Lab no 08: Interrupts and Timers

The purpose of this Lab is to know how to make blink led application using timers and interrupts (using switch) interfaces

Parts: -

1. What are Interrupts?
   a) What is ISR?
   b) External Interrupt vs Internal Interrupt?
   c) Maskable Interrupt vs Non Maskable Interrupt?
   d) Overview of Interrupt Process
   e) How to code Blink LED with Button?

2. Timers / Counters
   A) What is prescaler?
   B) Modes of Operation
   C) Timer/Counter 0 (8 Bits)
   D) How to code Blink LED in 1 second using timer 0?
   E) Difference Between Interrupt and Polling
   F) What's The difference between interrupts and delay in embedded programming?

Part 1. What are Interrupts?

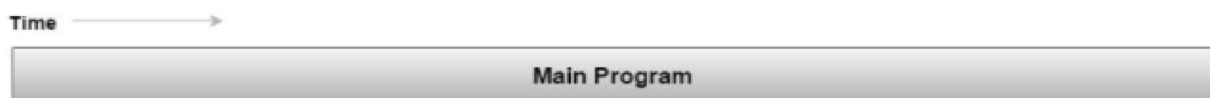
An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. Whenever an interrupt occurs, the controller completes the execution of the current instruction and starts the execution of an Interrupt Service Routine (ISR) or Interrupt Handler. ISR tells the processor or controller what to do when the interrupt occurs. The interrupts can be either hardware interrupts or software interrupts.

Part 1. A) Interrupt Service Routine

For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler. When an interrupt occurs, the microcontroller runs the interrupt service routine. For every interrupt, there is a fixed location in memory that holds the address of its interrupt service routine, ISR. The table of memory locations set aside to hold the addresses of ISRs is called as the Interrupt Vector Table.

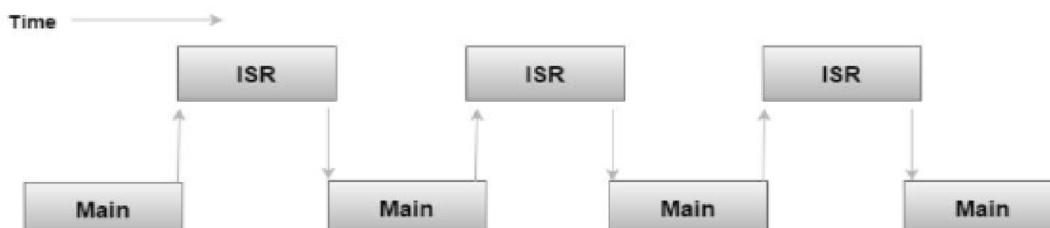
Program Execution without Interrupts

Time



Program Execution with Interrupts

Time



ISR : Interrupt Service Routine

Part 1. B) External Interrupt vs Internal Interrupt?

- **External interrupts**

  External interrupts come from input-output (I/0) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source.

- **Internal interrupts**

  Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps.

- The main difference between internal and external interrupts is that the internal interrupt is initiated by some exceptional condition caused by the program itself rather than by an external event.

External interrupts depend on external conditions that are independent of the program being executed at the time.

Part 1. C) Maskable Interrupt vs Non Maskable Interrupt?

Maskable Interrupt	Non Maskable Interrupt
Maskable interrupt is a hardware Interrupt that can be disabled or ignored by the instructions of CPU.	A non-maskable interrupt is a hardware interrupt that cannot be disabled or ignored by the instructions of CPU.
When maskable interrupt occur, it can be handled after executing the current instruction.	When non-maskable interrupts occur, the current instructions and status are stored in stack for the CPU to handle the interrupt.

- Interrupts are events detected by the MCU which cause normal program flow to be preempted. Interrupts pause the current program and transfer control to a specified user-written firmware routine called the Interrupt Service Routine (ISR). The ISR processes the interrupt event, then resumes normal program flow.

Part 1. D) Overview of Interrupt Process

Step 1. Program MCU to react to interrupts

The MCU must be programmed to enable interrupts to occur. Setting the Global Interrupt Enable (GIE) and, in many cases, the Peripheral Interrupt Enable (PEIE), enables the MCU to receive interrupts. GIE and PEIE are located in the Interrupt Control (INTCON) special function register.

