Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Estudio de los efectos de la temperatura en el consumo energético de microcontroladores

TITULACIÓN: Grado en Ingeniería de Sistemas Aeroespaciales

AUTOR: Andrés Fabricio Pisco Ortiz

DIRECTOR: Ferran Reverter

FECHA: 23 de noviembre del 2020

Título: Estudio de los efectos de la temperatura en el consumo energético de microcontroladores

Autor: Andrés Fabricio Pisco Ortiz

Director: Ferran Reverter

Fecha: 23 de noviembre del 2020

Resumen

Los microcontroladores son circuitos integrados programables que tienen como función automatizar procesos y procesar información dentro de una aplicación determinada. Debido a que son programables se pueden encontrar en diversos dispositivos electrónicos desde los más comunes en la vida diaria como ordenadores, tablets, smartphones, etc. a otros con funciones más complejas que forman parte de edificios y ciudades inteligentes, procesos industriales o la industria aeronáutica, por ejemplo.

Los microcontroladores están formados por varios bloques que cumplen una tarea específica. Los bloques esenciales son la unidad central de proceso (CPU), la cual se encarga de procesar y ejecutar las instrucciones guardadas en la memoria. La memoria donde se almacenan las instrucciones que ejecuta la CPU y elementos volátiles como variables o datos. Y los periféricos que son recursos hardware integrados dentro del mismo chip que pueden funcionar simultáneamente a la CPU y además permiten la interacción con el exterior.

Este proyecto está destinado al estudio del efecto de la temperatura en el consumo energético de un microcontrolador comercial y uno de sus periféricos, el timer. Para ello se dispondrá de una cámara climática donde se introducirá el dispositivo y se realizarán varios experimentos para medir su consumo. Finalmente, se compararán los resultados y se extraerán unas conclusiones sobre los efectos que tiene la temperatura en el consumo.

Title: Study of the effects of temperature on the energy consumption of microcontrollers

Author: Andrés Fabricio Pisco Ortiz

Director: Ferran Reverter

Date: November 23rd 2020

Overview

Microcontrollers are programmable integrated circuits whose function is to automate processes and process information within a given application. Because they are programmable, they can be found in various electronic devices from the most common ones in daily life such as computers, tablets, smartphones, etc. to others with more complex functions that are part of smart buildings and cities, industrial processes or the aeronautical industry, for example.

Microcontrollers are made up of several blocks that perform a specific task. The essential blocks are the central processing unit (CPU), which is responsible for processing and executing the instructions stored in memory. The memory where the instructions executed by the CPU and volatile elements such as variables or data are stored. And the peripherals that are hardware resources integrated within the same chip that can operate simultaneously with the CPU and also allow interaction with the outside.

This project is aimed at studying the effect of temperature on the energy consumption of a commercial microcontroller and one of its peripherals, the timer. To this end, a thermal chamber will be available where the device will be introduced and various experiments will be carried out to measure its consumption. Finally, the results will be compared and conclusions will be drawn on the effects of temperature on the power consumption.

ÍNDICE

INTRODUCCIÓN 1

Revolución tecnológica 1

Sensores 1

Objetivos 3
Objetivo principal 3
Objetivos secundarios 3

Estructura de la memoria 4

CAPÍTULO 1. MICROCONTROLADOR Y TIMER 5

1.1 ¿Qué es un microcontrolador? 5
1.1.1 CPU 5
1.1.2 Memoria 6
1.1.3 Periféricos 7

1.2 Timer 7

CAPÍTULO 2. MICROCONTROLADOR MSP430FR5969 9

2.1 Características 9
2.1.1 MSP-EXP430FR5969 10
2.1.2 MSP430FR5969 11

2.2 Timers 12

2.3 Modos de bajo consumo 13

CAPÍTULO 3. DESARROLLO DEL PROGRAMA 14

3.1 Escenarios 14

3.2 Entorno de programación 14

3.3 Descripción general 15

3.4 Código 16

CAPÍTULO 4. INSTRUMENTACIÓN Y MONTAJE 22

4.1 Instrumentación 22

4.2 Montaje 24

CAPÍTULO 5. RESULTADOS 28

5.1 Timer parado.....28

5.2 Timer activo 30

CAPÍTULO 6. CONCLUSIONES 38

BIBLIOGRAFÍA 39

ANEXOS..... 40

Anexo I. Registros entradas/salidas.....41

Anexo II. Registros del Sistema de reloj 42

Anexo III. Registros del timer.....45

Anexo IV. Programa completo 46

INTRODUCCIÓN

Revolución tecnológica

Desde la prehistoria el ser humano ha creado diversas máquinas y herramientas con el fin de superar sus limitaciones físicas. La lista de estas creaciones va desde las primeras armas prehistóricas a los dispositivos electrónicos más modernos que nos encontramos hoy en día.

Para que estas invenciones sean útiles y productivas necesitan de algo o alguien que las gobierne. Al principio y durante mucho tiempo este control fue realizado por el hombre, hasta el día de hoy donde ya podemos encontrar dispositivos que son capaces de gobernar y controlar a otra máquina con el fin de liberar al hombre de dichas tareas, realizar procesos automatizados y reducir costes y malos usos.

Estos dispositivos capaces de gobernar son los que conocemos como microcontroladores, que son pequeños circuitos integrados, en los cuales todas sus unidades funcionales se integran en un mismo microchip y son capaces de ejecutar instrucciones programables.

Sin embargo, los microcontroladores por si solos no son capaces de captar el gran número de señales analógicas por las que está caracterizado el mundo. El correcto tratamiento de estas señales tiene una gran importancia para el control de procesos y dispositivos. Es aquí donde entran en escena las redes de sensores autónomos, capaces de medir magnitudes y transmitirlas sin la intervención del hombre.

Sensores

Los sensores son el primer bloque en la cadena de medición y se encargan de proporcionar una señal en el dominio eléctrico. Los sensores se pueden clasificar de varias maneras como, por ejemplo, según sus requerimientos de fuente de energía (activos o pasivos), según la naturaleza de la señal a medir (mecánicos, térmicos, químicos, etc.), según la naturaleza de la señal de salida (analógicos, cuasi digitales, digitales), etc. Para este proyecto se ha decidido clasificarlos en función de la naturaleza de la señal de salida.

- Sensores analógicos. Proporcionan una señal analógica cuya amplitud es proporcional a la magnitud que se está midiendo (ver figura I.1a).

- Sensores cuasi digitales. Proporcionan una señal de salida analógica o digital con un parámetro relacionado con el tiempo (como el período, la frecuencia, el ciclo de trabajo o el intervalo de tiempo) que es modulado por la magnitud de entrada (ver figura I.1b). Aunque la información

relacionada con el tiempo es analógica, puede ser medida con un sistema digital que contenga un timer/contador, de ahí el nombre de cuasi digital.

- Sensores digitales. Proporcionan una señal de salida digital con una secuencia de niveles lógicos modulados por la señal de entrada (ver figura I.1c).

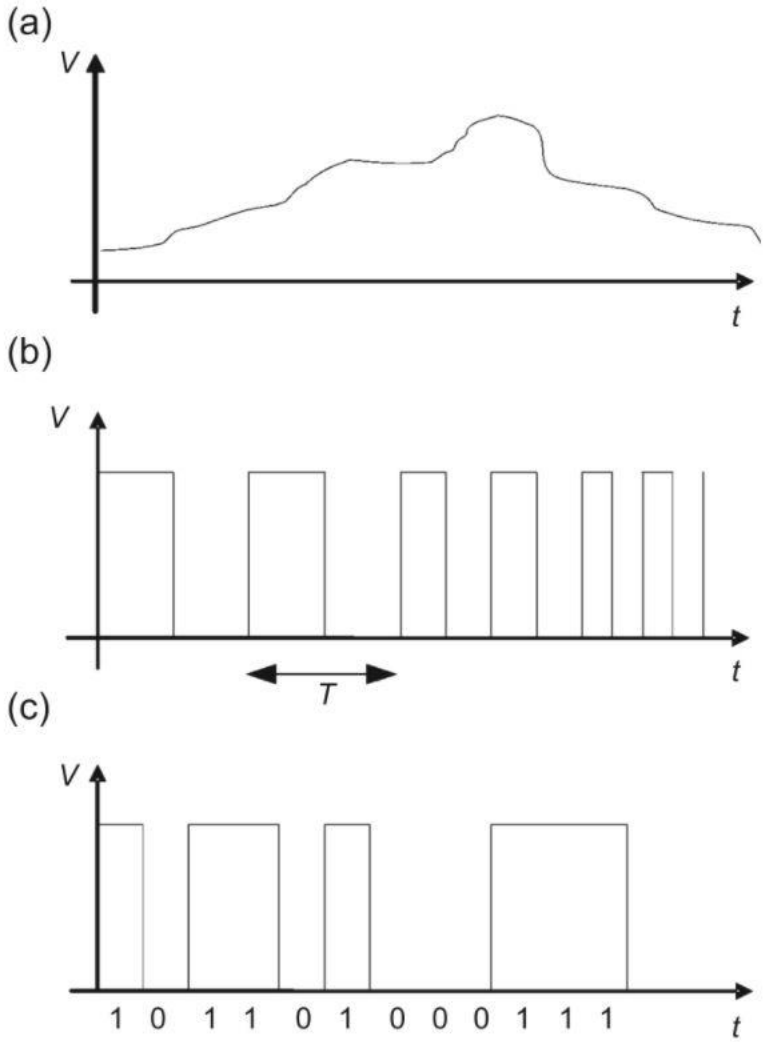

Fig. I.1 Señal analógica (a), señal cuasi digital (b) y señal digital (c)

Este proyecto se centrará en los sensores cuasi digitales, por ello también hay que destacar que los sensores analógicos con una correcta estimulación pueden llegar a convertirse en sensores cuasi digitales. Estos sensores son de gran interés para el proyecto ya el objetivo principal es el estudio de los efectos de la temperatura en el consumo energético de los MCUs y sus timers.

Algunos ejemplos de sensores cuasi digitales se pueden ver en la tabla I.1:

Tabla I.1 Sensores cuasi digitales comerciales

Nombre del sensor	Magnitud	Señal de salida
MLX90316	Posición angular	Señal analógica o digital modulada en frecuencia.
TMP03/TMP04	Temperatura	Señal digital modulada en frecuencia proporcional a la temperatura.
TSCDNNN150PGUCV	Presión	Señal proporcional a la diferencia entre las presiones aplicadas en cada puerto.
MAX6666/MAX6667	Temperatura	Señal modulada en frecuencia.

Objetivos

Este trabajo consta de varios objetivos en diferente grado de importancia que nos permitirán realizarlo con éxito.

Objetivo principal

Debido a la poca información facilitada por los fabricantes de microcontroladores en relación al consumo energético en función de la temperatura, el objetivo principal consiste en analizar este tema de forma experimental utilizando un microcontrolador comercial y una cámara climática. La finalidad es obtener los rangos de temperatura que nos permitan ver que situaciones nos son más favorables y así reducir el consumo energético para poder alargar el tiempo de vida de las baterías que alimentan los sensores y microcontroladores.

Para conseguir este objetivo se tiene que realizar un programa que permita analizar el consumo del MCU y del timer a diferentes frecuencias y modos de trabajo además de poder hacerlo de forma automatizada, lo que permitirá ahorrar tiempo a la hora de realizar las mediciones, ya que, no se necesitará estar abriendo y cerrando la cámara climática cada vez que se quiera analizar un escenario distinto.

Objetivos secundarios

1. Realizar un estudio de las características del microcontrolador elegido y de sus timers.

2. Entender el entorno de programación del microcontrolador.

3. Aprender a usar una cámara climática para el testeo térmico de dispositivos.

Estructura de la memoria

Una vez acabada esta breve introducción, planteado la problemática y los objetivos a conseguir, se explicará la estructura y contenido de los capítulos del proyecto.

- Capítulo 1. En este capítulo se explicará que es un microcontrolador y sus componentes más importantes. También se explicará que es un timer y sus características principales.

- Capítulo 2. En este capítulo se explicarán las características más destacadas tanto del microcontrolador como de la placa elegidos para el proyecto.

- Capítulo 3. En este capítulo se explicará brevemente el entorno de programación del microcontrolador y las diferentes partes que tiene el programa junto con los registros que han sido necesarios para su realización.

- Capítulo 4. En este capítulo se explicará cómo se han montado todos los dispositivos necesarios para la realización del proyecto, así como también la instrumentación necesaria para la realización del experimento y las mediciones de corriente.

- Capítulo 5. En este capítulo se expondrán y analizarán los resultados obtenidos en el laboratorio.

- Capítulo 6. En este capítulo finalmente se expondrán las conclusiones obtenidas al finalizar el proyecto.

CAPÍTULO 1. MICROCONTROLADOR Y TIMER

1.1 ¿Qué es un microcontrolador?

Un microcontrolador (MCU) es un circuito integrado digital basado en un procesador programable de bajo coste que en función de su diseño se usará en sistemas de control y/o medición de sensores. Existen otros sistemas digitales capaces de hacer estas tareas, pero debido a su bajo coste y consumo los MCUs parten con ventaja respecto a otros dispositivos.

Los MCUs tienen tres bloques principales, los cuales son: la unidad de procesamiento central (CPU), la memoria y los periféricos.

1.1.1 CPU

Es el centro de operaciones del microcontrolador y se encarga de procesar y ejecutar de forma secuencial las instrucciones almacenadas en su memoria para una tarea específica. Entre sus componentes se pueden encontrar (ver figura 1.1):

- Unidad lógica aritmética (ALU). Es la encargada de realizar todas las operaciones lógicas y aritméticas del sistema. Su capacidad de cálculo dependerá del número de bits que contiene el microcontrolador (8, 16 o 32 bits normalmente).

- Unidad de control (CU). Es la encargada de gobernar el funcionamiento de la CPU realizando continuamente ciclos de búsqueda, decodificación y ejecución. A este ciclo se le llama ciclo de instrucción.

- Un conjunto de registros. Proporcionan un almacenamiento temporal de datos, direcciones de memoria e información de control de forma que se pueda acceder rápidamente a ellos. Su contenido es volátil.

- Lógica de interfaz bus (BIL). Se refiere a las estructuras de la CPU que coordinan la interacción entre los buses internos y los buses del sistema.

Por otro lado, también hay que destacar que algunas CPUs tienen uno o varios modos de bajo consumo (modo sleep) de manera que suspende sus tareas y se mantiene en un estado “dormido” consiguiendo así que el consumo disminuya. Estos modos de bajo consumo son sumamente importantes para la implementación de nodos sensores para poder alargar el tiempo de vida de las baterías que los alimentan.

