Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Section: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Names: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| LABORATORY CHECK OFFS | |
| **Section 0.1 Making an LED blink using a timer**  numberOfLoopsRun with delay() after 10 seconds \_\_\_\_\_\_\_\_\_\_\_\_\_\_  numberOfLoopsRun with Timer after 10 seconds \_\_\_\_\_\_\_\_\_\_\_\_\_\_  Comment on the speed difference, is it significant?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Section 0.2 Creating an analog voltage using PWM from a timer**  Maximum voltage seen on pin 9 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (V)  Minimum voltage seen on pin 9 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (V)  Frequency of PWM signal on pin 9 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Hz)  Show oscilloscope trace of pin 9 showing varying pulse width to you lab instructor \_\_  **Section 0.3 Creating positive and negative analog voltages using two PWM signals**  Maximum positive voltage seen between pin 9 and 10 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (V)  Maximum negative voltage seen between pin 9 and 10 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ (V)  Show oscilloscope traces of pin 9 and 10 showing varying pulse width to you lab instructor \_\_  **Section 0.4 Controlling the angular position of an RC servo motor**  The frequency of the waveform is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Hz)  Show oscilloscope trace of pin 9 showing varying pulse width to your lab instructor \_\_  **Section 1 Viewing the output of Timer 1 using the oscilloscope**  Digital pin that is mostly 90% HIGH? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Digital pin that is mostly 90% LOW? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Digital pin affected by OCR1B? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Pin names for d9 ? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Pin names for d10 ? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Value of OCR1B to give same waveform duty cycle? \_\_\_\_\_\_\_\_\_  **Section 2 Changing the Timer 1 waveform using COM1B0, COM1B1 bits**  Identical waveforms ~10% on duty cycle. No phase offset.  Name of register affected by COM1B1/COM1B0 bits \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  After:  TCCR1A bin:\_\_\_\_\_\_\_\_\_\_\_\_ hex:\_\_\_\_\_\_\_\_\_\_\_\_  TCCR1B bin:\_\_\_\_\_\_\_\_\_\_\_\_ hex:\_\_\_\_\_\_\_\_\_\_  **Section 3 Changing the frequency of Timer 1 waveform using ICR1 register**  ICR1 for 10 Hz = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Section 4 Changing the frequency of Timer 1 waveform using clock prescaler**  Name of register affected by CS12 bits?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  ICR1 \_\_\_\_\_\_\_\_\_\_\_\_\_ TCCR1A\_\_\_\_\_\_\_\_\_\_\_\_\_ TCCR1B\_\_\_\_\_\_\_\_\_\_\_\_\_  **Section 5 Setting PWM frequency to 20 kHz**  ICR1\_\_\_\_\_\_\_\_\_\_\_ OCR1A\_\_\_\_\_\_\_\_\_\_ OCR1B \_\_\_\_\_\_\_\_\_\_\_\_  TCCR1A\_\_\_\_\_0xF0\_\_\_\_\_\_\_ TCCR1B\_\_0x11\_\_\_\_\_\_\_\_\_\_\_  **Section 6 Blinking the LED pin 13 using a hardware timer**  LED 13 is blinking on for 0.5 sec, off for 0.5 sec using Timer1 | \_\_\_\_  \_\_\_\_  \_\_\_\_  \_\_\_\_  \_\_\_\_  \_\_\_\_  \_\_\_\_  \_\_\_\_  \_\_\_\_  \_\_\_\_ |
| Points | \_\_\_\_ |

Prelab:

Read the AVR Atmel 328 microcontroller datasheet sections on Timer 1 and on the clock prescaler, pages 112 -141. The pages numbers refer to Revision G of the datasheet dated Feb 2013 as posted on mycourses. “8271G–AVR–02/2013”\

Overview: What Is a Timer/Counter?

A timer/counter is a special-purpose hardware device, built into a microcontroller, that counts events. Events can be something like a clock cycle. By counting clock cycles, your program can precisely measure elapsed time. By using a timer/counter, instead of a software loop, your main loop of code is left free to perform other tasks.

In many applications, the microcontroller must control actions that are accurate to less than a microsecond. If the main processor of the microcontroller is used to keep time, then there will be almost no processor time left to execute the control software. Hardware timers solve this problem.

