Department of Computer Engineering
Digital Hardware Systems
CpE 3201 - Embedded Systems

Practical Activity #6
Universal Synchronous Asynchronous Receiver Transmitter (USART)

Exercise Objectives:
1. Configure the Universal Synchronous Asynchronous Receiver Transmitter (USART) module of the PIC16F877A
2. Use asynchronous data transmit and receive
3. Applying USART in basic embedded systems application

Student Outcomes:
At the end of the exercise, students must be able to:
1. learn how to configure the USART module
2. use USART to transmit and receive asynchronously
3. develop an embedded systems application utilizing USART

Tools Required:
The following will be provided for the students:
1. MPLAB IDE v8.92 + MPLAB XC8 v1.33
2. Proteus v8.11

Delivery Process:
The instructor will conduct a briefing for this practical activity. Notes from the lecture class will provided for reference purposes. All inquiries pertaining to the latter must be referred to this manual.

Note:
Images and other contents in this manual are copyright protected. Use of these without consent from the respective authors is unauthorized.

Part I. USART Module of PIC16F877A

The Universal Synchronous Asynchronous Receiver Transmitter (USART) module is one of the two serial I/O modules. The USART can be configured as a full-duplex asynchronous system that can communicate with peripheral devices. It can be configured in the following modes:

- Asynchronous (full-duplex)
- Synchronous–Master (half-duplex)
- Synchronous–Slave (half-duplex)

The registers associated with the USART module are:

- TXSTA (Transmit Status & Control Register)
- RCSTA (Receive Status & Control Register)
- SPBRG (Baud Rate Register)
- PIE1 (Peripheral Interrupts Enable Register)
- PIR1 (Peripheral Interrupts Flag Register

<u>*USART Baud Rate Generator*</u>

The BRG supports both the Asynchronous and Synchronous modes of the USART. It is a dedicated 8-bit baud rate generator. The SPBRG register controls the period of a free running 8-bit timer. In Asynchronous mode, bit BRGH (TXSTA<2>) also controls the baud rate while in Synchronous mode, bit BRGH is ignored. For more information, see page 113 of the PIC16F87X data sheet.

TABLE 10-1: BAUD RATE FORMULA

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = FOSC/(64 (X + 1))	Baud Rate = FOSC/(16 (X + 1))
1	(Synchronous) Baud Rate = FOSC/(4 (X + 1))	N/A

Legend: X = value in SPBRG (0 to 255)

Baud Rate Calculation

From Table 10-1 and Tables 10-3* and 10-4*, we can calculate the value of SPBRG register depending on the desired baud rate and given Fosc. For example, if the desired baud rate for asynchronous, high speed operation is 9.6Kbd and Fosc = 4MHz then the value of SPBRG is calculated as:

$$Baud\ Rate = \frac{F_{OSC}}{16(X+1)}$$ From Table 10, where X is the value of SPBRG (0 to 255)

$$9.6Kbd = \frac{4MHz}{16(X+1)}$$

$$X = \frac{4MHz}{16(9.6Kbd)} - 1$$

$$X = 25.041\ or\ 25\ (0x19)$$

** see page 114 of the PIC16F87X data sheet for details*

Asynchronous Mode Transmit

In asynchronous communication, both module does not require a clock to synchronize the transmit and receive operations. Both modules require they have the same settings like baud rate, number of data bits, parity bit and number of stop bits.

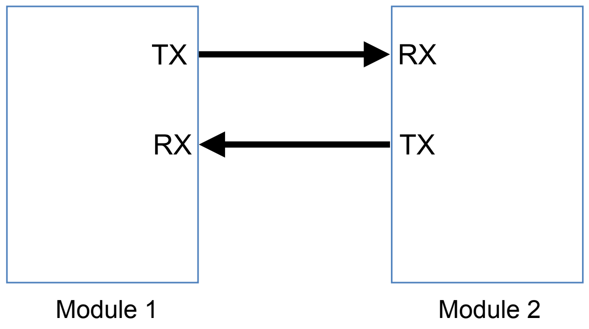


Fig.1. Synchronous communication

The advantage of asynchronous transmit receive is that it has one less wire and it can operate in full-duplex mode compared to synchronous transmission.

Steps in setting up asynchronous mode transmission:

1. Initialize the SPBRG register for the appropriate baud rate (Section 10.1 “USART Baud Rate Generator (BRG)”).
2. Enable the synchronous master serial port by setting bits SYNC, SPEN and CSRC.
3. If interrupts are desired, set enable bit TXIE.
4. If 9-bit transmission is desired, set bit TX9.
5. Enable the transmission by setting bit TXEN. 6.
6. If 9-bit transmission is selected, the ninth bit should be loaded in bit TX9D.
7. Start transmission by loading data to the TXREG register.
8. If using interrupts, ensure that GIE and PEIE (bits 7 and 6) of the INTCON register are set.

