Aitor Ingelmo Martin

Memoria de la práctica D de Sistemas electrónicos digitales

1. Material utilizado en la práctica

En esta sección se proporcionará información de los elementos externos al LPC1768 empleados para las siguientes especificaciones de la práctica.

Representación de temperaturas

Para representar la temperatura se han seleccionado dos Displays 7 segmentos cátodo común. La razón de elegir displays cátodo común fue el poder codificar los valores de los display de una manera más intuitiva, ya que se requieren salidas con un nivel alto para encender un segmento.

Selección de temperaturas

Para poder introducir al sistema las temperaturas medidas por los sensores, he dispuesto de dos microswitch 4 contactos para poder introducir los valores de forma cómoda y visual.

Leds adicionales

De manera adicional a los leds especificados por el guión, se ha dispuesto de un total de 3 leds adicionales para el montaje, cuya función es obtener información del estado del sistema de forma visual.

- **Led verde:** Se ilumina cada vez que pulsamos el botón EINT1. No tiene una implementación software, aprovecha los valores obtenidos del pin 2.11 al apretar el pulsador.
- **Led Amarillo:**Se ilumina cada vez que pulsamos el botón EINT2. No tiene una implementación software, aprovecha los valores obtenidos del pin 2.12 al apretar el pulsador
- **Led blanco:**Se iluminará cuando introducimos la temperatura programada.

2. Variables declaradas

En este apartado se revisarán las variables empleadas para el sistema así como la justificación de su declaración y uso. Todas las variables usadas en el código (a excepción de tres que se detallarán más adelante) son globales. Esta decisión se tomó para no tener que crear funciones que devolvieran parámetros, y poder leer desde cualquier otra función valores que definen el estado del sistema.

Variables lógicas

```
5  #define true 1
6  #define false 0
```

El uso de estos dos defines ha sido clave a la hora de emplear sentencias lógicas dicotómicas en el código. Todas las líneas de código emplean “true” o

“false” para definir si se cumplen o no ciertas situaciones. Cabe destacar que son dos variables que se pueden eliminar, y usar 1s y 0s en su lugar, pero su uso permite comprender el código más rápido.

Variables para la temperatura

```
8       //Temperatura media medida
9        int8_t temperaturaMediaMedida =0;
10       int8_t errorEnMedia=false;
```

Estas variables se emplean para obtener la temperatura media medida y para detectar si tenemos error en dicha medida. Las dos temperaturas se inicializan a 0 para un correcto funcionamiento inicial del sistema.

Variables para la representación

El uso de estas variables está relacionado con la obtención de información para la representación de valores en los displays.

```
17 //RepresentaciÐn
18 int8_t temperaturaMediaMedida = 0;
19 int8_t errorEnMedia = false;
20 int8_t temperaturaSensor1 = 0;
21 int8_t temperaturaSensor2 = 0;
22 int8_t conmutador = 0;
23 int8_t unidades = 0;
24 int8_t decenas = 0;
25 int8_t valorDelPin1 = 0;
26 int8_t valorDelPin2 = 0;
```

- **Temperatura Media Medida:** Almacena el valor medio medido por los sensores.
- **Conmutador:** Permite controlar el estado de la multiplexación de los displays.
- **Decenas/Unidades:**Contienen los valores de las unidades/decenas a representar.
- **Valor del pin 1/2:** Se emplea para determinar la posición a consultar de la matriz de valores del display.
- **Error en media:** Es una variable empleada para determinar si hay un error en media o no. Puede cambiarse por un boolean.
- Temperatura sensor 1/2: Almacenan las temperaturas medidas.

Variables para el display

Este conjunto de variables contienen todas las posibles posibilidades a representar en el display. Hay que tener en cuenta que la matriz Disp[10] está pensada para un display cátodo común.

```
28 //Displays
29 int8_t SignoMenos = 0x40;
30 int8_t LetraA = 0x77;
31 int8_t LetraL = 0x38;
32
33 uint16_t Disp[10] = {0x3F,  //0
34             0x06,  //1
35             0x5B,  //2
36             0x4F,  //3
37             0x66,  //4
38             0x6D,  //5
39             0x7C,  //6
40             0x07,  //7
41             0x7F,  //8
42             0x67}; //9
```

Variables para el modo de programa

Estas dos variables son empleadas para inicializar la temperatura programada predefinida (igualada a 12, pero es un valor irrelevante mientras esté en el rango de representación) y tener un conocimiento en todo momento del modo en el que estamos, normal o programar temperatura.

```
 9 int8_t temperaturaProgramada = 12;
10 int8_t mododeprograma = 0;
```

Variables para el systick

Estas variables son empleadas por el systick para su correcto funcionamiento. Hora representa la hora actual del sistema y horaFija es una variable empleada para comprobar si se está en las horas restringidas. Respecto a la variable temperaturaPredefinida cabe decir que no es necesaria para el funcionamiento del sistema, pero su uso permite modificar el valor de la temperatura fija sin necesidad de modificar el resto del código.

```
12 //Variables Systick
13 int8_t hora = 9;
14 int8_t horaFija = false;
15 int8_t ticks = 0;
```

3. Configuración de GPIOs

La selección de pines ha seguido como idea de diseño el disponer de pines contiguos, y que su implementación en una protoboard nunca implique el tener cables que crucen por encima de la placa. Además, sólo se seleccionaron bits del puerto 1 para la salida hacia los displays, por motivos de facilidad en el

código, y todos los pines conectados a led tienen sus correspondientes resistencias internas.
Bajo esta premisa, se han seleccionado las siguientes agrupaciones de pines y configuraciones.

