

Práctica 5:

Comunicación Hardware – Software:
desde CoOs a una aplicación en .Net

Objetivos

- Ilustrar el manejo de los puertos series asíncronos de un PC desde C#
- Implementar y desarrollar un mecanismo de comunicación de alto nivel.
- Empaquetado de la información:
 - Enviar al PC paquetes con el estado de los sensores:
 - Joystick digital y analógico.
 - Recibir comandos desde el PC s:
 - Mover servo / motor.
 - Lanzar animaciones de Leds.
- Implementar una aplicación en C# usando Windows Form:
 - Mostrar la información recibida a través del puerto serie al usuario.
 - Generar y enviar comandos por el puerto serie al sistema.

Empaquetado de la información

En esta práctica vamos a usar mecanismos de empaquetamiento de la información para la comunicación ya que es lo común en sistemas a un alto nivel.

Off-set	Descripción	Dato	Interpretación
+0	Inicio de trama	0x7E	Header
+1	Longitud de la trama	0x05	
+2	Dato [0]	0x01	A. Joy X MSB
+3	Dato [1]	0x55	A. Joy X LSB
+4	Dato [2]	0x00	A. Joy Y MSB
+5	Dato [3]	0xAA	A. Joy Y LSB
+6	Dato [4]	0x04	Joy Digital
+8	Fin de trama	0x0D	Trailing

Transmitiendo desde el STM32 – Recibiendo en C#

Con la función serialTxTask vamos a leer cada 100ms los valores de los dos ejes del joystick analógico y el digital. Además, vamos a enviar un paquete con la estructura arriba expuesta con los datos.

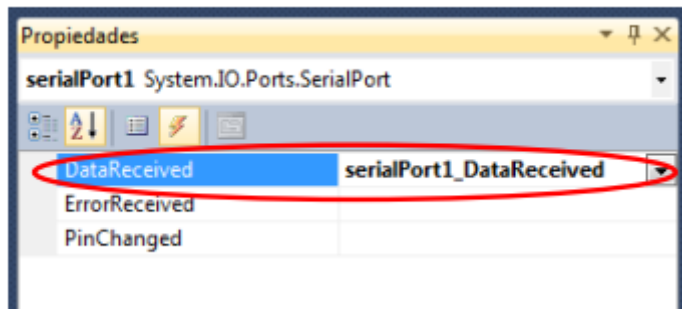
```
void serialTxTask(void * parg){  
    uint16_t x,y;  
    uint8_t b;  
    for(;;){  
        x = getAnalogJoy(0);  
        y = getAnalogJoy(1);  
  
        b = Read_Joy();  
  
        SerialSendByte(0x7E);  
        SerialSendByte(5);  
        SerialSendByte(x>>8);  
        SerialSendByte(x);  
        SerialSendByte(y>>8);  
        SerialSendByte(y);  
        SerialSendByte(b);  
        /*  
        SerialSendByte('H');  
        SerialSendByte('O');  
        SerialSendByte('L');  
        SerialSendByte('A');*/  
        SerialSendByte(0x0D);  
        CoTimeDelay(0,0,0,100); //Espera  
    }  
}
```

Para cerciorarnos de que enviamos correctamente usamos la herramienta X-CTU (vista hexadecimal).

Formulario en el Visual Studio

Creemos un proyecto en Visual C# (Aplicación de Windows forms) en nuestro Visual Studio. En nuestro diseño iremos añadiendo diferentes elementos desde la barra de herramientas como botones o cuadros de texto.

Para empezar, debemos añadir un serialPort para nuestra comunicación. Lo configuraremos a un BaudRate de 115200 y le cambiaremos el PortName al puerto que nos diga el X-CTU. Es importante que le añadamos un evento de datos recibidos.



Añadiremos también un evento FormClosing donde posteriormente cerraremos el puerto serie.

```
public Form1()
{
    InitializeComponent();
    i = 0;
    //Abrir el puerto serie al inicio.
    serialPort1.Open();
}

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    //Cerrar el puerto serie al cerrar el formulario.
    serialPort1.Close();
}
```

Recibiendo datos

Para recibir los datos vamos a implementar una máquina de estados en la función RxByteStateMachine para recibir los datos de cada paquete. El método es invocado byte a byte. Al finalizar la recepción de un paquete deberemos decodificarlo.

```
public int RxStatus, RxLen, RxIdx;
public int [] RxBuff;

public void RxByteStateMachine(int RxByte)
{
    switch(RxStatus){
        case 0:
            if(RxByte == 0x7E){
                RxStatus = 1;
            }
            break;

        case 1:
            RxLen = RxByte;
            RxIdx = 0;
            RxBuff = new int[RxLen];
            RxStatus = 2;
            break;

        case 2:
            RxBuff[RxIdx] = RxByte;
            RxIdx++;
            if(RxIdx == RxLen){
                RxStatus = 3;
            }
            break;

        case 3:
            if (RxByte == 0x0D)
            {
                ParsePacket(RxBuff);
            }
            RxStatus = 0;
            break;


        default:
            break;
    }
}
```

Descodificando la información

Para decodificar el paquete usamos la siguiente función.

```
int x, y, b;
public void ParsePacket(int[] RxBuff)
{
    x = RxBuff[0] * 256 + RxBuff[1];
    y = RxBuff[2] * 256 + RxBuff[3];
    b = RxBuff[4];

    this.Invalidate();
}
```

 void Control.Invalidate() (+ 5 sobrecargas)

Invalida toda la superficie del control y hace que se vuelva a dibujar el control.

Diseño del formulario

Usaremos un evento Paint para repintar el formulario.

