Microprocessors Lab

5- Internal Interrupts

Instructor: Dr. Mohsen Raji

Graders:

- Mohammad Hossein Hashemi
- Reza Hesami
- Negin Shirvani

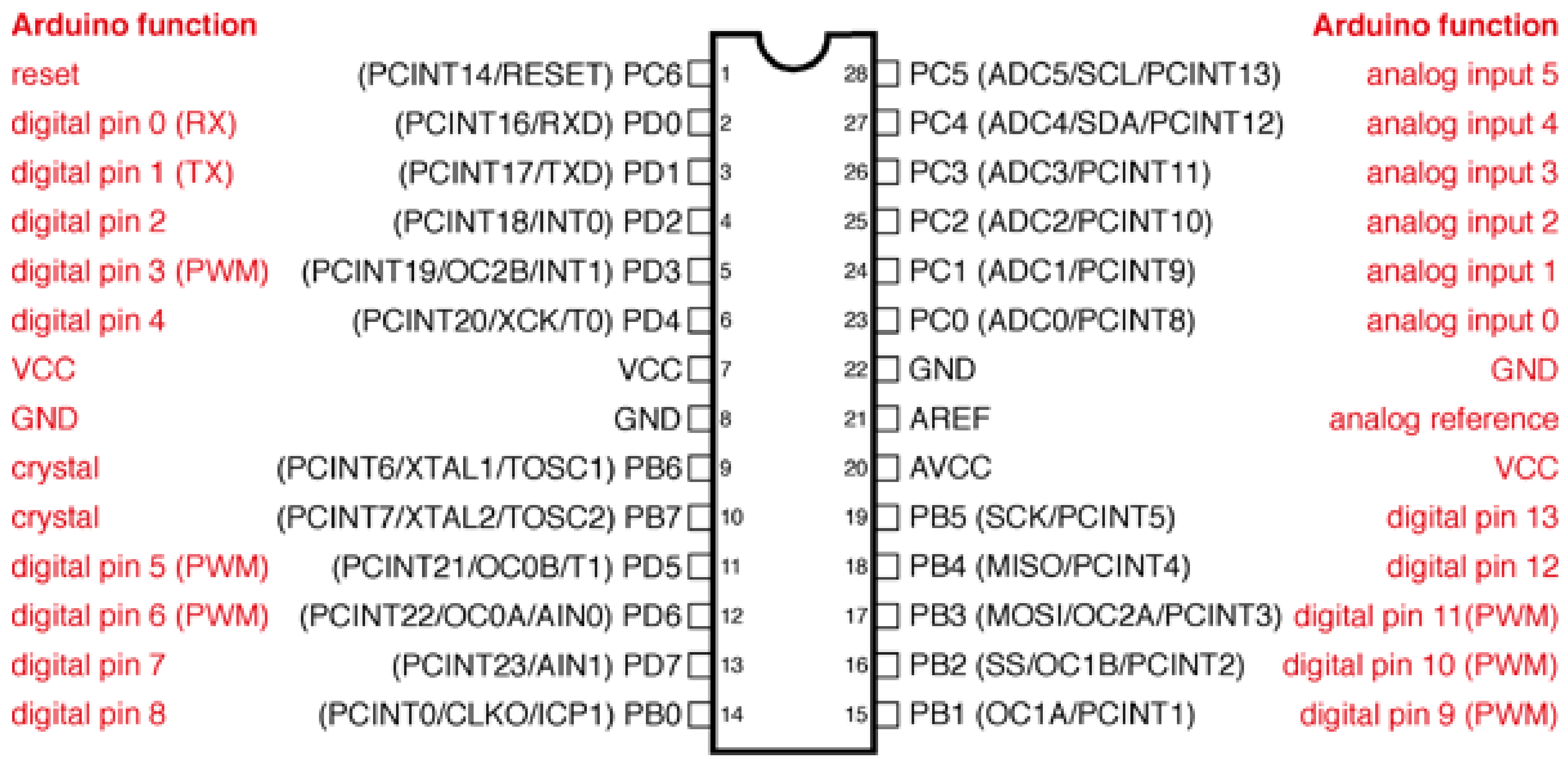
Shiraz University – Fall 2022

Based on the slides by:

- Hossein Dehghanipour
- Mohammad Hossein Allah Akbari

- Timer interrupts allow you to perform a task at very specifically timed intervals regardless of what else is going on in your code.
- Normally when you write an Arduino sketch the Arduino performs all the commands encapsulated in the loop() {} function in the order that they are written, however, it's difficult to time events in the loop(). Some commands take longer than others to execute, some depend on conditional statements (if, while...) and some Arduino library functions (like digitalWrite or analogRead) are made up of many commands. Arduino timer interrupts allow you to momentarily pause the normal sequence of events taking place in the loop() function at precisely timed intervals, while you execute a separate set of commands. Once these commands are done the Arduino picks up again where it was in the loop().
- Interrupts are useful for:
  - Measuring an incoming signal at equally spaced intervals (constant sampling frequency)
  - Calculating the time between two events
  - Sending out a signal of a specific frequency
  - Periodically checking for incoming serial data
  - much more...

Atmega168 Pin Mapping



Arduino function					Arduino function
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22	GND	GND
GND	GND	8	21	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11(PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

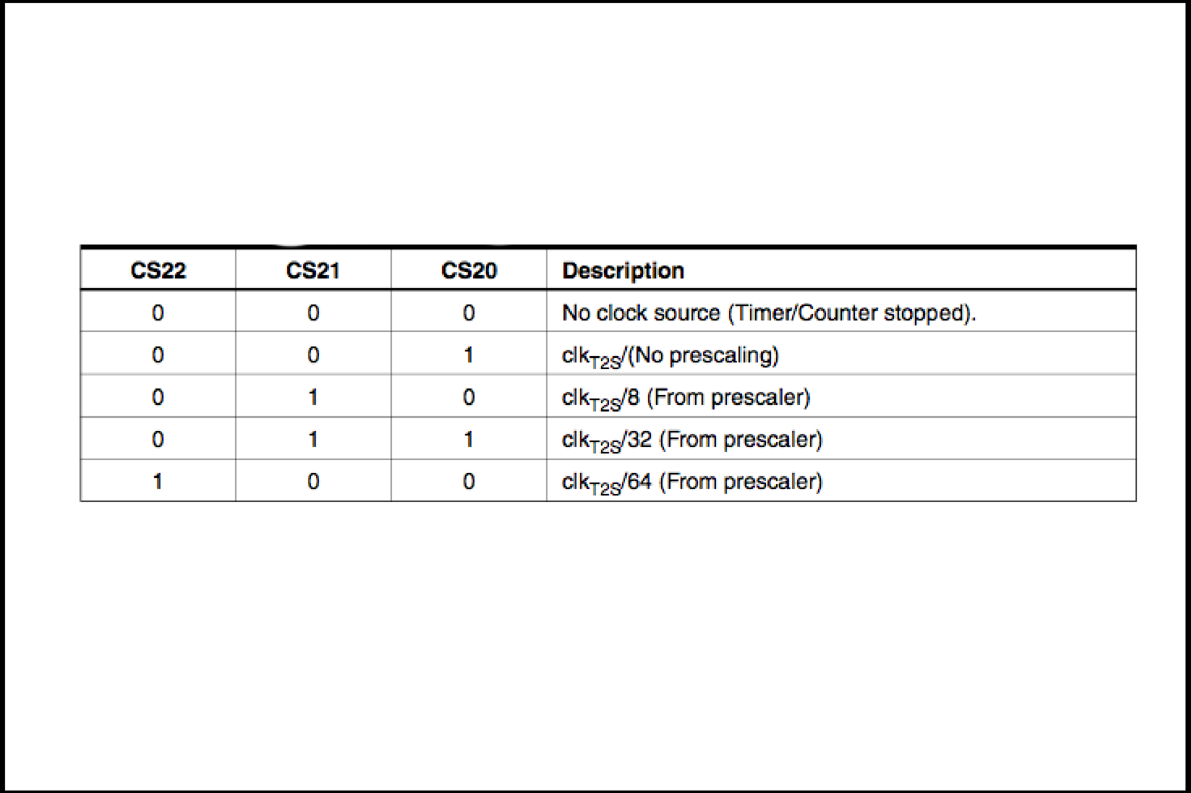
Digital Pins 11,12 & 13 are used by the ICSP header for MISO, MOSI, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