Step 2. Enable interrupts from selected peripherals

Each peripheral on the MCU has an individual enable bit. A peripheral's individual interrupt enable bit must be set, in addition to GIE/PEIE, before the peripheral can generate an interrupt. The individual interrupt enable bits are located in INTCON, PIE1, PIE2, and PIE3.

Step 3. Peripheral asserts an interrupt request

When a peripheral reaches a state where program intervention is needed, the peripheral sets an Interrupt Request Flag (xxIF). These interrupt flags are set regardless of the status of the GIE, PEIE, and individual interrupt enable bits. The interrupt flags are located in INTCON, PIR1, PIR2, and PIR3.

The interrupt request flags are latched high when set and must be cleared by the user-written ISR.

Step 4. Interrupt occurs

When an interrupt request flag is set and the interrupt is properly enabled, the interrupt process begins:

- Global Interrupts are disabled by clearing GIE to 0.
- The current program context is saved to the shadow registers.
- The value of the Program Counter is stored on the return stack.
- Program control is transferred to the interrupt vector at address 04h.

Step 5. ISR runs

The ISR is a function written by the user and placed at address 04h. The ISR does the following:

1. Checks the interrupt-enabled peripherals for the source of the interrupt request.
2. Performs the necessary peripheral tasks.
3. Clears the appropriate interrupt request flag.
4. Executes the Return from Interrupt instruction (RETFIE) as the final ISR instruction.

Step 6. Control is returned to the Main program

When RETFIE is executed:

1. Global Interrupts are enabled (GIE=1).
2. The program context is restored from the Shadow registers.

3. The return address from the stack is loaded into the Program Counter.
4. Execution resumes from point at which it was interrupted.

Part 1. E) How to code Blink LED with Button?

The first rule to code an interrupt is that we need to set the I (bit 7) of the AVR Status register. The I bit is global interrupt enable.

According to datasheet and AVR architecture the Global interrupt bit is a must to be set bit. It should be enabled first and then one can easily enable individual interrupts using separate control registers.

Interrupt in AVR is two-way lock first you need to set the global interrupt bit and then you need to set the individual interrupt bit of that particular peripheral then only you can receive interrupt.

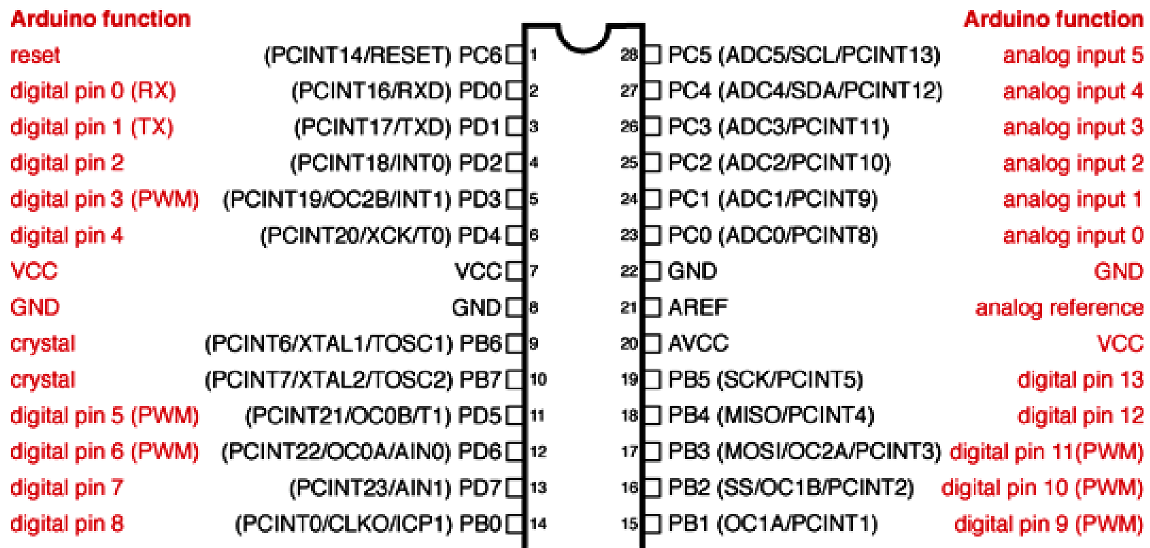
Below is the SREG register that is the status register for whole AVR. It contains all the necessary flags.

SREG – AVR Status Register

The AVR status register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

After setting this we need to see the pin diagram and select the pins which can be used for external interrupt and pin change interrupt.



