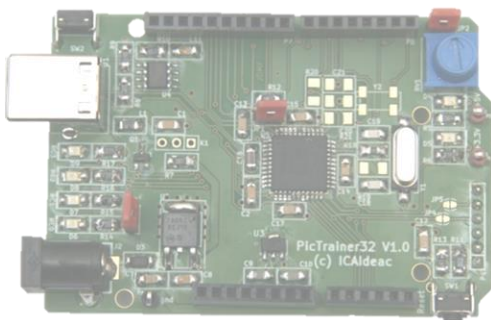


Práctica 6 y 7:10 Perfecto
CP:10
TC:10

Microprocesadores

Práctica 6&7: Comunicaciones serie

Francisco Martín Martínez



Jorge Calvar, Fernando Santana
9-4-2021

Contenido

Introducción	2
Comunicaciones serie I – Práctica 6	2
3. Recepción de caracteres mediante la UART1	2
4. Transmisión de caracteres mediante la UART1	3
5. Transmisión y recepción de caracteres	4
Driver UART.....	7
6. Eco (opcional).....	9
Comunicaciones serie II – Práctica 7	11
Driver UART con colas.....	11
2. Módulo para gestionar la UART1 con colas e interrupciones.....	13
3. Bluetooth bit Whacker.....	15
executeCommand(char* s).....	19
Conclusión	20

Introducción


En esta práctica nos familiarizaremos con las comunicaciones serie usando un módulo Bluetooth para comunicarnos con un ordenador o con un teléfono móvil mediante la UART del PIC32. Después utilizaremos colas e interrupciones para finalmente crear un programa que nos permita controlar los pines del microprocesador remotamente.

Comunicaciones serie I – Práctica 6

Primero conectamos el módulo bluetooth a nuestro dispositivo Android y utilizando la aplicación Bluetooth Terminal HC-05 probamos el correcto funcionamiento. Para ello realizamos un puente entre RX y TX en la tarjeta y vemos que lo que escribimos en el terminal aparece en pantalla. Probamos también el modulo bluetooth conectado al ordenador utilizando el programa PuTTY para asegurarnos de que sabemos configurarlo en ambos dispositivos.

3. Recepción de caracteres mediante la UART1

En esta sección diseñamos un programa que recibe un carácter de la UART y utilizando los LEDs de la placa y representa los cuatro bits menos significativos. Utilizaremos polling, el formato 8N1 a 9600 baudios, conectaremos U1RX al pin RB13 y el receptor se mantendrá habilitado.



```
/*
 * File:    main4.c
 * Author:  Fernando & Jorge
 *
 * Created on 26 March 2021, 10:23
 */
#include <xc.h>
#include "Pic32Ini.h"

#define PIN_RX 13

int main(void) {

    // Establecemos RB13 como digital y entrada
    ANSELB  &= ~(1<<PIN_RX);

    TRISA = 0;
    TRISB = 1<<PIN_RX;
    TRISC = 0;

    LATA = 0;
    LATB = 0;
    LATC = 0xF; // Leds apagados


    // Conectamos U1RX con RB13
    SYSKEY = 0xAA996655;
    SYSKEY = 0x556699AA;
    U1RXR = 3;
    SYSKEY = 0x0;

    U1BRG = 32;           // 9600 baudios

    U1STAbits.URXEN = 1;  // Habilitamos el receptor
    U1MODE = 0x8000;      // Activamos la UART, formato 8N1

    InicializarReloj();

    char c;
```



```
while(1) {

    while(U1STAbits.URXDA == 0) ;    // Esperamos la recepción
    c = U1RXREG;    // Leemos el caracter


    LATCSET = 0xF;    // Apagamos los LEDs
    LATCLR = (c & 0xF);    // Encendemos los correspondientes
}

return 0;
}
```

Mientras `U1STAbits.URXDA == 0` no hay ningún carácter disponible en la FIFO y por tanto paramos el programa utilizando el `while`; hasta que llega un carácter y se lee del registro `U1RXREG`. Si quisiéramos hacer más cosas en el programa aun cuando no llega un carácter sería conveniente usar `if (U1STAbits.URXDA != 0)` para que el programa no se detenga.

