Taller de Sistemas Embebidos
STM32 MCU – Analog to Digital Conversion

Información relevante

Taller de Sistemas Embebidos

Asignatura correspondiente a la **actualización 2023** del Plan de Estudios 2020 y resoluciones modificatorias, de Ingeniería Electrónica de FIUBA

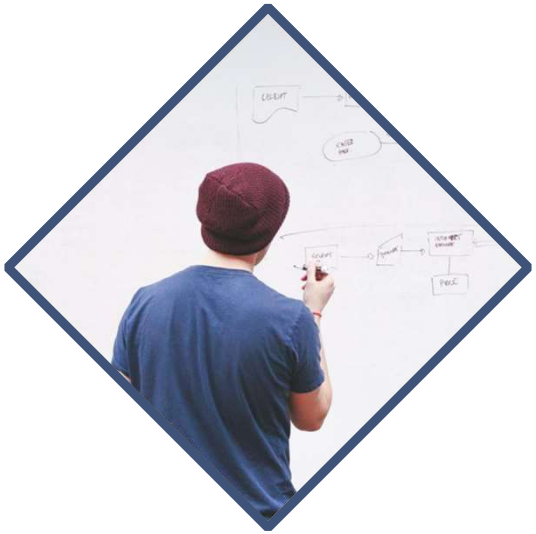
Estructura Curricular de la Carrera

El **Proyecto Intermedio** se desarrolla en la asignatura **Taller de Sistemas Embebidos**, la cual tiene un enfoque centrado en la **práctica propia de la carrera** más que en el desarrollo teórico disciplinar, con eje en la **participación de las y los estudiantes**

Más información . . .

. . . sobre la **actualización 2023** . . . https://www.fi.uba.ar/grado/carreras/ingenieria-electronica/plan-de-estudios

. . . sobre el **Taller de Sistemas Embebidos** . . . https://campusgrado.fi.uba.ar/course/view.php?id=1217



¡Hola!

Soy Juan Manuel Cruz
Taller de Sistemas Embebidos
Consultas a: jcruz@fi.uba.ar

1

Introducción

Actualización 2023 del Plan de Estudios 2020 y resoluciones . . .

Conceptos básicos

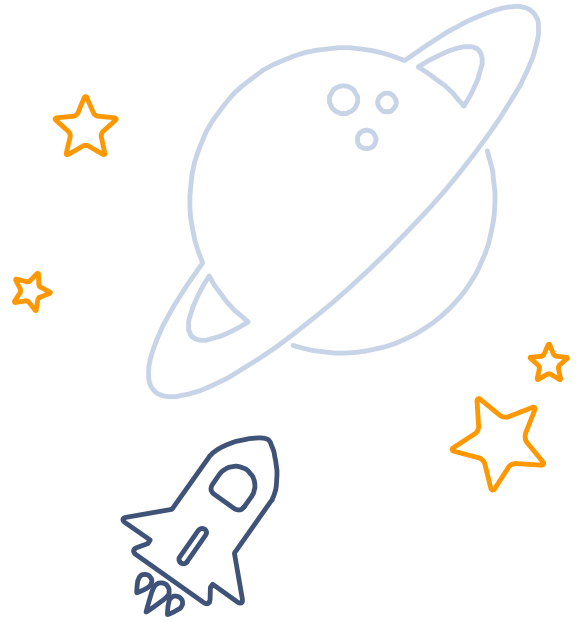
- Referencia:
  - Programming whith STM32, Getting Started with the Nucleo Board and C/C++ - Donal Norris (Author)
  - Chapter 9: Analog-to-Digital Conversion
    - Este capítulo cubre un tema muy importante, la conversión de analógico a digital. Los dispositivos que realmente realizan las conversiones se denominan conversores analógico-digital (ADC). La línea STM de MCU tiene gran variedad de ADCs; sin embargo, todos funcionan y se programan de manera similar.
    - Hay un único ADC utilizado en la MCU STM32F302R8 de la placa de proyecto. Que puede aceptar como entradas analógicas de 1 de 15 líneas externas independientes.

Conceptos básicos

- Una decimosexta entrada ADC, pero está conectada permanentemente a un sensor de temperatura interno y no está disponible para ningún otro uso.
- Los principiantes en el desarrollo embebido, a veces cuestionan la necesidad de un periférico ADC. Pero las mediciones en el mundo real, se toman normalmente utilizando sensores analógicos, que proporcionan un rango continuo de voltajes que son proporcionales al parámetro ambiental que se está monitoreando.
- Por ejemplo, el sensor de temperatura TMP36 de Analog Devices, que se muestra en la Figura 9-1, proporciona un rango de tensión de aproximado de 0,1V a 2,0 V, que es directamente proporcional a un rango de temperaturas aproximado de –40 °C a 150 °C. Esta tensión analógica debe convertirse a un número digital equivalente antes de poder ser procesado por la MCU.
- Ese es el papel de un ADC. La siguiente discusión proporciona una explicación detallada de cómo funciona un ADC de STM.

Figure 9-1 *Analog Devices TMP36 temperature sensor.*

Solución Adecuada



. . . lo más simple posible, previa determinación del objetivo de excelencia a cumplir, obviamente contando con la documentación debida y recurriendo a la metodología de trabajo adecuada

2

Documentación debida

1er Cuatrimestre de 2024, dictado por primera vez . . .

Programming whith STM32, Getting Started with the Nucleo Board and C/C++ - Donal Norris (Author)

Chapter 9: Analog-toDigital Conversion

ADC Functions

- La técnica utilizada en los ADC de STM, se conoce como Registro de Aproximación Sucesiva (SAR). La arquitectura de un ADC SAR es bastante simple, como se puede ver en el diagrama de bloques, como se muestra en la Figura 9-2.
  - Primero se introduce una tensión analógica en un circuito de seguimiento/retención.
  - Este circuito también se denomina comúnmente circuito de muestra/retención, que toma una muestra instantánea de la tensión analógica variable y utiliza esa muestra como valor que se convertirá en un número digital.

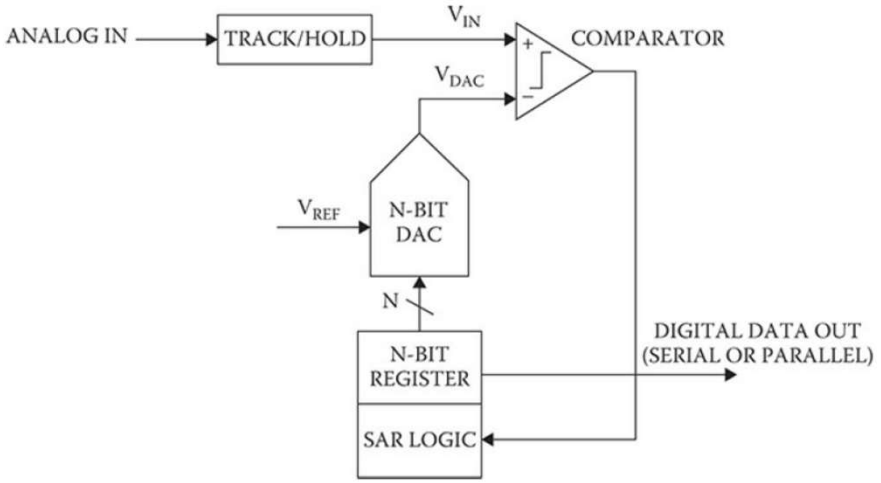


Figure 9-2 *SAR ADC block diagram.*

ADC Functions

- El bloque N-BIT REGISTER, que se muestra en la Figura 9-2, está configurado en un valor medio de escala.
- Este valor depende de la cantidad de bits de la conversión final.
- Para esta discusión, usaré sólo 4 bits para la conversión, lo cual será adecuado para explicar el proceso de conversión.
- En realidad, el STM ADC real utiliza 12 bits. Volviendo al registro significará que se preestablecerá en el registro un valor medio de escala de 1000.

