TITULO *TITLE*	**Vatímetro Digital Registrador**				
REF.	**UTN-FRM-TDII-Plantilla Trabajo Final-2022**				
EDICIÓN *ISSUE*					
FECHA *DATE*					

	NOMBRE Y CARGO *NAME &POSISTION*	**FIRMA** *SIGNATURE*	**FECHA** *DATE*
ESCRITO POR *WRITTEN BY*	Guerrero Valentin: guerrerovalentin6@gmail.com Magni Exequiel: exequiel.juan.magni@gmail.com Villarroel Francisco: villarroelf233@gmail.com Viñolo Franco: juanvinolo@outlook.com		
VERIFICADO POR *CHECKED BY*	**Dr. Cristian Pérez - JTP - Catedra Técnicas Digitales II**		
G. de CALIDAD *PRODUCT ASSURANCE*	**Ing Jorge Abraham - Prof Adjunto - Catedra Técnicas Digitales II**		
AUTORIZADO POR *AUTHORISED BY*	**Ing Gustavo Mercado - Prof Titular - Cátedra Técnicas Digitales II**		

ARGENTINA

REGISTRO DE REVISIONES

Revisión Preliminar de Diseño PDR (Preliminary Design Review)					
Fecha	18/10/22				
Autores	Guerrero	Magni	Villarroel	Viñolo	
Documentación Presentada	Informe de anteproyecto presentado.				
Revisión					
Profesores	Nombre		Nombre		
	Firma		Firma		

REGISTRO DE REVISIONES

Revisión Crítica de Diseño CDR (Critical Design Review)					
Fecha	17/11/22				
Autores	Guerrero	Magni	Villarroel	Viñolo	
Documentación Presentada	Presentación de avances del proyecto hasta el momento				
Revisión					
Profesores	Nombre		Nombre		
	Firma		Firma		

REGISTRO DE REVISIONES

Revisión Final de Proyecto FPR (Final Project Review)					
Fecha	23/11/22				
Autores	Guerrero	Magni	Villarroel	Viñolo	
Documentación Presentada	Presentación de prototipo				
Revisión					
Profesores	Nombre		Nombre		
	Firma		Firma		

Entrega de Proyecto PD (Project Delivery)					
Fecha	31/11/22				
Autores	Guerrero	Magni	Villarroel	Viñolo	
Documentación Presentada	Presentación de Documento de Proyecto (este documento versión final)				
Profesores	Nombre		Nombre		
	Firma		Firma		

REGISTRO DEL CAMBIO DEL DOCUMENTO

DOCUMENT CHANGE RECORD

EDIC. *ISSUE*	FECHA *DATE*	Nota de Cambio *Change Notice*	DESCRIPCION DEL CAMBIO *CHANGE DESCRIPTION*

INDICE

1. INTRODUCCIÓN

1.1. OBJETO:

El objeto del Proyecto es el diseño y fabricación de un instrumento que sea capaz de medir potencia aparente, potencia activa, potencia reactiva y factor de potencia de una línea monofásica, con la idea de, efectuar mediciones con la mayor exactitud y precisión posible, minimizar los errores presentes en la medición, alcanzar valores de resolución y sensibilidad óptimos para el campo propuesto (medición de líneas monofásicas) y registrar mediciones para el uso posterior de los datos.

1.2. ABREVIATURAS:

UTN	Universidad Tecnológica Nacional
FRM	Facultad Regional Mendoza

2. DOCUMENTOS APLICABLES

2.1. DOCUMENTOS DE REFERENCIA:

Utilizamos la hoja de datos de cada componente utilizado, como también del microcontrolador “Blue Pill”.

3. DESARROLLO (Vatímetro digital registrador)

- Medidor completo de potencias y calidad de línea.
- Realizamos un instrumento digital que permita realizar mediciones de potencia en una línea monofásica y permita almacenar datos de la medición.
- Funcionamiento: El instrumento consiste en dos circuitos diseñados para medir tensión de la línea monofásica y corriente, este ultimo utiliza un sensor de efecto hall el cual el usuario deberá colocar alrededor del conductor en el cual se encuentra la carga cuyo consumo de potencia se va a medir, el usuario no deberá conectar el circuito de tensión a la línea debido a que el propio instrumento mide por su cuenta esta tensión a través de su propia alimentación.

  Luego de encender el instrumento, este dará, en un display, las mediciones de tensión, corriente, potencia aparente, potencia activa, potencia reactiva y factor de potencia, debiendo acceder a algunas de estas mediciones a través de un menú, manejado por pulsadores.

3.1. High level design:

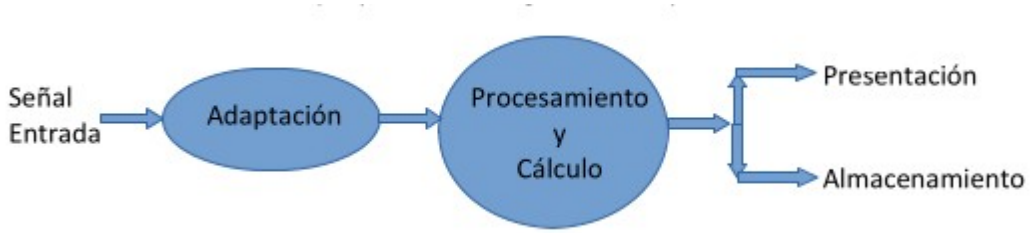
La medición de potencia de una línea monofásica es una medición indirecta. Se debe medir directamente las señales de tensión y corriente, las cuales se adaptarán con el uso de amplificadores y atenuadores para su procesamiento. El parámetro de interés es la amplitud de las señales para el cálculo de potencia aparente, potencia activa, potencia reactiva y factor de potencia.

$$Vrms = \sqrt{\frac{X_1^2 + X_2^2 + ... + X_n^2}{n}} \qquad Irms = \sqrt{\frac{X_1^2 + X_2^2 + ... + X_n^2}{n}}$$

$$S = U \cdot I \qquad \cos\varphi = \frac{P_a}{S} \qquad P = \frac{1}{T}\int_0^T v(t).i(t).dt$$

$$Q = \sqrt{S^2 - P^2}$$

Se dividió el circuito del proyecto en las siguientes etapas:



La señal de entrada (sea corriente o tensión), ingresa a una etapa encargada de adaptar los valores de amplitud ya sea atenuando o amplificando para la utilización de conversores análogo-digital (ADC). Como los conversores tienen referencia en +Vref = +Vdd, y -Vref = -Vss = GND, todos los valores que ingresen al conversor deberán comprender valores entre 0 V y 3,3V.

El procesamiento de las señales será realizado con el Módulo de Desarrollo Stm32f103c8t6 Blue Pill Stm32. A partir del valor pico ingresado en los conversores.

