INSTITUTO TECNOLÓGICO SUPERIOR DE SAN ANDRÉS TUXTLA (I.T.S.S.A.T.)

DIVISIÓN INGENIERÍA MECATRÓNICA

IMCT-2010-229

MICROCONTROLADORES

DOCENTE

Dr. José Ángel Nieves Vázquez

711-A

PERÍODO AGOSTO-DICIEMBRE 2025

UNIDAD V

PROGRAMACIÓN DE PERIFÉRICOS DEL MICROCONTROLADOR

INVESTIGACIÓN

PRESENTAN

Juan José Jiménez Reyes	*221U0541*
Juan José Marcial Fiscal	*221U0547*
Pólito Cerón Miguel de Jesús	*221U0552*
Quino Caixba Perla Joselin	*221U0555*
Teoba Herrera Rocio	*221U0562*

SAN ANDRÉS TUXTLA, VER. A 01 DE DICIEMBRE DE 2025

EQUIPO 3

INSTITUTO TECNOLÓGICO SUPERIOR DE SAN ANDRÉS TUXTLA

ÍNDICE

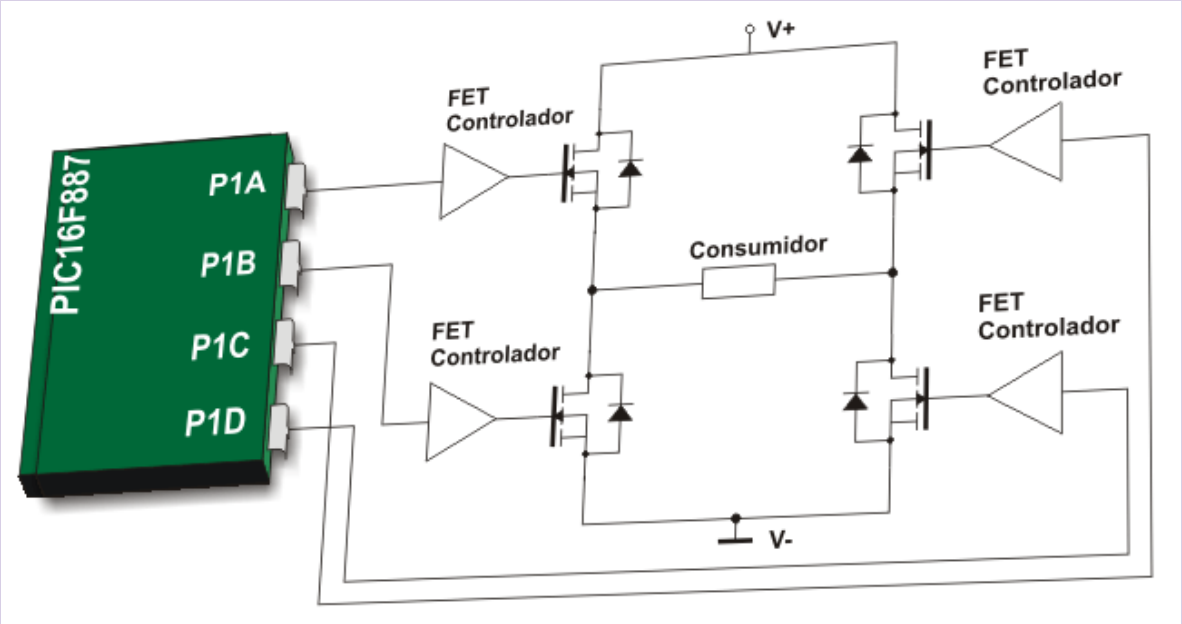
INTRODUCCIÓN

En el ámbito de los sistemas embebidos, la eficiencia de un diseño no se mide solo por la velocidad de procesamiento, sino por la capacidad del microcontrolador para gestionar eventos temporales y señales externas de manera autónoma. La presente investigación aborda la unidad temática 5: **"Programación del módulo CCP del microcontrolador"**, un periférico multifuncional que integra las capacidades de **C**aptura, **C**omparación y **P**WM (Modulación por Ancho de Pulso).

El estudio comienza con la **descripción del módulo (5.1)**, detallando su arquitectura interna y su dependencia crítica de los recursos de temporización (Timers), lo cual sienta las bases para comprender sus tres modos operativos. Se analizará profundamente la **Configuración como Comparador (5.2)**, esencial para la generación de eventos temporales precisos, y la **Configuración como Captura (5.3)**, que permite medir intervalos de señales externas con exactitud de hardware. Asimismo, se examina el modo **PWM (5.4)**, una herramienta estándar en la industria para el control de potencia y motores, donde la relación entre frecuencia y ciclo de trabajo es vital. Finalmente, el informe integra estos conceptos en el **Desarrollo de aplicaciones (5.5)**, demostrando cómo la correcta programación de registros como CCPxCON y CCPRx permite descargar a la CPU de tareas repetitivas, optimizando el rendimiento global del sistema.

5.1 DESCRIPCIÓN DEL MÓDULO CCP.

El módulo CCP se caracteriza por su versatilidad, lograda a través de una arquitectura configurable que comparte recursos físicos para realizar tres funciones distintas. En la mayoría de los dispositivos de gama media, como el PIC16F887 o el PIC16F877A, se encuentran dos instancias de este módulo: CCP1 y CCP2. Aunque funcionalmente idénticos en su operación básica, su integración con el resto del microcontrolador presenta matices importantes.

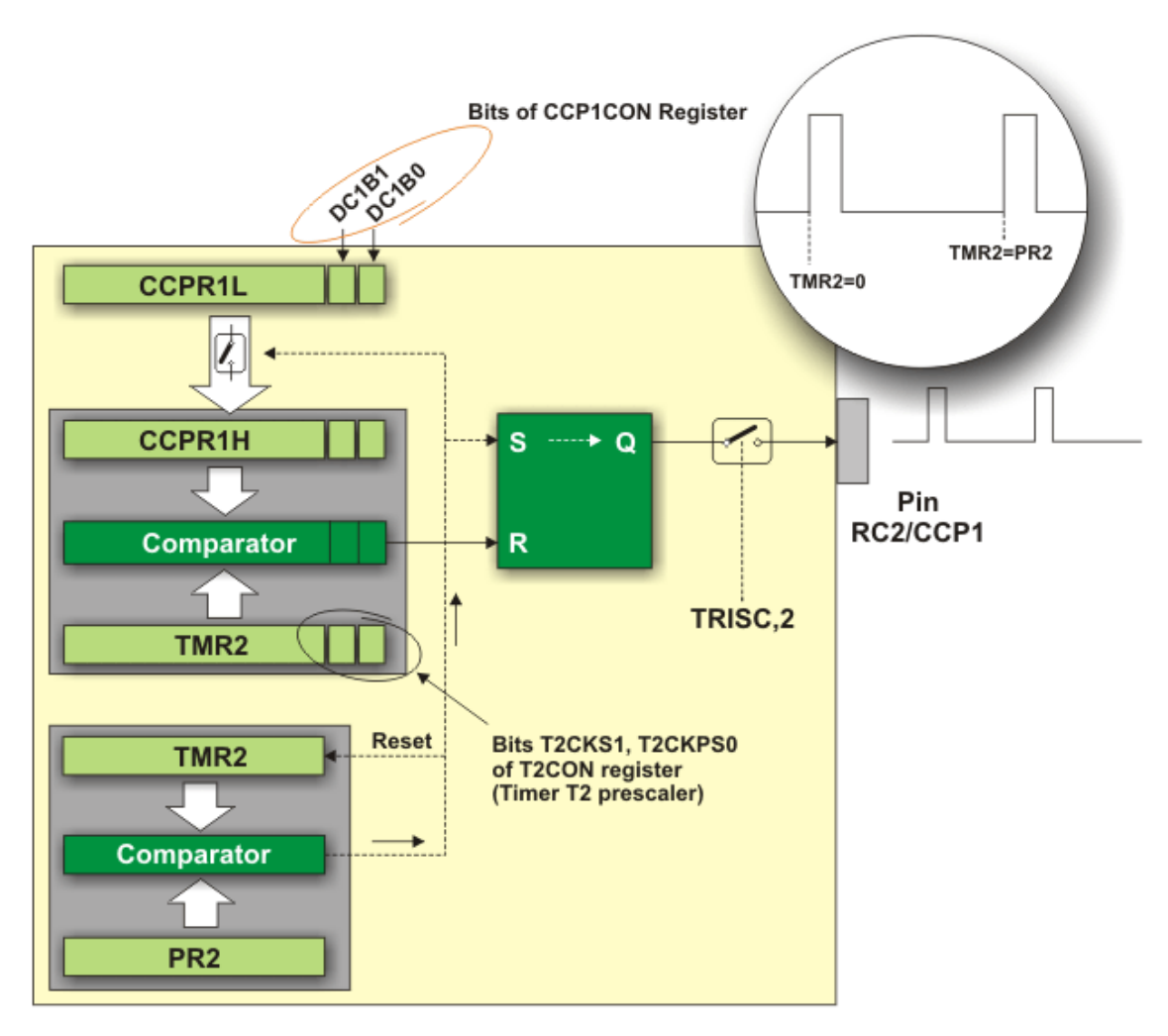


El Núcleo del Módulo: Registro de 16 bits

En el centro de la arquitectura del módulo CCP reside un registro de datos de 16 bits. Dado que los microcontroladores PIC de 8 bits (como las series PIC10, PIC12, PIC16 y PIC18) operan con un bus de datos de 8 bits, este registro de 16 bits se divide físicamente en dos registros de funciones especiales (SFR) de 8 bits cada uno:

- **CCPRxL (CCP Register Low):** Contiene los 8 bits menos significativos.

- **CCPRxH (CCP Register High):** Contiene los 8 bits más significativos.



Donde 'x' representa el número del módulo (1 o 2).

La función de este par de registros es polimórfica; su comportamiento cambia radicalmente según el modo seleccionado en el registro de control CCPxCON :

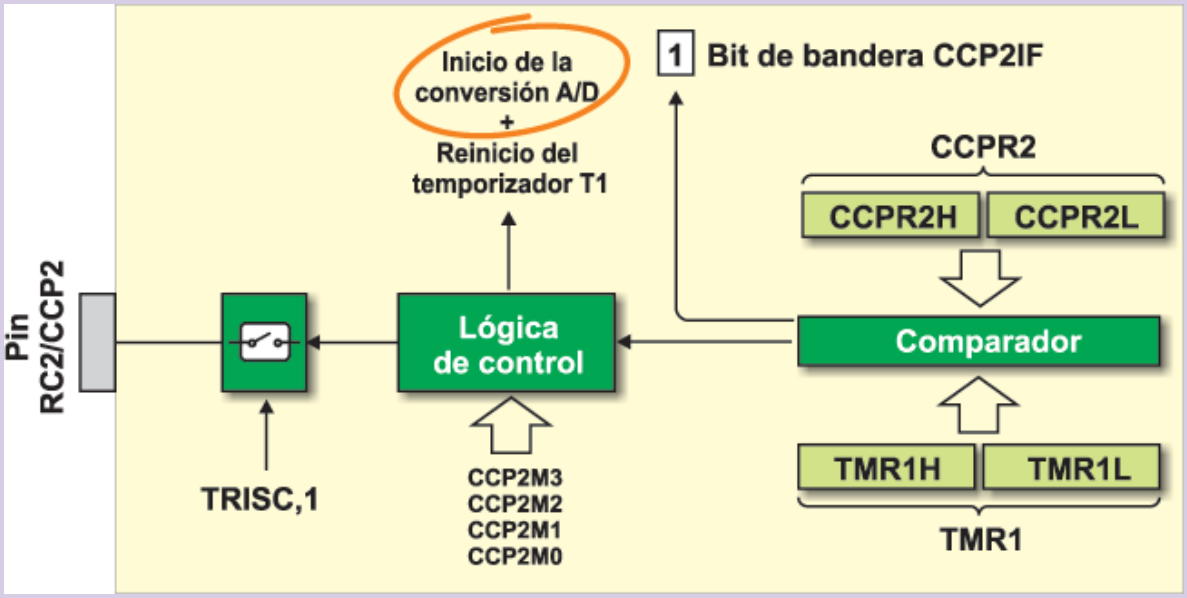
1. **En Modo Captura:** El par CCPRxH:CCPRxL actúa como un **registro de destino**. Está conectado internamente al bus de datos del Timer1 (o Timer3 en PIC18). Cuando un evento de disparo ocurre en el pin físico, el hardware transfiere instantáneamente el valor del temporizador a estos registros, "capturando" el tiempo del evento.

2. **En Modo Comparación:** El par actúa como un **registro de comparación**. El usuario carga un valor de tiempo objetivo en ellos. Un comparador digital de hardware monitorea constantemente la igualdad entre este registro y el temporizador del sistema.
3. **En Modo PWM:** La arquitectura cambia. El registro de 16 bits se utiliza para almacenar el **ciclo de trabajo** (Duty Cycle). Sin embargo, aquí ocurre una particularidad: CCPRxL almacena los 8 bits más significativos del ciclo de trabajo, mientras que CCPRxH se convierte en un registro de "sombra" (shadow register) o buffer de lectura solamente, gestionado por el hardware para evitar glitches durante las transiciones de pulso. Los 2 bits menos significativos del ciclo de trabajo se reubican en el registro de control CCPxCON, logrando así una resolución de 10 bits.

