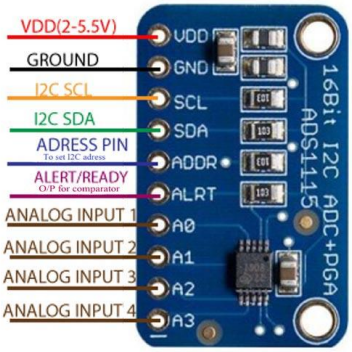
Utilizando el ADS1115

	ADC de 16 bits
Data Rate	**8sps a 860sps**
	Bajo consumo (150uA)
Iterfaz	**I2C (TwoWireInterface en AVR)**
Entrada	**4 canales single-ended o 2 canales diferenciales**
Rangos de Entrada (PGA)	**±256mV, ±512mV, ±1.024V, ±2.048V, ±4.096V, ±6.144V**
Alimentación	**2V a 5.5V**
	Referencia y Oscilador internos



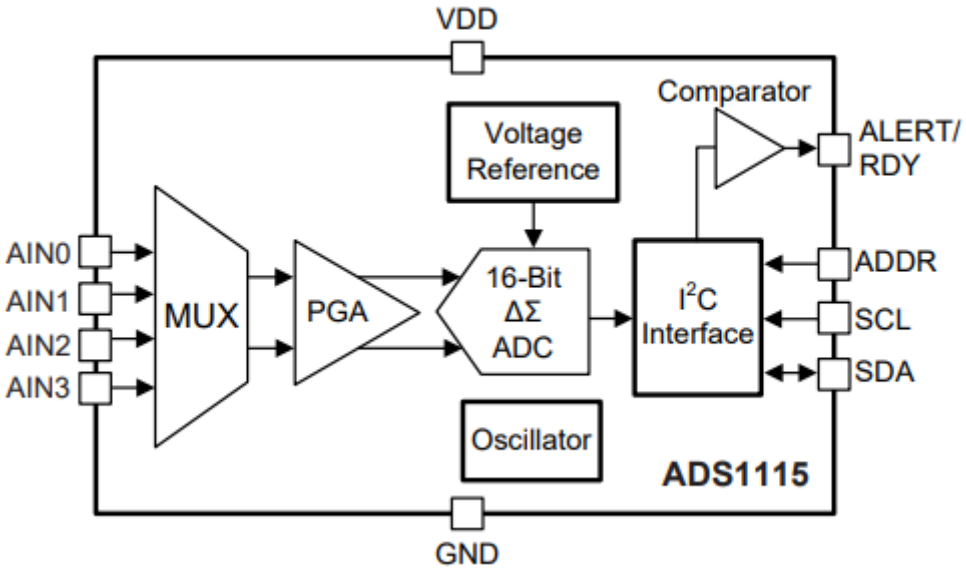


FIGURA 1 -DIAGRAMA DE BLOQUES DEL ADS1115-

Breve descripción del **I2C** y su funcionamiento en bajo nivel

La interfaz "Two Wire Interface" (TWI) es ideal para aplicaciones típicas de microcontroladores. El protocolo TWI permite diseñar sistemas para interconectar hasta 128 dispositivos diferentes usando solo dos líneas de bus bidireccionales, una para reloj (SCL) y otra para datos (SDA).

Esta interfaz utiliza un controlador principal, conocido como "maestro", para comunicarse con dispositivos "esclavos" conectados al bus.

El único hardware externo necesario para implementar el bus es una única resistencia *pull-up* para cada una de las líneas del bus TWI.

Todos los dispositivos conectados al bus tienen direcciones individuales de 7 bits, y los mecanismos para resolver la contención del bus son inherentes al protocolo TWI.

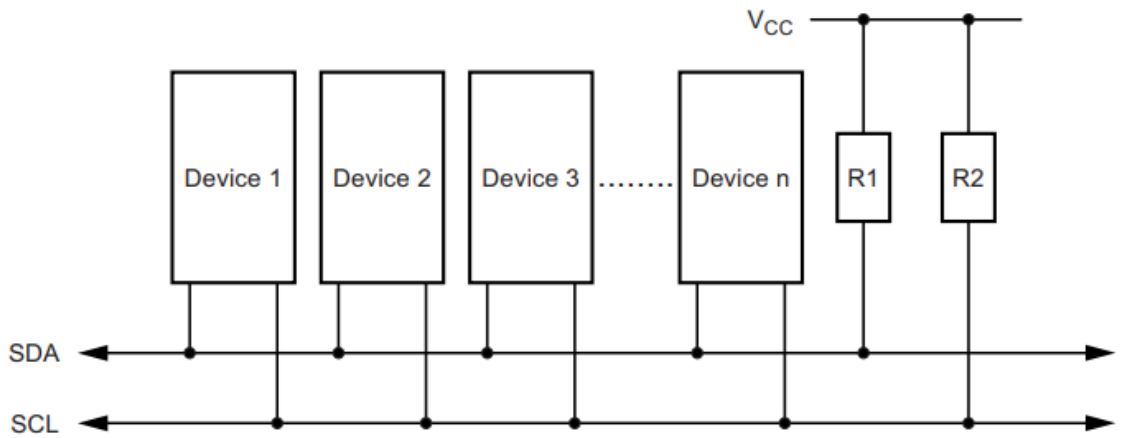


FIGURA 2 -BUS I2C-

La Figura 2 muestra el esquema de conexión general para la comunicación I2C, incluyendo la línea de datos (SDA) y la línea de reloj (SCL), ambas con resistores de *pull-up* hacia la alimentación. Los distintos dispositivos (maestros y esclavos) se conectan con sendas líneas.

