8-bit PIC MCUs - Hands On

Lab Manual

Table of Contents

Blank Sheet

Goals:

1. To configure I/O PORT and upon the switch press, to display 1010/0101 on the LEDs.
2. To configure Timer to generate 0.5sec tick, and to increment the LED display value on every tick.
3. To configure ADC, to measure the potentiometer voltage and display the digital value on LEDs.
4. To configure USART and transmit the read potentiometer voltage on to the console

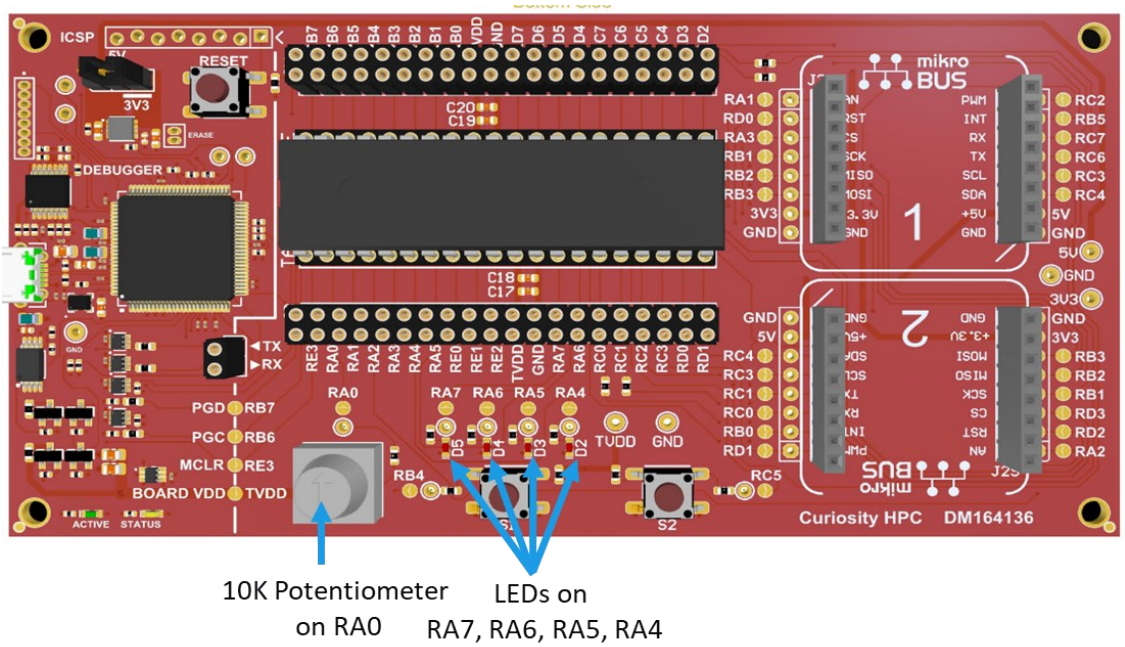
Pre-requisites:

1. MPLAB– X IDE usage
2. 'C'- language programming
3. Basic knowledge of PIC18F devices
4. Basic knowledge Data Visualizer plugin and its terminal settings.

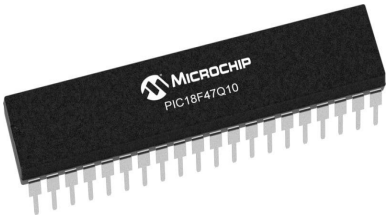
Blank Sheet

Required Tools:

1. Curiosity HPC board



2. PIC18F47Q10 device



3. MPLAB-X IDE

4. XC8 compiler

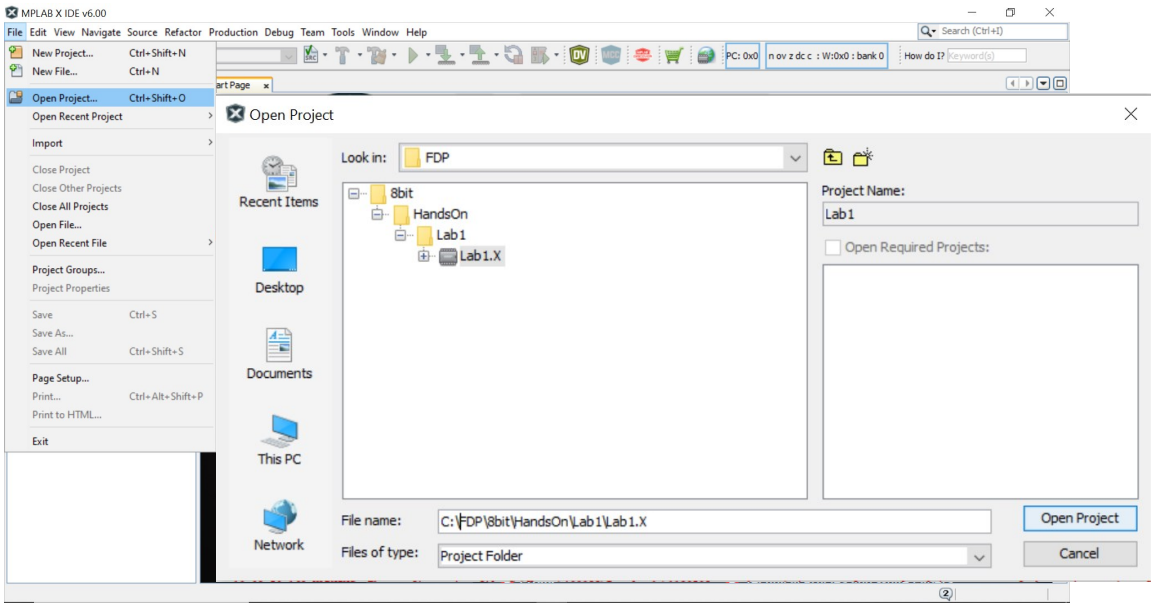
Blank Sheet

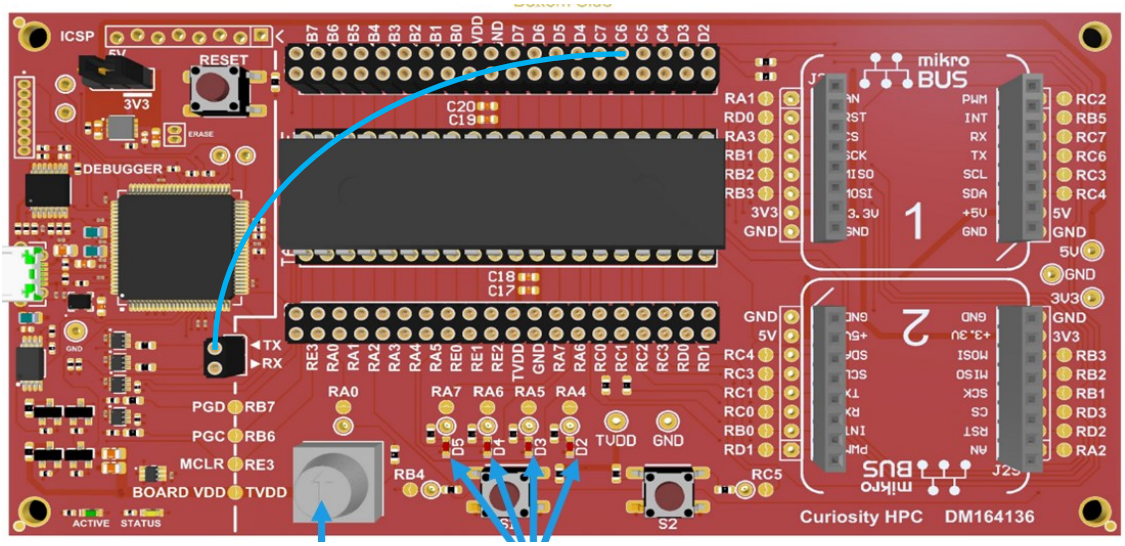
Setting up the Labs

1. **Open MPLAB-X IDE**
2. **Opening a project, say C:\FDP\8bit\HandsOn\Lab1\Lab1.x**
3. **Connect Curiosity HPC Board to your Laptop using the USB cable**
4. **Connect RC6 and TX jumpers using the jumper wire provided.**

The PIC18F47Q10 data sheet can be found at C:\FDP\8bit\HandsOn

The pic18F47q10.h file can be found at C:\Program Files\Microchip\xc8\v2.46\pic\include\proc





ANSELA Analog Select Register

Bit	7	6	5	4	3	2	1	0
	ANSELA7	ANSELA6	ANSELA5	ANSELA4	ANSELA3	ANSELA2	ANSELA1	ANSELA0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1

