Introducción a LPCXpresso y repaso del lenguaje C

Seminario de Sistemas Embebidos - FIUBA

Alan Kharsansky

Agosto - 2011

Índice

Introducción

El LPCXpresso es un toolchain completo para evaluación y desarrollo con microcontroladores de NXP.

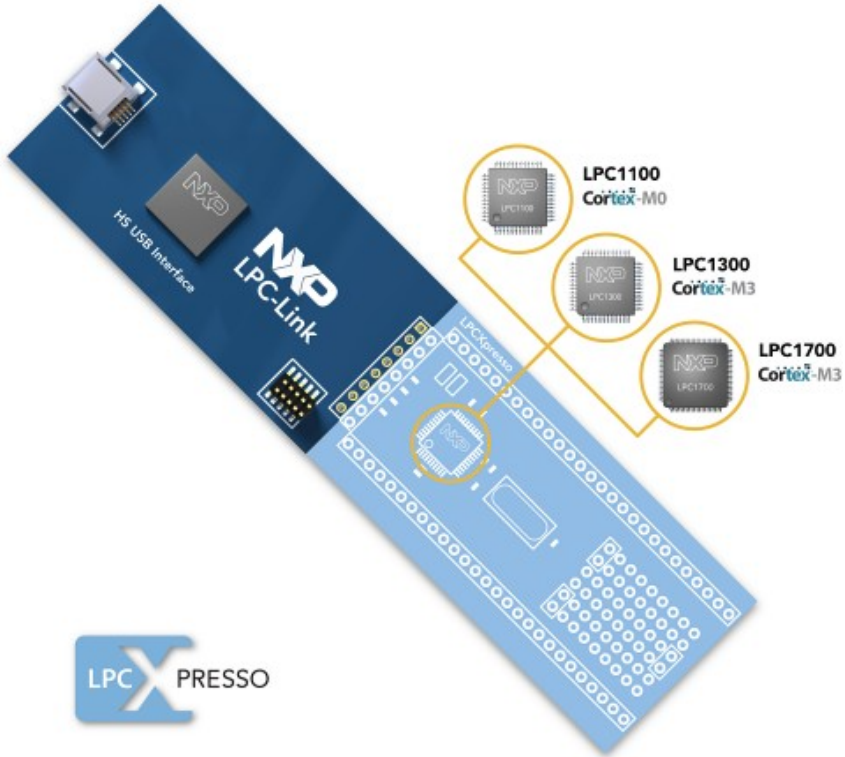
Esta compuesto por:

- LPCXpresso IDE y "development tools"
  - IDE basado en Eclipse
  - compiler y linker GNU
  - GDB debugger
- LPCXpresso target board (stick)
- BaseBoard o hardware adicional (opcional)

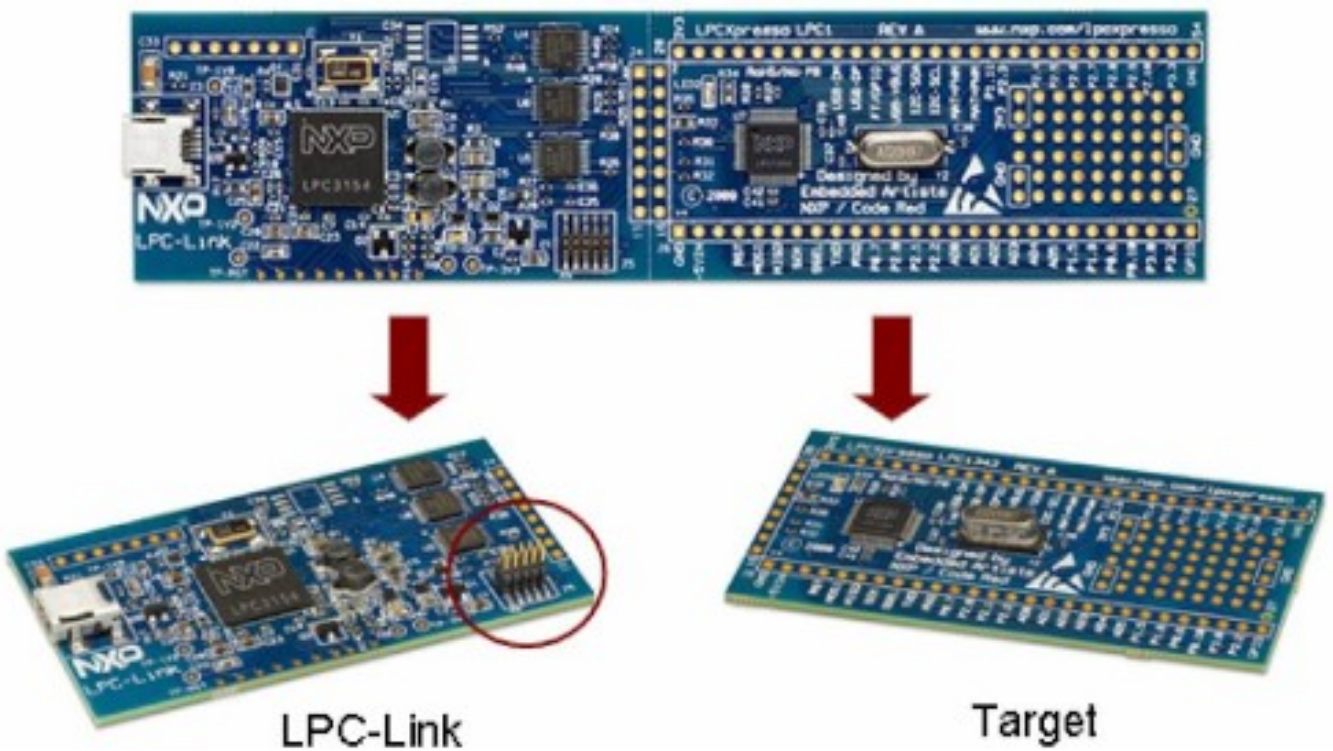
LPCXpresso target board

El target board es un una herramienta de desarrollo que incluye

- Un uC de las familias LPC1100, LPC1300 o LPC1700 junto con eletrónica mínima necesaria para su funcionamiento
- Un programador y debugger JTAG