Prescalers and the Compare Match Register

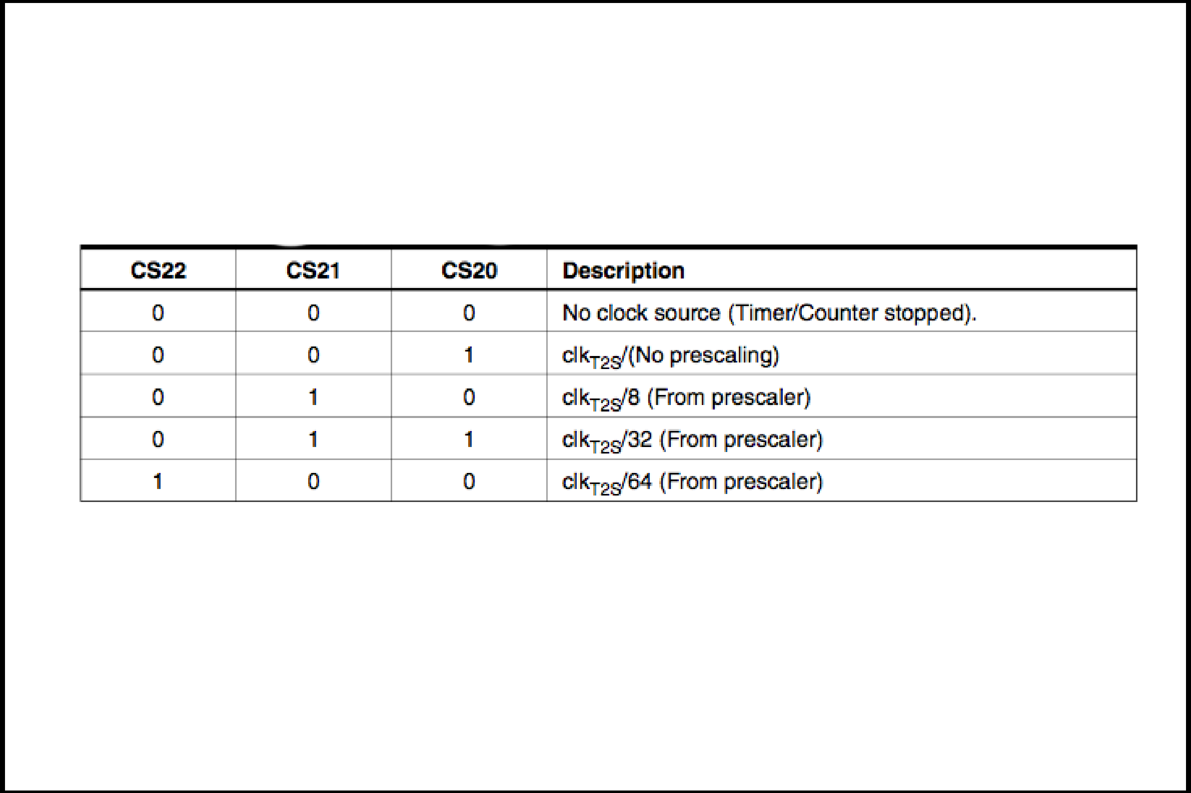
- The Uno has three timers called timer0, timer1, and timer2. Each of the timers has a counter that is incremented on each tick of the timer's clock. ***CTC*** timer interrupts are triggered when the counter reaches a specified value, stored in the compare match register. Once a timer counter reaches this value it will clear (reset to zero) on the next tick of the timer's clock, then it will continue to count up to the compare match value again. By choosing the compare match value and setting the speed at which the timer increments the counter, you can control the *frequency* of timer interrupts.
- The first parameter we'll discuss is the speed at which the timer increments the counter. The Arduino clock runs at 16MHz, this is the fastest speed that the timers can increment their counters. At 16MHz each tick of the counter represents 1/16,000,000 of a second (~63ns), so a counter will take 10/16,000,000 seconds to reach a value of 9 (counters are 0 indexed), and 100/16,000,000 seconds to reach a value of 99.
- In many situations, you will find that setting the counter speed to 16MHz is too fast. Timer0 and timer2 are 8 bit timers, meaning they can store a maximum counter value of 255. Timer1 is a 16 bit timer, meaning it can store a maximum counter value of 65535. Once a counter reaches its maximum, it will tick back to zero (this is called overflow). This means at 16MHz, even if we set the compare match register to the max counter value, interrupts will occur every 256/16,000,000 seconds (~16us) for the 8 bit counters, and every 65,536/16,000,000 (~4 ms) seconds for the 16 bit counter. Clearly, this is not very useful if you only want to interrupt once a second (WHY?).

