SISTEMA DE CONTROL DE TEMPERATURA ON-OFF

Laime Suárez, Gustavo
laimesuarezgustavo12@gmail.com
Carlassara, Fabrizio
fabrizio.carlassara@gmail.com
Quiroga, Laureano Agustín
laureanoquiroga98@gmail.com

ÍNDICE

RESUMEN: *El proyecto consiste en realizar una medición de la temperatura de un ambiente a partir de un sensor de temperatura, y con este valor modificar o mantener la temperatura a partir de valores dados por el usuario.*

PALABRAS CLAVE: *LPC1769 – MCUXpresso – FreeRTOS – Sensor – Temperatura – Control.*

1 INTRODUCCIÓN

El principio de funcionamiento de este controlador de temperatura se basa en la utilización de una celda Peltier controlada por relés. Los relés serán activados por el microcontrolador cada vez que necesitemos bajar/subir la temperatura.

El microcontrolador recibirá constantemente la señal del sensor de temperatura y la transformará a una señal digital, luego, esta señal será transformada a un valor numérico el cuál será comparado con un valor seleccionado en la interfaz por el usuario y de ser necesario activará él relé.

Se mostrará al usuario la temperatura actual a través de un 7 segmentos y en caso de querer seleccionar una nueva temperatura, a través de un pulsador el usuario cambiará de pantalla en el 7 segmentos y podrá setear con otros pulsadores la temperatura deseada.

2 DESCRIPCIÓN DE LOS MODULOS DE HARDWARE PRINCIPALES

2.1 CELDA PELTIER

Una celda Peltier es un dispositivo electrotérmico que permite generar frío o calor a partir de electricidad. Actúa como una bomba de calor de estado sólido, es decir, una de sus caras se calienta mientras que la otra se enfría.

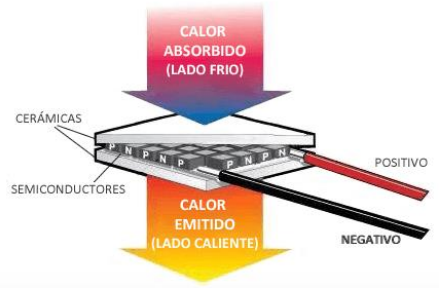


Figura 1. Funcionamiento Celda Peltier.

Para que la celda Peltier disipe el calor de la cara caliente y también para su correcto funcionamiento se le debe colocar un sistema de disipación, en este caso

utilizaremos un ventilador (cooler) colocado directamente a la celda utilizando grasa siliconada para una mejor transmisión del calor.

En este proyecto utilizaremos la capacidad de la celda Peltier de utilizar una misma cara para enfriar/calentar, para poder hacer esto se le debe invertir la polaridad de la misma.

2.2 SENSOR DE TEMPERATURA

Utilizaremos como sensor de temperatura al LM35-DZ el cuál es un dispositivo que por cada grado centígrado en el ambiente que se encuentre entrega 10mV. Es un muy buen sensor para esta aplicación ya que es lineal, relativamente barato y trabaja entre las temperaturas -55ºC y 150ºC, aunque nosotros lo utilizaremos para medir temperaturas entre 0ºC y 100ºC aproximadamente.

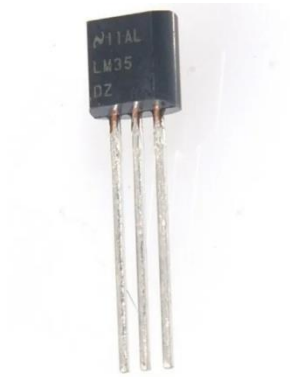


Figura Nº2. Sensor LM35-DZ.

2.3 MÓDULO LECTOR DE TARJETA MICRO SD.

Utilizaremos un módulo lector de tarjetas SD el cuál es compatible con memorias del tipo Micro SD y Micro SDHC (de alta velocidad). El módulo permite alimentación de 5V. Se utilizará la interfaz de comunicación SPI.

(Figura Nº3 – Módulo lector de tarjetas micro SD)

2.4 MÓDULO CONTROLADOR DE 7 SEGMENTOS

Utilizaremos un 7 segmentos de 3 dígitos ánodo común (KYX-3361BGG), para ellos utilizaremos además de las salidas para cada uno de los pines de los dígitos, tres salidas extras para seleccionar que digito se activa.

(Figura Nº4 – 7 segmentos de 3 dígitos)

2.5 PLACA LPCXPRESSO

El proyecto se llevará a cabo utilizando una plataforma de desarrollo con un microcontrolador LPC1769 de 32 bits ARM Cortex M3. Es programable mediante el entorno de desarrollo MCUXpresso. Esta plataforma cuenta con una gran variedad de módulos con distintas funciones, entre ellos nosotros utilizaremos:

El módulo GPIO (General Purpose Input/Output) el cual nos permitirá controlar la tensión salida de los pines GPIO.

