Informe Breve: Trabajo Voluntario 02B

Campus de Gijón - Universidad de Oviedo

Curso: Dispositivos Electrónicos Programables

Programa: EU4M – European Union Master in Mechatronics

Profesor: Ing. Miguel Angel José Prieto

Estudiante: Luis Antonio Orbegoso Moreno

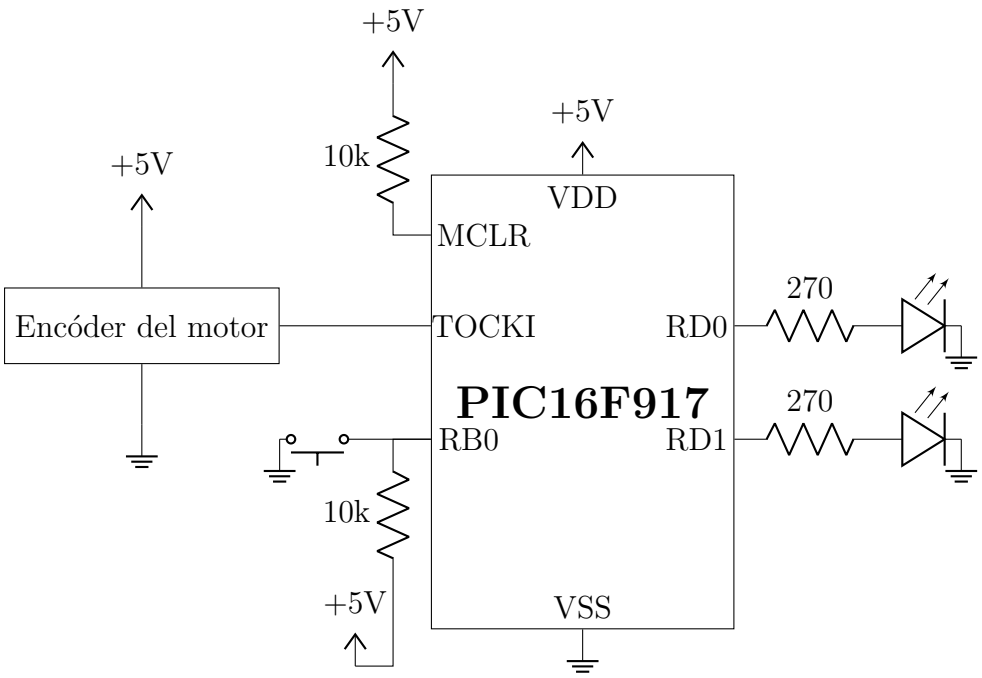
Fecha: 2 de noviembre de 2025

Lugar: Laboratorio de Electrónica - Campus Gijón

Introducción

Este informe describe el desarrollo de un sistema para medir la velocidad de un motor utilizando un microcontrolador PIC16F917. Se emplearon los temporizadores TMR0 y TMR1, interrupciones externas y lógica de control para activar indicadores según la velocidad calculada.

1 Circuito Eléctrico



2 Calculos previos

2.1 Calculo del TMR1

Se tienen los siguientes datos:

- Periodo de la interrupcion (T = 100ms).
- Frecuencia del oscilador interno ($F_{OSC} = 8MHz$).
- Prescaler 1:4.

Aplicando la formula del TMR1 se tiene:

$$T = \frac{4 \cdot PRESCALER}{F_{OSC}} \cdot (65535 - TMR1) \tag{1}$$

Reemplazando los datos se tiene que el número con el que se tiene que cargar al registro es $TMR1 = 15535$.

2.2 Calculo de la velocidad

Se saben los siguientes datos:

- Tiempo de muestreo ($T_m = 100ms$).
- Se desea obtener la velocidad angular en RPM.
- El número de pulsos por vuelta es de 100.
- El número de pulsos (N_{pulsos}) contados se almacenan en el TMR0.

Sabiendo esto, el número de vueltas que da el motor sabiéndose el número de pulsos recibidos es:

$$N_{vueltas} = \frac{N_{pulsos}}{100} \tag{2}$$

Y si esas vueltas se dieron durante un tiempo T_m, entonces la velocidad en RMP es:

$$V_{motor} = \frac{N_{vueltas}}{T_m} = \frac{N_{pulsos}}{100 \cdot 100ms} = \frac{N_{pulsos}}{10s \cdot \frac{1min}{60s}} = N_{pulsos} \cdot 6 \tag{3}$$

3 Pseudocódigo

```
//Configuracion de los fusibles

Delay con oscilador interno ← 8MHz
MCLR activado
WDT desactivado
Protección de codigo desactivada
```

```
Brown-out desactivado

//Configuracion del oscilador

Interno de alta frecuencia
Fosc ← 8MHz

//Configuracion de los pines

RD0 (LED1) y RD1 (LED2) → salidas
RB0 (PULS) y RA4 (TOCKI) → entradas
LED1 ← 0
LED2 ← 0

//Configuracion del TMR0

Modo ← contador externo (RA4)
Prescaler ← 1
TMR0 ← 0

//Configuracion del TMR1

Fuente ← Fosc/4
Prescaler ← 4
TMR1 ← 15535

//Configuracion de las interrupciones

INTCON.GIE ← 1      // Global interrupt enable
INTCON.PEIE ← 1     // Peripheral interrupt enable
INTCON.TMR0IE ← 1   // TMR0 interrupt enable
INTCON.INT0IE ← 1   // External interrupt enable
PIE1.TMR1IE ← 1     // TMR1 interrupt enable
```

