ENGINE

Teaching online electronics, microcontrollers and programming in Higher Education

Output 2: Online Course for Microcontrollers: syllabus, open educational resources

Practice leaflet: Module_2-8 Timers

Lead Partner: International Hellenic University (IHU)

Authors: Theodosios Sapounidis [IHU], Aristotelis Kazakopoulos [IHU], Aggelos Giakoumis [IHU], Sokratis Tselegkaridis [IHU]

Erasmus+

GRANT NUMBER: 2020-1-PL01-KA226-HE-095653

Declaration

This report has been prepared in the context of the ENGINE project. Where other published and unpublished source materials have been used, these have been acknowledged.

Copyright

Funding Disclaimer

This project has been funded with support from the European Commission. This report reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Table of Contents

Executive summary 4

Chapter 1: Overview 5

Chapter 2: Activities 7

2.1 Activity 1. Interrupt every 10ms 7

2.2 Activity 2. Delay function using Timer0 10

Chapter 3: Recapitulation 13

References 14

Appendix. Figures with high resolution 15

Executive summary

In this Module we will use PIC18F4550 with the Timer0.

Chapter 1: **Overview**

Table 1. Overview

Title / short summary	**8. Timer0**
Expected learning outcomes	• The student will be able to configure the Timer0 on the microcontroller • The student will be able to handle an interrupt from Timer0 • The student will be able to design simple circuits with a the Timer0 • The student will be able to load and animate a microcontroller program in the Proteus Design Suite
Keywords	Timers, internal interrup
Duration	The duration of the module_2-8 is 3 hours • Presentation of the module_2-8 by the teacher, 45 minutes • 1st activity, interrupt every 10ms, 60 minutes • 2nd activity, delay function using Timer0, 75 minutes
Involved	**The teacher:** Presents the slides associated with the module_2-8 and answers question **The students:** Draw circuits in Proteus Schematic, write programs in C language, load programs to a microcontroller and run the simulation using the Proteus Design Suite

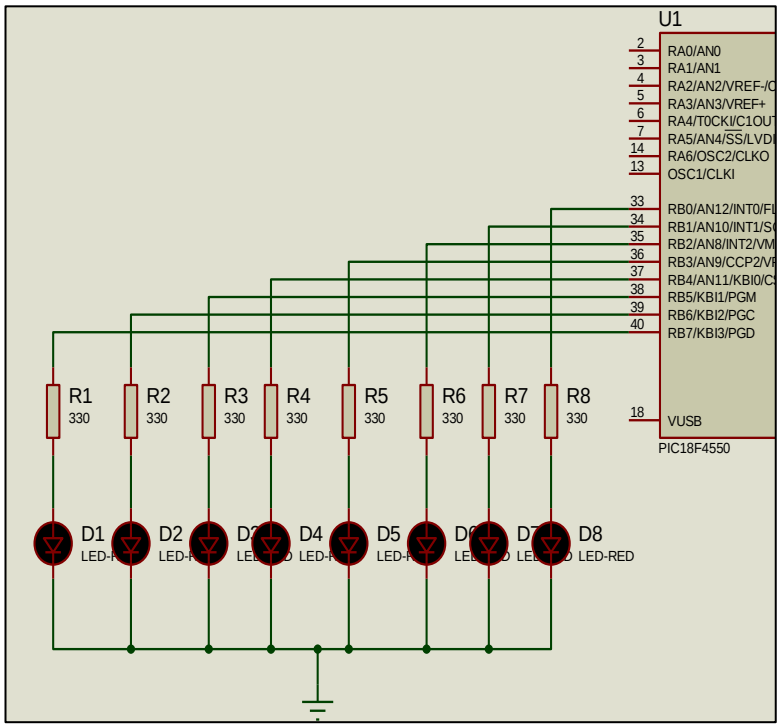
Assignment	At the end of the Module_2-8 will be given: • Open Project
Educational tools and equipment	• **Material:** PC • **Software:** CCS C compiler, Proteus Design Suite
Prerequisites / pre-existing knowledge	• The student must be familiarized with the Proteus Design Suite (link1) • The student must be completed Module_2-1 and Module_2-2
Educational content	• CCS C Compiler manual (C Compiler Reference Manual) • MICROCHIP, PIC18F2455/2550/4455/4550 Data Sheet • Module_2-8 slides • Module_2-8 Evaluation leaflet • Module_2-8 Open project leaflet • Module_2-8 Programs, Schematic Proteus (Compressed folder)
Tips	***Tip.*** *8-bit vs 16-bit register*

Chapter 2: **Activities**

2.1 Activity 1. Interrupt every 10ms

The purpose of this activity is to write a program with an interrupt service routine from Timer0 that blinks the LEDs connected to PORTB every 200 ms. Caution! No delay functions will be used in the program. As a time base, Timer0 will be set to trigger an interrupt every 10ms.

