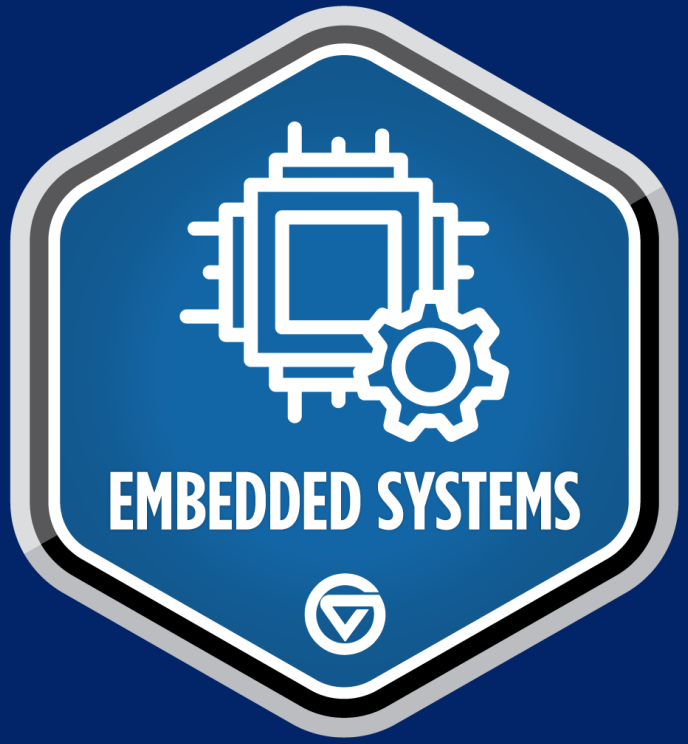
Today session Objective

D-BUGERZ

Recap on External Interrupts, And know how to use Timer peripheral then will discuss project description



Timer Peripheral

D-BUGERZ

- Recall Last session
- Exercise on Interrupts
- Introduction to Timers
- Start Coding with Timers
- Final Project Description

Code Techniques

D-BUGERZ

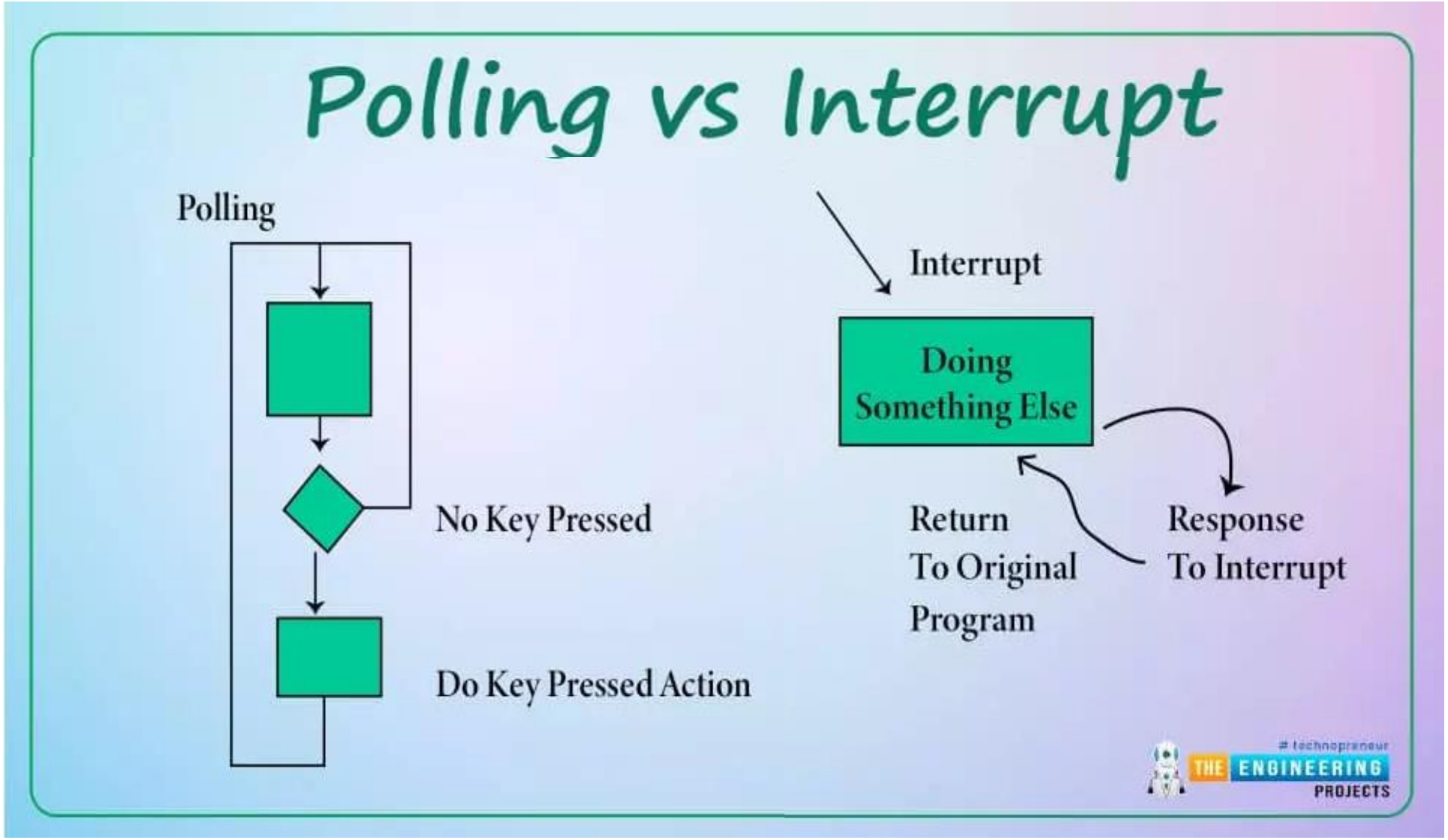
1.

2.

3.

4.

5.

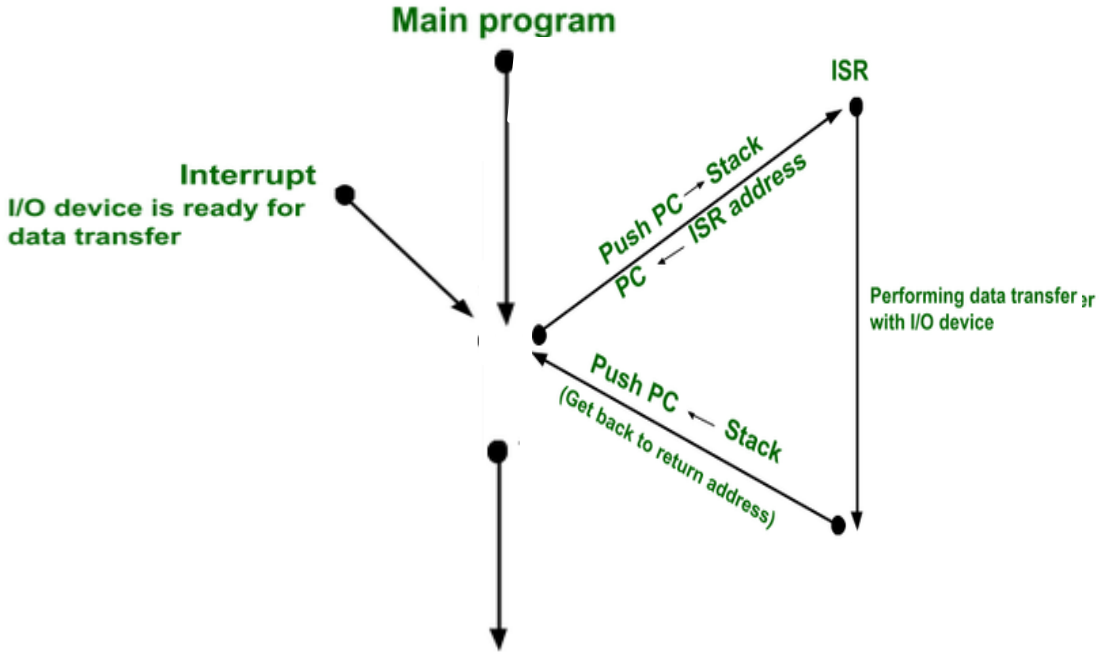


Interrupt Overview

- Interrupts are typically generated in most cases by hardware, for example peripherals or external input pins. When a peripheral or hardware needs service from the processor, this will lead to changes in program flow control outside the normal programmed sequence. Typically, the following sequences would occur

1.
2.
3.
4.
5.

1. The peripheral asserts an interrupt request to the processor.
2. The currently running task is suspended by the processor
3. The processor executes an Interrupt Service Routine (ISR) to service the peripheral
4. The processor resumes the previously suspended task.



Interrupt Overview

D-BUGERZ

Interrupt Service Routine (ISR) or Interrupt Handler

1. ❑ Piece of code that should be execute when an interrupt is triggered.

2. ❑ For every interrupt there must be a program associated with it. When an interrupt occurs this program is executed to perform certain service for the interrupt.

3. ❑ This program is commonly referred to as an Interrupt Service Routine (ISR) or Interrupt Handler When an interrupt occurs, the CPU runs the ISR. (No need to call this ISR)

4.

5.

```
8  #include <avr/io.h>
9  #include <avr/interrupt.h>
10
11 /* External INT0 Interrupt Service Routine */
12 ISR(INT0_vect)
13 {
14     PORTC = PORTC ^ (1<<PC0); //Toggle value of PIN 0 in PORTC (Led Toggle)
15 }
16
17
18 int main(void)
19 {
20     INT0_Init();              // Enable external INT0
21
22     DDRC  = DDRC | (1<<PC0);  // Configure pin PC0 in PORTC as output pin
23     PORTC = PORTC | (1<<PC0); // Set Value of PIN 0 in PORTC to 1 at the beginning
24
25     while(1)
26     {
27
28     }
29 }
30
```

Interrupt Overview

D-BUGERZ

- **Interrupt Vector Table for Atmega32**

→ Each interrupt has specific location in the interrupt vector table for it's ISR

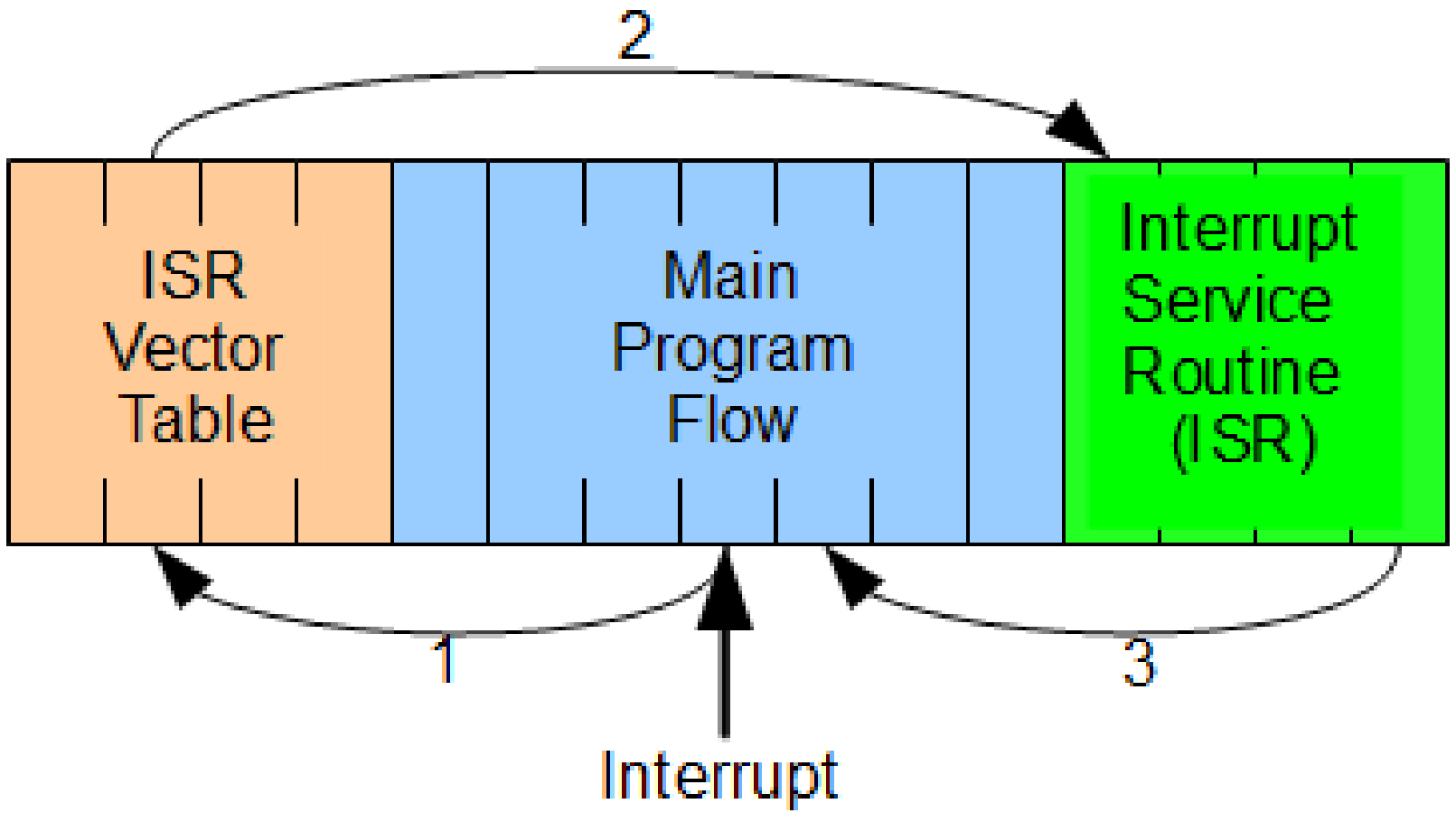
1.
2.
3.
4.
5.

