Universidad del Valle de Guatemala
Departamento de Ingeniería Electrónica y Mecatrónica
Prof. Kurt Kellner
Prof. Pablo Mazariegos
Digital 2

Diego Mencos
Carné 18300
Sección 20

“Mini proyecto SPI: Pseudocódigo”

Pseudocódigo del Master

```
//Master.c

#define _XTAL_FREQ 8000000

#include <xc.h>
//Incluimos todas las librerías a usar como la de SPI, USART, la LCD.


//Definicion de variables
uint8_t slave1;
uint8_t slave2;
uint8_t slave3;

//Definino todas las variables que se utilizaran para las funciones de conversión de int a char

// BEGIN CONFIG
#pragma config FOSC = HS // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT enabled)
#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = ON // Brown-out Reset Enable bit (BOR enabled)
#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)
#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)
#pragma config WRT = OFF // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to by EECON control)
#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)
//END CONFIG

#pragma config BOR4V = BOR40V   // Brown-out Reset Selection bit (Brown-out Reset set to 4.0V)
#pragma config WRT = OFF        // Flash Program Memory Self Write Enable bits (Write protection off)


//Prototipo de FUnciones
void setup(void);
void __interrupt() isr(void);
// Incluimos las demás funciones que se realicen (entre ellas estarán las que se utilicen para poner los valores en la LCD y en la terminal virtual)



//Interrupciones
void __interrupt() isr(void) {
```

```
    if (PIR1bits.RCIF == 1) {
        valorRX = UART_RECIBIMOSVALOR (); //Aqui función que se realice para recibir el valor de la terminal
        PIR1bits.RCIF = 0;
    }
}

void main() {
    setup(); //Llamamos a la configuracion
    Lcd_Init(); //Inicializamos el LCD
    Lcd_Clear(); //Limpiamos el LCD
    UART_CONFIG//Funcion que se utilizara para configurar la comunicación USART

    //Menu cual nos da la opcion de seleccinar que variable queremos ver

    UART_send_string("PRESIONE: A) ADC B) CONTADOR D) TEMPERATURA"); //Enviaremos el usuarios
los comandos para desplegar lo que el quiera

    //Mostramos en la LCD los comando S1,S2,S3
    Lcd_Set_Cursor(1, 1);
    Lcd_Write_String("S1:");
    Lcd_Set_Cursor(1, 8);
    Lcd_Write_String("S2:");
    Lcd_Set_Cursor(1, 14);
    Lcd_Write_String("S3:");


    while (1) {

        //Llamamos al primer SLAVE 1-ADC
        PORTCbits.RC2 = 0; //Slave Select 1 – EL RC2 Sera el bit de selector para el slave 1
        __delay_ms(1);

        spiWrite(1);
        slave1 = spiRead(); //Copiamos a la variable el ADC

        __delay_ms(1);
        PORTCbits.RC2 = 1; //Slave Deselect 1

        //Convertimos el valor en distintos char para poder desplegarlos en la
        //LCD
        Numero_a_charADC(slave1);//función que servirá para convertir el valor en ADC en un char para la LCD
     //  Con las funciones de la LCD- Lcd_Set_Cursor() y Lcd_Write_Char() escribimos los valores obtenidas de
//  la funcion anterior en la LCD, de igual manera servirán para imprimirlos en la terminal


        //Llamamos al segundo SLAVE 2-Contador
        PORTCbits.RC1 = 0; //Slave Select 2-Contador- EL RC1 sera el bit de selección para el slave 2
        __delay_ms(1);

        spiWrite(1);
        slave2 = spiRead();

        __delay_ms(1);
        PORTCbits.RC1 = 1; //Slave Deselect 2



        //Convertimos el valor en distintos char para poder desplegarlos en la
```

```
        //LCD
        Numero_a_charCONTADOR(slave2);

     //  Con las funciones de la LCD- Lcd_Set_Cursor() y Lcd_Write_Char() escribimos los valores obtenidas de
//  la funcion anterior en la LCD, de igual manera servirán para imprimirlos en la terminal


        //Llamamos al tercer SLAVE 3-Termometro
        PORTCbits.RC0 = 0; //Slave Select 3-Termometro
        __delay_ms(1);

        spiWrite(1);
        slave3 = spiRead();

        __delay_ms(1);
        PORTCbits.RC0 = 1; //Slave Deselect 3



        //Convertimos el valor en distintos char para poder desplegarlos en la
        //LCD
         Numero_a_charTEMPERATURA (slave3);

     //  Con las funciones de la LCD- Lcd_Set_Cursor() y Lcd_Write_Char() escribimos los valores obtenidas de
//  la funcion anterior en la LCD, de igual manera servirán para imprimirlos en la terminal

}

void setup(void) {

    ANSEL = 0;
    ANSELH = 0;
    TRISA = 0;
    PORTA = 0;
    TRISB = 0b00000011;
    PORTB = 0;
    TRISC = 0b10010000;
    PORTC = 0;
    TRISD = 0;
    PORTD = 0;
    TRISE = 0;
    PORTE = 0;
    INTCONbits.GIE = 1; //Habilitamos las interrupciones
    INTCONbits.PEIE = 1; //Habilitamos las interrupciones perifericas
    //Llamamos a la configuración del SPI
    //Configuracion de la comunicacion SPI

}
```