LPCXpresso target board



LPCXpresso target board

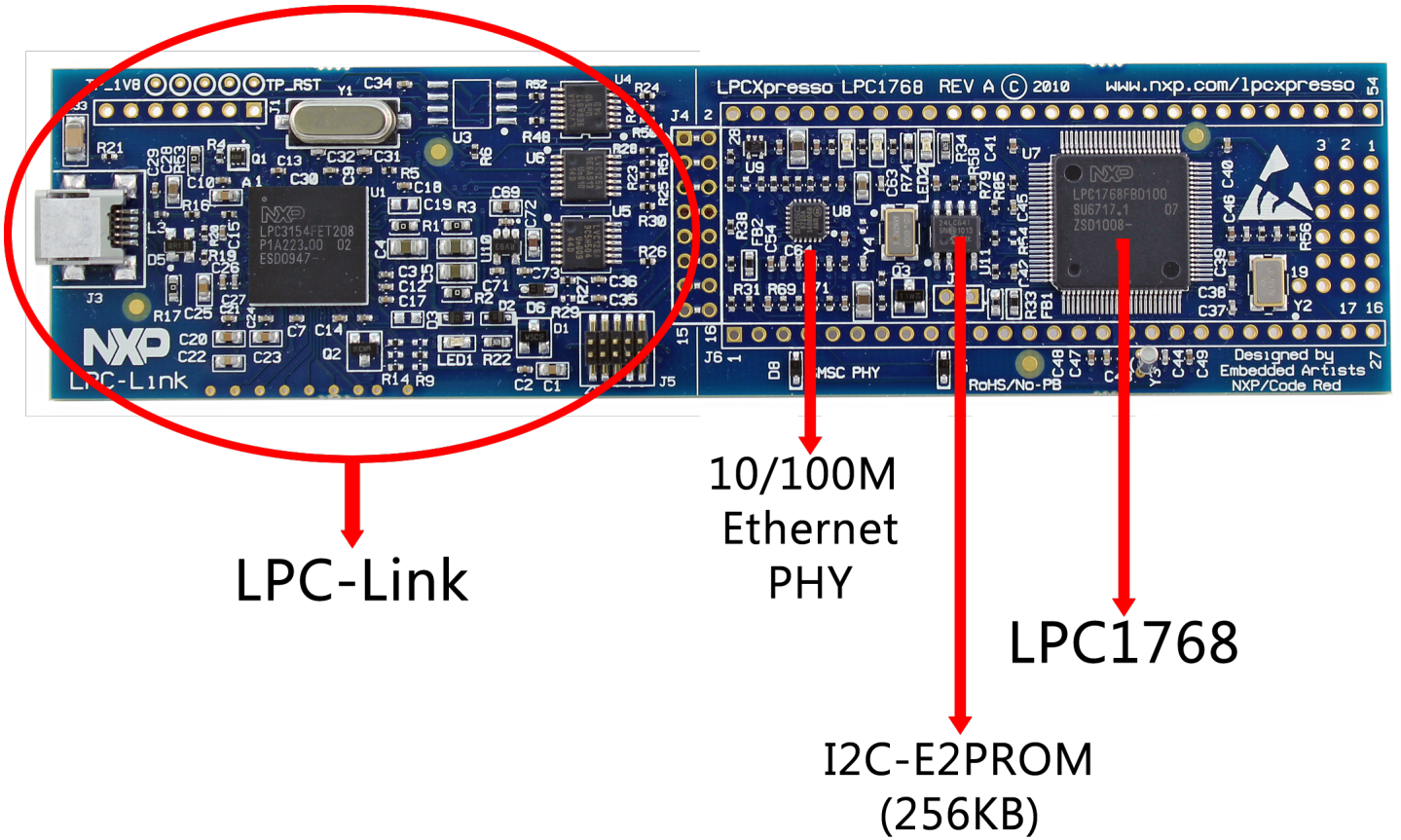
Del lado del target este incluye algunos periféricos básicos y se comercializan con diferentes micrcontroladores. Por el momento se encuentran disponibles:

- **LPC1114:** ARM Cortex-M0, 32KB flash, 4/8KB SRAM, 50 Mhz.
- **LPC1343:** ARM Cortex-M3, 32KB flash, 8K SRAM, USB, 72 Mhz.
- **LPC1769:** ARM Cortex-M3, 512KB flash, 64KB SRAM, Ethernet, USB On the go, 120 Mhz.

Nota: En su primera versión existió el LPC1768, muy similar al LPC1769

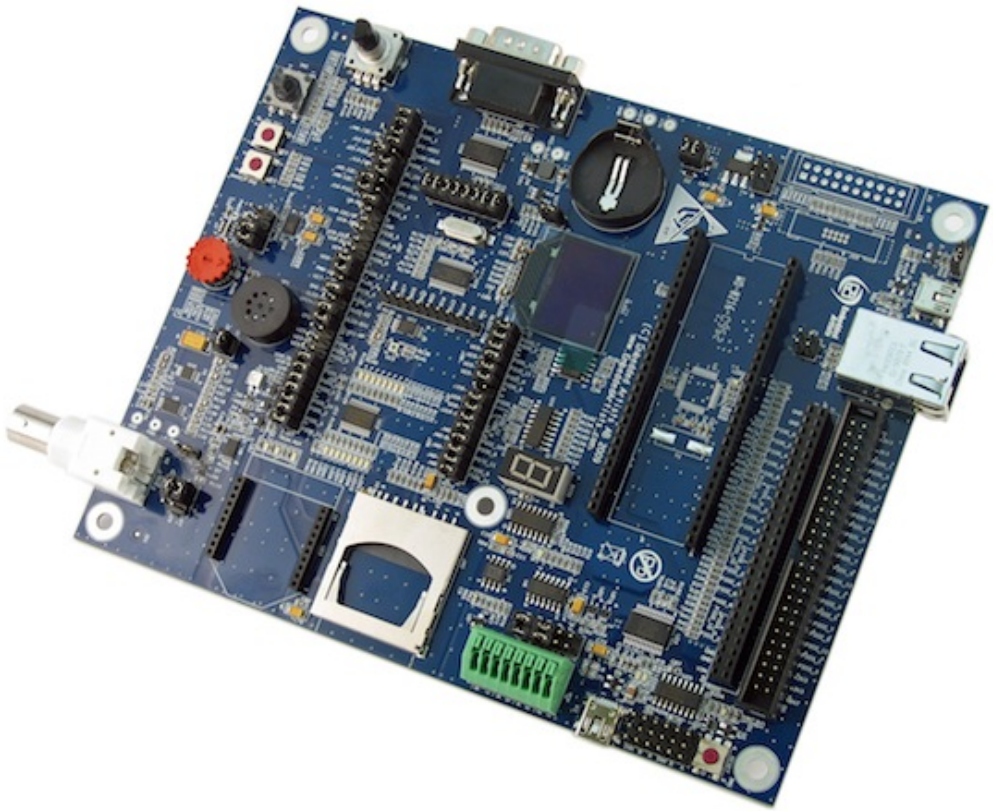
LPCXpresso target board

En este curso vamos a utilizar el target que viene con el LPC1768/9.