- Instead you can control the speed of the timer counter incrementation by using something called a prescaler. A prescaler dictates the speed of your timer according the the following equation:

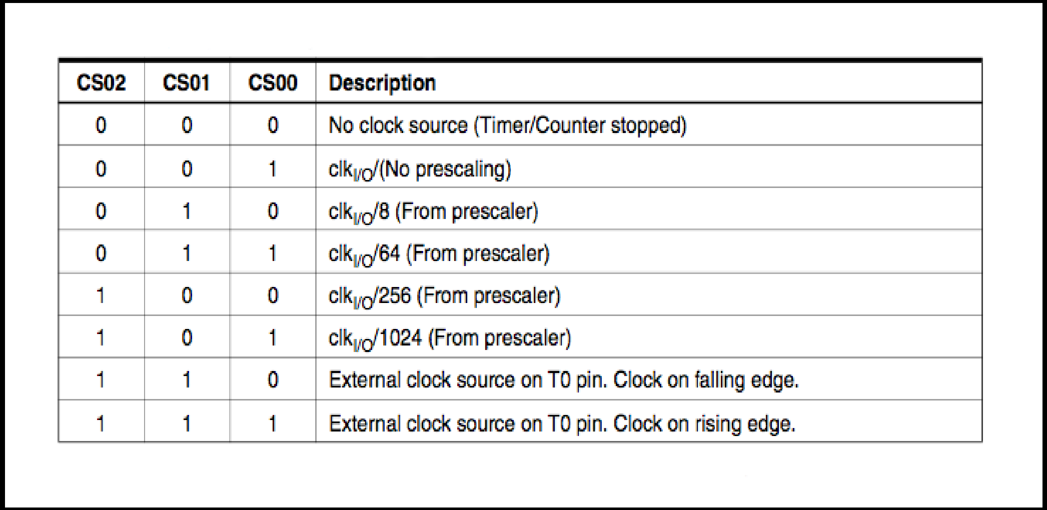
(timer speed (Hz)) = (Arduino clock speed (16MHz)) / prescaler



CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_{T2S}/(No prescaling)
0	1	0	clk_{T2S}/8 (From prescaler)
0	1	1	clk_{T2S}/32 (From prescaler)
1	0	0	clk_{T2S}/64 (From prescaler)



CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_{T2S}/(No prescaling)
0	1	0	clk_{T2S}/8 (From prescaler)
0	1	1	clk_{T2S}/32 (From prescaler)
1	0	0	clk_{T2S}/64 (From prescaler)



CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(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 T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

So a 1 prescaler will increment the counter at 16MHz, an 8 prescaler will increment it at 2MHz, a 64 prescaler = 250kHz, and so on. As indicated in the tables above, the prescaler can equal 1, 8, 64, 256, and 1024. (We'll explain the meaning of CS12, CS11, and CS10 in the next step.)

Now you can calculate the interrupt frequency with the following equation:

interrupt frequency (Hz) =
(Arduino clock speed 16,000,000Hz) / (prescaler * (compare match register + 1))

The +1 is there because the compare match register is zero indexed.

Rearranging the equation above, you can solve for the compare match register value that will give your desired interrupt frequency:

compare match register =
[16,000,000Hz / (prescaler * desired interrupt frequency)] – 1

Remember that when you use timers 0 and 2, this number must be less than 256, and less than 65536 for timer1.