- **Entradas de interrupción:** Se seleccionaron los botones ya integrados en la placa. En la función "configuracionDelSistema" se configuran los registros extmode/extpolar para que se activen por flanco de subida.
- **Pines A0/A1:** Son los pines correspondientes a la multiplexación. Se irán activando y desactivando en función de que display se debe representar. Se han adjudicado los pines "P2.03", "P2.04" respectivamente
- **Leds:** Se establecieron los pines "P2.00", "P2.01" y "P2.06" para los leds de frío, caliente y modo programa respectivamente.

```
59  //----led modo prog
60    LPC_PINCON->PINSEL4 &= ~(0x03 << 6);  // Función GPIO
61    LPC_PINCON->PINMODE4 &= ~(0x02 << 6); // Nada
62    LPC_GPIO2->FIODIR &= ~(0x1 << 6);     // bit P2.06 definido como entrada
63    //----led caliente
64    LPC_PINCON->PINSEL4 &= ~(0x3 << 4);   // Función GPIO
65    LPC_PINCON->PINMODE4 &= ~(0x3 << 4);  // Resistencia pull-up
66    LPC_GPIO2->FIODIR &= ~(0x1 << 4);     // bit P2.04 definido como entradae
67
68  //----led modo frio
69    LPC_PINCON->PINSEL4 &= ~(0x3);       // Función GPIO
70    LPC_PINCON->PINMODE4 &= ~(0x2);      // Nada
71    LPC_GPIO2->FIODIR &= ~(0x1);         // bit P2.00 definido como entrada
```

4. Configuración de las interrupciones

Siguiendo las especificaciones del guión de la práctica, para este diseño se han utilizado un total de 5 interrupciones con las siguientes configuraciones.

```
87  //Asignacion de prioridades
88  NVIC_SetPriorityGrouping(3);
89  NVIC_SetPriority(TIMER0_IRQn, 0x0); //Timer0
90  NVIC_SetPriority(SysTick_IRQn, 0x1);
91  NVIC_SetPriority(EINT0_IRQn, 0x20); //BOTONES
92  NVIC_SetPriority(EINT1_IRQn, 0x40);
93  NVIC_SetPriority(EINT2_IRQn, 0x40);
```

Estos niveles de prioridad se han fijado de esta manera para que el timer tenga la prioridad más alta seguida del systick, y a continuación las prioridades de los botones. De esta forma, configurando el "PriorityGrouping" a 3, las interrupciones de los botones pueden con una menor prioridad, pueden desalojar tanto al timer como al systick sin perturbar el correcto funcionamiento del sistema.

5. Configuración del Timer/Systick

En este apartado se va a justificar los valores establecidos en los registros tanto del timer1 como del systick.

Systick

El registro del Systick CTRL tiene el valor "0x7", con lo que se habilita la cuenta del systick, el que pueda interrumpir y que se tome como CLKSOURCE es de la CPU, es

decir, 100Mhz. Además, VAL está a 0 para que empiece a contar desde ahí, y LOAD se ha fijado a “0x11E1A2FF”, lo que significa que interrumpe cada 3 segundos.

Time1

El TIMER1 se ha configurado con un PCLK=PCLK/4, es decir, con una frecuencia de 25Mhz. Este valor se ha tomado ya que, como se debe leer cada 7 segundos la temperatura de los sensores, no requerimos un PCLK demasiado rápido. Además, PR se ha fijado a 0, con lo que se contará cada ciclo de reloj.

Los registros MR0 y MR1 se han configurado con 1 y 100000 respectivamente y se ha configurando MCR para que cuando TC alcance el valor de MR0 interrumpa, y cuando alcance el de MR1 interrumpa y se resetee. Esto nos permite ir actualizando el display a una velocidad mayor que la del ojo, y a la vez marcar un tiempo que usaré para leer los sensores.

Las interrupciones explicadas en los siguientes apartados describen su funcionamiento y algunas un pequeño grafo junto a su código. La justificación de esto es poder entender fácilmente la intención de la función, así como su lógica de funcionamiento.

6. Funcionalidad correcta de ISRs

Este apartado define el funcionamiento de las rutinas de atención a la interrupción empleadas por el sistema por orden de aparición en el código fuente:

TIMER1_IRQHandler

Timer1 interrumpe cada ciclo del reloj programado, y cada 7 segundos (en el código está con un valor más pequeño para hacer pruebas más cómodamente). Las interrupciones ocurridas en cada ciclo se emplean para poder ir realizando la multiplexación, mientras que la interrupción ocurrida cada 7 segundos se emplea para leer los sensores y así actualizar los valores almacenados.

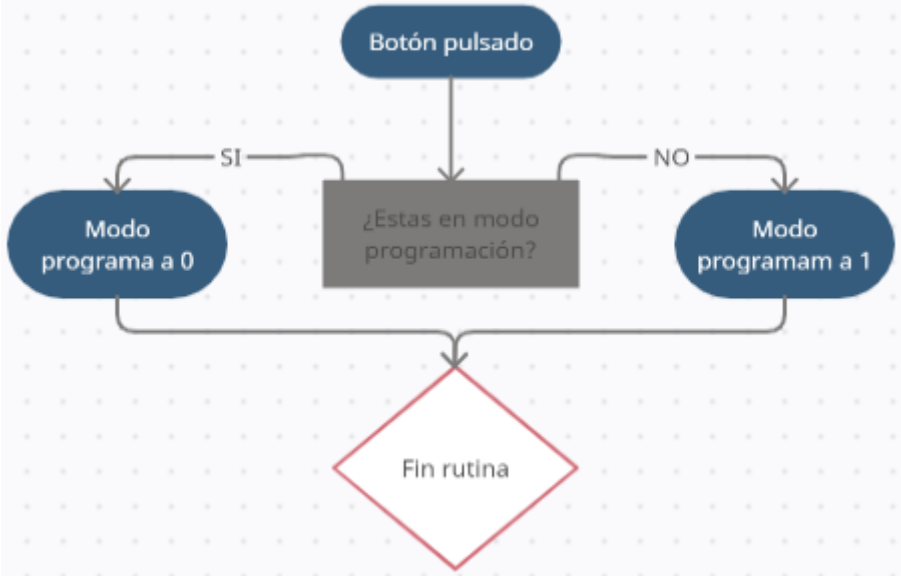
```
346 void TIMER1_IRQHandler(void){
347
348   LPC_TIM1->IR = 0x1 << 0;
349   if (LPC_TIM1->MR1 == LPC_TIM1->TC){
350     leerSensores();
351   }
352   if (horaFija == false){
353     if (mododeprograma == 0){
354       //Modo "normal"
355       if (errorEnMedia == false){
356         representarTemperatura(0);
357       }
358       else{
359         LPC_GPIO2->FIOSET = ((0x01) | (0x01 << 1) | (0x01 << 6));
360         multiplexacion(2);
361       }
362     }
363     else{
364       representarTemperatura(1);
365     }
366   }
367   else{
368     LPC_GPIO2->FIOSET = ((0x01) | (0x01 << 1) | (0x01 << 6));
369     mododeprograma=0;
370     multiplexacion(3);
371   }
372 }
```