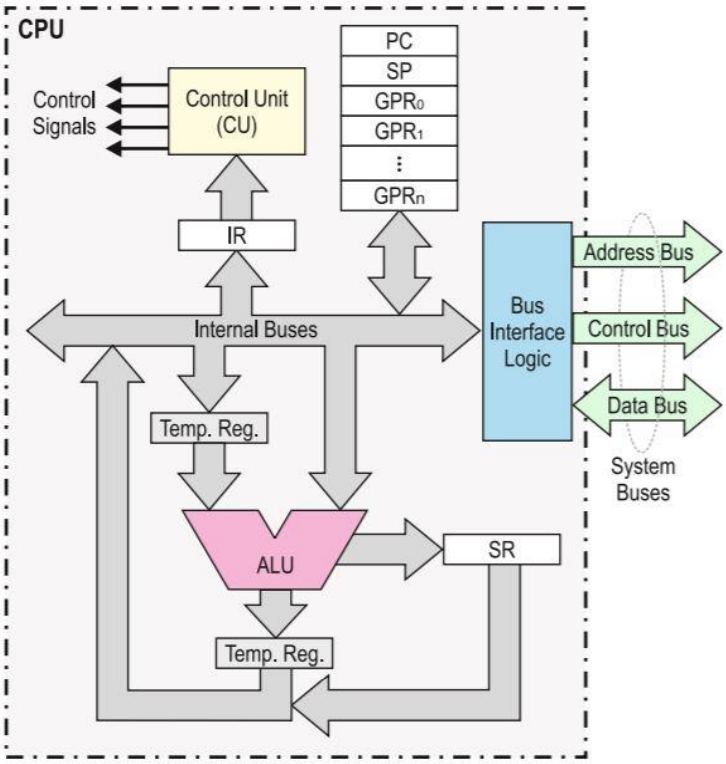

Fig. 1.1 Diagrama de bloques de la CPU

1.1.2 Memoria

La memoria está integrada en el microcontrolador y se encarga de las funciones fundamentales de almacenamiento de programas y datos.

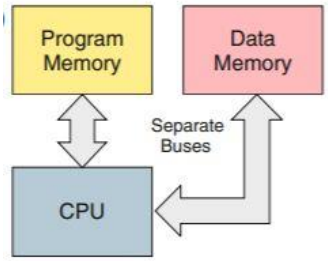

Fig. 1.2 Diagrama de bloques de la memoria

Actualmente la tendencia en los MCUs es usar la arquitectura Harvard (ver figura 1.2), que consiste en tener dos tipos de memoria que puedan trabajar simultáneamente y en consecuencia obtener mayor velocidad en la ejecución de programas. Estas memorias son:

- La memoria de programa, como se deduce de su nombre, se refiere a la porción de memoria que almacena los programas del sistema en una forma directamente accesible por la CPU. Suelen ser del tipo OTP

(programables una vez) o tipo flash. Este tipo de memorias son no volátiles, lo que quiere decir que no necesitan de energía para mantener guardada la información en ellas.

- La memoria de datos, es la que se utiliza para almacenar variables y datos que se esperan que cambien durante la ejecución del programa. Por lo tanto, este tipo de memoria debería permitir modificar fácilmente su contenido. Esta memoria suele ser de acceso aleatorio (RAM), que es volátil.

1.1.3 Periféricos

Los periféricos son recursos hardware integrados en el MCU que permiten la interacción con el entorno de la aplicación. Estos pueden interactuar paralelamente a la CPU. Se pueden clasificar en tres tipos digitales, analógicos o mixtos.

Entre los periféricos más comunes se encuentran los puertos de entrada/salida (I/O), temporizadores (timers), convertidores analógicos/digitales (ADC), puertos de comunicación serie, entre otros muchos.

1.2 Timer

El timer se puede definir como un contador impulsado por una señal de reloj periódica. Su funcionamiento puede resumirse de la siguiente manera: Partiendo de un valor arbitrario, normalmente cero, el contador se incrementa cada vez que la señal del reloj hace una transición, por ejemplo: un flanco de subida. Cada transición del reloj se llama “evento del reloj”. El contador lleva la cuenta del número de eventos del reloj. Por ejemplo, un timer alimentado por una señal de reloj con frecuencia f será capaz de calcular el tiempo transcurrido al realizar la operación $n \cdot T$, siendo n el número de eventos de reloj contados por el timer y $T = {}^{1}/_{f}$ el periodo de la señal de reloj.

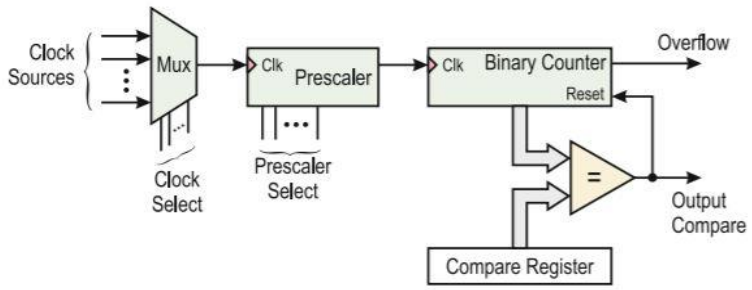

Fig. 1.3 Diagrama de bloques de un timer base.

Por lo general, los timers tienen una estructura base (ver figura 1.3) a la que se le pueden añadir bloques adicionales con el fin de mejorar sus capacidades. Sus componentes fundamentales son:

- Un selector de reloj (MUX), que permite elegir una entre las múltiples fuentes de reloj.

- Un preescalador, que permite dividir la frecuencia del reloj antes de su entrada al timer.

- Un contador binario de n bits, que le da la capacidad de conteo.

- Un registro de comparación de n bits, que le permite limitar el valor máximo del contador que puede alcanzar.

- Un comparador de hardware que le permite saber cuando el contador binario alcanza el valor almacenado en el registro de comparación.

CAPÍTULO 2. MICROCONTROLADOR MSP430FR5969

En la actualidad existe una amplia gama de microcontroladores de diferentes fabricantes, entre los que destacan Atmel, Intel, Texas Instruments o Microchip Technology entre otros muchos.

Para este proyecto el microcontrolador seleccionado ha sido el MSP430FR5969 del fabricante Texas Instruments (figura 2.1).

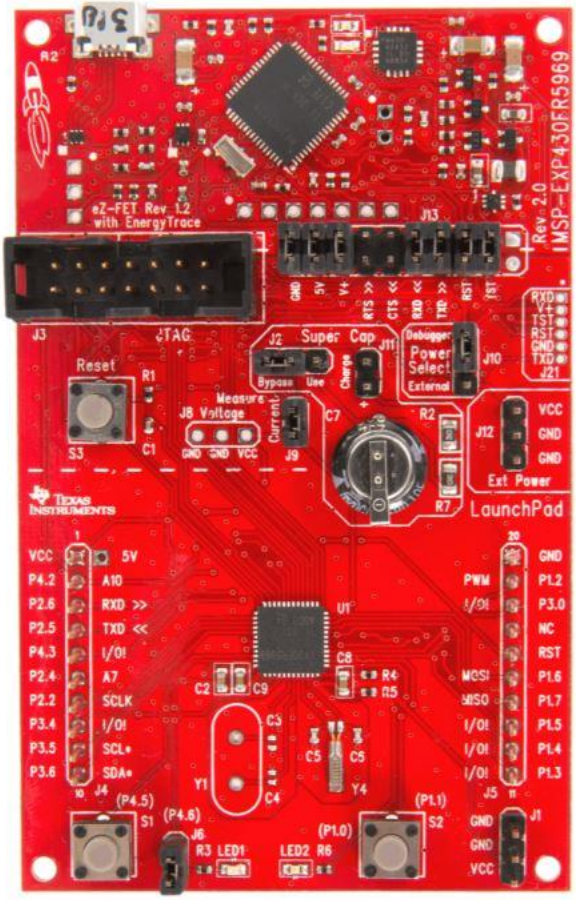

Fig. 2.1 MSP-EXP430FR5969

Este MCU ha sido seleccionado ya que su fabricante es el que ofrece más información sobre consumo y por tanto permite una comparación. Además, ofrecen una gran flexibilidad a la hora de configurar los modos de bajo consumo.

2.1 Características

En este apartado se explicarán las principales características tanto de la placa de desarrollo como del microcontrolador.

2.1.1 MSP-EXP430FR5969

Es un módulo de evaluación (EVM) fácil de usar para el microcontrolador MSP-430FR5969, que contiene todo lo necesario para comenzar a desarrollar proyectos en la plataforma MSP430 FRAM (Memoria Ferroeléctrica de Acceso Aleatorio), incluyendo la emulación a bordo para programar, compilar y realizar mediciones de energía.

Esta placa cuenta con (figura 2.2):

- El microcontrolador MSP430FR5969 de 16 bits.
- 20 pines BoosterPack, esto quiere decir que permite conectar directamente los circuitos deseados a la placa para test sencillos y desarrollo del código.
- Un supercondensador de 0,1F para energía autónoma.
- Un emulador a bordo (eZ-FET) que permite depurar la programación, así como la comunicación con un PC.
- Tecnología EnergyTrace que permite medir el consumo de corriente y ver el estado en el que se encuentra el MCU en tiempo real.
- Tres botones, dos de ellos programables y otro para resetear.
- Dos leds.
- Comunicación UART-PC a través del puerto USB.
- Jumpers que permiten aislar el emulador del MCU.

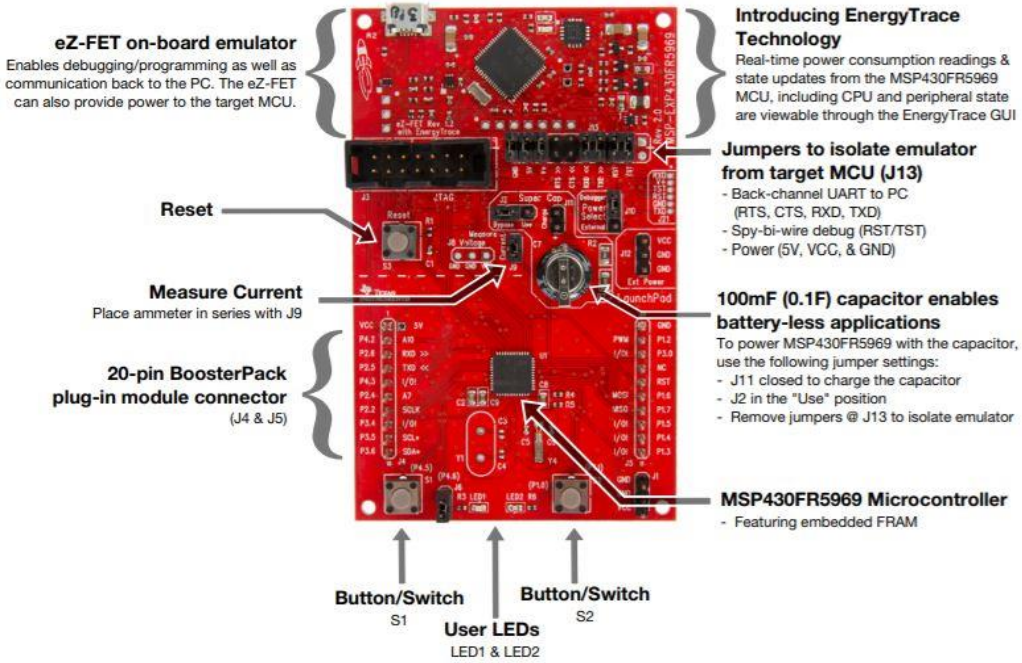

Fig. 2.2 Vista general MSP-EXP430FR5969

2.1.2 MSP430FR5969

El microcontrolador MSP430FR5969 es un dispositivo que contiene la última tecnología FRAM que es una memoria de vanguardia que combina las mejores características del tipo flash y la RAM en una memoria no volátil.

Entre las características más destacadas que han servido de ayuda para la realización del proyecto son las siguientes:

- Tiene una arquitectura RISC de 16 bits y un reloj de hasta 16 MHz y 8 MHz de acceso FRAM.
- Amplio rango de voltaje de entrada 1,8V a 3,6V.
- Amplio rango de temperatura de operación de -40ºC a 85ºC.
- 7 modos de bajo consumo
- Puertos de entrada/salida multifuncionales
- 5 timers de 16 bits con hasta 7 registros de captura/comparación cada uno.
- Tres interfaces serie (UART, SPI, I^2C)
- Tres relojes:

- MCLK: reloj utilizado por la CPU.
- SMCLK: reloj utilizado por los periféricos en alta frecuencia.
- ACLK: reloj utilizado por los periféricos a baja frecuencia.

- Sistema de reloj flexible:
  - DCO de frecuencia fija con 10 frecuencias seleccionables.
  - Reloj interno de baja potencia y baja frecuencia (VLO).
  - Cristales de 32kHz (LFTX).
  - Cristales de alta frecuencia (HFTX).

2.2 Timers

Este MCU tiene cinco timers como ya se ha mencionado en el apartado anterior, éstos pueden ser de dos tipos Timer_A y Timer_B habiendo 4 y 1 de cada tipo respectivamente.

MODULE	TEST CONDITIONS	REFERENCE CLOCK	MIN	TYP	MAX	UNIT
Timer_A		Module input clock		3		μA/MHz
Timer_B		Module input clock		5		μA/MHz

Fig. 2.3 Consumo del Timer_A y Timer_B

Para este proyecto se ha seleccionado el Timer_A ya que su consumo es menor que el del Timer_B como se puede ver en la figura 2.3. Por lo tanto, ahora se explicarán las principales características del Timer_A:

- Contador asíncrono de 16 bits con cuatro modos de funcionamiento.
- Fuente de reloj seleccionable y configurable.
- Siete registros de comparación/captura (TAxCCRn).
- Salidas configurables con capacidad de modulación de ancho de pulso (PWM).
- Registro de vectores de interrupción (TAxIV) para una rápida decodificación de las interrupciones del timer.

2.3 Modos de bajo consumo

El MSP430FR5969 dispone de 7 modos de bajo consumo (LPM) los cuales pueden ser seleccionados a placer por el usuario para su uso. Entre estos modos se encuentran LPM0, LPM1, LPM2, LPM3, LPM4, LPM3.5 y LPM4.5.

MODE	AM		LPM0	LPM1	LPM2	LPM3	LPM4	LPM3.5	LPM4.5	
	ACTIVE	ACTIVE, FRAM OFF[1]	CPU Off[2]	CPU OFF	STANDBY	STANDBY	OFF	RTC ONLY	SHUTDOWN WITH SVS	SHUTDOWN WITHOUT SVS
Maximum system clock	16 MHz		16 MHz	16 MHz	50 kHz	50 kHz	0[3]	50 kHz	0[3]	
Typical current consumption, T_A = 25°C	103 µA/MHz	65 µA/MHz	70 µA at 1 MHz	35 µA at 1 MHz	0.7 µA	0.4 µA	0.3 µA	0.25 µA	0.2 µA	0.02 µA
Typical wake-up time	N/A		instant	6 µs	6 µs	7 µs	7 µs	250 µs	250 µs	1000 µs
Wake-up events	N/A		all	all	LF I/O Comp	LF I/O Comp	I/O Comp	RTC I/O	I/O	
CPU	on		off	off	off	off	off	reset	reset	
FRAM	on	off[1]	standby (or off[1])	off	off	off	off	off	off	
Peripherals in high-frequency state[4]	yes		yes	yes	no	no	no	reset	reset	
Peripherals in low-frequency state[4]	yes		yes	yes	yes	yes[5]	no	RTC	reset	
Peripherals in unclocked state[4]	yes		yes	yes	yes	yes[5]	yes[5]	reset	reset	
MCLK	on (16 MHz_{MAX})		off	off	off	off	off	off	off	
SMCLK	optional[6] (16 MHz_{MAX})		optional[6] (16 MHz_{MAX})	optional[6] (16 MHz_{MAX})	off	off	off	off	off	
ACLK	on (50 kHz_{MAX})		on (50 kHz_{MAX})	on (50 kHz_{MAX})	on (50 kHz_{MAX})	on (50 kHz_{MAX})	off	off	off	
External clock	optional (16 MHz_{MAX})		optional (16 MHz_{MAX})	optional (16 MHz_{MAX})	optional (50 kHz_{MAX})	optional (50 kHz_{MAX})	optional (50 kHz_{MAX})	off	off	
Full retention	yes		yes	yes	yes	yes	yes	no	no	

Fig. 2.4 Modos de operación

Debido a las características de los experimentos, se han seleccionado para este proyecto los siguientes modos:

LPM1 ya que dentro de los que pueden trabajar con periféricos a altas frecuencias (>50kHz) es el que tiene menor consumo (ver figura 2.4).