Bits 0, 1, 2, 3, 4, 5, 6, 7 – ANSELAn Analog Select on Pins RA[7:0]

Value	Description
1	Digital Input buffers are disabled
0	ST and TTL input buffers are enabled

TRISA Tri-State Control Register

Bit	7	6	5	4	3	2	1	0
	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1

Bits 0, 1, 2, 3, 4, 5, 6, 7 – TRISAn TRISA Port I/O Tri-state Control bits

Value	Description
1	Port output driver is disabled
0	Port output driver is enabled

RxyPPS **Pin Rxy Output Source Selection Register**

Bit	7	6	5	4	3	2	1	0
				RxyPPS[4:0]				
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bits 4:0 – RxyPPS[4:0] Pin Rxy Output Source Selection bits
See output source selection table for source codes.

```
// Register: ANSELC
#define ANSELC ANSELC
extern volatile unsigned char           ANSELC                  __at(0xF1C);
#ifndef _LIB_BUILD
asm("ANSELC equ 0F1Ch");
#endif
// bitfield definitions
typedef union {
    struct {
        unsigned ANSELC0                :1;
        unsigned ANSELC1                :1;
        unsigned ANSELC2                :1;
        unsigned ANSELC3                :1;
        unsigned ANSELC4                :1;
        unsigned ANSELC5                :1;
        unsigned ANSELC6                :1;
        unsigned ANSELC7                :1;
    };
} ANSELCbits_t;
```

Lab1: Configure and Use I/O Ports

In this Lab we use four port pins to drive LEDs, a port pin to sense switch press
Display "10100101" on LEDs when the switch is pressed.

The Ports/Pins used in this project are:
Pins RA7:RA4 used as digital outputs to drive the LEDs.
Pin RC5 is used as digital input connected to Switch S2
Pin RA0 is used as analog input connected to POT
Pin RC6 used as Tx pin (Digital).

To configure the Port Pins as:
Analog/Digital, ANSEL Registers are used
'0' -> Digital and '1' -> analog (default)
Input/Output, TRIS Registers are used
'0' -> Output and '1' -> Input (default)
LAT Register is used to output data
PORT Register is used to Read the input data
To assign a output function output function PPS Registers are used

Steps to do:

1. **Open the project C:\FDP\8bit\HandsOn\Lab1\Lab1.x**
2. **Open the file ports.c**
3. **Within ports_Initialize() function**
   Configure RA4:RA7 as digital output

```
ANSELA = 0b00001111;    //RA4:RA7 -> Digital
TRISA = 0b00001111;     //RA4:RA7 -> Output
                        //RA0 is maintained as  Analog Input
```

4. **Configure RC5 and RC6 as digital input**

```
ANSELCbits.ANSELC5 = 0;  //RC5 -> Digital
ANSELCbits.ANSELC6 = 0;  //RC6 -> Digital
```

5. **Open the file ports.h.**

Define the below #defines

```
#define PortADataOut      LATA
#define RB4DataIn PORTBbits.RB4
#define RC5DataIn PORTCbits.RC5
#define RC6PinFun RC6PPS
```

6. **Prototype the ports Initialize function**

```
void ports_Initialize(void);
```

```
# Syntax: config setting mapping
# <config macro>:<#pragma config setting>
# Syntax: null mapping
# <config macro>:
FEXTOSC_LP:FEXTOSC=LP
FEXTOSC_XT:FEXTOSC=XT
FEXTOSC_HS:FEXTOSC=HS
FEXTOSC_OFF:FEXTOSC=OFF
FEXTOSC_ECL:FEXTOSC=ECL
FEXTOSC_ECM:FEXTOSC=ECM
FEXTOSC_ECH:FEXTOSC=ECH
RSTOSC_HFINTOSC_64MHZ:RSTOSC=HFINTOSC_64MHZ
RSTOSC_RESERVED_1:RSTOSC=RESERVED_1
RSTOSC_EXTOSC_4PLL:RSTOSC=EXTOSC_4PLL
```

PIC18F47Q10 is the device used in these labs.
We need to configure the device using the device configuration bits.
System Clock -> 64MHz Internal oscillator
Watchdog Timer -> OFF

These are done by configuring the Config Bits by using the below syntax:

#pragma config <ConfigbitName = ConfigbitSetting>;

To use compiler provided Delay routine "_XTAL_FREQ" should be defined

7. **Open the Lab1.c file**

In this lab we are outputting "00000000" to "11111111" on the LEDs connected on RA7:RA4 incrementing on every press of the switch S2 connected on RC5.

