

**City University of Hong Kong**  
**Department of Electronic Engineering**

**EE2004**

**Microcomputer Systems**

**Laboratory Manual**

**Important notes to students:**

Laboratory work assessment consists of four components: Prelab exercises, Demonstration on checkpoints, Logbook and Laboratory report.

1. Prelab exercises: Starting from Lab 2, students must complete the assigned prelab exercises before attending the laboratory session. Answers to prelab exercises must be written in a logbook dedicated to this course. **Each group will make one submission to the lab instructor when entering the lab. Students failing to submit the prelab exercise within the beginning 20 minutes of the lab session will receive a zero for the prelab component.** The prelab exercises will be marked on a pass-or-fail basis and the logbook will be returned to students immediately after a grade is assigned so that students are able to use the same logbook to record findings in the lab session. The grade you receive in prelab exercises carries a weight in the calculation of your overall laboratory score in the course.
2. Demonstration on checkpoints: Students should demonstrate their work to a demonstrator or the laboratory supervisor at each checkpoint. Students should also answer all questions marked with (?) in the logbook. These answers will be checked by the demonstrator or the laboratory supervisor at each checkpoint. Checkpoints are numbered and clearly indicated in the laboratory manual. Some tasks in this laboratory manual are optional. However, students will obtain a higher mark for the laboratory section of the course if they perform these optional tasks and have the results demonstrated to the demonstrator or laboratory supervisor.
3. Policy for early leave: Students finishing the lab exercise before the end of the lab session can choose to leave early. However, students choosing to leave early **MUST** get approval from the lab supervisor. The lab supervisor will ensure that the tasks have been demonstrated, and that the questions marked with (?) have been answered in a satisfactory manner before allowing a student or a group of students to leave the session.
4. Logbook: Each group must keep a logbook to record of their laboratory work, and submit the logbook to Mr. Szeto by Friday of Week 14. Mr. Szeto will assess the logbook based on how thorough students have responded to questions in the lab manual. **Students are only allowed to submit the logbook once.** Therefore, please finalize your logbook before the submission.
5. Lab report: Students will choose to write a report on one of lab sessions from Labs 2-5. This report will be submitted via Canvas by Friday of Week 13. Submission instructions are provided later in Canvas.

### **Overall objectives for the laboratory sessions in EE2004**

- Use the PIC18 microcontroller program development environment to edit, assemble and link programs.
- Use the in-circuit debugger (ICD) and simulator (MPSIM) to debug PIC18 programs.
- Use the PIC18 ports:
  - to drive 7-segment displays and
  - to read input from a key matrix
- Implement a frequency measurement device using the PIC18 on-chip timers and counters.
- Apply the following two serial communication protocols to interface between PIC18 microcontroller and various devices:
  - Asynchronous serial communication: USART to PC serial port (COM)
  - Synchronous serial communication: I<sup>2</sup>C to EEPROM
- Use the logic analyzer and the oscilloscope to validate various applications.

## **Table of Contents**

Laboratory 1: Simple LED HEX Counter

Laboratory 2: Driving the 7-Segment LED module

Laboratory 3: 8 discrete button input and 4 x 4 Key matrix input

Laboratory 4: Implementation of a frequency counter using PIC18 timer module

Laboratory 5

Part A: Asynchronous serial communication using PIC18 UASRT module

Part B: Interfacing with EEPROM 24C02 using PIC18 I<sup>2</sup>C bus

## **Equipment**

1. ND118-4520 development kit equipped with PIC18F4520 Microcontroller
2. Microchip's PICkit3 In-Circuit Debugger/Programmer
3. IBM PC or equivalent with at least 512MB extended memory recommended
4. USB cables connecting Equipment 3 with Equipments #1 and #2
5. 9-pin RS-232 Extension Cable connect Equipment #3 to Equipment #1
6. Tektronix Oscilloscope (TDS3032 or equivalent)
7. Saleae logic analyzer
8. Function Module #1
9. MPLAB IDE V8, Saleae Logic 1.1.12

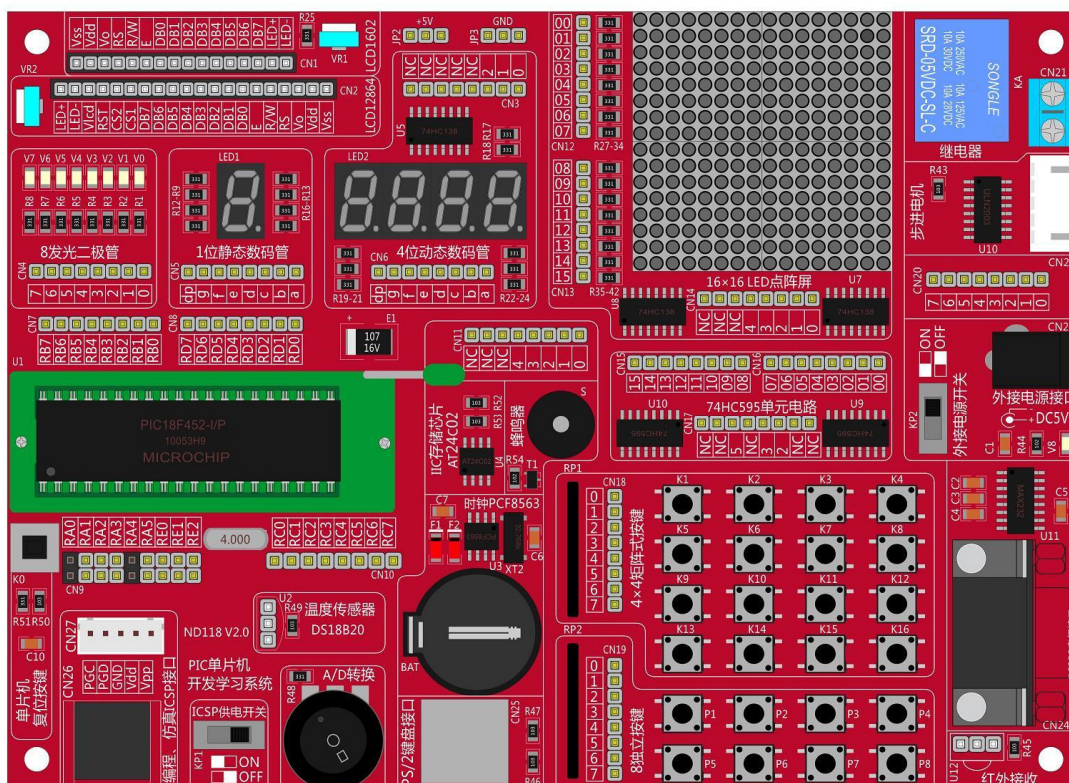
## **REFERENCES**

1. Huang, HW. "PIC Microcontroller: An Introduction to Software and Hardware Interfacing." Delmar Cengage Learning, Clifton Park, NY. 2007
2. MPASM™ Assembler, MPLINK™ Object Linker MPLIB™ Object Librarian User's Guide
3. PICkit™ 3 Programmer/Debugger User's Guide
4. Data Sheet of PIC18F4520, 24C02, 74HC138, MCP4921, NE555

## **ND118-4520 Development Kit**

ND118-4520 Development Kit is a single board multi-function PIC18F4520 microcontroller platform. The development kit is designed based on the concept of modular open systems design. You may upload programs to the system kit from a PC via the USB port connected to PICkit3 In-Circuit Debugger for program debugging and code execution. In this course, you will use only part of the module.

The layout of the ND118-4520 Development Kit is shown below:



### Figure 1 - ND118-4520 Development Kit Layout

### ND118-4520 Schematic

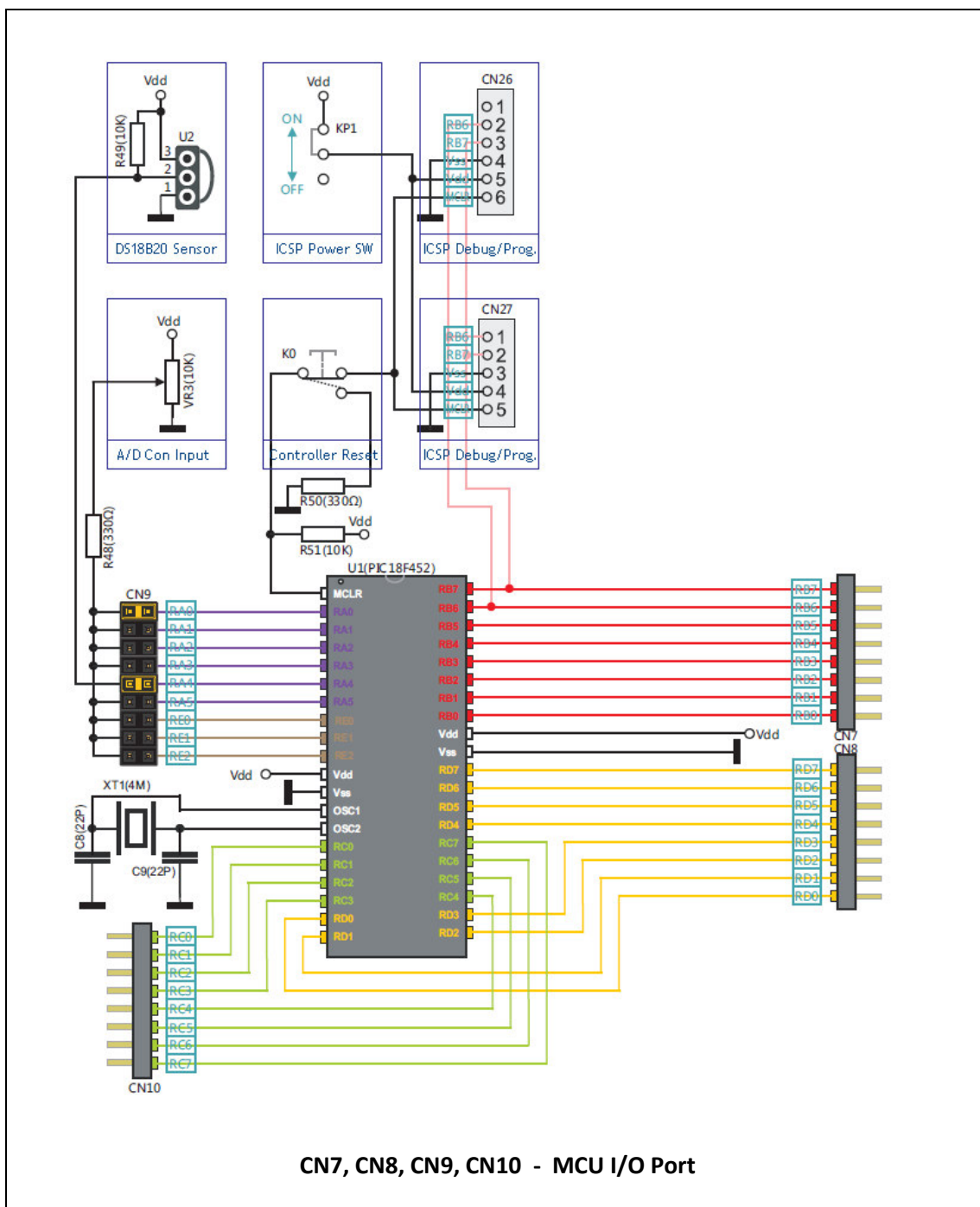
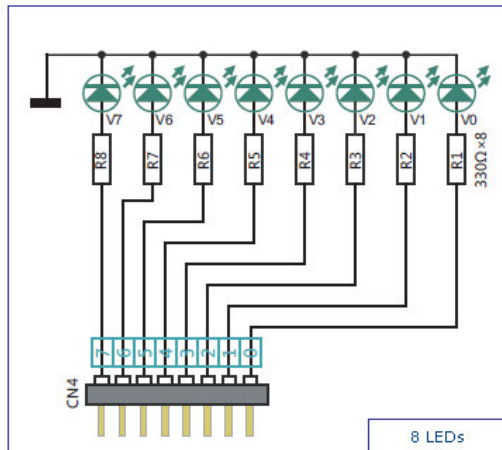
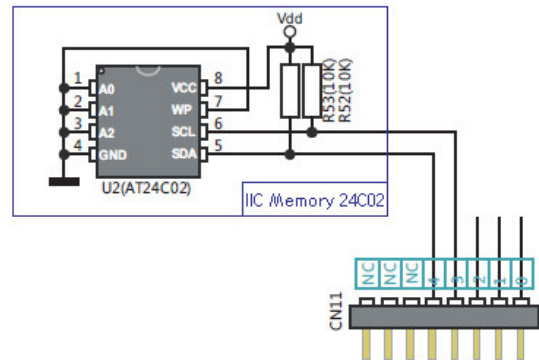


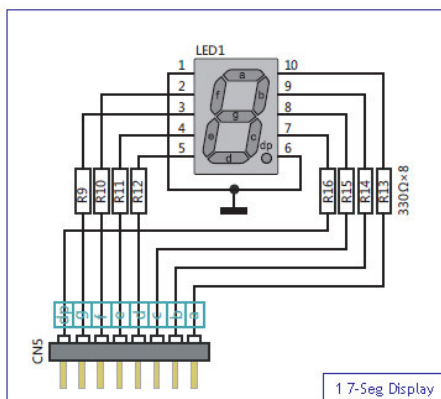
**Figure 2 - ND118-4520 Schematics (PIC18F4520)**