So, if you have a look at pin diagram you can see that we have digital pin 2 and 3 as the external interrupt pin that is INT1 and INT0. And if you want to program PCINT interrupt then you need to look at the other sets of registers present for pin change interrupt.

External Interrupt configuration

For **External Interrupt** – Now if you go through the Register description of external interrupt you will find various registers that you need to program while writing a code. Registers like EIMSK and EICRA – External Interrupt Control Register would be altered according to requirement.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

EICRA Register

Here EICRA is used to select what type of interrupt you want like- Level triggered or Edge triggered. So you can set particular bit for the type of interrupt you want to be configured.

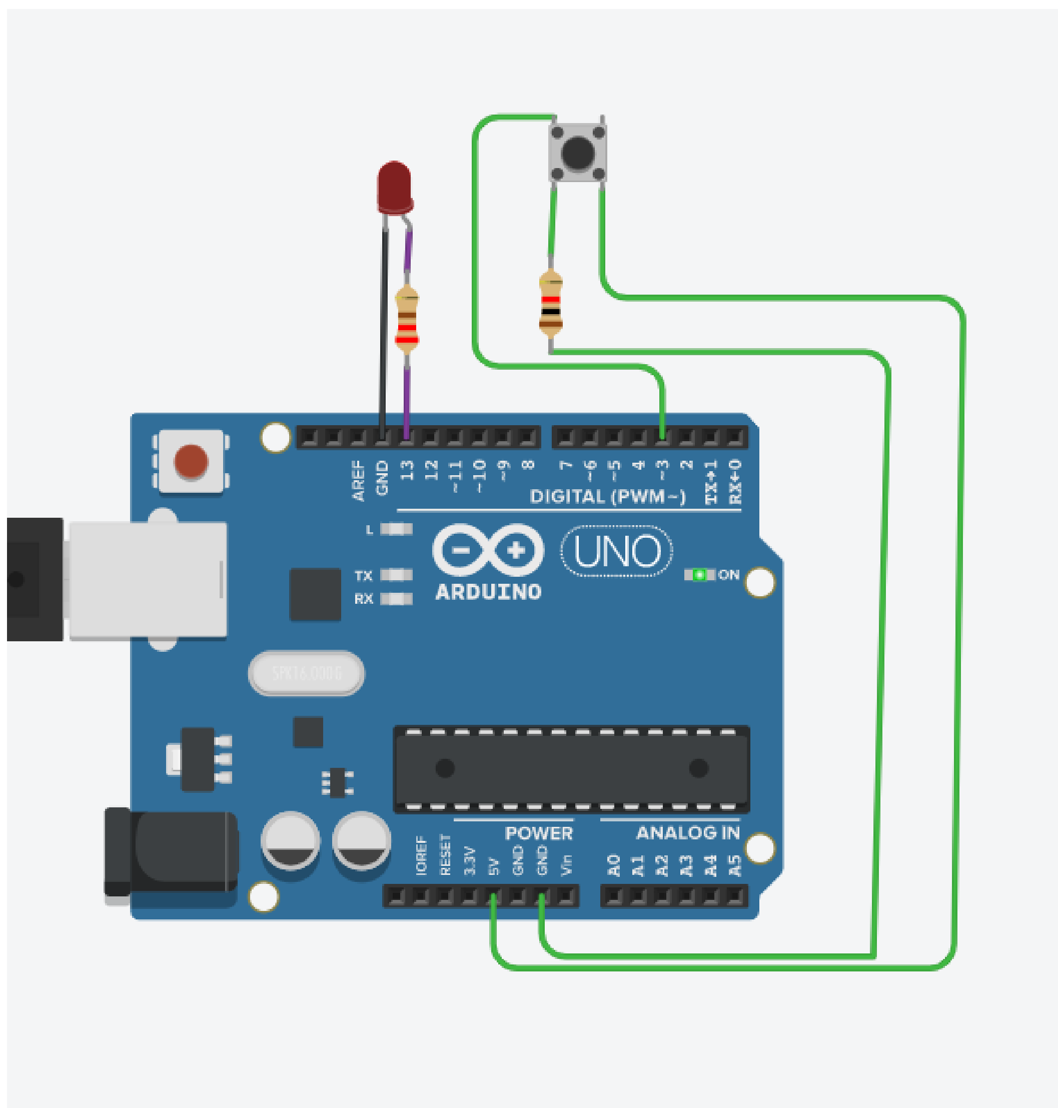
Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

EIMSK Register

Then In EIMSK you can set bit 1 and 2 for the pin from where you have connected your button in the circuit. It would be set according to your connection. If you have connected PD2 then INT0 would be set and if you have connected to PD3 the INT1 would be selected.