Below is the `main()` with the USART configuration to operate in asynchronous mode. The following example will send a character.

```
void main(void)
{
      SPBRG = 0x19;    // 9.6K baud rate @ FOSC=4MHz, asynchronous high speed
                       // (see formula in Table 10-1)
      SYNC = 0;        // asynchronous mode (TXSTA reg)
      SPEN = 1;        // enable serial port (RCSTA reg)
      TX9 = 0;         // 8-bit transmission (TXSTA reg)
      BRGH = 1;        // asynchronous high-speed (TXSTA reg)
      TXEN = 1;        // transmit enable (TXSTA reg)

      for(;;)          // foreground routine
      {
          while(!TRMT);  // wait until transmit shift register is empty
          TXREG = 'A';   // write data to be sent to TXREG
      }
}
```

Before transmitting, the transmit shift register should be empty by checking TMRT bit (0-register full, 1-register empty).

Open Proteus and construct the required circuit with filename "LE6-1.pdsprj". Connect a Virtual Terminal to the MCU via the TX (RC6) and RX (RC7). A virtual terminal will act as a device with a UART module to communicate with the MCU. It can display the received characters and can send characters by typing characters in the virtual terminal window. To send, click the virtual terminal window and press any character on the keyboard (no need to press enter).

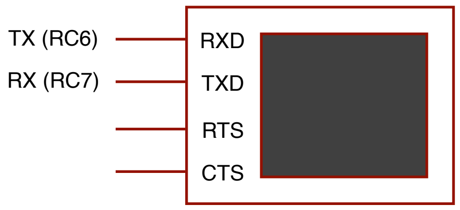


Fig. 2. Virtual Terminal in Proteus

In MPLAB, create a new project using the project wizard with the name "CpE 3201-LE6". Create a new source file with the filename "LE6-1.c" and add it to the project. Write the source code with the pre-processor directives. Build the program and debug if necessary. After successfully building the source code, simulate the program in Proteus ("LE6-1.pdsprj") by loading the generated .hex file to the microcontroller. The virtual terminal shall display letter 'A' continuously.

Asynchronous Mode Receive

Steps in setting up asynchronous mode reception:

1. Initialize the SPBRG register for the appropriate baud rate. If a high-speed baud rate is desired, set bit BRGH (Section10.1 "USART Baud Rate Generator (BRG)").
2. Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN.
3. If interrupts are desired, then set enable bit RCIE.
4. If 9-bit reception is desired, then set bit RX9.
5. Enable the reception by setting bit CREN
6. Flag bit RCIF will be set when reception is com- plete and an interrupt will be generated if enable bit RCIE is set.
7. Read the RCSTA register to get the ninth bit (if enabled) and determine if any error occurred during reception.
8. Read the 8-bit received data by reading the RCREG register.
9. If any error occurred, clear the error by clearing enable bit CREN.
10. If using interrupts, ensure that GIE and PEIE (bits 7 and 6) of the INTCON register are set.

Below is the `main()` with the USART configuration to operate in asynchronous mode. The following example will receive a character and outputs the ASCII code to PORTB (via 8 LEDs).

```
void main(void)
{
      SPBRG = 0x19;    // 9.6K baud rate @ FOSC=4MHz, asynchronous high speed
                       // (see formula in Table 10-1)
      SYNC = 0;        // asynchronous mode (TXSTA reg)
      SPEN = 1;        // enable serial port (RCSTA reg)
      RX9 = 0;         // 8-bit reception (TXSTA reg)
      BRGH = 1;        // asynchronous high-speed (TXSTA reg)
      CREN = 1;        // enable continuous receive (RCSTA reg)
      TRISA = 0x00;    // set all ports in PORTB to output
      PORTB = 0x00;    // all LEDs are off

      for(;;)          // foreground routine
      {
          while(!RCIF);     // wait until receive buffer is full
          PORTB = RCREG;    // read the receive register
      }
}
```

RCIF is set to 1 when a frame is received. This flag is set irregardless of RCIE (either enabled or not).

Open Proteus and construct the required circuit with filename “LE6-2.pdsprj”. Connect a Virtual Terminal to the MCU via the TX (RC6) and RX (RC7). Use the *LED-BARGRAPH-RED* for the LEDs and connect it to PORTB. Configure the virtual terminal to the same configuration of the MCU.

