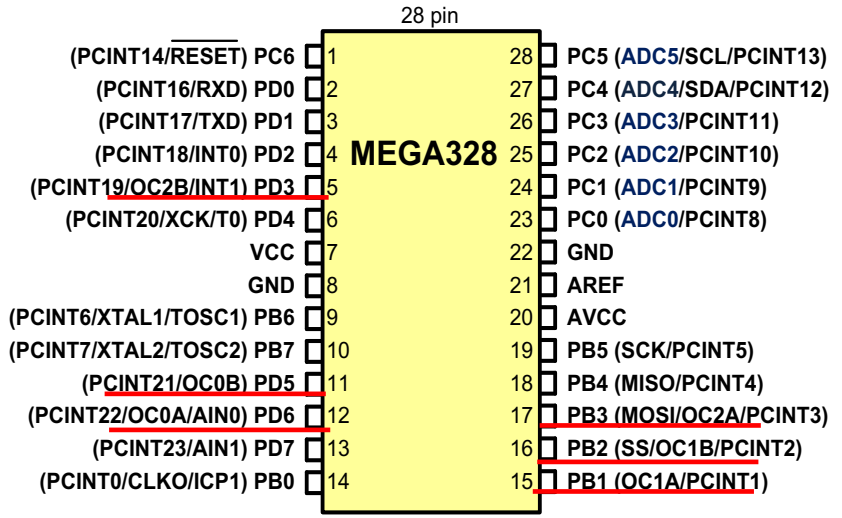




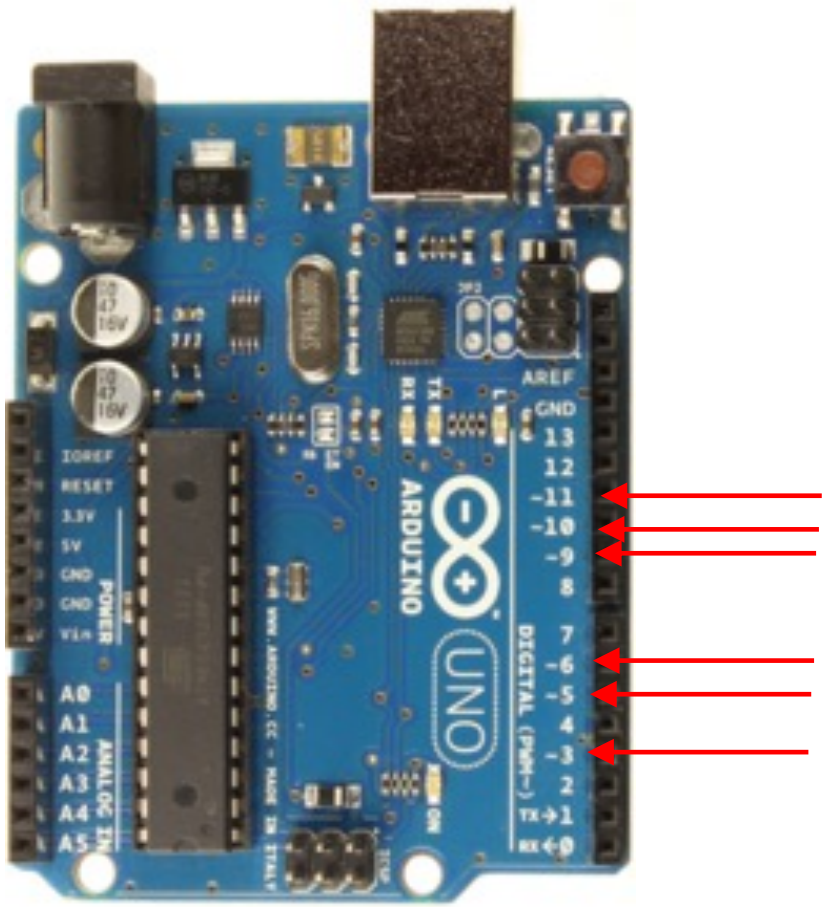


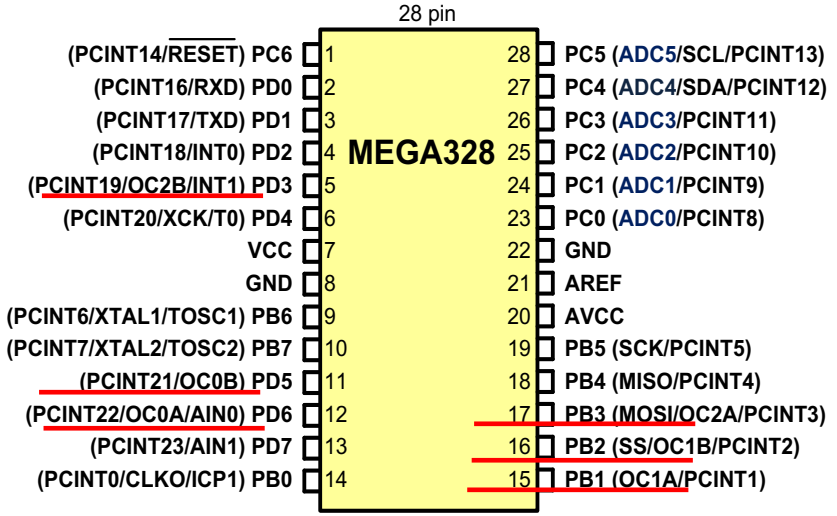
Each timer/counter has 2 waveform generators