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

Interrupt Overview

D-BUGERZ

1.
2.
3.
4.
5.



How To Trigger Interrupt Request

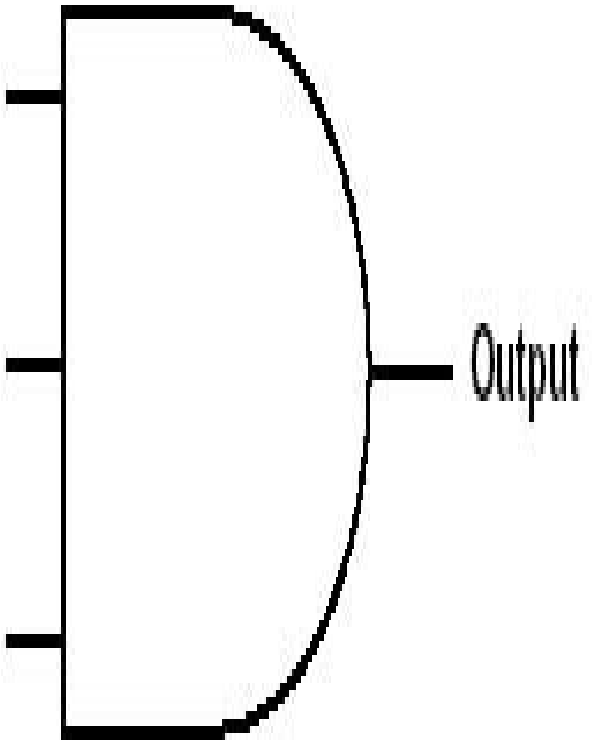
D-BUGERZ

- Each peripheral device has a group of special function registers that must be used to access the device for configuration. For a given peripheral interrupt to take effect, the interrupt for that peripheral must be enabled. The special function registers for that device provide the way to enable the interrupts.

1. In case disable all the interrupts is required just **clear** the **Global Interrupt Enable (GIE).** All Maskable interrupts are disabled whatever the MIE value was 1 or 0.

2. Each Maskable interrupt in the MC has a specific bit called **Module Interrupt Enable (MIE).** It is used to enable or disable the interrupt for this module or peripheral.

3. Also each Maskable interrupt in the microcontroller has a specific bit **called Module Interrupt flag (MIF).** This bit is set whenever the interrupt event occur. Whether or not the interrupt is enabled. This flag is also called Automatic flag as it is set automatically by hardware.



How To Trigger Interrupt Request

D-BUGERZ

1. ❑ **Global Interrupt Enable (GIE).**

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
	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	

2. • **Bit 7 – I: Global Interrupt Enable**

3. ❑ **Module Interrupt Enable (MIE)**

General Interrupt Control Register – GICR

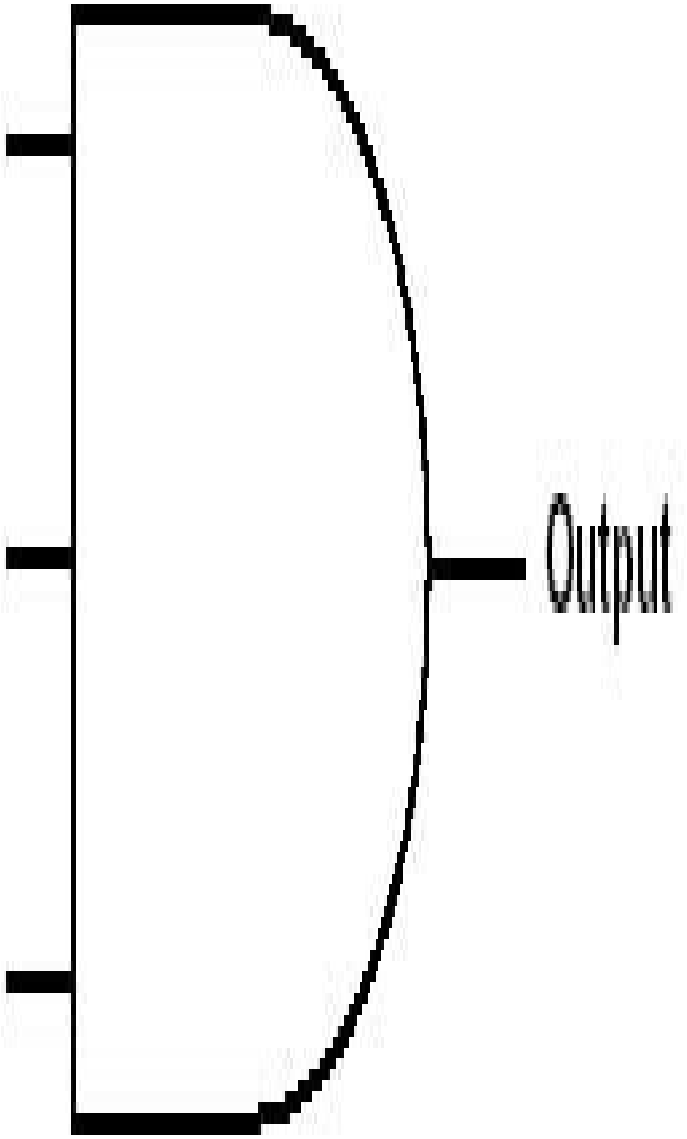
Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

4. → **There are 2nd step here to configure when you want to receive interrupt**

5. ❑ **Module Interrupt flag (MIF).**

General Interrupt Flag Register – GIFR

Bit	7	6	5	4	3	2	1	0	
	INTF1	INTF0	INTF2	–	–	–	–	–	GIFR
Read/Write	R/W	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	



Registers of External Interrupt

D-BUGERZ

- Page 67 in Datasheet

MCU Control Register – MCUCR

The MCU Control Register contains control bits for interrupt sense control and general MCU functions.

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
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	

1.

2.

3.

4.

5.

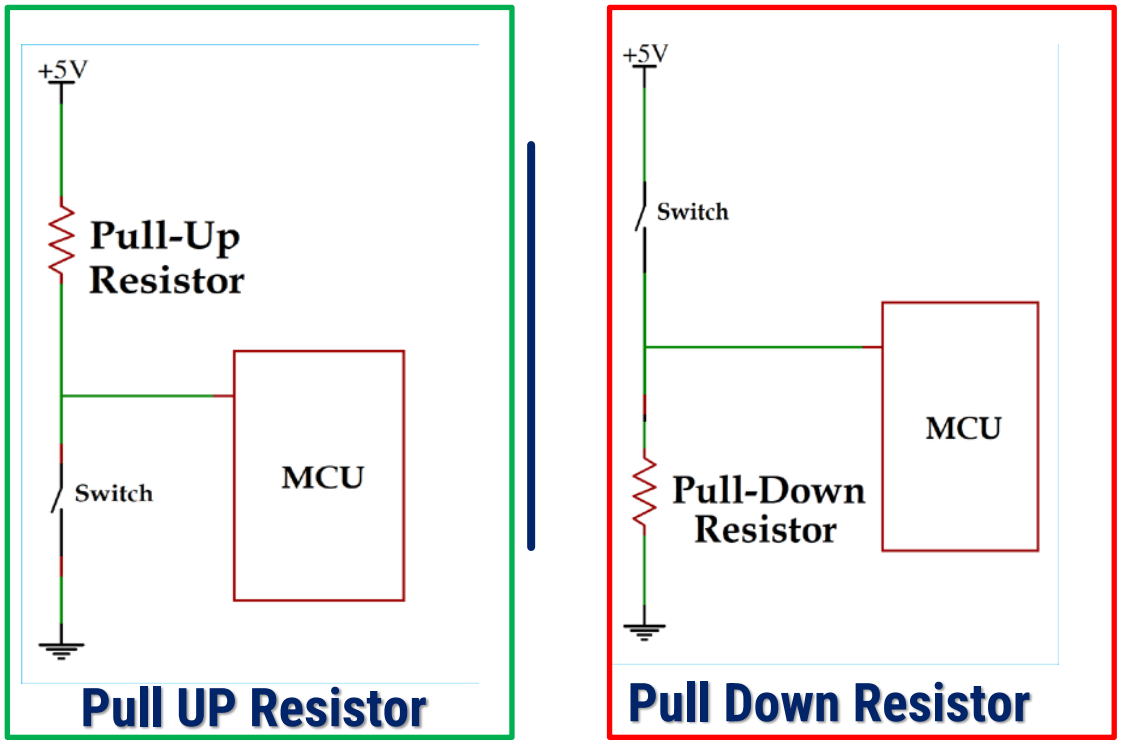
ISC01	ISC00		Description
0	0		The low level on INT0 pin generates an interrupt request.
0	1		Any logical change on INT0 pin generates an interrupt request.
1	0		The falling edge on INT0 pin generates an interrupt request.
1	1		The rising edge on INT0 pin generates an interrupt request.

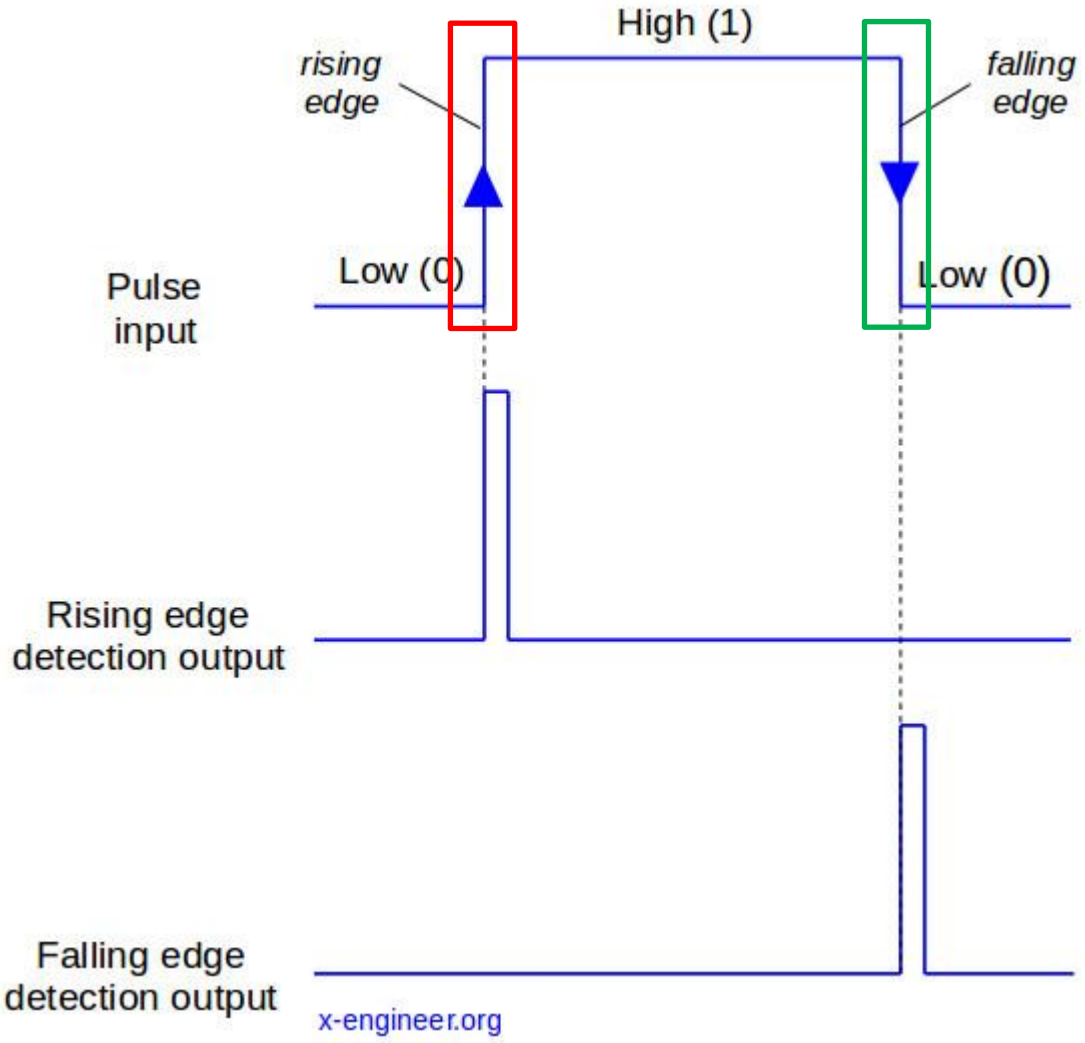
Registers of External Interrupt

D-BUGERZ

1.
2.
3.
4.
5.