El módulo ADC (Analog to Digital converter) el cual convierte un valor de tensión de entrada al microcontrolador en un valor digital con el que podemos trabajar.

El módulo SysTick (SysTick Timer) el cual nos permite generar una interrupción con la cual podremos generar un delay.

El módulo SPI (Serial Peripheral Interface) es un módulo utilizado para comunicación serie con otros dispositivos de forma sincrónica para comunicaciones de corta distancia. En este caso lo utilizaremos para enviar datos a la tarjeta SD.

(Figura Nº5 – Placa de desarrollo LPCXpresso 1769)

3 DIAGRAMAS DE CONEXIÓN

3.1 DIAGRAMA DE CONEXIÓN DE LA CELDA PELTIER

Para poder controlar la celda peltier vamos a hacer uso de relés de 12V los cuales controlaremos con salidas GPIO del microcontrolador, haremos uso de 3 en total, con uno controlaremos la alimentación de los ventiladores del sistema de disipación de la celda y la alimentación de la celda (cuya alimentación, al igual que la bobina de los relés, es de 12V), y con los restantes nos encargaremos de seleccionar la polaridad deseada en la celda, en caso de querer enfriar o calentar el ambiente.

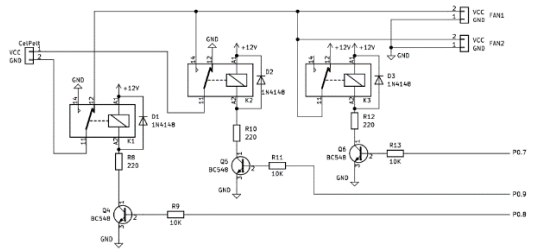


Figura Nº6 – Diagrama de conexión de la celda peltier y relés para la polaridad.

3.2 DIAGRAMA DE CONEXIÓN DEL SENSOR DE TEMPERATURA

Utilizaremos al sensor LM35-DZ con un filtro pasa bajos de segundo orden el cual realizaremos con un amplificador operacional LM358 ya que el mismo nos permite alimentarlo con 3,3V y GND al igual que al sensor de temperatura. Utilizaremos este filtro para evitar variaciones en la entrada del micro, además colocamos una resistencia de pull-down de 10k como es recomendado por el fabricante.

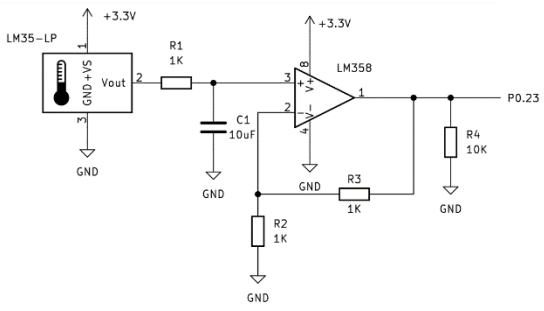


Figura Nº7 – Diagrama de conexión del LM35.

3.3 DIAGRAMA DE CONEXIÓN DE LOS PULSADORES PARA USUARIO

Como se mencionó anteriormente, utilizamos una serie de pulsadores para permitir al usuario seleccionar la temperatura deseada y cambiar entre si desea mostrar la temperatura actual o el modo de seteo. Para esto se utilizaron 3 pulsadores normales y 1 pulsador con retención con un pull-up.

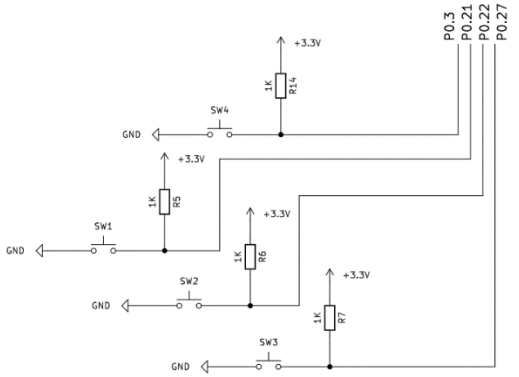


Figura Nº8 – Diagrama de conexión de pulsadores.

3.4 DIAGRAMA DE CONEXIÓN DEL DISPLAY 7 SEGMENTOS

Para la conexión del 7 segmentos se tuvo en cuenta que es del tipo ánodo común, por lo que se necesitaron tres transistores del tipo PNP, uno para cada digito. Se colocaron resistencias en las entradas de cada segmento para limitar la corriente.