6 waveform outputs share 6 GPIO pins:
Timer0: OC0A/PD6 & OC0B/PD5
Timer1: OC1A/PB1 & OC1B/PB2
Timer2: OC2A/PB3 & OC2B/PD3

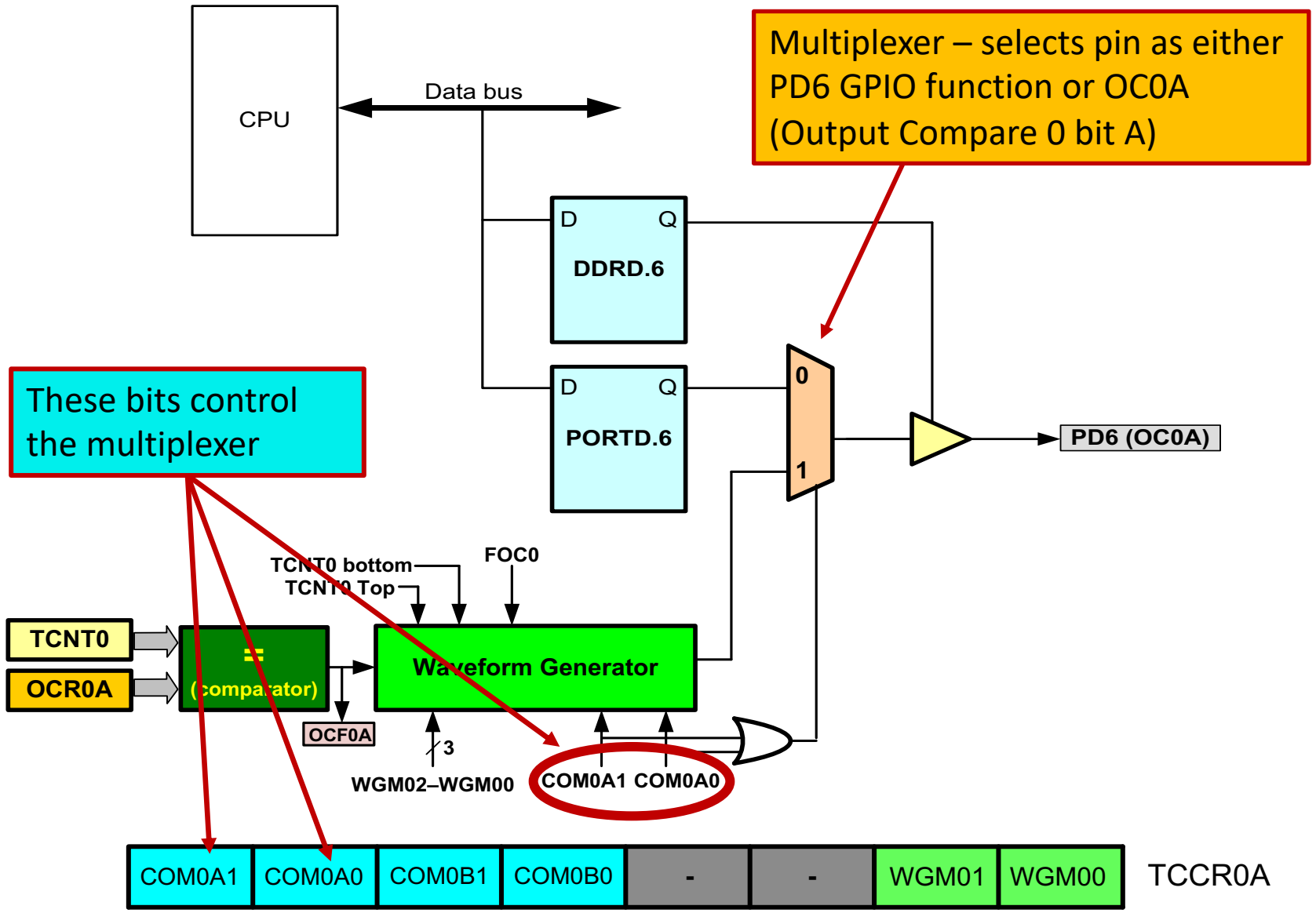


FYI: these 6 pins correspond to Arduino Uno pins 3, 5, 6, 9, 10, 11 (~PWM pins)