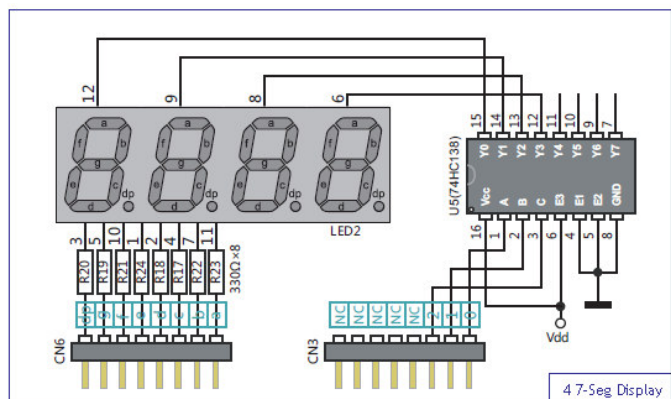
**CN4 – 8 x LED**



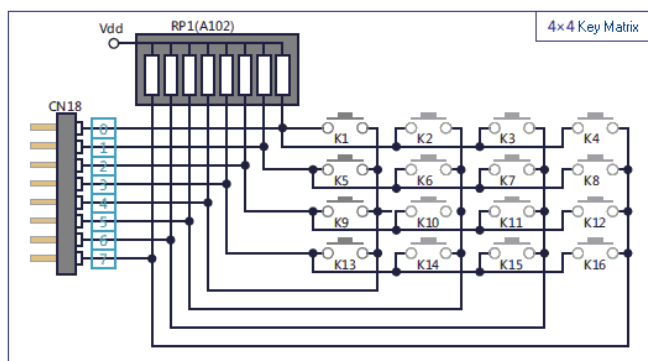
**CN11 - 24C02 EEPROM**



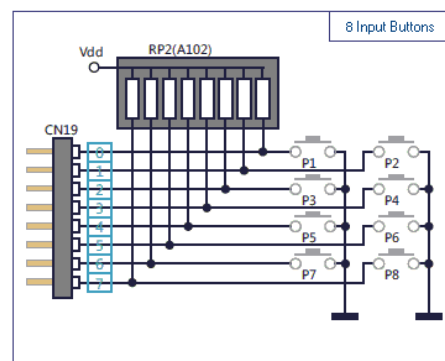
**CN5 - 7Seg LED**



**CN6, CN3 - 4 x 7Seg LED**



**CN18 - 4x4 Key Matrix**



**CN19 - 8 Key Input**

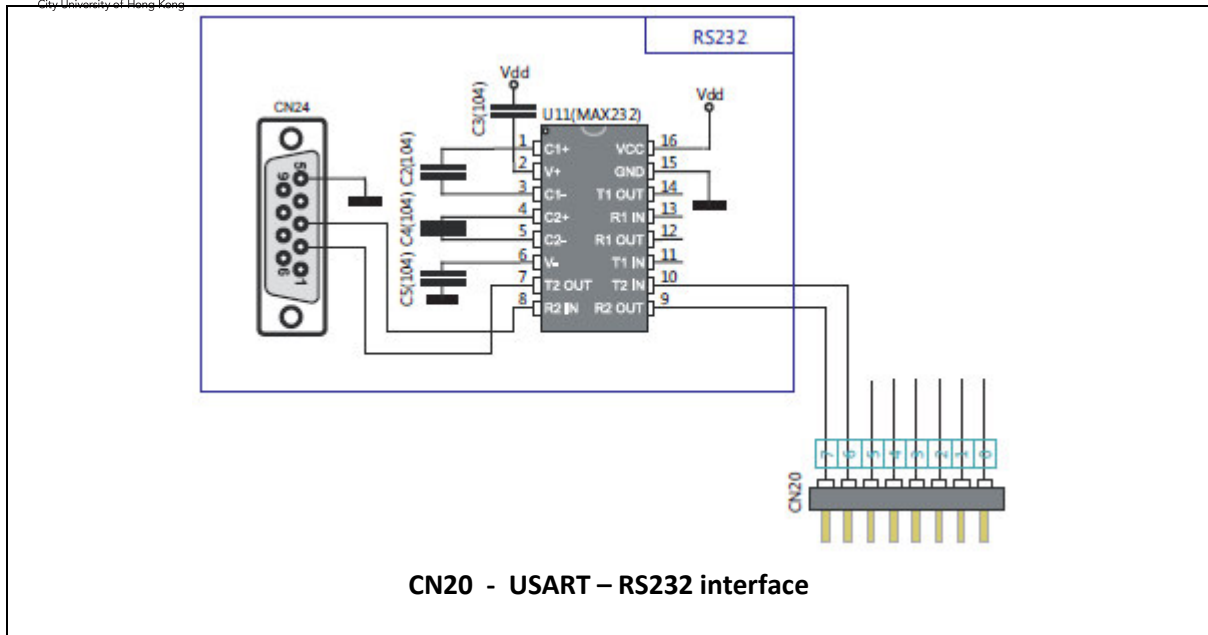


Figure 3 - ND118-4520 Schematics (Peripherals)

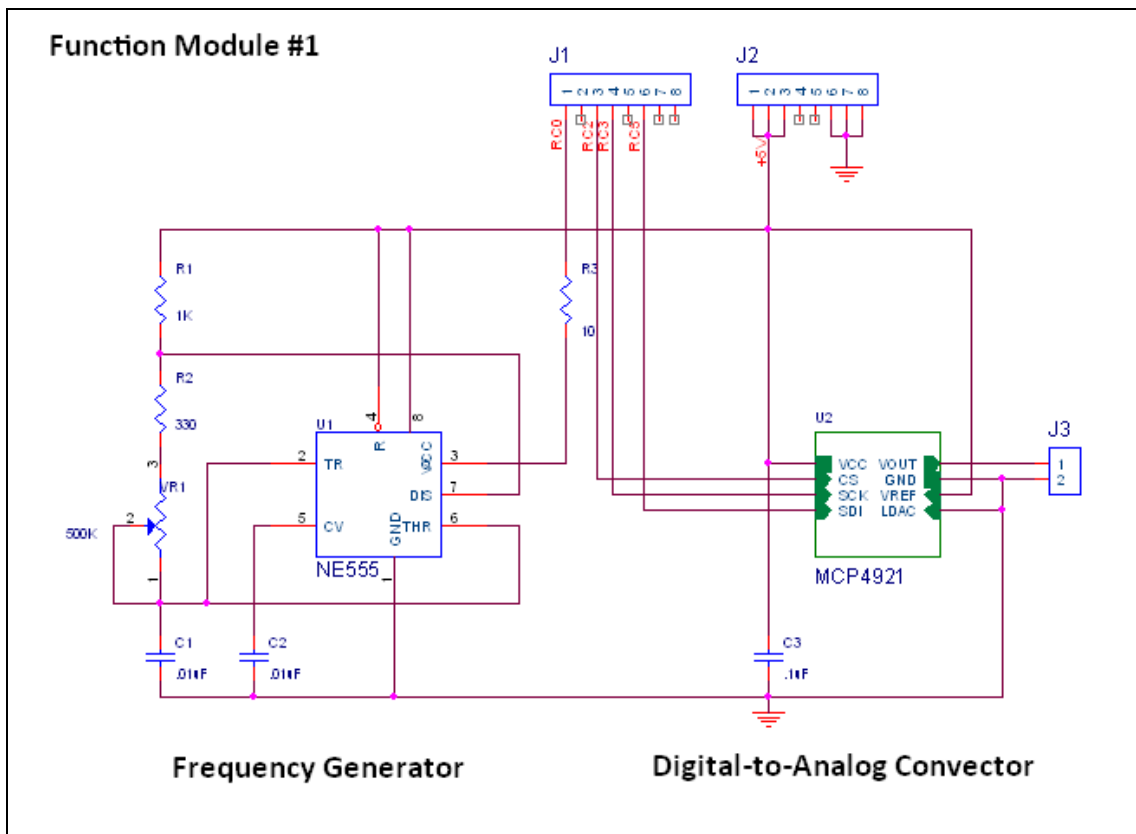


Figure 4 – Function Module #1 Schematics

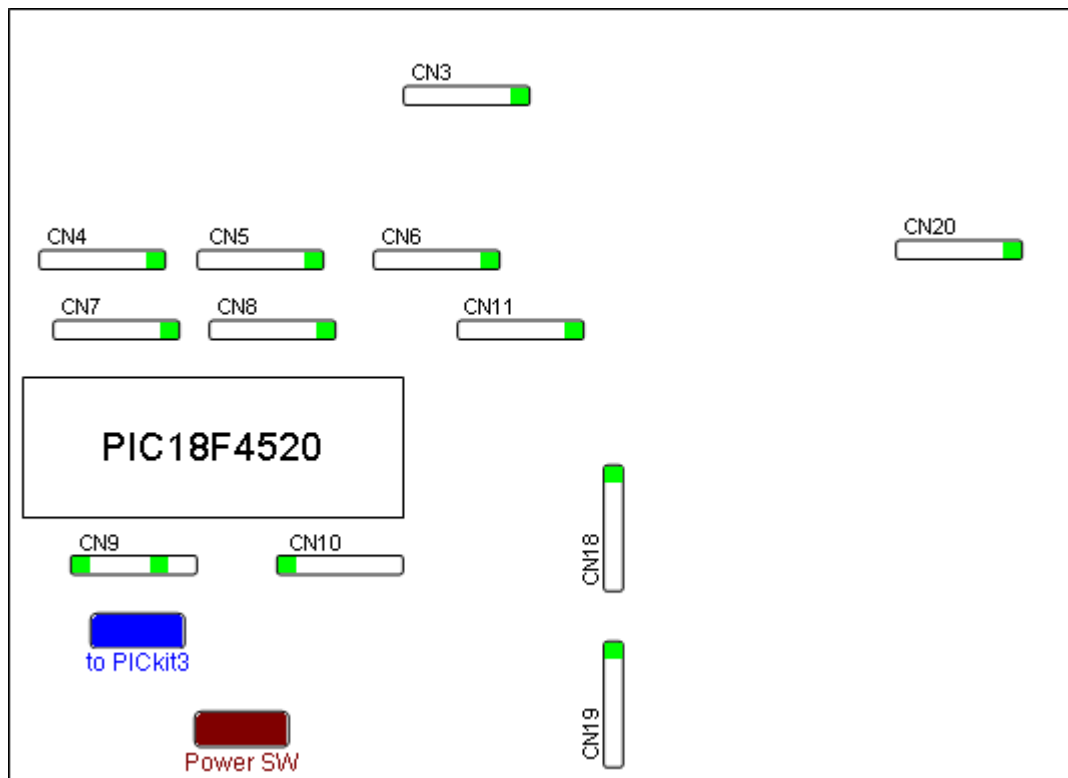


Figure 5 - Connector Location (only connectors used in this Lab. are shown)

❖ *Note: Green mark indicate the lowest order bit*



Figure 6 – PICkit3 ND118-4520 Connection Diagram

## Device Programmer

The software program must be programmed into the on-chip program memory of the microcontroller by a device programmer before it is tested. This programmer is connected to serial/USB port of a PC and is controlled by the MPLAB IDE software.

## In-Circuit Debugger

PICKit 3 is designed to be a low-cost debugging device that supports In-Circuit Serial Programming (ICSP) protocol. To enable in-circuit debugging, PICKit 3 programs a small debug executive module into target PIC microcontroller device. This debug executive module is stored at the end of the program memory. The debug executive works by communicating with special on-chip functions built inside the PIC18 microcontroller.

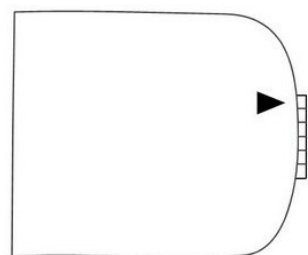
The PICKit 3 module contains the logic for debugging, programming and control. It also acts as a device programmer.

## PICKit 3 Features

- USB (Full speed 12 Mbits/s interface to host PC)
- Real-time execution
- MPLAB IDE compatible (free copy included)
- Built-in over-voltage/short circuit monitor
- Firmware upgradeable from PC/web download
- Totally enclosed
- Supports low voltage to 2.0 volts (2.0v to 6.0v range)
- Diagnostic LEDs (power, busy, error)
- Read/write program and data memory of microcontroller
- Erase of program memory space with verification
- Freeze-peripherals at breakpoint
- Program up to 512K byte flash with the Programmer-to-Go

## NOTES:

- ❖ The connector uses two device I/O pins (**RB6 & RB7**) and the reset line (**MCLR**) to implement in-circuit debugging and In-Circuit Serial Programming™.



| Pin Descriptions |              |
|------------------|--------------|
| 1                | VPP/MCLR     |
| 2                | VDD Target   |
| 3                | Vss (Ground) |
| 4                | ICSPDAT/PGD  |
| 5                | ICSPCLK/PGC  |
| 6                | Auxiliary    |

**Note:** The connector is a 6-pin header with 0.100" spacing and can accept 0.025" square pins.

**Reading Material:** PICKit™ 3 Programmer/Debugger User's Guide

[ww1.microchip.com/downloads/en/DeviceDoc/PICKit\\_3\\_User\\_Guide\\_51795A.pdf](http://ww1.microchip.com/downloads/en/DeviceDoc/PICKit_3_User_Guide_51795A.pdf)

## Laboratory 1: Simple LED HEX Counter

### Background:

The purpose of this laboratory is to familiarize students with the operations of I/O ports. Students will connect Port D of the microcontroller to the LEDs. As the contents of Port D change, the LEDs switch on and off accordingly.

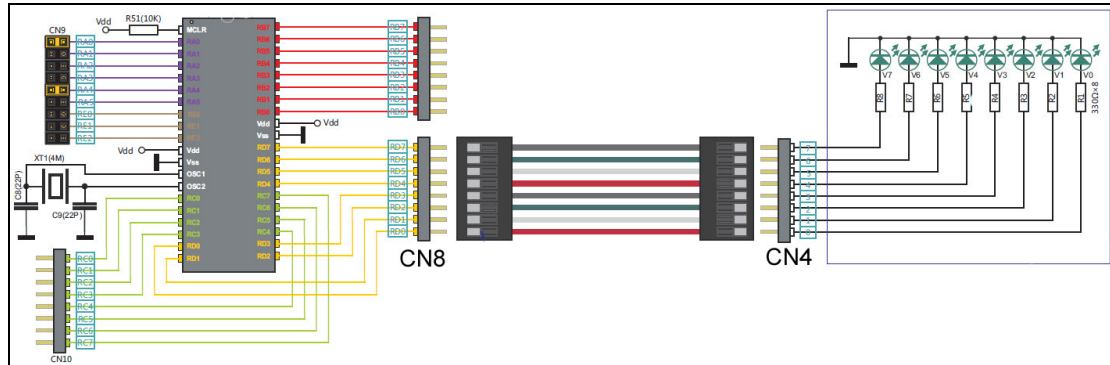


Figure 7a – Schematic for circuit connection

### Hardware connections:

1. Connect the PICKit3 as shown in the Fig. 6.
2. Connect CN4 with CN8 using a ribbon cable. Please pay attention to the *orientation* of the parallel connection (i.e., RD<x> of CN8 should be connected to pin<x> of CN4, where <x> = 0 to 7. See Fig. 7b)

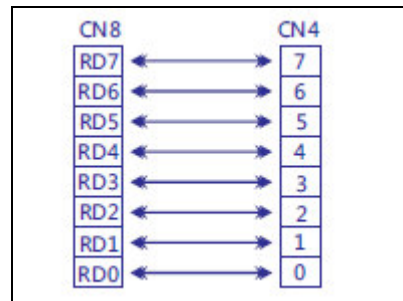


Figure 7b - Cable connection diagram

### Task 1: Assemble program and build machine code

1. Run the MPLAB IDE program.
2. Click **“File”**; **“New”** and type the following code into the file editor window.

```

LIST    P=18F4520, R=DEC    ;define processor, Decimal radix
#include <P18F4520.INC>      ;CPU specific variable definitions

;-----
;Configuration bits definitions
;
        CONFIG OSC = HS, WDT = OFF, LVP = OFF, PBADEN = OFF

;-----
;Variable definitions
;
        CBLOCK 0x000
        DELAY_H
        DELAY_L
        ENDC

;-----
;Reset vector
;
        ORG    0x0000        ;code origin, program start from here
        goto   Main

;-----
;Start of main program
;
Main:    movlw   0x0F          ;Set all Ports digital I/O
        movwf  ADCON1
        clrf   TRISD         ;Set PortD Direction "Output"
        clrf   PORTD

MainLoop: incf    PORTD        ;increment PortD value
        call   Delay
        bra    MainLoop

;.....
Delay:   clrf    DELAY_H      ;Delay Subroutine
        clrf    DELAY_L
DelayLoop: decfsz DELAY_L
        bra     DelayLoop
        decfsz  DELAY_H
        bra     DelayLoop
        return

;-----
        END                  ;End of program

```

**Figure 8 - Task #1 Program Code**

- **Note:** Each line of the source file contains up to four components:
- Labels,
  - Mnemonics, Directives or Macros,
  - Operands,
  - Comments

The order and position of these are important. **Labels** should start in **column one**, mnemonics and other fields should start in **column two or beyond**.

3. Click **“File”**, **“Save As”**. Create and select the **“C:\Code\Expt1”** folder and type **“led.asm”** as the program file name. (Make sure you save the file into the Expt1 folder).

