7. Señales analógicas: El Convertidor Analógico a Digital.

Un convertidor analógico a digital es imprescindible en las aplicaciones con microcontroladores, pues es una forma de medir una señal analógica entregada por un transductor.

El microcontrolador STM32F103C8T6 tiene 2 convertidores analógicos a digitales de 12 bits por el método de aproximaciones sucesivas, con 10 canales de entrada o pines (PA0 a PA7, PB0 y PB1). En la tarjeta azul o negra, los voltajes de referencia son fijos entre 0 y 3.3 voltios y la señal de conversión debe estar entre este mismo rango. Hay cuatro formas de convertir las señales:

- un solo evento.
- en forma continua.
- en forma escaneada.
- en forma discontinua.

Entre otras características, hay un modo del convertidor ADC llamado watchdog analógico que detecta cuando la señal analógica de entrada está por fuera de un rango de voltaje programable, hay un medidor de temperatura que se puede medir desde el ADC1 o el ADC2 y la forma de tener una lectura múltiple con canales inyectados y normales. El bloque ADC se muestra en la figura 46.

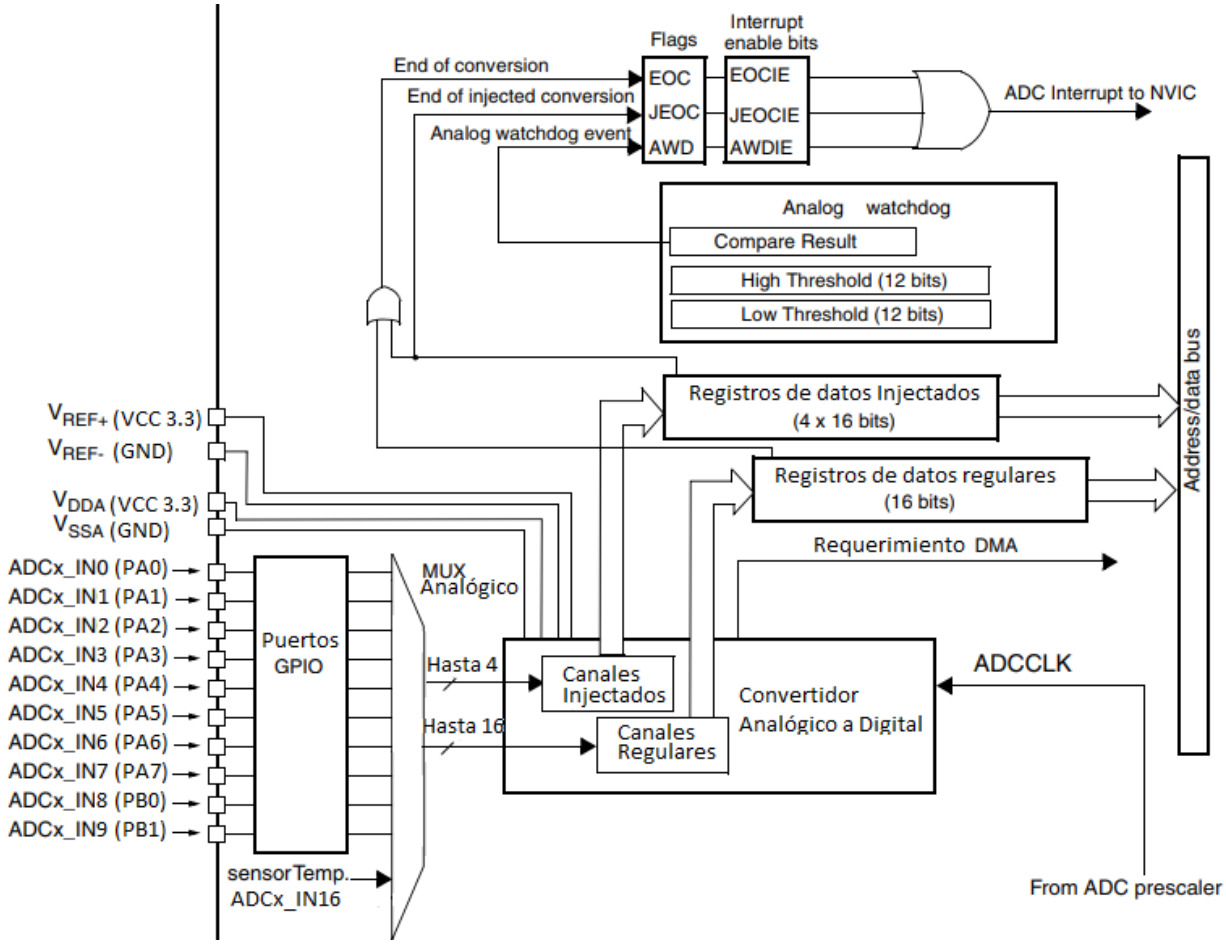

Figura 46. Módulo ADC en el microcontrolador STM32F103C8T6.

Para poder operar el módulo ADC es necesario hacer una configuración básica:

1. Fijar el reloj del ADC en no más de 14 Mhz. De esto depende la velocidad de conversión.
2. Habilitar el reloj del ADC con la función `RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADCy, ENABLE)`, donde *y* del parámetro `RCC_APB2Periph_ADCy` puede ser 1 o 2 para referirse al ADC1 o ADC2, respectivamente. El ADC está en el bus APB2.
