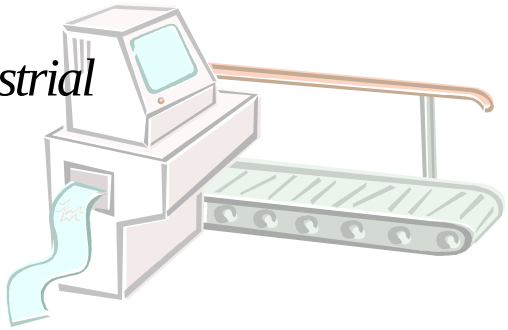
Página Web: https://web08.uah.es/
e-mail: eliseo.garcia@uah.es
Despacho: N-245
Departamento de Automática

Práctica 3

Puertos de Entrada/Salida

Laboratorio de Informática Industrial



Puertos de Entrada/Salida

Introducción

El microcontrolador LPC1768

Introducción

La familia de microcontroladores LPC17xx está basada en el ARM Cortex-M3. Esta familia está diseñada para usarse en aplicaciones que requieren un alto nivel de integración y un bajo consumo. El procesador ARM Cortex-M3 es un núcleo de nueva generación que ofrece mejoras en el sistema como fortalecimiento de las características de depuración y un mayor nivel de apoyo a la integración de bloques. Entre sus características principales se tienen las siguientes:

- Trabaja a una frecuencia de CPU de hasta 100 MHz.
- La CPU Cortex-M3 incorpora pipeline de 3-estados y usa arquitectura Harvard con bus de datos e instrucciones separados, además incorpora un tercer bus para comunicación con los periféricos.
- El LPC17xx incluye los siguientes periféricos:
  - Hasta 512 kB de memoria flash.
  - Hasta 64 kB de memoria de datos.
  - Ethernet MAC.
  - USB interfase.
  - Un controlador de DMA de 8 canales.
  - 4 UARTs.
  - 2 canales CAN, 2 controladores SSP, y un SPI interfase.
  - 3 I2C interfaces.
  - I2S interfase.
  - Un ADC de 12 bits y 8 canales de entrada y un DAC de 10-bit.
  - 4 timers de propósito general.
  - 6 salidas de PWM.
  - Un reloj de tiempo real de bajo consumo con batería separada.
  - Hasta 70 pines de entrada/salida de propósito general.
- Se comercializa en dos tipos de encapsulado: LQFP100 y TFBGA100, ver Figuras 1 y 2.

De forma general, en los microcontroladores, un pin se puede utilizar para realizar varias funciones, de acuerdo con los periféricos que se usen. Por ello, generalmente, es necesario configurar algunos registros donde se indique la funcionalidad del pin. Además, en el caso de los puertos de entrada/salida también será necesario indicar el sentido de las líneas de los mismos (entradas o salida).

En la figura 1 se puede observar como el pin 1 tiene dos funcionalidades, el pin 2 tiene tres y el pin 3 tiene una sola. Esto implica la necesidad de que exista la forma de configurar o seleccionar la función que va tener cada pin.

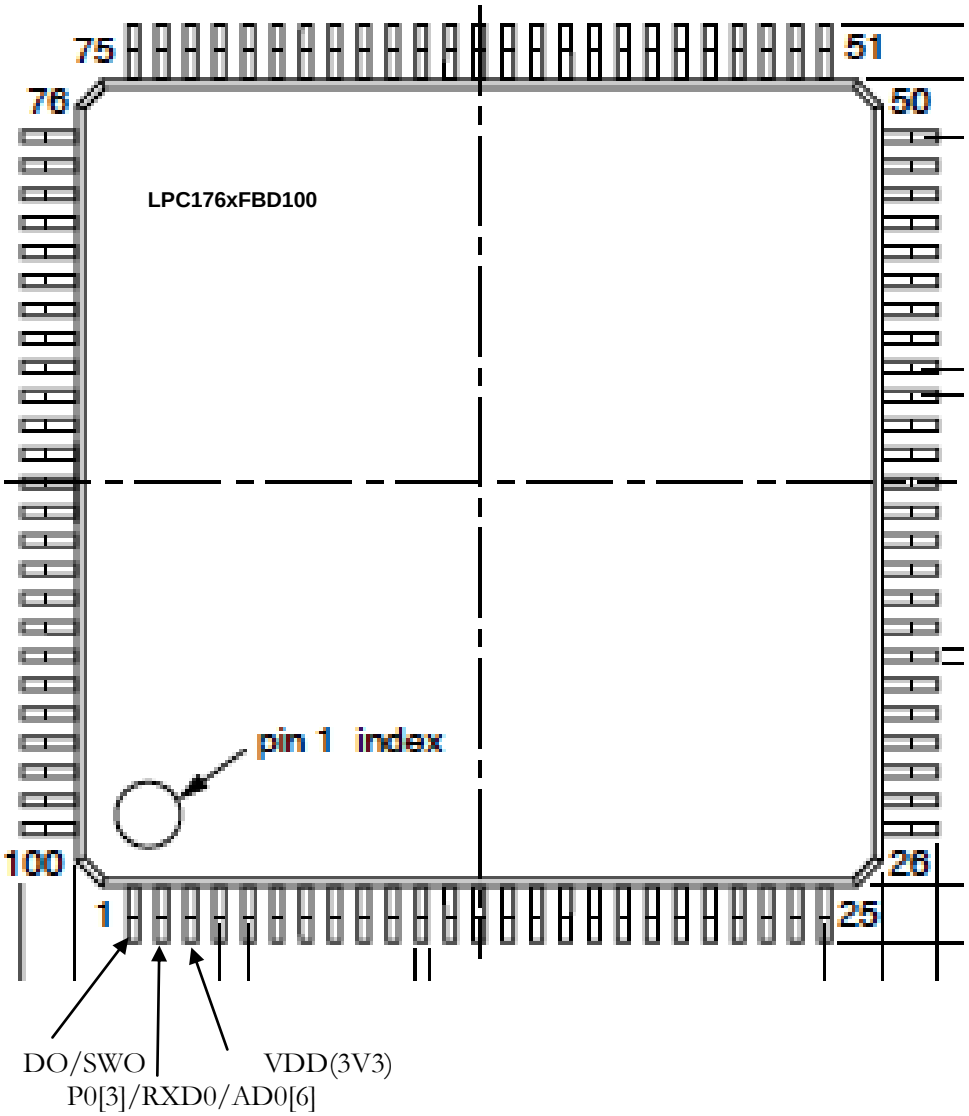


Figura 1. Encapsulado LQFP100.

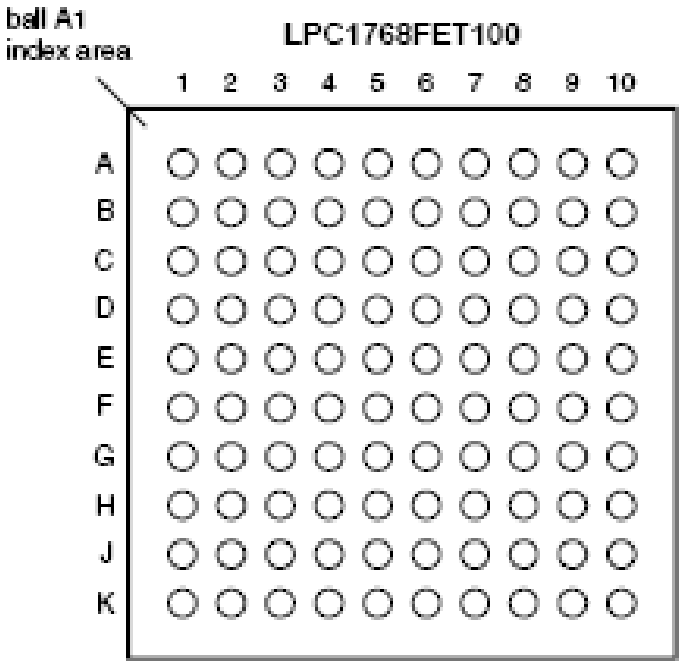


Figura 2. Encapsulado TFBGA1008.

Bloque de conexión de los pines

El bloque de conexión de los pines (*Pin connect block*) permite que un mismo pin del microcontrolador pueda realizar más de una función. Para ello este bloque contiene un conjunto de registros de configuración que controlan a unos multiplexores que conectan al pin con los periféricos del chip. Los periféricos deben estar conectados a las patillas adecuadas antes de ser activado y antes de que cualquier interrupción relacionada esté habilitada.

Existen dos tipos de registros de configuración:

- Registros de selección de la función del pin (PINSEL).
- Registros de selección del modo de conexión del pin (PINMODE).

Registro de selección de funciones (PINSELx)

Los registros PINSELx controlan las funciones de los pines del dispositivo como se muestra a continuación. Mediante dos bits en estos registros se asigna la función de los pines de dispositivo específico.

Nota: En lo que sigue se utilizarán varias tablas extraídas del manual de usuario del LPC17xx para describir la funcionalidad de los registros. Estas tablas en ocasiones se han simplificado, por lo que ante cualquier duda se recomienda recurrir a dicho manual.

Table 75. 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	

Ejemplo: El registro **PINSEL0** (Dir 0x4002 C000) controla las funciones de la parte baja (P0.0 a P0.15) del puerto 0. Si se escribe **00** (que a su vez es el valor por defecto) en los 2 bit menos significativos (mSB) del registro **PINSEL0**, se conecta el **pin P0.0** a la **línea 0** del **puerto 0** de entrada/salida (**GPIO Port 0.0**); si se escribe **01**, se conecta a la **línea RD1** del controlador **CAN1;** si se escribe **10**, se conecta a la **línea TXD3** de la **UART3;** si se escribe **11**, se conecta la **línea SDA1** del bus **I²C1.** Ver las tablas 73, 75 y 79 del manual de usuario.

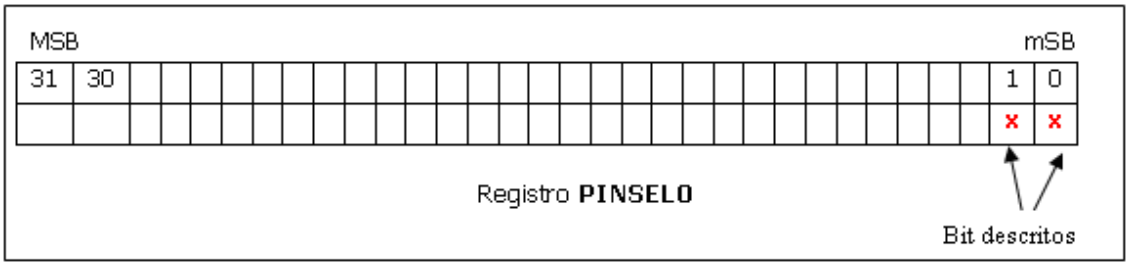


Table 79. Pin function select register 0 (PINSEL0 - address 0x4002 C000) bit description

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
⋮	⋮				⋮	⋮
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

Registros de selección del modo de conexión del pin (PINMODE)

Para controlar el modo de un pin del puerto se utilizan tres bits: dos en un registro **PINMODE**, y uno adicional en un registro **PINMODE_OD**.

Como se puede observar en la Tabla 76, configurando los registros PINMODE0:9 se puede conectar a la entrada de cada pin la resistencia interna de *pull-up* (00), de *pull-down* (11), ninguna de las dos resistencias (10) o seleccionar el modo repetidor (01). En el modo repetidor si a la entrada se aplica un nivel alto, se conecta el *pull-up* y se aplica un nivel bajo, se conecta el *pull-down*. Esto permite mantener el último valor aplicado.

Table 76. 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.	

Los registros PINMODE_OD0:4 se usan para configurar la salida en colector abierto o de forma normal.

Table 77. Open Drain Pin Mode Select register Bits

PINMODE_OD0 to PINMODE_OD4 Values	Function	Value after Reset
0	Pin is in the normal (not open drain) mode.	00
1	Pin is in the open drain mode.	

Aunque normalmente, el valor de PINMODE0:9 se aplica a un pin cuando está en el modo de entrada, puede ser usado conjuntamente con los registros PINMODE_OD0:4 para configurar una salida en drenador abierto y utilizar la resistencia de *pull-up* interna.

Cuando un pin se encuentra en el modo de drenador abierto, causado por un 1 en el bit correspondiente de uno de los registros PINMODE_OD, el modo de entrada no se aplica cuando el pin está sacando un 0. Sin embargo, cuando el valor del PIN es de 1, el PINMODE se aplica ya que este estado apaga el driver del pin de salida. Esto permite la posibilidad de configurar un pin en drenador abierto con la resistencia *pull-up* interna. El *pull-up* en este caso que sólo está activo cuando el pin no está forzado a un nivel bajo por la salida propia del pin.

Puertos de entrada/salida de propósito general

Para utilizar los pines como puertos de entrada/salida de propósito general (GPIOs), hay que tener en cuenta lo siguiente:

- La alimentación debe estar en todo momento habilitada.
- Hay que seleccionar los pines en su función GPIO.

