Computer Engineering Department

Course Name: Microcontroller Lab **Number: 66496**

Lab Report Grading Sheet

Instructor: **Dr. Abdulla Rashed.**
Academic Year: **2022/2023.**
Semester: **second semester.**

Experiment Number: **3.**
Experiment Name: **Universal Asynchronous Receiver Transmitter**

Students	
1- Fanan yamak	**2- Shahid Bilal.**
Performed on: **21/2/2023.**	Submitted on: **28/2/2023.**

Report's Outcomes				
ILO ___=( ) %	ILO ___=( ) %	ILO ___=( ) %	ILO =( ) %	ILO =( ) %

Evaluation Criterion	Grade	Points
Abstract: a very short summary (around 150-250 words) of what the experiment is about, what you found, and why it may be important.	**1**	
Introduction: Sufficient, clear and complete statement of objectives. In addition to Presents sufficiently the theoretical basis.	**1**	
Materials and Methods: Technical details about the experiment and how it was performed.	**2**	
Experimental Results (including code and figures): provide detailed explanation of the experimental results; outline all of your findings while commenting your code and describing your figures.	**2**	
Discussion: show off your understanding of the results and the data	**2**	
Conclusions and Recommendations: Conclusions summarize the major findings from the experimental results with adequate specificity. Recommendations appropriate in light of conclusions.	**1**	
Appearance: Title page is complete, page are numbered, content is well organized, correct spelling, fonts are consistent, good visual appeal.	**1**	
Total	**10**	

