COMUNICACIÓN MODO CDC

Por: Julio Fabio De La Cruz G – http://micropinguino.blogspot.com

Uno de los modos de comunicación mas utilizados es el serial asíncrono, sin embargo ya la gran mayoría de equipos de computo portátiles no incluyen el puerto serie. Pinguino permite configurarse como un dispositivo USB a serial asíncrono ofreciendo este modo de comunicación.

EL PUERTO USB - SERIE

Un uso común de la tarjeta **Pinguino** es la de utilizarla de interfaz entre nuestras aplicaciones y el computador, una forma de hacer es configurarla como una adaptador **USB** a serial asíncrono. Esto se hace por que una gran mayoría de entornos de programación disponen de ejemplos y librerías para el tipo de comunicación serial asíncrona **RS232**, para este caso **Pinguino** actúa como una caja negra que nos permite comunicación serial emulando un puerto de este tipo en el computador. Esto se logra con la configuración **CDC** para un dispositivo **USB**.

ESPECIFICACIONES DE CLASES USB

USB no es un protocolo de comunicación, es una especificación para dispositivos que se conecten al computador y que provee diferentes clases según la configuración utilizada.

Entre los microcontroladores las mas comunes son:

HDI

La **HID** (Human Interface Device) se utiliza cuando queremos configurar el microcontrolador como dispositivo de interfaz hombre computador como por ejemplo teclados, ratones, gamepads, etc.

Una de sus características es que no necesita driver pues utiliza los estándar del sistema operativo.

MSD

MSD (Mass Storage Device Class) se utiliza cuando queremos configurar el dispositivo como unidad de almacenamiento de datos como son los discos duros externos, memorias flash , etc.

Una de sus características es que no necesita driver pues utiliza los estándar del sistema operativo.

CDC

CDC (Communications Device Class) Es el que utilizaremos en **Pinguino** para configurarlo como dispositivo de comunicación serial. Con esta configuración el computador nos crea un puerto serie virtual de manera que la comunicación entre el dispositivo y el computador se hará tal como si se tuviese un puerto serie físico.

Este también esta soportado por los drivers estándar del sistema operativo, pero en el caso de windows se recomienda utilizar el suministrado por Microchip.

CONFIGURACIONES

Según el sistema operativo a utilizar se configura el dispositivo de las siguientes maneras:

> EN GNU LINUX

Normalmente no hay que instalar algún driver o similar para comunicarnos con la tarjeta **Pinguino** utilizando la configuración **CDC**. Sin embargo en **Processing** tenemos que crear un enlace simbólico por que el modulo **RXTXbin** de **JAVA** no lo reconoce como puerto **/dev/ttyACM0**.

Para hacer nuestro enlace simbólico abrimos una consola y digitamos la siguiente instrucción :

```
sudo ln -s /dev/ttyACM0 /dev/ttyS20
```

donde **ttyS20** es una declaración mas común para un puerto serie en **GNU LINUX.**

> EN WINDOWS

Para este sistema operativo necesitamos un driver en este caso **mchpcdc.inf** que nos suministra la empresa Microchip, este driver nos va a servir de enlace entre el puerto **USB** y **Pinguino** pero emulado como un puerto serial **COM**.
También necesitamos el archivo **mchpusb.ini** este archivo contiene información del driver y se utiliza para que el sistema operativo windows sepa que driver asignar al dispositivo cuando se conecta por primera vez.

PROGRAMACIÓN

A continuación se dará un ejemplo sencillo para controlar el tiempo de encendido y apagado de un led desde **Processing**.