…as horas determinadas. …permite contar cuántos …dos, y poder simular que …eja diciendo que cada 60 …ar el paso del tiempo y comprobar que de 8 a 22 la temperatura es fija. En caso de estar entre las 8 y las 22, aunque las rutinas EINT0/1/2 se ejecuten, sus funciones no hacen nada al estar restringidas por la condición de que no se esté en las horas fijadas.

```
329 void SysTick_Handler(void){
330   ticks++;
331   if (ticks == 60){
332     hora++;
333     if (hora > 24){
334       hora = 0;
335     }
336
337     if ((hora >= 22) || (hora <= 8)){
338       horaFija = true;
339     }
340     else{
341       horaFija = false;
342     }
343     ticks = 0;
344   }
345 }
```

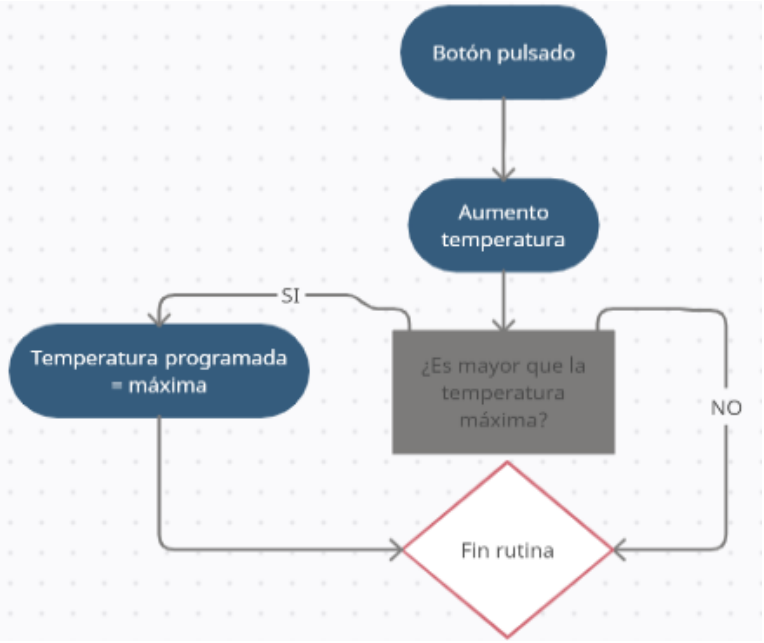
EINT0_IRQHandler

Esta rutina está asociada al botón para cambiar el modo del sistema. Su funcionamiento es muy básico y se basa en la evaluación de sentencias lógicas para determinar el valor de la variable modoPrograma.



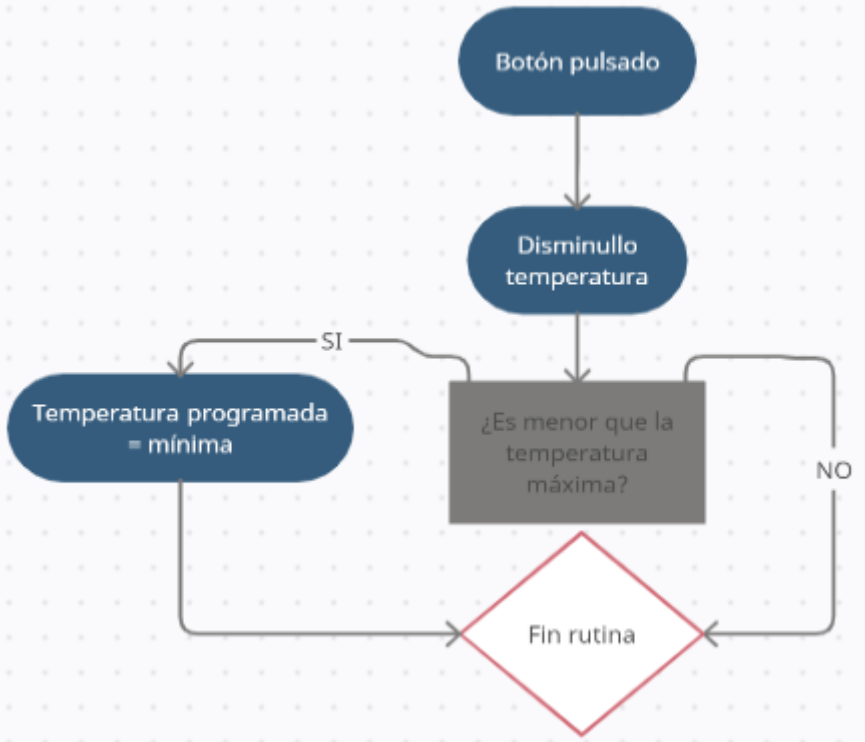
EINT1_IRQHandler

Esta rutina es la encargada de aumentar en una unidad la temperatura programada. Mediante una sentencia “if” se controla que no se salga del valor máximo.



EINT2_IRQHandler

Igual que la EINT1, pero restando temperatura



Códigos de las ISR

```
126 //ISR de las interrupciones
127 void EINT0_IRQHandler(){ //Rutina para el modo del programa. Entro si pulso el ISP
128
129   LPC_SC->EXTINT |= (1);
130   if (mododeprograma == 1){
131     mododeprograma = 0;
132     LPC_GPIO2->FIOSET |= (0x01 << 6);
133     LPC_GPIO2->FIOCLR = ((0x01 << 1) || (0x01));
134   }
135   else{
136     mododeprograma = 1;
137     LPC_GPIO2->FIOCLR |= (0x01 << 6);
138     LPC_GPIO2->FIODIR |= (0x01 << 6);
139   }
140 }
141 //Rutina KEY1
142 void EINT1_IRQHandler(){ //Rutina para aumentar la temperatura
143
144   LPC_SC->EXTINT |= (1 << 1);
145   if (mododeprograma == 1){
146     temperaturaProgramada++;
147
148     if ((temperaturaProgramada > 35)){
149       temperaturaProgramada = 35;
150     }
151   }
152 }
153
154 //Rutina KEY2
155 void EINT2_IRQHandler(){ //Rutina para disminuir la temperatura
156
157   LPC_SC->EXTINT |= (1 << 2);
158   if (mododeprograma == 1){
159     temperaturaProgramada--;
160     if (temperaturaProgramada < (-9)){
161       temperaturaProgramada = (-9);
162     }
163   }
164 }
```

