03 - Register Operation

CEG 4330/6330 - Microprocessor-Based Embedded Systems

Max Gilson

Arduino Bitwise Operators

- & - AND
- | - OR
- ^ - XOR
- ~ - NOT
- << - Shift left
- >> - Shift right
- These are equivalent:
  - Y |= 0x20;
  - Y = Y | 0x20;
  - If Y is 0x82 then the above operation results in Y = xA2
- These are equivalent:
  - Y = 0x20;
  - Y = 1<<5;

Register/Parallel Port Operation

- Y: a generic register which can be registers for ports B/C/D (PORTB, DDRB, PINB, ...) or others (TCCR1B, ...).
- Contents of Y: Y_7 Y_6 Y_5 Y_4 Y_3 Y_2 Y_1 Y_0
- Output (Use Y5 as an example)
  - Set bits : Y |= 0x20; or Y |= 1<<5; or Y = B00100000;
  - Clear bits : Y &= ~0x20; or Y &= 0xDF;
  - Toggle bits : Y ^= 0x20;
- Input
  - Check if set : if(Y & 0x20)
  - Check if clear : if(!(Y & 0x20))
  - Wait until set : while(!(Y & 0x20)) ;
  - Wait until clear: while(Y & 0x20) ;

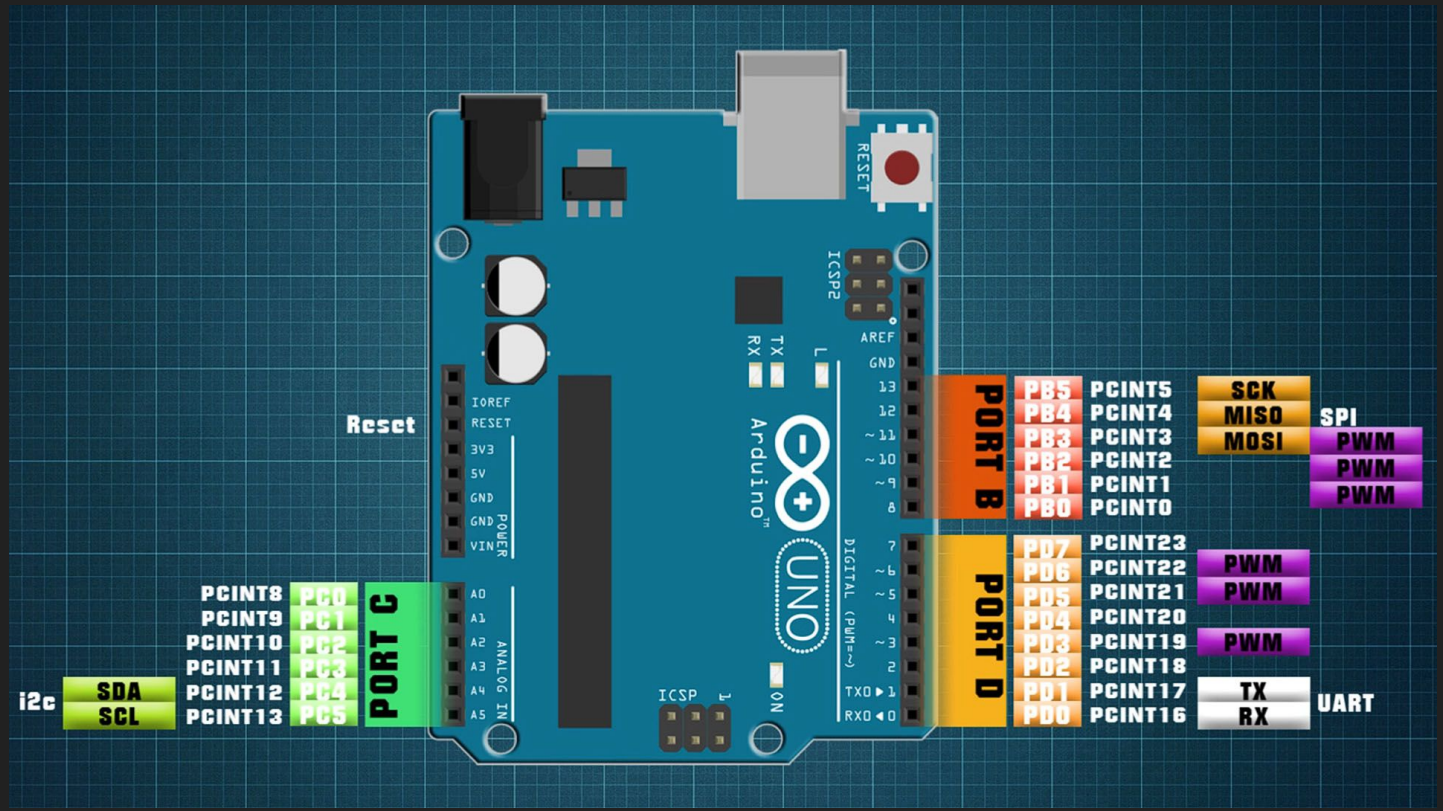
Register List

- DDRD - The Port D Data Direction Register - read/write
- PORTD - The Port D Data Register - read/write
- PIND - The Port D Input Pins Register - read only

- DDRB - The Port B Data Direction Register - read/write
- PORTB - The Port B Data Register - read/write
- PINB - The Port B Input Pins Register - read only

- DDRC - The Port C Data Direction Register - read/write
- PORTC - The Port C Data Register - read/write
- PINC - The Port C Input Pins Register - read only

Register/Port List (cont.)



