Acceso a Registros

Para el caso de las familias de microcontroladores que tengan los registros de sus periféricos mapeados en memoria, existen dos formas de arbitrar los recursos para acceder a ellos.

1) *Declarando una variable que tenga el tamaño del registro en su dirección de memoria asociada.*
2) *Con Punteros*
   a. *Declarando una variable tipo puntero que sea capaz de acceder de forma monolítica al registro.*
   b. *Mediante una macro que tenga la dirección del registro casteada debidamente para poder acceder de forma monolítica al registro.*

Tanto para 1) como para 2) deberemos investigar en el compilador que usemos para la tarea cual es la manera de su implementación.
Par nuestro caso, en el que utilizaremos el compilador gcc desarrollado para la familia Cortex de 32 bits de la empresa nxp incorporado en el IDE CodeRed y tomando como ejemplo un registro ubicado en la dirección 2009C000H seria:

Para la opción 1)

```
unsigned int        registro        __attribute__((at(0x2000C000L)));
```

Pero lamentablemente para el gcc desarrollado para estos micros esta opción ***no está implementada***

Para la opción 2.a)

```
unsigned int *   registro = 0x2000C000L;
```

Para la opción 2.b)

```
#define         registro            ( ( unsigned int* ) 0x2009C000UL )
```

En el desarrollo de las diferentes guías de ejercicios, utilizaremos la opción 2.b. La razón es: que la opción 2.a consume memoria RAM para la variable y el código assembler generado para su utilización es mayor.
Probablemente, esto último provoque risa al lector, ya que seguramente a esta altura conocerá los recursos del cortex M3 LPC1769 (que es el que se utilizará en esta guía) en donde abunda memoria RAM y memoria ROM (flash).

¿Entonces porque nuestra preocupación?

- *No todos los microcontroladores que existen en el mercado son de 32 bits y cuentan con estas capacidades.*
- *Existe una línea Cortex con recursos mínimos tanto de periféricos como de memoria, en formatos que van desde DIP de 8 patas,*
  - *Aparecen para reemplazar definitivamente a microcontroladores de 8 y 16 bits*
  - *Lograron inserción en ambientes de educación media*

Es por eso que en Informática II desarrollaremos las herramientas y rudimentos indispensables para que el alumno se desenvuelva con la misma soltura tanto con un microcontrolador de 32 bits que como con uno de 8 o 16 bits.
Sin perjuicio de lo dicho anteriormente, también desarrollaremos otras formas de acceder a los registros que si bien será de uso particular para estas familias nos permitirá:

- *Conjugar otras herramientas del lenguaje (estructuras, uniones y campos de bits)*
- *Acercar al alumno a herramientas utilizadas en las librerías CMSIS desarrolladas especialmente para estos microcontroladores.*

VERSION PRELIMINAR

REGISTROS DE CONFIGURACION

Registro PINSEL

Los registros PINSEL son los que permiten seleccionar la función que desarrollará cada Pin, tal como se muestra a continuación. Dentro del PINSEL cada pin se encuentra representado por un par de bits, que de acuerdo a la combinación que se adopte dará por resultado el dispositivo que estará presente sobre el Pin.
Cada uno de los pines tiene al menos asociada 2 funciones que se seleccionan con estos registros, cave destacar que en muchos casos las funciones se repiten en varios pines, esto es para facilitar al diseñador el desarrollo del circuito impreso.

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	

Registro PINMODE

Los registros PINMODE controlan el modo de las entradas de todos los puertos del microcontrolador. Incluye el uso de las funciones del resistor de pull-up / pull-down y un nodo especial de funcionamiento de open drain. La resistencia pull-up / pull-down se puede seleccionar para cada pin del puerto, independientemente de la función que tenga asociada, con la excepción de los pines para la interfaz I2C para la interface I2C0 y los pines asociados con el USB.

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.	

Registro PINMODE_OD

Los registros PINMODE_OD controlan los modos open drain sobre los puertos. El modo de open drain hace que el pin se vaya normalmente a estado bajo si está configurado como salida y el valor del dato es 0. Si el valor del dato es 1, la unidad de salida se apaga, lo que equivale a cambiarla dirección el pin. Esta combinación simula una salida de ***open drain***

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.	

