Seminario de Electrónica: Sistemas Embebidos - Trabajo Práctico N° 2

LPC43xx Entradas y Salidas (Digitales) de Propósito General (GPIO)

Objetivo:

- **Uso del IDE** (edición, compilación y depuración de programas)
- **Uso de GPIO** (manejo de Salidas y de Entradas Digitales)
- **Documentar lo que se solicita en c/ítems**

Referencias (descargar del Campus Vitual del curso a fin de usarlas durante la realización del TP)**:**

- **LPCXpresso-Intro:** http://campus.fi.uba.ar/pluginfile.php/155949/mod_resource/content/4/Sistemas_Embebidos-2016_2doC-LPCXpresso-Intro-Cruz.pdf
- **LPCXpresso-Salidas:** http://campus.fi.uba.ar/pluginfile.php/156011/mod_resource/content/4/Sistemas_Embebidos-2016_2doC-LPCXpresso-Salidas-Cruz.pdf
- **LPCXpresso-Entradas:** http://campus.fi.uba.ar/pluginfile.php/156013/mod_resource/content/4/Sistemas_Embebidos-2016_2doC-LPCXpresso-Entradas-Cruz.pdf
- **LPCXpresso-Systick:** http://campus.fi.uba.ar/pluginfile.php/156031/mod_resource/content/5/Sistemas_Embebidos-2016_2doC-LPCXpresso-Systick-Cruz.pdf
- **LPC435X_3X_2X_1X Product Data Sheet:** http://campus.fi.uba.ar/mod/resource/view.php?id=28519
- **LPC43XX User Manual** (Chapter 1, 18 & 19)**:** http://campus.fi.uba.ar/mod/resource/view.php?id=77765
- **EDU-CIAA-NXP (web site):** http://proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:edu-ciaa:edu-ciaa-nxp
- **EDU-CIAA-NXP (esquemático):** http://www.proyecto-ciaa.com.ar/devwiki/lib/exe/fetch.php?media=desarrollo:edu-ciaa:edu-ciaa-nxp:edu-ciaa-nxp_color.pdf
- **EDU-CIAA-NXP (pinout):** http://proyecto-ciaa.com.ar/devwiki/lib/exe/fetch.php?media=desarrollo:edu-ciaa:edu-ciaa-nxp_pinout_a4_v4r2_es.pdf

1. Uso del **IDE** (Integrated Development Environment) **GNU MCU Eclipse** (p/Linux o p/Windows)
   a. Previo a éste TP ya se Descargó, Instaló, Ejecutó, Configuró y Licenció todo lo necesario para desarrollar éste TP
      i. Instaló **CIAA-LAUCHER**, ejecutó **GNU MCU Eclipse**, agregó **firmware_v3**, configuró Debug para EDU-CIAA-NXP, agregó el plug-in de **eGIT**, agregó **Yakindu StateChart Tools** (**STC**), gestionó la licencia respectiva y al recibirla la cargó

      ii. Antes de ejecutar asegúrese tener conectada la placa **EDU-CIAA-NXP** a su PC (recuerde conectarla **siempre al mismo puerto USB**) a través de la interfaz **Debug**
         1. Seleccionar el Workspace: **C:\CIAA\CIAA_Software_1.1-Win\workspaces\eclipse-ws**
         2. En el archivo **program.mk** podrá configurar el **programa** en el que se trabajara:
            PROGRAM_PATH = **examples/c**
            PROGRAM_NAME = **app**
         3. En el archivo **board.mk** podrá configurar la **placa** a utilizar:
            BOARD = **edu_ciaa_nxp**
         4. Verifique tener en carpetas **examples/c/app/**, **examples/c/app/inc/** y **examples/c/app/src/** los archivos:
            a. **config.mk Compile options & Libraries** file
            b. **inc**, **out** y **src** folder
            c. **app.h** y **app.c Source & Include** files
         5. Para **Editar** el **fuente**: Doble clic sobre **app.c**
         6. Para **Compilar** el **fuente**: Clic derecho sobre **firmware_v3** -> **Clean Project**
            Clic derecho sobre **firmware_v3** -> **Build Project**
         7. Para **Debug** el **fuente**: Clic derecho sobre **firmware_v3** -> **Debug As** -> **2 Local C/C++ Application**
         8. **Identificar** la estructura de **archivos**, su tipo/contenido (especialmente **readme.txt**) de c/proyecto **examples/c/app**
         9. **Ejecutar** la **secuencia de comandos**: Clean **firmware_v3** -> Build **firmware_v3** -> Debug **firmware_v3** -> Ejecutar **app** (ejemplo de aplicación)
            a. Completo (**Resume**), detener (**Suspend**) y resetear (**Restart**)
            b. Por etapas colocando **breakpoints** (Resume)
            c. Por línea de código (**Step Into**, **Step Over**, **Step Return**)
            d. Recuerde siempre abandonar Debug (**Terminate**) antes de Editar o Compilar algún archivo, o Abandonar el IDE (**Exit**)

b. Migrar el proyecto **examples/c/app** (parpadeo del **LEDs** c/**sAPI**) a: **examples/c/projects/TP2**
   i. **Documentar** los pasos a seguir para concretar una **migración exitosa**

