CMSIS y Drivers para periféricos

Alan Kharsansky

Abril - 2011

Índice

Recursos de NXP

NXP proveé en su sitio web una implementación del CMSIS junto con Drivers básicos para sus periféricos. Desde su última versión también incluye funciones de DSP optimizadas para distintas arquitecturas.

Website del LPC1768 ▸ Ir

Se lo encuentra bajo el nombre:
LPC17xx CMSIS-Compliant Standard Peripheral Firmware Driver Library (GNU, Keil, IAR)

Recursos de NXP

Los archivos provistos por NXP no estan contenidos dentro de un proyecto de CodeRed por lo que se arregló esta distribución para poder hacerlo. Este recurso se encuentra en el website de la materia para descargar.

El paquete también original incluye un archivo de ayuda (.chm) que fue descomprimido en archivos HTML para poder hacerlo portable. Este puede ser descargado del website de la materia.

Estructura del paquete

El proyecto esta estructurado de la siguiente forma:

- Core
  - CM3
    - CoreSupport
    - DeviceSupport
  - Documentation
  - DSP_Lib
- Drivers
- Examples

Utilización

Para poder utilizar estas bibliotecas el proyecto se debe encontrar en el mismo workspace que nuestra aplicación. Como todo (CMSIS, Drivers y DSP_Lib) se encuentra bajo el mismo proyecto, solo es necesario linkear con una sola biblioteca para su utilización.

LPC17XX_CMSIS_Drivers

Inclusión

Recordar: Se deberá incluir los archivos de cabeceras (.h) que se deseen utilizar. Por ejemplo:

CMSIS

```
#include "LPC17xx.h"
```

Drivers

```
#include "  lpc17xx_uart.h"
#include "  lpc17xx_gpio.h"
```

DSP

```
#include "arm_math.h"
```

Un ejemplo - Blinky con Drivers

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

#include "lpc17xx_gpio.h"

int main(void) {
   GPIO_SetDir(0,(1<<22),1);
   while(1) {
      int i;
      GPIO_SetValue(0,(1<<22));
      for (i=0;i<10000000;i++);
      GPIO_ClearValue(0,(1<<22));
      for (i=0;i<10000000;i++);
   }
   return 0 ;
}
```

Configuración de pines. El PINSEL

Muchos de los pines externos que posee el encapsulado del LPC1768 pueden cumplir varias funciones diferentes. Pueden ser digitales, analógicos, de comunicación, etc.

También pueden tener activados Pull-Ups o Pull-Down y otras diferentes características. Es por eso que debemos antes de usar cualquier pin, configurarlo para cumpla con la función que nosotros buscamos.

Configuración de pines. El PINSEL

Para poder seleccionar la funcin de cada pin, se incluyen los registros PINSELn. Estos registros permiten seleccionar (de a 2 bits) la función del pin. Siendo posibles las siguientes opciones:

Table 74. Pin function select register bits

PINSEL0 to PINSEL9 Values	Function	Value after Reset
00	Primary (default) function, typically GPIO port	00
01	First alternate function	
10	Second alternate function	
11	Third alternate function	

Configuración de pines. El PINMODE

Los registros PINMODEn nos permiten configurar el comportamiento eléctrico de los pines. Siendo los posibles valores:

Table 75. Pin Mode Select register Bits

PINMODE0 to PINMODE9 Values	Function	Value after Reset
00	Pin has an on-chip pull-up resistor enabled.	00
01	Repeater mode (see text below).	
10	Pin has neither pull-up nor pull-down resistor enabled.	
11	Pin has an on-chip pull-down resistor enabled.	

También existe un registro PINMODE_OD que permite configurar el funcionamiento o no como Open Drain.

Configuración de pines. Ejemplo

Se puede encontrar en el manual de usuario del microcontrolador una tabla que resumen los distintos campos para un registro PINSEL. Por ejemplo:

Table 78. Pin function select register 0 (PINSEL0 - address 0x4002 C000) bit description

PINSEL0	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P0.0	GPIO Port 0.0	RD1	TXD3	SDA1	00
3:2	P0.1	GPIO Port 0.1	TD1	RXD3	SCL1	00
5:4	P0.2	GPIO Port 0.2	TXD0	AD0.7	Reserved	00
7:6	P0.3	GPIO Port 0.3	RXD0	AD0.6	Reserved	00
9:8	P0.4[1]	GPIO Port 0.4	I2SRX_CLK	RD2	CAP2.0	00
11:10	P0.5[1]	GPIO Port 0.5	I2SRX_WS	TD2	CAP2.1	00
13:12	P0.6	GPIO Port 0.6	I2SRX_SDA	SSEL1	MAT2.0	00
15:14	P0.7	GPIO Port 0.7	I2STX_CLK	SCK1	MAT2.1	00
17:16	P0.8	GPIO Port 0.8	I2STX_WS	MISO1	MAT2.2	00
19:18	P0.9	GPIO Port 0.9	I2STX_SDA	MOSI1	MAT2.3	00
21:20	P0.10	GPIO Port 0.10	TXD2	SDA2	MAT3.0	00
23:22	P0.11	GPIO Port 0.11	RXD2	SCL2	MAT3.1	00
29:24	-	Reserved	Reserved	Reserved	Reserved	0
31:30	P0.15	GPIO Port 0.15	TXD1	SCK0	SCK	00

Configuración de pines. Uso

Para facilitar la configuración de los distintos pines del microcontrolador se puede utilizar uno de los drivers que provee NXP: **lpc17xx_pinsel.h**. La función que vamos a utilizar tiene el siguiente prototipo:

```
void PINSEL_ConfigPin (PINSEL_CFG_Type *PinCfg);
```

Configuración de pines. Uso

El tipo de dato que hay que pasarle es una estructura de configuración. Esta está compuesta por:

- Portnum = PINSEL_PORT_x (x:0-3)
- Pinnum = PINSEL_PIN_x (x:0-31)
- Funcnum = PINSEL_FUNC_x (x:0-3)
- Pinmode = Pullup, Pulldown o TriState
- OpenDrain = Normal u Open Drain

Configuración de pines. Ejemplo

Si se desea configurar los pines 0 y 1 del Port 0 para que esten conectados a las lineas de Tx y Rx de la UART3, podemos hacer lo siguiente:

```
PINSEL_CFG_Type PinCfg;

PinCfg.Funcnum = PINSEL_FUNC_2;
PinCfg.OpenDrain = PINSEL_PINMODE_NORMAL;
PinCfg.Pinmode = PINSEL_PINMODE_PULLUP;
PinCfg.Pinnum = PINSEL_PIN_0;
PinCfg.Portnum = PINSEL_PORT_0;

PINSEL_ConfigPin(&PinCfg);

PinCfg.Pinnum = PINSEL_PIN_1;

PINSEL_ConfigPin(&PinCfg);
```

Actividad Nro 1 - Consigna

Se desea hacer un programa que permita utilizar alguna de las UARTs que trae el LPC1768. Para ello realizaremos un programa que:

- Configure la UART en: 115200-8-N-1
- Envie un mensaje de bienvenida al encender
- Espere un caracter nuevo y lo repita por la salida (modo ECHO)

Puede ver el ejercicio completo en el workspace de esta clase

Actividad Nro 1 - Tips

Las siguiente funciones deben ser utilizadas para inicializar correctamente el periferico:

```
// Configuramos la UART
UARTConfigStruct.Baud_rate = 115200;
UARTConfigStruct.Databits = UART_DATABIT_8;
UARTConfigStruct.Parity = UART_PARITY_NONE;
UARTConfigStruct.Stopbits = UART_STOPBIT_1;

// Inicializamos la UART
UART_Init(LPC_UART3, &UARTConfigStruct);
UART_TxCmd(LPC_UART3, ENABLE);
```

Actividad Nro 1 - Tips

Y para enviar y recibir podemos utilizar:

```
UART_SendByte(LPC_UART3,c);

UART_Send(LPC_UART3, msg, strlen(msg), BLOCKING);

buff = UART_ReceiveByte(LPC_UART3);
```

Actividad Nro 1 - Adicional

Se propone utilizar interrupciones para recibir los datos en vez de hacerlo en el programa principal. Para poder realizarlo se deben modificar 2 cosas:

- Activar la interrupción tanto en el periférico como en el NVIC
- Escribir un handler que reciba los datos y los reenvie

Actividad Nro 1 - Adicional

Para poder activar la interrupción usamos el siguiente código:

```
// Habilitamos la interrupci n de recepci n de la UART
UART_IntConfig(LPC_UART3, UART_INTCFG_RBR, ENABLE);

// Habilitamos la interrupci n de la UART3 en el NVIC
NVIC_EnableIRQ(UART3_IRQn);
```

Actividad Nro 1 - Adicional

El handler podría tener la siguiente forma:

```
void UART3_IRQHandler(void)
{
    char c;
    // Leo un dato nuevo
    c = UART_ReceiveByte(LPC_UART3);

    // Envio el dato leido
    UART_SendByte(LPC_UART3,c);
}
```

Cuidado: El handler es llamado cuando ocurre alguna interrupción en el modulo de UART3 como por ejemplo, Tx buffer vacio, un nuevo dato para leer, una linea de control cambió su valor, etc. **¿Por qué en este caso no debemos asegurarnos de donde vino la interrupción?**

Actividad Nro 2 - UART con ADC

Se desea hacer un programa que permita enviar el valor de un canal analógico por la UART (utilizando el ADC). Para eso debemos:

- Configurar los pines para que use la función analógica
- Configurar el conversor A/D
- Tomar una muestra y enviarla

Actividad Nro 2 - UART con ADC

Utilizaremos el trimpot que esta en el BaseBoard. El mismo está conectado al pin **GPIO0.23** que corresponde al canal analógico **AD0**. Para encender la UART debemos utilizar el siguiente código:

```
ADC_Init(LPC_ADC, 200000);
ADC_IntConfig(LPC_ADC,ADC_ADINTEN0,DISABLE);
ADC_ChannelCmd(LPC_ADC,ADC_CHANNEL_0,ENABLE);
```

Actividad Nro 2 - UART con ADC

Para convertir un dato y luego leerlo debemos utilizar siguiente código:

```
// Start conversion
ADC_StartCmd(LPC_ADC,ADC_START_NOW);

//Wait conversion complete
while (!(ADC_ChannelGetStatus(LPC_ADC,ADC_CHANNEL_0 ,
                                        ADC_DATA_DONE)));