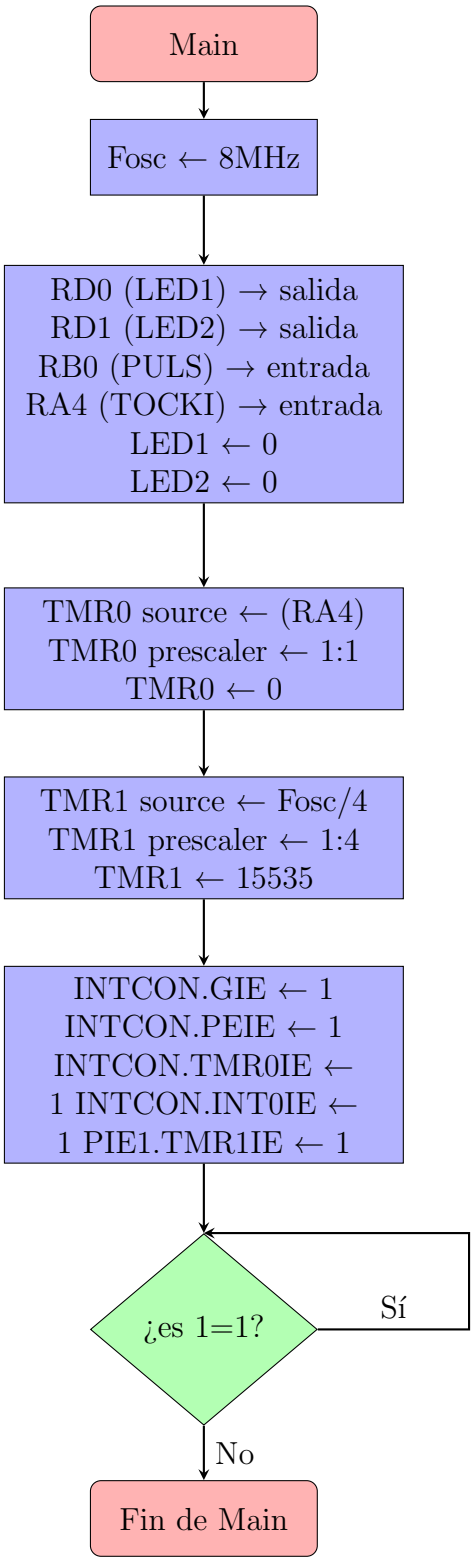
```
/Bucle principal
Hacer nada.
```

```
// INTERRUPCIÓN EXTERNA (RB0)
Esperar 20 ms (antirrebote)
Si RB0 = 0:
    Alternar LED1
Limpiar bandera INT

// INTERRUPCIÓN TMR0
pulsos ← pulsos + 256
Limpiar bandera TMR0

// INTERRUPCIÓN TMR1
velocidad ← (pulsos + TMR0)  6
Si velocidad > 800:
    LED2 ← 1
```

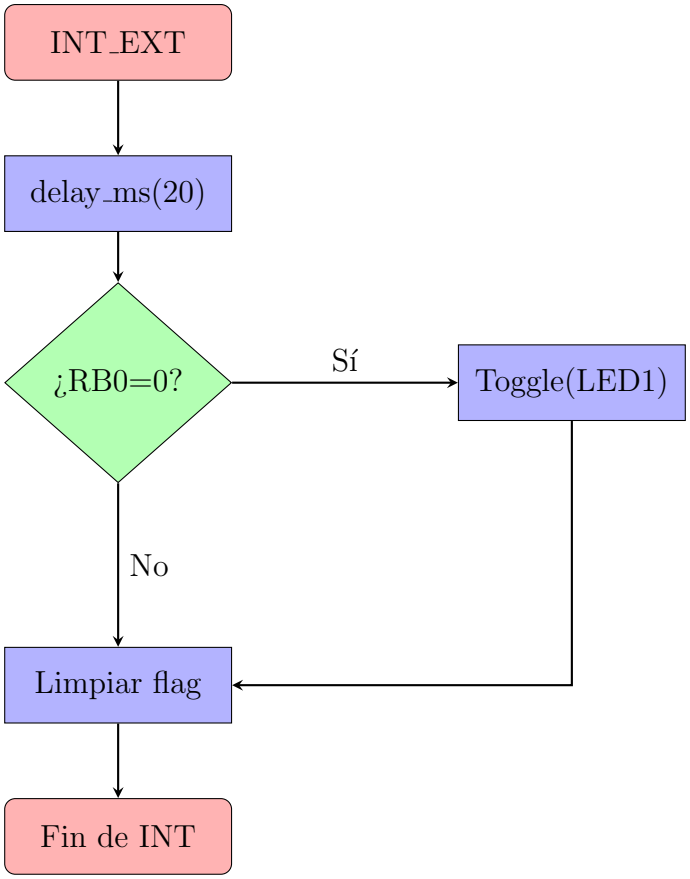
```
Sino:
    LED2 ← 0
pulsos ← 0
TMR0 ← 0
Recargar TMR1 ← 15535
Limpiar bandera TMR1
```