Switches(Push button, Keypad etc.)





D-BUGERZ

1.

2.

3.

4.

5.

D-BUGERZ

Recall Last session

Exercise on Interrupts

Introduction to Timers

Start Coding with Timers

Final Project Description

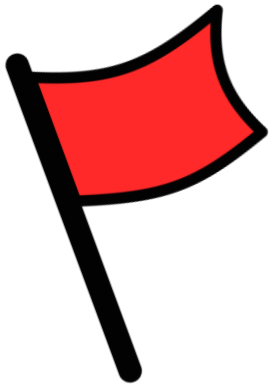
Interrupt Coding Notes

D-BUGERZ

- 0xF8 = 1111 1000
- 0x07 = 0000 0111
- We don't need to write long code in ISR so what we do?

1.
2.
3.
4.
5.

```
unsigned char g_Interrupt_Flag = 0;

/* External INT1 Interrupt Service Routine */
ISR(INT1_vect)
{
    // set the interrupt flag to indicate that INT1 occur
    g_Interrupt_Flag = 1;
}
```



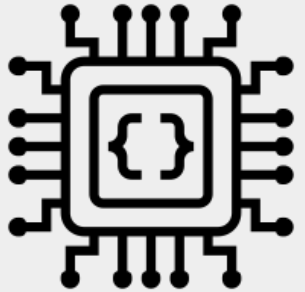
```
while(1)
{
    if(g_Interrupt_Flag == 0)
    {

    }
    else if(g_Interrupt_Flag == 1)
    {
```

Exercise 2 - Interrupts

D-BUGERZ

1.
2.
3.
4.
5.

Write Embedded C code using ATmega16/32 µC to control 3-LEDs by external interrupt INT1



Requirements:

1- Configure the µC clock with 16Mhz Crystal Oscillator.
2- Use the 3 LEDs at PC0, PC1 and PC2.
3- LEDs are connected using Positive Logic configuration.
4- A roll action is perform using the LEDs each led for half second. The first LED is lit and roll down to the last LED then back to the first LED. This operation is done continuously.
5- When the INT1 is triggered all three LEDs flash five times for five seconds.

Challenge 4

D-BUGERZ

Write Embedded C code using ATmega16/32 µC to control a 7-segment using a INT2

Requirements:

1.

- ✓ **Configure the µC clock with internal 1Mhz Clock.**

2.

- ✓ **The 7-segment is connected to first 4-pins of PORTC.**

3.

- ✓ **The 7-Segment type is common anode.**

4.

- ✓ **Connect the switch using Pull down configuration on INT2/PB2 pin.**

5.

- ✓ **When the INT2 is triggered just increase the number appeared in the 7-Segment display, and if the 7-segment reaches the maximum number (9) overflow occurs.**

D-BUGERZ

1.

2.

3.

4.

5.

6.

BREAK 10 MIN

D-BUGERZ

Recall Last session

Exercise on Interrupts

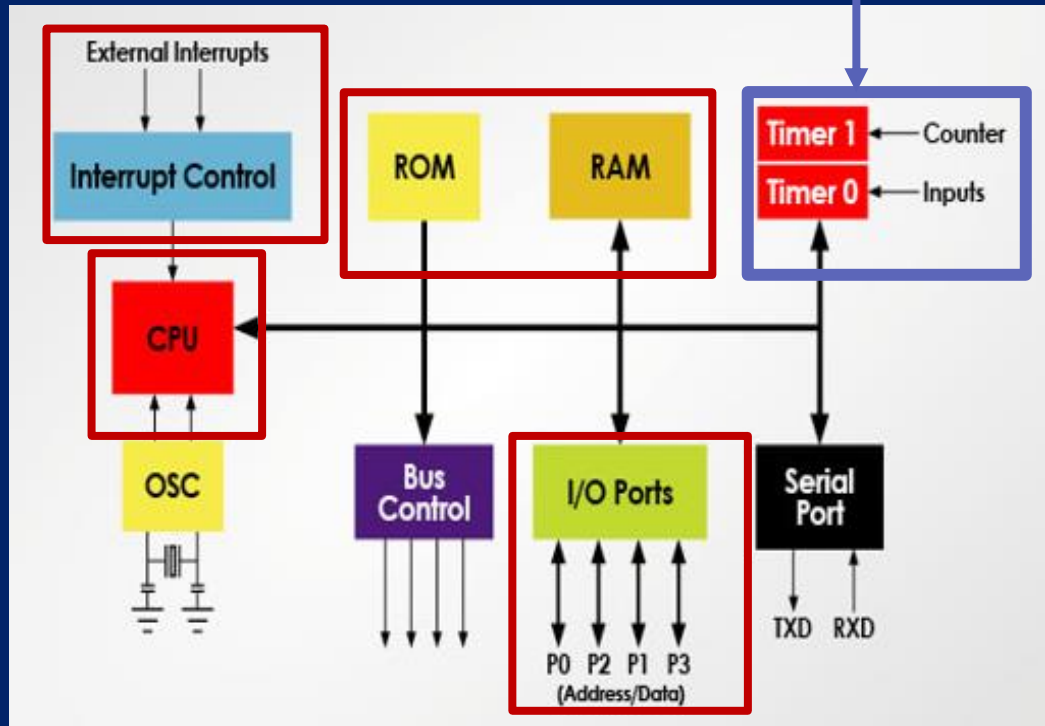
Introduction to Timers

Start Coding with Timers

Final Project Description

1. Central Processing Unit (CPU)
2. Memory Units
3. Input and Output Ports (GPIO or DIO)
4. Interrupt Unit
5. Timers

Today Session



Introduction to Timers

D-BUGERZ

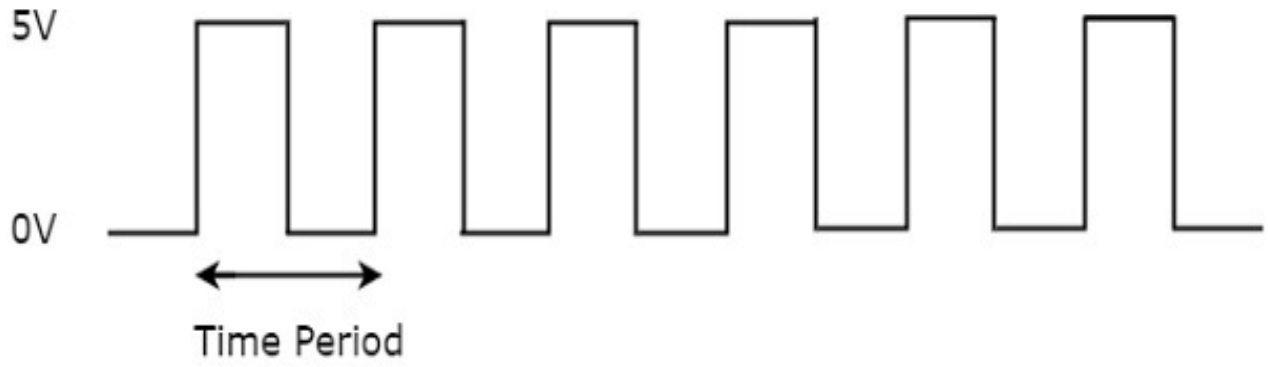
1. Clock signal is a particular type of signal that oscillates between a high and a low state, Circuits use the clock signal for **synchronization** may become active at either the rising edge, falling edge, or, in the case of double data rate, both in the rising and in the falling edges of the clock cycle.

2. Microcontroller clock generator source could be internal or external.

μC Clock

3. **Any Microcontroller have it's Clock and you can modify this number to your needs as programmer and this clock is the main parameter of controller speed you can modify by 1 MHZ , 8 MHZ , 12 MHZ , 16MHZ**

4. **Clock is the synchronization of μC and it make all peripherals in μC have same reference**



5.

Introduction to Timers

D-BUGERZ

1. ❑ Timers are **standard features of almost every microcontroller**. So it is very important to learn their use.

2. ❑ Generally, we use timer/counter to **generate time delays**, **waveforms** or to count events. Also, timer is used for **PWM generation**, capturing events etc·

3. ❑ A **timer in simplest term is a register !** But not a normal one. Timers generally have are solution of 8 or 16 or 32 or 64 bits.

4. ❑ 8 bits Timer for example is 8-bits wide, it is capable of counting 2^8 = 255 steps from 0 to 255

5. ❑ But this register has a magical property! Its value increases/decreases automatically at a predefined rate (supplied by user). This is the timer clock. And this operation does not need CPU's intervention.

❑ Similarly a 16 bits Timer is capable of counting 2^{16} = 65536 steps from 0 to 65535.

Introduction to Timers

D-BUGERZ

- A **timer in simplest term is a register !** But not a normal one. Timers generally have are solution of 8 or 16 or 32 or 64 bits.
- 8 bits Timer for example is 8-bits wide, it is capable of counting 2^8 = 255 steps from 0 to 255

1.
2.
3.
4.
5.

Binary **Decimal**

MSB **LSB**

00000000 = 000

2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0 10^2 10^1 10^0

0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 = 0 + 0 + 0

D-BUGERZ

1.

2.

3.

4.

5.

Introduction to Timers

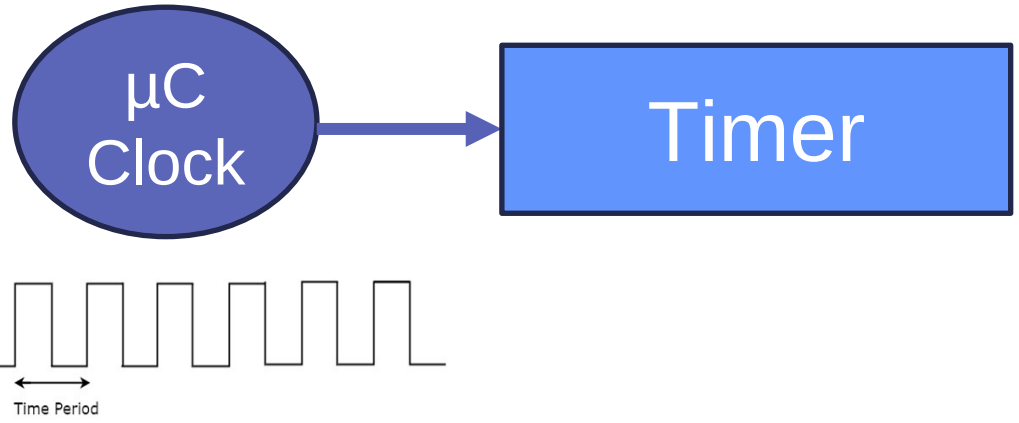
D-BUGERZ

1. Timers/counters are an independent unit inside a micro-controller. It runs parallel to the CPU and they basically run independently of what task CPU is performing.