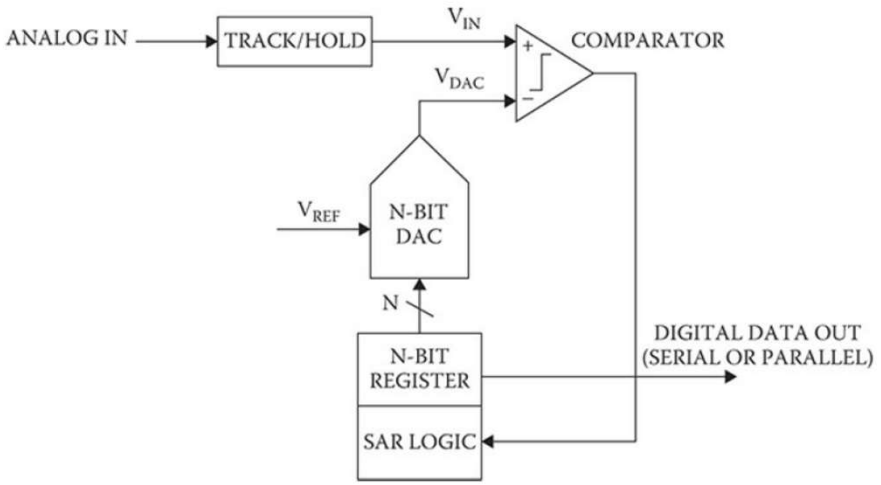


Figure 9-2 *SAR ADC block diagram.*

ADC Functions

- Por lo tanto, este valor obligará al conversor digital a analógico (DAC) que se muestra en la figura como DAC N-BIT a generar una tensión igual a $V_{REF}/2$.
- V_{REF} es un voltaje de referencia utilizado por el ADC y, a menudo, es el mismo que la tensión de alimentación.
- Eso sería 3,3 V para la mayoría de las MCU de STM.

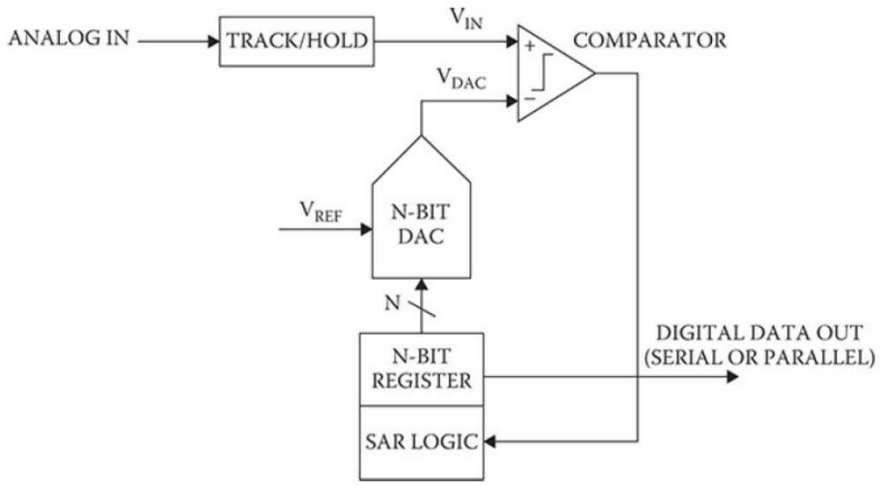


Figure 9-2 *SAR ADC block diagram.*

ADC Functions

- Una comparación entre la tensión analógica muestreada/retenida por el módulo TRACK/HOLD (V_{IN}) y la salida DAC (V_{DAC}) se logra mediante un módulo COMPARADOR que se muestra en la parte superior derecha de la Figura 9-2.
- El COMPARADOR generará un 1 si V_{IN} es mayor que V_{DAC}; de lo contrario, generará un 0.
- Si se genera un 1, se retendrá el bit más significativo en el REGISTRO N-BIT; de lo contrario, se restablecerá a 0.

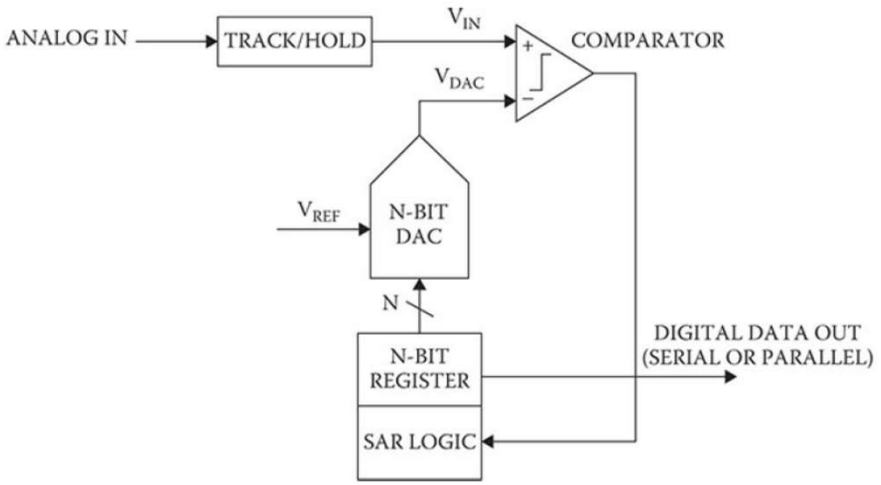


Figure 9-2 *SAR ADC block diagram.*

ADC Functions

- El módulo SAR LOGIC se desplazará 1 bit hacia la derecha y repetirá el proceso de comparación
- Esta secuencia se repite hasta que se alcanza y procesa la posición del bit menos significativo (LSB).
- Una vez que se haya hecho ese punto, el REGISTRO N-BIT contendrá el equivalente digital completo de la tensión de entrada analógica muestreada.
- El módulo SAR LOGIC permitirá que el valor digital completamente convertido se libere en formato paralelo o en serie.

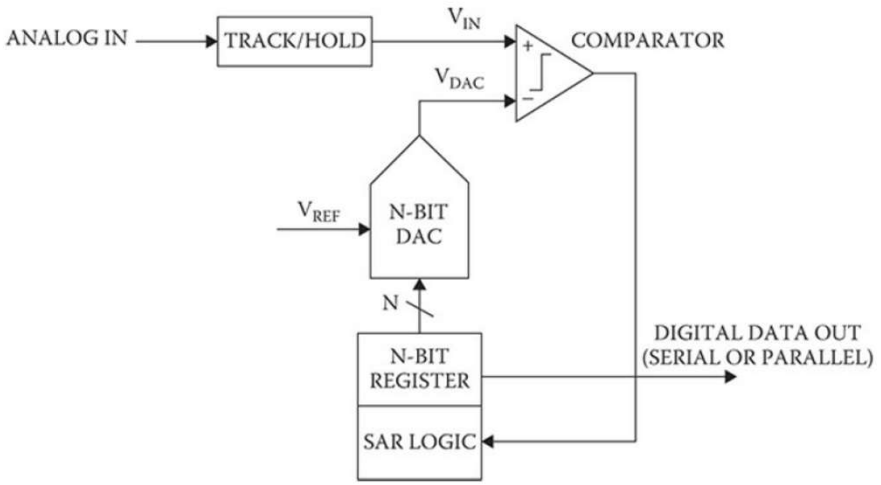


Figure 9-2 *SAR ADC block diagram.*

ADC Functions

- La Figura 9-3 muestra una secuencia de conversión completa de 4 bits.
- El eje vertical representa el voltaje V_{DAC} y el eje horizontal representa el tiempo.
- El resultado de la primera conversión es establecer el MSB en 0 porque V_{IN} es menor que $V_{REF}/2$.
- El módulo SAR LOGIC luego restablecerá el V_{DAC} a $V_{REF}/4$ antes de la siguiente comparación.
- En este caso, V_{IN} es mayor que $V_{REF}/4$ y la posición del bit uno a la derecha del MSB se establecerá en 1.