Todos los registros que hacen referencia a los GPIOs son accesibles en tamaño palabra, media palabra ó byte; y que por defecto todos los puertos están configurados como entradas con "pull-up" tras el reset.

La familia de microcontroladores LPC17xx dispone de cinco puertos de propósito general P0, P1, P2, P3 y P4, con diferentes cantidades de líneas. En la tabla 100 se muestra un resumen de los mismos.

Table 100. GPIO pin description

Pin Name	Type	Description
P0[30:0][1]; P1[31:0][2]; P2[13:0]; P3[26:25]; P4[29:28]	Input/ Output	General purpose input/output. These are typically shared with other peripherals functions and will therefore not all be available in an application. Packaging options may affect the number of GPIOs available in a particular device. Some pins may be limited by requirements of the alternate functions of the pin. For example, the pins containing the I^2C0 functions are open-drain for any function selected on that pin. Details may be found in Section 7.1.1.

[1] P0[14:12] are not available.

[2] P1[2], P1[3], P1[7:5], P1[13:11] are not available.

Los puertos GPIO son controlados por un conjunto de registros que se encuentran localizados en el bus de periféricos.

En la siguiente tabla se muestra el mapa de registros:

Table 101. GPIO register map (local bus accessible registers - enhanced GPIO features)

Generic Name	Description	Access	Reset value[1]	PORTn Register Name & Address
FIODIR	Fast GPIO Port Direction control register. This register individually controls the direction of each port pin.	R/W	0	FIO0DIR - 0x2009 C000 FIO1DIR - 0x2009 C020 FIO2DIR - 0x2009 C040 FIO3DIR - 0x2009 C060 FIO4DIR - 0x2009 C080
FIOMASK	Fast Mask register for port. Writes, sets, clears, and reads to port (done via writes to FIOPIN, FIOSET, and FIOCLR, and reads of FIOPIN) alter or return only the bits enabled by zeros in this register.	R/W	0	FIO0MASK - 0x2009 C010 FIO1MASK - 0x2009 C030 FIO2MASK - 0x2009 C050 FIO3MASK - 0x2009 C070 FIO4MASK - 0x2009 C090
FIOPIN	Fast Port Pin value register using FIOMASK. The current state of digital port pins can be read from this register, regardless of pin direction or alternate function selection (as long as pins are not configured as an input to ADC). The value read is masked by ANDing with inverted FIOMASK. Writing to this register places corresponding values in all bits enabled by zeros in FIOMASK. **Important:** if an FIOPIN register is read, its bit(s) masked with 1 in the FIOMASK register will be read as 0 regardless of the physical pin state.	R/W	0	FIO0PIN - 0x2009 C014 FIO1PIN - 0x2009 C034 FIO2PIN - 0x2009 C054 FIO3PIN - 0x2009 C074 FIO4PIN - 0x2009 C094
FIOSET	Fast Port Output Set register using FIOMASK. This register controls the state of output pins. Writing 1s produces highs at the corresponding port pins. Writing 0s has no effect. Reading this register returns the current contents of the port output register. Only bits enabled by 0 in FIOMASK can be altered.	R/W	0	FIO0SET - 0x2009 C018 FIO1SET - 0x2009 C038 FIO2SET - 0x2009 C058 FIO3SET - 0x2009 C078 FIO4SET - 0x2009 C098
FIOCLR	Fast Port Output Clear register using FIOMASK. This register controls the state of output pins. Writing 1s produces lows at the corresponding port pins. Writing 0s has no effect. Only bits enabled by 0 in FIOMASK can be altered.	WO	0	FIO0CLR - 0x2009 C01C FIO1CLR - 0x2009 C03C FIO2CLR - 0x2009 C05C FIO3CLR - 0x2009 C07C FIO4CLR - 0x2009 C09C

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

El motivo de que existan 32 bits para cada puerto, aun cuando no estén todos disponibles, es que es necesario para hacerlo compatible con los productos de la familia LPC2300 de ARM.

Con el fin de acceder más rápidamente al GPIO para modificar su funcionamiento, es posible escribir en tamaño byte o media palabra, en lugar de tamaño palabra, en sus registros de configuración. Para ello hay que referirse con un nombre de registro u otro según el tamaño y la parte del registro a modificar.

No hay que olvidar que cada bit de los registros de configuración hace referencia a un pin del puerto, que el bit sea '0' ó '1' será lo que decida el funcionamiento del pin. A continuación se describen los registros de configuración de los GPIOs:

Registro de dirección de datos (FIOxDIR)

El contenido de este registro determina si los pines que funcionan como GPIOs son de entrada ó de salida. Un '0' indica que funciona como entrada, y un '1' indica que el GPIO funciona como salida.

Para acceder al puerto completo, se nombra el registro como FIO**x**DIR, siendo **x** el número de puerto (de 0 a 4). Para acceder a media palabra, se usa FIO**x**DIR**L** (parte baja **L**) y FIOxDIR**U** (parte alta **U**) y para acceder a un byte se nombra FIO**x**DIR**n**, donde **n** es el número de byte (0 es el byte más bajo y 3 el byte más alto).

Table 103. Fast GPIO port Direction register FIO0DIR to FIO4DIR - addresses 0x2009 C000 to 0x2009 C080) bit description

Bit	Symbol	Value	Description	Reset value
31:0	FIO0DIR FIO1DIR FIO2DIR FIO3DIR FIO4DIR		Fast GPIO Direction PORTx control bits. Bit 0 in FIOxDIR controls pin Px.0, bit 31 in FIOxDIR controls pin Px.31.	0x0
		0	Controlled pin is input.	
		1	Controlled pin is output.	

Table 104. Fast GPIO port Direction control byte and half-word accessible register description

Generic Register name	Description	Register length (bits) & access	Reset value	PORTn Register Address & Name
FIOxDIR0	Fast GPIO Port x Direction control register 0. Bit 0 in FIOxDIR0 register corresponds to pin Px.0 … bit 7 to pin Px.7.	8 (byte) R/W	0x00	FIO0DIR0 - 0x2009 C000 FIO1DIR0 - 0x2009 C020 FIO2DIR0 - 0x2009 C040 FIO3DIR0 - 0x2009 C060 FIO4DIR0 - 0x2009 C080
FIOxDIR1	Fast GPIO Port x Direction control register 1. Bit 0 in FIOxDIR1 register corresponds to pin Px.8 … bit 7 to pin Px.15.	8 (byte) R/W	0x00	FIO0DIR1 - 0x2009 C001 FIO1DIR1 - 0x2009 C021 FIO2DIR1 - 0x2009 C041 FIO3DIR1 - 0x2009 C061 FIO4DIR1 - 0x2009 C081
FIOxDIR2	Fast GPIO Port x Direction control register 2. Bit 0 in FIOxDIR2 register corresponds to pin Px.16 … bit 7 to pin Px.23.	8 (byte) R/W	0x00	FIO0DIR2 - 0x2009 C002 FIO1DIR2 - 0x2009 C022 FIO2DIR2 - 0x2009 C042 FIO3DIR2 - 0x2009 C062 FIO4DIR2 - 0x2009 C082
FIOxDIR3	Fast GPIO Port x Direction control register 3. Bit 0 in FIOxDIR3 register corresponds to pin Px.24 … bit 7 to pin Px.31.	8 (byte) R/W	0x00	FIO0DIR3 - 0x2009 C003 FIO1DIR3 - 0x2009 C023 FIO2DIR3 - 0x2009 C043 FIO3DIR3 - 0x2009 C063 FIO4DIR3 - 0x2009 C083

Table 104. Fast GPIO port Direction control byte and half-word accessible register description

Generic Register name	Description	Register length (bits) & access	Reset value	PORTn Register Address & Name
FIOxDIRL	Fast GPIO Port x Direction control Lower half-word register. Bit 0 in FIOxDIRL register corresponds to pin Px.0 ... bit 15 to pin Px.15.	16 (half-word) R/W	0x0000	FIO0DIRL - 0x2009 C000 FIO1DIRL - 0x2009 C020 FIO2DIRL - 0x2009 C040 FIO3DIRL - 0x2009 C060 FIO4DIRL - 0x2009 C080
FIOxDIRU	Fast GPIO Port x Direction control Upper half-word register. Bit 0 in FIOxDIRU register corresponds to Px.16 ... bit 15 to Px.31.	16 (half-word) R/W	0x0000	FIO0DIRU - 0x2009 C002 FIO1DIRU - 0x2009 C022 FIO2DIRU - 0x2009 C042 FIO3DIRU - 0x2009 C062 FIO4DIRU - 0x2009 C082

***FIO0DIR*: bit 0 controla P0.0…bit 30 controla P0.30**
***FIO1DIR*: bit 16 controla P1.16…bit 30 controla P1.30**

Ejemplo: *FIO1DIR = 0x 00FF0000; pines 16-23 del puerto 1 como salida*

Port Pin value register (FIOxPIN)

Este registro se puede usar con los puertos que estén llevando a cabo funcionalidades digitales, aunque los pines no están configurados como GPIO.

La lectura de éste registro entrega el valor del pin al que hace referencia cada bit, devolviendo un '1' si el pin está a nivel alto, y un '0' si está a nivel bajo. Si el pin está llevando a cabo funciones analógicas (como por ejemplo la entrada al ADC) no se podrá leer de él.

La escritura en este puerto transmite el valor escrito en cada bit a su pin correspondiente. Al escribir un '0' se produce un nivel bajo en el pin referenciado, y un '1' produce un nivel alto en el mismo pin.

Al escribir en este registro se evita tener que usar los registros FIOxSET y FIOxCLR, por lo que hay que ser cuidadoso, especialmente cuando se escribe en todo el puerto a la vez.

El acceso al puerto por parte de este registro está condicionado por el contenido del registro FIOxMASK.

Table 110. Fast GPIO port Pin value byte and half-word accessible register description

Generic Register name	Description	Register length (bits) & access	Reset value	PORTn Register Address & Name
FIOxPIN0	Fast GPIO Port x Pin value register 0. Bit 0 in FIOxPIN0 register corresponds to pin Px.0 … bit 7 to pin Px.7.	8 (byte) R/W	0x00	FIO0PIN0 - 0x2009 C014 FIO1PIN0 - 0x2009 C034 FIO2PIN0 - 0x2009 C054 FIO3PIN0 - 0x2009 C074 FIO4PIN0 - 0x2009 C094
FIOxPIN1	Fast GPIO Port x Pin value register 1. Bit 0 in FIOxPIN1 register corresponds to pin Px.8 … bit 7 to pin Px.15.	8 (byte) R/W	0x00	FIO0PIN1 - 0x2009 C015 FIO1PIN1 - 0x2009 C035 FIO2PIN1 - 0x2009 C055 FIO3PIN1 - 0x2009 C075 FIO4PIN1 - 0x2009 C095
FIOxPIN2	Fast GPIO Port x Pin value register 2. Bit 0 in FIOxPIN2 register corresponds to pin Px.16 … bit 7 to pin Px.23.	8 (byte) R/W	0x00	FIO0PIN2 - 0x2009 C016 FIO1PIN2 - 0x2009 C036 FIO2PIN2 - 0x2009 C056 FIO3PIN2 - 0x2009 C076 FIO4PIN2 - 0x2009 C096

Port output Set register (FIOxSET)

Este registro sirve para poner un '1' en los pines configurados como GPIO en modo de salida. Al escribir un '1' en el bit correspondiente, el pin al que hace referencia se pone a nivel alto. Escribir un '0' no tiene ningún efecto. Si el pin referenciado no funciona en modo GPIO, lo que se escriba no tiene ningún efecto (aunque sea un '1').

Al leer de este registro se recibe el valor del mismo, que se verá influenciado por las escrituras que se han llevado a cabo previamente en este mismo registro, el registro FIOxCLR y el registro FIOxPIN. Hay que tener en cuenta que el valor que retorna la lectura es el valor del registro, no necesariamente el del pin (puede darse el caso de que el pin no funcione como GPIO, por ejemplo).

Table 105. Fast GPIO port output Set register (FIO0SET to FIO4SET - addresses 0x2009 C018 to 0x2009 C098) bit description

Bit	Symbol	Value	Description	Reset value
31:0	FIO0SET FIO1SET FIO2SET FIO3SET FIO4SET		Fast GPIO output value Set bits. Bit 0 in FIOxSET controls pin Px.0, bit 31 in FIOxSET controls pin Px.31.	0x0
		0	Controlled pin output is unchanged.	
		1	Controlled pin output is set to HIGH.	