Register Manipulation

```
// Use   |=   to safely set values to 1
// Use   &=  to safely set values to 0

DDRD |= B00000100;  //Sets only D2 as OUTPUT
PORTD |= B00000100;//Sets only D2 to HIGH

DDRB &= B11100111;  //Sets only D11 and D12 as INPUT
PORTB |= B00011000; //Sets only D11 and D12 as INPUT_PULLUP
PORTB &= B11100111;//Sets only D11 and D12 as INPUT (no pullup)
```

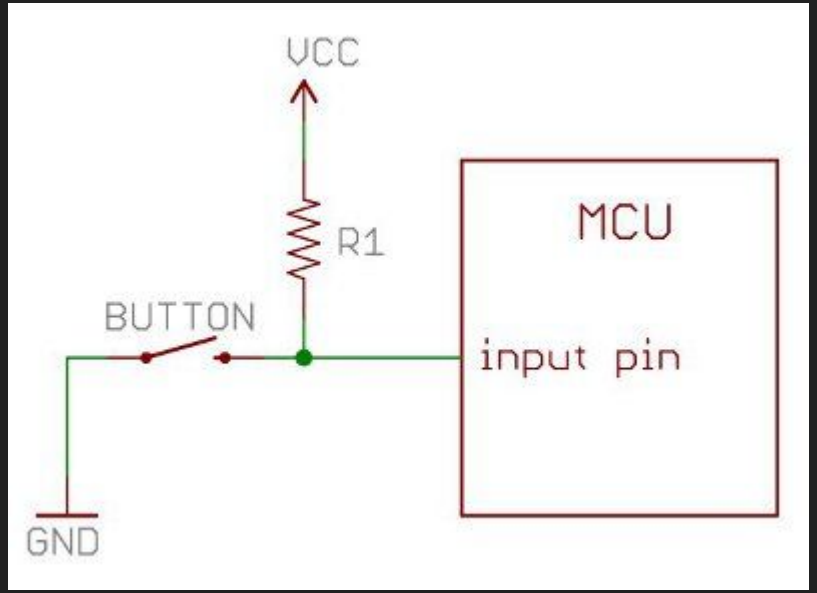
Reading Input Registers

```
// If D5 is HIGH:execute the code inside if statement
if(PIND & B00100000)
{
    //Add your code...
}

// If D5 is HIGH:value = 32
// If D5 is LOW: value = 0
int value = PIND & B00100000;

// If D5 is HIGH:value = 1
// If D5 is LOW: value = 0
// This is equivalent to digitalRead(5)
value = PIND >> 5 & B00100000 >> 5;
```

Registers/Ports are Fast

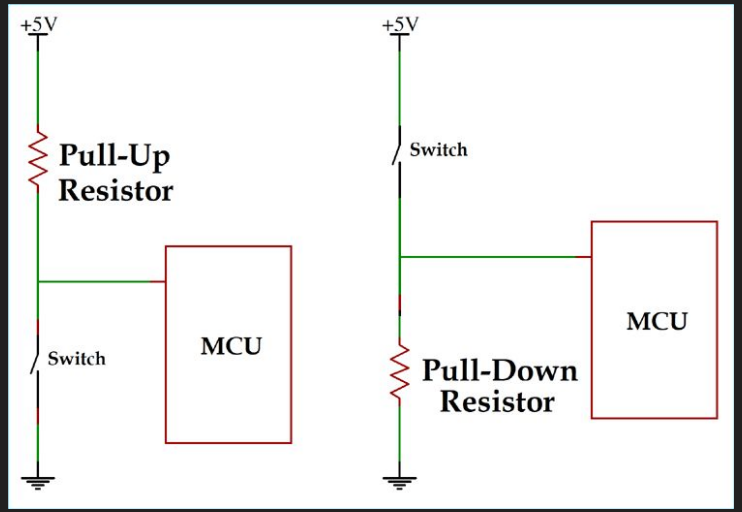
Pull Up and Pull Down Resistors

- Never leave a voltage floating
  - A floating voltage is undefined, it is not high, it is not low, it's something else (unknown)
  - In some cases devices will have internal pull up and pull down resistors to avoid this issue
- A pull up resistor works by assuring a voltage is never floating
  - When the switch is closed, the voltage is 0V, when the switch is open, the voltage is VCC



Pull Up and Pull Down Resistors (cont.)

- Sometimes, you need a pull down resistor (not as common)
  - This resistor pulls the voltage to ground when the switch is not active
- To choose a pull up/down resistor calculate the resistor using the desired current draw
- Typical values should be between 1KΩ - 10KΩ
- If the resistance is too low, it will draw a lot of power (wasteful)
- If the resistance is too high, you have a weak connection to ground or VCC (not much better than floating)



$$R1 = \frac{V_{CC}}{current\ through\ R1} = \frac{5\text{V}}{0.001\text{A}} = 5\text{k}\,Ohms$$

Pull Up and Pull Down Resistors (cont.)

- Some devices will give you the option of enabling or disabling internal pull up and pull down resistors
- This can be helpful if you don’t want to use extra components outside of the microcontroller