8. **Define the below definitions**

```
#define  LEDs              PORTADataOut
#define  S2_Pressed        !RC5DataIn
#define  Event             S2_Pressed
```

9. **Declare a variable to store the value to be displayed on LEDs and assign '0' toit**

```
uint8_t  LED_Data = 0x00;
```

10. **In the main() function on every event we output 0x00 for 200ms, MS nibble of LED_Data for 400ms and then LS nibble of LED_Data for 400ms on LEDs.**

Before these initialize the Ports by calling Initialize function, refer the file Ports.h for the function.

```
        ports_Initialize();

        while(1){
                if (Event){
                   LEDs = 0x00;
                   __delay_ms(200);

                   LEDs = LED_Data;
                  __delay_ms(400);

                   LEDs = LED_Data << 4;
                   __delay_ms(400);

                  LED_Data = LED_Data++;
                  while(Event);
                }
        }
```

T0CON1 Timer0 Control Register 1

Bit	7	6	5	4	3	2	1	0
	T0CS[2:0]			T0ASYNC	T0CKPS[3:0]			
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:5 – T0CS[2:0] Timer0 Clock Source Select
Refer the clock source selection table

Bits 3:0 – T0CKPS[3:0] Prescaler Rate Select

Value	Description
1111	1:32768
1110	1:16384
1101	1:8192
1100	1:4096
1011	1:2048
1010	1:1024
1001	1:512
1000	1:256
0111	1:128
0110	1:64
0101	1:32
0100	1:16
0011	1:8
0010	1:4
0001	1:2
0000	1:1

T0CS	Clock Source
111	CLC1_out
110	MFINTOSC (500 kHz)
101	SOSC
100	LFINTOSC
011	HFINTOSC
010	$F_{OSC}/4$
001	Pin selected by T0CKIPPS (Inverted)
000	Pin selected by T0CKIPPS (Non-inverted)

T0CON0 Timer0 Control Register 0

Bit	7	6	5	4	3	2	1	0
	T0EN		T0OUT	T016BIT	T0OUTPS[3:0]			
Access	R/W		R	R/W	R/W	R/W	R/W	R/W
Reset	0		0	0	0	0	0	0

Bit 7 – T0EN TMR0 Enable

Value	Description
1	The module is enabled and operating
0	The module is disabled

Bit 5 – T0OUT TMR0 Output

Bit 4 – T016BIT TMR0 Operating as 16-Bit Timer Select

Value	Description
1	TMR0 is a 16-bit timer
0	TMR0 is an 8-bit timer

Bits 3:0 – T0OUTPS[3:0] TMR0 Output Postscaler (Divider) Select

Value	Description
1111	1:16 Postscaler
1110	1:15 Postscaler
1101	1:14 Postscaler
1100	1:13 Postscaler
1011	1:12 Postscaler
1010	1:11 Postscaler
1001	1:10 Postscaler
1000	1:9 Postscaler
0111	1:8 Postscaler
0110	1:7 Postscaler
0101	1:6 Postscaler
0100	1:5 Postscaler
0011	1:4 Postscaler
0010	1:3 Postscaler
0001	1:2 Postscaler
0000	1:1 Postscaler

Lab2: Configure and Use Timer

In this Lab similar to Lab1 we use four port pins to drive LEDs but
Display "10100101" on LEDs every 1s.

Configure Timer 0 as a 8-bit periodic Timer.
Its clock source to System clock/4 (FOSC/4) -> 64MHz/4 = 16MHz
Its Prescaler to Max (1:32768) -> 16MHz/32768 = ~488Hz

To get 0.5s (2Hz) Ticks we need to load the period register (TMR0H) with (488/2)-1
To get 1s (1Hz) we need to select Postscaler 1:2

Steps to do:

1. **open project C:\FDP\8bit\HandsOn\Lab2\Lab2.x**
2. **Open the file tmr.c**
3. **Within tmr_Initialize() function**
   Configure Timer 0 as a 8-bit periodic Timer.