La distribución de las funciones de de los puertos en PINSEL y PINMODE tienen la misma ubicación dentro de ellos. Esto pude verse en la tabla1 en donde tomamos como ejemplo a PINSEL0 y PINMODE0. Esto nos permitirá realizar funciones compactas en el desarrollo los programas.

tabla 1

PINMODE0	Symbol	Value	Description	Reset value
1:0	P0.00		Port 0 pin 0 on-chip pull-up/down resistor control.	00
3:2	P0.01		Port 0 pin 1 control, see P0.00MODE.	00
5:4	P0.02		Port 0 pin 2 control, see P0.00MODE.	00
7:6	P0.03		Port 0 pin 3 control, see P0.00MODE.	00
9:8	P0.04		Port 0 pin 4 control, see P0.00MODE.	00
11:10	P0.05		Port 0 pin 5 control, see P0.00MODE.	00
13:12	P0.06		Port 0 pin 6 control, see P0.00MODE.	00
15:14	P0.07		Port 0 pin 7 control, see P0.00MODE.	00
17:16	P0.08		Port 0 pin 8 control, see P0.00MODE.	00
19:18	P0.09		Port 0 pin 9control, see P0.00MODE.	00
21:20	P0.10		Port 0 pin 10 control, see P0.00MODE.	00
23:22	P0.11		Port 0 pin 11 control, see P0.00MODE.	00
29:24	-		Reserved.	NA
31:30	P0.15		Port 0 pin 15 control, see P0.00MODE.	00

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

En resumen

Ejemplo

PINSEL	Function	
00	GPIO pin.	P0[26]
01	A/D converter 0, input 3.	AD0[3]
10	D/A converter output.	AOUT
11	Receiver input for UART3.	RXD3

figura 1

output

GPIO

input

PINMODE_OD	Function
0	Pin is in the normal (not open drain) mode.
1	Pin is in the open drain mode.

PINMODE	Function
00	Pin has an on-chip pull-up resistor enabled.
01	Repeater mode
10	Pin has neither pull-up nor pull-down resistor enabled.
11	Pin has an on-chip pull-down resistor enabled.

La tabla 2 nos muestra su ubicación dentro del mapa de memoria.

VERSION PRELIMINAR

tabla 2

Name	Description	Access	Reset Value	Address
PINSEL0	Pin function select register 0.	R/W	0	0x4002 C000
PINSEL1	Pin function select register 1.	R/W	0	0x4002 C004
PINSEL2	Pin function select register 2.	R/W	0	0x4002 C008
PINSEL3	Pin function select register 3.	R/W	0	0x4002 C00C
PINSEL4	Pin function select register 4	R/W	0	0x4002 C010
PINSEL7	Pin function select register 7	R/W	0	0x4002 C01C
PINSEL8	Pin function select register 8	R/W	0	0x4002 C020
PINSEL9	Pin function select register 9	R/W	0	0x4002 C024
PINSEL10	Pin function select register 10	R/W	0	0x4002 C028
PINMODE0	Pin mode select register 0	R/W	0	0x4002 C040
PINMODE1	Pin mode select register 1	R/W	0	0x4002 C044
PINMODE2	Pin mode select register 2	R/W	0	0x4002 C048
PINMODE3	Pin mode select register 3.	R/W	0	0x4002 C04C
PINMODE4	Pin mode select register 4	R/W	0	0x4002 C050
PINMODE5	Pin mode select register 5	R/W	0	0x4002 C054
PINMODE6	Pin mode select register 6	R/W	0	0x4002 C058
PINMODE7	Pin mode select register 7	R/W	0	0x4002 C05C
PINMODE9	Pin mode select register 9	R/W	0	0x4002 C064
PINMODE_OD0	Open drain mode control register 0	R/W	0	0x4002 C068
PINMODE_OD1	Open drain mode control register 1	R/W	0	0x4002 C06C
PINMODE_OD2	Open drain mode control register 2	R/W	0	0x4002 C070
PINMODE_OD3	Open drain mode control register 3	R/W	0	0x4002 C074
PINMODE_OD4	Open drain mode control register 4	R/W	0	0x4002 C078
I2CPADCFG	I^2C Pin Configuration register	R/W	0	0x4002 C07C

Puertos de entrada salida de propósito general

Tal como en esta y en otras oportunidades nos resultará complejo encarar la explicación de estos registros puesto que los alumnos que serán usuarios de estas guías provienen con conocimientos previos dados por formaciones disciplinares diferentes.