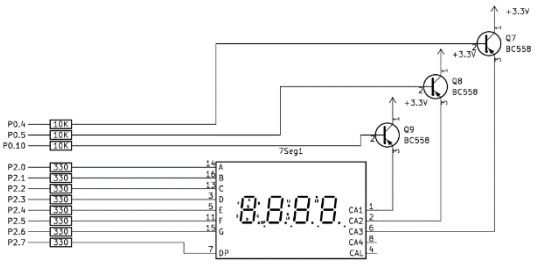
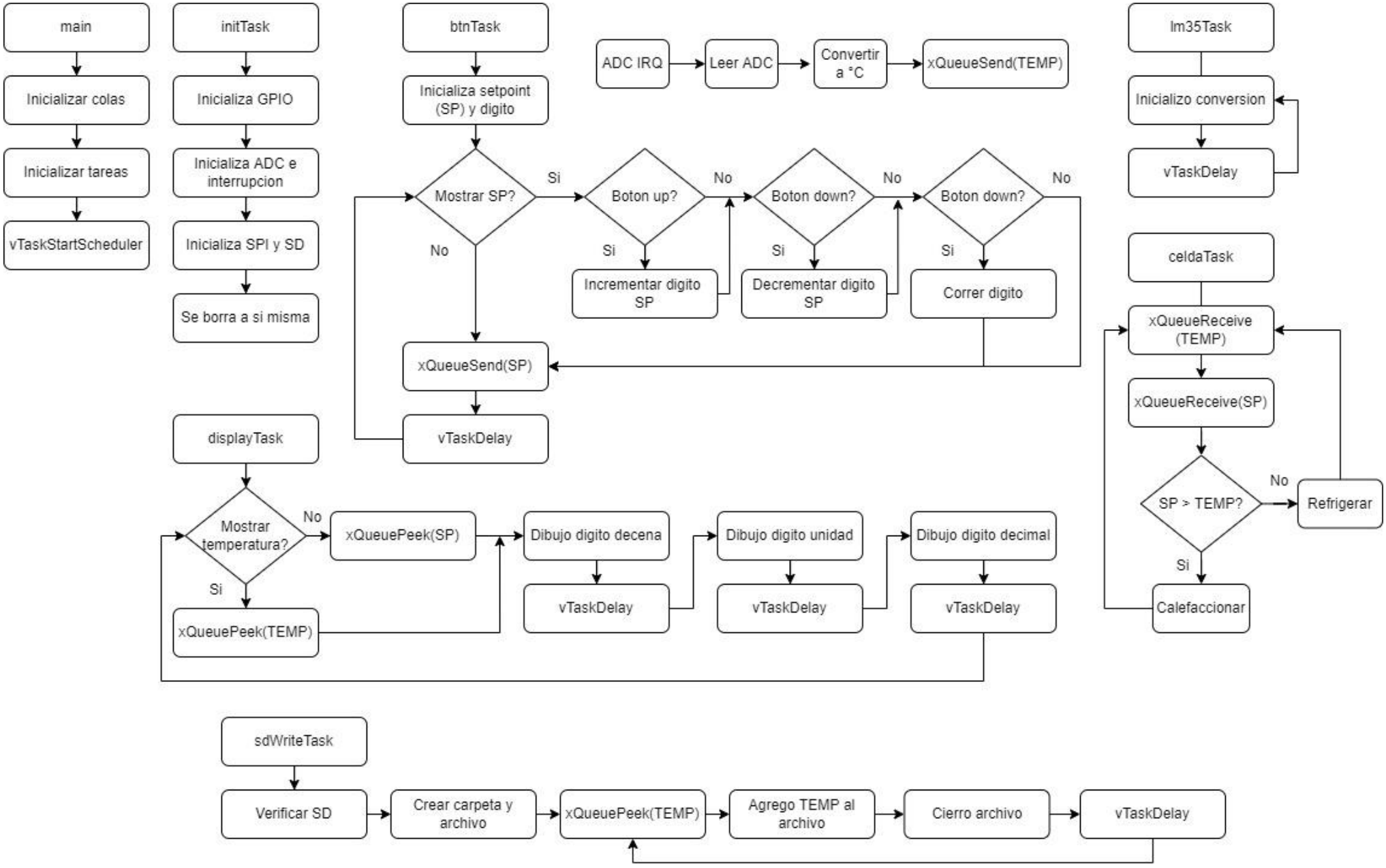


Figura Nº9 – Diagrama de conexión display 7 segmentos.