   ```
   T0CON0bits.T016BIT = 0;
   ```

4. **Clock Source to FOSC/4**

   ```
   T0CON1bits.T0CS = 0b010;
   ```

5. **Prescaler to 1:32768**

   ```
   T0CON1bits.T0CKPS = 0b1110;
   ```

6. **Period to 243 and load Timer with 0x00**

   ```
   TMR0H = 243;
   TMR0L = 0;
   ```

7. **Select the Postscaler as 1:2**

   ```
   T0CON1bits.T0OUTPS = 0b0001;
   ```

8. **Turn ON the timer**

   ```
   T0CON0bits.T0EN = 1;
   ```


MICROCHIP
ACADEMIC
PROGRAM

9. **Open the file tmr.h**

10. **Define the below #defines**

```
#define TimerTick           TMR0IF
```

11. **Prototype the timer Initialize function**

```
void tmr0_Initialize(void);
```

12. **Open the Lab2.c file**

In this Lab our event is TimerTick and not S2_Pressed

```
//#define  Event              S2_Pressed
#define  Event               TimerTick
```

13. **In the main()**

In addition to initialization the Ports initialize Timer0 by calling Initialize function, refer the file tmr0.h for the function.

In this lab as the event is not switch pressed (which was hardware set/cleared), it is TimerTick (which is hardware set but software cleared), we need to clear the event.

```
        ports_Initialize();

        tmr0_Initialize();

        while(1){
                if (Event){
                   LEDs = 0x00;
                   __delay_ms(200);

                   LEDs = LED_Data;
                  __delay_ms(400);

                   LEDs = LED_Data << 4;
                   __delay_ms(400);

                  LED_Data = LED_Data++;

                   //while(Event);
                   Event = 0;
                  }
        }
```

ADCON0 ADC Control Register 0

Bit	7	6	5	4	3	2	1	0
	ADON	ADCONT		ADCS		ADFM		ADGO
Access	R/W	R/W		R/W		R/W		R/W/HC
Reset	0	0		0		0		0

Bit 7 – ADON ADC Enable bit

Value	Description
1	ADC is enabled
0	ADC is disabled

Bit 4 – ADCS ADC Clock Selection bit

Value	Description
1	Clock supplied from FRC dedicated oscillator
0	Clock supplied by Fosc, divided according to ADCLK register

Bit 2 – ADFM ADC results Format/alignment Selection

Value	Description
1	ADRES and ADPREV data are right justified
0	ADRES and ADPREV data are left justified, zero-filled

Bit 0 – ADGO ADC Conversion Status bit

Value	Description
1	ADC conversion cycle in progress. Setting this bit starts an ADC conversion cycle. The bit is cleared by hardware as determined by the ADCONT bit
0	ADC conversion completed/not in progress

ADCLK ADC Clock Selection Register

Bit	7	6	5	4	3	2	1	0
			ADCS[5:0]					
Access			R/W	R/W	R/W	R/W	R/W	R/W
Reset			0	0	0	0	0	0

Bits 5:0 – ADCS[5:0] ADC Conversion Clock Select bits

Value	Description
n	ADC Clock frequency = $F_{OSC}/(2*(n+1))$

ADACQ ADC Acquisition Time Control Register

Bit	7	6	5	4	3	2	1	0
	ADACQ[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – ADACQ[7:0] Acquisition (charge share time) Select bits

Value	Description
0x01 to 0xFF	Number of ADC clock periods in the acquisition time
0x00	Acquisition time is not included in the data conversion cycle

ADPCH ADC Positive Channel Selection Register

Bit	7	6	5	4	3	2	1	0
			ADPCH[5:0]					
Access			R/W	R/W	R/W	R/W	R/W	R/W
Reset			0	0	0	0	0	0

Bits 5:0 – ADPCH[5:0] ADC Positive Input Channel Selection bits
See Channel Selection Table for input selection details.

Lab3: Configure and Use ADC

In this Lab similar to Lab2 we use TimerTick as our Event
But instead of "10100101" we display ADC Result on LEDs every 0.5s.

Configure ADC:
Switch ON ADC
Configure Result as Left justified as the 8-bits of our interest are in one register.
Select System clock (FOSC) as ADC clock source.
Load ADCLK to get The conversion clock (1/TAD) 1MHz, the ADC Result of this chip is not guaranteed if conversion clock is more than 1MHz.
As AN0 is default channel no need to select the channel
As VDD and VSS are default references for ADC no need to change the references.