7. Explicación de las funciones

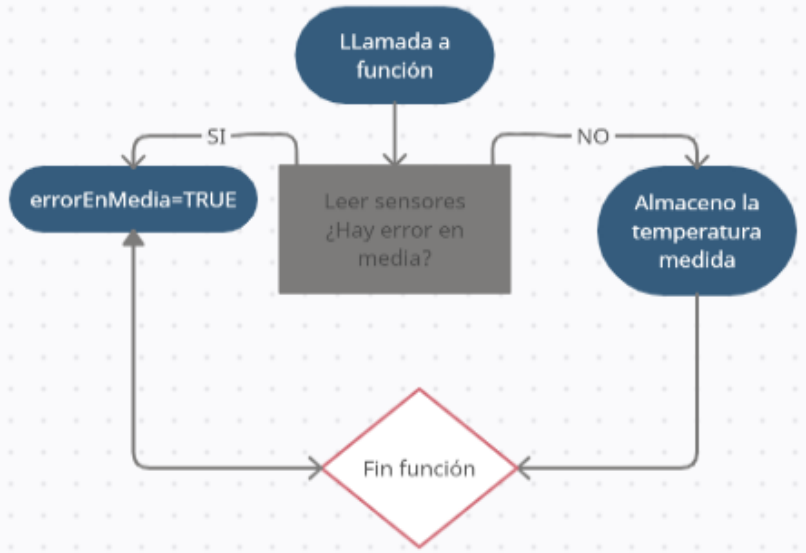
En este apartado se desarrollaran las explicaciones y motivaciones de cada función, junto con algunos grafos, para explicarlas de una manera más visual. En todas las funciones se ha aprovechado el uso de variables globales para no tener que construir funciones que devuelvan un resultado.

En este apartado no se incluye las funciones “inicioHardware” ni “configuracionDelSistema” ya que controlan registros y no presentan lógica alguna.

leerSensores

Esta función se creó para permitir ahorrar la acción de leer siempre que tenga se tenga una temperatura fija. De esta forma, no se consumen recursos en un proceso que no se necesita en ese momento.

Esta función se encarga de leer los sensores, detectar sus errores y obtener el valor representado según una función matemática que devuelve valores entre 35 y -9.

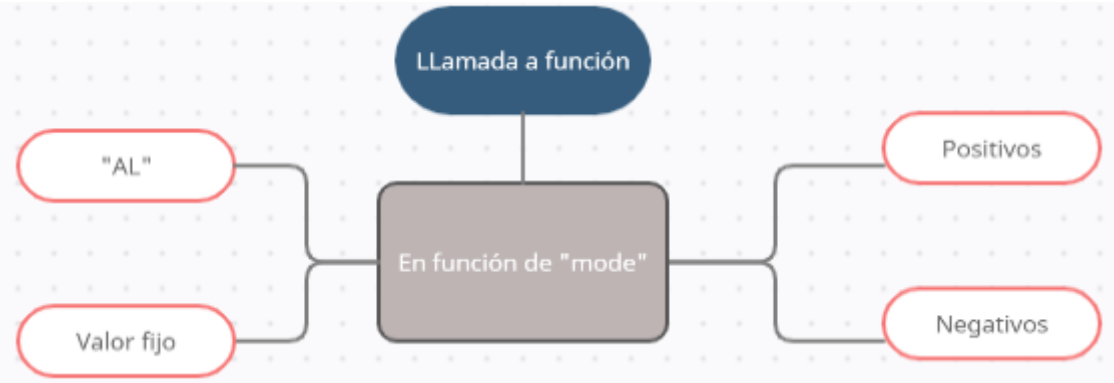


Multiplexación

La función de “multiplexación” se ideó como solución para poder representar en los displays los valores deseados. Su estructura es la unión de las 4 posibilidades a representar (valores medidos, valores programados, fuera de rango, valor fijado).

Originalmente fue diseñada con 4 variables pasadas por parámetro y un total de 6 posibles modos de representación, pero tras detectar que 2 de los estados eran redundantes y que las 4 variables podrían sustituirse por una, se llegó a la estructura actual de 4 modos y una variable por parámetro.

La idea de la función es que en base al valor de “modo” proporcionado al invocar la función, se realice un tipo de representación u otro. Para ello se emplean sentencias If que permiten discriminar el tipo de representación. Computacionalmente es mejor el uso de estructuras “switch-case” cuando se trabaja con un número amplio de sentencias If. Pero al tratarse de 4 he preferido dejarlas para poder separar cada uno de los modos y poder clonarlos más cómodamente.



representarTemperatura

Esta función se construyó por la necesidad de discriminar por un lado el modo de la función multiplexación y por otro, qué temperatura, entre la programada y la medida, es la mayor.
Al ser invocada, primero se aplican una serie de sentencias lógicas mediante las cuales se determina si la temperatura programada es mayor, menor o igual a la temperatura medida. En función del resultado se encendera el leds de calor/frío según corresponda.
Tras esta primera etapa, la función evalúa si el sistema está en el modo programación o no mediante la evaluación del parámetro “modo”.La única diferencia entre ambos modos está en cómo pasa se guardan las unidades en el caso en que sean negativas.

