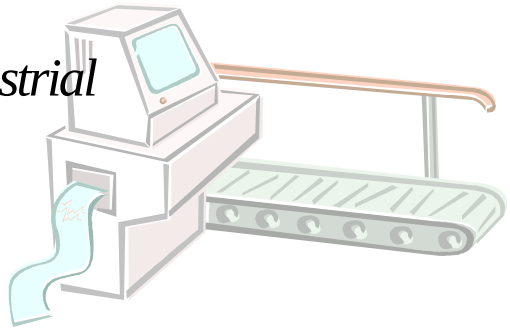
e-mail: eliseo.garcia@uah.es
Despacho: N-245
Departamento de Automática

Práctica 3

Puertos de Entrada/Salida

Laboratorio de Informática Industrial



Puertos de Entrada/Salida

Introducción

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).

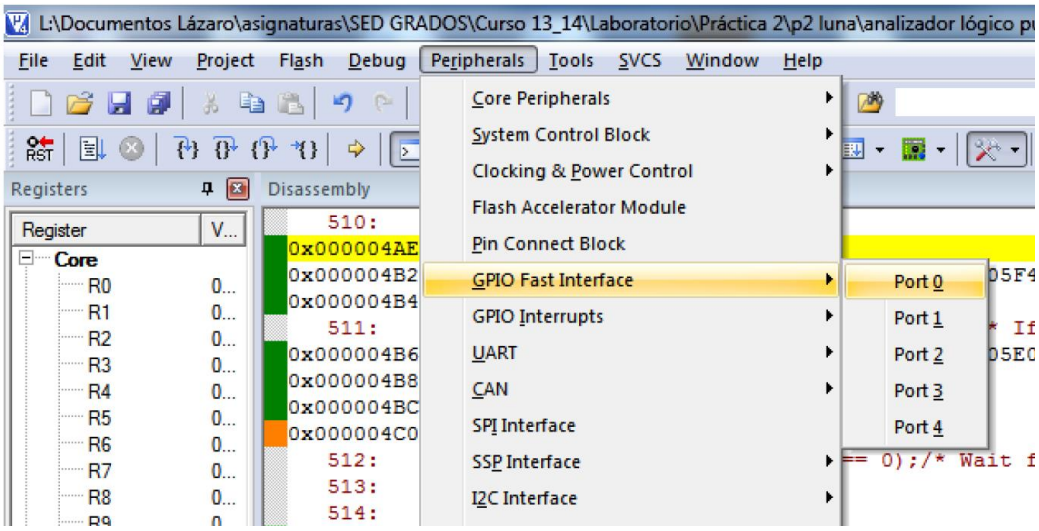


Figura 1. Visualización de la ventana del puerto 0.

La ventana que aparece es la de la figura 2. Ejecute el programa paso a paso y observe cómo se activan progresivamente los pines P0.[0..3]. Nótese además que en las patillas configuradas como entradas, se puede modificar el valor actuando sobre el menú.

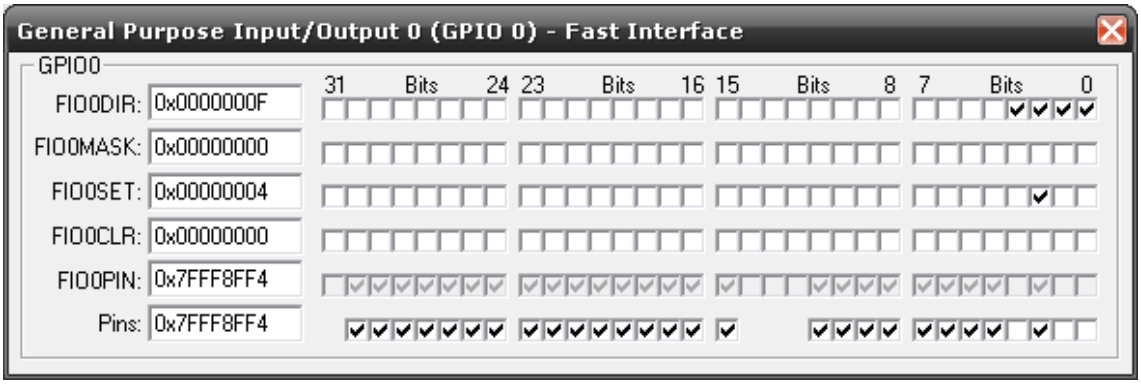


Figura 2. Ventana de visualización del puerto P0.