Table of Contents

Abstract 3

1. Introduction 3

2. Materials and Methods 5

2.1. Materials 5

2.2. Methods 5

3. Experimental Results & Discussion 6

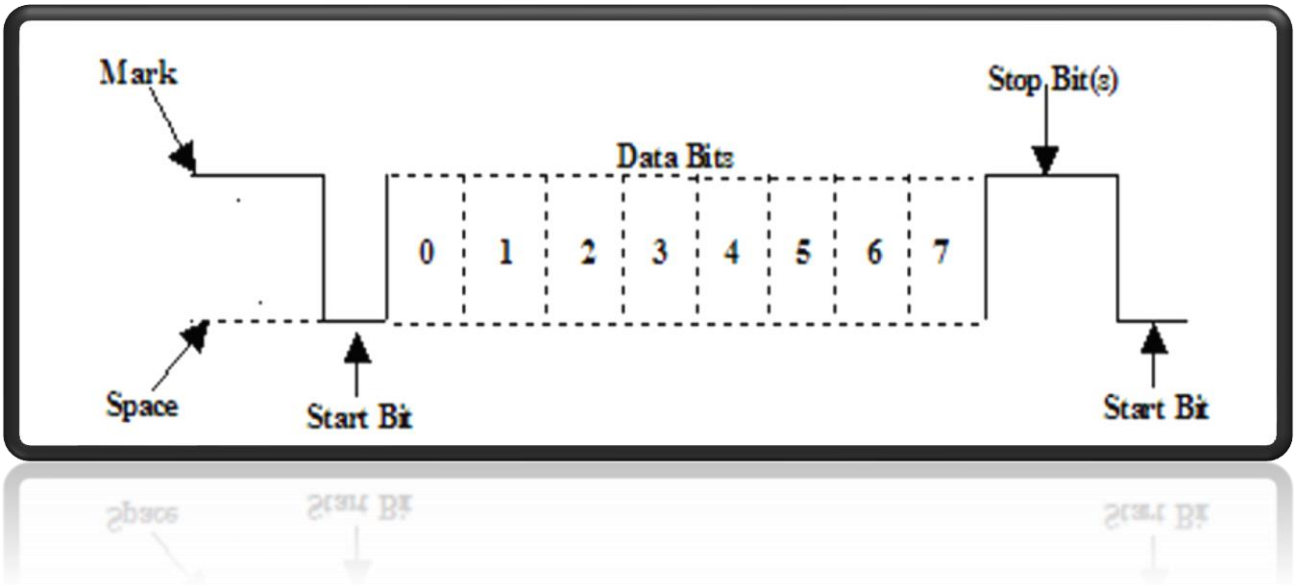
4. Conclusion 11

Abstract

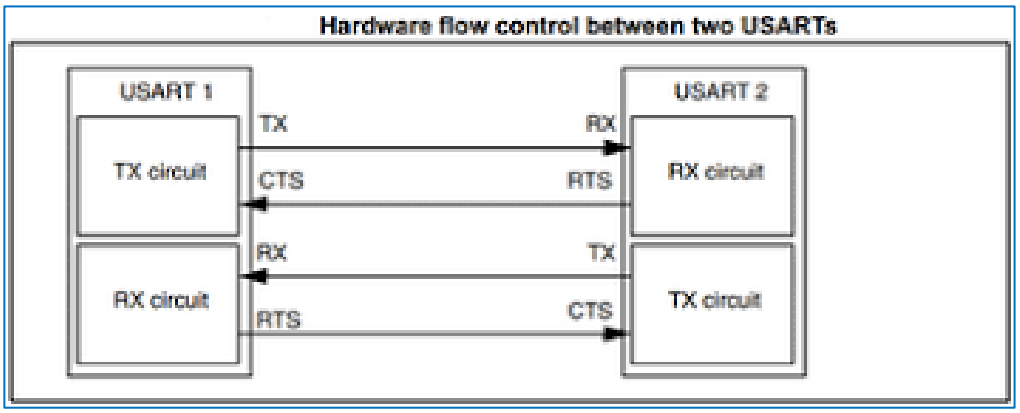
In this experiment we learn to Enable UART communication with PIC Microcontroller and how to transfer data to and from your Computer, using a terminal emulation program to implement a point-to point-serial link between them

1. Introduction

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) is the most common method of serial communication between two devices which is carried out using two lines Tx (transmission line) and Rx (reception line). And supports the full duplex mode
It is a computer hardware for serial communication based on communication protocols like RS 232, in which the data format and transmission speeds are configurable. It sends data bits one by one, from the least significant to the most significant, framed by start and stop bits so that precise timing is handled by the communication channel
It is common to define communications speed as bits per second. which is defined as the inverse of the period of a unit symbol, where the sender and receiver use the same rate , in our experiment it will be 9600



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. 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**



2. Materials and Methods

2.1. Materials

* ChipKITTM Pro MX7 processor board with USB cable.
* Microchip MPLAB ® X IDE
* MPLAB ® XC32++ Compiler
* MPLAB Harmony Framework
* HyperTerminal.

2.2. Methods

As a start we prepared the work environment and defined the inputs and the outputs (where LEDs and TX are the output and buttons and RX are the inputs), now we need to initialize and configure USART by using the MPLAB Harmony Configurator (MHC)

from the **Option-tab** choose **Drivers** then, **USART** and set the Driver Implementation as **STATIC**, **remove** the tick from **Interrupt Mode** and set **Number of USART Driver Instances** to **1**.

we used the USART Driver **default options** ,. and needed to configure the PC-based terminal program to use these same settings , which are :

- **Baud Rate: 9600** -**Data**: **8-bit** -**Parity**: **none**
-**Number of Stop bits**: **1** - **Flow control**: **none**

After that we were ready to start writing the code. Which have two parts :

3. Experimental Results & Discussion

3.1. Receive/echo byte data

In this part, we were 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.

In order to do that we defined (in the app.h)two states as we were either sending or receiving

```
typedef enum
{
    /* Application's state machine's initial state. */
    APP_STATE_INIT=0,
    APP_STATE_SERVICE_TASKS,
            sender,
            rec,
    /* TODO: Define states used by the application state machine. */

} APP_STATES;
```

.

then we defined the variable that was used for communication

```
typedef struct
{
    /* The application's current state */
    APP_STATES state;
    char c;
    /* TODO: Define any additional data used by the application. */

} APP_DATA;
```

in the app. c file

initially app state machine will be at receive state waiting until the receiving buffer contain a value , this value will be read using DRV USART0 ReadByte() Function which defined in the Harmony library as well as other DRV_USART functions that we used

after the value is stored in the previously defined variable (c) the machine will move to sender state where the variable will be incremented and sending back to PC using library's writing function as shown

```
void APP_Tasks ( void )
{

    /* Check the application's current state. */
    switch ( appData.state )
    {
        /* Application's initial state. */
        case APP_STATE_INIT:
        {
            bool appInitialized = true;


            if (appInitialized)
            {

                appData.state = rec;
            }
            break;
        }

        case rec:
        {
            if(!DRV_USART0_ReceiverBufferIsEmpty()){
                 appData.c=DRV_USART0_ReadByte();
                   appData.state = sender;
            }

            break;
        }
        case sender:
        {
            appData.c=appData.c+1;
        DRV_USART0_WriteByte(appData.c);
        appData.state = rec;
            break;
        }
        /* TODO: implement your application state machine.*/


        /* The default state should never be executed. */
        default:
        {
            /* TODO: Handle error in application's state machine. */
```

