Facultad de Informática
Universidad Complutense de Madrid

Prática 4 EC - Comunicación Serie universal

15 de febrero de 2024

1. Interfaces de baja velocidad

Los tres interfaces de baja velocidad más utilizados son: I2C, SPI y UART. Son interfaces de transmisión serie (se transmite un solo bit por ciclo de reloj) y la frecuencia de reloj es diferente en cada sistema.

1.1. I2C (Inter-Integrated Circuit)

El interfaz serie I2C utiliza dos pines: SDA (Slave Data Adress) y SCL (Slave Clock). Su funcionamiento incluye las siguientes fases:

- Se envía una señal al pin SCL para indicar que se quiere iniciar una transferencia.
- Los siguientes 7 bits que se envían es la dirección del dispositivo esclavo (máximo 128 dispositivos esclavos).
- Los siguientes 8 bits son la transferencia de datos entre el dispositivo maestro y el esclavo.

La mayoría de dispositivos con una interfaz I2C la utilizan para realizar actualizaciones del firmware interno o simplemente para iniciar el protocolo de comunicación entre dos dispositivos. Por ejemplo puede ser que la CPU necesite enviar datos a un periférico en el sistema, y antes de utilizar los puertos de alta velocidad envía un bit a través del I2C para informar al dispositivo esclavo que quiere empezar una transferencia de datos.

1.2. SPI (Serial Port Interface)

SPI es un protocolo de comunicación ampliamente utilizado por una gran cantidad de dispositivos, por ejemplo los lectores de tarjetas RFID, las tarjetas SD y microSD de baja velocidad y los dispositivos Bluetooth utilizan este tipo de interfaz de manera interna. Al igual que el I2C se trata de un bus de un solo bit entre el dispositivo maestro y el esclavo, pero es diferente en la forma en la que transmite los datos ya que funciona con los siguientes pines:

- MOSI (Master Out Slave In): Este pin indica que el dispositivo maestro es el que envía el dato hacía el dispositivo esclavo.
- MISO (Master In Slave Out): Si este pin está activo son los dispositivos esclavos quienes envían los datos hacía el dispositivo maestro.
- SCLK: Envía la señal se reloj, la cual es necesaria para enviar los datos tanto desde la interfaz SPI maestro como desde la interfaz SPI esclavo. La frecuencia de reloj indica la cantidad de datos que se van a enviar por segundo.
- CS (Chip Select): Si el pin está activo indica que se pueden enviar los datos a ese dispositivo esclavo. Un dispositivo SPI maestro va a tener tantos pines CS como dispositivos SPI esclavos haya en el sistema.

La ventaja del SPI es que se pueden conectar varios SPI esclavos en serie para emular un bus paralelo, aunque no es tan eficiente en cuanto a la cantidad de pines ya que cada interfaz SPI requiere 3 pines.

1.3. UART (Universal Asynchronous Receiver-Transmitter)

UART es un protocolo para el intercambio de datos en serie entre dos dispositivos. Es sumamente simple y utiliza solo dos hilos entre el transmisor y el receptor para transmitir y recibir datos en ambas direcciones. Las principales características son:

- Datos de 7 u 8 bits.
- Generador / Detector de paridad por hardware: (odd, even, none)
- Registros de desplazamiento y buffers independientes para transmisión y recepción.
- Transmisión y recepción de datos partiendo por bit LSB.
- Detección de bit de start para salir de modos de bajo consumo.
- Generador de Baudios con modulación.
- Flags de estado para detección de errores y detección de dirección.
- Interrupciones independientes para transmisión y recepción.

La UART de la placa se comunica a través de los pines 4 y 5 del puerto 3 (P3.4 y P3.5). Estos pines están multiplexados, y para su utilización por la UART, deben usarse los registros P3SEL0 y P3SEL1, configurándolos en sus bits correspondientes para seleccionar la función del módulo primario (veáse el guión de la práctica 2).

Podemos ver la documentación relacionada con la UART en la guía `slau367p.pdf`, sección 30, y en concreto sus registros en la sección 30.4

Dentro de los registros de la UART, son de especial interés los registros UCAxIE, que se utiliza para habilitar/deshabilitar las interrupciones de la UART y el registro UCAxIFG, que contiene los flags de interrupción.

Registro UCAxIE

Bit	Campo	Tipo	Reset	Descripción
1	UCTXIE	RW	0	Activa interrupción en transmisión (0:desactivada, 1:activada)
1	UCRXIE	RW	0	Activa interrupción en recepción (0:desactivada, 1:activada)

Registro UCAxIFG

Bit	Campo	Tipo	Reset	Descripción
1	UCTXIFG	RW	0	Flag de interrupción TX (0:no pendiente, 1: pendiente)
1	UCRXIFG	RW	0	Flag de interrupción RX (0:no pendiente, 1: pendiente)

2. Configuración previa

Previo al uso de la UART, hay que configurar sus registros, así como una señal de reloj a una frecuencia específica para que pueda comunicarse.

