GUÍA DE USO: Librería ADC para dsPIC33FJ32MC204

==========

📋 TABLA DE CONTENIDOS

1. 🕮 INTRODUCCIÓN

Esta librería proporciona una interfaz completa para el módulo ADC de 10-bit, 16 canales del dsPIC33FJ32MC204. Está diseñada para ser fácil de usar mientras mantiene flexibilidad para aplicaciones avanzadas.

2. ★ CARACTERÍSTICAS PRINCIPALES

- **16 canales analógicos** (AN0-AN15)
- **Hasta 1.1 Msps** de tasa de muestreo
- **Múltiples modos** de operación
- **Soporte para promediado** (2-32 muestras)
- **Interrupciones** configurables
- **Buffer circular** para almacenamiento
- **Lectura de temperatura** interna
- **Conversión automática** a voltaje

3. ⚙ CONFIGURACIÓN INICIAL

3.1 Inclusión de la librería

```
#include "ADC.h"
```

3.2 Configuración mínima

```
ADC_Config_t adc_config = ADC_CONFIG_DEFAULT;
ADC_Init(&adc_config);
```

4. 🔄 FUNCIONES DISPONIBLES

4.1 Funciones Básicas

Función	Descripción	Retorno
`ADC_Init()`	Inicializa el ADC con configuración	void
`ADC_ReadSingle()`	Lee un solo canal	uint16_t (0-1023)
`ADC_ReadVoltage()`	Lee voltaje de un canal	float (V)
`ADC_StartConversion()`	Inicia conversión manual	void

4.2 Funciones Avanzadas

Función	Descripción
`ADC_Calibrate()`	Calibra el módulo ADC
`ADC_SetVREF()`	Configura voltajes de referencia
`ADC_ScanChannels()`	Configura modo scan múltiple
`ADC_ConfigureBuffer()`	Habilita buffer circular

Función	Descripción
`ADC_SetInterruptCallback()`	Configura callback para interrupciones

4.3 Funciones Especializadas

Función	Descripción
`ADC_ReadTemperatureCelsius()`	Lee temperatura interna (°C)
`ADC_ReadTemperatureFahrenheit()`	Lee temperatura interna (°F)
`ADC_RawToVoltage()`	Convierte valor crudo a voltaje
`ADC_VoltageToRaw()`	Convierte voltaje a valor crudo

4.4 Funciones de Control

Función	Descripción
`ADC_Enable()` / `ADC_Disable()`	Habilita/deshabilita ADC
`ADC_SelectChannel()`	Selecciona canal activo
`ADC_WaitForConversion()`	Espera bloqueante para conversión
`ADC_IsConversionComplete()`	Verifica estado de conversión

5. 🎛️ MODOS DE OPERACIÓN

5.1 Modo Single (por defecto)

```
adc_config.mode = ADC_MODE_SINGLE;
```

5.2 Modo Continuo

```
adc_config.mode = ADC_MODE_CONTINUOUS;
adc_config.trigger = ADC_TRIGGER_AUTO;
ADC_Init(&adc_config);
ADC_StartConversion();  // Inicia conversión continua
```

5.3 Modo Scan

```
ADC_Channel_t scan_list[] = {ADC_CHANNEL_0, ADC_CHANNEL_1, ADC_CHANNEL_2};
ADC_ScanChannels(scan_list, 3);
```

5.4 Modo con Interrupciones

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

6. 💡 EJEMPLOS DE USO

6.1 Ejemplo Mínimo: Lectura de un canal

```
#include "ADC.h"

int main(void) {
    // 1. Configurar ADC con valores por defecto
    ADC_Config_t config = ADC_CONFIG_DEFAULT;
    ADC_Init(&config);

    // 2. Leer canal AN0
    uint16_t valor = ADC_ReadSingle(ADC_CHANNEL_0);
```

```
    // 3. Convertir a voltaje (asumiendo VREF = 3.3V)
    float voltaje = ADC_RawToVoltage(valor);

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

6.2 Ejemplo: Lectura con Promediado

```
ADC_Config_t config = ADC_CONFIG_DEFAULT;
config.averaging = ADC_AVG_16;      // Promediar 16 muestras
config.sample_rate = 50000;         // 50 kHz
ADC_Init(&config);

// Leer canal con promediado automático
uint16_t valor_promediado = ADC_ReadSingle(ADC_CHANNEL_1);
```

6.3 Ejemplo: Uso con Interrupciones

```
void mi_callback(uint16_t valor_adc) {
    float v = ADC_RawToVoltage(valor_adc);
    // Procesar voltaje aquí
}

int main(void) {
    ADC_Config_t config = ADC_CONFIG_DEFAULT;
    config.interrupt_enable = true;
    config.mode = ADC_MODE_CONTINUOUS;

    ADC_Init(&config);
    ADC_SetInterruptCallback(mi_callback);
    ADC_SelectChannel(ADC_CHANNEL_2);
    ADC_StartConversion();  // Inicia conversiones continuas

    while(1) {
        // El ADC funciona en segundo plano
        // Las interrupciones llaman a mi_callback()
    }
}
```

6.4 Ejemplo: Lectura de Temperatura

```
ADC_Init(&ADC_CONFIG_DEFAULT);

// Leer temperatura en diferentes formatos
uint16_t temp_raw = ADC_ReadTemperature();
float temp_c = ADC_ReadTemperatureCelsius();
float temp_f = ADC_ReadTemperatureFahrenheit();

printf("Temperatura: %d raw, %.1f°C, %.1f°F\n",
       temp_raw, temp_c, temp_f);
```

6.5 Ejemplo: Buffer Circular

```
// Habilitar buffer de 8 muestras
ADC_ConfigureBuffer(true, 8);