4. Transmisión de caracteres mediante la UART1

Una vez realizado el emisor creamos un transmisor que envíe una cadena de caracteres cada vez que se pulse un pulsador. De nuevo en modo polling con formato 8N1, 9600 baudios, U1TX (pin de transmisión de la UART1) ahora conectado al pin RB7. Esta vez el transmisor se habilitará cuando se vaya a enviar la cadena y se inhabilitará cuando se termine de enviar como vemos en la línea `U1STAbits.UTXEN = 1` y `U1STAbits.UTXEN = 0`.



```
/*
 * File:    main4.c
 * Author:  Fernando & Jorge
 *
 * Created on 26 March 2021, 10:32
 */

#include <xc.h>
#include "Pic32Ini.h"

#define PIN_TX 7
#define PIN_PULSADOR 5

int main(void) {

    // Establecemos los pines como digitales
    ANSELB &= ~(1<<PIN_TX | 1<<PIN_PULSADOR);

    // Todo salidas, excepto el pulsador
    TRISA = 0;
    TRISB = 1<<PIN_PULSADOR;
    TRISC = 0;

    LATA = 0;
    LATB = 0;
    LATC = 0xF; // Leds apagados

    // Conectamos U1TX con RB7
    SYSKEY = 0xAA996655;
    SYSKEY = 0x556699AA;
    RPB7R = 1;
    SYSKEY = 0x1CA11CA1;
}
```



```

U1BRG = 32;

U1MODE = 0x8000;    // Activamos la UART, formato 8N1

InicializarReloj();

int puls_act, puls_ant = (PORTB >> PIN_PULSADOR) & 1;

char cadena[] = "Hola Mundo\r\n";
int i;
char c;

while(1) {

    puls_act = (PORTB >> PIN_PULSADOR) & 1;

    if (puls_act < puls_ant) { // Flanco de subida

        U1STABits.UTXEN = 1;    // Habilitamos el transmisor

        i = 0;
        c = cadena[i];
        while(c != '\0') { // Comprobamos que la cadena no se ha acabado

            U1TXREG = c;
            while (U1STABits.TXMT == 0) ;

            i++;
            c = cadena[i];
        }

        U1STABits.UTXEN = 0;    // Deshabilitamos el transmisor

    }

    puls_ant = puls_act;

}

return 0;
}

```

Tras comprobar que la cadena no se ha acabado viendo si hemos llegado al '\0', esperamos al envío del carácter. Avanzamos posición en la cadena de caracteres y repetimos hasta llegar a '\0' que indica el final donde deshabilitamos el transmisor.

5. Transmisión y recepción de caracteres


Ahora procedemos a combinar los dos programas anteriores.

```

/*
 * File:    main5.c
 * Author: Fernando & Jorge
 *
 * Created on 26 March 2021, 16:57
 */

#include <xc.h>
#include "Pic32Ini.h"