1) *Los que vienen de escuelas no técnicas y descubran el tema por primera vez y no quisiéramos que se queden con la idea de que todos los microcontroladores de plaza tienen estos recursos.*
2) *Los que vienen de escuela técnica y que seguramente conocen sobre microcontroladores de 8 bits, inevitablemente vincularan lo aprendido con sus conocimientos previos, con lo cual estarán en conflicto permanente puesto que el mundo de 8 bits y el de 32 bits son bastante diferentes.*

Hagamos sinergia entre ambos grupos y razonemos en conjunto, imaginemos que nosotros seremos los diseñadores del microcontrolador LPC1769 y por ende quienes decidamos cuales serán las prestaciones de las que dotaremos a sus periféricos y en particular a las GPIO.
Uno o varios de los registros que resultan infaltables serán los que reflejen el estado de sus pines asociados.

Para nuestro caso le daremos el nombre de FIOPIN (Fast Input Output PIN), diferenciándose con un numero para seleccionar el puerto: FIO0PIN, FIO1PIN, FIO2PIN, FIO3PIN, FIO4PIN.
Para aquellos que tengan conocimientos sobre la familia de microcontroladores 8x51 creerán que esto es suficiente y seguramente lo asociaran inmediatamente con los puertos P0, P1, P2 y P3.

Ejemplo como salida Cortex:
FIO1PIN =0xFFFFFFFF; // todos las salidas se van a estado alto.

Ejemplo como salida 8x51:
P1 = 0xFF; // todos las salidas se van a estado alto.

¿ Y cómo entrada ?
Los que conozcan 8x51 seguro ya tienen la respuesta:
unsigned char in;
in = P1; // en in quedo guardado el estado de todas las entradas

Pero para nuestro microcontrolador no lo haremos tan directo. Tomemos el ejemplo de familias tales como freescale o los AVR de ATMEL en donde se cuenta con registros que indican si el puerto será de entrada o salida.
Démosle el nombre de FIODIR (Fast Input Output DIRection), diferenciándolo con un numero para la selección del puerto: FIO0DIR, FIO1DIR, FIO2DIR, FIO3DIR y FIO4DIR

Ejemplo para freescale en donde los puertos se identifican con letras:
```
unsigned char in;
…..
DDRB= 0x0f;        // 0: entrada 1: salida
PTB = 0x05;        // el pin 0 y el 2 se fueron a estado alto
…..
in = PTB ;
```
Para el caso de nuestro microcontrolador será:

```
unsigned int in;
…..
FIO1DIR = 0x0000FFFF;      // 0: entrada 1: salida
FIO1PIN = 0x00005555;      // los pines 0,2,4,6,8,10,12 y 14 se fueron a estado alto
…..
in = FIO1PIN; // lectura del estado de las entradas
```

Si bien, con esto nos podríamos dar por satisfechos, porque no agregar más prestaciones. Seguramente usted sabe, que a medida que avanza la tecnología en el desarrollo de microcontroladores, se hace cada vez más compleja su programación en assembler y más habitual en C, de hecho los mismos se diseñan pensando que serán programados en este lenguaje. El C es un lenguaje estructurado en donde al programar intentamos ir siempre por el camino verdadero, simbolizado habitualmente con un 1 y por contraposición al falso con 0.
Pensemos….Si con el registro FIOPIN….