El procedimiento general para que el maestro acceda al dispositivo esclavo es el siguiente:

1. Suponga que un maestro quiere enviar datos a un esclavo:

- El transmisor maestro envía una condición de START y se dirige al receptor esclavo enviando su ADDRESS correspondiente seguido de un bit de W (escritura).
- El transmisor maestro envía datos al receptor esclavo.
- El transmisor maestro finaliza la transferencia con una condición de STOP.

2. Si un maestro desea recibir / leer datos de un esclavo:

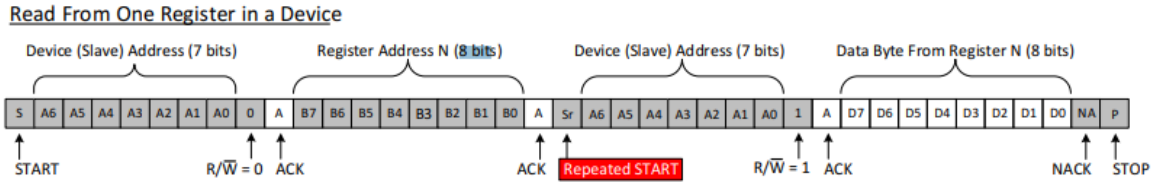
- El receptor maestro envía una condición de START y se dirige al transmisor esclavo enviando su ADDRESS correspondiente seguido de un bit de W (escritura).
- El receptor maestro envía la dirección del registro que se desea leer al transmisor esclavo.

• El receptor maestro envía nuevamente una condición de START y se dirige al transmisor esclavo enviando su ADDRESS correspondiente seguido de un bit de R (lectura).

• El receptor maestro recibe datos del transmisor esclavo.

• El receptor maestro finaliza la transferencia con una condición de STOP.

Una representación de lectura de registro de un esclavo se muestra a continuación:



Cada "paquete" de datos enviado o leído entre maestro-esclavo se conforma por 8 bits. Luego de recibido cada uno de estos "paquetes", tanto el maestro como el esclavo pueden responder con un bit de ACK (acknowledge o reconocimiento).

El bus SCL y los eventos de START y STOP son administrados por el dispositivo maestro y generalmente están implementados internamente en el microcontrolador utilizado como maestro.