Table 2. Activity 1

Activity 1st (60 minutes)	**Step 1.** The circuit is drawn in the Proteus Design Suite. **Step 2.** Timer0 configurations. **Step 3.** The program in C language is written. **Step 4.** The program is compiled with the use of CCS C compiler to the microcontroller machine code. **Step 5.** The machine code is loaded to the microcontroller. **Step 6.** The animation is activated.
Step 1 (10 minutes)	Draw the circuit of the picture in the Proteus Design Suite.  *Figure 1. Connections*

Step 2
(25 minutes)

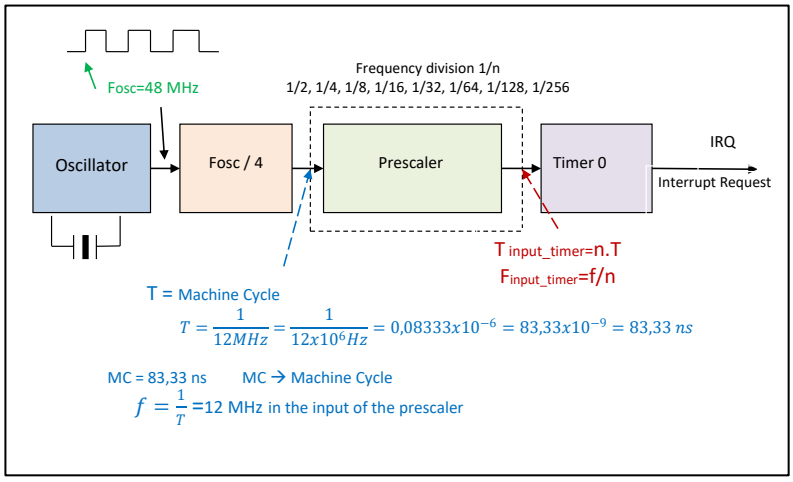


Figure 2. Timer0 block diagram

The frequency in the input of the Prescaler is 12MHz.

We chose from the Prescaler the value of 1/64, therefore, the frequency in the output of the Prescaler is 12 / 64 = 0.1875MHz.

Accordingly, the period at the input of timer 0 (Timer0) will be: 83.33 ns X 64 =5333ns

Timer0 should be initialized so that timer0 overflows every 10ms.

Overflowing timer0 means going from the value FFFF to the value 0000. The time it takes for timer0 to go from the initial value given to it until it overflows (and thus interrupts) should be:

10ms=10 000 μs=10 000 000 ns

It is reminded that (FFFF)h = (65535)d

The number of steps from the initial value of Timer0 until it overflows will be: 65536-(Initial value of Timer0). That is:

[65536-(Initial value of Timer0)]x5333 **ns**= 10 000 000 **ns**.

(Initial value of Timer0) = 65536-1875 =**63661**

Step 3 (15 minutes)	Write in CCS Compiler the program in C language

```
#include <main.h> // the file main.h with the
          // initial settings is included.
          // This file must be placed in the same
          // folder with the project.
          // Also the 18F4550.h file must exist
          // in the same folder with the project


#byte PORTB =0xF81
/* We attribute to the memory position 0xF81 the
name PORTB. This means that we define an 8-bit
variable whose value will be stored to the memory
position F81h.*/
#byte PORTD =0xF83
// The position F83h is the PORTD data register.

void init (void);
int counter1=20;

void main()
{
   init();
   while (TRUE){
        ;
   }
}

// Interrupt Service Routine
#INT_TIMER0
void timer0_int(void){
   set_timer0(63661);
   counter1--;
   if (counter1==0){
      counter1=20;
      PORTB=PORTB^0b11111111;
   }
}


void init (void)
{
   // Prescaler value = 1/64
   SETUP_TIMER_0(T0_INTERNAL | T0_DIV_64 );
   set_timer0(63661);
   enable_interrupts(INT_TIMER0);
   enable_interrupts(GLOBAL);

   set_tris_b(0x00);
   PORTB=0x00;
}

/* Interrupt time
 (65536-63661)*[1/(Fclock/4)]*Prescaler = 9.994ms
  */
```

Step 4 (4 minutes)	Compile the program in C in order to create the program in the microcontroller machine code (hex file).
Step 5 (1 minutes)	Load to the microcontroller the hex file (program in machine code) that was created from the CCS Compiler.
Step 6 (5 minute)	Run the simulation and check the correct operation of the circuit.