4 Diagramas de Flujo

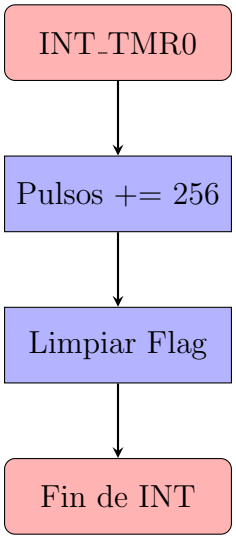
4.1 Programa Principal



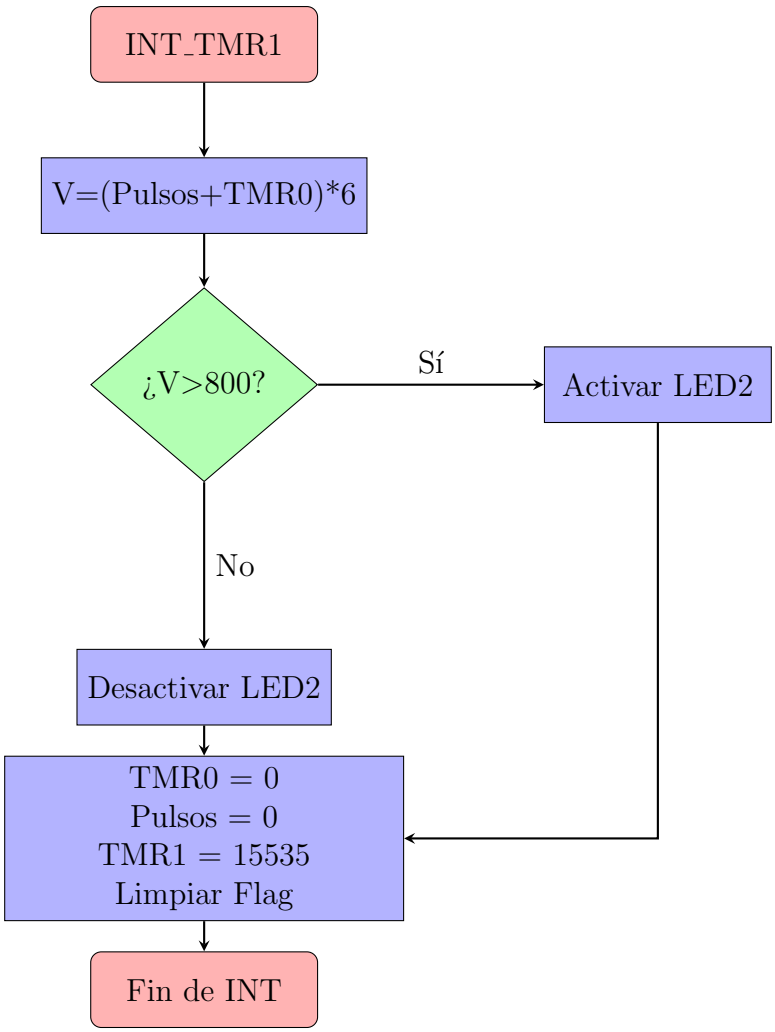
4.2 Interrupción Externa



4.3 Interrupción por Desbordamiento del TMR0



4.4 Interrupción por Desbordamiento del TMR1



Conclusiones

- El uso de interrupciones en microcontroladores PIC permite una **mayor eficiencia del sistema**, evitando el desperdicio de tiempo en bucles de espera (*polling*).
- Las interrupciones garantizan una **respuesta rápida a eventos críticos**, como la pulsación de botones o el desbordamiento de temporizadores, mejorando la precisión temporal.
- Se logra una **reducción de carga en el CPU**, ya que el procesador no necesita verificar constantemente condiciones, optimizando el rendimiento global.
- El manejo por interrupciones facilita la **modularidad y escalabilidad** del sistema, permitiendo implementar aplicaciones más complejas con múltiples eventos asíncronos.
- Se comprobó la **importancia de la configuración correcta** de registros (PIE, PIR, INTCON) y la habilitación/deshabilitación de interrupciones para evitar comportamientos indeseados.

- El uso de interrupciones por temporizador (TMR1) permitió **sincronización precisa** en retardos sin bloquear el flujo principal, lo cual es esencial en aplicaciones de tiempo real.
- Es fundamental **limpiar las banderas de interrupción** para evitar ejecuciones repetidas y considerar la gestión de prioridades para prevenir conflictos.

Repositorio del Proyecto

El código fuente completo y la simulación en Proteus se encuentran disponibles en el siguiente enlace:

`https://github.com/LuisAntonio1929/CalculoVelocidad_PIC16F917.git`