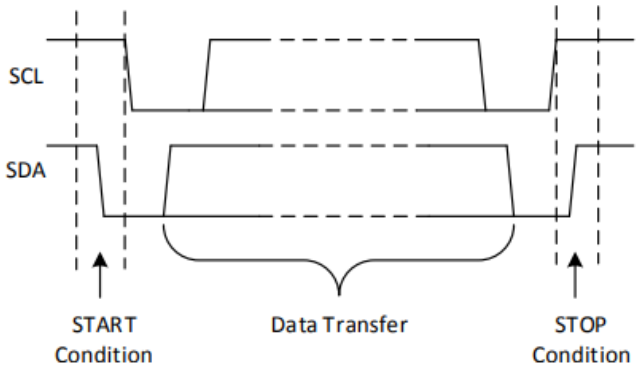


FIGURA 3 -EJEMPLO DE CONDICIÓN DE START Y STOP-

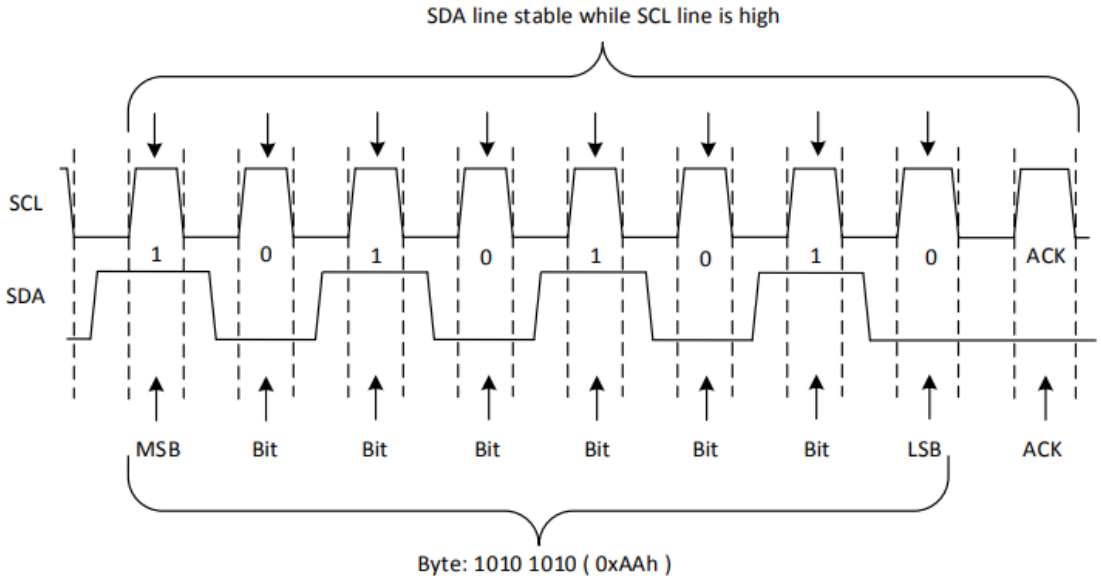


FIGURA 4 -EJEMPLO DE TRANSFERENCIA DE 1 BYTE-

Programación del ADS1115

- Selección del ADDRESS

El ADS1115 tiene un pin de dirección, ADDR, que configura la dirección I2C de 7 bits (ADDRESS) del dispositivo. Este pin puede ser conectado a GND, VDD, SDA o SCL, lo que permite seleccionar cuatro direcciones diferentes con un pin, como se muestra en la Tabla 1. El estado del pin de dirección ADDR es muestreado continuamente por el conversor.

ADDR PIN CONNECTION	SLAVE ADDRESS
GND	1001000
VDD	1001001
SDA	1001010
SCL	1001011

TABLA 1

El ADS1115 solo opera en modo esclavo y no puede manejar el bus de SCL.

- **Modo recepción**

En el modo de recepción esclavo, el primer byte transmitido del maestro (MCU) al esclavo consiste en la dirección del esclavo de 7 bits seguido de un bit $R/\overline{W}$ en bajo. El siguiente byte transmitido por el maestro se escribe en el registro del Puntero de Dirección de Registros. El ADS1115 luego acusa la recepción de este byte. Los siguientes dos bytes se escriben en la dirección dada por los bits de este Puntero de Dirección de Registros, P [1: 0]. El ADS1115 acusa recibo de cada byte enviado. Los bytes que escriben los registros son enviado con el byte más significativo primero, seguido del byte menos significativo (tener en cuenta que todos los registros son de 16 bits, a excepción del Puntero de Dirección de Registros, el cual se escribe con un solo byte).

- **Modo transmisión**

En el modo de transmisión esclavo, el primer byte transmitido por el maestro (MCU) es la dirección esclava de 7 bits seguido de un bit $R/\overline{W}$ en alto. Este byte coloca al esclavo en modo de transmisión e indica que se está leyendo el ADS1115. El siguiente byte transmitido por el esclavo es el byte más significativo del registro apuntado por el Puntero de Dirección de Registros, P [1: 0] (tener en cuenta que previamente se debe haber escrito por el maestro este

registro de direcciones para determinar el registro a leer luego). Este byte deberá estar seguido de un acuse de recibo (ACK) del maestro. El byte menos significativo restante se envía a continuación. Este byte podrá estar seguido de un acuse de recibo (ACK) del maestro, pero si éste desea terminar la transmisión o reiniciarla inmediatamente puede no reconocer recibo (NACK) y emitir una condición de STOP o RESTART.

- **Escribiendo y leyendo los registros**

Para acceder a un registro específico del ADS1115, el maestro primero debe escribir la dirección deseada en el Puntero de Dirección de Registros, P [1: 0]. Este registro de direcciones se escribe directamente después del acuse de recibo exitoso (ACK) por parte del esclavo para el byte que contiene el ADDRESS y el bit $R/\overline{W}$ en bajo. Luego de escribir el Puntero de Dirección de Registros, el esclavo reconoce y el maestro emite una condición STOP o RESTART.

Ahora, al querer leer del ADS1115, el valor anteriormente escrito en los bits P [1: 0] determina el registro que se lee. Para cambiar el registro que se desea leer, se debe escribir un nuevo valor en P [1: 0]. Para escribir un nuevo valor en P [1: 0], el maestro debe emitir un byte con la dirección del esclavo y el bit $R/\overline{W}$ en bajo, seguido de un byte con la dirección del registro al que se desea acceder. El maestro ahora puede emitir una condición de STOP o RESTART y enviar el byte que contiene el ADDRESS y el bit $R/\overline{W}$ en alto para comenzar la lectura. Si se desean lecturas repetidas del mismo registro, no es necesario enviar continuamente la dirección que se desea leer, porque el ADS111x almacena el valor de P [1: 0] hasta que es modificado por una operación de escritura. Sin embargo, para cada operación de escritura, el Puntero de Dirección de Registros debe escribirse con los valores adecuados.