4. Click "Project", "Project Wizard...", "Next >", select device "PIC18F4520", click "Next >", select "Microchip MPASM Toolsuite", click "Next >".
5. Select the item "Create New Project File" and click "Browse". A new window with title "Save Project As" pops up. Choose the folder "C:\Code\Expt1" in the window, type "Expt1" in the "File name" field and click "Save".
6. Click "Next >" and expand the folder tree and locate the file **led.asm**. Click "Add >>" and "Next >" to insert the **led.asm** file to the Project. Check the project parameters list and click "Finish" to finish the project definition process.
7. Click "Project", "Build All" and select "Absolute".
8. "**BUILD SUCCEEDED**" should appear at Output window. Should "**BUILD FAILED**" appear instead, read the error messages, fix the errors and repeat the build process described in Step 7.

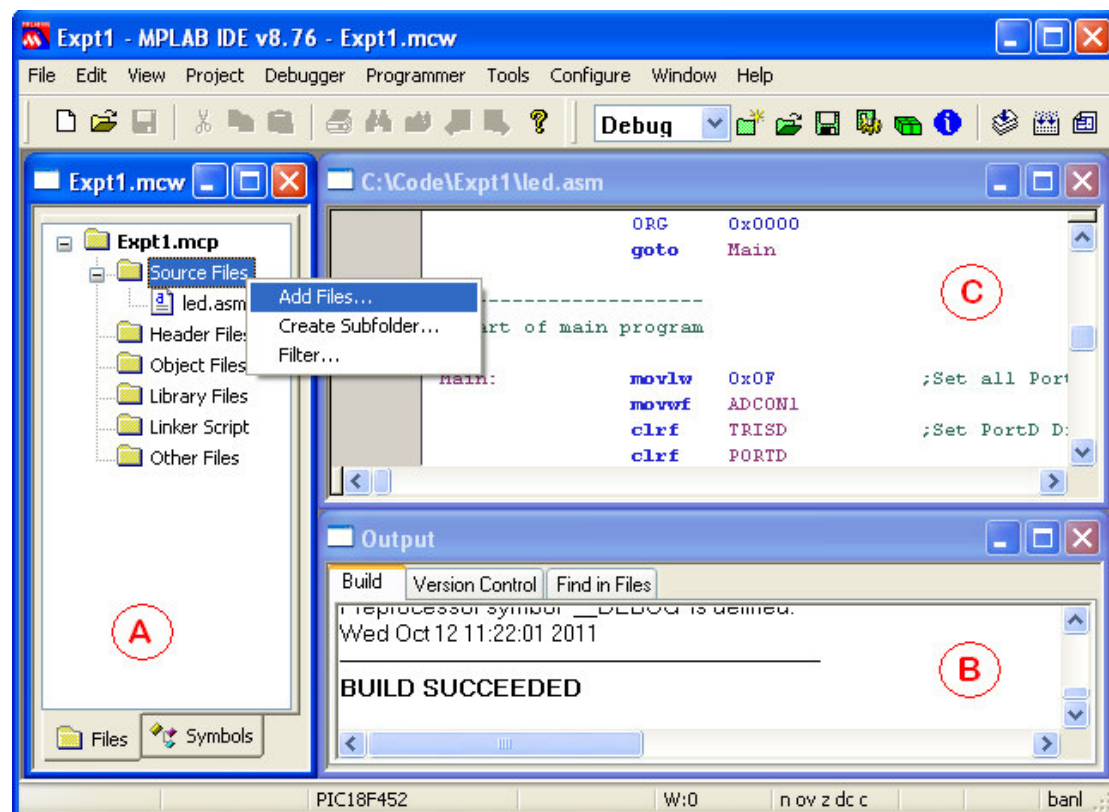


Figure 9 - MPLAB IDE Screen Example

A. Project window      B. Output window      C. Source file window

9. Examine the Project folder "C:\Code\Expt1". Find a file called **led.hex**. The program code **led.asm** was translated by the assembler to PIC18 executable machine codes and written to the file **led.hex** in intel Hex format.
10. Click "File", "Save Workspace" to save your work. By doing this, all your current project-related parameters are saved into the file with extension "mcw". You can double click file **Expt1.mcw** later to continue your development.

## **Task 2: Device Programming – Burn code into Device Flash**

11. Connect one end of the PICkit3 In-Circuit Debugger to PC using a USB cord and the other end to the ND118-4520 Development board. (See Fig. 6). Connect pin connector CN4 to CN8 with a ribbon cable. (See Fig. 7a & b)
12. Set ICSP Power Switch to “ON” position. Doing this allows the PICkit3 to supply power to the development board. Click “**Programmer**”, “**Setting...**”, “**Power**” and check the “**Power target circuit from PICkit 3**” check box. Click “OK”. You should see an LED at right edge of the development board turned on, indicating system power is ON.
  - ❖ **NOTE:** Debugger & Programmer ICSP Power Setting are set to **OFF** by default in order to protect target board devices.
13. Click “**Programmer**” and “**Reconnect**” to make sure connection is well established. Output screen should display a similar screen as shown below:

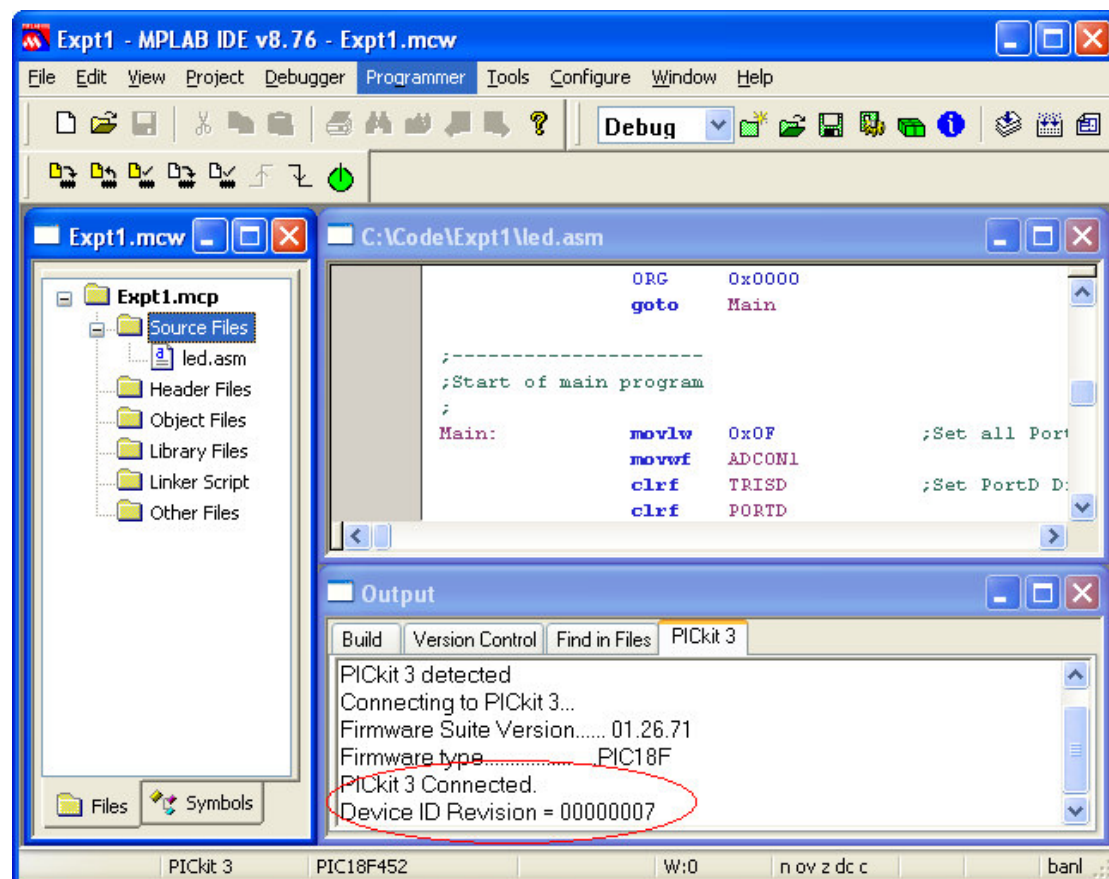


**Figure 10 - MAPLAB IDE & PICKIT3 connected output**

14. Click “**Programmer**”, “**Program...**”. MPLAB IDE changes from DEBUG to RELEASE mode and a dialog box pops up to ask you whether to assemble the code again. PICKIT3 then begins the device programming process. “Programming/Verify complete” should be shown at output window when programming completes and the LED should blink accordingly.

### Task 3: In-Circuit Debugger (ICD)

You would usually encounter errors when you first test the code you have written. Program debugging is required to locate the errors. PICKit3 is not only a device programmer but also a tool for code debugging.

15. Click **“Debugger”, “Select Tool”, “2 PICKit 3”** to start an ICD connection. You may need to turn on the PICKit3 ICSP power feature, which can be done by clicking **“Debugger”, “Settings...”, “Power”** and check the **“Power target circuit from PICKit 3”** check box. Board power status is always indicated by the LED located at the right edge of the board.
16. You can also click **“Debugger”, “Reconnect”** to restart an ICD session. Session connection status will be shown in the Output window. Note that a green arrow now appears in the source file window indicating the initial Program Counter position. A new icon tab for debugging also appears. You may use it as a debug command shortcut.

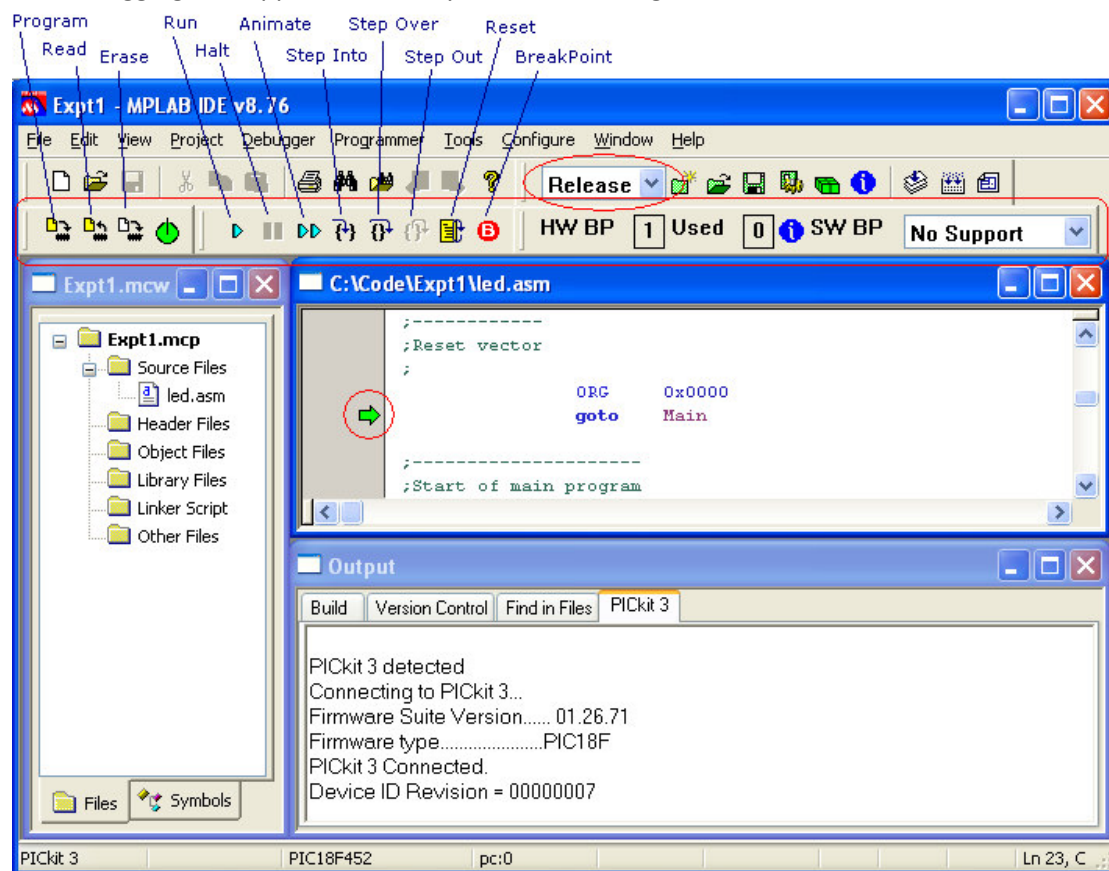


Figure 11 – MPLAB Debugger activated Screen

17. Click **“Debugger”, “Program”** to program the code into the microcontroller. You may be asked to rebuild the code again. Note that the Build Configuration will be changed from **Release** to **Debug** after rebuild. This change of Build Configuration is necessary because the debugger requires a small monitor agent and program debug information to be loaded to the microcontroller, which is only performed with Build Configuration set to **Debug**.
18. Click **“Debugger”** and play with the **“Run”, “Animate”, “Halt”** and **“Reset”** debugger commands. Take note of the position of the green arrow in the source file window and how the code affects blinking status of the 8 LEDs on the board.

19. Click “View”, “Watch” to bring out the Registers watch window. Select “PCL” from the **left listbox** and click “AddSFR”. Repeat the above actions for “WREG” and “PORTD”, select “DELAY\_H” and “DELAY\_L” from **right listbox** and click “Add Symbol”. After adding these registers, you should see the following screen:

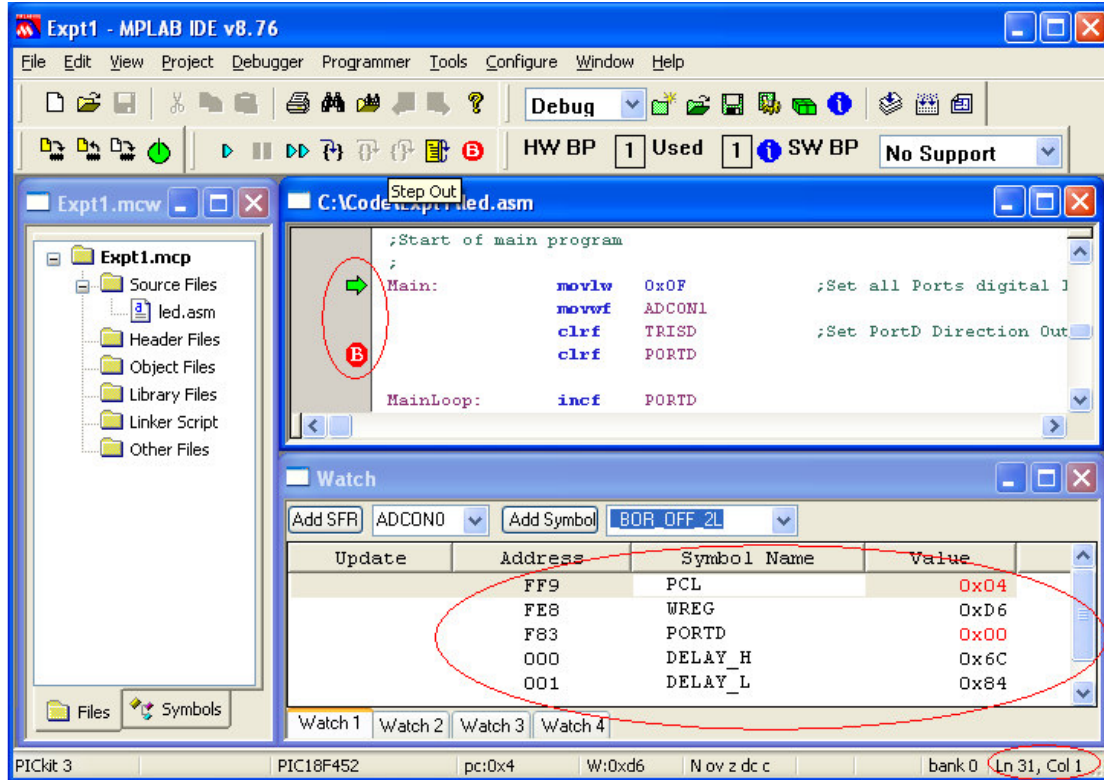


Figure 12 – MPLAB Debugger with Register Watch Point

20. Double-click the source file windows at line #29 “clrf PORTD”. A breakpoint mark appears at the left indicating a program breakpoint has been set at this location. Click “Debugger”, “Reset”, “Processor Reset” to reset the Program Counter (PC) to the program starting point (i.e., 0x0000). (See Fig. 12)
21. Click “Debugger”, “Step Into” to execute one line of instruction. Observe the change of the values of the registers in the Watch window.
22. Click “Debugger”, “Run” to allow the processor to run the remaining code at full speed until a Breakpoint is met. Play with the four debugger commands listed in Step 18 and notice how the register values are changed.
23. (?) Describe a way to remove a Breakpoint from the code.
24. (?) When the green arrow is pointing at the instruction “call Delay”, click “Debugger”, “Step Into”. Describe your observation. What happens if you click “Step Over” instead?
25. (?) When the program is inside the subroutine Delay, click “Debugger”, “Step Out”. Describe your observation.