4 GUIA DE CODIGO


main
Inicializar colas
Inicializar tareas
vTaskStartScheduler
initTask
Inicializa GPIO
Inicializa ADC e interrupcion
Inicializa SPI y SD
Se borra a si misma
btnTask
Inicializa setpoint (SP) y digito
Mostrar SP?
Si
No
Boton up?
Si
No
Incrementar digito SP
Boton down?
Si
No
Decrementar digito SP
Boton down?
Si
No
Correr digito
xQueueSend(SP)
vTaskDelay
ADC IRQ
Leer ADC
Convertir a °C
xQueueSend(TEMP)
lm35Task
Inicializo conversion
vTaskDelay
celdaTask
xQueueReceive (TEMP)
xQueueReceive(SP)
SP > TEMP?
No
Refrigerar
Si
Calefaccionar
displayTask
Mostrar temperatura?
No
Si
xQueuePeek(SP)
xQueuePeek(TEMP)
Dibujo digito decena
vTaskDelay
Dibujo digito unidad
vTaskDelay
Dibujo digito decimal
vTaskDelay
sdWriteTask
Verificar SD
Crear carpeta y archivo
xQueuePeek(TEMP)
Agrego TEMP al archivo
Cierro archivo
vTaskDelay

5 FUNCIONAMIENTO

Se observa arriba un diagrama de flujo con los distintos hilos y tareas del programa. En principio, tenemos dos que no son tareas de FreeRTOS, estos son:

- El programa principal (main) que se encarga de inicializar las colas que se van a usar, (solo dos que llamaremos TEMP y SP de aquí en adelante) y las tareas. Luego, inicializa el scheduler de FreeRTOS y le cede el control al sistema operativo.

- La interrupción del ADC (ADC IRQ). Este es un handler que se ejecuta cada vez que una conversión del ADC se termine. Este handler solo se encarga de tomar ese valor, convertirlo a temperatura y pasarlo a la cola de TEMP.

Sacando lo anteriormente explicado, existen cinco tareas de FreeRTOS que están en ejecución en el programa:

- initTask que solo inicializa los periféricos que la aplicación va a utilizar y luego se elimina a sí misma. Es una tarea de alta prioridad para asegurar que se ejecute primero.

- btnTask es una tarea que lidia con la interacción del usuario, por lo que tiene una prioridad alta. Esta se encarga de tomar los valores de entrada de los pulsadores y variar el valor del SP de acuerdo a si el usuario desea subir o bajar el valor. Aun así, esto solo lo hace si el usuario elige mostrar el SP en el display. De otra forma, el accionar de los botones no producirá ningún cambio en este. Con el botón de enter, el usuario cambia de digito para cambiar, mientras que con los de up y down, sube o baja el SP en el digito elegido.

- displayTask es otra tarea de alta prioridad. Se encarga de mostrar el SP o la TEMP en el display 7 segmentos. Dependiendo de cual de
- los dos el usuario desee mostrar con un botón adicional, toma el dato de la cola correspondiente. Luego, se encarga de tomar cada uno de los dígitos en orden decreciente de forma individual, mostrarlo en el siete segmentos correspondiente, y luego hacer una demora corta para dar la ilusión óptica de que están todos prendidos.

- sdWriteTask es una tarea de baja prioridad. Esta se encarga de inicializar la SD, crear un archivo dentro de una carpeta de la misma y luego, tomar de la cola de TEMP el valor para escribirlo en la SD de forma periódica.

- celdaTask es la tarea que se encarga de controlar la celda peltier. Es una tarea de baja prioridad y se encarga de tomar los valores de SP y TEMP, compararlos, y luego calentar o refrigerar dependiendo de la diferencia entre estas temperaturas. Esta es la única tarea que hace un xQueueReceive en lugar de un xQueuePeek, ya que los mismos datos son compartidos entre varias tareas. Se hizo de esta una tarea de baja prioridad para asegurar de que sea la que se encargue de remover los valores leídos de las colas correspondientes sin interferir con las otras tareas.

6 BIBLIOGRAFÍA

[1] Manual de usuario LPC1769: UM10360 – LPC176x/5x – Rev 3.1

[2] Hoja de datos celda peltier: https://pdf1.alldatasheet.com/datasheet-pdf/view/313841/HB/TEC1-12706.html

[3] Hoja de datos LM35-DZ: https://pdf1.alldatasheet.com/datasheet-pdf/view/8866/NSC/LM35.html

[4] Hoja de datos BC548: https://pdf1.alldatasheet.com/datasheet-pdf/view/11552/ONSEMI/BC548.html

[5] Hoja de datos BC558: https://pdf1.alldatasheet.com/datasheet-pdf/view/42388/SEMTECH/BC558.html