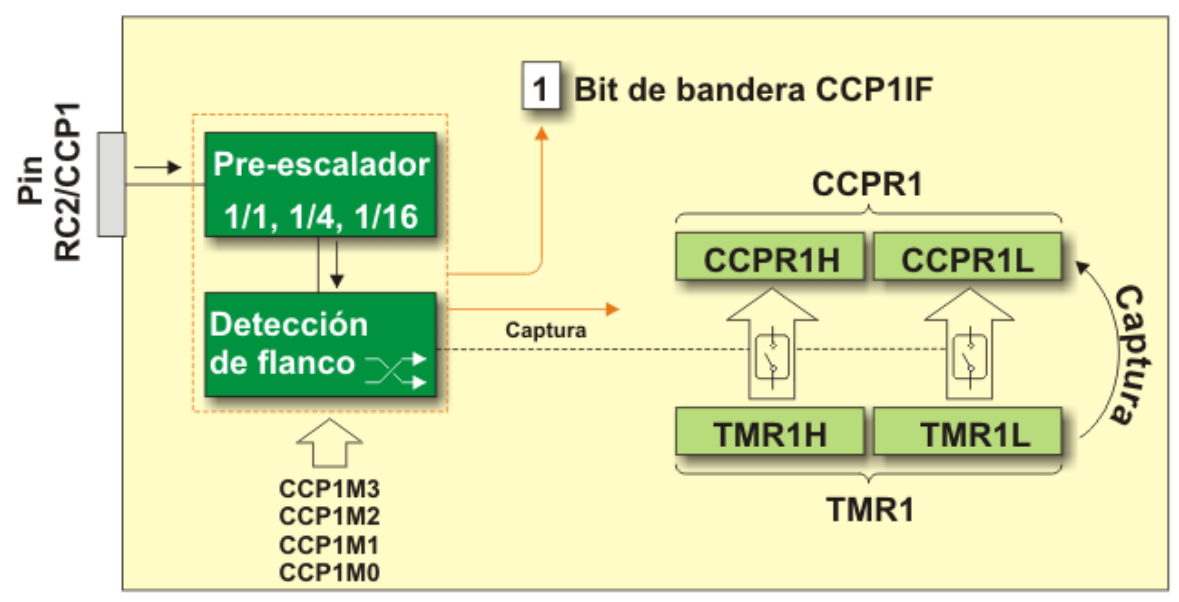
Pines de Entrada y Salida (Multiplexación)

El módulo CCP interactúa con el mundo exterior a través de pines físicos. Debido a la limitación de pines en los encapsulados, estos pines están multiplexados con otras funciones, generalmente puertos de E/S digitales.

- **CCP1:** Típicamente asignado al pin **RC2** (Puerto C, bit 2).



- **CCP2:** Típicamente asignado al pin **RC1** (Puerto C, bit 1).



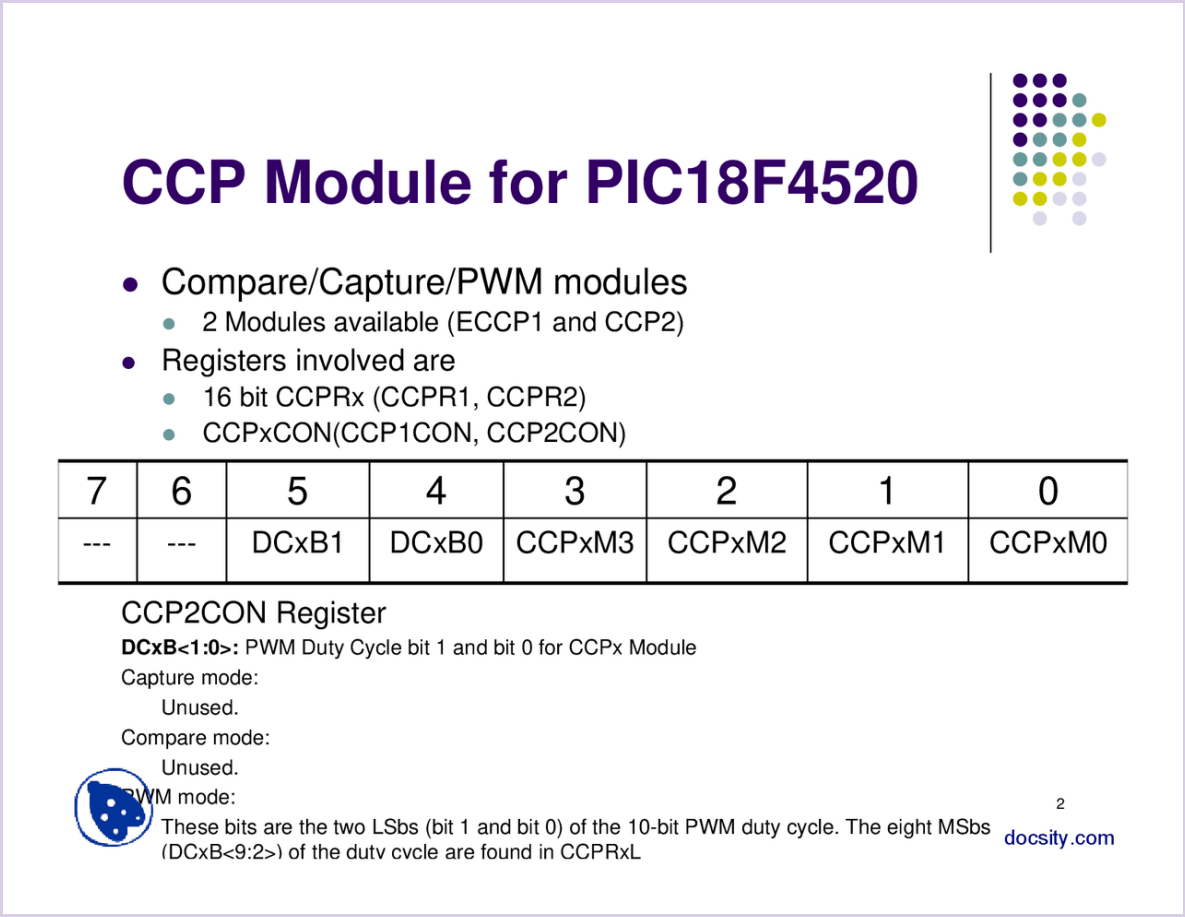
Es crucial notar que en ciertos dispositivos, como el PIC16F887 o PIC18F4550, el pin del módulo CCP2 puede ser reconfigurado o "movido" a otro pin (como **RB3**) mediante bits de configuración (Configuration Bits o Fuses) al momento de programar el chip. Esto otorga flexibilidad al diseñador de PCB (Printed Circuit Board) para facilitar el enrutamiento de pistas.

Consideración de Hardware: Para que el módulo CCP funcione, el registro de dirección de datos (TRIS) correspondiente al pin debe configurarse correctamente.

- En **Modo Captura**, el pin debe ser configurado como **Entrada** (TRIS = 1).
- En **Modos Comparación y PWM**, el pin debe ser configurado como **Salida** (TRIS = 0). Si se configura como entrada en estos modos, la operación lógica interna ocurrirá, pero la señal no será visible en el pin físico.

Análisis Detallado de los Registros de Control

La "Programación del módulo CCP" (Tema 5 del syllabus) se realiza fundamentalmente manipulando los bits de los registros de control. Un entendimiento superficial de estos bits lleva a implementaciones erróneas. A continuación, se presenta un análisis exhaustivo del registro CCPxCON.



CCP Module for PIC18F4520

- Compare/Capture/PWM modules
  - 2 Modules available (ECCP1 and CCP2)
- Registers involved are
  - 16 bit CCPRx (CCPR1, CCPR2)
  - CCPxCON(CCP1CON, CCP2CON)

7	6	5	4	3	2	1	0
---	---	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0

CCP2CON Register

DCxB<1:0>: PWM Duty Cycle bit 1 and bit 0 for CCPx Module

Capture mode:

Unused.

Compare mode:

Unused.

PWM mode:

These bits are the two LSbs (bit 1 and bit 0) of the 10-bit PWM duty cycle. The eight MSbs (DCxB<9:2>) of the duty cycle are found in CCPRxL

2

docsity.com

5.2 CONFIGURACIÓN Y PROGRAMACIÓN DEL MÓDULO CCP COMO COMPARADOR.

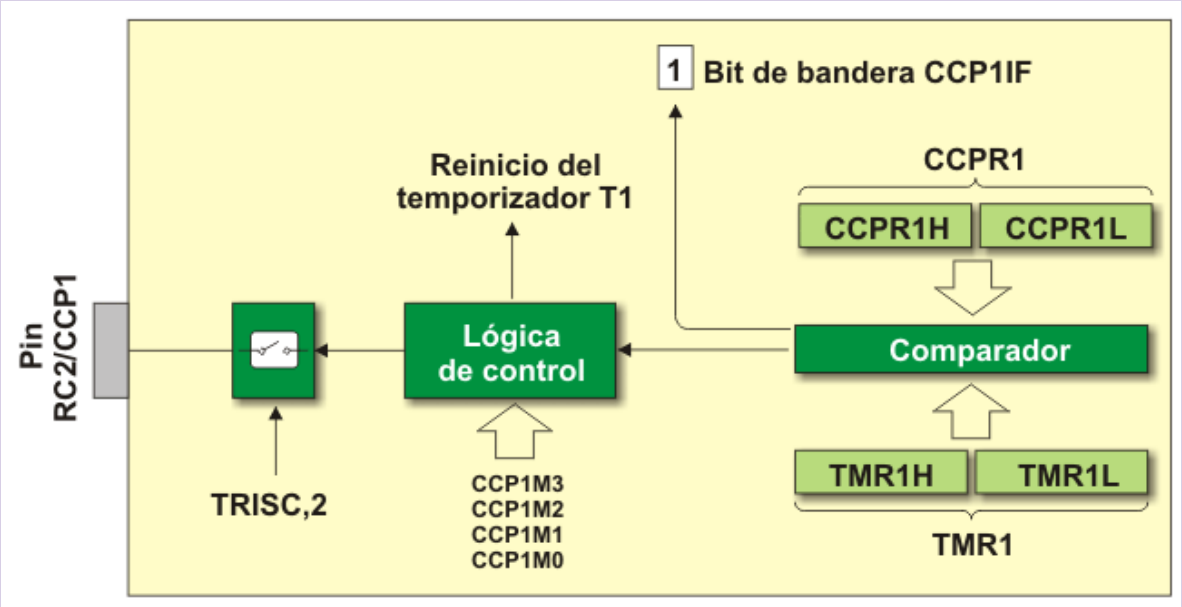
El módulo **CCP (Capture/Compare/PWM)** representa uno de los periféricos más versátiles y robustos incorporados en microcontroladores de arquitectura de 8 y 16 bits, especialmente en dispositivos de la familia PIC. Su finalidad es asistir en operaciones críticas de temporización y control que exigen una respuesta determinística y con baja latencia. Dentro de sus tres modos de operación, el **modo Comparador** destaca por permitir la generación de eventos basados en coincidencias entre un temporizador y un valor previamente configurado por el usuario, lo cual facilita la creación de secuencias temporales exactas, señales de control y mecanismos de sincronización entre módulos internos y externos del sistema.

Este modo opera apoyándose en un temporizador de 16 bits, comúnmente **TMR1**, el cual actúa como base de tiempo. Los valores de comparación se almacenan en el registro **CCPRx**, compuesto por CCPRxH y CCPRxL. Cuando la cuenta del temporizador coincide con el valor configurado en CCPRx, el hardware produce un evento inmediato, ya sea una interrupción, un cambio de estado en el pin CCP o la ejecución de una función especial. Debido a que el proceso ocurre sin intervención del CPU en el ciclo crítico, el comportamiento es altamente confiable y presenta un desfase prácticamente inexistente, lo que resulta esencial en procesos industriales o sistemas embebidos donde el tiempo real es un requerimiento.

Arquitectura Interna del Modo Comparador

Desde una perspectiva funcional, el comparador del CCP está integrado por:

- Un **latch** de 16 bits para almacenar el valor de referencia.
- Una lógica de comparación que analiza permanentemente el valor del temporizador.
- Un conjunto de multiplicadores lógicos que determinan la acción a ejecutar en el pin de salida.
- Una unidad de eventos especiales que puede interactuar con periféricos como el ADC o módulos de reset de temporizadores.



El diseño está optimizado para realizar la comparación en una sola operación lógica interna, evitando ciclos adicionales y minimizando la latencia. Esto significa que incluso en sistemas con carga elevada, el tiempo de respuesta del comparador permanece estable y confiable, aun cuando se ejecuten rutinas pesadas en paralelo. Cabe destacar que un error mínimo en la configuración del temporizador puede producir desalineación temporal, lo cual debe evitarse mediante cálculos detallados de preescalers y frecuencias.

Procedimiento Detallado de Configuración

1. Selección del Modo de Comparación

El primer paso consiste en escribir los bits **CCPxM3:CCPxM0** en el registro **CCPxCON**. Algunas modalidades típicas incluyen:

- **1001**: Forzar salida CCPx a nivel alto en coincidencia.
- **1000**: Forzar salida CCPx a nivel bajo en coincidencia.
- **1010**: Generar una interrupción sin modificar el pin.
- **1011**: Activar evento especial, útil para resetear TMR1 o iniciar una conversión ADC.

Una selección adecuada depende de la naturaleza del proceso a controlar. Por ejemplo, si se requiere generar un pulso preciso, se recomienda utilizar los modos que modifican el pin de salida directamente.

2. Configuración del Valor de Comparación

El valor preestablecido del evento se carga en:

- **CCPRxH** (bits altos)
- **CCPRxL** (bits bajos)

La escritura debe realizarse preferentemente con el temporizador detenido para evitar que un valor parcial produzca una coincidencia accidental durante la carga. Este detalle suele causar errores en principiantes, generando activaciones no deseadas del comparador.

3. Programación del Temporizador Asociado (TMR1)

El temporizador debe configurarse cuidadosamente:

- Fuente de reloj interna o externa.
- Preescaler (1:1, 1:2, 1:4, 1:8).
- Selección de sincronización en caso de usar reloj externo.
- Activación del oscilador especial T1OSC (si se requiere alta estabilidad).

El valor inicial del temporizador puede ser configurado en cero o en cualquier número específico si se requiere un desplazamiento temporal inicial.

4. Configuración del Pin CCPx

El pin físico asociado debe programarse como **salida digital** usando TRISx.
La incorrecta configuración de este pin es una de las causas más comunes de mal funcionamiento, ya que el hardware del CCP puede generar un evento correcto, pero si el pin está configurado como entrada, la salida no se reflejará externamente.

5. Habilitación de Interrupciones (Opcional)

Si se necesita reaccionar inmediatamente al evento del comparador, se activan:

- Bit de interrupción CCPxIE en PIE1.
- Bit de habilitación global GIE.
- Bit de prioridad si el microcontrolador lo soporta.