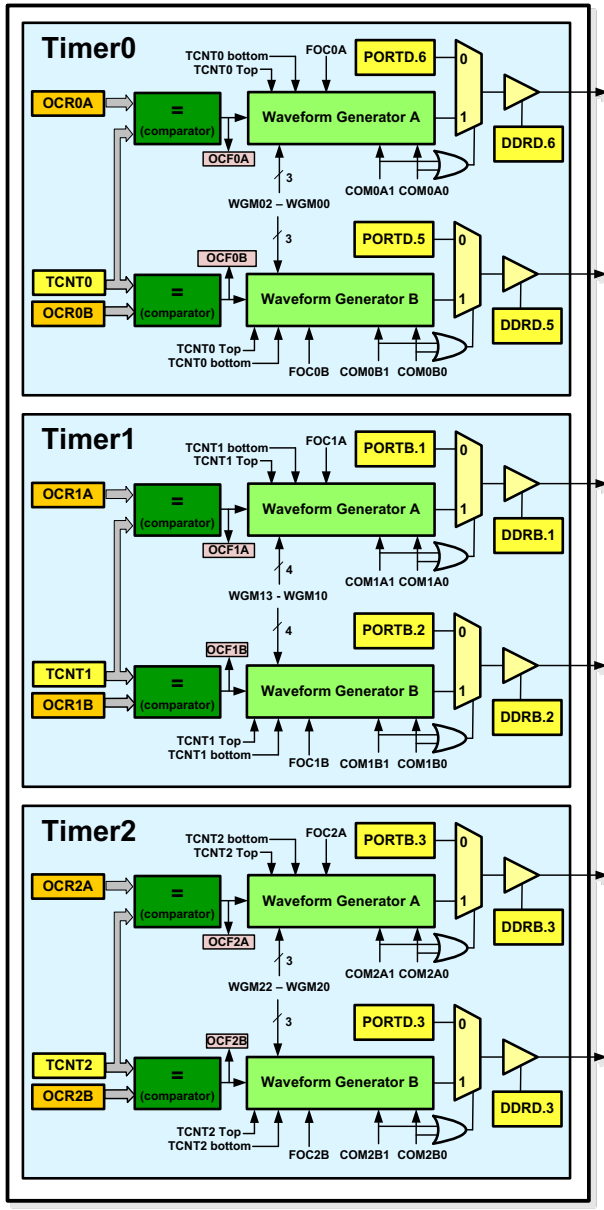


Waveform Generator 0, output A



34

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



Timer0 Output Comparator 0 Waveform A (OC0A)

Timer0 Output Comparator 0 Waveform B (OC0B)

Timer1 Output Comparator 1 Waveform A (OC1A)

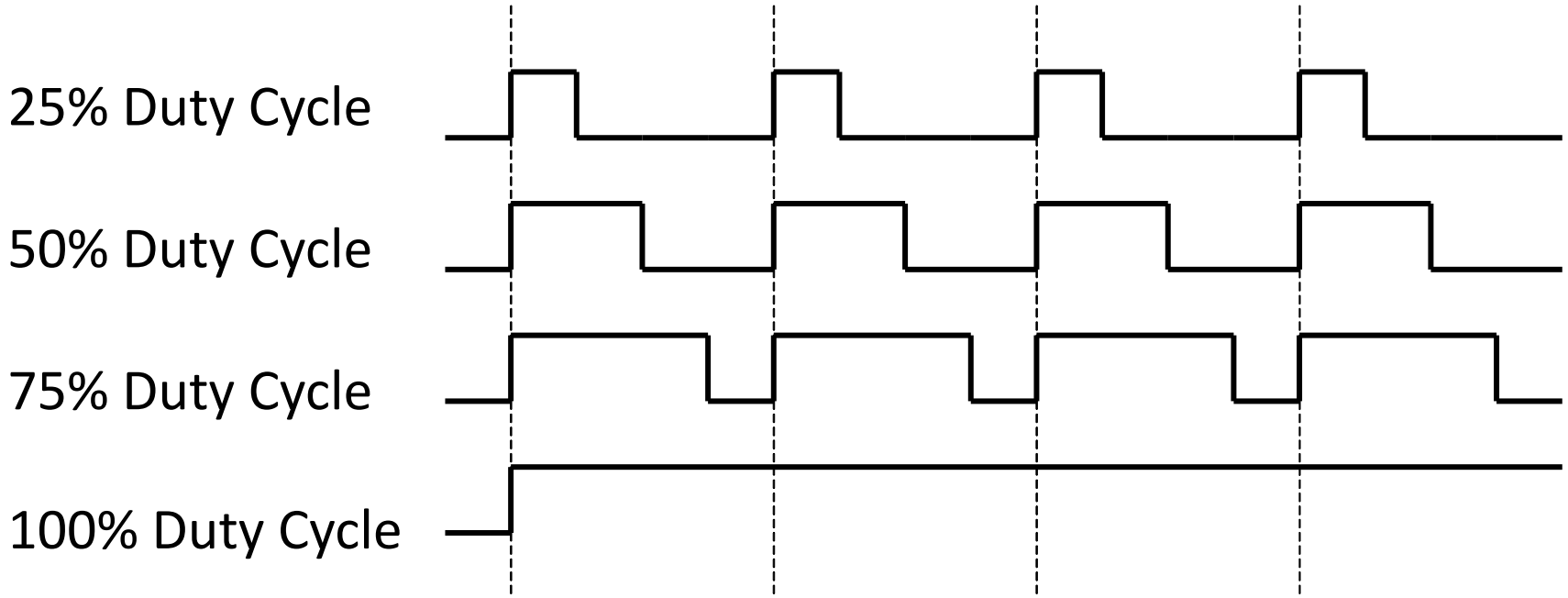
Timer1 Output Comparator 1 Waveform B (OC1B)