LPM3 ya que es un modo en el que los periféricos solo trabajan a bajas frecuencias (<50kHz) y dentro de este grupo es el que tiene menor consumo (ver figura 2.4)

CAPÍTULO 3. DESARROLLO DEL PROGRAMA

En este capítulo se especificarán los diferentes escenarios que se analizaran y se hará una descripción del entorno de programación del MCU y de las partes que componen el programa realizado para este proyecto.

3.1 Escenarios

Los escenarios que se van a analizar son los siguientes:

Tabla 3.1 Escenarios a analizar.

	Frecuencia del timer		
	LPM1		**LPM3**
Divisor de frecuencia del timer	**DCO a 8MHz**	**DCO a 1MHz**	**LFXT a 32kHz**
1	8 MHz	1 MHz	32 kHz
2	4 MHz	500 kHz	16 kHz
4	2 MHz	250 kHz	8 kHz
8	1 MHz	125 kHz	4 kHz

Los escenarios mostrados en la tabla 3.1 se realizarán con el timer parado y con el timer activado, lo que da un total de 24 escenarios diferentes. Este proceso se repetirá para cada una de las temperaturas a las que se configurará la cámara climática, 5ºC, 25ºC, 40ºC, 60ºC y 85ºC.

3.2 Entorno de programación

El software empleado para la realización de este proyecto ha sido Code Composer Studio (CCS), que ha sido desarrollado por el mismo fabricante del MCU Texas Instruments (TI). Es un software gratuito exceptuando su versión más actualizada, que es de pago. Para este proyecto se ha utilizado la versión 6.2.0.

Entre el contenido que incluye CCS se pueden destacar:

- Entorno de desarrollo integrado (IDE de sus siglas en inglés). Se refiere a un entorno que reúne las herramientas necesarias para el desarrollo de software. Esto significa que incluye un editor, un sistema de construcción y un depurador. Esto permite el desarrollo de software sin tener que cambiar continuamente entre las herramientas.

- Compilador. Proporciona compiladores de C/C++ para cada una de las familias de placas desarrolladas por TI. Además, se pueden conseguir compiladores adicionales en la App Center del desarrollador.

- Explorador de recursos. El explorador de recursos ayuda a encontrar los últimos ejemplos, bibliotecas, aplicaciones de demostración, hojas de datos y más para la plataforma elegida.

- App center. Se utiliza para obtener elementos adicionales necesarios para el desarrollo de software como pueden ser los compiladores.

3.3 Descripción general

El objetivo final del proyecto es analizar los efectos de la temperatura en el consumo del MCU y uno de sus periféricos (el timer). Para alcanzar el objetivo se ha desarrollado un programa capaz de ejecutar diferentes escenarios y poder cambiar fácilmente de uno a otro. A cada escenario le corresponderá una combinación de unos y ceros obtenidos de los pines de entrada. El cambio de escenario se debe de ser autónomo, es decir, que no sea necesario abrir la cámara climática cada vez que se quiera cambiar de un escenario a otro. Los pasos a seguir para lograr un programa que se adecue a el objetivo son los siguientes:

- Seleccionar las librerías necesarias.

- Reiniciar todos los pines y configurarlos como salidas.

- Configurar las entradas necesarias.

- Crear todos los escenarios que se desean analizar y dentro de éstos:

    - Configurar el reloj.

    - Configurar el timer.

    - Selección de modo de bajo consumo.

Este proceso se puede ver en la figura 3.1.

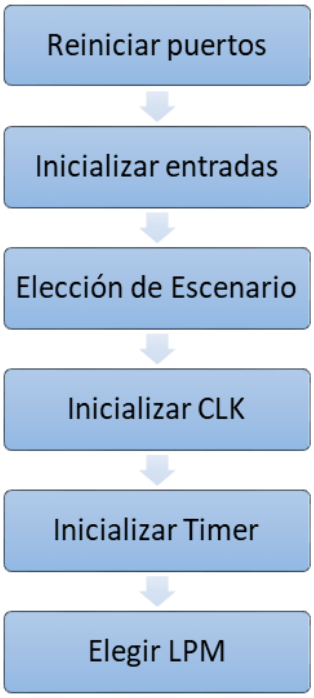

Fig. 3.1. Flujograma del programa

A continuación, en el apartado siguiente se detallarán los pasos comentados anteriormente.

3.4 Código

Primero de todo se deberá inicializar el programa incluyendo las bibliotecas que se utilizaran en el proyecto (ver figura 3.2). Se han seleccionado las siguientes:

- msp430.h que permite reconocer al MCU por el compilador.
- stdio.h que permite usar las funciones de entrada/salida.
- stdint.h que permite definir datos enteros.

```
1 #include <msp430.h>
2 #include <stdio.h>
3 #include <stdint.h>
4
```

Fig. 3.2 Librerías utilizadas en el proyecto

El siguiente paso consiste en reiniciar todos los pines y configurarlos como salidas (ver figura 3.3). Con este paso se busca que se reduzca el consumo al mínimo y evitar posibles errores. Para la realización de este paso se han utilizado los siguientes registros:

WDTCTL

Con este registro se da acceso al watchdog timer, el cual hay que desactivar para esto primero se deberá acceder a él mediante los bits del campo WDTPW y se desactivará poniendo un 1 en el bit de WDTHOLD.

PxOUT

Este registro sirve para hacer una limpieza en los puertos. Poniendo un 0 en este registro se conseguirá resetear los pines.

PxDIR

Con este registro se direccionarán los pines como entradas o salidas, poniendo un 1 para configurarlo como salida o un 0 para entrada. En este caso inicialmente todos los puertos se direccionan como salidas para luego seleccionar como entradas los pines que se deseen. En este programa se han seleccionado como entradas los pines 2, 3, 5 ,6 y 7 del puerto 1 de la placa por facilidad de uso y encontrarse en la misma línea. Se ha descartado los pines 0 y 1 al ser éstos un led y un pulsador y el pin 4 debido que a la hora de realizar las mediciones daba errores.

```
6 int main(void)
7 {
8     WDTCTL = WDTPW | WDTHOLD;   // Stop watchdog timer
9
10
11    // Configure GPIO
12    // Configure GPIOs to it's lowest power state
13      P1OUT = 0;                                // All P1.x reset
14      P1DIR = 0xFF;                             // All P1.x outputs
15      P2OUT = 0;                                // All P2.x reset
16      P2DIR = 0xFF;                             // All P2.x outputs
17      P3OUT = 0;                                // All P3.x reset
18      P3DIR = 0xFF;                             // All P3.x outputs
19      P4OUT = 0;                                // All P4.x reset
20      P4DIR = 0xFF;                             // All P4.x outputs
21      PJOUT = 0;                                // All PJ.x reset
22      PJDIR = 0xFFFF;                           // All PJ.x outputs
23
24     PJSEL0 = BIT4 | BIT5;           // For XT1
25
26     P1DIR &= ~(BIT3 | BIT5 | BIT7 | BIT6 | BIT2);  // P1.3/P1.5/P1.7/P1.6/P1.2 inputs
27
28     // Disable the GPIO power-on default high-impedance mode to activate
29      // previously configured port settings
30      PM5CTL0 &= ~LOCKLPM5;
```

Fig. 3.3 Configuración de los puertos de entrada/salida

A continuación, se empiezan definir los escenarios. Para ello se le asigna un valor de 1 o 0 a los pines de entrada y una combinación de éstos (tabla 3.2) formará un número binario que se le asignará al registro PxIN (ver figura 3.4).

Tabla 3.2 Asignación de escenarios

Caso	P1.2	P1.3	P1.5	P1.6	P1.7	Nº hexadecimal
LPM1, timer parado, DCO a 8MHz y f = 8MHz	0	1	0	0	0	08
LPM1, timer parado, DCO a 8MHz y f = 4MHz	0	0	1	0	0	20
LPM1, timer parado, DCO a 8MHz y f = 2MHz	0	1	1	0	0	28
LPM1, timer parado, DCO a 8MHz y f = 1MHz	0	0	0	0	1	80
LPM1, timer parado, DCO a 1MHz y f = 1MHz	0	1	0	0	1	88
LPM1, timer parado, DCO a 1MHz y f = 500kHz	0	0	1	0	1	A0
LPM1, timer parado, DCO a 1MHz y f = 250kHz	0	1	1	0	1	A8
LPM1, timer parado, DCO a 1MHz y f = 125kHz	0	0	0	1	0	40
LPM3, timer parado, LFXT a 32kHz y f = 32kHz	0	1	0	1	0	48
LPM3, timer parado, LFXT a 32kHz y f = 16kHz	0	0	1	1	0	60
LPM3, timer parado, LFXT a 32kHz y f = 8kHz	0	1	1	1	0	68
LPM3, timer parado, LFXT a 32kHz y f = 4kHz	0	0	0	1	1	C0
LPM1, timer activo, DCO a 8MHz y f = 8MHz	0	1	0	1	1	C8
LPM1, timer activo, DCO a 8MHz y f = 4MHz	0	0	1	1	1	E0
LPM1, timer activo, DCO a 8MHz y f = 2MHz	0	1	1	1	1	E8
LPM1, timer activo, DCO a 8MHz y f = 1MHz	1	0	0	0	0	04
LPM1, timer activo, DCO a 1MHz y f = 1MHz	1	1	0	0	0	0C
LPM1, timer activo, DCO a 1MHz y f = 500kHz	1	0	1	0	0	24
LPM1, timer activo, DCO a 1MHz y f = 250kHz	1	1	1	0	0	2C
LPM1, timer activo, DCO a 1MHz y f = 125kHz	1	0	0	0	1	84
LPM3, timer activo, LFXT a 32kHz y f = 32kHz	1	1	0	0	1	8C
LPM3, timer activo, LFXT a 32kHz y f = 16kHz	1	0	1	0	1	A4
LPM3, timer activo, LFXT a 32kHz y f = 8kHz	1	1	1	0	1	AC
LPM3, timer activo, LFXT a 32kHz y f = 4kHz	1	0	0	1	0	44

Una vez se le ha asignado un número al registro PxIN se pasa a la configuración del reloj donde se podrá elegir el reloj principal y su frecuencia, luego se configura el timer y por último se elige el modo de bajo consumo que se desea emplear. Para esta sección del programa se han usado los siguientes registros:

PxIN

Cada bit en este registro refleja el valor de la señal de entrada a su correspondiente pin de entrada/salida. Este registro es solo de lectura y con él se leen los valores de los pines de entrada.

CSCTL0

Con este registro permite el acceso para escribir sobre los registros del reloj,

CSCTL1

Con este registro se elige la frecuencia del DCO. Si el bit de DCOSEL se le asigna el numero hexadecimal 7 la frecuencia será de 8MHz (ver figura 3.4) y si se le asigna un 0 la frecuencia será de (1MHz ver figura 3.5).

CSCTL2

Con este registro se le asigna la frecuencia al reloj. Para altas frecuencias se le asigna la frecuencia del DCO al SMCLK (ver figura 3.4 y 3.5) y para bajas frecuencias el DCO tiene que estar a 1MHz y al ACLK se le asigna la frecuencia de 32 kHz del LFXT (ver figura 3.6).

Estas selecciones se han hecho mediante los bits:

- SELA, elige la frecuencia del ACLK
- SELS, elige la frecuencia del SMCLK
- SELM, elige la frecuencia del MCLK

CSCTL3

Este registro permite hacer divisiones en la frecuencia de los relojes, pero para este proyecto se ha decidido que sean valores fijos por lo que todos los divisores serán 1.

CSCTL4

Este registro da la posibilidad de apagar el LFXT cuando éste no sea la fuente de frecuencia de ninguno de los relojes.

```
if (P1IN==0x08){
    // Clock System Setup
    CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
    CSCTL1 = DCOFSEL_7;                      // Set DCO to 8MHz
    CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set SMCLK = MCLK = DCO ACLK = VLOCLK
    CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to 1
    CSCTL4 &= ~LFXTOFF;

    do
      {
        CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault flag
        SFRIFG1 &= ~OFIFG;
      } while (SFRIFG1 & OFIFG);                // Test oscillator fault flag

    CSCTL0_H = 0x00;                          // Lock CS module (use byte mode to upper byte)

    TA1CTL = TASSEL__SMCLK | ID_0 | MC__STOP ;  // Use SMCLK as clock source, Stop timer.

    __bis_SR_register(LPM1_bits);  //For SMCLK as main source
```

Fig. 3.4 LPM1 con DCO a 8MHz y timer parado

```
if(P1IN==0x88){
    // Clock System Setup
        CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
        CSCTL1 = DCOFSEL_0;                      // Set DCO to 1MHz
        CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set SMCLK = MCLK = DCO ACLK = VLOCLK
        CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to 1
        CSCTL4 &= ~LFXTOFF;

        do
          {
            CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault flag
            SFRIFG1 &= ~OFIFG;
          } while (SFRIFG1 & OFIFG);                // Test oscillator fault flag

        CSCTL0_H = 0x00;                          // Lock CS module (use byte mode to upper byte)

        TA1CTL = TASSEL__SMCLK | ID_0 | MC__STOP ;  // Use SMCLK as clock source, Stop timer.

        __bis_SR_register(LPM1_bits);  //For SMCLK as main source
```

Fig. 3.5 LPM1 con DCO a 1MHz y timer parado

```
if(P1IN==0x48){
    // Clock System Setup
        CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
        CSCTL1 = DCOFSEL_0;                      // Set DCO to 1MHz
        CSCTL2 = SELA__LFXTCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set SMCLK = MCLK = DCO ACLK = LFXT
        CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to 1
        CSCTL4 &= ~LFXTOFF;

        do
          {
            CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault flag
            SFRIFG1 &= ~OFIFG;
          } while (SFRIFG1 & OFIFG);                // Test oscillator fault flag

        CSCTL0_H = 0x00;                          // Lock CS module (use byte mode to upper byte)

        TA1CTL = TASSEL__ACLK | ID_0 | MC__STOP ;  // Use ACLK as clock source, Stop timer.

        __bis_SR_register(LPM3_bits);  //For ACLK as main source
```

Fig. 3.6 LPM3 con LFXT a 32kHz y timer parado

Para la configuración del timer se ha usado el registro TAxCTL.