```



```
#define PIN_RX 13
#define PIN_TX 7
#define PIN_PULSADOR 5

int main(void) {

    // Establecemos como digitales los pines correspondientes
    ANSELB &= ~(1<<PIN_RX | 1<<PIN_TX | 1<<PIN_PULSADOR);

    TRISA = 0;
    TRISB = 1<<PIN_RX | 1<<PIN_PULSADOR;
    TRISC = 0;

    LATA = 0;
    LATB = 0;
    LATC = 0xF; // Leds apagados

    // Conectamos U1RX con RB13
    SYSKEY = 0xAA996655;
    SYSKEY = 0x556699AA;
    U1RXR = 3;
    RPB7R = 1;
    SYSKEY = 0x0;

    U1BRG = 32;          // 9600 baudios

    U1STAbits.URXEN = 1; // Habilitamos el receptor
    U1MODE = 0x8000;     // Activamos la UART, formato 8N1

    InicializarReloj();

    int puls_act, puls_ant = (PORTB >> PIN_PULSADOR) & 1;

    char cadena[] = "Hola Mundo\r\n";
    int i;
    char c;

    while(1) {

        // RECEPCIÓN

        if(U1STAbits.URXDA == 1) { // Vemos si hay recepción

            c = U1RXREG; // Leemos el caracter

            LATC |= 0xF; // Apagamos los LEDs
            LATC &= ~(c & 0xF); // Encendemos los correspondientes

        }

        // TRANSMISIÓN

        puls_act = (PORTB >> PIN_PULSADOR) & 1;

        if (puls_act < puls_ant) { // Flanco de subida

            U1STAbits.UTXEN = 1; // Habilitamos el transmisor
```

```
i = 0;
c = cadena[i];
while(c != '\0') {

    U1TXREG = c;
    while (U1STAbits.TRMT == 0) ;

    i++;
    c = cadena[i];
}

U1STAbits.UTXEN = 0;    // Deshabilitamos el transmisor
}

puls_ant = puls_act;

}

return 0;
}
```

Para un mejor diseño se ha dividido el programa utilizando un driver para la UART que se incluye a continuación

```
/*
 * File:    main5b.c
 * Author:  Fernando & Jorge
 * Created on 26 March 2021, 12:48
 */
#include <xc.h>
#include "Pic32Ini.h"
#include "driver_uart.h"

#define PIN_RX 13
#define PIN_TX 7
#define PIN_PULSADOR 5


int main(void) {

    // Configuramos como digitales
    ANSELB &= ~(1<<PIN_RX | 1<<PIN_TX | 1<<PIN_PULSADOR);

    // El pulsador y el rx son entradas
    TRISA = 0;
    TRISB = 1<<PIN_PULSADOR | 1<<PIN_RX;
    TRISC = 0;

    // LEDs apagados
    LATA = 0;
    LATB = 1<<PIN_TX;    // Transmisor inhabilitado
    LATC = 0xF;

    // Conectamos U1RX y U1TX
    SYSKEY = 0xAA996655;
    SYSKEY = 0x556699AA;
    U1RXR = 3;
    RPB7R = 1;
}
```



```

SYSKEY = 0x0;

// Inicializar UART
InicializarUART();

InicializarReloj();

// Iniciamos interrupciones
INTCONbits.MVEC = 1;

char c;
char cadena[] = "Hola Mundo!";
int puls_act, puls_ant = (PORTB >> PIN_PULSADOR) & 1;

while(1) {
    puls_act = (PORTB >> PIN_PULSADOR) & 1;

    if (puls_act < puls_ant) { // Flanco de bajada
        putsUART(cadena);
    }

    c = getcUART();
    if(c != '\0') {
        LATC |= 0xF; // Apagamos los LEDs
        LATC &= ~(c & 0xF); // Encendemos los correspondientes
    }
    puls_ant = puls_act;
}
return 0;
}

```

Utilizamos `putsUART (cadena)` para el envío de un string `getcUART ()` y para la recepción de un caracter.

Driver UART

```

/*
 * File:    driver_uart.h
 * Author:  Fernando & Jorge
 *
 * Created on 26 March 2021, 13:19
 */

#ifndef DRIVER_UART_H
#define DRIVER_UART_H

void InicializarUART();
char getcUART();
void putsUART(char *ps);

#endif /* DRIVER_UART_H */

```

```

/*
 * File:    driver_uart.c
 * Author:  Fernando & Jorge
 *
 * Created on 26 March 2021, 13:19
 */

#include <xc.h>
#include "driver_uart.h"