1. *Quiero activar una salida con un estado alto, lo hacemos con un 1 → Verdadero*
2. *Quiero activar una salida con un estado bajo, lo hacemos con un 0 → Falso !!!!!*

Siguiendo con lo manifestado anteriormente, lo que sería deseable es que el punto dos sea Verdadero, puesto que poner un 0 es lo que se pretende hacer.
Pues bien, hagámoslo, creemos un registro que ponga estados altos en los pines y otro que ponga bajos.
Démosle el nombre de FIOSET (Fast Input Output SET), al que permita colocar estados altos en los pines y FIOCLR (Fast Input Output CLearR) al que permita colocar estados bajos en los pines, diferenciándolos con un numero para indicar a que puerto pertenecen: FIO0SET, FIO1SET, FIO2SET, FIO3SET, FIO4SET y FIO0CLR, FIO1CLR, FIO2CLR, FIO3CLR, FIO4CLR.
Y por ultimo hagamos que funcione de la siguiente forma:
FIOnSET :

VERSION PRELIMINAR

Un 1 en alguno de sus bits produce un estado alto en su pin asociada → Verdadero !!!
Un 0 en alguno de sus bits no produce ningún efecto en su pin asociado
FIOnCLR :
Un 1 en alguno de sus bits produce un estado bajo en su pin asociada→ Verdadero !!!
Un 0 en alguno de sus bits no produce ningún efecto en su pin asociado.
Y de este modo hemos logrado actuar siempre por verdadero.
FIOnPIN y FIOnSET, FIOnCLR mas no son excluyentes entre sí, dejemos libertad al programador que de que utilice los que le resulten más cómodos para su tarea.

Hagamos un mismo ejemplo con ambas opciones

Poner un estado alto en P0[22] y un estado bajo en P0[21]

tabla 3

Opción 1	Opción 2
FIO0PIN = FIO0PIN | 0x00400000; FIO0PIN = FIO0PIN & 0xFFDFFFFF;	FIO0SET = FIO0SET | 0x00400000; FIO0CLR = FIO0CLR | 0x00200000;

Sin embargo en ambos casos hubiera sido más cómodo escribir:

tabla 4

Opción 1	Opción 3
FIO0PIN = 0x00400000; FIO0PIN = 0x00000000;	FIO0SET = 0x00400000; FIO0CLR = 0x00200000;

Pero sabemos que si hacemos lo indicado en la tabla 4 pondremos los valores deseados en los pines requeridos pero descuidamos los valores del resto de los bits correspondientes a P0
Entonces como nos dieron libertad para el diseño sigamos incorporando registros. Creemos un registro que oficie de mascara, es decir si el registro que asociemos al P0 tiene sus bits en 0 y se permita que todo lo que hagamos con FIOPIN,PIOGLR o FIOSET se refleje en los pines respectivos, y si los tiene en 1 que no se vean afectados.
*Démosle el nombre de FIOMASK (**F**ast **I**nput **O**utput **MASK**), diferenciándolos entre si, tal como venimos haciendo con un numero para asociarlo a su puerto: FIO0MASK, FIO1MASK, FIO2MASK, FIO3MASK, FIO4MASK.*

Entonces nuestro ejemplo quedaría:

tabla 5

Opción 1	Opción 2

FIO0MASK = 0xFFBFFFFF;
FIO0PIN = 0x??4?????; // ?: puede ser cualquier valor
FIO0MASK = 0xFFDFFFFF;
FIO0PIN = 0x??0?????; // ?: puede ser cualquier valor
	FIO0MASK = 0xFFBFFFFF;
	FIO0SET = 0x??4?????; // ?: puede ser cualquier valor
	FIO0MASK = 0xFFDFFFFF;
	FIO0CLR = 0x00200000; // ?: puede ser cualquier valor

Con los cinco registros desarrollados para cada puerto podemos darnos por satisfechos y continuar más adelante agregándoles algunas prestaciones.
La tabla 6 muestra el mapa de memoria y una descripción simplificada de cada uno de ellos tomada de las hojas de datos del LPC1769

VERSION PRELIMINAR

Tabla 6

Generic Name	Description	Access	Reset value	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

VERSIO

TPC N°1: Declaración de Variables y Registros

El objetivo de pese pretende ejercitar en este TPC es de cómo seleccionar el dispositivo que quedara asociado a un pin y su uso en particular como entrada/salidas de propósito general.

