Laboratorio de control de dispositivos utilizando un SoC ARM

José Manuel Mendías Cuadros
Juan Carlos Fabero Jiménez
Hortensia Mecha López
Carlos González Calvo
Juan Antonio Clemente Barreira

12 de febrero de 2015

Índice general

Práctica 1

Sistema objetivo

La plataforma objetivo que vamos a utilizar en las prácticas es una placa de prototipado Embest modelo S3CEV40 que consta de un System-on-Chip (SoC) Samsung S3C44B0X conectado a 2MB de Flash ROM, a 8MB de SDRAM y a una colección de dispositivos externos. El sistema muestra en la figura 1.1.

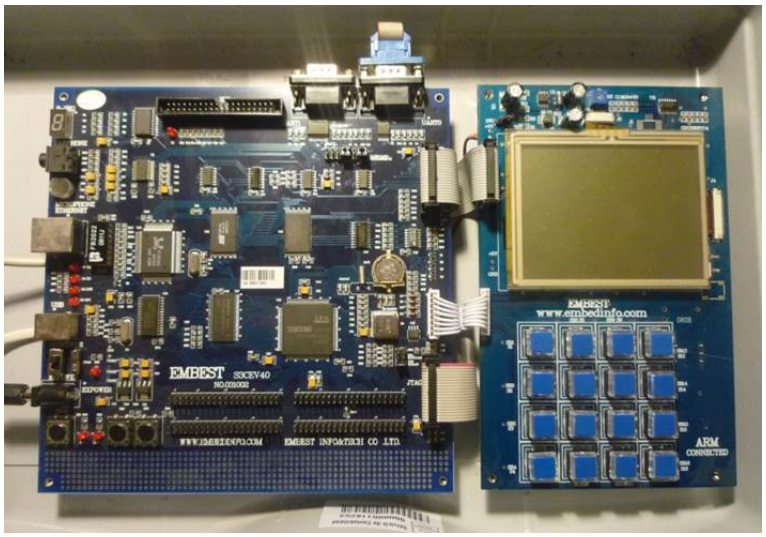

Figura 1.1: Placa de prototipado Embest S3CEV40

El microcontrolador S3C44B0X consta de:

- 1 core ARM7TDMI a 66 MHz.
- 8KB de memoria SRAM configurable por programa como cache o scratchpad.
- 2 buses: uno de sistema y otro de periféricos.

- 25 controladores de dispositivos configurables por programa.

Los controladores de dispositivo integrados en dicho SoC son los siguientes:

- Controlador de memoria.
- Gestor de reloj y energía.
- Controlador de cache.
- Árbitro de bus.
- 4 controladores de DMA (Direct Memory Access).
- Controlador de pines de E/S (71 pines multiplexados organizados en 7 puertos).
- 6 temporizadores.
- 2 UART (Universal Asynchronous Receiver Transmitter).
- Controlador de interrupciones.
- Controlador de LCD (Liquid Cristal Display).
- Conversor A/D (Analog to Digital) de 8 canales.
- Reloj de tiempo real.
- Watchdog.
- Controlador de bus IIC (Inter Integrated Circuits).
- Controlador de bus IIS (Integrated Interchip Sound).
- Controlador de bus SIO (Synchronous Input/Output).

Los anteriores controladores están conectados a alguno de siguientes dispositivos, todos ellos montados sobre la placa del sistema objetivo:

- NAND Flash 16 MB
- IIC EEPROM 4Kb
- 2 pulsadores / 2 LED
- Display 7 segmentos
- Teclado matricial 4×4
- Controlador Ethernet IEEE 802.3 (conector RJ-45)
- IIS Audio CODEC stéreo (micrófono electrect / jack 3,5 mm)

- Controlador USB 1.0 / 1.1 (conector tipo B)
- 2 drivers RS-232 (conectores DB-9)
- Touchpad LCD 320×240
- Conector IDE
- Conector JTAG

Desde el punto de vista del programador, estos dispositivos se hallan o bien mapeados en memoria o bien en puertos de E/S dando lugar al mapa de puertos mostrado en la tabla 1.1.

La placa de expansión se conecta al sistema objetivo a través de un conector ad-hoc que encaja en las 4 tiras de 20×2 pines que este último dispone, tal como muestra la figura 1.2.

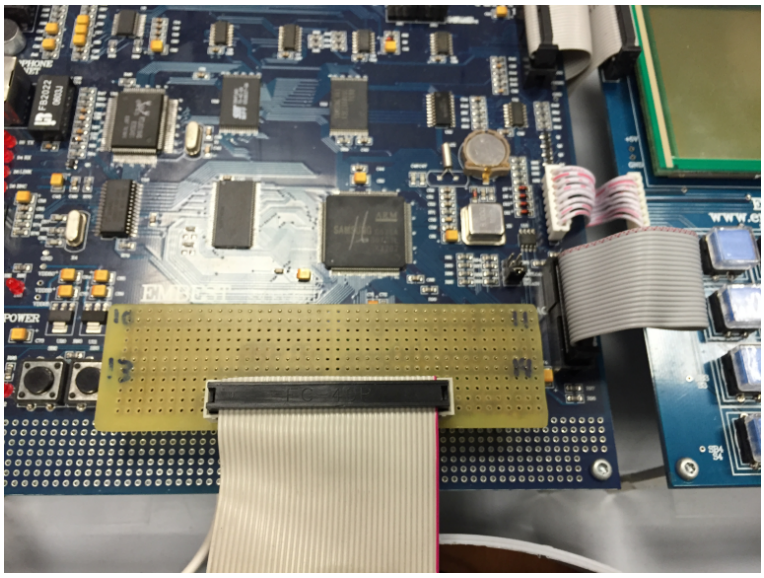

Figura 1.2: Conexión con la placa Embest S3CEV40

De todos los periféricos de la placa de expansión, en este conjunto de prácticas, solo utilizaremos aquellos que no sean redundantes con los ya existentes en el sistema objetivo. Además, al tener estos periféricos muchos de sus pines compartidos, no podrán usarse todos simultáneamente. Así, la placa de expansión podrá utilizarse en alguna de las 3 configuraciones mostradas en la tabla 1.2 en la que se detallan los dispositivos a los que el SoC podrá acceder simultáneamente. Cada una de las configuraciones requiere que los jumpers de la placa de expansión se dispongan según lo indicado en la tabla 1.3. Todos los dispositivos de la placa objetivo se pueden usar en simultaneidad con los de la placa de expansión, exceptuando la UART1 cuyo uso es mutuamente exclusivo con el uso del motor.

Desde el punto de vista del programador, cada una de las 3 configuraciones da lugar a un mapa de puertos distinto. Dichos mapas se muestran en las tablas 1.4 (configuración 1), 1.5(configuración 2) y 1.6 (configuración 3).

