# Experiment 3: Universal Asynchronous Receiver Transmitter

## Objectives

The purpose of this experiment is to learn about asynchronous communications and how to communicate with a microcontroller using a terminal emulation program to implement a point-to point-serial link between the ChipKIT$^{TM}$ MX7 and a PC.

## Equipment List

- ChipKIT$^{TM}$ Pro MX7 processor board with USB cable
- Microchip MPLAB ® X IDE
- MPLAB ® XC32++ Compiler
- PC-based terminal emulations (HyperTerminal®)

## Overview

Asynchronous communication is a serial data protocol that has been in use for many years. Normally eight bits of data are transmitted at a time. There are other less commonly used modes that can send 5, 6, or 7 bits of data. Each byte of data is framed by a start bit and a stop bit (Table 2). A symbol is defined as a start, data, parity, or stop bit. It is common to define communications speed as bits per second. The bit rate is defined as the inverse of the period of a unit symbol. Although the common standard bit rates are 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, and 115200, communications are possible at any rate provided that the sender and receiver use the same rate. For most asynchronous communications, the term "baud" is commonly used interchangeably with the term "bit rate."

The Universal Asynchronous Receiver Transmitter (UART) module is one of the serial I/O modules available in PIC32MX Family devices. The UART is a full-duplex, asynchronous communication channel that communicates with peripheral devices and personal computers through protocols such as RS-232, RS-485, LIN 1.2 and IrDA®.

| Bit Number | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Start Bit | 5-8 Data Bits | | | | | | | | Stop Bit(s) | |
| | Start | Data 0 | Data 1 | Data 2 | Data 3 | Data 4 | Data 5 | Data 6 | Data 7 | Stop | |

Table 2: Simple serial communication

The PIC32MX795 microcontroller can provide up to six UARTs. Due to conflicting uses of many of the pins used by the UARTs, the Cerebot MX7cK is designed to allow use of two of them: UART1 and UART2. The UARTs can provide either a 2-wire or a 4- wire asynchronous serial interface. The 2-wire interface provides receive (RX) and transmit (TX) pins. The 4-wire interface includes request-to-send (RTS) and clear-to-send (CTS) in addition to receive and transmit.

UART1 can be accessed from Pmod connector JE and UART2 can be accessed from Pmod connector JF using the following pins:

- U1CTS - JE-01
- U1TX - JE-02
- U1RX - JE-03
- U1RTS - JE-04
- U2CTS - JF-01
- U2TX - JF-02
- U2RX - JF-03
- U2RTS - JF-04

# UART Programming

The UART peripheral must be initialized to set the communications bit rate, the parity option, and the number of data bits to send and receive. Additionally, the transmit and receive have separable enabled control bits.

The MPLAB Harmony Framework provides the possibility of initializing and configuring USART by using the MPLAB Harmony Configurator (MHC). To do so, follow the following steps:

- Click on the **Option** tab in the MHC window.
- Choose **Harmony Framework Configuration>Drivers>USART**.
- Check the box beside **Use USART Driver?**.
- We will use **STATIC** as driver implementation.
- Un-check the box beside **Interrupt Mode** since we will use a polled implementation.
- Check the other USART options. We can notice that the USART Driver default options configure the USART with the following serial port settings. You will need to configure the PC-based terminal program to use these same settings:
  - Baud Rate: 9600
  - Data: 8-bit
  - Parity: none
  - Number of Stop bits: 1
  - Flow control: none

# Experiment

## Part 1: Receive/echo byte data

In this part, you are asked to send text characters (bytes) from the serial port (HyperTerminal) to PIC32. The PIC32 will increment the ASCII value of the received byte and send the new byte back to PC.