Timer2 Output Comparator 2 Waveform A (OC2A)

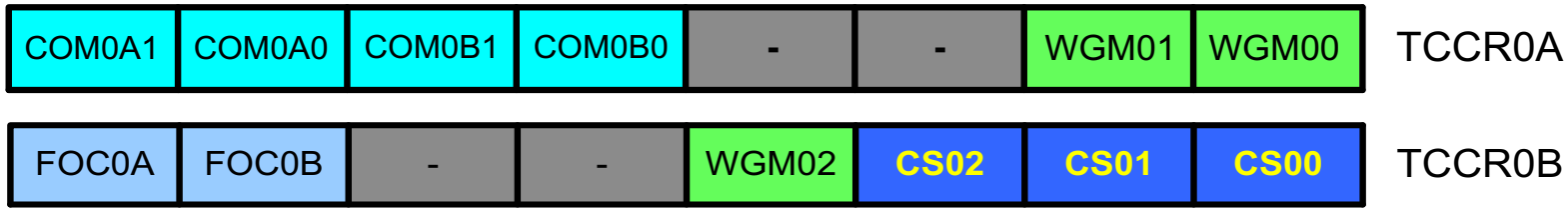
Timer2 Output Comparator 2 Waveform B (OC2B)

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)



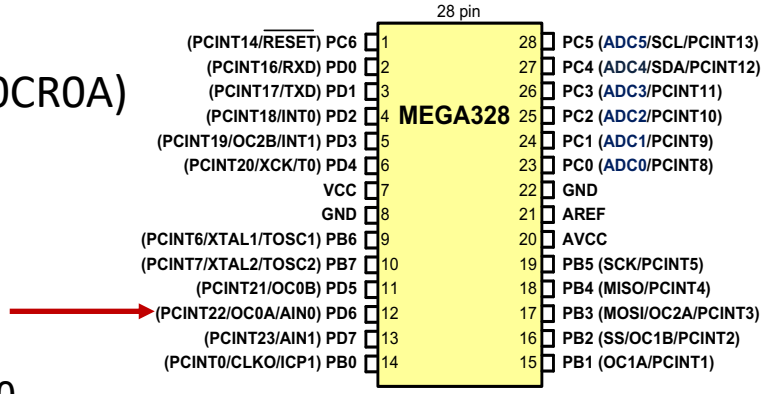
COM0A1:COM0A0 = 10 (clear OC0A on compare match; set on top)
WGM02:WGM00 = 011 (fast PWM mode)
CS02:CS00 = 011 (64 pre-scaler)