Puerto	bit number	15	14	13	12	11	10	9	8
A	subsistema							audio	libre
	nombre							L3DATA	J10-P29
	tipo							out	(out)
B	subsistema						leds	leds	keypad
	nombre						LED DER	LED IZQ	nGCS3
	tipo						out	out	interna
C	subsistema	libre	libre	uart- 1	uart-1	uart-1	uart-1	libre	libre
	nombre	J13-37	J13-36	RxD1	TxD1	CTS1	RTS1	J13-31	J13-30
	tipo	(out)	(out)	interna	interna	interna	interna	(out)	(out)
D	subsistema								
	nombre								
	tipo								
E	subsistema								audio
	nombre								CODECLK
	tipo								interna
F	subsistema								jumpers
	nombre								SW3
	tipo								in
G	subsistema								
	nombre								
	tipo								

Puerto	bit number	7	6	5	4	3	2	1	0
A	subsistema	bus sistema	bus sistema	bus sistema	bus sistema	bus sistema	bus sistema	bus sistema	libre
	nombre	ADDR22	ADDR21	ADDR20	ADDR19	ADDR18	ADDR17	ADDR16	J10-P7
	tipo	interna	interna	interna	interna	interna	interna	interna	(out)
B	subsistema	libre	varios	audio	audio	bus	bus	bus	bus
	nombre	J11-P7	nGCS1	L3CLOCK	L3MODE	nSRAS	nSCAS	SCLK	SCKE
	tipo	(out)	interna	out	out	interna	interna	interna	interna
C	subsistema	libre	libre	libre	libre	audio	audio	audio	audio
	nombre	J13-P28	J13-P27	J13-P26	J13-P25	IISCLK	IISDI	IISDO	IISLRCK
	tipo	(out)	(out)	(out)	(out)	interna	interna	interna	interna
D	subsistema	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen
	nombre	VFRAME	VM	VLINE	VCLK	VD3	VD2	VD1	VD0
	tipo	interna	interna	interna	interna	interna	interna	interna	interna
E	subsistema	touchpad	touchpad	touchpad	touchpad	timer-0	uart-0	uart-0	libre
	nombre	Y-	X-	Y+	X+	TOUT0	RxD0	TxD0	J11-P13
	tipo	out	out	out	out	interna	interna	interna	(out)
F	subsistema	jumpers	disk	disk	disk	disk	libre	bus IIC	bus IIC
	nombre	SW4	ALE	CLE	CE	R/B	J11-P19	IICSDA	IICSCL
	tipo	in	out	out	out	in	(out)	interna	interna
G	subsistema	pulsador der.	pulsador izq.	libre	libre	ethernet	touchpad	keypad	usb
	nombre	EINT7	EINT6	J11-P28	J11-P27	EINT3	EINT2	EINT1	EINT0
	tipo	interna	interna	(out)	(out)	interna	interna	interna	interna

Tabla 1.1: Mapa de puertos de E/S

Dispositivo	**CONF1**	**CONF2**	**CONF3**
motor paso a paso	X	X	X
dispositivos IIC	X	X	X
LED RGB	X		
matriz de puntos		X	
pantalla LCD			X
emisor-receptor de infrarrojos		X	X
zumbador / ventilador		X	X
altavoz		X	X

Tabla 1.2: Configuraciones de dispositivos

Jumper	**Función**	**CONF1**	**CONF2**	**CONF3**
JP1	matriz de puntos	open	close	open
JP2	display LCD	open	open	close
JP3	retorno keypad	open	open	open
JP4	ADC-OE	open	open	open
JP5	ADC-EOC	open	open	open
JP6	scan keypad	open	open	open
JP7	emisor infrarrojos	open	close	close
JP8	receptor infrarrojos	open	close	close
JP9 (1-3)	LED RGB 0	close	open	open
JP9 (4-6)	LED RGB 1	close	open	open
JP9 (7-9)	LED RGB 2	close	open	open

Tabla 1.3: Configuración de los *jumpers*

Conocido el mapa de puertos a utilizar, el acceso a los periféricos de la placa de expansión simplemente supondrá realizar escrituras/lecturas de los registros, según el caso, del controlador de pines de E/S o del controlador del bus IIC integrados en el SoC.

Puerto	bit number	15	14	13	12	11	10	9	8
A	subsistema							audio	libre
	nombre							L3DATA	J10-P29
	tipo							out	(out)
B	subsistema						leds	leds	keypad
	nombre						LED DER	LED IZQ	nGCS3
	tipo						out	out	interna
C	subsistema	RGB-LED	RGB-LED	motor	motor	motor	motor/vent.	RGB-LED	RGB-LED
	nombre	R1	B0	PH_A2	PH_B2	PH_A1	PH_B1	G0	R0
	tipo	out	out	out	out	out	out	out	out
D	subsistema								
	nombre								
	tipo								
E	subsistema								audio
	nombre								CODECLK
	tipo								interna
F	subsistema								jumpers
	nombre								SW3
	tipo								in
G	subsistema								
	nombre								
	tipo								

Puerto	bit number	7	6	5	4	3	2	1	0
A	subsistema	bus sistema	bus sistema	bus sistema	bus sistema	bus sistema	bus sistema	bus sistema	libre
	nombre	ADDR22	ADDR21	ADDR20	ADDR19	ADDR18	ADDR17	ADDR16	J10-P7
	tipo	interna	interna	interna	interna	interna	interna	interna	(out)
B	subsistema	RGB-LED	varios	audio	audio	bus	bus	bus	bus
	nombre	G1	nGCS1	L3CLOCK	L3MODE	nSRAS	nSCAS	SCLK	SCKE
	tipo	out	interna	out	out	interna	interna	interna	interna
C	subsistema	libre	libre	libre	libre	audio	audio	audio	audio
	nombre	J13-P28	J13-P27	J13-P26	J13-P25	IISCLK	IISDI	IISDO	IISLRCK
	tipo	(out)	(out)	(out)	(out)	interna	interna	interna	interna
D	subsistema	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen
	nombre	VFRAME	VM	VLINE	VCLK	VD3	VD2	VD1	VD0
	tipo	interna	interna	interna	interna	interna	interna	interna	interna
E	subsistema	touchpad	touchpad	touchpad	touchpad	RGB-LED	uart-0	uart-0	RGB-LED
	nombre	Y-	X-	Y+	X+	B2	RxD0	TxD0	G2
	tipo	out	out	out	out	out	interna	interna	out
F	subsistema	jumpers	disk	disk	disk	disk	RGB-LED	bus IIC	bus IIC
	nombre	SW4	ALE	CLE	CE	R/B	B1	IICSDA	IICSCL
	tipo	in	out	out	out	in	out	interna	interna
G	subsistema	pulsador der.	pulsador izq.	bus IIC/vent.	RGB-LED	ethernet	touchpad	keypad	usb
	nombre	EINT7	EINT6	IICINT/FANRTN	R2	EINT3	EINT2	EINT1	EINT0
	tipo	interna	interna	in	out	interna	interna	interna	interna

