Configuración de ADC en STM32CubeIDE

El ADC de 12-bits es un convertidor analogico-digital de aproximación sucesiva. Tiene hasta 19 canales multiplexados, permitiendo medir señales de hasta 16 fuentes externas, dos internas, y el canal V_{BAT}. La conversión A/D de los canales puede ser hecha en modo único, continuo, de escaneo o discontinuo. El resultado es almacenado en un registro de 16-bit.

Se crea un proyecto en STM32CubeIDE dando clic en “NewProject” y seleccionando la tarjeta STM32F401RE-Nucleo. Una vez creado el proyecto, del lado izquierdo de la vista de STM32CubeMX se puede apreciar la pestaña de Analog, donde dándole clic se desplegará la opción de ADC1, a la cual si se le da clic, mostrará los canales de entrada disponibles y los que estén ocupados o reservados, estarán en rojo.

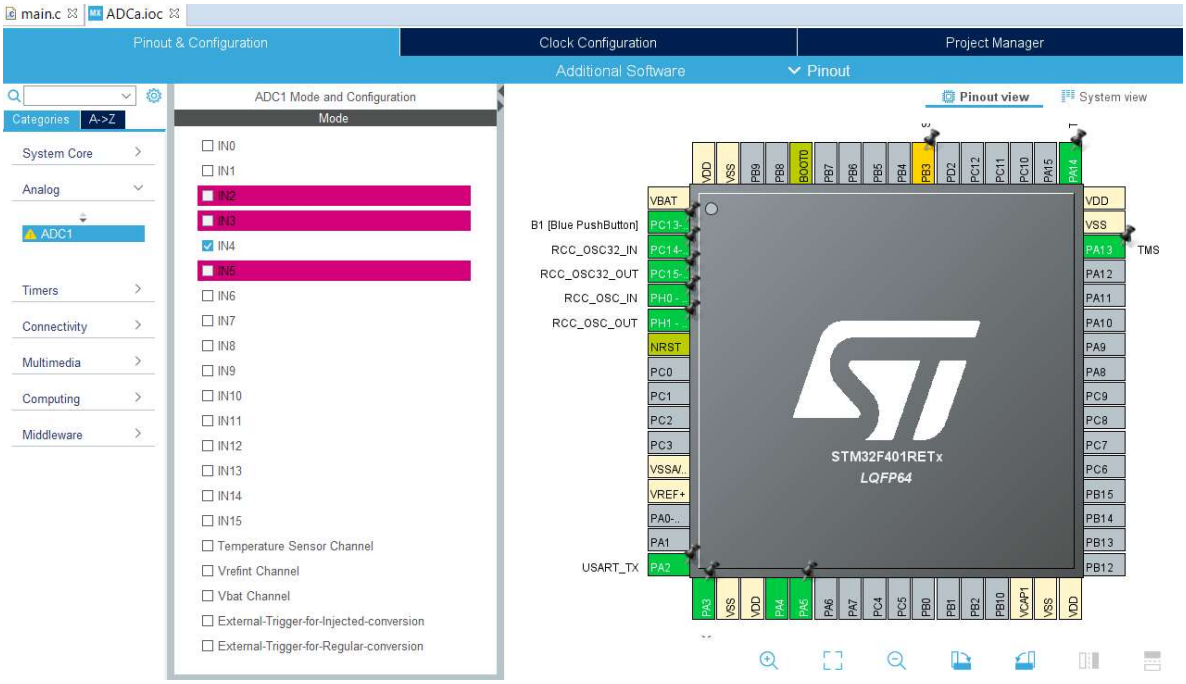

Figura 1: Vista de canales de ADC en STM32CubeMX

Ahora se procede a habilitar el ADC1, activando la casilla de IN4

Figura 2: IN4 habilitado para ADC

Ahora hay que desplegar el menú de configuración debajo de la selección de canales, y revisar que los valores estén idénticos a la foto, debajo serán explicados.