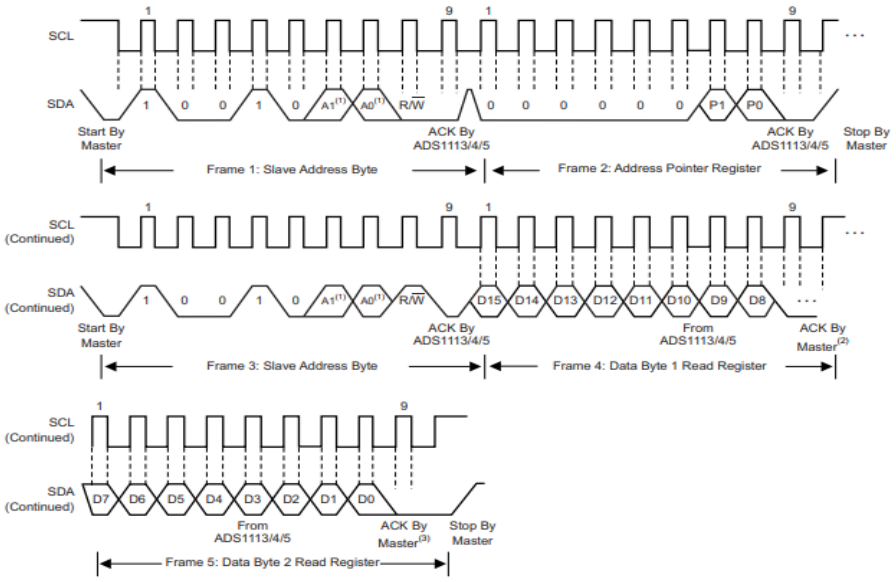


FIGURA 5 -SECUENCIA DE LECTURA DE UN REGISTRO-

Utilizando el ADS1115 con el ATmega328P

A continuación se describe una implementación básica realizada utilizando una placa Arduino Uno para controlar y tomar muestras del conversor ADS1115 y escribirlas por puerto serie.

I2C en el ATmega328P (TWI)

- Algunas características de la interfaz "Two Wire Interface" de este microcontrolador:
  - Compatible con la operación de maestro y esclavo.
  - El dispositivo puede funcionar como transmisor o receptor.
  - El espacio de direcciones de 7 bits permite hasta 128 direcciones esclavas diferentes.
  - Soporte de arbitraje multimaestro.
  - Velocidad de transferencia de datos de **hasta 400 kHz**.
  - Circuito de supresión de ruido para rechazo de picos en las líneas de bus.
  - Dirección de esclavo totalmente programable con soporte de llamada general.
  - El reconocimiento de direcciones hace que se despierte cuando el AVR® está en modo de suspensión.

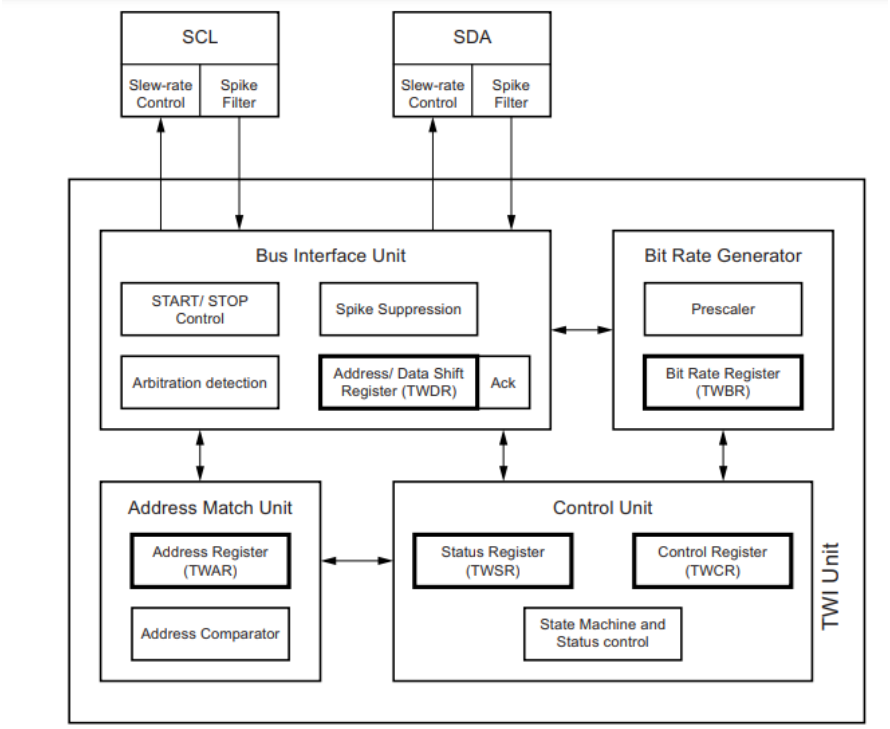


FIGURA 6 -VISTA GENERAL DEL MÓDULO TWI DEL ATMEGA328P-

- Se recomienda la lectura del Datasheet del MCU para la comprensión del funcionamiento de esta interfaz.

La comunicación se implementa a través de la escritura y lectura de diversos registros en el MCU, mientras que el Módulo TWI se encarga de administrar internamente la interfaz (manejo del SCL, condiciones de START y STOP, envío y recepción de la data, control de estado, etc.).