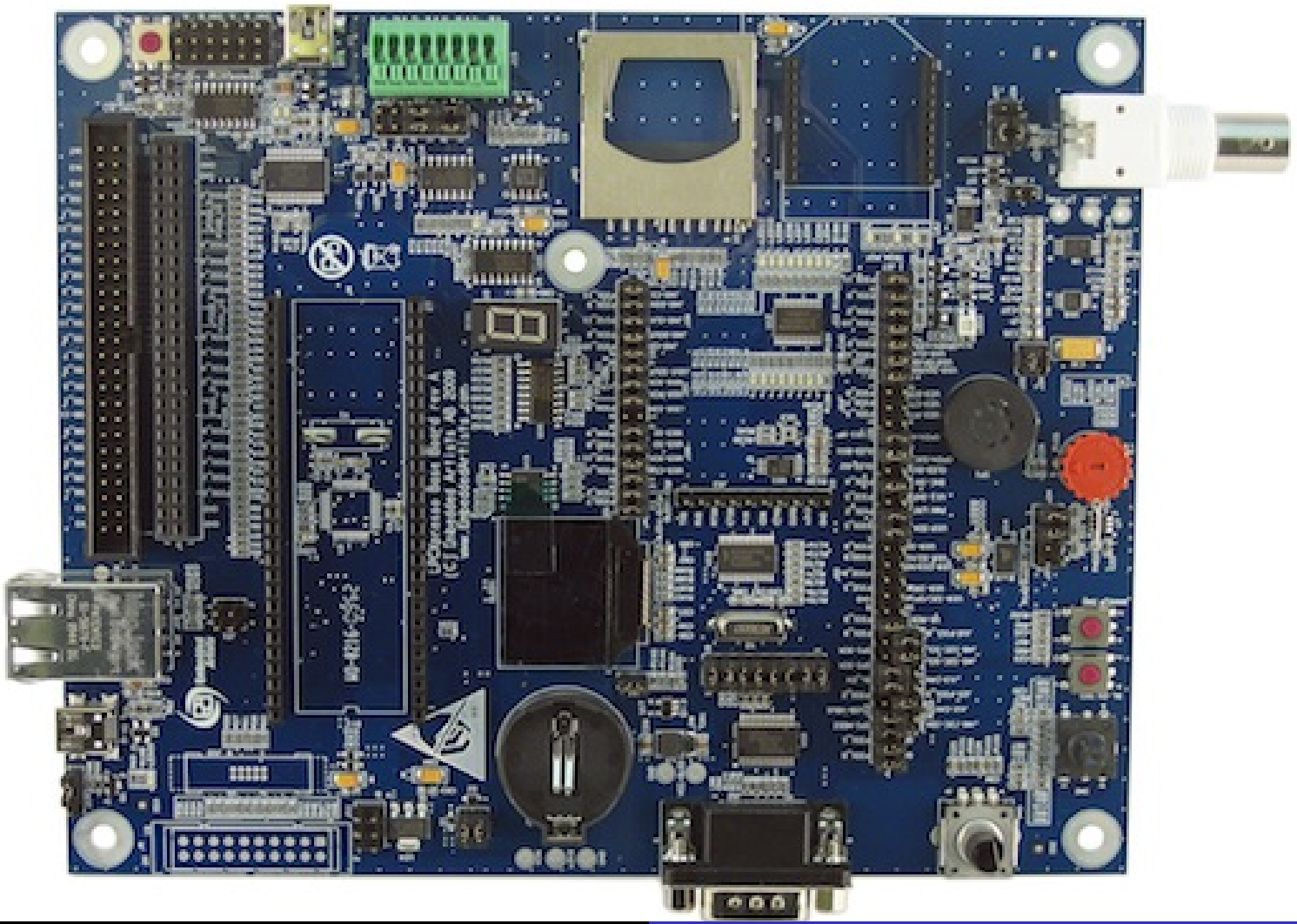


LPCXpresso BaseBoard

El BeseBoard fue diseñado por Embedded Artists Junto con NXP. Permite conectarle un MBed o un LPCXpresso Target.



LPCXpresso BaseBoard



LPCXpresso BaseBoard

La placa contiene periféricos para desarrollo y experimentación:

Generales:

- Socket for LPCXpresso and mbed module
- 50 pin expansion dual row pin/header list connector
- Battery powering (small coin battery)
- USB interface
- Reset pushbutton

Digitales:

- RGB-LED (can be PWM controlled)
- 5-key joystick switch
- 2 pushbuttons, one for activating bootloader
- Rotary switch with quadrature encoding (timer capture)
- Temperature sensor with PWM output (timer capture)

Analógicos:

- Trimming potentiometer input (analog input)
- PWM to analog LP-filtering (PWM output and analog input)
- Speaker output (PWM output)
- Oscilloscope probe inout stage

Serial - UART:

- USB-to-serial bridge, with automatic ISP activation
- RS422/485 interface
- Interface socket for XBee RF-module

LPCXpresso BaseBoard

Continuación:

Serial - SPI:

- Shift register driving 7-segment LED
- SD/MMC memory card interface
- Dataflash SPI-NOR flash

Serial - I2C:

- PCA9532 port expander connected to 16 LEDs
- 8kbit E2PROM
- MMA7455L accelerometer with I2C interface
- Light sensor

Serial - I2C/SPI

- SC16IS752 - I2C/SPI to 2xUART bridge; connected to RS232 full-modem interface and one expansion UART
- 96x64 pixel white OLED (alternative I2C/SPI interface)

Extras

- CAN bus interface (can be simulated with LPCXpresso LPC1114/LPC1343)
- Ethernet RJ45 connector with integrated magnetic