26. (?) Remember the 8 LED status on board. Double-click on the value of PORTD register in the Watch window, change its value to 0xF5 and press “Enter” key. Describe your observation.

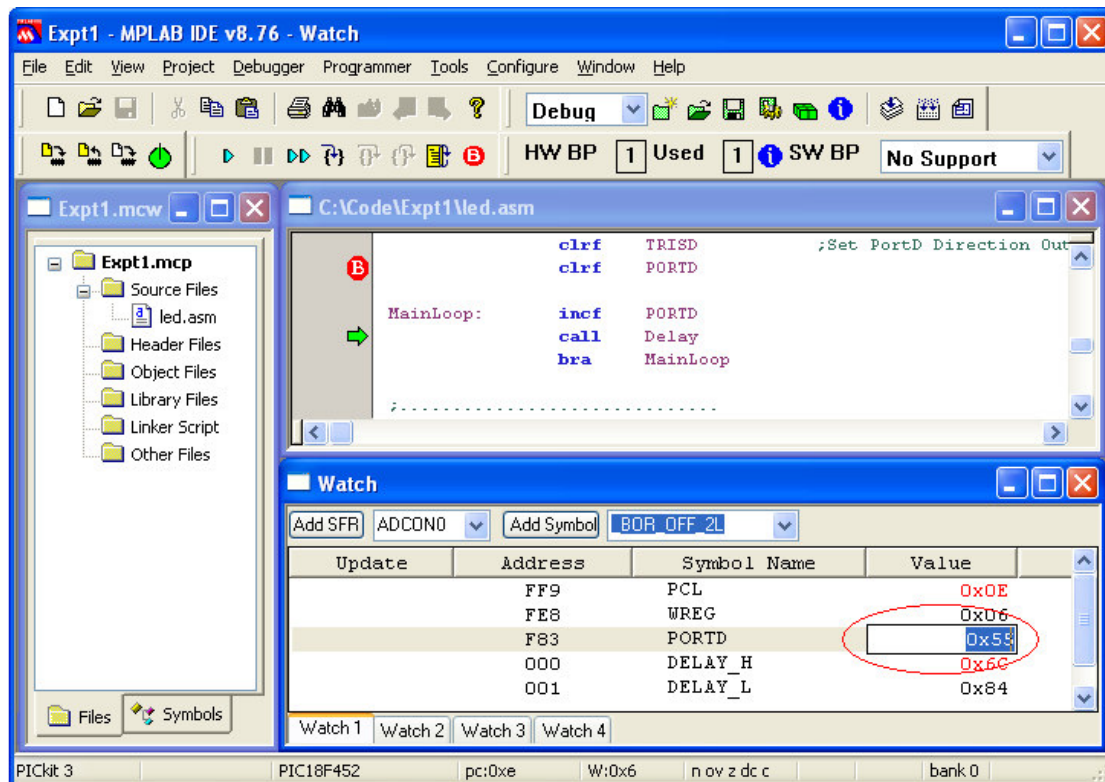


Figure 13 – Modify Register Value

27. (?) Modify the code so that it drives the same 8 LEDs from Port B instead of Port D. Change the hardware connection as well and describe the connection changes you have made. Build the program and download it into the device by clicking “**Debugger**”, “**Program**”, which should be performed every time when the code has been modified. Run the code and observe the status of the 8 LEDs. Add PORTB into the Watch list and repeat Step 26. Repeat Step 22, describe and explain any difference in the behaviour of the LEDs as compared to your previous observation.
28. (?) Repeat Task #2 to activate the **Programmer** (*Release mode*) and burn the code into the device. Do you find any difference in the behaviour of LEDs as compared to Step 27? Explain what has happened.

➤ **Checkpoint 1: Show your work to a demonstrator.**

### Task 4: PC Simulator (MPSIM)

You may not have access to the sophisticated hardware tools such as the PICkit3 and the development board at home to test your program. However, you can still develop your microcontroller projects using MPSIM. MPSIM is a PC simulator that can simulate PIC18 microcontroller internal activities with a host PC. MPSIM allows multiple breakpoints to be set, provides a stopwatch to measure activity timing and accepts stimulus to simulate I/O input activities.

29. Set the board ICSP power switch to the “Off” position. Disconnect PICkit3 from the host PC. Click “Debugger”, “Select Tool”, “4 MPLAB SIM” to start the MPSIM simulator.
30. Reload the code used in Task #1 (Fig. 8). Set two breakpoints at locations before and after the subroutine DELAY (see Fig. 14). Click “Debugger”, “Setting...”, “OSC / Trace”, set Processor Frequency to “4” MHz. and click “OK”.
31. Click “Debugger”, “StopWatch” to bring out the Stopwatch window. Click “Debugger”, “Reset”, “Processor Reset” to reset the microcontroller. Click “Debugger”, “Run”. The processor will stop at the first breakpoint. Note from the Stopwatch window that it took 7 instruction cycles to reach the breakpoint from start. Click the “Zero” button to reset the stopwatch to 0. Click “Debugger”, “Run” again. The processor would then stop at the 2<sup>nd</sup> breakpoint. The Stopwatch window would display that it took 197125 instruction cycles (i.e., 197.125 mSecs) to travel from the first to the second two breakpoints, during which the Delay subroutine executes.