Simulación on-chip

El programa que se va a utilizar como punto de partida es el programa llamado *DepuracionJLINK*, cuyo funcionamiento consiste en hacer parpadear el LED LD2 de la tarjeta MiniDK2. La cadencia de encendido y apagado del LED se incrementa si se mantiene pulsado el pulsador KEY2. 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 según se detalla en el Anexo 1 (al final del presente documento), 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 LD2.

2. Una vez puestos los *breakpoints* se puede realizar una ejecución continua del programa, comprobando que la ejecución se detiene en la línea que produce el encendido del LED LD2 y que cambia el estado del LED LD2 de la tarjeta.

3. A continuación se debe hacer un seguimiento del estado de los pines a través de la ventana del puerto GPIO 3 del *Fast Interface*. En esta ventana se debe ver reflejado el estado del pin 3.25 durante la ejecución del programa.

4. También se puede hacer un seguimiento del estado del pin P3.25 empleando el analizador lógico (véase documento Anexo II. Analizador Lógico). Para ello, basta con añadir el registro **FIO3PIN.25** 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 SW4.

Programa ejemplo

El siguiente programa produce el parpadeo del LED D7 de la tarjeta de desarrollo MiniDK2 con una frecuencia que depende de la pulsación del SW2.

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

//    Funciones de retardo
void delay(uint32_t n)
{  int32_t i;
  for(i=0;i<n;i++);
}
int main (void)
{   LPC_GPIO3->FIODIR |= (1<<25);     // P3.25 definido como salida
  LPC_GPIO2->FIODIR &= ~(1<<12);    // P2.12 definido como entrada
  LPC_GPIO3->FIOSET |= (1<<25);       // P3.25 apagado

