PICF458 BASED VOLTMETER

EMBEDDED SYSTEMS

MODULE CONVENOR: DR. SHERRIF WELSEN

VIGNESHWAREN SUNDER
20021065
BE HONS IN MECHATRONICS

Table of Contents

1. INTRODUCTION: 1
1.2 AIM: 2
1.3 OBJECTIVE: 2
2. PIC18f 2
2.1 PIC18F FEATURES 2
2.2 INSTRUCTION CYCLE: 3
2.3 PIN CONFIGURATION: 3
2.4 PIC18F INTERRUPTS: 4
2.5 On Chip Analog to Digital Converter Module: 5
2.5.1 A/D Convertion Time: 5
2.5.2 STEP SIZE: 5
2.5.3 DIGITAL DATA OUTPUT 6
2.6 Working Principle of 7-Segment Led Display 7
3. DESIGN AND WORKFLOW OF THE ALGORITHM: 7
4. FLOWCHART 8
5. SIMULATION RESULTS: 9
6. CONCLUSION: 10
7. REFERENCES: 11
8. APPENDIX: 11
9. ASSEMBLY CODES: 11

1. INTRODUCTION:

Before the introduction of embedded systems, people were using analog voltmeter to measure the voltage. This analog voltmeter works on the simple principle of Ohm's law. However, this system is big and hectic to calibrate to get the right output. Now the use of analog voltmeter is no more in practice after the invention of embedded systems. These systems can be programmed to perform various task on a microchip. Microcontrollers are easier to program and integrate several inputs and outputs onto a single chip and consumes less space. It is being widely preferred for many engineering applications due to its flexibility and size. In this project we have demonstrated on how to design an efficient voltmeter using PIC18F458 microcontroller.

1.2 AIM:

The aim of this report is to describe how to design and code a simple microcontroller based digital voltmeter using a PIC18F458 microcontroller. The range of the voltmeter is chosen to be 0-5V based on the given specification. The PIC mc reads the input voltage through one of the 8 analog channels and converts it to a 10-bit digital number using the internal ADC. The voltage is then displayed in two 7-segment LED displays with one real and one fractional value.

1.3 OBJECTIVE:

The primary objective of the report are as follows:

- To design a circuit for Pic18f458 based voltmeter
- To program the microcontroller using Interrupt Service Routine to convert A/D and show the output in two 7-seg LED Display
- To demonstrate a fully working circuit with real time analog input and to get the output on the LED

2. PIC18f

2.1 PIC18F FEATURES

Parametrics

Name	Value
Program Memory Type	Flash
Program Memory Size (KB)	32
CPU Speed (MIPS/DMIPS)	10
SRAM (KB)	1,536
Data EEPROM/HEF (bytes)	256
Digital Communication Peripherals	1-UART, 1-SPI, 1-I2C1-MSSP(SPI/I2C)
Capture/Compare/PWM Peripherals	1 CCP, 1 ECCP,
Timers	1 x 8-bit, 3 x 16-bit
ADC Input	8 ch, 10-bit
Number of Comparators	2
Number of CAN Modules	1 CAN
Temperature Range (°C)	-40 to 125
Operating Voltage Range (V)	2 to 5.5
Pin Count	40

Additional Features

- 5 x 10-bit PWM
- 40 MHz Max. Speed
- Full CAN 2.0B Active 3Tx Buffers
- 2RX Buffers
- 6 Full Acceptance Filters
- 2 Filter Masks
- PSP
- ICD
- Self-Programming

2.2 INSTRUCTION CYCLE:

The time taken to execute an instruction on PIC is known as instruction cycle. In pic microcontrollers, one instruction cycle takes place in 4 oscillatory periods. In our case,