```



```
#define TAM_CAD 100

static char c_rx = '\0';
static int i_c_tx = 0;
static char c_tx[TAM_CAD];

void InicializarUART() {
    // Interrupciones
    IFS1bits.U1RXIF = 0;
    IEC1bits.U1RXIE = 1;
    IFS1bits.U1TXIF = 0;

    IPC8bits.U1IP = 3;      // Prioridad 3
    IPC8bits.U1IS = 1;      // Subprioridad 1

    // Inicializar UART
    U1BRG = 32;
    U1STAbits.URXISEL = 0;  // 1 char interrumpe
    U1STAbits.UTXISEL = 2;  // Se interrumpe al estar vac 
    U1STAbits.URXEN = 1;
    U1STAbits.UTXEN = 1;
    U1MODE = 0x8000;        // Activamos la UART
}

__attribute__((vector(32), interrupt(IPL3SOFT), nomips16))
void InterrupcionUART1(void) {

    if(IFS1bits.U1RXIF == 1) {
        c_rx = U1RXREG;
        IFS1bits.U1RXIF = 0;
    }

    if(IFS1bits.U1TXIF == 1) {
        if(c_tx[i_c_tx] == '\0') {
            IEC1bits.U1TXIE = 0;
            i_c_tx = 0;
            c_tx[i_c_tx] = '\0'; //No volver a enviar cadena cndo interrumpa recep
        } else {
            U1TXREG = c_tx[i_c_tx];
            i_c_tx++;
        }
        IFS1bits.U1RXIF = 0;
    }
}

char getcUART() {
    char c_ret = c_rx;
    c_rx = '\0';
    return c_ret;
}

void putsUART(char *ps) {
    char *pc;
    pc = c_tx;
    while(*ps != '\0') {
        *pc = *ps;
        pc++;
        ps++;
    }
}
```

```
*pc = *ps; // Hay que añadir \0 al final de la cadena, para el transmisor
IEC1bits.U1TXIE = 1;
}
```

6. Eco (opcional)

Finalmente creamos un programa que envíe por la UART1 los caracteres que reciba para hacer un eco por software.

```
/*
 * File:    main6.c
 * Author:  Fernando & Jorge
 *
 * Created on 26 March 2021, 18:48
 */

#include <xc.h>
#include "Pic32Ini.h"

#define PIN_RX 13
#define PIN_TX 7

int main(void) {

    // Establecemos como digitales los pines correspondientes
    ANSELB &= ~(1<<PIN_RX | 1<<PIN_TX);

    TRISA = 0;
    TRISB = 1<<PIN_RX;
    TRISC = 0;

    LATA = 0;
    LATB = 0;
    LATC = 0xF; // Leds apagados

    // Conectamos U1RX con RB13
    SYSKEY = 0xAA996655;
    SYSKEY = 0x556699AA;
    U1RXR = 3;
    RPB7R = 1;
    SYSKEY = 0x0;

    U1BRG = 32; // 9600 baudios

    U1STAbits.URXEN = 1; // Habilitamos el receptor
    U1MODE = 0x8000; // Activamos la UART, formato 8N1

    InicializarReloj();

    char c;

    while(1) {

        // RECEPCIÓN


        if(U1STAbits.URXDA == 1) { // Vemos si hay recepción
            c = U1RXREG; // Leemos el caracter


            // TRANSMISION
            U1STAbits.UTXEN = 1; // Habilitamos el transmisor
            U1TXREG = c;
        }
    }
}
```

```
    while (U1STAbits.TRMT == 0) ;  
    U1STAbits.UTXEN = 0;    // Deshabilitamos el transmisor  
  
    }  
}  
return 0;  
}
```



Para hacer un eco por software simplemente utilizamos el transmisor y receptor ya programados de manera conjunta: leemos el carácter del teclado y lo enviamos por la UART. Lo comprobamos utilizando PuTTY.

 COM9 - PuTTY

Hola, estamos probando el main6 que hace eco 

Comunicaciones serie II – Práctica 7

En esta segunda parte de comunicaciones serie vamos a utilizar un buffer circular que será probado con un eco y finalmente se empleará en un programa más complejo.