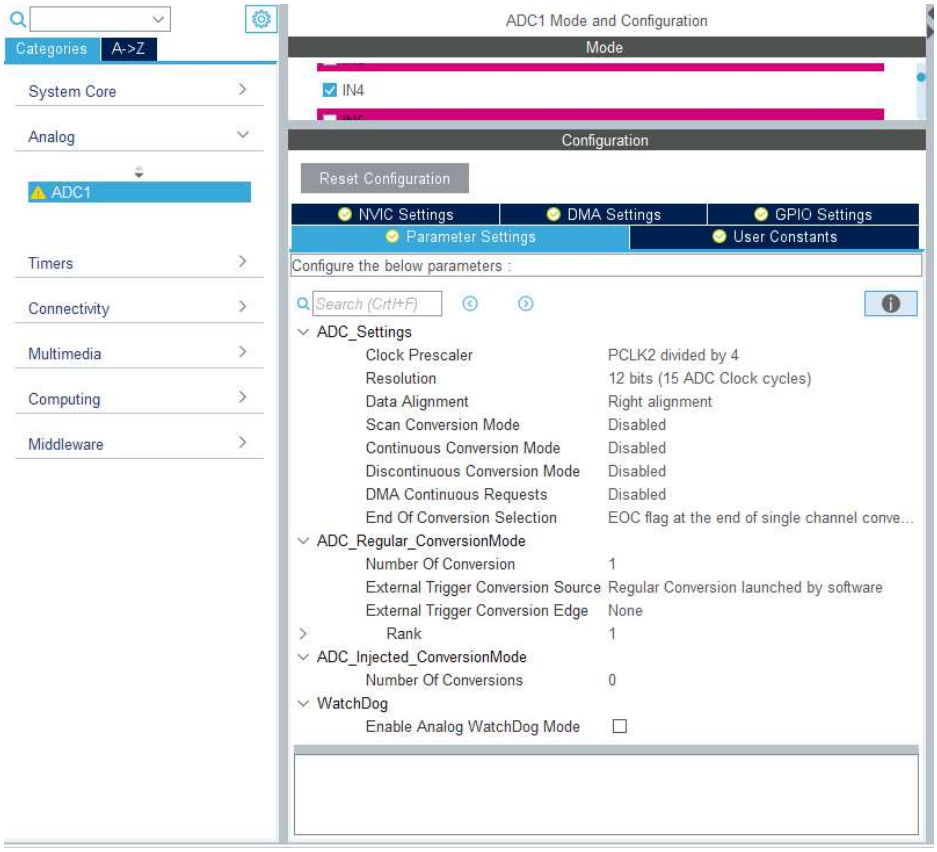

Figura 3: configuración ADC

- Resolution: Selecciona la resolución para la conversión del ADC.
- Data Alignment: Selecciona la alineación de los datos leídos por el ADC.
- Scan conversion mode: Conversión en un arreglo de canales uno detrás de otro
- Continuous Conversion Mode: Modo de conversión continuo.
- Discontinuous Conversion Mode: Modo discontinuo de conversión.
- DMA Continuos Requests: Solicitud para lectura continua del DMA.
- End of Conversion Selection: Fin de conversión en el canal
- Number of Conversion: Número de canales a convertir (debe de estar entre 1 y 16)
- External Trigger Conversion Source: Selecciona el evento externo para iniciar la conversión

- External Trigger Conversion Edge: Disparo que selecciona polaridad externa y habilita el inicio de un grupo regular (Este parámetro es cambiado automáticamente por el software)
- Rank: Rango para la secuencia de conversión (Este debe ser entre 1 y 16)
- Channel: Canal de conversión
- Sampling Time: Tiempo de muestreo
- Enable Analog WatchDog Mode: Habilita o deshabilita el modo de perro guardian

Después de cerciorarse que los parámetros estén iguales a la figura 6, hay que ir a Project manager -> Code Generator, y activar la casilla "*Generate peripherial initialization as a pairo f '.c/.h' files per peripherial"*, y posteriormente se genera el código.