La rutina de servicio debe ser lo más breve posible para evitar retardo acumulado en sistemas de alta frecuencia.

Limitaciones Operativas

Aunque el módulo es altamente confiable, presenta algunas limitaciones:

- Si el temporizador TMR1 se comparte con otros procesos, puede generar conflictos de sincronización.
- La resolución está limitada por el reloj del temporizador; valores muy pequeños pueden no ser alcanzables.
- El hardware del pin CCP puede presentar incompatibilidades si comparte funciones con módulos como USB u osciladores secundarios.
- Valores grandes de preescaler disminuyen la precisión temporal fina.

En algunos casos es necesario recalcular dinámicamente el registro CCPRx dentro de la rutina de interrupción, especialmente si se desea generar sistemas periódicos con intervalos variables.

Ejemplo Práctico: Generación de un Pulso Preciso para Control de una Válvula Industrial

A continuación, se presenta un ejemplo típico donde el modo comparador es indispensable: **generar un pulso de 10 ms para accionar una válvula solenoide utilizada en un sistema de dosificación**.

Objetivo del Sistema

La válvula debe activarse durante 10 ms exactos cada vez que se recibe una señal de inicio. La precisión es crítica, ya que la cantidad de fluido dosificado depende directamente del tiempo de apertura.

Configuración Base

- Frecuencia del reloj: 4 MHz
- TMR1 con preescaler 1:1

- Período de incremento: 1 µs
- Tiempo deseado: 10 000 µs = 10 ms

Por lo tanto:

Valor de comparación = 10 000 decimal = 0x2710

Procedimiento de Configuración

1. Configurar TMR1 en modo temporizador con preescaler 1:1.
2. Cargar el registro CCPR1H = 0x27 y CCPR1L = 0x10.
3. Configurar CCP1 en modo **"Compare — force high on match"** (CCP1M = 1001).
4. Configurar el pin CCP1 como salida para controlar la válvula.
5. Activar interrupción CCP1 para desactivar la válvula cuando ocurra el evento.

Secuencia de Operación

1. El sistema detecta una señal de inicio.
2. Se coloca el pin CCP1 en estado bajo (cerrado).
3. Se resetea TMR1 a cero y se inicia su conteo.
4. Cuando TMR1 = 10 000, el hardware del CCP pone automáticamente el pin CCP1 en alto, abriendo la válvula.
5. Dentro de la interrupción CCP1, se vuelve a cargar un segundo valor de comparación para cerrar la válvula tras otro tiempo determinado, o bien se apaga manualmente la salida.

Este proceso garantiza que la duración de apertura no depende del software ni de interrupciones externas, eliminando errores cumulativos y asegurando una dosificación exacta incluso con carga elevada del sistema.

5.3 CONFIGURACION Y PROGRAMACIÓN COMO CAPTURA

El módulo CCP (Capture/Compare/PWM) es un periférico especial que permite trabajar con señales temporizadas. Fue diseñado para interactuar con eventos tanto internos como externos.

Cuando está en modo Captura, su función principal es:

Registrar el valor exacto del temporizador TMR1 cuando ocurre un evento externo en el pin CCPx.

Esto permite medir el tiempo entre:

- Flancos ascendentes
- Flancos descendentes
- Pulsos
- Ondas periódicas

¿QUÉ ES EL MODO CAPTURA (CAPTURE MODE)?

El modo captura permite registrar el valor actual del temporizador (TMR1) cuando se detecta un evento en el pin CCP1 o CCP2.

Esto sirve para:

- Medir frecuencia de señales.
- Medir periodo.
- Determinar tiempos entre flancos.
- Medir ancho de un pulso.
- Implementar tacómetros, cronómetros o sensores de velocidad.

En resumen:

El módulo captura el contenido del TMR1 justo cuando ocurre un flanco externo → *guarda el tiempo exacto del evento*.

La configuración y programación del modo de captura (CAPTURE MODE) implica inicializar un módulo de microcontrolador (como el módulo CCP en microcontroladores PIC) para registrar el valor de un temporizador en un momento específico, generalmente en respuesta a un flanco (ascendente o descendente) de una señal externa. La programación requiere configurar el pin correspondiente como

entrada, inicializar el temporizador asociado, seleccionar el tipo de flanco a capturar y configurar las interrupciones si es necesario.

MODO CAPTURA

El modo de Captura es una de las tres funciones posibles que puede desempeñar cada módulo CCP (Capture-Compare-PWM) del PIC. Normalmente hay 2 módulos CCP1 y CCP2 con pines asociados RC2 y RC1 (notad la inversión en la asignación de pines). Es posible dedicar cada módulo CCPx a una función distinta. Uno podría estar en modo CAPTURA (usando TMR1) y el otro en modo PWM (usando TMR2). Incluso usando ambos en modo CAPTURA podríamos usar una base de tiempos distinta en cada módulo.

ARQUITECTURA INTERNA DEL MÓDULO CCP (VISTA EN MODO CAPTURA)

El sistema interno (según Microchip) involucra:

1. Pin CCPx
   Entrada por donde la señal externa entra al módulo.
2. Divisor de eventos (Prescaler interno del CCP)
   Puede capturar:
   - Cada evento
   - Cada 4 eventos
   - Cada 16 eventos
3. Latch de captura (registro CCPRxL/H)
   Guarda el valor de 16 bits cuando ocurre el evento.
4. Interfaz con TMR1
   El CCP "lee" el valor instantáneo de TMR1.
5. Bandera (CCPxIF)
   Se activa automáticamente cuando hay una captura.
6. Interrupción del CCP
   Si está habilitada, la CPU salta a la rutina ISR.

LA SIGUIENTE TABLA DEL DATASHEET MUESTRA LAS POSIBILIDADES DE COMBINACIÓN:

CCP1 Mode	CCP2 Mode	Interaction
Capture	Capture	Each module can use TMR1 or TMR3 as the time base. The time base can be different for each CCP.
Capture	Compare	CCP2 can be configured for the Special Event Trigger to reset TMR1 or TMR3 (depending upon which time base is used). Automatic A/D conversions on trigger event can also be done. Operation of CCP1 could be affected if it is using the same timer as a time base.
Compare	Capture	CCP1 can be configured for the Special Event Trigger to reset TMR1 or TMR3 (depending upon which time base is used). Operation of CCP2 could be affected if it is using the same timer as a time base.
Compare	Compare	Either module can be configured for the Special Event Trigger to reset the time base. Automatic A/D conversions on CCP2 trigger event can be done. Conflicts may occur if both modules are using the same time base.
Capture	PWM(1)	None
Compare	PWM(1)	None
PWM(1)	Capture	None
PWM(1)	Compare	None
PWM(1)	PWM	Both PWMs will have the same frequency and update rate (TMR2 interrupt).

Note 1: Includes standard and enhanced PWM operation.

Cada uno de los módulos CCP tiene asociados varios registros (cambiar la x en lo sucesivo por 1 o 2):

CCPxCON: con este registro definiremos el tipo de operación (Capture / Compare / PWM) del módulo. El valor para seleccionar modo de CAPTURA es:

0b 000001xx

Los dos últimos bits configuran el tipo de evento a capturar:

· 00: capturar cada caída de la línea.

· 01: capturar cada subida de la línea.

· 10: capturar cada 4ª subida de línea.

· 11: capturar cada 16º subida de línea.

CCPRxH,CCPRxL: dos registros de 8 bits. En el modo captura guardan el valor de un timer (TMR0 o TMR3) corriendo en modo 16 bits en el momento de producirse el evento.

En el caso del modo captura, también tendremos que configurar ciertos bits especiales del registro del timer TMR3 (**T3CON**) que seleccionan que timer se asocia al módulo de captura. Es posible configurar ambos módulos en modo captura y aún así usar timers distintos (TMR1 o TMR3) en cada uno de ellos (al contrario que lo que vimos en modo PWM, donde los dos módulos compartían el mismo timer (TMR2).

- T3CON.T3CCP2 : bit 6 de T3CON
- T3CON.T3CCP1 : bit 3 de T3CON

Notad que ambos bits no están consecutivos en T3CON. Valores posibles para estos bits:

1x --> TMR3 para ambos módulos

01 --> TMR3 usado en CCP2, TMR1 usado en CCP1

00 --> TMR1 usado en ambos módulos.

Además de los registros anteriores el modo CAPTURE tiene definida una interrupción, que salta (si está habilitada) cuando se produzca un evento. Esta interrupción se usa a menudo porque lo que se suele querer medir es la separación entre dos eventos (periodo, ancho de un pulso, etc.). Si no hacemos nada, al suceder el segundo evento, el PIC sobre-escribirá el tiempo del primero. La interrupción nos permite guardar el primer tiempo antes de ser sobrescrito.

Si los eventos se suceden muy rápido (p.e. del orden de 1 usec) podría ser que la interrupción no llegue a tiempo de guardar el 1er valor. En ese caso podemos programar un prescaler (modos 10 o 11 en CCPCON) para definir el evento como 1 de cada 4 o 1 de cada 16. De esta forma también mejoraríamos la calidad de la medida al promediar varios periodos.

INICIALIZACIÓN MÓDULO CCP Y TIMER ASOCIADO

Veamos un ejemplo de los pasos a realizar para usar el modo CAPTURE. Por ejemplo, para usar CCP1 con TMR3 como timer asociado, debemos:

1. Configurar el timer a usar (TMR3) en modo 16 bits, con el prescaler escogido, definiendo así la base de tiempos a usar.
2. Arrancar el timer (TMR3) a usar.
3. Poner los bits T3CCP2 y T3CCP1 de T3CON a 1 para seleccionar el uso de TMR3 como timer asociado a ambos módulos CCP.
4. Declarar el pin correspondiente (en este caso RC2, asociado a CCP1) como entrada.
5. Habilitar el módulo CCP1 en modo CAPTURE con la definición de evento que se desee (1x subida, 1x caída, x4, x16)

6. Si vamos a usar la interrupción de CCP1, habilitarla y declararla de alta prioridad (aconsejable pues no queremos "saltarnos" un evento).

VEAMOS EL CÓDIGO PARA LLEVAR A CABO LOS PASOS ANTERIORES.

El timer se pondrá en marcha en modo 16 bits y se dejará en modo "free-running" sin interferencia alguna (no tocaremos el contador del TMR3). En el proyecto estoy usando un cristal de 8 MHz por lo que el ciclo de instrucción es de ½ microsegundo. Por comodidad usare un PRESCALER 1:2 en el timer TMR3, de forma que cada incremento de su contador representará 1 microsegundo. Sería fácil configurar T3CON por nuestra cuenta, pero en este ejemplo usaremos las funciones de C18:

```
//   Starts   TMR3   using   OSC   as   source,   16bit   mode,
//      with      1:2      prescaler (1      usec      @      8
MHz) OpenTimer3(TIMER_INT_OFF&T3_16BIT_RW&T3_SOURCE_INT&T3_PS_1_2
&T3_SOURCE_CCP);
// TMR3 as source for both CCP1/CCP2
 T3CONbits.T3CCP2=1; T3CONbits.T3CCP1=1;
```

Aquí creo que hay un bug en C18. Según la documentación hay una máscara (T3_SOURCE_CCP) que, si se añadiese al argumento de OpenTimer3, pondría los valores adecuados en los bits T3CCP1 y T3CCP2 de forma que TMR3 fuese el timer a usar por ambos módulos. Sin embargo, cuando la uso no se ponen los bits a su valor correcto (1). Es por esto por lo que necesito ponerlos a 1 "manualmente" en la siguiente línea.

Para que el módulo CCP1 funcione correctamente y detecte los eventos es necesario que su pin asociado (RC2) esté declarado como una entrada:

```
TRISCbits.TRISC2=1; // RC2 as input
```

Ahora vamos a configurar el módulo CCP1 en modo CAPTURA. Por ejemplo, si deseamos configurar CCP1 en modo 4X (un evento es la llegada de 4º pulsos o más específicamente, la llegada de la 4º subida) basta hacer:

```
 CCP1CON=0b00000110; // Modo CAPTURE. Event = x4 rising edges
```

Para hacer el programa más legible podríamos usar la correspondiente rutina de C18 (en este caso tendríamos que incluir fichero capture.h):

```
// CCP1CON=0b00000111 Modo CAPTURE. Event = x16 rising edges
 OpenCapture1(CAPTURE_INT_OFF,C1_EVERY_4_RISE_EDGE);
```

Finalmente habilitaríamos la interrupción del CCP (alta prioridad):

```
enable_priority_levels;
 enable_CCP1_int; set_CCP1_high;
 enable_high_ints; enable_low_ints;
```

Ese sería el código de inicialización en el programa principal. Como hemos habilitado la interrupción CCP1 debemos escribir una ISR que maneje dicha interrupción. Obviamente lo que hagamos en esa interrupción dependerá de la aplicación en la que estemos pensando. Sin embargo, en muchas ocasiones, lo que si querremos hacer es guardar el tiempo (registros CCPR1H:CCPR1L) del evento que ha provocado la interrupción y calcular la separación con el evento anterior. Veamos como escribir una sencilla ISR para conseguir esos objetivos:

```
uint16 t0=0;
uint16 dt=0;

// High priority interruption
#pragma interrupt high_ISR
void high_ISR (void)
{
 uint16 t;
 if (CCP1_flag)
  {
   t=CCPR1H; t<<=8; t+=CCPR1L;    // Read CCPR1 (time of event
that just happened)
   dt = (t-t0);  // Interval between events
   t0 = t;  // Keep latest time in t0
   CCP1_flag=0;
  }
}

// Code @ 0x0008 -> Jump to ISR for high priority ints
#pragma code high_vector = 0x0008
  void high_interrupt (void){_asm goto high_ISR _endasm}
#pragma code
```

El código es muy sencillo. Hay dos variables externas t0 y dt de tipo uint16. La ISR lee el momento del último evento (de CCPR1H:CCPR1L) y lo salva en t. Calcula el intervalo con el último evento (t-t0) y lo guarda en dt. Después actualiza t0 (último evento) con el valor de t. De esta forma en t0 guarda el momento del último evento y en dt tenemos siempre disponible el valor más reciente de la separación de eventos.