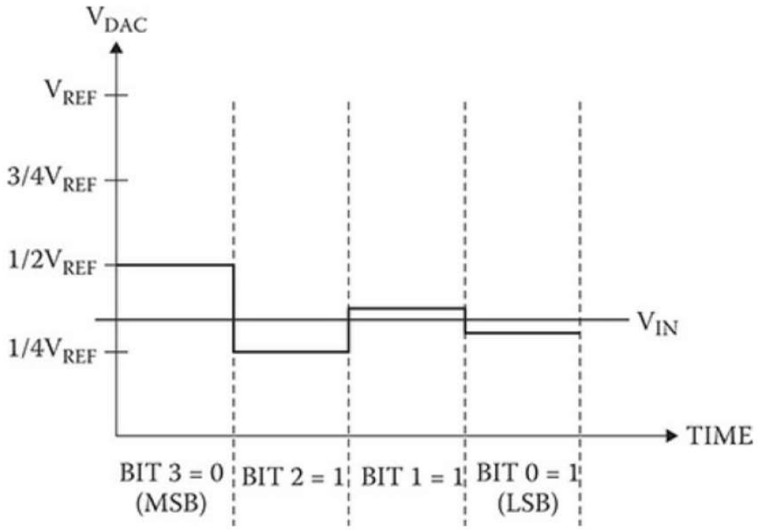


Figure 9-3 *Four-bit conversion sequence.*

ADC Functions

- A continuación, el módulo SAR LOGIC dividirá la diferencia entre $V_{REF}/2$ y $V_{REF}/4$, que es $3V_{REF}/8$.
- Esta nueva comparación da como resultado un 0 que sale del módulo COMPARADOR.
- En la actualidad, el módulo SAR LOGIC cambia la tensión de V_{DAC} en algún múltiplo de $V_{REF}/24$ o $V_{REF}/16$ para cada operación de comparación
- Esto da como resultado una forma de señal escalonada convergente, como se muestra en la Figura 9-3.

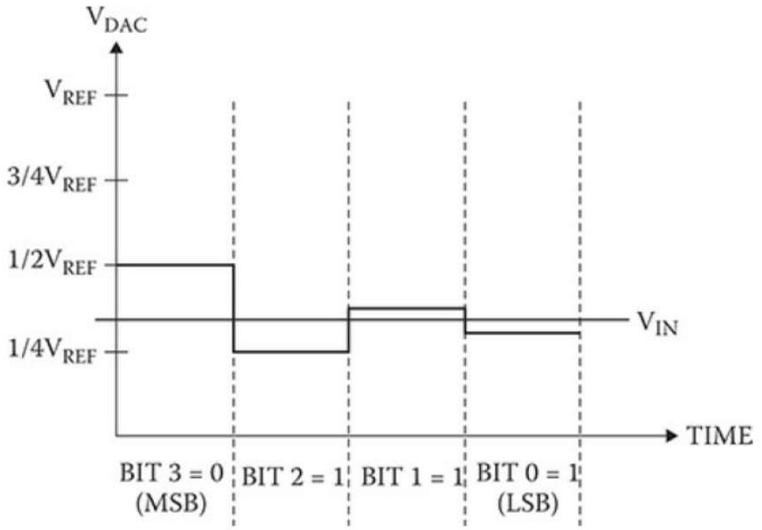


Figure 9-3 *Four-bit conversion sequence.*

ADC Functions

- Algunos múltiplos del valor del paso se suman o restan del V_{DAC} dependiendo del resultado de la comparación de bits anterior.
- En este caso, el valor digital resultante que posee el REGISTRO N-BIT es 0101, que es 5 en formato decimal.
- Todo esto tiene sentido si equiparas V_{REF} con 16, donde cada paso en el rango V_{REF} es 1. La línea V_{IN} en la figura estaría entonces aproximadamente en el nivel 5.

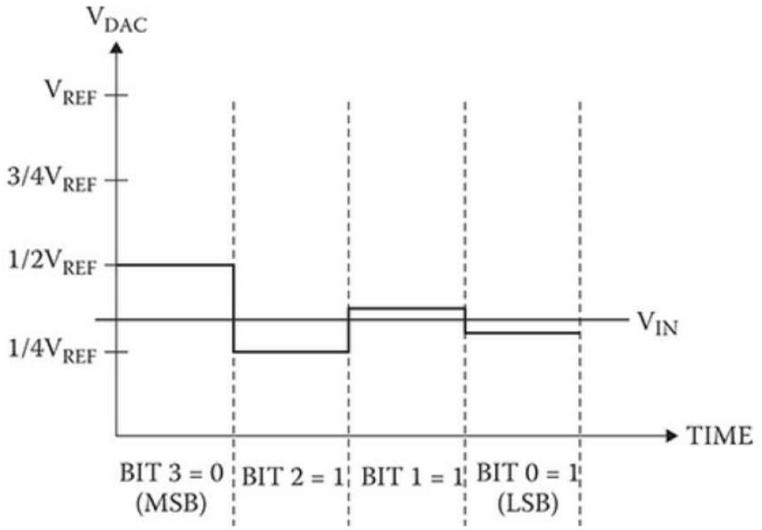


Figure 9-3 *Four-bit conversion sequence.*

ADC Functions

- Se puede ver que el proceso SAR converge con bastante rapidez hacia un valor final.
- En este caso, se necesitaron cuatro comparaciones, mientras que un ADC de 12 bits requeriría 12 comparaciones.
- La buena noticia es que todas las operaciones del módulo SAR LOGIC se realizan en firmware y no se requieren instrucciones de la MCU más que para iniciar la conversión.

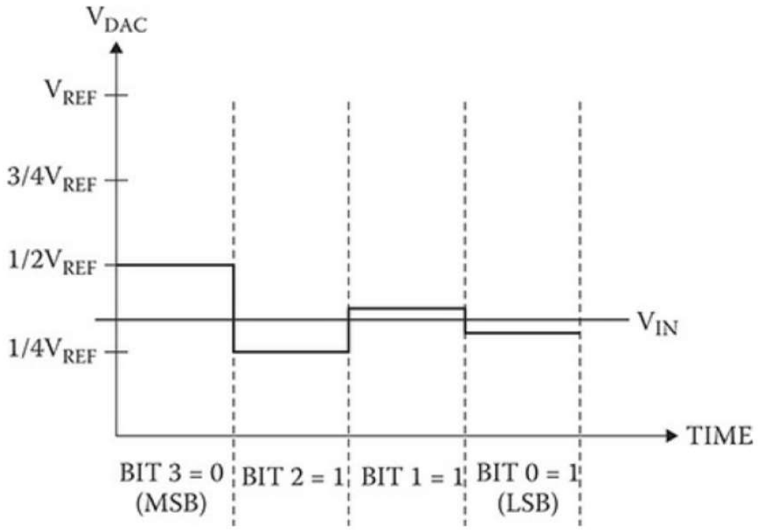


Figure 9-3 *Four-bit conversion sequence.*

ADC Module with HAL

- La HAL utiliza una estructura C similar para declarar periféricos ADC, como se señaló anteriormente, al analizar los pines GPIO y los periféricos del temporizador.
  - La siguiente estructura C está definida para un ADC.
  - Cada miembro de la estructura se analizará por separado después de la definición de la estructura.

```
typedef struct {
    ADC_TypeDef          *Instance;
    ADC_InitTypeDef      Init;
    __IO uint32_t        NbrOfCurrentConversionRank;
    DMA_HandleTypeDef    *DMA_Handle;
    HAL_LockTypeDef      Lock;
    __IO  uint32_t       State;
    __IO uint32_t        ErrorCode;
} ADC_HandleTypeDef;
```

The ADC_HandleTypeDef struct members are briefly defined in the following list:

ADC_TypeDef *Instance;—This is a pointer to the active ADC. For example, ADC1 would reference the first ADC.

ADC_InitTypeDef Init;—References the ADC_InitTypeDef C struct, which is used to configure the selected ADC. This struct will be discussed shortly.

__IO uint32_t NbrOfCurrentConversionRank;—References the rank or channel number in a conversion group. This will be shortly explained in detail.

DMA_HandleTypeDef *DMA_Handle;—This is a pointer to a DMA handler used for ADC to DMA conversions. This will be explained in the ADC/DMA section.

HAL_LockTypeDef Lock;—A locking object used for the ADC process.