In MPLAB, create a new project using the project wizard with the name “CpE 3201-LE6”. Create a new source file with the filename “LE6-2.c” and add it to the project. Write the source code with the pre-processor directives. Build the program and debug if necessary. After successfully building the source code, simulate the program in Proteus (“LE6-2.pdsprj”) by loading the generated .hex file to the microcontroller. To send, type a character on the virtual terminal window and observe the output in PORTB. The ASCII code of the character sent shall be displayed via the LED bar graph.

Part IV. Hands-on Exercise

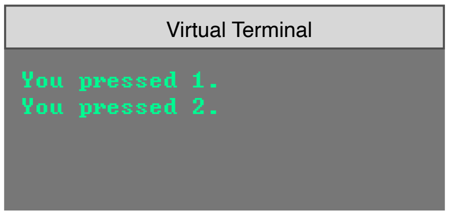
Part 1

Open the firmware project file “LE6-1.pdsprj” and save it as “LE6-3.pdsprj”. Connect a 3x4 keypad with a 74C922 encoder to PORTB. Connect a Virtual Terminal to the MCU via the TX (RC6) and RX (RC7).

In MPLAB, create a new source file with the filename “LE6-2.c” and add it to the project “CpE 3201-LE5”. Write a program that will send a string to the the virtual terminal when a key in the keypad is pressed. For example:

key	string
1	“You pressed 1.”
2	“You pressed 2.”
..	

After a string is sent, send a “new line” character to the move cursor of the virtual terminal to the next line (see Figure 3).



```
You pressed 1.
You pressed 2.
```

Fig. 3. Display in the virtual terminal

Build the program and debug if necessary. After successfully building the source code, simulate the program in Proteus (“LE6-3.pdsprj”) by loading the generated .hex file to the microcontroller. The appropriate string should be displayed based on the key being pressed.

Part 2

Open the firmware project file “LE6-3.pdsprj” and save it as “LE6-4.pdsprj”. Remove the virtual and add a new instance of PIC16F877A which we will call MCU2. Connect a power supply and configure MCU2. Connect also a 7-segment display to PORTB of MCU2. Connect the RX of MCU1 to the TX of MCU2 and TX of MCU1 to RX of MCU similar to Figure 1.

(For MCU1)

In MPLAB, create a new project using the project wizard with the name “CpE 3201-LE6-TX”. Create a new source file with the filename “LE6-3-TX.c” and add it to the project. Write a program that will send a character based on the key pressed. For example, pressing 1 will send the character ‘1’.

(For MCU2)

In MPLAB, create a new project using the project wizard with the name “CpE 3201-LE6-RX”. Create a new source file with the filename “LE6-3-RX.c” and add it to the project. Write a program that will read the character sent by MCU1 and the character will be displayed on the 7-segment display. For characters ‘*’ and ‘#’, the display will be blank.

Build the programs and debug if necessary. After successfully building the source code, simulate the program in Proteus (“LE6-4.pdsprj”) by loading the generated .hex files to MCU1 (“CpE 3201-LE6-TX.hex”) and MCU2 (“CpE 3201-LE6-RX.hex”). Press a key on the keypad in MCU1 and verify the display in MCU2.

Instructions for submission:

- Submit the following:
  - LE6-3.pdsprj
  - LE6-3.c
  - LE6-4.pdsprj
  - LE6-4-TX.c
  - LE6-4-RX.c
- Submit the files in Canvas in the correct order and file format.

Assessment

Criteria	Outstanding (4 pts)	Competent (3 pts)	Marginal (2 pts)	Not Acceptable (1 pt)	None (0 pts)
Feature Configuration	Configuration was properly done.	Configuration has minor errors.	-	Configuration is incorrect.	No project created.
Functionality	The systems function perfectly.	The system functions with minor issues.	The system has several issues which already affect the functionality.	The presented system is non-functioning.	No project created.

Firmware	Firmware created successfully without issues and followed the requirements.	Firmware created successfully with some issues and followed the requirements.	Firmware has issues and missing some of the requirements.	Firmware has several issues and did not follow the requirements.	No project created.

Copyright Information

References

PIC16F87X Data Sheet, Microchip Technology Inc. 2003.
CpE 3201 Lecture Notes on USART in PIC16F877A.

Change Log:

Date	Version	Author	Changes
March 16, 2021	1.0	Van B. Patiluna	- Initial draft.
March 17, 2021	1.0.1	Van B. Patiluna	- Corrected the exercise objectives.
March 22, 2021	1.0.2	Van B. Patiluna	- Added details about the virtual terminal.
March 23, 2021	1.0.3	Van B. Patiluna	- Corrected some typographical errors.