Sistemas Embebidos para Tiempo Real

Laboratorio 1 - Introducción al entorno de desarrollo CCS

Introducción

En este laboratorio se darán los primeros pasos de utilización del entorno de desarrollo integrado (IDE, Integrated Development Environment) Code Composer Studio (CCS) para la creación de aplicaciones embebidas y se introducirá la plataforma de hardware Launchpad MSP-EXP430G2 o MSP-EXP430G2ET con un microcontrolador MSP430G2553 a utilizar durante los laboratorios, desarrollando parcialmente un módulo de reloj de tiempo real.

En muchas aplicaciones donde es necesario manejar la noción del tiempo se utiliza un reloj de tiempo real para el monitoreo, disparo o etiquetado de eventos. Para el seguimiento del tiempo típicamente se utilizan dos métodos: contar los milisegundos transcurridos a partir de cierto instante de tiempo o directamente contar milisegundos, segundos, minutos y horas. Si además interesa la noción del tiempo al pasar de los días o a lo largo del año, se lleva la cuenta de los días, día del mes, mes, etc. Ambos métodos presentan ventajas y desventajas. En este laboratorio se desarrollará una primera parte de un módulo de reloj (que se llamará "timer") utilizando el segundo método. En el laboratorio 2 se continuará el módulo utilizando un temporizador hardware.

Este laboratorio también servirá como una introducción a GIT y a las buenas prácticas de programación.

Objetivos

Objetivos generales

- Emplear las herramientas del entorno de desarrollo CCS para compilar, ejecutar y depurar programas en plataformas de hardware.
- Emplear una herramienta de control de versiones.
- Emplear una herramienta para documentación automática de código.
- Repasar y aplicar conceptos básicos de programación en lenguaje C orientados a sistemas embebidos.

Objetivos particulares

1. Crear un proyecto en CCS y configurar sus opciones básicas. Escribir y agregar archivos fuente al proyecto.
2. Compilar y enlazar la aplicación. Interpretar los archivos generados.
3. Depurar la aplicación ejecutando en hardware, utilizando el Launchpad MSP-EXP430G2 o MSP-EXP430G2ET.
4. Observar la dependencia entre algunas decisiones en el código fuente y el programa generado por el compilador/linker.
5. Mantener un repositorio con diferentes versiones de una aplicación utilizando GIT.
6. Repasar y aplicar conceptos básicos de programación en C: visibilidad, alcance, tiempo de vida y almacenamiento de variables, cualificadores, tipos de datos, y modularización (interfaz pública/privada, utilizar funciones de acceso a variable, funciones privadas a un módulo, utilización de guarda).

Materiales

- PC con los siguientes programas instalados
    - CCS v8.3 o superior
    - Git y herramientas asociadas (por ejemplo Fork, Gitkraken o similar)
- Paquetes opcionales:
    - En Windows: minGW, o una herramienta de su preferencia que le permita compilar con GCC
    - En Linux: paquetes meld, gcc, tig
- Repositorio grupal de los Laboratorios, según se describe en el documento "Guía de instalación y

puesta a punto de herramientas para el Laboratorio de Sistemas Embebidos”
- Plataforma de desarrollo Launchpad MSP-EXP430G2 o MSP-EXP430G2ET con un microcontrolador MSP430G2553.

Actividades a realizar

Parte 1) Instalación de entorno de desarrollo y estudio del hardware

1. Lectura obligatoria de:
   a) Guía de instalación y puesta a punto de herramientas (disponible en el EVA del curso).
   b) MSP430G2553 LaunchPad™ Development Kit (MSP-EXP430G2ET) User's Guide [1].
   c) Slides del teórico “Introducción y conceptos básicos”, “Introducción al lenguaje C”, “Herramientas de desarrollo de software embebido”, y “Git: introducción al control de versiones”.
2. Creación del repositorio de Laboratorios (según la guía antes mencionada)

Parte 2) Destello de un LED (Blink LED)