Tabla 1.4: Mapa de puertos en configuración 1

Puerto	bit number	15	14	13	12	11	10	9	8
A	subsistema							audio	libre
	nombre							L3DATA	J10-P29
	tipo							out	(out)
B	subsistema						leds	leds	keypad
	nombre						LED DER	LED IZQ	nGCS3
	tipo						out	out	interna
C	subsistema	libre	banner	motor	motor	motor	motor/vent.	banner	banner
	nombre	J13-37	ROWCLK	PH_A2	PH_B2	PH_A1	PH_B1	ROWRST	ROWDATA
	tipo	(out)	out	out	out	out	out	out	out
D	subsistema								
	nombre								
	tipo								
E	subsistema								audio
	nombre								CODECLK
	tipo								interna
F	subsistema								jumpers
	nombre								SW3
	tipo								in
G	subsistema								
	nombre								
	tipo								

Puerto	bit number	7	6	5	4	3	2	1	0
A	subsistema	bus sistema	bus sistema	bus sistema	bus sistema	bus sistema	bus sistema	bus sistema	libre
	nombre	ADDR22	ADDR21	ADDR20	ADDR19	ADDR18	ADDR17	ADDR16	J10-P7
	tipo	interna	interna	interna	interna	interna	interna	interna	(out)
B	subsistema	infrarrojos	varios	audio	audio	bus	bus	bus	bus
	nombre	IR-LED	nGCS1	L3CLOCK	L3MODE	nSRAS	nSCAS	SCLK	SCKE
	tipo	out	interna	out	out	interna	interna	interna	interna
C	subsistema	libre	banner	banner	banner	audio	audio	audio	audio
	nombre	J13-P28	COLRST	COLCLK	COLDATA	IISCLK	IISDI	IISDO	IISLRCK
	tipo	(out)	out	out	out	interna	interna	interna	interna
D	subsistema	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen
	nombre	VFRAME	VM	VLINE	VCLK	VD3	VD2	VD1	VD0
	tipo	interna	interna	interna	interna	interna	interna	interna	interna
E	subsistema	touchpad	touchpad	touchpad	touchpad	altavoz	uart-0	uart-0	zumbador
	nombre	Y-	X-	Y+	X+	SPEAKER	RxD0	TxD0	BUZZER
	tipo	out	out	out	out	out	interna	interna	out
F	subsistema	jumpers	disk	disk	disk	disk	libre	bus IIC	bus IIC
	nombre	SW4	ALE	CLE	CE	R/B	J11-P19	IICSDA	IICSCL
	tipo	in	out	out	out	in	(out)	interna	interna
G	subsistema	pulsador der.	pulsador izq.	bus IIC/vent.	infrarrojos	ethernet	touchpad	keypad	usb
	nombre	EINT7	EINT6	IICINT/FANRTN	IR-RCV	EINT3	EINT2	EINT1	EINT0
	tipo	interna	interna	in	in	interna	interna	interna	interna

Tabla 1.5: Mapa de puertos en configuración 2

Puerto	bit number	15	14	13	12	11	10	9	8
A	subsistema							audio	libre
	nombre							L3DATA	J10-P29
	tipo							out	(out)
B	subsistema						leds	leds	keypad
	nombre						LED DER	LED IZQ	nGCS3
	tipo						out	out	interna
C	subsistema	libre	libre	motor	motor	motor	motor/vent.	display	display
	nombre	J13-37	J13-30	PH_A2	PH_B2	PH_A1	PH_B1	LCD-E	LCD-RS
	tipo	(out)	(out)	out	out	out	out	out	out
D	subsistema								
	nombre								
	tipo								
E	subsistema								audio
	nombre								CODECLK
	tipo								interna
F	subsistema								jumpers
	nombre								SW3
	tipo								in
G	subsistema								
	nombre								
	tipo								

Puerto	bit number	7	6	5	4	3	2	1	0
A	subsistema	bus sistema	bus sistema	bus sistema	bus sistema	bus sistema	bus sistema	bus sistema	libre
	nombre	ADDR22	ADDR21	ADDR20	ADDR19	ADDR18	ADDR17	ADDR16	J10-P7
	tipo	interna	interna	interna	interna	interna	interna	interna	(out)
B	subsistema	infrarrojos	varios	audio	audio	bus	bus	bus	bus
	nombre	IR-LED	nGCS1	L3CLOCK	L3MODE	nSRAS	nSCAS	SCLK	SCKE
	tipo	out	interna	out	out	interna	interna	interna	interna
C	subsistema	display	display	display	display	audio	audio	audio	audio
	nombre	LCD-D7	LCD-D6	LCD-D5	LCD-D4	IISCLK	IISDI	IISDO	IISLRCK
	tipo	out	out	out	out	interna	interna	interna	interna
D	subsistema	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen	lcd screen
	nombre	VFRAME	VM	VLINE	VCLK	VD3	VD2	VD1	VD0
	tipo	interna	interna	interna	interna	interna	interna	interna	interna
E	subsistema	touchpad	touchpad	touchpad	touchpad	altavoz	uart-0	uart-0	zumbador
	nombre	Y-	X-	Y+	X+	SPEAKER	RxD0	TxD0	BUZZER
	tipo	out	out	out	out	out	interna	interna	out
F	subsistema	jumpers	disk	disk	disk	disk	libre	bus IIC	bus IIC
	nombre	SW4	ALE	CLE	CE	R/B	J11-P19	IICSDA	IICSCL
	tipo	in	out	out	out	in	(out)	interna	interna
G	subsistema	pulsador der.	pulsador izq.	bus IIC/vent.	infrarrojos	ethernet	touchpad	keypad	usb
	nombre	EINT7	EINT6	IICINT/FANRTN	IR-RCV	EINT3	EINT2	EINT1	EINT0
	tipo	interna	interna	in	in	interna	interna	interna	interna

Tabla 1.6: Mapa de puertos en configuración 3

Práctica 2

Control de una matriz de puntos o Banner

2.1. Introducción

La placa de expansión incorpora un una matriz de 40×7 leds multiplexada (lógica directa para filas, e inversa para columnas) accesible indirectamente a través de 2 registros de desplazamiento de 40 y 7 bits.

El registro de desplazamiento de filas (40 bits) se controla a través de 3 líneas: datos serie (`ROWDATA`), reloj (`ROWCLK`) y reset (`ROWRST`). El registro de desplazamiento de columnas (7 bits) también se controla a través de 3 líneas: datos serie (`COLDATA`), reloj (`COLCLK`) y reset (`ROWRST`).

Para visualizar una información estática en el banner es necesario refrescar los leds, bien por filas o bien por columnas, con una cierta frecuencia.

En el caso de refresco por filas, sucesivamente debe seleccionarse 1 de las 7 columnas (haciendo rotar un 1 en el registro de desplazamiento de columnas) e introducir en serie los 40 bits de información (1 por cada led de la fila) en el registro de desplazamiento de filas. La columna seleccionada y los 40 bits de información deberán estar fijos durante un cierto periodo de persistencia.

En el caso de refresco por columnas, sucesivamente debe seleccionarse 1 de las 40 filas (haciendo rotar un 1 en el registro de desplazamiento de filas) e introducir en serie los 7 bits de información (1 por cada led de la columna) en el registro de desplazamiento de columnas. La fila seleccionada y los 7 bits de información deberán estar fijos durante un cierto periodo de persistencia.