// Read the value
  adc_value = ADC_ChannelGetData(LPC_ADC,ADC_CHANNEL_0);
```

Actividad Nro 3 - UART con Maquinas de Estado

Se desea hacer un programa que permita recibir datos por una UART y dependiendo del estado en que se encuentre realizar diferentes acciones con los datos recibidos. Para ello se utilizará el siguiente diagrama de estados:

Diagrama de estados

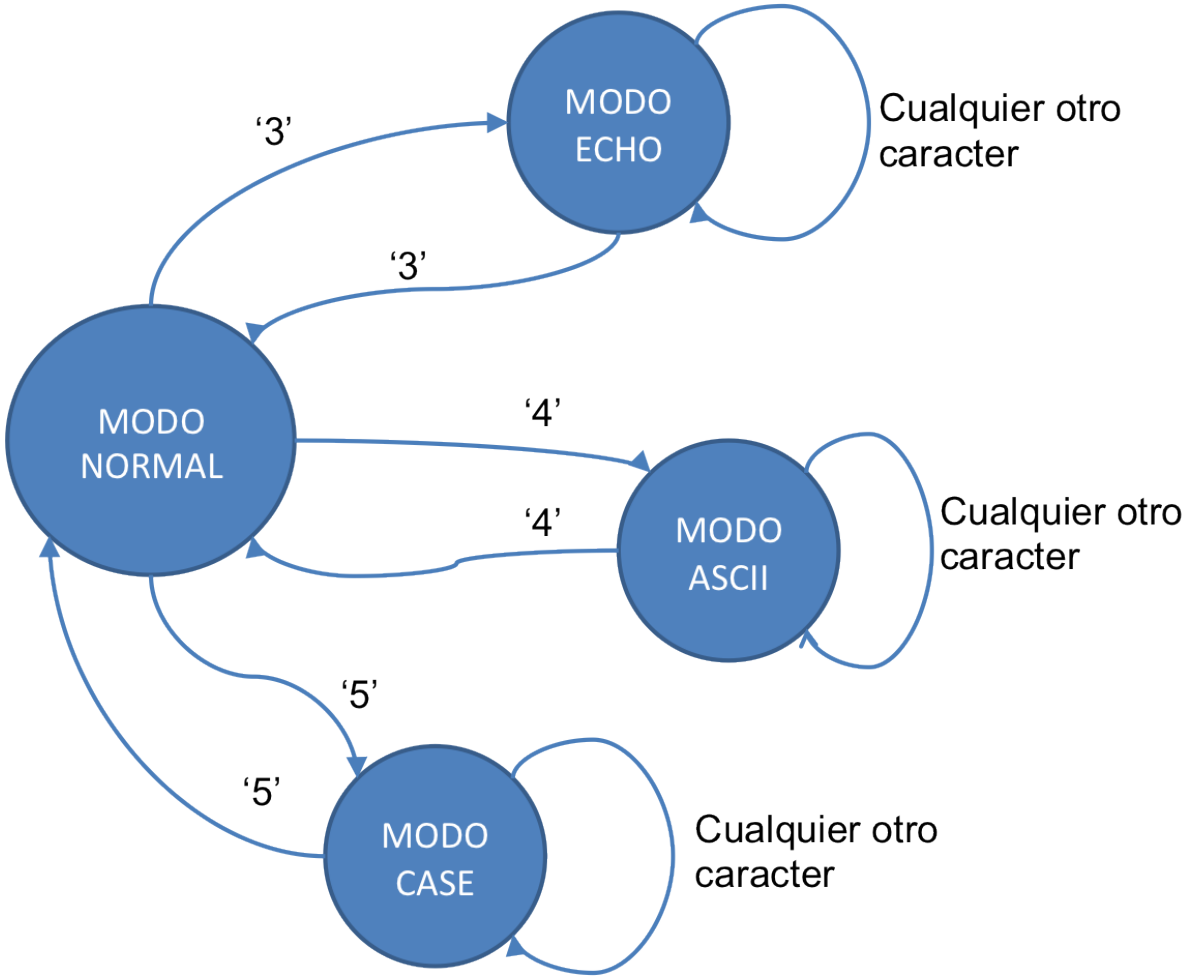

Diagrama de estados - MODO NORMAL

Este modo "espía" el último caracter recibido que luego se interpretan como comandos. Se definen los comandos:

- **1:** Muestro todo lo que hay guardado en el buffer
- **2:** Vacio el buffer
- **3:** Cambio a modo ECHO
- **4:** Cambio a modo ASCII
- **5:** Cambio a modo CASE

Diagrama de estados - MODO ECHO

En este modo todo lo que se recibe por Rx se repite por Tx

Para salir del modo se debe volver a enviar el numero 3

Diagrama de estados - MODO ASCII

Los caracteres recibidos se mostran de la siguiente manera:

- Si son no imprimibles (¡0x20) se muestra su nombre. Por ejemplo: EOL
- Si son imprimibles, se muestra su numero Hexa. Por ejemplo, si se recibe una 'a' se deberá enviar 0x61

Para salir del modo se debe volver a enviar el numero 4

Diagrama de estados - MODO CASE

En este modo, todos los caracteres alfabeticos que se reciban por Rx en minusculas se deben enviar por Tx en mayusculas y viceversa

Para salir del modo se debe volver a enviar el numero 5

Consideraciones generales

La recepción de datos se debe hacer de forma asincronica. Es decir, manejada por interrupciones.

Como la rutina de interrupción deberá ser lo más corta posible, los datos se almacenaran en una buffer y luego serán procesados por el loop principal.

Solución a la actividad

```
if (c<0x20){
    UARTSendString(ascii[c]);
}else{
    char buff[3];
    intToString((int)c,buff,3,16);
    UARTSendString("0x");
    UARTSendString(buff);
    UARTSendString(("\n\r");
}
```

```
if ((c>='a') && (c<='z')){
    c = c+('A'-'a');
    UART_SendByte(LPC_UART3,c);
}else if ((c>='A') && (c<='Z')){
    c = c-('A'-'a');
    UART_SendByte(LPC_UART3,c);
}
```

¿Dudas? ¿Consultas?

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

seminario-embebidos@googlegroups.com

Muchas gracias