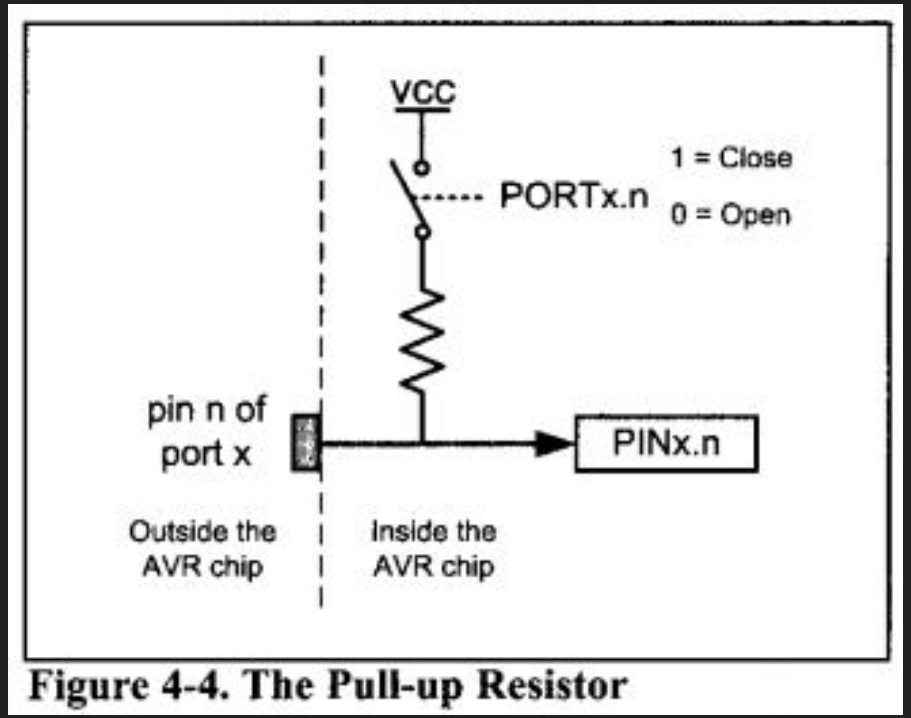
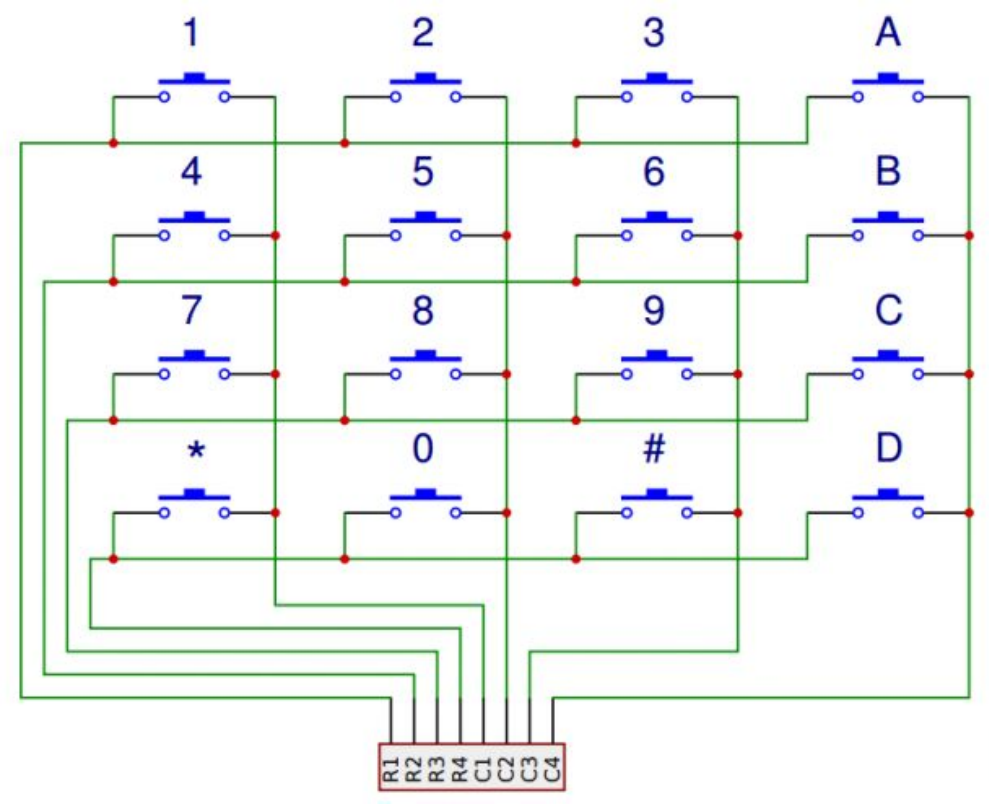


Figure 4-4. The Pull-up Resistor

Keypad Scanning



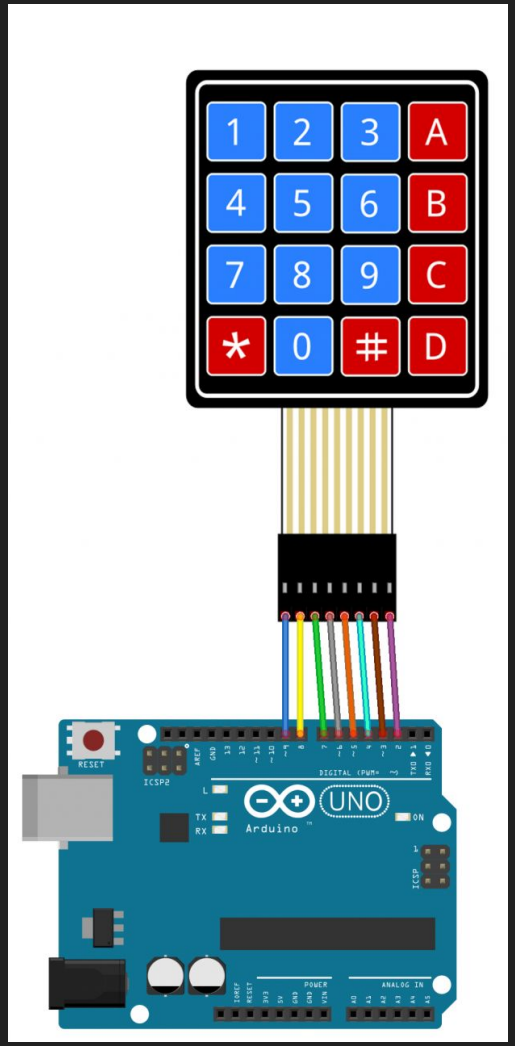
Keypad Scan Sample Code

```
int rows = 4;
int cols = 4;

int rowPins[4] = {9,8,7,6};
int colPins[4] = {5,4,3,2};

char keys[4][4] = {     {'1', '2', '3', 'A'},
                        {'4', '5', '6', 'B'},
                        {'7', '8', '9', 'C'},
                        {'*', '0', '#', 'D'}};

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  scanKeys();
  delay(5);
}
```

```
void scanKeys()
{
  // Set all rows to inputs with a pull up resistor (all row pins become HIGH)
  for (int r = 0; r < rows; r++)
  {
            pinMode(rowPins[r], INPUT_PULLUP);
  }

  // Send a pulse through each column and check all rows for the pulse
  for (int c = 0; c < cols; c++)
  {
    pinMode(colPins[c], OUTPUT);
    digitalWrite(colPins[c], LOW);          // Begin column pulse output.
    // Check each row to see if they are LOW
    for (int r = 0; r < rows; r++)
    {
      if(!digitalRead(rowPins[r]))
      {
        Serial.print(keys[r][c]);
        delay(200);
      }
    }

    // Set pin to high impedance input. Effectively ends column pulse.
    digitalWrite(colPins[c], HIGH);
    pinMode(colPins[c], INPUT);
  }
}
```

Keypad Scan Sample Schematic

- Column Pins:
  - 5, 4, 3, 2
- Row Pins:
  - 9, 8, 7, 6



Timer

- Binary counter: Clock source, pre-scaler, overflow, TOP/BOTTOM
- Input capture
  - Control options, set/clear Flag bit (see TIFR1)
  - Digital pin 8; Pulse width measurement
- Output compare
  - Control options, Flag bit (TIFR0, 1, 2)
  - Delay loop, output waveform generation
  - analogWrite( ): Timer 0 A/B: pins 6/5, fast PWM;
  - Timer 1: pins 9/10, 8-bit phase correct PWM;
  - Timer 2: pins 11/3, phase correct PWM