La presentación de la información se hará en una pantalla LCD presentando valores de tensión rms, corriente rms, potencia activa, potencia reactiva, potencia aparente y factor de potencia.

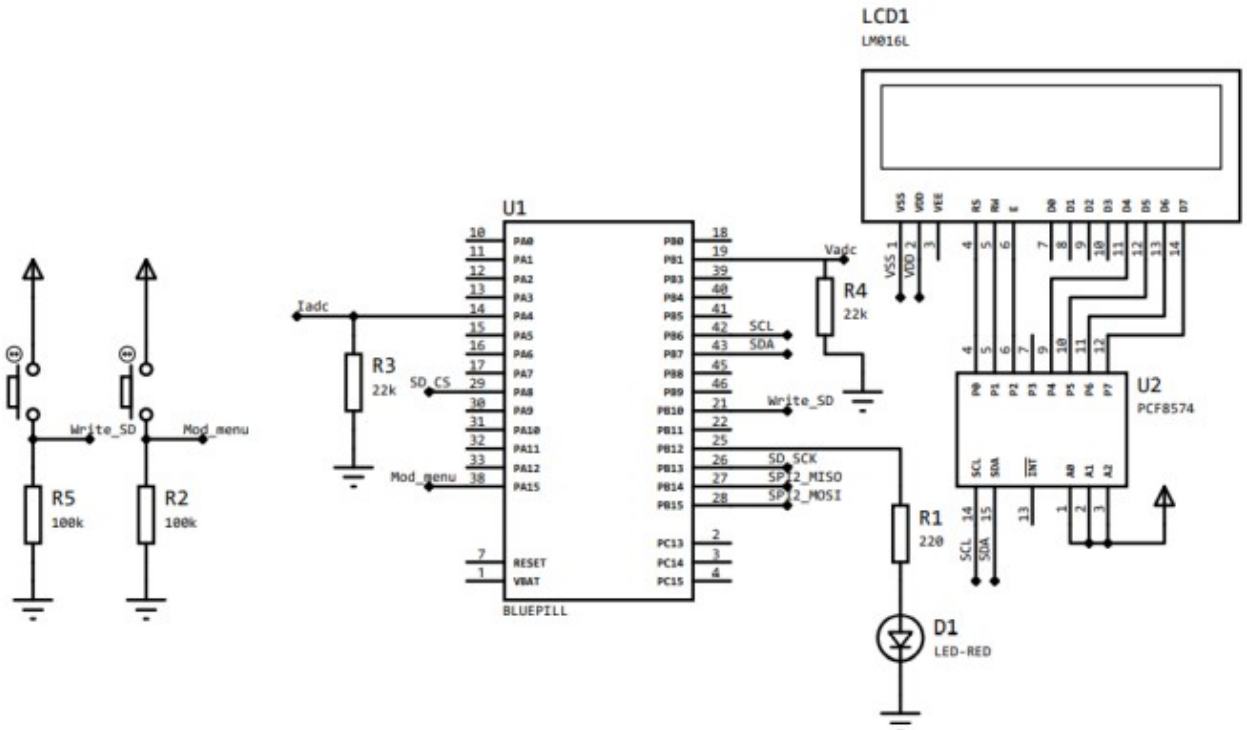
Se dispondrá la posibilidad de guardar las mediciones realizadas en una memoria micro SD CARD junto con la fecha de la medición proporcionada por el RTC interno en la Blue Pill.

3.2. Program/hardware design:

Diseño de Circuitos (Hardware)

Por la parte digital, los elementos utilizados fueron:

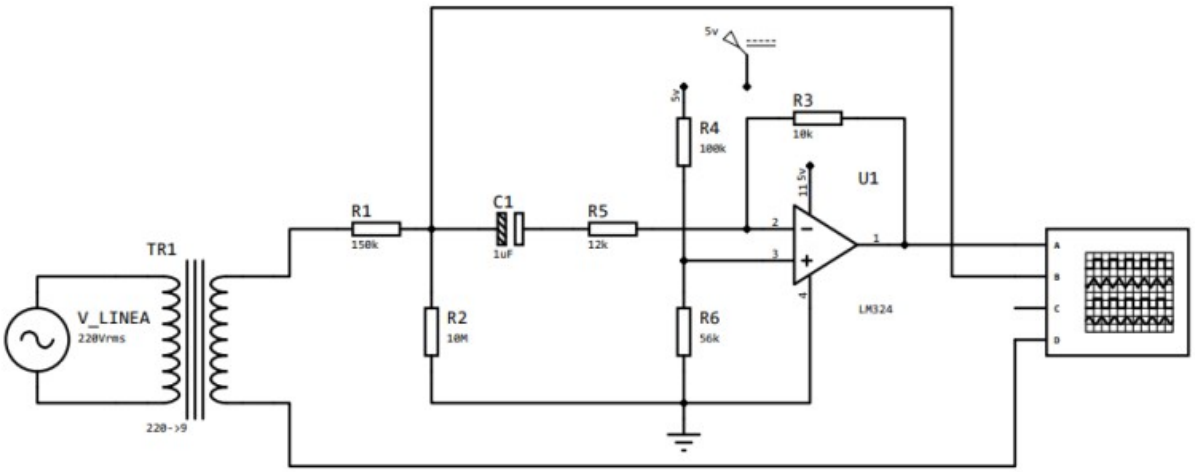
- **El microcontrolador Blue Pill**, del cual, a consideración a nuestro proyecto, una Arm® 32-bit Cortex®-M3 CPU core con máxima frecuencia de funcionamiento de 72 MHz, memoria FLASH 64 KB, memoria SRAM de 20KB, RTC, TIM2 y TIM3 (Timers de 16 bits)
- **Interfaces SPI e I2C**
- **2 conversores análogo/digital** con resolución de 12 bits y pines GPIO
- 2 **push-buttons,** utilizados para la interacción de parte del usuario para guardar las mediciones en memoria e intercambiar los menús en el LCD
- 1 **módulo SPI para micro SD**
- 1 **LCD** con un **módulo PCF8574** (I²C)
- 1 **LED-ROJO** indicador de si los datos han sido guardados en la memoria micro SD.