TAxCTL

- Este registro permite configurar las características más básicas del Timer_A, tales como:

- El reloj que hace de fuente (TASSEL). Este bit permite el cambio entre los dos relojes que se van a usar SMCLK o ACLK según se requiera.

- El divisor de frecuencia (ID). A este bit se le pueden asignarlos valores 0, 1, 2 y 3 que corresponden respectivamente a los divisores /1, /2, /4 y /8.

- Modo de control (MC). Este bit controla el modo en el que se encuentra el timer, pudiendo elegir entre STOP cuando se requiera que el timer este parado y CONTINUOUS para cuando se requiera que el timer este activo.

Por último, solo queda elegir el modo de bajo consumo para esto se utiliza el registro **_bis_SR_register** pudiendo elegir entre LPM1 (ver figura 3.3 y 3.4) o LPM3 (ver figura 3.5)

Este patrón (figuras 3.3, 3.4 y 3.5) se repite para cada uno de los escenarios mostrados en el apartado 3.1 haciendo sus respectivos cambios en la selección de reloj, la frecuencia de los relojes, divisores de frecuencia del timer, modos de control del timer y modo de bajo consumo.

Para más información sobre los registros utilizados ver Anexos.

CAPÍTULO 4. INSTRUMENTACIÓN Y MONTAJE

4.1 Instrumentación

Para el montaje de este experimento se han necesitado de los siguientes instrumentos.

En primer lugar, se necesitará un instrumento capaz de alimentar externamente al MCU, para esta función se ha seleccionado el Keysight E3631A (figura 4.1). Este dispositivo suministrará una tensión de 3V al MCU.

Fig. 4.1 Keysight E3631A

Para las mediciones de corriente se ha seleccionado el multímetro Agilent 34410ª (figura 4.2) que tiene una resolución de $6^{1/2}$ dígitos y una exactitud de 0,050% de lectura + 0,025% del rango al medir corrientes continuas en la escala de 100 µA.

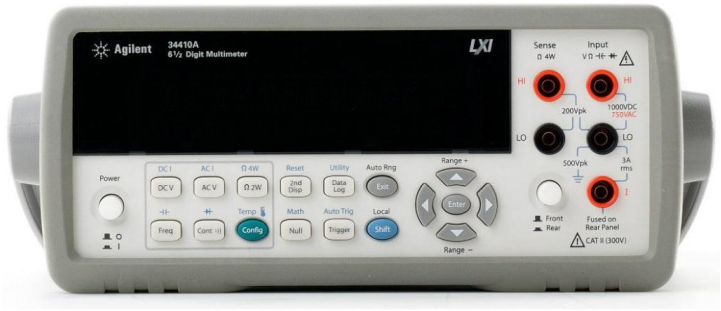

Fig. 4.2 Agilen 34410A

Para poder asignar valores al registro PxIN se necesita de una placa protoboard que estará alimentada a 3V al igual que el MCU.

Por último, se necesitará una cámara climática que será la encargada de crear el ambiente para los distintos escenarios (5ºC, 25ºC, 40ºC, 60ºC y 85ºC). Se ha elegido la cámara Espec SU-242 (figura 4.3).

Fig. 4.3 Espec SU-242

Entre sus características principales se destaca las siguientes:

- Volumen interior de 22L.

- Rango de temperatura de -40ºC a 150ºC.
- Fluctuación de la temperatura de ±0,3ºC.
- Tasa promedio de cambio:
  - 3ºC/min calentando.
  - 1,2ºC/min enfriando.

Los valores de temperatura elegidos para el experimento se han ido consiguiendo primero aplicando un descenso hasta los 5ºC y luego aplicando escalones graduales en sentido ascendente.

4.2 Montaje

El montaje del experimento se realizará de la siguiente manera (figura 4.4);

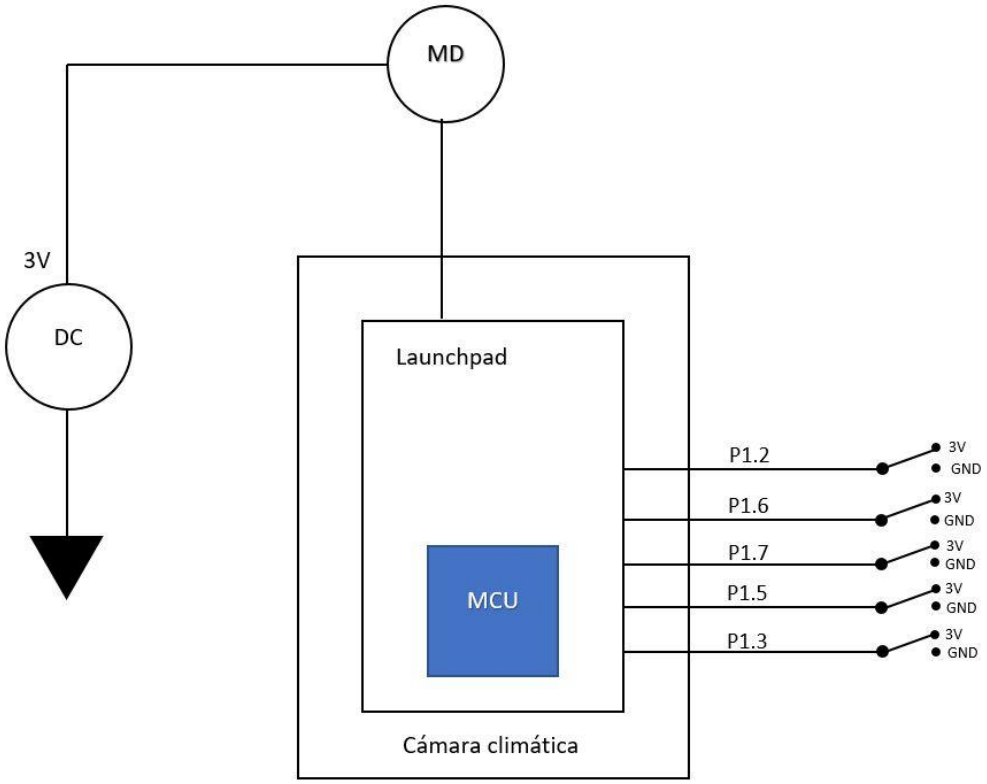

Fig. 4.4 Diagrama de bloques del montaje

Para el montaje de este experimento en primer lugar se ha de cargar el programa en el MCU, para ello se ha conectado el MCU a un ordenador mediante el conector USB. El MCU además deberá estar alimentado

externamente. Por otro lado, en la placa tiene que haber dos jumpers conectados a los pines RST y TST, otro jumper en configuración de alimentación externa (J10) y otro en current (ver figura 4.5).

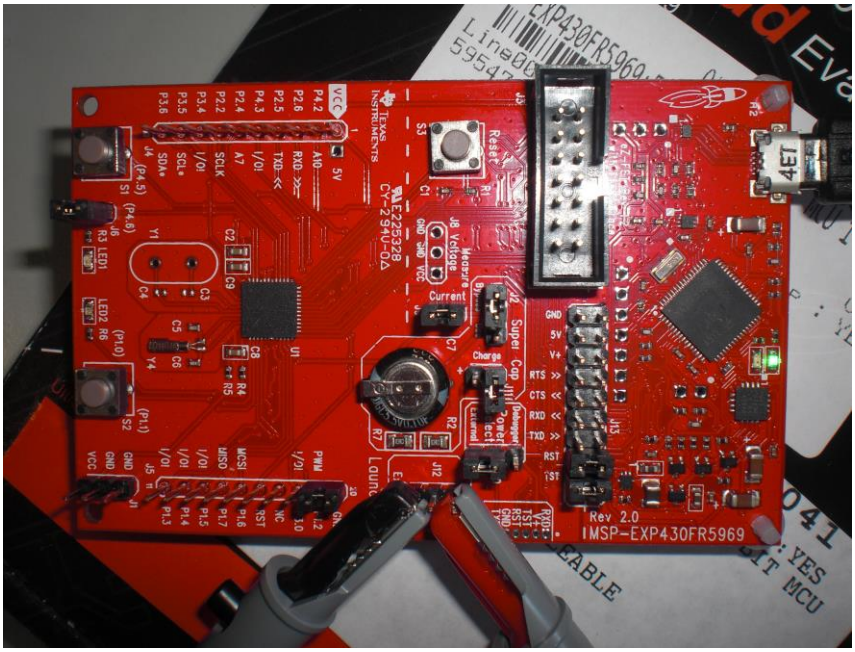

Fig. 4.5 Configuración de alimentación externa y carga del programa

Una vez se ha cargado el programa se quitan los jumpers de los pines RTS y TST y el jumper de current. Luego se introduce la placa en la cámara climática y se le conecta el multímetro en serie en los pines de current. En los pines de entrada se les pone un conector con un cable soldado, lo suficientemente largo como para que llegue al exterior de la cámara, a la pata que le corresponde a cada pin con un protector térmico (ver figura 4.6). También se soldarán dos cables al botón de reset por la parte de abajo de la placa.

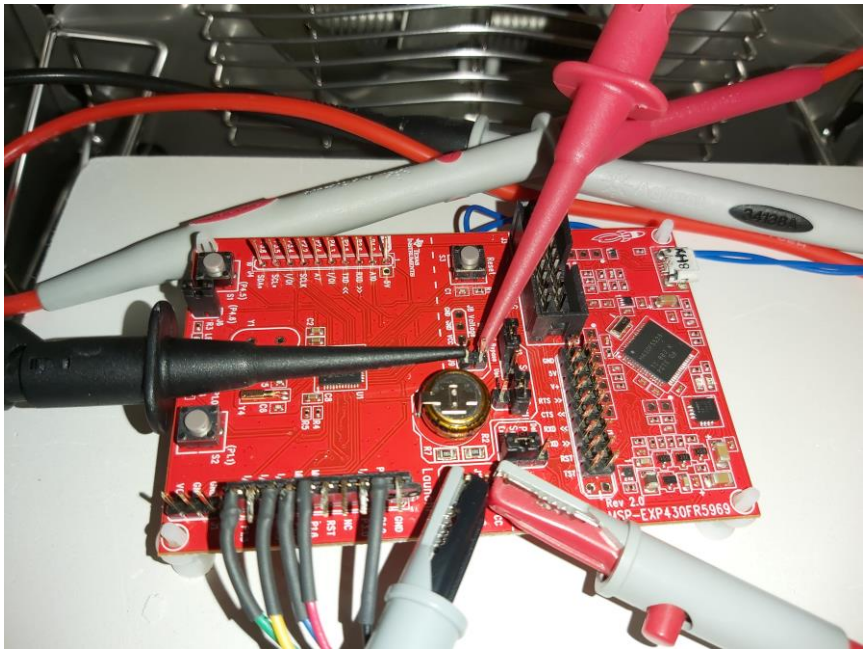

Fig. 4.6 Vista de la placa en el interior de la cámara climática

Para terminar el montaje en el exterior de la cámara climática se encuentra la placa protoboard, que al igual que el MCU está alimentada a 3V y donde se conectarán los cables que vienen de los pines de entrada de la placa. En la placa protoboard hay una línea conectada a los 3V y otra línea conectada a tierra. En función de que valor se quiera asignar al registro PxIN se conectará el cable a la línea de 3V si se quiere un 1 o a la línea de tierra si se quiere un 0 (figura 4.7). Además, también estarán conectados a la protoboard los cables del botón reset que al provocar un cortocircuito con estos seria lo equivalente a pulsar el botón. Esto último será de gran ayuda ya que cada vez que se quiera cambiar de escenario es necesario pulsar el botón de reset.

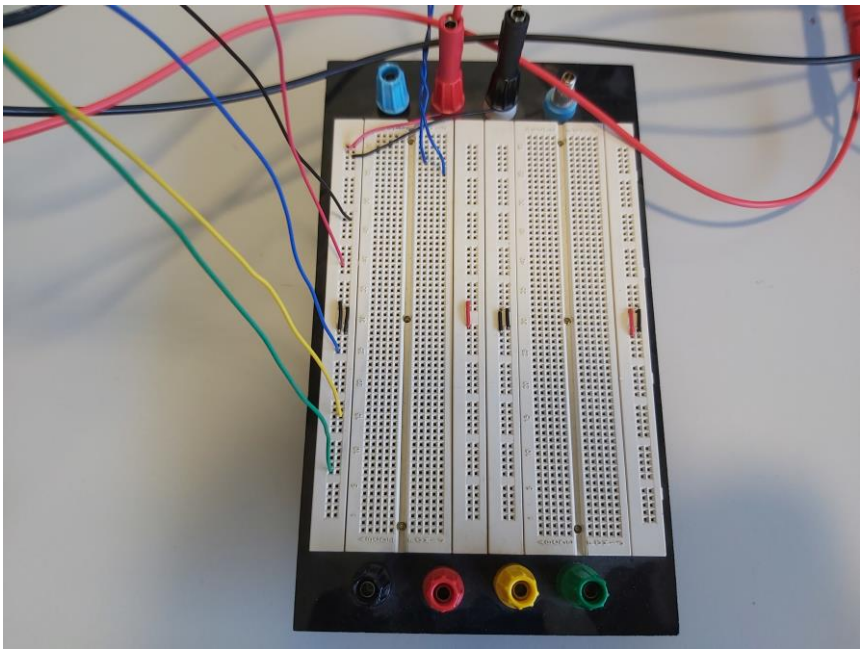

Fig. 4.7 Placa protoboard alimentada a 3V y conexiones realizadas

La figura 4.8 muestra una vista general del montaje antes de encender la cámara climática.

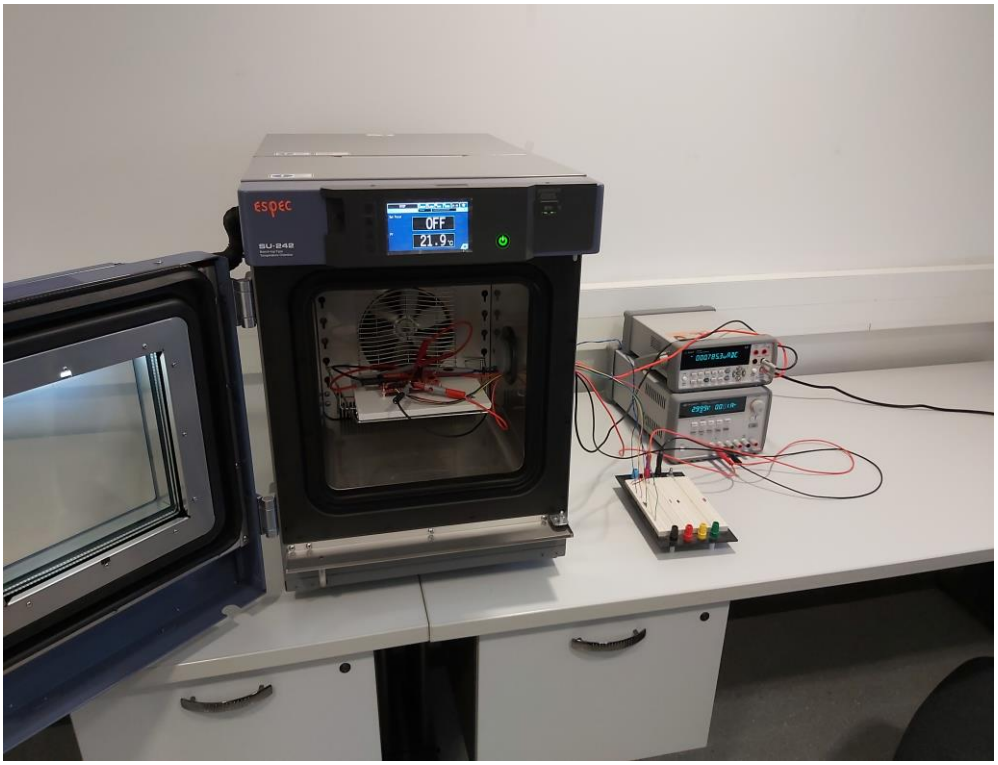

Fig. 4.8 Vista general del montaje

CAPÍTULO 5. RESULTADOS

Una vez se han planteado los diferentes escenarios a estudiar y se ha conseguido un programa que funcione, se procede a la realizar las medidas. Para cada una de las mediciones se han tomado 10 muestras y se ha hecho la media aritmética de éstas.