2. Timers could be **count up/down** each clock.

3. All Timers in AVR Microcontrollers are count up timers.

4. Timers is an important concept in the field of electronics and embedded applications.



5.

- Count Down Timer : 01:00
- Count Up Timer : 00:00

Introduction to Timers

D-BUGERZ

- The Prescaler is used to divide the clock frequency and produce a clock for TIMER
- The Prescaler is used when it is necessary to measure longer periods of time.

1.
2.
3.
4.
5.

Introduction to Timers

D-BUGERZ

- The Prescaler is used to divide the clock frequency and produce a clock for TIMER
- For example in AVR Microcontrollers Timers. The Prescaler can be used to get the following clock for timer:

1. No Clock (Timer Stop) → **If you want Timer to stop Count**
2. Clock = F_CPU (No Prescaler)
3. Clock = F CPU/8
4. Clock = F CPU/64
5. Clock = F CPU/256
   Clock = F CPU/1024

- We **do not reduce the actual F_CPU** used by AVR processor. The actual F_CPU remains the same

D-BUGERZ

1.

2.

3.

4.

5.

Introduction to Timers

D-BUGERZ

1.
2.
3.
4.
5.

- If MCU is clocked at **1MHz** **without prescaler** counter fills up to 255 value and get back very **fast-in 256µs**, while **with prescaler 1024** it will **fill up in 0.26s**. It is easy to calculate these values if you know what timer resolution is. Simply speaking - resolution is a smallest time period of one timer count. It can be easily calculated by using simple formula

$$\textbf{Resolution} = \frac{1}{\textbf{Frequency}}$$

- So if microcontroller is clocked with 1MHz source, then 8 bit timer without prescaler will run with resolution:

$$\textbf{Resolution} = \frac{1}{\textbf{1MHz}} = \textbf{1µs}$$

- So if timer counts 256 ticks (8-bits timer) until overflow then it takes:

T= Resolution * Ticks = 1µs * 256 = 256µs

- **If we receive interrupt each overflow so we receive 1/ 256µs = 3902 interrupt to reach 1 second**

Introduction to Timers

D-BUGERZ

- If we use 1024 prescaler we get

$$\text{Frequency enter to Timer} = \frac{\text{Frequency}}{\text{Prescale}} = \frac{1\ \text{MHZ}}{1024} = 1\ KHZ$$

- So Resolution in Prescalar Case :

$$\text{Resolution} = \frac{1}{\text{Frequency}}$$

$$\text{Resolution} = \frac{1}{1\text{khz}} = 1\ \text{msec}$$

- Then to count up to overflow takes

T = Resolution * Ticks =1 msec * 256 = 256 msec

- **So if we make interrupt each overflow 1/256 ms = 4 so we will reach 1 sec after 4 interrupts**

Introduction to Timers

- 8 bits Timer for example is 8-bits wide, it is capable of counting 2^8 = 255 steps from 0 to 255
- Resolution = 1µs

1.
2.
3.
4.
5.

Binary **Decimal**

MSB **LSB**

00000000 = 000

2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0 10^2 10^1 10^0

0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 = 0 + 0 + 0

D-BUGERZ

1.

2.

3.

4.

5.

Atmega32 Timers

D-BUGERZ

1.
2.
3.
4.
5.

- In ATmega16/32, there are **three timers** each one with separate prescaler:
  - Timer 0 : **8** bit timer.
  - Timer 1 : **16** bit timer.
  - Timer 2 : **8** bit timer.

  - **If timer size increase so can counts more so give less interrupts**

- These three timers can be either operated in:
  1- Normal mode (Overflow).
  2- Clear Timer on Compare mode (CTC) or Compare Match mode
  3- Pulse Width Modulation mode (PWM).

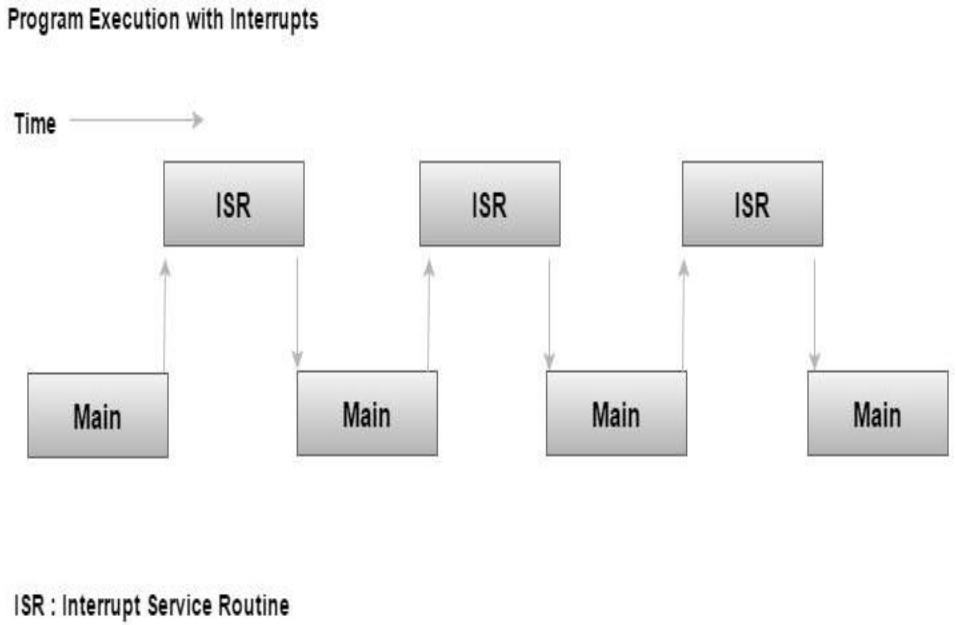
Atmega32 Timers

D-BUGERZ

- We will use Timer0 as example of Timer different usages in this slides
- Timer0 is an 8-bits timer/counter which can count from 0 to 0xFF(255)
- The timer can be operated either in the **polling mode** or **in the interrupt mode**

1. 2. 3. 4. 5.

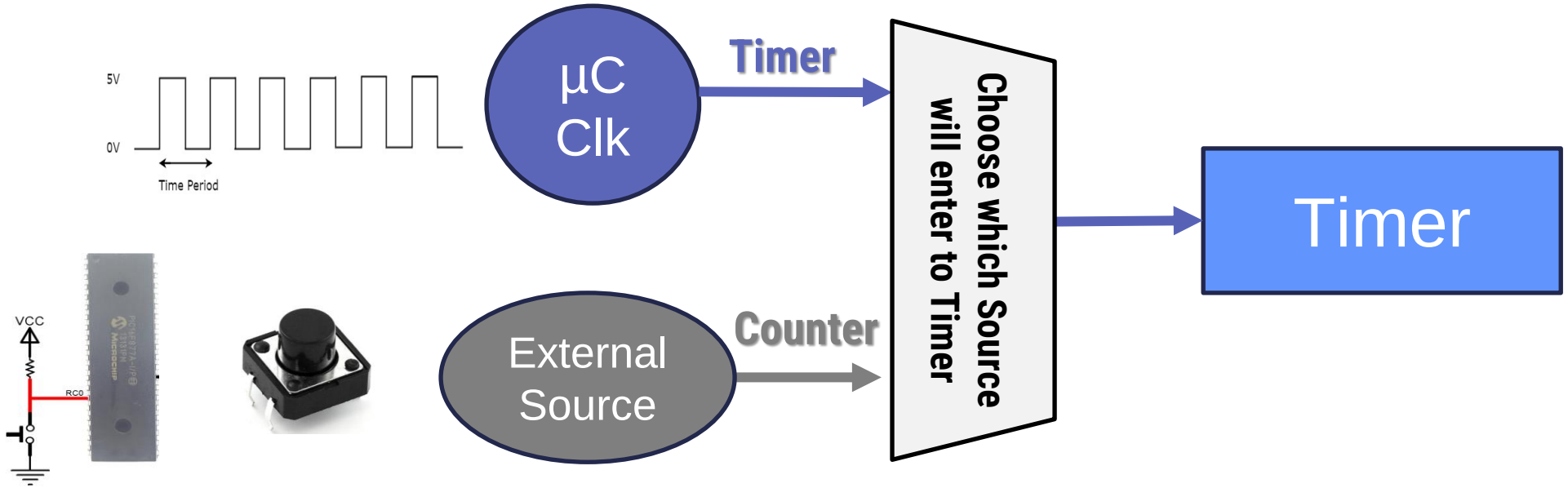
polling mode



interrupt mode

Atmega32 Timers

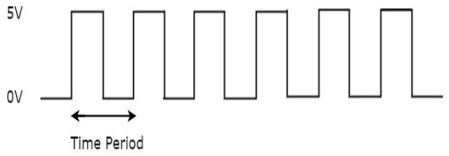
- Timer0 is an 8-bits timer/counter which can count from 0 to 0xFF(255)
- Timer clock could be **internal** or **external** as we will see in datasheet.
- In the timer mode this peripheral uses an internal clock signal
- In the counter mode an external signal on PINO(TO) in PORTB



Atmega32 Timers

D-BUGERZ

- Timers/Counter is the same hardware unit inside the MCU that is used either as Timers or Counter they come in very handy, and are primarily **used for the following**:



1. ❑ **Internal Timer**

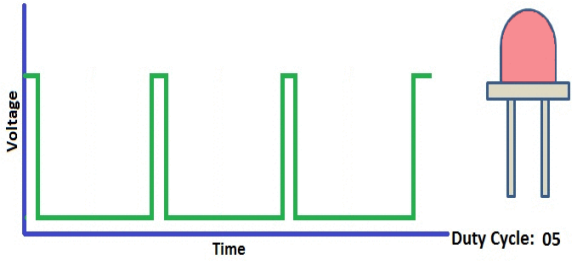
2. As an internal timer the unit, ticks on the oscillator frequency. The oscillator frequency can be directly feed to the timer or it can be pre-scaled. In this mode it used generate precise delays. Or as precise time counting machine

3. ❑ **External Counter**

4. In this mode the unit is used to count events on a specific external pin on a MCU

5. ❑ **Pulse width Modulation (PWM) Generator**

PWM is used in speed control of motors and various other applications



D-BUGERZ

1.

2.

3.

4.

5.

Timer Overflow (Normal) Mode

D-BUGERZ

- In overflow mode the counter start counts from **certain initial** value **until it reaches its maximum value** and then start counting from the beginning again.

- The resolution of the timer determines the maximum value of that timer for example Timer0 is 8-bits timer courts from 0 to 255.

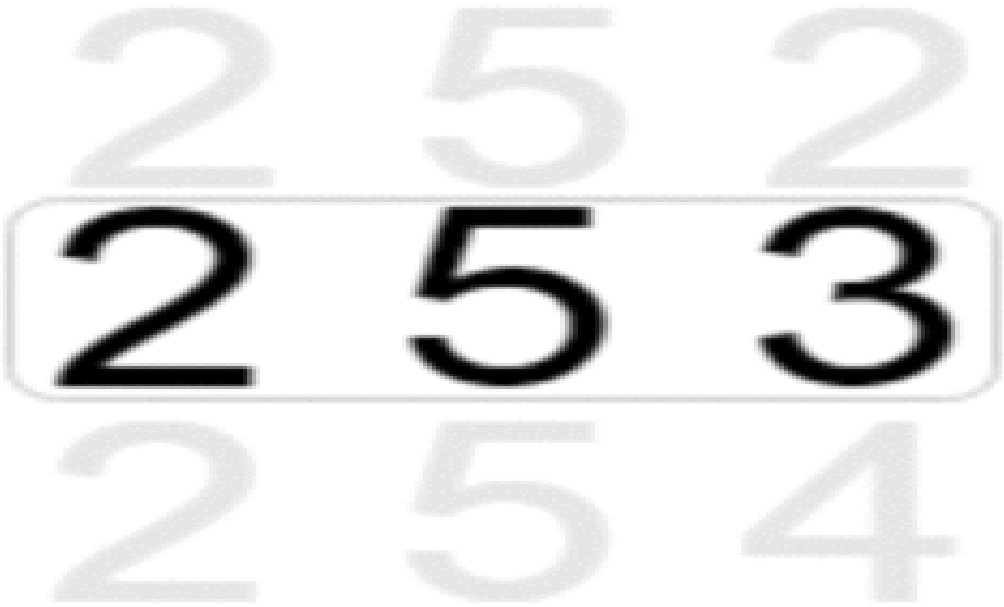
1.

