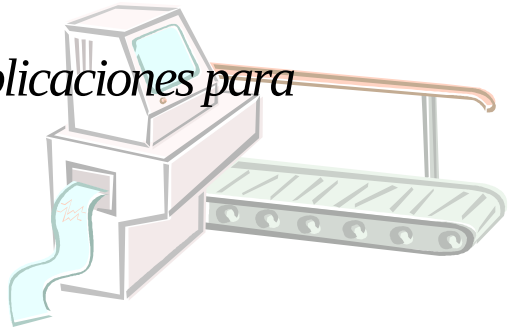
e-mail: eliseo.garcia@uah.es
Despacho: N-245
Departamento de Automática

Práctica 3

Puertos de Entrada/Salida

Laboratorio de Desarrollo de Aplicaciones para Sistemas Industriales



Puertos de Entrada/Salida

Introducción

Durante la ejecución del programa, es muy probable que un microcontrolador, además de hacer distintos cálculos, necesite comunicarse con otros dispositivos a los que se encuentre conectado. El método más simple es la utilización de puertos de Entrada/Salida, donde podrá leer/escribir datos binarios.

El LPC1768 cuenta con 5 Puertos de Entrada/Salida, cada uno de ellos con varias patillas. La forma de utilizarlo es mediante la escritura en una serie de registros, que son los que define la arquitectura ARM en la que está basado.

El entorno de desarrollo Keil uVision ofrece una serie de identificadores para poder referenciar los registros en el código que se desarrolle sobre él. Estos identificadores difieren del nombre que ARM utiliza para los registros, pero existe una equivalencia, que es la que a continuación se detalla:

Registros de Arquitectura ARM	Identificador del entorno Keil
Registro PINSEL**X**	LPC_PINCON->PINSEL**X**
Registro PINMODE**X**	LPC_PINCON->PINMODE**X**
Registro PINMODE_OD**X**	LPC_PINCON->PINMODE_OD**X**
Registro FIO**X**DIR	LPC_GPIO**X**->FIODIR
Registro FIO**X**DIR**Y**	LPC_GPIO**X**->FIODIR**Y**
Registro FIO**X**PIN	LPC_GPIO**X**->FIOPIN
Registro FIO**X**PIN**Y**	LPC_GPIO**X**->FIOPIN**Y**
Registro FIO**X**SET	LPC_GPIO**X**->FIOSET
Registro FIO**X**SET**Y**	LPC_GPIO**X**->FIOSET**Y**
Registro FIO**X**CLR	LPC_GPIO**X**->FIOCLR
Registro FIO**X**CLR**Y**	LPC_GPIO**X**->FIOCLR**Y**
Registro FIO**X**MASK	LPC_GPIO**X**->FIOMASK
Registro FIO**X**MAST**Y**	LPC_GPIO**X**->FIOMASK**Y**

donde X hace referencia al numero de registro, e Y hace referencia al tamaño de acceso (Y puede ser 0,1,2,3,H,L)

Simulacion de los puertos

En este apartado se continua con la presentación del manejo del entorno de desarrollo Keil µVision comenzado en la práctica anterior. En este caso se trata de la utilización de la herramienta para visualizar/modificar el valor de los puertos y el uso de *breakpoints*.

Para eso, Keil uVision nos da la posibilidad de ver los valores de los distintos puertos. Por ejemplo, si se quisiera visualizar las señales del puerto P0, se puede hacer mostrando su ventana a través del menú *Peripherals*→*GPIO Fast Interface*→*Port 0* (figura 1).