5.1 Timer parado

En esta primera prueba se ha medido el consumo del MCU con el timer parado, por lo que los resultados obtenidos (tabla 5.1, 5.2 y 5.3) serán el consumo del microcontrolador estando en el correspondiente modo de bajo consumo.

Tabla 5.1 Consumo del MCU en LPM1, timer parado y DCO a 8MHz

	T=5ºC	T=25ºC	T=40ºC	T=60ºC
Frecuencia	Corriente (µA)	Corriente (µA)	Corriente (µA)	Corriente (µA)
8 MHz	109,42	109,84	110,23	111,09
4 MHz	109,41	109,83	110,22	111,08
2 MHz	109,41	109,82	110,24	111,07
1 MHz	109,41	109,84	110,24	111,07

Tabla 5.2 Consumo del MCU en LPM1, timer parado y DCO a 1MHz

	T=5ºC	T=25ºC	T=40ºC	T=60ºC
Frecuencia	Corriente (µA)	Corriente (µA)	Corriente (µA)	Corriente (µA)
1 MHz	32,98	33,51	34,00	34,99
500 kHz	32,98	33,51	34,00	34,97
250 kHz	32,98	33,51	34,00	34,97
125 kHz	32,98	33,51	34,00	34,98

Tabla 5.3 Consumo del MCU en LPM3, timer parado y LFXT a 32kHz

	T=5ºC	T=25ºC	T=40ºC	T=60ºC	T=85ºC
Frecuencia	Corriente (nA)	Corriente (nA)	Corriente (nA)	Corriente (nA)	Corriente (nA)
32 kHz	631,26	665,52	718,76	888,59	1560,44
16 kHz	631,24	665,46	718,61	888,61	1558,97
8 kHz	631,21	665,46	718,47	888,23	1557,54
4 kHz	631,25	665,43	718,34	888,22	1557,51

Al observar los datos de las tablas anteriores se ve como hay un aumento en el consumo a medida que aumenta la temperatura y se mantiene constante en frecuencia. Esto se debe a que el divisor de frecuencia se le aplica al timer y éste al estar parado no afecta en el consumo del MCU. Para apreciar mejor este crecimiento se ha decido hacer una representación gráfica de los resultados (ver figura 5.1, 5.2 y 5.3).

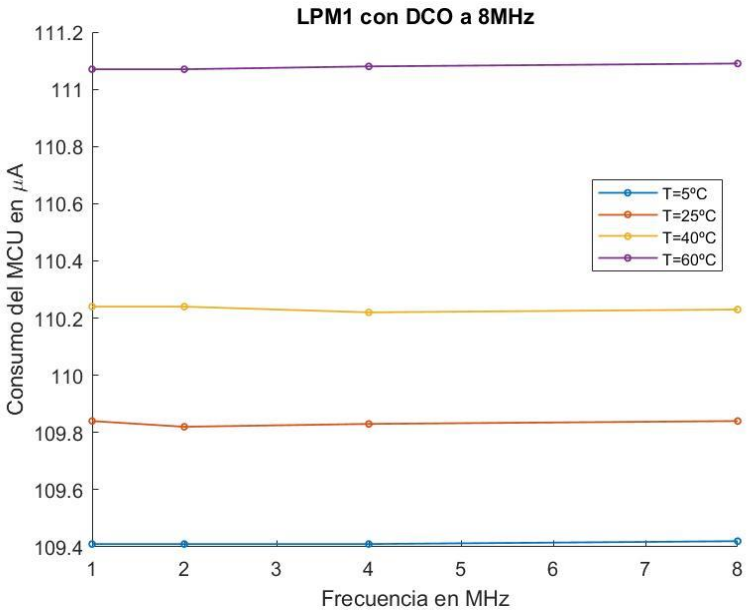

Fig. 5.1 Consumo del MCU respecto a la frecuencia en LPM1 con el timer parado y DCO a 8MHz

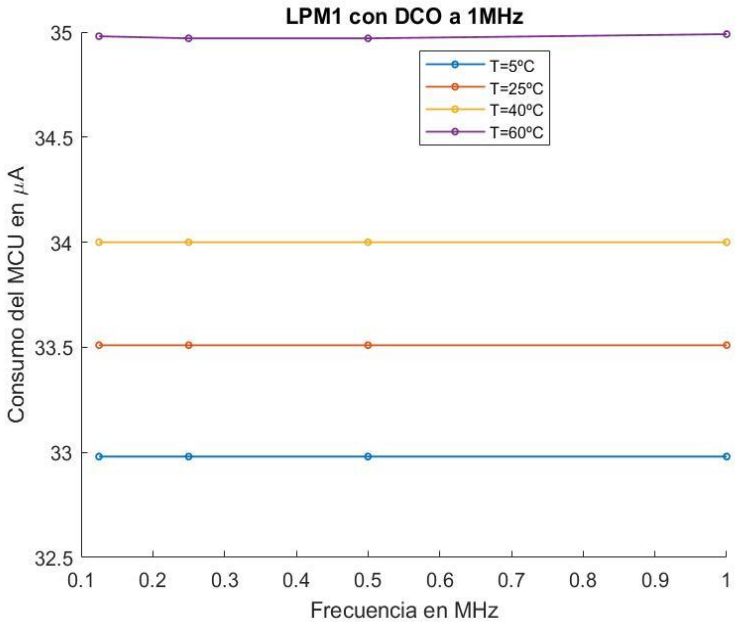

Fig. 5.2 Consumo del MCU respecto a la frecuencia en LPM1 con el timer parado y DCO a 1MHz

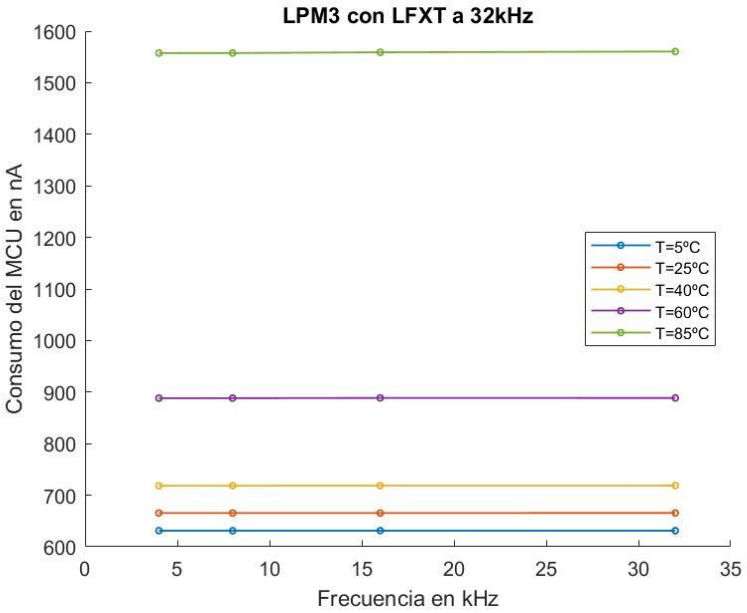

Fig. 5.3 Consumo del MCU respecto a la frecuencia en LPM3 con el timer parado y LFXT a 32kHz

5.2 Timer activo

En esta ocasión se han vuelto a repetir las pruebas del apartado anterior pero ahora con el timer activado. De esta manera se podrá saber el consumo del timer.

Primeramente, se muestran los datos de consumo del MCU (tabla 5.4, 5.5 y 5.6).

Tabla 5.4 Consumo del MCU en LPM1, timer activo y DCO a 8MHz

	T=5ºC	T=25ºC	T=40ºC	T=60ºC
Frecuencia	Corriente (µA)	Corriente (µA)	Corriente (µA)	Corriente (µA)
8 MHz	133,60	134,15	134,70	135,68
4 MHz	125,50	126,04	126,32	127,48
2 MHz	121,57	122,08	122,58	123,48
1 MHz	119,49	119,99	120,47	121,36

Tabla 5.5 Consumo del MCU en LPM1, timer activo y DCO a 1MHz

	T=5ºC	T=25ºC	T=40ºC	T=60ºC
Frecuencia	Corriente (µA)	Corriente (µA)	Corriente (µA)	Corriente (µA)

1 MHz	36,00	36,52	37,05	38,05
500 kHz	34,99	35,53	36,03	37,02
250 kHz	34,49	35,04	35,54	36,52
125 kHz	34,23	34,78	35,28	36,26

Tabla 5.6 Consumo del MCU en LPM3, timer activo y LFXT a 32kHz

	T=5ºC	T=25ºC	T=40ºC	T=60ºC	T=85ºC
Frecuencia	Corriente (nA)	Corriente (nA)	Corriente (nA)	Corriente (nA)	Corriente (nA)
32 kHz	751,21	789,75	849,94	1046,82	1838,45
16 kHz	717,60	756,26	816,47	1013,11	1800,19
8 kHz	700,84	739,51	800,01	997,41	1777,58
4 kHz	692,53	730,92	790,95	988,68	1777,55

En segundo lugar, se muestran los datos de consumo del timer (tabla 5.7, 5.8 y 5.9). Para la obtención de estos datos, al consumo del MCU con el timer activo se le ha restado el consumo del MCU con el timer parado. Los resultados son los siguientes:

Tabla 5.7 Consumo del timer en LPM1 y DCO a 8MHz

	T=5ºC	T=25ºC	T=40ºC	T=60ºC
Frecuencia	Corriente (µA)	Corriente (µA)	Corriente (µA)	Corriente (µA)
8 MHz	24,18	24,31	24,47	24,59
4 MHz	16,09	16,21	16,10	16,40
2 MHz	12,16	12,26	12,34	12,41
1 MHz	10,08	10,15	10,23	10,29

Tabla 5.8 Consumo del timer en LPM1 y DCO a 1MHz

	T=5ºC	T=25ºC	T=40ºC	T=60ºC
Frecuencia	Corriente (µA)	Corriente (µA)	Corriente (µA)	Corriente (µA)
1 MHz	3,02	3,01	3,05	3,06
500 kHz	2,01	2,02	2,03	2,05
250 kHz	1,51	1,53	1,54	1,55
125 kHz	1,25	1,27	1,28	1,28

Tabla 5.9 Consumo del timer en LPM1 y LFXT a 32kHz

	T=5ºC	T=25ºC	T=40ºC	T=60ºC	T=85ºC
Frecuencia	Corriente (nA)	Corriente (nA)	Corriente (nA)	Corriente (nA)	Corriente (nA)
32 kHz	119,95	124,23	131,18	158,23	278,01

16 kHz	86,36	90,80	97,86	124,50	241,22
8 kHz	69,63	74,05	81,54	109,18	220,04
4 kHz	61,28	65,49	72,61	100,46	220,04

En esta ocasión nuevamente se vuelve a apreciar que al aumentar la temperatura el consumo tanto del MCU como del timer aumentan, pero en esta ocasión al aumentar la frecuencia también aumenta el consumo debido a que ahora si está el timer activado. Para una mejor comprensión de los datos se ha representado gráficamente tanto el consumo del MCU (figura 5.4, 5.5 y 5.6) como el consumo del timer (figura 5.7, 5.8 y 5.9).