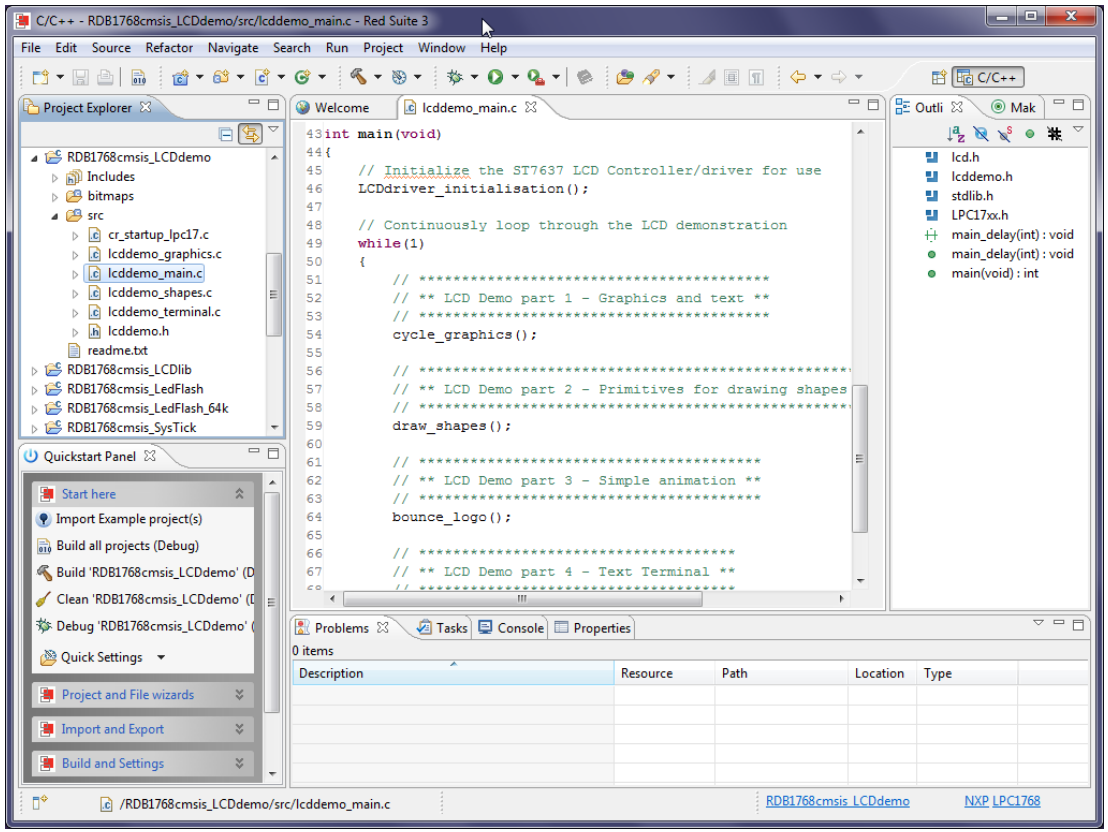
Información adicional

Se recomienda revisar los siguientes documentos:

- LPC1768 User Manual ▸ Ver
- LPC1768 Datasheet ▸ Ver
- LPCXpresso 1768 Target board: Esquemáticos ▸ Ver
- LPCXpresso BaseBoard: Guía de usuario ▸ Ver
- LPCXpresso BaseBoard: Esquemáticos ▸ Ver

LPCXpresso IDE (Eclipse)

El LPCXpresso IDE fue desarrollado por CodeRed junto a NXP. El mismo incluye un entorno de Eclipse específicamente adaptado para interactuar con el target board.



Conceptos básicos

Eclipse utiliza algunos conceptos que no siempre son comunes a otros entornos de desarrollo por lo que vamos a ver algunos de ellos.

Workspace Es el contenedor de nuestros proyectos. Estos proyectos pueden ser **aplicaciones** y/o **bibliotecas**. También almacena todas las configuraciones del entorno por lo que se puede mover muy fácilmente de computadora en computadora.

Proyecto Este puede ser de dos tipos. Biblioteca estática o una aplicación ejecutable. Contiene archivos de código fuente (.c), encabezados (.h) y cualquier otro archivo que se desee.

En general utilizaremos el workspace para intercambiar proyectos (en el sentido convencional de la palabra) ya que el mismo incluíra todas las bibliotecas necesarias.

Tipos de proyectos

Los proyectos pueden ser de dos tipos:

- **Aplicaciones:** Se compilan y se pueden descargar directamente al target.
- **Bibliotecas estáticas:** Se pueden compilar, pero para usarlas, un proyecto de tipo aplicación debe hacer llamadas a las funciones que este contiene. Es decir, no puede tener un **main()**. Este tipo de proyectos no se puede descargar por si solo al microcontrolador.

Tipos de proyectos - Ejemplo

Para ejemplificar pensemos en un ejemplo de un sistema embebido: un reproductor de MP3. Este podría estar compuesto por una memoria SD, una pantalla táctil y un decodificaro de MP3. Todos estos periféricos están controlados por un microcontrolador, por ejemplo un LPC1768.

Tipos de proyectos - Ejemplo

Las bibliotecas estáticas que podríamos tener son:

- Para el manejo de una memoria SD
- Para el manejo del display
- Para el manejo del touchscreen
- Para el manejo del decodificador

Tipos de proyectos - Ejemplo

Nuestra aplicación en sí sería la que tendrá el programa principal y desde donde se ejecutara nuestro programa. Pero las bibliotecas nos proveen funciones para el manejo de estos periféricos. Puede darse el caso en el que el fabricante nos provea de estas bibliotecas ya compiladas. En ese caso solo debemos conocer los **prototipos** de las funciones.

Estos prototipos se agregan a los archivos fuente utilizando la directiva #include y generalmente de archivos con extensión .h.

Repaso de lenguaje C

Tenemos que recordar que vamos a estar programando en el lenguaje C estandar y que nuestra plataforma donde se ejecutará el código es distinta a donde lo compilaremos. Es por eso que decimos que estamos usando un:

Cross Compiler

Proceso de creación de software en C

Al programar en C, generalmente el proceso que se sigue es el siguiente:

Edición de código

En C tenemos dos tipos de archivos,

.c Archivos compilables. Sólo puede haber uno por ejecutable que contenga un **main**. Cada programa puede utilizar tantos como se requiera.

.h Archivos no compilables. Suelen incluirse en los .c para usar definiciones y prototipos.