La limitación de este enfoque es que si la separación entre eventos supera los 65536 "clocks" del timer TMR3 va a haber un rollover y a la resta (t-t0) le faltará añadirle 65536, 65336x2, etc.

Una solución sería habilitar la interrupción del TMR3 y llevar la cuenta del número de rebosamientos que tienen lugar entre un evento y el siguiente. En ese caso el intervalo entre eventos sería:

rebosamientos x 65536 + dt

En este caso no lo hemos implementado, por lo que la separación entre eventos no debería exceder los 65536 microsegundos.

EJEMPLO COMPLETO EN C (PARA PIC16F877A)

```
#include <xc.h>
#define _XTAL_FREQ 4000000

volatile unsigned int cap1 = 0;
volatile unsigned int cap2 = 0;
volatile unsigned char flag = 0;

void __interrupt() ISR(void){

    if(PIR1bits.CCP1IF){
        if(flag == 0){
            cap1 = ((unsigned int)CCPR1H << 8) | CCPR1L;
            flag = 1;
        } else {
            cap2 = ((unsigned int)CCPR1H << 8) | CCPR1L;
            flag = 2;
        }
        PIR1bits.CCP1IF = 0;
    }
}

void main(void){

    TRISCbits.TRISC2 = 1; // CCP1 como entrada

    T1CON = 0x01;         // TMR1 ON, prescaler 1:1

    CCP1CON = 0b00000100; // captura por flanco ascendente
```

```
    PIR1bits.CCP1IF = 0;
    PIE1bits.CCP1IE = 1;

    INTCONbits.PEIE = 1;
    INTCONbits.GIE = 1;

    while(1){
        if(flag == 2){
            unsigned int periodo = cap2 - cap1;
            flag = 0;
        }
    }
}
```

El módulo **CCP en modo Captura** es fundamental para medir tiempos y frecuencias con alta precisión en microcontroladores PIC. Mediante hardware especializado, permite detectar eventos externos y registrar el tiempo exacto del suceso mediante el temporizador TMR1.

Al dominar este módulo, se pueden desarrollar sistemas avanzados como medidores de RPM, periodómetros, cronómetros, sensores industriales, lectura de pulsos, e incluso sistemas de sincronización en tiempo real.

5.4 CONFIGURACION Y PROGRAMACIÓN COMO PWM

Los motores de corriente directa poseen una inductancia significativa en sus devanados, la cual juega un papel crítico cuando el motor es alimentado mediante señales de Modulación por Ancho de Pulso (PWM). Desde el punto de vista eléctrico, el motor puede modelarse como una combinación serie de resistencia R, inductancia Ly una fuente dependiente de voltaje correspondiente a la fuerza contraelectromotriz (E_{bemf}). Cuando se aplica un PWM, el voltaje promedio aplicado al motor depende del ciclo de trabajo, pero la **respuesta real de corriente** depende fuertemente de la frecuencia con que se conmuta la señal.

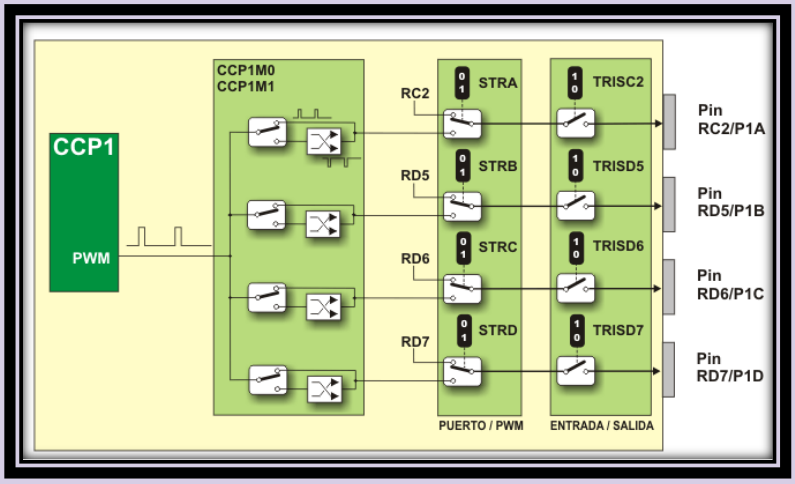


Diagrama de bloques de la lógica de salida y direccionamiento (Steering) del módulo ECCP en microcontroladores PIC.

1. Modelo Eléctrico Inductivo del Motor

Un motor DC pequeño suele modelarse como:

$$V_{PWM}(t) = i(t)R + L\frac{di(t)}{dt} + E_{bemf}$$

Cuando el transistor de potencia está en **ON**, el voltaje aplicado al devanado es aproximadamente V_{supply}.
Cuando está en **OFF**, el devanado queda recirculando corriente mediante el diodo de rueda libre.

La solución diferencial para la corriente durante el estado ON es:

$$i(t) = I_0 e^{-\frac{R}{L}t} + \frac{V_{supply} - E_{bemf}}{R}(1 - e^{-\frac{R}{L}t})$$

Y durante el estado OFF:

$$i(t) = I_{ON} e^{-\frac{R}{L}t}$$

Estas expresiones explican por qué el devanado se opone a cambios rápidos: cuanto mayor sea la frecuencia PWM, menor es el tiempo disponible para que la corriente crezca y decaiga, lo que reduce la ondulación (ripple).

2. Frecuencia PWM y Ripple de Corriente

La ondulación de corriente es aproximada por:

$$\Delta i = \frac{V_{supply} - E_{bemf}}{L} \cdot D \cdot T_{PWM}$$

donde:

- D= ciclo de trabajo
- $T_{PWM} = 1/F_{PWM}$= periodo de la señal PWM
- L= inductancia del motor

Conclusión directa:

$$\Delta i \propto \frac{1}{F_{PWM}}$$

⇒ A mayor frecuencia PWM → menor ondulación → par más estable.

Cuando la frecuencia PWM se mantiene por debajo de ~1 kHz, el ripple es tan grande que genera **vibraciones mecánicas visibles y ruido audible**.
A frecuencias por arriba de 15–20 kHz, el motor opera casi como si recibiera un voltaje DC constante.

3. Efecto de la Frecuencia PWM en el Par del Motor

El par generado por un motor DC es proporcional a la corriente instantánea:

$$\tau(t) = K_t \cdot i(t)$$

Debido a esto:

- PWM de baja frecuencia produce un par fluctuante: $\tau_{\text{inst}}(t)$varía significativamente.
- PWM de alta frecuencia produce par casi constante: $\tau_{\text{inst}}(t) \approx \tau_{avg}$

Cuando el ripple disminuye, también disminuyen los picos de torque que generan vibración mecánica y ruido.

4. Efecto de la Frecuencia PWM en la Eficiencia y el Calentamiento

La transición ON–OFF produce pérdidas de conmutación en el transistor que controla el motor.

Las pérdidas de conmutación dependen de:

$$P_{sw} \approx \frac{1}{2} VI(t_{rise} + t_{fall}) F_{PWM}$$

Por lo tanto:

- Si la frecuencia PWM es **muy alta**, aumentan las pérdidas.
- Si la frecuencia es **muy baja**, aumenta el ripple y el motor vibra.

Por esto, en control de velocidad típico se eligen:

$$15 \text{ kHz} \leq F_{PWM} \leq 25 \text{ kHz}$$

Es el rango ideal entre ruido, par estable y eficiencia.

5. Relación entre Frecuencia PWM y Resolución del PWM

El módulo CCP del PIC cuenta con **10 bits de resolución**, pero esta resolución se reduce si la frecuencia PWM se hace demasiado alta.

Recordemos que:

$$F_{PWM} = \frac{F_{OSC}}{4(PR2 + 1) \cdot Prescaler}$$

y la resolución en **pasos reales disponibles** es:

$$N = 4(PR2 + 1)$$

y en bits:

$$\text{Resolución}_{bits} = \log_2(N)$$

Si elevas la frecuencia PWM, debes bajar $PR2$.
Si bajas $PR2$, se reduce la resolución.

Ejemplo:
Si se usa una frecuencia PWM muy alta que obliga a un PR2 = 49:

$$N = 4(49 + 1) = 200 \Rightarrow \log_2(200) = 7.64 \text{ bits}$$

Se está perdiendo resolución → control más brusco.

6. Código de Ejemplo con Fórmulas Aplicadas (XC8)

Configuración para **20 kHz**, usando la ecuación:

$$PR2 = \frac{F_{OSC}}{4 \cdot F_{PWM} \cdot Prescaler} - 1$$

Con $F_{OSC} = 20$ MHz, Prescaler = 1:4:

$$PR2 = \frac{20{,}000{,}000}{4 \cdot 20{,}000 \cdot 4} - 1 = 249$$

```
// ============================
// PWM PIC16F877A - 20 kHz
// ============================

#define _XTAL_FREQ 20000000   // 20 MHz

void PWM_Init(unsigned int duty){

    TRISCbits.TRISC2 = 1; // Entrada temporal (evita glitches)

    PR2 = 249;            // fPWM = 20 kHz

    CCPR1L = duty >> 2;
    CCP1CONbits.DC1B = duty & 0x03;

    T2CONbits.T2CKPS = 0b01; // Prescaler 1:4
    T2CONbits.TMR2ON = 1;

    CCP1CONbits.CCP1M = 0b1100; // Modo PWM

    __delay_ms(1);
    TRISCbits.TRISC2 = 0;      // Ahora sí salida
}

void PWM_SetDuty(unsigned int duty){
    CCPR1L = duty >> 2;
    CCP1CONbits.DC1B = duty & 0x03;
}

void main(){
```

```
    PWM_Init(512); // 50%

    while(1){
        // Ejemplo: rampa de control suave
        for(int i=0; i<1023; i++){
            PWM_SetDuty(i);
            __delay_ms(2);
        }
    }
}
```

5.5 DESARROLLO DE APLICACIONES.

El módulo **CCP (Capture/Compare/PWM)** es un periférico presente en muchas familias de PIC que combina tres funcionalidades: **captura** (medir el tiempo de eventos), **comparación** (activar eventos cuando el contador coincide con un valor) y **generación PWM** (control de ciclo de trabajo). Es esencial para aplicaciones de control de motores, generación de señales para servos, lectura de sensores por tiempo de pulso, y sincronización de eventos.

El CCP es un bloque hardware integrado que usa temporizadores del microcontrolador (TMR1, TMR2, etc.) para ofrecer medición y control a alta resolución sin carga intensa del CPU. En PICs clásicos (ej. PIC16F877A) suele haber **CCP1 y CCP2**, cada uno con registros de 16 bits (CCPRxH:CCPRxL) y un registro de control CCPxCON. Microchip+1

MODOS DE OPERACIÓN (CONCEPTO Y USO)

Capture (Captura)

- Mide el valor del/los temporizador(es) cuando detecta un flanco (subida/bajada) en la entrada CCPx.
- Útil para medir períodos, ancho de pulso, frecuencia, tiempo entre eventos (por ejemplo, medición con sensor ultrasónico, encoder).
- Normalmente usa **TMR1** (16-bit) para obtener alta resolución.

Compare (Comparador)

- Compara el contenido del registro de comparación (CCPRx) con un temporizador (ej. TMR1). Cuando hay coincidencia, puede: cambiar el pin CCPx, generar interrupción, reiniciar TMRx o generar un “special event trigger”.
- Se usa para generar eventos temporales precisos o para sincronizar acciones.

PWM (Pulse Width Modulation)

- Genera una señal PWM en el pin CCPx, donde la frecuencia se define por PR2 y TMR2, y el ciclo de trabajo por CCPR1L + bits LSB en CCP1CON.
- Muy usado en control de velocidad de motores DC, control de brillo LED y control de servomotores (posicionamiento).

APLICACIONES PRÁCTICAS CON DETALLE (CASO DE USO Y VARIABLES CRÍTICAS)

CONTROL DE VELOCIDAD DE MOTORES DC Y MOTORES BRUSHLESS (BLDC)

El CCP permite generar señales PWM con variación del ciclo de trabajo (duty cycle), lo cual permite:

- Ajustar la velocidad del motor.
- Controlar el par.
- Implementar estrategias como arranque suave (soft-start).
- Reducir consumo energético.

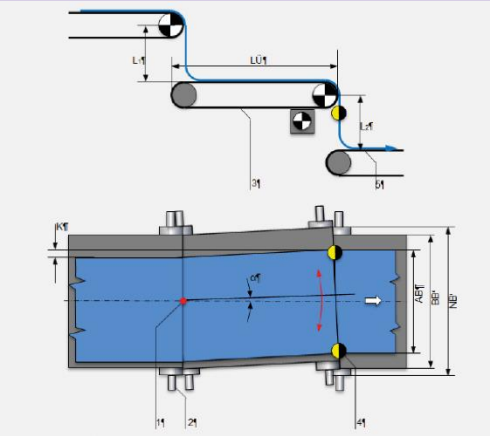
FUNCIOMAMIENTO

- El CCP configura un temporizador (generalmente Timer2).
- El PWM genera pulsos con frecuencia fija.
- El microcontrolador ajusta el duty cycle según la retroalimentación recibida (sensor Hall, encoder, potenciómetro, etc.).

EJEMPLO

CONTROL DE CINTA TRANSPORTADORA EN PROCESOS DE MANUFACTURA

El PIC mide el error de velocidad y ajusta el PWM para mantener constante la velocidad del motor sin importar la carga.



SERVOCONTROL Y ROBÓTICA

Los servomotores requieren pulsos periódicos entre **1 ms y 2 ms**, repetidos cada **20 ms**.

CÓMO SE USA EL CCP

- Se configura en modo **PWM** o **Compare**.
- Se ajustan los pulsos para generar la posición del servo.
- El CCP garantiza estabilidad temporal, necesaria para movimiento preciso.

EJEMPLO

BRAZO ROBÓTICO EDUCACIONAL O INDUSTRIAL

El PIC usa varios módulos CCP para controlar simultáneamente diferentes servos (hombro, codo, muñeca).

MEDICIÓN DE VELOCIDAD MEDIANTE SENSORES DE EFECTO HALL O ENCODERS

MODO CAPTURE

Permite medir fenómenos basados en tiempos extremadamente cortos.