2.

3.

4.

5.



Timer Overflow (Normal) Mode

D-BUGERZ

- In overflow mode the counter start counts from **certain initial** value **until it reaches its maximum value** and then start counting from the beginning again.

- The resolution of the timer determines the maximum value of that timer for example Timer0 is 8-bits timer courts from 0 to 255.

1.

2.

3.

4.

5.

Binary **Decimal**

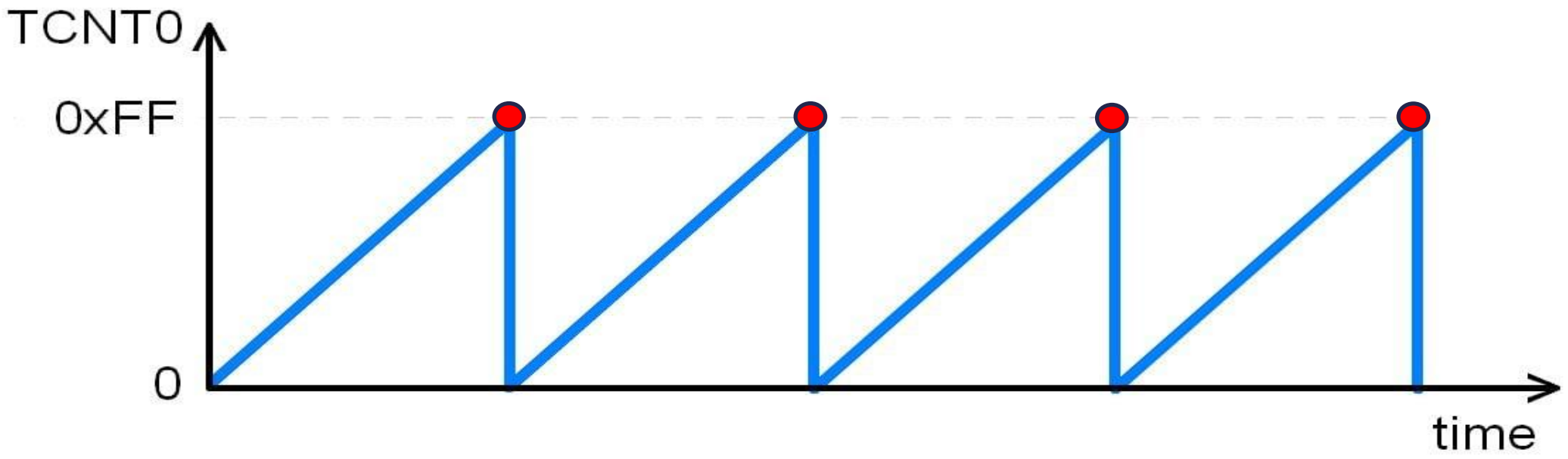
MSB **LSB**

00000000 = 000

Timer Overflow (Normal) Mode

D-BUGERZ

1.
2.
3.
4.
5.

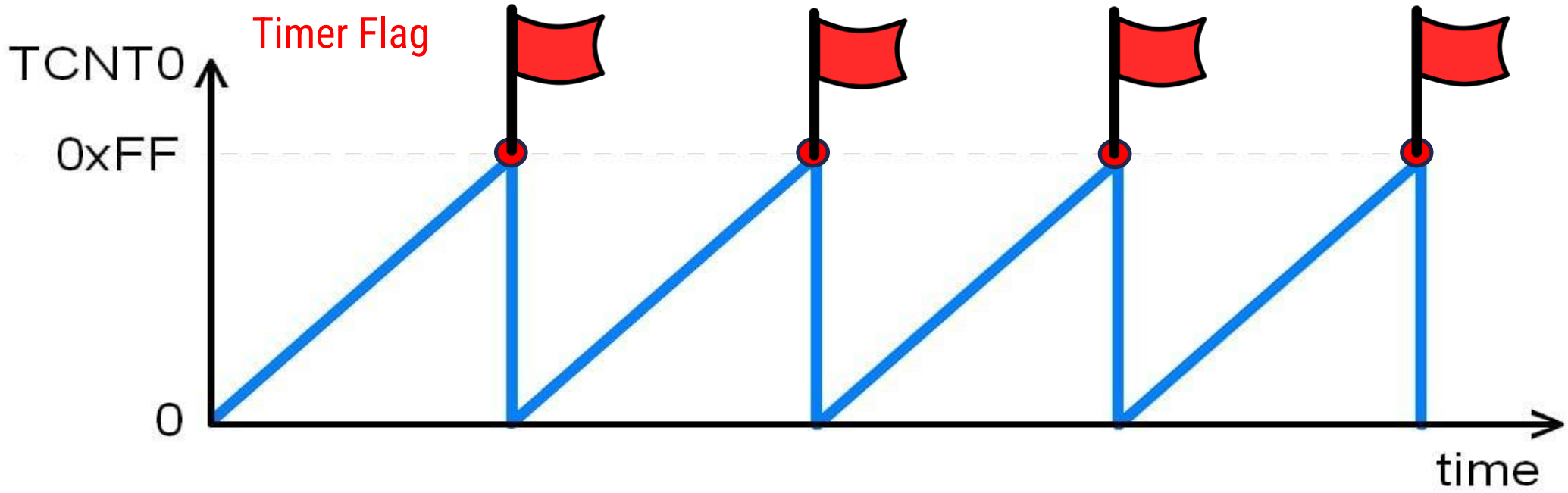


Timer Overflow (Normal) Mode

D-BUGERZ

- Timer0 is a 8 bit timer. It basically means it can count from **0 to 255**. The operation of timer is straight forward. The **TCNT0 register hold the timer count** and it is incremented on every timer clock "tick". If the timer has counted up to its maximum value which is 255 and is reset to zero in the next timer clock cycle then overflow occurs

- The Timer Overflow event causes Timer0 Overflow Flag (TOV0) to be set in the Timer Interrupt Flag Register (TIFR) and The Timer Counter Register TCNT0 will start count again from ZERO



1.
2.
3.
4.
5.

D-BUGERZ

1.

2.

3.

4.

5.

D-BUGERZ

Recall Last session

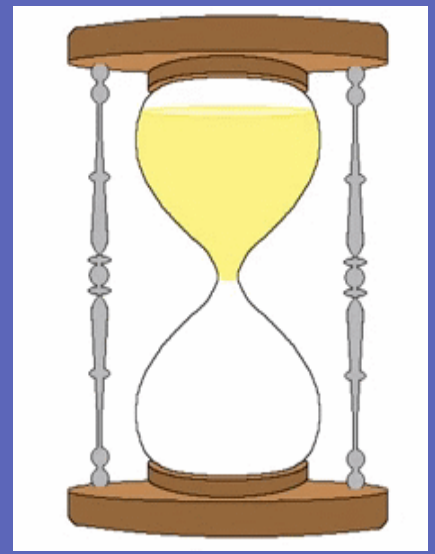
Exercise on Interrupts

Introduction to Timers

Start Coding with Timers

Final Project Description

Let’s Start with Timers



Timers Register (Timer0)

D-BUGERZ

1.
2.
3.
4.
5.

- Page 80 Datasheet

Timer/Counter Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	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	

- **Bit 7 – FOC0: Force Output Compare**

The FOC0 bit is only active when the WGM00 bit specifies a non-PWM mode.

- **Bit 6, 3 – WGM01:0: Waveform Generation Mode**

These bits control the counting sequence of the counter, the source for the maximum (TOP) counter value, and what type of Waveform Generation to be used

Table 38. Waveform Generation Mode Bit Description(1)

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

Timers Register (Timer0)

D-BUGERZ

- Page 80 Datasheet

Timer/Counter Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	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	

1.

2.

3.

4.

5.

- **Bit 5:4 – COM01:0: Compare Match Output Mode**

These bits control the Output Compare pin (OC0) behavior

Table 39. Compare Output Mode, non-PWM Mode

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

Timers Register (Timer0)

D-BUGERZ

- Page 80 Datasheet

Timer/Counter Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	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	

1.

2.

3.

4.

5.

- **Bit 5:4 – COM01:0: Compare Match Output Mode**

These bits control the Output Compare pin (OC0) behavior

Table 40. Compare Output Mode, Fast PWM Mode(1)

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at BOTTOM, (nin-inverting mode)
1	1	Set OC0 on compare match, clear OC0 at BOTTOM, (inverting mode)

Timers Register (Timer0)

D-BUGERZ

- Page 80 Datasheet

Timer/Counter Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	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	

1.

2.

3.

4.

5.

- **Bit 2:0 – CS02:0: Clock Select**

The three Clock Select bits select the clock source to be used by the Timer/Counter.

Table 42. Clock Select Bit Description

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.

Timers Register (Timer0)

1.
2.
3.
4.
5.

- Page 82 Datasheet

Timer/Counter Register – TCNT0

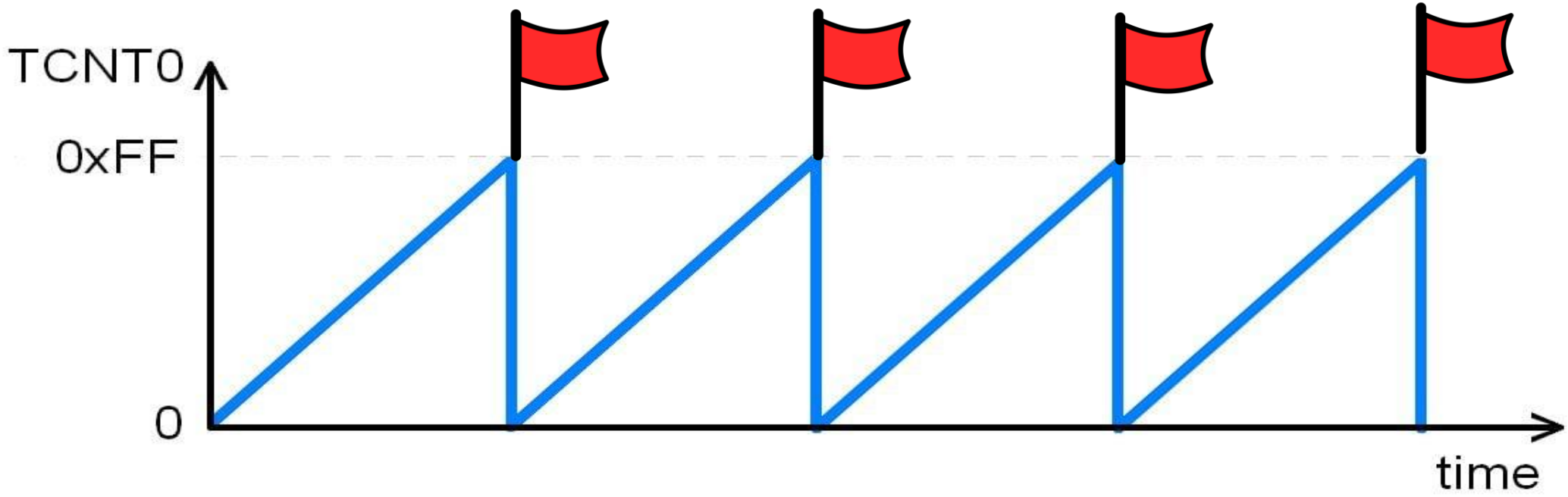
Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- This is where the 8-bit counter of the timer resides. The value of the counter is stored here and it increases automatically each clock cycle. Data can be both read/written from this register. The initial value of the counter is set by writing it.

Timer Overflow (Normal) Mode

D-BUGERZ

1.
2.
3.
4.
5.



Timers Register (Timer0)

D-BUGERZ

1.
2.
3.
4.
5.

- Page 83 Datasheet

Timer/Counter Interrupt Flag Register – TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
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	

- **Bit 0 -TOVO: Timer/Counter0 Overflow Flag**
  - It is set (one) when an overflow occurs in Timer/Counter0.
  - TOVO is **cleared by writing a logic one to the bit flag.**
  - Alternatively, TOVO is **cleared by hardware when executing** the Timer0 Overflow **ISR**.