Table 106. Fast GPIO port output Set byte and half-word accessible register description

Generic Register name	Description	Register length (bits) & access	Reset value	PORTn Register Address & Name
FIOxSET0	Fast GPIO Port x output Set register 0. Bit 0 in FIOxSET0 register corresponds to pin Px.0 … bit 7 to pin Px.7.	8 (byte) R/W	0x00	FIO0SET0 - 0x2009 C018 FIO1SET0 - 0x2009 C038 FIO2SET0 - 0x2009 C058 FIO3SET0 - 0x2009 C078 FIO4SET0 - 0x2009 C098
FIOxSET1	Fast GPIO Port x output Set register 1. Bit 0 in FIOxSET1 register corresponds to pin Px.8 … bit 7 to pin Px.15.	8 (byte) R/W	0x00	FIO0SET1 - 0x2009 C019 FIO1SET1 - 0x2009 C039 FIO2SET1 - 0x2009 C059 FIO3SET1 - 0x2009 C079 FIO4SET1 - 0x2009 C099
FIOxSET2	Fast GPIO Port x output Set register 2. Bit 0 in FIOxSET2 register corresponds to pin Px.16 … bit 7 to pin Px.23.	8 (byte) R/W	0x00	FIO0SET2 - 0x2009 C01A FIO1SET2 - 0x2009 C03A FIO2SET2 - 0x2009 C05A FIO3SET2 - 0x2009 C07A FIO4SET2 - 0x2009 C09A
FIOxSET3	Fast GPIO Port x output Set register 3. Bit 0 in FIOxSET3 register corresponds to pin Px.24 … bit 7 to pin Px.31.	8 (byte) R/W	0x00	FIO0SET3 - 0x2009 C01B FIO1SET3 - 0x2009 C03B FIO2SET3 - 0x2009 C05B FIO3SET3 - 0x2009 C07B FIO4SET3 - 0x2009 C09B
FIOxSETL	Fast GPIO Port x output Set Lower half-word register. Bit 0 in FIOxSETL register corresponds to pin Px.0 … bit 15 to pin Px.15.	16 (half-word) R/W	0x0000	FIO0SETL - 0x2009 C018 FIO1SETL - 0x2009 C038 FIO2SETL - 0x2009 C058 FIO3SETL - 0x2009 C078 FIO4SETL - 0x2009 C098
FIOxSETU	Fast GPIO Port x output Set Upper half-word register. Bit 0 in FIOxSETU register corresponds to Px.16 … bit 15 to Px.31.	16 (half-word) R/W	0x0000	FIO0SETU - 0x2009 C01A FIO1SETU - 0x2009 C03A FIO2SETU - 0x2009 C05A FIO3SETU - 0x2009 C07A FIO4SETU - 0x2009 C09A

Port output Clear register (FIOxCLR)

Este registro funciona de manera análoga al registro FIOxSET, solo que ahora consiste en poner las salidas a '0' en vez de a '1'. Escribiendo un '1' en el bit correspondiente del puerto, el pin referenciado se pone a '0'. Además en el registro FIOxSET, el bit correspondiente a ese pin, también se pone a '0'. Escribir un '0' no tiene ningún efecto. Si el pin referenciado no funciona como GPIO, ninguna escritura tiene ningún efecto. Éste registro es de sólo escritura.

El acceso al puerto por parte de este registro está condicionado por el contenido del registro FIOxMASK.

Table 107. Fast GPIO port output Clear register (FIO0CLR to FIO4CLR- addresses 0x2009 C01C to 0x2009 C09C) bit description

Bit	Symbol	Value	Description	Reset value
31:0	FIO0CLR FIO1CLR FIO2CLR FIO3CLR FIO4CLR		Fast GPIO output value Clear bits. Bit 0 in FIOxCLR controls pin Px.0, bit 31 controls pin Px.31.	0x0
		0	Controlled pin output is unchanged.	
		1	Controlled pin output is set to LOW.	

Table 108. Fast GPIO port output Clear byte and half-word accessible register description

Generic Register name	Description	Register length (bits) & access	Reset value	PORTn Register Address & Name
FIOxCLR0	Fast GPIO Port x output Clear register 0. Bit 0 in FIOxCLR0 register corresponds to pin Px.0 … bit 7 to pin Px.7.	8 (byte) WO	0x00	FIO0CLR0 - 0x2009 C01C FIO1CLR0 - 0x2009 C03C FIO2CLR0 - 0x2009 C05C FIO3CLR0 - 0x2009 C07C FIO4CLR0 - 0x2009 C09C
FIOxCLR1	Fast GPIO Port x output Clear register 1. Bit 0 in FIOxCLR1 register corresponds to pin Px.8 … bit 7 to pin Px.15.	8 (byte) WO	0x00	FIO0CLR1 - 0x2009 C01D FIO1CLR1 - 0x2009 C03D FIO2CLR1 - 0x2009 C05D FIO3CLR1 - 0x2009 C07D FIO4CLR1 - 0x2009 C09D
FIOxCLR2	Fast GPIO Port x output Clear register 2. Bit 0 in FIOxCLR2 register corresponds to pin Px.16 … bit 7 to pin Px.23.	8 (byte) WO	0x00	FIO0CLR2 - 0x2009 C01E FIO1CLR2 - 0x2009 C03E FIO2CLR2 - 0x2009 C05E FIO3CLR2 - 0x2009 C07E FIO4CLR2 - 0x2009 C09E
FIOxCLR3	Fast GPIO Port x output Clear register 3. Bit 0 in FIOxCLR3 register corresponds to pin Px.24 … bit 7 to pin Px.31.	8 (byte) WO	0x00	FIO0CLR3 - 0x2009 C01F FIO1CLR3 - 0x2009 C03F FIO2CLR3 - 0x2009 C05F FIO3CLR3 - 0x2009 C07F FIO4CLR3 - 0x2009 C09F
FIOxCLRL	Fast GPIO Port x output Clear Lower half-word register. Bit 0 in FIOxCLRL register corresponds to pin Px.0 … bit 15 to pin Px.15.	16 (half-word) WO	0x0000	FIO0CLRL - 0x2009 C01C FIO1CLRL - 0x2009 C03C FIO2CLRL - 0x2009 C05C FIO3CLRL - 0x2009 C07C FIO4CLRL - 0x2009 C09C
FIOxCLRU	Fast GPIO Port x output Clear Upper half-word register. Bit 0 in FIOxCLRU register corresponds to pin Px.16 … bit 15 to Px.31.	16 (half-word) WO	0x0000	FIO0CLRU - 0x2009 C01E FIO1CLRU - 0x2009 C03E FIO2CLRU - 0x2009 C05E FIO3CLRU - 0x2009 C07E FIO4CLRU - 0x2009 C09E

Port Mask register (FIOxMASK)

Este registro se usa para enmascarar las acciones que pueden llevar a cabo los registros FIOxSET, FIOxCLR y FIOxPIN. Esto quiere decir que puede eliminar la capacidad de leer o escribir en estos puertos.

Un '0' en un bit de este registro provoca que el pin al que hace referencia, pueda verse afectado por los valores de los registros anteriormente mencionados.

Un '1' en un bit de este registro desconecta físicamente la conexión entre el pin referenciado en ese bit, y los registros FIOxSET, FIOxCLR y FIOxPIN.

Si además se utilizan los puertos 0 y 2 para generar interrupciones (sólo puede hacerse con estos dos puertos) hay que tener en cuenta otra serie de registros.

Para conocer si algún puerto tiene una interrupción pendiente, hay que leer del registro IOIntStatus. El resto de los registros están duplicados, uno para el puerto 0 y otro para el puerto 2, de manera que cada bit del registro se corresponde con un pin del puerto.

Si se desea eliminar alguna interrupción que esté pendiente en algún pin del puerto, puede hacerse mediante el registro IOxIntClr. En los registros IOxIntStatR y IOxIntStatF se puede comprobar qué pin ha recibido una interrupción por flanco de subida ó de bajada respectivamente.

Table 111. Fast GPIO port Mask register (FIO0MASK to FIO4MASK - addresses 0x2009 C010 to 0x2009 C090) bit description

Bit	Symbol	Value	Description	Reset value
31:0	FIO0MASK FIO1MASK FIO2MASK FIO3MASK FIO4MASK		Fast GPIO physical pin access control.	0x0
		0	Controlled pin is affected by writes to the port's FIOxSET, FIOxCLR, and FIOxPIN register(s). Current state of the pin can be read from the FIOxPIN register.	
		1	Controlled pin is not affected by writes into the port's FIOxSET, FIOxCLR and FIOxPIN register(s). When the FIOxPIN register is read, this bit will not be updated with the state of the physical pin.	

Ejemplo: Configuración, escritura y lectura de puertos

```
#include <stdio.h>
#include <LPC17xx.H>    /* Definiciones para el LPC17xx */

main()
{
  int value;
  LPC_GPIO0 -> FIODIR = 0x0000000F;  //Los pines P0.[0..3] configurados como salidas,
                                     //P0.[4..31] como entradas

  LPC_GPIO0->FIOCLR=0x0000000F;  //Ponemos a cero los pines P0.[0..3].

 //También serviría para poner los pines P0.[0..3] a cero: LPC_GPIO0->FIOPIN=0x00000000;

  LPC_GPIO0->FIOSET=0x00000001; //Activar la salida P0.0

  LPC_GPIO0->FIOSET=0x0000000A; //Activar las salidas P0.1 y P0.3

  LPC_GPIO0->FIOSET |= ((1<<3)| (1<<1));

  LPC_GPIO0->FIOCLR=0x00000001; //Desactivar la misma salida P0.0

  value = ((LPC_GPIO0->FIOPIN & (1 <<4))>> 4); // Lee el estado del pin P0.4 (Nota 1)

}
```

Nota 1: Recuerde las operaciones lógicas y de desplazamiento en lenguaje C. A continuación se muestra a modo de ejemplo la última instrucción.

31			**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
0		0	0	0	0	1	0	0	0	0	1 <<4

&

31			**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
x		x	x	x	x	X	x	x	x	x	LPC_GPIO0

=

31			**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
0		0	0	0	0	X	0	0	0	0	

value =

31			**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
0		0	0	0	0	0	0	0	0	X	>> 4

Ficheros de inclusión stdint.h y LPC17xx.h. Algunos aspectos importantes.

stdint.h

El estándar C99 incluye la definición de nuevos tipos de datos con el objetivo de facilitar la portabilidad de códigos. Estos nuevos tipos están declarados en el fichero cabecera *stdint.h*:

```
   /* exact-width signed integer types */
typedef  signed          char int8_t;    // 8 bits
typedef  signed short    int int16_t;    // 16 bits
typedef  signed          int int32_t;    // 32 bits
typedef  signed          __int64 int64_t;  // 64 bits

   /* exact-width unsigned integer types */
```

```
typedef unsigned          char uint8_t;
typedef unsigned short    int uint16_t;
typedef unsigned          int uint32_t;
typedef unsigned          __int64 uint64_t;
```

LPC17xx.h

Para facilitar la programación en lenguaje C de los microcontroladores de la familia LPC17xx, ARM proporciona el fichero **LPC17xx.h.** En dicho fichero podemos encontrar la definición de los registros estudiados anteriormente, utilizando la misma nomenclatura que en Manual de Usuario del LPC17xx.

En el caso de los puertos GPIO, la estructura LPC_GPIO_TypeDef define la estructura de dichos puertos. Como se puede ver en la tabla siguiente, dicha estructura está formada por varias uniones, donde se definen los registros de configuración. En dichas uniones podemos ver que se declaran los registros enteros, divididos en parte alta y parte baja, y divididos en cuatro bytes.

```
0193  typedef struct
0194  {
0195    union {
0196      __IO uint32_t FIODIR;
0197      struct {
0198        __IO uint16_t FIODIRL;
0199        __IO uint16_t FIODIRH;
0200      };
0201      struct {
0202        __IO uint8_t  FIODIR0;
0203        __IO uint8_t  FIODIR1;
0204        __IO uint8_t  FIODIR2;
0205        __IO uint8_t  FIODIR3;
0206      };
0207    };
0208    uint32_t RESERVED0[3];
0209    union {
0210      __IO uint32_t FIOMASK;
0211      struct {
0212        __IO uint16_t FIOMASKL;
0213        __IO uint16_t FIOMASKH;
0214      };
0215      struct {
0216        __IO uint8_t  FIOMASK0;
0217        __IO uint8_t  FIOMASK1;
0218        __IO uint8_t  FIOMASK2;
0219        __IO uint8_t  FIOMASK3;
0220      };
0221    };
..............................
0248    union {
0249      __O  uint32_t FIOCLR;
0250      struct {
0251        __O  uint16_t FIOCLRL;
0252        __O  uint16_t FIOCLRH;
0253      };
0254      struct {
0255        __O  uint8_t  FIOCLR0;
0256        __O  uint8_t  FIOCLR1;
0257        __O  uint8_t  FIOCLR2;
0258        __O  uint8_t  FIOCLR3;
0259      };
0260    };
0261  } LPC_GPIO_TypeDef;
```

Registro FIODIR entero

Parte baja y alta del registro FIODIR

Los cuatro bytes del registro FIODIR

Registro FIOCLR entero

Parte baja y alta del registro FIOCLR

Los cuatro bytes del registro FIOCLR

En el fichero **LPC17xx.h,** también se encuentran definidas:

- La dirección base de la zona de memoria de los GPIO:

```
#define LPC_GPIO_BASE        (0x2009C000)
```

- La dirección base de cada puerto:

```
#define LPC_GPIO0_BASE       (LPC_GPIO_BASE + 0x00000)
#define LPC_GPIO1_BASE       (LPC_GPIO_BASE + 0x00020)
#define LPC_GPIO2_BASE       (LPC_GPIO_BASE + 0x00040)
#define LPC_GPIO3_BASE       (LPC_GPIO_BASE + 0x00060)
#define LPC_GPIO4_BASE       (LPC_GPIO_BASE + 0x00080)
```

- Punteros del tipo LPC_GPIO_TypeDef a la dirección base del puerto:

```
#define LPC_GPIO0        ((LPC_GPIO_TypeDef    *) LPC_GPIO0_BASE)
#define LPC_GPIO1        ((LPC_GPIO_TypeDef    *) LPC_GPIO1_BASE)
#define LPC_GPIO2        ((LPC_GPIO_TypeDef    *) LPC_GPIO2_BASE)
#define LPC_GPIO3        ((LPC_GPIO_TypeDef    *) LPC_GPIO3_BASE)
#define LPC_GPIO4        ((LPC_GPIO_TypeDef    *) LPC_GPIO4_BASE)
```

Las definiciones vistas anteriormente brindan la posibilidad de acceder a los diferentes registros de los puertos utilizando una nomenclatura sencilla. Esto se puede ver en el siguiente ejemplo:

```
LPC_GPIO0->FIOSET=0x0000000A; //Activar las salidas P0.1 y P0.3
LPC_GPIO0->FIOSET |= ((1<<3)| (1<<1));
```

En ambas líneas de código se escribe un '1' en los bits 1 y 3 del registro FIOSET del Puerto 0, sin necesidad de recordar la localización de memoria de dicho registro..