Timer Specs

- Timer 0
  - 8 bit counter (Range: 0 - 255)
  - Compare match
  - Overflow
- Timer 1
  - 16 bit counter (Range: 0 - 65546)
  - Compare match
  - Overflow
  - Input Capture
- Timer 2
  - 8 bit counter (Range: 0 - 255)
  - Compare match
  - Overflow

Timer Register List

- TCCRx - Timer/Counter Control Register. The prescaler can be configured here.
- TCNTx - Timer/Counter Register. The actual timer value is stored here.
- OCRx - Output Compare Register
- ICRx - Input Capture Register (only for 16bit timer)
- TIMSKx - Timer/Counter Interrupt Mask Register. To enable/disable timer interrupts.
- TIFRx - Timer/Counter Interrupt Flag Register. Indicates a pending timer interrupt.

500ms Blink Sample Code

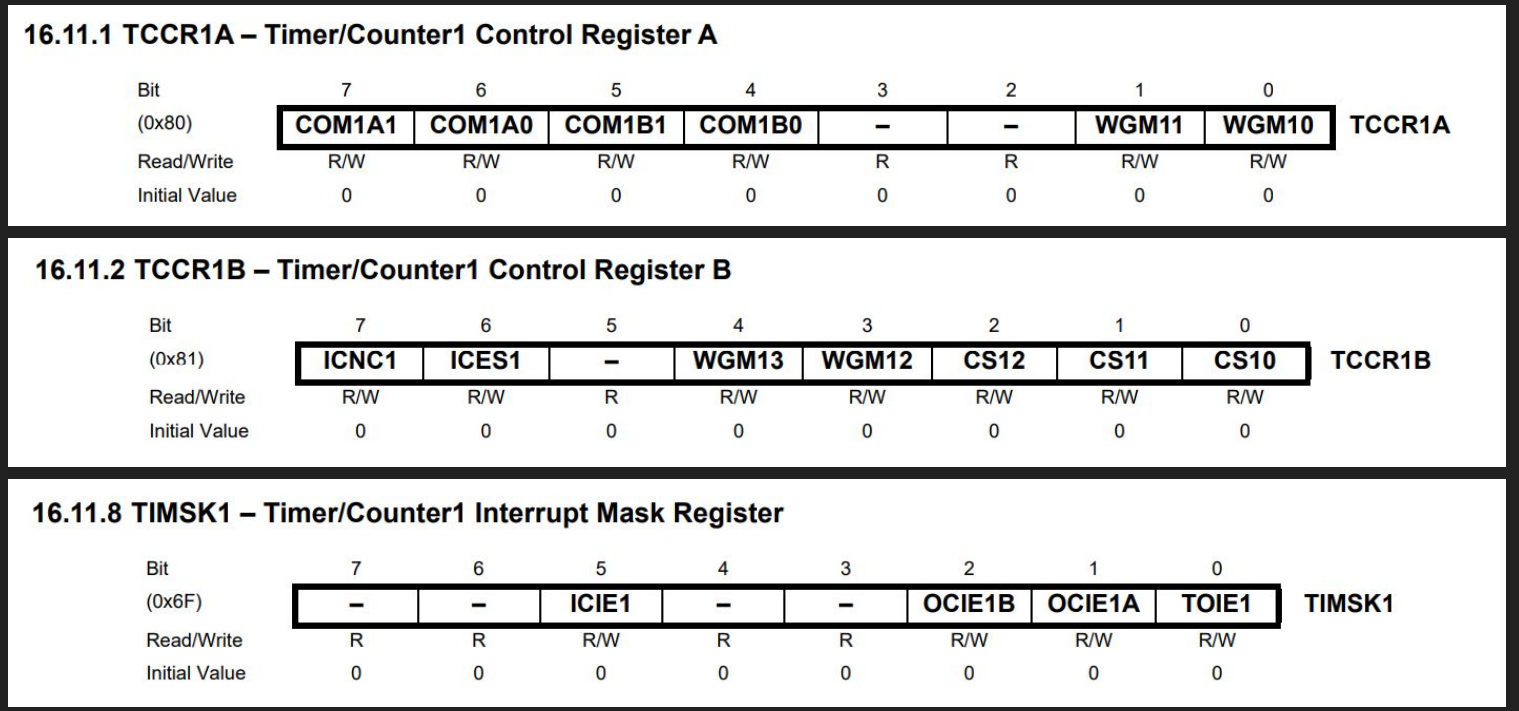
```
bool LED_STATE = true;

void setup() {
  pinMode(13, OUTPUT);        //Set the pin to be OUTPUT
  cli();                      //stop interrupts for till we make the settings
  /*1. First we reset the control register to make sure we start with everything disabled.*/
  TCCR1A = 0;                 // Reset entire TCCR1A to 0
  TCCR1B = 0;                 // Reset entire TCCR1B to 0

  /*2. We set the prescalar to the desired value by changing the CS10 CS12 and CS12 bits. */
  TCCR1B |= B00000100;        //Set CS12 to 1 so we get prescalar 256

  /*3. We enable compare match mode on Timer 1 Output Compare A*/
  TIMSK1 |= B00000010;        //Set OCIE1A to 1 so we enable compare match A

  /*4. Set the value of register A to 31250*/
  OCR1A = 31250;              //Finally we set compare register A to this value
  sei();                      //Enable back the interrupts
}

void loop() {
  // put your main code here, to run repeatedly:
}

//With the settings above, this IRS will trigger each 500ms.
ISR(TIMER1_COMPA_vect){
  TCNT1  = 0;                 //First, set the timer back to 0 so it resets for next interrupt
  LED_STATE = !LED_STATE;     //Invert LED state
  digitalWrite(13,LED_STATE);  //Write new state to the LED on pin D5
}
```

How It Works

- Timer1 counts from 0 - 65546 without prescaler
  - Each increment occurs at 1 clock cycle (16Mhz, 62.5ns)
  - Counting from 0 - 65546 takes 0.004096625 seconds
- Timer1 counts from 0 - 65546 with 256 prescaler
  - Each increment occurs at 256 clock cycles (62.5Khz, 16us)
  - Counting from 0 - 65546 takes 1.048736 seconds
- To oscillate at 500ms we use Compare Match to interrupt when Timer1 reaches 31250 and reset the timer
  - 500ms / 16us = 31250

What is TCCR1A, TCCR1B, TIMSK1?