Code for Blink LED with Button

Let's take external interrupt from a switch and use that interrupt to blink a led. Now here below you can find the circuit.



So, we have connected the switch to digital pin 3 that is the **INT1** now we need to set the bits of various register accordingly. So as to get the desired result.

- The code for getting the desired result is below.

  THE EXPECTED OUTPUT FROM CODE IS THAT WHENEVER THERE IS AN INTERRUPT FROM SWITCH THE LED SHOULD TURN ON.

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

//These three are set of pre-processors that can be used to
Set, Reset and toogle the bit
 #define SET_BIT(port,bit) port|=(1<<bit)
 #define CLEAR_BIT(port,bit) port&=~(1<<bit)
 #define TOGGLE(PORT,BIT) PORT^=(1<<BIT)

void init();//Declaration of Function

int main()
 {
   init();
  while(1)//infinite loop
   {
    //nothing in loop cause it will work on interrupts
   }
   return 0;
 }

//the ISR-INTERRUPT SERVICE ROUTINE

 ISR(INT1_vect)
 {
   PORTB=~PORTB; //toggling the portB
 }
```

```
//We have created a separate Function for all initialization
so as to make our program look more readable.

 void init()
 {
//GPIO CODE
   DDRB|=(1<<PB5);//LED CONNECTED TO DIGITAL PIN 13 ,VALUE 1
   DDRB&=~(1<<PD3);//SLIDE BUTTON CONNECTED TO PIN 11 AS INPUT
THUS VALUE ZERO

//the two way lock for interrupt is shown here first I //bit
is set then external interrupt bit -EIMSK is set