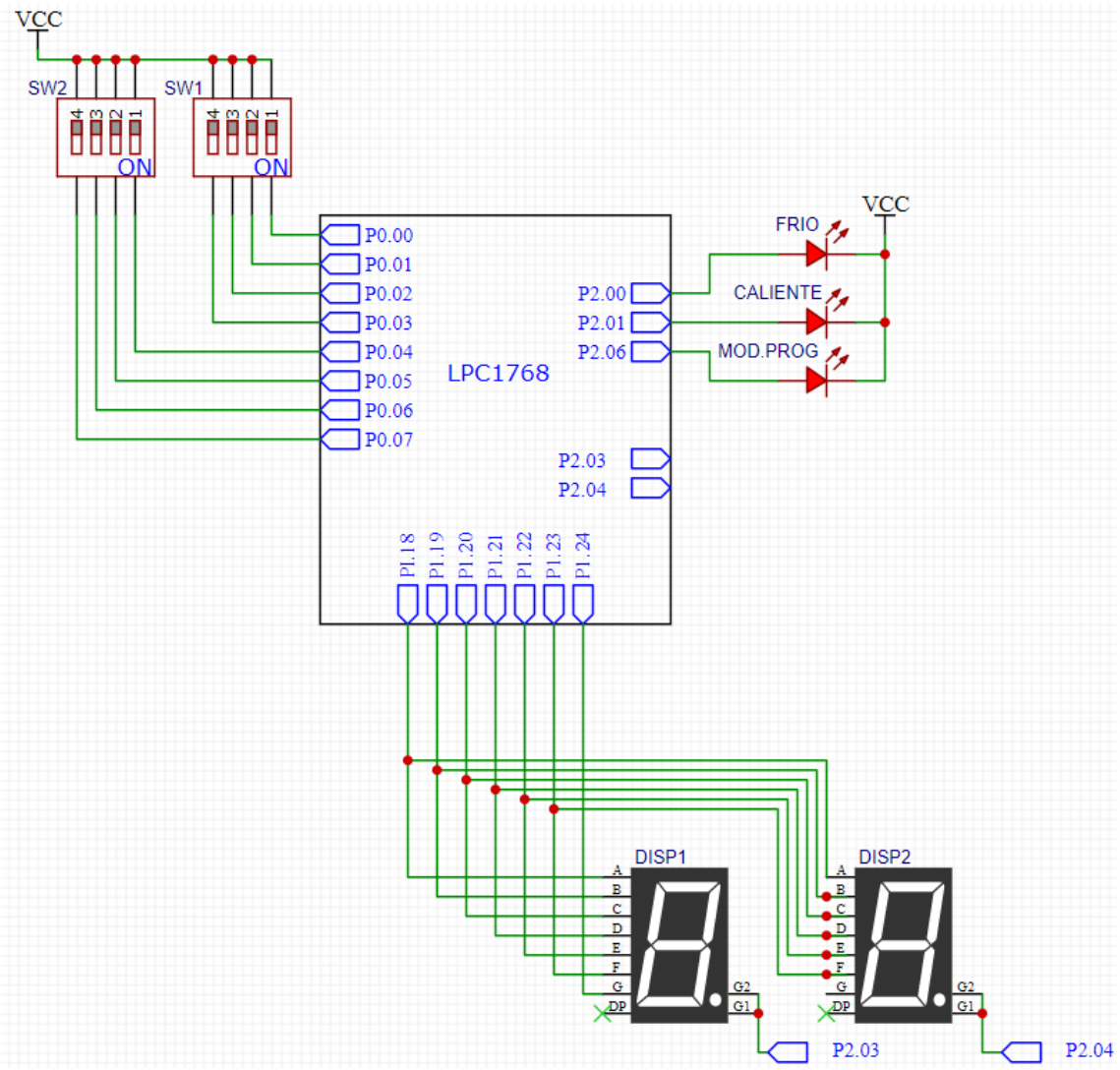
8. Evolución de Ideas

En este apartado se va a detallar brevemente algunas ideas iniciales o desechadas durante la realización de la práctica.

- En un inicio se plantearon 3 funciones de inicio, una para las interrupciones, otra para el systick y otra para el timer. Finalmente esas 3 funciones han sido absorbidas por una mucho mayor llamada configuracionDelSIstema.
- La función multiplexación no se creó hasta el final del desarrollo, inicialmente a partir de una serie de sentencias “if” y comprobaciones repetitivas de unidades, se ejecutaba tal cual el fragmento de código que representa los displays. Esta opción era insostenible y repetía operaciones, con lo que se simplificó la lógica y se idearon varias versiones de esta función hasta llegar a la actual.
- Desde un inicio del desarrollo, se programaron los pines escribiendo en los registros valores hexadecimales muy grandes. A mitad se decidió cambiar por la modificación selectiva de bits mediante máscaras, lo cual es mucho más legible y cómodo.
- En un principio los pines de los displays se limpiaban de manera selectiva, pero como pertenecen al mismo puerto se prefirió limpiar todo el puerto.
- En un inicio se crearon 2 variables para los displays, con el paso del tiempo me di cuenta de que sólo hacía falta una, con lo que se eliminó la segunda y se llamó a la restante Disp.