A timer is just a counter, that counts up, down or both ways, and is triggered by a consistent pulse waveform, e.g. the system clock. Counting to a specific counter value will take a predictable amount of time. This is the relationship between the counter value and time.

A hardware timer can be used to generate pulse width modulated signals for controlling a motor without consuming software resources. The hardware timer is an independent unit that just needs to be configured to do its task using registers. It has minimal interaction with the main software loop.

Read this overview of the ATMEGA328 Timers <https://www.arduino.cc/en/Tutorial/SecretsOfArduinoPWM>

Reasons to Use Timers:

1) Use a timer when you want to create specific frequencies for driving external circuits e.g. blinking an LED or generating a tone for a speaker, especially when you want to keep the main processor loop free to execute other software instructions.

2) Use a timer when you want to create an analog voltage. To do this, generate a pulse-width-modulated (PWM) digital signal and apply it to an output pin. Changing the pulse width creates different analog voltages. Wider pulses produce higher average voltages. Narrow pulses produce lower average voltages. Arduino’s analogWrite() function uses this form of pulse width modulation.

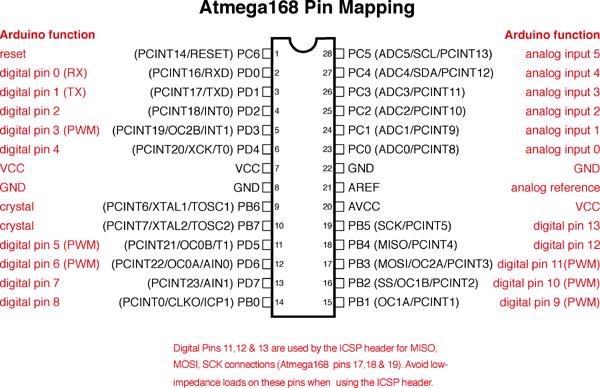
3) Use a timer when you want to drive a motor in forward and backward directions. Create two different PWM signals on different two pins, which will allow you to produce positive and negative voltages. For more power, it possible to attach an H-bridge motor driver that can provide several amps of current to a motor. This will be done in later labs with the robot platform.

4) Use a timer when you want to control the angular position of an RC-type servo. Generate a PWM signal where the pulse width conveys what angle you want to servo to go to.

Learning Outcomes

By the end of this lab you will be able to:

1. Configure hardware timers in the microcontroller to create pulse width modulated signals.
2. Check what values are in a register (8 bit and 16 bit registers) by printing the register values in binary, hex, and decimal format.
3. Configure a hardware timer in the microcontroller by setting register values.
4. Use existing #defined constants such as COM1B1, COM1B0 to make it easier to read the code that sets the register values.
5. Use the LED on pin 13 to display the digital values from other pins (e.g. pin 9) by using a jumper wire.
6. Explain when and why it is beneficial to use timers instead of delays.
7. Explain when and why it is beneficial to use pulse width modulation.



**What is a timer?** Source: <https://arduino-info.wikispaces.com/Timers-Arduino>  
  
A timer or to be more precise a timer / counter is a piece of hardware built into the Arduino controller (other controllers have timer hardware, too). It is like a clock, and can be used to measure time events. The timer can be programmed by some special registers. You can configure the prescaler for the timer, or the mode of operation and many other things.  
  
The controller of the Arduino is the Atmel AVR ATMega 168 or ATmega328. These chips are pin compatible and only differ in the size of internal memory. Both have 3 timers, called timer0, timer1 and timer2. Timer0 and timer2 are 8bit timers, where timer1 is a 16bit timer. The most important difference between 8bit and 16bit timer is the timer resolution. 8bits means 256 values where 16bit means 65536 values for higher resolution or longer count.  
  
The controller for the Arduino Mega series is the Atmel AVR ATmega1280 or the ATmega2560. Again, identical but differs in memory size. These controllers have 6 timers. Timer 0, timer1 and timer2 are identical to the ATmega168/328. The timer3, timer4 and timer5 are all 16bit timers, similar to timer1.  
  
All timers depend on the system clock of your Arduino system. Normally the system clock is 16MHz, but for the Arduino Pro 3.3V it is 8Mhz. So be careful when writing your own timer functions.  
  