SREG|=(1<<7); //GLOBAL INTERRUPT

//EXTERNAL INTERRUPT Configuration
 EICRA|=(1<<ISC10); // Setting control register to Any logical
change on INT1 means whenever the logic will change interrupt
will occur
 EIMSK|=(1<<INT1); // Enabling interrupt from INT1
 }
```

For writing Interrupt code in AVR we always need to follow this convention. We need to write ISR (name of Interrupt Vector) and then we can write our Interrupt service routine i.e. the piece of code we want to execute whenever the interrupt occurs. Like in this case we wanted to toggle the state of LED OFF/ON whenever the Button is pushed.

Part 2. Timers / Counters

- The Atmega328P has a total of three timer/counters named Timer/counter 0, Timer/counter 1, and Timer/counter 2.

- The first and last of these are both 8-bit timer/counters and have a maximum value of **255**, while Timer/Counter 1 is 16 bits and its maximum is **65,535**

- In much of what follows, you may see references to TOP, MAX, and/or BOTTOM. These are definitions that are used in the data sheet for the ATmega328P and refer to the following:

  - BOTTOM is easy. It is always zero.

  - MAX is also easy. It is always the maximum value that can be held in the timer/counter's TCNTn register according to however many bits the timer/counter is configured for. This is calculated as 2bits – 1.

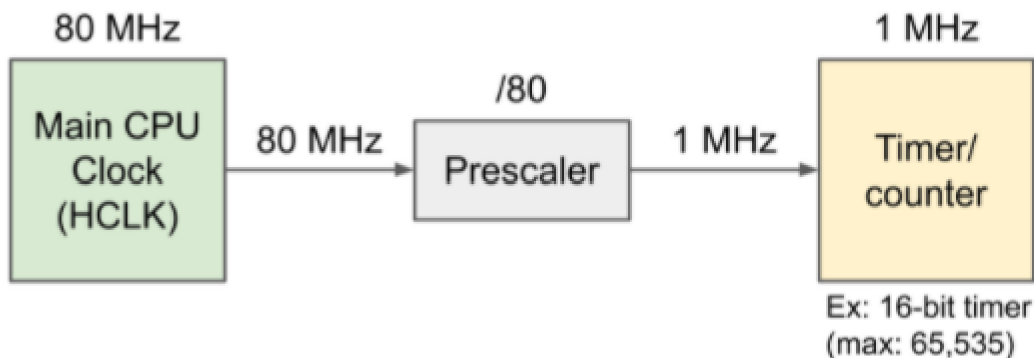
    - For Timer/counters 0 and 2, this is always 8 bits, and so MAX always equals 255. For Timer/counter 1, MAX varies as follows:

    - In 8-bit mode, MAX = 255.
    - In 9-bit mode, MAX = 511.
    - In 10-bit mode, MAX = 1,023.
    - In 16-bit mode, MAX = 65,535.

- TOP depends on the timer/counter's mode and is either MAX for some modes or as defined by various other timer/counter registers such as OCRnA, OCRnB, ICR1, etc.

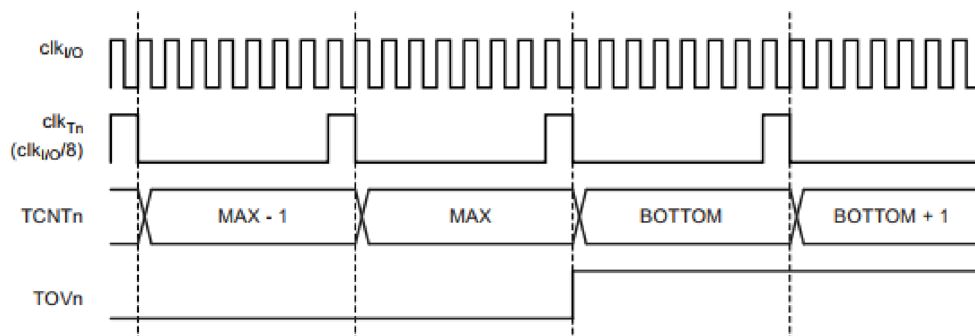
Part 2.A) What is prescaler?

A **prescaler** is an electronic counting circuit used to reduce a high frequency electrical signal to a lower frequency by integer division.