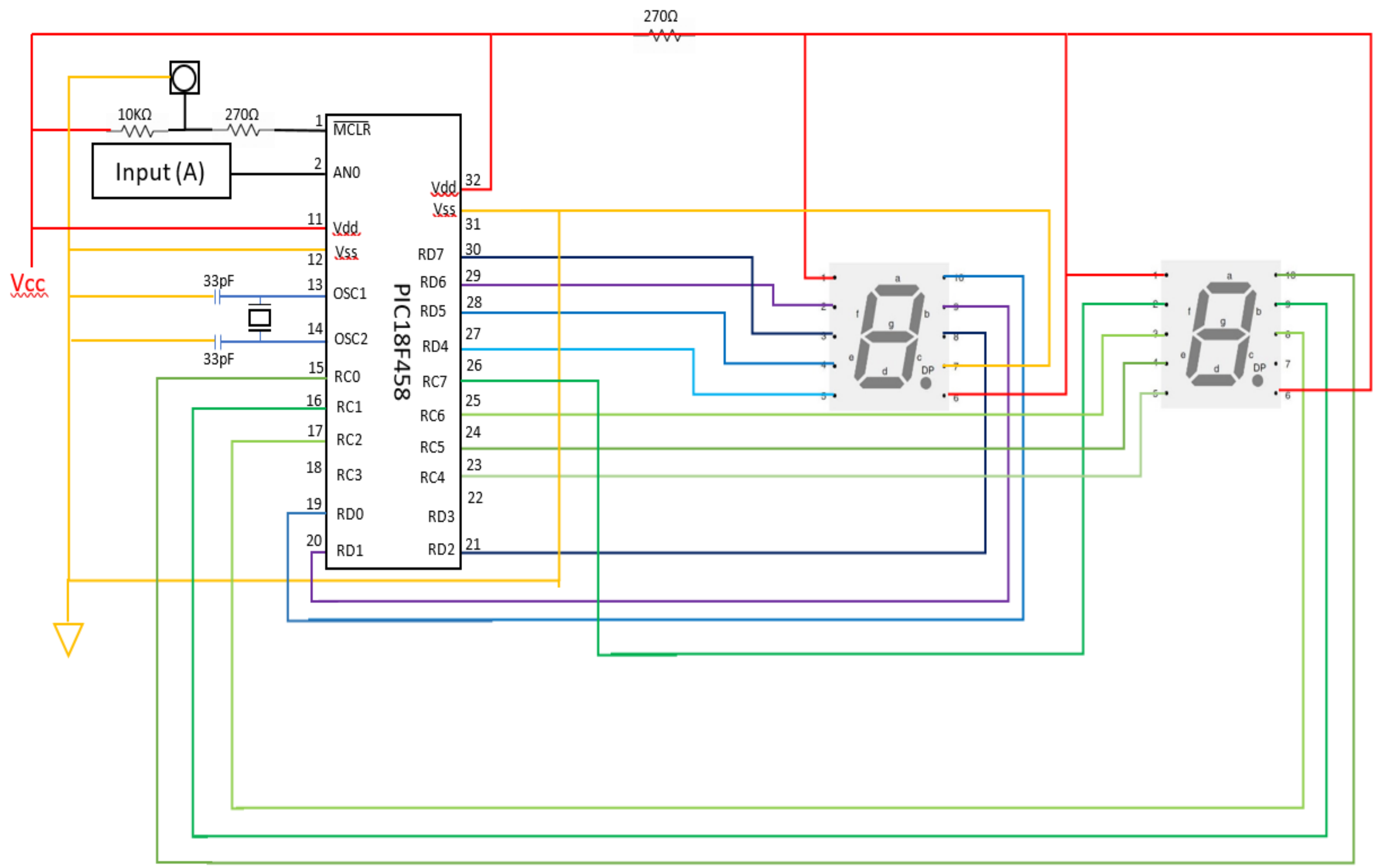
XTAL Frequency = 10 MHz

Timer's Clock Frequency = Fosc(XTAL f)/4 = 10Mhz=4 = 2.5MHz

Time period to execute per instruction = 1/2.5Mhz = 0.4 micro seconds

2.3 PIN CONFIGURATION:

Pic18 has 40 pins with 33 pins set for port A,B,C and D. The remaining pins are set for Vdd, GND (Vss), OSC1, OSC2, and MCLR (master clear reset).



In our circuit, we made use of PORTC and PORTD to give our output; Vdd and Vss for voltage source, and OSC1 and OSC2 for external oscillator, and MCLR to reset. We have used pin2 (AN0) as analog input port. At the register level, we made use of various Special Function registers as follows: INTCON, ADCON0, ADCON1, ADRESL, ADRESH and status register.

During the construction of the circuit we need to make sure that the external oscillator is placed close with the PIC18 microcontroller. It is used to provide stable output for a long time.

Since the oscillator works at very high speed, if kept far it might affect the signal transmission going to the microcontroller and might receive unexpected delays in the clock signal.

2.4 PIC18F INTERRUPTS:

A microcontroller can serve to more than one device. Interrupts are used when we want to mention which device has to be enabled and executed. There are two types of interrupts in PIC18F, External Interrupts and Internal Interrupts. External interrupts are used by external devices connected tot the microcontroller. It can be used as a power failure interrupt.

The interrupt service routine (ISR) can be written to store critical data in nonvolatile memory and the interrupt program can continue without any loss of data when the power returns. When the interrupts are executed, the CPU executes current instruction, saves the next instruction address to the program counter onto the hardware stack automatically and loads the program counter with an address called ISR.

We used internal interrupts to activate interrupt conditions after completing the conversion of ADC. We made use of Interrupt service Routine to load the interrupts.

A/D Converter Interrupt Flag Bits and their Registers

Interrupt	Flag bit	Register	Enable bit	Register
ADIF (ADC)	ADIF	PIR1	ADIE	PIE1

8-BIT of PIR1, PIE1 and INTCON Register

PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE

BCF PIR1, ADIF ;clear A/D interrupt flag for the first round

BSF PIE1, ADIE ;enable A/D Interrupt

INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF

INTCON,PEIE ;enable peripheral interrupt

BSF INTCON,GIE ;enable interrupt globally . Thus interrupted

2.5 On Chip Analog to Digital Converter Module:

The PIC18F contains an on-chip A/D converter module with 13 channels(AN0-AN12). We used channel AN0 to read our analog input. This channel converts the analog input to a 10-bit digital number and it can be stored in two SFRs, ADRESH and ADREL. The A/D module has two 8-bit ADCON control registers for PICF458.

The ADCON0 control register controls the operation of the A/D module. We used 01H to configure the input port to be AN0.