ESQUEMÁTICO SECCIÓN DIGITAL

Por la parte de medición, los circuitos utilizados fueron los siguientes:

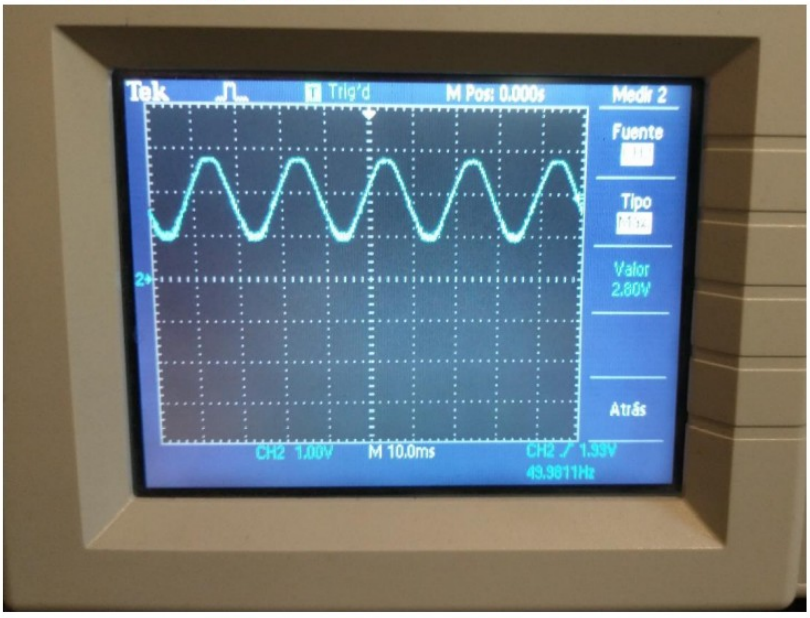
- **Tensión:** Para el circuito desarrollado para la adaptación de tensión se realizó lo que detalla el siguiente esquema:



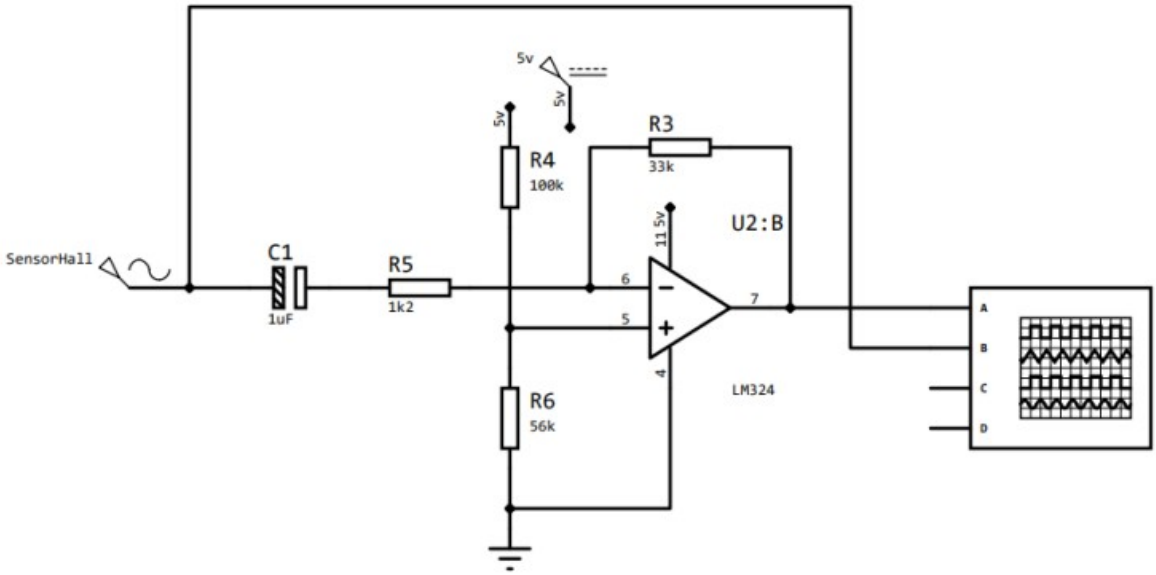
ESQUEMÁTICO ADAPTACIÓN DE TENSIÓN

Se redujo la tensión con un transformador monofásico, con una salida de 9V, cuya tensión se redujo posteriormente con un divisor resistivo. El conversor propio del microcontrolador tiene un rango de tensiones que va de 0V a 3,3V por lo que se tuvo que montar la señal alterna a una continua a traves de un amplificador operacional Se realizaron los cálculos mencionados anteriormente para la tensión en valores RMS.

Salida del amplificador operacional del circuito de tensión



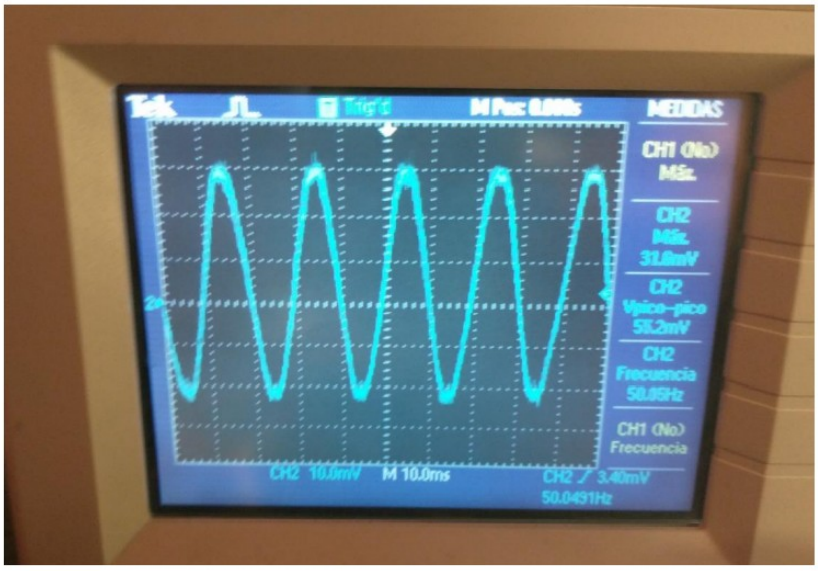
- **Corriente:** Se realizó el siguiente esquema:



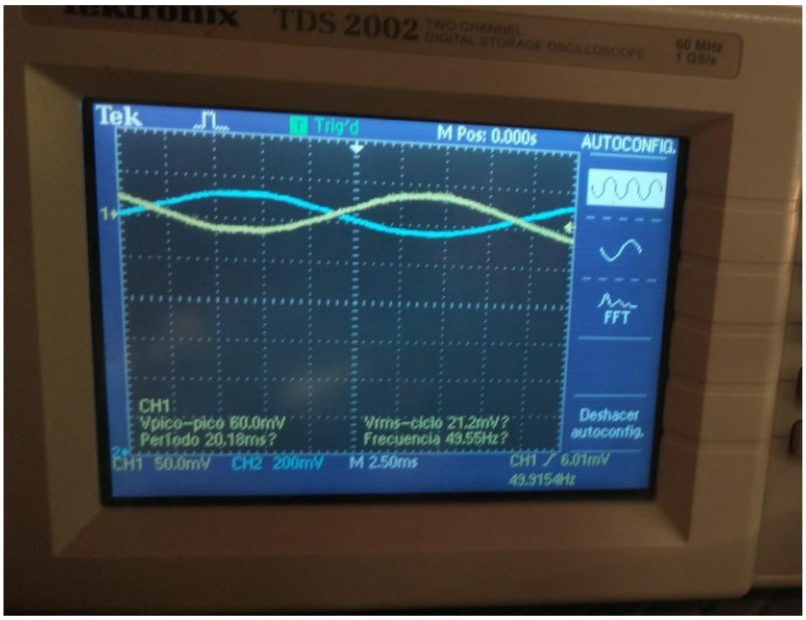
ESQUEMÁTICO ADAPTACIÓN DE CORRIENTE

Vemos que la salida del sensor de efecto Hall, se conectó directamente al operacional donde se realizó la misma lógica que para tensión, montar la señal alterna sobre un valor de continua debido a las especificaciones de los conversores A/D. Para este caso no fue necesario un atenuador resistivo ya que el sensor nos da valores de tensiones salida muy pequeños. NOTA: Los amplificadores operacionales utilizados corresponden a la pastilla del integrado LM324, el cual posee 4 operacionales diferentes.