Driver UART con colas

```
/*
 * File:    driver_uart.h
 * Author:  Fernando & Jorge
 *
 * Created on 26 March 2021, 13:19
 */


#ifndef DRIVER_UART_H
#define DRIVER_UART_H

void InicializarUART1(int baudios);
char getcUART();
void putsUART(char *ps);

#endif /* DRIVER_UART_H */
```

Primero creamos una cabecera con las funciones para inicializar, recibir caracteres y enviar cadenas.

Como vemos a continuación, vamos a definir una estructura de datos para utilizar las colas.



```
/*
 * File:    driver_uart.c
 * Author:  Fernando & Jorge
 *
 * Created on  March 2021, 13:19
 */

#include <xc.h>
#include "driver_uart.h"

#define TAM_COLA 100

typedef struct {
    char array[TAM_COLA];
    int cab;
    int col;
} cola_t;

static cola_t rx;
static cola_t tx;

void InicializarUART1(int baudios) {

    rx.cab = 0;
    rx.col = 0;
    tx.cab = 0;
    tx.col = 0;

    // Interrupciones
    IFS1bits.U1RXIF = 0;
    IEC1bits.U1RXIE = 1;
    IFS1bits.U1TXIF = 0;

    IPC8bits.U1IP = 3; // Prioridad 3
```



```

IPC8bits.U1IS = 1; // Subprioridad 1

U1MODE = 0;
// Velocidad
double brg;
if (baudios > 38400) {
    U1MODEbits.BRGH = 1;
    brg = 5000000 / (4 * baudios) - 1;
} else {
    U1MODEbits.BRGH = 0;
    brg = 5000000 / (16 * baudios) - 1;
}
U1BRG = (int) brg;

// Inicializar UART

U1STAbits.URXISEL = 0; // 1 char interrumpe
U1STAbits.UTXISEL = 2; // Se interrumpe al estar vacío
U1STAbits.URXEN = 1;
U1STAbits.UTXEN = 1;
U1MODEbits.ON = 1; // Activamos la UART
}

__attribute__((vector(32), interrupt(IPL3SOFT), nomips16))
void InterrupcionUART1(void) {

    if (IFS1bits.U1RXIF == 1) {
        if ((rx.cab + 1 == rx.col) ||
            (rx.cab + 1 == TAM_COLA && rx.col == 0)) {
            // La cola está llena
        } else {
            rx.array[rx.cab] = U1RXREG;
            rx.cab++;
            if (rx.cab == TAM_COLA) {
                rx.cab = 0;
            }
        }
        IFS1bits.U1RXIF = 0;
    }

    if (IFS1bits.U1TXIF == 1) {
        // Se extrae un carácter de la cola y se env
        if (tx.col != tx.cab) { // Hay datos nuevos
            U1TXREG = tx.array[tx.col];
            tx.col++;
            if (tx.col == TAM_COLA) {
                tx.col = 0;
            }
        } else { // Se ha vaciado la cola.
            IEC1bits.U1TXIE = 0; // Para evitar bucle sin fin
        }
        IFS1bits.U1RXIF = 0;
    }
}

char getcUART() {
    char c;
    if (rx.col != rx.cab) { // Hay datos nuevos
        c = rx.array[rx.col];
        rx.col++;
        if (rx.col == TAM_COLA) {

```

```

        rx.col = 0;
    }
    } else { // no ha llegado nada
        c = '\0';
    }
    return c;
}

void putsUART(char *ps) {
    while (*ps != '\0') {
        if ((tx.cab + 1 == tx.col) ||
            (tx.cab + 1 == TAM_COLA && tx.col == 0)) {
            // La cola está llena.
            break;
        } else {
            tx.array[tx.cab] = *ps; // Copia el carácter en la cola
            ps++; // Apunto al siguiente carácter de la cadena
            tx.cab++;
            if (tx.cab == TAM_COLA) {
                tx.cab = 0;
            }
        }
    }
    // Se habilitan las interrupciones del transmisor para
    // comenzar a enviar
    IEC1bits.U1TXIE = 1;
}

```

En la primera función inicializar UART permitimos elegir la velocidad de la línea serie. Para ello hemos tenido en cuenta que para velocidades superiores a 38 400 baudios debemos usar el divisor por 4 en lugar del divisor por 16 para evitar un error elevado. Teniendo eso en cuenta seleccionamos el valor de U1BRG con la fórmula $5000000 / (\text{div} * \text{baudios}) - 1$.