The ADCON1 control register controls the function of the port pins and to select the Vref. We used it to make the result of 8-bit output from analog channel to sit in ADRESH in right justified format. We loaded 04EH into this register to perform the desired operation. More information can be found on code comments.

GO/DONE enables us to initiate the ADC conversion by making the bit to 1. And, 0 means no ADC initiated. It is programmed to be cleared at the End of Conversion (EOC)

ADRESH	A/D Result Register High Byte							
ADRESL	A/D Result Register Low Byte							
ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0

```
MOVLW 0X81          ;Fosc/64, ch. 0,A/D
MOVWF ADCON0        ;load ADCON0 with 10 000 001
MOVLW 04EH          ;right justified, Fosc/64, AN0=analog
MOVWF ADCON1        ;load ADCON1 with 01 001 110
```

2.5.1 A/D Convertion Time:

It is the time taken to convert the analog input into digital output in binary form. It is based on the clock source connected to the PIC18 and also its fabrication technology of ADC chips, like MOS or TTL.

We used clock cycle of Fosc/64 =156250 Hz . Hence Tad = 1/156250 = 6.4 micro seconds

2.5.2 STEP SIZE:

Our Vref = 5V

Step Size = 5/1024 = 4.8mV for 10bit

Step Size = 5/256 = 19.53mV for 8bit

The resolution of our ADC module can be changed by changing the Vref.

2.5.3 DIGITAL DATA OUTPUT

In the 10bit ADC the data output is D0-D9. However, we used 8-bit ADC for our voltmeter. To calculate the output voltage, we use the following formula:

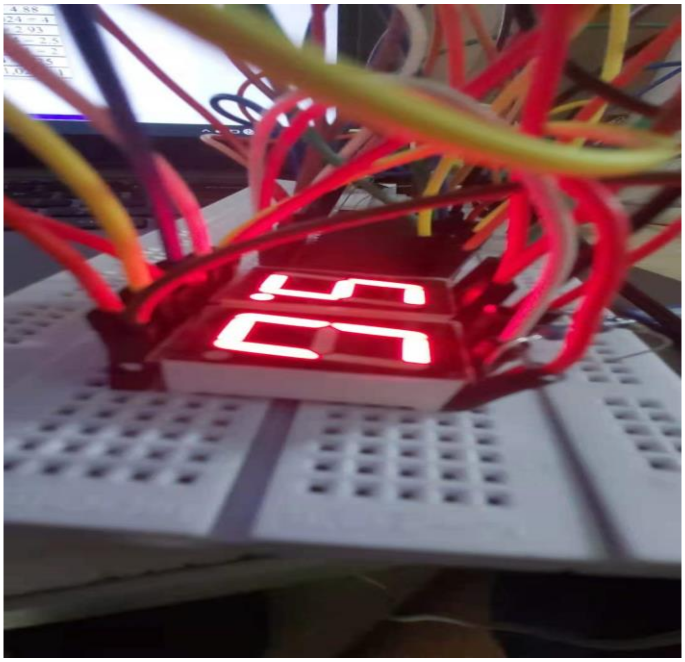
$$Dout = Vin\ /stepsize$$

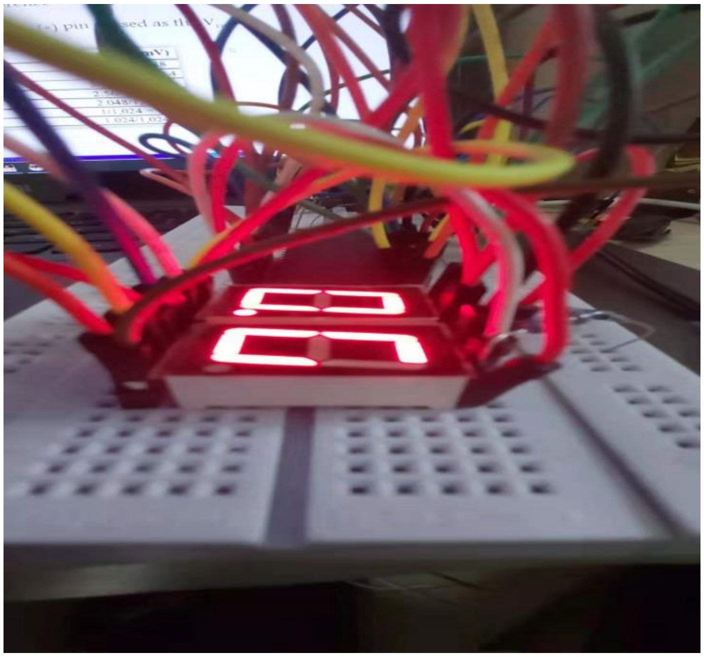
Where, Dout= digital data output(in decimal)

Eg: Dout= 2.1V/4.88mV = 430 in decimal, which gives us 11010111 00 in binary for D0-D9(10-bit).

Dout= 2.1V/19.53mV = 111 in decimal, which gives us 101011110 in binary for D0-D7(8-bit)

And, we used parallel ADC to bring this data from ADC to the PORTS.