__IO uint32_t State;—A state object used with the ADC communication process.

__IO uint32_t ErrorCode;—An object that can hold an error value that might arise.

An ADC declared using the ADC_HandleTypeDef struct must further be configured using the ADC_InitTypeDef C struct, which is defined as follows:

```
typedef struct {
uint32_t ClockPrescaler;
uint32_t Resolution;
uint32_t DataAlign;
uint32_t ScanConvMode;
uint32_t EOCSelection
uint32_t ContinuousConvMode;
uint32_t NbrOfConversion;
uint32_t DiscontinuousConvMode;
uint32_t NbrOfDiscConversion;
uint32_t ExternalTrigConv;
uint32_t ExternalTrigConvEdge;
uint32_t DMAContinuousRequests;
} ADC_InitTypeDef;
```

ADC Module with HAL

- Los miembros de la estructura ADC_InitTypeDef se definen en la siguiente lista:
  - uint32_t ClockPrescaler; Esto define la velocidad de reloj del ADC (ADCCLK) utilizada en el ADC.
    - Esta frecuencia de reloj establece indirectamente la frecuencia de muestreo máxima posible con el ADC.
    - El preescalador ADC tiene tasas de divisor preestablecidas que comienzan en uno y progresan en múltiplos de 2, es decir, 2, 4, 6 u 8.
    - La velocidad del reloj del ADC afecta a todos los ADC de MCU. Este parámetro puede ser cualquier valor de lo siguiente que define ADC_ClockPrescaler:

```
ADC_CLOCK_SYNC_PCLK_DIV1
ADC_CLOCK_SYNC_PCLK_DIV2
ADC_CLOCK_SYNC_PCLK_DIV4
ADC_CLOCK_SYNC_PCLK_DIV6
ADC_CLOCK_SYNC_PCLK_DIV8
```

ADC Module with HAL

- uint32_t Resolución; Este valor establece la resolución del ADC.
  - Este parámetro puede ser cualquier valor de lo siguiente que ADC_Resolution define:

```
ADC_RESOLUTION_12B - 12 bits
ADC_RESOLUTION_6B - 6 bits
```

  - Existe una relación definida entre la resolución y las conversiones máximas posibles que se pueden lograr por segundo. Esta regla es que cuanto mayor sea la resolución, menor será la tasa de conversión máxima.
- uint32_t Alineación de datos; Este valor establece cómo se alinean los bits en el registro N-Bit.
  - Este parámetro puede ser cualquier valor de lo siguiente que ADC_DATAALIGN define:

```
ADC_DATAALIGN_LEFT—MSB to the left
ADC_DATAALIGN_RIGHT—MSB to the right
```

ADC Module with HAL

- uint32_t ScanConvMode; Este valor especifica el modo ADC.
  - Es único o continuo. Este parámetro puede ser cualquiera de los siguientes valores definidos por ADC_SCAN:

    ADC_SCAN_DISABLE—Scan mode disabled
    ADC_SCAN_ENABLE—Scan mode enabled

- uint32_t EOCSelection; Este valor establece el tipo de flag que indica el final de una conversión (EOC).
  - El valor del tipo de flag también depende del modo ADC, que puede ser único o continuo.
  - Este parámetro se puede establecer en cualquiera de las siguientes definiciones ADC_EOC:

    ```
    ADC_EOC_SINGLE_CONV—Single conversions
    ADC_EOC_SEQ_CONV—Continuous conversion
    ```

ADC Module with HAL

- uint32_t ModoConvContinuo; Especifica si la conversión se realiza en modo único (una conversión) o en modo continuo para un grupo normal, después de que se produjo el trigger seleccionado (inicio del software o trigger externo).
  - Este parámetro se puede establecer en ENABLE o DISABLE.
- uint32_t NbrOfConversion; Este valor especifica el número de rangos o canales que se convertirán dentro del secuenciador de grupo normal.
  - Para utilizar un secuenciador de grupo normal y convertir varios rangos, se debe habilitar el parámetro 'ScanConvMode'.
  - Este parámetro debe ser un número entre Min_Data = 1 y Max_Data = 16.

ADC Module with HAL

- uint32_t ModoConvDiscontinuo; Este valor especifica si la secuencia de conversión del grupo regular se realiza en secuencia completa/secuencia discontinua (secuencia principal subdividida en partes sucesivas).
  - El modo discontinuo se utiliza sólo si el secuenciador está habilitado (parámetro 'ScanConvMode'). Si el secuenciador está deshabilitado, este parámetro se descarta.
  - El modo discontinuo sólo se puede habilitar si el modo continuo está deshabilitado. Si el modo continuo está habilitado, esta configuración de parámetro se descarta.
  - Este parámetro se puede establecer en ENABLE o DISABLE.

ADC Module with HAL

- uint32_t NbrOfDiscConversion; Este valor especifica el número de conversiones discontinuas en las que se subdividirá la secuencia principal del grupo regular (parámetro NbrOfConversion).
  - Si el parámetro 'DiscontinuousConvMode' está deshabilitado, este parámetro se descarta.
  - Este parámetro debe ser un número entre Min_Data = 1 y Max_Data = 8.
- uint32_t ExternalTrigConv; Este valor selecciona el evento externo utilizado para desencadenar el inicio de la conversión del grupo normal.
  - Si se establece en ADC_SOFTWARE_START, los activadores externos están deshabilitados. Si se configura en una fuente de activación externa, la activación se produce en el flanco ascendente del evento de forma predeterminada.

ADC Module with HAL

- Este parámetro puede ser un valor en el formato ADC_External_trigger_Source_Regular y es uno de los siguientes define:

```
ADC_EXTERNALTRIGCONVEDGE_NONE
ADC_EXTERNALTRIGCONVEDGE_RISING
ADC_EXTERNALTRIGCONVEDGE_FALLING
ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING
```

- uint32_t DMAContinuousRequests; Este valor especifica si las solicitudes de DMA se realizan en modo de una sola vez (la transferencia de DMA se detiene cuando se alcanza el número de conversiones) o en modo continuo (transferencia de DMA ilimitada, independientemente del número de conversiones).
  - Nota: En modo continuo, DMA debe configurarse en modo circular. De lo contrario, se activará una saturación cuando se alcance el puntero máximo del búfer DMA.

ADC Module with HAL

- Nota: Este parámetro debe modificarse cuando no hay ninguna conversión en curso en los grupos regular e inyectado (ADC deshabilitado o ADC habilitado sin modo continuo o disparador externo que pueda iniciar una conversión).
- Este parámetro se puede establecer en ENABLE o DISABLE.

- No se desanime demasiado por el alcance de los listados anteriores. La declaración y configuración real de un periférico STM ADC es sencilla y razonablemente fácil de entender, como se dará cuenta en breve al examinar una demostración. Sin embargo, todavía necesito cubrir algunos temas restantes del ADC antes de proceder a una demostración real.

ADC Conversion Modes

- Existen algunas formas comunes de utilizar uno o más canales de entrada con un ADC.
- Estas formas se conocen como modos de conversión y a continuación cubriré los modos de conversión más populares.
  - Single Channel/Single Conversion
    - Es el modo más simple. La Figura 9-4 muestra un diagrama para este modo.
    - En este modo, el ADC toma o muestrea con mayor precisión la tensión analógica presente en una línea o canal de entrada seleccionado y convierte esa tensión en un número digital. Luego, el número se lee del ADC y se utiliza en cualquier aplicación que lo requiera.

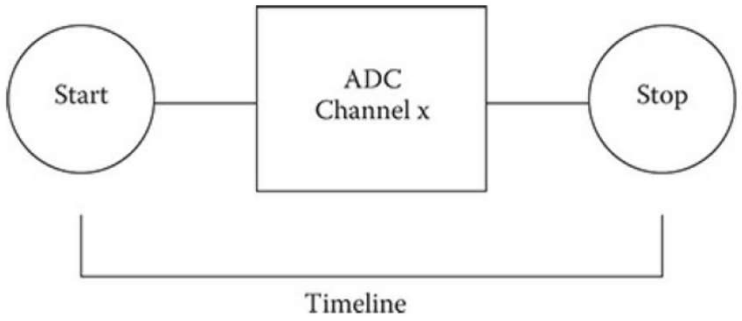


Figure 9-4 *Single channel/single conversion ADC mode block diagram.*

ADC Conversion Modes

- Multi-channel Scan/Single Conversion
  - Este modo es un poco más complejo que el anterior.
  - En este modo, se muestrean múltiples líneas analógicas de entrada y los niveles de voltaje muestreados luego se convierten en números digitales.
  - El diagrama de bloques para este modo se muestra en la Figura 9-5.