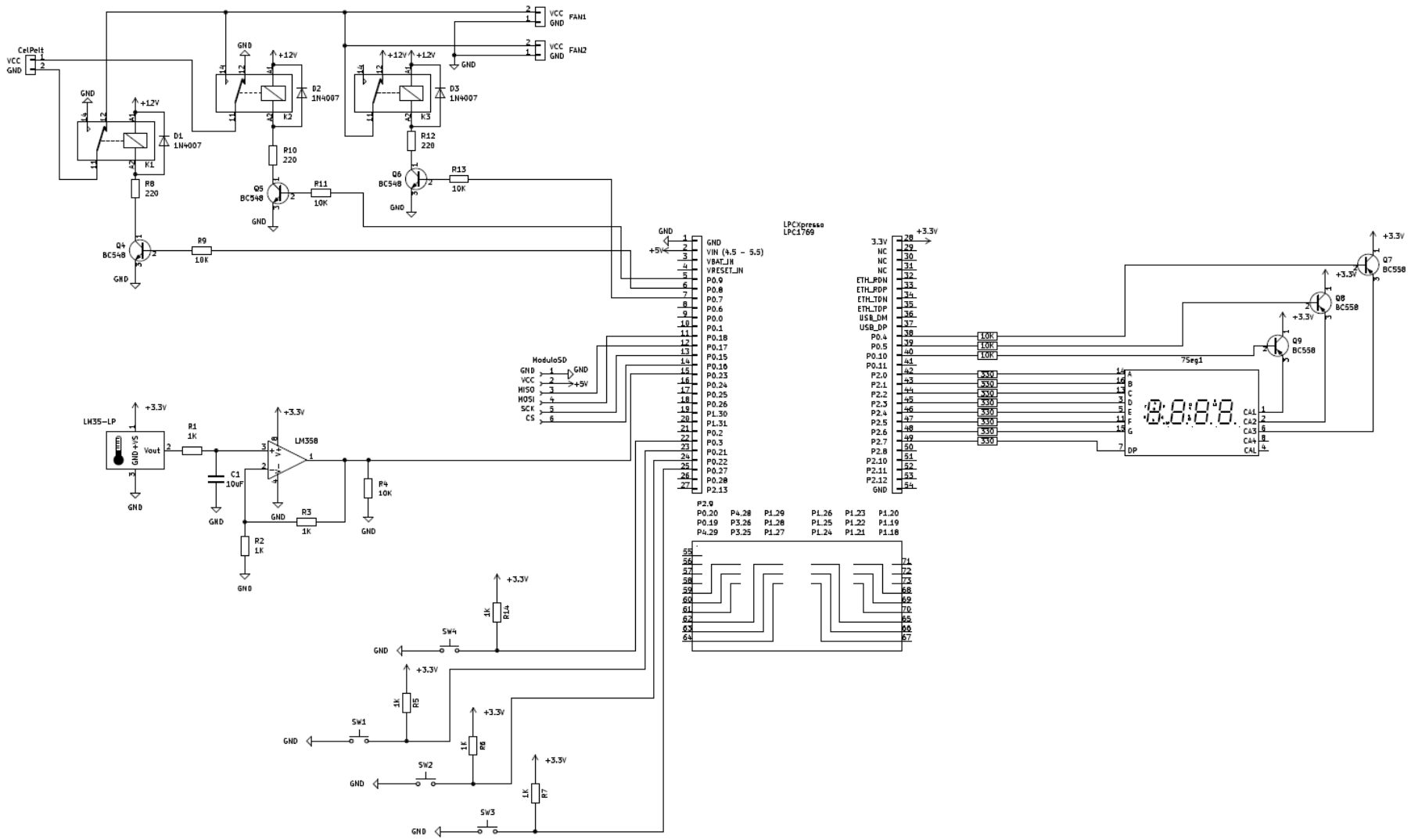
[6] Hoja de datos LM358: https://pdf1.alldatasheet.com/datasheet-pdf/view/3067/MOTOROLA/LM358.html

[7] Hoja de datos Relés: https://datasheet4u.com/datasheet-pdf/TONGLING/JQC-3FF-S-Z/pdf.php?id=1480970

[8] Hoja de datos display 7 segmentos: http://www.xlitx.com/Products/7-segment-led-dot-matrix/3361bgg.html