Ejemplo 1.1

Crear un conjunto de macros que definan a los registros PINSEL.

```
#define          PINSEL          ( ( uint32_t  * ) 0x4002C000UL )

#define          PINSEL0         PINSEL[0]   //PINSEL0------>P0[15:0]
#define          PINSEL1         PINSEL[1]   //PINSEL1------>P0[31:16]
#define          PINSEL2         PINSEL[2]   //PINSEL2------>P1[15:0]
#define          PINSEL3         PINSEL[3]   //PINSEL3------>P1[31:16]
#define          PINSEL4         PINSEL[4]   //PINSEL4------>P2[15:0]
#define          PINSEL5         PINSEL[5]   //PINSEL5------>P2[31:16]    NOT USED
#define          PINSEL6         PINSEL[6]   //PINSEL6------>P3[15:0]     NOT USED
#define          PINSEL7         PINSEL[7]   //PINSEL7------>P3[31:16]
#define          PINSEL8         PINSEL[8]   //PINSEL8------>P4[15:0]     NOT USED
#define          PINSEL9         PINSEL[9]   //PINSEL9------>P4[31:16]
```

Ejemplo 2.1

Configurar el PINSEL que corresponda para que P0[26] quede configurado como GPIO.

```
// De las hojas de datos
```

PINSEL1	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P0.16	GPIO Port 0.16	RXD1	SSEL0	SSEL	00
3:2	P0.17	GPIO Port 0.17	CTS1	MISO0	MISO	00
5:4	P0.18	GPIO Port 0.18	DCD1	MOSI0	MOSI	00
7:6	P0.19	GPIO Port 0.19	DSR1	Reserved	SDA1	00
9:8	P0.20	GPIO Port 0.20	DTR1	Reserved	SCL1	00
11:10	P0.21	GPIO Port 0.21	RI1	Reserved	RD1	00
13:12	P0.22	GPIO Port 0.22	RTS1	Reserved	TD1	00
15:14	P0.23	GPIO Port 0.23	AD0.0	I2SRX_CLK	CAP3.0	00
17:16	P0.24	GPIO Port 0.24	AD0.1	I2SRX_WS	CAP3.1	00
19:18	P0.25	GPIO Port 0.25	AD0.2	I2SRX_SDA	TXD3	00
21:20	P0.26	GPIO Port 0.26	AD0.3	AOUT	RXD3	00
23:22	P0.27	GPIO Port 0.27	SDA0	USB_SDA	Reserved	00
25:24	P0.28	GPIO Port 0.28	SCL0	USB_SCL	Reserved	00

```
PINSEL1 &= 0xFFCFFFFF
```

Configurar el PINSEL que corresponda para que P0[26] quede configurado como AD0.3.

PINSEL1	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P0.16	GPIO Port 0.16	RXD1	SSEL0	SSEL	00
3:2	P0.17	GPIO Port 0.17	CTS1	MISO0	MISO	00
5:4	P0.18	GPIO Port 0.18	DCD1	MOSI0	MOSI	00
7:6	P0.19	GPIO Port 0.19	DSR1	Reserved	SDA1	00
9:8	P0.20	GPIO Port 0.20	DTR1	Reserved	SCL1	00
11:10	P0.21	GPIO Port 0.21	RI1	Reserved	RD1	00
13:12	P0.22	GPIO Port 0.22	RTS1	Reserved	TD1	00
15:14	P0.23	GPIO Port 0.23	AD0.0	I2SRX_CLK	CAP3.0	00
17:16	P0.24	GPIO Port 0.24	AD0.1	I2SRX_WS	CAP3.1	00
19:18	P0.25	GPIO Port 0.25	AD0.2	I2SRX_SDA	TXD3	00
21:20	P0.26	GPIO Port 0.26	AD0.3	AOUT	RXD3	00
23:22	P0.27	GPIO Port 0.27	SDA0	USB_SDA	Reserved	00
25:24	P0.28	GPIO Port 0.28	SCL0	USB_SCL	Reserved	00

```
PINSEL1 = ( PINSEL1 & 0xFFCFFFFF )| 0x00001000 ;
```

```
   b??????????**??**????????????????????? ----- Contenido de PINSEL1
&  b1111111111**00**1111111111111111111111 ----- Limpiamos PINSEL1[20:21]
   b??????????**00**????????????????????? ----- Contenido de PINSEL1
|  b0000000000**01**0000000000000000000000 ----- Colocamos la función 1
   b??????????**01**????????????????????? ----- Contenido de PINSEL1 actualizado
```