  while(1) {
    if (!(LPC_GPIO2->FIOPIN & (1<<12))){ // Comprueba si el pin P2.12 está nivel bajo (pulsado). Si est· pulsado
      LPC_GPIO3->FIOPIN |= (1<<25);  //… Enciendo LED
      delay (1000000);                          //… Dejo pasar un tiempo
      LPC_GPIO3->FIOPIN &= ~(1<<25); // ..Apago LED
      delay (1000000);                          //… Dejo pasar un tiempo
    }    else    {      // Si no está pulsado
      contador ++;
      LPC_GPIO3->FIOCLR = (1<<25);   // Enciendo LED
      delay (5000000);
      LPC_GPIO3->FIOSET = (1<<25);   // Apago LED
      delay (5000000);
    }
  }
}
```

Practica Propuesta

Se propone el desarrollo de un proyecto 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 almacenar en ningún momento 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) de los pines del Conector P1 de la placa Minidk2. En concreto, lo obtendrá según se describe en el siguiente esquema:

Pines del Conector P1	**Bits del valor inicial**
Patilla 6 (GPIO P0.26)	Bit 0
Patilla 7 (GPIO P0.25)	Bit 1
Patilla 8 (GPIO P0.24)	Bit 2
Patilla 9 (GPIO P0.23)	Bit 3
Patilla 20 (GPIO P1.31)	Bit 4
Patilla 21 (GPIO P1.30)	Bit 5
Patilla 24 (GPIO P0.28)	Bit 6
Patilla 25 (GPIO P0.27)	Bit 7
Patilla 29 (GPIO P0.29)	Bit 8
Patilla 30 (GPIO P0.30)	Bit 9
Patilla 32 (GPIO P1.18)	Bit 10
Patilla 33 (GPIO P1.19)	Bit 11
Patilla 34 (GPIO P1.20)	Bit 12
Patilla 35 (GPIO P1.21)	Bit 13
Patilla 36 (GPIO P1.22)	Bit 14
Patilla 37 (GPIO P1.23)	Bit 15

Tabla1. Descripcion de pines del valor inicial

Conectado a cada una de las patillas anteriores, se tendrá un circuito como el mostrado a continuación:

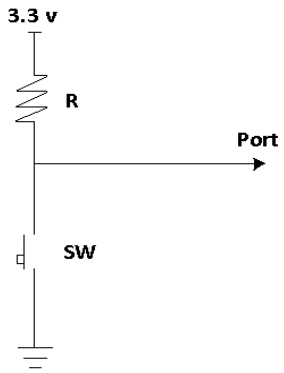


Figura 3. Circuito de conexión a cada patilla del puerto de entrada.

donde SW será una de las patilla de un microswitch que nos permite en una posición unir eléctricamente los dos extremos del microswitch (con lo que Port estaría conectado directamente a masa), o, en la otra posición no hacer esa unión (con lo que Port estaría conecatado a 3.3v a través de la resistencia R).

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

- En los pines del conector P2 que se indican en la siguiente tabla. En este conector se mostrara el resultado en dos formatos en función del valor del pulsador ISP. En el caso de que este pulsado, se mostrara el número el formato binario en las primeras 16 patillas de señal del conector, y en caso de que se encuentre libre se mostrara el número en formato BCD en las 19 primeras patillas de señal

del conector. **Recuerde realizar la conexión al diodo utilizando unas resistencias.¿Cúal sería el valor en ohmios de éstas?**

Pines del Conector P2	Bits del número primo
Patilla 2 (GPIO P0.3)	Bit 0
Patilla 3 (GPIO P0.2)	Bit 1
Patilla 6 (GPIO P1.0)	Bit 2
Patilla 7 (GPIO P1.1)	Bit 3
Patilla 8 (GPIO P1.4)	Bit 4
Patilla 9 (GPIO P1.8)	Bit 5
Patilla 10 (GPIO P1.9)	Bit 6
Patilla 11 (GPIO P1.10)	Bit 7
Patilla 12 (GPIO P1.14)	Bit 8
Patilla 13 (GPIO P1.15)	Bit 9
Patilla 14 (GPIO P1.16)	Bit 10
Patilla 15 (GPIO P1.17)	Bit 11
Patilla 16 (GPIO P4.29)	Bit 12
Patilla 19 (GPIO P4.28)	Bit 13
Patilla 20 (GPIO P0.4)	Bit 14
Patilla 21 (GPIO P0.5)	Bit 15
Patilla 22 (GPIO P0.6)	Bit 16
Patilla 23 (GPIO P0.7)	Bit 17
Patilla 24 (GPIO P0.8)	Bit 18

Tabla 2. Esquema de representación del numero primo en binario/bcd

A cada uno de estas patillas del conector se conectará un diodo led que emitirá luz cuando el valor del bit sea '1' y estará apagado si el valor de bit asociado vale '0'. **Recuerde realizar la conexión al diodo utilizando una resistencia. Calcule cual sería el valor de la resistencia adecuado según las características del diodo led.**

- En los pines del conector P2 que se indican en la siguiente tabla. 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. **Recuerde realizar la conexión a cada segmento utilizando una resistencia.**

Pines del Conector P2	Segmentos del display
Patilla 26 (GPIO P2.0)	Segmento A
Patilla 27 (GPIO P2.1)	Segmento B
Patilla 28 (GPIO P2.2)	Segmento C
Patilla 31 (GPIO P2.3)	Segmento D
Patilla 32 (GPIO P2.4)	Segmento E
Patilla 33 (GPIO P2.5)	Segmento F
Patilla 34 (GPIO P2.6)	Segmento G
Patilla 35 (GPIO P2.7)	Ánodo

Tabla 3. Esquema de representación de los dígitos del numero primo en display de 7 segmentos

- En los diodos LD2 y LD3. El estado del diodo LD2 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 LD3 estará iluminando si el bit correspondiente del número primo es un 1 o estará apagado si su bit correspondiente es 0, comenzando por el bit menos significativo (LSB).

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.Toma de datos

Realícese un proyecto que tome los datos de los pines mencionados en la Tabla 1. Para eso, realícese el montaje hardware con la conexión a las patillas de los microswitches. A continuación, créese en el entorno de desarrollo un proyecto donde se defina la función *"uint16_t leedato()"*, que cuando se invoca retorna el número formado por los pines mencionados en dicha tabla.

```
#include "cmsis os.h"
#include "lpc17XX.h"
uint16_t leedato(){
uint16 t dato;
dato=…
…


return dato;
}

int main(){
uint16 t valor;
…
valor=leedato();
while(1);
}
```

Compruebese, mediante depuración y situando breakpoints, que variando los valores del microswitch conectado al conector P1, los valores que se toman son los correctos.

Paso 2. Representación en binario

Realícese un proyecto que muestre un numero en binario en los pines mencionados en la Tabla 2. Para eso, añada al montaje realizado en el paso 1, el circuito de los diodos (y su resistencias asociadas), conectados a las patillas más bajas del conector P2. En el entorno de depuración, créese un proyecto donde esté definida la función *"void muestrabinario(uint16_t numero)"*, que cuando se invoca muestra cada bit del número que se le pasa por parámetro en uno de los diodos.