// Leer múltiples veces (se almacenan en buffer)
for(int i = 0; i < 10; i++) {
    ADC_ReadSingle(ADC_CHANNEL_3);
    __delay_ms(10);
}

// Acceder a valores del buffer
for(int i = 0; i < 8; i++) {
    uint16_t valor = ADC_GetBufferValue(i);
    // Procesar valor
}
```

6.6 Ejemplo: Configuración Avanzada

```
ADC_Config_t config = {
    .mode = ADC_MODE_CONTINUOUS,
    .format = ADC_FORMAT_INTEGER,
    .trigger = ADC_TRIGGER_TMR1,     // Trigger por Timer1
    .averaging = ADC_AVG_8,
    .sample_rate = 100000,           // 100 kHz
    .vref_positive = 3.3,
    .vref_negative = 0.0,
```

```
    .interrupt_enable = true,
    .auto_sample = true,
    .alternate_mux = false,
    .calibrate = true
};

ADC_Init(&config);
```

7. ⚙ CONFIGURACIONES SOPORTADAS

7.1 Canales Disponibles

```
ADC_CHANNEL_0    // AN0
ADC_CHANNEL_1    // AN1
...
ADC_CHANNEL_15   // AN15
ADC_CHANNEL_TEMP // Sensor de temperatura
ADC_CHANNEL_DAC1 // Salida DAC1
ADC_CHANNEL_FVR  // Fixed Voltage Reference
```

7.2 Modos de Conversión

```
ADC_MODE_SINGLE      // Conversión única
ADC_MODE_CONTINUOUS  // Conversión continua
ADC_MODE_SCAN        // Escaneo de múltiples canales
ADC_MODE_MUX_SAMPLE  // Muestreo multiplexado
```

7.3 Fuentes de Trigger

```
ADC_TRIGGER_MANUAL   // Trigger manual
ADC_TRIGGER_TMR1     // Timer1
ADC_TRIGGER_TMR2     // Timer2
ADC_TRIGGER_TMR3     // Timer3
ADC_TRIGGER_PWM      // PWM
ADC_TRIGGER_EXT_INT  // Interrupción externa
ADC_TRIGGER_AUTO     // Auto-sampling
```

7.4 Niveles de Promediado

```
ADC_AVG_NONE  // Sin promediado (1 muestra)
ADC_AVG_2     // 2 muestras
ADC_AVG_4     // 4 muestras
ADC_AVG_8     // 8 muestras
ADC_AVG_16    // 16 muestras
ADC_AVG_32    // 32 muestras
```

8. ⚠ CONSIDERACIONES IMPORTANTES

8.1 Configuración de Pines

- Los pines analógicos deben configurarse como entrada
- Las funciones digitales deben deshabilitarse
- La librería configura automáticamente los pines usados

8.2 Tiempos de Muestreo

- Tad mínimo: 75ns (3 ciclos de instrucción @ 40MHz)
- Tasa máxima: 1.1 Msps para 10-bit
- Considerar tiempo de establecimiento del sensor

8.3 Referencia de Voltaje

- Por defecto: AVDD (3.3V) y AVSS (0V)
- Se pueden usar referencias externas en AN2/AN3
- Ajustar `vref_positive` y `vref_negative` para conversión precisa

8.4 Consumo de Energía

- Deshabilitar ADC cuando no se use: `ADC_Disable()`
- Mayor tasa de muestreo = mayor consumo
- Considerar usar modo bajo consumo entre lecturas

9. 🔧 SOLUCIÓN DE PROBLEMAS

9.1 Problemas Comunes y Soluciones

Problema	Posible Causa	Solución
Valores erráticos	Pines no configurados	Usar `ADC_SelectChannel()`
Conversión lenta	Tad muy grande	Reducir `AD1CON3bits.SAMC`
Lecturas incorrectas	VREF mal configurado	Verificar `ADC_SetVREF()`
Sin interrupciones	IE no habilitado	Configurar `interrupt_enable=true`

9.2 Verificación de Configuración

```
c
// Imprimir configuración actual
ADC_PrintConfiguration();

// Verificar estado del ADC
if (AD1CON1bits.ADON) {
    // ADC está habilitado
}

// Verificar si hay error
if (AD1CON1bits.DONE) {
    // Última conversión completada
}
```

9.3 Debug con LED

```
// Ejemplo simple para debug
ADC_Init(&ADC_CONFIG_DEFAULT);
uint16_t valor = ADC_ReadSingle(ADC_CHANNEL_0);

if (valor > 512) {
    LED_ON();   // Voltaje > 1.65V
} else {
    LED_OFF();  // Voltaje <= 1.65V
}
```

10. 📊 MEJORES PRÁCTICAS

1. **Calibrar periódicamente** en aplicaciones de precisión
2. **Usar promediado** para reducir ruido
3. **Deshabilitar ADC** en modo bajo consumo
4. **Validar rangos** de voltaje de entrada
5. **Considerar impedancia** de la fuente de señal
6. **Usar buffer** para aplicaciones en tiempo real
7. **Configurar prioridades** de interrupción adecuadamente

11. 📞 SOPORTE

Para problemas o preguntas:

1. Verificar la configuración con `ADC_PrintConfiguration()`
2. Revisar el datasheet del dsPIC33FJ32MC204
3. Asegurar que los pines estén correctamente configurados
4. Verificar la alimentación y referencia de voltaje

Nota: Esta librería está optimizada para el compilador XC16 v1.70 o superior. Para versiones anteriores, pueden requerirse ajustes menores.

Versión: 1.0.0
Fecha: 04/12/2025
Autor: Carlos Olivera Ylla
Compatibilidad: dsPIC33FJ32MC204