3. Configurar los pines del ADC que se van a usar seleccionando los canales de entrada en los puertos Gpio. Primero se debe habilitar el puerto Gpio donde está el pin que se va a usar con `RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOx, ENABLE)`, donde x es el puerto A, B. Luego se debe crear una estructura del tipo `GPIO_InitTypeDef GPIOx_Struct`. Esta estructura debe inicializarse con el valor del pin o pines, `GPIOx_Struct.GPIO_Pin = GPIO_Pin_n`, donde n es el número del pin desde 0 hasta 15. También debe inicializarse la frecuencia, puede ser con `GPIOx_Struct.GPIO_Speed = GPIO_Speed_2MHz`, e inicializarse la entrada al pin como entrada analógica, `GPIOx_Struct.GPIO_Mode = GPIO_Mode_AIN`. Luego de esta información en la estructura, se llama la función `GPIO_Init(GPIOx, &GPIOx_Struct)`.
4. Configurar los detalles de la conversión, según si se quiere un solo canal, varios canales, una sola muestra, muestra contínua, la forma de activación de la conversión,

etc. Para esto se debe crear una estructura del tipo `ADC_InitTypeDef ADC_InitStruct`, y se debe inicializar, preferiblemente con la función `ADC_StructInit(&ADC_InitStruct)`, la cuál inicializa la estructura con la información necesaria. Algunos parámetros de esta estructura necesitan ser cambiados, como el modo de muestreo con `ADC_InitStruct.ADC_ContinuousConvMode = ENABLE`, si es muestreo contínuo, y luego llamar la función que tiene como parámetro esta misma estructura, `ADC_Init(ADCx, &ADC_InitStruct)`, donde x es 1 o 2.

5. Escoger el ADC y el pin de entrada, fijar el tiempo entre muestras y luego iniciar la conversión. Se usa la función `ADC_RegularChannelConfig(ADCx, ADC_Channel_n, 1, ADC_SampleTime_13Cycles5)` para fijar el ADC, el canal, el rango y el tiempo de muestreo, que puede ser 1.5, 7.5, 13.5, 28.5, 41.5, 55.5, 71.5 o 239.5 ciclos. Entre menor sea el tiempo de muestreo, más tomas va a tomar. En este ejemplo el tiempo de muestreo es de 13.5 ciclos. Luego se debe habilitar el ADC con la función `ADC_Cmd(ADCx, ENABLE)`. Para iniciar la conversión, se usa la función `ADC_SoftwareStartConvCmd(ADCx, ENABLE)`.