So if you want an interrupt every second (frequency of 1Hz):
compare match register = [16,000,000 / (prescaler * 1)] -1

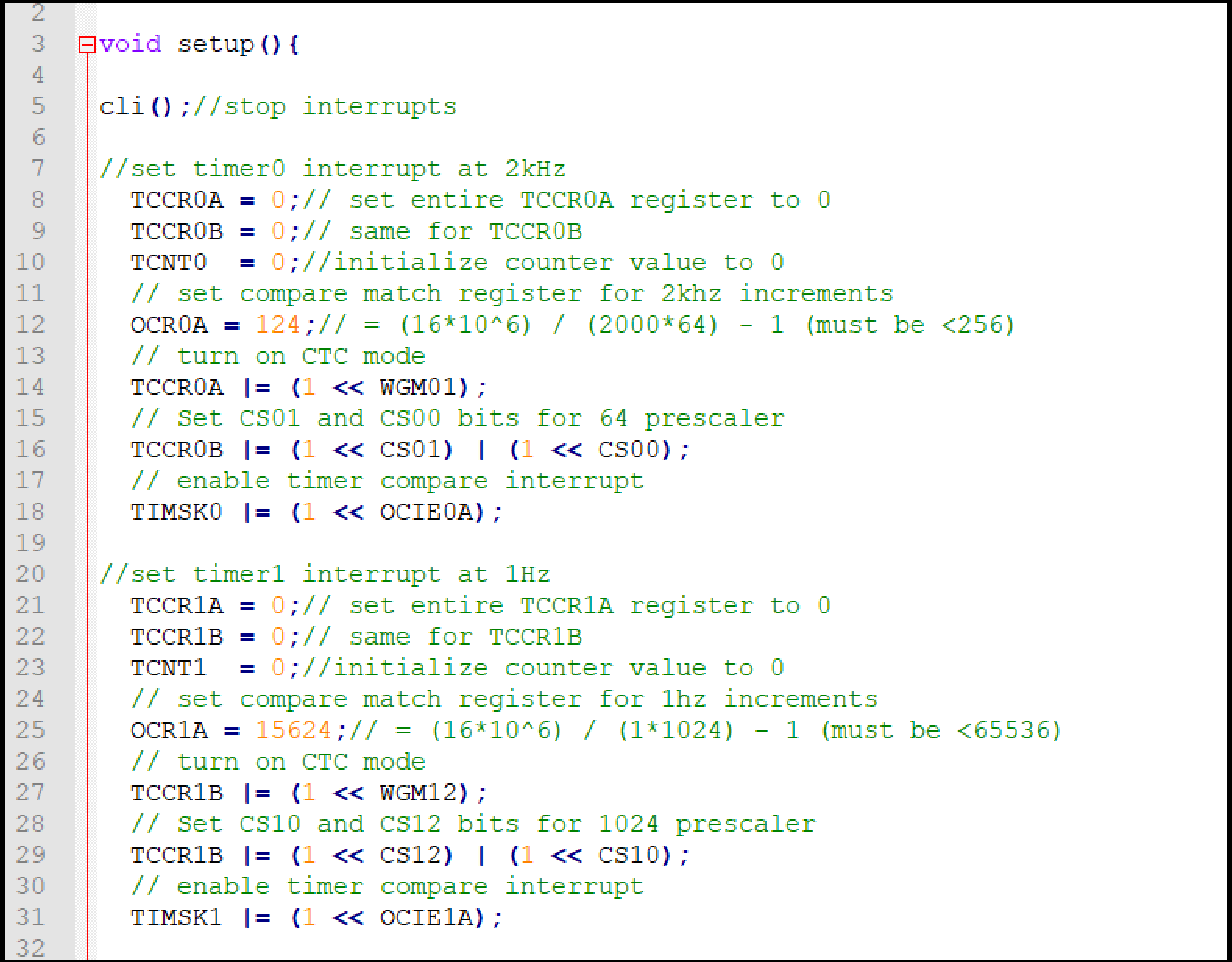
With a prescaler of 1024 you get:
compare match register = [16,000,000 / (1024 * 1)] -1
= 15,624

And since 256 < 15,624 < 65,536, you must use timer1 for this interrupt.

Structuring Timer Interrupts

- Timer setup code is done inside the setup() function in an Arduino sketch.
- Although there is no limitation for using them in loop() function, it is usually not needed except for some specific registers like OCR.
- The code involved for setting up timer interrupts is a little daunting to look at, but it's actually not that hard. You can pretty much just copy the same main chunk of code and change the prescaler and compare match registers to set the correct interrupt frequency.
- However, you need to know about registers’ functionalities in order to change them effectively.