Edición de código - un ejemplo

```c
#include "leds.h"
#include "delay.h"

int main(void){
 int a, b;
 a = 10000;
 b = 2*a;

 while(1){
  ledOn(0,22);
  delay(b);
  ledOff(0,22);
  delay(b);
 }

 return 1;
}
```

app.c

```c
void delay(int t){
 int i;
 for (i=0;i<t*10000;i++){
  //No hago nada
 }
}
```

delay.c

```c
void delay(int t);
```

delay.h

Compilación

Al querer generar un programa ejecutable. Primero necesitaremos ejecutar el compilador. Este antes de empezar a compilar, ejecuta el **pre-procesador**.

Algunas de las tareas que este realiza son:

- Inclusión de archivos.
- Compilación condicional.
- Interpretación de macros y definiciones.
- Comprobación de errores de sintaxis.

Compilación

Ahora el compilador es quien se encarga de generar código objeto.

Incluir archivos .h (o prototipos de funciones) en el archivo .c le indican al compilador que las funciones existen, tienen un prototipo determinada y en algún momento serán linkeadas. Es decir que el código tendra llamadas a las funciones pero no su implementación.

En nuestro ejemplo, se deberá entonces ejecutar el compilador para compilar leds.c y app.c El resultado del mismo son archivos .o:

- leds.o
- app.o

Linkeo

Cuando se tienen todos los archivos .o, se puede proceder al **linkeo** o unión de los archivos para generar un archivo ejecutable. La salida generalmente es un archivo **.axf** o **.elf** que contienen el código ejecutable. Luego se puede pasar a un archivo **.hex**

Zonas de memoria

Es importante tener en cuenta en donde se va a ubicar cada parte del código generado. Se tienen los sectores de memoria:

.text Código ejecutable

.data Variables globales read-only y read-write inicializadas

.bss Variables sin inicializar (son inicializadas en 0 automticamente)

Notas: Es común en microcontroladores utilizar el segmento text para guardar variables const. Las variables locales de una función se alojan en el stack. La memoria dinámica, en el heap.

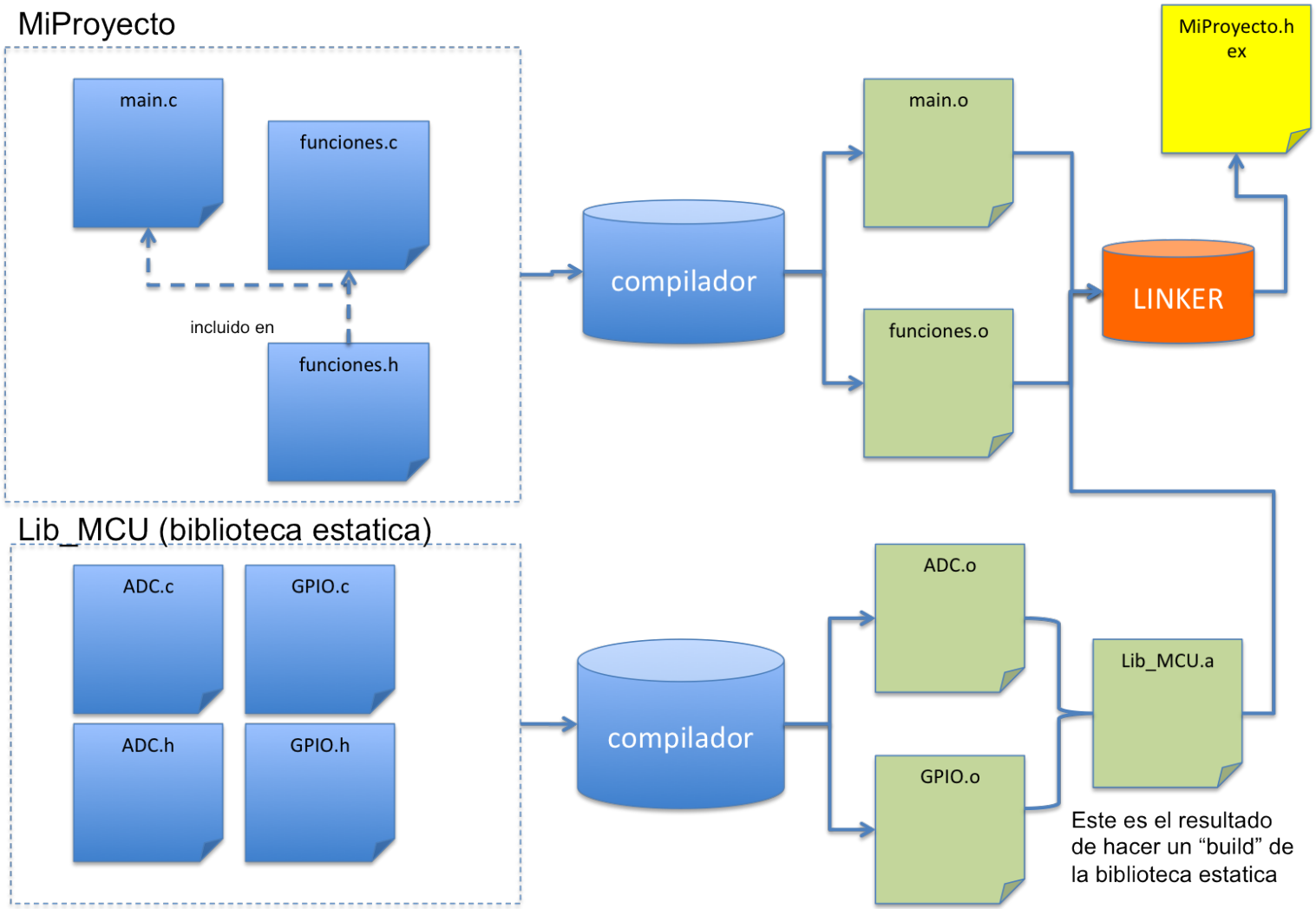
Linkeo - Linker Script

El linker script es un archivo de configuración que le indica al linker (ld) donde ubicar cada segmento de memoria.
Por defaut, en el IDE de LPCXpresso los segmentos .data y .bss se alojan en el primer banco de RAM aunque es posible ubicarlos en uno diferente.

Resumen

Cuando utilizamos Eclipse (en este caso el CodeRed) podemos, entonces, tener diferentes proyectos dentro de un workspace y luego linkearlos para producir una salida final.