7.1. Muestras continuas de un solo canal:

La conversión por parte del ADC se puede hacer con una sola conversión o haciendo que el ADC convierta contínuamente la muestra automáticamente. El siguiente ejemplo lee continuamente el canal 0 del ADC1 y lo muestra en el puerto serial.

```
int main(void)
{
    uint16_t adc_value;
    Sysclk_56M();
    UART_Init();

    // configuracion del PIN0 del puerto GPIOA
    GPIO_InitTypeDef GPIOA_Struct;
    GPIOA_Struct.GPIO_Pin  = GPIO_Pin_0; // Canal 0
    GPIOA_Struct.GPIO_Speed = GPIO_Speed_2MHz;
    GPIOA_Struct.GPIO_Mode = GPIO_Mode_AIN;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    GPIO_Init(GPIOA, &GPIOA_Struct);

    // configuración del ADC
    // reloj para ADC (max 14MHz --> 56Mhz/4=14MHz)
    RCC_ADCCLKConfig (RCC_PCLK2_Div4);
    // habilita el reloj del ADC1
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
    ADC_InitTypeDef ADC_InitStruct;
    ADC_StructInit(&ADC_InitStruct);
    // convierte continuamente
    ADC_InitStruct.ADC_ContinuousConvMode = ENABLE;
    ADC_InitStruct.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_Init(ADC1, &ADC_InitStruct); //ADC1
```

```
    // ADC1, canal 0
    ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1,  ADC_SampleTime_13Cycles5);
    // habilita ADC1
    ADC_Cmd(ADC1, ENABLE);

    // calibracion del ADC
    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));

    // habilita ADC1
    ADC_Cmd(ADC1, ENABLE);
    // inicia conversion por software
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
    while(1)
        {
            adc_value = ADC_GetConversionValue(ADC1);
            UART_numero(adc_value);
            for (int i = 0; i < 2000000; ++i) asm("nop");
        }
}
```

En este ejemplo se obtiene el dato convertido con la función `adc_value = ADC_GetConversionValue(ADC1)`.

7.2. Una sola muestra de un solo canal:

Para la toma de una sola muestra de un solo canal se deshabilita el modo de conversión continuo (`ADC_InitStruct.ADC_ContinuousConvMode = DISABLE`). La conversión es de una sola vez, por lo que hay que iniciar la conversión por cada vez que se quiera convertir la señal. La modificación al programa anterior es la siguiente dentro del while(). Observe que es necesario leer la bandera de final de conversión para hacer la lectura. Una vez convertida la señal, se puede reiniciar la conversión por software.

```
while(1)
        {
            while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC)==  RESET);
            adc_value = ADC_GetConversionValue(ADC1);
            UART_numero(adc_value);
            for (int i = 0; i < 2000000; ++i) asm("nop");
            ADC_SoftwareStartConvCmd(ADC1, ENABLE);
        }
```