The main structure of the interrupt setup looks like this:



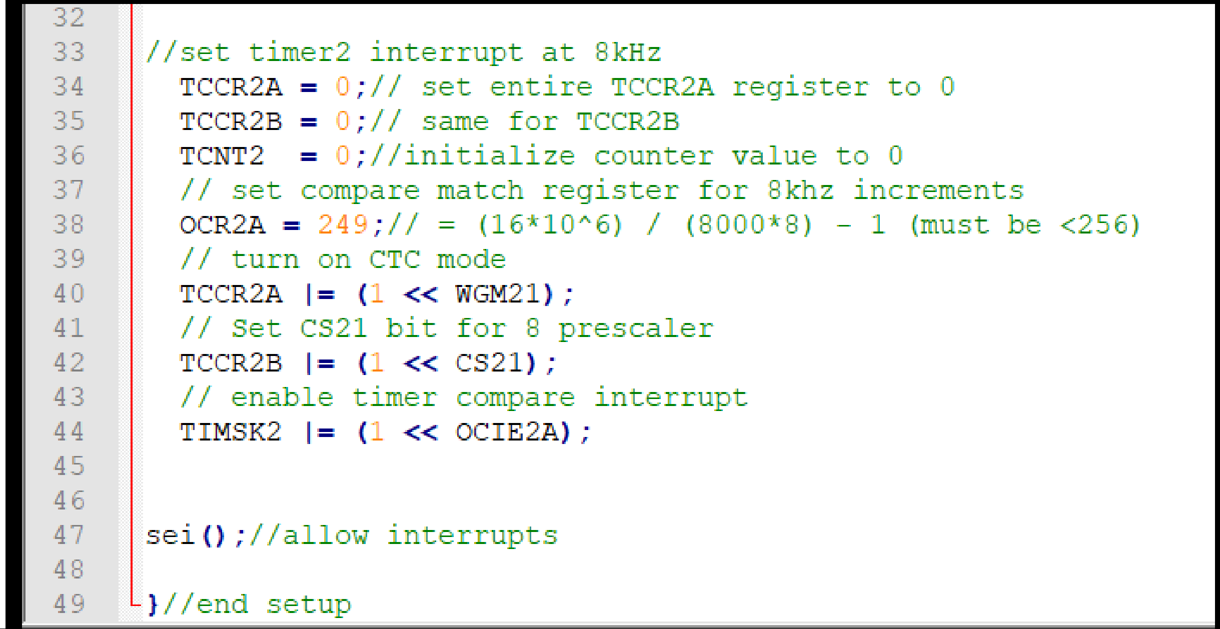
```
void setup(){

cli();//stop interrupts

//set timer0 interrupt at 2kHz
  TCCR0A = 0;// set entire TCCR0A register to 0
  TCCR0B = 0;// same for TCCR0B
  TCNT0  = 0;//initialize counter value to 0
  // set compare match register for 2khz increments
  OCR0A = 124;// = (16*10^6) / (2000*64) - 1 (must be <256)
  // turn on CTC mode
  TCCR0A |= (1 << WGM01);
  // Set CS01 and CS00 bits for 64 prescaler
  TCCR0B |= (1 << CS01) | (1 << CS00);
  // enable timer compare interrupt
  TIMSK0 |= (1 << OCIE0A);

//set timer1 interrupt at 1Hz
  TCCR1A = 0;// set entire TCCR1A register to 0
  TCCR1B = 0;// same for TCCR1B
  TCNT1  = 0;//initialize counter value to 0
  // set compare match register for 1hz increments
  OCR1A = 15624;// = (16*10^6) / (1*1024) - 1 (must be <65536)
  // turn on CTC mode
  TCCR1B |= (1 << WGM12);
  // Set CS10 and CS12 bits for 1024 prescaler
  TCCR1B |= (1 << CS12) | (1 << CS10);
  // enable timer compare interrupt
  TIMSK1 |= (1 << OCIE1A);