Pseudocódigo del Slave 1

Este código no se modificó mucho con respecto al laboratorio que se realizó anteriormente, únicamente se agregó el slave de la comunicación SPI

```
//Slave 1-ADC

#define _XTAL_FREQ 8000000
```

```
#include <xc.h>
#include "ADC.h"
//Importamos la librería del SPI


//Definimos variables
uint8_t contador;


// BEGIN CONFIG
#pragma config FOSC = HS // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT enabled)
#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = ON // Brown-out Reset Enable bit (BOR enabled)
#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is
digital I/O, HV on MCLR must be used for programming)
#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code protection
off)
#pragma config WRT = OFF // Flash Program Memory Write Enable bits (Write protection off; all program
memory may be written to by EECON control)
#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)
//END CONFIG


//Prototipos de funciones
void setup(void);
void __interrupt() isr(void);

//Configuramos interrupciones
void __interrupt() isr(void) {

    if (PIR1bits.ADIF == 1) {
        PORTD = ADRESH;
        contador = PORTD;
        PIR1bits.ADIF = 0;
        __delay_ms(0.4);
        ADCON0bits.GO = 1;
    } else if (SSPIF == 1) {
        PORTE = spiRead();
        spiWrite(PORTD);
        SSPIF = 0;
    }
}

int main() {
    setup();
    ADC(0, 0); //Seleccionamos canal AN0 y justificado a la izquierda
    ADCON0bits.GO_nDONE = 1;
    while (1) {

    }
    return 0;
}

void setup(void) {
    ANSEL = 0;
    ANSELH = 0;
```

```
    TRISA = 0;
    TRISA=0b00100000;
    PORTA = 0;
    TRISB = 0b00000011;
    PORTB = 0;
    TRISC = 0b00011000;
    PORTC = 0;
    TRISD = 0;
    PORTD = 0;
    TRISE = 0;
    PORTE = 0;
    INTCONbits.GIE = 1; //Habilitamos las interrupciones
    INTCONbits.PEIE = 1;
    contador = 0;


    INTCONbits.GIE = 1; // Habilitamos interrupciones
    INTCONbits.PEIE = 1; // Habilitamos interrupciones PEIE
    PIR1bits.SSPIF = 0; // Borramos bandera interrupción MSSP
    PIE1bits.SSPIE = 1; // Habilitamos interrupción MSSP


    //Configuramos la comunicacion SPI
}
```

Pseudocódigo del Slave 2

Este código no se modificó mucho con respecto al laboratorio que se realizó anteriormente, únicamente se agregó el slave de la comunicación SPI

```
//Slave 2-Contador

#define _XTAL_FREQ 8000000
#include <xc.h>
//Importamos librerías del SPI

// CONFIG1
#pragma config FOSC = INTRC_NOCLKOUT        // Oscillator Selection bits (XT oscillator: Crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN)
#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled and can be enabled by SWDTEN bit of the WDTCON register)
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT disabled)
#pragma config MCLRE = ON       // RE3/MCLR pin function select bit (RE3/MCLR pin function is MCLR)
#pragma config CP = OFF         // Code Protection bit (Program memory code protection is disabled)
#pragma config CPD = OFF        // Data Code Protection bit (Data memory code protection is disabled)
#pragma config BOREN = OFF      // Brown Out Reset Selection bits (BOR disabled)
#pragma config IESO = OFF       // Internal External Switchover bit (Internal/External Switchover mode is disabled)
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enabled bit (Fail-Safe Clock Monitor is disabled)
#pragma config LVP = OFF        // Low Voltage Programming Enable bit (RB3 pin has digital I/O, HV on MCLR must be used for programming)

// CONFIG2
#pragma config BOR4V = BOR40V   // Brown-out Reset Selection bit (Brown-out Reset set to 4.0V)
#pragma config WRT = OFF        // Flash Program Memory Self Write Enable bits (Write protection off)
```

```
//*****************************************************************************
//Variables
//*****************************************************************************
#define incrementar PORTBbits.RB0
#define decrementar PORTBbits.RB1
uint8_t contador; //entero de 8 bits sin signo



//*****************************************************************************
//Prototipos de funciones
//*****************************************************************************
void setup(void);
void __interrupt() isr(void);

//*****************************************************************************
//Interrupciones
//*****************************************************************************

void __interrupt() isr(void) {
   if (INTCONbits.RBIF == 1) {
      if (incrementar == 1) {
         PORTD++; //Incrementar el contador
         contador = PORTD;

      } else if (decrementar == 1) {
         PORTD--; //Decrementar el contador
         contador = PORTD;

      }
      else if (SSPIF == 1) {
         PORTE = spiRead();
         spiWrite(PORTD);
         SSPIF = 0;
      }
      INTCONbits.RBIF = 0;
   }

}


//*****************************************************************************
//Ciclo Principal
//*****************************************************************************

void main(void) {
   setup();
   while (1) {
      spiWrite(contador);
   return;

//*****************************************************************************
//Configuracion
//*****************************************************************************
```

```
void setup(void) {

    ANSEL = 0;
    ANSELH = 0;
    TRISA = 0b00100000;
    PORTA = 0;
    TRISB = 0b00000011;
    PORTB = 0;
    TRISC = 0b00011000;
    PORTC = 0;
    TRISD = 0;
    PORTD = 0;
    TRISE = 0;
    PORTE = 0;
    INTCONbits.GIE = 1; //Habilitamos las interrupciones
    INTCONbits.PEIE = 1;
    INTCONbits.RBIE = 1; //
    INTCONbits.RBIF = 0;
    IOCBbits.IOCB0 = 1; //Interrupt on change del B0
    IOCBbits.IOCB1 = 1; //iNTERUPTN ON CHANGE DEL B1

    INTCONbits.GIE = 1; // Habilitamos interrupciones
    INTCONbits.PEIE = 1; // Habilitamos interrupciones PEIE

    //Configuramos la comunicacion SPI
}
```