Simulacion de los puertos en el analizador lógico

En este apartado se continua con la presentación del manejo del entorno de desarrollo Keil µVision. En este caso se trata de la utilización del analizador lógico para visualizar señales o variables en el dominio del tiempo, uso de puertos y *breakpoints*.

Utilización del analizador lógico

El simulador dispone de un analizador lógico que permite ver variables y señales de puertos en el dominio del tiempo. Para visualizar las variables en el analizador, estas deben ser estáticas o globales; no locales. A continuación se muestran los pasos para visualizar con el analizador lógico la variable *i* que se actualiza constantemente en el bucle *while*:

- Cree un nuevo proyecto y compile el siguiente programa:

```
#include <LPC17xx.H>      /* Definiciones para el LPC17xx */
#define CONST_25ms_100MHz 500000

int32_t i,on_off;

//Función que de demora
void Retardo(int32_t tiempo)
{
  int32_t x;

  //El bucle for tarda 25 ms a 100MHz.
 for(x=0;x<tiempo;x++);
}
int main(void)
{
  while(1){
    if(i==0) {
      i=1;
      Retardo(CONST_25ms_100MHz);
   }
    else{
      i=0;
      Retardo(CONST_25ms_100MHz/2);
    }
   }
}
```

- Inicia la simuación del programa.
- Haga click sobre el icono  para visualizar el analizador lógico.
- Introduzca la variable *i* en el analizador, simplemente arrastrándola y soltándola en la ventana del analizador. Con un conjunto de variables, procederíamos del mismo modo, arrastrándolas y soltándolas de una en una.
- Ajuste la escala vertical haciendo click en el botón *Setup…* de la ventana del analizador y rellene con *0x0* y *0x1* las cajas de texto *Min:* y *Max:*, respectivamente. Acepte y cierre haciendo click en el botón *Close*. También se puede ajustar la escala vertical abriendo el menú contextual sobre el analizador y seleccionando la opción 'Bit'.
- Ajuste la escala de tiempo a 20 ns/div (*Grid:* 20 ns) haciendo varios clicks sobre el botón *In* del *Zoom* (o situándose sobre el valor de Grid y moviendo la rueda del ratón).
- Habilite la casilla *Signal Info* para obtener información al colocar el puntero sobre la señal.
- Ejecute paso a paso hasta entrar en el bucle *while* y observe la señal cuadrada que aparece en la ventana del analizador lógico a medida que cambia el valor de la variable *i* en el bucle *while*.
- Haga click sobre cualquier flanco de la señal para colocar una marca y luego colocar el puntero del ratón sobre otro flanco para visualizar la diferencia de tiempos (*Delta*).

NOTA: Otra forma de especificar directamente un bit de una variable en el analizador lógico, por ejemplo el bit 0 de la variable *i*, sería la siguiente: hacer click en  de la ventana de setup del analizador lógico (figura 1) e introducir el texto *i.0*.

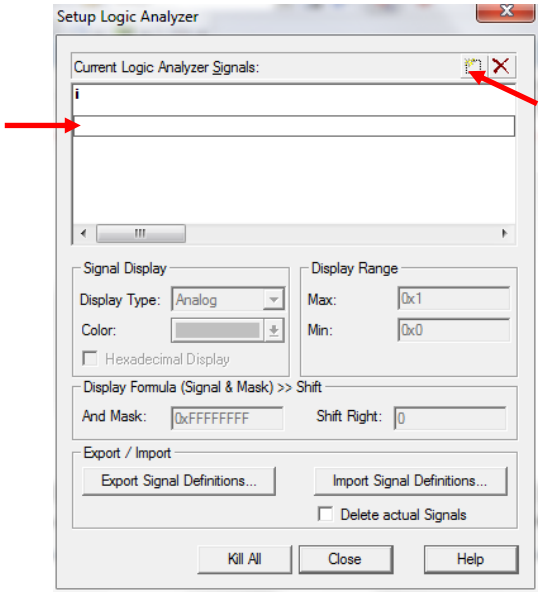


Figura 1. Añadir variable desde la ventana de setup del analizador.

En la figura 2 aparece un ejemplo que muestra una porción de la señal sobre el analizador configurado con una escala de 0.1us/div. Es importante resaltar que el microcontrolador ofrece varias posibilidades como fuente de reloj para su funcionamiento. Tras el reset, queda seleccionado como reloj por defecto la salida de un oscilador RC de 4MHz que posee internamente, pero la selección de un cristal de 12 MHz (figura 3), junto a los ficheros que hemos incluido para el arranque en la entrada StartUp del árbol de proyecto nos garantizan que la velocidad de reloj interna en la CPU sea de 100 MHz.

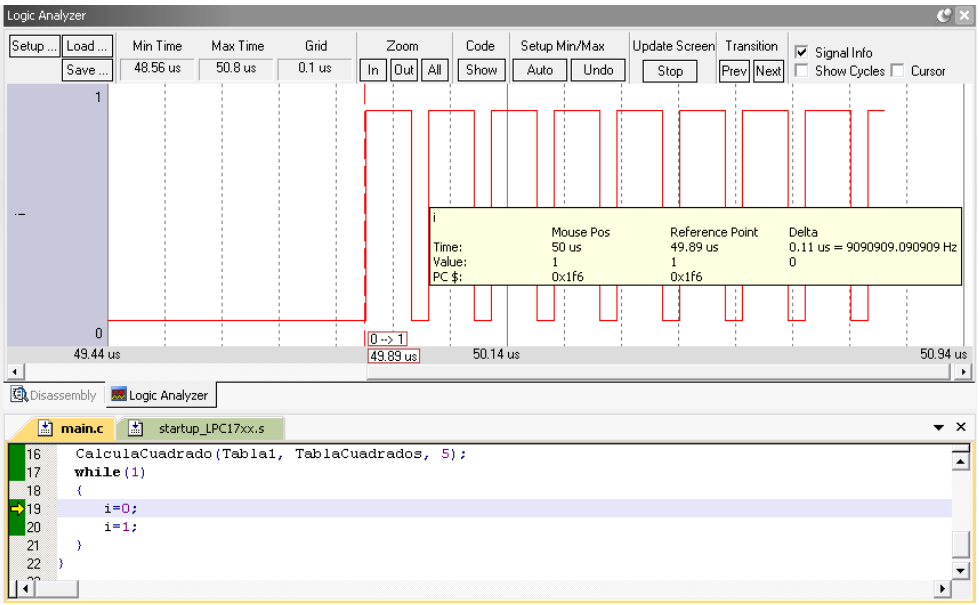


Figura 2. Ejemplo que visualiza la variable i sobre el analizador lógico.

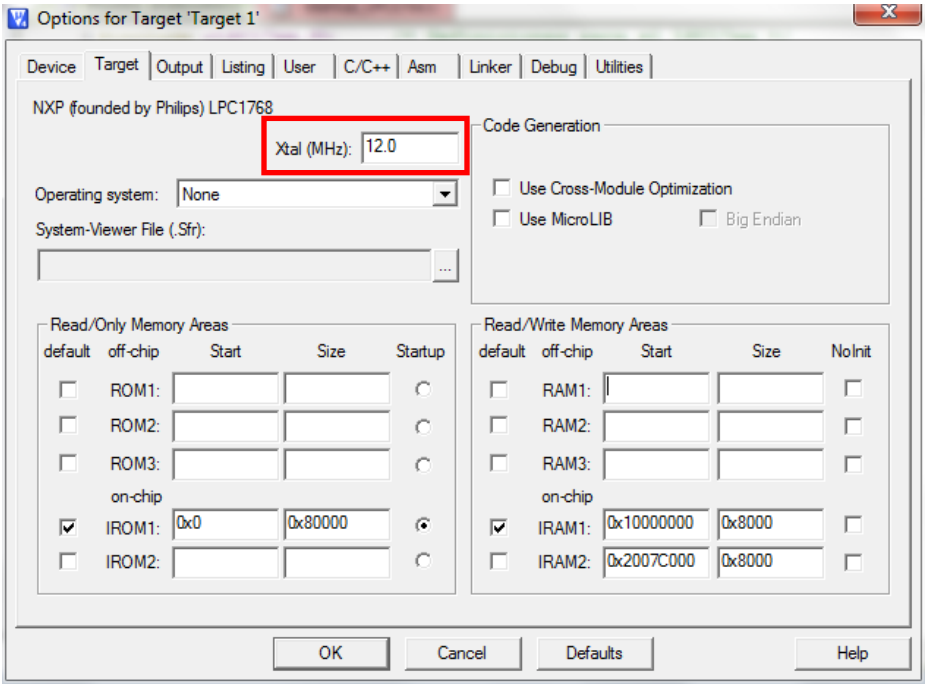


Figura 3. Configuración de la frecuencia del cristal en las opciones del proyecto.

Ejemplo de utilización de los puertos

En este caso se presenta un ejemplo que hace uso del puerto 0 para activar cuatro pines del mismo de manera secuencial, seleccionando el sentido de activación de acuerdo al valor de una variable que podrá ser modificada de forma interactiva en el simulador. Las señales serán visualizadas mediante el analizador lógico.

Comience cerrando el proyecto actual (*Project→Close Project*) y creando uno nuevo como ya se ha explicado en la práctica 2. Llame *Ejemplo2* al nuevo proyecto y almacénelo en una carpeta diferente, ya que también llamaremos *main.c* al fichero fuente que usará el nuevo proyecto, pero será uno diferente que debemos tener preparado en la nueva carpeta (no es necesario que sea definitivo, puesto que siempre podemos editarlo en la ventana del entorno de µVision).

Código fuente

```
#include <LPC17xx.H>      /* Definiciones para el LPC17xx */
#define CONST_25ms_100MHz 500000

int32_t sentido;

//Función que establece el tiempo que una salida del puertopermanecerá activada.
void Retardo(int32_t tiempo)
{
  int32_t i;
  for(i=0;i<tiempo;i++);    //El bucle for tarda 25 ms a 100MHz.
}

int main(void)
{
  int32_t n;

  LPC_GPIO0->FIODIR=0x0000000F;  // Los pines P0.[0..3]  configurados como salidas,
                                 //           P0.[4..31] configurados como entradas.

  LPC_GPIO0->FIOCLR=0x0000000F;  // Se ponen a cero los pines P0.[0..3].

    while(1){
    if(sentido) {
      n=n<<1;
      if(n==(1<<4)) n=1;
      }
    else{
      n=n>>1;
      if(n==0) n=(1<<3);
      }
    LPC_GPIO0->FIOPIN=(n<<0);         // Activar la salida que corresponda.
    Retardo(CONST_25ms_100MHz);       //Retardo de 25 ms.
      }
}
```

Visualización de puertos en el analizador lógico

Una vez compilado el proyecto sin errores inicie el simulador, muestre la ventana del analizador lógico y configurelo para visualizar los cuatro bits de menor peso del puerto 0 P0.[3..0]. Para poder visualizar estos cuatro bits, primero buscamos el puerto 0 (*FIO0PIN*) haciendo click en el símbolo '+' de la entrada *Peripheral Registers* (ventana *Symbols*) para expandirla (figura 4). Como *FIO0PIN* hace referencia conjunta a todos los bits del puerto 0, arrastraremos y soltaremos hasta cuatro veces *FIO0PIN* a la ventana del analizador lógico y a continuación seleccionaremos un bit diferente en cada variable *FIO0PIN*.