The prescaler takes the basic timer clock frequency (which may be the CPU clock frequency or may be some higher or lower frequency) and divides it by some value before feeding it to the timer, according to how the prescaler register(s) are configured.



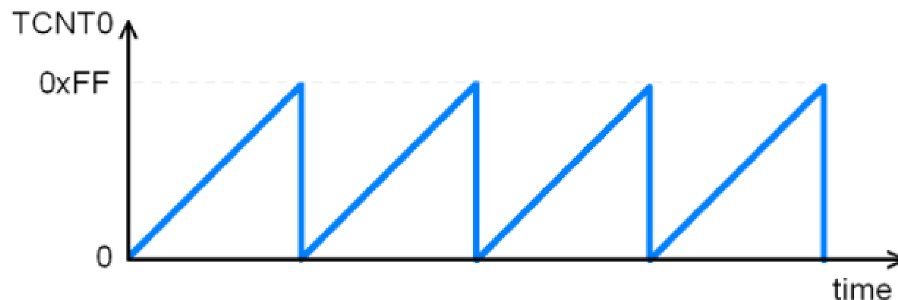
The prescaler values, referred to as prescales, that may be configured might be limited to a few fixed values (powers of 2), or they may be any integer value from 1 to 2^P, where P is the number of prescaler bits.

Figure 14-9. Timer/Counter Timing Diagram, with Prescaler ($f_{clk_I/O}/8$)



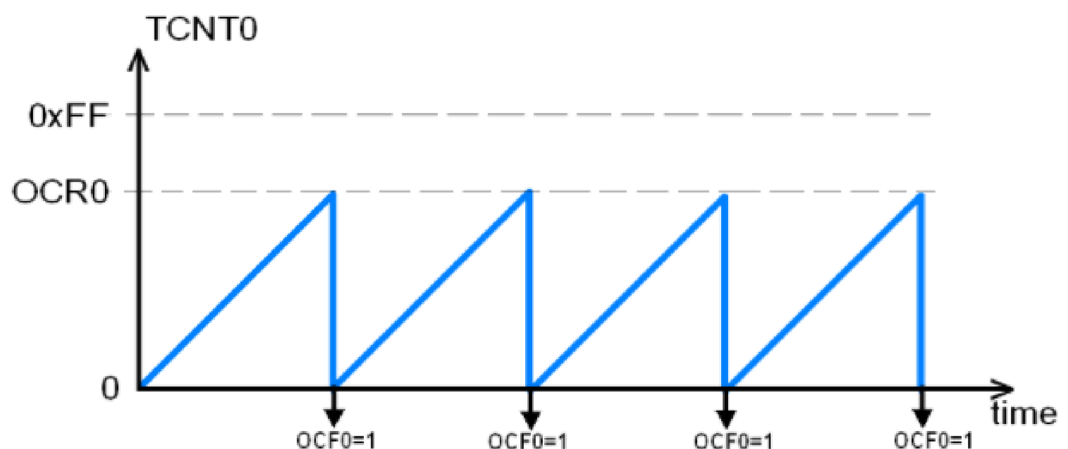
Part 2. B) Modes of Operation

- **Normal mode**: the simplest mode of operation
  In this mode the counting direction is always up (incrementing)
  The counter simply overruns when it passes its maximum 8-bits value (top =0xFF) and restart from bottom (0x00)



- **Clear Timer on Compare Match (CTC) Mode**
  In CTC mode the counter is cleared to zero when the counter value (TCNTX) matches the OCRX

  The OCRX defines the top value for counter hence also its resolution
  Fast PWM



- **Fast PWM Mode**
- **Phase Correct PWM Mode**

Part 2. C) Timers / Counters0

See Datasheet page74

Part 2. D)How to code Blink LED in 1 second using timer 0?

1. Load the TCNT0 register with the initial value (let's take 0x25).

2. For normal mode and the pre-scaler option of the clock, set the value in the TCCR0A register. As soon as the clock Prescaler value gets selected, the timer/counter starts to count, and each clock tick causes the value of the timer/counter to increment by 1.

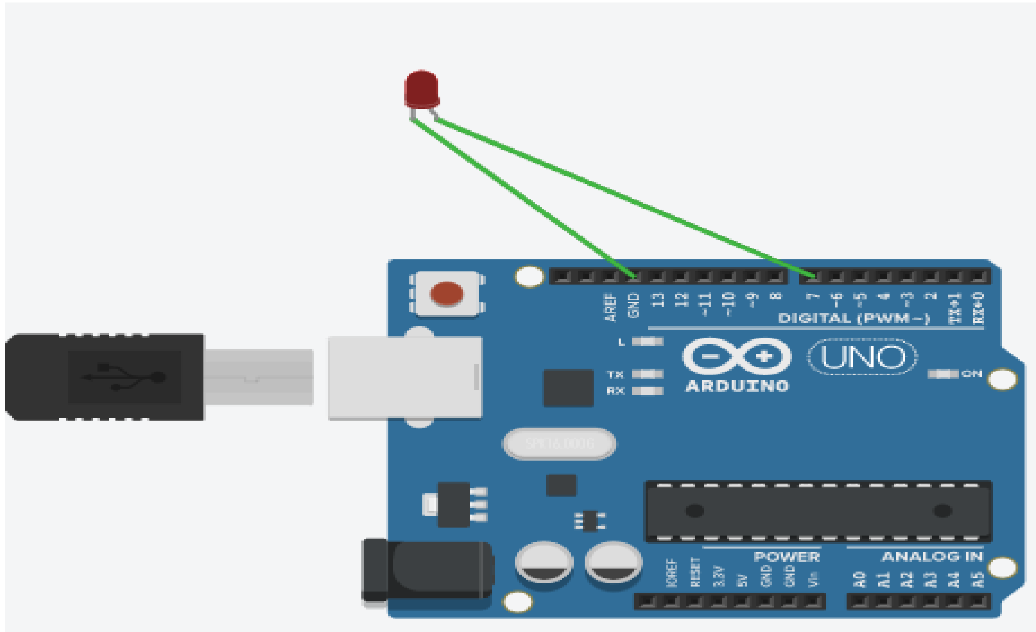
3.Timer keeps counting up, so keep monitoring for timer overflow i.e. TOV0 (Timer0 Overflow) flag to see if it is raised.

4. Stop the timer by putting 0 in the TCCR0 i.e. the clock source will get disconnected and the timer/counter will get stopped.

5. Clear the TOV0 flag. Note that we have to write 1 to the TOV0 bit to clear the flag.

6. Return to the main function.

Hardware Connection




