UART (Serial)

Last modified January 7, 2015

Introduction

Un UART permet de transférer des données entre une sortie TXD (transmission) et une entrée RXD (réception). Normalement, les broches sont sur des processeurs différents. Les données sont envoyées à partir d'une broche TXD, selon une séquence et à une vitesse prédéfinie. Les données transmisses sont reçues sur la broche RXD et traitées par le récepteur.

Généralement, les systèmes envoient et reçoivent des données, il y aura donc un TXD et RXD à chaque extrémité. Les connexions doivent donc être croisées (TxD1 ⇨ RxD2 et TxD2 ⇨ RxD1).

UART Technical Details

Le baud est l'unité de mesure du nombre de symboles transmissibles par seconde. Le baud rate, ou vitesse de transmission, est le nombre de caractères transmis par seconde. Les valeurs standards sont généralement: 9600; 19200; 115200; …

Lors de connexions directes entre les broches TXD / RXD, il ne faut utiliser que des niveaux TTL / 0V à 3,3V.

NETMF supporte les ports série (UART) de la même manière que le .NET framework complet. Les ports séries sur PC et sur NETMF sont appelés "ports COM." Le premier est le port COM1 (il n'y a pas de COM0). Sur les processeurs, le premier UART est généralement l'UART0 (et pas non pas l'UART1), donc, COM1 est l'UART0 ... etc.

Informations sur les ports: MSInfo32 et dans le gestionnaire de périphériques

Valeurs usuelles

Les valeurs les plus utilisées (par défaut) sont:

Nombre de bits:	8 bits,
Stop bits:	1 stop bit
Parité:	pas de parité
Bits par seconde:	9600 bits/s

Software UART

La classe SignalGenerator de GHI, permet de simuler un UART. L'avantage est que toutes les broches GPIO peuvent être utilisées pour envoyer les données UART. Un bon exemple est un affichage série de caractères. Généralement ce type de circuit ne nécessite pas de grande vitesse et n'est pas mis à jour fréquemment. Si tous les UART sont utilisés, chaque port GPIO peut être utilisé pour simuler une transmission UART.

RS232

Le processeur utilise des tensions de de 0 et 3,3V comme niveaux logiques. Cela permet le raccordement des sorties d'un micro aux entrées sur un autre microcontrôleur en toute sécurité. Ces niveaux sont appelés TTL. Dans notre cas, niveau TTL UART.

Pour permettre des connexions de grandes distances, le très ancien standard RS232 défini l'état bas au +12V. Sonne comme il est en arrière, mais voilà comment il est! Ce port se trouve sur de nombreux systèmes mais tend à disparaître des PC récents. On peut facilement en ajouter à l'aide d'un câble USB – Port série.

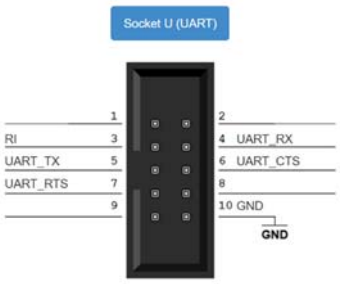
Il n'est pas possible de relier directement des broches aux niveaux RS232 à des niveaux TTL. Il faut pour cela utiliser un petit circuit qui convertit les niveaux RS232 en TTL, comme le circuit MAX232.

Connexions

Les UART de la carte Spider2

UART	Port série	Connecteur		Broches			
		type	N°	TX	RX	RTS	CTS
UART1*	COM2	U	4	4	5	6	7
UART2	COM3		8				
UART3	COM4		9				
UART0	COM1		11				

*Hardware Handshaking supporté



Tx: Transmit Data. Broche de transmission des données (aussi appelée TD ou TxD)
Rx: Receive Data. Broche de réception des données (aussi appelée RD ou RxD)
RTS: *Request To Send*. Demande d'autorisation à émettre
CTS: *Clear To Send*. Autorisation d'émettre

Exemple d'utilisation d'un UART

Le programme suivant envoie la valeur d'un compteur de 10 fois par seconde. Les données sont envoyées à 115200 bauds. Assurez-vous que la réception soit configurée de la même manière. Ce programme envoie les données sur COM1 d'une carte NETMF. Ce numéro de COM n'a rien à voir avec le numéro de COM sur votre PC. Par exemple, vous pouvez avoir un port série USB sur votre PC qui correspond au COM8 et si vous avez besoin d'ouvrir COM8 sur votre PC, pas COM1. Le programme NETMF utilisera toujours COM1 car il utilise l'UART0 (COM1).