Resumen


MiProyecto
main.c
funciones.c
incluido en
funciones.h
compilador
main.o
funciones.o
LINKER
MiProyecto.hex
Lib_MCU (biblioteca estatica)
ADC.c
GPIO.c
ADC.h
GPIO.h
compilador
ADC.o
GPIO.o
Lib_MCU.a
Este es el resultado de hacer un “build” de la biblioteca estatica

Resumen

Mini Tutorial para realizar proyectos con CodeRed

▸ Descargar

Diferencias

Al programar un microcontrolador tendremos que tener algunas consideraciones. Al no tener control (preciso) de las lineas de Assembly ni de donde se ubicará nuestro código en memoria, debemos usar nombres de funciones que el compilador luego ubicará en lugares especiales.

Punto de entrada

El punto de entrada que antes lo asociabamos a una posición de memoria, por ejemplo 0x0000, ahora lo debemos utilizar con un nombre de función especifico.

Para el LPCXpresso esta función se llama **Reset_Handler()**.

También existen otros para diferentes interrupciones y handlers

Punto de salida

A diferencia de un programa de computadora, los programas de los microcontroladores no pueden retornar del main. Esto se debe a que no fue llamado por ningún sistema operativo (por ejemplo). Por eso generalmente utilizamos la siguiente estructura:

```
int main(void){
  // Declaracion de variables
  // Rutinas de inicializacion

  while(1){
    // Programa principal
    // .......
    // .......
  }

  return 0;
}
```

Interrupciones

Las interrupciones en el microcontrolador, al igual que el main, se asocian a una posición de memoria. Solamente utilizamos su nombre. Por ejemplo:

```
void main(void){
 // Declaracion de variables
 // Rutinas de inicializacion

 while(1){
  // Programa principal
  // .......
  // .......
 }

}

void IRQ_Handler(void){
 // Servicio de interrupcion
}
```

El compilador y/o el microcontrolador se encargan de poner el código necesario para guardar y devolver el contexto.

Sistema de control de versiones (VCS)

Un sistema de control de versiones es una herramienta que permite administrar un código que va evolucionando, que debe ser compartido por más de una persona y que esta sujeto a constantes modificaciones por parte de los programadores de manera eficiente.

No más:

main_final_final_superfinal8_elposta.c

VCSs distribuido vs centralizado

Hay dos topologías de sistemas de control de versiones:

- **Centralizado**: existe un repositorio central de código. Todos los usuarios se conectan a este para trabajar sobre el proyecto.
- **Distribuido**: No existe un repositorio central de código. Todos los usuarios manejan copias locales del repositorio completo.

A veces se suele usar un sistema híbrido.

VCSs populares

Hoy en dia los más populares son:

- **Centralizado**: Subversion
- **Distribuido**: Mercurial (Hg) y Git

Herramientas para VCS

Durante el transcurso del seminario utilizaremos Mercurial como sistema de control de versiones y un repositorio central en BitBucket.org.
Para utilizar el VCS dentro de nuestro entorno de trabajo, utilizaremos un plugin para eclipse llamado MercurialEclipse que nos permitira integrar esta tecnología directamente en el entorno del LPCXpresso.

HgInit.com es un muy buen tutorial para comenzar ▸ Ver

Mercurial

Mercurial es un sistema distribuido muy simple de usar. Los conceptos más importantes que tenemos que saber para comenzar son:

- Clone
- Push
- Pull
- Commit

Commit

Cuando queremos utilizar Mercurial debemos crear un repositorio en alguna carpeta. Luego agregar los archivos que queremos administrar (lo ideal: la mayoría).

Un commit es la manera de guardar los cambios que se hicieron. Estos son incrementales por lo que no se copia cada versión sino solo las modificaciones. Estos son SIEMPRE en el repositorio local.

Clone, push y pull

El comando **clone** permite clonar un repositorio existente. Por ejemplo el central en bitBucket.org a mi computadora local.

El comando **pull** permite descargar los cambios de un repositorio ya clonado

El comando **push** permite guardar los cambios en un repositorio que no es el local.

Actividad Nro 1 - Blinky Led

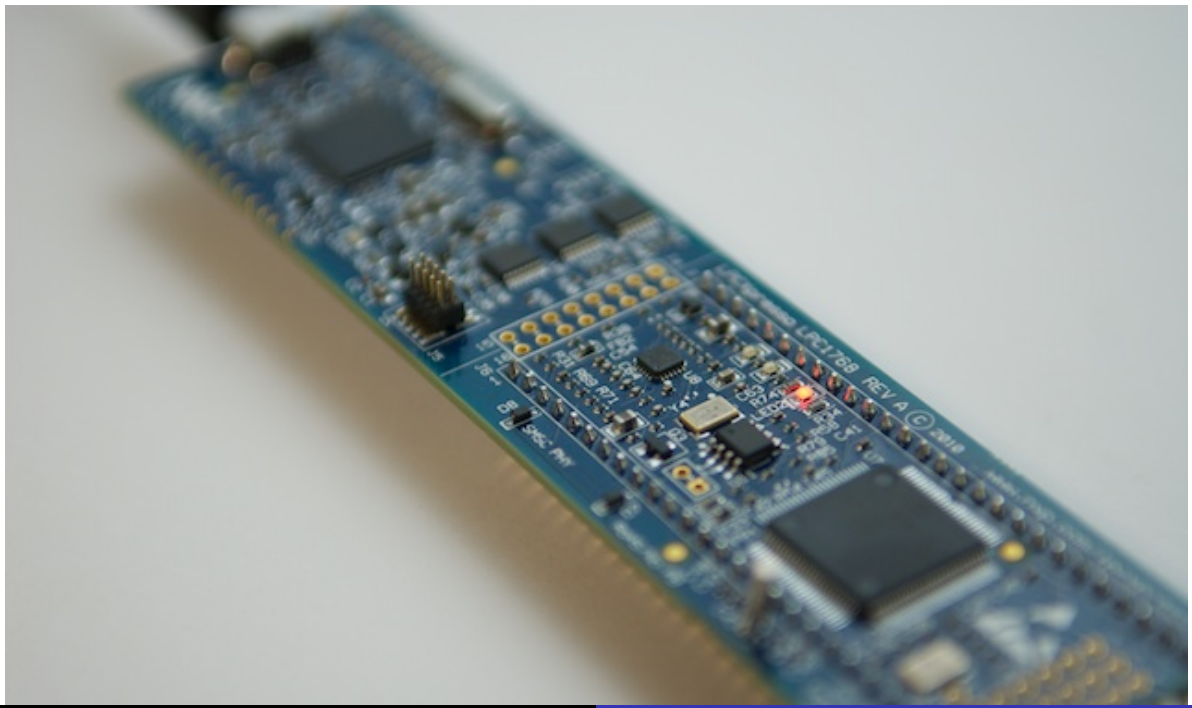
Blinky Led

Objetivo

Familiarizarse con el entorno y lograr hace nuestro primer:

Hello World!