```
#include "cmsis_os.h"
#include "lpc17XX.h"

void muestrabinario(uint16_t numero){
…


}

int main(){
uint16_t valor;
…
valor=…;
muestrabinario(valor);
while(1);
}
```

Compruebese, mediante depuración y situando breakpoints, que variando los valores de la variable *valor*, los valores que se muestran en los diodos conectados al conector P2 son los correctos.

Paso 3. Representación en bcd

Realícese un proyecto que muestre un numero en bcd en los pines mencionados en la Tabla 2. Para eso, usaremos el montaje hardware realizado en el paso 2. En el entorno de depuración, créese un proyecto donde estén definidas dos funciones

- *"void muestrabcd(uint32_t numero)"*, que cuando se invoca muestra cada bit del número que se le pasa por parámetro en uno de los diodos.
- *"uint32_t binario_a_bcd(uint16_t numero)",* que convierte el dato de entrada numero expresado en binario a su representación en bcd, que es la que retorna

```
#include "cmsis_os.h"
#include "lpc17XX.h"

void muestrabcd(uint32_t numero){
…


}

uint32_t binario_a_bcd(uint16_t numero){
…


}


int main(){
uint16_t valor;
uint32_t valorbcd
…
valor=…;
valorbcd=binario_a_bcd(valor);
valorbcd=muestrabcd(valorbcd);
while(1);
}
```

Compruebese mediante depuración y situando breakpoints, que variando los valores de la variable valor los valores que se muestran en los diodos conectados al conector P2 son los correctos.

Paso 4. Lector del pulsador ISP

Realícese un proyecto que muestre el valor que tiene en cada momento el swicth ISP. En el entorno de depuración, créese un proyecto donde esté definida la función *"uint8_t isp()"*, que cuando se invoca retorna 1 si el pulsador ISP se encuentra pulsado, y 0 en el caso de que no.

```
#include "cmsis os.h"
#include "lpc17XX.h"

uint8 t isp(){
…


}

int main(){
uint8_t pulsado;
…

pulsado=isp();
while(1);
}
```

Compruebese mediante depuración y situando breakpoints, que pulsando y liberando el pulsador ISP, la aplicación detecta que el valor obtenido es el correcto.

Paso 5. Muestra en binario o bcd

Realícese un proyecto que muestre el valor de un número en formato binario o bcd, en función del valor del pulsador ISP. En el entorno de depuración, créese un proyecto y utilícense las funciones desarrolladas en los tres proyectos anteriores, para conseguir que en los diodos se muestre el valor de un número en formato binario o bcd según se encuentre pulsado o no el pulsador ISP.

```
#include "cmsis_os.h"
#include "lpc17XX.h"

uint8_t isp(){
…
}
void muestrabcd(uint32_t numero){
…
}
uint32_t binario_a_bcd(uint16_t numero){
…
}
void muestrabinario(uint16_t numero){
…
}

int main(){
uint16 t valor;
…
valor=…;
…
while(1);
}
```

Compruebese mediante depuración y situando breakpoints, que pulsando y liberando el pulsador ISP, se muestra el valor correcto en los diodos en el formato adecuado.

Paso 6. Muestra en display de 7 segmentos

Realícese un proyecto que muestre el valor de un número en un display de 7 segmentos. En el entorno de depuración, créese un proyecto y defina la función *"void display(unit8_t digito)"*, que muestre en el display de 7 segmentos el valor de la variable dígito(teniendo en cuenta que su valor estará entre 0 y 9).

```
#include "cmsis os.h"
#include "lpc17XX.h"

display(uint8 t){
…
}

int main(){
uint16 t valor;
…
valor=…;
display(valor);
while(1);
}
```

Compruebese mediante depuración y situando breakpoints, que se muestra en el display el valor correcto de la variable introducida.

Paso 7. Muestra en display de 7 segmentos un digito de una variable

Realícese un proyecto que muestre el valor de un dígito concreto de un número en un display de 7 segmentos. En el entorno de depuración, créese un proyecto y defina la función *"void displaydigito(unit16_t numero,uint8_t digito)"*, que muestre en el display de 7 segmentos el dígito *digito* (que puede valer 1(unidades), 2(decenas), 3 (centenas) ,4 (unidades de millar) ó 5(decenas de millar)) de la variable numero. Puede utilizar para implementar esta función la función *display* obtenida en el paso anterior.

```
#include "cmsis_os.h"
#include "lpc17XX.h"

display(uint8_t){
…
}

void displaydigito(){
…
display(…)
}

int main(){
uint16_t valor;
…
valor=…;
displaydigito(1,valor);
while(1);
}
```

Compruebese mediante depuración y situando breakpoints, que se muestra en el display el valor correcto de la variable introducida.

Paso 8. Conmutación del estado del diodo LD2

Realícese un proyecto que conmute el estado del diodo LD2. En el entorno de depuración, créese un proyecto y defina la función *"void ld2()"*, que cambia el estado de un diodo. Es decir, si se encontraba apagado, lo enciende, y viceversa.