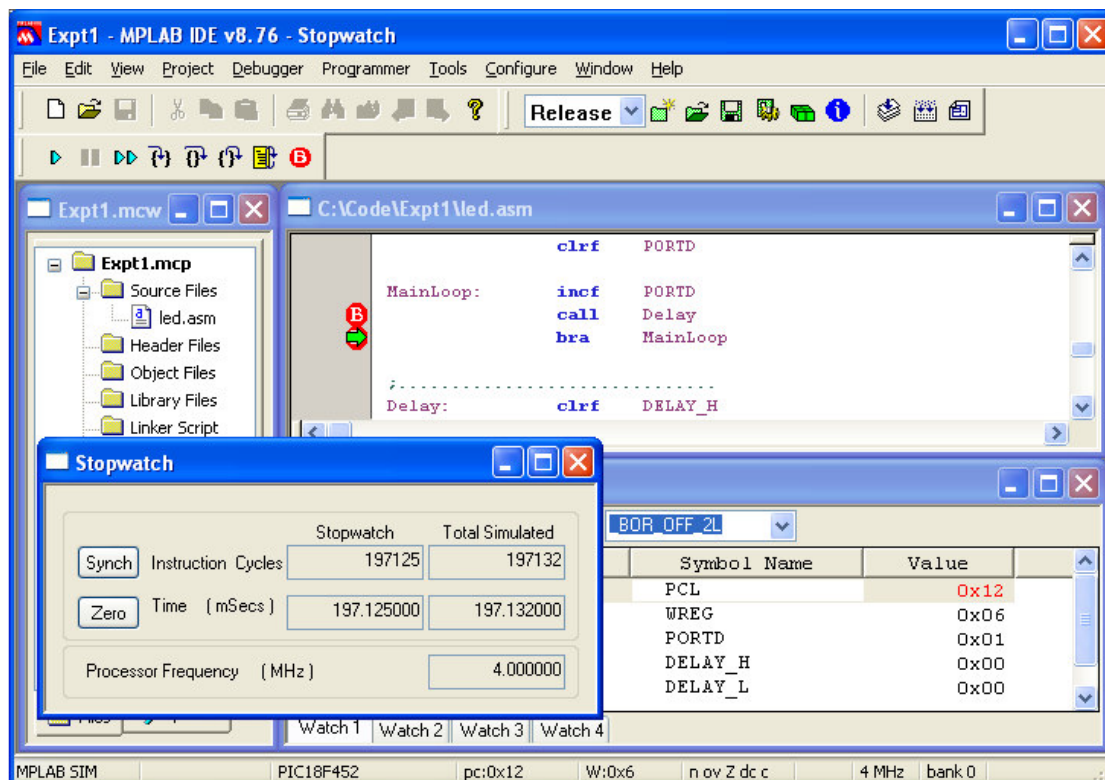


Figure 14 – MPSIM with Stopwatch

32. Modify the program shown in Fig. 8 by adding the following 3 lines of code at location near the MainLoop label (see Fig. 15, Label 1):

```
~~~~~
setf    TRISB           ;Set PortB Direction "Input"
~~~~~

MainLoop:  btfsc  PORTB, RB0
           bra    MainLoop
~~~~~
```

33. Click **"Project"**, **"Build All"** to rebuild the code. Add **"PORTB"** to the Watch window (see **Step 19**). Click **"Debugger"**, **"Stimulus"**, **"New Workbook"** to activate Stimulus. The Stimulus window would pop up. Select the first stimulus entry with **Pin/SFR = "RB0"** and **Action = "Set High"** (see the row labeled 4 in Fig. 15). This stimulus imitates the event in which RB0 (PORTB Bit 0) was set to logic high. Set 2<sup>nd</sup> Stimulus entry with **Pin/SFR = "RB0"**, **Action = "Pulse Low"**, **Width = "10"** and **Units = "cyc"** (see the row labeled 5 in Fig. 15).

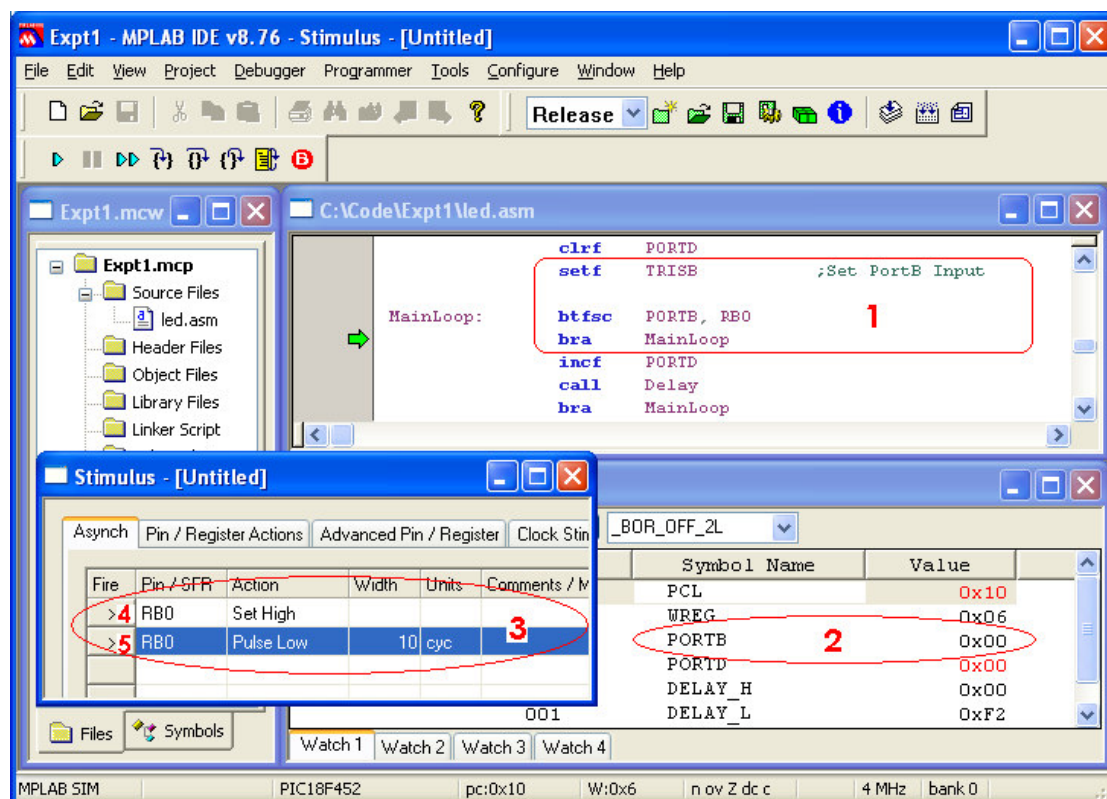


Figure 15 – MPSIM with Stimulus

34. (?) Click **"Debugger"**, **"Reset"**, **"Processor Reset"** to reset the microcontroller. Click Stimulus #1 Fire button **">"** (Fig. 15, Label 4) once to set RB0 to high. Click **"Debugger"**, **"Animate"** to run the code in animation mode. Describe your observation. Click Stimulus #2 Fire button **">"** (Fig. 15, Label 5) to set RB0 to low for 10 cycles. Describe your observation.

➤ **Checkpoint 2: Show your work to a demonstrator.**

### **Task 5: Optional**

Rewrite the program displayed in Fig. 8 so that the LED lights up sequentially from left-hand side until all LEDs are turned on. That is, V7 lights up first, followed by V6 etc. and V0 turns on last. All LEDs stay on after V0 turns on. (Hints: Use the `iorwf` and `rncf` instructions).

➤ **Checkpoint 3: Show your work to a demonstrator.**

## Laboratory 2: Driving 7-Segment LED module

### Background:

In this laboratory, students will drive a 7-segment display using a microcontroller I/O port. 7-segment display consists of 7 LED bar segments. When put together, these 7 LED segments form a numerical digit '8'. In addition, there is an extra LED dot for decimal point display purpose. (We do not use the decimal point in this laboratory). The 7-segment display used on ND-118 kit is of the common cathode type. Through this exercise, students will learn how the software table lookup and the hardware multiplexing techniques are applied.

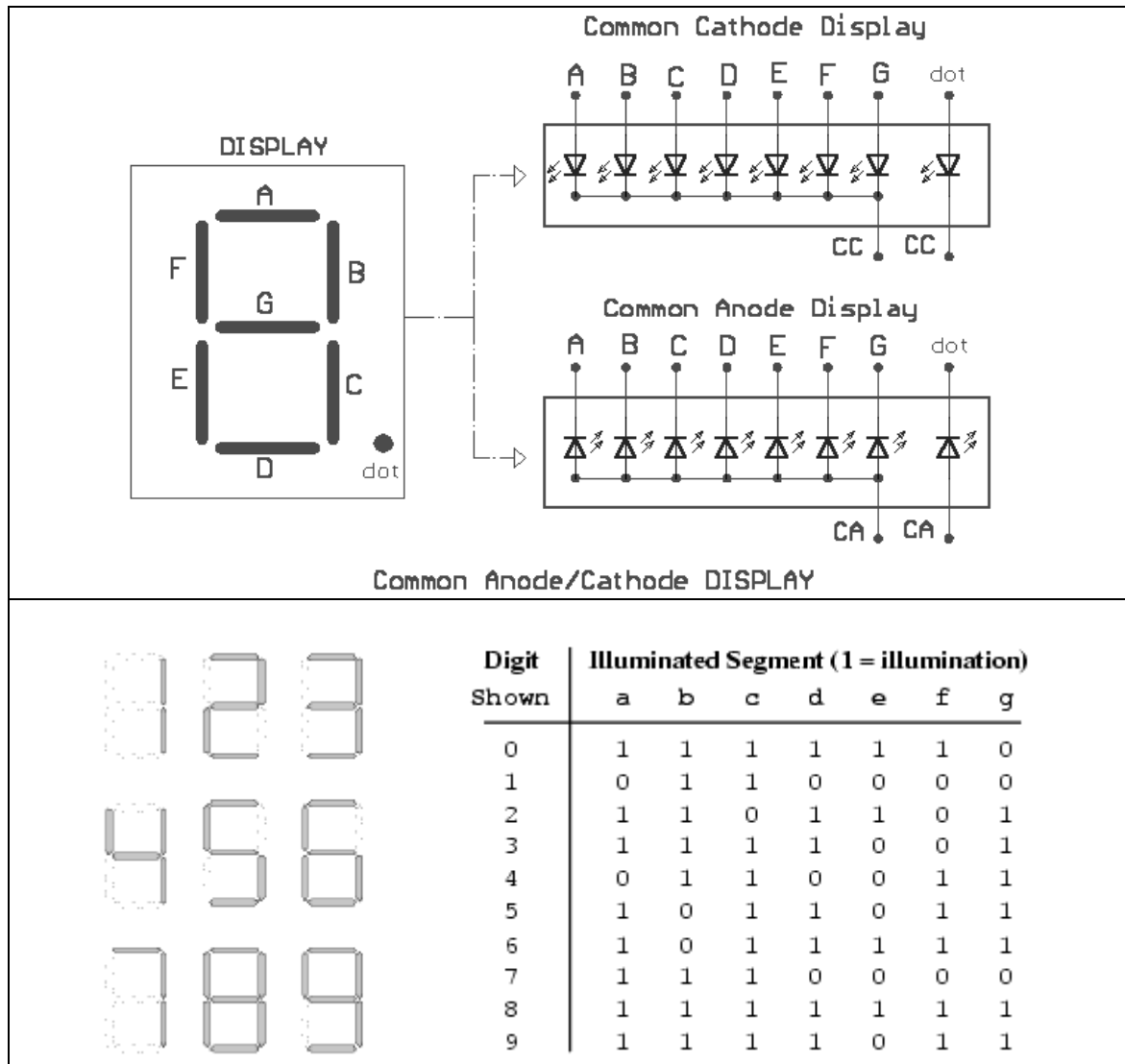


Figure 16 – 7-Segment LED display internal and digit display truth table

### Prelab work:

1. Write a subroutine SVN\_SEG implementing a lookup table for the patterns of the digits 0-9 to be displayed on the 7-segment LED. The input argument of the subroutine is the hexadecimal value of 0 to 9 and the output is the 7-segment pattern of the corresponding input. The subroutine can be implemented using the `retlw` or `tblrd` instructions. Write a program that uses this subroutine to write the 7-segment pattern of the digit '3' to WREG.
2. Describe the role of the 3-to-8 line decoder in the operation of the the four-digit 7-segment LED module. Specification of the 3-to-8 line decoder can be obtained at <http://www.fairchildsemi.com/ds/MM/MM74HC138.pdf>.
3. Explain why four digits can be seen simultaneously in a four-digit 7-segment LED module in terms of the theory of persistence of vision ([http://en.wikipedia.org/wiki/Persistence\\_of\\_vision](http://en.wikipedia.org/wiki/Persistence_of_vision)).

### Task 1 Single digit 7-Segment LED Display

**Objective:** Write a program to show digit pattern 0 to 9.

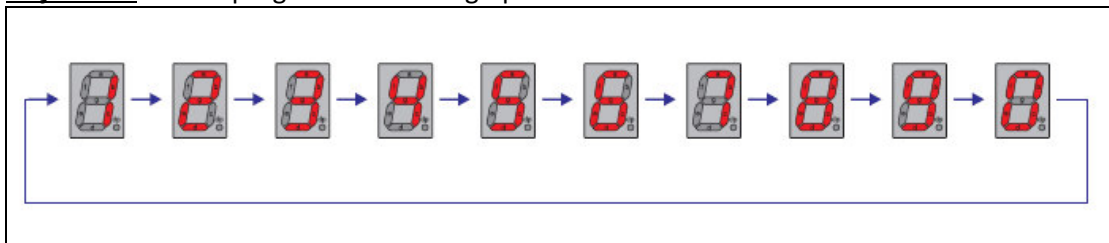


Figure 17a Schematic of the expected result

**Hardware setup:** Connect microcontroller Port D to single digit 7-Segment LED, i.e., connector CN8 – CN5. In your program, please ensure that you have set all I/O ports to be digital using the following two instructions: `movlw 0x0F; movwf ADCON1`.

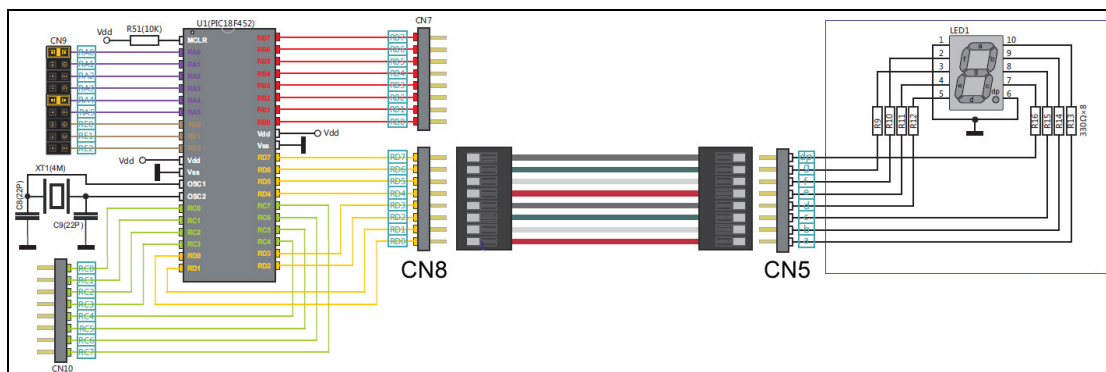


Figure 17b Schematic of circuit connection

**Algorithm:** Write a loop that increments a variable *COUNT* from 0. At each iteration of the loop, obtain the 7-segment pattern corresponding to *COUNT* and send it to PORTD. Reset *COUNT* to 0 after displaying 9.

➤ **Checkpoint 1: Show your work to a demonstrator.**

## Task 2 4-digit 7-segment LED display using time-multiplexing

**Objective:** Write a program to display your group number in the 4-digit 7-segment LED display. For example, if your group number is 4, display 0004.

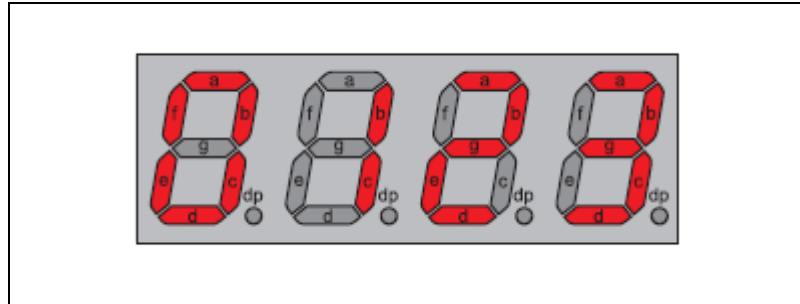


Figure 18a Schematic of the expected result (except that you should display your group number)

**Hardware setup:** Connect microcontroller Port D & Port B to 4-digit 7-segment LED, i.e., connector CN8 – CN6, CN7 – CN3. In your program, please ensure that you have set all I/O ports to be digital using the following two instructions: `movlw 0x0F`; `movwf ADCON1`.

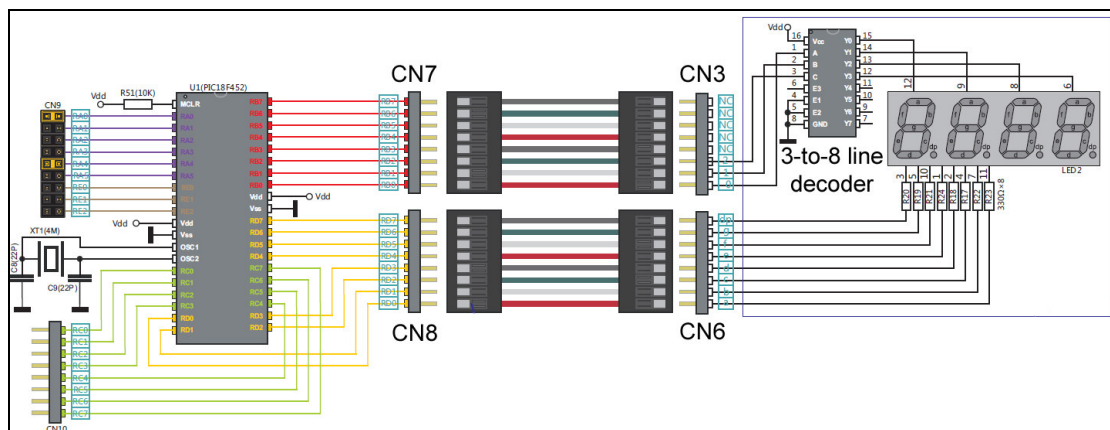


Figure 18b Schematic of circuit connection

**Algorithm:** Your group number consists of four digits. Display the most significant digit in the leftmost digit of the 4-digit 7-segment LED by appropriately setting (i) the input digit pattern and (ii) the input of the 3-to-8 line decoder. Control the duration for which this digit is displayed by calling a delay subroutine (see Fig. 8). Display the second, third and fourth digits similarly by setting the two inputs listed above followed by a call of the delay subroutine.

**Question for discussion (Give your answer in your logbook):**

(?) Vary the duration of the delay time generated by the delay subroutine and describe the effect of the length of the delay time on how the 4 digits are displayed. Relate your observations with the theory of “persistent of vision” you learned in your prelab work.

➤ **Checkpoint 2: Show your work to a demonstrator.**

### **Laboratory 3: 8-button input and 4 x 4 key matrix input**

#### **Background:**

In this laboratory, you will learn how devices can be used input data via I/O ports to the microcontroller, which in turns output data to I/O ports to drive the 7-segment LEDs.

#### **Prelab work:**

You are provided a template file called Lab3Template.asm below. Write codes to choose input/output setting appropriately for PORTC and PORTD in Tasks 1 and 2.

```

;*****
; Template for EE2004 Lab 3
; This template is designed for student to implement Tasks 1 and 2 in Lab 3.

; Student should work on the following three components for implementing the two
; tasks in Lab 3:

; (1) Select input/output setting of PORTD and PORTC
; (2) Implement the KeyScan subroutine. This subroutine returns the identification
; number of the button being pressed.
; (3) Implement the SVN_SEG subroutine as a lookup table.
;*****

LIST      P=18F4520, R=DEC
;define processor, decimal radix
;include <P18F4520.INC>
;processor specific variable definitions

;-----
;Configuration bits, see above .inc file for details on notation.
;
;          CONFIG  OSC = HS, WDT = OFF, LVP = OFF, PBADEN = OFF
;-----
;Variable definitions
;
;          CBLOCK  0x000
;          DELAY_H
;          DELAY_L
;          PAT_NUM
;          ENDC
;-----
;          ORG      0x0000
;          goto     Main
;go to start of main code

;-----
;Start of main program

Main:      movlw    0x0F                      ;Set all Ports digital I/O
           movwf    ADCON1

           ; **Here, student should choose
           ; the input/output settings of PORTC and PORTD (Prelab)

MainLoop:  call     KeyScan
           movwf    PAT_NUM
           call     SVN_SEG
           movwf    PORTD
           call     Delay
           bra      MainLoop

;-----
; **Students should implement the SVN_SEG subroutine. This subroutine returns the 7-segment
; LED pattern corresponding to the value stored in WREG. The subroutine can be implemented
; using the retlw or tblrd instruction.

SVN_SEG:

; **Student should implement the KeyScan subroutine, which returns the identification
; number of the button being pressed and stores the number in WREG.