Para utilizar el banner es necesario previamente que los jumpers de la placa de expansión estén en la configuración 2 (véase tabla 1.2). En esta configuración las líneas de control del banner están conectadas al puerto C del SoC en los bits indicados a continuación:

- `ROWDATA : PC8`
- `ROWCLK : PC14`

- `ROWRST : PC9`
- `COLDATA : PC4`
- `COLCLK : PC5`
- `COLRST : PC6`

2.2. Control software

El módulo software (driver) para control del banner deberá disponer de las siguientes funciones públicas:

- `void dotmatrix_init( void )`: Inicializa el driver y borra la matriz de leds.
- `void dotmatrix_clear( void )`: Borra la matriz de leds.
- `void dotmatrix_putpixel( uint8 x, uint8 y, uint8 val )`: Pone el pixel (x,y) en el valor indicado.
- `uint8 dotmatrix_getpixel( uint8 x, uint8 y )`: Devuelve el valor al que está el pixel (x,y).

Las definiciones de los tipos usados se encuentran en el fichero `common_types.h`

El refresco de la matriz de puntos se realizará por columnas y será efectuado por una rutina de tratamiento de interrupción para permitir que el programa principal que utiliza el módulo no se vea bloqueado por el refresco periódico de la matriz de puntos. Se utilizará el temporizador 0 integrado en el SoC para generar una interrupción cada 2 ms.

2.3. Aspectos de implementación

La comunicación entre las funciones públicas de acceso a los píxeles del banner y la rutina de tratamiento de interrupción que los refresca, se realizará a través de una variable global privada que almacena el valor de cada led de la matriz.

```
static uint64 dots_col[7] = { 0, 0, 0, 0, 0, 0, 0 };
```

Además de las funciones públicas, privadamente deberán implementarse las siguientes funciones auxiliares:

- `inline static void risingedge( uint16 bitnum )`: Genera un pulso de subida en el bit indicado del puerto C.
- `void static dotmatrix_putcol( uint64 data )`: Envía en serie por `COL_DATA` los 40 bits menos significativos del argumento comenzando por el de menor peso. Tras cada envío genera un pulso de subida en `COL_CLK`. Tras enviar los 40 bits genera un pulso de subida en `COL_CLK` adicional para compensar la latencia de 1 ciclo que tiene el registro de desplazamiento.

- `void static dotmatrix_refresh_by_cols( void ) __attribute__ ((interrupt ("IRQ")))`: Rutina de tratamiento de interrupción instalada en el vector de interrupciones por el temporizador 0. A cada iniciación, resetea la columna actual, selecciona la siguiente columna a refrescar y envía en serie los bits a refrescar (contenidos en la componente que corresponda del array `dots_col`). Las columnas se seleccionan haciendo rotar un 1 por el desplazador de filas, esto es, generando un pulso de subida en `ROW_CLK`, e insertando un 1 por `ROW_DATA` en 1 de cada 7 iniciaciones.

La función `dotmatrix_init` deberá configurar los bits correspondientes del puerto C como de salida, pondrá a 0 todas las líneas del control del banner, inicializará el temporizador 0, borrará la matriz de leds, instalará la rutina de tratamiento de refresco por columnas en el vector correspondiente y arrancará el temporizador 0 para que genere interrupciones periódicamente.

Las funciones `dotmatrix_putpixel` y `dotmatrix_getpixel`, respectivamente, escriben y leen bits concretos de cada una de las componentes del array `dots_col`.

Los prototipos de las funciones para acceso al temporizador 0 se encuentran en el fichero `timers.h`. Asimismo, los ficheros `s3c44b0x.h` y `s3cev40.h`, respectivamente, definen nemotécnicos para el acceso a los registros de los controladores de dispositivos internos del chip S3C44BOX y a los distintos elementos de la placa de prototipado Embest S3CEV40.

Práctica 3

Control de un display LCD alfanumérico

3.1. Introducción

El LCD alfanumérico que incorpora la placa de expansión está formado por 2 filas de 16 caracteres cada uno y cada carácter se visualiza como una matriz de 5×8 puntos. Internamente almacena los patrones de puntos de 208 caracteres diferentes (identificados por su código ASCII) y también permite la definición por el usuario del patrón de puntos de 8 caracteres adicionales. Tiene facilidades para el manejo del cursor y la inicialización de la pantalla.

Para comunicarse utiliza un protocolo paralelo de tipo *strobe* con señales de habilitación (`LCD-E`) selección de operación (lectura/escritura, `LCD-RW`, que en la placa de expansión está fijado a escritura), de selección de registro (instrucción-estado/dato, `LCD-RS`) y de datos paralelos seleccionable entre 4 u 8 bits (`LCD-D4..D7`).

Para utilizar el LCD es necesario previamente que los *jumpers* de la placa de expansión estén en la configuración 3 (véase tabla 1.2). En esta configuración las líneas de comunicación con el LCD están conectadas al puerto `C` del SoC en los bits indicados a continuación:

- `LCD-RS` : `PC8`
- `LCD-E` : `PC9`
- `LCD-D4` : `PC4`
- `LCD-D5` : `PC5`
- `LCD-D6` : `PC6`
- `LCD-D7` : `PC7`

3.2. Control software

El módulo *software* (*driver*) para control del LCD deberá disponer de las siguientes funciones públicas:

- `void display_init( void )`: Inicializa el *driver* y el *display* (interfaz de 4 bits, 2 líneas de caracteres, cursor apagado en posición inicial, sin *scroll* y encendido).
- `void display_on( void )`: Enciende el *display*.
- `void display_off( void )`: Apaga el *display*.
- `void display_cursor_on( void )`: Activa la visualización del cursor.
- `void display_cursor_off( void )`: Desactiva la visualización del cursor.
- `void display_putchar( uint8 x, uint8 y, char ch )`: Escribe un carácter en la posición indicada.
- `void display_puts( uint8 x, uint8 y, char *s )`: Escribe una cadena a partir en la posición indicada.

Las definiciones de los tipos usados se encuentran en el fichero `common_types.h`.

Dado que la comunicación con el LCD deberá tener una temporización adecuada, se utilizará el temporizador 0 integrado en el SoC para medir los tiempos.

3.3. Aspectos de implementación

El estado del LCD se almacenará en 2 variables globales privadas:

```
static uint8 display_status = ON;
static uint8 cursor_status  = OFF;
```

Además de las funciones públicas, privadamente deberán implementarse las siguientes funciones auxiliares:

- `static void display_putnibble( uint8 mode, uint8 nibble )`: escribe los 4 bits menos significativos de `nibble` en el bus paralelo `LCD-D4..D7`, escribe `mode` en la línea de selección de registro (comando/dato) `LCD-RS` y genera un pulso en alta de 1 ms en la línea de habilitación `LCD-E`.
- `static void display_putcommand( uint8 byte )`: envía un comando de 8 bits al LCD como una secuencia de 2 *nibbles* consecutivos, comenzando por el más significativo.
- `static void display_setcursor( uint8 x, uint8 y )`: indica al LCD que sitúe el cursor en la posición indicada, enviando el correspondiente comando (“`1yxxxxxx`”).