The timer hardware can be configured with some special timer registers. In the Arduino firmware all timers were configured to a 1kHz frequency and interrupts are generally enabled.

**Timer0: 8bit timer.**

In the Arduino world timer0 is been used for the software Sketch timer functions, like [\_\_delay()\_\_](http://arduino.cc/en/Reference/Delay),[\_\_millis()\_\_](http://arduino.cc/en/Reference/Millis) and[\_\_micros()\_\_](http://arduino.cc/en/Reference/Micros). If you change timer0 registers, this may influence the Arduino timer function. So you should know what you are doing.

**Timer1: 16bit timer.**

In the Arduino world the [\_\_Servo library\_\_](http://arduino.cc/en/Reference/Servo) uses timer1 on Arduino Uno (timer5 on Arduino Mega).

**Timer2: 8bit timer like Timer0.**

In the Arduino world the [\_\_tone()\_\_](http://arduino.cc/en/Reference/Tone) function uses timer2.

How to Succeed With This Lab:

Read the sections of the datasheet. Explore changing a single value in a register and observe what changes in the behavior of the signal. Print the value of each register to see if it has the expected value before it is changed and after it is changed. Use the oscilloscope to measure the pins directly and see what waveform is being produced.

HINT: For the your calculations in this lab, it is helpful to know that the Arduino system clock runs at 16 MHz.

**SECTION 0.1 – Making an LED blink using a timer**

In this section, you will be comparing the performance of the main loop code when blinking an LED using a Timer instead of a Delay() statement

**Procedure:**

1) Run the code in the textbox below. After about 10 seconds, record the number of times the loop has run and put this value on the signoff sheet.

// Lab4\_BlinkUsingDelay

// Written By: C. Hochgraf

// Date: Feb 2017

unsigned long numberOfLoopsRun=0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void setup() {

pinMode(13,OUTPUT);

Serial.begin(9600);

Serial.println(F("Lab 4: BlinkUsingDelay"));

Serial.println(F("I'm ready to blink now."));

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void loop() {

digitalWrite(13,HIGH);

delay(1000);

digitalWrite(13,LOW);

delay(1000);

Serial.print(F("The number of times loop code has run is: "));

Serial.println(numberOfLoopsRun);

numberOfLoopsRun++;

} // Arduino loop()

2) Now run the code below that blinks the LED but uses Timer to toggle pin 9 once every second. There is not an LED on pin 9 but if you connect a jumper wire from pin 9 to pin 13, the LED on pin 13 will flash on and off. After about 10 seconds of the program running, record the number of times the loop has run and put this value on the signoff sheet.

// Lab4\_BlinkUsingTimer1

// Written By: C. Hochgraf

// Date: Feb 2017

unsigned long numberOfLoopsRun=0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void setup()

{

pinMode(9,OUTPUT); // to drive led on pin 13

pinMode(13,INPUT); // to receive digital signal from pin 9

Serial.begin(9600);

Serial.println(F("Lab 4: BlinkUsingTimer1"));

TCCR1A=0x50; // Configures timer to CTC mode 4 with OCR1A as top

TCCR1B=0x0D; // toggles pin 9 every one second

OCR1A=0x3D09; // one second interval

Serial.println("I'm ready to blink now. Attach a wire from pin 9 to 13");

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void loop()

{

// every time numberOfLoopsRun is a multiple of 10000, print the number of loops run so far

if ((numberOfLoopsRun % 10000)==0) {

Serial.print(F("The number of times loop code has run is: "));

Serial.println(numberOfLoopsRun);

}

numberOfLoopsRun++;

} // Arduino loop()

**SECTION 0.2 – Creating an analog voltage using PWM from a timer**

In this section, you will create a slowly varying analog voltage and use that voltage to dim an LED. You will also measure the voltage on the PWM pin using both the oscilloscope and a DC voltmeter. The code generates a PWM signal of varying pulse width on pin 9. The PWM signal is used to slowly dim an LED. In this case, the built-in LED on pin 13 is used.

**Procedure:**

1) Run the code in the textbox below. Connect a jumper wire from pin 9 to pin 13, so that the LED on pin 13 can be used to show the varying voltage on PWM pin 9. Pin 13 is configured as an input so that it has no effect on the PWM signal that is coming from pin 9, but the pin 13 can still be driven by pin 9’s signal.