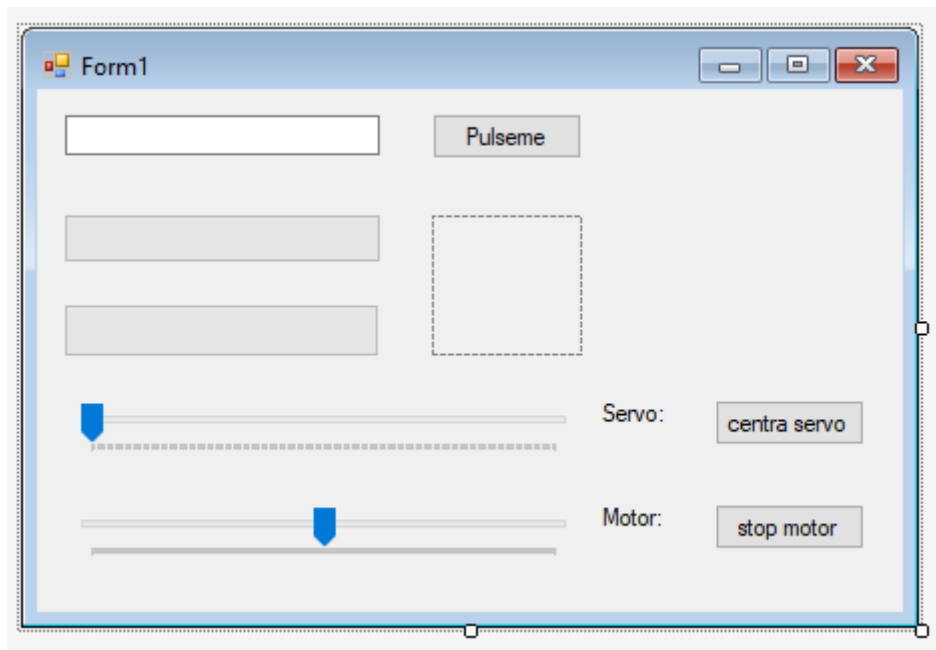
```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    textBox1.Text = "Eje x: " + x + " Eje y: " + y + "Bot: " + b;
    progressBar1.Value = x/16;
    progressBar2.Value = y/16;

    switch(b){
        case 0:
            break;
        case 1:
            panel1.BackColor = Color.Yellow;
            break;
        case 2:
            panel1.BackColor = Color.Green;
            break;
        case 3:
            panel1.BackColor = Color.Orange;
            break;
        case 4:
            panel1.BackColor = Color.Violet;
            break;
        case 5:
            panel1.BackColor = Color.Blue;
            break;
        default:
            panel1.BackColor = Color.White;
            break;
    }
}
```

Nuestro formulario constará de:

- Un cuadro de texto y un botón.
- Dos barras progresivas para pintar la posición de los joysticks analógicos.
- Un panel para representar el valor del joystick digital.
- Una barra para posicionar nuestro servo al gusto.
- Una barra para hacer que nuestro motor gire en el sentido y velocidad que deseemos.
- Dos botones para centrar el servo o parar el motor.

Quedando finalmente así:



Enviando comandos al CoOs

Vamos a enviar comandos al STM32 desde la aplicación para:

- Lanzar animaciones LED.
- Posicionar el servo.
- Dar velocidad y dirección al motor.

Los empaquetaremos en 2 bytes, comando y valor:

Comando	Valor	Descripción
'L'	0 a 3	Lanza la animación de leds indicada.
'S'	0 a 180	Fija la posición del servo.
'M'	-128 a 128	Establece la velocidad y sentido del motor.

Para ello vamos a crear un método llamado TxCmd:


```

public void TxCmd(char cmd, byte value) {
    byte[] buffer = new Byte[5];

    buffer[0] = 0x7E;
    buffer[1] = 2;
    buffer[2] = (byte)cmd;
    buffer[3] = value;
    buffer[4] = 0x0D;

    serialPort1.Write(buffer, 0, 5);
}

```

Para enviar comandos a los leds usaremos el primer botón.

```

private void button1_Click(object sender, EventArgs e)
{
    textBox1.Text = "Boton pulsado!: " + i;
    i++;
    if(i == 4){
        i = 0;
    }

    TxCmd('L', (byte) i);
}

```

Recibiendo datos en el STM32

Modificamos la tarea serialRxTask para que quede así:

```
void serialRxTask(void * parg){
    uint8_t c = 0x00;
    uint8_t RxLen = 0;
    uint8_t RxIndex = 0;
    uint8_t RxBuffer[8];
    StatusType err;

    for(;;){
        //c=CoPendQueueMail(queueRxId,0,&err);
        //LED_Toggle(2);

        c = 0;
        //Esperar recepcion inicio trama
        while(c != 0x7E){
            c = CoPendQueueMail(queueRxId,0,&err);
        }

        //Longitud del paquete
        RxLen = CoPendQueueMail(queueRxId,0,&err);

        //Recepcion de datos
        for(RxIndex = 0; RxIndex < RxLen; RxIndex++){
            RxBuffer[RxIndex] = CoPendQueueMail(queueRxId,0,&err);
        }

        //Fin de transmision

        c = CoPendQueueMail(queueRxId,0,&err);

        if(c == 0x0D){
            parsePacket(RxBuffer);
        }
    }
}
```

Al final de la tarea debemos decodificar el paquete:

```
void parsePacket (uint8_t * buff){

    //Switch/Case
    switch(buff[0]){
        case 'L':
            SetFlagKey(buff[1]);
            break;
        case 'S':
            setServoPos(buff[1]);
            break;
        case 'M':
            setMotorSpeed(buff[1]);
            break;
        default:
            break;
    }
}
```