3.2. Receive/echo String of data

Now as a second part instead of byte we were asked to receive a whole string of data that represent set of commands to turn on corresponding LED for each command and send back a statement as a response to that command

To implement that we used the same states in part 1 but with extra variables that include some counters and the array of characters to store the string

```
typedef struct
{
    /* The application's current state */
    APP_STATES state;
    char c ;
    char s[30];

    int i;
     int j;
      int m;
       int k;

    /* TODO: Define any additional data used by the application. */

} APP_DATA;
```

In the app.c file , at first we needed to define and initialize the variables and string which were used in the communication process , as PORTG which its valued displayed on LEDs as explained before , initially all LEDs are OFF

```
void APP_Tasks ( void )
{

char l1[]="led1 is on \r\n";
   char l2[]="led2 is on \r\n";
   char l3[]="wrong command \r\n";
   /* Check the application's current state. */
   switch ( appData.state )
   {
       /* Application's initial state. */
       case APP_STATE_INIT:
       {
           bool appInitialized = true;


           if (appInitialized)
           {

               PORTG=0x0000;
               appData.i=0;

               appData.state = rec;
           }
           break;
       }

//       case APP_STATE_SERVICE_TASKS:
//       {
//
//           break;
//       }
```

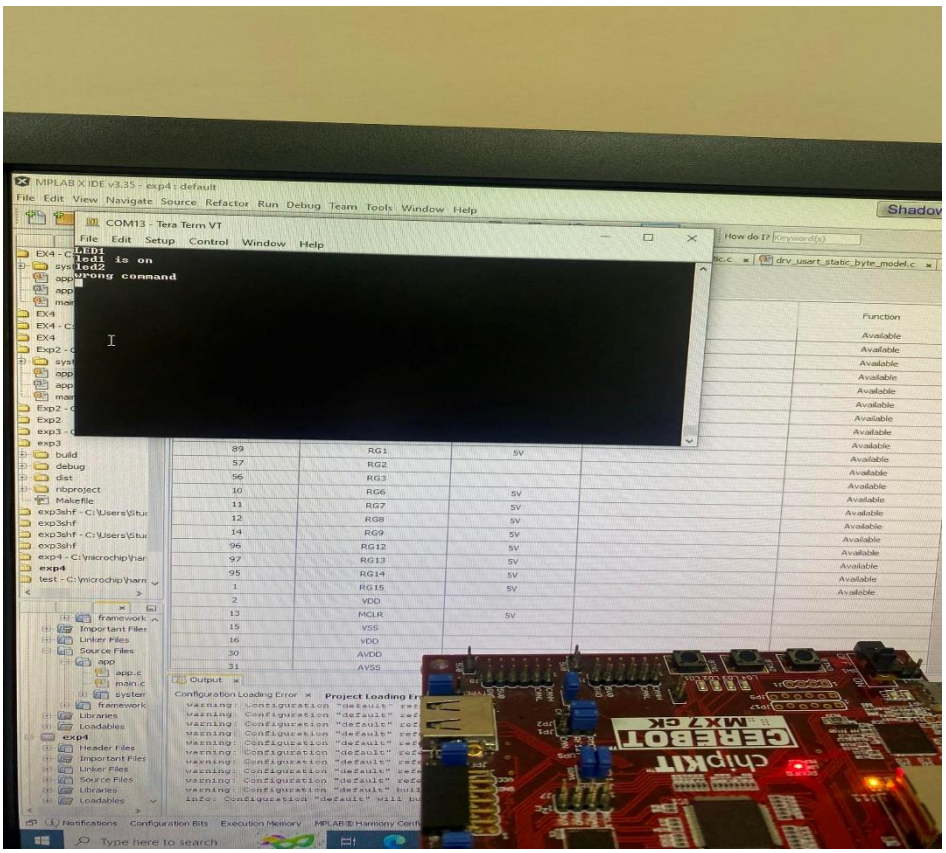
Also here the machine will be initially in receiving state