Les données envoyées peuvent être affichées sur un programme de terminal, comme TeraTerm. Notez comment nous avons fini la chaîne avec "\r \n". Le "\r" est le code pour dire au terminal de "retour" retour au début de la ligne et "\ n" est d'ajouter «nouvelle» ligne. Lorsque les données sont reçues sur l'UART, il est automatiquement mis en attente dans la queue de réception de sorte que vous ne perdiez pas de données.

L'utilisation d'un port série requiert l'assembly ***Microsoft.SPOT.Hardware.SerialPort***.
Pour obtenir les énumérations Parity ou StopBits, l'assembly ***Microsoft.SPOT.Hardware*** est également nécessaire.

C# P:\Programmation\C#\M242\Bus\UART\UART-COM01\UART-COM01\UART_COM_01.cs

```
using System.IO.Ports;
using System.Text;
using System.Threading;

public class Program
{
```

```
    public static void Main()
    {
        SerialPort UART = new SerialPort("COM1", 115200);
        int counter = 0;
        UART.Open();
        while (true)
        {
            // create a string
            string counter_string = "Count: " + counter.ToString() + "\r\n";

            // convert the string to bytes
            byte[] buffer = Encoding.UTF8.GetBytes(counter_string);
            // send the bytes on the serial port
            UART.Write(buffer, 0, buffer.Length);
            // increment the counter;
            counter++;
            //wait...
            Thread.Sleep(100);
        }
... }
}
```

Receiving Data

Le programme suivant attend de recevoir un byte sur le port et de le retourner sous forme d'une chaine de caractère.

C#

```
using System.Threading;
using System.IO.Ports;
using System.Text;

public class Program
{
    public static void Main()
    {
        SerialPort UART = new SerialPort("COM1", 115200);
        int read_count = 0;
        byte[] rx_byte = new byte[1];

        UART.Open();
        while (true)
        {
            // Lecture du byte
            read_count = UART.Read(rx_byte, 0, 1);
            if (read_count > 0)// do we have data?
            {
```

```
                // Crée une chaine
                string counter_string =
                        "You typed: " + rx_byte[0].ToString() + "\r\n";
                // Converti la chaine en bytes
                byte[] buffer = Encoding.UTF8.GetBytes(counter_string);
                // Envoie les bytes sur le port série
                UART.Write(buffer, 0, buffer.Length);
                //Attend...
                Thread.Sleep(10);
            }
        }
    }
}
```

Sending and Receiving

L'exemple suivant est une boucle de retour. Connecter un fil entre les broches TX et RX du port et contrôler que les données envoyées sont reçues correctement.

C#

```
using System.IO.Ports;
using System.Text;
using System.Threading;
using Microsoft.SPOT;

public class Program
{
    public static void Main()
    {
        SerialPort UART = new SerialPort("COM1", 115200);
        int read_count = 0;
        byte[] tx_data; // Buffer d'envoi
        byte[] rx_data = new byte[10]; // Buffer de réception
        tx_data = Encoding.UTF8.GetBytes("FEZ");
        UART.ReadTimeout = 0;
        UART.Open();

        while (true)
        {
            // Envoie toutes les données en attente dans le buffer d'envoi et vide le buffer
            UART.Flush();
            // Envoie un nombre spécifié de data depuis le buffer
            UART.Write(tx_data, 0, tx_data.Length);
            // Attend que la transmission soie terminée
            Thread.Sleep(100);
            // Lecture des données
            read_count = UART.Read(rx_data, 0, rx_data.Length);
            if (read_count != 3)
```

```
            {
                // we sent 3 so we should have 3 back
                Debug.Print("Wrong size: " + read_count.ToString());
            }
            else
            {
                // the count is correct so check the values
                // I am doing this the easy way so the code is more clear
                if (tx_data[0] == rx_data[0])
                {
                    if (tx_data[1] == rx_data[1])
                    {
                        if (tx_data[2] == rx_data[2])
                        {
                            Debug.Print("Perfect data!");
                        }
                    }
                }
            }
            Thread.Sleep(100);
        }
    }
}
```