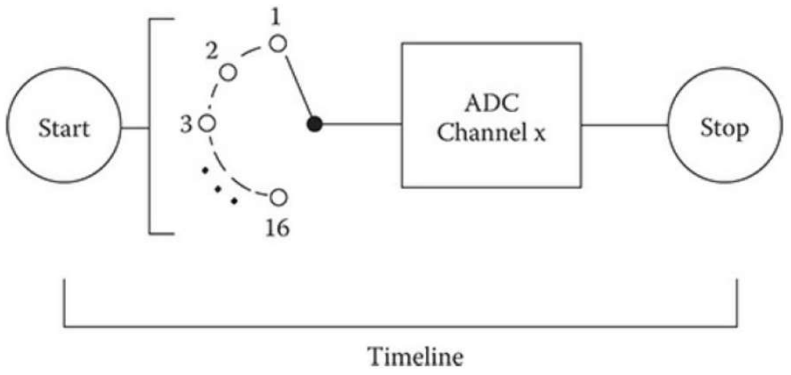


Figure 9-5 *Multi-channel scan/single conversion ADC mode block diagram.*

ADC Conversion Modes

- Una de las características excepcionalmente interesantes de este modo es que cada canal es independiente y puede configurarse para tener diferentes frecuencias de muestreo, fuentes de trigger y otros elementos personalizados.
- Esta característica se implementa mediante el uso de rangos. Al utilizar rangos, la MCU no tiene que detener y reconfigurar cada canal para su configuración personalizada. Este modo solo realiza un escaneo y los números de resultados a menudo se almacenan directamente en una memoria mediante un acceso directo a la memoria (DMA).

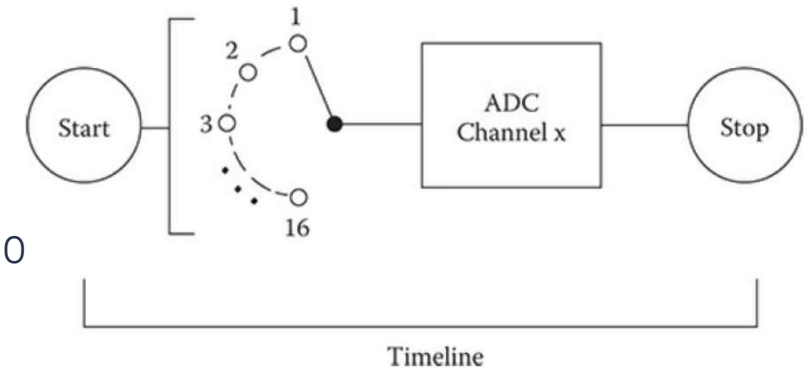


Figure 9-5 *Multi-channel scan/single conversion ADC mode block diagram.*

ADC Conversion Modes

- Single Channel/Continuous Conversion
  - En este modo, se muestrea continuamente un solo canal y los números digitales resultantes se almacenan continuamente en la memoria. Este modo casi siempre usa DMA para aliviar la carga excesiva en la MCU.
  - Este modo se utiliza normalmente para el monitoreo en tiempo real de sensores u otros dispositivos ambientales.
- Multi-Channel Scan/Continuous Conversion
  - En este modo, se muestrean continuamente múltiples canales y los números digitales resultantes se almacenan en la memoria. Este modo exige que se utilice DMA para aliviar una carga de cálculo intolerable en la MCU.
  - Este modo se utiliza normalmente para el monitoreo integral en tiempo real de sensores u otros dispositivos ambientales.

ADC Conversion Modes

- Channels, Groups, and Ranks
  - La cantidad de canales que se pueden convertir varía según la MCU STM particular que se utilice. Ese número se establece en 16 para la MCU STM32F302R8 utilizada en la placa del proyecto Nucleo. Se pueden configurar secuencias de conversiones de canales dentro de colecciones independientes conocidas como grupos. Los canales pueden organizarse en cualquier orden dentro de un grupo específico.
  - Los canales de entrada están basados en hardware y, por lo tanto, están fijos y vinculados a pines MCU específicos, es decir, IN0 es el primer canal, IN1 el segundo, y así sucesivamente. Sin embargo, se pueden reordenar lógicamente para formar una secuencia de muestreo personalizada dentro de un grupo. La reordenación de canales se realiza asignando el canal de entrada a un índice que va del 1 al 16. Este índice se llama rango en el HAL.

ADC Conversion Modes

- La Figura 9-6 muestra este concepto. Aunque el canal IN3 en la figura es fijo, por ejemplo, está conectado al pin PA3 en una MCU STM32F302R8, se puede asignar al rango 1 para convertirlo en el primer canal en ser muestreado.

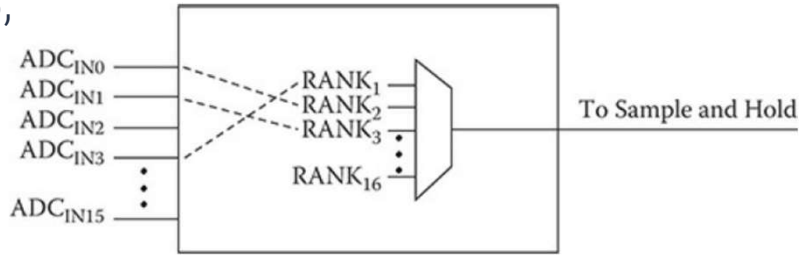


Figure 9-6 *ADC diagram showing channels and ranks.*

- La siguiente estructura C está definida para ADC_ChannelConfTypeDef. Cada miembro de la estructura se analizará por separado después de la definición de la estructura.

```
typedef struct {
uint32_t Channel;
uint32_t Rank;
uint32_t SamplingTime;
uint32_t Offset;
} ADC_ChannelConfTypeDef;
```

ADC Conversion Modes

- Los miembros de la estructura ADC_ChannelConfTypeDef se definen en la siguiente lista:
  - uint32_t Channel; Este valor especifica el canal que se configurará en el grupo regular de ADC. Este parámetro puede ser cualquier valor de lo siguiente que ADC_CHANNEL define:

```
ADC_CHANNEL_0
to
ADC_CHANNEL_15
ADC_CHANNEL_VREFINT
ADC_CHANNEL_VBAT
```

  - uint32_t Rango; Este valor especifica el rango en el secuenciador de grupo normal. Este parámetro debe ser un número entre Min_Data = 1 y Max_Data = 16: Este parámetro puede ser cualquier valor de lo siguiente que define ADC_REGULAR_RANK:

```
ADC_REGULAR_RANK_1
to
ADC_REGULAR_RANK16
```

ADC Conversion Modes

- uint32_t Tiempo de muestreo; El valor del tiempo de muestreo que se establecerá para el canal seleccionado en unidades de ciclos de reloj ADC. El tiempo total de conversión es la suma del tiempo de muestreo y el tiempo de procesamiento (12 ciclos de reloj ADC con resolución ADC de 12 bits, 11 ciclos a 10 bits, 9 ciclos a 8 bits, 7 ciclos a 6 bits). Este parámetro puede ser cualquier valor de lo siguiente que ADC_SAMPLETIME define:

PRECAUCIÓN: Este parámetro actualiza la propiedad del parámetro del canal, que se puede utilizar en grupos regulares y/o inyectados. Si este mismo canal ha sido configurado previamente en el otro grupo (regular/inyectado), se actualizará a la última configuración. En caso de utilizar canales de medición internos (VrefInt/Vbat/TempSensor), se deben respetar las restricciones de tiempo de muestreo (el tiempo de muestreo se puede ajustar en función de la frecuencia del reloj del ADC y la configuración del tiempo de muestreo).

```
ADC_SAMPLETIME_3CYCLES
ADC_SAMPLETIME_15CYCLES
ADC_SAMPLETIME_28CYCLES
ADC_SAMPLETIME_56CYCLES
ADC_SAMPLETIME_84CYCLES
ADC_SAMPLETIME_112CYCLES
ADC_SAMPLETIME_144CYCLES
ADC_SAMPLETIME_480CYCLES
```

- uint32_t Desplazamiento; Reservado para uso futuro, se puede establecer en 0.

