# Experiment 7 (DCMP Lab. 2017-2018/II)

## Objective:

To realize an 8-bit ADC (Analog-to-Digital Converter) with PIC16F877A and display the digital values on an LED Bank.

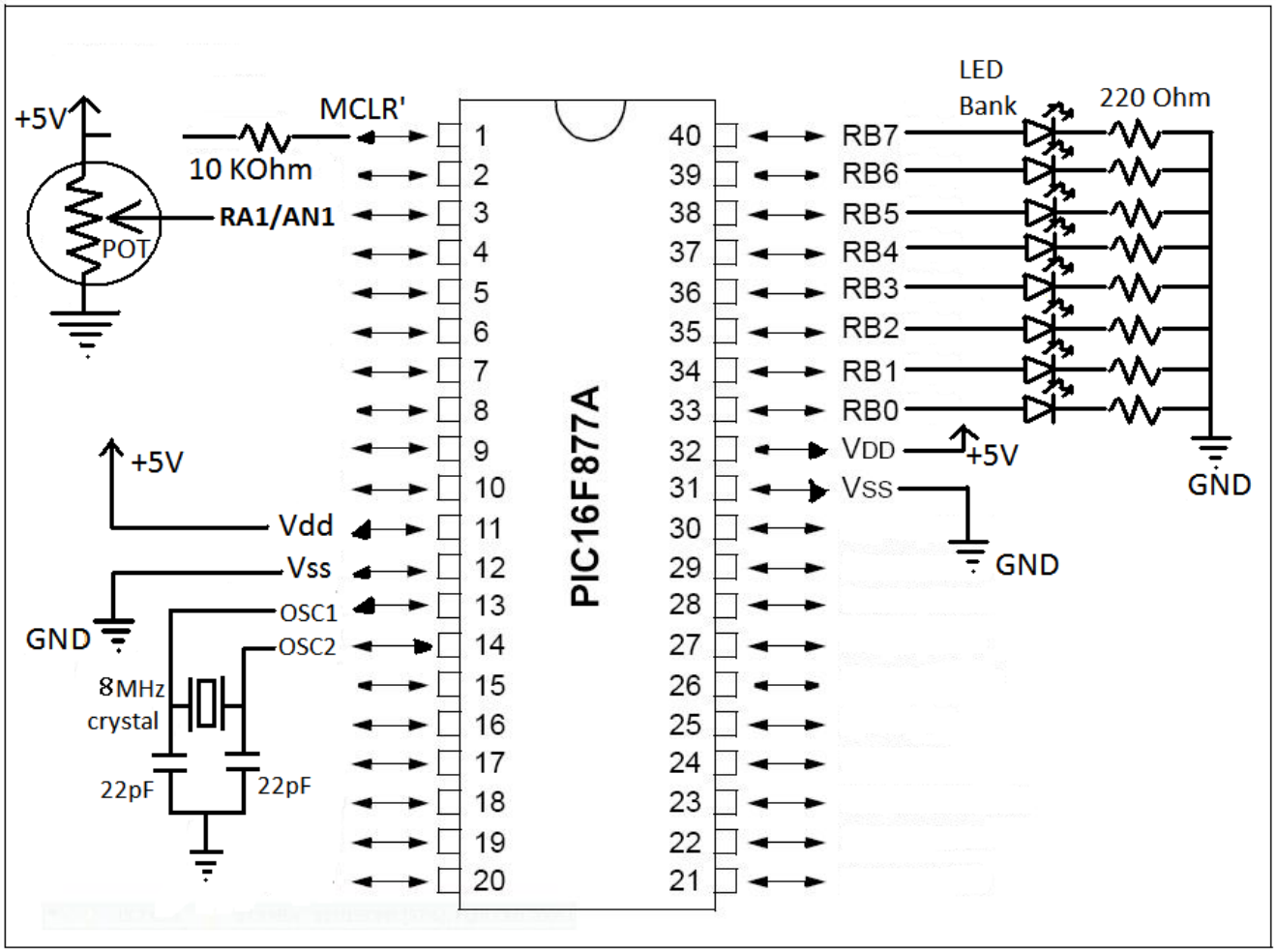
## Equipment:

1) PIC16F877A IC
2) Programmer of PIC16F877A
3) +5 V DC Supply
4) 2 Capacitors: 22 pF & a 8 MHz/ 10 MHz Crystal
5) A 10 kOhm resistor for $\overline{MCLR}$ Pin
6) LED's (each with a 220 Ohm resistor)
7) Connecting Wires

**Problem Statement:**
PIC16F877A has a 10-bit ADC with 8 analog channels. We have to take 8 most significant bits as the result and display the result through PORT B.