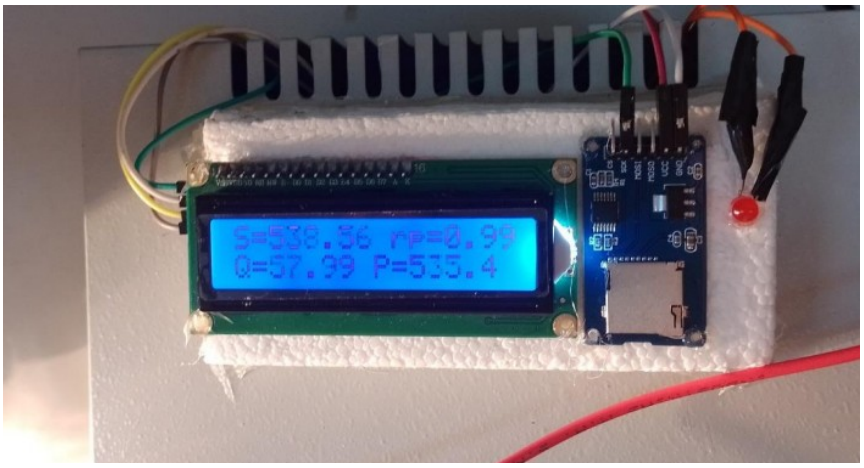
Salida del sensor de corriente con una carga de 100W



Comparación entre la entrada y salida del amplificador operacional del circuito de corriente (Canal 1 entrada y canal 2 salida)

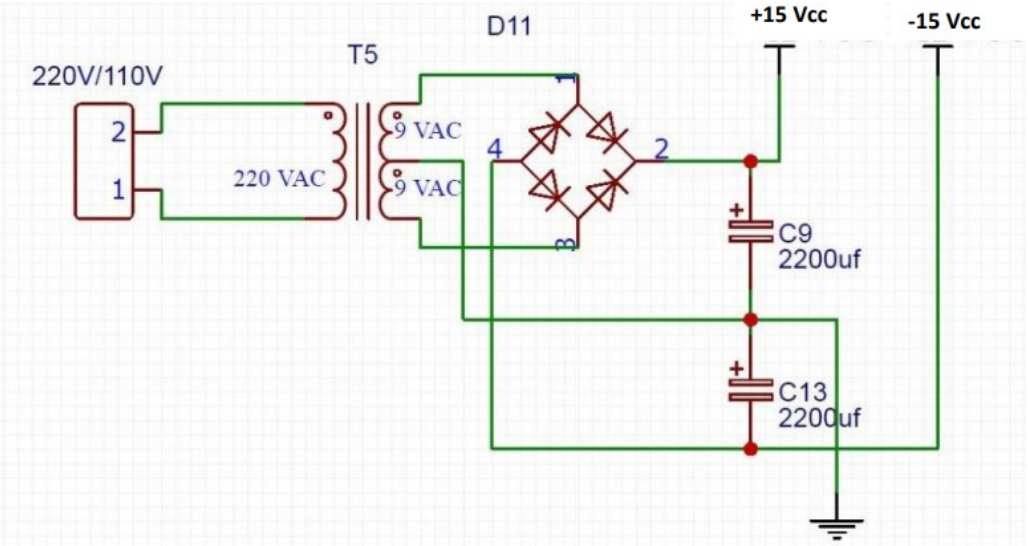


- **Muestra de datos**: Los datos son mostrados a través de un display LCD 16x2, el cual posee comunicación I 2C con el MCU. Se mostrará con una tasa de refresco muy alta. Además, se desarrolló la función de registrar los datos en una memoria SD, para que luego de haber realizado la medición, se pueda tener registro de los valores obtenidos en una tabla.

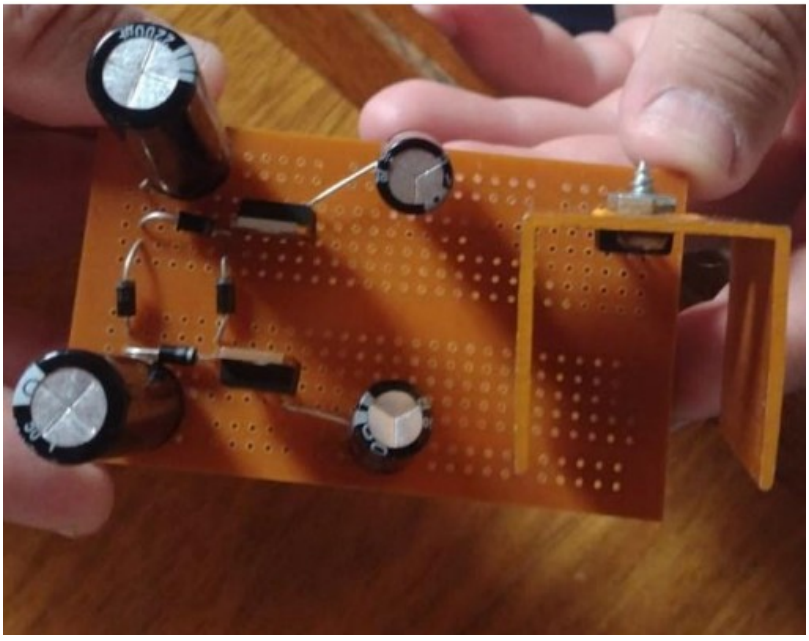


Se observan en las imágenes anteriores los valores de tensión, corriente. En la segunda imagen, donde se aumentó la carga, tenemos los valores de potencia aparente, reactiva y activa junto con el factor de potencia.

- **Fuente de alimentación**: Para la alimentación del instrumento se propuso realizar una fuente de tensión. Esta tiene que ser de tipo simétrica, ya que la alimentación del sensor es de +/-15V, se realizó con el siguiente circuito:



A la salida de los terminales de +15V y -15V se le conectó un regulador 7815 y 7915, respectivamente, para obtener una salida de tensión regulada. Ademas, se tuvo en cuenta la idea de alimentar los operacionales con 5V. Para ello, añadimos otro regulador 7805.