2) When the code runs, you should see the pin 13 led get slowly brighter.

3) Connect a DC voltmeter from pin 9 to **ground**. Record the maximum DC voltage seen on pin 9 and the minium voltage seen on pin 9.

4) Connect the oscilloscope probe to pin 9 and display the pulse width modulation signal. Record the **frequency** of the PWM signal on the sign off sheet.

// Lab4 PWM\_dim\_LED\_compact.ino

// Written By: C. Hochgraf

// Date: Feb 2017

int sreg;

unsigned long fadeValue=0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void setup()

{

pinMode(9,OUTPUT); // to drive led on pin 13

pinMode(13,INPUT); // to receive digital signal from pin 9

Serial.begin(9600);

Serial.println(F("Lab 4: PWM\_dim\_LED\_compact"));

TCCR1A=0xA3; // configure registers for Timer 1 Mode 7 fast PWM 10bit, clock scale=8x

TCCR1B=0x0A; // configure registers for Timer 1 Mode 7 fast PWM 10bit, clock scale=8x

Serial.println();

Serial.println("I'm ready to blink now. Attach a wire from pin 9 to 13");

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void loop()

{ delay(10);

fadeValue++;

OCR1A=fadeValue; // load PWM compare register (OCR1A) with new PWM value (fadeValue)

//OCR1A=512; // generates fixed PWM "analog voltage" of around 2.5 volts dc

if (fadeValue>=1023) fadeValue=0; // PWM counts up to 10 bit value (1023)

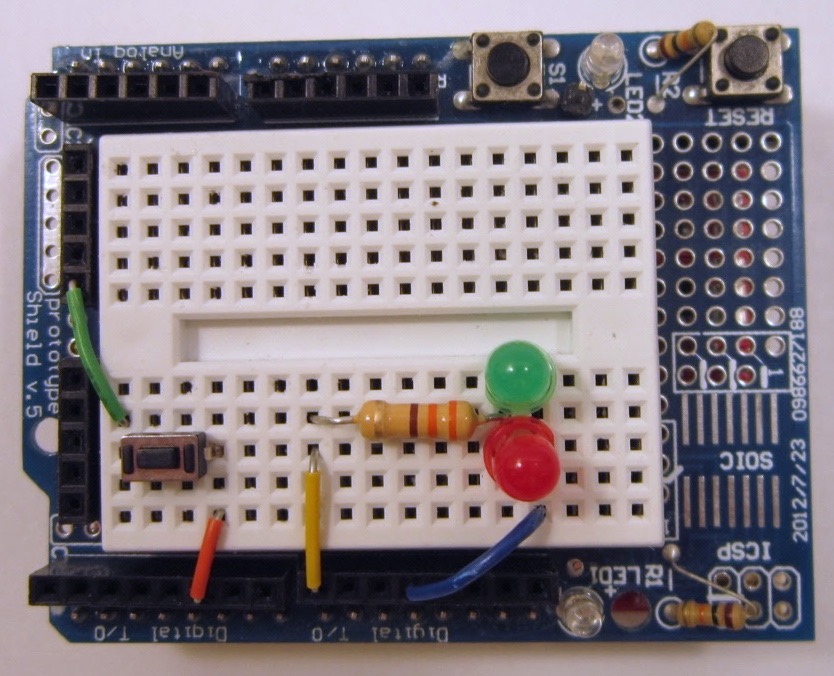
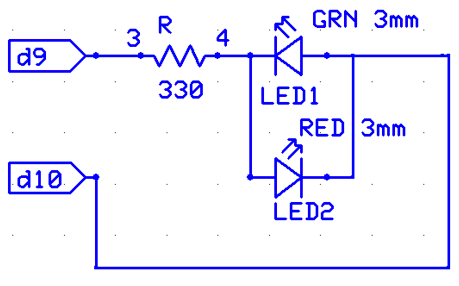
} // Arduino loop()

**SECTION 0.3 – Creating positive and negative analog voltages using two PWM signals**

In this section, you will use Timer 1 to create two different PWM signals that can be used when you want to drive a motor in the forward and backward directions. The two PWM signals will allow you to produce positive and negative voltages. For more power, an H-bridge motor driver can be attached, allowing you to provide several amps of current to a motor.