ADC Demonstration

- El ADC, como muchos periféricos STM, se puede controlar mediante uno de tres métodos:
  - Encuesta (Polling)
  - Interrupción (Interrupt)
  - Acceso Directo a Memoria (DMA)
- Esta primera demostración utilizará el Encuesta (Polling) como tipo de control porque es el más simple y permitirá centrarse principalmente en el proceso ADC.
  - El paso inicial es crear una referencia a una estructura ADC_HandleTypeDef y luego asignar los valores apropiados a los miembros de la estructura que definirán la instancia de ADC que se implementará.
  - Esta nueva instancia se configurará a continuación utilizando la estructura ADC_InitTypeDef, que requiere una referencia a la estructura ADC_HandleTypeDef ahora definida.

ADC Demonstration

- Será el momento de inicializar e iniciar el ADC una vez que se hayan definido todos los miembros de la estructura ADC_InitTypeDef. Los comandos de inicialización e inicio se logran mediante estas llamadas:

```
HAL_ADC_Init()
HAL_ADC_Start()
```

- Luego, el ADC realizará una conversión si se seleccionó el modo único o continuará con un flujo de conversiones si se seleccionó el modo continuo. En el caso del modo único, primero se debe llamar al método HAL_ADC_Stop() para detener el hilo ADC y luego se debe llamar nuevamente al método HAL_ADC_Start() para realizar otra conversión.
- Cualquier aplicación que desee acceder a los datos convertidos debe utilizar el siguiente método al utilizar el sondeo para determinar cuándo ha finalizado una conversión de ADC y los datos están disponibles:

```
HAL_StatusTypeDef HAL_ADC_PollForConversion(ADC_Handle-
TypeDef* hadc, uint32_t Timeout);
```

ADC Demonstration

- Los argumentos del método son un puntero a la estructura ADC_HandleTypeDef y un número entero llamado Timeout que representa el número máximo de milisegundos para esperar a que los datos estén disponibles en el registro ADC.
- También es posible utilizar la definición HAL_MAX_DELAY para el valor de tiempo de espera, lo que hará que el sistema espere indefinidamente datos. No recomendaría ese enfoque porque tiene el potencial de "colgar" el sistema sin ningún síntoma aparente.
- Debe utilizar el siguiente método para transferir los datos del registro ADC a cualquier aplicación de usuario:

```
uint32_t HAL_ADC_GetValue(ADC_HandleTypeDef* hadc);
```

- Ahora es apropiado hablar del sensor de temperatura TMP36 de Analog Devices, que presenté al comienzo del capítulo. Este sensor será la fuente de tensión analógica utilizada para la demostración del ADC.

ADC Demonstration

- Analog Devices TMP36 Temperature Sensor
  - En la Figura 9-1 se muestra una imagen de este sensor. Viene en una cápsula plástica TO-92 normal con tres pines, como se muestra en la figura.
  - Sin embargo, contiene bastantes componentes internos que le permiten generar una tensión de CC que es proporcional a la temperatura ambiente que rodea su cápsula.
  - Este sensor también es muy económico y está disponible en varias fuentes en línea.
  - Las especificaciones del sensor se enumeran a continuación:
    - Tamaño: encapsulado TO-92 (aproximadamente 0,2" × 0,2" × 0,2") con tres cables

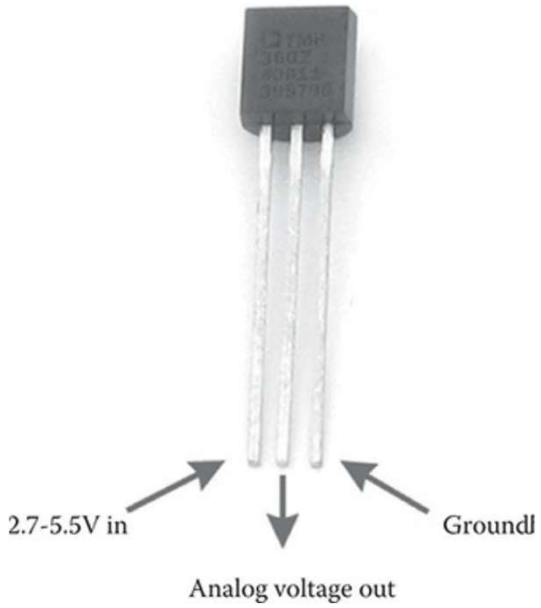


Figure 9-7 *TMP36 lead connections.*

ADC Demonstration

- Rango de temperatura: –40 °C a 150 °C/–40 °F a 302 °F
- Rango de salida: 0,1 V (–40 °C) a 2,0 V (150 °C) (precisión decreciente después de 125 °C)
- Fuente de alimentación: 2,7 V a 5,5 V únicamente, con un consumo de corriente típico de 0,05 mA
- La Figura 9-7 muestra a qué se debe conectar cada pin.
- Los pines deben conectarse al protoboard Arduino, como se detalla en la Tabla 9-1.
- La temperatura es muy fácil de calcular usando la siguiente ecuación:
  - °C = [(Vout en mV) – 500] /10

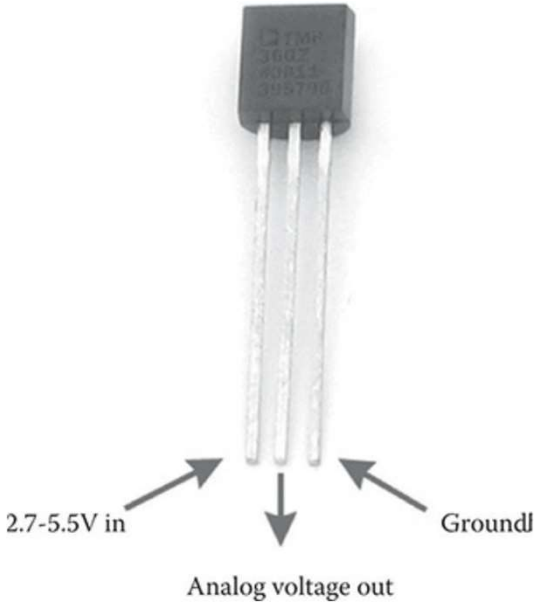


Figure 9-7 *TMP36 lead connections.*

ADC Demonstration

- La temperatura equivalente en °F también se calcula fácilmente usando esta ecuación de conversión:
  - °F = (9 * °C/5) + 32
- Ahora es el momento de discutir el archivo main.c ya que se han introducido y explicado todos los elementos precursores.

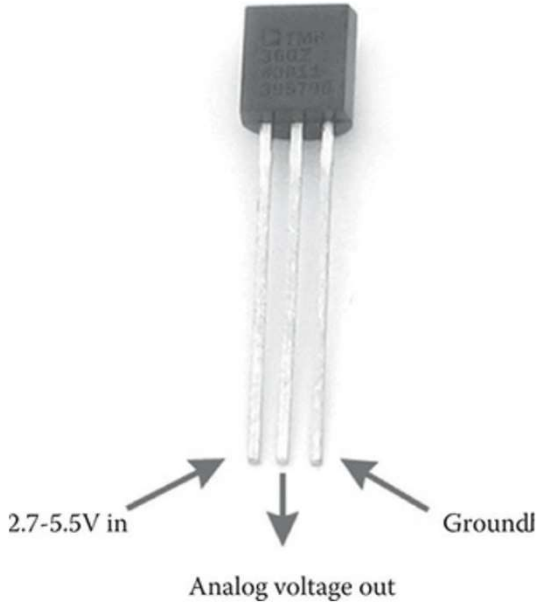


Figure 9-7 *TMP36 lead connections.*

ADC Demonstration

Enabling ADC1

- Ahora necesitaré habilitar la función alternativa ADC1 para tener la función de conversión de analógico a digital. Configura ADC1 al crear un nuevo proyecto y se muestra la ventana Pinout.
- La Figura 9-8 muestra cómo se habilita el periférico ADC1 seleccionando IN1 de extremo único en el menú desplegable ADC1 IN1.
- La aplicación CubeMX insertará automáticamente un método de inicialización estático void MX_ADC1_Init(void) en el archivo main.c cuando la funcionalidad ADC1 esté habilitada.