[9] Repositorio en GitHub: https://github.com/carlassaraf/td3-integrador-1

ANEXO I: ESQUEMATICO



ANEXO II: CODIGO

MAIN.C

```
#include "proj_tasks.h"

/*
 *    @brief Programa principal
 */
int main(void) {
      /* Actualizo el clock del sistema */
    SystemCoreClockUpdate();
    /* Creo la cola para la temperatura medida */
    queueTEMP = xQueueCreate(1, sizeof(float));
    /* Creo la cola para los datos del setpoint */
    queueSP = xQueueCreate(1, sizeof(float));
    /* Registro las colas para debuggear */
    vQueueAddToRegistry( queueTEMP, (signed char *) "Cola de temperatura");
    vQueueAddToRegistry( queueSP, (signed char *) "Cola de SP");

    /* Creacion de tareas */
    xTaskCreate(
            initTask,                           /* Callback para la tarea */
            (const signed char *) "Init Task",  /* Nombre de la tarea para debugging */
            configMINIMAL_STACK_SIZE,           /* Minimo stack para la tarea */
            NULL,                               /* Sin parametros */
            tskINIT_PRIORITY,                   /* Prioridad */
            NULL                                /* Sin handler */
    );

      xTaskCreate(
            btnTask,                            /* Callback para la tarea */
            (const signed char *) "Botones",    /* Nombre de la tarea para debugging */
      configMINIMAL_STACK_SIZE,                 /* Minimo stack para la tarea */
            NULL,                               /* Sin parametros */
            tskBTN_PRIORITY,                    /* Prioridad */
            NULL                                /* Sin handler */
      );

      xTaskCreate(
```

```
        displayTask,                            /* Callback para la tarea */
        (const signed char *) "7 Segment Task",/* Nombre de la tarea para debugging */
        configMINIMAL_STACK_SIZE,               /* Minimo stack para la tarea */
        NULL,                                   /* Sin parametros */
        tsk7SEG_PRIORITY,                       /* Prioridad */
        NULL                                    /* Sin handler */
    );

    xTaskCreate(
        lm35Task,                               /* Callback para la tarea */
        (const signed char *) "LM35 Task",      /* Nombre de la tarea para debugging */
        configMINIMAL_STACK_SIZE,               /* Minimo stack para la tarea */
        NULL,                                   /* Sin parametros */
        tskLM35_PRIORITY,                       /* Prioridad */
        NULL                                    /* Sin handler */
    );

    xTaskCreate(
        sdWriteTask,                            /* Callback para la tarea */
        (const signed char *) "Escritura SD",   /* Nombre de la tarea para debugging */
        configSD_TASK_SIZE,                     /* Minimo stack para la tarea */
        NULL,                                   /* Sin parametros */
        tskSDWRITE_PRIOTITY,                    /* Prioridad */
        NULL                                    /* Sin handler */
    );

    xTaskCreate(
        celdaTask,                              /* Callback para la tarea */
        (const signed char *) "Celda Peltier", /* Nombre de la tarea para debugging */
        configMINIMAL_STACK_SIZE,               /* Minimo stack para la tarea */
        NULL,                                   /* Sin parametros */
        tskCELDA_PRIOTITY,                      /* Prioridad */
        NULL                                    /* Sin handler */
    );

    /* Inicio el scheduler */
    vTaskStartScheduler();

while(1);

return 0;
```

```
}
```