Timer Overflow (Normal) Mode

D-BUGERZ

Steps to Program Timer0 with polling

1. **You must make your clock calculations to know overflow will occur in how many time to know how many number of overflow you will need to reach your desired time**
2. Load the TCNTO register with the initial value (let's take 0).
3. For **normal mode** and **the pre-scaler** option of the clock, set the value in the TCCRO 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.
4. Timer keeps counting up, so keep monitoring for timer overflow i.e. TOVO (Timer0 Overflow) flag to see if it is raised.

```
14  */
15 void Timer0_Delay(void)
16 {
17     TCNT0 = 0; // Set Timer0 initial value to 0
18
19     /* Configure the timer control register
20      * 1. Non PWM mode FOC0=1
21      * 2. Normal Mode WGM01=0 & WGM00=0
22      * 3. Normal Mode COM00=0 & COM01=0
23      * 4. clock = F_CPU/1024 CS00=1 CS01=0 CS02=1
24      */
25     TCCR0 = (1<<FOC0) | (1<<CS02) | (1<<CS00);
26
27     while(!(TIFR & (1<<TOV0))); // Wait until the Timer0 Overflow occurs (wait until TOV0 = 1)
28
29     TIFR |= (1<<TOV0); // Clear TOV0 bit by set its value
30
31     TCCR0 = 0; // Stop Timer0 by clear the Clock bits (CS00, CS01 and CS02)
32 }
```

2

3

4

Timer Overflow (Normal) Mode

D-BUGERZ

Steps to Program Timer0 with polling

5. Clear the TOVO flag. Note that we have to write 1 to the TOVO bit to clear the flag
6. Stop the timer by putting 0 in the TCCR0 ie the clock source will get disconnectedand the timer/counter will get stopped.
7. Return to the main function.
8. This design technique is called polling as we wait until the Timer0 finishes the counts

1.
2.
3.
4.
5.

```
14  */
15 void Timer0_Delay(void)
16 {
17     TCNT0 = 0; // Set Timer0 initial value to 0
18
19     /* Configure the timer control register
20      * 1. Non PWM mode FOC0=1
21      * 2. Normal Mode WGM01=0 & WGM00=0
22      * 3. Normal Mode COM00=0 & COM01=0
23      * 4. clock = F_CPU/1024 CS00=1 CS01=0 CS02=1
24      */
25     TCCR0 = (1<<FOC0) | (1<<CS02) | (1<<CS00);
26
27     while(!(TIFR & (1<<TOV0))); // Wait until the Timer0 Overflow occurs (wait until TOV0 = 1)
28
29     TIFR |= (1<<TOV0); // Clear TOV0 bit by set its value
30
31     TCCR0 = 0; // Stop Timer0 by clear the Clock bits (CS00, CS01 and CS02)
32 }
```

5

6

Timer Overflow (Normal) Mode

D-BUGERZ

Steps to Program Timer0 with polling

1. **You must make your clock calculations to know overflow will occur in how many time to know how many number of overflow you will need to reach your desired time**

1.
2.
3.
4.
5.

```
// Wait for half second
for(count = 0; count < 2; count++)
{
  Timer0_Delay();
}
```

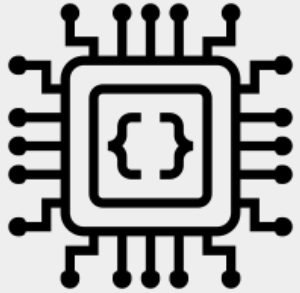
```
/*
void Timer0_Delay(void)
{
    TCNT0 = 0; // Set Timer0 initial value to 0

    /* Configure the timer control register
     * 1. Non PWM mode FOC0=1
     * 2. Normal Mode WGM01=0 & WGM00=0
     * 3. Normal Mode COM00=0 & COM01=0
     * 4. clock = F_CPU/1024 CS00=1 CS01=0 CS02=1
     */
    TCCR0 = (1<<FOC0) | (1<<CS02) | (1<<CS00);

    while(!(TIFR & (1<<TOV0))); // Wait until the Timer0 Overflow occurs (wait until TOV0 = 1)

    TIFR |= (1<<TOV0); // Clear TOV0 bit by set its value

    TCCR0 = 0; // Stop Timer0 by clear the Clock bits (CS00, CS01 and CS02)
}
```

Exercise 1

D-BUGERZ

1.
2.
3.
4.
5.

Write Embedded C code using ATmega16/32 µC to control led using Timer0



Requirements:

1- Configure the µC clock with internal 1Mhz Clock
2- The led is connected to pin 0 in PORTC
3- Connect the Led using Positive Logic configuration
4- Configure the timer clock to F_CPU/1024.
5- Toggle the led every half second
6- Use the Timer0 Normal Mode with Polling

D-BUGERZ

1.

2.

3.

4.

5.

Timer Overflow (Normal) Mode

D-BUGERZ

- One of the basic concepts is the situation when timer overflow occurs then **Timer Overflow Interrupt can be generated.**

- In this situation timer can issue an interrupt if **the global interrupt mask bit is set** and the **timer overflow interrupt is enable** and you must **write an Interrupt Service Routine (ISR)** to handle the event.

- You can as well **load a count value in TCNTO** and start the timer from a specific count but the timer will start from ZERO again after overflow. So if you need to start from the same value you will need to reload the count value again in TCNT0, it always done inside the timer overflow ISR

1.
2.
3.
4.
5.

Timers Register (Timer0)

D-BUGERZ

1.

2.

3.

4.

5.

- Page 82 Datasheet

Timer/Counter Interrupt Mask Register – TIMSK

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
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	

- **Bit 1 – OCIE0: Timer/Counter0 Output Compare Match Interrupt Enable**
  - 0 : Disable compare mode interrupt for Timer0
  - 1 : Enable compare mode interrupt for Timer0
- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**
  - 0 : Disable Overflow Interrupt mode for Timer0
  - 1 : Enable Overflow Interrupt mode for Timer0

How To Trigger Interrupt Request in Timers

D-BUGERZ

1. ❑ **Global Interrupt Enable (GIE).**

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
	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	

- **Bit 7 – I: Global Interrupt Enable**

2. ❑ **Module Interrupt Enable (MIE)**

3. **Timer/Counter Interrupt Mask Register – TIMSK**

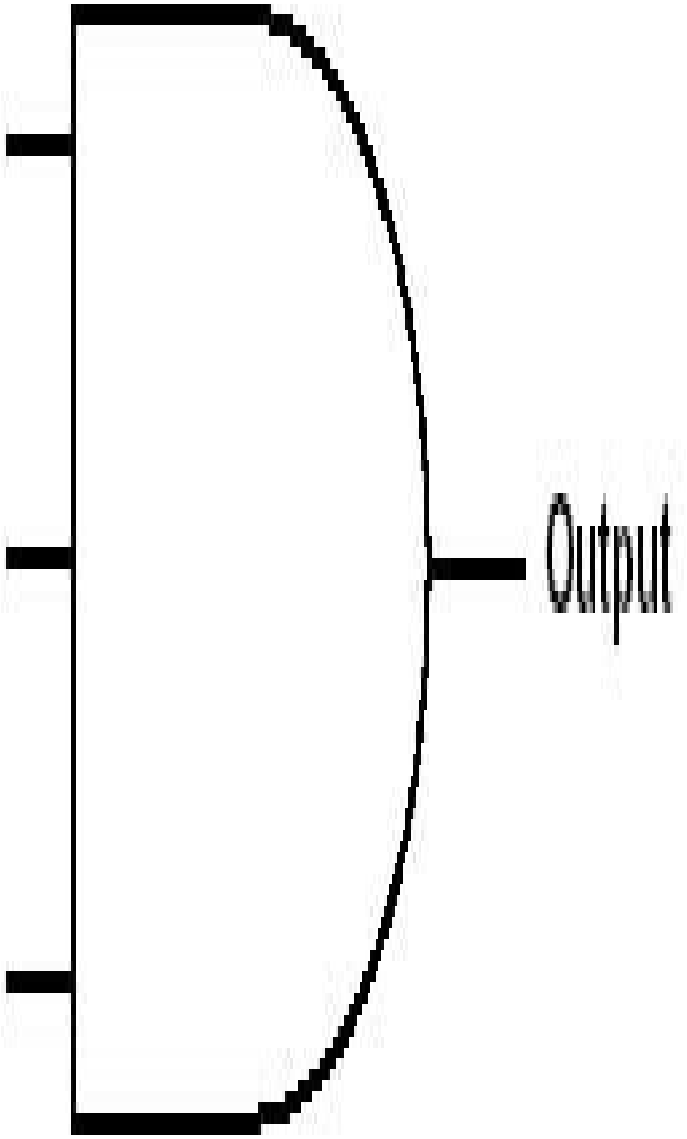
Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

4. ➔ **There are 2nd step here to configure when you want to receive interrupt**

5. ❑ **Module Interrupt flag (MIF).**

Timer/Counter Interrupt Flag Register – TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
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	



Timer Overflow (Normal) Mode

D-BUGERZ

Steps to Program Timer0 with Interrupt

1. **You must make your clock calculations to know overflow will occur in how many time to know how many number of overflow you will need to reach your desired time**
2. Load the TCNTO register with the initial value (let's take 0).
3. Enable Timer Interrupt to give interrupt for each time overflow done
4. For **normal mode** and **the pre-scaler** option of the clock, set the value in the TCCRO 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.

```
34  */
35 void Timer0_Init_Normal_Mode(void)
36 {
37     TCNT0 = 6; //Set Timer initial value to 6                    2
38
39     TIMSK |= (1<<TOIE0); // Enable Timer0 Overflow Interrupt     3
40
41     /* Configure the timer control register
42      * 1. Non PWM mode FOC0=1
43      * 2. Normal Mode WGM01=0 & WGM00=0
44      * 3. Normal Mode COM00=0 & COM01=0
45      * 4. clock = F_CPU/1024 CS00=1 CS01=0 CS02=1
46      */
47     TCCR0 = (1<<FOC0) | (1<<CS02) | (1<<CS00);                   4
48 }
49
```

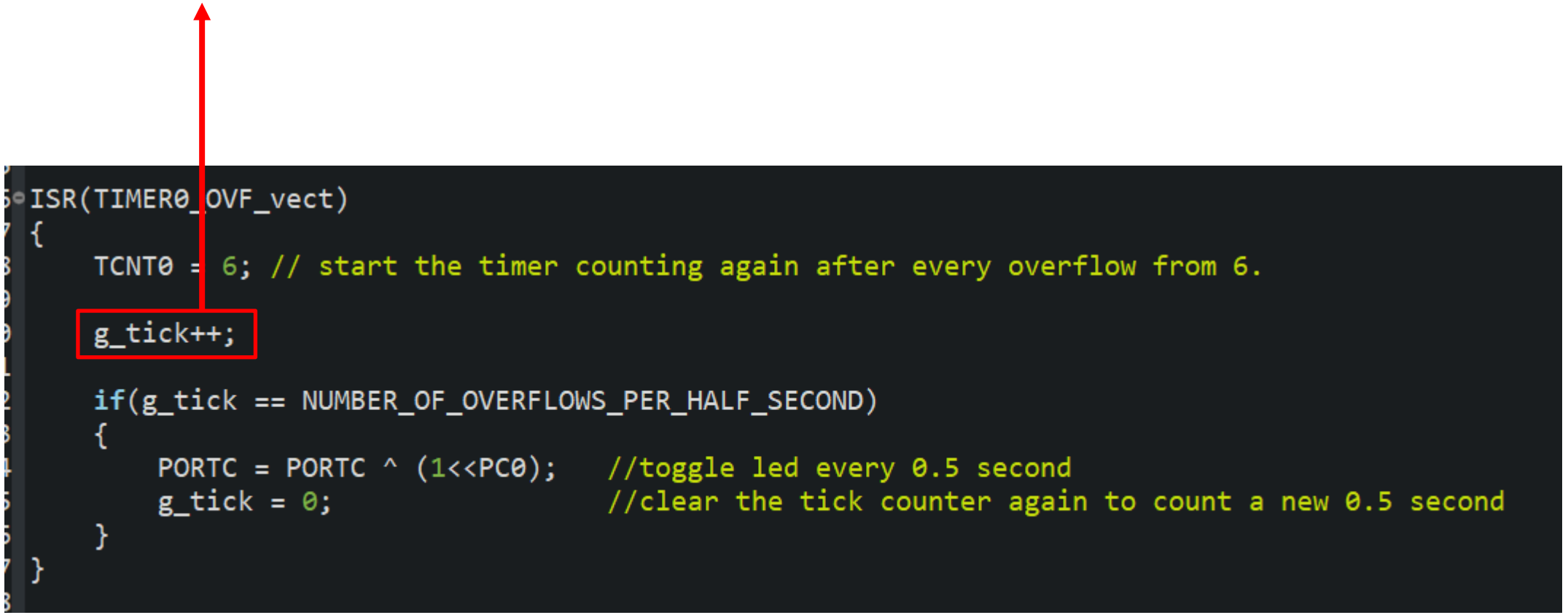
Timer Overflow (Normal) Mode

D-BUGERZ

Steps to Program Timer0 with polling

1. **You must make your clock calculations to know overflow will occur in how many time to know how many number of overflow you will need to reach your desired time**

