Sistemas Embebidos para Tiempo Real

Laboratorio 2 - Interrupciones

Introducción

En este laboratorio se implementará un reloj de tiempo real utilizando un timer hardware de un microcontrolador y el módulo de timer software desarrollado en el Laboratorio 1. También se incorporará un módulo de adquisición de temperatura utilizando el sensor de temperatura y el conversor analógico digital (ADC) del microcontrolador. Para estas tareas se utilizarán interrupciones, generadas por el timer hardware y el ADC.

Objetivos

Objetivos generales

1. Escribir rutinas de atención a interrupciones y explicar el funcionamiento de las interrupciones en los microcontroladores.
2. Describir el problema de datos compartidos.
3. Adquirir buenas prácticas de codificación y documentación del código.

Objetivos particulares

1. Configurar periféricos internos del microcontrolador, tomando como ejemplo el Basic Clock Module+.
2. Manejar las distintas fuentes de reloj que posee el microcontrolador.
3. Utilizar el ADC del microcontrolador junto con su sensor de temperatura.
4. Provocar una inconsistencia de datos (problema de datos compartidos) forzando una interrupción oportunamente.
5. Escribir rutinas de atención a interrupciones que eviten el problema de datos compartidos, utilizando una solución básica.
6. Verificar el funcionamiento de una aplicación sencilla en hardware.
7. Reconocer los beneficios de la programación modular y aplicarla adecuadamente en un ejemplo sencillo.
8. Mantener un repositorio con diferentes versiones de una aplicación utilizando GIT.
9. Generar la documentación del código utilizando Doxygen.

Material

- PC con el IDE CCS (v8.3 o superior), Git y herramientas asociadas.
- Repositorio grupal de los Laboratorios en Gitlab de Facultad de Ingeniería.
- Plataforma de desarrollo Launchpad MSP-EXP430G2ET o MSP-EXP430G2 con el microcontrolador MSP430G2553.

Actividades a realizar

La placa de desarrollo Launchpad MSP-EXP430G2ET tiene soldado un cristal externo de 32768 Hz conectado a los pines XIN y XOUT del microcontrolador MSP430G2553. Además el microcontrolador posee un oscilador interno de baja frecuencia (12 kHz) denominado VLO. Estas dos fuentes se utilizarán como base de tiempos, la señal del cristal es LFXT1CLK y la del VLO es VLOCLK.

Se programará y probará una aplicación para generar una interrupción cada 250 ms, utilizando el TIMER_A del microcontrolador y el cristal o el oscilador interno, para incrementar el tiempo del reloj de tiempo real a

implementar. Este módulo estará basado en el módulo iniciado en el Laboratorio 1.

También se programará y probará una aplicación para utilizar el sensor de temperatura que funciona mediante una interrupción del ADC del microcontrolador. El código del módulo del sensor de temperatura será provisto por los docentes.

Parte 1) Estudio del hardware e implementación de módulos relacionados al timer hardware

Se pide:

1. Lectura obligatoria de las secciones que involucre la configuración del Basic Clock Module+ y el TIMER_A del MSP430G2553 [2, 3] y materiales de consulta relacionados [1].
2. Implementación del módulo hardware del timer (`timer_hw.h y timer_hw.c)` y la aplicación de prueba `test_timer.c`, descriptos a continuación.

Requerimientos:

1. `timer_hw.h`

   Interfaz pública del módulo hardware del timer con los prototipos de las funciones de inicialización (una para inicializar con el cristal, y otra con el VLO). Los archivos de encabezado deberán ser comentados utilizando la sintaxis Doxygen (siguiendo el ejemplo de comentarios que se encuentra en la web del curso).

2. `timer_hw.c` incluyendo las siguientes funciones:

   - `void config_timer_crystal(void)` Rutina de inicialización del timer hardware usando el cristal como fuente de reloj.
   - `void config_timer_VLO(void)` Rutina de inicialización del timer hardware usando el VLO como fuente de reloj.

   En primera instancia se programará el timer hardware tomando como fuente de reloj el cristal y en segunda instancia el VLO[1]. En los dos casos, comentar junto con la escritura de cada registro qué se está configurando, explicando claramente cómo se obtiene el "tick" de 250 ms utilizando el TIMER_A del microcontrolador.

   - rutina de atención a la interrupción (ISR)

   En esta instancia se conmuta un LED para verificar el funcionamiento.

3. `test_timer.c` incluyendo la función main

   Aplicación de prueba en la cual se configura el timer hardware y se entra en un bucle infinito. Recordar habilitar las interrupciones antes de entrar al bucle infinito.

Notas:

- Los archivos fuentes directamente relacionados con la aplicación de prueba se ubicarán en la carpeta `test`, mientras que los archivos de los módulos (.h y .c) que son reutilizados por diferentes aplicaciones (y probados con los test respectivos), se ubicarán en las carpetas `include` y `src`.
- Al iniciar las tareas de la parte 1), se debe crear un proyecto de nombre `lab2-parte1` en la carpeta `test/lab2-parte1`. Al finalizar las tareas de la parte 1), para identificar la entrega en la historia del repositorio, crear un tag de nombre `lab2-parte1`.

Se deberá tener previsto la posibilidad de mostrar el repositorio del grupo con la estructura de directorios conteniendo el código pedido.

Se diseñará y realizará una demo de validación de la aplicación de prueba del timer hardware que muestre el funcionamiento de la ISR conmutando un LED en cada interrupción (resultando en 2 destellos por segundo). Debe funcionar para los dos casos de fuentes de reloj.