La función `display_init` deberá configurar los bits correspondientes del puerto `C` como de salida, pondrá a 0 todas las líneas del control del *display*, realizará una espera de 30 ms y seguidamente configurará el *display* para que use un interfaz de 4 bits, utilice las 2 líneas de caracteres y lo encienda (enviando como comandos la secuencia de *nibbles*: "`0010`", "`0010`" y "`1100`"). Seguidamente enviará la siguiente secuencia de comandos:

1. `Display on`, `cursor off`, `blink off`. ("`00001100`").
2. Borrar el *display* y poner el cursor en posición inicial ("`00000001`").
3. Direccionar la DDRAM y poner a 0 el `address counter` ("`10000000`").
4. `Scroll off` ("`00001010`").

Las funciones `display_on`, `display_off`, `cursor_on` y `cursor_off`, además de actualizar adecuadamente las variables que almacenan en el módulo el estado del LCD, enviarán el correspondiente comando al LCD ("`00001dc1`").

Las funciones `display_putchar` y `display_puts` posicionarán el cursor en la posición indicada por los argumentos y seguidamente enviarán *nibble* a *nibble* (comenzando por el más significativo) el carácter o la cadena indicada.

Los prototipos de las funciones para acceso al temporizador 0 se encuentran en el fichero `timers.h`. Asimismo, los ficheros `s3c44b0x.h` y `s3cev40.h`, respectivamente, definen nemotécnicos para el acceso a los registros de los controladores de dispositivos internos del chip S3C44BOX y a los distintos elementos de la placa de prototipado Embest S3CEV40.

Práctica 4

Control de un zumbador

4.1. Introducción

La placa de expansión incorpora un zumbador (transductor electroacústico) capaz de producir un sonido continuo del mismo tono.

El zumbador se controla mediante una única línea de habilitación (`BUZZER`). Si vale 0, el zumbador suena. Si vale 1, el zumbador permanece en silencio. Para utilizar el zumbador es necesario previamente que los *jumpers* de la placa de expansión estén en la configuración 2 ó 3 (véase tabla 1.2). En esta configuración la línea de control del zumbador está conectada al bit 0 del puerto `E` del SoC:

- `BUZZER : PE0`

4.2. Control software

El módulo *software* (*driver*) para control del zumbador deberá disponer de las siguientes funciones públicas:

- `void buzzer_init( void )`: Inicializa el driver y silencia el zumbador.
- `void buzzer_on( void )`: Pone el zumbador a sonar.
- `void buzzer_off( void )`: Silencia el zumbador.
- `void buzzer_toggle( void )`: Conmuta el estado del zumbador.
- `uint8 buzzer_status( void )`: Devuelve el estado (`on/off`) del zumbador.

Las definiciones de los tipos usados se encuentran en el fichero `common_types.h`.

4.3. Aspectos de implementación

El estado del zumbador se almacenará en una variable global privada:

```
static uint8 state;
```

La función `buzzer_init` deberá configurar el bit correspondiente del puerto `E` como de salida y silenciará el zumbador.

Las funciones `buzzer_on`, `buzzer_off`, `buzzer_toggle` además de actualizar adecuadamente la variable `state`, pondrán a 0, a 1 o conmutarán la línea `BUZZER`.

La función `buzzer_status` devolverá el valor de la variable `state`.

Práctica 5

Control de un altavoz

5.1. Introducción

La placa de expansión incorpora un altavoz (transductor electroacústico) capaz de generar una onda sonora análoga (en frecuencia y amplitud) a un señal eléctrica dada.

El altavoz se controla mediante una única línea de habilitación (`SPEAKER`). Si por ella generamos una señal digital periódica de una frecuencia en el rango audible (alternando 0 y 1 a una periodicidad fija), el altavoz generará una onda sonora de la misma frecuencia.

Para utilizar el altavoz es necesario previamente que los *jumpers* de la placa de expansión estén en la configuración 2 ó 3 (véase tabla 1.2). En estas configuraciones la línea de control del altavoz está conectada al bit 3 del puerto `E` del SoC:

- `SPEAKER : PE3`

5.2. Control software

El módulo *software* (*driver*) para control del altavoz deberá disponer de las siguientes funciones públicas:

- `void speaker_init( void )`: Inicializa el driver y silencia el altavoz.
- `void speaker_on( uint16 period )`: Reproduce un sonido con el periodo indicado (en microsegundos).
- `void speaker_off( void )`: Silencia el altavoz.
- `uint8 speaker_status( void )`: Devuelve el estado (`on/off`) del altavoz.

Las definiciones de los tipos usados se encuentran en el fichero `common_types.h`.

Para facilitar al programador la generación de sonidos, el módulo *software* dispondrá de las siguientes declaraciones públicas:

```
// Macros que definen los periodos en microsegundos de las notas
//   de la octava central (4 octava) del piano
#define DO      (3822)
#define DO_     (3608)
#define RE      (3405)
#define RE_     (3214)
#define MI      (3034)
#define FA      (2864)
#define FA_     (2703)
#define SOL     (2551)
#define SOL_    (2408)
#define LA      (2273)
#define LA_     (2145)
#define SI      (2025)

// Macros que permiten calcular los periodos en microsegundos de
//   las notas en función de la octava a la que pertenecen
#define OCTAVE1( note ) (note << 3)
#define OCTAVE2( note ) (note << 2)
#define OCTAVE3( note ) (note << 1)
#define OCTAVE4( note ) (note)
#define OCTAVE5( note ) (note >> 1)
#define OCTAVE6( note ) (note >> 2)
#define OCTAVE7( note ) (note >> 3)
```

La generación de las señal periódica la realizaremos utilizando el temporizador 0 integrado en el SoC. Esto es posible dado que este temporizador tiene capacidad de generar autónomamente señales periódicas con anchura de pulso programable (PWM) y además el pin 3 del puerto `E` puede conectarse internamente a dicho temporizador. Esto permitirá que el programa principal que utiliza el módulo no se vea bloqueado por la generación del tren de pulsos. Alternativamente, si no se deseara utilizar el temporizador, la generación de sonido podría realizarse por interrupción, en una manera similar a como se realizó el refresco de la matriz de puntos en la primera práctica.

5.3. Aspectos de implementación

El estado del altavoz se almacenará en una variable global privada:

```
static uint8 state;
```

La función `speaker_init` deberá configurar el bit correspondiente del puerto `E` como salida del temporizador 0 e inicializará dicho temporizador.

La función `speaker_on` actualizará adecuadamente la variable `state` y programará el temporizador 0 para que genere una señal periódica cuadrada del periodo indicado en el argumento.

La función `speaker_off` actualizará adecuadamente la variable `state` y parará el temporizador 0.

La función `speaker_status` devolverá el valor de la variable `state`.