➢ Ejemplo de envío de datos desde el MCU a un esclavo

1. El primer paso en una transmisión TWI es transmitir una condición de START. Esto se hace escribiendo un valor específico en el registro TWCR, indicando al hardware TWI que transmita una condición de START. El valor a escribir se describe más adelante, sin embargo, es importante escribir un uno en TWINT (bit de flag) para borrar la bandera. El TWI no iniciará ninguna operación mientras el bit TWINT en TWCR esté seteado. Inmediatamente después de que la aplicación borre TWINT (escribiendo un uno en él), el TWI iniciará la transmisión de la condición de START.

2. Cuando se ha transmitido la condición de START, se setea el flag TWINT en TWCR y TWSR se actualiza con un código de estado que indica que la condición de INICIO se ha enviado con éxito.

3. El software de la aplicación ahora debería examinar el valor de TWSR, para asegurarse de que la condición de INICIO fue transmitido con éxito. Si TWSR indica lo contrario, el software de la aplicación podría realizar alguna acción especial, como llamar a una rutina de error. Suponiendo que el código de estado es el esperado, la aplicación debe cargar SLA + W (ADDRESS + bit de escritura) en el registro TWDR. Recordar que TWDR se usa tanto para direcciones como para datos. Después de que TWDR se haya cargado con el SLA + W, se debe escribir un valor específico en TWCR (TWINT en uno), indicando al hardware TWI que transmita el SLA + W presente en TWDR. Inmediatamente después de que la aplicación haya borrado TWINT (escribiendo un uno en él), el TWI iniciará la transmisión del paquete de direcciones.

4. Cuando se ha transmitido el paquete de direcciones, se establece la bandera TWINT en TWCR y TWSR se actualiza con un código de estado que indica que el paquete de direcciones se ha enviado correctamente. El código de estado también reflejará si un esclavo reconoció el paquete o no.

5. El software de la aplicación debería examinar ahora el valor de TWSR, para asegurarse de que el paquete de direcciones fue transmitido con éxito, y que el valor del bit ACK fue el esperado. Si TWSR indica lo contrario, el el software de la aplicación puede realizar alguna acción especial, como llamar a una rutina de error.

Suponiendo que el código de estado es como se esperaba, la aplicación debe cargar un paquete de datos en TWDR. Posteriormente, se debe escribir un valor específico en TWCR (TWINT en uno), instruyendo al hardware TWI para transmitir el paquete de datos presente en TWDR.

El TWI no iniciará ninguna operación mientras el bit TWINT en TWCR esté seteado. Inmediatamente después de que la aplicación borre TWINT (escribiendo un uno en él), el TWI iniciará la transmisión del paquete de datos.

6. Cuando se ha transmitido el paquete de datos, se establece la bandera TWINT en TWCR y TWSR se actualiza con un estado código que indica que el paquete de datos se ha enviado correctamente. El código de estado también reflejará si un esclavo reconoció el paquete o no.

7. El software de la aplicación debería examinar ahora el valor de TWSR, para asegurarse de que el paquete de datos fue transmitido con éxito, y que el valor del bit ACK fue el esperado. Si TWSR indica lo contrario, el el software de la aplicación puede realizar alguna acción especial, como llamar a una rutina de error. Suponiendo que el código de estado es el esperado, la aplicación debe escribir un valor específico en TWCR (TWINT en uno), indicando al hardware TWI que transmita una condición de STOP. El TWI no iniciará ninguna operación mientras el bit TWINT en TWCR está seteado. Inmediatamente después de que la aplicación haya borrado TWINT, el TWI iniciará la transmisión de la condición de STOP. Tener en cuenta que TWINT NO se establece después de que se ha enviado una condición de STOP.

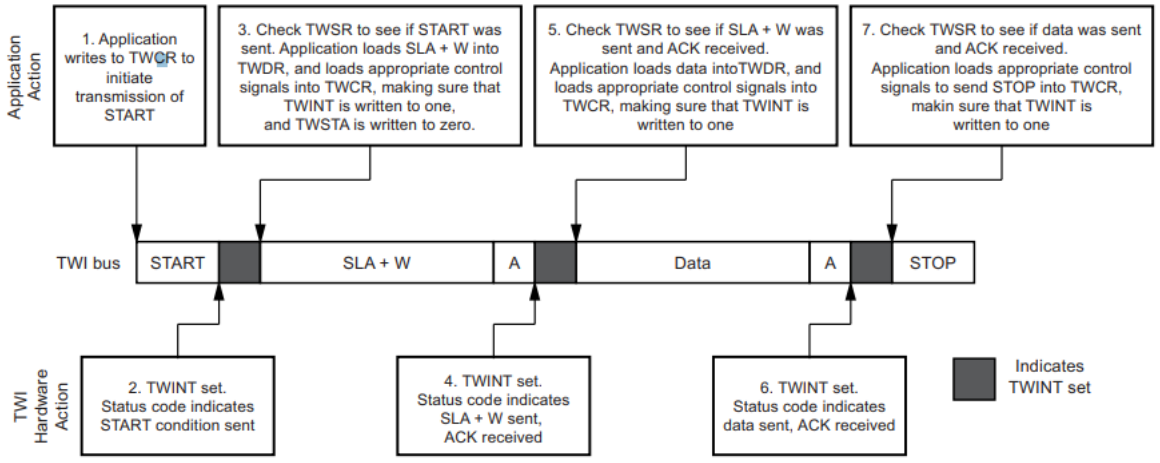


FIGURA 7 -EJEMPLO ESCRITURA DE 1 BYTE-

➢ Algunos Registros relevantes para el modo maestro del MCU.