- These registers control the interrupts and timers/counters:
  (HINT: Read the datasheet!)



16.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

16.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

16.11.8 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Prescalers and Counting

- The prescaler is another clock signal that has a divided frequency based on system clock:
  - Prescaler frequency = $\frac{\text{Clock frequency}}{\text{Prescaling value}}$
  - For a prescaler of 64 on Arduino Uno:
    - 250 Khz = $\frac{\text{16 Mhz}}{64}$

Prescalers and Counting (cont.)

- What is the time difference between increments in the timer?
  - (1/frequency) = (1/250Khz)
    - 4us
- What is the total time for counting assuming max value of 255 or 65546?
  - (T*(Max Value+1)) = (4us * (255 + 1))
    - 0.001024 sec for 255
    - 0.262184 sec for 65546

What is Our Limit + Customize Our Own Signal

- What is the highest frequency using interrupts?
  - A little above 200 Khz
- Can we change the duty cycle?
  - Yes
- What prescaler and compare match needed for 1 Khz 50% duty cycle signal?
- There are many options but here is one:
  - Prescaler = 8
    - Prescaled frequency = 2 Mhz
    - Increment time = 500ns
  - Max Value = 65546
  - Compare Match = 1000
    - 1000 * 500ns = 500us
    - (1/500us)/2 = 1 Khz
  - Use pin toggle every 500us gives a 1Khz square wave with 50% duty cycle

Changing The Duty Cycle

- We can change the duty cycle
  - If we are toggling the pin every 500us we get a 1Khz square wave with 50% duty cycle
  - What if we toggle at different compare match values?
- In the previous example we have compare match = 1000
- For 10% duty cycle we can do this:
  - Timer = 200, then set pin LOW and change Compare match = 1800
  - Timer = 1800, then set pin HIGH and change Compare match = 200
  - Notice: Total compare matches = 2000
    - This should be expected since for 50% duty cycle high and low compare match = 1000

Formula Cheat Sheet

- Compare match value for 50% duty cycle:

$$x_c = \frac{1}{2 \cdot f_{desired} \cdot T_{prescaled}}$$

where:
x_c is the compare match value

$f_{desired}$ is the desired PWM/square wave frequency

$T_{prescaled}$ is the period of the prescaler frequency

Note: if x_c requires rounding down, or is very small, try different prescaler

- Duty cycle for high/low compare match value:

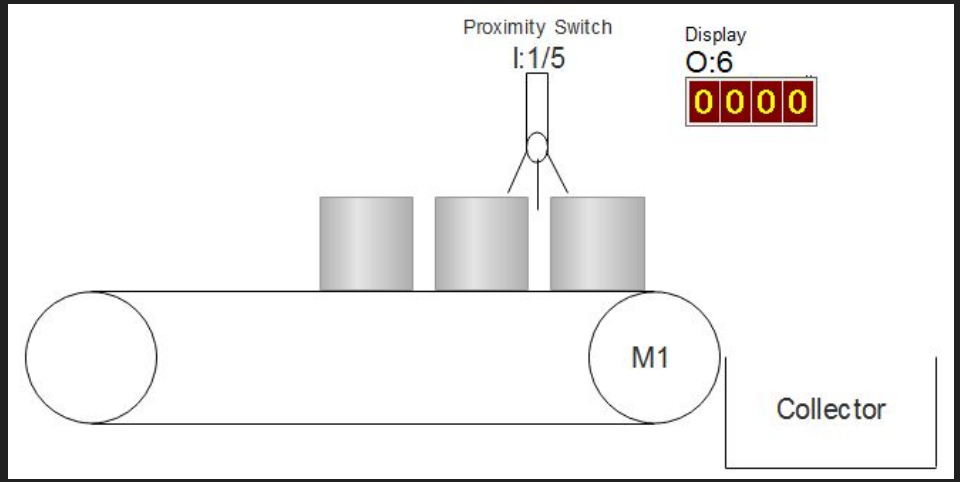
$$x_{HIGH} = 2 \cdot x_c \cdot D$$
$$x_{LOW} = 2 \cdot x_c \cdot (1 - D)$$

Input Capture and Overflow

- Input Capture
  - When a pin rises or falls, trigger the interrupt
- Overflow
  - When the timer overflows, trigger the interrupt
- When the input capture interrupt occurs, track the timer’s count
- We can use this to track the time between rising and falling edges
- Overflow interrupt allows us to check for longer times
- Use your knowledge on timers, prescalers, prescaler frequency, etc. to measure pulses using these new interrupts

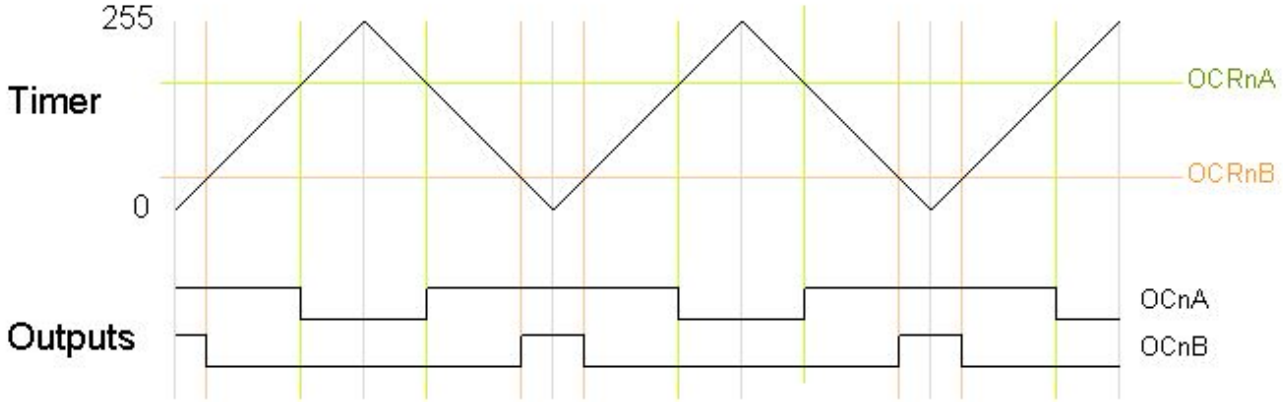
Input Capture for Conveyor Belt

- Assume a sensor outputs 5V (HIGH) when an object crosses a light beam, and outputs 0V (LOW) when there is no object
- Assume all products are 1 inch long and are always equally spaced
- Create an Arduino sketch that uses input capture interrupts to measure the speed of the conveyor system
- Add a PWM output that would control a motor to regulate the conveyor belt to move 5 products per second
- Assume a PWM with 0% duty cycle stops the conveyor belt and scales linearly to 4 feet/sec at 100%