7.3. Whatchdog analógico con muestra continua:

En este modo se puede programar unos niveles analógicos, bajo y alto de la señal de entrada, para que indiquen cuando una o varias señales en los canales de entrada están por fuera de esos umbrales. Como ejemplo, se toma una muestra en modo continuo y se configura el watchdog analógico con interrupción para que encienda un led cuando la señal de entrada está por fuera de los umbrales.

```
int main(void)
{
    int adc_value;
    Sysclk_56M();
    UART_Init();
    LED_Init();

    // configuracion del PIN0 del puerto GPIOA
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    GPIO_InitTypeDef GPIOA_Struct;
    GPIOA_Struct.GPIO_Pin  = GPIO_Pin_0; // Canal 0
    GPIOA_Struct.GPIO_Speed = GPIO_Speed_2MHz;
    GPIOA_Struct.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIOA_Struct);

    // configuración del ADC
        RCC_ADCCLKConfig (RCC_PCLK2_Div4); // reloj para ADC (max 14MHz --> 56Mhz/4=14MHz)
    // habilita el reloj del ADC1
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
    ADC_InitTypeDef ADC_InitStruct;
    ADC_StructInit(&ADC_InitStruct);// inicializa la estructura
    // convierte continuamente
    ADC_InitStruct.ADC_ContinuousConvMode = ENABLE;
    ADC_InitStruct.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_Init(ADC1, &ADC_InitStruct); //ADC1
    // ADC1 canal 0
    ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1,  ADC_SampleTime_13Cycles5);

    // configura los umbrales del watchdog analogico
    uint16_t HighThreshold= 2000;
    uint16_t LowThreshold= 1000;
    ADC_AnalogWatchdogThresholdsConfig(ADC1, HighThreshold, LowThreshold);
    // escoge el canal para conversion- canal 0
    ADC_AnalogWatchdogSingleChannelConfig(ADC1, ADC_Channel_0);
    // habilita el watchdog analogico
    ADC_AnalogWatchdogCmd(ADC1, ADC_AnalogWatchdog_SingleRegEnable);
    // habilita ADC1
    ADC_Cmd(ADC1, ENABLE);
    // calibracion del ADC
        ADC_ResetCalibration(ADC1);
        while(ADC_GetResetCalibrationStatus(ADC1));
        ADC_StartCalibration(ADC1);
        while(ADC_GetCalibrationStatus(ADC1));
        // habilita ADC1
    ADC_Cmd(ADC1, ENABLE);
    // inicia conversion por software
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);

    // configuración de la interrupción
```

```
    ADC_ITConfig(ADC1, ADC_IT_AWD, ENABLE);
    NVIC_InitTypeDef NVIC_InitStruct;
    NVIC_InitStruct.NVIC_IRQChannel = ADC1_2_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);

    while(1)
        {
            adc_value = ADC_GetConversionValue(ADC1);
            UART_numero(adc_value); // lee y manda lo que lee al puerto serial
            GPIO_SetBits(GPIOB, GPIO_Pin_12);// apaga led
            for (int i = 0; i < 2000000; ++i) asm("nop"); // retardo
        }

    return 0;
}
```

La función de tratamiento a interrupción enciende el led y borra la bandera de interrupción.

```
void ADC1_2_IRQHandler(void)
{
    GPIO_ResetBits(GPIOB, GPIO_Pin_12); // enciende el led
    ADC_ClearITPendingBit(ADC1, ADC_IT_AWD);
    return;
}
```

Como se puede observar, se lee la bandera de activación de watchdog analógico y si la lectura está por fuera de la ventana entre 1.000 y 2.000, se enciende el led.