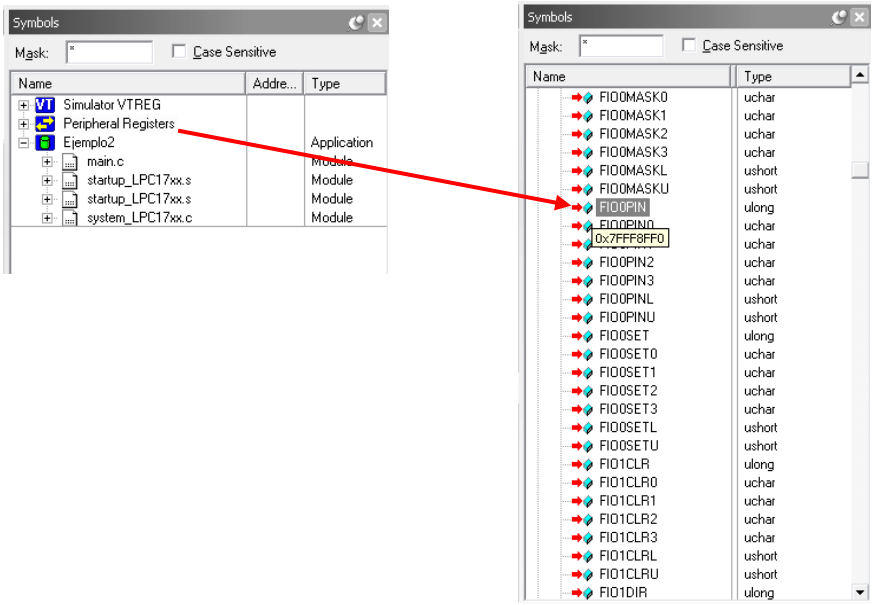


Figura 4. Localización del puerto P0 en la ventana Symbols.

Para configurar el bit a visualizar pulse en *Setup* de la ventana del analizador. La figura 5 ilustra este proceso para seleccionar el bit de menor peso (b0) de la primera variable *FIO0PIN*, configurando el valor 0x00000001 en el campo *And Mask* y estableciendo un rango de amplitud *Min/Max* de 0x0-0x1. Siguiendo un proceso similar, configuramos el resto de variables. Así, para seleccionar el bit b3 en la *FIO0PIN* inferior escribimos 0x00000008 en su campo *And Mask*.

NOTA: Otra forma de especificar directamente un pin de un puerto en el analizador lógico, por ejemplo el pin 18 del puerto 1, sería la siguiente: hacer click en  de la ventana del analizador lógico (figura 1) e introducir el texto FIO1PIN.18, o bien PORT1.18 (esta última sintaxis sólo es válida usando el simulador, pero no lo reconocería el emulador JLINK (unknown pin)).

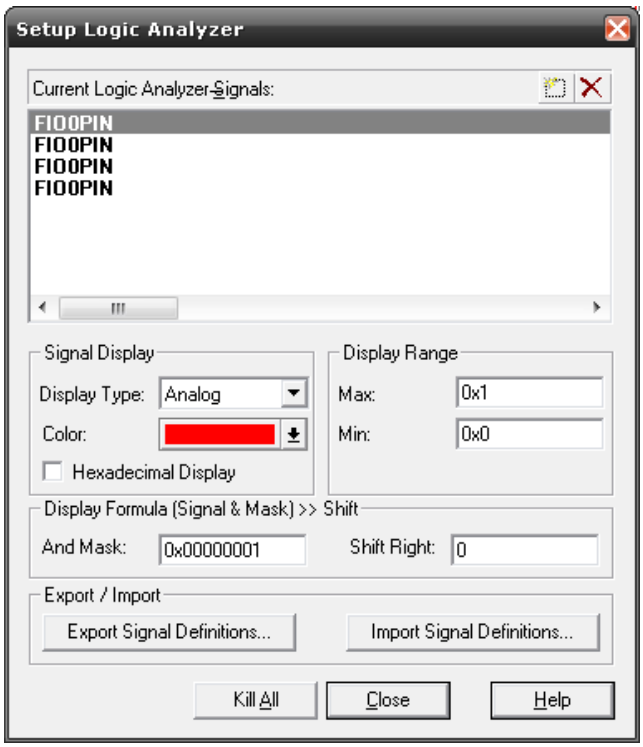


Figura 5. Selección del bit b0 del puerto P0 y configuración de su escala.

Modificación interactiva de variables

Para poder interactuar con el programa y así modificar la variable *sentido* en tiempo de ejecución y que cambie el sentido de desplazamiento en la activación de los pines P0.[0..3], crearemos una ventana compuesta de dos botones: '*right*' y '*left*'. Al hacer click sobre ellos asignarán los valores '*sentido=0*' y '*sentido=1*', respectivamente. Para ello, comenzamos visualizando la ventana Command haciendo click sobre el icono  , si no estaba ya visualizada en el interface gráfico del simulador.

Escriba a continuación del símbolo de prompt (>) de la ventana *Comand* el comando *define button "right","sentido=0"* y pulse *enter* (figura 6).

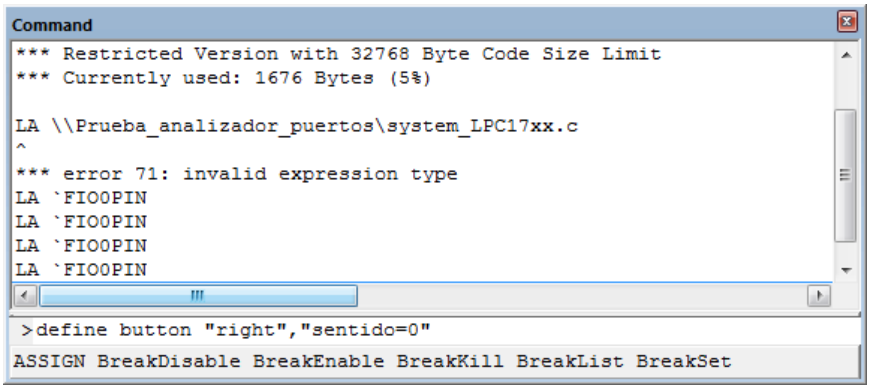


Figura 6. Introducción de una orden en la ventana de *Command*.

A continuación teclee el otro comando *define button "left","sentido=1"* y pulse *enter*. El resultado es la ventana con dos botones que aparece en la figura 7. Tenga presente que esto sólo tiene sentido en el simulador. En una aplicación ejecutándose sobre una placa con un microcontrolador real se podría controlar el sentido de activación, leyendo el valor que se introduzca por otro pin de un puerto determinado.

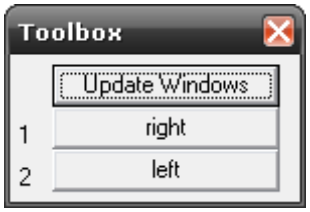


Figura 7. Ventana con dos botones para interactuar con el programa.

Ejecute el programa paso a paso y vea cómo evolucionan las señales en el analizador lógico. Experimente pulsando los botones *right* y *left* y vea cómo afectan a las señales en el analizador. Reflexione sobre los cambios que se producen. En la figura 8 se visualiza una porción de las señales cuando la variable *sentido* vale cero.

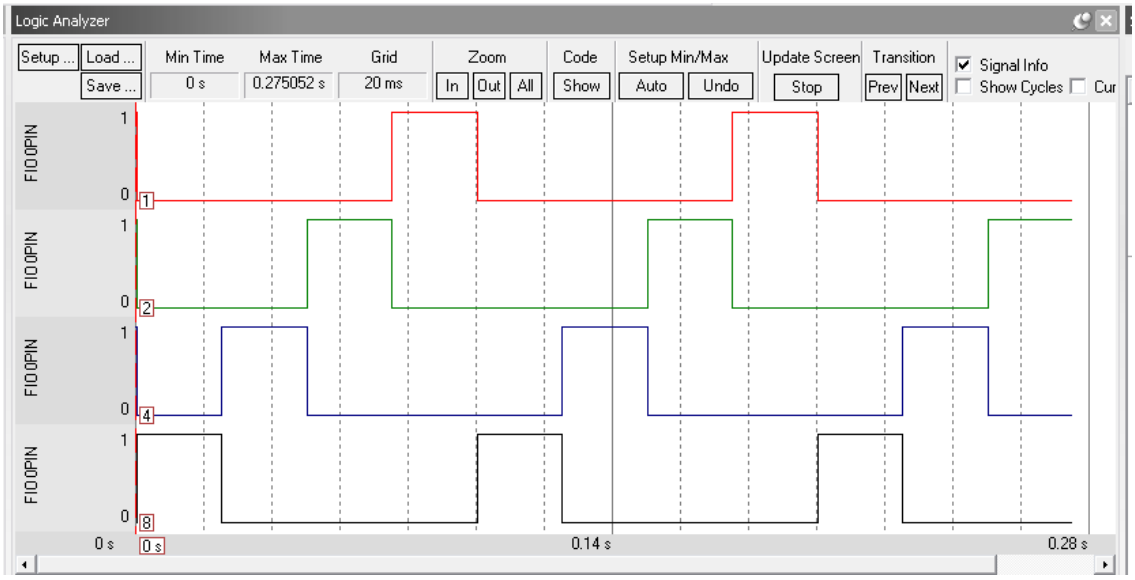


Figura 8. Señales P0.[0..3] cuando la variable *sentido* vale cero.

Otra posibilidad de visualizar las señales del puerto P0 es mostrando su ventana a través del menú *Peripherals*→*GPIO Fast Interface*→*Port 0* (figura 9).

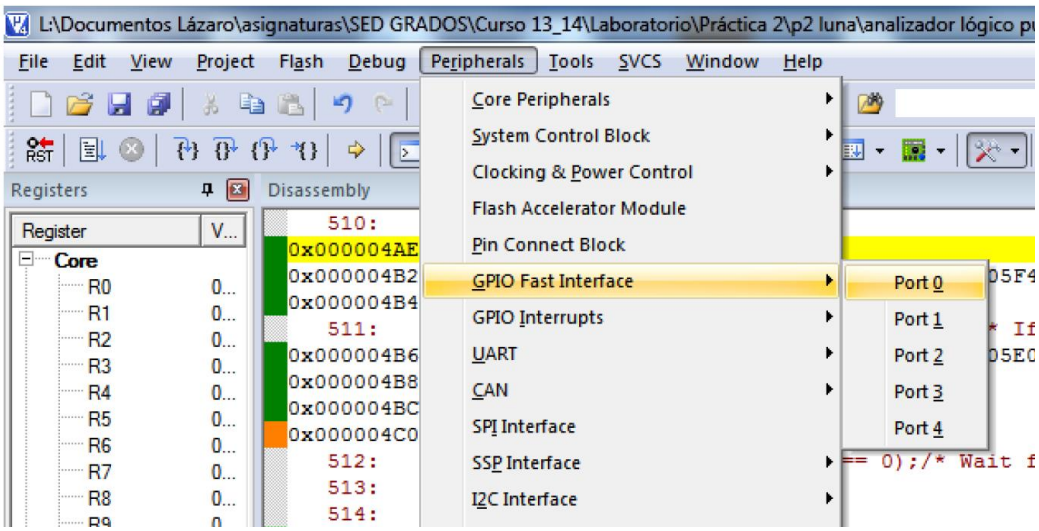


Figura 9. Visualización de la ventana del puerto 0.

La ventana que aparece es la de la figura 10. Ejecute el programa paso a paso y observe cómo se activan progresivamente los pines P0.[0..3].

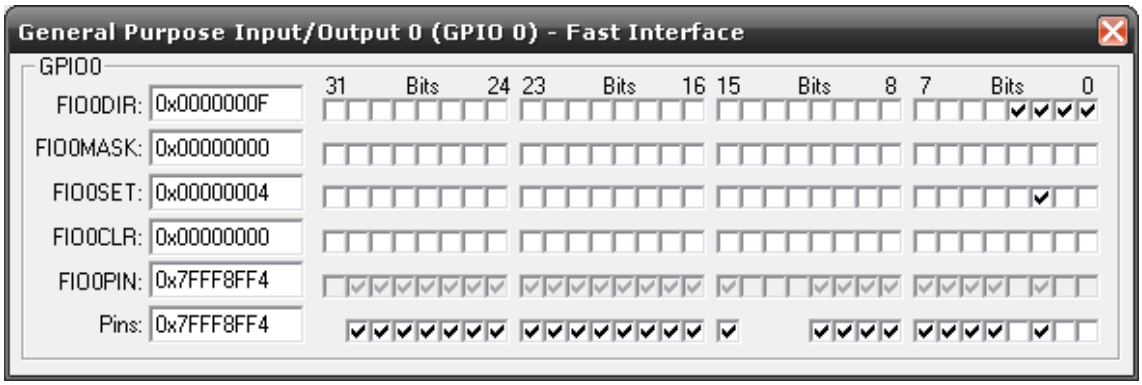


Figura 10. Ventana de visualización del puerto P0.