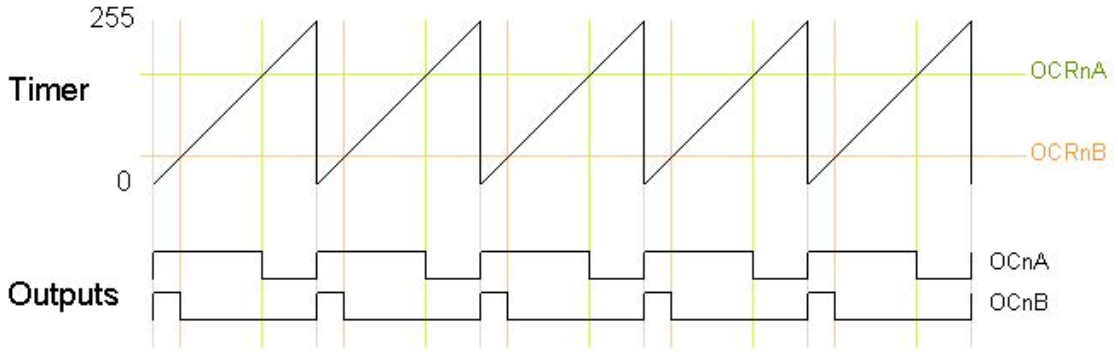
Phase Correct PWM

The second PWM mode is called phase-correct PWM. In this mode, the timer counts from 0 to 255 and then back down to 0. The output turns off as the timer hits the output compare register value on the way up, and turns back on as the timer hits the output compare register value on the way down. The result is a more symmetrical output. The output frequency will be approximately half of the value for fast PWM mode, because the timer runs both up and down.



Fast PWM

In the simplest PWM mode, the timer repeatedly counts from 0 to 255. The output turns on when the timer is at 0, and turns off when the timer matches the output compare register. The higher the value in the output compare register, the higher the duty cycle. This mode is known as Fast PWM Mode. The following diagram shows the outputs for two particular values of OCRnA and OCRnB. Note that both outputs have the same frequency, matching the frequency of a complete timer cycle.



What is OCnA and OCnB?

- OCnA and OCnB are very fast
- Toggle a pin without needing an interrupt
- Easy to achieve square wave in many Mhz range

The Art of Datasheet Reading

- When in doubt, the answer is probably in the datasheet
- The datasheet can provide insight into how a device works
- Helpful when the code does not describe what is going on or when you need to interface directly with the hardware
  - Example:

```
/*2. We set the prescalar to the desired value by changing the CS10 CS12 and CS12 bits. */
TCCR1B |= B00000100;        //Set CS12 to 1 so we get prescalar 256
```

  - How does this work? The datasheet will tell us

The Arduino Datasheet vs ATmega328P Datasheet

- Arduino datasheet is short (14 pages)
  https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf
  - Does not give detailed information on operation
- ATmega328P datasheet is very long (294 pages)
  https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
  - Provides all information and details on the microcontroller that controls the Arduino

Timer1 Specifications

- The datasheet contains all details regarding timer 1 and its functionality
- We are fortunate, interfacing with timer 1 is already heavily documented online and in official Arduino documentation
- Some day you may have to work on a microprocessor or microcontroller that has very little or no online support
- You will have to decipher the datasheet yourself to understand how it works
- We will focus on timer 1 for now since it has the most capabilities and is complex enough

Timer1 Specifications (cont.)

- Specifications for timer 1 from pages 89 - 115 (26 pages):
  - True 16-bit design (i.e., allows 16-bit PWM)
  - Two independent output compare units
  - Double buffered output compare registers
  - One input capture unit
  - Input capture noise canceler
  - Clear timer on compare match (auto reload)
  - Glitch-free, phase correct pulse width modulator (PWM)
  - Variable PWM period
  - Frequency generator
  - External event counter
  - Four independent interrupt sources (TOV1, OCF1A, OCF1B, and ICF1)
- How do are we accessing all these features?

TCCR1A – Timer/Counter1 Control Register A

- Set Compare Output Mode
  - Bits: COM1A1, COM1A0, COM1B1, COM1B0
  - OC1A/OC1B set, clear, toggle
  - Dependent on Waveform Generation Mode
- Set Waveform Generation Mode
  - Bits: WGM11, WGM10
    - Note: WGM13, WGM12 are in TCCR1B
  - Normal
  - PWM Phase Correct
  - Fast PWM

TCCR1A – Timer/Counter1 Control Register A (cont.)

15.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – COM1A1:0: Compare Output Mode for Channel A**
- **Bit 5:4 – COM1B1:0: Compare Output Mode for Channel B**

The COM1A1:0 and COM1B1:0 control the output compare pins (OC1A and OC1B respectively) behavior. If one or both of the COM1A1:0 bits are written to one, the OC1A output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COM1B1:0 bit are written to one, the OC1B output overrides the normal port functionality of the I/O pin it is connected to. However, note that the data direction register (DDR) bit corresponding to the OC1A or OC1B pin must be set in order to enable the output driver.

When the OC1A or OC1B is connected to the pin, the function of the COM1x1:0 bits is dependent of the WGM13:0 bits setting. Table 15-2 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to a normal or a CTC mode (non-PWM).

Table 15-2. Compare Output Mode, non-PWM

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on compare match.
1	0	Clear OC1A/OC1B on compare match (set output to low level).
1	1	Set OC1A/OC1B on compare match (set output to high level).

Table 15-3 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to the fast PWM mode.