7.4. Modo de rastreo:

En este modo se hace conversión de señales analógicas una por una de un grupo de entradas previamente programadas.
Para programar el ADC en este modo, se debe inicializar la estructura con `ADC_InitStruct.ADC_ScanConvMode = ENABLE`, previamente al llamado de la función de inicialización, `ADC_Init(ADC1, &ADC_InitStruct)`.
En este modo cada muestra se hace una por una dentro del grupo de señales que se convierten, pero la bandera de final de conversión se activa al final de convertir todas las señales del grupo. En este caso, si se fuera a leer las señales convertidas, solo estará disponible la última. El modo de rastreo solo se puede usar con el módulo DMA. De esta forma las señales convertidas van directamente a memoria. El siguiente ejemplo muestra un rastreo de ocho señales de los canales desde el 0 hasta el 7 usando DMA.

```
// definición de funciones
void UART_Init(void);
void UART_numero(uint32_t numero);
void Sysclk_56M(void);
void LED_Init(void);
```

```
#define ARRAYSIZE 8
volatile uint32_t i;

int main(void)
{
    Sysclk_56M();
    UART_Init();
    LED_Init();

    uint16_t destination[ARRAYSIZE];
    //inicializa arreglo
    for (i=0; i<ARRAYSIZE;i++)
    {
        destination[i]= 0;
        UART_numero(destination[i]);
    }
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
    USART_SendData(USART1,10);

    // activación del reloj del DMA
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
    // crea una estructura para DMA
    DMA_InitTypeDef  DMA_InitStruct;
    //reset el canal channel1 del DMA1 a sus valores de inicio
    DMA_DeInit(DMA1_Channel1);
    // este canal se va a usar para transferencia desde periférico a memoria
    DMA_InitStruct.DMA_M2M = DMA_M2M_Disable;
    //selección de modo circular
    DMA_InitStruct.DMA_Mode = DMA_Mode_Circular;
    //prioridad media
    DMA_InitStruct.DMA_Priority = DMA_Priority_Medium;
    //tamaño del dato de la fuente y el destino= 16bit
    DMA_InitStruct.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
    DMA_InitStruct.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
    //habilitación de incremento automático en destino
    DMA_InitStruct.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStruct.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    //La posición asignada al registro periférico será la fuente
    DMA_InitStruct.DMA_DIR = DMA_DIR_PeripheralSRC;
    //tamaño de los datos que se transfieren
    DMA_InitStruct.DMA_BufferSize = ARRAYSIZE;
    //dirección de inicio de la fuente y el destino
    DMA_InitStruct.DMA_PeripheralBaseAddr = (uint32_t)&ADC1->DR;
    DMA_InitStruct.DMA_MemoryBaseAddr = (uint32_t)destination;
    //programa registros del DMA
    DMA_Init(DMA1_Channel1, &DMA_InitStruct);

    // habilita la interrupción por transferencia completa en el DMA1
    DMA_ITConfig(DMA1_Channel1, DMA_IT_TC, ENABLE);
    // programación de interrupción
    NVIC_InitTypeDef NVIC_InitStruct;
    NVIC_InitStruct.NVIC_IRQChannel = DMA1_Channel1_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);
```

```
    //habilita transferencia en el canal de DMA1
    DMA_Cmd(DMA1_Channel1, ENABLE);

    // habilitación del ADC1 y GPIOA
    RCC_ADCCLKConfig(RCC_PCLK2_Div4);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOA, ENABLE);
    GPIO_InitTypeDef GPIO_InitStruct; // estrutura para configurar los pines
    // configuración de los pines del ADC (PA0 -> canal 0 a PA7 -> canal 7) como entradas analoógicas
    GPIO_StructInit(&GPIO_InitStruct); // inicialización de la estructura
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1| GPIO_Pin_2| GPIO_Pin_3| GPIO_Pin_4| GPIO_Pin_5|
GPIO_Pin_6| GPIO_Pin_7;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStruct);

    ADC_InitTypeDef ADC_InitStruct;
    // configuración del ADC1
    ADC_InitStruct.ADC_Mode = ADC_Mode_Independent;
    // multiples canales
    ADC_InitStruct.ADC_ScanConvMode = ENABLE;
    // modo de conversión contínuo
    ADC_InitStruct.ADC_ContinuousConvMode = ENABLE;
    // sin inicio de conversión externo
    ADC_InitStruct.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    // alineamiento de presentación de datos hacia la derecha
    ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right;
    // 8 canales de conversión
    ADC_InitStruct.ADC_NbrOfChannel = 8;
    // carga información de configuración
    ADC_Init(ADC1, &ADC_InitStruct);
    // configuración de cada canal
    ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_239Cycles5);
    ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 2, ADC_SampleTime_239Cycles5);
    ADC_RegularChannelConfig(ADC1, ADC_Channel_2, 3, ADC_SampleTime_239Cycles5);
    ADC_RegularChannelConfig(ADC1, ADC_Channel_3, 4, ADC_SampleTime_239Cycles5);
    ADC_RegularChannelConfig(ADC1, ADC_Channel_4, 5, ADC_SampleTime_239Cycles5);
    ADC_RegularChannelConfig(ADC1, ADC_Channel_5, 6, ADC_SampleTime_239Cycles5);
    ADC_RegularChannelConfig(ADC1, ADC_Channel_6, 7, ADC_SampleTime_239Cycles5);
    ADC_RegularChannelConfig(ADC1, ADC_Channel_7, 8, ADC_SampleTime_239Cycles5);
    // habilitación de ADC1
    ADC_Cmd(ADC1, ENABLE);
    // habilitación de DMA para ADC
    ADC_DMACmd(ADC1, ENABLE);
    // calibración
    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));

    ADC_SoftwareStartConvCmd(ADC1 , ENABLE);

    while(1)
    {
        if(GPIO_ReadOutputDataBit(GPIOB, GPIO_Pin_12) == Bit_RESET)
        {
            for(i=0; i<ARRAYSIZE;i++)
            {
                UART_numero(destination[i]);
```

```
            }
            while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
            USART_SendData(USART1,10);
            /// apaga el led pb12
            GPIO_SetBits(GPIOB, GPIO_Pin_12);
        }
    }
}

void DMA1_Channel1_IRQHandler(void)
{
  //verifica que la interrupción se hizo con la terminación completa de la transferencia
  if(DMA_GetITStatus(DMA1_IT_TC1))
  {
      GPIO_ResetBits(GPIOB, GPIO_Pin_12);// enciende el led
      // borra la bandera de interrupción
      DMA_ClearITPendingBit(DMA1_IT_GL1);
  }
  return;
}
```