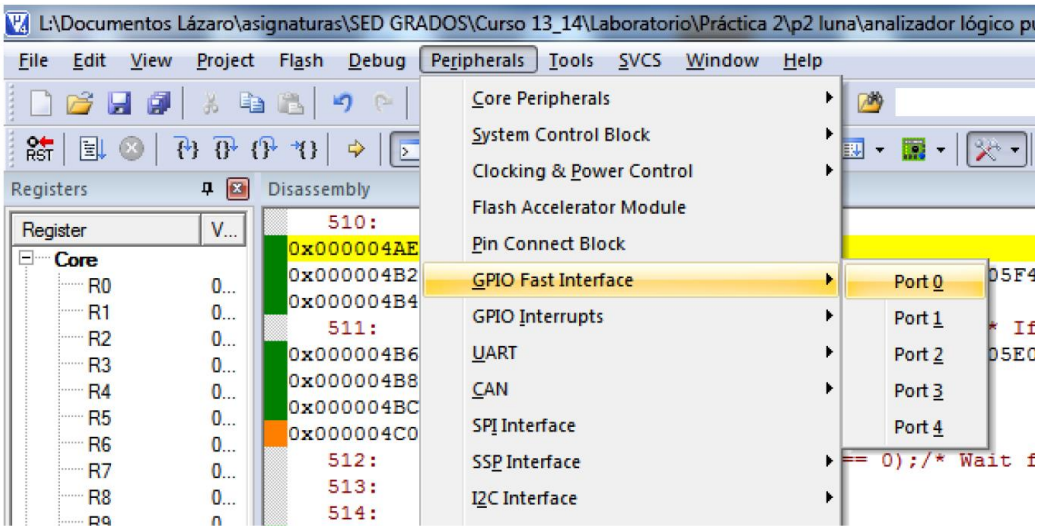


Figura 1. Visualización de la ventana del puerto 0.

La ventana que aparece es la de la figura 2. Se puede ejecutar el programa paso a paso y así se observa cómo pueden cambiar los estados de los distintos registros y patillas asociadas al puerto P0 durante la ejecución de un programa. Nótese además que en aquellas patillas que se configuren como entradas, se puede modificar el valor de las patillas actuando sobre la ultima fila del menú (*Pins*), simulando así los valores que un elemento externo situaría en la patilla del microcontrolador.

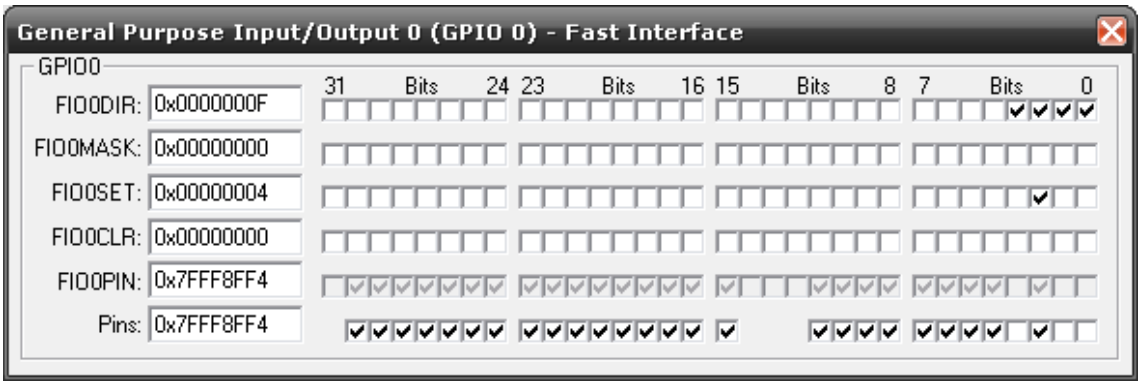


Figura 2. Ventana de visualización del puerto P0.

Practica Propuesta

Se propone la realización de un programa cuyo funcionamiento consista en hacer parpadear los LED LD2 y LD3 de la tarjeta MiniDK2. La cadencia de encendido y apagado de ambos LEDs se incrementará si se mantienen pulsado los pulsadores KEY1 y KEY2, respectivamente.

Una vez generado el proyecto, se podrá configurar el adaptador J-LINK y realizar el volcado del programa en la memoria Flash del microcontrolador (pulsando *Start Debug Session*), observando en la placa el funcionamiento del programa desarrollado.

Realización paso a paso de la Práctica

Una vez conocidos los requisitos que se plantean en la práctica, vamos a construir paso a paso funciones que nos permitan llegar hasta la aplicación que resuelva el problema

Paso 1. Configuracion de funcionalidad GPIO

Realícese un proyecto que realice la configuración como GPIO de una determinada patilla en un determinado puerto . A continuación, créese en el entorno de desarrollo un proyecto donde se definan las siguientes funciones:

- La función *"uint8_t IsAValidPin(uint8_t puerto, uint8_t patilla)"*, que cuando se invoca retorna si la patilla del puerto indicado es una patilla válida, es decir, se encuentra implementada en el microcontrolador. La función retornará 1 si la patilla esta implementada y 0 en caso contrario. Considerese que en el LPC1768, las patillas implementadas son las que se especifican en la siguiente tabla.

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.