Table 15-3. Compare Output Mode, Fast PWM(1)

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 14 or 15: Toggle OC1A on compare match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on compare match, set OC1A/OC1B at BOTTOM (non-inverting mode)
1	1	Set OC1A/OC1B on compare match, clear OC1A/OC1B at BOTTOM (inverting mode)

Note: 1. A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. In this case the compare match is ignored, but the set or clear is done at BOTTOM. See Section 15.9.3 "Fast PWM Mode" on page 101 for more details.

Table 15-4 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to the phase correct or the phase and frequency correct, PWM mode.

Table 15-4. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM(1)

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 9 or 11: Toggle OC1A on compare match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on compare match when up-counting. Set OC1A/OC1B on compare match when down counting.
1	1	Set OC1A/OC1B on compare match when up-counting. Clear OC1A/OC1B on compare match when down counting.

Note: 1. A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. See Section 15.9.4 "Phase Correct PWM Mode" on page 103 for more details.

- **Bit 1:0 – WGM11:0: Waveform Generation Mode**

Combined with the WGM13:2 bits found in the TCCR1B register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see Table 15-5. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), clear timer on compare match (CTC) mode, and three types of pulse width modulation (PWM) modes. See (Section 15.9 "Modes of Operation" on page 100).

Table 15-5. Waveform Generation Mode Bit Description(1)

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, phase correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, phase correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, phase correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, phase and frequency correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, phase and frequency correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, phase correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, phase correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

TCCR1B – Timer/Counter1 Control Register B

- Set Input Capture Features
  - Bits: ICNC1, ICES1
  - Input Capture Noise Canceler
  - Input Capture Edge Select
- Set Prescaler
  - Bits: CS12, CS11, CS10
  - Disable clock, prescaled 1, 8, 64, 256, 1024, or external clock
- Set Waveform Generation Mode
  - Bits: WGM13, WGM12
    - Note: WGM11, WGM10 are in TCCR1A
  - Normal
  - PWM Phase Correct
  - Fast PWM

TCCR1B – Timer/Counter1 Control Register B (cont.)

15.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ICNC1: Input Capture Noise Canceler**

Setting this bit (to one) activates the input capture noise canceler. When the noise canceler is activated, the input from the input capture pin (ICP1) is filtered. The filter function requires four successive equal valued samples of the ICP1 pin for changing its output. The input capture is therefore delayed by four oscillator cycles when the noise canceler is enabled.

- **Bit 6 – ICES1: Input Capture Edge Select**

This bit selects which edge on the input capture pin (ICP1) that is used to trigger a capture event. When the ICES1 bit is written to zero, a falling (negative) edge is used as trigger, and when the ICES1 bit is written to one, a rising (positive) edge will trigger the capture.

When a capture is triggered according to the ICES1 setting, the counter value is copied into the input capture register (ICR1). The event will also set the input capture flag (ICF1), and this can be used to cause an input capture interrupt, if this interrupt is enabled.

When the ICR1 is used as TOP value (see description of the WGM13:0 bits located in the TCCR1A and the TCCR1B register), the ICP1 is disconnected and consequently the input capture function is disabled.

- **Bit 5 – Reserved Bit**

This bit is reserved for future use. For ensuring compatibility with future devices, this bit must be written to zero when TCCR1B is written.

- **Bit 4:3 – WGM13:2: Waveform Generation Mode**

See TCCR1A register description.

- **Bit 2:0 – CS12:0: Clock Select**

The three clock select bits select the clock source to be used by the Timer/Counter, see Figure 15-10 on page 106 and Figure 15-11 on page 106.

Table 15-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)
1	0	0	$clk_{I/O}$/256 (from prescaler)
1	0	1	$clk_{I/O}$/1024 (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter1, transitions on the T1 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

More Info on PWM

- https://docs.arduino.cc/tutorials/generic/secrets-of-arduino-pwm

Changing The Duty Cycle Sample Code

```
bool isHigh = false;

void setup() {
  DDRB |= B00010000;            //Sets only D12 as OUTPUT
  cli();                  //stop interrupts for till we make the settings

  /*1. First we reset the control register to make sure we start with everything disabled.*/
  TCCR1A = 0;               // Reset entire TCCR1A to 0
  TCCR1B = 0;               // Reset entire TCCR1B to 0

  /*2. We set the prescalar to the desired value by changing the CS10 CS12 and CS12 bits. */
  TCCR1B |= B00000010;      //Set CS11 to 1 so we get prescalar 8

  /*3. We enable compare match mode on Timer 1 Output Compare A*/
  TIMSK1 |= B00000010;      //Set OCIE1A to 1 so we enable compare match A

  OCR1A = 1800;             //For timing while pin is LOW
  sei();                  //Enable back the interrupts
}

void loop() {
  // put your main code here, to run repeatedly:
}

//With the settings above, this IRS will trigger each 500ms.
ISR(TIMER1_COMPA_vect)
{
  TCNT1  = 0;               //First, set the timer back to 0 so it resets for next interrupt
  isHigh = !isHigh;
  PORTB ^= 1 << 4;

  if(isHigh)
  {
    OCR1A = 200;            //For timing while pin is HIGH
  }
  else
  {
    OCR1A = 1800;           //For timing while pin is LOW
  }

}
```

Changing The Duty Cycle + Sweep Sample Code

```
bool isHigh = false;
int sweep = 1;

void setup() {
  DDRB |= B00010000;            //Sets only D12 as OUTPUT
  cli();                        //stop interrupts for till we make the settings

  /*1. First we reset the control register to make sure we start with everything disabled.*/
  TCCR1A = 0;                   // Reset entire TCCR1A to 0
  TCCR1B = 0;                   // Reset entire TCCR1B to 0

  /*2. We set the prescalar to the desired value by changing the CS10 CS12 and CS12 bits. */
  TCCR1B |= B00000010;      //Set CS11 to 1 so we get prescalar 8

  /*3. We enable compare match mode on Timer 1 Output Compare A*/
  TIMSK1 |= B00000010;      //Set OCIE1A to 1 so we enable compare match A

  OCR1A = 1800;             //For timing while pin is LOW
  sei();                    //Enable back the interrupts
}

void loop() {
  // put your main code here, to run repeatedly:
}

//With the settings above, this IRS will trigger each 500ms.
ISR(TIMER1_COMPA_vect)
{
  TCNT1  = 0;               //First, set the timer back to 0 so it resets for next interrupt
  isHigh = !isHigh;
  PORTB ^= 1 << 4;

  if(isHigh)
  {
    OCR1A = sweep;          //For timing while pin is HIGH
  }
  else
  {
    OCR1A = 2000 - sweep;           //For timing while pin is LOW
  }

  sweep++;

  if(sweep >= 2000)
  {
    sweep = 1;
  }

}
```