Use the AN1 channel as the analog input, while the entire PORTB (8-bit) as Output. Program the IC to convert any analog input at the AN1 Channel into an equivalent 8-bit Digital Word that can be displayed on PORTB.

**Circuit Connections:**
Connect a potentiometer at the AN1 Channel (Pin No.3 of IC), so as to vary its analog voltage from 0 to 5 Volts. Connect 8 LEDs on all the 8 channels of PORTB, as shown.



## Hints for Writing the Program:

### 1) Configuring the A/D Module

The A/D Module has to be configured by assigning appropriate values to the ADCON0 & ADCON1 Registers (AD Control Registers).

Refer to the PIC16F877A Datasheet to understand, how to fill up all the bits of these two registers.

To simplify the process, just put the value of ADCON1 Register as ‘ 0x00 ‘ (0000 0000).
(Ten bit result is left justified in ADRESH and ADRESL registers)

Details for filling up ADCON0 are given as follows.

**ADCON0 REGISTER (ADDRESS 1Fh)**

| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | U-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| ADCS1 | ADCS0 | CHS2 | CHS1 | CHS0 | GO/$\overline{\text{DONE}}$ | — | ADON |
| bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 |

Clear the following bits to zero:

bit7: ADCS1
bit6: ADCS0

ADCS1 and ADCS0 are AD clock select bits. (ADCS1=ADCS0=0 indicates AD clock to be fosc/2. You can choose your own frequency if required.)

bit1: Unimplemented

bits <5:3> must be chosen to choose one of the eight Analog Input Channels

| Bit 5 | Bit 4 | Bit 3 | Analog Channel |
|---|---|---|---|
| 0 | 0 | 0 | AN0 (RA0) |
| 0 | 0 | 1 | AN1 (RA1) |
| 0 | 1 | 0 | AN2 (RA2) |
| 0 | 1 | 1 | AN3 (RA3) |
| 1 | 0 | 0 | AN4 (RA5)) |
| 1 | 0 | 1 | AN5 (RE0) |
| 1 | 1 | 0 | AN6 (RE1) |
| 1 | 1 | 1 | AN7 (RE2) |

Choose the appropriate bits.

bit0: **ADON** decides whether the A/D Conversion is to be done or not.

1 = A/D Module is powered up i.e. working
0 = A/D Module is switched off.

bit2: **GO/** $\overline{DONE}$ is the most important bit.

1 = start the A/D conversion (this bit is automatically cleared to 0, when the conversion is complete).
0 = A/D conversion has stopped / not in progress.

After you choose the appropriate values for each bit, assign the appropriate word to the ADCON0 Register.
(e.g. : If you have decided the values of <bit7:bit0> as 1011 1100, then assign 0xBC to ADCON0)

**Writing values to Registers:**

*Every time you need to write some values into the Special Function Registers (SFR’s) : ADCON1, ADCON0, TRISA, TRISB, etc., you need to select the corresponding banks that contain these registers. (refer the IC Datasheet for this). Set/Clear RP1 & RP0 bits of STATUS Register for doing this.*

**2) Configure the Ports**

Configure RA1(AN1) Channel of Port A as Input Channel (by setting to '1' the appropriate bit of TRISA Register.)
Configure entire PORTB as Output Port (Set all bits of TRISB)

**3) Do A/D Conversion**

Write an assembly language program that can run continuously.

To start A/D Conversion, you will have to Set (make the bit '1') the GO/ $\overline{DONE}$ bit (bit2) of ADCON0 Register.
As soon as this is done, program will convert the analog input at AN1 and store it as a 10-bit Digital Word in ADRESH and ADRESL Register. Since the result will be left justified, ADRESH will contain the higher 8-bits of result. The AD conversion will take some time (a few 10's of μs).
After AD conversion is over, the bit2 of ADCON0 will automatically be cleared (to 0).

The completion of A/D conversion can be checked through **polling**.

Write a main program with polling (checking GO/ $\overline{DONE}$ bit) to read the A/D converter data (ADRESH) and output that to PORTB continuously. Change the potentiometer to vary the analog input and observe the change in the LEDs connected to PORTB.

Compile and test the program in MPLAB itself, using the MPLAB SIM simulator. If it works fine, proceed to burn the .hex file into the IC & test the circuit.

## OPTIONAL

The completion of A/D conversion can also be checked through A/D **interrupt**.

Write the main program and the interrupt service routine to read the A/D converter data (ADRESH) and output that to PORTB continuously. (You will have to configure suitable interrupt related registers for this. Especially, configure ADIE, ADIF, GIE, PEIE bits suitably. Refer to the datasheet of PIC16F877A, if required.)

Write a goto instruction so that program runs continuously. Change the potentiometer and observe the change in the LEDs connected to PORTB.