1.
2.
3.
4.
5.



```
ISR(TIMER0_OVF_vect)
{
    TCNT0 = 6; // start the timer counting again after every overflow from 6.

    g_tick++;

    if(g_tick == NUMBER_OF_OVERFLOWS_PER_HALF_SECOND)
    {
        PORTC = PORTC ^ (1<<PC0);    //toggle led every 0.5 second
        g_tick = 0;                  //clear the tick counter again to count a new 0.5 second
    }
}
```

Timer Overflow (Normal) Mode

D-BUGERZ

Important Notes when Use Timer Interrupt

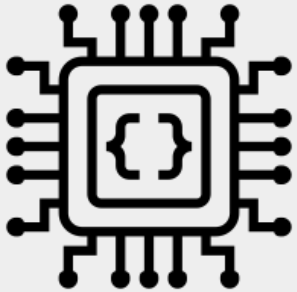
1. **You must put intial value of timer in each time enter ISR to start count from it (TCNT0)**
2. **If you enter ISR you don't need to delete Flag bit (TIFR)**
3.
4.
5.

Exercise 2

D-BUGERZ

1.
2.
3.
4.
5.

Write Embedded C code using ATmega16/32 µC to control led using Timer0.



Requirements:

1- Configure the µC clock with internal 1Mhz Clock
2- The led is connected to pin 0 in PORTC
3- Connect the Led using Positive Logic configuration.
4- Configure the timer clock to F CPU/1024.
5- Toggle the led every half second.
6- Use the Timer0 Normal Mode with Interrupt technique.

D-BUGERZ

1.

2.

3.

4.

5.

Exercise 3

D-BUGERZ

1.
2.
3.
4.
5.

Write Embedded C code using ATmega16/32 µC to control a 7-segment using Timer0.

Requirements:

1- Configure the µC clock with internal 1Mhz Clock.
2- 7-segment connected to PORTC.
3- Configure the timer clock to F_CPU/1024
4- Every second the 7-segment should be incremented by one if the 7-segment reaches the maximum number (9) overflow occurs
5- Use the Timer Normal Mode with Interrupt technique

Formative Challenge

D-BUGERZ

Write Embedded C code using ATmega16/32 µC to control a 7-segment using Timer0.

Requirements:

1. ✓ **Configure the µC clock with 16Mhz Crystal Oscillator**
2. ✓ **The 7-Segment is connected directly to PORTA from PA1 PA7 without a decoder**
3. ✓ **The 7-Segment type is common cathode and enable the first 7-segment which its common pin controlled by PC6 pin**
4. ✓ **Configure the timer clock to F CPU/256**
5. ✓ **Every second the 7-segment should be incremented by one, if the 7-segmentreaches the maximum number (9) overflow occurs.**
✓ **Use the Timer0 Normal Mode with Interrupt technique.**

D-BUGERZ

1.

2.

3.

4.

5.

BREAK 10 MIN

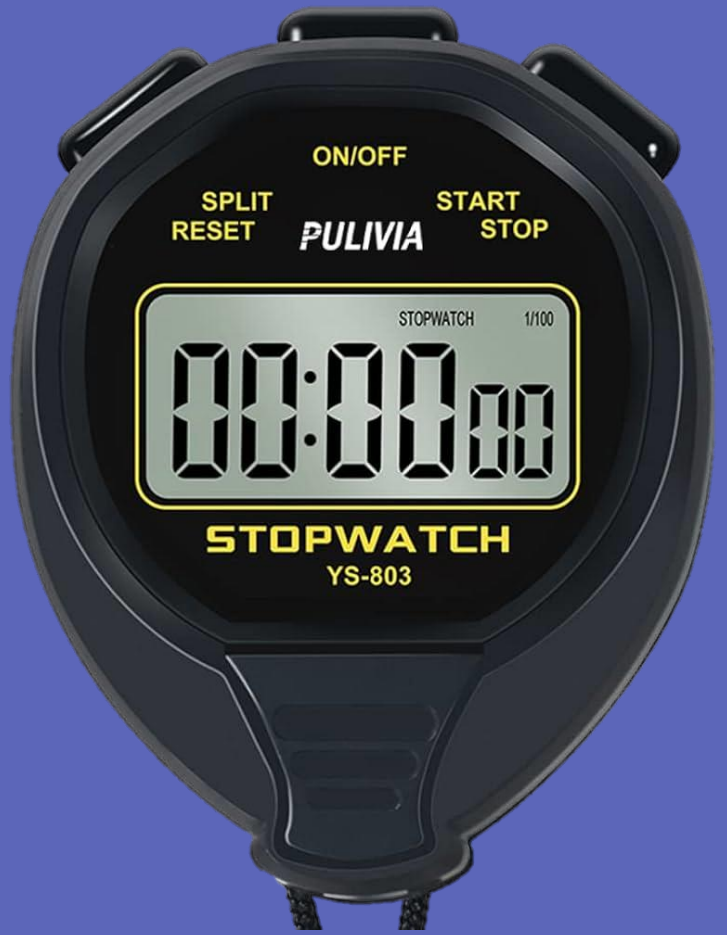
Recall Last session

Exercise on Interrupts

Introduction to Timers

Start Coding with Timers

Final Project Description



D-BUGERZ

Project Description

D-BUGERZ

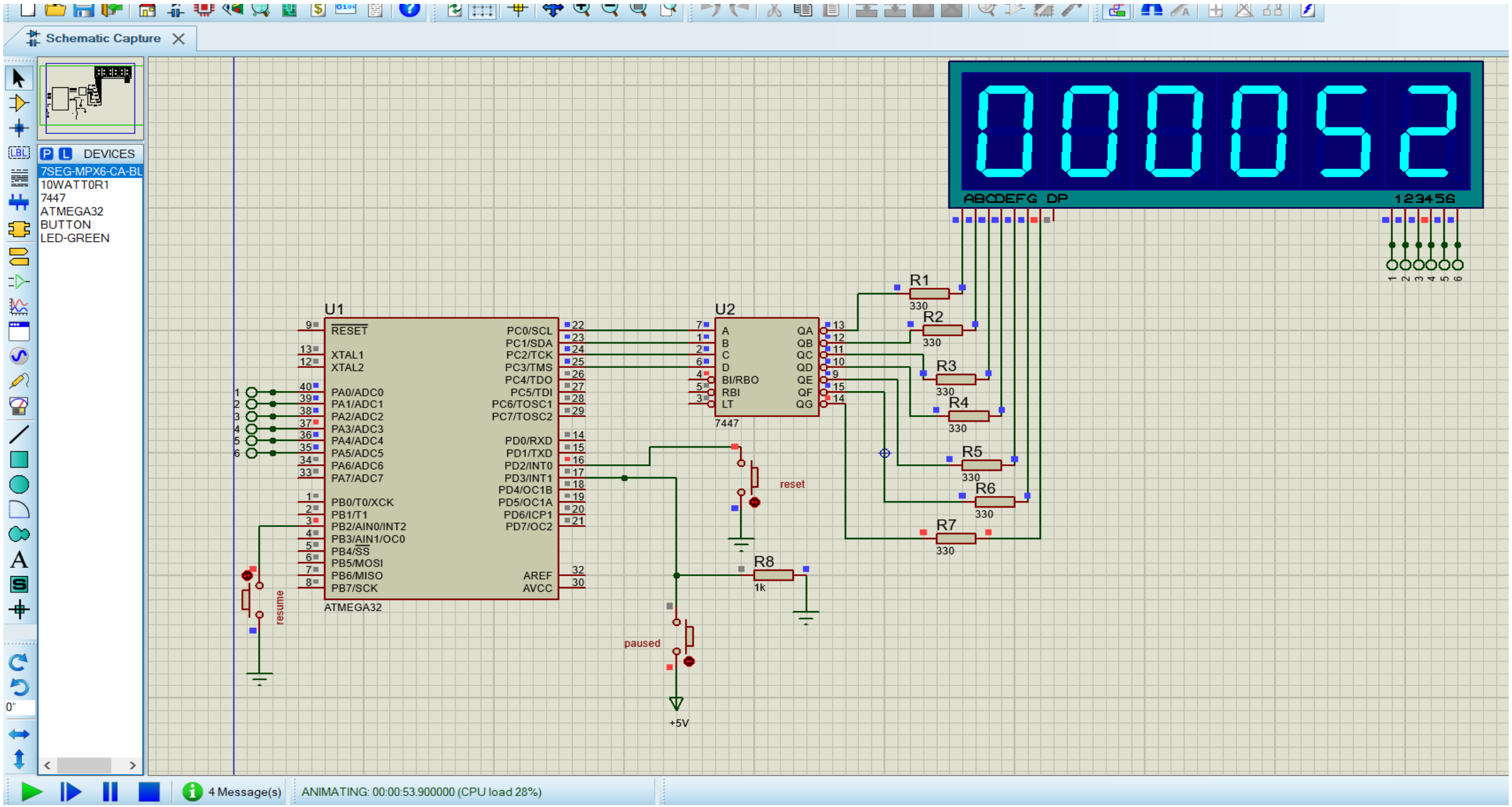
1.

2.

3.

4.

5.



Project Description

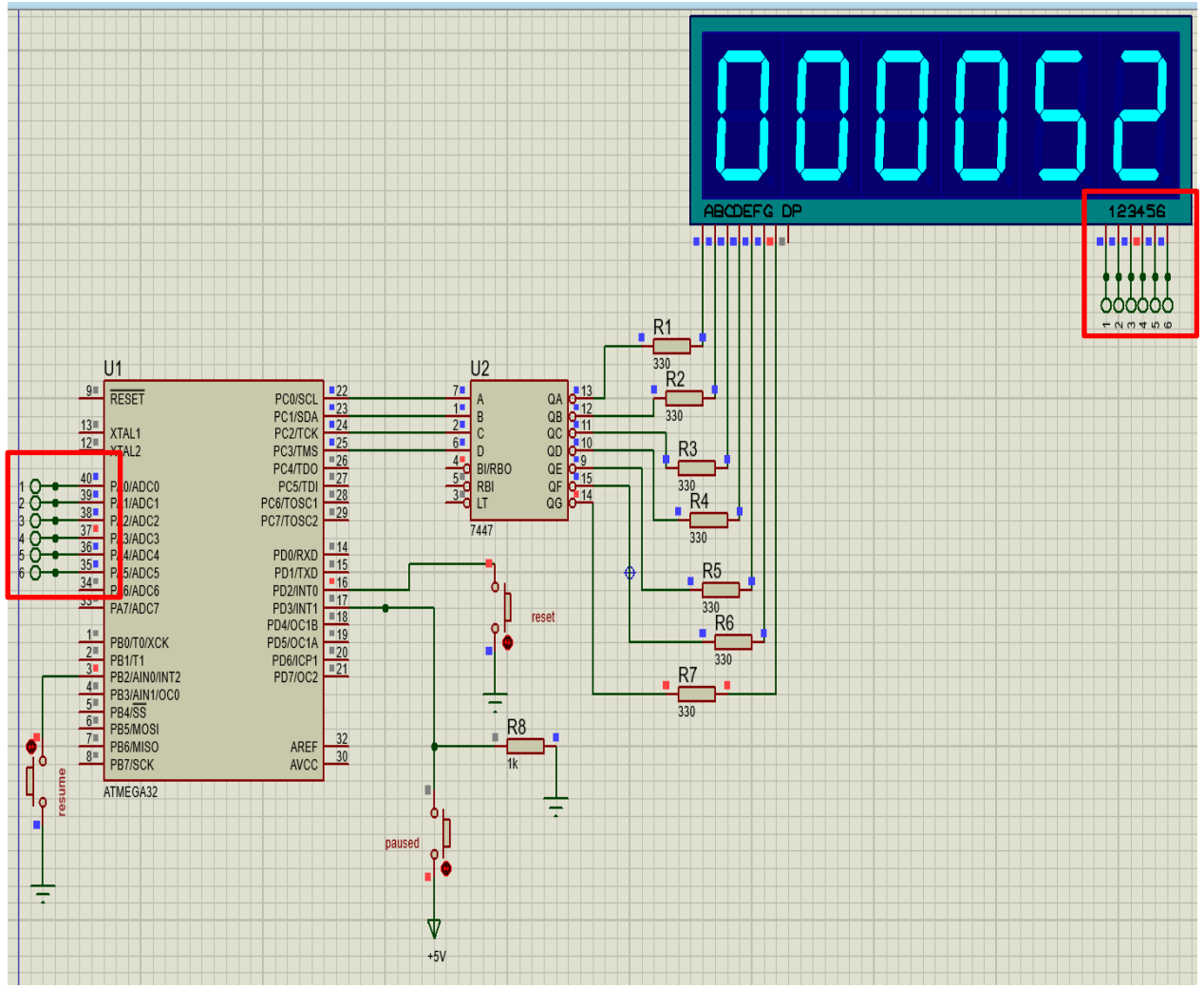
D-BUGERZ

1.
2.
3.
4.
5.