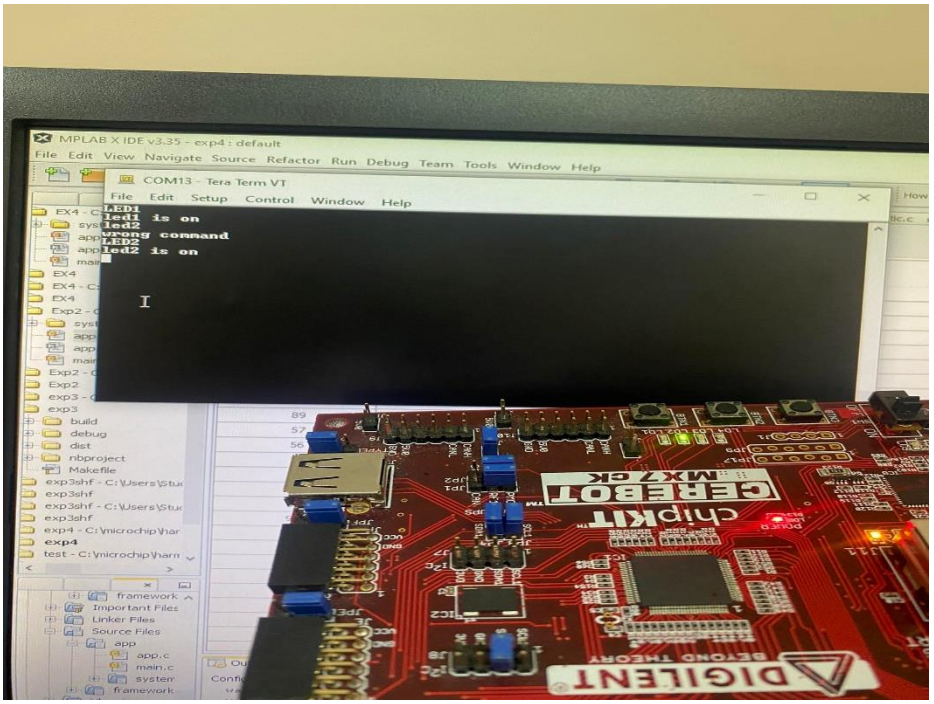
```
case rec:
{

   if(!DRV_USART0_ReceiverBufferIsEmpty()){
        appData.c=DRV_USART0_ReadByte();
        // appData.state = sender;
        if (appData.c == '\n' || appData.c == '\r'){
            appData.s[appData.i]='\0';
            appData.i=0;
            appData.j=0;
       appData.k=0;
       appData.m=0;
            appData.state = sender;
        }
        else {
            appData.s[appData.i]=appData.c;
            appData.i++;
        }

         //
   }

   break;
}
```

In receiving state as before it will be waiting for receiver buffer to contain a value ,using pre-defined functions , and for each read byte it will be stored in the character array until new line is read it will stop reading and move to sender state

```
case sender:
{

 if(!strcmp(appData.s,"LED1"))
 {
    for(;appData.j<12;appData.j++){
      DRV_USART0_WriteByte(l1[appData.j]);
    }
    PORTG= 0x1000;
    appData.state = rec;

 }
 else if(!strcmp(appData.s,"LED2"))
 {

      for(;appData.k<12;appData.k++){
          DRV_USART0_WriteByte(l2[appData.k]);
      }
    PORTG= 0x2000;
    appData.state = rec;

 }


 else{

      for(;appData.m<15;appData.m++){
          DRV_USART0_WriteByte(l3[appData.m]);
      }
    PORTG= 0x0000;


    appData.state = rec;
 }
     DRV_USART0_WriteByte(appData.c);
     appData.state = rec;
```

In the sender state the string stored in the array will be compared with some command , and Turn on the corresponding LED by set the proper bit in PORTG , also a loop will be used to send the response to PC byte by byte using USART write Byte function , if the command not defined "wrong command" will be sent and all LEDs will be OFF


COM13 - Tera Term VT
File Edit Setup Control Window Help
LED1
led1 is on
led2
wrong command
MPLAB X IDE v3.35 - exp4 : default
chipKIT
CEREBOT
MX7 CK


COM13 - Tera Term VT
File Edit Setup Control Window Help
LED1
led1 is on
led2
wrong command
LED2
led2 is on
MPLAB X IDE v3.35 - exp4 : default
chipKIT
CEREBOT
MX7 CK
DIGILENT

4. Conclusion

By the end of this experiment we were able to use the USART built in the ChipKit and initialize and configure it to receive and transmit data between PC and PIC32, starting with single byte to whole string and commands , also to perform actions in response to these commands