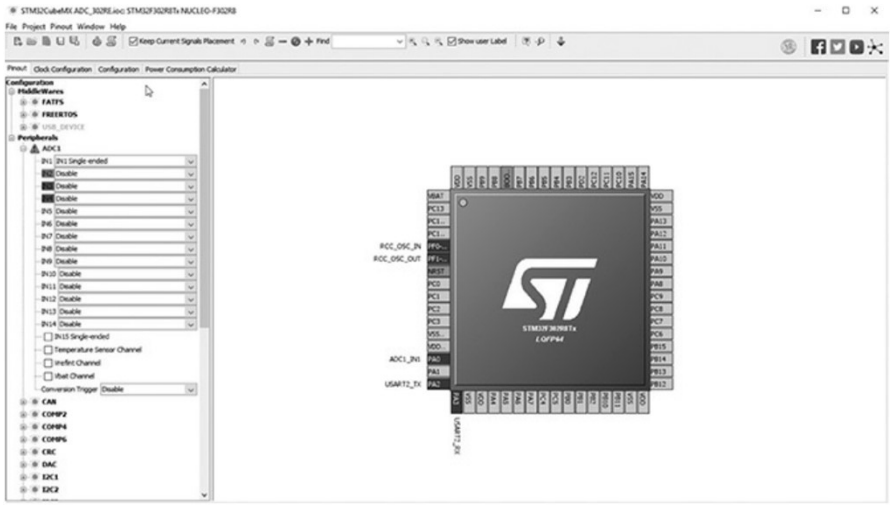


Figure 9-8 *Enabling the IN1 channel in ADC1.*

ADC Demonstration

- Los parámetros de ADC apropiados se configuran automáticamente según los parámetros que establezca en la vista de árbol de IP de ADC1, como se muestra en el siguiente fragmento de código:

```
static void MX_ADC1_Init(void)
{
  ADC_ChannelConfTypeDef sConfig;
  // Common config
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = ENABLE;
  hadc1.Init.NbrOfDiscConversion = 1;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc1.Init.LowPowerAutoWait = DISABLE;
  hadc1.Init.Overrun = ADC_OVR_DATA_OVERWRITTEN;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }
  // Configure Regular Channel
  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = 1;
  sConfig.SingleDiff = ADC_SINGLE_ENDED;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  sConfig.OffsetNumber = ADC_OFFSET_NONE;
  sConfig.Offset = 0;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }
}
```

ADC Demonstration

Connecting ADC1 to the TMP36 Sensor

- La Figura 9-9 muestra cómo conecté el sensor de temperatura TMP36 al periférico interno ADC1 usando una placa sin soldadura, cables de puente y el protoboard Arduino.
- Las conexiones de los pines del TMP36 se detallaron anteriormente en la Tabla 9-1. Necesitará tres cables de puente para conectar el sensor al protoboard Arduino.

TMP36 lead	Arduino protoboard (refer to Figure 5-11)
Power	CN6, pin 5 (+5 V)
Ground	CN6, pin 6 (GND)
Output	CN8, pin 1 (PA0)

Table 9-1 *TMP36 Lead Connections*

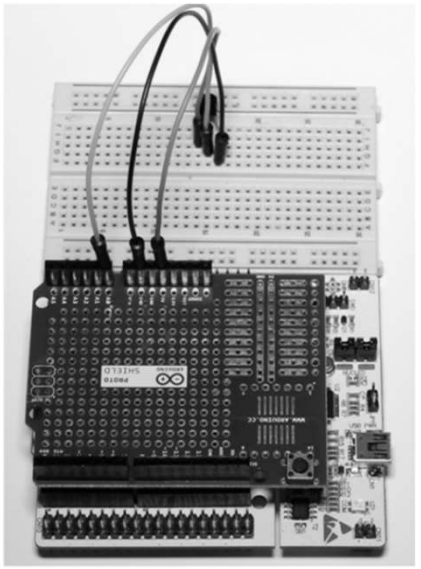


Figure 9-9 *Interconnections between the TMP36 sensor and the Arduino protoboard.*

- En este punto, se han introducido todos los requisitos previos necesarios para demostrar un programa ADC real.

ADC Demonstration

- ADC Demonstration Software
  - El siguiente programa se basa en el programa USART que se muestra en el capítulo anterior. Este paso se tomó porque se requiere que la salida ADC procesada se muestre en una ventana de terminal. El archivo main.c ahora tiene un nuevo método que configura e inicializa el periférico interno ADC1.
  - También hay un nuevo código funcional que se ejecuta dentro del bucle eterno. Este código se muestra en el siguiente fragmento de código con explicaciones detalladas después del listado:

```
while (1)
{
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
          HAL_ADC_Start(&hadc1);
          HAL_ADC_PollForConversion(&hadc1, 100);
          raw = (double) HAL_ADC_GetValue(&hadc1);
          raw = raw * 0.452
          value = (raw - 500.0)/10;
          value = (9*value)/5 + 32;
          sprintf(msg, "%f\r\n", value);
          HAL_UART_Transmit(&huart2, (uint8_t*) msg, 20, 100);
          HAL_Delay(1000);
}
/* USER CODE END 3 */
}
```

ADC Demonstration

- Todas las declaraciones dentro del fragmento de código se analizan a continuación:
  - HAL_ADC_Start(&hadc1); Inicia el periférico interno ADC1.
  - HAL_ADC_PollForConversion(&hadc1, 100); Espera una señal de fin de conversión (EOC) antes de pasar a la siguiente instrucción. El valor 100 es el tiempo máximo en milisegundos para esperar la señal EOC. Esta es una instrucción de tipo bloqueo.

```
while (1)
{
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
          HAL_ADC_Start(&hadc1);
          HAL_ADC_PollForConversion(&hadc1, 100);
          raw = (double) HAL_ADC_GetValue(&hadc1);
          raw = raw * 0.452
          value = (raw - 500.0)/10;
          value = (9*value)/5 + 32;
          sprintf(msg, "%f\r\n", value);
          HAL_UART_Transmit(&huart2, (uint8_t*) msg, 20, 100);
          HAL_Delay(1000);
}
/* USER CODE END 3 */
}
```

ADC Demonstration

- raw = raw * 0.452; — Conversión de un recuento bruto a una salida absoluta de mV. El factor de escala 0.452 se determinó mediante una medición de voltaje de precisión.
- value = (raw – 500.0)/10; El valor raw mV se convierte a una temperatura en °C utilizando la ecuación TMP36.
- value = (9* value)/5 + 32; Ecuación clásica para convertir de °C a °F.

```
while (1)
{
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
          HAL_ADC_Start(&hadc1);
          HAL_ADC_PollForConversion(&hadc1, 100);
          raw = (double) HAL_ADC_GetValue(&hadc1);
          raw = raw * 0.452
          value = (raw - 500.0)/10;
          value = (9*value)/5 + 32;
          sprintf(msg, "%f\r\n", value);
          HAL_UART_Transmit(&huart2, (uint8_t*) msg, 20, 100);
          HAL_Delay(1000);
}
/* USER CODE END 3 */
}
```

ADC Demonstration

- sprintf(msg, “%f\r\n”, value); Hace que el valor de la variable double value se convierta en una cadena de caracteres con un retorno de carro y un avance de línea concatenado al valor.
- HAL_UART_Transmit(&huart2, (uint8_t*) msg, 20, 100); Realiza la transmisión real de los datos desde la MCU al programa terminal
- HAL_Delay(1000); Inserta un retraso de un segundo entre muestras.

```
while (1)
{
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
          HAL_ADC_Start(&hadc1);
          HAL_ADC_PollForConversion(&hadc1, 100);
          raw = (double) HAL_ADC_GetValue(&hadc1);
          raw = raw * 0.452
          value = (raw - 500.0)/10;
          value = (9*value)/5 + 32;
          sprintf(msg, "%f\r\n", value);
          HAL_UART_Transmit(&huart2, (uint8_t*) msg, 20, 100);
          HAL_Delay(1000);
}
/* USER CODE END 3 */
}
```