1st Challenge

- **Make 6 7-segment On together**

→ By looping on pins by with small delay so you will see all 7-segment are on



Project Description

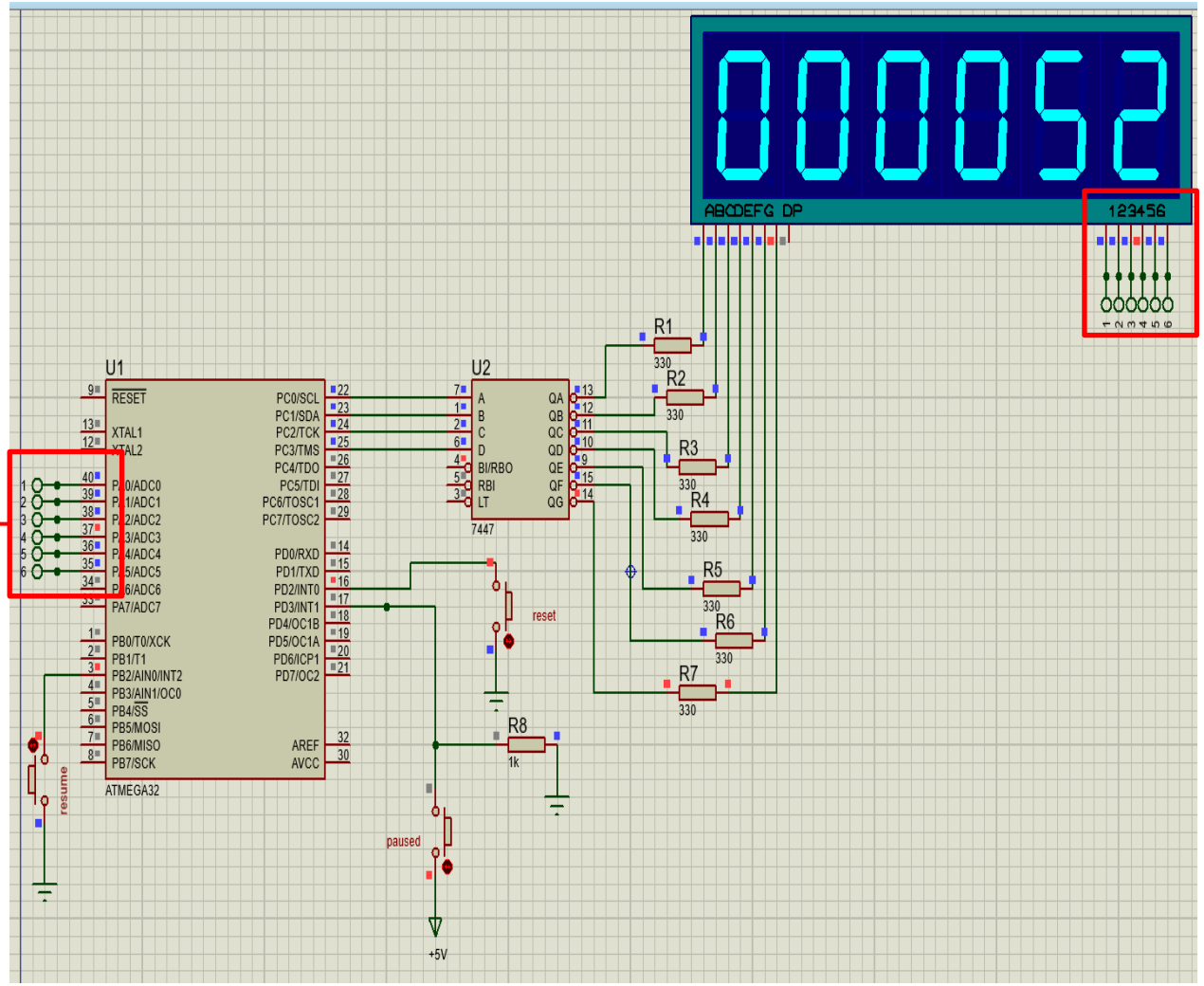
D-BUGERZ

1st Challenge

- Make 6 7-segment On together

1.
2.
3.
4.
5.

```
while(1)
{
    //display
    PORTA = (1<<0);  //enable first 7 segment and disable others
    PORTC= (PORTC & 0xf0) | (hours_1 & 0x0f);
    _delay_ms(2);

    PORTA = (1<<1);   //enable second 7 segment and disable others
    PORTC= (PORTC & 0xf0) | (hours_0 & 0x0f);
    _delay_ms(2);

    PORTA = (1<<2);  //enable third 7 segment and disable others
    PORTC= (PORTC & 0xf0) | (minutes_1 & 0x0f);
    _delay_ms(2);

    PORTA = (1<<3);  //enable fourth 7 segment and disable others
    PORTC= (PORTC & 0xf0) | (minutes_0 & 0x0f);
    _delay_ms(2);

    PORTA = (1<<4);  //enable fifth 7 segment and disable others
    PORTC= (PORTC & 0xf0) | (secound_1 & 0x0f);
    _delay_ms(2);

    PORTA = (1<<5);  //enable sixth 7 segment and disable others
    PORTC= (PORTC & 0xf0) | (secound_0 & 0x0f) ;
    _delay_ms(2);
```



Project Description

D-BUGERZ

1.
2.
3.
4.
5.

2nd Challenge

❖ **Timer Calculations**

→ Make your calculations first

→ You need to know how many **ticks** to make 1 sec and you must use configurations in project description **1MHZ** overflow mood or Compare mode

```
24  */
25 void Timer0_Init_Normal_Mode(void)
26 {
27     TCNT0 = 0; // Set Timer initial value to 0
28
29     TIMSK = (1<<TOIE0); // Enable Timer0 Overflow Interrupt
30
31     /* configure the timer
32      * 1. Non PWM mode FOC0=1
33      * 2. Normal Mode WGM01=0 & WGM00=0
34      * 3. Normal Mode COM00=0 & COM01=0
35      * 4. clock = F_CPU/256 CS00=0 CS01=0 CS02=1
36      */
37     TCCR0 = (1<<FOC0) | (1<<CS02);
38 }
```

```
0 ISR(TIMER0_OVF_vect)
1 {
2     g_tick++;
3     if(g_tick == NUMBER_OF_OVERFLOWS_PER_SECOND)
4     {
```

Project Description

D-BUGERZ

3rd Challenge

❖ **External Interrupts from Buttons**

1.
2.
3.
4.
5.

→ Each interrupt must have its **initialization function** and his own **ISR function**

```
void paused_INT1 ()
{
    //write your intilization
}

void resume_INT2()
{
    //write your intilization
}

void reset_INT0()
{
    //write your intilization
}
```

```
ISR(INT1_vect)
{
   // your code
}

ISR(INT0_vect)
{
  // your code
}


ISR(INT2_vect)
{
  // your code
}
```

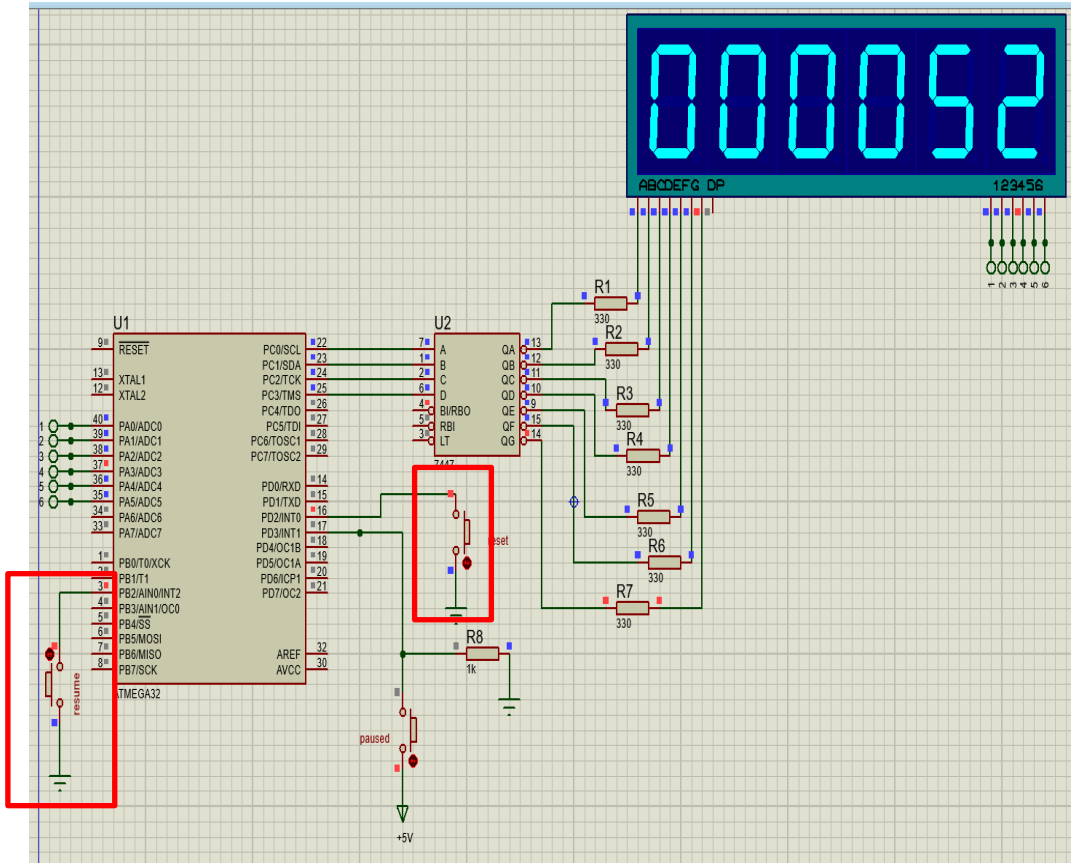
Project Description

D-BUGERZ

4th Challenge

❖ **Internal pull up Resistor**

1. → Open Datasheet **page 56** and try to Active internal pull up resistors
2. → Try to read description and of pin PUD and know how to use it in project
3.
4.
5.



Special Function I/O Register – SFIOR

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Project Description

D-BUGERZ

1.
2.
3.
4.
5.

Some Important Notes

1. Remember to call all initialization function in main before while(1)
2. You are allowed to put long code in Timer ISR **ONLY** in this project
3. Don’t forget enable Global interrupt bit
4. Don’t forget Notes of Timer overflow in slide 59
5. Don’t forget at start of code:

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

6. Remember how to stop Timer with smart way as describe in session
7. Try to open data sheet and be familiar with registers
8. Try to do your maximum effort 💔

D-BUGERZ

1.

2.

3.

4.

5.

WE DID IT

D-BUGERZ