- **Controles**: Se dispondrá de dos pulsadores que controlan:

  - Grabación en la memoria SD.

  - Acceso al menú de datos mostrados en el LCD.

Diseño de Programa (Software)

El diseño del software se realizó con apoyo en las herramientas STM32CubeMx, Visual Studio Code, Github-Copilot, PlatformIO y STM32CubeProgrammer para el quemado del programa en el microcontrolador.

La inicialización consta de una calibración de los conversores AD, establecimiento de las comunicaciones por SPI e I²C, inicialización de los timers tim2 y tim3 y del RTC, y el seteo de los pines GPIO. Dentro del bucle while, se realiza la tarea de medir la entrada de los conversores y se establece la lógica para el guardado en la memoria.

El resto de las funciones a cumplir se realizan por medio de interrupciones. El desborde del timer 3 controla el período para tomar muestras de las señales, las cuales, una vez alcanzado el valor correspondiente a las muestras máximas, se realizan los cálculos pertinentes. El timer 2 es utilizado para realizar los refrescos de la pantalla LCD. Los botones push también generan interrupciones externas cuando son presionados. Una rutina es la que cambia el contenido en la pantalla en el LCD y la otra rutina realizada por el segundo botón es la que se encarga de realizar la carga de datos en la memoria.

La mayoría de las funciones se llevaron a cabo por medio de las librerías HAL y otras funciones tuvieron que ser agregadas por librerías hechas a mano (aunque también estas recurren a las librerías HAL en ocasiones)

Lo más complejo a realizar fue la parte para medir frecuencia y fase propuestos en un comienzo del proyecto. Con un poco de investigación, logramos eliminar esas dependencias para la obtención de los parámetros objetivos por lo que se pudo reemplazar las mediciones de frecuencia y fase por cálculos que se realizan en el microcontrolador sacrificando memoria SRAM (ya que se tuvo que almacenar muestras de las señales). Además de esto, se le sumó que se debió ajustar por software la alinealidad de los conversores.

A continuación se muestra el contenido de los archivos de código que son de relevancia para la lógica explicada:

main.c

```
/* USER CODE BEGIN Header */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------
*/
#include "main.h"
#include "adc.h"
#include "fatfs.h"
#include "i2c.h"
#include "rtc.h"
#include "spi.h"
#include "tim.h"
#include "gpio.h"
```

```
/* Private includes ----------------------------------------------------------
*/
/* USER CODE BEGIN Includes */
// #include <stdio.h>
#include "lcd_i2c.h"  // LCD library
#include <math.h>
#include <stdio.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------
*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------
*/
/* USER CODE BEGIN PD */
FATFS fs; // File system object
FIL fil;  // File object
// Capacity and free space
FATFS *pfs;
DWORD fre_clust;
uint32_t free_space;

// RTC
RTC_TimeTypeDef sTime;
RTC_DateTypeDef sDate;
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------
*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------
*/

/* USER CODE BEGIN PV */
uint8_t menu = 0;
uint8_t write_sd = 0;
uint8_t counting_period = 0;

const uint16_t adc_cross_zero_v = 2000;
const uint16_t adc_cross_zero_i = 1417;
const uint16_t max_samples = 4096;

// uint8_t frecuency = 50;
uint16_t sample_count = 0;  // It must be same size as max_samples

uint16_t adc1_value = 0;
uint16_t adc2_value = 0;
uint16_t last_adc_value[] = {0, 0}; // 0: V, 1: I

float voltage_sum = 0;
float RMS_voltage = 0;
float current_sum = 0;
float RMS_current = 0;
```

```
uint8_t frecuency = 50;
uint32_t period = 0;

float P = 0;
float power_sum = 0;

float S = 0;
float Q = 0;
float PF = 0;

/* USER CODE END PV */

/* Private function prototypes ———————————————————————————————
*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */

/* Private user code ————————————————————————————————————————
*/
/* USER CODE BEGIN 0 */
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
  /* USER CODE END 1 */

  /* MCU Configuration————————————————————————————————————————
*/

  /* Reset of all peripherals, Initializes the Flash interface and the 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_I2C1_Init();
  MX_ADC1_Init();
  MX_TIM2_Init();
  MX_ADC2_Init();
  MX_TIM3_Init();
  MX_SPI2_Init();
  MX_FATFS_Init();
```

```
  MX_RTC_Init();
  /* USER CODE BEGIN 2 */
  // Function to calibrate ADC
  HAL_ADCEx_Calibration_Start(&hadc1);
  HAL_ADCEx_Calibration_Start(&hadc2);
  HAL_Delay(100);

  // Init LCD
  Lcd_Init();
  Lcd_Clear();

  // Init SD card
  uint8_t mounted = 1;
  if (f_mount(&fs, "", 1) ≠ FR_OK) {
    Lcd_Set_Cursor(1,1);
    Lcd_Send_String("SD error");
    mounted = 0;
    HAL_Delay(3000);
  }
  // Check card capacity
  f_getfree("", &fre_clust, &pfs);
  // total = (uint32_t)((pfs→n_fatent - 2) * pfs→csize * 0.5);
  free_space = (uint32_t)(fre_clust * pfs→csize * 0.5);
  if (free_space < 100 && mounted == 1) {
    Lcd_Set_Cursor(1,1);
    Lcd_Send_String("SD full ");
    HAL_Delay(3000);
  }

  // HAL_GPIO_WritePin(ALERT_LED_GPIO_Port, ALERT_LED_Pin, GPIO_PIN_SET);

  // // Check if SD is workink ONLY DEBUG
  // if (f_open(&fil, "debug.txt", FA_OPEN_ALWAYS | FA_WRITE | FA_READ) ≠
FR_OK) {
  //   Lcd_Set_Cursor(1,1);
  //   Lcd_Send_String("File error");
  //   HAL_Delay(1500);
  // }
  // // Go to end of file
  // f_lseek(&fil, fil.fsize);
  // // Write header
  // f_puts("Debugging SD Card\n", &fil);
  // f_close(&fil);

  // HAL_GPIO_WritePin(ALERT_LED_GPIO_Port,ALERT_LED_Pin, GPIO_PIN_RESET);

  // Print fixed menu=0 messages
  Lcd_Clear();
  // LCD Voltage
  Lcd_Set_Cursor(1,1);
  Lcd_Send_String("V=");
  // LCD Current
  Lcd_Set_Cursor(2,1);
  Lcd_Send_String("I=");

  // LCD Power Factor
  Lcd_Set_Cursor(1,10);
```

```
  Lcd_Send_String("Fp=");

  // LCD Power
  Lcd_Set_Cursor(2,9);
  Lcd_Send_String("P=");

  // Start TMR2: refresh LCD
  HAL_TIM_Base_Start_IT(&htim2);
  // Start TMR3: ADC sampling
  HAL_TIM_Base_Start_IT(&htim3);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

    // Start ADC1: Current
    HAL_ADC_Start(&hadc1);
    // Start ADC2: Voltage
    HAL_ADC_Start(&hadc2);

    // Read ADC1: Current
    adc1_value = HAL_ADC_GetValue(&hadc1);
    // Read ADC2: Voltage
    adc2_value = HAL_ADC_GetValue(&hadc2);

    // Calculate frecuency
    if (adc2_value)

    if (write_sd == 1)
    {
      // Stop timer 2 and 3
      HAL_TIM_Base_Stop_IT(&htim2);
      HAL_TIM_Base_Stop_IT(&htim3);
      // Disable external interrupts B3 and B4
      HAL_NVIC_DisableIRQ(EXTI3_IRQn);
      HAL_NVIC_DisableIRQ(EXTI4_IRQn);

      HAL_GPIO_WritePin(ALERT_LED_GPIO_Port, ALERT_LED_Pin, GPIO_PIN_SET);
      // Write SD
      if (f_open(&fil, "data.csv", FA_OPEN_ALWAYS | FA_WRITE | FA_READ) ≠
FR_OK) {
        Lcd_Set_Cursor(1,1);
        Lcd_Send_String("File error");
        HAL_Delay(1500);
      }
      // Go to end of file
      f_lseek(&fil, fil.fsize);
      // header
      // f_puts("FECHA;Vrms;Irms;S;P;Q;Fp;\n", &fil);

      // Get date and time
```

```
      HAL_RTC_GetTime(&hrtc, &sTime, RTC_FORMAT_BIN);
      HAL_RTC_GetDate(&hrtc, &sDate, RTC_FORMAT_BIN);

      char buffer[100];
      sprintf(buffer, "%d/%d/%d-%d:%d;%d.%d%d;%d.%d%d%d;%d.%d%d;%d.%d%d;%d.%d
%d;%d.%d%d;\n",
        sDate.Date, sDate.Month, sDate.Year, sTime.Hours, sTime.Minutes,
        (uint8_t) RMS_voltage, (uint8_t) (RMS_voltage * 10) % 10, (uint8_t)
(RMS_voltage * 100) % 10,
        (uint8_t) RMS_current, (uint8_t) (RMS_current * 10) % 10, (uint8_t)
(RMS_current * 100) % 10, (uint8_t) (RMS_current * 1000) % 10,
        (uint8_t) S, (uint8_t) (S * 10) % 10, (uint8_t) (S * 100) % 10,
        (uint8_t) P, (uint8_t) (P * 10) % 10, (uint8_t) (P * 100) % 10,
        (uint8_t) Q, (uint8_t) (Q * 10) % 10, (uint8_t) (Q * 100) % 10,
        (uint8_t) PF, (uint8_t) (PF * 10) % 10, (uint8_t) (PF * 100) % 10
        // frecuency
        );
      f_puts(buffer, &fil);

      f_close(&fil);

      write_sd = 0;

      // Enable external interrupts B3 and B4
      HAL_NVIC_EnableIRQ(EXTI3_IRQn);
      HAL_NVIC_EnableIRQ(EXTI4_IRQn);
      // Continue timer 2 and 3
      HAL_TIM_Base_Start_IT(&htim2);
      HAL_TIM_Base_Start_IT(&htim3);
      HAL_GPIO_WritePin(ALERT_LED_GPIO_Port, ALERT_LED_Pin, GPIO_PIN_RESET);
    }

  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_LSI|
RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.LSIState = RCC_LSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
```

```
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) ≠ HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses 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();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_RTC|RCC_PERIPHCLK_ADC;
  PeriphClkInit.RTCClockSelection = RCC_RTCCLKSOURCE_LSI;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV8;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) ≠ HAL_OK)
  {
    Error_Handler();
  }
}

/* USER CODE BEGIN 4 */

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if (htim→Instance == TIM2)
    {
      if (menu == 0)
      {
        // Clear voltage LCD
        Lcd_Set_Cursor(1,3);
        Lcd_Send_String("       ");
        Lcd_Set_Cursor(2,3);
        Lcd_Send_String("      ");

        // Print values
        // voltage
        Lcd_Set_Cursor(1,3);
        Lcd_Send_Float(RMS_voltage,2);
        // Lcd_Send_Float(adc2_value,0);
        // Lcd_Send_Float(adc1_value,0);
        // current
        Lcd_Set_Cursor(2,3);
        Lcd_Send_Float(RMS_current,3);
        // Lcd_Send_Float(adc1_value,0);
        // Lcd_Send_Float(adc2_value,0);

        // Lcd_Set_Cursor(1,10);
        // Lcd_Send_Float(frecuency,0);
      }
```

```
      else
      {
        // Clear voltage LCD
        Lcd_Set_Cursor(1,3);
        Lcd_Send_String("       ");
        Lcd_Set_Cursor(2,3);
        Lcd_Send_String("      ");

        // Print values
        Lcd_Set_Cursor(1,3);
        Lcd_Send_Float(S,2);
        Lcd_Set_Cursor(2,3);
        Lcd_Send_Float(Q,2);
      }
      // Clear LCD
      Lcd_Set_Cursor(1,13);
      Lcd_Send_String("    ");

      // power factor
      Lcd_Set_Cursor(1,13);
      Lcd_Send_Float(PF,2);
      Lcd_Set_Cursor(2,11);
      Lcd_Send_Float(P,1);
    }
    else if (htim→Instance == TIM3)
    {
      if (sample_count < max_samples)
      {
        sample_count++;
        voltage_sum += (adc2_value - adc_cross_zero_v) * (adc2_value -
adc_cross_zero_v);
        current_sum += (adc1_value - adc_cross_zero_i) * (adc1_value -
adc_cross_zero_i);
        power_sum += (adc2_value - adc_cross_zero_v) * (adc1_value -
adc_cross_zero_i);
      }
      else
      {
        RMS_voltage = sqrt(voltage_sum * (22.5 / 111.845 * 1.4522) * (22.5 /
111.845 * 1.4522) / max_samples );
        RMS_current = sqrt(current_sum * ( 0.0470 / 149.68 * 18) * ( 0.0470 /
149.68 * 18 ) / max_samples);
        P = power_sum * (22.5 / 111.845 * 1.4522) * (0.0470 / 149.68 * 18) /
max_samples;
        if (P < 0)
        {
          P = P * -1;
        }

        S = RMS_voltage * RMS_current;
        Q = sqrt( (S * S) - (P * P) );
        PF = P / S;

        // Reset all variables
        sample_count = 0;
        voltage_sum = 0;
        current_sum = 0;
```

```
        power_sum = 0;
      }
    }
}

// Handle GPIO external interrupt PB3 and PB4
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if (GPIO_Pin == WRITE_SD_Pin)
  {
    if (write_sd == 0)
    {
      write_sd = 1;
    }
  }
  else if (GPIO_Pin == MOD_MENU_Pin)
  {
    if (menu == 1)
    {
      menu = 0;

      // Print fixed messages
      Lcd_Set_Cursor(1,1);
      Lcd_Send_String("V=");
      Lcd_Set_Cursor(2,1);
      Lcd_Send_String("I=");
    }
    else
    {
      menu = 1;

      // Print fixed messages
      Lcd_Set_Cursor(1,1);
      Lcd_Send_String("S=");
      Lcd_Set_Cursor(2,1);
      Lcd_Send_String("Q=");
    }
  }
}

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* 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 CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

adc.c

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
  * @file    adc.c
  * @brief   This file provides code for the configuration
  *          of the ADC instances.

******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *

******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------
*/
#include "adc.h"

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;

/* ADC1 init function */
void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */
```

```
  /* USER CODE END ADC1_Init 1 */

  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_4;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}
/* ADC2 init function */
void MX_ADC2_Init(void)
{

  /* USER CODE BEGIN ADC2_Init 0 */

  /* USER CODE END ADC2_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC2_Init 1 */

  /* USER CODE END ADC2_Init 1 */

  /** Common config
  */
  hadc2.Instance = ADC2;
  hadc2.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc2.Init.ContinuousConvMode = DISABLE;
  hadc2.Init.DiscontinuousConvMode = DISABLE;
  hadc2.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc2.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc2.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc2) != HAL_OK)
  {
    Error_Handler();
```

```
  }

  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_9;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) ≠ HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC2_Init 2 */

  /* USER CODE END ADC2_Init 2 */

}

void HAL_ADC_MspInit(ADC_HandleTypeDef* adcHandle)
{

  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(adcHandle→Instance==ADC1)
  {
  /* USER CODE BEGIN ADC1_MspInit 0 */

  /* USER CODE END ADC1_MspInit 0 */
    /* ADC1 clock enable */
    __HAL_RCC_ADC1_CLK_ENABLE();

    __HAL_RCC_GPIOA_CLK_ENABLE();
    /**ADC1 GPIO Configuration
    PA4     ————→ ADC1_IN4
    */
    GPIO_InitStruct.Pin = GPIO_PIN_4;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* USER CODE BEGIN ADC1_MspInit 1 */

  /* USER CODE END ADC1_MspInit 1 */
  }
  else if(adcHandle→Instance==ADC2)
  {
  /* USER CODE BEGIN ADC2_MspInit 0 */

  /* USER CODE END ADC2_MspInit 0 */
    /* ADC2 clock enable */
    __HAL_RCC_ADC2_CLK_ENABLE();

    __HAL_RCC_GPIOB_CLK_ENABLE();
    /**ADC2 GPIO Configuration
    PB1     ————→ ADC2_IN9
    */
    GPIO_InitStruct.Pin = GPIO_PIN_1;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

