Luis Castillo
Ian Fernández

Práctica P7_I2C_V1

Bus I2C + Periférico SRF08 para medición de distancias

Sincronización de los dispositivos (encuesta o interrupción)

- **Pulsador S4:** Se gestiona por encuesta
- **Pulsadores S3, S5, S6:** Se gestionan por interrupción
- **Timers T5, T6 y T7:** Se gestionan por interrupción
- **Timers T2, T3:** El timer T2 se sincroniza con el módulo OC1 y el timer T3 se sincroniza con el módulo ADC1.
- **ADC1:** Se gestiona por interrupción.
- **UART2:** Se gestionan las interrupciones para recepción y transmisión de datos.
- **OC1 (PWM):** Se sincroniza con el timer T2 y se hacen escrituras en el registro OC1RS
- **Bus I2C1 y Sensor SRF08:** El temporizador T6 será el que indique cuándo se recogerán muestras y por tanto cuando se transmitirá la información por el bus.

Funciones que llama el programa principal

- **inic_oscilator** (en oscilator.c)
  - Inicializa y configura el módulo oscilator con una frecuencia de 40MHz
- **inic_leds** (GPIO.c)
  - Define e inicializa los puertos correspondientes a los leds
- **inic_pulsadores** (GPIO.c)
  - Define e inicializa los puertos correspondientes a los pulsadores
- **inic_LCD** (LCD.c)
  - inicializa el display LCD estableciendo los estados iniciales para data y pines de control e inicializa la secuencia de inicialización de la LCD
- **copiar_FLASH_RAM** (memoria.c)
  - Recibe una cadena de texto y asigna sus valores a la variable *Ventana_LCD* la cual se usa para mandar datos al display
- **line_1** (LCD.c)
  - Establece el cursor en la primera línea del display
- **puts_lcd** (LCD.c)
  - Recibe la variable *Ventana_LCD* y despliega su información en el display LCD
- **line_2** (LCD.c)
  - Establece el cursor en la segunda línea del display
- **inic_crono** (timers.c)
  - Inicializa las variables de cronómetro y cambia el valor del *flag_inic_crono* a 1
- **inic_Timer7** (timers.c)
  - Inicializa y configura el Timer 7: PR = 50000, prescaler = 1:8
- **inic_Timer5** (timers.c)
  - Inicializa y configura el Timer 5: PR = 12500, prescaler : 1:8
- **inic_Timer3** (timers.c)
  - Inicializa y configura el Timer 3
  - Se inicializa el registro de la cuenta a 0 con TMR3 y se le indica que ha de contar 40.000 ciclos con PR3
  - El preescaler TCKPS = 0 se establece a 1:1 porque no se excede del máximo 65.536 ciclos
  - El reloj interno y el modo gate se dejan en 0 TCS y TGATE
  - Se activan las interrupciones mediante T3IE y se pone a 0 el flag de interrupción T3IF

- Por último se pone en marcha el timer con TON = 1

• **inic_CN** (CN.c)
- Inicializa el módulo CN

• **Inic_UART2** (UART_RS232.c)
- Inicializa el módulo UART2 para recibir y transmitir información

• **Inic_ADC1** (ADC1.c)
- Inicializa el ADC1 para convertir señales analógicas en digitales
- SSRC está establecido en 7 que es el auto converter, para sincronizarlo con el timer 3, se pone a 2
- ASAM = 1 establece el muestreo automático
- AD1CON2, AD1CON3 y AD1CON4 inicializan los registros de control a 0
- SMAC y ADCS son los campos que indican cada cuantos ciclos de reloj se realiza el muestreo
- El registro CHOSA selecciona la entrada analógica.
- Con los registros AD1PCFGH y AD1PCFGL se ponen todas las AN como digitales exceptuando las que vayamos a usar (El potenciómetro, la temperatura, Px, Py y la palanca)
- Como siempre, tenemos el interrupt enable AD1IE y el flag de interrupción AD1IF, ambos a 0.
- Por último con ADON se habilita el ADC

• **crono** (timers.c)
- Realiza el conteo de las variables del cronómetro, en cada cambio de valor de una de estas variables se realiza la conversión de tiempo de número entero a ASCII con la función *conversion_tiempo* de modo que la variable *Ventana_LCD* actualiza su contenido.