2.2 Activity 2. Delay function using Timer0

In this activity we will use Timer0 to create our own "delay" function that lasts 100us

Table 3. Activity 2

Activity 2nd (75 minutes)	***** The circuit is the same as the circuit of Activity 1 ***** **Step 1.** Timer0 configurations. **Step 2.** The program in C language is written. **Step 3.** The program is compiled with the use of CCS C compiler to the microcontroller machine code. The machine code is loaded to the flash memory of the microcontroller. **Step 4.** The animation is activated. **Step 5.** Modifications and discussion.
Step 1 (20 minutes)	The frequency in the input of the Prescaler is 12MHz. We chose from the Prescaler the value of 1/1, therefore, the frequency in the output of the Prescaler is 12MHz. Accordingly, the period at the input of timer 0 (Timer0) will be: 83.33 X 1 = 83.33 ns. Timer0 should be initialized so that Timer0 overflows every 100us.

	Overflowing timer0 means going from the value FFFF to the value 0000. The time it takes for timer0 to go from the initial value given to it until it overflows (and thus interrupts) should be: 100us=100000ns It is reminded that (FFFF)h = (65535)d The number of steps from the initial value of Timer0 until it overflows will be: 65536-(Initial value of Timer0). That is: [65536-(Initial value of Timer0)]x83.33 **ns**= 100000 **ns**. (Initial value of Timer0) = 65536-1200 =**64336**

Step 2
(25 minutes)
	Write in CCS C Compiler the program
	
	`#include <main.h> // the file main.h with the`
	`// initial settings is included.`
	`// This file must be placed in the same`
	`// folder with the project.`
	`// Also the 18F4550.h file must exist`
	`// in the same folder with the project`
	
	`#byte PORTB =0xF81`
	`/* We attribute to the memory position 0xF81 the name PORTB. This means that we define an 8-bit variable whose value will be stored to the memory position F81h.*/`
	
	`void init (void);`
	`void timer0_int(void);`
	`//function for our delay`
	`void mydelay_100us(int);`
	
	`int32 counter_time=0;`
	`//Declare a variable to count the interrupts.`
	`//It will increment by 1 every 100 µs`
	
	`int32 counter_time_old=0;`
	`int32 aaa=1;`
	
	`void main(){`
	`init();`
	`while (TRUE){`
	`PORTB=PORTB^0b11111111;`
	`mydelay_100us(500);`
	`//delay 500X100µs=50 000 µs= 50 ms`
	`}`
	`}`
	
	`//Interrupt Service Routine`
	`#INT_TIMER0`
	`void timer0_int(void) {`
	`set_timer0(64336);`
	`counter_time++;`

	}
	
	void init (void) {
	SETUP_TIMER_0(T0_INTERNAL | T0_DIV_1);
	set_timer0(64336);
	enable_interrupts(INT_TIMER0);
	enable_interrupts(GLOBAL);
	set_tris_b(0x00);
	PORTB=0x00;
	}
	
	void mydelay_100us(aaa){
	counter_time_old=counter_time;
	while(counter_time < counter_time_old+aaa) { }
	}

Step 3 (5 minutes)	Use the CCS C Compiler to translate the programm from C language to the microcontroller machine code. Load to the microcontroller the hex file (machine code) that was created from the CCS Compiler.
Step 4 (5 minutes)	Run the simulation and check the correct operation of the circuit.
Step 5 (20 minutes)	If the microcontroller oscillator is 48MHz, what is the maximum time it can count before the Timer0 overflows?

Chapter 3: **Recapitulation**

- ☞ The schematic of the circuit was drawn with Proteus Design Suite.
- ☞ The programs in C was written in CCS C compiler.
- ☞ The programs in C was compiled to the microcontroller machine code (hex file).
- ☞ The machine code was “loaded” to the microcontroller and the animation was activated.

References

CCS C Compiler Manual. Ccsinfo.com. (2021). Retrieved from https://www.ccsinfo.com/downloads/ccs_c_manual.pdf.

PIC18F2455/2550/4455/4550 Data Sheet. Ww1.microchip.com. (2006). Retrieved from https://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf.

Proteus Tutorial : Getting Started with Proteus PCB Design (Version 8.6). Youtube.com. (2017). Retrieved from https://www.youtube.com/watch?v=GYAHwYUUs34.

Simple LED Circuits. Electronics Hub. (2017). Retrieved from https://www.electronicshub.org/simple-led-circuits/.