PRELIMINAR

Ejemplo 3.1

Realizar una función que sea capaz de configurar los modos de P0 que responda al siguiente prototipo.

```
/**
      \fn void SetPINSEL01 ( uint8_t , uint8_t )
      \brief Activa el modo de trabajo del P0
      \details Configura PINSEL0 y PINSEL1
      \param [in] pin: Número de pin del puerto
      \param [in] modo: Función del pin del puerto
      \return Retorna void
*/
void SetPINSEL01 ( uint8_t pin, uint8_t modo){
            if (pin >= 16){
                  pin -=16;
                  PINSEL1 &= ~(0x3 << (pin * 2));
                  PINSEL1 |= (modo << (pin * 2));
            }
            else {
                  PINSEL0 &= ~(0x3 << (pin * 2));
                  PINSEL0 |= (modo << (pin * 2));
            }
}
```

Ejercicio 4.1

Realizar una función que sea capaz de configurar las cuatro posibilidades sobre un pin del LPC1769 que responda al siguiente prototipo y que ponga de manifiesto a todos los registros PINSEL.

```
/**
      \fn void SetPINSEL ( uint8_t , uint8_t , uint8_t )
      \brief Activa el modo de trabajo de los puertos
      \details Configura el PINSEL correspondiente al puerto seleccionado
      \param [in] pin: Número de puerto
      \param [in] pin: Número de pin del puerto
      \param [in] modo: Función del pin del puerto
      \return Retorna void
*/
void SetPINSEL (uint8_t puerto, uint8_t pin, uint8_t modo){
      switch (puerto){
            case P0:
                  if (pin >= 16){
                        pin -=16;
                        PINSEL1 &= ~(0x11 << (pin * 2));
                        PINSEL1 |= (modo << (pin * 2));
                  }
                  else {
                        PINSEL0 &= ~(0x11 << (pin * 2));
                        PINSEL0 |= (modo << (pin * 2));
                  }
                  break;
            case P1:
                  |     Completar !!!!
                  |
                  |

      }
}
```

VERSIÓN PRELIMINAR

Ejercicio 5.1

Realizar una función que sea capaz configurar las cuatro posibilidades sobre un pin LPC1769 que responda al siguiente prototipo con la menor cantidad de código que pueda.

```
/**
      \fn void SetPINSEL ( uint8_t , uint8_t , uint8_t )
      \brief Activa el modo de trabajo de los puertos
      \details Configura el PINSEL correspondiente al puerto seleccionado
      \param [in] pin: Número de puerto
      \param [in] pin: Número de pin del puerto
      \param [in] modo: Función del pin del puerto
      \return Retorna void
*/
```

Ejemplo 6.1

A partir de la tabla 6 crear las macros necesarias para definir los registros de los GPIO de LPC1769