Utilización de Breakpoints

Los *breakpoints* o puntos de ruptura son marcas que podemos colocar delante de las líneas de código para ayudarnos a depurarlo durante la simulación. Evitan, por ejemplo, tener que ejecutar paso a paso una porción de código que lleva mucho tiempo y sabemos a priori que funciona correctamente. De este modo, nos permiten ejecutar el programa y que éste quede detenido cuando se encuentra una línea que hemos marcado en el simulador con un *breakpoint*. A partir de ahí, podemos seguir ejecutando paso a paso para depurar cuidadosamente el código que sigue o seguir ejecutando el programa hasta que se encuentre el siguiente breakpoint, etc. Para insertar un *breakpoint*, sitúese en una línea de código y hacer click en **Insert/Remove Breakpoint (F9)**.

Existen muchas otras posibilidades de utilizar *breakpoints* a través de la ventana de comandos, por ejemplo, condicionalmente. Se recomienda consultar el comando *BreakSet* en la ayuda del entorno µVision, pestaña 'índice'.

Carga del programa a la tarjeta mediante el modo ISP

Una de las formas de cargar el programa a la tarjeta de desarrollo *Blueboard* es mediante comunicación serie asíncrona a través de la UART0 (modo ISP) y el programa FlashMagic de NXP, que ha de estar instalado en el PC. Para ello, los pasos que hay que seguir son:

1. Conecte el cable USB-serie (GND, TX, RX) a las líneas de la UART0 (**TxD0**, **RxD0**) y a la masa (**GND**) del **conector J3** de la tarjeta, tal como muestra la figura 11.

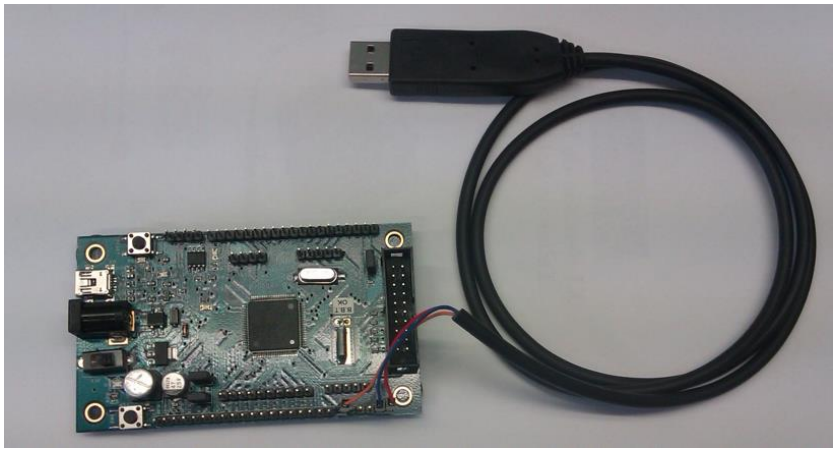


Figura 11. Tarjeta Blueboard y cable USB-serie.

2. **Entre en modo ISP**. Se lleva a cabo manteniendo pulsado un breve instante de tiempo el pulsador **SW2** (mantiene P2.10 a masa), tras hacer un reset mediante el pulsador SW1.

   NOTA: En algunas tarjetas no está soldada de fábrica la resistencia R27 (0 ohmios) por lo que para entrar en modo ISP hay que hacer un puente con un cable (hembra-hembra) entre el pin P2.10 (BSL) y masa (GND) de la tarjeta.

3. Cree un proyecto llamado Ejemplo3, e introduzca en el fichero main.c el siguiente contenido:

```
// ********************************************************
// Proyecto: Parpadea un LED
// Archivo: main.c
// ********************************************************
#include "lpc17xx.h"

// ********************************************************
//    Funciones de retardo
// ********************************************************
void delay(uint32_t n)
{
  int32_t i;
  for(i=0;i<n;i++);
}

int main (void)
{
  LPC_GPIO1->FIODIR |= (1<<18);      // P1.18 definido como salida
  LPC_GPIO1->FIOCLR |= (1<<18);      // P1.18 apagado

  while(1) {
     LPC_GPIO1->FIOPIN |= (1<<18);  // Enciendo LED
     delay (1000000);
     LPC_GPIO1->FIOPIN &= ~(1<<18); // Apago LED
     delay (1000000);
    }
}
```

4. No olvide compilar el proyecto "ejemplo 3" con la opción "create HEX file", en la pestaña **Output** del menú de opciones del proyecto (Figura 12).

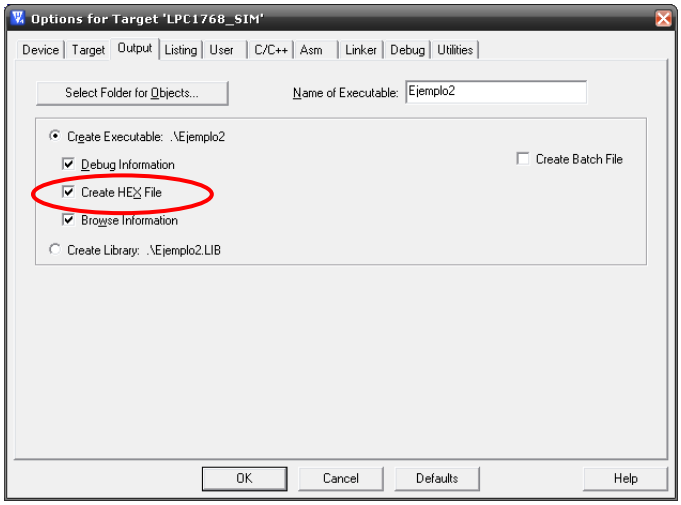


Figura 12. Configuración para que genere fichero ejecutable HEX.

5. Pulse desde Keil el icono *LOAD*  . Previamente es necesario configurar en la pestaña **Utilities** de las opciones del proyecto, el *path* de acceso al ejecutable (.exe) FlashMagic donde se instaló (figura 13).

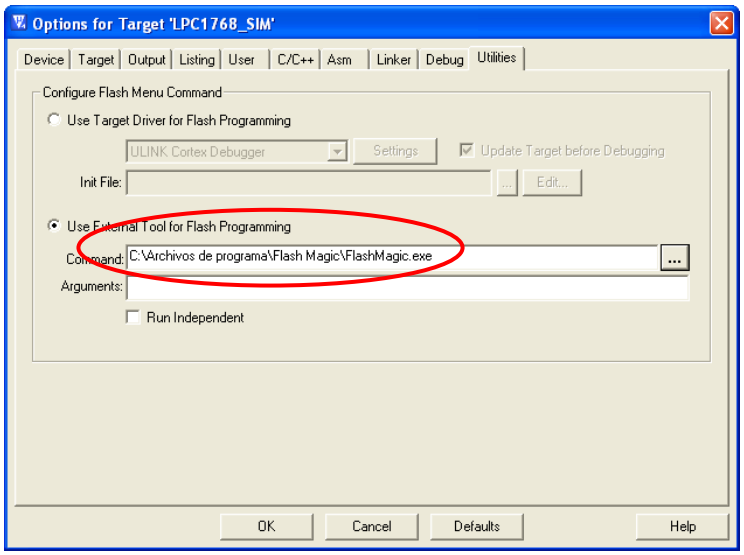


Figura 13. Configuración de la ruta del programa FlashMagic.

6. Se abrirá el programa FlashMagic. Configure el puerto COMx que se haya detectado, y seleccione el Baud Rate a 115200 baudios. Compruebe el *path* del archivo .HEX de carga. Si no detecta el puerto COMx automáticamente puede comprobar cuál es en el Panel de Control de Windows (Por ejemplo en Windows 7 está en Inicio -> Panel de control -> Dispositivos e impresoras).

7. Pinche en **Start** para que se cargue el programa en la memoria flash del LPC1768. Observe la barra inferior derecha de la ventana que muestra el proceso de la transferencia (figura 14).

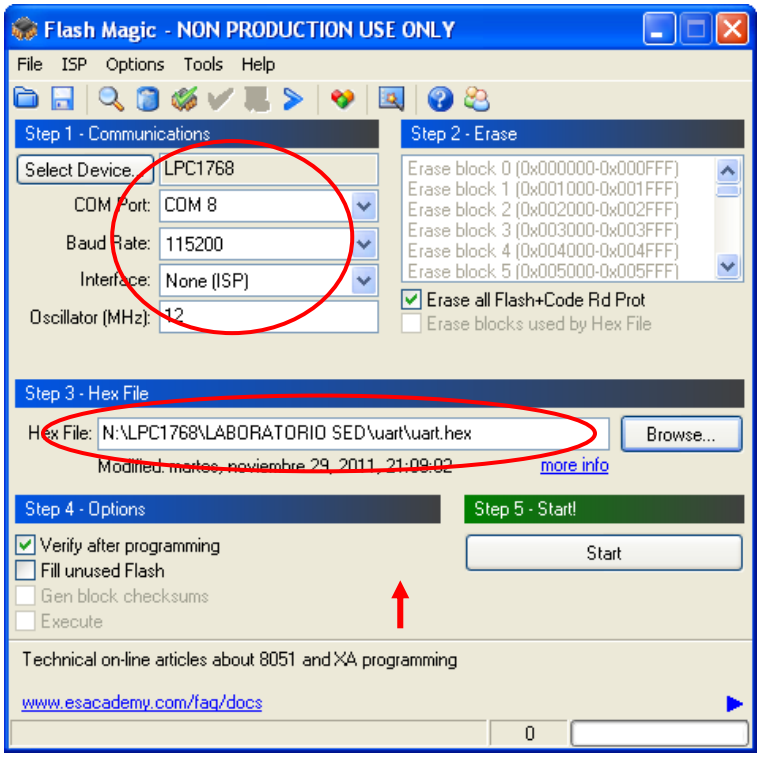


Figura 14. Carga del fichero HEX en la memoria flash de la tarjeta.

8. Cierre el programa FlashMagic para tener de nuevo el control sobre el entorno Keil, y poder lanzar la ejecución del programa que acabamos de cargar en la Blueboard desde el PC.

9. Para ejecutar el programa, pulse el botón de reset de la tarjeta (SW1).

10. Si todo funciona correctamente el LED conectado a P1.18 se debe encender y apagar de forma intermitente.

Configuracion del adaptador J-Link para carga y depuración

En este anexo se abordará la configuración del entorno de depuración Keil para posibilitar la utilización del adaptador J-Link de Segger para la depuración de programas. Esta configuración tiene dos partes, por un lado, la carga del controlador del dispositivo J-Link conectado al ordenador a través de un puerto USB, y por otro lado, la configuración del programa Keil para el empleo del adaptador J-link en la depuración de programas *on-chip*. Además se proponen una serie de ejercicios a realizar en el laboratorio a partir de un programa ejemplo, que a su vez sirve para que el usuario pueda comprobar que el adaptador ha sido instalado y configurado correctamente. Dicho programa se encuentra en la Blackboard y se denomina DepuracionJLINK.

Instalación del controlador de USB del adaptador J-Link

Antes de utilizar el adaptador J-Link de Segger se debe instalar el controlador del dispositivo para USB. Para ello basta con ejecutar el fichero *DPInst* que se encuentra en la ruta: Keil/ARM/Segger/USBDriver/x64, tal y como se muestra en la figura 15.

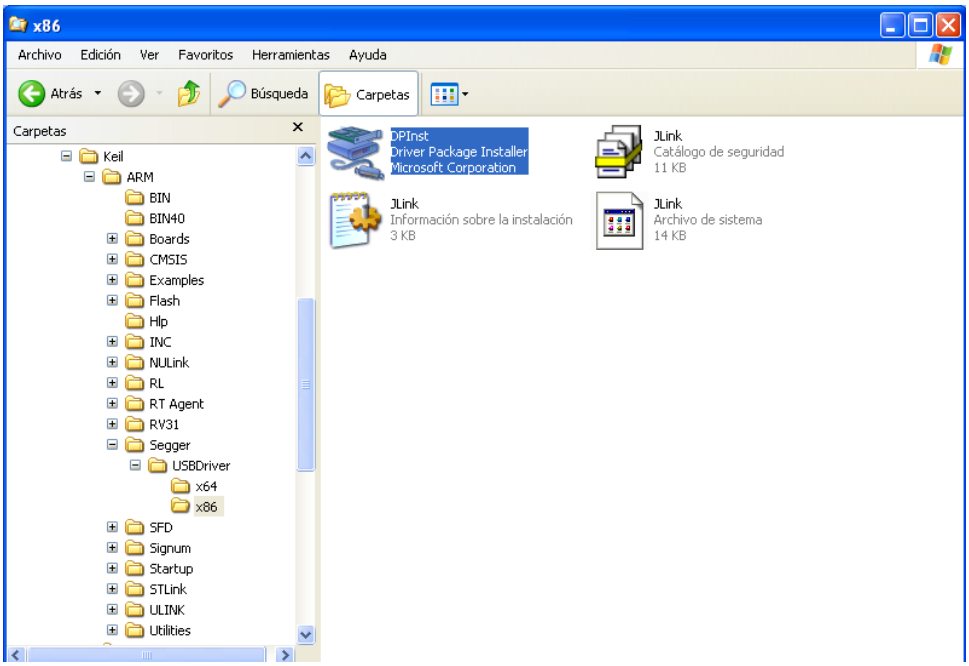


Figura 15. Ruta de acceso al instalador del controlador USB para el J-Link de Segger.