7.5. Modo Canal inyectado.

En este modo el ADC puede tener un grupo regular, como por ejemplo, el que se describe como modo de rastreo. El modo inyectado tiene mayor prioridad que el modo regular, por lo que en cualquier momento de la conversión se puede hacer que la conversión se haga con los canales inyectados. Esta conversión se hace, se toman las muestras y el ADC continúa con los canales regulares. En otras palabras, los canales inyectados se pueden poner a convertir en cualquier momento, aún si hay un grupo de canales regulares programado. El número de canales inyectados máximo es de cuatro.

```
int main(void)
{
    uint16_t ADC_valor0, ADC_valor1;

    Sysclk_56M();
    UART_Init();
    LED_Init();

    // habilitación del ADC1 y GPIOA
    // reloj para ADC (max 14MHz --> 56Mhz/4=14MHz)
    RCC_ADCCLKConfig(RCC_PCLK2_Div4);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOA, ENABLE);
    GPIO_InitTypeDef GPIO_InitStruct; // estrutura para configurar los pines
    // configuración de los pines del ADC como entradas analoógicas
    GPIO_StructInit(&GPIO_InitStruct); // inicialización de la estructura
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStruct);

    ADC_InitTypeDef ADC_InitStruct;
    // configuración del ADC1
```

```
    ADC_InitStruct.ADC_Mode = ADC_Mode_Independent;
    // multiples canales
    ADC_InitStruct.ADC_ScanConvMode = ENABLE;
    // modo de conversión una sola muestra
    ADC_InitStruct.ADC_ContinuousConvMode = DISABLE;
    // sin inicio de conversión externo
    ADC_InitStruct.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    // alineamiento de presentación de datos hacia la derecha
    ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right;
    // 8 canales de conversión
    ADC_InitStruct.ADC_NbrOfChannel = 2;
    // carga información de configuración
    ADC_Init(ADC1, &ADC_InitStruct);
    // configuración de canales inyectados
    ADC_InjectedSequencerLengthConfig(ADC1, 2);
    ADC_InjectedChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_7Cycles5);
    ADC_InjectedChannelConfig(ADC1, ADC_Channel_1, 2, ADC_SampleTime_7Cycles5);
    ADC_ExternalTrigInjectedConvConfig(ADC1, ADC_ExternalTrigInjecConv_None);
    // habilitación de ADC1
    ADC_Cmd(ADC1, ENABLE);
    // calibración
    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));

    ADC_AutoInjectedConvCmd(ADC1, ENABLE);

    while(1)
    {
        ADC_SoftwareStartInjectedConvCmd(ADC1 , ENABLE);

        while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);

        GPIO_ResetBits(GPIOB, GPIO_Pin_12);// enciende el led
        ADC_valor0 = ADC_GetInjectedConversionValue(ADC1, ADC_InjectedChannel_1);
        ADC_valor1 = ADC_GetInjectedConversionValue(ADC1, ADC_InjectedChannel_2);
        UART_numero(ADC_valor0);
        UART_numero(ADC_valor1);
        while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
        USART_SendData(USART1, 10);
        /// apaga el led pb12
        GPIO_SetBits(GPIOB, GPIO_Pin_12);

        GPIO_SetBits(GPIOB, GPIO_Pin_12);// apaga el led
        for(uint32_t j=0; j<2000000; j++)
        {
            asm("nop");
        }
    }
}
```

7.6. Modo dual.

En el modo dual se pueden usar los dos ADC disponibles para toma de una muestra del mismo canal en el mismo ADC o para toma de muestras en grupos.