Changing The Duty Cycle + Variable Sample Code

```
bool isHigh = false;
float dutyCycle = 0.1;

void setup() {
  DDRB |= B00010000;            //Sets only D12 as OUTPUT
  cli();                        //stop interrupts for till we make the settings

  /*1. First we reset the control register to make sure we start with everything disabled.*/
  TCCR1A = 0;                   // Reset entire TCCR1A to 0
  TCCR1B = 0;                   // Reset entire TCCR1B to 0

  /*2. We set the prescalar to the desired value by changing the CS10 CS12 and CS12 bits. */
  TCCR1B |= B00000010;      //Set CS11 to 1 so we get prescalar 8

  /*3. We enable compare match mode on Timer 1 Output Compare A*/
  TIMSK1 |= B00000010;      //Set OCIE1A to 1 so we enable compare match A

  OCR1A = 1800;             //For timing while pin is LOW
  sei();                    //Enable back the interrupts
}

void loop() {
  // put your main code here, to run repeatedly:
}

//With the settings above, this IRS will trigger each 500ms.
ISR(TIMER1_COMPA_vect)
{
  TCNT1  = 0;               //First, set the timer back to 0 so it resets for next interrupt
  isHigh = !isHigh;
  PORTB ^= 1 << 4;

  if(isHigh)
  {
    OCR1A = 2000 * dutyCycle;           //For timing while pin is HIGH
  }
  else
  {
    OCR1A = 2000 * (1- dutyCycle);          //For timing while pin is LOW
  }

}
```

Input Capture and Overflow Sample Code

```
//Input compare pin must be 8
#define inputComparePin 8

// Long integers are required for high counts
int long overflows, T1, T2, T, pulseCount = 0;
double pulseTime = 0;

// These are used for printing to the display
float printTime = 1.0f;
float prevPrintTime = 0.0f;

ISR(TIMER1_OVF_vect)
{
  // If the timer overflowed, increment the overflows integer
  overflows++;
}

ISR(TIMER1_CAPT_vect)
{
  // Record the input capture count ASAP to record it before it changes during the ISR
  int long count = ICR1;

  // If a rising edge is detected
  if(bit_is_set(TCCR1B,ICES1))
  {
    // Record the current time (counts @ rising edge) and reset overflows
    T1 = count;
    overflows = 0;
  }
  // If a falling edge is detected
  else
  {
    // Record the current time (counts @ falling edge)
    T2 = count;

    // If we overflowed at least once, multiply the overflows by the max value (65546)
    if(overflows > 0)
    {
      T2 += (overflows*65546);
    }

    // Calculate the difference between the rising edge counter and falling edge
    // Use the increment time step (1/16Mhz) to convert total counts (T) into seconds
    T = T2 - T1;
    pulseTime = float(T) * (1.0f/16000000.0f);
  }

  // Toggle to trigger on opposite edge
  TCCR1B ^= _BV(ICES1);
}
```

```
void setup()
{
  // Begin communicating on serial monitor
  Serial.begin(9600);
  pinMode(inputComparePin, INPUT);

  // We use pin 6 for pulsing the input compare (just for testing purposes)
  pinMode(6, OUTPUT);

  cli();
  // Reset control registers
  TCCR1A = TCCR1B = 0;
  // Set lowest prescale value
  TCCR1B = _BV(CS10);
  // Enable input capture and overflow interrupts
  TIMSK1 |= _BV(ICIE1) + _BV(TOIE1);
  sei();
}
void loop()
{
  // Toggle the pin very fast
  PORTD |= B01000000;
  delayMicroseconds(100);
  PORTD &= ~B01000000;
  delayMicroseconds(900);


  // Everything below is for printing time and count
  float currentTime = millis()/1000.0f;

  // If the time currently exceeds the last time we printed plus the print duration, print again (once every
second)
  if(currentTime >= (prevPrintTime + printTime))
  {
    prevPrintTime = currentTime;

    Serial.print("Pulse Time: ");
    Serial.println(pulseTime, 6);

    Serial.print("Counts: ");
    Serial.println(T);
  }

}
```

PWM Phase Correct Sample Code

```
// Choose a frequency and duty cycle
float frequency = 10000.0f;
float dutyCycle = 0.25f;

ISR(TIMER1_COMPA_vect)
{
  // When the compare match value reached toggle pin
  PORTB ^= B00000001;
}

void setup()
{
  // Disable interrupts
  cli();
  // Use pin 8 for PWM output
  pinMode(8, OUTPUT);
  // Reset control registers
  TCCR1A = TCCR1B = 0;
  // Set PWM phase corrected mode + prescaler = 1
  TCCR1A = _BV(WGM11);
  TCCR1B = _BV(WGM13) | _BV(CS10);
  // Enable compare match interrupt
  TIMSK1 |= (1 << OCIE1A);
  // Set TOP
  ICR1 = int(1.0f/(2.0f*frequency*(1.0f/16000000.0f)));
  // Set compare match
  OCR1A = int(ICR1*dutyCycle);
  // Enable interrupts
  sei();
}
void loop()
{

}
```

OCnA Sample Code

```
float frequency = 1000000.0f;

void setup() {
  pinMode(9,OUTPUT);
  TCCR1A = TCCR1B = 0;
  TCCR1A = (1<<COM1A0); // Toggle mode on OC1A
  TCCR1B = (1 << WGM12) | (1 << CS10); // CTC mode, no prescaler
  OCR1A = (1.0f/(2.0f*frequency*(1.0f/16000000.0f))) - 1; // Compare match value
}

void loop(){

}
```

Register/Timer/Port Additional Resources

- Port Control:
  - https://electronoobs.com/eng_arduino_tut130.php
- Timer Control:
  - https://electronoobs.com/eng_arduino_tut140.php
- Registers, timers, interrupts:
  - https://electronoobs.com/eng_arduino_tut12.php