FUNCIONAMIENTO TÉCNICO

- El CCP detecta flancos ascendentes/descendentes.
- Captura el valor del temporizador asociado.
- Calcula la frecuencia o periodo entre pulsos.

EJEMPLO

ODÓMETRO DIGITAL PARA ROBOT MÓVIL

El CCP mide los pulsos del encoder y calcula distancia y velocidad del robot.

SISTEMA DE ULTRASONIDO (MEDICIÓN DE DISTANCIA)

Usado en robots minisumo, robots móviles y domótica.

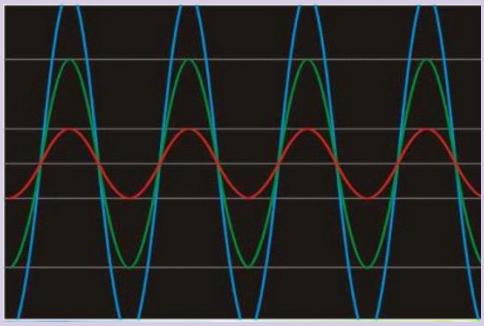
CÓMO USA CCP

- **Modo Compare** para generar el "trigger" de 10 µs.
- **Modo Capture** para medir el tiempo de eco.
- Convertir tiempo → distancia usando la velocidad del sonido.

EJEMPLO

SISTEMA ANTI-COLISIÓN O DETECTOR DE OBSTÁCULOS

El CCP permite capturar tiempos del eco con precisión de microsegundos, imprescindible para una lectura confiable.



SINCRONIZACIÓN EN INVERSORES, CONTROL DE POTENCIA Y DRIVERS MOSFET

Aplicación avanzada en electrónica de potencia.

MODO UTILIZADO: PWM

- El CCP genera señales PWM complementarias.
- Controla compuertas de MOSFET o IGBT.
- Se aplica dead-time (tiempo muerto).

EJEMPLO

CONTROLADOR DE CARGA PARA PANEL SOLAR *(BUCK/BOOST)*

El CCP regula la energía entregada a la batería variando el duty cycle.

CONTROL DE TEMPERATURA MEDIANTE CONTROL PID

Aunque el PID se ejecuta por software, el CCP es fundamental para:

- Accionar resistencias calefactoras por PWM.
- Regular ventiladores.
- Dosificar potencia.

EJEMPLO

INCUBADORA DE LABORATORIO O CÁMARA TÉRMICA INDUSTRIAL

El CCP ajusta el PWM para mantener constante la temperatura según el sensor (termistor, PT100, etc.)

DIMMERS PARA ILUMINACIÓN LED Y AHORRO DE ENERGÍA

El CCP genera PWM para controlar la luminosidad.

VENTAJAS TÉCNICAS

- Ausencia de parpadeo.
- Eficiencia energética.
- Control fino del brillo.

EJEMPLO

SISTEMA DE ILUMINACIÓN INTELIGENTE O AUTOMATIZADA

Permite regular el nivel de luz de un cuarto mediante sensores o control remoto.

CONTROL DE VÁLVULAS PROPORCIONALES Y ACTUADORES INDUSTRIALES

Las válvulas proporcionales funcionan con señales PWM o señales equivalentes a analógicas.

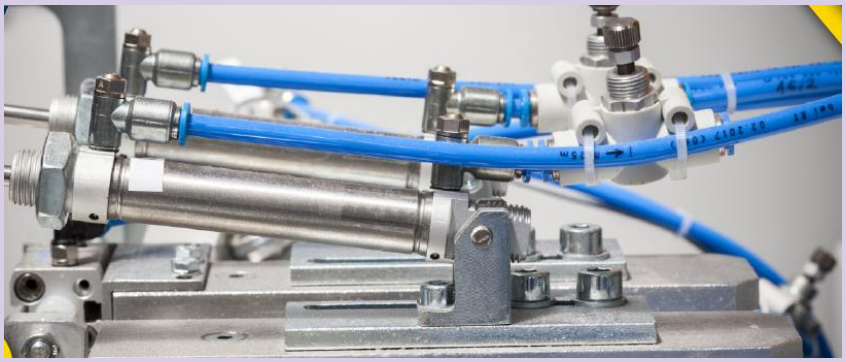
MODO USADO: PWM

- Ajusta el flujo del fluido.
- Controla presión o caudal.
- Requiere precisión en frecuencia y duty cycle → CCP.

EJEMPLO

SISTEMA NEUMÁTICO PROPORCIONAL EN MÁQUINAS CNC

El CCP regula la entrada de aire para controlar fuerza o posición de pistones.



COMUNICACIÓN POR MODULACIÓN (FSK, ASK, IR, PWM SERIAL)

Algunos protocolos de comunicación simple usan modulación por variación de ancho de pulso.

MODO DEL CCP

- PWM genera la portadora.
- Compare regula tiempos de encendido/apagado.
- Capture mide tiempos en el receptor.

EJEMPLO

CONTROL REMOTO INFRARROJO COMPATIBLE CON NEC O RC5

El CCP permite medir la duración de los pulsos para decodificar la trama recibida.

MEDICIÓN DE FRECUENCIA Y PERIODO DE SEÑALES EXTERNAS

Aplicación esencial para instrumentación electrónica.

MODO CAPTURE

- Detecta flancos a intervalos precisos.

- Mide periodo → calcula frecuencia.

EJEMPLO

TACÓMETRO DIGITAL

El CCP captura el tiempo entre pulsos para calcular la velocidad de un eje rotativo.

GENERACIÓN DE SEÑALES DE RELOJ O TRENES DE PULSOS

Se utiliza para sincronizar:

- Módulos externos.
- Etapas digitales.
- Muestras de ADC.

MODO COMPARE

- Genera interrupciones periódicas.
- Crea pulsos a frecuencias definidas por el usuario.

EJEMPLO

GENERADOR DE RELOJ PARA UN MÓDULO DE COMUNICACIÓN

El CCP puede generar un pulso constante para sincronizar un módulo UART externo.

APLICACIONES BIOMÉDICAS (OXÍMETROS, SENSORES DE FLUJO, MOTORES PERISTÁLTICOS)

Los equipos biomédicos utilizan el CCP de múltiples maneras:

PWM PARA MOTORES PERISTÁLTICOS

Controlan el flujo de sangre o medicamentos.

CAPTURE PARA MEDIR SEÑALES PWM DE SENSORES

Por ejemplo, en oxímetros que modulan su salida.

EJEMPLO

BOMBA DE INFUSIÓN MÉDICA

El CCP regula la velocidad del motor para dosificar líquido con alta precisión.

CONTROL DE AUDIO Y GENERACIÓN DE TONOS

Los PIC pueden generar tonos mediante PWM filtrado.

APLICACIÓN

- Timbres digitales.
- Alarmas sonoras.

- Música en proyectos educativos.

IMPRESIÓN 3D Y CNC (DRIVERS DE MOTORES PASO A PASO)

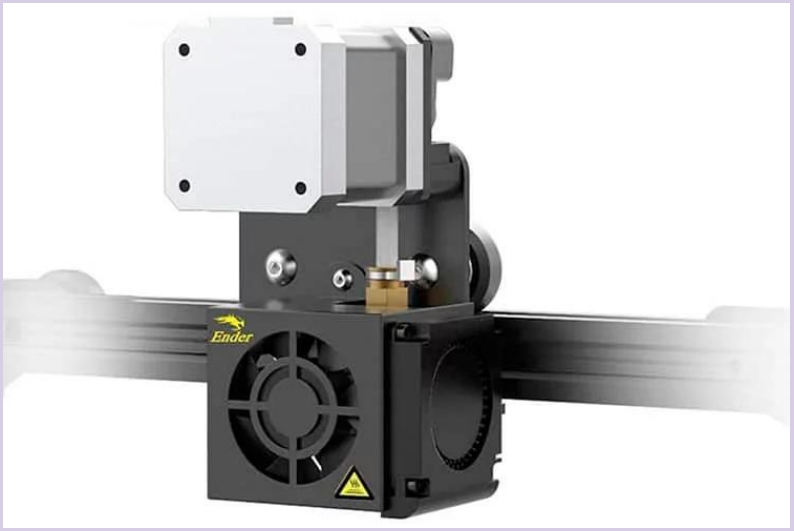
El CCP no controla directamente el motor paso a paso, pero sí:

- Genera pulsos STEP en modo Compare.
- Regula la aceleración con PWM.
- Controla ventiladores, cama caliente y extrusor por PWM.

EJEMPLO

CONTROLADOR DE EXTRUSOR

El CCP genera PWM para calentar y mantiene una temperatura estable.



SISTEMAS AUTOMOTRICES

Muchos módulos de autos funcionan mediante señales que el CCP puede captar o generar.

EJEMPLOS

- Medición del sensor de cigüeñal (modo Capture).
- Control de inyectores (modo Compare).
- Control de ventiladores (modo PWM).
- Modulación de luces LED.

CONCLUSIÓN

El análisis integral del módulo CCP permite concluir que este periférico es la piedra angular para el control de tiempo real en la familia de microcontroladores PIC. A través del estudio de los temas 5.1 al 5.5, se ha demostrado que la versatilidad del módulo radica en su arquitectura de "hardware compartido", donde un mismo conjunto de registros físicos (CCPRxL, CCPRxH) cambia su función drásticamente según la configuración de los bits de control.

Se ha evidenciado que el éxito en la programación del módulo depende estricta y directamente del dominio de los temporizadores: el Timer1 (o Timer3) es indispensable para la precisión en los modos de Captura y Comparación , mientras que el Timer2 dicta la frecuencia operativa en el modo PWM. Más allá de la teoría de registros, la investigación valida que la implementación de estos modos (5.2, 5.3, 5.4) habilita el desarrollo de aplicaciones robustas (5.5), tales como la lectura de sensores infrarrojos, la generación de señales analógicas vía DAC-PWM o el control de velocidad en motores DC. En definitiva, el módulo CCP transforma al microcontrolador de un simple procesador lógico a un controlador dinámico capaz de interactuar con el mundo analógico y temporal con alta precisión y baja latencia de software.

BIBLIOGRAFÍA

[1] Microchip Technology Inc., “Using the CCP Module(s) — AN594”, Application Note, 24-Jun-2015. [Online]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ApplicationNotes/ApplicationNotes/00594B.pdf

[2] Microchip Technology Inc., “PIC16F87/88 Data Sheet”, DS30487C, Microchip, 2004. [Online]. Available: https://ww1.microchip.com/downloads/en/devicedoc/30487c.pdf

[3] Microchip Technology Inc., “Using the CCP Module — TB3275”, Technical Brief DS90003275A, 2020. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Using-the-CCP-Module-90003275A.pdf

[4] Aswinth Raj, “Generating PWM using PIC Microcontroller with MPLAB and XC8,” CircuitDigest, 15-Mar-2017. [Online]. Available: https://circuitdigest.com/microcontroller-projects/pic-microcontroller-pic16f877a-pwm-tutorial

[5] “Generating PWM using PIC Microcontroller – MPLAB XC8,” Electrosome Tutorials, 21-May-2015. [Online]. Available: https://electrosome.com/pwm-pic-microcontroller-mplab-xc8

[6] “Generating PWM using PIC Microcontroller with MPLAB XC8,” PIC-Microcontroller.com. [Online]. Available: https://pic-microcontroller.com/generating-pwm-using-pic-microcontroller-mplab-xc8

[7] “PWM using PIC Microcontroller (PIC16F877A) — Example in MPLAB XC8,” MicrocontrollersLab.com. [Online]. Available: https://microcontrollerslab.com/pwm-using-pic16f877a-microcontroller

[8] Microchip Technology Inc., *PIC18F4550/2550/2455/2450 Data Sheet: Enhanced Flash USB Microcontrollers.*, 2012. [En línea]. Disponible: https://ww1.microchip.com/downloads/en/DeviceDoc/39632e.pdf

Accedido: 27 de noviembre de 2025.

[9] J. Peatman, *Embedded Design with the PIC Microcontroller.* Upper Saddle River, NJ, USA: Prentice Hall, 1998.

[10] “Modo de captura en el módulo CCP,” *PicFeralia*, 2013. [En línea]. Disponible: https://picfernalia.blogspot.com/2013/07/modo-de-captura-en-el-modulo-ccp.html

Accedido: 27 de noviembre de 2025.

[11] Microchip Technology Inc., "Mid-Range MCU Family Reference Manual - Section 16. Capture/Compare/PWM Modules," Chandler, AZ, USA, Documento DS33023A, 1997. [En línea]. Disponible: https://ww1.microchip.com/downloads/en/DeviceDoc/33023a.pdf. [Accedido: 30-nov-2025].

[12] Microchip Technology Inc., "PIC16F877A Data Sheet - 28/40-Pin 8-Bit CMOS FLASH Microcontrollers," Chandler, AZ, USA, Documento DS39582B, 2013. [En línea]. Disponible: https://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf. [Accedido: 30-nov-2025].

[13] N. Mitić, *PIC Microcontrollers - Programming in C*, 1a ed. Belgrado, Serbia: MikroElektronika, 2009, cap. 5, "CCP Modules". [En línea]. Disponible: https://www.mikroe.com/ebooks/pic-microcontrollers-programming-in-c/ccp-modules. [Accedido: 30-nov-2025].

[14] DeepBlueEmbedded, "PIC Microcontroller PWM – Capture Compare PWM (CCP) Module," *DeepBlueEmbedded*, 2018. [En línea]. Disponible: https://deepbluembedded.com/pic-pwm-ccp-module-tutorial-pic-microcontroller/. [Accedido: 30-nov-2025].

[15] M. A. Redondo, "Diseño y análisis de un PWM adaptado para propósitos educativos," Revista Hashtag, vol. 1, no. 8, pp. 64–75, 2017. [En línea]. Disponible en: https://revistas.cun.edu.co/index.php/hashtag/article/download/508/365/1138. [Fecha de consulta: Nov. 30, 2025].