Una vez instalado el controlador (Driver) USB para el adaptador J-Link, para depurar un programa son necesarios los pasos siguientes:

1. Conexión de la Tarjeta BlueBoard_LPC1768_H.
2. Configuración del entorno de desarrollo µVision4 de Keil.
3. Edición del programa.
4. Depuración del programa.

A continuación se describe detalladamente cada uno de estos pasos.

Conexión de la Tarjeta BlueBoard_LPC1768_H

En esta práctica se va a trabajar con la tarjeta BlueBoard_LPC1768_H alimentada por USB con el adaptador J-LINK para realizar la carga y depuración del programa.

1. Conecte el adaptador J-LINK a la Tarjeta BlueBoard_LPC1768_H.
2. Alimente la tarjeta BlueBoard por USB conectándola al ordenador con el cable USB-MiniUSB
3. Conecte el adaptador J-LINK al ordenador.

En esta práctica no se van a utilizar más componentes externos.

Configuración del entorno de desarrollo Keil µVision 4

Antes de iniciar el proceso de depuración *on-chip*, debemos tener el proyecto a depurar cargado en el entorno de desarrollo. El entorno de desarrollo µVision 4 permite la depuración *on-chip* a través de adaptadores JTAG, para ello debe acceder a las opciones del proyecto, *Options for Target 1*… y desde la solapa *Debug* activar la casilla "Use <Hardware de depuración>", tal y como muestra figura 16. El tipo de hardware que se debe seleccionar en nuestro caso es J-Link.

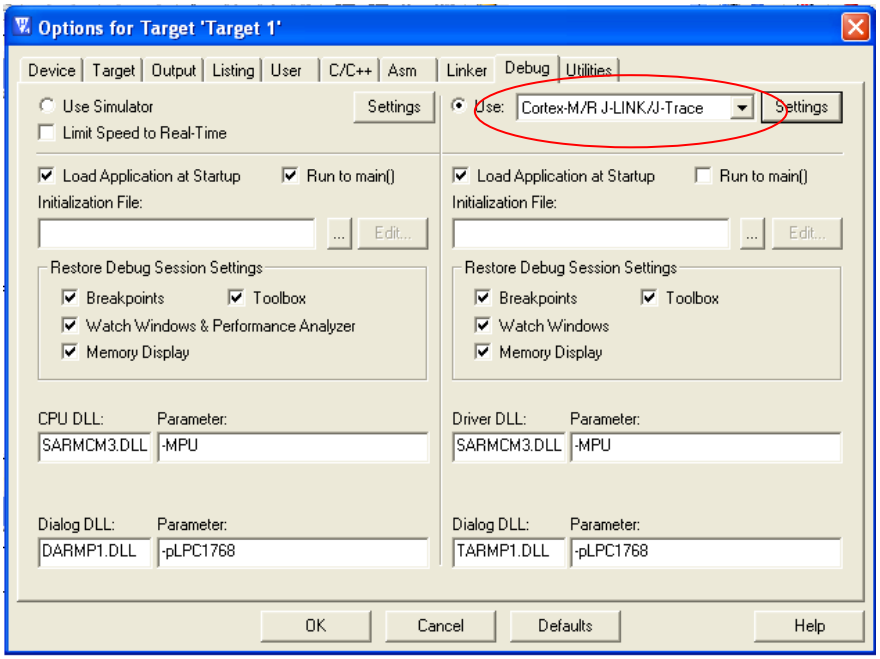


Figura 16. Menú en el que se puede seleccionar el tipo de depuración que se va a realizar: *Simulator* u *On-Chip*

Aparte de las demás opciones que proporciona el cuadro de diálogo mostrado en la figura 16, es importante que no se modifiquen las librerías (*CPU DLL*, *Dialog DLL*) y *Drive DLL*).

La depuración *on-chip* permite el volcado de programas en la memoria Flash, para ello, se debe seleccionar el hardware para depuración que se va a emplear desde la solapa *Utilities* de las mismas opciones del proyecto, según se muestra en la figura 17.

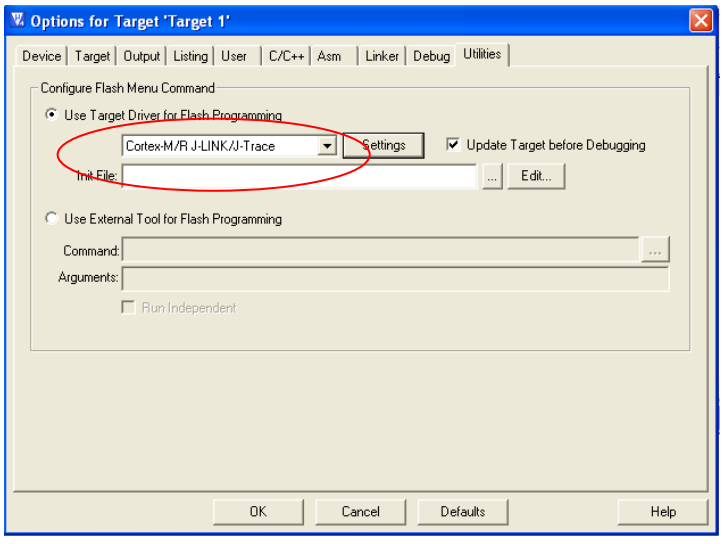


Figura 17. Menú que permite seleccionar el hardware con el que vamos a realizar el volcado en Flash de los programas a depurar.

Una vez seleccionado el Hardware de depuración, pulsamos *Settings* y en el siguiente cuadro de diálogo, solapa *Flash Download*, seleccionamos las características de la Flash de la tarjeta *Blueboard-H*, según se muestra en la figura 18. A esta misma ventana también se tiene acceso pulsando el botón *Settings* del cuadro de diálogo mostrado en la figura 16.

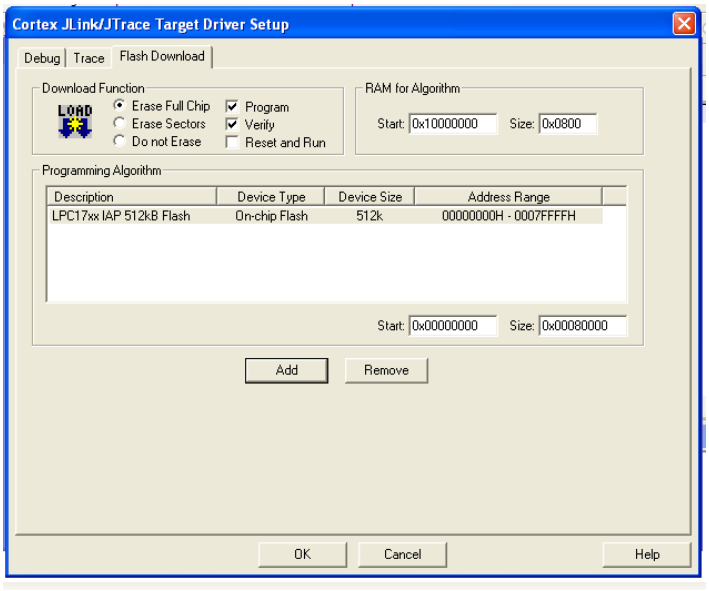


Figura 18. Especificaciones sobre la memoria flash de la tarjeta de desarrollo Blueboard LPC1768-H.

Desde esa misma ventana, en la pestaña *Debug* se debe seleccionar como *interface* el *USB*, mientras que el propio Keil actualiza los datos referidos al J-Link, como son el número de serie (*SN*) y la versión de Hardware (*HW*). Si deseamos hacer el seguimiento en tiempo real de una variable debemos seleccionar como *Port* el *SW* (*serial wire*) y pulsar sobre *Auto Clk* para que tome la frecuencia de reloj del dispositivo. Las opciones que deben estar seleccionadas se muestran en la figura 19, resaltando aquelas que es necesario modificar.

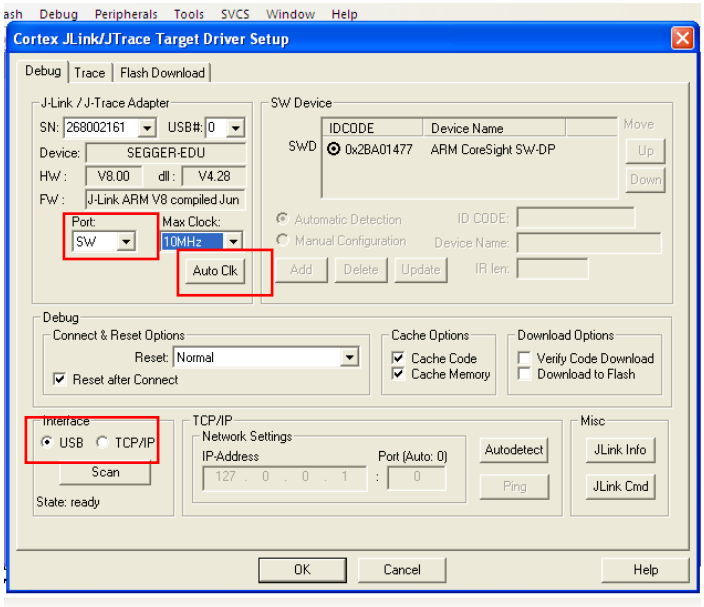


Figura 19. Especificaciones de depuración del J-link.

La pestaña *Trace* permite seleccionar las opciones de configuración de la toma de traza por el J-Link. Debemos tener activadas las opciones que se muestran en la figura 20. Entre ellas, las más importantes son:

- *PC sampling*, define la frecuencia de muestreo del ordenador sobre la tarjeta. El intervalo de muestreo debe ser lo suficientemente elevado como para permitir un refresco adecuado de los datos. La casilla "on data R/W Simple" debe estar activada para permitir que el PC realice un muestreado de las lecturas y escrituras de memoria.

- *Timestamps* permite reescalar los datos que se muestran en la pantalla del analizador lógico.

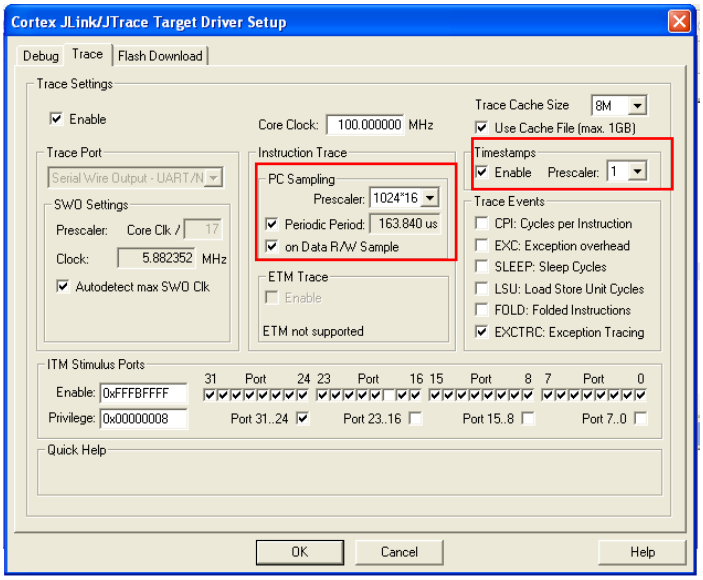


Figura 20. Opciones requeridas para el muestreo adecuado de variables en depuración.

Una vez configuradas las opciones anteriores, la depuración *on-chip* se realiza siguiendo los mismos pasos que en la opción de Simulación, iniciándola con la opción *Start/Stop Debug Session*, ver figura 21.

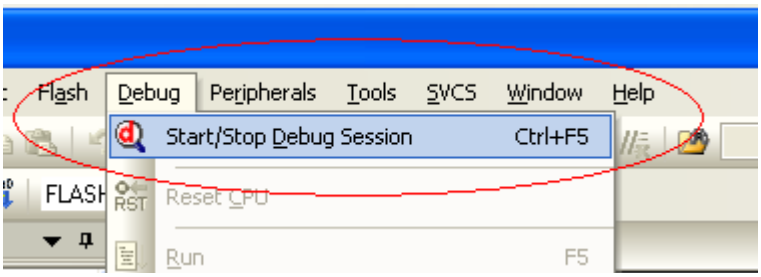


Figura 21. Opción de inicio de la depuración empleando J-Link.