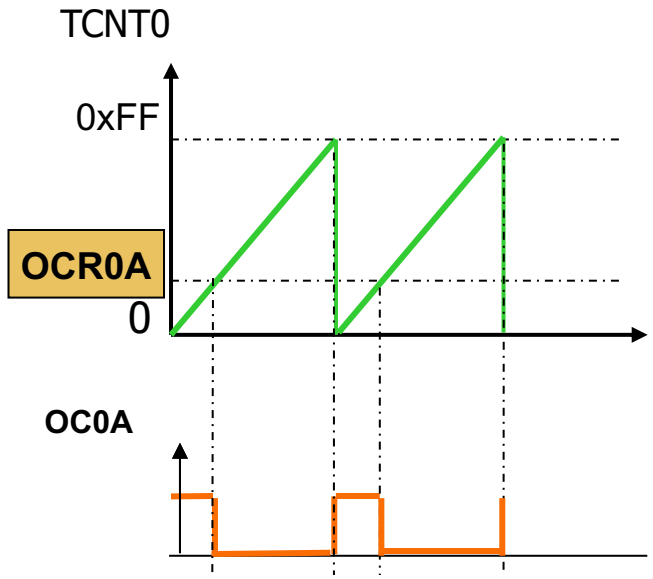
TCCR0A = 10000011=0x83

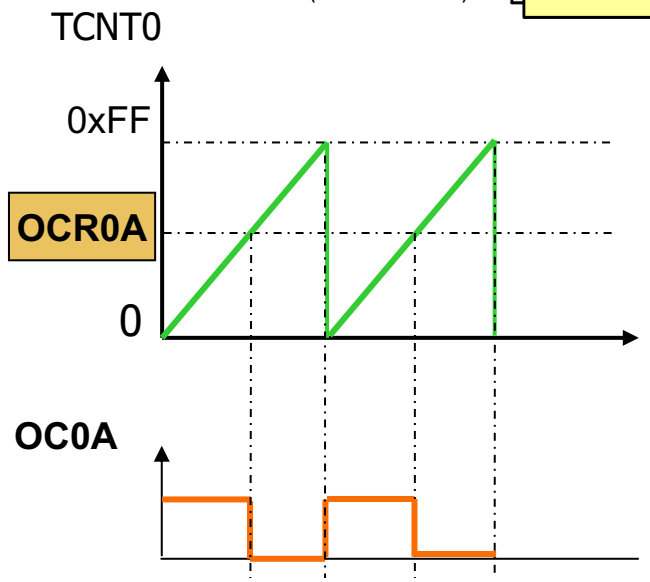
TCCR0B = 00000011 = 0x03

Fast PWM Mode
(Timer0, output OC0A)

- configure timer0 for Fast PWM Mode
- load value into OCR0A (Output Compare Register 0CR0A)
- timer continuously counts from 0x00 to 0xFF
- Output pin OC0A/PD6:
  - set when TCNT0 = 0xFF (top)
  - cleared when TCNT0=OCR0A





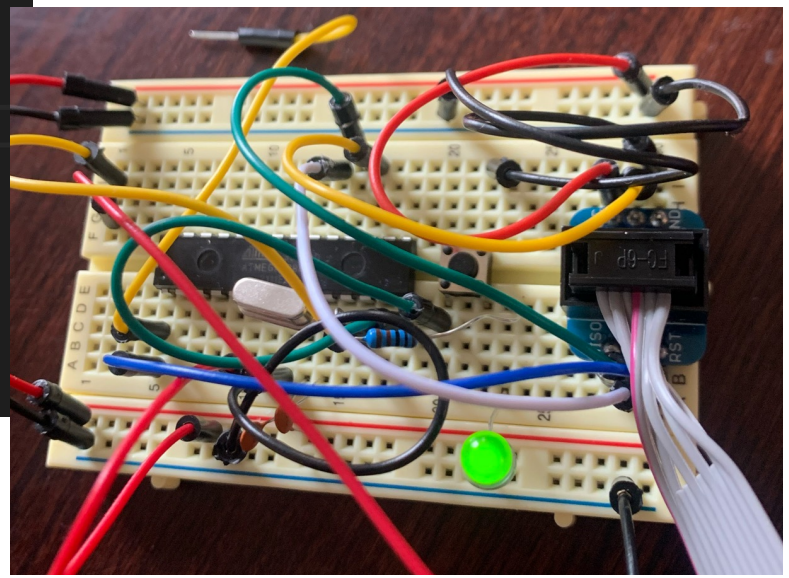


Increase & decrease duty cycle by increasing & decreasing OCR0A
OCR0A=0 → 0% duty cycle; OCR0A = 255 → 0% duty cycle