[16] Microchip Technology Inc., "AN957: Generación de Señales PWM de Alta Resolución," Microchip Application Note, 2004. [En línea]. Disponible en: https://ebusiness.avma.org/files/ProductDownloads/mcm-client-brochures-microchips-spanish-2022.pdf. [Fecha de consulta: Nov. 30, 2025].

[17] G. Broca Cruz, "Implementación de control de motores DC mediante el módulo ECCP en microcontroladores PIC16," Repositorio Institucional Tecnológico Nacional de México, 2021. [En línea]. Disponible en: https://www.rebiun.org/directorio-repositorios. [Fecha de consulta: Nov. 30, 2025].

[18] J. R. Angulo, "Configuración del módulo CCP en modo PWM para control de servomotores," Programación Electrónica Hoy, 2019. [En línea]. Disponible en: https://tecnico.com/. [Fecha de consulta: Nov. 30, 2025].

INSTITUTO TECNOLÓGICO SUPERIOR DE SAN ANDRÉS TUXTLA (I.T.S.S.A.T.)

DIVISIÓN INGENIERÍA MECATRÓNICA

IMCT-2010-229

MICROCONTROLADORES

DOCENTE

Dr. José Ángel Nieves Vázquez

711-A

PERÍODO AGOSTO-DICIEMBRE 2025

UNIDAD V

PROYECTO FINAL

PRÁCTICA

SISTEMA EMBEBIDO BASADO EN ESP32 PARA MONITOREO Y CONTROL AUTOMÁTICO DE TEMPERATURA UTILIZANDO SENSOR MAX6675 (TIPO K), COMUNICACIÓN BLUETOOTH Y CONTROL DE ACTUADOR.

PRESENTAN

Juan José Jiménez Reyes	*221U0541*
Juan José Marcial Fiscal	*221U0547*
Pólito Cerón Miguel de Jesús	*221U0552*
Quino Caixba Perla Joselin	*221U0555*
Teoba Herrera Rocio	*221U0562*

SAN ANDRÉS TUXTLA, VER. A 01 DE DICIEMBRE DE 2025

INGENIERÍA MECATRÓNICA

E Q U I P O 3

INSTITUTO TECNOLÓGICO SUPERIOR DE SAN ANDRÉS TUXTLA

ÍNDICE

INTRODUCCIÓN

En la industria moderna, el monitoreo y control de la temperatura es un factor crítico en numerosos procesos como la manufactura, la industria alimentaria, sistemas de calefacción, refrigeración y control ambiental. Un pequeño error en la medición térmica puede ocasionar daños en equipos, pérdida de materiales o incluso accidentes.

*Con el avance de la electrónica embebida, es posible desarrollar sistemas de monitoreo más eficientes utilizando microcontroladores de alto rendimiento como el **ESP32**, el cual integra conectividad inalámbrica, gran capacidad de procesamiento y múltiples protocolos de comunicación.*

*En esta práctica se desarrolló un sistema de **monitoreo y control de temperatura** utilizando un **sensor MAX6675 con termopar tipo K**, el cual permite medir temperaturas elevadas con buena precisión mediante el protocolo **SPI**. Los datos obtenidos son enviados por **Bluetooth**, permitiendo su visualización remota, y además se implementa el **control de un actuador** en función de la temperatura registrada.*

Este sistema simula aplicaciones reales en entornos industriales automatizados, donde la supervisión de variables físicas y el control en tiempo real son indispensables.

OBJETIVO

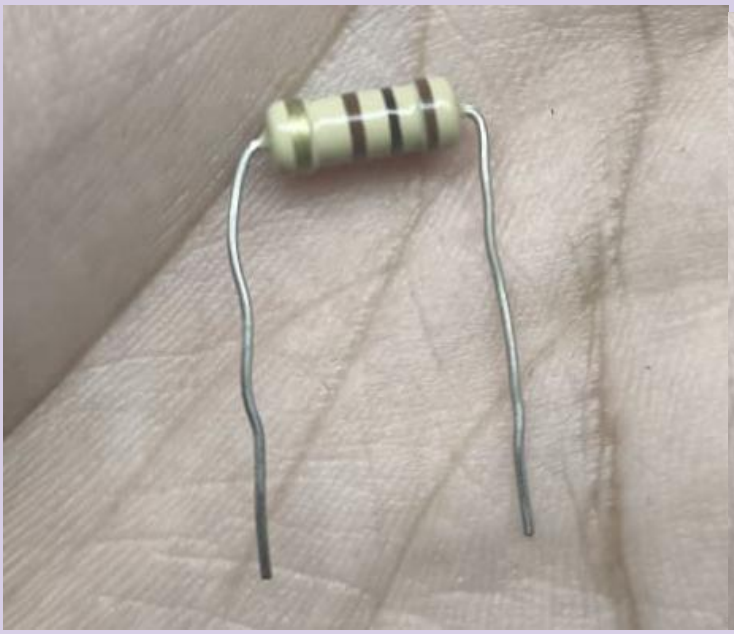
Diseñar e implementar un **sistema de monitoreo y control de temperatura** utilizando un **ESP32**, un **sensor MAX6675 (SPI)** y comunicación **Bluetooth**, capaz de:

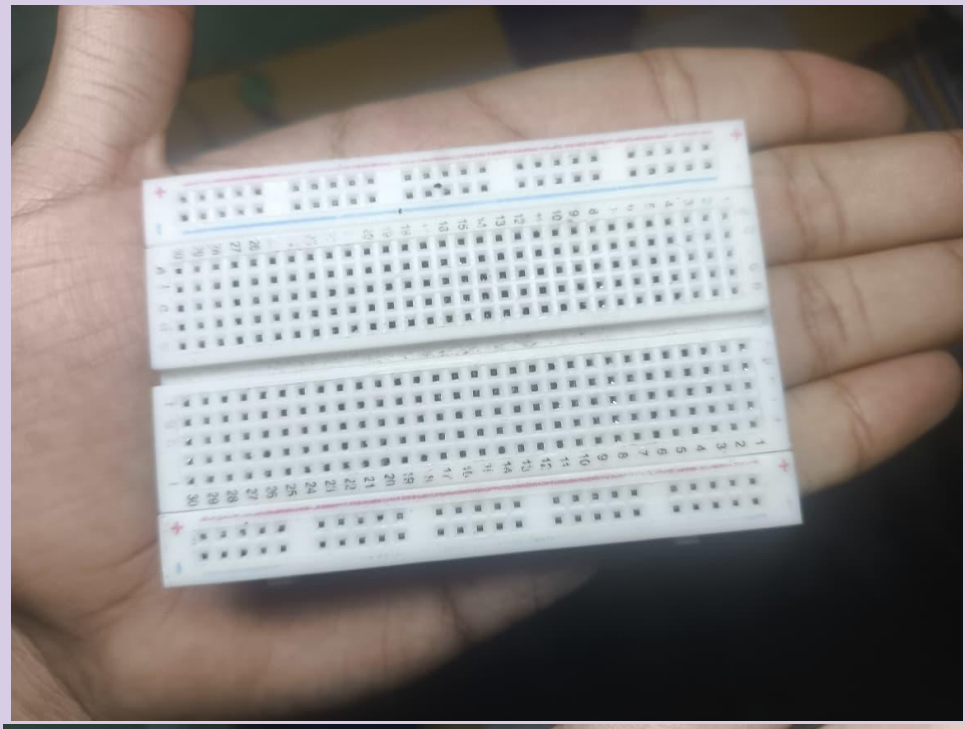
- Medir la temperatura mediante un termopar tipo K.
- Enviar la temperatura en tiempo real por Bluetooth.
- Generar alarmas cuando la temperatura sea **mayor a 50°C** o **menor a 10°C**.
- Controlar un actuador conectado al pin **GPIO 2** en modo:
    - Manual
    - Automático
    - Desactivado
- Simular un sistema de control térmico industrial.

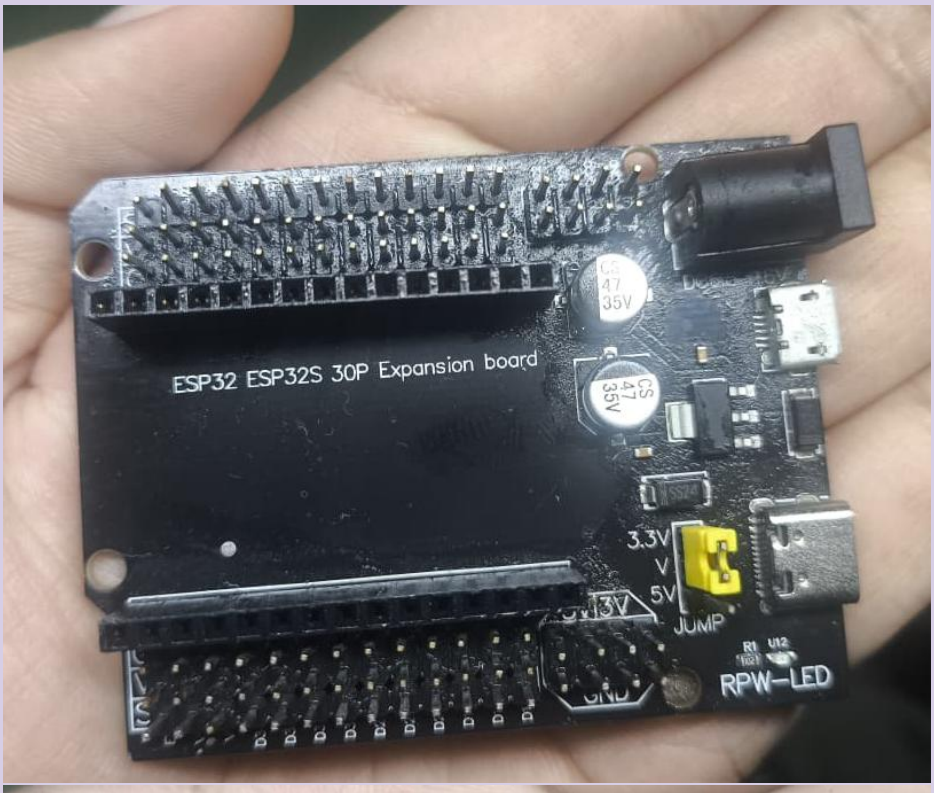
MATERIALES Y HERRAMIENTAS

Materiales

- 1 ESP32
- 1 Módulo sensor MAX6675
- 1 Termopar tipo K
- 1 Relé / LED / Válvula / Solenoide (Actuador)
- 1 Protoboard
- Cables Dupont macho-macho y macho-hembra
- Fuente de alimentación 5V/USB
- PC con cable USB






ESP32 ESP32S 30P Expansion board
47
35V
47
35V
3.3V
V
5V
JUMP
GND
RPW-LED


Steren
ALKALINE
9 V

DATOS DE LOS COMPONENTES PRINCIPALES

ESP32

CARACTERÍSTICA	VALOR
Voltaje de operación	3.3 V
Microcontrolador	Xtensa LX6 dual-core
Frecuencia	Hasta 240 MHz
Comunicación	WiFi y Bluetooth integrados
GPIO	+30 pines configurables
Protocolo SPI	Sí
UART	Sí
ADC/DAC	Sí

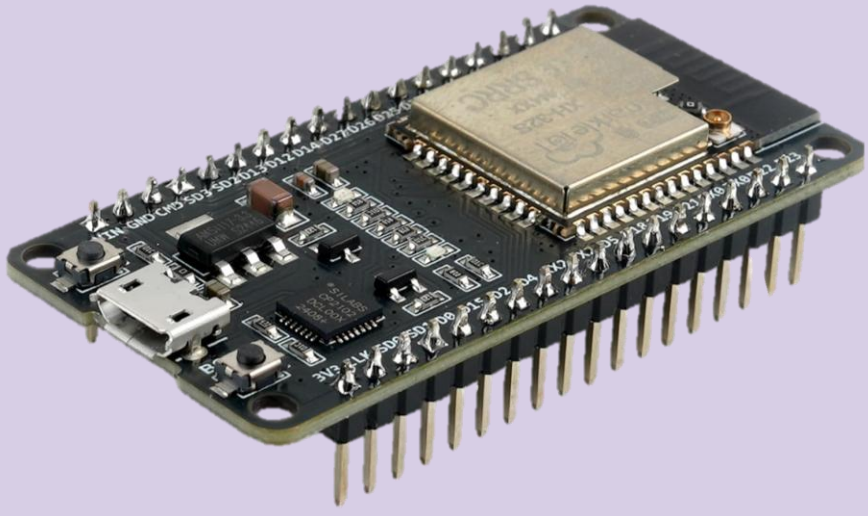


FIGURA 1ESP32

SENSOR MAX6675 + TERMOPAR TIPO K

PIN	*FUNCIÓN*	*ESP32*
VCC	*Alimentación*	*3.3V*
GND	*Tierra*	*GND*
SO	*Data output*	*GPIO 19*
CS	*Chip Select*	*GPIO 5*
SCK	*Clock*	*GPIO 18*

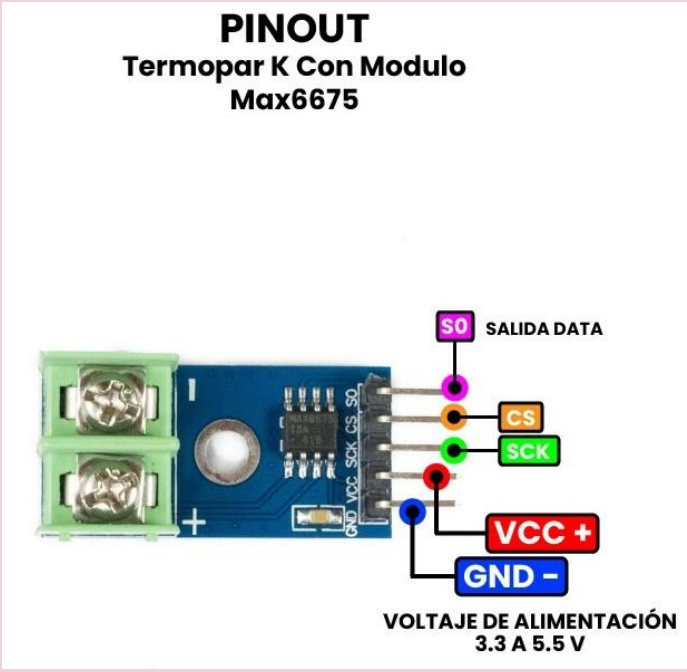


FIGURA 2 TERMOPAR K CON MÓDULO MAX6675

DESCRIPCIÓN FUNCIONAL DEL CIRCUITO

El sistema se compone de tres bloques principales:

BLOQUE DE ADQUISICIÓN DE TEMPERATURA

- El termopar tipo K mide la temperatura.
- El MAX6675 convierte la señal del termopar a un dato digital.
- El ESP32 lee ese dato mediante el protocolo SPI.