KeyScan:

;-----

```

```

Delay:      clrf    DELAY_H
            clrf    DELAY_L
DelayLoop:  decfsz  DELAY_L
            bra     DelayLoop
            decfsz  DELAY_H
            bra     DelayLoop
            return

;*****
;End of program
;
            END
  
```

Figure 19 Lab 3 template

### Task 1 8-button input

**Objective:** Write a program to show digit <X> on the 7-Segment LED display when button P<X> is pushed, where <X> = 1 to 8.

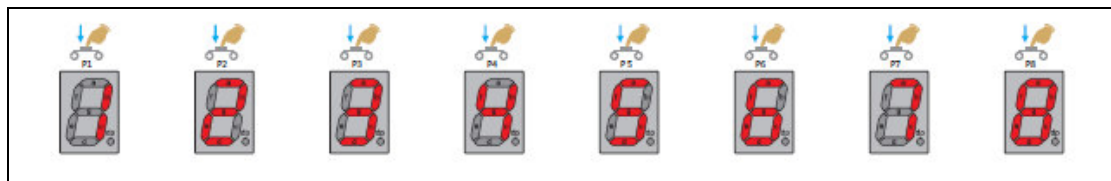


Figure 20a Schematic of the expected result

**Hardware setup:** Connect microcontroller Port D to 1-digit 7-Segment LED, Port C to 8-Button input, i.e., connector CN8 – CN5, CN10 – CN19. In your program, please ensure that you have set all I/O ports to be digital using the following two instructions: `movlw 0x0F;` `movwf ADCON1.`

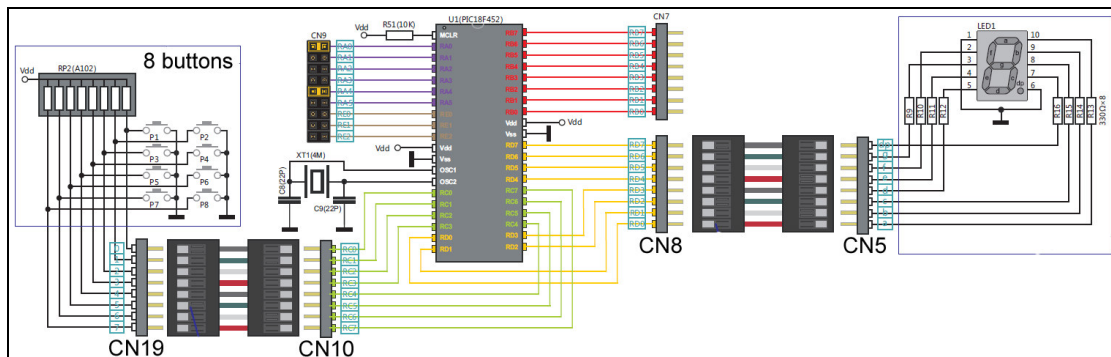


Figure 20b – Schematic for circuit connection

**Algorithm:** Please refer to Fig. 19. Upon returning to the main program from the KeyScan subroutine, the identification number of the button being pressed should be stored in WREG (i.e., <X> is stored when P<X> is being pressed, where <X> = 1, 2, ..., 8).

➤ **Checkpoint 1: Show your work to a demonstrator.**

## Task 2 4 x 4 Key matrix input

**Objective:** Write a program to show digit 0 to 9, 'A' to 'F' on the 7-Segment LED display when the corresponding button in the 4x4 key matrix is pressed.

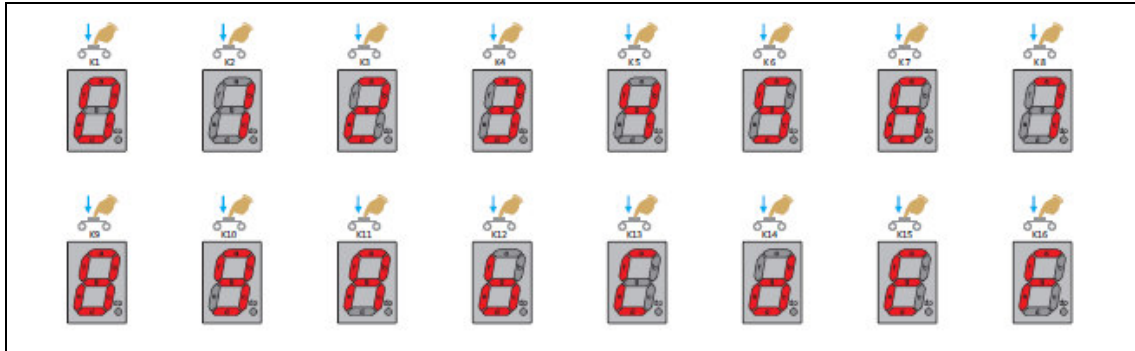


Figure 21a Schematic for the expected result

**Hardware setup:** Connect microcontroller Port D to 1-digit 7-Segment LED, Port C to 8-Button input, i.e., connector CN8 – CN5, CN10 – CN18. In your program, please ensure that you have set all I/O ports to be digital using the following two instructions: `movlw 0x0F;` `movwf ADCON1.`

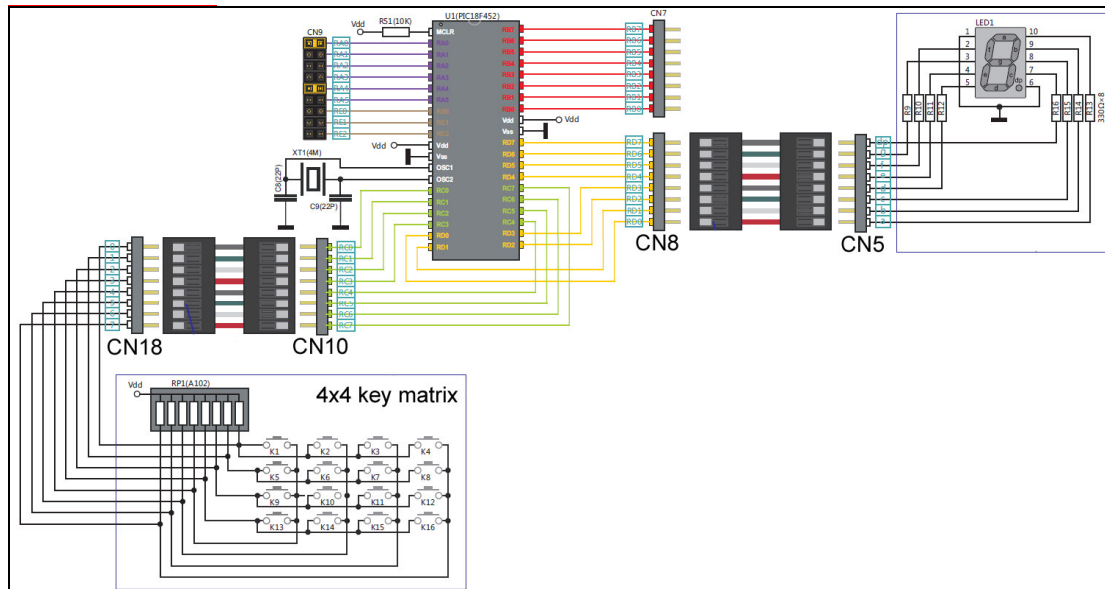


Figure 21b Schematic of circuit connection

### Suggested algorithm:

The KeyScan subroutine should perform the following operations:

1. Initialize variable KEYCODE to 0. KEYCODE is the variable that will be output by the KeyScan subroutine.
2. Set row1 (RC0) to logic '0' and others to logic '1' to scan row1 buttons only.
3. Define and call a subroutine DetectCol to read PortC. If logic '0' is appeared on any bit of RC4-RC7, a key connected to that port is pressed. Return value 1 to 4 to indicate which column key is pressed, return 0 if no key is pressed.
4. Check DetectCol return value. There are two possibilities:
  - If DetectCol returns 0 (i.e., no key is pressed in the current row), add 4 to KEYCODE. Continue to scan the next row by setting the next row to logic '0' and other rows to logic '1'.
  - If DetectCol returns Col, which is between 1 and 4, add Col to KEYCODE and exit KeyScan.
5. If no key press is detected after scanning the 4 rows, reset KEYCODE = 0 and exit KeyScan.

**Question for discussion (Give your answer in your logbook):**

(?) In Task 1, one port pin is required to read the status of a button into the microcontroller. How many I/O pins are required in Task 2 to read the statuses of 16 buttons? What is the advantage of the matrix topology implemented in Task 2?

➤ **Checkpoint 2: Show your work to a demonstrator.**

## **Laboratory 4: Implementation of a frequency counter using PIC18 Timer Module**

### **Objectives:**

- Learn the basic operations of the PIC18 Timer Module both in timer mode and counter mode.
- Use PIC18 to build a frequency counter.
- Learn how to use the oscilloscope.

### **Equipment required:**

- ND118-4520 Development Kit equipped with PIC18F4520 Microcontroller.
- DS Lab Function Module #1.
- Oscilloscope.

### **Prelab work:**

You are given the following code fragment that uses TMR0 to count 1 second. Follow the instructions below to modify the code so that the LED turns on for exactly 1 second.

```

Main:      clrf    TRISD                ;set PortD output
           clrf    PORTD

           movlw   0x03                ;TMR0, 16-bit, Int clk, 1:16 prescale
           movwf   T0CON
           movlw   0xYY                ;Students will determine initial YYXX to put in TMR0H: TMR0L
           movwf   TMR0H
           movlw   0xXX
           movwf   TMR0L

           bcf     INTCON, TMR0IF      ;
           bsf     PORTD, RD0          ;turn LED on

           bcf     INTCON, TMR0IF ; Set Breakpoint 1 here right after LED was turned on
           bsf     T0CON, TMR0ON        ;start TMR0
WaitT0done: btfss   INTCON, TMR0IF
           bra     WaitT0done

           bcf     PORTD, RD0          ;turn LED off.
           bra     $                  ;set Breakpoint 2 here right after LED was turned off
                                           ; jump to current PC, loop forever;

```

**Figure 22 – One second timer example code**

1. Determine YYXX, the initial value to be loaded to the registers TMR0H:TMR0L so that the LED turns on for 1 second. Here, you would consider only the time delay generated between the time point when the timer is turned on to the instant when the interrupt flag is raised. Do not consider overhead time delay generated by instructions.
2. Create a new project named “Timer”. Copy the code template from previous projects as shown in Figure 8. Replace the code segment between the labels “Main” and “END” with the code fragment shown above.
3. Select Debug Tools as MPLAB SIM.
4. Click “Debugger”, “StopWatch” to bring out the stopwatch window. Set processor clock to 4MHz in settings tab.

5. Set Breakpoints 1 and 2 as indicated in Figure 22.
6. When the program stops at Breakpoint 1, set the stopwatch to zero by clicking the “zero” button in the stopwatch window.
7. Run the code again and the program will stop running at Breakpoint 2. Write down the time recorded by the stopwatch. You should observe that the time recorded by the stopwatch slightly exceeds 1 second.
8. Modify the code so that time elapsed between Breakpoints 1 and 2 is exactly 1 second. *(Hints: Set initial values of TMR0H:TMR0L so that the time delay generated by the Timer 0 is slightly less than 1 second. Then, add nop instructions between the two breakpoints).* Bring your code to the lab and have it checked with a demonstrator.

### Task 1 TMR1 functions as a counter

|           |       |               |                                         |
|-----------|-------|---------------|-----------------------------------------|
| Main:     | clrf  | TRISD         | ;set PortD output                       |
|           | clrf  | PORTD         |                                         |
|           | movlw | 0x86          | ;TMR1: 16-bit, RC0, Async, 1:1 prescale |
|           | movwf | T1CON         |                                         |
|           | clrf  | TMR1H         |                                         |
|           | clrf  | TMR1L         |                                         |
|           | bsf   | T1CON, TMR1ON | ;Enable TMR1                            |
| MainLoop: | movff | TMR1L, PORTD  | ;copy TMR1 to PORTD                     |
|           | bra   | MainLoop      |                                         |

**Figure 23 – TMR1 counter example code**

1. Open a new project named “Counter”. Copy the code template from previous project shown in Figure 8. Replace the code segment between the labels “Main” and “END” with the code fragment shown above in Figure 23.
2. Connect CN10 with CN19 and CN8 with CN4. Build the code and execute it under the PICKit 3 debugger environment.
3. (?) Press P1 button several times and see how the LEDs respond to your actions. Briefly describe the change. Make sure the connection between CN10 and CN19 is made in the correct orientation. The LED would not respond to P1 (and respond to P8 instead) if the orientation is reversed.
4. (?) Change the prescale setting to 1:4 (0xA6) in T1CON and run the program again. Press P1 and see how the change in the prescale value affects the counter operation. Describe the change.
5. (?) Does LED change state at the point when you push down the button or at the point when it returns to the original position? Find an explanation of your observation by reading p. 127 of the PIC18F4520 data sheet.

➤ **Checkpoint 1: Show your work to a demonstrator.**

## Task 2 Constructing a frequency counter

### Background: Function Module #1 (Frequency Generator)

A variable frequency generator is used in this part of the laboratory. It is built with a NE555 chip connected in the astable mode. The frequency of the output signal is a function of the charge/discharge time of the capacitor C1, which is controlled by varying the resistor value R2 through turning a knob on the circuit board (Figure 24).

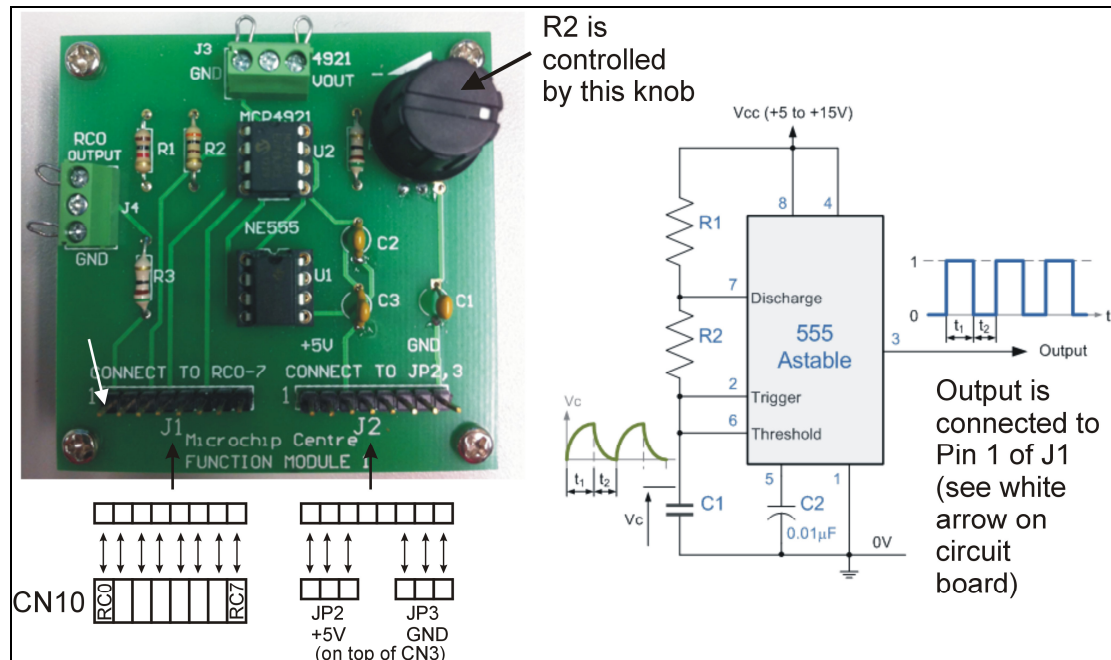


Figure 24 – The left panel shows the circuit board of Function module 1 and the connections required for this task. The right panel shows the circuit diagram of the NE555 chip, which is used as the variable frequency generator. Notice that the output of the frequency generator is connected to PIC18F4520 RC0, which is the counter input of TMR1.

**Hardware settings:** Connect CN6 with CN8 and CN3 with CN7 as in Lab 2 Task 2. Connect P1 (first button in the 8-button keypad) to RE2. Connect J1 and J2 of Function module 1 according to Figure 24. In your program, please ensure that you have set all I/O ports to be digital using the following two instructions: `movlw 0x0F; movwf ADCON1`.

**Specifications of this task:** Implement a frequency counter of a signal generated by a variable frequency generator and display the frequency measurement on the 4-digit 7-segment LED module. When P1 is pressed, the following two tasks will be performed by the microcontroller:

1. Measure frequency of signal generated by the NE555 chip: Write a subroutine MeasureFrequency that serves as the frequency counter. After executing this subroutine, you would expect that the frequency measurement is stored in TMR1. This subroutine can be implemented by combining your prelab work and work done in Task 1. **Note that you should validate your frequency measurement before displaying the reading in the second part.** Please follow the steps provided in the next section to validate your frequency measurements.
2. Display the result on the 4-digit 7-segment LED module: You can reuse code generated in Lab 2 Task 2 to display the frequency measurement on the 4-digit 7-segment LED module. The frequency reading stays until you press the P1 button that triggers the frequency measurement to be taken again.

### Steps to validate your frequency measurement

1. (?) Use the debug mode of PICkit 3 (Click “**Debugger**”, “**Select Tool**”, “**2 PICkit 3**”) to assemble your program. Set a breakpoint at the point when you exit the subroutine MeasureFrequency. Write down the value of the SFR TMR1 as observed from the Watch Window.
2. Connect the oscilloscope channel #1 probe to Function Module #1 JP4, which outputs the variable frequency signal.
3. Push the power button at bottom-left corner once to switch on the oscilloscope.
4. Press the “MENU” button in **VERTICAL** section. Select “DC” for Channel 1 coupling.
5. Press the “AUTOSET” button in **ACQUIRE** section. TDS3032 will setup the display parameters automatically.
6. Press the “MEASURE” button and select Frequency. You should see the real time frequency value of the generated sine wave. Select “Remove Measurement” to remove unnecessary measurement display values.
7. (?) Compare the TMR1 register value and the frequency you obtained in 6 using the oscilloscope. Calculate the percent difference between the two measurements. The difference should be within 0.1% range.

➤ **Checkpoint 2: Show your work to a demonstrator.**

## Laboratory 5: Asynchronous and synchronous serial communications

### Prelab work:

1. Explain overflow and framing errors in UART.
2. Answer the following questions regarding the I<sup>2</sup>C interface with 24C02 EEPROM. We have already gone over these materials during lectures. See the data sheet of 24C02 if you need more information (<http://www.atmel.com/Images/doc0180.pdf>).
  - a. What is a control byte? What does it specify?
  - b. When performing byte write, you have to specify the word address. What is the word address? How many bits do the word address consist of? Why?
  - c. Suppose the word address specified in a page write operation is  $n$  and five bytes are written consecutively, where is the last byte written stored in the EEPROM?
  - d. When performing the byte read operation (see Fig. 33), the control byte is specified twice, why?

## Part A: Asynchronous communication: PIC18 USART module

### Objectives:

- Study the basic operations of the USART.
- Learn to use the PIC18 USART module to communicate with a PC through COM Port.

**Equipment required:**

- ND118-4520 development kit equipped with a PIC18F4520 microcontroller.
- 9-pin serial extension cable.

**Hardware settings:** Connect Port C (CN10) to MAX232 Driver CN20. Connect the RS232 connector (CN24) to the PC com port with a 9-pin serial port extension cable. Connect Port D (CN8) to 8 LEDs (CN4).

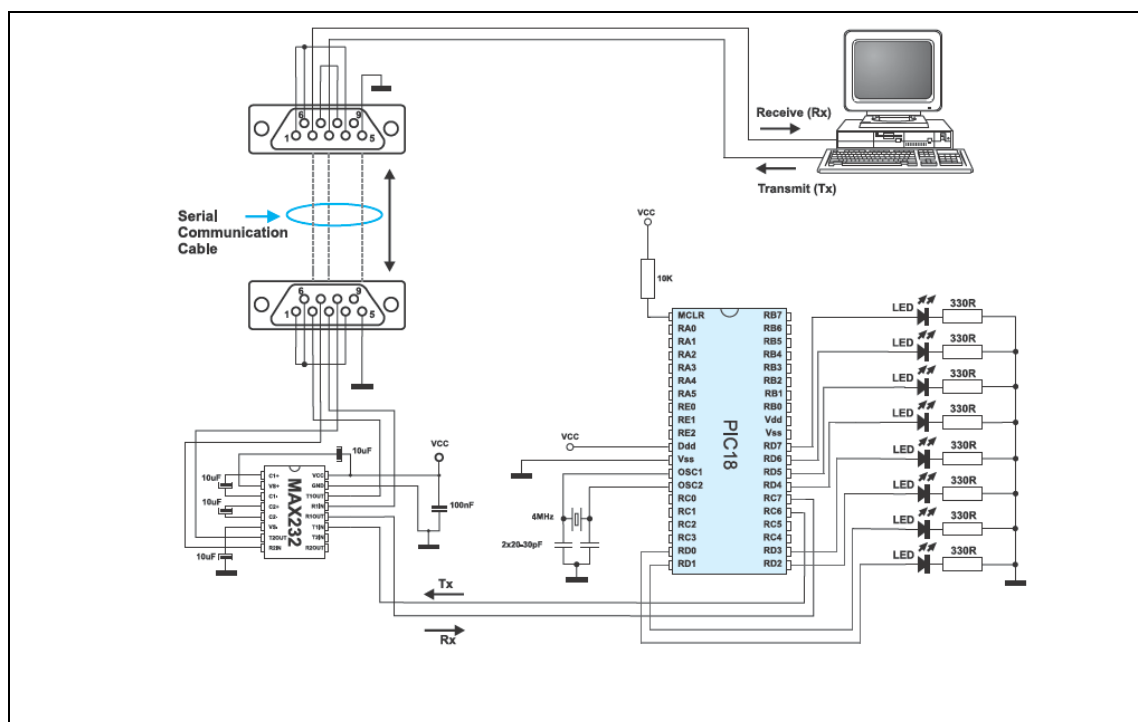


**Figure 25 – Schematic of circuit connections**

### UART Transmission

1. Open a new PIC18F4520 project. Open a new file and type in the code shown in Figure 28 and save file as “**Usart\_tx.asm**”. Add this file into the project.
2. (?) Run the PC application hyper-terminal and set up a connection with a COM port (in most case COM1). Configure the port setting with parameters set to the following values: 9600 baud, 8 data-bit, no parity, 1 stop bit and no flow-control. Describe your observation.