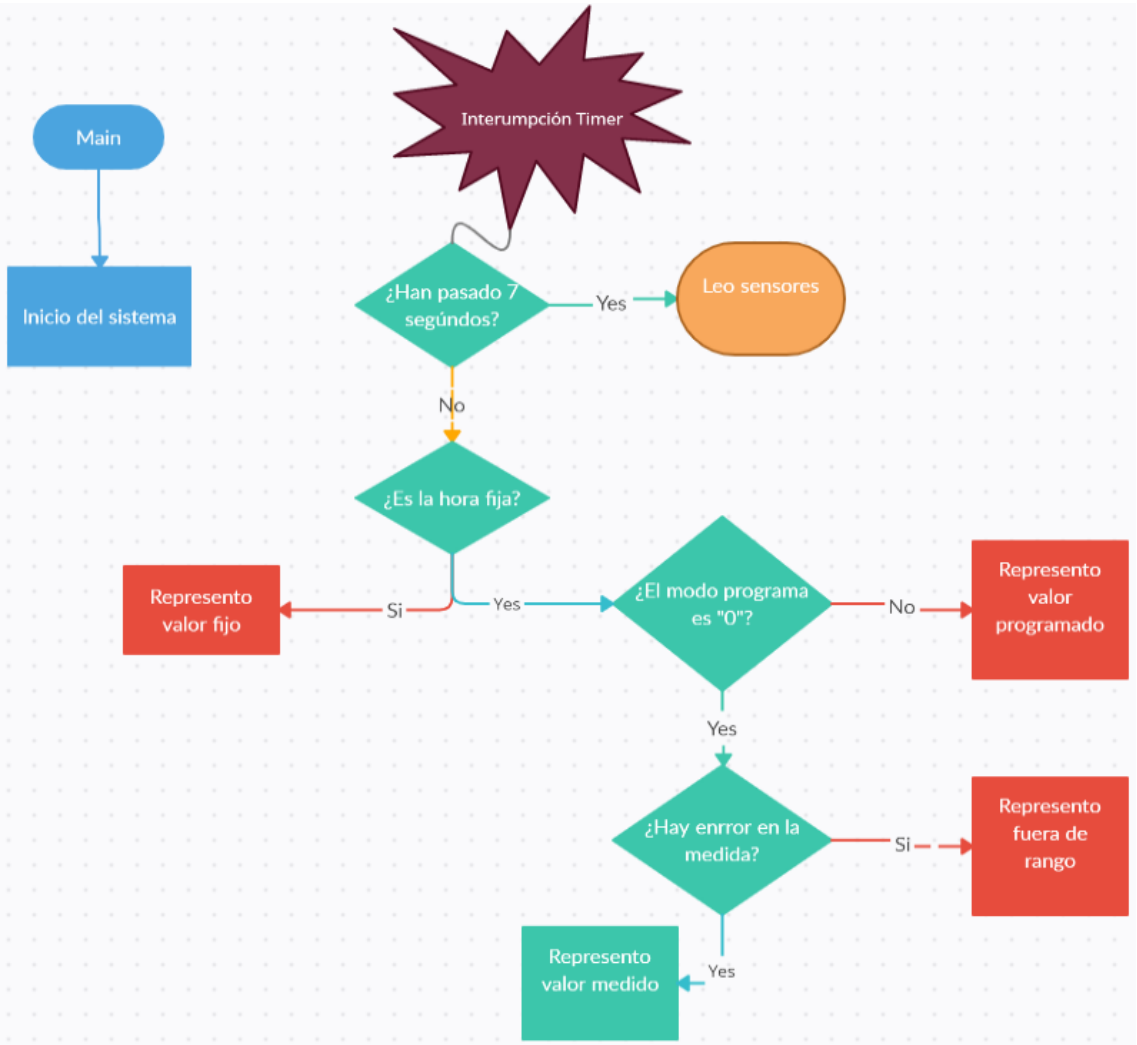
9. Diseño completo

   En este apartado se presenta un esquemático que representa el esquema de pines y los hardware empleados en la práctica. No se han representado las resistencias internas.



10. Funcionamiento correcto

Este grafo representa el funcionamiento general del sistema. Al margen y en azul están los estados de inicio del sistema.



11. Codificación realizada- Estilo de programación

El estilo de programación del código ha pretendido seguir siempre las siguientes intenciones sintácticas empleadas:

- Empleo de nombre metaexplicativos tanto para funciones como para variables.
- Utilizar (en la medida de lo posible) un sistema de tabulaciones tanto en el interior de funciones como en el desarrollo de sentencias lógicas.
- El uso de declarar funciones y variables compuestas siguiendo el siguiente esquema [nombreApellido1Apellido2].
- La realización de funciones con la idea de ser reutilizables en otros proyectos.

12. Alternativas incluidas

En el apartado de alternativas, se plantean 2 posibilidades con mejoras independientes. Una de ellas es sustituir de entrada el microcontrolador por otro menos potente (A) y otra en la que se amplía el desarrollo con el LPC1768 (B).

A)La opción de cambiar la placa viene motivada por el precio del LC1768 y la reducida existencia de módulos baratos adicionales para el mismo.Como este proyecto es modesto, podemos utilizar un microcontrolador más barato. En lo personal optaría por emplear alguna placa arduino oficial tanto por el precio de la placa como por su filosofía de apoyo entre usuarios, lo que otorga un abanico de librerías adicionales.

Suponiendo que elegimos un arduino ATMEGA por no tener problemas con los pines que necesitamos para el sistema, podríamos emplear multitud de sensores adicionales (de humo, de presión, de humedad...)para tener un mayor control a nave industrial. Aunque hay que tener en cuenta que no se puede emplear Arduino para uso comercial homologable.

B)La opción de mantener el LPC1768 para un proyecto tan pequeño significa que además de este sistema, queremos implementar más funcionalidades. En c, pasaría por emplear un menú con un LCD, el uso del DAC para la obtención de la temperatura de una manera más precisa y con decimales mediante un potenciómetro, aprovechar su puerto ethernet para poder configurar la placa desde el móvil y obtener por ahí los datos, o añadir los mísmos sensores de que dispone arduino adaptándolos al LPC.

Al margen de estas dos opciones, se podría sustituir los leds de calentado o enfriado por un led RGB que sólo se ilumine el rojo y el azúl para ahorrarte 2 leds en caso de querer pasar el diseño a una PCB y comercializarlo o reducir líneas de código sustituibles por una única línea, como el caso de a continuación que puede sustituirse por una línea que modifique el regístro FIOPIN.