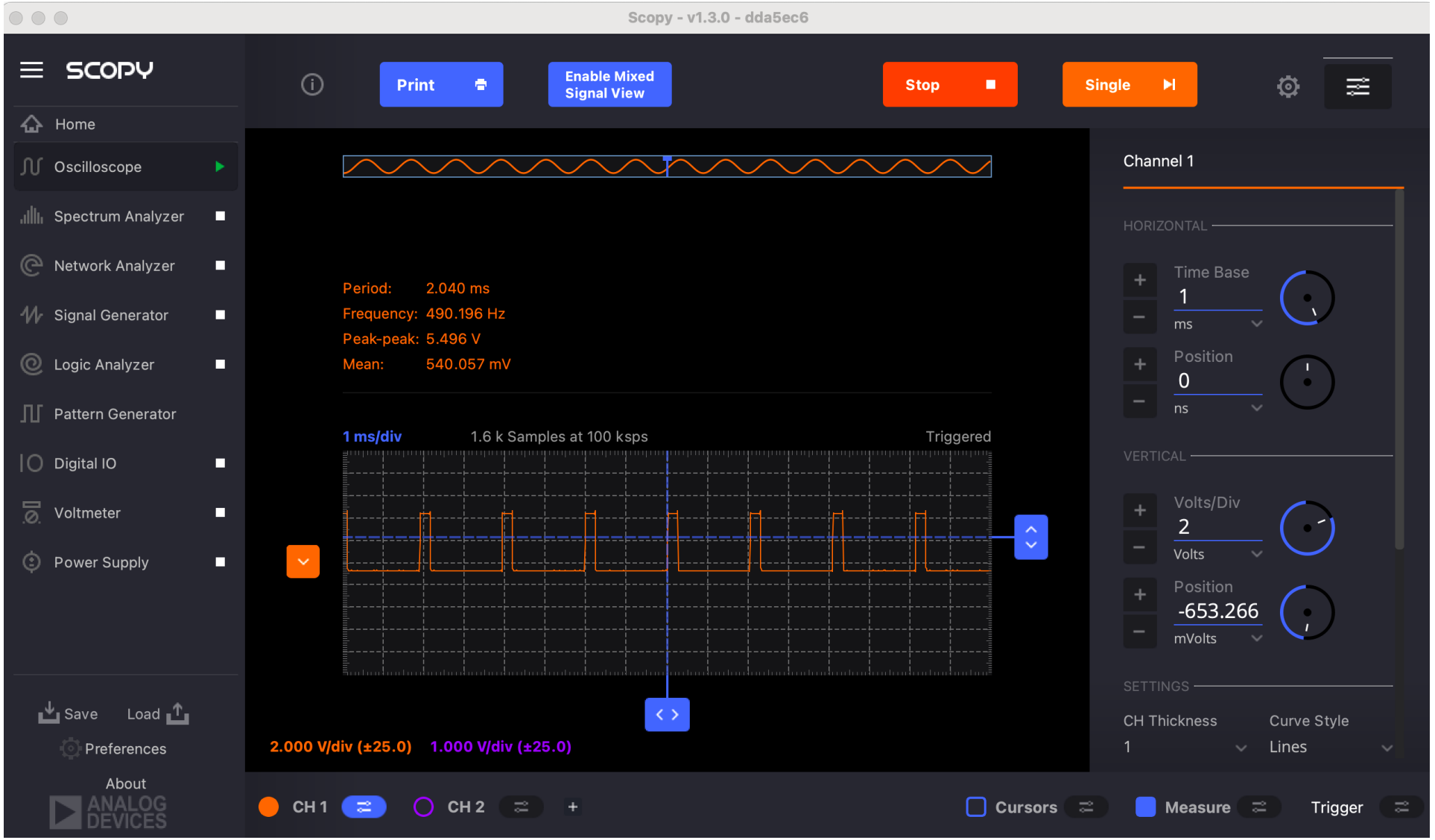
38

stpwm.c > main(void)

```
/* fastpwm.c  Creates 12.5%  duty cycle wafeform
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
}
```