Note: The upper left bit of the LED 2 isn't working because the pin for that bit on PIC18 has been broken while handling.

2.6 Working Principle of 7-Segment Led Display

Segments Inputs							7 Segment Display Output
a	b	c	d	e	f	g	
1	1	1	1	1	1	0	0
0	1	1	0	0	0	0	1
1	1	0	1	1	0	1	2
1	1	1	1	0	0	1	3
0	1	1	0	0	1	1	4
1	0	1	1	0	1	1	5
1	0	1	1	1	1	1	6
1	1	1	0	0	0	0	7
1	1	1	1	1	1	1	8
1	1	1	1	0	0	1	9

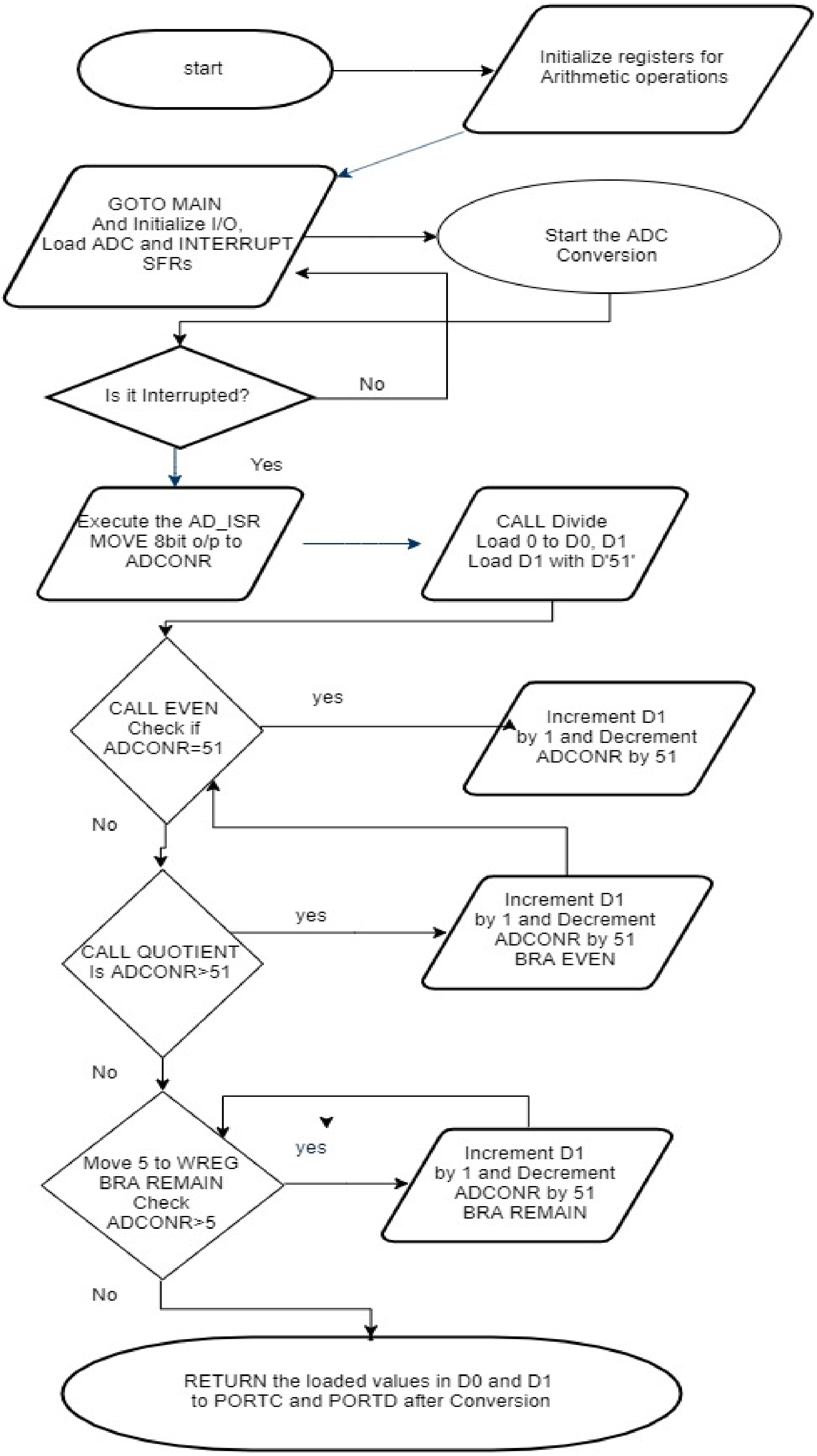
The output has to be converted in a way it gives the appropriate decimal output to the LEDs before sending the received data into PORTC and PORTD.

3. DESIGN AND WORKFLOW OF THE ALGORITHM:

Steps on how the program algorithm works:

- The O/p of A/D gets stored in the ADCONR. This ADCONR is subtracted by 51, and checks whether ADCONR>51. If yes, then contents of register D1 is incremented by 1.
- Now the ADCONR is subtracted by 5. Once the ADCONR<51, the REMAIN(REMAINDER) part of the ADCONR is determined. Each time the subtraction result is greater than 5, register D0 is incremented by 1.
- Therefore, the UNIT part is stored in D1 and Ten part is stored in D0.
- And the output of these registers are then converted to 7-segment format used in the table given above, and sent to PORTC and PORTD.