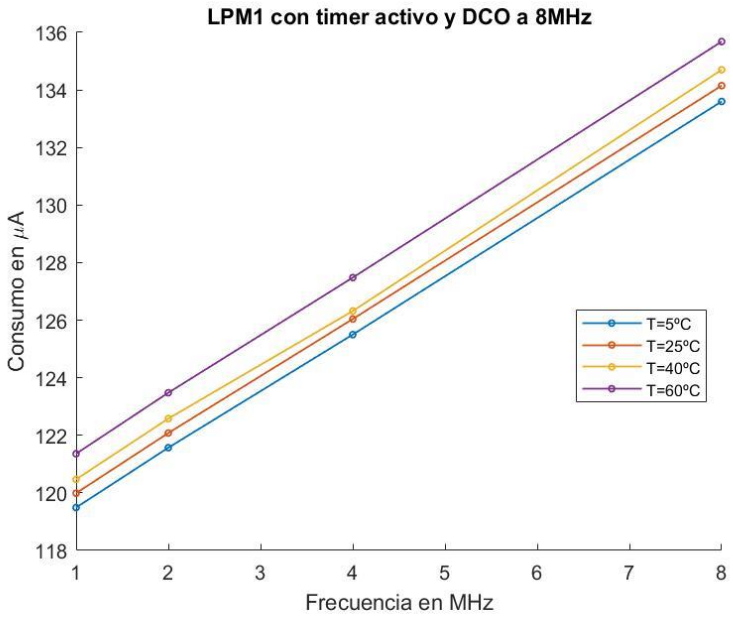

Fig. 5.4 Consumo MCU, timer activo y DCO a 8MHz

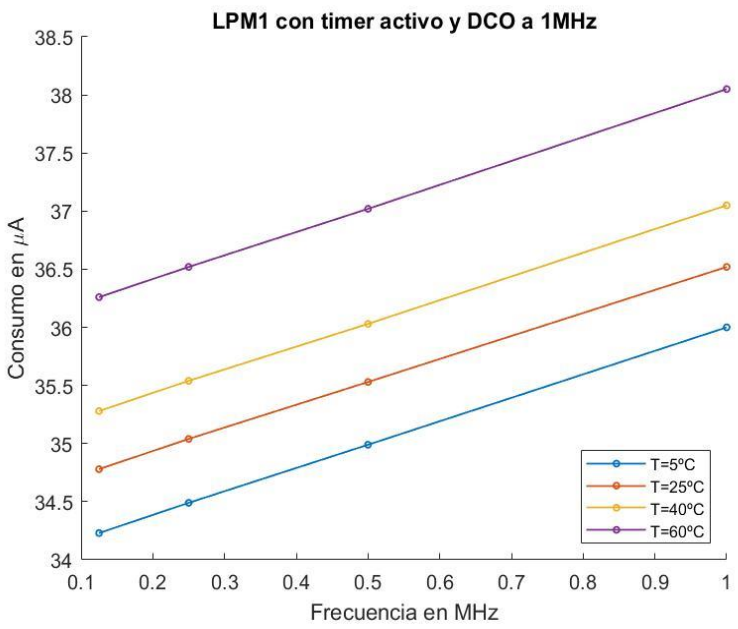

Fig. 5.5 Consumo MCU, timer activo y DCO a 1MHz

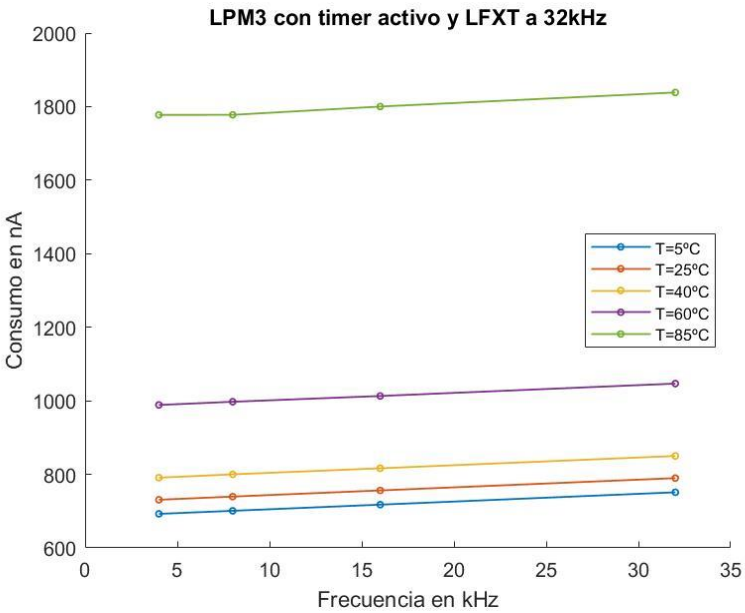

Fig. 5.6 Consumo MCU, timer activo y LFXT a 32kHz

Como muestran estas gráficas (figuras 5.4, 5.5 y 5.6) el aumento del consumo es lineal a medida que aumenta la frecuencia independientemente del modo que se esté ejecutando y de la temperatura. También se puede apreciar que las pendientes de las rectas resultantes son prácticamente iguales, es decir el consumo del MCU es mayor ya que el punto de partida es más alto pero el crecimiento de éste es prácticamente independiente a la temperatura

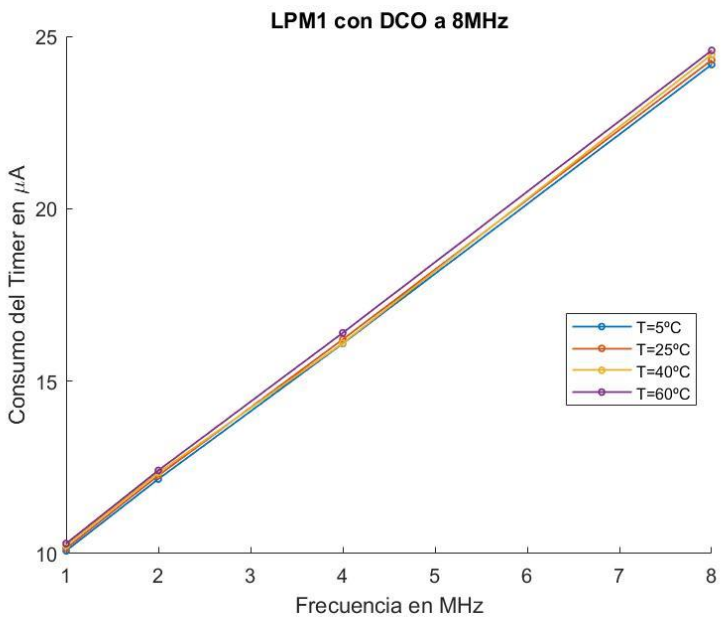

Fig. 5.7 Consumo del timer respecto de la frecuencia en LPM1 y DCO a 8MHz

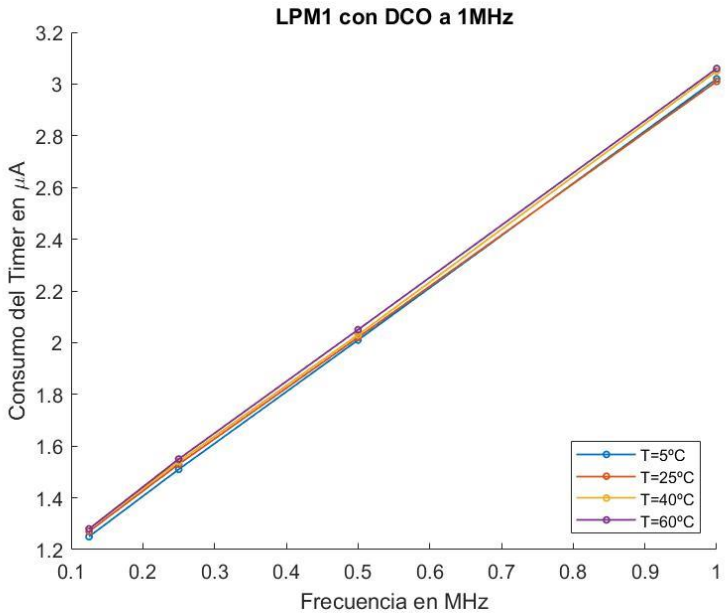

Fig. 5.8 Consumo del timer respecto de la frecuencia en LPM1 y DCO a 1MHz

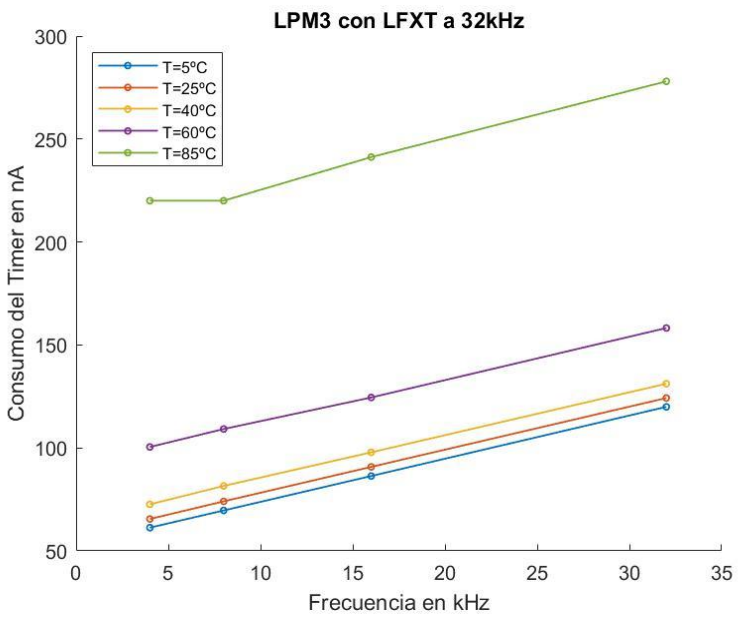

Fig. 5.9 Consumo del timer respecto de la frecuencia en LPM3 y LFXT a 32kHz

Al observar las figuras 5.7 y 5.8 no se pueden sacar datos concluyentes aparte de que el consumo del timer es prácticamente el mismo en ambos casos y crece de manera lineal con la frecuencia. Sin embargo, la figura 5.9 al mostrar el consumo en nanoamperios ofrece unos datos más interesantes. En este caso se vuelve a ver que el consumo aumenta al aumentar la temperatura y la frecuencia. Este aumento a pesar de ser pequeño (en torno a 2nA/Hz) puede llegar a ser significativo ya que el consumo total del MCU con el timer activo está en torno a 0,70µA en la situación más favorable y 1,8µA en la situación más desfavorable (tabla 5.6).

Como las figuras 5.7 y 5.8 no han servido de mucha ayuda para observar los efectos térmicos, también se ha decidido representar los valores de consumo respecto de la temperatura y establecer una línea de tendencia para cada caso.

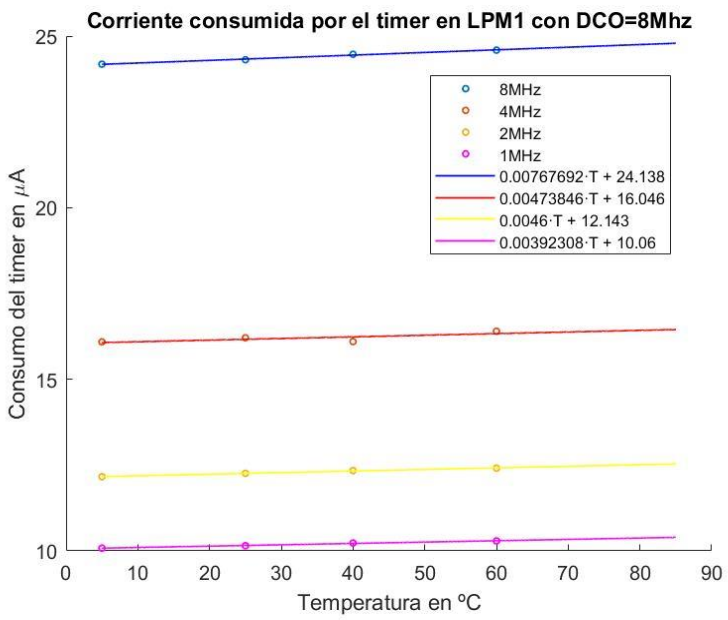

Fig. 5.10 Consumo del timer respecto de la temperatura en LPM1 y DCO a 8MHz

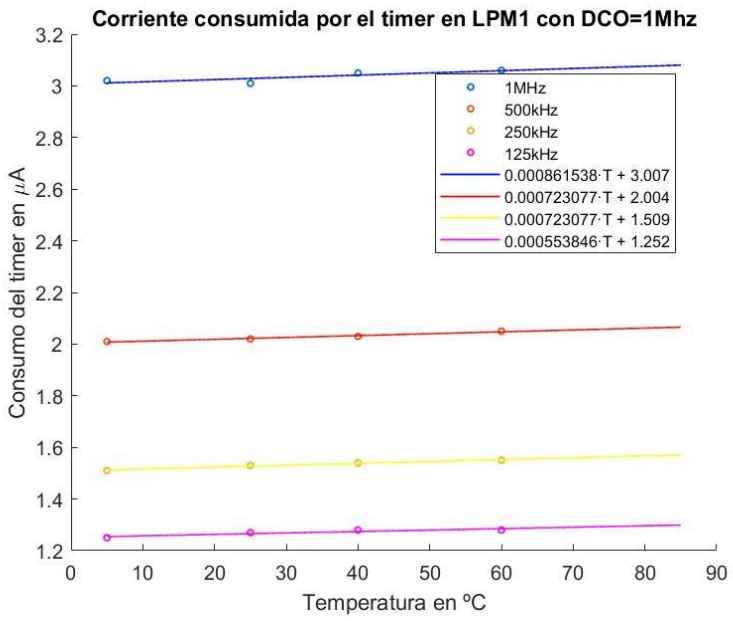

Fig. 5.11 Consumo del timer respecto de la temperatura en LPM1 y DCO a 1MHz

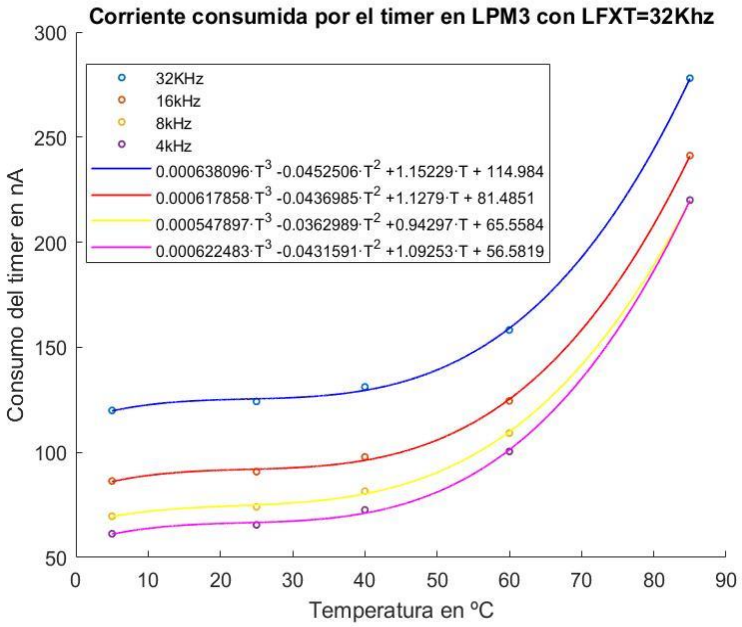

Fig. 5.12 Consumo del timer respecto de la temperatura en LPM3y LFXT a 32kHz

En LPM1 tanto para DCO a 8MHz (figura 5.10) y DCO a 1MHz (figura 5.11) se puede apreciar que los valores experimentales se ajustan casi a la perfección con la línea de tendencia. También se puede apreciar que cuando el DCO está a 8MHz las pendientes de las rectas se encuentran entre los valores 4nA/ºC y 7nA/ºC y cuando el DCO está a 1MHz las pendientes de las rectas se encuentran entre los valores 0,5nA/ºC y 0,8 nA/ºC. Estos valores representan un aumento del consumo muy bajo en comparación con el consumo global del MCU en estos casos, ya que éstos van desde los 119µA a 135µA para el caso de DCO a 8MHz y desde 34µA a 38µA para cuando el DCO está a 1MHZ.

Por otro lado, está el caso de LPM3 (figura 5.12), aunque no se termina de apreciar bien si hay un crecimiento exponencial o polinómico de grado 3 como muestra la línea de tendencia debido al bajo consumo del MCU en este modo, el consumo del timer puede llegar a tener un gran impacto en la vida útil del MCU.

Por último, hay que comentar que para el modo LPM1 se han descartado los valores para la temperatura de 85ºC, ya que daban resultados erróneos.

CAPÍTULO 6. CONCLUSIONES

Una vez finalizado el proyecto se han podido extraer las siguientes conclusiones.

En primer lugar, en cuanto a los objetivos planteados inicialmente:

- Se ha realizado un aprendizaje de las características y funciones de un microcontrolador comercial y sus periféricos poniendo más atención en el timer.

- Se han aprendido nuevas herramientas de programación dentro del entorno de programación ofrecido por el fabricante, consiguiendo realizar con éxito el desarrollo de una aplicación.

- Se ha aprendido a usar una cámara climática además de sus principales características y sus ciclos de trabajo.

El cumplimiento de los puntos anteriores ha permitido alcanzar el objetivo principal, es decir, se ha logrado realizar con éxito un estudio sobre los efectos de la temperatura en un microcontrolador comercial. A raíz de los resultados obtenidos del estudio se pueden extraer las siguientes conclusiones:

- El consumo del timer aumenta linealmente con la frecuencia y la temperatura cuando este trabaja a altas frecuencias y cuando trabaja a bajas frecuencias el consumo aumenta linealmente con la frecuencia, pero no de forma lineal con la temperatura.