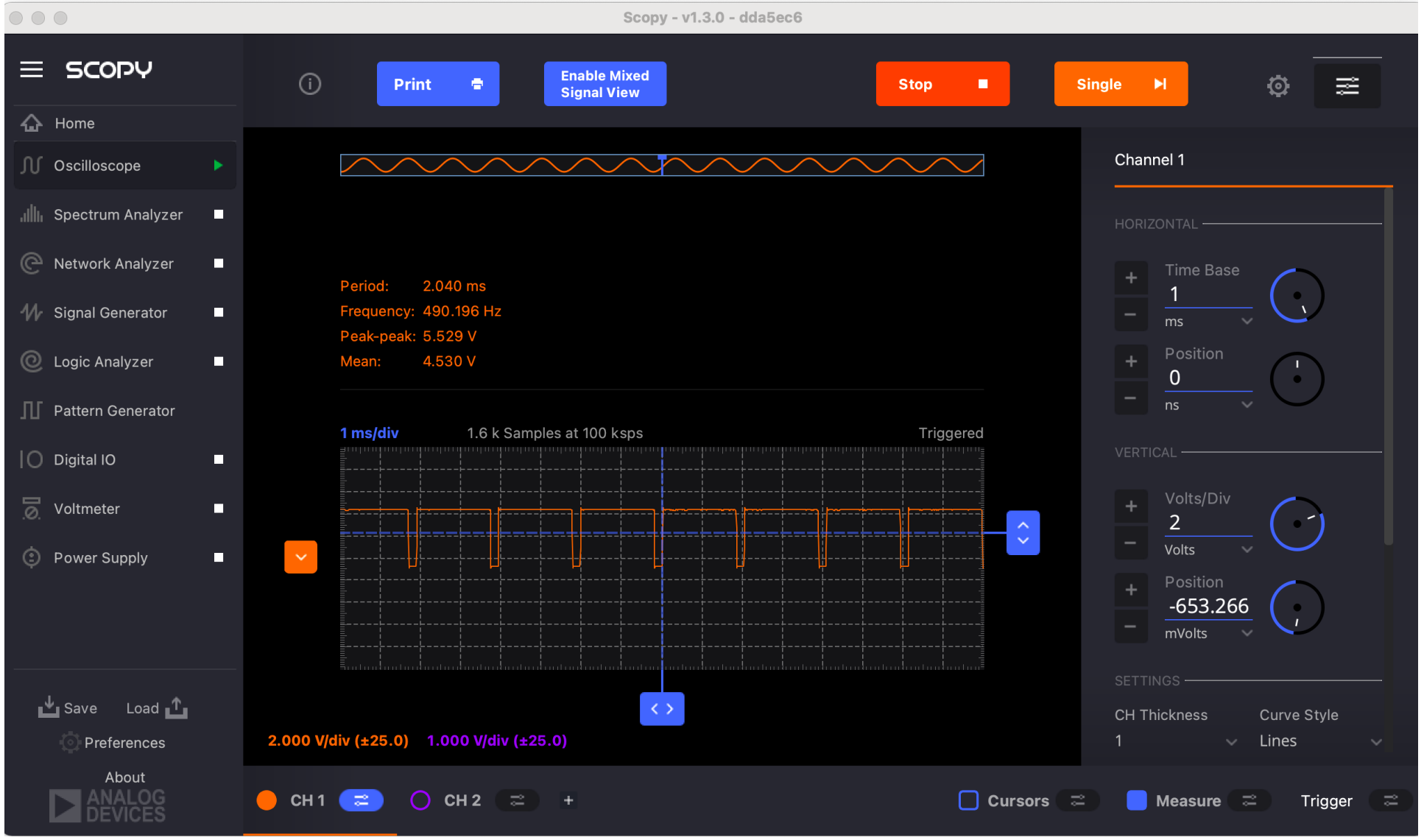
490 Hz pulse train, 12% duty cycle



fastpwm.c > main(void)

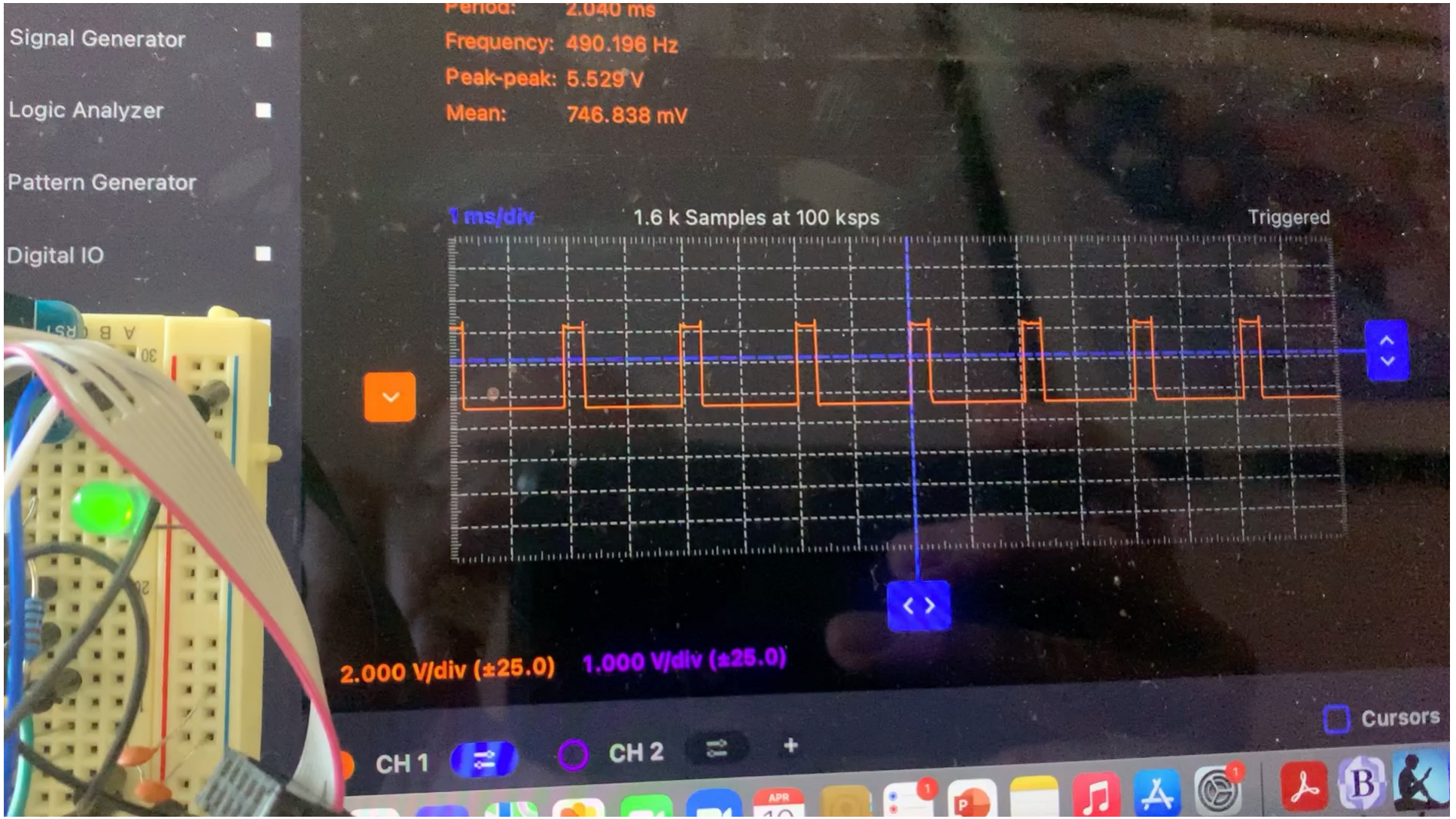
```c
/* fastpwm.c  Creates 12.5%  duty cycle wafeform
on OC0A (PB6). Uses timer0 waveform generator A
with /64 pre-scaler.
Waveform frequency: 16MHz/64/256/2 = 488 Hz
Duty cycle: 90% corresponds to OCR0A = 90% of 256 = 230
D. McLaughlin 4/6/22 for ECE-231 Demo Spring 2022 */

#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 = 230;     // 230 is 90% of 256 so 90% duty cycle

    while(1);           // Just sit there and do nothing
}
```