```
/*
 * timer_application.c
 *
 * Created: 4/22/2022 9:28:25 PM
 * Author : Miada_Pc
 */

#include <stdint.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#define toggle_BIT(PORT,BIT) PORT ^= (1<<BIT) //this will flip
the bit
#define SET_BIT(PORT,BIT) PORT |= (1<<BIT)
#define CLR_BIT(PORT,BIT) PORT &= ~(1<<BIT)
volatile unsigned int count = 0;

void timer()
{
	TCCR0A = 0x00; // Normal mode of operation
	SET_BIT(TCCR0B,0);  // no prescaler
	TIMSK0 |= 1<<TOIE0; // Enable timer0 overflow interrupt
	sei(); // Global interrupt
	SET_BIT(EIMSK,1);
}
```

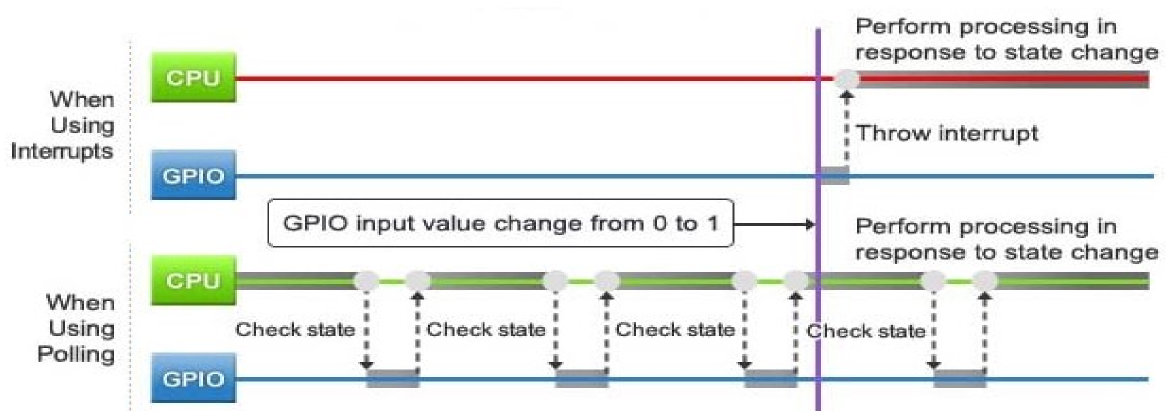
```
int main(void)
{
	SET_BIT(DDRD,PD7);
	timer();
	while(1)
	{

	}
	return 0;
}

ISR(TIMER0_OVF_vect)
{
	count++;
	if(count==31250)
	{
		toggle_BIT(PORTD,PD7);
		count=0;

	}

}
```

Part 2. E) Difference Between Interrupt and Polling

Polling: the mechanism that indicates the CPU that a device requires its attention. It is a continuous act to figure out whether the device is working properly.

Mainly, polling causes the wastage of many CPU cycles. Especially, if there are many devices to check, then the time taken to poll them could exceed the time available to service the I/O device.

❖ Interrupt VS Polling



- **Definition**

  An interrupt is an event that is triggered by external components other than the CPU that alerts the CPU to perform a certain action. In contrast, polling is a synchronous activity that samples the status of an external device by a client program. Thus, this describes the main difference between interrupt and polling.

- **Result**

  When an interrupt occurs, the interrupt handler is executed. On the other hand, in polling, the CPU provides the service.

- **Occurrence**

  Another difference between interrupt and polling is that interrupt can occur at any time while polling occurs at regular intervals.

- **Indication**

  Moreover, the interrupt-request line indicates that a device needs a service whereas command-ready bit indicates a device needs service.

- **CPU cycles**

  Interrupt does not waste many CPU cycles while polling wastes a lot of CPU cycles. Hence, this is also a difference between interrupt and polling.

- **Efficiency**

  Furthermore, in the case of interrupt, it is inefficient when the devices interrupt the CPU frequently. In contrast, polling is inefficient, when the CPU does not get much requests from the devices.

➢ The main difference between interrupt and polling is that in interrupt, the device notifies the CPU that it requires attention while, in polling, the CPU continuously checks the

status of the devices to find whether they require attention. In brief, an interrupt is asynchronous whereas polling is synchronous.

Part 2. F) What's The difference between timers and delay in embedded programming?

A delay is a operation which holds the execution of the current thread so the next operation that is in line will have to wait until the delay operation is over.

A timer on the other hand is not blocking your normal code. It runs in the other thread parallel to the execution on your normal code and only if the timer end is reached it will jump to the code that you provided for your timer end and after that code is complete it will jump back to the main program execution. Counter is just a normal variable that is incremented if you tell it to and it can be done either in normal variable in the code or special register that will be storing your counter.

- So the main target here to understand is that delay is blocking code execution and timer don’t. Counter is just counting and it can either be blocking or not.