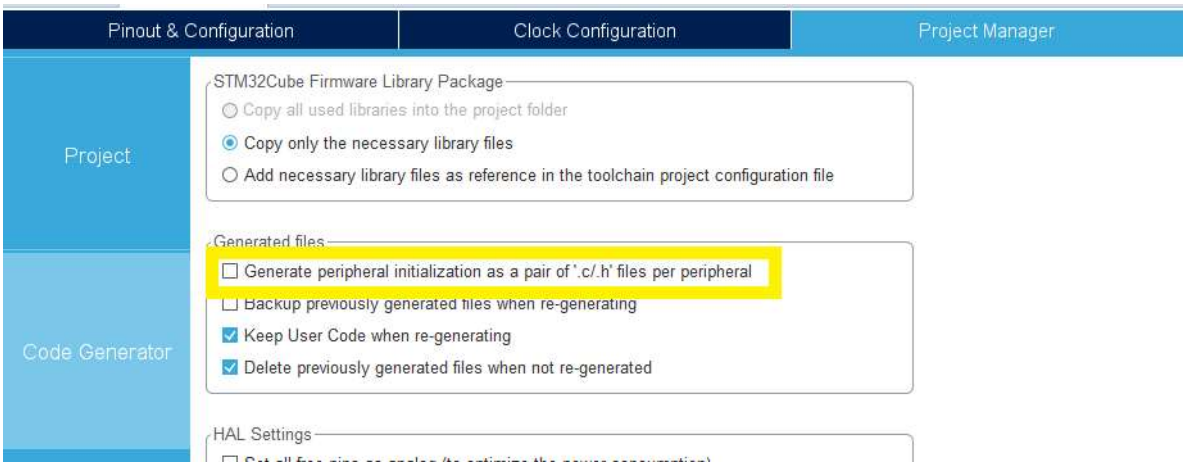

Figura 4: casilla para inicializaciones.

Después de pasar al avista C/C++, primero se agregan un par de etiquetas nuevas a las funciones GPIO_PIN_RESET y GPIO_PIN_SET como se muestra a continuación, para recordar mejor cual es el estado que enciende o apaga el led.

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#define Enc_LED GPIO_PIN_RESET
#define Apa_LED GPIO_PIN_SET
/* USER CODE END Includes */
```

Figura 5: etiquetas

Ahora hay que dirigirse a el *while* infinito, y escribir el siguiente código:

*main.c | ADCa.ioc | usart.c | startup_stm32f401retx.s

```
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
      HAL_ADC_Start(&hadc1);
          int Resultado = HAL_ADC_GetValue(&hadc1); //valor de conversion de bits