PROJ_TASKS.C

```
#include "proj_tasks.h"

/* Cola para el ADC */
xQueueHandle queueTEMP, queueSP;

/*
 *    @brief Tarea de inicializacion de perifericos
 */
void initTask(void *params) {
    /* Inicializo SPI */
    SPI_Inicializar();
    /* Inicializo ADC */
    adc_init();
    /* Inicializo los 7 Segmentos */
    gpio_7segments_init();
    /* Inicializo botones */
    gpio_btn_init();
    /* Inicialización de celda */
    gpio_celda_init();
    /* Elimino tarea */
    vTaskDelete(NULL);
}

/*
 *    @brief Tarea que lee el estado de los botones y actualiza el setpoint
 */
void btnTask(void *params) {
      /* Delay para captura de botones */
      const uint16_t DELAY_MS = 5;
      /* Setpoint inicial */
      float setpoint = 25.0;
      /* Variable para verificar que digito modificar */
      uint8_t digits = 0x04;
      /* Valor de incremento/decremento */
      int8_t inc = 0;

      while(1) {
```

```
/* Solo valido si se elige mostrar el setpoint */
if(SETPOINT_SELECTED) {
	/* Veo si se sube o baja el valor del setpoint */
	/* Si el boton up se apreto, se incrementa */
	if(!gpio_get_btn_up()) {
		while(!gpio_get_btn_up());
		inc = 1;
	}
	/* Si el boton down se apreto, se decrementa */
	else if(!gpio_get_btn_down()) {
		while(!gpio_get_btn_down());
		inc = -1;
	}
	/* Si el boton enter se apreto, no cambia el setpoint, pero cambio de digito */
	else if (!gpio_get_btn_enter()) {
		while(!gpio_get_btn_enter());
		/* No hay incremento */
		inc = 0;
		/* Voy al digito de la derecha*/
		digits >>= 1;
		/* Vuelvo al primer digito */
		if(digits == 0) { digits = 0x04; }
	}
	else{
		inc = 0;
	}
	/* Reviso el digito */
	switch (digits) {
		/* Si es el primer digito, cambio la decena */
		case 0b100:
			setpoint += inc * 10;
			break;
		/* Si es el segundo digito, cambio la unidad */
		case 0b010:
			setpoint += inc;
			break;
		/* Si es el tercer digito, cambio el decimal */
		case 0b001:
			setpoint += inc / 10.0;
			break;
	}
```

```
                        /* Si el setpoint es negativo, lo vuelvo a cero */
                        if(setpoint < 0) { setpoint = 0; }
                        /* Si el setpoint se excedio de 100, vuelvo a 99.9 */
                        if(setpoint >= 100.0) { setpoint = 99.9; }
                }
                /* Mando el setpoint a la cola */
                xQueueSendToBack(queueSP, &setpoint, portMAX_DELAY);
                /* Bloqueo la tarea por un rato */
                vTaskDelay(DELAY_MS / portTICK_RATE_MS);
        }
}

/*
 *     @brief Tarea que inicia las lecturas del LM35
 */
void lm35Task(void *params) {
        /* Delay entre conversiones */
        const uint16_t DELAY_MS = 5;

        while(1) {
                /* Inicio la conversion */
                adc_start();
                /* Bloqueo por un tiempo hasta empezar otra conversion*/
                vTaskDelay(DELAY_MS / portTICK_RATE_MS);
        }
}

/*
 *     @brief Muestreo del valor de temperatura/setpoint en el 7 segmentos
 */
void displayTask(void *params) {
        /* Delay entre cambio de digitos */
        const uint8_t DELAY_MS = 5;
        /* Variable para guardar la temperatura/setpoint */
        float temp;
        /* Variable para almacenar cada digito separado */
        uint8_t digit1, digit2, digit3;

        while(1) {
                /* Si se eligio mostrar la temperatura, bloqueo la tarea hasta que la interrupcion la cargue */
                if(TEMPERATURE_SELECTED) { xQueuePeek(queueTEMP, &temp, portMAX_DELAY); }
```

```
            /* Si se eligio mostrar el setpoint, bloqueo la tarea hasta que este en la cola */
            else { xQueuePeek(queueSP, &temp, portMAX_DELAY); }
            /* Obtengo el primer digito */
            digit1 = (uint8_t)(temp / 10);
            /* Escribo el numero en el primer digito, sin el punto */
            gpio_7segments_set(DIGIT_1, digit1, false);
            /* Bloqueo la tarea por unos momentos */
            vTaskDelay(DELAY_MS / portTICK_RATE_MS);
            /* Obtengo el segundo digito */
            digit2 = (uint8_t)temp % 10;
            /* Escribo el numero en el segundo digito, con el punto */
            gpio_7segments_set(DIGIT_2, digit2, true);
            /* Bloqueo la tarea por unos momentos */
            vTaskDelay(DELAY_MS / portTICK_RATE_MS);
            /* Obtengo los decimales */
            digit3 = (uint8_t)((temp - 10 * digit1 - digit2) * 10);
            /* Escribo el numero en el tercer digito, sin el punto */
            gpio_7segments_set(DIGIT_3, digit3, false);
            /* Bloqueo la tarea por unos momentos */
            vTaskDelay(DELAY_MS / portTICK_RATE_MS);
        }
}

/*
 *    @brief       Tarea que escribe el valor de temperatura en la SD