- La función *"uint8_t SetGPIO(uint8_t puerto, uint8_t patilla)"*, que cuando se invoca configura como GPIO la patilla del puerto indicado por los parámetros de entrada. La función retornará 1 si la configuración se realizó correctamente y 0 en caso contrario.

```
#include "lpc17XX.h"

uint8 t IsAValidPin(uint8 t puerto, uint8 t patilla){
/* Configura una patilla como GPIO.
 Parametros de entrada:
   puerto:   Puerto GPIO (De 0 a 4)
   patilla:   Patilla del puerto (de 0 a 31)
 Parametros de salida:
   exito:     1→ patilla existente  0→ patilla no implementada
*/
uint8 t exito;  // 1 si la configuracion fue correcta, 0 en caso contrario
…









return exito;
}
```

```
uint8 t SetGPIO(uint8 t puerto, uint8 t patilla){
/* Configura una patilla como GPIO.
 Parametros de entrada:
   puerto:   Puerto GPIO a configurar (De 0 a 4)
   patilla:   Patilla del puerto a configurar (de 0 a 31)
 Parametros de salida:
   exito:     1→ configuracion realizada correctamente  0→ en caso contrario
*/
uint8 t exito;  // 1 si la configuracion fue correcta, 0 en caso contrario
…




return exito;
}

int main(){
uint8_t resul, puerto, patilla, valida;
puerto = 3;
patilla = 25;
…

// Comprueba si la patilla esta implementada en el microcontronlador
valida = IsAValidPin (puerto, patilla);
…
// Configura como GPIO la patilla 3.25
if (valida){
   resul=SetGPIO(puerto, patilla);
   }
while(1);
}
```

Compruebese, mediante depuración y situando breakpoints, que variando los valores de los parámetros de entrada (puerto y patilla), las patillas quedan configuradas correctamente. Para eso, se puede utilizar, una vez comenzada la depuración, el panel Peripherals→Pin Connect Block.

Paso 2. Configuracion de la dirección de la comunicación GPIO

Realícese un proyecto que realice la configuración de la dirección de comunicación de una patilla configurada como GPIO entre entrada o salida. A continuación, créese en el entorno de desarrollo un proyecto donde se defina la función *"uint8_t SetGPIODir(uint8_t puerto, uint8_t patilla,uint8_t direccion)"*, que cuando se invoca configura la patilla GPIO del puerto indicado como entrada (*uint8_t dirección←0*) o salida(*uint8_t dirección←1*). La función retornará 1 si la configuración se realizó correctamente y 0 en caso contrario.

```
#include "lpc17XX.h"
uint8 t SetGPIODir(uint8 t puerto, uint8 t patilla, uint8 t direccion){
/* Configura la direccion de una patilla GPIO.
 Parametros de entrada:
  puerto:   Puerto GPIO a configurar (De 0 a 4)
  patilla:   Patilla del puerto a configurar (de 0 a 31)
  direccion: Direccion a configurar:  0→ entrada    1→salida
 Parametros de salida:
   exito:     1→ configuracion realizada correctamente  0→ en caso contrario
*/
uint8 t exito;  // 1 si la configuracion fue correcta, 0 en caso contrario
…
```

```
return exito;
}

int main(){
uint8 t resul, puerto, patilla, direccion, valida;
puerto = 3;
patilla = 25;
direccion = 1; //salida
…
// Comprueba si la patilla esta implementada en el microcontronlador
valida = IsAValidPin (puerto, patilla);
…
if (valida){
   // Configura como GPIO la patilla 3.25
   resul=SetGPIO(puerto, patilla);
…
   // Configura como salida la patilla GPIO 3.25
   resul=SetGPIODir(puerto, patilla,direccion);
   }
while(1);
}
```

Compruebese, mediante depuración y situando breakpoints, que variando los valores de los parámetros de entrada (puerto, patilla y direccion), las patillas quedan configuradas correctamente. Para eso, se puede utilizar, una vez comenzada la depuración, el panel Peripherals→GPIO Fast Interface.

Paso 3. Establecimiento de un nivel de salida en una patilla GPIO de salida.