El entorno µVision también permite la compilación de un mismo proyecto para ser depurado en simulación o cualquier otro medio, como el J-Link. Para ello se pueden generar distintos proyectos de manera paralela y que se modifican de manera simultánea. Para usar esta opción debemos crear un nuevo proyecto utilizando el gestor de componentes del proyecto. Pulsamos con el botón derecho sobre el nombre del proyecto (figura 22a) y seleccionamos la opción *Manage Componentes…* A continuación pulsamos sobre el icono dentro del grupo *Project Targets* e introducimos un nuevo nombre para el proyecto (figura 22b). Por ejemplo en uno de los proyectos podemos ponerle *LPC1768 J-Link* y lo configuraremos con las opciones de depuracion *on-chip*, y el otro *LPC1768 Simulacion*, que tendrá las opciones necesarias para poder simularlo.

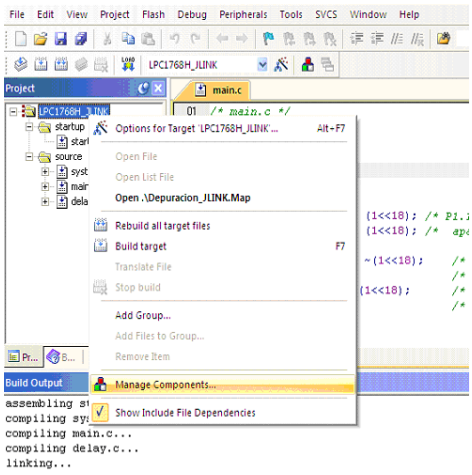


Figura 22a.

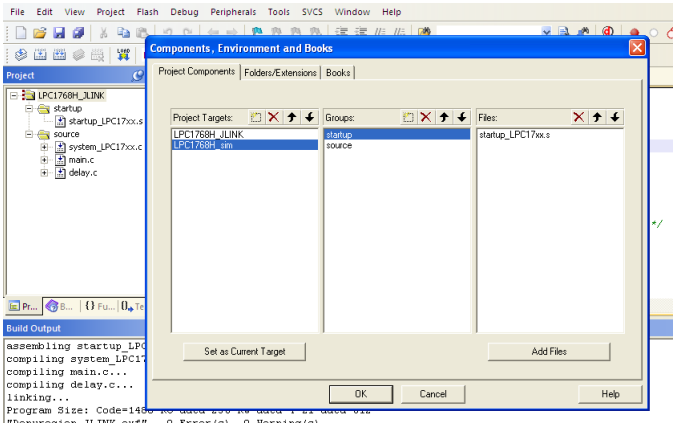


Figura 22b.

Una vez generadas las dos copias del proyecto, es posible cambiar de una copia a la otra usando el menú desplegable situado en la barra de comandos, como se recoge en la figura 23.

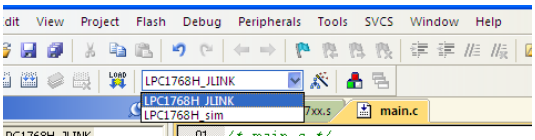


Figura 23. Menú desplegable que permite seleccionar el tipo de proyecto que se está depurando.

Depuración del programa

El programa que se va a utilizar como punto de partida es el programa llamado *DepuracionJLINK*, cuyo funcionamiento consiste en hacer parpadear el LED D7 de la tarjeta Blueboard. La cadencia de encendido y apagado del LED se incrementa si se mantiene pulsado el pulsador SW1. El código de este programa se encuentra al final de esta sección.

Una vez generado el proyecto basado en el programa *DepuracionJLINK* y de haber configurado el adaptador J-LINK, se debe proceder a realizar el volcado del programa en la memoria Flash del microcontrolador pulsando *Start Debug Session* ( ) y debe aparecer la pantalla del depurador. Los pasos para depurar el programa empleando el J-Link son los siguientes:

1. El primer paso para comprobar el funcionamiento del programa es realizar una ejecución paso a paso del mismo. Para ello, coloque *breakpoints* en las líneas del programa en las que se produce un cambio en el estado del pin de salida que lleva conectado el LED D7.

2. Una vez puestos los *breakpoints* se puede realizar una ejecución continua del programa, comprobando que la ejecución se para en la línea que produce el encendido del LED D7, tal y como se muestra en la figura 24, y que cambia el estado del LED D7 de la tarjeta.

3. A continuación se debe hacer un seguimiento del estado de los pines a través de la ventana del puerto GPIO 1 del *Fast Interface*. En esta ventana se debe ver reflejado el estado del pin 1.18 durante la ejecución del programa.

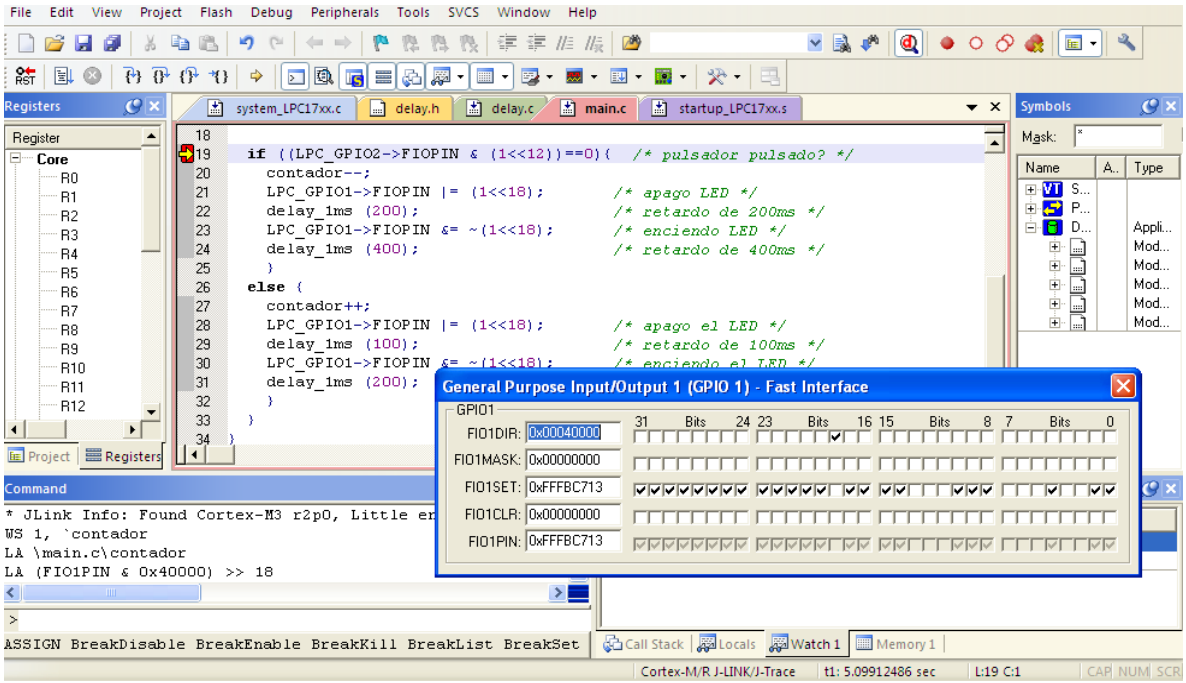


Figura 24. Depuración empleando *breakpoints* del programa ejemplo.

4. También se puede hacer un seguimiento del estado del pin P1.18 empleando el analizador lógico, tal y como se estudió en las secciones anteriores. Para ello, basta con añadir el registro **FIO1PIN.18** al analizador lógico. Depurando en ejecución continua y manteniendo los *breakpoints* podemos determinar el tiempo que está encendido y apagado el LED pulsando y sin pulsar el pulsador SW2.

Ejercicios propuestos para afianzar conocimientos

Una vez realizada una primera depuración se proponen los siguientes ejercicios, de los cuales se deben adjuntar evidencias y se debe mostrar al profesor.

1. *Visualización del estado del pulsador SW2 en tiempo real.*

   Para ello se debe mostrar el estado del pin P2.12 a través del analizador lógico y de la ventana de F-GPIO2..

2. *Modifique el programa para que la pulsación de SW2 produzca que el pin 18 del puerto 1 cambie de estado cada 500 ms. Por el contrario, cuando el pulsador SW2 no esté pulsado debe encenderse y apagarse el LED D8 de la tarjeta Blueaboard con la misma frecuencia.* Nota: tenga en cuenta que el LED D8 se encuentra conectado al pin 29 del puerto 1 y se activa a nivel alto.

Programa ejemplo

El siguiente programa produce el parpadeo del LED D7 de la tarjeta de desarrollo Blueboard LPC1768-H con una frecuencia que depende de la pulsación del SW2.

```
// Proyecto: Parpadea un LED.  Archivo: main.c
#include "lpc17xx.h"
#include "delay.h"

//    Funciones de retardo
void delay(uint32_t n)
{  int32_t i;
  for(i=0;i<n;i++);
}
int32_t contador = 0;
int main (void)
```

```
{
  LPC_GPIO1->FIODIR |= (1<<18);       // P1.18 definido como salida
  LPC_GPIO2->FIODIR &= ~(1<<12);    // P2.12 definido como entrada
  LPC_GPIO1->FIOCLR |= (1<<18);       // P1.18 apagado

  while(1) {
      contador --;
// Comprueba si el pin P2.12 está nivel bajo (pulsado)
    if (!(LPC_GPIO2->FIOPIN & (1<<12))){       // Si est· pulsado
      LPC_GPIO1->FIOPIN |= (1<<18);  // Enciendo LED
      delay (1000000);
      LPC_GPIO1->FIOPIN &= ~(1<<18); // Apago LED
      delay (1000000);
    }
    else    {     // Si no está pulsado
      contador ++;
      LPC_GPIO1->FIOSET = (1<<18);   // Enciendo LED
      delay (5000000);
      LPC_GPIO1->FIOCLR = (1<<18);   // Apago LED
      delay (5000000);
    }
  }
}
```

Practica Propuesta

Se propone el desarrollo de la práctica que debe realizar las siguientes acciones. El microcontrolador calculará y sacará el valor de los números primos contenidos entre un valor inicial y el valor 65535. El programa calculará el primer número primo, a continuación lo mostrará, y seguidamente realizará el cálculo del siguiente número, lo mostrará,… sin tener en ningún momento que almacenar los números calculados.

Una vez que se haya finalizado el cálculo y muestra de todos los números primos, el programa comenzará otra vez dicho cálculo desde el valor inicial hasta el 65535, y asi indefinidamente. Se debe permitir que en cada ciclo, el valor inicial pueda ser distinto al del ciclo anterior (por ejemplo, la primera vez que se realiza el cálculo, el valor inicial puede ser 1000 (con lo que se deben calcular los números primos de 1000 al 65535) y una vez finalizado estos cálculos y muestras, en el siguiente ciclo, el valor inicial puede ser el 5000 (números primos del 5000 al 65535)), y una vez calculado estos se continua con un tercer ciclo,….

El microcontrolador obtendrá el valor inicial (de 16 bits) uniendo el pin 4 y desde el 6 hasta el 15 del conector J14 de la placa Blueboard y los pines 1 hasta el 5 del conector J5. (El bit menos significativo (bit 0) del valor inicial es el pin 4 del conector J14, a continuación va el pin 6 del conector J14, a continuación va el pin 7 del conector J14,…, el bit 10 del valor inicial es el pin 15 del conector J14, el bit 11 del valor inicial es el pin 1 del conector J5, … y el bit 15 (MSB) del valor inicial se obtiene del pin 5 del conector J5.

Los números primos que se vayan obteniendo se irán sacando, manteniendo una duración de dos segundos por cada uno de ellos, a través de tres lugares:

- En los pines del conector J13. En este conector se mostrara el resultado en dos formatos en función del valor del pulsador SW2. En el caso de que este pulsado, se mostrara el número el formato binario en los primeros 16 bits del conector, y en caso de que se encuentre libre se mostrara el número en formato BCD en los 19 bits del conector.

- En los pines del conector J6. Se mostrara el número primo en ese conector para alimentar un display BCD de 7 segmentos en ánodo común (E10561-G). En este caso, se mostrara cada digito BCD durante un tiempo de 400mseg en el orden de decenas de millar, unidades de millar, centenas, decenas y unidades. En este caso, el pin 2 estará asociado con el segmento A, …

- En los diodos D7 y D8. El estado del diodo D7 ira conmutando cada 125 mseg. Para cada número primo, en cada uno de los 16 periodos de 125 mseg en los que el número esté activo, el diodo D8 estará iluminando si el bit correspondiente del número primo es un 1 o estará apagado si su bit correspondiente es 0.

Agradecimientos

La memoria de la presente práctica ha sido realizada utilizando material cedido por profesores del Departamento de Electrónica de la Universidad de Alcalá.