Provide an API to trigger and read the conversion result.

To get 1us TAD we need to load the ADCLK register with (64000000/1000000)-1 = 63

Steps to do:

1. **open project C:\FDP\8bit\HandsOn\Lab3\Lab3.x**
2. **Open the file adc.c**
3. **Within adc_Initialize() function**
   Turn ON ADC
   Result alignment as Left justified
   Clock source as System clock (FOSC)
   Conversion clock to 1MHz
   As we are using AN0 to read the potentiometer and it is the default channel selected we need not select any channel.

```
ADCON0bits.ADON = 1;
ADCON0bits.ADFM = 0;
ADCONbits.ADCS = 0b010;
ADCLK = 63;
```

4. **Within adc_ResultGet() function**
   Trigger conversion by setting GO bit
   Wait for conversion completion by waiting for GO to get cleared
   Return the result (as left justified the required 8-bits are stored in ADRESH)

```
ADCON0bits.ADGO = 1;
while (ADCON0bits.ADGO);
return (ADRESH);
```

ADREF ADC Reference Selection Register

Bit	7	6	5	4	3	2	1	0
				ADNREF			ADPREF[1:0]	
Access				R/W			R/W	R/W
Reset				0			0	0

Bit 4 – ADNREF ADC Negative Voltage Reference Selection bit

Value	Description
1	V_{REF-} is connected to external V_{REF-}
0	V_{REF-} is connected to AV_{SS}

Bits 1:0 – ADPREF[1:0] ADC Positive Voltage Reference Selection bits

Value	Description
11	V_{REF+} is connected to internal Fixed Voltage Reference (FVR) module
10	V_{REF+} is connected to external V_{REF+}
01	Reserved
00	V_{REF+} is connected to V_{DD}

5. **Open the file adc.h**
6. **Prototype the adc Initialize function**
   Prototype Result get function

```
void adc_Initialize(void);
uint8_t adc_ResultGet(void);
```

7. **Open the Lab3.c file**

In this Lab our event remains TimerTick, but what we display on LEDs is not "0x00 to 0xff" but the ADC result.

13. In the main()

In addition to initialization the Ports and Timer0 initialize ADC by calling Initialize function, refer the file adc.h for the function.

Load ADC Result into LED_Data
Comment incrementing of LED_Data

```
    ports_Initialize();

tmr0_Initialize();

    adc_Initialize();

    while(1){
        if (Event){
          LEDs = 0x00;
          __delay_ms(200);

        LED_Data = adc_ResultGet();

          LEDs = LED_Data;
          __delay_ms(400);

          LEDs = LED_Data << 4;
          __delay_ms(400);

        //LED_Data = LED_Data++;

          //while(Event);
          Event = 0;
        }
    }
```

TXxSTA Transmit Status and Control Register

Bit	7	6	5	4	3	2	1	0
	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D
Access	R/W	R/W	R/W	R/W	R/W	R/W	RO	R/W
Reset	0	0	0	0	0	0	1	0

Bit 6 – TX9 9-bit Transmit Enable bit

Value	Description
1	Selects 9-bit transmission
0	Selects 8-bit transmission

Bit 5 – TXEN Transmit Enable bit
Enables transmitter(1)

Value	Description
1	Transmit enabled
0	Transmit disabled

SPxBRG Baud Rate Determination Register

Bit	15	14	13	12	11	10	9	8
	SPBRGH[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	SPBRGL[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 15:8 – SPBRGH[7:0] Baud Rate High Byte Register

Bits 7:0 – SPBRGL[7:0] Baud Rate Low Byte Register

TXxREG Transmit Data Register

Bit	7	6	5	4	3	2	1	0
	TXREG[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – TXREG[7:0] Transmit Data

Lab4: Configure and Use UART

In this Lab similar to Lab3 we display ADC Result on LEDs every 0.5s, in addition we transmit the corresponding Voltage value to the terminal.

Configure UART:
Switch ON Serial Port
Configure Baudrate to 9600

Provide an API to transmit a character and putch() API for STDIO library.

To get 9600 Baud we need to load the SPBRG register with (64000000/(64*9600)) -1 = 103

Steps to do:

1. **open project C:\FDP\8bit\HandsOn\Lab4\Lab4.x**
2. **Open the file uart.c**
3. **Within uart_Initialize() function**
   Turn ON Serial Port
   Set Baudrate to 9600
   we are not enabling Transmit here.