- El programa main.c completo se enumera a continuación:

```
// STM disclaimer goes here
// Includes
#include "main.h"
#include "stm32f3xx_hal.h"
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */
// Private variables
ADC_HandleTypeDef hadc1;
UART_HandleTypeDef huart2;
/* USER CODE BEGIN PV */
// Private variables
double value = 1.0;
double raw =0.0;
char msg[20] = "";
/* USER CODE END PV */
// Private function prototypes
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_ADC1_Init(void);
/* USER CODE BEGIN PFP */
// Private function prototypes
/* USER CODE END PFP */
/* USER CODE BEGIN 0 */
/* USER CODE END 0 */
int main(void)
{
  /* USER CODE BEGIN 1 */
  /* USER CODE END 1 */
  // MCU Configuration
  // Reset all peripherals, initializes the flash and Systick.
  HAL_Init();
  /* USER CODE BEGIN Init */
  /* USER CODE END Init */
  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */
  /* USER CODE END SysInit */
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_ADC1_Init();
  /* USER CODE BEGIN 2 */
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */
          HAL_ADC_Start(&hadc1);
          HAL_ADC_PollForConversion(&hadc1, 100);
          raw = (double) HAL_ADC_GetValue(&hadc1);
          raw = raw 8 0.452;
          value = (raw - 500.0)/10;
          value = (9*value)/5 + 32;
          sprintf(msg, "%f\r\n", value);
          HAL_UART_Transmit(&huart2, (uint8_t*) msg, 20, 100);
          HAL_Delay(1000);
  }
  /* USER CODE END 3 */
}
// System Clock Configuration
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInit;
  // Initialize the CPU, AHB and APB buss clocks
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL16;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }
  // Initialize the CPU, AHB and APB buss clocks
  RCC_ClkInitStruct.ClockType =
  RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
  |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) !=
HAL_OK)
  {
```

```
    _Error_Handler(__FILE__, __LINE__);
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC1;
  PeriphClkInit.Adc1ClockSelection = RCC_ADC1PLLCLK_DIV1;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  // Configure the Systick interrupt time
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);
  // Configure the Systick
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}
/* ADC1 init function */
static void MX_ADC1_Init(void)
{
  ADC_ChannelConfTypeDef sConfig;
  // Common config
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = ENABLE;
  hadc1.Init.NbrOfDiscConversion = 1;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc1.Init.LowPowerAutoWait = DISABLE;
  hadc1.Init.Overrun = ADC_OVR_DATA_OVERWRITTEN;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }
  // Configure Regular Channel
  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = 1;
  sConfig.SingleDiff = ADC_SINGLE_ENDED;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  sConfig.OffsetNumber = ADC_OFFSET_NONE;
  sConfig.Offset = 0;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}
```

```
/* USART2 init function */
static void MX_USART2_UART_Init(void)
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 9600;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }
}
/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
}
/* USER CODE BEGIN 4 */
/* USER CODE END 4 */
/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void _Error_Handler(char * file, int line)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error re-
turn state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
#ifdef USE_FULL_ASSERT
/**
   * @brief Reports the name of the source file and the source line
number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
  /* USER CODE END 6 */
}
#endif
/**
  * @}
  */
/**
  * @}
*/
/**** (C) COPYRIGHT STMicroelectronics ****/
```

ADC Demonstration

- Test Run
  - Luego, el programa se creó y se descargó en la placa del proyecto. Luego inicié el programa del terminal Realterm para mostrar el flujo de comunicación del tablero. La Figura 9-10 muestra la salida del terminal.
  - Intenté enfriar y calentar el sensor y vi que el sensor reaccionaba de acuerdo con la temperatura ambiente.

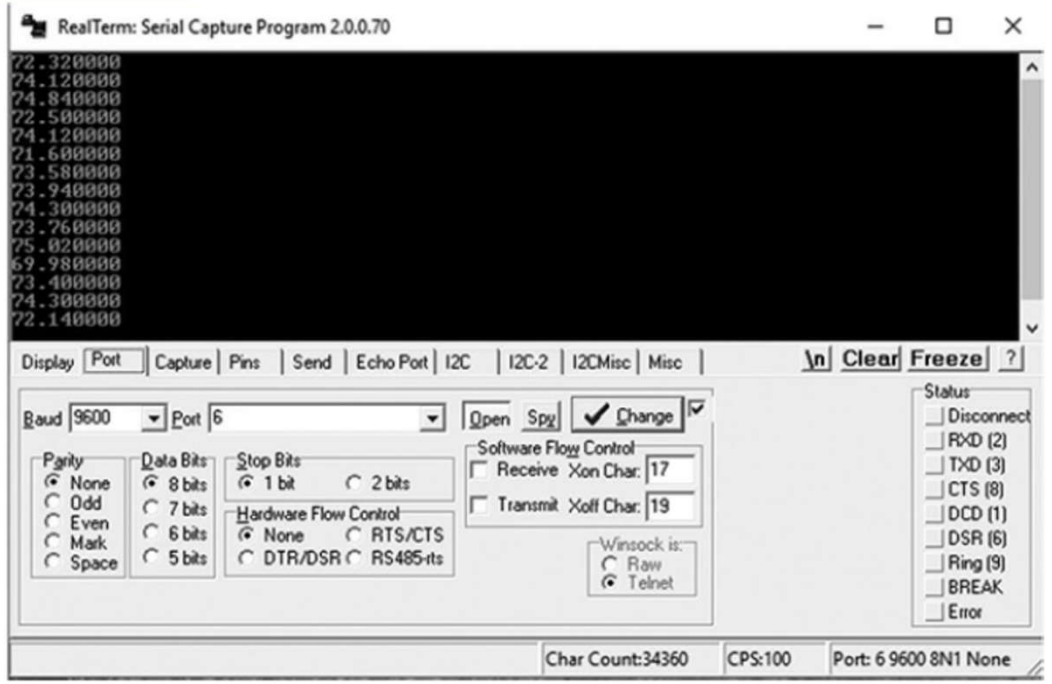


Figure 9-10 *Terminal output.*

Summary

- La conversión de analógico a digital (ADC) fue el tema del capítulo. Hubo una explicación de cómo funciona el ADC de 12 bits y 16 canales dentro de la MCU. Utiliza una técnica de aproximación sucesiva para adquirir muestras numéricas precisas de una entrada analógica que varía continuamente.
- A continuación, la discusión se centró en cómo el marco HAL respalda la función ADC. Revisé las dos estructuras C que configuran e inicializan el periférico ADC. La discusión también incluyó los distintos modos de conversión, canales, grupos y clasificaciones.
- Una demostración a gran escala del ADC cerró el capítulo. Se utilizó un sensor de temperatura TMP36 como fuente de voltaje analógico para el ADC. El programa utilizó el periférico USART2 para transmitir lecturas continuas de temperatura a una ventana de terminal.

Referencias

- Programming with STM32: Getting Started with the Nucleo Board and C/C++ 1st Edición - Donal Norris (Author)
- Nucleo Boards Programming with the STM32CubeIDE, Hands-on in more than 50 projects - Dogan Ibrahim (Author)
- STM32 Arm Programming for Embedded Systems, Using C Language with STM32 Nucleo - Muhammad Ali Mazidi (Author), Shujen Chen (Author), Eshragh Ghaemi (Author)

Manos a la obra con el . . .

. . . Proyecto Intermedio

. . . un enfoque centrado en la **práctica propia de la carrera** más que en el desarrollo teórico disciplinar, con eje en la **participación de las y los estudiantes**


Las y los estudiantes preguntarán:
¿en qué lío nos metimos?

¡Muchas gracias!

¿Preguntas?

. . .

Consultas a: jcruz@fi.uba.ar