Los prototipos de las funciones para acceso al temporizador 0 se encuentran en el fichero `timers.h`. Asimismo, los ficheros `s3c44b0x.h` y `s3cev40.h`, respectivamente, definen nemotécnicos para el acceso a los registros de los controladores de dispositivos internos del chip S3C44BOX y a los distintos elementos de la placa de prototipado Embest S3CEV40.

Práctica 6

Control de LED RGB

6.1. Introducción

La placa de expansión incorpora 3 leds tricolor.

Cada led se controla mediante 3 líneas de habilitación correspondientes a cada uno de los 3 colores (R, G y B). Si se fija a 1 permanentemente la línea correspondiente a un color, el led se ilumina a dicho color con máxima intensidad. Si queremos regular su intensidad, bastará con generar una señal digital periódica con un factor de trabajo (porcentaje de tiempo que está a 1) adecuado. Habilitando más de un color, pueden obtenerse colores compuestos. Si además, cada uno de los colores se habilita con señales periódicas de factores de trabajo diferentes (y por tanto la intensidad cada color es distinta), podrán generar una amplia gama de colores.

Para utilizar los leds RGB es necesario previamente que los *jumpers* de la placa de expansión estén en la configuración 1 (véase tabla 1.2). En esta configuración las líneas de control de los leds tricolor están a los puertos `B`, `C`, `E`, `F` y `G` del SoC en los bits indicados a continuación:

- `R0 : PC8`
- `G0 : PC9`
- `B0 : PC14`
- `R1 : PC15`
- `G1 : PB7`
- `B1 : PF2`
- `R2 : PG4`
- `G2 : PE0`
- `B2 : PE3`

6.2. Control software

El módulo *software* (*driver*) para control de los leds tricolor deberá disponer de las siguientes funciones públicas:

- `void rgbleds_init( void )`: Inicializa el *driver* y apaga los leds.
- `void rgbled_on( uint8 led, color_t *color )`: Enciende el led indicado en el color indicado.
- `void rgbled_off( uint8 led )`: Apaga el led indicado.
- `color_t *rgbled_status( uint8 led )`: Devuelve el color del led indicado.

Las definiciones de los tipos usados se encuentran en el fichero `common_types.h`.

Para facilitar al programador la denominación de los leds y de colores, el módulo *software* dispondrá de las siguientes declaraciones públicas:

```
// Macros que identifican a cada uno de los leds
#define RGB_LED0 (0)
#define RGB_LED1 (1)
#define RGB_LED2 (2)

// Tipo que permite definir el porcentaje de tiempo (0-100) que de cada
// componente del color debe activarse
typedef struct {
        uint8 r, g, b;
} color_t ;
```

6.3. Aspectos de implementación

El estado del cada led se almacenará en una componente de un array global privado:

```
static color\_t state[3];
```

Además de las funciones públicas, privadamente deberá implementarse la siguiente función auxiliar:

- `static void rgbleds_tick( void ) __attribute__ ((interrupt ("IRQ")))`: Rutina de tratamiento de interrupción instalada en el vector de interrupciones por el temporizador 0. Es la encargada de generar las señales PWM que controlan los leds. Para ello, lleva la cuenta (módulo 100) del número de ticks que cada una de las líneas de habilitación de cada led lleva activada. Si dicho número es superior al indicado en el estado del led correspondiente, desactiva la línea. Cada vez que este número vuelve a 0, todas las líneas se activan.

La función `rgbleds_init` deberá configurar los bits correspondientes de los puertos `B`, `C`, `E`, `F` y `G` como salida, apagará todos los leds, instalará la rutina de tratamiento de interrupción para la generación de las señales PWM y arrancará el temporizador 0 para que genere periódicamente interrupciones cada 10 us.

La función `rgbled_on` actualizará adecuadamente la variable de estado del led indicado.

La función `rgbled_off` actualizará adecuadamente la variable de estado del led indicado. Si todos los leds están apagados, parará el temporizador 0.

La función `rgbled_status` devolverá el valor de la variable de estado del led indicado.

Los prototipos de las funciones para acceso al temporizador 0 se encuentran en el fichero `timers.h`. Asimismo, los ficheros `s3c44b0x.h` y `s3cev40.h`, respectivamente, definen nemotécnicos para el acceso a los registros de los controladores de dispositivos internos del chip S3C44BOX y a los distintos elementos de la placa de prototipado Embest S3CEV40.

Práctica 7

Control de un par de dispositivos emisor-receptor de infrarrojos

7.1. Introducción

La placa de expansión incorpora un par emisor-receptor de luz infrarroja. Los receptores de infrarrojos no son capaces de capturar cualquier señal, sino sólo aquéllas que contengan una determinada frecuencia portadora. En este caso, el par emisor-receptor de infrarrojos que utilizaremos trabaja con una frecuencia portadora de 38 KHz que deberá ser modulada por la señal a transmitir.

El led emisor se controla mediante 1 línea de habilitación. Cuando vale 1, emite luz infrarroja; cuando vale 0, no lo hace.

El receptor, filtra la portadora de la señal modulada y la resultante (la señal original) la envía por 1 pin de salida.

Para utilizar el emisor o el receptor de infrarrojos es necesario previamente que los *jumpers* de la placa de expansión estén en la configuración 1 (véase tabla 1.2). En esta configuración las líneas de control y estado de los dispositivos están a los puertos `B` y `G` del SoC en los bits indicados a continuación:

- `LED : PB7`
- `RCV : PG4`

7.2. Control software

El módulo *software* (*driver*) para control del par emisor-receptor deberá disponer de las siguientes funciones públicas:

- `void ir_init( void )`: Inicializa el *driver* y desactiva el led.
- `void irTX_pulseBurst( uint16 us )`: Emite una ráfaga de pulsos a 38 KHz durante el tiempo indicado (medido en us).

- `void irTX_space( uint16 us )`: Desactiva el emisor durante el tiempo indicado (medido en us).
- `uint16 irRX_downLength( void )`: Espera la recepción de un pulso a baja y devuelve el intervalo que ha durado (medido en us).
- `uint16 irRX_upLength( void )`: Espera la recepción de un pulso a alta y devuelve el intervalo que ha durado (medido en us).

Con estas funciones podrán implementarse protocolos de comunicación que usen codificación basada en distancia de pulsos. Por ejemplo, el protocolo de infrarrojos de NEC codifica el 1 lógico como un *pulse burst* de 562.5 us seguido de un *space* de 562.5, y el 0 lógico como un *pulse burst* de 562.5 us seguido de un *space* de 1687.5 us.

Las definiciones de los tipos usados se encuentran en el fichero `common_types.h`.

7.3. Aspectos de implementación

La función `ir_init` deberá configurar el bit correspondiente del puerto `B` como salida, el del puerto `G` como entrada y apagará el led emisor.

La función `irTX_pulseBurst`, usando 2 temporizadores, enviará al led emisor una señal cuadrada de 38 KHz durante el tiempo indicado en el argumento.

La función `irTX_space` apagará el led emisor durante el tiempo indicado en el argumento.

La función `irRX_downLength` esperará la recepción de un flanco de bajada y, usando un temporizador adicional, medirá el tiempo (en us) que la señal permanece a 0.