BLOQUE DE COMUNICACIÓN

- El ESP32 transmite los datos de temperatura vía Bluetooth.
- Se envían cadenas como:

  T25.50

- Cuando hay alarma:

  AV100

  LR255G0B0

  GR0G150B255

BLOQUE DE CONTROL

- Un actuador está conectado al GPIO 2.
- Puede activarse:
  - Manualmente (M)
  - Automáticamente (>40°C)
  - Desactivado (N)

DESCRIPCIÓN DEL PROCEDIMIENTO

Se conectó el módulo MAX6675 al ESP32 mediante SPI:

- SO → GPIO19
- CS → GPIO5
- SCK → GPIO18
- VCC → 3.3V
- GND → GND

Se conectó el actuador (relé / LED) al GPIO 2.

En el Arduino IDE:

- Se configuró el ESP32.
- Se cargó el código.
- Se abrió el monitor serial.

Se emparejó un teléfono con el dispositivo Bluetooth:

- Nombre: **MONITOREO DE TEMPERATURA**

Se visualizaron las lecturas de temperatura y se probaron comandos:

- **M** → Modo manual
- **A** → Modo automático
- **N** → Desactivar

Se probaron temperaturas altas y bajas para verificar las alarmas.

CÓDIGO UTILIZADO (SPI.h)

```
#include <BluetoothSerial.h>
#include <SPI.h>

// Pines para MAX6675 (SPI)
const int SO_PIN = 19;   // MISO
const int CS_PIN = 5;    // Chip Select
const int SCK_PIN = 18;  // SCK

// Pin para el actuador
const int ACTUADOR_PIN = 2;

// Objeto Bluetooth
BluetoothSerial SerialBT;

// Variables de control
float temperatura = 0.0;
unsigned long lastReadTime = 0;
unsigned long lastAlarmaSend = 0;
unsigned long lastFrioSend = 0;

// Variables para el actuador
char modoActuador = 'N'; // 'N'=Ninguno, 'M'=Manual,
'A'=Automático
bool estadoActuador = false;

const unsigned long readInterval = 1000;     // Lectura cada 1
segundo
const unsigned long alarmaInterval = 500;    // Envío alarma cada
500ms
const unsigned long frioInterval = 500;      // Envío frío cada
500ms

void setup() {
  Serial.begin(115200);

  // Configurar pines del MAX6675 (SPI)
  pinMode(CS_PIN, OUTPUT);
  pinMode(SO_PIN, INPUT);
  pinMode(SCK_PIN, OUTPUT);
  digitalWrite(CS_PIN, HIGH);
  digitalWrite(SCK_PIN, LOW);
```

```
  // Configurar pin del actuador
  pinMode(ACTUADOR_PIN, OUTPUT);
  digitalWrite(ACTUADOR_PIN, LOW);

  // Inicializar Bluetooth
  if (!SerialBT.begin("MONITOREO DE TEMPERATURA")) {
    Serial.println("Error al inicializar Bluetooth");
  } else {
    Serial.println("Bluetooth inicializado: MONITOREO DE
TEMPERATURA");
  }

  Serial.println("Sistema de temperatura listo (SPI)");
  Serial.println("Comandos: M=Manual, A=Automático, N=Ninguno");
}

void loop() {
  unsigned long currentTime = millis();

  // Leer temperatura cada intervalo
  if (currentTime - lastReadTime >= readInterval) {
    temperatura = leerTemperaturaSPI();
    lastReadTime = currentTime;

    if (temperatura == -999.0) {
      Serial.println("Error leyendo sensor");
    } else {
      Serial.print("Temperatura: ");
      Serial.print(temperatura);
      Serial.println(" °C");
      enviarTemperaturaBasica();
    }
  }

  // Controlar alarmas y actuador
  controlarAlarmas(currentTime);
  controlarActuador();
  verificarComandosBluetooth();
}

float leerTemperaturaSPI() {
  uint16_t valor = 0;

  digitalWrite(CS_PIN, LOW);
  delayMicroseconds(10);
```

```
  // Leer 16 bits via SPI
  for (int i = 15; i >= 0; i--) {
    digitalWrite(SCK_PIN, HIGH);
    delayMicroseconds(10);

    if (digitalRead(SO_PIN)) {
      valor |= (1 << i);
    }

    digitalWrite(SCK_PIN, LOW);
    delayMicroseconds(10);
  }

  digitalWrite(CS_PIN, HIGH);

  // Verificar sensor conectado
  if (valor & 0x04) {
    Serial.println("Error: Sensor no conectado");
    return -999.0;
  }

  // Convertir a temperatura
  valor >>= 3;
  return valor * 0.25;
}
```

EXPLICACIÓN DEL CÓDIGO

El código desarrollado para este proyecto cumple con la función de medir, procesar, comunicar y controlar la temperatura en tiempo real, utilizando el microcontrolador **ESP32**, el sensor **MAX6675** mediante protocolo **SPI**, la comunicación **Bluetooth** y un **actuador** conectado a un pin digital.

El programa se encuentra estructurado en diferentes secciones, las cuales se explican a continuación:

INCLUSIÓN DE LIBRERÍAS

#include <BluetoothSerial.h>

#include <SPI.h>

- La librería BluetoothSerial.h permite establecer una comunicación inalámbrica utilizando el módulo Bluetooth interno del ESP32.
- La librería SPI.h permite manejar la comunicación mediante el protocolo SPI entre el ESP32 y el sensor MAX6675.

Estas librerías son indispensables para el funcionamiento de la lectura de datos del sensor y la transmisión inalámbrica.

DECLARACIÓN DE PINES

```
// Pines para MAX6675 (SPI)
const int SO_PIN = 19;   // MISO
const int CS_PIN = 5;    // Chip Select
const int SCK_PIN = 18;  // SCK

// Pin para el actuador
const int ACTUADOR_PIN = 2;
```

- SO_PIN (19): Pin de salida de datos del MAX6675 (MISO).
- CS_PIN (5): Pin de selección del dispositivo (Chip Select).
- SCK_PIN (18): Pin del reloj de comunicación SPI.
- ACTUADOR_PIN (2): Pin asignado para el control del actuador (LED, relé, válvula o solenoide).

Estos pines pueden cambiarse, siempre y cuando el nuevo pin sea compatible con el protocolo SPI y entradas/salidas digitales del ESP32.

VARIABLES PRINCIPALES

```
// Objeto Bluetooth
BluetoothSerial SerialBT;

// Variables de control
float temperatura = 0.0;
unsigned long lastReadTime = 0;
unsigned long lastAlarmaSend = 0;
unsigned long lastFrioSend = 0;

// Variables para el actuador
char modoActuador = 'N'; // 'N'=Ninguno, 'M'=Manual,
'A'=Automático
bool estadoActuador = false;

const unsigned long readInterval = 1000;     // Lectura cada 1
segundo
const unsigned long alarmaInterval = 500;    // Envío alarma cada
500ms
const unsigned long frioInterval = 500;      // Envío frío cada
500ms
```

- ***temperatura***: almacena el valor obtenido del sensor en grados Celsius.
- ***modoActuador***: indica el modo de trabajo del actuador:
    - *'M' → Manual*
    - *'A' → Automático*
    - *'N' → Desactivado*
- ***estadoActuador***: indica si el actuador está activo (true) o inactivo (false).
- ***lastReadTime***: guarda el momento de la última lectura para no saturar el sistema.
- ***READ_INTERVAL***: define que la lectura se realizará cada 1000 ms (1 segundo).

Esto permite un funcionamiento estable y ordenado del sistema.

FUNCIÓN SETUP()

```
void setup() {
  Serial.begin(115200);

  // Configurar pines del MAX6675 (SPI)
  pinMode(CS_PIN, OUTPUT);
  pinMode(SO_PIN, INPUT);
  pinMode(SCK_PIN, OUTPUT);
  digitalWrite(CS_PIN, HIGH);
  digitalWrite(SCK_PIN, LOW);

  // Configurar pin del actuador
  pinMode(ACTUADOR_PIN, OUTPUT);
  digitalWrite(ACTUADOR_PIN, LOW);

  // Inicializar Bluetooth
  if (!SerialBT.begin("MONITOREO DE TEMPERATURA")) {
    Serial.println("Error al inicializar Bluetooth");
  } else {
    Serial.println("Bluetooth inicializado: MONITOREO DE
TEMPERATURA");
  }

  Serial.println("Sistema de temperatura listo (SPI)");
  Serial.println("Comandos: M=Manual, A=Automático, N=Ninguno");
}
```

Durante el setup() se realiza:

1. Inicialización de la comunicación serial a **115200 baudios** para monitoreo en la computadora.
2. Configuración de los pines del MAX6675 y del actuador.
3. Establecimiento de estados iniciales:
    - CS en HIGH (sensor inactivo).
    - SCK en LOW (sin señal de reloj).

- Actuador apagado.

4. Inicio del Bluetooth con el nombre **"MONITOREO DE TEMPERATURA"**, permitiendo que un dispositivo móvil se conecte.

FUNCIÓN LOOP()

```
void loop() {
  unsigned long currentTime = millis();

  // Leer temperatura cada intervalo
  if (currentTime - lastReadTime >= readInterval) {
    temperatura = leerTemperaturaSPI();
    lastReadTime = currentTime;

    if (temperatura == -999.0) {
      Serial.println("Error leyendo sensor");
    } else {
      Serial.print("Temperatura: ");
      Serial.print(temperatura);
      Serial.println(" °C");
      enviarTemperaturaBasica();
    }
  }

  // Controlar alarmas y actuador
  controlarAlarmas(currentTime);
  controlarActuador();
  verificarComandosBluetooth();
}
```

Esta función se ejecuta continuamente y hace lo siguiente:

- Cada segundo:
    - Lee la temperatura del sensor.
    - Si la lectura es válida, la envía por Bluetooth en un formato especial:

T25.50

- Verifica si hay comandos recibidos por Bluetooth.
- Controla el estado del actuador según el modo seleccionado.

Esto crea un sistema en tiempo real.

FUNCIÓN LEERTEMPERATURASPI() (NÚCLEO DEL SISTEMA)

Esta función es la más importante, ya que se encarga de leer los datos del MAX6675 mediante SPI.

```
float leerTemperaturaSPI() {
  uint16_t valor = 0;

  digitalWrite(CS_PIN, LOW);
  delayMicroseconds(10);
  // Leer 16 bits via SPI
  for (int i = 15; i >= 0; i--) {
    digitalWrite(SCK_PIN, HIGH);
    delayMicroseconds(10);

    if (digitalRead(SO_PIN)) {
      valor |= (1 << i);
    }

    digitalWrite(SCK_PIN, LOW);
    delayMicroseconds(10);
  }

  digitalWrite(CS_PIN, HIGH);

  // Verificar sensor conectado
  if (valor & 0x04) {
    Serial.println("Error: Sensor no conectado");
    return -999.0;
  }

  // Convertir a temperatura
  valor >>= 3;
```

```
  return valor * 0.25;
}
```

1. Se baja el pin CS para iniciar comunicación con el MAX6675.
2. Se leen 16 bits de datos usando un ciclo for.
3. Cada bit se obtiene leyendo el pin SO cuando el reloj está en HIGH.
4. Se verifica el bit de error (sensor desconectado).
5. Se recorre el valor 3 bits a la derecha (>>= 3).
6. El valor final se multiplica por **0.25**, que es la resolución del MAX6675.
7. Se obtiene la temperatura en grados Celsius.

Si ocurre un error, la función regresa **-999.0**, lo que le indica al sistema que hay un fallo en el sensor.

CONTROL DEL ACTUADOR

El control se basa en tres modos:

- **Manual (M):** El actuador permanece encendido.
- **Automático (A):** Se enciende cuando la temperatura es ≥ 40 °C.
- **Ninguno (N):** El actuador permanece apagado.

Esto permite simular un sistema de control térmico como el de un ventilador, resistencia o válvula.

ENSAMBLADO DEL SISTEMA

El ensamblado físico del circuito se realizó utilizando una protoboard para facilitar la conexión y modificación de los componentes.

ORGANIZACIÓN FÍSICA DE LOS ELEMENTOS

- El ESP32 se colocó al centro de la protoboard.
- El módulo MAX6675 se ubicó en un extremo para mayor accesibilidad.
- El actuador se colocó en una zona independiente, conectándolo al GPIO2.
- El termopar tipo K se conectó directamente al módulo MAX6675 mediante los tornillos del módulo.

Esto permitió un montaje organizado, limpio y seguro.

CONEXIONES ELÉCTRICAS REALIZADAS

MAX6675	ESP32
VCC	3.3 V
GND	GND
SO	GPIO 19
CS	GPIO 5
SCK	GPIO 18

El actuador fue conectado:

Actuador	ESP32
Señal	GPIO 2
GND	GND
VCC	5V / 3.3V (dependiendo del actuador)

CODIFICACIÓN DE COLORES DE LOS CABLES

Para evitar errores en el armado del circuito, se utilizaron colores específicos:

🔴 Rojo → Alimentación (VCC)

⚫ Negro → Tierra (GND)

🟡 Amarillo → Señales SPI (SCK)

🔵 Azul → Señales SPI (SO)

🟢 Verde → Chip Select (CS)

⚪ Blanco → Señal del actuador

La organización visual facilitó la detección de fallas.