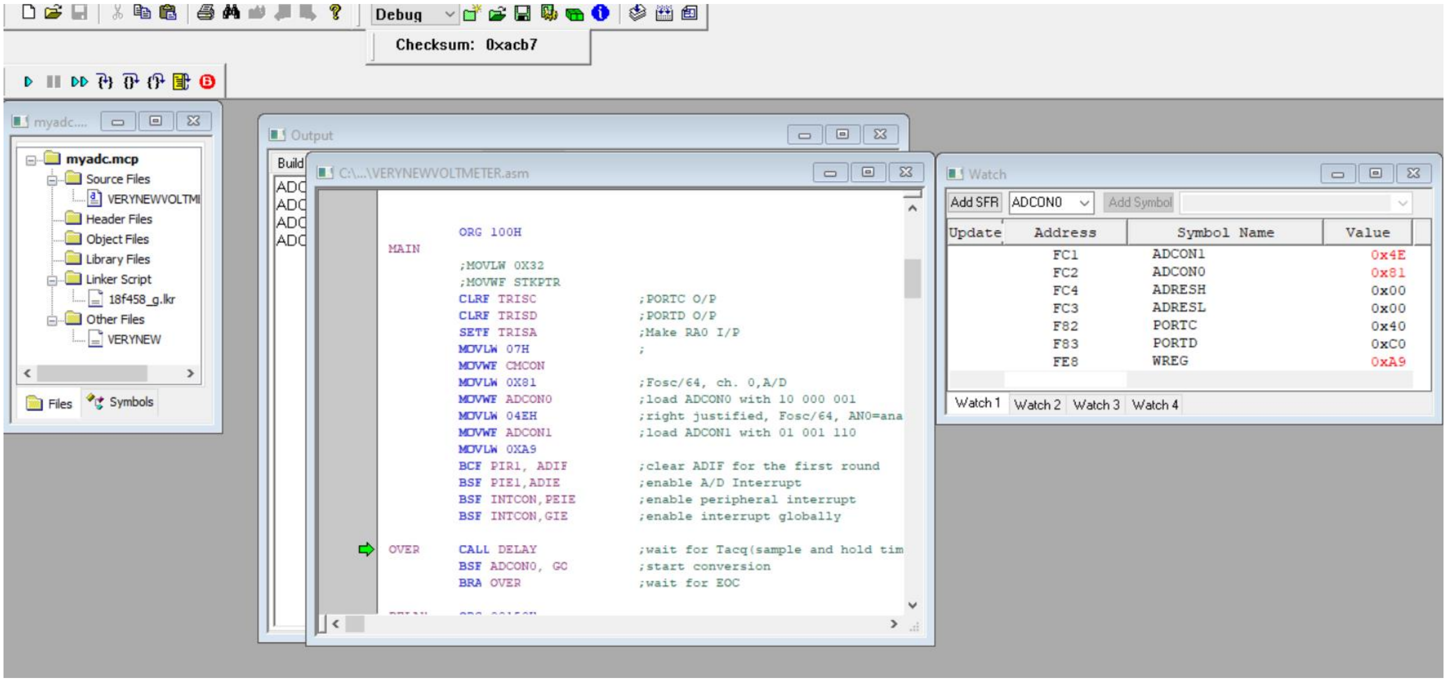
4. FLOWCHART

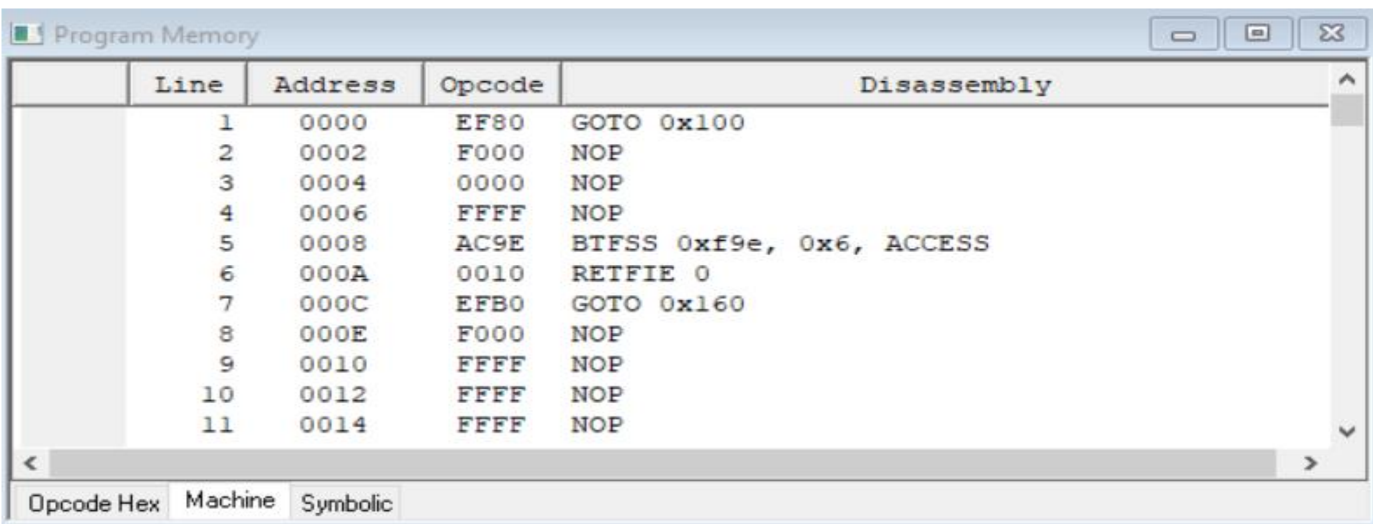


start
Initialize registers for
Arithmetic operations
GOTO MAIN
And Initialize I/O,
Load ADC and INTERRUPT
SFRs
Start the ADC
Conversion
Is it Interrupted?
No
Yes
Execute the AD_ISR
MOVE 8bit o/p to
ADCONR
CALL Divide
Load 0 to D0, D1
Load D1 with D'51'
CALL EVEN
Check if
ADCONR=51
yes
Increment D1
by 1 and Decrement
ADCONR by 51
No
CALL QUOTIENT
Is ADCONR>51
yes
Increment D1
by 1 and Decrement
ADCONR by 51
BRA EVEN
No
Move 5 to WREG
BRA REMAIN
Check
ADCONR>5
yes
Increment D1
by 1 and Decrement
ADCONR by 51
BRA REMAIN
No
RETURN the loaded values in D0 and D1
to PORTC and PORTD after Conversion

5. SIMULATION RESULTS:

The values stored in the SFRs are shown in this simulation




Program Memory

Line	Address	Opcode	Disassembly
1	0000	EF80	GOTO 0x100
2	0002	F000	NOP
3	0004	0000	NOP
4	0006	FFFF	NOP
5	0008	AC9E	BTFSS 0xf9e, 0x6, ACCESS
6	000A	0010	RETFIE 0
7	000C	EFB0	GOTO 0x160
8	000E	F000	NOP
9	0010	FFFF	NOP
10	0012	FFFF	NOP
11	0014	FFFF	NOP