• **comprobar_inic_crono** (timers.c)
- Evalúa el cambio del *flag_inic_crono* y realiza la conversión del tiempo de las variables del cronómetro, esto se realiza para que estas operaciones no se hagan directamente en la rutina de atención

• **calcularMediaMuestras** (ADC1.c)
- Cuando el ADC1 ha recogido 8 muestras por cada entrada analógica, realiza una media de esas 8 muestras para cada una de las entradas. Cuando realiza esto, vuelve a habilitar el ADC que en la llamada a esta función se había deshabilitado y pone a 0 el *flag_muestras* que indica cuando se han realizado todas las muestras de cada entrada.
- Si el modo de control del servomotor es con el potenciómetro entonces se llama a la función *relacion_adc_pwm* con el valor de la media de las muestras del potenciómetro.

• **inic_OC1** (OCPWM.c)
- Inicializa el módulo OC1 y el pulso con duración intermedia.

• **mostrar_OC1** (OCPWM.c)
- Llama a la función *conversion_4digitos* para convertir los 4 dígitos del registro OC1RS a una cadena de caracteres para ser mostrados en la LCD o en la terminal con UART

• **inic_Timer2** (timers.c)
- Inicializa el timer 2 con PR de 12500 para conseguir 20 ms con un prescaler de 1:64

• **inic_Timer6** (timers.c)
- Inicializa el timer 6 con PR de 43750 para conseguir 70ms con un prescaler de 1:64

- **InitI2C_1** (i2c_funciones.c)
  - Inicializa el bus I2C 1, asignándole un Baud Rate adecuado (en I2C1BRG) para la frecuencia con la que trabajamos (40 MHz). Por último, se indica que se quiere iniciar en modo Master en el registro I2C1CON e inicializamos el bus con I2CEN a 1.
- **inic_medicion_dist** (srf08.c)
  - Inicia una medición en el periférico SRF08 mediante el bus I2C 1 haciendo uso de la función *LDByteWriteI2C_1*. Devuelve un código de error (0 si va todo bien).
- **leer_medición** (srf08.c)
  - Función para leer la distancia medida, 2 Bytes y devuelve un código de error (0 si va todo bien).

Otras funciones importantes

- **lcd_cmd** (en LCD.c)
  - Se encarga de enviar los comandos necesarios para el funcionamiento de la pantalla LCD.
- **lcd_data** (en LCD.c)
  - Subrutina encargada de imprimir un carácter en la pantalla LCD.
- **Delay_ms** (en timers.c)
  - Función que espera el tiempo pasado por parámetro en milisegundos.
- **Delay_us** (en timers.c)
  - Función que espera el tiempo pasado por parámetro en microsegundos.
- **conversion_tiempo** (en utilidades.c)
  - Función que, dados dos dígitos, devuelve estos en forma de caracteres.
- **conversion_4digitos** (en utilidades.c)
  - Función que, dados cuatro dígitos, devuelve estos en forma de caracteres. Es una ampliación de la función *conversión_tiempo* para poder representar los valores obtenidos por el ADC, también la usamos para representar los valores del registro OC1RS.
- **recoger_valorADC1** (en ADC1.c)
  - Esta función se encarga de tomar las muestras de cada entrada analógica y guardarla en un vector para posteriormente mostrar el valor medio de las conversiones realizadas y no el ADC no toma muestras constantemente. Se evalúa el registro CHOSA y se almacena el valor recogido por la entrada analógica correspondiente, después se cede el ADC para la siguiente entrada. Cuando se han recogido 8 muestras de todas las entradas, se activa el *flag_muestras* para realizar la media y se inhabilita el ADC.
- **relación_adc_pwm** (en OCPWM.c)
  - Permite realizar un escalado del valor digital de la entrada ADC (0 - 1023) a valores entre DUTY_MIN y DUTY_MAX (312-1312).
- ***Funciones relacionadas con el I2C*** (en i2c_funciones.c)
  - *StartI2C_1*: Genera una condición de inicio para I2C y devuelve esa condición.
  - *RestartI2C_1*: Genera una condición de reinicio para I2C y devuelve esa condición.
  - *StopI2C_1*: Genera una condición de stop para I2C y devuelve esa condición.
  - *IdleI2C_1*: Espera a que el bus esté inactivo.
  - *WriteI2C_1*: Escribe 1 Byte en el bus.
  - *ACKCheck_1*: Devuelve el estado de reconocimiento del bus.
  - *NotAckI2C_1*: Genera un NO ACK.
  - *AckI2C_1*: Genera un ACK.
  - *getI2C_1*: Lee 1 Byte del bus.
  - *getsI2C_1*: Lee la cantidad especificada de bytes del bus usando la función anterior.
- **LDByteReadI2C_1** (en i2c_funciones.c)

- Haciendo uso de las funciones anteriores para el I2C 1, realiza una lectura de baja densidad y almacena los datos leídos en una dirección de memoria de tal forma que posteriormente se puedan procesar esos datos.