Parte 2) Integración de módulo timer del laboratorio 1 con Parte 1)

En esta parte, se modificará el código de prueba del módulo timer software del laboratorio 1 para que el tiempo sea incrementado por el timer hardware. La función que incrementa el tiempo se llamará desde la

1 El timer hardware no puede tener dos fuentes de reloj simultáneas, por lo tanto deberá pensarse una manera adecuada de presentarlo en el código.

rutina de atención a la interrupción del TIMER_A.

1. Crear un nuevo proyecto `lab2-timer-test` en `test/lab2-timer-test` integrando los módulos implementados en el Laboratorio 1 con los de este laboratorio (archivos: `timer.h`, `timer.c`, `timer_hw.h`, `timer_hw.c` y `test_timer.c`).
2. Modificar la rutina de atención a la interrupción del TIMER_A para que llame la función que incrementa el tiempo. Recordar habilitar las interrupciones en el programa principal (`test_timer.c`) antes de entrar al bucle infinito.
3. Para verificar el correcto funcionamiento, primero configurar el timer hardware con la fuente de reloj del cristal y mantener la parte del código de la ISR que conmuta el LED con cada interrupción. Luego de comprobar el correcto funcionamiento con esa configuración, testear el funcionamiento con el VLO.
4. Comparar los resultados, según las distintas fuentes de reloj, en términos de costo, tamaño y precisión[2]. Para la precisión se sugiere contrastar utilizando un cronómetro durante al menos 2 minutos.

De ahora en más utilizar el timer hardware configurado con el cristal como fuente de reloj.

5. Modificar el main para realizar dentro de un bucle infinito una copia del tiempo a una variable privada del módulo main. El tiempo se copia repetidamente en cada ciclo.
6. Verificar el correcto funcionamiento de la aplicación inicializando el tiempo en un valor, correr la aplicación por un tiempo conocido (cronometrado) y verificar que el tiempo obtenido en el main es correcto.

Parte 3) Problemas de datos compartidos: generación de inconsistencia de datos

1. En la aplicación de prueba inicializar el reloj en un tiempo adecuado para mostrar la inconsistencia de datos, por ejemplo un tiempo cercano a las 00:00:00:000.
2. Configurar un breakpoint en la función que incrementa el reloj para detener la ejecución en el “tick” anterior de “dar vuelta el reloj”.
3. Ejecutar paso a paso hasta llegar a la llamada de `get_time(...)` en que se copia el valor del reloj (dentro del bucle infinito en el main).
4. Forzar una interrupción, escribiendo el flag correspondiente, en el momento adecuado para provocar una inconsistencia de datos y observando cómo se obtiene un dato de tiempo corrupto en el main.
5. Modificar el código para asegurar que no se produce el problema de datos compartidos.

Parte 4) Prueba del módulo de temperatura

El módulo de adquisición de temperatura está disponible en la sección Laboratorios de la página web del curso. Para probar el módulo se deberá adquirir una medida de temperatura y guardarla en una variable. Para lograr dicho objetivo hay que:

1. Crear un nuevo proyecto `temp-test` en `test/lab2-temp-test` integrando el módulo temperatura provisto por los docentes (archivos: temperatura.h, temperatura.c)
2. En el programa de prueba inicializar el módulo de temperatura con la función `tempInit()`.
3. Determinar la bandera en la cual la ISR del ADC avisará que la medida de temperatura está pronta usando la función `setFlagTemp(char* flag_main)`.
4. Lanzar una medida de temperatura con la función `runTemp()`, indicando la bandera con la cual la ISR del ADC avisará que la medida de temperatura está pronta.
5. Leer la medida de temperatura con la función `getTemp()` y guardarla en una variable. Al leer la temperatura, el módulo automáticamente reseteará la bandera.
6. Para finalizar esta parte crear un tag en el repositorio de proyectos de nombre `lab2-temp`.

Parte 5) Generación de documentación usando Doxygen

1. Ejecutar el “Doxygen GUI frontend” (doxywizard) para generar la documentación del proyecto.

2 También hay un diferencia importante en términos de consumo.

2. Se sugiere inicialmente usar el “Wizard” especificando:
    - Project:
        - Especificar *Project name*
        - Especificar *Source code directory*
        - Especificar *Destination directory* y seleccionar *Scan recursively.*
    - Mode
        - Seleccionar *Optimize for C or PHP*
    - Output
        - Seleccionar solamente *HTML*
    - Diagrams
        - Seleccionar *No diagrams*
3. Ejecutar haciendo clic en *RUN*
4. Abrir la página index.html en la carpeta de salida y navegar por la documentación generada.
5. Experimentar cambiando las opciones, generando nuevamente la documentación y volviendo a observar la salida.

Referencias

Documentos técnicos

1. MSP430G2553 LaunchPad™ Development Kit (MSP-EXP430G2ET) User's Guide
2. MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller datasheet (Rev. J)
3. MSP430x2xx Family User's Guide (Rev. J)
4. MSP430G2553 Device Erratasheet (Rev. I)
5. MSP430 Optimizing C/C++ Compiler v18.1.0 LTS User's Guide

Lecturas recomendadas

1. Problema de datos compartidos: “An Embedded Software Primer", Chapter 4: Interrupts, Section 4.3: The Shared-Data Problem
2. "Implementing a Real-Time Clock on the MSP430", Application Report, January 2001. [slaa076.pdf]
3. “MSP430 Microcontroller Basics”, Chapter 8: Timers, John Davies, 2008.

Enlaces

1. Launchpad MSP-EXP430G2: http://www.ti.com/tool/msp-exp430g2et
2. MSP430G2553: http://www.ti.com/product/MSP430G2553
3. Code Composer Studio: http://www.ti.com/tool/ccstudio
4. Code Composer Studio para MSP430 :http://www.ti.com/tool/ccstudio-msp
5. TI Clouds tools: http://dev.ti.com
6. TI Resource Explorer: http://dev.ti.com/tirex/