Opcode Hex | Machine | Symbolic

Program Memory

Line	Address	Opcode	Disassembly
512	03FE	FFFF	NOP
513	0400	0E00	MOVLW 0
514	0402	6211	CPFSEQ 0x11, ACCESS
515	0404	D003	BRA 0x40c
516	0406	0E40	MOVLW 0x40
517	0408	6E82	MOVWF 0xf82, ACCESS
518	040A	0012	RETURN 0
519	040C	0E01	MOVLW 0x1
520	040E	6211	CPFSEQ 0x11, ACCESS
521	0410	D003	BRA 0x418
522	0412	0E79	MOVLW 0x79
523	0414	6E82	MOVWF 0xf82, ACCESS
524	0416	0012	RETURN 0
525	0418	0E02	MOVLW 0x2
526	041A	6211	CPFSEQ 0x11, ACCESS
527	041C	D003	BRA 0x424
528	041E	0E24	MOVLW 0x24
529	0420	6E82	MOVWF 0xf82, ACCESS
530	0422	0012	RETURN 0
531	0424	0E03	MOVLW 0x3
532	0426	6211	CPFSEQ 0x11, ACCESS
533	0428	D003	BRA 0x430
534	042A	0E30	MOVLW 0x30
535	042C	6E82	MOVWF 0xf82, ACCESS
536	042E	0012	RETURN 0
537	0430	0E04	MOVLW 0x4
538	0432	6211	CPFSEQ 0x11, ACCESS
539	0434	D003	BRA 0x43c
540	0436	0E19	MOVLW 0x19
541	0438	6E82	MOVWF 0xf82, ACCESS
542	043A	0012	RETURN 0
543	043C	0E05	MOVLW 0x5
544	043E	0E12	MOVLW 0x12
545	0440	6E82	MOVWF 0xf82, ACCESS
546	0442	0012	RETURN 0
547	0444	FFFF	NOP