Tanto la rutina de interrupción como las funciones `getcUART()` y `putsUART(char *ps)` han sido vistas y explicadas en clase.

Normalmente los sistemas de comunicaciones se encuentran haciendo otra cosa y sin hacer nada cuando de repente llega un mensaje. Si el programa está ocupado y la velocidad es alta es posible que se pierdan caracteres. Las colas solucionan este problema permitiendo al programa principal leer los caracteres cuando tenga tiempo desde la cola y procesarlos mientras esta sea suficientemente grande para almacenarlos mientras el microprocesador está ocupado.

2. Módulo para gestionar la UART1 con colas e interrupciones


```

/*
 * File:    main2.c
 * Author:  Fernando & Jorge
 *
 * Created on 8 April 2021, 12:48
 */

#include <xc.h>
#include "Pic32Ini.h"
#include "driver_uart.h"

#define PIN_RX 13
#define PIN_TX 7

```



```
int main(void) {

    // Configuramos como digitales
    ANSELB &= ~(1<<PIN_RX | 1<<PIN_TX);

    // El pulsador y el rx son entradas
    TRISA = 0;
    TRISB = 1<<PIN_RX;
    TRISC = 0;

    // LEDs apagados
    LATA = 0;
    LATB = 1<<PIN_TX;    // Transmisor inhabilitado
    LATC = 0xF;

    // Conectamos U1RX y U1TX
    SYSKEY = 0xAA996655;
    SYSKEY = 0x556699AA;
    U1RXR = 3;
    RPB7R = 1;
    SYSKEY = 0x0;

    // Inicializar UART
    InicializarUART1(9600);

    InicializarReloj();

    // Iniciamos interrupciones
    INTCONbits.MVEC = 1;
    asm(" ei");


    char c;
    char s[2];
    s[1] = '\0';


    while(1) {

        c =getcUART();
        if (c != '\0') {
            s[0] = c;
            putsUART(s);
        }
    }
    return 0;
}
```

Para probar el driver de la UART con el buffer circular vamos a crear un programa que haga eco.

Para utilizar `putsUART(s)` que está diseñada para enviar cadenas y poder enviar caracteres, simplemente creamos una cadena de tamaño 2 con el carácter deseado y `'\0'`. Por otro lado, `getcUART()` ya está diseñada para recibir un carácter. Comprobamos utilizando PuTTY que se recibe el carácter del teclado y lo envía por la UART.

 COM9 - PuTTY

Probamos el driverUART con un main que hace eco 

3. Bluetooth bit Whacker

Finalmente vamos a crear un programa para poder controlar los pines de la tarjeta desde el PC o el móvil.

A continuación, se presentan los comandos a utilizar.

PD (de **Pin Direction**). Selecciona la dirección de un pin.

PD,<puerto>,<pin>,<dirección>\n.

El comando retornará "OK\n" si los parámetros son correctos o "Error\n" si no lo son

PI (de **Pin Input**). Devuelve el estado de un pin.

PI,<puerto>,<pin>\n.

El comando retornará "PI,1\n" si el pin está a 1, "PI,0\n" si el pin está a 0 o "Error\n"

PO (de **Pin Output**). Cambia el estado de un pin.

PO,<puerto>,<pin>,<valor>\n.

El comando retornará "OK\n" si los parámetros son correctos o "Error\n" si no lo son

<puerto> es la letra A, B o C según el puerto al que pertenezca el pin que queremos cambiar.

<pin> es el número del pin que queremos cambiar, expresado en hexadecimal (0 a F).

<dirección> es 0 para salida o 1 para entrada.