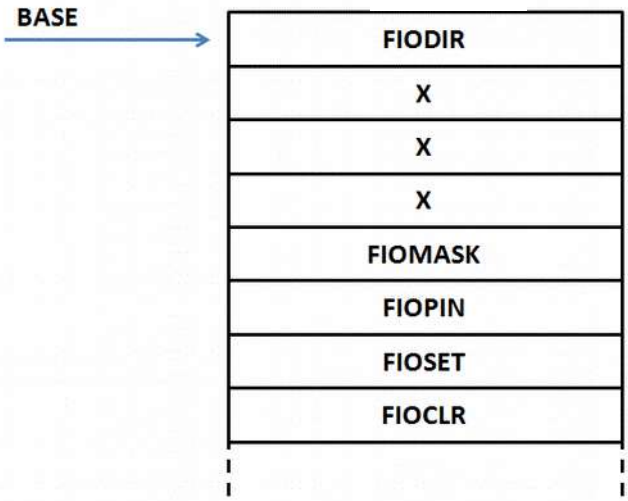
```
#define           GPIOs             ( ( uint32_t  * ) 0x2009C000UL )

#define           FIO0DIR           GPIOs[0]    //!< 0x2009C000
#define           FIO1DIR           GPIOs[8]    //!< 0x2009C020
#define           FIO2DIR           GPIOs[16]   //!< 0x2009C040
#define           FIO3DIR           GPIOs[24]   //!< 0x2009C060
#define           FIO4DIR           GPIOs[32]   //!< 0x2009C080

#define           FIO0MASK          GPIOs[4]    //!< 0x2009C010
#define           FIO1MASK          GPIOs[12]   //!< 0x2009C030
#define           FIO2MASK          GPIOs[20]   //!< 0x2009C050
#define           FIO3MASK          GPIOs[28]   //!< 0x2009C070
#define           FIO4MASK          GPIOs[36]   //!< 0x2009C090

#define           FIO0PIN           GPIOs[5]    //!< 0x2009C014
#define           FIO1PIN           GPIOs[13]   //!< 0x2009C034
#define           FIO2PIN           GPIOs[21]   //!< 0x2009C054
#define           FIO3PIN           GPIOs[29]   //!< 0x2009C074
#define           FIO4PIN           GPIOs[37]   //!< 0x2009C094

#define           FIO0SET           GPIOs[6]    //!< 0x2009C018
#define           FIO1SET           GPIOs[14]   //!< 0x2009C038
#define           FIO2SET           GPIOs[22]   //!< 0x2009C058
#define           FIO3SET           GPIOs[30]   //!< 0x2009C078
#define           FIO4SET           GPIOs[38]   //!< 0x2009C098

#define           FIO0CLR           GPIOs[7]    //!< 0x2009C01C
#define           FIO1CLR           GPIOs[15]   //!< 0x2009C03C
#define           FIO2CLR           GPIOs[23]   //!< 0x2009C05C
#define           FIO3CLR           GPIOs[31]   //!< 0x2009C07C
#define           FIO4CLR           GPIOs[39]   //!< 0x2009C09C
```

VERSION PRELIMINAR

Ejemplo 7.1

A partir de la tabla 6 (y con la ayuda de la figura 2) intente crear las macros necesarias para definir los registros de los GPIO de LPC1769 tal como lo realizo en el ejemplo 6 pero mediante la utilización de estructuras.

figura 2



PUERTO	BASE
P0	0x2009C000
P1	0x2009C020
P2	0x2009C040
P3	0x2009C060
P4	0x2009C080

Ejercicio 8.1

Crear un conjunto de macros que definan a los registros PINMODE.
Crear un conjunto de macros que definan a los registros PINMODE_OD.

Ejercicio 9.1

A partir de lo expresado en la figura 1 realice todas las acciones necesarias para configurar el P022 como salida y escribirle un uno a su pin asociado. Realice el ejercicio al menos de dos formas diferentes.

Ejercicio 10.1

A partir de lo expresado en la figura 1 realice todas las acciones necesarias para configurar el el P021 como entrada tipo pull up.

Ejercicio 11.1

A partir de lo expresado en la figura 1 realice todas las acciones necesarias para configura el P022, P024 y P027 como salida y escribirle un uno a sus pines asociados.
NOTA: Realizar cada una de las configuraciones del conjunto de los tres pines en un solo paso.

VERSION PRELIMINAR

Ejercicio 12.1

Declare e inicialice un vector de char en ROM con los códigos de conversión a 7 segmentos en orden de 0 a 9

Ejercicio 13.1

Declare e inicialice un vector de char en ROM con los códigos ascii que van del 0 al 9 y A a la F en orden consecutivo.