```



```

//set timer2 interrupt at 8kHz
  TCCR2A = 0;// set entire TCCR2A register to 0
  TCCR2B = 0;// same for TCCR2B
  TCNT2  = 0;//initialize counter value to 0
  // set compare match register for 8khz increments
  OCR2A = 249;// = (16*10^6) / (8000*8) - 1 (must be <256)
  // turn on CTC mode
  TCCR2A |= (1 << WGM21);
  // Set CS21 bit for 8 prescaler
  TCCR2B |= (1 << CS21);
  // enable timer compare interrupt
  TIMSK2 |= (1 << OCIE2A);


sei();//allow interrupts

}//end setup
```

Reference: https://www.instructables.com/Arduino-Timer-Interrupts/

- Notice how the value of OCR#A (the compare match value) changes for each of these timer setups. As explained in the last step, this was calculated according to the following equation:

  compare match register = [16,000,000Hz/ (prescaler * desired interrupt frequency)] – 1

  Remember that when you use timers 0 and 2 this number must be less than 256, and less than 65536 for timer1

- Also notice how the setups between the three timers differ slightly in the line which turns on CTC mode:

  ```
  TCCR0A |= (1 << WGM01);//for timer0
  TCCR1B |= (1 << WGM12);//for timer1
  TCCR2A |= (1 << WGM21);//for timer2
  ```

  This follows directly from the datasheet of the ATMEL 328/168.

- Finally, notice how the setup for the prescalers follows the tables in the last step (the table for timer 0 is repeated above),

  ```
  TCCR2B |= (1 << CS22);  // Set CS#2 bit for 64 prescaler for timer 2
  TCCR1B |= (1 << CS11);  // Set CS#1 bit for 8 prescaler for timer 1
  TCCR0B |= (1 << CS02) | (1 << CS00);  // Set CS#2 and CS#0 bits for 1024 prescaler for timer 0
  ```

  Notice in the last step that there are different prescaling options for the different timers. For example, timer2 does not have the option of 1024 prescaler.

Timer Library

- Normally, internal interrupts are created by timer registers. Arduino has three timers. They are timer zero, timer one, timer two (we have already mentioned them). In this tutorial we will create internal interrupt using timer one interrupt. So, we need timer one library. By using this interrupt we can create three types of internal interrupts. They are Timer Overflow Interrupt, Output Compare Match Interrupt, Timer Input Capture Interrupt. We won’t go too much deep in this tutorial. We will just make an internal interrupt using some functions. This will be easy to understand. Our aim is to create an interrupt which will blink an LED using pin 10.