- El consumo del timer es insignificante en comparación con el consumo global del MCU cuando trabaja a altas frecuencias y para bajas frecuencias el consumo tiene un gran peso dentro del consumo global del MCU.

BIBLIOGRAFÍA

[1] Martí Campoy, A. “Microcontroladores”, Cap.10 en *Un viaje a la historia de la informática,* Molero Prieto, pp 113-126, Universitat Politècnica de València, Valencia, España (2016)

[2] Reverter, F. “Interfacing sensors to microcontrollers: A direct approach”, Cap.2 en *Smart sensors and MEMS*, Nihtianov, S. and Luque, A., pp 23-34, Woodhead Publishing, 2a ed., Duxford, UK, (2018).

[3] Jimenez, M. Palomera, R. Couvertier, I. “Microcomputer Organization”, Cap.3 en *Introducction to embedded systems*, pp 81-155, Springer, Mayagüez, Puerto Rico, USA, (2014)

[4] F. Reverter and M. Gasulla, "Experimental Characterization of the Energy Consumption of ADC Embedded into Microcontrollers Operating in Low Power," *IEEE International Instrumentation and Measurement Technology Conference (I2MTC*), Auckland, New Zealand, pp. 1-5, (2019),

[5] F. Reverter, "Toward Non-CPU Activity in Low-Power MCU-Based Measurement Systems," in *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 1, pp. 15-17, (2020)

[6] Texas Instruments, MSP430FR59xx Mixed-Signal Microcontrollers, (PDF)

[7] Texas Instruments, MSP430FR5969 LaunchPad™ Development Kit (MSP‑EXP430FR5969)

[8] Texas Instruments, MSP430FR58xx, MSP430FR59xx, and MSP430FR6xx Family User´s Guide

[9] Texas Instruments, Code Composer Studio User´s Guide

ANEXOS

Anexo I. Registros entradas/salidas

Registro PxIN

7	6	5	4	3	2	1	0
PxIN							
r	r	r	r	r	r	r	r

Bit	Field	Type	Reset	Description
7-0	PxIN	R	Undefined	Port x input 0b = Input is low 1b = Input is high

Registro PxOUT

7	6	5	4	3	2	1	0
PxOUT							
rw	rw	rw	rw	rw	rw	rw	rw

Bit	Field	Type	Reset	Description
7-0	PxOUT	RW	Undefined	Port x output When I/O configured to output mode: 0b = Output is low. 1b = Output is high. When I/O configured to input mode and pullups/pulldowns enabled: 0b = Pulldown selected 1b = Pullup selected

Registro PxDIR

7	6	5	4	3	2	1	0
PxDIR							
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

Bit	Field	Type	Reset	Description
7-0	PxDIR	RW	0h	Port x direction 0b = Port configured as input 1b = Port configured as output

Anexo II. Registros del Sistema de reloj

Registro CSCTL0

15	14	13	12	11	10	9	8
CSKEY							
rw-1	rw-0	rw-0	rw-1	rw-0	rw-1	rw-1	rw-0
7	6	5	4	3	2	1	0
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0

Bit	Field	Type	Reset	Description
15-8	CSKEY	RW	96h	CSKEY password. Must always be written with A5h; a PUC is generated if any other value is written. Always reads as 96h. After the correct password is written, all CS registers are available for writing.
7-0	Reserved	R	0h	Reserved. Always reads as 0.

Registro CSCTL1

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
Reserved	DCORSEL	Reserved		DCOFSEL			Reserved
r0	rw-[0]	r0	r0	rw-[1]	rw-[1]	rw-[0]	r0

Bit	Field	Type	Reset	Description
15-7	Reserved	R	0h	Reserved. Always reads as 0.
6	DCORSEL	RW	0h	DCO range select. For high-speed devices, this bit can be written by the user. For low-speed devices, it is always 0. See description of DCOFSEL bit for details.
5-4	Reserved	R	0h	Reserved. Always reads as 0.
3-1	DCOFSEL	RW	6h	DCO frequency select. Selects frequency settings for the DCO. Values shown below are approximate. Refer to the device-specific data sheet. 000b = If DCORSEL = 0: 1 MHz; If DCORSEL = 1: 1 MHz 001b = If DCORSEL = 0: 2.67 MHz; If DCORSEL = 1: 5.33 MHz 010b = If DCORSEL = 0: 3.5 MHz; If DCORSEL = 1: 7 MHz 011b = If DCORSEL = 0: 4 MHz; If DCORSEL = 1: 8 MHz 100b = If DCORSEL = 0: 5.33 MHz; If DCORSEL = 1: 16 MHz 101b = If DCORSEL = 0: 7 MHz; If DCORSEL = 1: 21 MHz 110b = If DCORSEL = 0: 8 MHz; If DCORSEL = 1: 24 MHz 111b = If DCORSEL = 0: Reserved. Defaults to 8 MHz. It is not recommended to use this setting; If DCORSEL = 1: Reserved. Defaults to 24 MHz. It is not recommended to use this setting
0	Reserved	R	0h	Reserved. Always reads as 0.

Registro CSCTL2

15	14	13	12	11	10	9	8
Reserved					SELA		
r0	r0	r0	r0	r0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
Reserved	SELS			Reserved	SELM		
r0	rw-0	rw-1	rw-1	r0	rw-0	rw-1	rw-1

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved. Always reads as 0.
10-8	SELA	RW	0h	Selects the ACLK source 000b = LFXTCLK when LFXT available, otherwise VLOCLK. 001b = VLOCLK 010b = LFMODCLK 011b = Reserved. Defaults to LFMODCLK. Not recommended for use to ensure future compatibility. 100b = Reserved. Defaults to LFMODCLK. Not recommended for use to ensure future compatibility. 101b = Reserved. Defaults to LFMODCLK. Not recommended for use to ensure future compatibility. 110b = Reserved. Defaults to LFMODCLK. Not recommended for use to ensure future compatibility. 111b = Reserved. Defaults to LFMODCLK. Not recommended for use to ensure future compatibility.
7	Reserved	R	0h	Reserved. Always reads as 0.
6-4	SELS	RW	3h	Selects the SMCLK source 000b = LFXTCLK when LFXT available, otherwise VLOCLK. 001b = VLOCLK 010b = LFMODCLK 011b = DCOCLK 100b = MODCLK 101b = HFXTCLK when HFXT available, otherwise DCOCLK. 110b = Reserved. Defaults to HFXTCLK. Not recommended for use to ensure future compatibility. 111b = Reserved. Defaults to HFXTCLK. Not recommended for use to ensure future compatibility.
3	Reserved	R	0h	Reserved. Always reads as 0.
2-0	SELM	RW	3h	Selects the MCLK source 000b = LFXTCLK when LFXT available, otherwise VLOCLK 001b = VLOCLK 010b = LFMODCLK 011b = DCOCLK 100b = MODCLK 101b = HFXTCLK when HFXT available, otherwise DCOCLK 110b = Reserved. Defaults to HFXTCLK. Not recommended for use to ensure future compatibility. 111b = Reserved. Defaults to HFXTCLK. Not recommended for use to ensure future compatibility.

Registro CSCTL3

15	14	13	12	11	10	9	8
Reserved					DIVA		
r0	r0	r0	r0	r0	rw-0	rw-0	rw-0
7	6	5	4	3	2	1	0
Reserved	DIVS			Reserved	DIVM		
r0	rw-0	rw-1	rw-1	r0	rw-0	rw-1	rw-1

Bit	Field	Type	Reset	Description
15-11	Reserved	R	0h	Reserved. Always reads as 0.
10-8	DIVA	RW	0h	ACLK source divider. Divides the frequency of the ACLK clock source. 000b = /1 001b = /2 010b = /4 011b = /8 100b = /16 101b = /32 110b = Reserved. Defaults to /32. Not recommended for use to ensure future compatibility. 111b = Reserved. Defaults to /32. Not recommended for use to ensure future compatibility.
7	Reserved	R	0h	Reserved. Always reads as 0.
6-4	DIVS	RW	3h	SMCLK source divider. Divides the frequency of the SMCLK clock source. 000b = /1 001b = /2 010b = /4 011b = /8 100b = /16 101b = /32 110b = Reserved. Defaults to /32. Not recommended for use to ensure future compatibility. 111b = Reserved. Defaults to /32. Not recommended for use to ensure future compatibility.
3	Reserved	R	0h	Reserved. Always reads as 0.
2-0	DIVM	RW	3h	MCLK source divider. Divides the frequency of the MCLK clock source. 000b = /1 001b = /2 010b = /4 011b = /8 100b = /16 101b = /32 110b = Reserved. Defaults to /32. Not recommended for use to ensure future compatibility. 111b = Reserved. Defaults to /32. Not recommended for use to ensure future compatibility.

Registro CSCTL4

15	14	13	12	11	10	9	8
HFXTDRIVE		Reserved	HFXTBYPASS	HFFREQ		Reserved	HFXTOFF
rw-1	rw-1	r0	rw-0	rw-1	rw-1	r0	rw-1
7	6	5	4	3	2	1	0
LFXTDRIVE		Reserved	LFXTBYPASS	VLOOFF	Reserved	SMCLKOFF	LFXTOFF
rw-1	rw-1	rw-0	rw-0	rw-1	r0	rw-0	rw-1

Bit	Field	Type	Reset	Description
0	LFXTOFF	RW	1h	LFXT off. This bit turns off the LFXT. 0b = LFXT is on if LFXT is selected through the port selection and LFXT is not in bypass mode of operation 1b = LFXT is off if it is not used as a source for ACLK, MCLK, or SMCLK

Anexo III. Registros del timer

Registro TAxCTL

15	14	13	12	11	10	9	8
Reserved						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID		MC		Reserved	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

Bit	Field	Type	Reset	Description
15-10	Reserved	RW	0h	Reserved
9-8	TASSEL	RW	0h	Timer_A clock source select 00b = TAxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	Input divider. These bits along with the TAIDEX bits select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	Mode control. Setting MC = 00h when Timer_A is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TAxCCR0 10b = Continuous mode: Timer counts up to 0FFFFh 11b = Up/down mode: Timer counts up to TAxCCR0 then down to 0000h
3	Reserved	RW	0h	Reserved
2	TACLR	RW	0h	Timer_A clear. Setting this bit clears TAR, the clock divider logic (the divider setting remains unchanged), and the count direction. The TACLR bit is automatically reset and is always read as zero.
1	TAIE	RW	0h	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
0	TAIFG	RW	0h	Timer_A interrupt flag 0b = No interrupt pending 1b = Interrupt pending