El programa que se presenta en la Figura 1 tiene el propósito de hacer conmutar una salida digital a cierta frecuencia para hacer destellar un LED.

```
#include <msp430.h>
#define LED1 (0x0001)

int main(void)
{
    volatile unsigned int i;
    WDTCTL = WDTPW + WDTHOLD;       // Detiene el watchdog timer
    P1DIR |= LED1;                  // Configura pin LED1 salida
    while(1)
    {
        P1OUT ^= LED1;              // Conmuta LED1 usando XOR
        for (i=10000; i>0; i--);    // Retardo software
    }
}
```

Figura 1

1. Crear un proyecto nuevo vacío de nombre `lab1-blink` según se explica al final del documento “Guía de instalación Code Composer Studio v8” pero en la carpeta test/blink.
2. Crear un nuevo archivo fuente en el proyecto (`main.c`), y copiar el código anterior.[1]
3. Identificar las carpetas y los archivos creados para el proyecto (carpeta donde CCS ubica los proyectos) con un administrador o navegador de archivos y en el propio CCS antes de compilar.
4. Compilar el proyecto e identificar los archivos creados (Menú: Project > Build Project).
5. Verificar el correcto funcionamiento de la aplicación iniciando la depuración o debug (Menú: Run > Debug) y luego seleccionar ejecutar (Menú: Run > Resume).
6. Para finalizar crear un tag en el repositorio de proyectos de nombre `lab1-blink.`

Parte 3) Módulo Timer

Se pide Implementar el código del módulo timer (`timer.h` y `timer.c`) descripto a continuación, prestando especial atención a las técnicas de modularización.

El módulo de manejo del tiempo lleva la cuenta de las horas del día, minutos, segundos y milisegundos. El tiempo será incrementado usando un periférico del microcontrolador que implementa un temporizador de hardware (timer), aunque en esta versión no será implementado. En próximos laboratorios, la función que incrementa el tiempo será llamada desde la rutina de servicio a la interrupción del timer. En esta instancia para su evaluación se llamará desde el main.

1 Si el proyecto se creó como “empty project with main” modificar el código de `main.c` provisto.

Requerimientos:

1. La interfaz pública incluye funciones para:
    a) inicializar el reloj en un cierto tiempo
    b) incrementar el tiempo en 250 ms
    c) devolver el tiempo actual
    Observación: para determinar la firma de las funciones ver el uso del módulo en la Figura 2.
2. Para representar el tiempo definir un nuevo tipo utilizando una estructura para guardar: horas, minutos, segundos, milisegundos. La variable donde se guarda el tiempo actual no será accesible fuera del módulo.
3. Las funciones auxiliares, si las hubiera, deberán ser privadas al módulo.

El código fuente puede ser compilado usando el IDE CCS o GCC.

Se deberá tener previsto la posibilidad de mostrar el repositorio del grupo con la estructura de directorios conteniendo el código pedido. Para identificar este punto en la historia del repositorio crear un tag de nombre `lab1-parte3`. Verificar que los cambios son visibles para todos los miembros del proyecto.

Se diseñará y realizará una demo de validación del timer compilando una aplicación de prueba similar al código provisto en la Figura 2.

```
tiempo_t t_inicial = {23, 59, 59, 500};
tiempo_t t1;
tiempo_t t2;

int main(void)
{
	tiempo_t t_local;

	WDTCTL = WDTPW | WDTHOLD;	// Detiene el watchdog timer

	set_time(t_inicial);
	get_time(&t1);

	inc_time();
	inc_time();
	inc_time();
	inc_time();

	get_time(&t2);
	get_time(&t_local);

	return 0;
}
```

Figura 2

Compilación (compiler/linker)