**Procedure:**

1) First, connect a 330 ohm resistor and two back to back LEDs (one red, one green) between pin 9 and pin 10. This will look very much like the setup used in lab 2 state machines for red/green blinking except that the LEDs will be connected to pins 9&10 instead of pins 8&11.

2) Remove the jumper wire to pin 13. The pin 13 LED will not be used.

3) Copy and run the code from the textbox below. When the code runs, the LED brightness should cross fade slowly from bright green to bright red.

4) Connect a DC voltmeter with the positive lead attached to pin 9 and the negative lead attached to pin 10, **not to ground.** Record the maximum voltage seen on the voltage meter and the minimum voltage on the voltmeter on the signoff sheet.

5) Connect the oscilloscope channel 1 to pin 9 and channel 2 to pin 10. Observe the PWM waveforms and show them to your lab instructor for a signoff.

// Lab4 PWM\_motor\_pos\_neg\_compact.ino

// Written By: C. Hochgraf

// Date: Feb 2017

unsigned long fadeValue=0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void setup()

{

pinMode(9,OUTPUT); // motor lead +

pinMode(10,OUTPUT); // motor lead -

Serial.begin(9600);

Serial.println(F("Lab 4: PWM\_motor\_pos\_neg\_compact"));

TCCR1A=0xB3; // configure registers for Timer 1 Mode 7 fast PWM 10bit, clock scale=8x

TCCR1B=0x0A; // configure registers for Timer 1 Mode 7 fast PWM 10bit, clock scale=8x

Serial.println();

Serial.println("I'm ready to blink now. Attach 330 ohm resistor and back to back LEDs");

Serial.println("(just like in lab 2) between pins 9 and 10");

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void loop()

{ delay(20);

fadeValue++;

OCR1A=fadeValue; // load PWM compare register (OCR1A) with new PWM value (fadeValue)

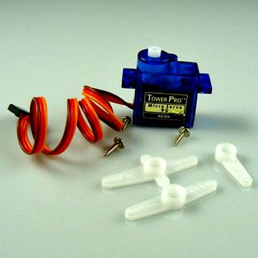
OCR1B=fadeValue;

if (fadeValue>=1023) fadeValue=0; // PWM counts up to 10 bit value (1023)

} // Arduino loop()

**SECTION 0.4 – Controlling the angular position of an RC servo motor**

In this section, you will use Timer 1 to create a PWM signal that controls the angular position of an RC-type servo.



**Procedure:**

1) Disconnect the red and green LEDs from the previous section.

2) Connect an RC servo (blue motor in your kit) to the Arduino pin 9. The color code on the servo wires is as follows: Red= +5v, Brown= Ground, Orange=PWM signal from pin 9.

3) Install one of the white “horns” on the servo motor so that you can see the motor turn. You do not need to install the screw to hold the horn on at this time.

4) Copy and run the code from the textbox below. When the code runs, the arm on the servo should move slowly over a range of 90 or more degrees.

5) Connect the oscilloscope channel 1 to pin 9. Observe the PWM waveform. Notice that the pulse width varies over a small range from 575usec to 2460 usec as the arm moves over an angle of 90 degrees or more. The middle value of the pulse width is 1500 usec. Wider pulses move the arm one direction. Narrower pulses move the arm the other direction. Show the waveform to your lab instructor for a signoff.

// Lab4 PWM\_servo\_compact.ino

// Written By: C. Hochgraf

// Date: Feb 2017

unsigned long servoValue=0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void setup()

{

pinMode(9,OUTPUT); // to drive servo on pin 9

Serial.begin(9600);

Serial.println(F("Lab 4: PWM\_servo\_compact"));

TCCR1A=0xB0;

TCCR1B=0x13;

ICR1=0x9C4;

Serial.println();

Serial.println("I'm ready to run a servo, attach one to pin 9");

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void loop()

{ delay(100);

servoValue++;

OCR1A=72+servoValue; // load PWM compare register (OCR1A) with new PWM value (fadeValue)

if (servoValue>=235) servoValue=0; //

// for HS-311 servo, min time= 0.575 ms to max time 2.460ms

// 250 counts per ms, but divide by 2 due to phase correct = 125 counts per ms

} // Arduino loop()