Código Final

```
#include <LPC17xx.H>
#include <string.h>
#include <stdio.h>

#define true 1
#define false 0

//Modo programador
int8_t temperaturaProgramada = 12;
int8_t mododeprograma = 0;

//Variables Systick
int8_t hora = 9;
int8_t horaFija = false;
int8_t ticks = 0;

//Representaci�n
int8_t temperaturaMediaMedida = 0;
int8_t errorEnMedia = false;
int8_t temperaturaSensor1 = 0;
int8_t temperaturaSensor2 = 0;
int8_t conmutador = 0;
int8_t unidades = 0;
int8_t decenas = 0;
int8_t valorDelPin1 = 0;
int8_t valorDelPin2 = 0;

//Displays
int8_t SignoMenos = 0x40;
int8_t LetraA = 0x77;
int8_t LetraL = 0x38;

uint16_t Disp[10] = {0x3F,      //0
                                0x06,  //1
                                0x5B,  //2
                                0x4F,  //3
                                0x66,  //4
                                0x6D, //5
                                0x7C, //6
                                0x07,  //7
                                0x7F,  //8
                                0x67}; //9

void inicioHardware(){
```

```
	//Configuracion de los pines P2.10 a P2.12, como entradas de interrupcion
	LPC_PINCON->PINSEL4 |= 1 << (10 * 2); //ISP para el modo de programacion
	LPC_PINCON->PINSEL4 |= 1 << (11 * 2); //KEY1 para el "+"
	LPC_PINCON->PINSEL4 |= 1 << (12 * 2); //KEY2 para el "-"

	//----Pines AO/A1 multiplexaci□n
	LPC_PINCON->PINSEL4 &= ~(0x3 << 3);	// AO
	LPC_PINCON->PINMODE4 &= ~(0x01 << 3); // Sin resistencia interna

	LPC_PINCON->PINSEL4 &= ~(0x3 << 4);	// A1
	LPC_PINCON->PINMODE4 &= ~(0x01 << 4); // Sin resistencia interna

	//----led modo prog
	LPC_PINCON->PINSEL4 &= ~(0x3 << 6);	// Funci□n GPIO
	LPC_PINCON->PINMODE4 &= ~(0x3 << 6); // Resistencia pull-up
	LPC_GPIO2->FIODIR &= ~(0x0 << 6);		// bit P2.06 definido como entrada

	//----led caliente
	LPC_PINCON->PINSEL4 &= ~(0x3 << 1);	// Funci□n GPIO
	LPC_PINCON->PINMODE4 &= ~(0x3 << 1); // Resistencia pull-up
	LPC_GPIO2->FIODIR &= ~(0x0 << 1);		// bit P2.06 definido como entrada

	//----led modo frio
	LPC_PINCON->PINSEL4 &= ~(0x3);// Funci□n GPIO
	LPC_PINCON->PINMODE4 &= ~(0x3); // Resistencia pull-up
	LPC_GPIO2->FIODIR &= ~(0x0);	// bit P2.06 definido como entrada

	//----Entradas y salidas
	LPC_GPIO2->FIODIR = ((0x01) | (0x01 << 1) | (0x01 << 3) | (0x01 << 4) | (0x00 << 6));

	//LPC_GPIO2->FIOCLR = (0x01<<6);
	LPC_GPIO2->FIODIR &= ~(1 << 10); /* ISP definido como entrada */
	LPC_GPIO2->FIODIR &= ~(1 << 11); /* Key_1 definido como entrada */
	LPC_GPIO2->FIODIR &= ~(1 << 12); /* Key_2 definido como entrada */

	//P1.18-P1.24 como salidas para los displays
	LPC_GPIO1->FIODIR = (0x7F) << 18;
}

void configuracionDelSistema(){

	//Asignacion de prioridades
	NVIC_SetPriorityGrouping(3);
```

```
	NVIC_SetPriority(TIMER0_IRQn, 0x0); //Timer0
	NVIC_SetPriority(SysTick_IRQn, 0x1);
	NVIC_SetPriority(EINT0_IRQn, 0x20); //BOTONES
	NVIC_SetPriority(EINT1_IRQn, 0x40);
	NVIC_SetPriority(EINT2_IRQn, 0x40);

	//Habilitacion de las interrupciones
	NVIC_EnableIRQ(EINT0_IRQn);
	NVIC_EnableIRQ(EINT1_IRQn);
	NVIC_EnableIRQ(EINT2_IRQn);
	NVIC_EnableIRQ(TIMER1_IRQn);

	//Configuración EXTMODE/EXTPOLAR
	LPC_SC->EXTMODE |= (1 << 0);
	LPC_SC->EXTPOLAR |= (0 << 0);

	LPC_SC->EXTMODE |= (1 << 1);
	LPC_SC->EXTPOLAR |= (0 << 1);

	LPC_SC->EXTMODE |= (1 << 2);
	LPC_SC->EXTPOLAR &= ~(0 << 2);

	//Configuración SysTick
	SysTick->LOAD = 0x11E1A2FF;                    /* set reload
register */
	SysTick->VAL = 0;                              /* Load the SysTick
Counter Value */
	SysTick->CTRL = 7;                             /* Enable SysTick
IRQ and SysTick Timer */

	//Configuracion timer
	LPC_SC->PCLKSEL1 |= 0 << 1;
	LPC_TIM1->MR0 = 1;
	LPC_TIM1->MR1 = 0x100000;
	LPC_TIM1->MCR = ((0x01) | (0x03 << 3));
	LPC_TIM1->PR = 0;
	LPC_TIM1->TCR = 0x01;

}

//ISR de las interrupciones
void EINT0_IRQHandler(){ //Rutina para el modo del programa. Entro si pulso
el ISP

	LPC_SC->EXTINT |= (1);
	if (mododeprograma == 1){
		mododeprograma = 0;
		LPC_GPIO2->FIOSET |= (0x01 << 6);
```

```
			LPC_GPIO2->FIOCLR = ((0x01 << 1) || (0x01));
		}
		else{
			mododeprograma = 1;
			LPC_GPIO2->FIOCLR |= (0x01 << 6);
			LPC_GPIO2->FIODIR |= (0x01 << 6);
		}
}
//Rutina KEY1
void EINT1_IRQHandler(){ //Rutina para aumentar la temperatura

		LPC_SC->EXTINT |= (1 << 1);
		if (mododeprograma == 1){
			temperaturaProgramada++;

			if ((temperaturaProgramada > 35)){
				temperaturaProgramada = 35;
			}
		}
}

//Rutina KEY2
void EINT2_IRQHandler(){ //Rutina para disminuir la temperatura

		LPC_SC->EXTINT |= (1 << 2);
		if (mododeprograma == 1){
			temperaturaProgramada--;
			if (temperaturaProgramada < (-9)){
				temperaturaProgramada = (-9);
			}
		}
}
void leerSensores(void){

		int8_t media = 0;
		int8_t temperaturaObtenida = 0;

		temperaturaSensor1 = (LPC_GPIO0->FIOPIN &= (0x0F));
		temperaturaSensor2 = (LPC_GPIO0->FIOPIN >> 4) & 0x0F;

		media = (temperaturaSensor1 + temperaturaSensor2) / 2;

		if ((temperaturaSensor1 - temperaturaSensor2) >= 3 ||
(temperaturaSensor2 - temperaturaSensor1) >= 3){
			errorEnMedia = true;
		}
		else{
			errorEnMedia = false;