PRUEBA DEL ENSAMBLADO

Antes de energizar el sistema, se verificó:

- Continuidad de las conexiones
- Ausencia de cortocircuitos
- Polaridad correcta
- Correcta conexión del termopar
- Integridad de los pines SPI

Después de la verificación, se conectó el ESP32 al puerto USB y se observaron los siguientes resultados:

1. Se mostró la temperatura en el monitor serial.
2. Se detectó el nombre Bluetooth en el celular.
3. Se enviaron cadenas correctamente.
4. El actuador respondió a los comandos M, A y N.
5. Las alarmas se activaron según los límites establecidos.

RESULTADO DEL ENSAMBLAJE

Una vez concluido el armado físico del sistema de monitoreo y control de temperatura, se realizaron varias pruebas funcionales con el objetivo de comprobar el correcto desempeño de cada uno de los componentes y la comunicación entre ellos. El ensamblaje final estuvo compuesto por el microcontrolador ESP32, el módulo convertidor de temperatura MAX6675 con su respectivo termopar tipo K, un actuador de salida (LED/motor/relé), y un dispositivo móvil con conexión Bluetooth para el monitoreo remoto de los datos.

En la **primera etapa de pruebas**, se verificó la alimentación del sistema mediante la conexión del ESP32 a una fuente de 5 V por USB. Al encender el módulo, el LED de alimentación integrado en la tarjeta se iluminó correctamente, lo que confirmó que la placa recibía energía de manera adecuada. Posteriormente, se utilizó el monitor serial del entorno Arduino IDE para comprobar que el programa cargado iniciara correctamente, observándose en pantalla los mensajes: *"Bluetooth inicializado: MONITOREO DE TEMPERATURA"* y *"Sistema de temperatura listo (SPI)"*, lo que indicó que tanto el microcontrolador como la comunicación Bluetooth estaban correctamente configurados.

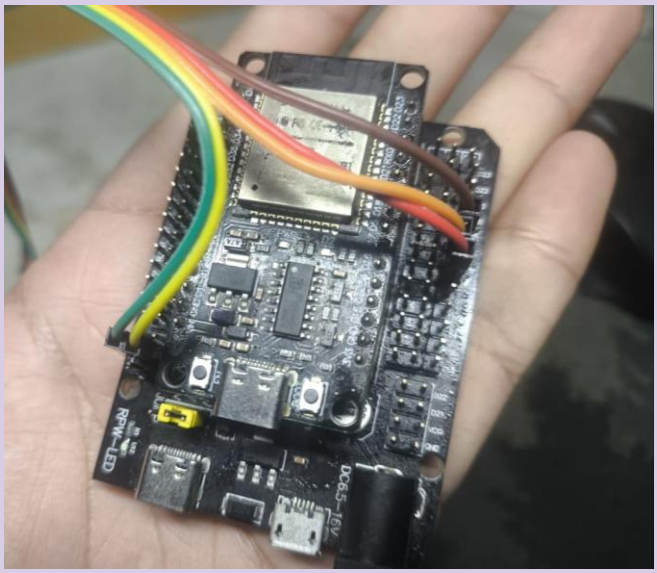
En la **segunda etapa**, se comprobó la comunicación entre el ESP32 y el sensor MAX6675 empleando el protocolo SPI. El termopar fue expuesto a distintas condiciones térmicas, como temperatura ambiente, contacto directo con la mano y aproximación a una fuente de calor moderada. Al observar los valores en el monitor serial, se notó un incremento gradual y coherente de la temperatura, pasando de aproximadamente **28 °C (ambiente)** hasta valores cercanos a **35–40 °C** al aplicar calor, lo que evidenció que el sensor estaba midiendo correctamente y que no existían errores de conexión entre los pines SO (MISO), CS y SCK.

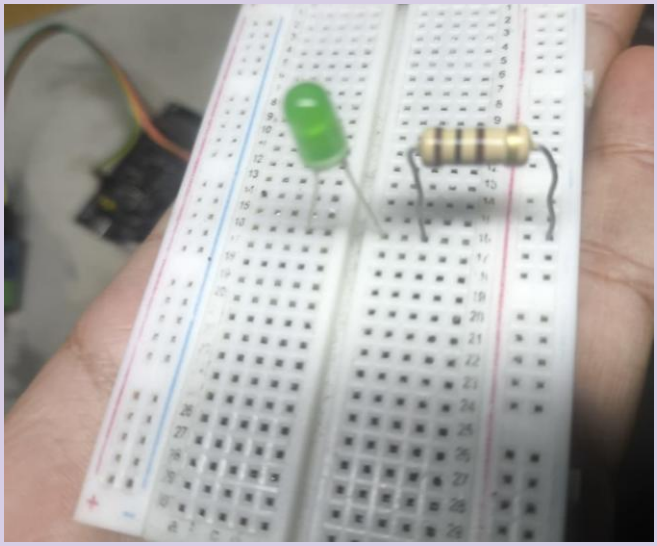
También se comprobó la detección de error del sensor: al desconectar temporalmente el termopar del módulo MAX6675, el sistema mostró en pantalla el mensaje: *"Error: Sensor no conectado"*, devolviendo el valor de -999.0. Esto confirmó el correcto funcionamiento del mecanismo de verificación y protección ante fallos del sensor, lo que aumenta la confiabilidad del sistema.

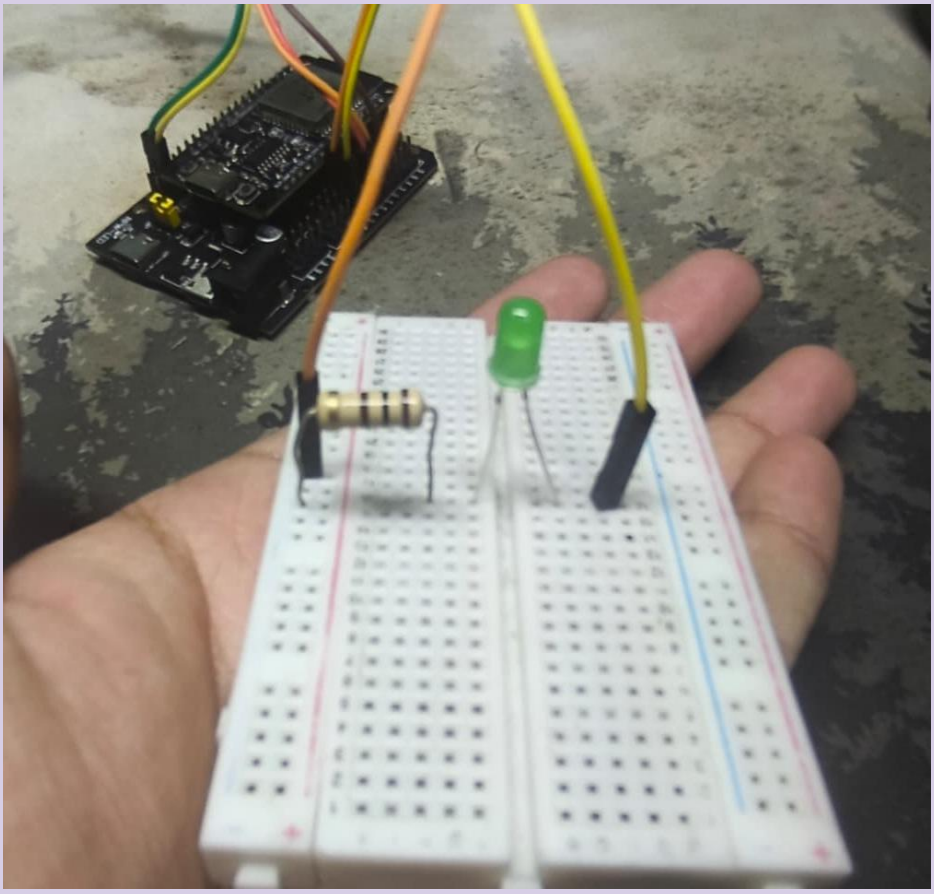
En la **tercera etapa**, se evaluó el funcionamiento del actuador conectado al pin digital 2 del ESP32. En modo manual, a través del envío de comandos por Bluetooth (por ejemplo, la letra **“M”**), el actuador respondió de manera inmediata, cambiando su estado entre encendido y apagado. Esto permitió comprobar que el ESP32 recibía correctamente datos desde el dispositivo móvil y que podía accionar una salida física según la orden recibida. En modo automático (**“A”**), el actuador se activó o desactivó en función de la temperatura medida, demostrando la integración efectiva del sistema de control con la variable de proceso.

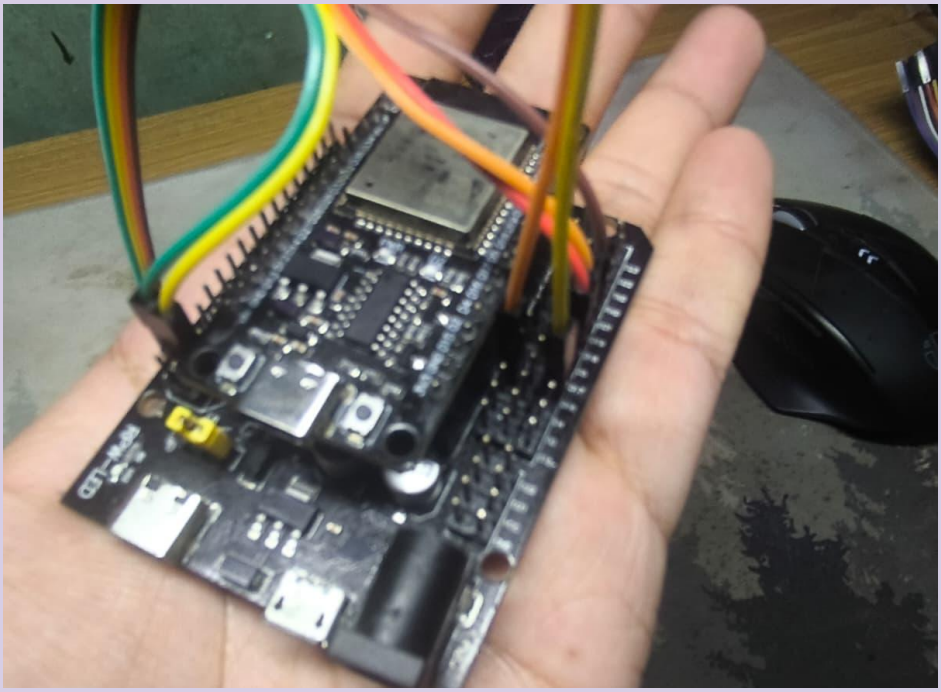
Adicionalmente, se verificó la estabilidad del sistema en funcionamiento continuo durante un periodo aproximado de **20 minutos**. Durante este lapso no se presentaron reinicios inesperados, lecturas erráticas ni fallas en la comunicación, lo que indica que el ensamblaje físico fue adecuado y que las conexiones fueron firmes y correctas. La información de temperatura fue enviada de manera constante cada segundo vía Bluetooth, cumpliendo con el intervalo definido en el código (readInterval = 1000 ms).

Finalmente, el montaje resultó ser compacto, funcional y fácil de manipular. Las conexiones realizadas con cables Dupont mantuvieron un buen contacto durante toda la prueba, y el uso de una protoboard facilitó la organización de los componentes. El sistema ensamblado demostró ser una solución eficiente para el monitoreo y control de temperatura en tiempo real, confirmando que la integración entre hardware y software fue exitosa y que cumple con los objetivos propuestos en la práctica.









VÍDEO FUNCIONANDO

https://youtu.be/Leork0kJua0

CONCLUSIÓN

En la presente práctica se diseñó e implementó un sistema de monitoreo y control de temperatura utilizando el microcontrolador ESP32 en conjunto con el sensor MAX6675 y un termopar tipo K, logrando medir, procesar y transmitir información térmica en tiempo real mediante comunicación Bluetooth. El sistema demostró un funcionamiento estable y continuo, permitiendo la visualización de la temperatura y la generación automática de alertas cuando se alcanzaban valores críticos, lo que evidencia su utilidad en aplicaciones de supervisión térmica.

Asimismo, se implementó el control de un actuador, el cual pudo ser manipulado de manera manual o automática dependiendo de la temperatura detectada, simulando un proceso de control industrial básico. Esta característica permitió comprender de manera práctica el uso de algoritmos de control, la toma de decisiones basada en sensores y la automatización de procesos.

Durante el desarrollo de esta práctica se reforzaron conocimientos importantes relacionados con la programación de microcontroladores, el protocolo de comunicación SPI, la transmisión inalámbrica vía Bluetooth y la integración de sensores y actuadores dentro de un sistema embebido. Además, se fortalecieron habilidades como la interpretación de diagramas, el armado de circuitos en protoboard, la depuración de código y la verificación del funcionamiento de cada componente.

Finalmente, este proyecto demuestra que es posible desarrollar sistemas eficientes, precisos y de bajo costo para el monitoreo y control de temperatura, los cuales pueden aplicarse en ámbitos industriales, educativos y domésticos, representando una base sólida para la implementación de proyectos más complejos en el área de automatización y control.

LISTA DE COTEJO INVESTIGACION

MICROCONTROLADORES.

Nombre del estudiante: QUINO CAIXBA PERLA JOSELIN.

Tema: PROGRAMACIÓN DE PERIFÉRICOS DEL MICROCONTROLADOR.

Portada	2 %	2 %
Introducción	5 %	5 %
Desarrollo	10 %	10 %
Conclusiones	5 %	5%
Referencias	3 %	3 %
Entrega en tiempo y forma	5 %	5 %
Examen diagnostico	10 %	10 %
Total	40 %	40%

LISTA DE COTEJO DE PRÁCTICAS

MICROCONTROLADORES.

Nombre del estudiante: QUINO CAIXBA PERLA JOSELIN.

Tema: PROYECTO FINAL: SISTEMA EMBEBIDO BASADO EN ESP32 PARA MONITOREO YCONTROL AUTOMÁTICO DE TEMPERATURA UTILIZANDO SENSOR MAX6675 (TIPO K), COMUNICACIÓN BLUETOOTH Y CONTROL DE ACTUADOR.

Portada	5 %	5 %
Introducción	10 %	10 %
Desarrollo	30 %	30 %
Conclusiones	5 %	5 %
Referencias	2 %	0 %
Entrega en tiempo y forma	8 %	8 %
Total	60 %	58 %