```
  /* USER CODE BEGIN ADC2_MspInit 1 */

  /* USER CODE END ADC2_MspInit 1 */
  }
}

void HAL_ADC_MspDeInit(ADC_HandleTypeDef* adcHandle)
{

  if(adcHandle->Instance==ADC1)
  {
  /* USER CODE BEGIN ADC1_MspDeInit 0 */

  /* USER CODE END ADC1_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_ADC1_CLK_DISABLE();

    /**ADC1 GPIO Configuration
    PA4     ------> ADC1_IN4
    */
    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_4);

  /* USER CODE BEGIN ADC1_MspDeInit 1 */

  /* USER CODE END ADC1_MspDeInit 1 */
  }
  else if(adcHandle->Instance==ADC2)
  {
  /* USER CODE BEGIN ADC2_MspDeInit 0 */

  /* USER CODE END ADC2_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_ADC2_CLK_DISABLE();

    /**ADC2 GPIO Configuration
    PB1     ------> ADC2_IN9
    */
    HAL_GPIO_DeInit(GPIOB, GPIO_PIN_1);

  /* USER CODE BEGIN ADC2_MspDeInit 1 */

  /* USER CODE END ADC2_MspDeInit 1 */
  }
}

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
```

lcd_i2c.c

```
#include "i2c.h"
#include "lcd_i2c.h"
#include <stdio.h>

void Lcd_Send_Cmd(char cmd)
```

```
{
	char data_u, data_l;
	uint8_t data_t[4];
	data_u = (cmd & 0xF0);
	data_l = ((cmd<<4) & 0xF0);
	data_t[0] = data_u|0x0C;
	data_t[1] = data_u|0x08;
	data_t[2] = data_l|0x0C;
	data_t[3] = data_l|0x08;
	HAL_I2C_Master_Transmit(&hi2c1, LCD_ADDRESS,(uint8_t*) data_t, 4, 100);
}

void Lcd_Send_Char(char data)
{
	char data_u, data_l;
	uint8_t data_t[4];
	data_u = (data & 0xF0);
	data_l = ((data<<4) & 0xF0);
	data_t[0] = data_u|0x0D;
	data_t[1] = data_u|0x09;
	data_t[2] = data_l|0x0D;
	data_t[3] = data_l|0x09;
	HAL_I2C_Master_Transmit(&hi2c1, LCD_ADDRESS,(uint8_t*) data_t, 4, 100);
}

void Lcd_Init(void)
{
	HAL_Delay(60);
	Lcd_Send_Cmd(0x02);
	Lcd_Send_Cmd(0x28);
	Lcd_Send_Cmd(0x0C);
	Lcd_Send_Cmd(0x80);
	Lcd_Send_Cmd(0x01);
}

void Lcd_Clear(void)
{
	Lcd_Send_Cmd(0x01);
	HAL_Delay(2);
}

void Lcd_Set_Cursor(int row, int col)
{
	uint8_t address;
	switch(row)
	{
		case 1:
			address = 0x00;
			break;
		case 2:
			address = 0x40;
			break;
		case 3:
			address = 0x14;
			break;
		case 4:
			address = 0x54;
```

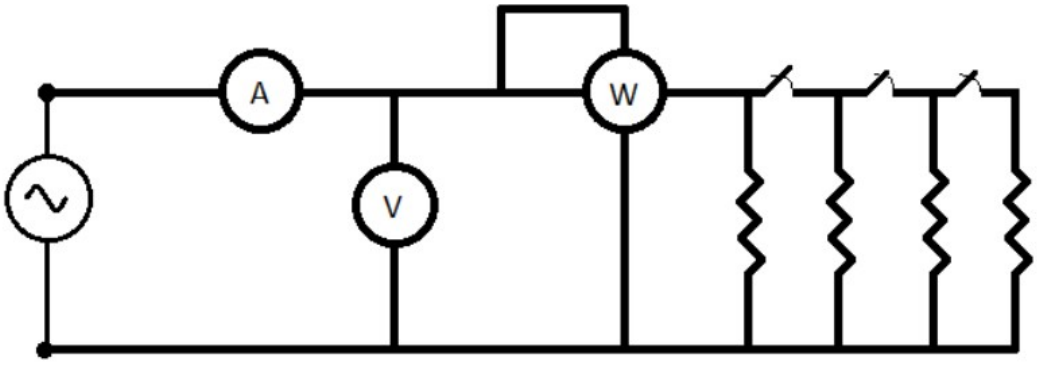
```
                break;
        }
        address += col - 1;
        Lcd_Send_Cmd(0x80 | address);