- ❖ TWBR (Bit Rate Register)
- ❖ TWSR (Status Register)

Estos dos registros se utilizan principalmente para definir la frecuencia del SCL y, por defecto, la velocidad de la comunicación. Los últimos 3 bits de TWSR determinan el PrescalerValue, y los primeros determinan los diferentes códigos de estado posibles, lo cuales son importantes para la depuración de errores.

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \times (\text{PrescalerValue})}$$

FIGURA 8 -CÁLCULO PARA FRECUENCIA DE SCL-

❖ TWCR (Control Register)

Este registro es el principal controlador de la interfaz.

Bit	7	6	5	4	3	2	1	0	
(0xBC)	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

FIGURA 9 -REGISTRO TWCR-

❖ TWDR (Data Register)

En modo transmisión este registro contiene el byte próximo a ser enviado.

En modo recepción contiene el último byte recibido.

▪ Entorno y Programación

Se utiliza el Atmel Studio 7.0 para la codificación, programación del MCU y comunicación serie con el Arduino UNO. Además se hace uso de extensiones del Atmel Studio tales como el AVRdude (flasheo del ATmega328P) y TerminalWindow para la recepción de datos en la PC por puerto serie, a través de USB.

▪ Implementación

El objetivo principal es la puesta en funcionamiento de la interfaz I2C entre el MCU y el ADS1115, realizando un seteo inicial del conversor y una posterior captura de muestras para ser visualizadas en la PC.

➢ En primer lugar se definen algunos parámetros a configurar:

❖ El ADDRES queda determinado conectando el pin ADDR y GND del ADS.

```
I2C_ADDRESS 0b1001000
```

❖ Se utiliza la velocidad máxima disponible en la interfaz por el MCU (ver Figura 8)

```
TWBR = 0x0B;          //SCL = FMCU/40 = 400kHz
TWSR = 0x00;
```

- Los valores del registro de configuración se establecen en los siguientes valores:

```
CONFIG_REG_MSB.bits.OS            = 0b0;        //default
CONFIG_REG_MSB.bits.MUX           = 0b111;      //Input entre AIN3 y GND
CONFIG_REG_MSB.bits.PGA           = 0b0;        //Rango de entrada +-6.144V
CONFIG_REG_MSB.bits.MODE          = 0b0;        //Conversión continua
CONFIG_REG_LSB.bits.DATA_RATE     = 0b100;      //Datarate 128sps (default)
CONFIG_REG_LSB.bits.COMP_MODE     = 0b0;        //default
CONFIG_REG_LSB.bits.COMP_POL      = 0b0;        //default
CONFIG_REG_LSB.bits.COMP_LAT      = 0b0;        //default
CONFIG_REG_LSB.bits.COMP_QUE      = 0b11;       //default
```

- La tensión de alimentación para el conversor es de 5V y se toma desde la Placa Arduino. **[Tener en cuenta que el rango de alimentación del conversor es de 2V a 5.5V]**
- La señal a muestrear se toma desde un potenciómetro de 10k, estableciendo un rango entre GND y 5V, tomados también desde la Placa Arduino. **[Tener en cuenta que los rangos de valores de tensión de señal a la entrada van desde ±256mV hasta ±6.144V (este último con 5V de alimentación]**

- Se arman algunas funciones para cada comando o evento de la interfaz con el fin de simplificar y hacer más legible el código correspondiente a la comunicación I2C:

```
void I2C_START()
       {TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);
       while (!(TWCR & (1<<TWINT))); }

void I2C_SEND(uint8_t dato)
       {TWDR = dato;
       TWCR = (1<<TWINT) |   (1<<TWEN);
       while (!(TWCR & (1<<TWINT))); }

void I2C_SEND_ADDR(uint8_t ADDR, uint8_t RW)
       {TWDR = (ADDR << 1) | RW;
       TWCR = (1<<TWINT) |   (1<<TWEN);
       while (!(TWCR & (1<<TWINT))); }

void I2C_READ_ACK(uint8_t *dato)                    //lectura con acknowledge
       {TWCR |= (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
       while (!(TWCR & (1<<TWINT)));
       *dato = TWDR; }

void I2C_READ_NACK(uint8_t *dato)                   //lectura sin acknowledge
       {TWCR |= (1<<TWINT) | (1<<TWEN);
       while (!(TWCR & (1<<TWINT)));
       *dato = TWDR; }

void I2C_STOP()
       {TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);
       while(TWCR & (1<<TWSTO)); }
```