```

```
		}
		temperaturaObtenida = (50 / 15) * media - 15;
		temperaturaMediaMedida = temperaturaObtenida;
}
void multiplexacion(char tarea){

		//Representar negativos
		if (tarea == 0){
			if (conmutador == 0){
				//Represento decenas
				LPC_GPIO2->FIOCLR = (0X01 << 3);
				LPC_GPIO2->FIOSET = (0X01 << 4);
				LPC_GPIO1->FIOCLR = 0xFFFFFFFF;
				valorDelPin2 = SignoMenos;
				LPC_GPIO1->FIOPIN = (valorDelPin2) << 18;
				conmutador = 1;
			}
			else{
				//Represento unidades
				LPC_GPIO2->FIOSET = (0X01 << 3);
				LPC_GPIO2->FIOCLR = (0X01 << 4);
				LPC_GPIO1->FIOCLR = 0xFFFFFFFF;
				valorDelPin2 = Disp[unidades];
				LPC_GPIO1->FIOPIN = (valorDelPin2) << 18;
				conmutador = 0;
			}
		}
		//Represenar pisitivos
		if (tarea == 1){
			if (conmutador == 0){
				//Represento decenas
				LPC_GPIO2->FIOCLR = (0X01 << 3);
				LPC_GPIO2->FIOSET = (0X01 << 4);
				LPC_GPIO1->FIOCLR = 0xFFFFFFFF;
				valorDelPin2 = Disp[decenas];
				LPC_GPIO1->FIOPIN = (valorDelPin2) << 18;
				conmutador = 1;
			}
			else{
				//Represento unidades
				LPC_GPIO2->FIOSET = (0X01 << 3);
				LPC_GPIO2->FIOCLR = (0X01 << 4);
				LPC_GPIO1->FIOCLR = 0xFFFFFFFF;
				valorDelPin2 = Disp[unidades];
				LPC_GPIO1->FIOPIN = (valorDelPin2) << 18;
				conmutador = 0;
			}
		}
```

```
	//Fuera de rango
	if (tarea == 2){
		if (conmutador == 0){
			//Represento decenas
			LPC_GPIO2->FIOCLR = (0X01 << 3);
			LPC_GPIO2->FIOSET = (0X01 << 4);
			LPC_GPIO1->FIOCLR = 0xFFFFFFFF;
			valorDelPin2 = LetraA;
			LPC_GPIO1->FIOPIN = (valorDelPin2) << 18;
			conmutador = 1;
		}
		else{
			//Represento unidades
			LPC_GPIO2->FIOSET = (0X01 << 3);
			LPC_GPIO2->FIOCLR = (0X01 << 4);
			LPC_GPIO1->FIOCLR = 0xFFFFFFFF;
			valorDelPin2 = LetraL;
			LPC_GPIO1->FIOPIN = (valorDelPin2) << 18;
			conmutador = 0;
		}
	}
	//Representar T fijada
	if (tarea == 3){
		if (conmutador == 0){
			//Represento decenas
			LPC_GPIO2->FIOCLR = (0X01 << 3);
			LPC_GPIO2->FIOSET = (0X01 << 4);
			LPC_GPIO1->FIOCLR = 0xFFFFFFFF;
			valorDelPin2 = Disp[1];
			LPC_GPIO1->FIOPIN = (valorDelPin2) << 18;
			conmutador = 1;
		}
		else{
			//Represento unidades
			LPC_GPIO2->FIOSET = (0X01 << 3);
			LPC_GPIO2->FIOCLR = (0X01 << 4);
			LPC_GPIO1->FIOCLR = 0xFFFFFFFF;
			valorDelPin2 = Disp[2];
			LPC_GPIO1->FIOPIN = (valorDelPin2) << 18;
			conmutador = 0;
		}	}}
void representarTemperatura(int8_t modo){
	if (temperaturaProgramada == temperaturaMediaMedida){
		LPC_GPIO2->FIOSET = ((0x01) | (0x01 << 1));
	}
	else{
		if (temperaturaProgramada > temperaturaMediaMedida){
			//Led de calentado
```

```
                        LPC_GPIO2->FIOSET = (0x01);
                        LPC_GPIO2->FIOCLR = (0x01 << 1);
                }
                else{
                        //Led de enfriado
                        LPC_GPIO2->FIOSET = (0x01 << 1);
                        LPC_GPIO2->FIOCLR = (0x01);
                }
        }

        if (modo == 0){
                if (temperaturaMediaMedida > 0){
                        if ((temperaturaMediaMedida - 10) >= 0){
                                decenas = (temperaturaMediaMedida / 10);
                                unidades = (temperaturaMediaMedida - (decenas
* 10));
                        }
                        else{
                                decenas = 0;
                                unidades = temperaturaMediaMedida;
                        }
                        multiplexacion(1);
                }
                else{
                        multiplexacion(0);
                }
        }

        if (modo == 1){
                if (temperaturaProgramada >= 0){
                        if ((temperaturaProgramada - 10) >= 0){
                                decenas = (temperaturaProgramada / 10);
                                unidades = (temperaturaProgramada - (decenas *
10));
                        }
                        else{
                                decenas = 0;
                                unidades = temperaturaProgramada;
                        }
                        multiplexacion(1);
                }
                else{
                        unidades = (-temperaturaProgramada);
                        multiplexacion(0);
                }
        }
}
//Inicio main
```

```
int main(void){
	inicioHardware();
	configuracionDelSistema();
	while (1){}
}
void SysTick_Handler(void){
	ticks++;
	if (ticks == 60){
		hora++;
		if (hora > 24){
			hora = 0;
		}

		if ((hora >= 22) || (hora <= 8)){
			horaFija = true;
		}
		else{
			horaFija = false;
		}
		ticks = 0;
	}
}
void TIMER1_IRQHandler(void){

	LPC_TIM1->IR = 0x1 << 0;
	if (LPC_TIM1->MR1 == LPC_TIM1->TC){
		leerSensores();
	}
	if (horaFija == false){
		if (mododeprograma == 0){
			//Modo "normal"
			if (errorEnMedia == false){
				representarTemperatura(0);			}
			else{
				LPC_GPIO2->FIOSET = ((0x01) | (0x01 << 1) |
(0x01 << 6));
				multiplexacion(2);
			}		}
		else{
			representarTemperatura(1);
		}
	}
	else{
		LPC_GPIO2->FIOSET = ((0x01) | (0x01 << 1) | (0x01 << 6));
		mododeprograma=0;
		multiplexacion(3);
	}
}
```