```
#include "cmsis os.h"
#include "lpc17XX.h"

void ld2(){
…
}

int main(){

ld2();
while(1);
}
```

Compruebese mediante depuración y breakpoints que el diodo varia su estado en la invocación a la función.

Paso 9. Representación en el diodo LD3 de un bit de una variable

Realícese un proyecto que represente en el diodo LD3 el valor de un determinado bit de una variable. En el entorno de depuración, créese un proyecto y defina la función *"void ld3(uint8_t bit,uint16_t numero)"*, que representa en el diodo LD3 el valor del bit *bit* (variable cuyo valor estará entre 0 y 15) de la variable *numero*. Es decir, si se invoca la función ld3(5,valor), el diodo ld3 emitirá luz si el bit 5 de la variable valor es '1', y estará apagado si el bit 5 de la variable valor es '0'.

```
#include "cmsis os.h"
#include "lpc17XX.h"

void ld3(uint8 t bit, uint16 t numero){
…
}

int main(){
uint16 t valor;
valor=…;
ld3(5,valor);
while(1);
}
```

Compruebese mediante depuración y breakpoints que el diodo se luce o no en función del valor del bit que se esté representando en él.

Paso Final. Desarrollo de la práctica.

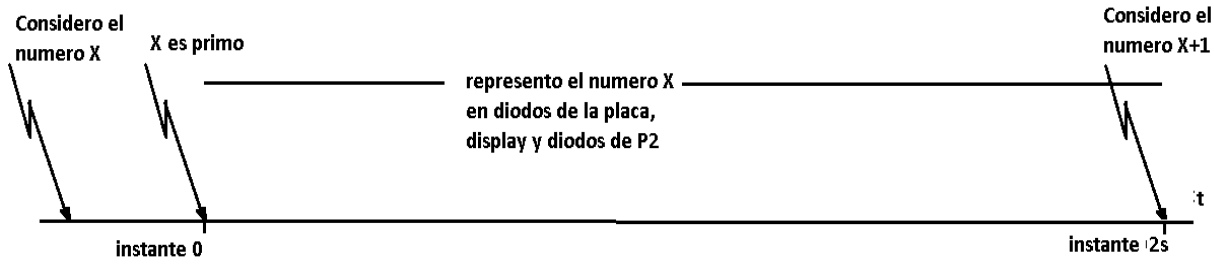
Una vez llegados a este punto, se encuentran desarrolladas todas las funciones que se van a utilizar en el desarrollo de la práctica. Estas funciones son todas las desarrolladas en los pasos anteriores, así como la función delay() (definida en el programa de ejemplo de esta practica, en la sección "Simulación on-chip"), y la función esprimo() definida en la practica anterior. Este paso final consiste en organizar y sincronizar las llamadas a cada una de las funciones desarrolladas.

Para eso, es conveniente, antes de nada, realizar un flujograma que nos indique las acciones que tiene que llevar a cabo nuestro programa. Realícese, pues, dicho flujograma, que describa este funcionamiento.

Dentro de dicho flujograma existirá una "caja" que se refiera a la representación de un número primo en display y diodos (ld2, ld3 y los conectados a P2).

Para abordar la realización de la mencionada caja, puede resultar útil la elaboración de un cronograma que nos permita aclarar, para cada número primo calculado, las acciones que se deben realizar así como el instante en que se deben llevar a cabo esas acciones. Para tal fin, rellénese el siguiente gráfico con esa información, basándonos en los requisitos que nos solicitan en el enunciado de la práctica. Por ejemplo, se podría rellenar que

- En el instante 0 se debe representar en LD3 el bit 0 del numero X (llamando a la función ld3)
- En el instante 125 mseg se debe representar en LD3 el bit1 del numero X(usando la función ld3)
- En el instante 125 mseg se debe conmutar el valor de LD2 (mediante la función ld2)
- …



Se observa rellenando el cronograma anterior, que se debe realizar una acción en un momento puntual, y a continuación, dejar transcurrir un determinado tiempo hasta que se realice la siguiente acción. Para dejar pasar ese tiempo determinado se puede utilizar la función "*void delay(uint32_t n)*" . Para conocer cuanto tiempo transcurre durante su ejecución para un determinado valor de n, se puede utilizar la información del tiempo que transcurre durante la ejecución del programa (mostrado en la parte inferior derecha de la

herramienta Keil uVision cuando se encuentra en simulación). Así, si nos fijamos en el valor del tiempo justo antes de la invocación a la función delay(), y justo tras la finalización de la ejecución de dicha función, el tiempo empleado en ejecutar dicha función será la resta de ambas cantidades. De esa forma, para conseguir un tiempo determinado, se puede ajustar el valor de n siguiendo este procedimiento.