Pseudocódigo del Slave 3

```
//Slave 3-Termometro

#define _XTAL_FREQ 8000000
//Importamos librerias
#include <xc.h>
#include "ADC.h"
//Importamos la librería del SPI


//Definimos variables
signed int temperatura;


// BEGIN CONFIG
#pragma config FOSC = HS // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT enabled)
#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = ON // Brown-out Reset Enable bit (BOR enabled)
#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is
digital I/O, HV on MCLR must be used for programming)
#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code protection
off)
#pragma config WRT = OFF // Flash Program Memory Write Enable bits (Write protection off; all program
memory may be written to by EECON control)
#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)
//END CONFIG
//Prototipo de funciones
void setup(void);
```

```
void __interrupt() isr(void);

//Configuramos las interrupciones
void __interrupt() isr(void) {
    if (PIR1bits.ADIF == 1) {
        temperatura = ADRESH;
        PORTD=temperatura;
        PIR1bits.ADIF = 0;
        __delay_ms(0.8);
        ADCON0bits.GO = 1;
    }
}

int main() {
    setup(); //Llamamos a la configuracion
    ADC(0, 0); //Seleccionamos canal 0 y justificacion a la izquierda
    ADCON0bits.GO_nDONE = 1;
    PORTBbits.RB3 = 0; //Ponemos las salidas del semoforo que se utilizaran el RB3,RB4,RB5
    PORTBbits.RB4 = 0;
    PORTBbits.RB5 = 0;
    while (1) {

        spiWrite(temperatura); //Escribimos constantementes el valor de la
                        // temperatura en la comunicacion

        if (PORTD >= 36) { //Semaforo en rojo
            PORTBbits.RB3 = 1;
            PORTBbits.RB4 = 0;
            PORTBbits.RB5 = 0;
        } else if (PORTD < 36 & PORTD >=24) {  //Semaforo en amarillo
            PORTBbits.RB3 = 0;
            PORTBbits.RB4 = 1;
            PORTBbits.RB5 = 0;

        } else { //Semaforo en verde
            PORTBbits.RB3 = 0;
            PORTBbits.RB4 = 0;
            PORTBbits.RB5 = 1;
        } //Los valores de 36 y 24 cambiaran con respecto al ADC, al momento de  hacer el proteus se verá el
//rango de voltaje del LM35

    }
    return 0;
}

void setup(void) {
    ANSEL = 0;
    ANSELH = 0;
    TRISA=0b00100000;
    PORTA = 0;
    TRISB = 0b00000011;
    PORTB = 0;
    TRISC = 0b00011000;
    PORTC = 0;
    TRISD = 0;
    PORTD = 0;
    TRISE = 0;
    PORTE = 0;
```

```
    INTCONbits.GIE = 1; //Habilitamos las interrupciones
    INTCONbits.PEIE = 1;

    temperatura=0;

    INTCONbits.GIE = 1; // Habilitamos interrupciones
    INTCONbits.PEIE = 1; // Habilitamos interrupciones PEIE

    //Configuracion de la comunicacion SPI

}
```