GUÍA DE USO: Librería I2C para dsPIC33FJ32MC204

==========

1. 📖 INTRODUCCIÓN

Librería completa para el módulo I2C del microcontrolador dsPIC33FJ32MC204.

2. ⭐ CARACTERÍSTICAS PRINCIPALES

- ☑ Soporte maestro y esclavo
- ☑ Velocidades: 100kHz, 400kHz, 1MHz
- ☑ Operaciones síncronas y asíncronas
- ☑ Buffer para recepción/transmisión
- ☑ Timeout y manejo de errores
- ☑ Compatible con SMBus/PMBus
- ☑ Callbacks por eventos
- ☑ Escaneo automático de bus

3. ⚙ CONFIGURACIÓN INICIAL

3.1 Inclusión de la librería

```
#include "i2c.h"
```

3.2 Configuración mínima (maestro)

```
#include "i2c.h"

int main(void) {
    // Configurar I2C como maestro a 100kHz
```

```
    I2C_Config_t config = I2C_CONFIG_DEFAULT_MASTER;
    config.speed = I2C_SPEED_100KHZ;

    I2C_Init(&config);

    // Escribir datos a un dispositivo
    uint8_t data[] = {0x01, 0x02, 0x03};
    I2C_WriteData(I2C_MODULE_1, 0x50, data, 3);

    // Leer datos de un dispositivo
    uint8_t buffer[3];
    I2C_ReadData(I2C_MODULE_1, 0x50, buffer, 3);

    return 0;
}
```

3.2 Configuración mínima (esclavo)

```
void mi_callback(I2C_Event_t evento, uint8_t dato) {
    switch(evento) {
        case I2C_EVENT_DATA_RECEIVED:
            printf("Dato recibido: 0x%02X\n", dato);
            break;
        case I2C_EVENT_DATA_REQUESTED:
            I2C_PutByte(I2C_MODULE_1, 0xAA); // Responder
            break;
    }
}

int main(void) {
    // Configurar como esclavo
    I2C_Config_t config = I2C_CONFIG_DEFAULT_SLAVE;
    config.slave_address = 0x40;
    config.callback = mi_callback;

    I2C_Init(&config);

    while(1);
    return 0;
}
```

4. 📚 FUNCIONES DISPONIBLES

4.1 Inicialización y Control

- `I2C_Init()` - Inicializa el módulo I2C
- `I2C_Start()` / `I2C_Stop()` - Control del bus
- `I2C_IsBusy()` - Verifica estado del bus

Transmisión/Recepción

- `I2C_WriteData()` - Escribe múltiples bytes
- `I2C_ReadData()` - Lee múltiples bytes
- `I2C_WriteRegister()` - Escribe a registro específico
- `I2C_ReadRegister()` - Lee de registro específico

Utilidades

- `I2C_ScanBus()` - Escanea dispositivos conectados
- `I2C_CheckDevice()` - Verifica si un dispositivo responde
- `I2C_PrintConfig()` - Muestra configuración actual

5. 🎛️ MODOS DE OPERACIÓN

5.1 Escanear el Bus I2C

```
uint8_t dispositivos[16];
if (I2C_ScanBus(I2C_MODULE_1, dispositivos, 16)) {
    printf("Dispositivos encontrados:\n");
    for(uint8_t i = 0; i < 16 && dispositivos[i] != 0; i++) {
        printf("  - 0x%02X\n", dispositivos[i]);
    }
}
```

5.2 Comunicación con EEPROM 24LC256

```
// Escribir en EEPROM
uint8_t datos[] = {0x00, 0x00, 0xAB}; // Dirección + dato
```

```
I2C_WriteData(I2C_MODULE_1, 0x50, datos, 3);

// Leer de EEPROM
uint8_t direccion[] = {0x00, 0x00};
I2C_WriteData(I2C_MODULE_1, 0x50, direccion, 2);
uint8_t dato_leido;
I2C_ReadData(I2C_MODULE_1, 0x50, &dato_leido, 1);
```