La función `irRX_upLength` esperará la recepción de un flanco de subida y, usando un temporizador adicional, medirá el tiempo (en us) que la señal permanece a 1.

Los ficheros `s3c44b0x.h` y `s3cev40.h`, respectivamente, definen nemotécnicos para el acceso a los registros de los controladores de dispositivos internos del chip S3C44BOX y a los distintos elementos de la placa de prototipado Embest S3CEV40.

Práctica 8

Control de un motor paso a paso

8.1. Introducción

La placa de expansión incorpora un motor paso a paso de 4 terminales.

El motor se puede hacer girar enviando alternativamente pulsos por cada uno de sus terminales a una frecuencia determinada que determina su velocidad de giro.

Para utilizar el motor paso a paso los *jumpers* de la placa de expansión deben estar en alguna de las tres configuraciones mostradas en la tabla 1.2. Con independencia de la configuración adoptada, los terminales del motor están conectados al puerto `C` del SoC en los bits indicados a continuación:

- `PH_B1 (pink): PC10`
- `PH_A1 (yellow): PC11`
- `PH_B2 (orange): PC12`
- `PH_A2 (blue): PC13`

8.2. Control software

El módulo *software* (*driver*) para control del motor deberá disponer de las siguientes funciones públicas:

- `void stepper_init( uint8 mode )`: Inicializa el *driver* indicando el modo de giro y detiene el motor.
- `void stepper_on( uint8 sense, uint16 sps )`: Gira el motor en el sentido y a la velocidad (en pasos por segundo) indicados.
- `void stepper_off( void )`: Detiene el motor.
- `uint8 stepper_status( void )`: Devuelve el estado (`on/off`) del motor.

Las definiciones de los tipos usados se encuentran en el fichero `common_types.h`.

Para facilitar al programador la especificación del modo y sentido del giro deseado, el módulo *software* dispondrá de las siguientes declaraciones públicas:

```
// Sentido de giro
#define STEPPER_CW     (0)      // Horario
#define STEPPER_CCW    (1)      // Antihorario

// Modo de giro
#define HIGH_TORQUE    (0)      // Full stepping
#define WAVE_STEPPING  (1)      // Full stepping
#define HALF_STEPPING  (2)      // Half stepping
```

La generación de los pulsos de reloj en cada uno de los terminales del motor será efectuado por una rutina de tratamiento de interrupción para permitir que el programa principal que utiliza el módulo no se vea bloqueado por el control del movimiento del motor. Se utilizará el temporizador 0 integrado en el SoC para generar una interrupción periódica a una cadencia dependiente de la velocidad de giro que se desee alcanzar.

8.3. Aspectos de implementación

El estado del motor se almacenará en 3 variables globales privadas:

```
static uint8 state;
static uint8 mode;
static uint8 sense;
```

Las tablas de consulta que almacenan la forma de onda a generar en los terminales según cada modo de giro (en sentido contrareloj) se declaran como constantes globales privadas:

```
static const uint8 high_torque[8] = {
    0x3, 0x6, 0xC, 0x9, 0x3, 0x6, 0xC, 0x9
};

static const uint8 wave_stepping[8] = {
    0x1, 0x2, 0x4, 0x8, 0x1, 0x2, 0x4, 0x8
};

static const uint8 half_stepping[8] = {
    0x1, 0x3, 0x2, 0x6, 0x4, 0xC, 0x8, 0x9
};
```

La tabla actual a usar es apuntada por una variable privada:

```
static const uint8 *lookup_table;
```

Además de las funciones públicas, privadamente deberá implementarse la siguiente función auxiliar:

- `static void stepper_step( void ) __attribute__ ((interrupt ("IRQ")))`: Rutina de tratamiento de interrupción instalada en el vector de interrupciones por el temporizador 0. A cada iniciación, genera un paso en los terminales del motor según lo indicado en la tabla de consulta actual.

La función `stepper_init` deberá configurar los bits correspondientes del puerto `C` como salidas, inicializará el temporizador 0 y apuntará la variable `lookup_table` a la tabla de formas de onda que corresponda según el valor de su argumento.

La función `stepper_on` actualizará adecuadamente las variables de estado, instalará la rutina de tratamiento de interrupción para generación de pasos en el vector correspondiente y arrancará el temporizador 0 para que genere periódicamente interrupciones a la cadencia indicada por el argumento.

La función `stepper_off` actualizará adecuadamente las variables de estado y parará el temporizador 0.

La función `stepper_status` devolverá el valor de la variable `state`.

Los prototipos de las funciones para acceso al temporizador 0 se encuentran en el fichero `timers.h`. Asimismo, los ficheros `s3c44b0x.h` y `s3cev40.h`, respectivamente, definen nemotécnicos para el acceso a los registros de los controladores de dispositivos internos del chip S3C44BOX y a los distintos elementos de la placa de prototipado Embest S3CEV40.

Práctica 9

Control de un ventilador

9.1. Introducción

La placa de expansión incorpora un ventilador controlable por PWM.

El ventilador se controla mediante una única línea de habilitación (`FAN`). Si vale 0, el ventilador permanece parado. Si vale 1, el ventilador gira a máxima velocidad. Si por dicha línea se genera una señal digital periódica de una frecuencia superior al rango audible (mayor de 20 KHz, para evitar ruidos) es posible controlar la velocidad de giro regulando la anchura de pulso.

Para utilizar el ventilador es necesario previamente que los *jumpers* de la placa de expansión estén en la configuración 2 ó 3 (véase tabla 1.2). En estas configuraciones la línea de control del ventilador está conectada al bit 3 del puerto `E` del SoC:

- `FAN : PE3`

Adicionalmente, algunos ventiladores son capaces de generar pulsos con una frecuencia directamente proporcional a al velocidad de giro, estos pulsos puede recibirlos el SoC a través de la línea siguiente:

- `FANRTN: PG5`

9.2. Control software

El módulo *software* (*driver*) para control del ventilador deberá disponer de las siguientes funciones públicas:

- `void fan_init( void )`: Inicializa el *driver* y detiene el ventilador.
- `void fan_on( uint8 pw )`: Pone a girar el ventilador y a un porcentaje dado de la velocidad máxima.
- `void fan_off( void )`: Detiene el ventilador.

- `uint8 fan_status( void )`: Devuelve el estado (`on/off`) del ventilador.

Las definiciones de los tipos usados se encuentran en el fichero `common_types.h`.

La generación de la señal periódica la realizaremos utilizando el temporizador 0 integrado en el SoC. Esto es posible dado que este temporizador tiene capacidad de generar autónomamente señales periódicas con anchura de pulso programable (PWM) y además el pin 3 del puerto `E` puede conectarse internamente a dicho temporizador. Esto permitirá que el programa principal que utiliza el módulo no se vea bloqueado por la generación del tren de pulsos. Alternativamente, si no se deseara utilizar el temporizador, el control del ventilador podría realizarse por interrupción, en una manera similar a como se realizó el refresco de la matriz de puntos en la primera práctica.