**SECTION 1 – Viewing the output of Timer 1 using the oscilloscope**

**Procedure:**

1. Load and run the code Lab4\_Timer1\_p1.
2. Connect two oscilloscope probes, one on pin d10, one on pin d9. Set the vertical scale to around 2V/div and the horizontal scale to around 20 msec/div. Vertically offset channel 1 from channel 2 so that you can clearly see the two signals and they are non-overlapping. Your signals should look roughly like the ones in the figure below.
3. From the oscilloscope display, which digital pin has the signal that is mostly high, but dips low ~10% of the time? \_\_\_\_\_\_\_\_ Which pin is mostly low, but goes high ~10% of the time?\_\_\_\_\_\_\_\_\_
4. Change the value of the register OCR1B, compile and upload the code. Observe how changing the OCR1B value changes waveforms. Are both waveforms affected by the value in OCR1B or just one of the waveforms? Which digital pin’s waveform is affected by changing OCR1B?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
5. Refer back to the figure that shows the ATMEGA 168 pin mapping for the Arduino. What are all the different pin names that go with digital pin 9? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ What are all the different pin names that go with digital pin 10?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
6. Change just the value of OCR1B until both waveforms have about the same percent of time that that they are HIGH. What value of OCR1B gives it the same % duty cycle as the other waveform? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. Notice how when the waveforms have the same duty cycle (~10% HIGH, ~90% LOW) there is still a phase difference between the two waveforms. This is in part due to the way the reference values in OCR1A and OCR1B are being compared to a triangle wave that counts up to TOP and counts back down to BOTTOM. It may help to look at the waveforms in the datasheet showing counting up and counting down in phase correct and frequency correct mode.

**SECTION 2 – Changing the Timer 1 waveform using COM1B0, COM1B1 bits.**

1. Set the value of OCR1B back to 625 so that you have the original waveforms from the very start of the lab.
2. It is possible to make the waveforms on pin d10 and d9 look identical just by changing how the output pins are configured using the Compare Output Mode bits. You will need to read the datasheet section to understand how these bits are used.



1. Changing only the COM1Bx bits, make the output waveforms on d10 and d9 identical. OCR1B must remain unchanged at 625. Show your waveforms to your instructor for a signoff. Why did this do what it did? Does it make sense?
2. When you changed the values of the COM1B0 and/or COM1B1 bits, which **register (name)** was affected? \_\_\_\_\_\_\_\_\_\_\_\_
3. Print the binary and hex value of TCCR1A and TCCR1B before and after the changes to COM1Bx bits.
   1. Before
      1. TCCR1A bin:\_\_\_\_\_\_\_\_\_\_\_\_\_\_ hex:\_\_\_\_\_\_\_\_\_\_\_\_
      2. TCCR1B bin:\_\_\_\_\_\_\_\_\_\_\_\_\_\_ hex:\_\_\_\_\_\_\_\_\_\_\_\_
   2. After
      1. TCCR1A bin:\_\_\_\_\_\_\_\_\_\_\_\_\_\_ hex:\_\_\_\_\_\_\_\_\_\_\_\_
      2. TCCR1B bin:\_\_\_\_\_\_\_\_\_\_\_\_\_\_ hex:\_\_\_\_\_\_\_\_\_\_\_\_

Does the change in the register values make sense?

**SECTION 3 – Changing the frequency of Timer 1 waveform using ICR1 register.**

1. Use cursors on the scope to measure the frequency of the signal on pin 10. The period should be about 80 milliseconds for a frequency of 12.5 Hz. By changing only the value of ICR1, modify the frequency to be 10 Hz, (i.e. a period of 100 milliseconds.) What value of ICR1 do you need to use? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Don’t worry about what the duty cycle of the waveform is (i.e. don’t worry about what % of time the signal is HIGH versus LOW).