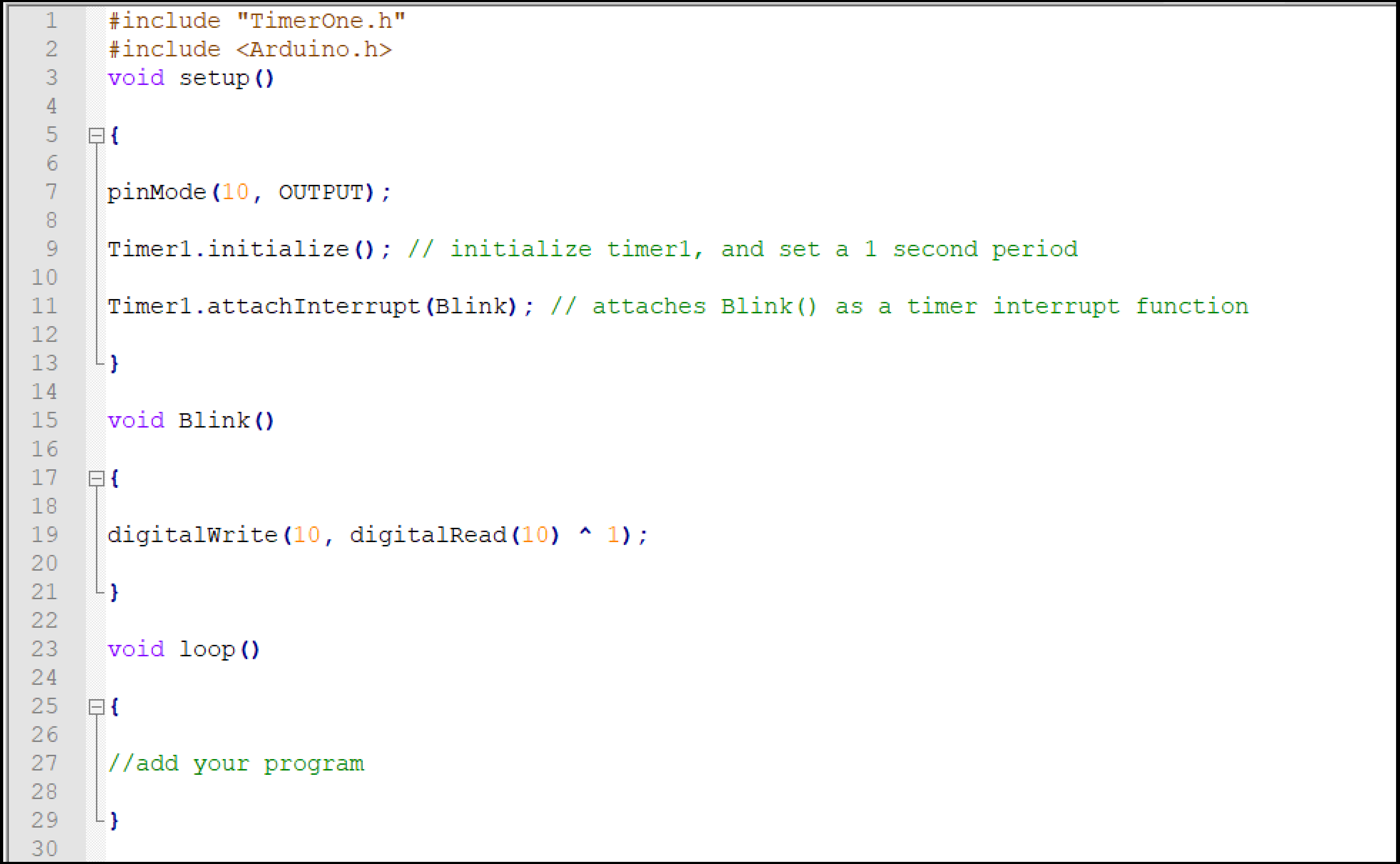
- First you have to include the library in your program. That is,
  - #include "TimerOne.h"

- We need to declare a pin to make a LED blink. Let, declared pin 10 as OUTPUT.
  - pinMode(10, OUTPUT);

- Now, we have to initialize internal interrupt to create internal interrupt. The name of the function which initialize interrupt is
  - "TimerX.initialize(period)"
  - here ,
    - X= 0,1,2; (Since arduino uno has three timer zero,one and two)
    - Period = period mean time in microsecond. By default period is set at 1 second.
- Since, we will use timer one. So function will be,
  - Timer1.initialize(); //set a 1 second period

- In the previous step we initialize internal interrupt. Now, time comes to call interrupt function to do the work in the interrupt. To call the interrupt function we need to use,
  - "TimerX.attachInterrupt(function)"
    - X = 0,1,2; (Since, arduino uno has three timer zero,one and two)
    - function = function is the name of the function that we are going to use as interrupt function.
- I want to use the name of the of the function is "Blink". So the command is,
  - Timer1.attachInterrupt(Blink); //initialize internal interrupt function Blink

- We want to blink LED. So, we need to toggle pin 10 in the interrupt function in "Blink" function.
  - void Blink()
  - {
  - digitalWrite(10, digitalRead(10) ^ 1);//Toggle pin 10 for blink
  - }



```
#include "TimerOne.h"
#include <Arduino.h>
void setup()

{

pinMode(10, OUTPUT);

Timer1.initialize(); // initialize timer1, and set a 1 second period

Timer1.attachInterrupt(Blink); // attaches Blink() as a timer interrupt function

}

void Blink()

{

digitalWrite(10, digitalRead(10) ^ 1);

}

void loop()

{

//add your program

}

```

After all this time, if you still have any questions and you don’t know what to do, go ahead; ask your TAs.

I don’t care anymore.

Good Luck ☺

References

https://www.instructables.com/Arduino-Timer-Interrupts/

https://www.instructables.com/Create-Internal-Interrupt-In-Arduino/

More To read:

https://www.robotshop.com/community/forum/t/arduino-101-timers-and-interrupts/13072