   ```
   RC1STAbits.SPEN = 1;
   SPBRG = 103;
   ```

4. **Within uart_Transmit() function**
   Turn On transmission by setting TXEN bit
   Load data to be transmitted (The character passed as an argument)
   Wait till the transmission is complete
   Turn OFF transmission.

   ```
   TX1STAbits.TXEN = 1;
   TX1REG = PotVoltage;
   while (!TX1STAbits.TRMT);
   TX1STAbits.TXEN = 0;
   ```

Table 17-2. Peripheral PPS Output Selection Codes

RxyPPS	Pin Rxy Output Source	PORT To Which Output Can Be Directed							
		28-Pin Devices			40-Pin Devices				
0x1F	CLC8OUT	—	B	C	—	B	—	D	—
0x1E	CLC7OUT	—	B	C	—	B	—	D	—
0x1D	CLC6OUT	A	—	C	A	—	C	—	—
0x1C	CLC5OUT	A	—	C	A	—	C	—	—
0x1B	CLC4OUT	—	B	C	—	B	—	D	—
0x09	EUSART1 (TX/CK)	—	B	C	—	B	C	—	—

5. **Open the file uart.h**
6. **Define UART1TX as 9**
7. **Prototype the uart Initialize function**
   Prototype uart_Transmit function
   Prototype putch() function

```
void uartc_Initialize(void);
void uart_Transmit(uint8_t PotVoltage);
```

8. **Open the Lab4.c file**

In this Lab our event remains TimerTick, but what we display on LEDs is not "10100101" but the ADC result.

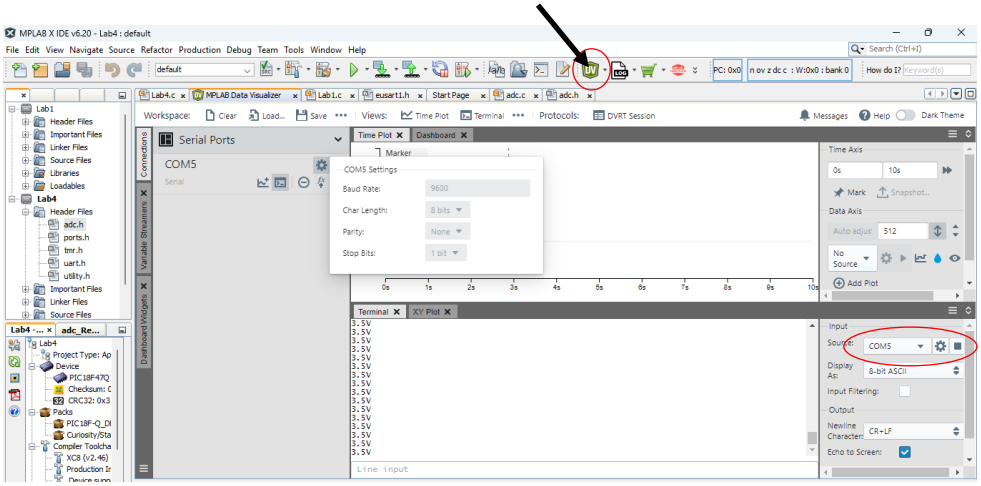
9. **In the main()**
   In addition to initialization the Ports and Timer0 initialize ADC by calling Initialize function, refer the file uart.h for the function. Set RC6 as Transmit pin.

```
ports_Initialize();

tmr0_Initialize();

adc_Initialize();

uart_Initialize();
RC6PinFun =  UART1TX;

while(1){
        if (Event){
          LEDs = 0x00;
          __delay_ms(200);

         LED_Data = adc_ResultGet();

          LEDs = LED_Data;
          __delay_ms(400);

          LEDs = LED_Data << 4;
          __delay_ms(400);

        printf(ConvertToVoltage(LED_Data));

          //while(Event);
          Event = 0;
        }
}
```

To set up the terminal refer the below picture

1. **Click on DV**
2. **Select COM5 for the terminal**
3. **Set the Baudrate as 9600,**
   Data length as 8-bits,
   Parity as None,
   Stop Bits as 1-bit
4. **Click on RUN**

Click DV



Select COM5 and RUN

Verify the settings
Baudrate : 9600
Data length : 8
Parity : None
Stop Bits : 1