Anexo IV. Programa completo

```
#include <msp430.h>
#include <stdio.h>
#include <stdint.h>


int main(void)
 {
    WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer


    // Configure GPIO
    // Configure GPIOs to it's lowest power state
      P1OUT = 0;                                // All P1.x reset
      P1DIR = 0xFF;                             // All P1.x outputs
      P2OUT = 0;                                // All P2.x reset
      P2DIR = 0xFF;                             // All P2.x outputs
      P3OUT = 0;                                // All P3.x reset
      P3DIR = 0xFF;                             // All P3.x outputs
      P4OUT = 0;                                // All P4.x reset
      P4DIR = 0xFF;                             // All P4.x outputs
      PJOUT = 0;                                // All PJ.x reset
      PJDIR = 0xFFFF;                           // All PJ.x outputs

     PJSEL0 = BIT4 | BIT5;           // For XT1

     P1DIR &= ~(BIT3 | BIT5 | BIT7 | BIT6 | BIT2);  //
P1.3/P1.5/P1.7/P1.6/P1.2 inputs

     // Disable the GPIO power-on default high-impedance mode to activate
      // previously configured port settings
      PM5CTL0 &= ~LOCKLPM5;

      if (P1IN==0x08){
          // Clock System Setup
          CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
          CSCTL1 = DCOFSEL_7;                     // Set DCO to 8MHz
          CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set SMCLK
= MCLK = DCO ACLK = VLOCLK
          CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to 1
          CSCTL4 &= ~LFXTOFF;

          do
            {
              CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault flag
              SFRIFG1 &= ~OFIFG;
            } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

          CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)

          TA1CTL = TASSEL__SMCLK | ID_0 | MC__STOP ;  // Use SMCLK as clock
source, Stop timer.

          __bis_SR_register(LPM1_bits);  //For SMCLK as main source
```

```
      }
      if (P1IN==0x20){
            // Clock System Setup
            CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
            CSCTL1 = DCOFSEL_7;                      // Set DCO to 8MHz
            CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set
SMCLK = MCLK = DCO ACLK = VLOCLK
            CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to
1
            CSCTL4 &= ~LFXTOFF;

            do
              {
                CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault
flag
                SFRIFG1 &= ~OFIFG;
              } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

            CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)

            TA1CTL = TASSEL__SMCLK | ID_1 | MC__STOP ;  // Use SMCLK as
clock source, Stop timer.

            __bis_SR_register(LPM1_bits);  //For SMCLK as main source
      }
      if(P1IN==0x28){
          // Clock System Setup
          CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
          CSCTL1 = DCOFSEL_7;                      // Set DCO to 8MHz
          CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set SMCLK
= MCLK = DCO ACLK = VLOCLK
          CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to 1
          CSCTL4 &= ~LFXTOFF;

          do
            {
              CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault flag
              SFRIFG1 &= ~OFIFG;
            } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

          CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)


          TA1CTL = TASSEL__SMCLK | ID_2 | MC__STOP ;  // Use SMCLK as clock
source, Stop timer.

          __bis_SR_register(LPM1_bits);  //For SMCLK as main source
      }
      if(P1IN==0x80){
          // Clock System Setup
          CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
          CSCTL1 = DCOFSEL_7;                      // Set DCO to 8MHz
```

```
          CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set SMCLK
= MCLK = DCO ACLK = VLOCLK
          CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to 1
          CSCTL4 &= ~LFXTOFF;

          do
            {
              CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault flag
              SFRIFG1 &= ~OFIFG;
            } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

          CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)


          TA1CTL = TASSEL__SMCLK | ID_3 | MC__STOP ;  // Use SMCLK as clock
source, Stop timer.

          __bis_SR_register(LPM1_bits);  //For SMCLK as main source
      }
      if(P1IN==0x88){
        // Clock System Setup
              CSCTL0_H = CSKEY >> 8;                    // Unlock CS
registers
              CSCTL1 = DCOFSEL_0;                                       // Set
DCO to 1MHz
              CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set
SMCLK = MCLK = DCO ACLK = VLOCLK
              CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers
to 1
              CSCTL4 &= ~LFXTOFF;

              do
                {
                  CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault
flag
                  SFRIFG1 &= ~OFIFG;
                } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

              CSCTL0_H = 0x00;                          // Lock CS module
(use byte mode to upper byte)

              TA1CTL = TASSEL__SMCLK | ID_0 | MC__STOP ;  // Use SMCLK as
clock source, Stop timer.

              __bis_SR_register(LPM1_bits);  //For SMCLK as main source
      }
      if(P1IN==0xA0){
        // Clock System Setup
              CSCTL0_H = CSKEY >> 8;                    // Unlock CS
registers
              CSCTL1 = DCOFSEL_0;                                       // Set
DCO to 1MHz
              CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set
SMCLK = MCLK = DCO ACLK = VLOCLK
              CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers
to 1
```

```
            CSCTL4 &= ~LFXTOFF;

            do
              {
                CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault
flag
                SFRIFG1 &= ~OFIFG;
              } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

            CSCTL0_H = 0x00;                          // Lock CS module
(use byte mode to upper byte)

            TA1CTL = TASSEL__SMCLK | ID_1 | MC__STOP ;  // Use SMCLK as
clock source, Stop timer.

            __bis_SR_register(LPM1_bits);  //For SMCLK as main source
      }
      if(P1IN==0xA8){
        // Clock System Setup
            CSCTL0_H = CSKEY >> 8;                    // Unlock CS
registers
            CSCTL1 = DCOFSEL_0;                                       // Set
DCO to 1MHz
            CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set
SMCLK = MCLK = DCO ACLK = VLOCLK
            CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers
to 1
            CSCTL4 &= ~LFXTOFF;

            do
              {
                CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault
flag
                SFRIFG1 &= ~OFIFG;
              } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

            CSCTL0_H = 0x00;                          // Lock CS module
(use byte mode to upper byte)

            TA1CTL = TASSEL__SMCLK | ID_2 | MC__STOP ;  // Use SMCLK as
clock source, Stop timer.

            __bis_SR_register(LPM1_bits);  //For SMCLK as main source
      }
      if(P1IN==0x40){
        // Clock System Setup
            CSCTL0_H = CSKEY >> 8;                    // Unlock CS
registers
            CSCTL1 = DCOFSEL_0;                                       // Set
DCO to 1MHz
            CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set
SMCLK = MCLK = DCO ACLK = VLOCLK
            CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers
to 1
            CSCTL4 &= ~LFXTOFF;

            do
```

```
                {
                  CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault
flag
                  SFRIFG1 &= ~OFIFG;
                } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

              CSCTL0_H = 0x00;                          // Lock CS module
(use byte mode to upper byte)

              TA1CTL = TASSEL__SMCLK | ID_3 | MC__STOP ;  // Use SMCLK as
clock source, Stop timer.

              __bis_SR_register(LPM1_bits);  //For SMCLK as main source
      }
      if(P1IN==0x48){
               // Clock System Setup
                    CSCTL0_H = CSKEY >> 8;                    // Unlock CS
registers
                    CSCTL1 = DCOFSEL_0;
      // Set DCO to 1MHz
                    CSCTL2 = SELA__LFXTCLK | SELS__DCOCLK | SELM__DCOCLK;  //
Set SMCLK = MCLK = DCO ACLK = LFXT
                    CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all
dividers to 1
                    CSCTL4 &= ~LFXTOFF;

                    do
                      {
                        CSCTL5 &= ~LFXTOFFG;                    // Clear XT1
fault flag
                        SFRIFG1 &= ~OFIFG;
                      } while (SFRIFG1 & OFIFG);                // Test
oscillator fault flag

                    CSCTL0_H = 0x00;                          // Lock CS
module (use byte mode to upper byte)

                    TA1CTL = TASSEL__ACLK | ID_0 | MC__STOP ;  // Use ACLK as
clock source, Stop timer.

                    __bis_SR_register(LPM3_bits);  //For ACLK as main source
      }
      if(P1IN==0x60){
               // Clock System Setup
                    CSCTL0_H = CSKEY >> 8;                    // Unlock CS
registers
                    CSCTL1 = DCOFSEL_0;
      // Set DCO to 1MHz
                    CSCTL2 = SELA__LFXTCLK | SELS__DCOCLK | SELM__DCOCLK;  //
Set SMCLK = MCLK = DCO ACLK = LFXT
                    CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all
dividers to 1
                    CSCTL4 &= ~LFXTOFF;

                    do
                      {
                        CSCTL5 &= ~LFXTOFFG;                    // Clear XT1
fault flag
```

```
                        SFRIFG1 &= ~OFIFG;
                      } while (SFRIFG1 & OFIFG);                // Test
oscillator fault flag

                    CSCTL0_H = 0x00;                          // Lock CS
module (use byte mode to upper byte)

                    TA1CTL = TASSEL__ACLK | ID_1 | MC__STOP ;  // Use ACLK as
clock source, Stop timer.

                    __bis_SR_register(LPM3_bits);  //For ACLK as main source
      }
      if(P1IN==0x68){
               // Clock System Setup
                    CSCTL0_H = CSKEY >> 8;                    // Unlock CS
registers
                    CSCTL1 = DCOFSEL_0;
      // Set DCO to 1MHz
                    CSCTL2 = SELA__LFXTCLK | SELS__DCOCLK | SELM__DCOCLK;  //
Set SMCLK = MCLK = DCO ACLK = LFXT
                    CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all
dividers to 1
                    CSCTL4 &= ~LFXTOFF;

                    do
                      {
                        CSCTL5 &= ~LFXTOFFG;                    // Clear XT1
fault flag
                        SFRIFG1 &= ~OFIFG;
                      } while (SFRIFG1 & OFIFG);                // Test
oscillator fault flag

                    CSCTL0_H = 0x00;                          // Lock CS
module (use byte mode to upper byte)

                    TA1CTL = TASSEL__ACLK | ID_2 | MC__STOP ;  // Use ACLK as
clock source, Stop timer.

                    __bis_SR_register(LPM3_bits);  //For ACLK as main source
      }
      if(P1IN==0xC0){
               // Clock System Setup
                    CSCTL0_H = CSKEY >> 8;                    // Unlock CS
registers
                    CSCTL1 = DCOFSEL_0;
      // Set DCO to 1MHz
                    CSCTL2 = SELA__LFXTCLK | SELS__DCOCLK | SELM__DCOCLK;  //
Set SMCLK = MCLK = DCO ACLK = LFXT
                    CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all
dividers to 1
                    CSCTL4 &= ~LFXTOFF;

                    do
                      {
                        CSCTL5 &= ~LFXTOFFG;                    // Clear XT1
fault flag
                        SFRIFG1 &= ~OFIFG;
                      } while (SFRIFG1 & OFIFG);                // Test
oscillator fault flag
```

```
                    CSCTL0_H = 0x00;                          // Lock CS
module (use byte mode to upper byte)

                    TA1CTL = TASSEL__ACLK | ID_3 | MC__STOP ;  // Use ACLK as
clock source, Stop timer.

                    __bis_SR_register(LPM3_bits);  //For ACLK as main source
      }
      if(P1IN==0xC8){
          // Clock System Setup
          CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
          CSCTL1 = DCOFSEL_7;                      // Set DCO to 8MHz
          CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set SMCLK
= MCLK = DCO ACLK = VLOCLK
          CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to 1
          CSCTL4 &= ~LFXTOFF;

          do
            {
              CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault flag
              SFRIFG1 &= ~OFIFG;
            } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

          CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)

          TA1CTL = TASSEL__SMCLK | ID_0 | MC__CONTINUOUS ;  // Use SMCLK as
clock source, Start timer in continuous mode.
          __bis_SR_register(LPM1_bits);  //For SMCLK as main source
      }
      if(P1IN==0xE0){
          // Clock System Setup
          CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
          CSCTL1 = DCOFSEL_7;                      // Set DCO to 8MHz
          CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set SMCLK
= MCLK = DCO ACLK = VLOCLK
          CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to 1
          CSCTL4 &= ~LFXTOFF;

          do
            {
              CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault flag
              SFRIFG1 &= ~OFIFG;
            } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

          CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)

          TA1CTL = TASSEL__SMCLK | ID_1 | MC__CONTINUOUS ;  // Use SMCLK as
clock source, Start timer in continuous mode.
          __bis_SR_register(LPM1_bits);  //For SMCLK as main source
      }
      if(P1IN==0xE8){
          // Clock System Setup
          CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
          CSCTL1 = DCOFSEL_7;                      // Set DCO to 8MHz
```

```
          CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set SMCLK
= MCLK = DCO ACLK = VLOCLK
          CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to 1
          CSCTL4 &= ~LFXTOFF;

          do
            {
              CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault flag
              SFRIFG1 &= ~OFIFG;
            } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

          CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)

          TA1CTL = TASSEL__SMCLK | ID_2 | MC__CONTINUOUS ;  // Use SMCLK as
clock source, Start timer in continuous mode.
          __bis_SR_register(LPM1_bits);  //For SMCLK as main source
      }
      if(P1IN==0x04){
          // Clock System Setup
          CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
          CSCTL1 = DCOFSEL_7;                     // Set DCO to 8MHz
          CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set SMCLK
= MCLK = DCO ACLK = VLOCLK
          CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to 1
          CSCTL4 &= ~LFXTOFF;

          do
            {
              CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault flag
              SFRIFG1 &= ~OFIFG;
            } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

          CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)

          TA1CTL = TASSEL__SMCLK | ID_3 | MC__CONTINUOUS ;  // Use SMCLK as
clock source, Start timer in continuous mode.
          __bis_SR_register(LPM1_bits);  //For SMCLK as main source
      }
      if(P1IN==0x0C){
            // Clock System Setup
            CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
            CSCTL1 = DCOFSEL_0;                                 // Set DCO
to 1MHz
            CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set
SMCLK = MCLK = DCO ACLK = VLOCLK
            CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to
1
            CSCTL4 &= ~LFXTOFF;

            do
              {
                CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault
flag
                SFRIFG1 &= ~OFIFG;
```

```
              } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

            CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)

            TA1CTL = TASSEL__SMCLK | ID_0 | MC__CONTINUOUS ;  // Use SMCLK
as clock source, Start timer in continuous mode.

            __bis_SR_register(LPM1_bits);  //For SMCLK as main source
      }
      if(P1IN==0x24){
            // Clock System Setup
            CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
            CSCTL1 = DCOFSEL_0;                                // Set DCO
to 1MHz
            CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set
SMCLK = MCLK = DCO ACLK = VLOCLK
            CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to
1
            CSCTL4 &= ~LFXTOFF;

            do
              {
                CSCTL5 &= ~LFXTOFFG;                  // Clear XT1 fault
flag
                SFRIFG1 &= ~OFIFG;
              } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

            CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)

            TA1CTL = TASSEL__SMCLK | ID_1 | MC__CONTINUOUS ;  // Use SMCLK
as clock source, Start timer in continuous mode.

            __bis_SR_register(LPM1_bits);  //For SMCLK as main source
      }
      if(P1IN==0x2C){
            // Clock System Setup
            CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
            CSCTL1 = DCOFSEL_0;                                // Set DCO
to 1MHz
            CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set
SMCLK = MCLK = DCO ACLK = VLOCLK
            CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to
1
            CSCTL4 &= ~LFXTOFF;

            do
              {
                CSCTL5 &= ~LFXTOFFG;                  // Clear XT1 fault
flag
                SFRIFG1 &= ~OFIFG;
              } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

            CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)
```

```
            TA1CTL = TASSEL__SMCLK | ID_2 | MC__CONTINUOUS ;  // Use SMCLK
as clock source, Start timer in continuous mode.

            __bis_SR_register(LPM1_bits);  //For SMCLK as main source
      }
      if(P1IN==0x84){
            // Clock System Setup
            CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
            CSCTL1 = DCOFSEL_0;                                 // Set DCO
to 1MHz
            CSCTL2 = SELA__VLOCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set
SMCLK = MCLK = DCO ACLK = VLOCLK
            CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to
1
            CSCTL4 &= ~LFXTOFF;

            do
              {
                CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault
flag
                SFRIFG1 &= ~OFIFG;
              } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

            CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)

            TA1CTL = TASSEL__SMCLK | ID_3 | MC__CONTINUOUS ;  // Use SMCLK
as clock source, Start timer in continuous mode.

            __bis_SR_register(LPM1_bits);  //For SMCLK as main source
      }
      if(P1IN==0x8C){
            // Clock System Setup
            CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
            CSCTL1 = DCOFSEL_0;                                 // Set DCO
to 1MHz
            CSCTL2 = SELA__LFXTCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set
SMCLK = MCLK = DCO ACLK = LFXT
            CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to
1
            CSCTL4 &= ~LFXTOFF;

            do
              {
                CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault
flag
                SFRIFG1 &= ~OFIFG;
              } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

            CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)

            TA1CTL = TASSEL__ACLK | ID_0 | MC__CONTINUOUS | TACLR | TAIE;
//Use ACLK as clock source, Start timer in continuous mode.

            __bis_SR_register(LPM3_bits);  //For ACLK as main source
```

```
        }
        if(P1IN==0xA4){
                // Clock System Setup
                CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
                CSCTL1 = DCOFSEL_0;                                 // Set DCO
to 1MHz
                CSCTL2 = SELA__LFXTCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set
SMCLK = MCLK = DCO ACLK = LFXT
                CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to
1
                CSCTL4 &= ~LFXTOFF;

                do
                  {
                    CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault
flag
                    SFRIFG1 &= ~OFIFG;
                  } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

                CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)

                TA1CTL = TASSEL__ACLK | ID_1 | MC__CONTINUOUS | TACLR | TAIE;
//Use ACLK as clock source, Start timer in continuous mode.

                __bis_SR_register(LPM3_bits);  //For ACLK as main source
        }
        if(P1IN==0xAC){
                // Clock System Setup
                CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
                CSCTL1 = DCOFSEL_0;                                 // Set DCO
to 1MHz
                CSCTL2 = SELA__LFXTCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set
SMCLK = MCLK = DCO ACLK = LFXT
                CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to
1
                CSCTL4 &= ~LFXTOFF;

                do
                  {
                    CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault
flag
                    SFRIFG1 &= ~OFIFG;
                  } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

                CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)

                TA1CTL = TASSEL__ACLK | ID_2 | MC__CONTINUOUS | TACLR | TAIE;
//Use ACLK as clock source, Start timer in continuous mode.

                __bis_SR_register(LPM3_bits);  //For ACLK as main source
        }
        if(P1IN==0x44){
                // Clock System Setup
                CSCTL0_H = CSKEY >> 8;                    // Unlock CS registers
```

```
            CSCTL1 = DCOFSEL_0;                                  // Set DCO
to 1MHz
            CSCTL2 = SELA__LFXTCLK | SELS__DCOCLK | SELM__DCOCLK;  // Set
SMCLK = MCLK = DCO ACLK = LFXT
            CSCTL3 = DIVA__1 | DIVS__1 | DIVM__1;     // Set all dividers to
1
            CSCTL4 &= ~LFXTOFF;

            do
              {
                CSCTL5 &= ~LFXTOFFG;                    // Clear XT1 fault
flag
                SFRIFG1 &= ~OFIFG;
              } while (SFRIFG1 & OFIFG);                // Test oscillator
fault flag

            CSCTL0_H = 0x00;                          // Lock CS module (use
byte mode to upper byte)

            TA1CTL = TASSEL__ACLK | ID_3 | MC__CONTINUOUS | TACLR | TAIE;
//Use ACLK as clock source, Start timer in continuous mode.

            __bis_SR_register(LPM3_bits);  //For ACLK as main source
      }
      while(1);

      return 0;
      }
```