Opcode Hex | Machine | Symbolic

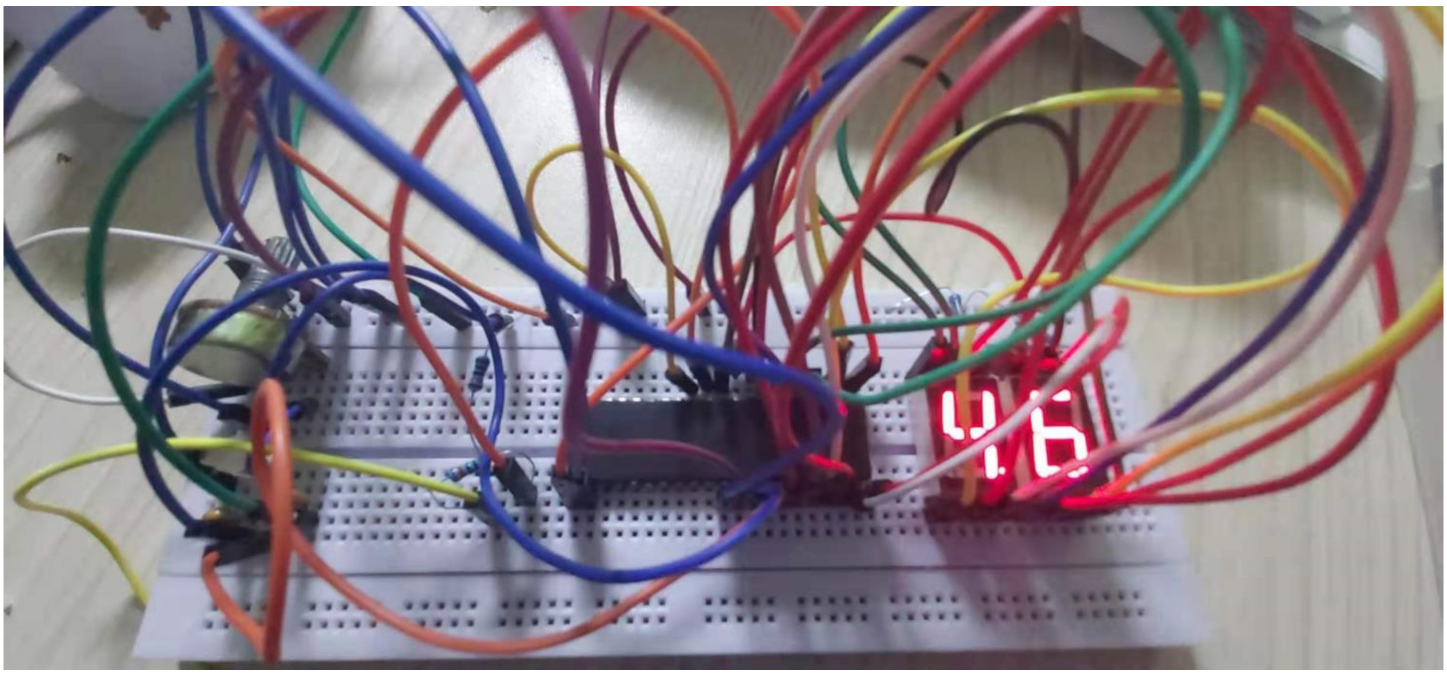
6. CONCLUSION:

Thus, we have demonstrated successfully the working of PIC18F458 based Voltmeter. Our design can be further developed to have higher range by calibrating the Vref for ADC module. This design can also be constructed using BCD-7seg LED Decoder, however, considering the need for space and circuit complexity, we have programmed our assembly code to convert the received BCD to 7-Segment code. The ADC module gives output in 8-bit which gives the output with less complex algorithm. If we use 10-bit, then we need to decode the data from the address ADRESH and ADRESL separately. We can make use of only one PORT (eg: PORTC) to send the output decimal value to many 7-seg LED displays by using a multiplexer circuit and transferring the decoded value to each LED after some minute delay. This may be considered if we need to make use of PORTD and PORTB for something else.

7. REFERENCES:

1. William Brumby, Gaston Mulisanga, Travis Ram, and Vladimir Tsarkov. The Smart Digital Voltmeter. University of Central Florida, Orlando, Florida, 32816-2450
2. Shagun Malhotra , Abhishek Verma , Twishish Shrimali. Design of Digital Voltmeter for AC Voltage Measurement Using PIC Microcontroller. International Journal of Science and Research (IJSR) ISSN (Online): 2319-7064.
3. Rafiquzzaman M. PIC18F Timers and Analog Interface. First published: 04 December 2017 https://doi.org/10.1002/9781119448457.ch10

8. APPENDIX:



9. ASSEMBLY CODES:

;AUTHOR: VIGNESHWAREN SUNDER

;PIC18F458 BASED VOLTMETER (0-5V)

