LAB 3 RAPORT

Creator: Sampo Kajaste 1881563

SISÄLTÖ

1 GOAL

In this lab, you work with interrupt. You learn how to write an Interrupt Service Routine (ISR) to handle timer interrupts.

1.1 Objectives

Upon completion of this lab, you will be able to:

- Understand the behaviour of interrupts.
- What is an ISR and how to write it.
- How to measure frequency of a pulse signal using a combination of timer and interrupt.

1.2 Interrupt

An interrupt is an unpredictable event that requires CPU to stop its normal operation and provide some services related to the event. An interrupt can be generated by an event inside processor chip (internal interrupt) or by an event outside processor chip (external interrupt). An internal interrupt is generated by the peripherals inside microcontroller or by software errors such as divide by zero or invalid op-codes. An external interrupt is generated when an external device asserts an interrupt signal. When an interrupt occurs, microcontroller stops running current program and jumps to a fixed address to execute Interrupt Service Routine (ISR). ISR is a program that handles an interrupt. When ISR is finished, the CPU returns to the interrupted program. In PIC18, the default value for address of ISR is 0x08 for high-priority interrupts and 0x18 for low-priority interrupts. For details of interrupt, refer to chapter 11 of the textbook.

1.3 Frequency Measurement

In this part, you write an assembly program to measures frequency of a square wave. Use a function generator to generate a 1-KHz square wave. Connect the function generator to RA4. Figure 1 shows counter0 and timer3 that you will use in this section of the lab. 2 Counter0: The clock of counter0 is connected to RA4. Configure counter0 to increment on the rising edge of the clock. Timer3: Timer3 should be configured to generate 1-second intervals using interrupt. After 1- seond, read counter0 (TMR0L, TMR0H). The value of counter0 shows frequency of the 1-kHz signal. Timer3 sends measured frequency to the output LEDs. (hint1: when you read from TMR0, you need to read from TMR0L first, and then read from TMR0H. Hint2: Counter0 is 16- bit but the board has 8-LED. Timer3 should send low and high bytes of Counter0 to the LEDs, alternatively. Hint3: set T1OSCEN in T1CON)

2 CODE

```
; PIC18F87J11 Configuration Bit Settings
; Assembly source line config statements
#include "p18f87j11.inc"
; CONFIG1L
 CONFIG WDTEN = OFF ; Watchdog Timer Enable bit (WDT disabled (control is placed on SWDTEN
bit))
 CONFIG STVREN = OFF ; Stack Overflow/Underflow Reset Enable bit (Reset on stack
overflow/underflow disabled)
 CONFIG XINST = OFF ; Extended Instruction Set Enable bit (Instruction set extension and Indexed
Addressing mode disabled (Legacy mode))
; CONFIG1H
 CONFIG CP0 = OFF ; Code Protection bit (Program memory is not code-protected)
; CONFIG2L
 CONFIG FOSC = HS ; Oscillator Selection bits (HS oscillator)
 CONFIG FCMEN = ON ; Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor enabled)
 CONFIG IESO = ON ; Two-Speed Start-up (Internal/External Oscillator Switchover) Control bit
(Two Speed Start-up enabled)
; CONFIG2H
 CONFIG WDTPS = 32768 ; Watchdog Timer Postscaler Select bits (1:32768)
; CONFIG3L
 CONFIG EASHFT = ON ; External Address Bus Shift Enable bit (Address shifting enabled, address
on
external bus is offset to start at 000000h)
 CONFIG MODE = MM ; External Memory Bus Configuration bits (Microcontroller mode - External
bus disabled)
 CONFIG BW = 16 ; Data Bus Width Select bit (16-bit external bus mode)
 CONFIG WAIT = OFF ; External Bus Wait Enable bit (Wait states on the external bus are disabled)
```

```
; CONFIG3H
 CONFIG CCP2MX = DEFAULT ; ECCP2 MUX bit (ECCP2/P2A is multiplexed with RC1)
 CONFIG ECCPMX = DEFAULT ; ECCPx MUX bit (ECCP1 outputs (P1B/P1C) are multiplexed with RE6 and
RE5; ECCP3 outputs (P3B/P3C) are multiplexed with RE4 and RE3)
 CONFIG PMPMX = DEFAULT ; PMP Pin Multiplex bit (PMP port pins connected to EMB (PORTD and
PORTE))
 CONFIG MSSPMSK = MSK7 ; MSSP Address Masking Mode Select bit (7-Bit Address Masking mode
enable)
 ; TCON setup:
 ; TMR1ON = 0
 ; TMR1CS = 1
 ; T1SYNC = 1
 ; T1OSCEN = 1
 ; T1CKPS0 = 0
 ; T1CKPS1 = 0
 ; T1RUN = 0
 ; RD16 = 0

 ;TMR1H&TMR1L: RegValue =
 ;32kHz/4 = 8kHz
 ;1ms= (1/(8khz))*(FFFF)-init
 ;init=65536-8=65528 = FFF7
org 0x0
goto start
settimer:
 bcf T1CON,TMR1ON
 ;Write into TMR1H first and then TMR1L
 movlw 0xFF
```

```
 movwf TMR1H
 movlw 0xF7
 movwf TMR1L
 ;Turn on timer1
 bsf T1CON, TMR1ON
 return



start:
 bcf TRISE,3
 movlw 0x14
 movwf T1CON
restart:
; Clear TMR1IF and go to set the timer
 bcf T1CON,TMR1IF
 call settimer
;Stay in the loop untill interrupt flag is set
loop:
 btfss PIR1, TMR1IF
 bra loop
 ;Turn off timer1
 bcf T1CON,TMR1ON
 ;Toggle PRTE.3
 btg PORTE,3
 ;Go back to restart
 Bra restart
 end
```

3 SELF-REFLECT

Timers are trivial to write. The challenges for me came from trying to understand how to calculate the initial value. Since I have little or no practice how to change mHz to time. Once I understood how to change them it made a lot more sense how to calculate these. Also I should have verified from the data sheet what register to use in which timer. This was the reason why my program didn't work first. I was using wrong registers to check the overflow flag.