1. Crear un proyecto nuevo vacío de nombre lab1 en la carpeta test/lab1 creada previamente siguiendo la guía.
2. Guardar los archivos `timer.h` y `timer.c` en los directorios `/src e /include` según corresponda.
3. Copiar el fragmento de código de la Figura 2 en un archivo de nombre `main.c` y modifíquelo de forma de que incluya el header del módulo timer.
4. Compilar corrigiendo los eventuales errores.
5. Deducir a partir de la inclusión del archivo de encabezado `<msp430.h>` y siguiendo las dependencias cómo se definen y mapean en memoria los registros del microcontrolador.
6. Inspeccionar el reporte del linker (archivo .map) identificando los diferentes segmentos y los objetos que contienen (funciones, variables, etc.). Calcular el tamaño a ocupar por la Flash y la RAM.
7. Verificar que los resultados anteriores concuerdan con los obtenidos por Memory Allocation (Menú: View > Memory Allocation)

Depurado (debugging)

1. Iniciar el depurador o debugger.
2. Correr paso a paso utilizando todas formas posibles (Step Over, Step Into, Run-to, etc.) y deduzca la utilidad de cada una (Menú: Run > ...).

3. Asociar el código en C con el código assembler generado (Menú: View > Disassembly).
4. Experimentar utilizando diferentes vistas para inspeccionar (Menú: View):
   - Variables (locales)
   - Expressions (locales, globales, estáticas, expresiones válidas de C, registros)
   - Register (registros del procesador)
   - Otros (Module, etc.)
5. Agregar un breakpoint para detener la ejecución del programa, por ejemplo, al inicio de la función para incrementar el timer.

Variables y Funciones

1. Observar (ver código assembler) cómo accede a las variables según su tipo de almacenamiento (estático y automático).
2. Repetir cambiando alguna variable por un puntero.
3. Agregar el modificador `const` a la variable `t_inicial` y observar los cambios, por ejemplo inspeccionando en el .map, o a través de los detalles dados por Memory Allocation.
4. Correr paso a paso para inspeccionar el stack antes, durante y después de la llamada a la función (ver ventana Debug).
5. Deducir el uso del Stack y la convención de llamadas a función usada por el compilador.
6. Experimentar la ejecución introduciendo *breakpoints*.
7. Para finalizar realizar los últimos cambios a los archivos fuentes y crear un tag en el repositorio de proyectos de nombre `lab1.`

Parte 4) Conversión de tipos

El MSP430G2553 tiene un sensor de temperatura integrado que da el valor medido mediante un entero de 16 bits, que llamaremos `datotemp`. Según los datos de calibración de fábrica, la relación entre la temperatura medida (en °C) y esa indicación está dada por

```
temperatura = datotemp * 0.413 - 277.75
```

1. Crear un proyecto nuevo vacío nombre `lab1-test-temp` en la carpeta test/test-temp.
2. Implementar una función que realice ese cálculo y un nuevo main.c como programa principal para probar esa función. La función recibirá como entradas el valor de `datotemp` y devolverá el resultado en un float. Para probar la función usar valores de `datotemp` entre 700 y 800.
3. Inspeccionar el código assembler generado y observar las conversiones de tipo que se realizan para obtener el resultado.
4. Determinar cuántos ciclos de reloj demora en su ejecución.
5. Proponer una función alternativa que realice todas las operaciones entre enteros sin perder precisión.
6. Repetir los pasos 3 y 4 con la función alternativa.
7. Para finalizar esta parte crear un tag en el repositorio de proyectos de nombre `lab1-test-temp`.

Tenga en cuenta que el microcontrolador (MSP430G2553) no tiene unidad de punto flotante y tampoco multiplicador hardware.

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

Enlaces

6. Launchpad MSP-EXP430G2: http://www.ti.com/tool/msp-exp430g2et
7. MSP430G2553: http://www.ti.com/product/MSP430G2553
8. Code Composer Studio: http://www.ti.com/tool/ccstudio

9. Code Composer Studio para MSP430 :http://www.ti.com/tool/ccstudio-msp
10. TI Clouds tools: http://dev.ti.com | TI Resource Explorer: http://dev.ti.com/tirex/