```

Main:      movlw  b'00100100'      ;Tx Enable
           movwf  TXSTA
           movlw  b'10010000'      ;Serial Port Enable
           movwf  RCSTA
           bcf    BAUDCON, BRG16
           movlw  d'25'             ;Set Baud rate for 9600
           movwf  SPBRG

           movlw  'A'
           call   putChar
           movlw  'B'
           call   putChar
           movlw  'C'
           call   putChar
           bra     $                ;loop forever

putChar:    btfss  TXSTA, TRMT
           bra     putChar
           movwf  TXREG
           return
  
```

Figure 26 – Code for USART Transmission

```

Main:      clrf    TRISD            ;set PORTD output
           movlw  b'00100100'
           movwf  TXSTA
           movlw  b'10010000'      ;Serial Port Enable
           movwf  RCSTA
           bcf    BAUDCON, BRG16
           movlw  d'25'
           movwf  SPBRG

MainLoop:  btfss  PIR1, RCIF
           bra     MainLoop
           movff  RCREG, PORTD
           bra     MainLoop
  
```

Figure 27 – Code for USART Reception

### **UART Reception**

3. Open another project and run the code shown in Figure 27.
4. (?) Type something on the hyper-terminal and send it. Describe how the LEDs change relates to the input.
5. (?) Change the baud rate specified in the hyper-terminal to 19200 bauds without changing the code. What do you observe? Explain your observation.
6. (?) Change the baud rate to 57600 bauds both on the microcontroller and the hyper-terminal sides and then run the code again. Try sending different bytes from the hyper-terminal. What did you observe? Explain your observation. (*Hints: How many bits are received correctly? What is the % error of the sampling frequency on the microcontroller side?*)
7. (Optional) (?) Write code to detect *framing error* occurred during reception. Assign an LED to indicate whether a framing error has occurred after receiving a byte. (*Hint: you can consider using one of the LED in the 7-segment LED (CN5)*). Framing error occurs frequently under Setting 6. Give an example of a situation where framing error occurs and show it to a demonstrator.
8. (Optional) (?) Write code to detect *overflow error*. Give an example of a situation where overflow error occurs and show it to a demonstrator.

➤ **Checkpoint 1: Show your work to a demonstrator.**

## **Part B: Synchronous serial communication: PIC18 I<sup>2</sup>C bus**

### **Objectives:**

- Learn the I<sup>2</sup>C serial communication protocol through programming the microcontroller to interface with an external EEPROM device.
- Use the logic analyzer in hardware and software debugging.

### **Equipment required**

- Saleae Logic16 logic analyzer.
- ND118-4520 Development Kit equipped with PIC18F4520 Microcontroller.

### **Introduction to the logic analyzer**

A logic analyzer is an instrument used for hardware debugging, software debugging or both in embedded systems. A logic analyzer consists of a block of memory, trigger unit, clock unit, data acquisition unit and display unit (some logic analyzer models may be connected to a PC for control and display purposes). The operation of a logic analyzer is synchronized with a clock signal. Upon receiving a trigger signal, the datum is sampled in each clock pulse and stored in the memory.

#### *Sampling rates:*

It is vital to have the correct time-base setting in order to accurately capture data. The logic analyzer samples at time intervals based on the settings of the logic analyzer's time-base or by reference to an external clock. The sampled data is then recreated and displayed on the logic analyzer's display screen.

A general rule of thumb is that the sampling rate should be a minimum of twice the frequency of the fastest signal of the system. If the logic analyzer is not sampling fast enough, aliasing occurs. For example, the sampling rate of at least 4 MHz should be set for a system running at 2 MHz. However, a higher sampling rate is sometimes preferable, e.g., 5 samples of data will be obtained for a sampling rate of 10 MHz. The faster the sampling the better the resolution, but increasing the sampling rate decreases the capturing time. Capturing time can be calculated as sample memory depth multiplied by the sampling period.

#### *Post-triggering and Pre-triggering:*

The size of the post-triggering buffer can be selected to control the number of samples recorded and stored after the trigger (e.g., rising/falling edge). Events after the trigger are usually the focus of the investigation. The trigger is usually positioned near the beginning of the logic analyser sample buffer, leaving enough room to store events that occur after the trigger.

The size of the pre-triggering buffer determines the number of samples recorded and stored before the trigger. Events before the trigger may be of interest, especially when you want to know what caused the trigger in the first place.

### Task 1 Basic I<sup>2</sup>C interface with 24C02 2Kbit EEPROM

You are provided with code that performs the following two functions:

- When the P7 button is pressed, 4 bytes are written to the EEPROM: 0x04, 0x05, 0x02 and 0x00 are respectively written to device locations 0, 1, 2, 3.
- When the P8 button is pressed, 4 bytes are read from the EEPROM: Device locations 0, 1, 2 and 3 are read and respectively stored in the variables DIGIT0, DIGIT1, DIGIT2 and DIGIT3.
- When the P6 button is pressed the variables DIGIT0, DIGIT1, DIGIT2 and DIGIT3 are reset to 0.

You are required to use the logic analyzer to monitor the sequence of events and describe your observations.

#### Hardware connections:

1. Connect a 9-wire harness to Port A of the logic analyzer, with the black wire connected to label 1 of the port and the grey wire to the ground label. Each port has 8 channels. It is conventional to connect the black wire of the harness to Channel 0 (which is labelled 1).
  2. Connect test-clips to the channel wires. In this exercise, we monitor 3 signals, i.e., a key button and the 2-wire I<sup>2</sup>C bus. Thus, 3 channels + 1 ground test-clips are used. Connect them to wire black, brown, red and grey.
  3. Hook test-clips to according to the following list:
    - Channel0 (black) to CN19-6,
    - Channel1 (brown) to CN11-4,
    - Channel2 (red) to CN11-3 and
    - Ground (Grey) to JP3. JP3 is located at above of 4-digit 7-Segment LEDs.
- *Pay extra attention while hooking test-clips onto target board, they are soft and can be damaged easily.*