Ejercicio 14.1

Realizar una función que configure a los puertos del LPC1769 como entrada o salida y que responda al siguiente prototipo.

```
/*****************************************************************************
      \fn  void SetDIR (uint8_t puerto, uint8_t pin, uint8_t direccion)
      \brief Establece si un determinado PIN de un determinado PUERTO (previamente
                configurado como GPIO) es entrada o salida.
      \author : Usted
      \param [in] puerto:     port con el que se va a trabajar
      \param [in] pin:        pin a configurar sentido
      \param [in] direccion:  0 = entrada - 1 = salida.
      \return void
*/
void SetDIR(uint8_t puerto,uint8_t pin,uint8_t direccion);
```

Ejercicio 15.1

Realizar una función que escriba un uno o un cero en los puertos del LPC1769 que fueron configurados como salida que responda al siguiente prototipo.

```
/*****************************************************************************
      \fn  void SetPIN (uint8_t puerto, uint8_t pin, uint8_t estado)
      \brief Establece un ESTADO en un determinado PIN de un determinado PUERTO.
      \author: Usted
      \param [in] puerto: port con el que se va a trabajar
      \param [in] pin:  pin a setear estado
      \param [in] estado:     valor a establecer en el PIN del PUERTO [0-1].
      \return void
*/
void SetPIN(uint8_t puerto,uint8_t pin,uint8_t estado);
```

VERSION PRELIMINAR

Ejercicio 16.1

Realizar una función que lea los puertos del LPC1769 que fueron configurados como entradas que responda al siguiente prototipo.

```
/*****************************************************************************
      \fn  uint8_t GetPIN (uint8_t puerto, uint8_t pin , uint8_t actividad)
      \brief Devuelve el ESTADO de un determinado PIN de un determinado PUERTO.
      \author: Usted
      \param [in] puerto: port con el que se va a trabajar
      \param [in] pin:  pin a consultar estado
      \param [in] actividad define si es activo bajo o activo alto
      \return:    estado del pin consultado (uint_8)
*/
uint8_t GetPIN(uint8_t puerto,uint8_t pin , uint8_t actividad);
```

Ejercicio 17.1

Realizar una función que seleccione la función que desarrollara el pin del puerto seleccionado del LPC1769 .

```
/*****************************************************************************
      \fn  void SetPINSEL ( uint8_t puerto , uint8_t pin ,uint8_t sel)
      \brief selección de la función del PIN de un determinado PUERTO.
      \author Usted
      \param [in] puerto: port con el que se va a trabajar
      \param [in] pin:  pin configurar
      \param [in] sel:  selección deseada (0,1,2 o 3)
      \return:    void
*/
void SetPINSEL( uint8_t puerto , uint8_t pin ,uint8_t sel)
```

Ejercicio 18.1

Realizar una función que seleccione el comportamiento del puerto (pull-up, pull-down, etc) cuando este fue configurado como entrada digital

```
/*****************************************************************************
      \fn  void SetPINMODE( uint8_t puerto, uint8_t pin ,uint8_t modo)
      \brief selecciona el comportamiento de un pin cuando fue configurado como
           entrada.
      \author Usted
      \param [in] puerto: puerto con el que se va a trabajar
      \param [in] pin:  pin configurar
      \param [in] modo: modo de trabajo deseado (0,1,2 o 3)
      \return:    void
*/
void SetPINMODE( uint8_t puerto , uint8_t pin ,uint8_t modo)
```

VERSION PRELIMINAR

Ejercicio 19.1

Realizar una función que seleccione el comportamiento del puerto (pull-up, pull-down, etc) cuando este fue configurado como entrada digital

```
/*****************************************************************************
      \fn  void SetTOGGLE( uint8_t puerto, uint8_t pin )
      \brief Invierte el estado del pin del puerto.
      \author Usted
      \param [in] puerto: puerto con el que se va a trabajar
      \param [in] pin:  pin a invertir
      \return:    void
*/
void SetTOGGLE( uint8_t puerto , uint8_t pin)
```

VERSION PRELIMINAR