Realícese un proyecto que establezca un determinado valor de tensión (0V o 3,3V) en una patilla GPIO configurada como salida. A continuación, créese en el entorno de desarrollo un proyecto donde se defina la función *"uint8_t GPIOSetValue(uint8_t puerto, uint8_t patilla,uint8_t value)"*, que cuando se invoca establece en la patilla GPIO del puerto configurado como salida un valor de 0 Voltios (*uint8_t value←0*) o 3.3 V (*uint8_t value←1*). La función retornará 1 si el establecimiento de tensió se realizó correctamente y 0 en caso contrario.

```
#include "lpc17XX.h"
uint8_t GPIOSetValue(uint8_t puerto, uint8_t patilla, uint8_t value){
/* Configura la direccion de una patilla GPIO.
 Parametros de entrada:
  puerto:   Puerto GPIO a configurar (De 0 a 4)
  patilla:   Patilla del puerto a configurar (de 0 a 31)
  value:    Tension a enviar:  0→ 0 Voltios    1→ 3.3 Voltios
 Parametros de salida:
   exito:     1→ configuracion realizada correctamente  0→ en caso contrario
*/
uint8 t exito;  // 1 si la configuracion fue correcta, 0 en caso contrario
…



return exito;
}

int main(){
uint8 t resul, puerto, patilla, direccion, value;
puerto = 3;
patilla = 25;
```

```
direccion = 1; //salida
value = 1; //3.3V

// Comprueba si la patilla esta implementada en el microcontronlador
valida = IsAValidPin (puerto, patilla);
…
if (valida){
…
   // Configura como GPIO la patilla 3.25
   resul=SetGPIO(puerto, patilla);
…
   // Configura como salida la patilla GPIO 3.25
   resul=SetGPIODir(puerto, patilla,direccion);
…
   // Establece 3.3 V la salida la patilla GPIO 3.25
   resul=GPIOSetValue(puerto, patilla,value);
   }
while(1);
}
```

Compruebese, mediante depuración y situando breakpoints, que variando los valores de los parámetros de entrada (puerto, patilla y value), las patillas quedan configuradas y que el nivel de tensión se establece correctamente. Para eso, se puede utilizar, una vez comenzada la depuración, el panel Peripherals→GPIO Fast Interface. Una vez se ha comprobado el funcionamiento en el simulador, configúrese el proyecto para que modifique los valores de la patilla asociada al led LD2, cárguese el proyecto en la placa minidk2 mediante el cable JTAG y compruébese el cambio de estado del diodo.

Paso 4. Obtención del valor de un nivel de tension en una patilla GPIO de entrada.

Realícese un proyecto que obtenga el valor de tensión (0V o 3,3V) presente en una patilla GPIO configurada como entrada. A continuación, créese en el entorno de desarrollo un proyecto donde se defina la función *"uint8_t GPIOGetValue(uint8_t puerto, uint8_t patilla,uint8_t* value)"*, que cuando se invoca obtenga en valor de tensión, 0 Voltios (**value←0*) o 3.3 V (**value←1*), presente en la patilla GPIO del puerto configurado como entrada . La función retornará 1 si la obtención de la tensión se realizó correctamente y 0 en caso contrario.

```
#include "lpc17XX.h"
uint8_t GPIOGetValue(uint8_t puerto, uint8_t patilla, uint8_t* value){
/* Configura la direccion de una patilla GPIO.
 Parametros de entrada:
  puerto:   Puerto GPIO a configurar (De 0 a 4)
  patilla:   Patilla del puerto a configurar (de 0 a 31)
  value:    Puntero dende se guarda la tension recibida:  0→ 0 Voltios    1→ 3.3 Voltios
 Parametros de salida:
   exito:     1→ configuracion realizada correctamente  0→ en caso contrario
*/
uint8_t exito;  // 1 si la configuracion fue correcta, 0 en caso contrario
…



return exito;
}

int main(){
uint8_t resul, puerto, patilla, direccion;
uint8 t* value;
puerto = 3;
patilla = 25;
```

```
direccion = 0; //entrada

// Comprueba si la patilla esta implementada en el microcontronlador
valida = IsAValidPin (puerto, patilla);
…
if (valida){
…
   // Configura como GPIO la patilla 3.25
   resul=SetGPIO(puerto, patilla);
…
   // Configura como entrada la patilla GPIO 3.25
   resul=SetGPIODir(puerto, patilla,direccion);
…
   // Obtengo en la variable value el valor de tension presente en la patilla GPIO 3.25
   resul=GPIOGetValue(puerto, patilla,value);
   }
while(1);
}
```

Compruebese, mediante depuración y situando breakpoints, que variando los valores de los parámetros de entrada (puerto y patilla), las patillas quedan configuradas y que el nivel de tensión se obtiene correctamente. Para eso, se puede utilizar, una vez comenzada la depuración, el panel Peripherals→GPIO Fast Interface. Una vez se ha comprobado el funcionamiento en el simulador, configúrese el proyecto para que modifique los valores de la patilla asociada al pulsador KEY1, cárguese el proyecto en la placa minidk2 mediante el cable JTAG y compruébese el cambio del valor recibido en función del estado del pulsador.