<valor> es 1 o 0 según el valor que queramos poner en el pin.

```
/*
 * File:   main3.c
 * Author: jorge
 *
 * Created on 09 April 2021, 10:54
 */

#include <xc.h>
#include "Pic32Ini.h"
#include "driver_uart.h"

#define PIN_RX 13
#define PIN_TX 7

void resultError()
{
    putsUART("Error\n");
}

int convertPin(char s) {
    if (s >= 48 && s <= 57)
        return s - 48;
    else if (s >= 65 && s <= 70)
        return s - 65 + 10;
    return -1;
}
```



```
void executeCommand(char* s) {

    // Comrpobar formato
    if (s[0] != 'P' || s[2] != ',' || s[4] != ',') {
        resultError();
        return;
    }

    ///PD///
    if (s[1] == 'D') {

        // Comrpobar formato
        if (s[6] != ',' || s[8] != '\n') {
            resultError();
            return;
        }

        // Leer valores
        char puerto = s[3];
        int pin = convertPin(s[5]);

        if (pin == -1) {
            resultError();
            return;
        }

        int dir = convertPin(s[7]);

        if (dir != 0 && dir != 1) {
            resultError();
            return;
        }

        // Cambiar modo del pin
        if (puerto == 'A') {
            if (dir == 0)
                TRISACLR = 1 << pin;
            else
                TRISASET = 1 << pin;
        } else if (puerto == 'B') {
            if (dir == 0)
                TRISBCLR = 1 << pin;
            else
                TRISBSET = 1 << pin;
        } else if (puerto == 'C') {
            if (dir == 0)
                TRISCCLR = 1 << pin;
            else
                TRISCSET = 1 << pin;
        } else {
            resultError();
            return;
        }

        putsUART("OK\n");

    }

    ///PI///
    } else if (s[1] == 'I') {

        // Comprobar formato
```

```

if (s[6] != '\n') {
    resultError();
    return;
}

// Leer valores
char puerto = s[3];
int pin = convertPin(s[5]);


if (pin == -1) {
    resultError();
    return;
}

// Leer le pin
int v;
if (puerto == 'A') {
    v = (PORTA >> pin) & 1;
} else if (puerto == 'B') {
    v = (PORTB >> pin) & 1;
} else if (puerto == 'C') {
    v = (PORTC >> pin) & 1;
} else {
    resultError();
    return;
}

char s_ret[5];
s_ret[0] = 'P';
s_ret[1] = 'I';
s_ret[2] = ',';
s_ret[3] = v + 48;
s_ret[4] = '\n';
putsUART(s_ret);

///

```



```

    // Poner el valor adecuado
    if (puerto == 'A') {
        if(valor == 0)
            LATACLR = 1 << pin;
        else
            LATASET = 1 << pin;
    } else if (puerto == 'B') {
        if(valor == 0)
            LATBCLR = 1 << pin;
        else
            LATBSET = 1 << pin;
    } else if (puerto == 'C') {
        if(valor == 0)
            LATCCLR = 1 << pin;
        else
            LATCSET = 1 << pin;
    } else {
        resultError();
        return;
    }

    putsUART("OK\n");

} else {
    resultError();
}
}

int main(void) {

    // Configuramos como digitales
    ANSELB &= ~(1 << PIN_RX | 1 << PIN_TX);

    // El pulsador y el rx son entradas
    TRISA = 0;
    TRISB = 1 << PIN_RX;
    TRISC = 0;

    // LEDs apagados
    LATA = 0;
    LATB = 1 << PIN_TX; // Transmisor inhabilitado
    LATC = 0xF;

    // Conectamos U1RX y U1TX
    SYSKEY = 0xAA996655;
    SYSKEY = 0x556699AA;
    U1RXR = 3;
    RPB7R = 1;
    SYSKEY = 0x0;


    // Inicializar UART
    InicializarUART1(9600);

    InicializarReloj();

    // Iniciamos interrupciones
    INTCONbits.MVEC = 1;
    asm(" ei");

    char c;