En microcontroladores, esto equivale a hacer parpadear un LED conectado a algun pin de I/O del microcontrolador.



Paso 1

Debemos clonar el repositorio de la materia. Este nos descargara un workspace que contiene bibliotecas estáticas y recursos adicionales:

- **Lib_CMSIS_and_Drivers** CMSIS provisto por NXP y Drivers básicos para perifericos
- **FreeRTOS-Library** Kernel del FreeRTOS
- **Documentación**
- **Ejemplos**

Paso 1

Crearemos un nuevo proyecto. Elegir que el target es un NXP LPC17XX C project. Poner un nombre, por ejemplo: "Blinky" y elegir el microcontrolador LPC1768/9. Luego presionar finish.

Debemos configurar al proyecto para que sepa los paths con los que trabajaremos. También debemos definir el macro global __USE_CMSIS

Paso 2

Ahora que ya tenemos configurado el proyecto, podemos empezar a escribir nuestro código. Para ello lo primero que debemos hacer es configurar el pin como salida digital. Esto se debe a que cada pin del LPC1768 puede ser utilizado para diferentes funciones.

Por ahora para poder configurar perifericos, accederemos a traves de los registros especiales. El Led que utilizaremos está ubicado en el stick y conectado al puerto 0 y pin 22 (GPIO0_22).

Paso 2

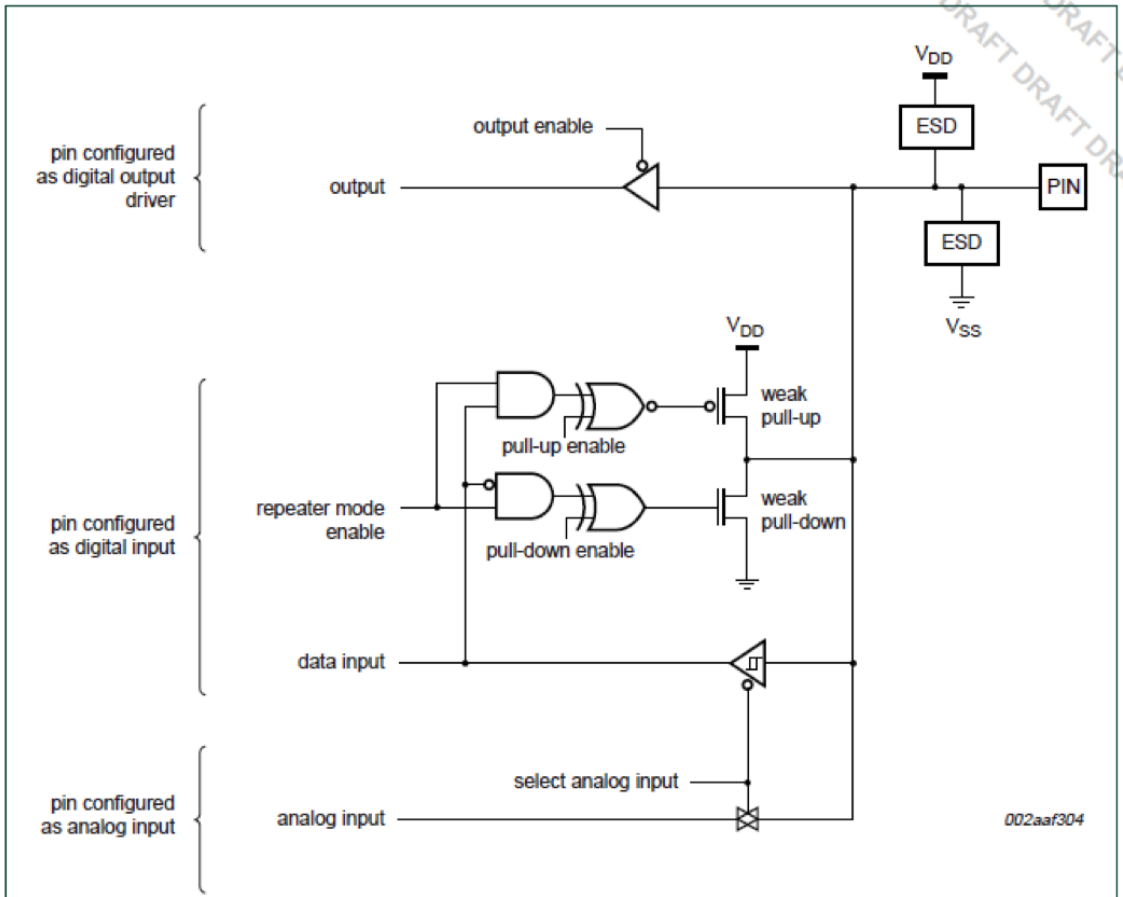


Fig 5. Standard I/O pin configuration

Paso 2

Para poder acceder a estos registros, NXP nos provee de archivos para cada uC con definiciones y estructuras en las que se incluyen todos los registros de cada periférico. En este caso como usamos el puerto 0, la estructura que utilizaremos será:

LPC_GPIO0

Paso 2

La estructura LPC_GPIO0 esta formado por los siguientes registros:

- FIODIR: Permite establecer la dirección del pin. 1 para salida, 0 para entrada
- FIOSET: Permite poner un 1 lógico en un pin.
- FIOCLEAR: Permite poner un 0 lógico en un pin.
- FIOPIN: Permite leer el estado de los pines
- FIOMAS: Establece una mascara para las acciones de PIN, SET o CLEAR

FIO: Fast I/O

Paso 2

Para setear la dirección de un pin utilizamos el siguiente código:

```
LPC_GPIO0->FIODIR |= (1<<22);
```

Para setear el bit o borrarlo, podemos hacer:

```
LPC_GPIO0->FIOSET = (1<<22);
LPC_GPIO0->FIOCLR = (1<<22);
```

Notar que SET y CLEAR no se usan con máscaras, ¿Por qué?