}

void Lcd_Send_String(char *str)
{
        while(*str) Lcd_Send_Char(*str++);
}

// void Lcd_Send_Float(float num)
// {
//     char buf[10];
//     int num_int = (int)(num);
//     int num_dec1 = (int)((num - num_int) * 10);
//     int num_dec2 = (int)((num - num_int - num_dec1 / 10.0) * 100);
//     sprintf(buf, "%d.%d%d", num_int, num_dec1, num_dec2);
//     Lcd_Send_String(buf);
// }

void Lcd_Send_Float(float num, int decimals)
{
        char buf[10];
        int num_int = (int)(num);
        if (decimals == 0)
        {
                sprintf(buf, "%d", num_int);
        }
        else if (decimals == 1)
        {
                int num_dec1 = (int)((num - num_int) * 10);
                sprintf(buf, "%d.%d", num_int, num_dec1);
        }
        else if (decimals == 2)
        {
                int num_dec1 = (int)((num - num_int) * 10);
                int num_dec2 = (int)((num - num_int - num_dec1 / 10.0) * 100);
                sprintf(buf, "%d.%d%d", num_int, num_dec1, num_dec2);
        }
        else if (decimals == 3)
        {
                int num_dec1 = (int)((num - num_int) * 10);
                int num_dec2 = (int)((num - num_int - num_dec1 / 10.0) * 100);
                int num_dec3 = (int)((num - num_int - num_dec1 / 10.0 - num_dec2 /
100.0) * 1000);
                sprintf(buf, "%d.%d%d%d", num_int, num_dec1, num_dec2, num_dec3);
        }
        Lcd_Send_String(buf);
}

void Lcd_Shift_Right(void)
{
        Lcd_Send_Cmd(0x1C);
}
```

```
void Lcd_Shift_Left(void)
{
      Lcd_Send_Cmd(0x18);
}

void Lcd_Blink(void)
{
      Lcd_Send_Cmd(0x0F);
}

void Lcd_NoBlink(void)
{
      Lcd_Send_Cmd(0x0C);
}

void Lcd_CGRAM_CreateChar(unsigned char pos, const char*msg)
{
    if(pos < 8)
    {
        Lcd_Send_Cmd(0x40 + (pos*8));
        for(unsigned char i=0; i<8; i++)
        {
            Lcd_Send_Char(msg[i]);
        }
    }
}

void Lcd_CGRAM_WriteChar(char pos)
{
    Lcd_Send_Char(pos);
}
```

No se incluirá código correspondiente al manejo de la memoria SD debido a que el módulo implica la creación de un objeto y funciones para la interfaz SPI.

3.3. Results of the design:

Se realizó la contrastación con un instrumento del laboratorio, y con diferentes cargas:



Para el ensayo se colocaron focos de filamento de 150W cada uno, y se fueron sumando en paralelo uno a uno. Para este ensayo de laboratorio se comprobó que la linealidad del sensor de corriente no es la especificada en la hoja de datos. El error para cada una de las cargas fue contradictorio. El vatímetro monofásico utilizado en el ensayo de contrastación es el mostrado en la foto a continuación.

TABLAS DEL ENSAYO DE LABORATORIO:

Contrastación					
FOCOS	Vatímetro Patrón En W	Vatímetro Dig. En W	Tensión indicada en el vatímetro	Corriente Indicada en el vatímetro	Error relativo
1	145	141	233,3 V	0,604 A	0.027
2	290	279	232,9 V	1,197 A	0.037
3	440	435	232,8 V	1,868 A	0.011
4	600	587,5	233,2 V	2,519 A	0.020
5	740	736,5	232,9 V	3,162 A	0.004
4	600	587,5	233,2 V	2,519 A	0.011
3	445	441	233,0 V	1,893 A	0.008
2	300	289	232,9 V	1,240 A	0.037
1	150	145	233,3 V	0,596 A	0.033

ERROR DETERMINADO:

POTENCIA ACTIVA: Error relativo: 0,037 – Error rel. porcentual: 3,7%

4. CONCLUSIONES

- Los resultados obtenidos fueron los esperados en un principio, ya que en el planteo de los mismo tuvimos en cuenta las limitaciones a causa de los materiales utilizados, el tiempo requerido, y otros factores en el desarrollo. En base a esto, se pueden realizar algunas mejorías mas allá de lo propuesto en un principio, ya que, partiendo del hecho de que el proyecto cumple su fin perfectamente podemos extender su alcance tanto en los rangos de medición como en la optimización tanto de componentes como de tamaño y exactitud en la medición.
- En base a lo mencionado anteriormente, pensamos que este proyecto es ampliable y perfeccionable en aspectos como:
    - Mayor precisión, sensibilidad y resolución de la medición
    - Disminución de componentes electrónicos
    - Reducción de circuitos y tamaño en general
    - Añadir un selector de rangos
    - Mejorar el sensor de corriente (principal limitación del proyecto), haciéndolo, por ejemplo, no invasivo, más lineal, preciso, etc
    - Permitir la lectura en redes trifásicas o de media/alta potencia.
    - Reducir los costos de fabricación
- No existen problemas legales ni de permisos, ya que el proyecto tiene fines educativos y no se pretende comercializar ni sacar provecho económico. El código es de nuestra autoría, aunque tiene referencia en repositorios de Github que sirvieron de guía en el desarrollo del mismo. Las herramientas de software utilizadas se basan en una política de software libre y licencias GNU.
- Gran cantidad de información se obtuvo a partir de los apuntes y bibliografía disponible en las cátedras de Medidas Electrónicas I y Técnicas Digitales II. Por parte, en el diseño del Hardware también se recurrió a la materia de Electrónica Aplicada II para la adaptación de las señales hacia el conversor análogo/digital por medio de amplificadores.

5. ANEXOS

HOJAS DE DATOS:

- Hoja de datos Sensor de corriente.
- Hoja de datos LM324.
- Hoja de datos STM32F103C8T6.
- Hoja de datos LCD 16x2 I2C.
- Hoja de datos de LM7815.
- Hoja de datos de LM7915.
- Hoja de datos de LM7805.