5.3 Sensor de Temperatura LM75

```
uint8_t buffer[2];
I2C_ReadData(I2C_MODULE_1, 0x48, buffer, 2);
int16_t temp_raw = (buffer[0] << 8) | buffer[1];
float temperatura = (float)(temp_raw >> 5) * 0.125;
printf("Temperatura: %.2f°C\n", temperatura);
```

5.4 Modo con Interrupciones

```
adc_config.interrupt_enable = true;
ADC_Init(&adc_config);
ADC_SetInterruptCallback(my_callback_function);
```

6. 💡 Configuración de Pines

Para I2C1 (por defecto):

- **SCL1**: RC3
- **SDA1**: RC4

Para I2C2:

- **SCL2**: RG2
- **SDA2**: RG3

La librería configura automáticamente los pines como entradas con pull-up.

Consideraciones Importantes

1. Resistores Pull-up

El bus I2C **requiere resistores pull-up externos**:

- **SCL**: 4.7kΩ a 10kΩ a VDD
- **SDA**: 4.7kΩ a 10kΩ a VDD

2. Velocidad Máxima

- **Standard mode**: 100 kHz
- **Fast mode**: 400 kHz
- **Fast mode plus**: 1 MHz

3. Timeout

Configurar timeout adecuado según la velocidad:

```
c
config.timeout_ms = 100;  // 100ms para operaciones normales
```

4. Interrupciones

Para modo asíncrono, habilitar interrupciones:

```
c
config.interrupt_enable = true;
config.callback = mi_funcion_callback;
```

🔧 Solución de Problemas

Problema: No se detectan dispositivos

1. Verificar conexiones físicas
2. Confirmar resistores pull-up (4.7kΩ)
3. Verificar direcciones I2C (generalmente 0x00-0x7F)
4. Usar osciloscopio para ver señales SCL/SDA

Problema: Errores de ACK

1. Dispositivo no conectado o apagado

2. Dirección incorrecta
3. Velocidad muy alta para el dispositivo

Problema: Señales ruidosas

1. Añadir capacitores de desacople (10-100nF)
2. Acortar cables de conexión
3. Usar twisted pair para SCL/SDA

📊 Diagrama de Conexión

```
text
dsPIC33FJ32MC204              Dispositivo I2C
     RC3 (SCL1) --------┬-------- SCL
                        |
                        4.7kΩ
                        |
                        VDD (3.3V)

     RC4 (SDA1) --------┬-------- SDA
                        |
                        4.7kΩ
                        |
                        VDD (3.3V)
```

📝 Ejemplo Completo

```c
c
#include "i2c.h"
#include <stdio.h>

int main(void) {
    // 1. Inicializar sistema
    SYSTEM_Initialize();

    // 2. Configurar I2C
    I2C_Config_t i2c_config = I2C_CONFIG_DEFAULT_MASTER;
    i2c_config.speed = I2C_SPEED_400KHZ;
    i2c_config.timeout_ms = 500;

    I2C_Init(&i2c_config);
```

```
    // 3. Escanear bus
    printf("Escaneando bus I2C...\n");
    uint8_t devices[10];
    if (I2C_ScanBus(I2C_MODULE_1, devices, 10)) {
        printf("Dispositivos encontrados!\n");
    }

    // 4. Comunicación con dispositivo
    uint8_t test_data[] = {0x01, 0x02, 0x03};
    if (I2C_WriteData(I2C_MODULE_1, 0x50, test_data, 3)) {
        printf("Datos escritos correctamente\n");
    }

    while(1);
    return 0;
}
```

📄 Licencia

MIT License - Ver archivo LICENSE para detalles.

🤝 Contribuir

1. Fork el repositorio
2. Crea una rama para tu feature
3. Commit tus cambios
4. Push a la rama
5. Abre un Pull Request

✉️ Contacto

Para preguntas o soporte:

- **Autor**: Carlos Olivera Vila
- **Email**: [c_olivera_y@hotmail.com]
- **GitHub**: [Carlos-Olivera-Vila]

Versión: 1.0.0
Última actualización: 04/12/2025
Compatibilidad: dsPIC33FJ32MC204, XC16 v1.70+