- Initially, generate a new harmony project. Initialize the system as in Experiment 2 (Input/Output pins, clock diagram, ...).
- Communications will use the PC terminal emulation program for a data rate of 57.6k, no parity, 8 data bits and one stop bit.
- Initialize the USART driver in the MHC as described earlier.
- Modify the source code of the created project to perform the following tasks:
  - Add any necessary States in app.h to represent the transmit and receive tasks.
  - In the same file, modify the APP_DATA structure to insert two character objects in which received and sent bytes will be kept.
  - Modify the app.c code to implement the transmit and receive states in the APP_Tasks() function.
  - Notice that before reading bytes from USART, make sure that the buffer is not empty first.
  - The same logic is applied to the transmit state. Make sure that the transmit buffer is not full before trying to write bytes.

**Note:** You can use functions from the USART harmony library, such as drv_usart_static_byte_model.c. In this file, you can find functions as DRV_USART0_ReceiverBufferIsEmpty() and DRV_USART0_ReadByte(). Check the content of this source file for more functions and details. This file and others related to USART can be found in Source Files>system_config>default >framework>driver>usart>src.

## Part 2: Receive/echo String of data

In this experiment, you are asked to send text data from the serial port (HyperTerminal) to PIC32 that will turn on of the LEDs (1-4) on. Then, the text from the serial port will be echoed back to the HyperTerminal.

- Re-do the same steps of the previous part to create a new Harmony Project, initialize and configure USART.
- Modify the source code of the created project to perform the following tasks:
  - Wait for line of text from the serial port
  - Echo the string to the HyperTerminal
  - Turn on LEDs based on the received text. For example, if text is equal to message "LED1", then turn on LED1 and send an acknowledgment message to PC "LED1 is on". Do the same for LED2, LED3 and LED4. If any other message is received, then turn off all LEDs and send the message "LEDs are OFF" to PC.

Figure 13 describes the flowchart of this code.

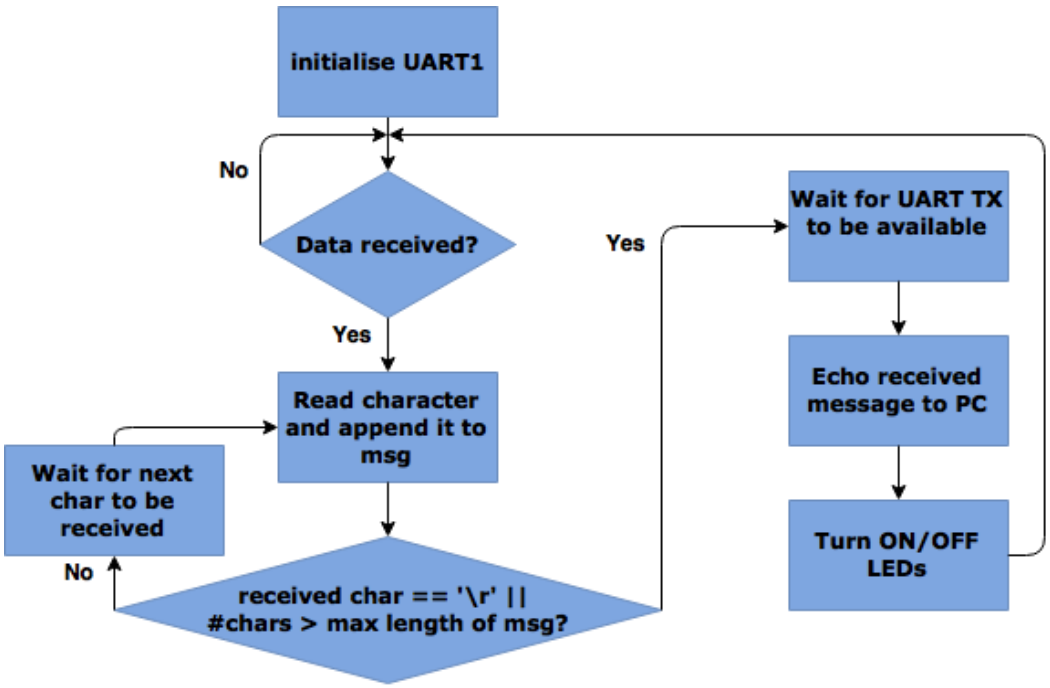


Figure 13: Flowchart of UART code.