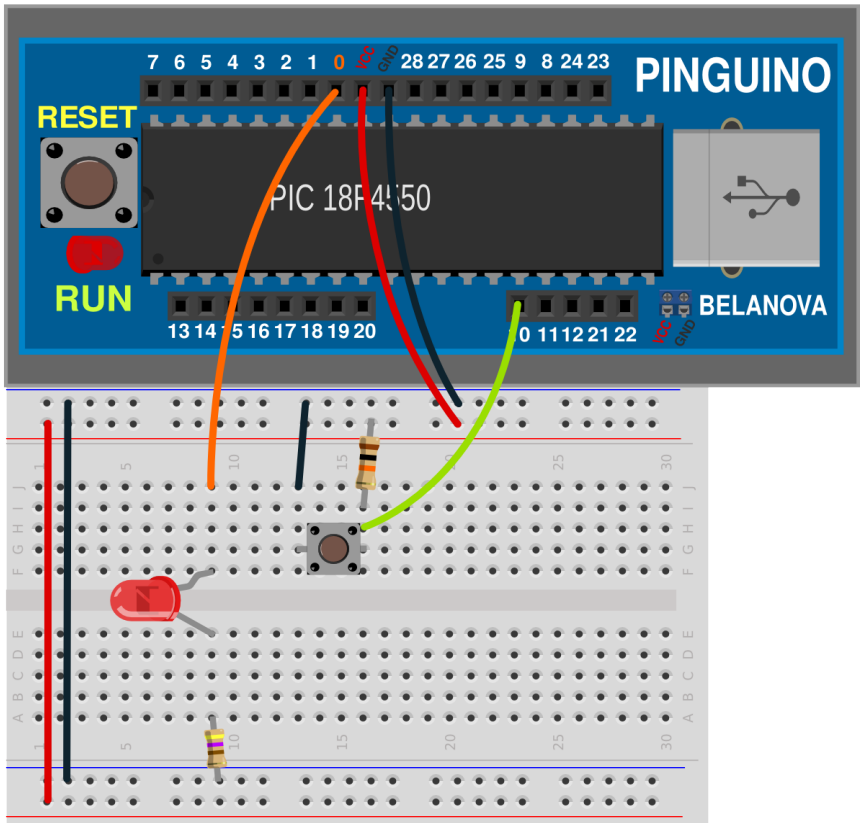
PROGRAMA 1 PINGUINO

Codigo1 – Recepción de un dato

```
#define PIC18F4550
unsigned char rxstr[64];
unsigned char ucDatosRecibidos;
unsigned char ucDato;
int nRetardo;
void setup(){
  ucDatosRecibidos=0;
  pinMode(0,OUTPUT);
  pinMode(10,INPUT);
}
void loop(){
  ucDatosRecibidos=CDC.read(rxstr);
  if (ucDatosRecibidos>0){
    ucDato=rxstr[0];
     nRetardo = ucDato * 100;
     digitalWrite(0,HIGH);
     delay(nRetardo);
     digitalWrite(0,LOW);
  }
}
```

Las variables para el manejo de datos son declaradas como **char** sin signo así que sus valores van de **0** a **255**.

Figura 1 – Montaje



Los datos que se envían son caracteres alfanuméricos entre letras, números y símbolos especiales. A cada caracteres le corresponde un numero decimal comprendido entre **0** y **255** según como especifica el estándar **ASCII**.
Por este método de comunicación podemos enviar mensajes de texto que serian un arreglo de caracteres y también valores numéricos de un byte ya que a cada carácter le corresponde un valor numérico único.

La variable **rxstr[64]** es la que recibe los caracteres enviados, es un arreglo de tamaño 64 indicando la cantidad de caracteres que puede almacenar en una recepción de datos. Esto quiere decir que estamos en capacidad de enviar 64 datos a la vez a nuestra tarjeta **Pinguino**.
La instrucción

```
ucDatosRecibidos=CDC.read(rxstr);
```

sirve para saber cuantos caracteres son envidados, es decir si se envían 2 caracteres la variable **ucDatosRecibidos** sera igual a 2. También esta instrucción hace ya la lectura de los datos y los guarda en el arreglo **rxstr**, recordemos que podemos enviar solo hasta **64** caracteres a la vez desde nuestra aplicación en este caso desde **Processing**.
La instrucción

```
if (ucDatosRecibidos>0)
```

Sirve para preguntar si hay un nuevo dato y así leerlo, para este ejemplo solo enviaremos un dato a la vez desde **Processing**, así que solo leemos el dato guardado en la posición cero del arreglo **rxstr**.

```
ucDato=rxstr[0];
```

Si no hay un nuevo envío de datos la variable **ucDatosRecibidos** no es mayor a cero.

PROGRAMA 1 PROCESSING

Codigo1 – Envío De Un Dato

```
import processing.serial.*;

Serial miPuerto;
String puerto[];
int i;

void setup(){
puerto=Serial.list();
println(puerto[0]);
  miPuerto=new Serial(this,puerto[0],115200);
}

void draw(){
   miPuerto.write("2");
   delay(5000);
   miPuerto.write("d");
   delay(5000);
}

void stop(){
  miPuerto.stop();
}
```