```



```
char s[10]; // El comando más largo ocupa 9
int icab_s = 0;

while (1) {

    c = getcUART();
    if (c != '\0') {

        s[icab_s] = c;
        icab_s++;

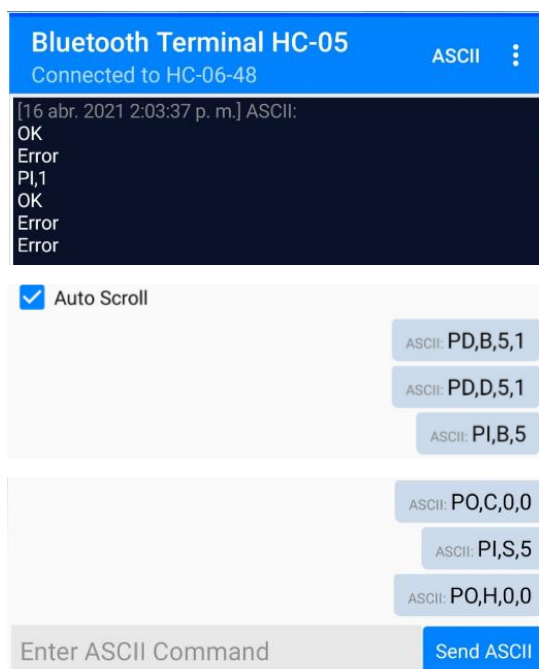
        if (c == '\n') {
            executeCommand(s);
            icab_s = 0;
        }
    }
    return 0;
}
```

Como vemos en el último while, toda la lógica del programa se encuentra en la función `executeCommand(char* s)` que actúa al presionar enter. Para facilitar la conversión ASCII hemos creado también la función `convertPin(char s)` y la función `resultError()` para escribir error.

Consultando la tabla ASCII vemos que los números se encuentran entre las posiciones 48 y 57 de la tabla y las letras a partir de la 65.

`executeCommand(char* s)`

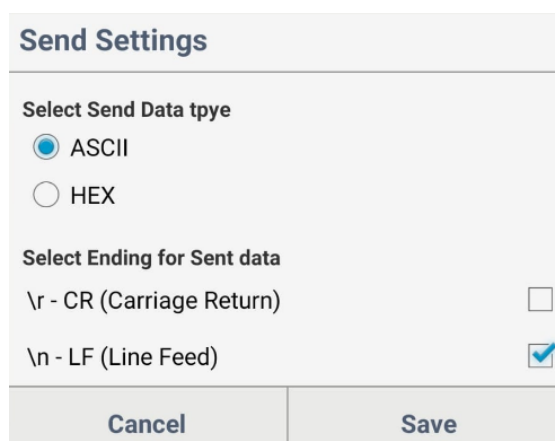
Primero comprobamos genéricamente que el formato coincide. Después utilizando ifs comprobamos el comando introducido. En PD volvemos a comprobar el formato de forma específica, leemos los parámetros y cambiamos el modo del pin. En PO de nuevo comprobamos el formato específicamente, leemos parámetros, leemos el pin y devolvemos el estado en el que se encuentra por la UART. Finalmente, en PO tras comprobar el formato y leer los parámetros, ponemos el valor indicado en el pin indicado. En caso de algún error de formato escribimos error en todos los casos.



Probamos varios comandos.

En clase se probó mas exhaustivamente comprobando el estado de un pulsador, encendiendo Leds etc.

Importante configurarlo en formato ASCII y con \n



Conclusión

En esta práctica hemos aprendido a:

- Configurar el módulo bluetooth HC-06
- Utilizar PuTTY y la aplicación Bluetooth Terminal HC-05 para enviar caracteres y así probar el correcto funcionamiento de los módulos diseñados
- Crear un transmisor y receptor para la UART en modo polling
- Diseñar un driver para utilizar un buffer circular con la UART
- Crear un programa para controlar los pines del microprocesador de forma remota