Appendix. Figures with high resolution

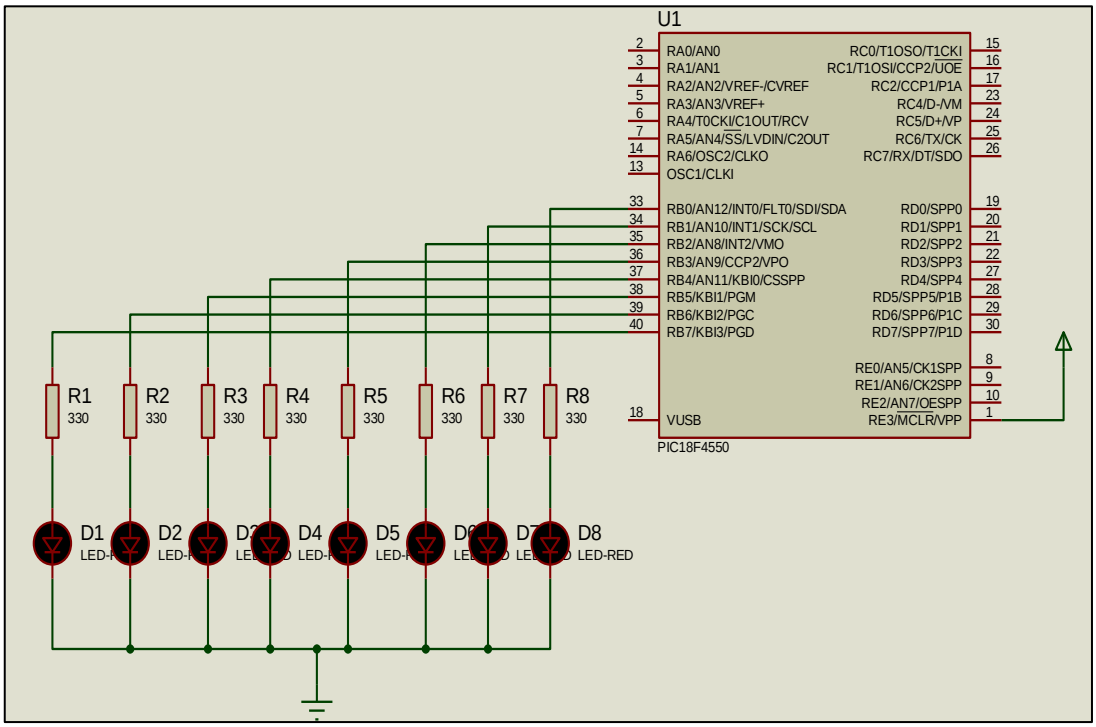


Figure 1. Connections

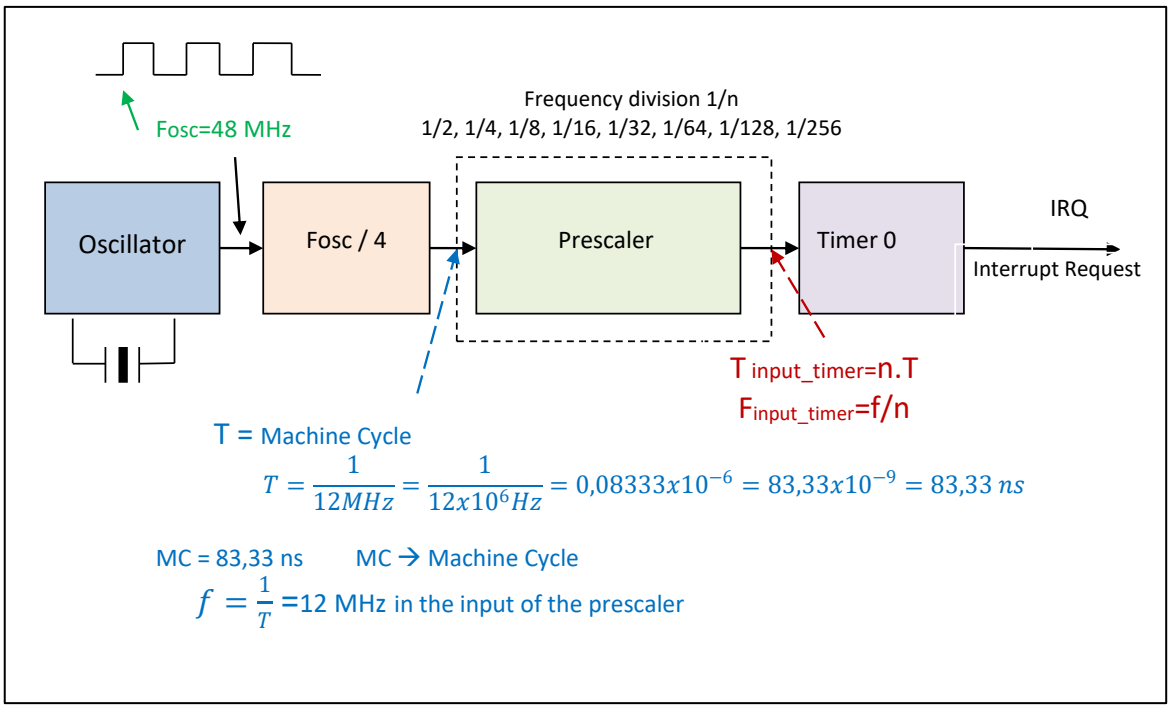


Figure 2. Timer0 block diagram