En **Processing** utilizamos la librería Serial que trae ya por defecto incluida, esta librería no permite la realización de comunicación serial asíncrona.
Con la instrucción

```
Serial miPuerto;
```

instaciamos el objeto **miPuerto** para el manejo de la comunicación serial.
La variable

```
String puerto[];
```

sirve para almacenar el nombre de los puerto seriales encontrados en el sistema.
Por defecto vamos a asignar el primer puerto encontrado que esta almacenado en **puerto[0]**, la instrucciones que realizan este proceso son

```
port=Serial.list();
println(port[0]);
```

donde utilizamos un **println** para que nos muestre en pantalla el nombre del puerto a utilizar. Si nuestro computador tiene otros puertos seriales procedemos a verificar cual es el que se asigno para nuestra tarjeta **Pinguino** y procedemos a ver en que posición quedo guardada utilizando la instrucción

```
println(port[numero]);
```

donde numero es el valor que queremos enviar, por ejemplo si queremos saber cual es el nombre que se asigno en la posición 1 colocariamos

```
println(port[ 1]);
```

y así sucesivamente. Una vez encontrado nuestro puerto, trabajamos con este en nuestro programa.
La instrucción

```
miPuerto=new
Serial(this,puerto[0],115200);
```

Crea y configura nuestro puerto serial, es este caso a una velocidad de transmisión de **115200 baudios** por que así también esta especificado en el el programa de la tarjeta **Pinguino** para la comunicación en modo **CDC.**
Con la instrucción

```
myPort.write("2");
```

enviamos un carácter, es por esto que esta el **2** entre comillas, en el conjunto de caracteres **ASCII** al **2** le corresponde el valor de **49**, por lo tanto el led encenderá y apagara cada 500 milisegundos o medio segundo según esta instrucciones se **Pinguino**

```
    nRetardo = ucDato * 100;
    digitalWrite(0,HIGH);
    delay(nRetardo);
    digitalWrite(0,LOW);
```

Para el carácter **d** encenderá y apagara cada segundo.

PROGRAMA 2 PINGUINO

En este otro ejemplo enviaremos dos caracteres a la vez, esta vez uno para seleccionar una opción y otro como dato. En ejemplo tendremos dos opciones, la de visualizar dato enviado y la de apagar dato.
Los comandos estarán indicados con las letras **V** y **L**, donde **V** es para visualizar y **L** para limpiar.

En este ejemplo enviaremos el **1** pero recordemos que en realidad se visualizara su equivalente en el código **ASCII** el cual es el **49** en decimal o el **00110001** en binario que el que veríamos en los leds. Los valores **ASCII** para los caracteres numéricos son

CARACTER	DECIMAL	BINARIO
0	48	110000
1	49	110001
2	50	110010
3	51	110011
4	52	110100
5	53	110101
6	54	110110

7	55	110111
8	56	111000
9	57	111001

De la tabla podemos observar que los 4 bits menos significativos corresponden al equivalente numérico en formato binario.
Una manera de convertir el carácter numérico enviado a su equivalente numérico es restarle 48 al dato enviado, pues 48 corresponde al valor **ASCII** del 0, por ejemplo, si enviamos los siguientes dato y les restamos 48 tendríamos:

- Dato = "0" - 48 = 48-48=0
- Dato = "1" - 48 = 49-48=1
- Dato = "2" - 48 = 50-48=2

Codigo2 – Recepción opción y dato

```
#define PIC18F4550
unsigned char rxstr[64];
unsigned char ucDatosRecibidos;
unsigned char ucDato;
int nRetardo;
void setup(){
  ucDatosRecibidos=0;
  pinMode(0,OUTPUT);
  pinMode(10,INPUT);
}
void loop(){
  ucDatosRecibidos=CDC.read(rxstr);
  if (ucDatosRecibidos>0){
    if(rxstr[0]=='V'){
      ucDato=rxstr[1];
      PORTB = ucDato;
    }
     if(rxstr[0]=='L'){
      PORTB = 0;
    }
  }
}
```

- Dato = "3" - 48 = 51-48=3

En este ejemplo si desde **Processing** se envía la opción V y el dato 1 tenemos que:
ucDatosRecibidos = 2

rxstr[0] = "V"
rxstr[1] = "1"
luego el dato es
ucDato = rxstr[1] = 49= 00110001

y esto es lo que veríamos en los leds

Codigo2 – Envío De Un Opción Y Dato

```
import processing.serial.*;

Serial miPuerto;
String puerto[];
int i;

void setup(){
puerto=Serial.list();
println(puerto[0]);
 miPuerto=new Serial(this,puerto[0],115200);
}

void draw(){
   miPuerto.write("V"+"1");
   delay(500);
   miPuerto.write("V"+"2");
   delay(500);
   miPuerto.write("L");
   delay(500);
}

void stop(){
 miPuerto.stop();
}
```

por 5 segundos.

PROGRAMA 2 PROCESSING

En **processing** enviamos los caracteres **1** y **2** cada 5 segundos y después enviamos la opción de limpiar o apagar los leds conectados en el **puerto B** de la tarjeta **Pinguino**.