Ejercicio completo

El código completo queda entonces:

```
#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

int main(void) {

 LPC_GPIO0->FIODIR |= (1<<22);                    // Direccion

    while(1) {
       LPC_GPIO0->FIOSET = (1<<22);               // Prendo
       LPC_GPIO0->FIOCLR = (1<<22);               // Apago
    }

    return 0 ;
}
```

Adicional

El ejemplo anterior solo es visible si se debuggea el programa, ¿Por qué?

Como podríamos implementar un "delay" en nuestro código que nos permita ver parpadear el led (sin utilizar Timers).

Actividad Nro 2 - RGB Led

RGB Led driver para el BaseBoard

Objetivo

El objetivo de este ejercicio es poder crear una biblioteca estática de funciones y poder vincularla con nuestro proyecto. La misma deberá tener funciones para poder cambiar los colores del led RGB que incluye el BaseBoard.

Consigna

Se deberá crear una biblioteca estatica que permita manejar el Led RGB que trae el BaseBoard. El mismo se controla mediante 3 pines de salida:

- RED: GPIO2, pin 0
- GREEN: GPIO2, pin 1
- BLUE: GPIO0, 26

Ojo con los jumpers del BaseBoard. Chequear que esten conectados.

Consigna

Las funciones que debera tener esta biblioteca son:

- Inicializar el hardware
- Apagar todos los leds
- Prender los leds independientemente para mostrar los colores primarios
- Opcional: Hacer funciones para generar colores a partir de los primarios.

Consigna

La intención es lograr una biblioteca (o driver) totalmente separado de nuestro código para luego tener un main que sea similar al siguiente:

```
#include "RGBLed.h"

int main(void) {

    ledsInit();

    while(1) {
        redLedOn();
        ledsOff();
        greenLedOn();
        ledsOff();
        blueLedOn();
     }
     return 0 ;
}
```

Actividad Nro 3 - Fading Led (PWM)

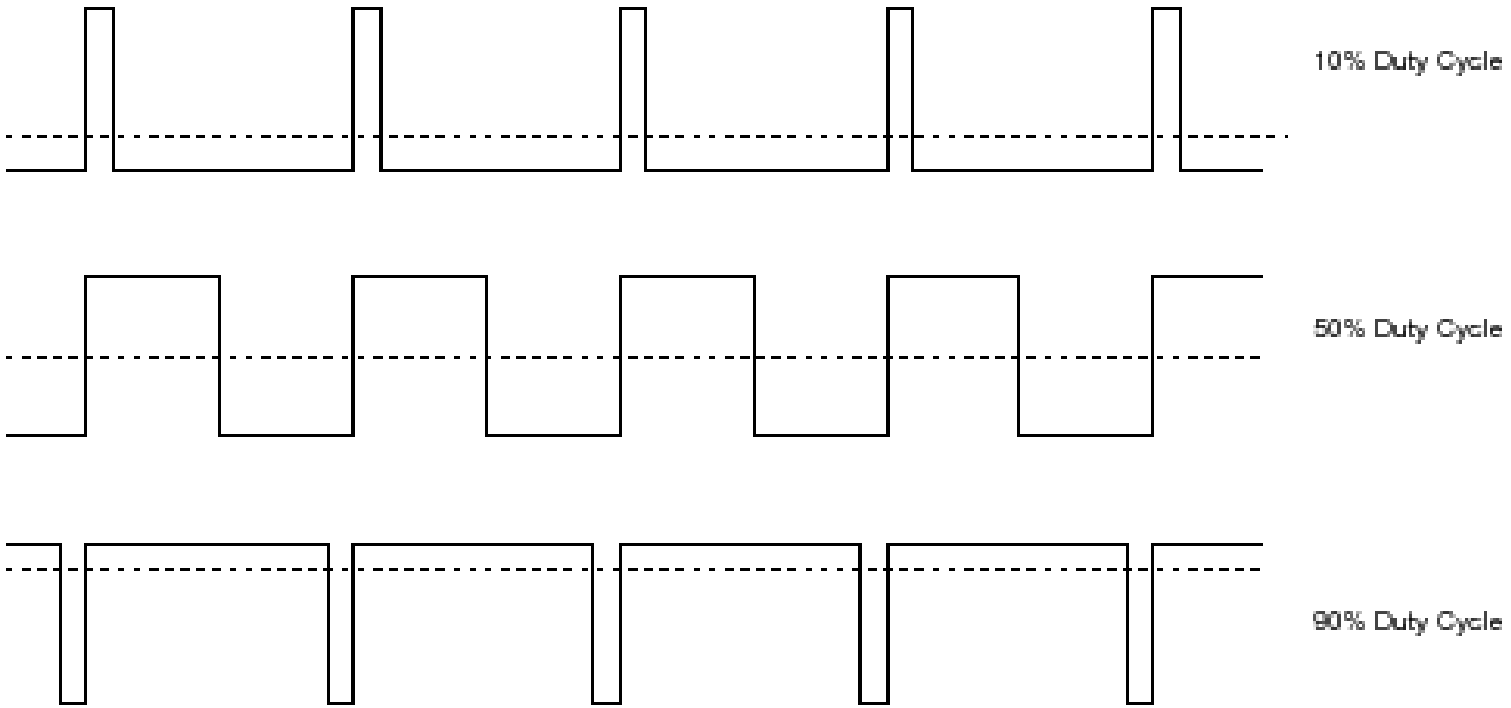
Fading LED

Consigna

El objetivo de esta actividad es hacer un programa que permita variar la intensidad de brillo de un Led. Para ello solamente podremos usar las funciones básicas C y las de entrada y salida vistas en la actividad Nro 1.

Se deberá diseñar entonces un programa que permita variar el periodo de actividad de un pulso cuadrado (PWM) **por software**.

Actividad Nro 3 - Fading Led (PWM)



Resumen

Los temas que vimos hoy fueron:

- LPCxpresso y su toolchain: target, BaseBoard, IDE
- Proceso de desarrollo de software: edicion, compilación, linkeo, descarga y depuración
- Sistemas de control de versiones
- Repaso de C: punteros, estructuras, operadores de bits y mascaras.
- Ejemplos básicos: manejos de I/O y estructuras repetitivas.

Resumen

Prácticamos sobre la herramienta LPCXPresso:

- Descargar un repositorio
- Escribir un programa nuevo
- Compilarlo y descargarlo
- Debuggear paso por paso nuestra aplicación

Resumen

¿Dudas? ¿Consultas?

Cualquier comentario o consulta lo pueden hacer a la lista del grupo:

seminario-embebidos@googlegroups.com

Resumen

Muchas gracias