- Más arriba se muestra el seteo inicial del registro de configuración del conversor. Este formato intenta emular la apariencia del registro real en el conversor y también poder acceder a él de a un byte por vez (el mismo es de 16 bits). A continuación se muestra una forma de lograr esto a través de uniones, estructuras y campos de bits:

```
union {
	uint8_t byte;
	struct {
		unsigned int COMP_QUE	: 2; //L
		unsigned int COMP_LAT	: 1;
		unsigned int COMP_POL	: 1;
		unsigned int COMP_MODE	: 1;
		unsigned int DATA_RATE	: 3; //H
	}bits;
}CONFIG_REG_LSB; //byte menos significativo del registro

union {
	uint8_t byte;
	struct {
		unsigned int MODE		: 1; //L
		unsigned int PGA		: 3;
		unsigned int MUX		: 3;
		unsigned int OS			: 1; //H
	}bits;
}CONFIG_REG_MSB; //byte más significativo del registro
```

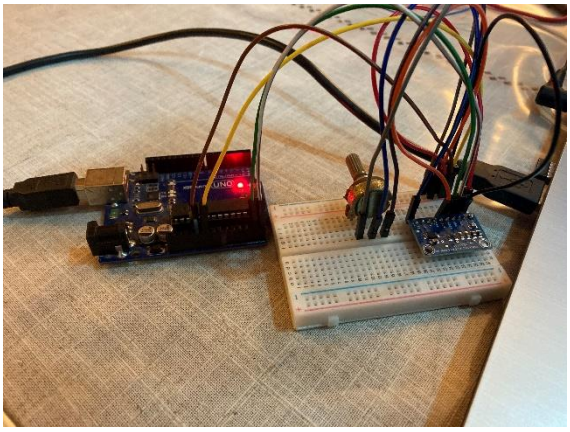
- Luego de setear tanto la configuración del ADS1115 como del MCU (clock, baudrate, UART, TWI, etc.) se implementa una pequeña rutina para leer un total de 100 muestras del conversor, una cada 500ms.

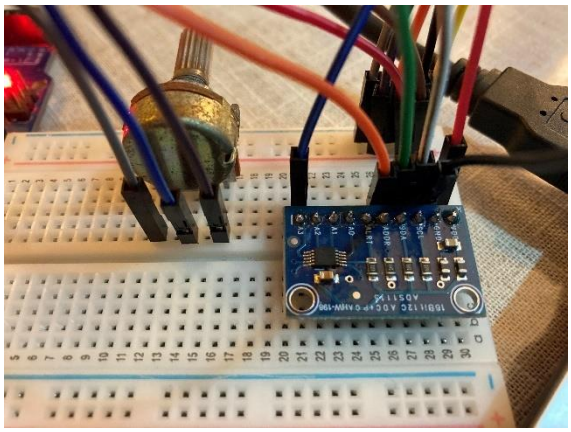
```
int main(void)
{
……/*aquí setear configuración del ADS1115, registros del MCU, etc.*/
……
I2C_START();
I2C_SEND_ADDR(I2C_ADDRESS, I2C_W); //envía ADDRESS y bit de escritura
I2C_SEND(CONFIG_REG_POINT);        //apunta al registro Config
I2C_SEND(CONFIG_REG_MSB.byte);     //escribe el MSB del registro Config
I2C_SEND(CONFIG_REG_LSB.byte);     //escribe el LSB del registro Config
I2C_RESTART();
I2C_SEND_ADDR(I2C_ADDRESS, I2C_W); //envía ADDRESS y bit de escritura
I2C_SEND(CONVER_REG_POINT);        //apunta al registro Conversion
```

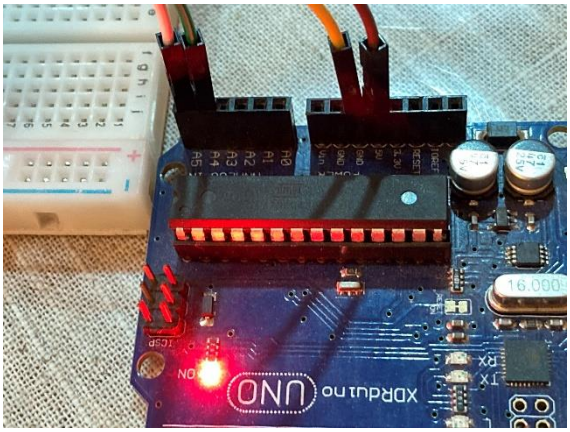
```
        while (i<100)
    {
                I2C_START();
                I2C_SEND_ADDR(I2C_ADDRESS, I2C_R); //envía ADDRESS y bit de lectura
                I2C_READ_ACK(&CONV_MSB);           //leer primer byte
                I2C_READ_ACK(&CONV_LSB);           //leer segundo byte
                I2C_STOP();
                _delay_ms(500);
                        escribir_byte_UART(CONV_MSB); escribir_byte_UART(CONV_LSB);
        //escribe por UART los 16bits
                i++;
    }
}
```

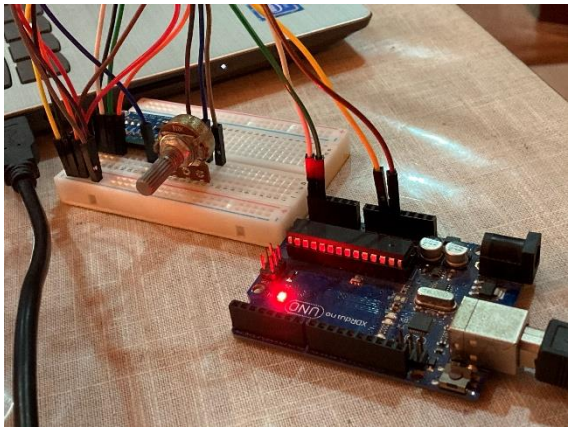
Validación de Resultados

- A continuación se muestran algunas capturas de la configuración y conexiones entre Placa y ADS1115.