Configuración de relok de 8MHz

```
CSCTL0_H = CSKEY >> 8;                    // Unlock clock registers
CSCTL1 = DCOFSEL_3 | DCORSEL;             // Set DCO to 8MHz
CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;
CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers
CSCTL0_H = 0;                             // Lock CS registers
```

Configuración del periférico USCI de la placa como UART

```
// Configure USCI_A1 for UART mode
UCA1CTLW0 = UCSWRST;                        // Put eUSCI in reset
UCA1CTLW0 |= UCSSEL__SMCLK;                // CLK = SMCLK

// Baud Rate calculation
// 8000000/(16*9600) = 52.083
// Fractional portion = 0.083
// User Guide Table 21-4: UCBRSx = 0x04
// UCBRFx = int ( (52.083-52)*16) = 1
```

```
UCA1BR0 = 52;                                // 8000000/16/9600
UCA1BR1 = 0x00;
UCA1MCTLW |= UCOS16 | UCBRF_1 | 0x4900;
UCA1CTLW0 &= ~UCSWRST;                      // Initialize eUSCI

//////////////////////////////////////////////////////
// Configure interruptions to receive and transmit data in register UCA1IE
// Por completar
//////////////////////////////////////////////////////
```

RTI para las interrupciones de la UART

```
#pragma vector=USCI_A1_VECTOR
__interrupt void USCI_A1_ISR(void) {
    //a rellenar
}
```

NOTA: La interrupción de la UART se dispara cada vez que se recibe un dato nuevo, y **cada vez que se puede enviar un dato** nuevo. Hay que evitar enviar datos en todas las interrupciones, o la terminal podrá saturarse en el ordenador, impidiéndonos escribir desde allí y por tanto recibir en la placa.

3. Ejercicios para placa de desarrollo MSP430FR6989

1. Realizar un programa para que la UART transmita el abecedario con la siguiente configuración: 9600, 8 bits de datos, 1 bit de stop, y sin paridad y que se pueda visualizar a través del USB en el hyperteminal. PISTA: se puede utilizar un temporizador en cuya interrupción se mande una nueva letra a la UART solo si esta ha disparado previamente una interrupción de transmisión.

2. Realizar un programa para que desde el hyperterminal se envíen letras mayúsculas del abecedario a la UART con la siguiente configuración: 9600 baudios, 8 bits de datos, 1bit de stop, y sin paridad. Las letras enviadas se introducen mediante el teclado del PC y una vez recibidas se visualizan en el LCD de la placa. Se visualizarán al mismo tiempo las últimas 6 letras recibidas. Utilizar las funciones del LCD de la práctica anterior.

3. Unificar las dos funcionalidades anteriores en un único programa.

4. Uso del terminal para envío/recepción de datos a través del USB

Para conectar el ordenador a la placa, tenemos que abrir una terminal UART en nuestro entorno de desarrollo. Esta se conectará a través del USB con la UART de nuestra placa.

1. Abrir `view > terminal`

2. Hacer click en “Open a terminal”

3. Seleccionar la configuración de la UART. En nuestro caso son 9600 baudios, 8 bits de datos, sin paridad, 1 bit de stop, y codificación por defecto ISO-8859-1. En el puerto serie habrá que seleccionar el correspondiente a la placa. Normalmente se llama `COMx` o `/dev/ttyACMx`. Si hay varios, se puede probar a conectar y desconectar la placa a ver cuál aparece/desaparece. Ese será el de nuestra MSP430.

5. Utilidades para el desarrollo de la práctica

```
//diccionario para transformar letras a segmentos del LCD
const char alphabetBig[26][2] =
{
    {0xEF, 0x00},  /* "A" LCD segments a+b+c+e+f+g+m */
    {0xF1, 0x50},  /* "B" */
    {0x9C, 0x00},  /* "C" */
    {0xF0, 0x50},  /* "D" */
```

```
    {0x9F, 0x00},  /* "E" */
    {0x8F, 0x00},  /* "F" */
    {0xBD, 0x00},  /* "G" */
    {0x6F, 0x00},  /* "H" */
    {0x90, 0x50},  /* "I" */
    {0x78, 0x00},  /* "J" */
    {0x0E, 0x22},  /* "K" */
    {0x1C, 0x00},  /* "L" */
    {0x6C, 0xA0},  /* "M" */
    {0x6C, 0x82},  /* "N" */
    {0xFC, 0x00},  /* "O" */
    {0xCF, 0x00},  /* "P" */
    {0xFC, 0x02},  /* "Q" */
    {0xCF, 0x02},  /* "R" */
    {0xB7, 0x00},  /* "S" */
    {0x80, 0x50},  /* "T" */
    {0x7C, 0x00},  /* "U" */
    {0x0C, 0x28},  /* "V" */
    {0x6C, 0x0A},  /* "W" */
    {0x00, 0xAA},  /* "X" */
    {0x00, 0xB0},  /* "Y" */
    {0x90, 0x28}   /* "Z" */
};

//Función para mostrar las letras mayúsculas del alfabeto contenidas
//en un vector de 6 elementos en el LCD de la placa.
void ShowBuffer(int buffer[]) {
    LCDMEM[9] = alphabetBig[(buffer[5])-65][0];
    LCDMEM[10] = alphabetBig[(buffer[5])-65][1];
    LCDMEM[5] = alphabetBig[(buffer[4])-65][0];
    LCDMEM[6] = alphabetBig[(buffer[4])-65][1];
    LCDMEM[3] = alphabetBig[(buffer[3])-65][0];
    LCDMEM[4] = alphabetBig[(buffer[3])-65][1];
    LCDMEM[18] = alphabetBig[(buffer[2])-65][0];
    LCDMEM[19] = alphabetBig[(buffer[2])-65][1];
    LCDMEM[14] = alphabetBig[(buffer[1])-65][0];
    LCDMEM[15] = alphabetBig[(buffer[1])-65][1];
    LCDMEM[7] = alphabetBig[(buffer[0])-65][0];
    LCDMEM[8] = alphabetBig[(buffer[0])-65][1];
}
```