          if (Resultado<=1024){
              HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, Enc_LED);
          }
          else if (Resultado>1024 && Resultado<=2048){
            HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, Enc_LED);
              HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
              HAL_Delay(500);
              HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
              HAL_Delay(500);
              HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
          }
          else if (Resultado>2048 && Resultado<=3072){
            HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, Apa_LED);
            HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
            HAL_Delay(100);
            HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
            HAL_Delay(100);
            HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
           }
          else if (Resultado>3072){
            HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, Apa_LED);
              //HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
              //HAL_Delay(2000);
        //        HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
          }
  }
  /* USER CODE END 3 */
}
```

Figura 6: Código principal del programa

El código comienza con *HAL_ADC_Start(&hadc1)*, que inicia la conversión del ADC. Después con *AL_ADC_GetValue(&hadc1)* se obtiene el valor que entrega el ADC y se guarda en una variable llamada *Resultado*.

A continuación, el código compara el valor obtenido (que puede ser de 0 a 4095) contra un rango de valores prestablecidos, haciendo lo siguiente:

- Si el valor del ADC es menor a 1024, se encenderá el LED verde.
- Si el valor del ADC está entre 1025 y 2048, el LED parpadeará lentamente.
- Si el valor del ADC está entre 2049 y 3072, el LED parpadeará rápidamente.
- Si el valor del ADC es mayor a 3071, el LED se apagará.

El código correrá de forma infinita; es hora de preparar el hardware para poner a prueba el programa. Se requieren: 3 jumpers o cables macho-macho, una Protoboard, un potenciómetro de preferencia mayor a 1K, de preferencia 5K, y la tarjeta STM32 Nucleo. Se conectará un cable al 3.3v de la tarjeta, y a un extremo del potenciómetro; otro en GND de la tarjeta, y el extremo opuesto del potenciómetro; y finalmente, un cable del centro del potenciómetro a PA4, que es el pin que fue configurado como entrada del ADC.

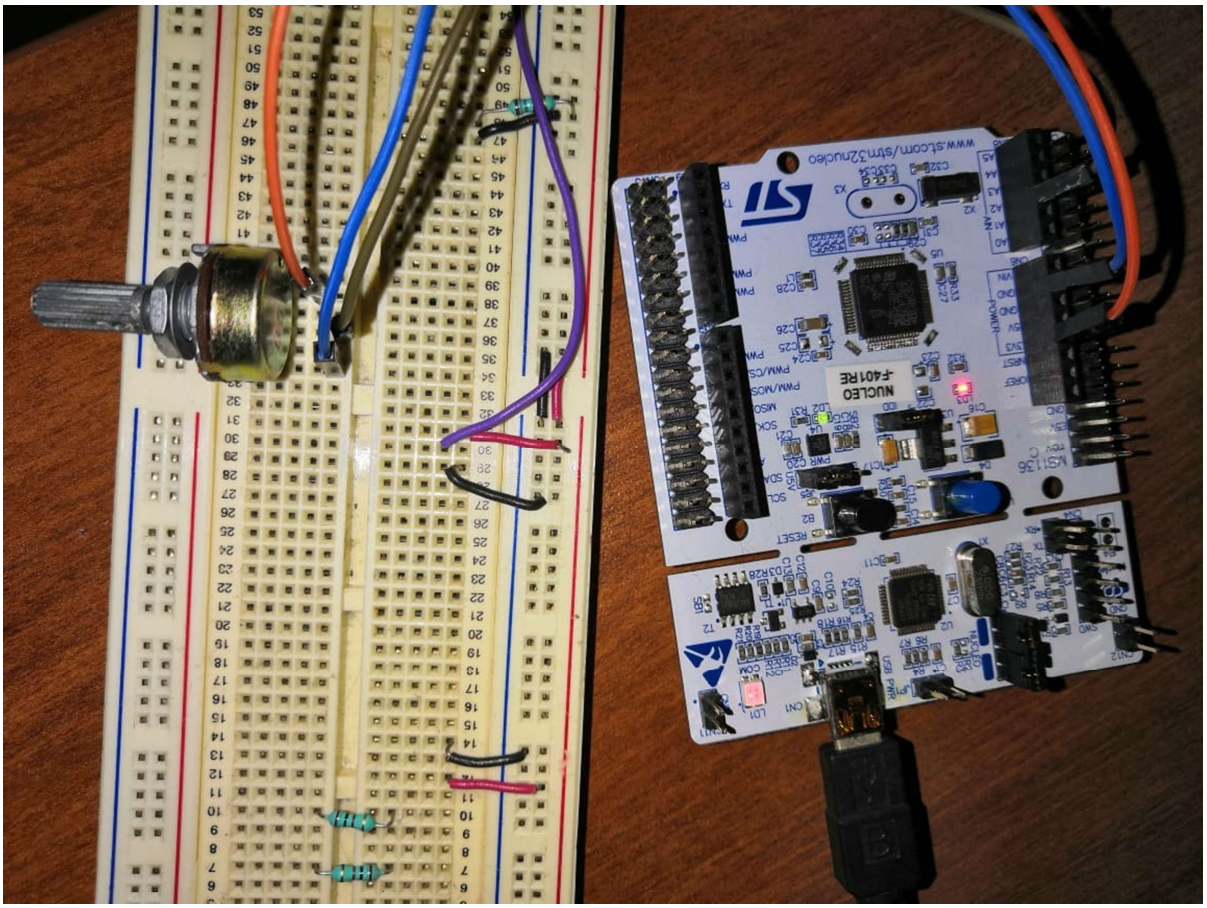

Figura 7: ejemplo de conexión al potenciómetro.

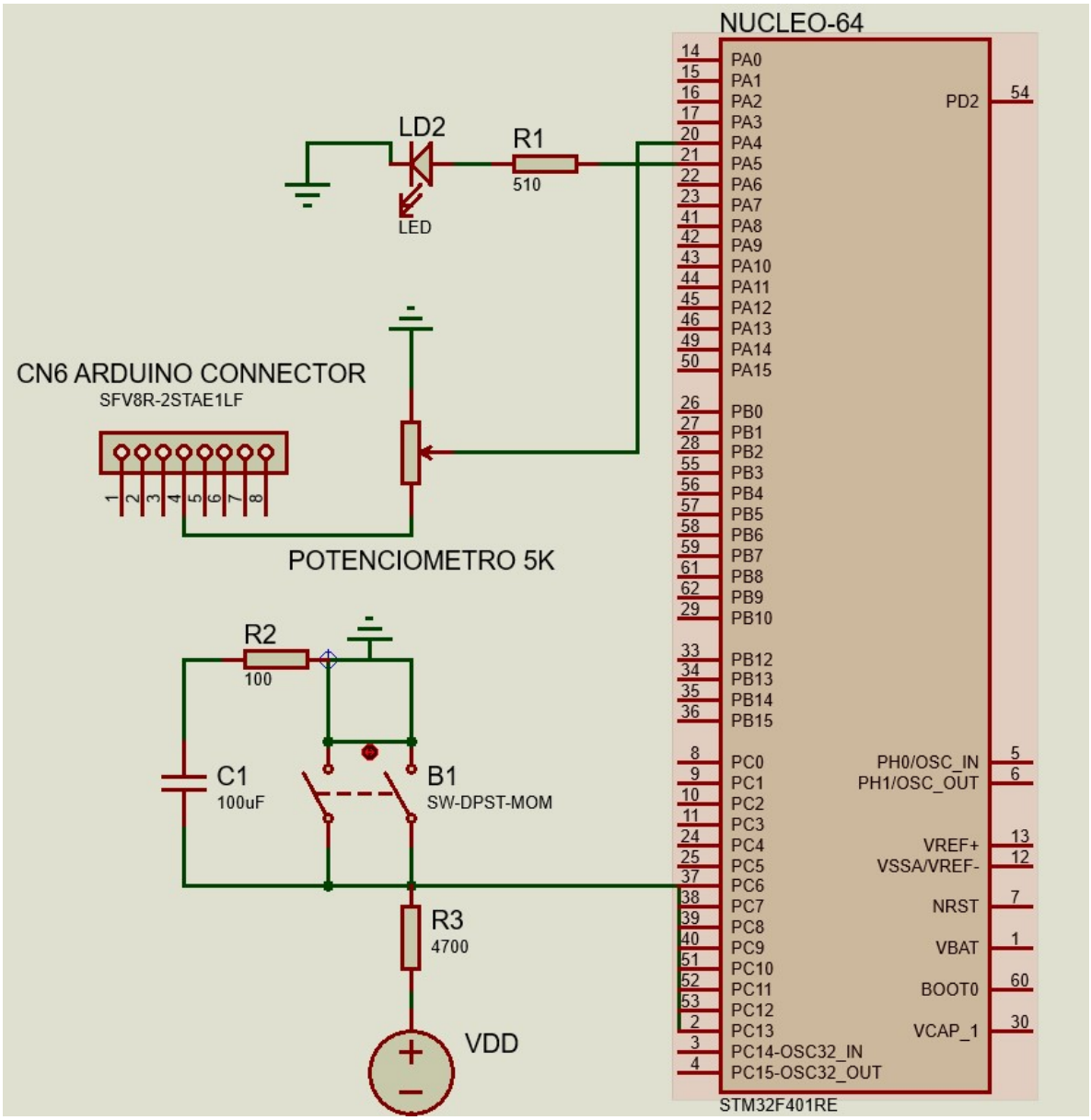

Figura 8: Esquemático de la conexión realizada.

Posteriormente cargamos el programa a la tarjeta y entramos al modo de depuración.