7.7. Medición de temperatura.

Este procesador es capaz de medir la temperatura del chip, sin embargo en las especificaciones mencionan que el error de conversión puede ser hasta de 45 grados °C, por lo que no es conveniente usarlo como lector de temperatura absoluta, pero si funciona muy bien para leer cambios de temperatura.

7.8. Funciones de la librería del ADC:

La librería SPL de Embitz dispone de un gran repertorio de funciones para configurar el ADC en sus diferentes características. Las funciones disponibles son las siguientes.

```
1.  void ADC_DeInit(ADC_TypeDef* ADCx);
2.  void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct);
3.  void ADC_StructInit(ADC_InitTypeDef* ADC_InitStruct);
4.  void ADC_Cmd(ADC_TypeDef* ADCx, FunctionalState NewState);
5.  void ADC_DMACmd(ADC_TypeDef* ADCx, FunctionalState NewState);
6.  void ADC_ITConfig(ADC_TypeDef* ADCx, uint16_t ADC_IT, FunctionalState NewState);
7.  void ADC_ResetCalibration(ADC_TypeDef* ADCx);
8.  FlagStatus ADC_GetResetCalibrationStatus(ADC_TypeDef* ADCx);
9.  void ADC_StartCalibration(ADC_TypeDef* ADCx);
10. FlagStatus ADC_GetCalibrationStatus(ADC_TypeDef* ADCx);
11. void ADC_SoftwareStartConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState);
12. FlagStatus ADC_GetSoftwareStartConvStatus(ADC_TypeDef* ADCx);
13. void ADC_DiscModeChannelCountConfig(ADC_TypeDef* ADCx, uint8_t Number);
14. void ADC_DiscModeCmd(ADC_TypeDef* ADCx, FunctionalState NewState);
15. void ADC_RegularChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint8_t Rank,
    uint8_t ADC_SampleTime);
16. void ADC_ExternalTrigConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState);
17. uint16_t ADC_GetConversionValue(ADC_TypeDef* ADCx);
18. uint32_t ADC_GetDualModeConversionValue(void);
19. void ADC_AutoInjectedConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState);
20. void ADC_InjectedDiscModeCmd(ADC_TypeDef* ADCx, FunctionalState NewState);
21. void ADC_ExternalTrigInjectedConvConfig(ADC_TypeDef* ADCx, uint32_t
    ADC_ExternalTrigInjecConv);
22. void ADC_ExternalTrigInjectedConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState);
23. void ADC_SoftwareStartInjectedConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState);
24. FlagStatus ADC_GetSoftwareStartInjectedConvCmdStatus(ADC_TypeDef* ADCx);
25. void ADC_InjectedChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint8_t Rank,
    uint8_t ADC_SampleTime);
26. void ADC_InjectedSequencerLengthConfig(ADC_TypeDef* ADCx, uint8_t Length);
27. void ADC_SetInjectedOffset(ADC_TypeDef* ADCx, uint8_t ADC_InjectedChannel, uint16_t
    Offset);
28. uint16_t ADC_GetInjectedConversionValue(ADC_TypeDef* ADCx, uint8_t
    ADC_InjectedChannel);
29. void ADC_AnalogWatchdogCmd(ADC_TypeDef* ADCx, uint32_t ADC_AnalogWatchdog);
```

```
30. void ADC_AnalogWatchdogThresholdsConfig(ADC_TypeDef* ADCx, uint16_t HighThreshold,
    uint16_t LowThreshold);
31. void ADC_AnalogWatchdogSingleChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel);
32. void ADC_TempSensorVrefintCmd(FunctionalState NewState);
33. FlagStatus ADC_GetFlagStatus(ADC_TypeDef* ADCx, uint8_t ADC_FLAG);
34. void ADC_ClearFlag(ADC_TypeDef* ADCx, uint8_t ADC_FLAG);
35. ITStatus ADC_GetITStatus(ADC_TypeDef* ADCx, uint16_t ADC_IT);
36. void ADC_ClearITPendingBit(ADC_TypeDef* ADCx, uint16_t ADC_IT);
```