4. Connect CN8 to CN19, CN10 to CN11 (see Figure 28). Test-clips are in between the CN pins and cable plugs.

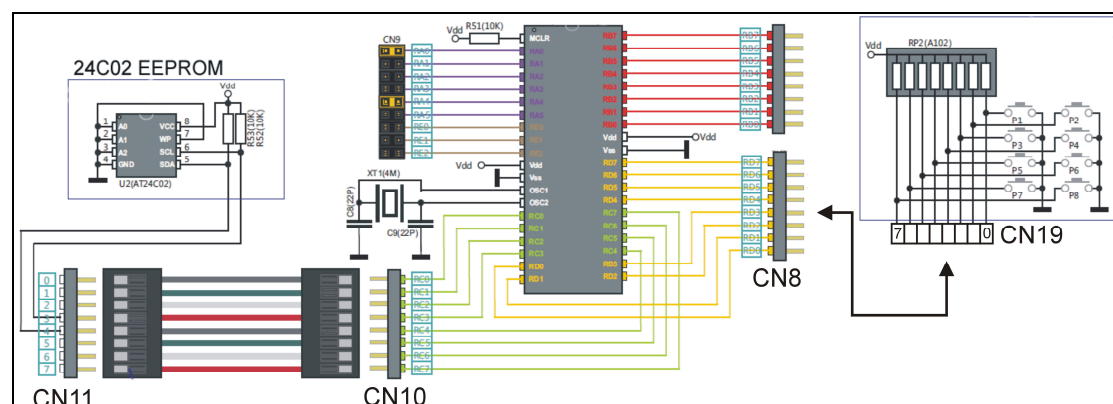


Figure 28 – Schematic of circuit connection

### Instructions:

- Run the Saleae Logic application. In the application window, specify the following parameters:
  - At the top left-hand corner of the window, select “2M Samples @ 2MHz”, setting the sampling rate to 2MHz and post-trigger buffer to 2M samples. This configuration allows one to store one second of sampling data after the trigger.
  - Locate the “Options” drop-down box at the top right-hand corner of the window. Click “Options”, “Preferences”, “Capture”. Select “1M samples” for pre-trigger buffer size. This configuration allows one to store 0.5 second of sampling data before the trigger (because 2MHz was set as the sampling frequency).
  - Click “Options”, “Logic16 channels”, “Use Channels 0-2”, setting up 3 channels to be monitored.
  - Click “Options”, “Logic16 input voltages” and make sure “3.6V to 5.0V” is selected, which corresponds to the logic ‘1’ voltage region for 5V TTL devices.
  - Click “Options”, “Display in Hexadecimal” and make sure “Hexadecimal” is checked.
  - Click setup icon of the Measurements window (see Figure 29) and make sure the followings are checked. This setting allows you to display the measured time period between pulses.

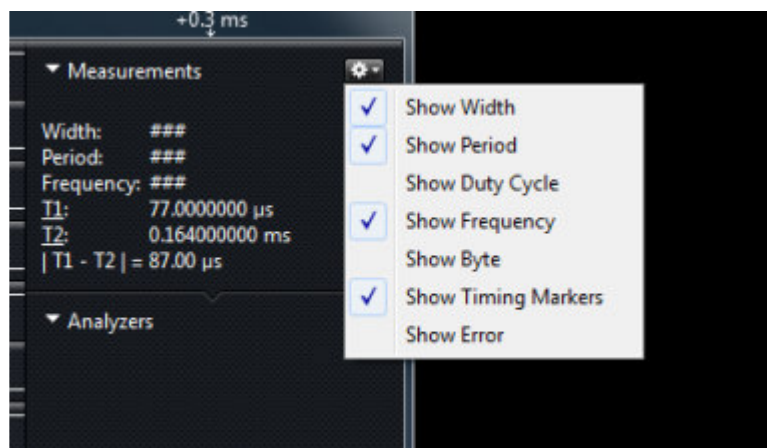


Figure 29 – Measurement Setup

- Modify the channel label as follows:
  - 0 – Channel 0 → 0 – Key
  - 1 – Channel 1 → 1 – SDA
  - 2 – Channel 2 → 2 – SCL
- Click the Plus icon next to the Analyzers tab. Select I<sup>2</sup>C protocol to be analyzed. An I<sup>2</sup>C section will be added under the Analyzers window. Click the setup icon and click “Edit Settings” and assign the appropriate channels for I<sup>2</sup>C SDA and SCL signals as shown in Figure 30.

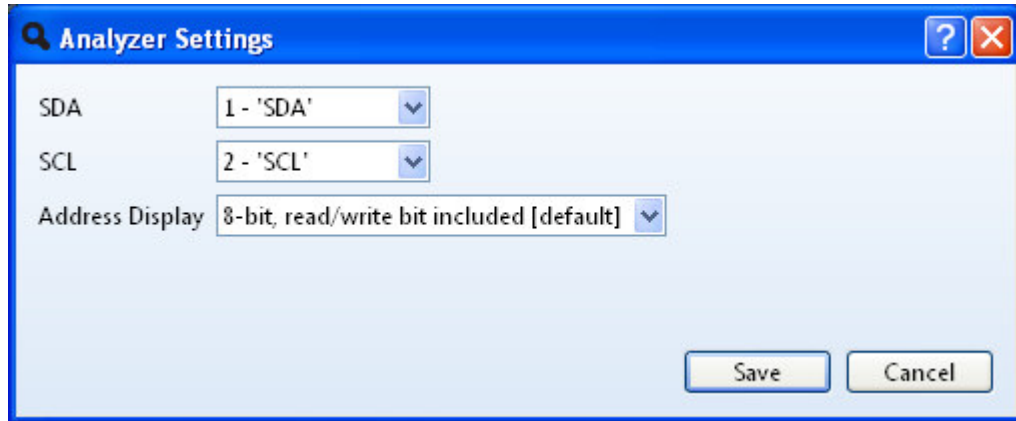


**Figure 30 – I<sup>2</sup>C Protocol Analyzer Setup**

- Set the trigger edge. Since the key button signal goes LOW when pressed, we choose Channel 0 falling edge as the trigger. Click the highlighted button shown below:

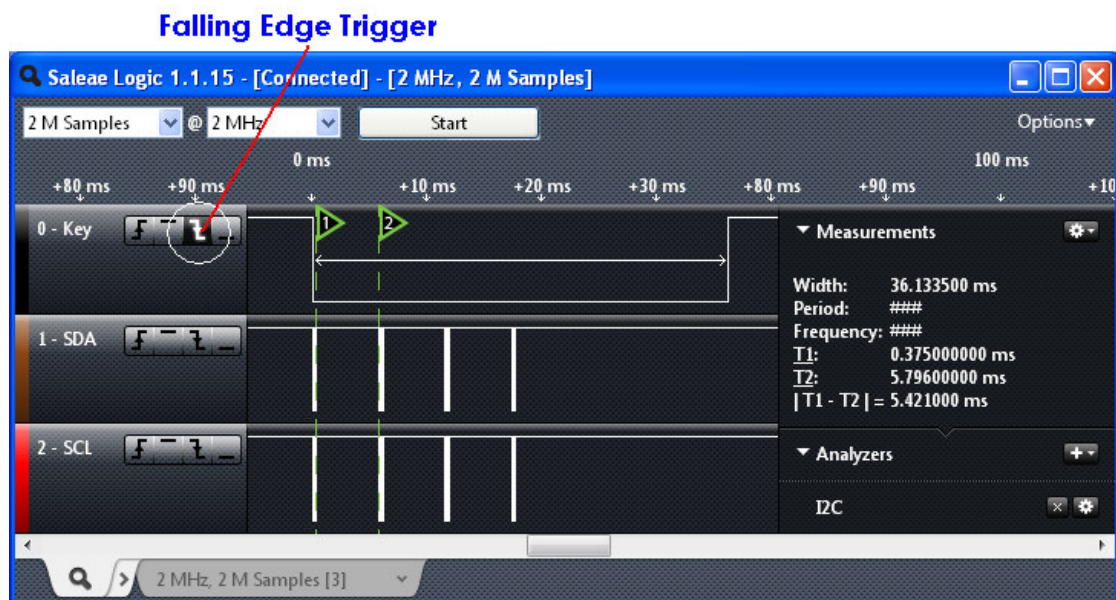


**Figure 31 – Trigger point setup**

- Click on the i2c.mcw MPLAB workspace file to start MPLAB. Rebuild the code and download it into the PCKIT3 debugger and run the program.
- Click the start button within the Saleae Logic Window. The logic analyzer is now waiting for the trigger, which you defined as the falling edge of Channel 0 in Step 4.
- Press the P7 button. A screen similar to Figure 32 will appear. Move the mouse cursor to Saleae Logic window channel region and scroll the mouse wheel in order to zoom in and zoom out of the signals. 4 pulse sequences should appear. By moving your mouse cursor over the window, you would select two closest signal transitions to the mouse cursor and the duration between these two signal transitions is displayed as Width under the Measurements window. For example, you can observe that the P7 key was pressed for a 36.13 ms duration from Figure 31.
- You can also place timing markers T1 and T2 by following the steps listed below:

- Single-click the **T1** or **T2** label. (or use **1** and **2** keys on your keyboard)
  - Move your mouse to the desired location. Note that you can still pan and zoom.
  - Single click to place the marker. Your mouse must not be moving when you place the marker.
  - To move the marker after it has been placed, click it.
  - To cancel placing a marker, right-click or press the escape key.
9. Scroll the mouse wheel to zoom into and capture the first pulse sequence as shown in Figure 32:

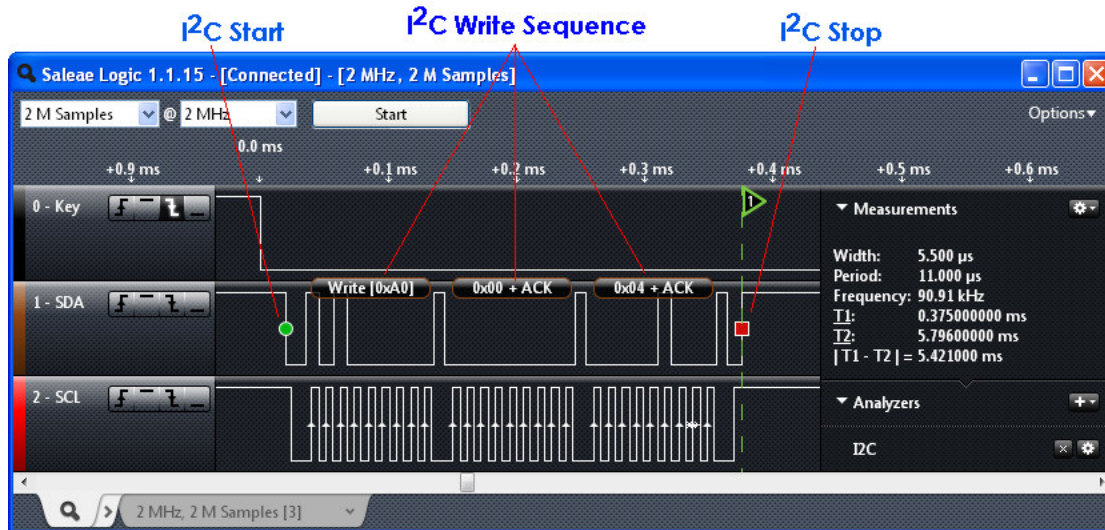


Figure 32 – I<sup>2</sup>C Write Sequence Timing

10. (?) Compare the captured signals to the 24C02's datasheet. Identify and write down the control byte, word address and the data in each byte write cycle. Understand how the byte write operation is programmed by studying the source code.

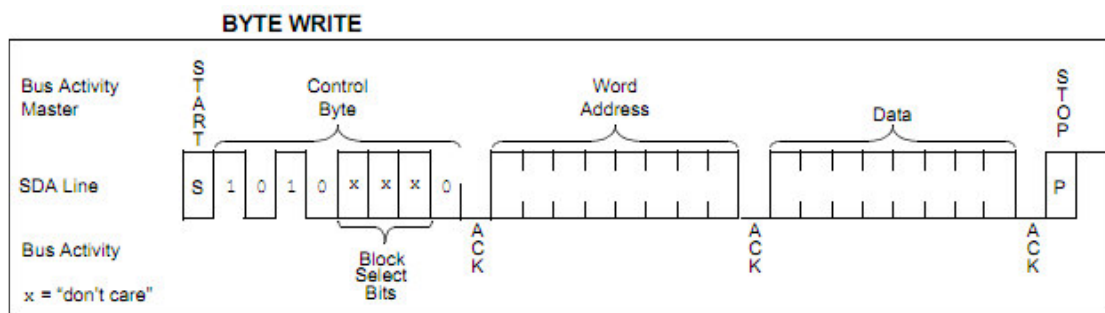


Figure 33 – Timing diagram of byte write operation

11. (?) Change the delay constant from decimal 7 to decimal 1 (see Fig. 34) and perform the byte write operation again. By observing the display on the logic analyzer, you can see that separation between the four pulse sequences becomes narrower. Notice that a NAK is received in some byte write operations. Explain why this happens by referring to the course notes. Also refer to the 24C02 datasheet to find out the minimum timing separation required between two byte write operations.

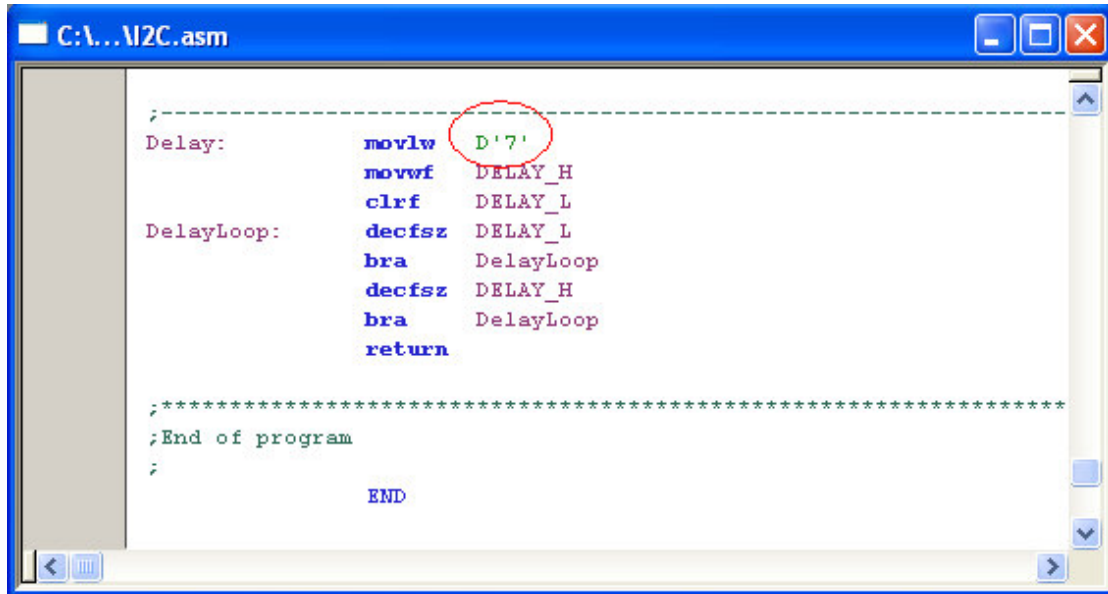


Figure 34 – Modify the delay constant circled in red

12. (?) Connect Channel 0 to the CN19-7 and press the P8 button. Provide a screen capture of the logic analyzer display and identify the control byte, word address and the byte the microcontroller receives. Compare the screen capture with Figure 35. Also monitor the variables DIGIT0-DIGIT3 from the watch window in MPLAB. List the contents stored in these variables. Click the P6 button and observe the changes in the contents stored in these variables.

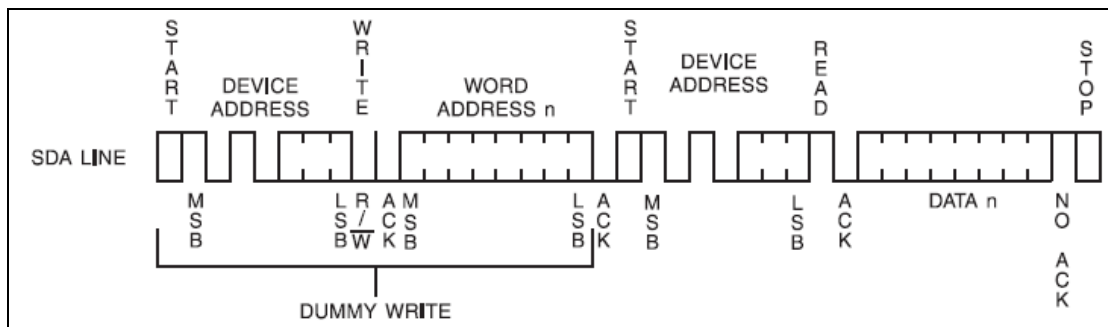


Figure 35 – Timing diagram of byte read operation

13. (?) Explain why the lines following the WaitP7Up and WaitP8Up labels are required in the program source code. What happens if they were not there?

➤ **Checkpoint 2: Show your work to a demonstrator.**

### Task 2: Page read and write

- Write a subroutine called `EEPROM_Page_Write` that writes your group ID in 4 digits into the EEPROM address 0x00 to 0x03 using the page write method. The timing diagram is shown in Figure 36. Change the MainLoop so that pressing P7 will call `EEPROM_Page_Write` instead of `EEPROM_Write`. Provide a screen capture from the logic analyzer.

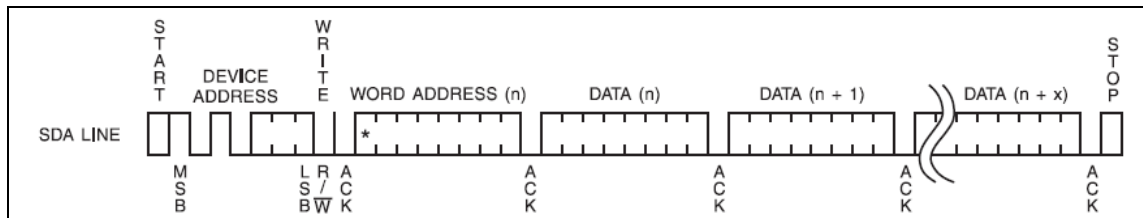


Figure 36 – EEPROM Page Write timing

#### ➤ Checkpoint 3: Show your work to a demonstrator.

- (Optional) Write a subroutine called `EEPROM_Page_Read` that reads the contents stored in memory location 0x00 to 0x03 of the EEPROM by using the page read method. The timing diagram is shown in Figure 36. Change the MainLoop so that pressing P8 will call `EEPROM_Page_Read` instead of `EEPROM_Read`. Verify the digits of your group ID are successfully retrieved. Provide a screen capture from the logic analyzer.

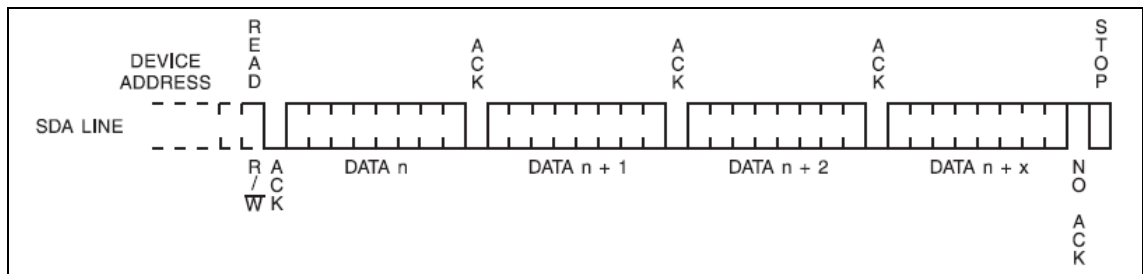


Figure 37 – EEPROM Page Read timing

#### ➤ Checkpoint 4: Show your work to a demonstrator.