**SECTION 4 – Changing the frequency of Timer 1 waveform using clock prescaler**

1. Configure Timer 1 so that the signal on d10 has a frequency of 1 Hz (a period of 1 second). Try changing the value in the register ICR1 to accomplish this. Are you able to get this to work? You may need to change the clock prescaler as well. At this point, you want to be able calculate out what the register values for ICR1 and the bit values for the clock prescaler should be. To understand how this works, refer to the datasheet. One table that might be helpful is the clock select bit description table given below.
2. You also might find it helpful to uncomment the line of code in your program that prints the WGM, COM, and CS1 values

…

Serial.println(F("you can comment out the printing of WGM values to save time"));

//printWGMCOMCS1Values();

…



1. What register is affected when you change the clock select bits CS12, CS11, CS10? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. Change the values ICR1 and the clock prescaler bits, to get a signal with a frequency of 1 Hz (period of 1 second). What values do you have for each of these registers?
   1. ICR1 = \_\_\_\_\_\_\_\_\_\_\_\_ TCCR1A= \_\_\_\_\_\_\_\_\_\_\_\_ TCCR1B = \_\_\_\_\_\_\_\_\_\_\_\_\_

**SECTION 5 – Setting PWM frequency to 20 kHz**

Pulse Width Modulation (PWM) is widely used to control the speed and direction of DC motors. The pulse width modulation waveforms can be applied directly to the motor, for example using an H-bridge motor drive with high power transistors. Often, it is desirable to perform the pulse width modulation at a frequency that is so high that humans can not hear it (e.g. freq> 20kHz). If the PWM frequency is low (e.g. 1kHz) then the motor driver can produce an objectionable whining noise.

**Procedure:**

1. Configure Timer 1 so that the frequency is 20kHz and use a clock prescale value so that ICR1 is >= 100. Use the following settings for the compare output mode COM1A1 = 1, COM1A0 = 1, COM1B1 = 1, COM1B0 = 1. Set the duty cycle on both pins 9 and 10 to be 50% duty cycle. Connect both pin 9 and pin 10 to the oscilloscope so that you can see both waveforms at the same time.
2. Fill in the values you used:
   1. ICR1 = \_\_\_\_\_\_\_\_\_\_\_\_ OCR1A= \_\_\_\_\_\_\_\_\_\_\_\_ OCR1B = \_\_\_\_\_\_\_\_\_\_\_\_\_
   2. TCCR1A= \_\_\_\_\_\_\_\_\_\_\_\_ TCCR1B = \_\_\_\_\_\_\_\_\_\_\_\_\_
3. Try setting OCR1A = 300 and OCR1B =100 and observe the PWM waveforms on the scope. Try setting OCR1A = 100 and OCR1B =300 and notice how the waveforms change. Place a dc voltmeter between pins 9 and 10 and notice how the dc voltage reading changes between the case where OCR1A = 300, OCR1B =100 and where OCR1A = 100, OCR1B =300. In one case you should see a positive voltage reading, in the other you will see a negative voltage reading. This type of PWM allows you to smoothly control a DC motor’s speed, allowing you to turn the motor forward and backward by applying positive or negative voltage to the terminals of the motor.

**SECTION 6 – Blinking the LED pin 13 using a hardware timer.**

In the first lab, you ran code called “Blink” that turned the pin 13 LED on and off at a one second interval. That code used a delay() function. In this section, you will use a hardware timer do the same thing, however, the LED is on pin 13 and not on pin 10 or pin 9 where the hardware timer is attached. The good news is that there is a way to light the LED on pin 13 using the digital output on pin 10. You can set pin 13’s pin mode to be an INPUT and then place a jumper wire from digital pin 10 to pin 13. Pin 10 is an output and it can drive the LED which still attached to pin 13, even if pin 13’s pin mode is INPUT.

**Procedure:**

1. Make sure your code configures pin 13’s pin mode to be an INPUT.
2. Connect a jumper wire from pin 10 to pin 13. Depending upon the duty cycle of the signal on pin 10, the LED might be flashing on and off dimly or brightly.
3. Configure the OCR1B register so that the signal on pin 10 has a duty cycle of 50% on, 50% off and period of 1 second. Make any other register adjustments as needed to accomplish this. You should see the LED blinking on for 0.5 sec and off for 0.5 sec.
4. Note that while the LED is blinking, you could run additional code at the same time to other tasks, without having the processor stuck on a delay(500); statement for most of the time.

**Write Up:**

There is no write up for this lab, just get the items on the cover sheet (next page) signed off by your lab instructor and submit just the cover sheet in class.