- **LDByteWriteI2C_1** (en i2c_funciones.c)
  - Haciendo uso de las funciones anteriores para el I2C 1, realiza una escritura del dato que se le otorgue a un dispositivo de baja densidad en el registro indicado.

Rutinas de atención

- **Rutina del módulo CN** (en CN.c): _CNInterrupt()
  - Comprueba el pulsador S3 y si está pulsado para o pone en marcha el timer 7.
  - Comprueba el pulsador S6 y si está pulsado reinicia el cronómetro y para el timer 7.
  - Comprueba el pulsador S5 y si está pulsado cambia el modo de control del módulo OC1 (si es 0 es con UART y si es 1 es con el potenciometro)
- **Rutina del timer T7** (en timers.c): _T7Interrupt()
  - Simplemente actualiza la variable mili cada 10ms e inhabilita el flag de interrupción.
- **Rutina del timer T6** (en timers.c): _T6Interrupt()
  - Simplemente establece el flag_T6 a 1 para notificar que el módulo SRF08 puede realizar la siguiente medición e inhabilita el flag de interrupción.
- **Rutina del timer T5** (en timers.c): _T5Interrupt()
  - Se implementa la máquina de estados para que envíe datos y comandos al LCD cada 2,5ms.
- **Rutina de la UART2** (en UART2_RS232.c): _U2RXInterrupt()
  - Evalúa el registro recepción de la UART2, si el carácter recibido es “p” o “P” para el cronómetro, si es “i” o “I” lo inicializa y si es “c” o “C” lo pone en marcha, para controlar el servo con PWM mediante la UART, si el carácter recibido es “m” o “M” se aumenta un valor de 10 al registro OC1RS validando que este no se encuentre en el límite del DUTY_MAX, si el valor recibido es “d” o “D” entonces realiza la misma validación pero a DUTY_MIN y resta 10 al OC1RS.
- **Rutina de la UART2** (en UART2_RS232.c): _U2TXInterrupt()
  - Contiene una máquina de estados que enviará los datos y comandos necesarios a una velocidad 4800 baudios mediante la UART2 (10 bits) y se verán reflejados en el terminal del Tera Term.

FUNCIONAMIENTO GENERAL

En esta práctica se configura el módulo SFR08, que hace uso de las ondas ultrasónicas para calcular distancias. Se busca que el dispositivo realice una medición cada 70 ms que es lo aconsejado por el fabricante, para ello, hacemos uso del temporizador T6 programado para que interrumpa 1 vez cada 70 ms.

Para realizar la medición de las distancias, usamos el bus I2C 1 y hay que seguir el protocolo de comunicaciones que este utiliza. A grandes rasgos lo que hay que hacer es establecer una comunicación inicial con el periférico que se quiera comunicar e indicando la operación que se quiera realizar (lectura | escritura), esto establecerá la placa como Master y el periférico como Slave. Una vez establecida la comunicación estos dos dispositivos seguirán un proceso de recibimiento y envío de datos y ACKs con los que se obtendrán los datos necesarios del periférico. Para finalizar con la comunicación, se envía un NO ACK.

En el programa principal esto sería de la siguiente forma:
Se realiza la comunicación inicial con *inic_medición_dis* , luego esperamos a que hayan pasado 70 ms y leemos la medición que SRF08 ha realizado *leer_medicion* y procesamos los 2 Bytes que ha recogido con *conversion_BytesToInt* que nos dejará el resultado en un formato legible (cm). Por último lo mostramos por pantalla.

Es importante que cuando se vaya a leer la medición, se pare el temporizador y se reinicie el registro de la cuenta para que este no se descuadre y podamos realizar las mediciones dentro del tiempo establecido. (Después de todo este proceso se vuelve a iniciar el contador)