;CODE MADE AS A PART OF MY COURSEWORK FOR EMBEDDED SYSTEMS

;ASSEMBLY CODE STARTS

```
LIST P=PIC18F458

#include P18F458.INC

CONFIG OSC =HS, OSCS=OFF

CONFIG WDT=OFF

CONFIG BORV=45, PWRT=ON,BOR=ON

CONFIG DEBUG=OFF, LVP=OFF, STVR=OFF


D0      EQU 10H

D1      EQU 11H

ADCONR   EQU 14H

MYREG    EQU 5H

        ORG 000H

        GOTO MAIN           ;Bypass the interrupt vector table

        ORG 0008H                                          ;interrupt vector table

                  BTFSS PIR1, ADIF                         ;did we get here due to A/D int?

                  RETFIE                                               ;No. Then return to main

                  GOTO AD_ISR           ;Yes. Then go to INT0 ISR





        ORG 100H

MAIN

        ;MOVLW 0X32

        ;MOVWF STKPTR

        CLRF TRISC                        ;PORTC O/P

        CLRF TRISD                                         ;PORTD O/P

        SETF TRISA                              ;Make RA0 I/P

                  MOVLW 07H                                  ;

                  MOVWF CMCON
```

```
        MOVLW 0X81          ;Fosc/64, ch. 0,A/D
        MOVWF ADCON0                ;load ADCON0 with 10 000 001
        MOVLW 04EH                          ;right justified, Fosc/64, AN0=analog
        MOVWF ADCON1                ;load ADCON1 with 01 001 110
        MOVLW 0XA9
        BCF PIR1, ADIF      ;clear ADIF for the first round
                BSF PIE1,ADIE               ;enable A/D Interrupt
                BSF INTCON,PEIE             ;enable peripheral interrupt
                BSF INTCON,GIE                      ;enable interrupt globally

OVER    CALL DELAY                          ;wait for Tacq(sample and hold time)
                BSF ADCON0, GO      ;start conversion
                BRA OVER        ;wait for EOC

DELAY   ORG 00150H
                MOVLW 008H
                MOVWF MYREG
AGAIN   NOP
                NOP
                NOP
                DECF MYREG,F
                BNZ AGAIN
                RETURN

AD_ISR
        ORG 00200H

;START  BSF ADCON0,GO
INCONV  ;BTFSC ADCON0,DONE
        ;BRA INCONV
        ;MOVFF ADRESL,ADCONR
```

```
        MOVFF ADRESH,ADCONR     ;Give High byte to ADCONR
    CALL DIVIDE           ;Call Divide Subroutine
    ;CALL DISPLAY         ;Call Display subroutine
    CALL TEN              ;Call Ten Subroutine
    CALL UNIT             ;Call UNIT Subroutine
        BCF PIR1, ADIF         ;clear ADIF interrupt flag bit
    ;BRA START
    RETFIE



DIVIDE   CLRF D0              ;Clears D0
    CLRF D1              ;Clears D1
    MOVLW D'51'           ;#1 Load 51 into WREG
EVEN     CPFSEQ ADCONR         ;#2
    BRA QUOTIENT          ;#3
    INCF D1,F             ;#4
    SUBWF ADCONR,F         ;#5
QUOTIENT CPFSGT ADCONR         ;#6

    BRA DECIMAL                          ;#7
    INCF D1,F             ;#8 Increment D1 for each time
                ;ADCONR is Greater than 51

    SUBWF ADCONR,F         ;#9 Subtract 51 from ADCONR

    BRA EVEN              ;#10
DECIMAL  MOVLW 0X05            ;#11
REMAIN   CPFSGT ADCONR         ;#12 Checks if ADCONR>5

    BRA DIVDONE           ;#13
    INCF D0,F             ;#14
    SUBWF ADCONR,F         ;#15 Subtract 5
```

```
        BRA REMAIN
DIVDONE  RETURN                ;#16
;DISPLAY  MOVFF D1,PORTC        ;#17 Output D1 on integer 7-seg
 ;       MOVFF D0,PORTD         ;#18 Output D0 on fractional 7-seg
 ;       RETURN
 ;       END


ORG 300H
UNIT
L1    MOVLW D'0'
        CPFSEQ D0
        BRA L2
        MOVLW 0C0H
        MOVWF PORTD
        RETURN
L2    MOVLW D'1'
        CPFSEQ D0
        BRA L3
        MOVLW 0F9H
        MOVWF PORTD
        RETURN
L3    MOVLW D'2'
        CPFSEQ D0
        BRA L4
        MOVLW 0A4H
        MOVWF PORTD
        RETURN
L4    MOVLW D'3'
        CPFSEQ D0
        BRA L5
```

```
        MOVLW 0XB0
        MOVWF PORTD
        RETURN
L5      MOVLW D'4'
        CPFSEQ D0
        BRA L6
        MOVLW 099H
        MOVWF PORTD
        RETURN
L6      MOVLW D'5'
        CPFSEQ D0
        BRA L7
        MOVLW 092H
        MOVWF PORTD
        RETURN
L7      MOVLW D'6'
        CPFSEQ D0
        BRA L8
        MOVLW 082H
        MOVWF PORTD
        RETURN
L8      MOVLW D'7'
        CPFSEQ D0
        BRA L9
        MOVLW 0F8H
        MOVWF PORTD
        RETURN
L9      MOVLW D'8'
        CPFSEQ D0
        BRA L10
        MOVLW 080H
        MOVWF PORTD
```

```
        RETURN
L10   MOVLW D'9'
        MOVLW 090H
        MOVWF PORTD
        RETURN

ORG 400H
TEN

LOOP1    MOVLW D'0'
        CPFSEQ D1
        BRA LOOP2
        MOVLW B'01000000'
        MOVWF PORTC
        RETURN
LOOP2    MOVLW D'1'
        CPFSEQ D1
        BRA LOOP3
        MOVLW B'01111001'
        MOVWF PORTC
        RETURN
LOOP3    MOVLW D'2'
        CPFSEQ D1
        BRA LOOP4
        MOVLW B'00100100'
        MOVWF PORTC
        RETURN
LOOP4    MOVLW D'3'
        CPFSEQ D1
        BRA LOOP5
        MOVLW B'00110000'
        MOVWF PORTC
```

```
        RETURN
LOOP5   MOVLW D'4'
        CPFSEQ D1
        BRA LOOP6
        MOVLW B'00011001'
        MOVWF PORTC
        RETURN
LOOP6   MOVLW D'5'
        MOVLW B'00010010'
        MOVWF PORTC
        RETURN
END
```