Paso 5. Realizacion del parpadeo del diodo LD2.

Realícese un proyecto que realice el parpadeo del diodo LD2. Para eso, se deben modificar cada cierto tiempo los niveles de tensión en la patilla conectada a ese diodo. El periodo de tiempo entre apagado y encendido del diodo se realizará utilizando la función delay(), mostrada en el ejemplo. Observese que en función del valor de entrada de dicha función, dicho periodo es distinto.

Créese en el entorno de desarrollo, utilizando las funciones creadas en los paso anteriores y la función delay, un proyecto donde se defina el código que permite este comportamiento

```
#include "lpc17XX.h"

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

int main(){
…
while(1){  //Indefinidamente…
   // Enciendo el diodo
…
   // Dejo pasar el tiempo
   delay(100000);
   // Apago el diodo
…
   // Dejo pasar el tiempo
   delay(100000);
   }
…
}
```

Compruebese, mediante depuración y situando breakpoints, que varian los valores de la patilla asociada al diodo LD2. Para eso, se puede utilizar, una vez comenzada la depuración, el panel Peripherals→GPIO Fast Interface. Una vez se ha comprobado el funcionamiento en el simulador cárguese el proyecto en la placa minidk2 mediante el cable JTAG y compruébese el cambio de estado del diodo.

Paso 6. Realizacion de la modificación del periodo de parpadeo del diodo LD2.

Realícese un proyecto que permita un parpadeo mas rápido o mas lento del diodo LD2 en función de si se encuentra pulsado o no el pulsador KEY1. Para eso, se deben modificar cada cierto tiempo los niveles de tensión en la patilla conectada a ese diodo. El periodo de tiempo entre apagado y encendido del diodo, que se realizará utilizando la función delay(), será diferente en función de si se encuentra pulsado o no el pulsador KEY1.

Créese en el entorno de desarrollo donde se defina el código que permite este comportamiento

```
#include "lpc17XX.h"

// Funcion de retardo
void delay(uint32_t n)
{  uint32 t i;
  for(i=0;i<n;i++);
}

int main(){
…
while(1){  //Indefinidamente…
   if (…){   // Pulsador pulsado
       // Enciendo el diodo
       …
       // Dejo pasar el tiempo
       delay(100000);
       // Apago el diodo
       …
       // Dejo pasar el tiempo
       delay(100000);
   }else{
       // Enciendo el diodo
       …
       // Dejo pasar más tiempo
       delay(200000);
       // Apago el diodo
       …
       // Dejo pasar más tiempo
       delay(200000);
       }
   }
…
}
```

Compruebese, mediante depuración y situando breakpoints, que varian los valores de la patilla asociada al diodo LD2 y el tiempo entre cambios en función del valor de la patilla asociada al pulsador KEY1. Para eso, se puede utilizar, una vez comenzada la depuración, el panel Peripherals→GPIO Fast Interface. Una vez se ha comprobado el funcionamiento en el simulador cárguese el proyecto en la placa minidk2 mediante el cable JTAG y compruébese el cambio de estado del diodo.

Paso 6. Realizacion de la modificación del periodo de parpadeo del diodo LD2 y LD3.

Realícese un proyecto que permita un parpadeo mas rápido o mas lento los diodos LD2 y LD3 en función de si se encuentran pulsados o no los pulsadores KEY1 y KEY2, respectivamente.

Créese en el entorno de desarrollo donde se defina el código que permite este comportamiento

```
#include "lpc17XX.h"

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

int main(){
…
}
```

Compruebese, mediante depuración y situando breakpoints, que varian los valores de la patilla asociada a los diodos LD2 y LD3, y que el tiempo entre cambios en función del valor de la patilla asociada al pulsador KEY1 y KEY2, respectivamente. Para eso, se puede utilizar, una vez comenzada la depuración, el panel Peripherals→GPIO Fast Interface. Una vez se ha comprobado el funcionamiento en el simulador cárguese el proyecto en la placa minidk2 mediante el cable JTAG y compruébese el cambio de estado del diodo.