   ii. Identificar funciones de librería **sAPI** útiles para el **parpadeo de un led** y **printf** (**UART_DEBUG**)
      1. **Identificar** la secuencia de **funciones** invocadas durante la ejecución del ejemplo de aplicación, en qué archivo se encuentran, su descripción detallada, qué efecto tiene la aplicación sobre el hardware (identificar circuitos, puertos, pines, niveles, etc.) así como la interacción entre las mismas (tanto en **ResetISR()** como en **main()**)
      2. **Idem c** pero con **datos** (definiciones, constantes, variables, estructuras, etc.) (tanto en **ResetISR()** como en **main()**)

   iii. Para facilitar la corrección de su trabajo se le ha pedido previamente:
      1. **Ingresar** a https://github.com/
      2. **Crear** una **cuenta** si no dispone de una
      3. **Informar** su nombre de usuario al docente (para ser agregado al *team* correspondiente a su grupo, o lo que hayan recomendado los ayudantes)
      4. **Crear** un **repositorio** denominado **TP2**. (La URL del mismo será parecida a https://user...name/TP2, o lo que hayan recomendado los ayudantes)
      5. **Realizar** un **commit/push** inicial del **modelo** actual.
      6. De ahora en adelante, **actualizar** **su repositorio** mediante **commit**/**push**

2. El objetivo a continuación es crear las funciones **gpioInit**, **gpioWrite** y **gpioRead**, las cuales implementan completamente el uso de los GPIOs de la placa en una manera simple y general:

   a. **gpioInit**: Inicializa el pin utilizado como GPIO. En la familia de chips LPC43xx esta instrucción consiste en dos partes:
      i. Configurar eléctricamente el pin a utilizar a través del periférico System Control Unit (**SCU**).
      ii. Configurar el modo de uso del GPIO a través de sus propios registros en el memory mapping.

   b. **gpioWrite**: Escribe un valor binario en el pin del GPIO cuando está configurado como salida.

   c. **gpioRead**: Lee el valor del pin utilizado como GPIO

   d. **Documentar** los pasos a seguir para mantener en el archivo **app.c** los fuentes del TP1-1 y **TP1-2** (**compilación condicional**)

      i. Para **implementar** correctamente estas funciones, primero hay que hacer varias cosas:

         1. **Identificar** las estructuras que representan los periféricos **SCU** y **GPIO** y las **funciones** implementadas para manejarlas, provistos por el fabricante. En el caso de la EDU-CIAA, las librerías del chip **LPC4337** se encuentran en la carpeta **firmware_v3/libs/lpcopen/lpc_chip_43xx**

         2. **Enumerar** los diferentes tipos de **configuraciones** que puede adoptar el pin GPIO (**output**, **pullup**, etc.) a partir de la información provista por la hoja de datos, y crear un tipo enumerativo **gpioInit_t** que contenga los nombres de cada uno de ellos.

         3. **Crear** una **estructura** que contenga los siguientes campos:
            a. el **puerto** y el **pin** a configurar por el **SCU**,
            b. la **función** para que el **pin** anterior se **configure** como **GPIO**, y
            c. el **puerto** y el **pin** del **GPIO** utilizado

         4. Luego, **crear** un **vector global** de estas estructuras que contenga todas las posibles configuraciones que pueden adoptar los **GPIOs** que quiero configurar. Por ejemplo, si tengo que configurar un conjunto de leds (**LED1**, **LED2** y **LED3**), voy a crear un vector **gpioPinsInit** de (en este caso 3) estructuras que contengan la información de configuración de cada led. Es decir, que sí el **LED1** se conecta al **puerto 2**, **pin 10** del **SCU**, la **función** para configurarlo como GPIO tiene un valor igual a **0**, y el **GPIO** configurado es el **GPIO0[14]**, entonces la configuración correspondiente al **LED1** en este vector tiene la forma:

```
const conf_t gpioPinsInit[]{
        // ...
        { 2,10, 0, 0,14 },   // Configuración del LED1: { P2_10, 0, GPIO0[14] }
        // ...
}
```

donde **conf_t** es una estructura (la del punto 3) de la siguiente forma:

```
typedef struct {
    int8_t scu_port;
    int8_t scu_pin;
    int8_t func;
    int8_t gpio_port;
    int8_t gpio_pin;
} conf_t;
```

5. También resulta muy conveniente **crear** un tipo enumerativo **gpioMap_t** con todos los posibles **GPIOs** (en el ejemplo anterior serían **LED1**, **LED2**, **LED3**).

ii. Con todo esto, ya es posible hacer las funciones **gpioInit**, **gpioWrite** y **gpioRead**:

```
void gpioInit( gpioMap_t pin, gpioInit_t conf );
void gpioWrite( gpioMap_t pin, bool_t value );
void gpioRead( gpioMap_t pin );
```

Dentro de cada una de ellas se pretende que se utilicen las funciones de la librería Chip de **LPCOpen** analizadas en el punto **2.d.i.1.**

3. **Implementación** para el **chip utilizado en el TPF**.

a. Luego de haber completado los pasos del ítem 2, se pretende que estas funciones sirvan para la implementación del trabajo final. En el caso de no utilizar un chip de la familia **LPC43xx** o equivalentes, va a ser necesario buscar las funciones provistas por el fabricante correspondiente y realizar un trabajo similar al del apartado anterior.

b. Mediante **compilación condicional**, mantener en el archivo **app.c** los fuentes del TP1-1, TP1-2 y **TP1-3**