 */
void sdWriteTask(void *params){

       sd_variables_t carpeta;
       uint32_t i = 0;

    /* Inicializamos y verificamos que se inicialice correctamente el SD*/
       if (SD_Init (&carpeta.tipo) == SD_FALSE)
    {
       // No se pudo iniciar la tarjeta. por favor, revisa la tarjeta
         while (1)
              i++;
    }

    /* Verificamos que se pueda leer */
    if (SD_ReadConfiguration (&carpeta.CardConfig) == SD_FALSE)
```

```
    {
      // No se pudo leer
            while (1)
                  i++;
    }
    // Se inicializó con exito.

      /* Abro una carpeta que exista o creo una nueva */
    carpeta.fr = f_mount(&carpeta.fs, "0:", 0); // Registro un objeto del sistema de archivos para una unidad lógica "0:",
es decir es el Driver.
    carpeta.fr = f_open(&carpeta.fil, "td3.csv", FA_CREATE_ALWAYS); // Si no existe crea.
    if (carpeta.fr == FR_OK) {
      //carpeta.fr = f_write (&carpeta.fil, carpeta.bufferWrite, sizeof(carpeta.bufferWrite), &carpeta.BytesWritten); //
Escribe
      // Buscar hasta el final del archivo para agregar datos
      //carpeta.fr = f_lseek(&carpeta.fil, f_size(&carpeta.fil));

       // En caso de que no poder escribirse la carpeta se debe cerrarla.
       if (carpeta.fr != FR_OK)
           f_close(&carpeta.fil);
    }

    // Verifico que se haya creado correctamente.
      if (carpeta.fr != FR_OK)
      {
            // Si no crea nada, viene acá.
            while (1);
      }
      // Se creó la carpeta TecnicasDigitalesIII, debemos cerrar la carpeta.
      f_close(&carpeta.fil);

      float temp;
      char vector[40];
      /* vacio el vector*/
      for (uint8_t i = 0; i < 40; i++)
            vector[i] ='\0';
      while(1)
      {
            xQueuePeek(queueTEMP, &temp, portMAX_DELAY);
            carpeta.fr = f_mount(&carpeta.fs, "0:", 0); // Registro un objeto del sistema de archivos para una unidad
lógica "0:", es decir es el Driver.
```

```
		carpeta.fr = f_open(&carpeta.fil, "td3.csv", FA_READ | FA_WRITE); // Si no existe crea.
		if (carpeta.fr == FR_OK){
			imprimir(vector, &temp);
			carpeta.tamanioArchivo = f_size(&carpeta.fil);
			carpeta.bufferRead = malloc(carpeta.tamanioArchivo);
			while(carpeta.tamanioArchivo > 0)
			{
				carpeta.fr = f_read (&carpeta.fil, carpeta.bufferRead, (UINT)sizeof(carpeta.bufferWrite),
&carpeta.ByteRead);
				carpeta.tamanioArchivo -= sizeof(carpeta.bufferWrite);

			}
			carpeta.fr = f_write (&carpeta.fil, vector, sizeof(carpeta.bufferWrite), &carpeta.BytesWritten); //
Escribe
			// Buscar hasta el final del archivo para agregar datos
			carpeta.fr = f_lseek(&carpeta.fil, f_size(&carpeta.fil));
			// En caso de que no poder escribirse la carpeta se debe cerrarla.
			for (uint8_t i = 0; i < 40; i++)
				vector[i] ='\0';
			if (carpeta.fr != FR_OK)
				f_close(&carpeta.fil);
			// Se creó la carpeta TecnicasDigitalesIII, debemos cerrar la carpeta.
		}
		f_close(&carpeta.fil);
		vTaskDelay(1000 / portTICK_RATE_MS);
	}

}

/*
 *	@brief Tarea que se encarga de manejar la celda peltier
 */
void celdaTask(void *params){
	/* Variables para guardar los valores de las colas */
	float temp, sp;
	/* Versiones enteras para la temperatura y setpoint */
	uint8_t iTemp, iSp;

	while(1) {
		/* Se bbloquea hasta recibir la temperatura */
		xQueueReceive(queueTEMP, &temp, portMAX_DELAY);
```

```
			/* Se bloquea hasta recibir el setpoint */
			xQueueReceive(queueSP, &sp, portMAX_DELAY);
			/* Obtengo la temperatura y setpoint como enteros */
			iTemp = temp;
			iSp = sp;
			/* Enciendo el fan */
			gpio_fan_on(true);
			/* Si la temperatura es menor al setpoint, calefacciono */
			if(iTemp < iSp) {
				/* Enciendo la calefaccion */
				gpio_heat_on();
			}
			/* Si la temperatura es mayor al setpoint, refrigero */
			else if(iTemp > iSp) {
				/* Enciendo la refrigeracion */
				gpio_cold_on();
			}
			/* Si son iguales descontando los decimales, apago la celda */
			else {
				/* Apago la celda */
				gpio_celda_off();
			}
		}
}

/*
 *	@brief Handler para las interrupciones del ADC
 */
void ADC_IRQHandler(void) {
		/* Control para cambiar de tarea al salir de la interrupcion */
		static portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;
		/* Variable auxiliar para el valor  del ADC */
		uint16_t adc;
		/* Variable para la temperatura */
		float temp;
		/* Obtengo el valor del ADC */
		adc_read(adc);
		/* Convierto el valor del ADC en temperatura */
		temp = CONV_FACTOR * (float)adc * 50;
		/* Ingreso el valor a la cola */
		xQueueSendToBackFromISR(queueTEMP, &temp, &xHigherPriorityTaskWoken);
```

```
	/* Solicitar un cambio de contexto si fuese necesario */
	portEND_SWITCHING_ISR(xHigherPriorityTaskWoken);
}
```