- Una vez programado el MCU, se inicia la terminal y se conecta con la baudrate establecida.

  Se reinicia la placa mediante el botón de reset y se visualizan las muestras por terminal.

  Durante la lectura iterativa de muestras (duración aproximada de 50 segundos), con el potenciómetro se barre lentamente desde los 5V hasta GND.

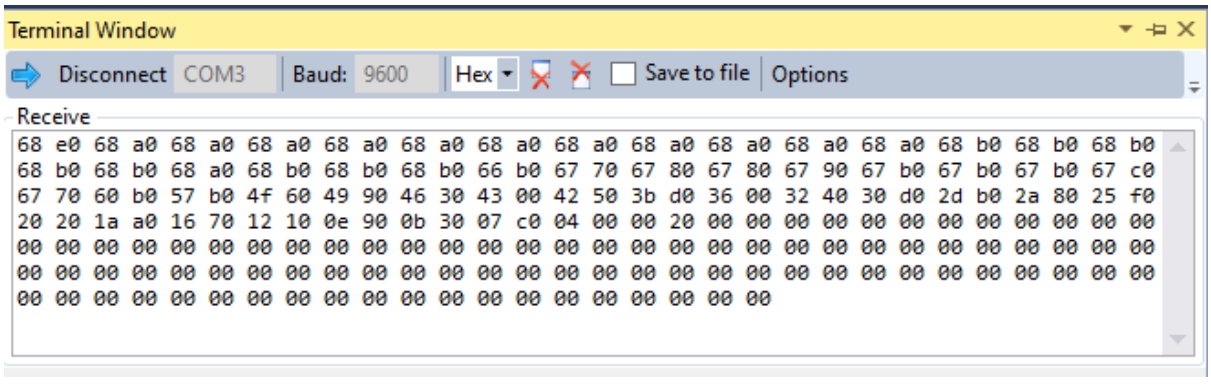


FIGURA 10- VISUALIZACIÓN POR TERMINAL DE LAS LECTURAS CON BARRIDO ENTRE 5V Y GND-

En la Figura 10**¡Error! No se encuentra el origen de la referencia.** se pueden ver las lecturas que devuelve el MCU partiendo de mantener el potenciómetro al máximo (5V), y barriendo hacia GND.

Para corroborar los resultados se utiliza la siguiente tabla obtenida de la hoja de datos:

INPUT SIGNAL $V_{IN} = (V_{AINP} - V_{AINN})$	IDEAL OUTPUT CODE(1)(1)
≥ +FS $(2^{15} - 1)/2^{15}$	7FFFh
$+FS/2^{15}$	0001h
0	0000h
$-FS/2^{15}$	FFFFh
≤ –FS	8000h

TABLA 2

Aquí se puede ver que una salida de 0x01 equivale a $FS/2^{15}$, donde FS es el "fondo de escala". Para este caso $FS = 6.144V$

- Entonces, para el primer valor visualizado por terminal: $(0x68E0)$

$$V_1 = 0x68E0 * \frac{FS}{2^{15}} = 26848 * \frac{6.144V}{2^{15}} = 5.034V$$

- Para un valor intermedio leído: $(0x4250)$

$$V_{int} = 0x4250 * \frac{FS}{2^{15}} = 16976 * \frac{6.144V}{2^{15}} = 3.183V$$

Y para los últimos valores, corresponden a GND, con un valor de cero $0x0000$

$$V_{GND} = 0$$

Estos valores corresponden con los resultados esperados.

Un detalle observado es el valor constante presente en el nibble menos significativo, el cual se mantiene siempre en cero. A primera vista no se identifica la causa de esta anomalía, más adelante se deberá corroborar y justificar.

Queda pendiente una implementación muestreando a la máxima velocidad disponible del conversor (860sps) y con una señal de prueba variable y alterna.

Referencias

1. ADS1115 Data Sheet [https://www.ti.com/lit/ds/symlink/ads1113.pdf?ts=1627432843792&ref_url=https%253A%252F%252Fwww.google.com%252F]
2. ATmega328P Data Sheet [https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf]
3. "Understanding the I2C bus" -Texas Instruments- [https://www.ti.com/lit/an/slva704/slva704.pdf?ts=1627479529400&ref_url=https%253A%252F%252Fwww.google.com%252F]
4. Código disponible en los repositorios del laboratorio GIBIC. [https://bitbucket.org/gibic/ads1115_atmega328p/src/master/]