490 Hz pulse train, 90% duty cycle



C fastpwmramp.c > WAIT

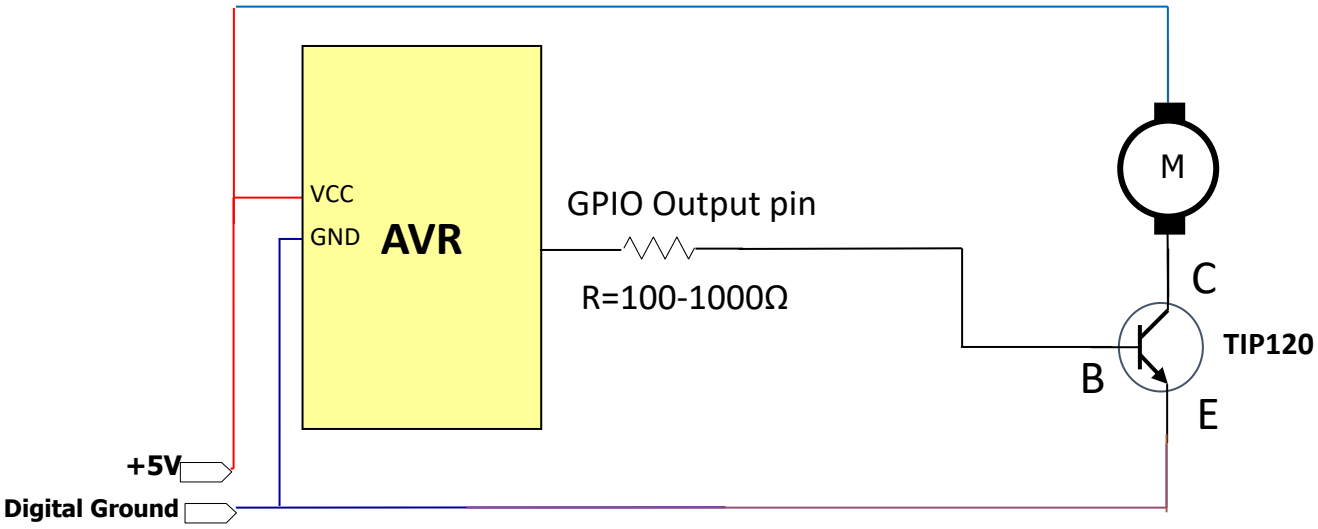
```c
/* fastpwmramp.c  Ramps up duty cycle on OC0A (PB6)
between 0 and 100%. Uses timer0 waveform generator
with /64 pre-scaler
D. McLaughlin 4/6/22 for ECE-231 Demo Spring 2022 */

#include <avr/io.h>
#include <util/delay.h>

#define WAIT 250
int main(void){
    unsigned char i=0;
    DDRD = 1<<DDD6;     // Output must be PD6 for this code

    // Initialize timer0 for FASTPWM, /64 prescaler
    TCCR0A = 0x81;
    TCCR0B = 0x03;


    while(1){
        i=i+10;
        OCR0A=i;        // pulse turns off when TCNT1 = OCR0A
        _delay_ms(WAIT);
    }
}
```

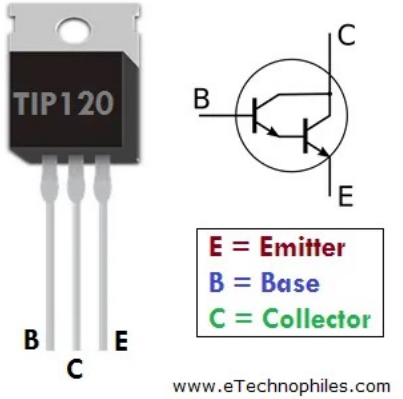

Signal Generator
Logic Analyzer
Pattern Generator
Digital IO
Period: 2.040 ms
Frequency: 490.196 Hz
Peak-peak: 5.529 V
Mean: 746.838 mV
1 ms/div
1.6 k Samples at 100 ksps
Triggered
2.000 V/div (±25.0)
1.000 V/div (±25.0)
Cursors
CH 1
CH 2

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)