9.3. Aspectos de implementación

El estado del ventilador se almacenará en una variable global privada:

```
static uint8 state;
```

La función `fan_init` deberá configurar el bit correspondiente del puerto `E` como salida del temporizador 0 e inicializará dicho temporizador.

La función `fan_on` actualizará adecuadamente la variable `state`, programará el temporizador 0 para que genere una señal periódica cuadrada de unos 25 KHz (aprox.) y con el ancho de pulso indicado en el argumento.

La función `fan_off` actualizará adecuadamente la variable `state` y parará el temporizador 0.

La función `fan_status` devolverá el valor de la variable `state`.

Los prototipos de las funciones para acceso al temporizador 0 se encuentran en el fichero `timers.h`. Asimismo, los ficheros `s3c44b0x.h` y `s3cev40.h`, respectivamente, definen nemotécnicos para el acceso a los registros de los controladores de dispositivos internos del chip S3C44BOX y a los distintos elementos de la placa de prototipado Embest S3CEV40.

Práctica 10

Control de un expansor de puertos IIC

10.1. Introdución

La placa de expansión incorpora 2 unidades del chip PCF8574, que es un expansor de 8 puertos bidireccionales de E/S controlable por bus serie IIC.

Para controlar este chip, bastará direccionarlo y enviar/recibir los datos a escribir/leer de los puertos a través del bus IIC.

Para utilizar este chip los *jumpers* de la placa de expansión deben estar en cualquiera de las tres configuraciones mostradas en la tabla 1.2. Con independencia de la configuración adoptada, las líneas del bus IIC están conectados al puerto `F` del SoC en los bits indicados a continuación:

- `IICSCL: PF0`
- `IICSDA: PF1`

Adicionalmente, este chip es capaz de generar interrupciones que el SoC recibe a través de la línea siguiente:

- `IICINT: PG5`

10.2. Control software

El módulo *software* (*driver*) para control del expansor de puertos deberá disponer de las siguientes funciones públicas:

- `void pcf8574_init( void )`: Inicializa el *driver* y pone a 1 los puertos.
- `void pcf8574_putbyte( uint8 dev, uint8 data )`: Escribe el argumento en los 8 bits del puerto indicado.
- `uint8 pcf8574_getbyte( uint8 dev )`: Lee el valor de los 8 bits del puerto indicado.

Las definiciones de los tipos usados se encuentran en el fichero `common_types.h`.

Para facilitar al programador la denominación de los puertos, el módulo *software* dispondrá de las siguientes declaraciones públicas:

```
// Macros que identifican a cada uno de los dispositivos
#define pcf8574_UP   (0)
#define pcf8574_DOWN (1)
```

10.3. Aspectos de implementación

La función `pcf8574_init` deberá inicializar el *driver* de IIC y enviar `0xff` a cada uno de los chips pcf8574.

La función `pcf8574_putbyte` iniciará como *master* un flujo de transmisión con el chip indicado direccionando el dispositivo (`0x70` para pcf8574_UP y `0x71` para pcf8574_DOWN), enviará el argumento y cerrará la transmisión.

La función `pcf8574_getbyte` iniciará como *master* un flujo de recepción con el chip inidicado direccionando el dispositivo (`0x70` para pcf8574_UP y `0x71` para pcf8574_DOWN), esperará a recibir un dato y cerrará la transmisión.

Los prototipos de las funciones para acceso al bus IIC se encuentran en el fichero `iic.h`. Asimismo, los ficheros `s3c44b0x.h` y `s3cev40.h`, respectivamente, definen nemotécnicos para el acceso a los registros de los controladores de dispositivos internos del chip S3C44BOX y a los distintos elementos de la placa de prototipado Embest S3CEV40.

Práctica 11

Control de un conversor A/D IIC

11.1. Introdución

La placa de expansión incorpora 2 unidades del chip PCF8591, que es un conversor A/D y D/A de 8 bits con 4 canales de entrada y 1 de salida controlable por bus serie IIC.

Para controlar este chip, bastará direccionarlo y enviar/recibir los datos a escribir/leer de los conversores a través del bus IIC.

Para utilizar el este chip los *jumpers* de la placa de expansión deben estar en cualquiera de las tres configuraciones mostradas en la tabla 1.2. Con independencia de la configuración adoptada, las líneas del bus IIC están conectados al puerto `F` del SoC en los bits indicados a continuación:

- `IICSCL: PF0`
- `IICSDA: PF1`

Adicionalmente, este chip es capaz de generar interrupciones que el SoC recibe a través de la línea siguiente:

- `IICINT: PG5`

11.2. Control software

El módulo *software* (*driver*) para control del conversor AD/DA deberá disponer de las siguientes funciones públicas:

- `void pcf8591_init ( void )`: Inicializa el *driver* y establece el modo de funcionamiento de los conversores.
- `void pcf8591_putsample( uint8 dev, uint8 data )`: Envía una muestra al conversor D/A indicado.

- `uint8 pcf8591_getsample( uint8 dev, uint8 ch )`: Lee una muestra del canal (0-3) del conversor A/D indicado.

Las definiciones de los tipos usados se encuentran en el fichero `common_types.h`.

Para facilitar al programador la denominación de los puertos, el módulo *software* dispondrá de las siguientes declaraciones públicas:

```
// Macros que identifican a cada uno de los dispositivos
#define pcf8591_UP   (0)
#define pcf8591_DOWN (1)
```

11.3. Aspectos de implementación

La función `pcf8591_init` deberá inicializar el *driver* de IIC, iniciar como *master* un flujo de transmisión con el chip direccionando el dispositivo (`0x90` para pcf8591_UP y `0x91` para pcf8591_DOWN), configurar el chip (*DAC on, 4 single-ended inputs, no auto-increment, ch0*) enviando `0x40` y cerrar la trasmisión.

La función `pcf8591_putsample` iniciará como master un flujo de transmisión con el chip direccionando el dispositivo (`0x90` para pcf8591_UP y `0x91` para pcf8591_DOWN), configurará nuevamente el chip (*DAC on, 4 single-ended inputs, no auto-increment, ch0*) enviando `0x40`, enviará el argumento y cerrará la trasmisión.

La función `pcf8591_getsample` iniciará como *master* un flujo de transmisión con el chip pcf8591 direccionando el dispositivo (`0x90` para pcf8591_UP y `0x91` para pcf8591_DOWN), configurará nuevamente el chip (*DAC on, 4 single-ended inputs, no auto-increment, ch0*) enviando `0x40`, iniciará como *master* un flujo de recepción con el chip pcf8574 direccionando el dispositivo (`0x90`), esperará a recibir un dato vacío, esperará a recibir la muestra y cerrará la transmisión.

Los prototipos de las funciones para acceso al bus IIC se encuentran en el fichero `iic.h`. Asimismo, los ficheros `s3c44b0x.h` y `s3cev40.h`, respectivamente, definen nemotécnicos para el acceso a los registros de los controladores de dispositivos internos del chip S3C44BOX y a los distintos elementos de la placa de prototipado Embest S3CEV40.