# Implementación de un algoritmo basado en FFT en un microcontrolador con motor DSP para la medición de distorsión armónica, potencia compleja, valores RMS, y factor de potencia de una carga monofásica para integrar en una red IoT.

Andrés Alejandro Moreno Sánchez
andres.moreno-s@mail.escuelaing.edu.co
Programa de Ingeniería Electrónica
Escuela Colombiana de Ingeniería Julio Garavito
Bogotá D.C., Colombia

***Resumen* - En este trabajo se implementa un sistema para analizar la red colombiana en tiempo real con equipos de bajo costo utilizando un microcontrolador de la familia ARM.**

***Palabras Clave - Armónicos, Transformada de Fourier,Series de Fourier, FFT, DSP***

***Abstract* - This work implements a system to analyze the Colombian network in real time with low-cost equipment using a microcontroller of the ARM family.**

***Key Words - Harmonics, Fourier Transform, Fourier Series, FFT, DSP***

## I. Introducción

La necesidad de monitorizar y medir nuestro consumo de energía siempre a sido un tema a tener en cuenta en la industria eléctrica, dado que la energía eléctrica no es un recurso ilimitado y su valor en el mercado varia constantemente. Se encuentra que los sistemas modernos de medición no son adecuados a las cargas no lineales que han invadido nuestras residencias en los últimos años generando problemas con sobre corrientes, incendios y multas a los residentes. Los métodos normales para poder medir correctamente estas cargas son muy costosos y demasiado grandes, por lo que son ineficientes para el proceso de medición en casas. El presente proyecto presenta la implementación de un algoritmo basado en FFT en un microcontroladores con motor DSP para la medición de distorsión armónica.

## II. Descripción Del Problema

Actualmente con la cantidad de dispositivos no lineales con los que contamos en las casas, los métodos de medición tradicional se vuelven ineficientes y poco fiables, esto es debido a que no están diseñados para tomar mediciones sobre todos los armónicos de la red, la solución convencional para medir estos redes es mediante un analizador de espectro, que es un elemento sumamente costoso por sus características, además de poco portable, sin contar que este dispositivo no realiza el calculo de las variables por cada ciclo de la señal, con el fin de obtener mas datos para su posterior análisis en una red IoT se requiere que se tomen los datos de cada ciclo de la señal, por lo que se requiere un método rápido y económico para poder realizar estas mediciones.

## III. Objetivos

Implementar un algoritmo basado en FFT, en un microcontrolador con motor DSP que permita el análisis de múltiples variables para la toma de datos de la red eléctrica que permita medir la distorsión armónica, potencia compleja, valores RMS y factor de potencia de una carga monofásica, para ser monitorizada e integrar en una red loT.

## IV. Justificación

Actualmente se busca medir y analizar una señal con las nuevas tecnologías de comunicación que existen para integrarlas a bases de datos en donde pueden ser monitorizadas. En estas redes se busca la implementación de una gran cantidad de dispositivos generando que el servidor no pueda procesar toda la información de los dispositivos entrantes en tiempo real. Para solucionar esto el procesamiento de la señal debe hacerse en el dispositivo, esto genera un problema, al utilizar microcontroladores normales como lo es un pic, este se queda corto por la gran cantidad de instrucciones que debe realizar, Utilizando microcontroladores de la familia DSPIC o ARM se puede reducir drásticamente la cantidad de instrucciones que debe llevar a cabo el microcontrolador, de esta forma se puede tener en tiempo real el análisis de la señal y subir al servidor de manera eficiente.

## V. Marco Teórico

### *V-A. Distorsión armónica en redes eléctricas*

Los armónicos se pueden definir como corrientes o tensiones de tipo sinusoidal estos tienen como característica que se

conforma de frecuencias las cuales son múltiplos enteros de la frecuencia en la que el sistema esta diseñado para operar [5].

Cuando ocurre una distorsión armónica es debido a características no lineales de los equipos y cargas de un sistema de potencia. Las ondas distorsionadas pueden ser descompuestas en una suma de la señal de frecuencia fundamental y las armónicas[1]. Para poder cuantificar el nivel de distorsión que puede presentar un armónico:

1 Factor de potencias.
2 Factor de crestas.
3 Potencia de distorsión.
4 Espectro en frecuencia.
5 Tasa de distorsión armónica.

El factor de la potencia se define como la relación entre potencia activa P y potencia aparente S. Una primera indicación de la presencia significativa de armónicos es cuando el factor de potencia medio es diferente del "cos" ( el factor de potencia será inferior)[3]. Se define como el valor de la cresta de corriente o de tensión (Im oVm) y el valor eficaz. El valor de cresta típico de corrientes absorbidas por cargas no lineales es mucho mayor que la raíz cuadrada de dos. Puede tomar valores de 1, 5, o 2 llegando incluso a 5 en casos críticos. La potencia P Activa de una señal distorsionada por armónicos es la suma de las potencias activas correspondientes a las tensiones e intencionalidades del mismo orden. Adicional a la potencia Activa se encuentra la Potencia Reactiva y la potencia de la distorsión. Identificar estos valores permite describir las tasas de distorsión armónica, la Distorsión armónica total y la distorsión total de demanda.[13].

### *V-B. Principales generadores de armónicos*

Entre las Fuentes Armónicas se encuentran los convertidores que son dispositivos que inyectan armónicas al sistema corriente alterna debido a la operación de los elementos switcheo (tiristores). Los Hornos de Inducción, utilizados en la industria manufacturera, es un rectificador e inversor, el cual controla la frecuencia de alimentación de una bobina. Los compensadores estáticos de Potencia utilizan tiristores para el control de la potencia reactiva. Los Hornos de Arco Eléctrico emplean electrodos los cuales a l hacer contacto con el acero se crea una arco eléctrico de tal magnitud que funde el acero. La Saturación de transformadores provoca generación de armónicas, pues se trata de un elemento no lineal generada por la saturación de las armónicas impares. Lámparas Fluorescentes se emplean como un, medio para reducir el consumo de energía [5].

### *V-C. Transformada de Fourier*

Este concepto matemático tiene su origen en 1811 de la mano de Joseph B. Fourier en conjunción con un tratado sobre la propagación de del calor, mediante un argumento de paso al limite a partir de las series de su misma autoría [2]. Esta transformada tiene un papel semejante al de la transformada de Laplace en las ecuaciones diferenciales, generando simplificaciones significativas para la solución del las mismas. El papel que la transformada de Fourier juega en el terreno de las aplicaciones, fundamentalmente en teoría de la señal, teoría cuántica de campos, tomografía y tratamiento y digitalización de imágenes, es también significativo e importante[6].
Dada una fucnion $f : R \Rightarrow C$, se define formalmente su transformada de Fourier como la función de variable real $\widehat{f} : R \Rightarrow C$, definida como:

$$\widehat{f}(\xi) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} e^{-i\xi x} f(x)\, dx,\ \xi\ \epsilon\ R \tag{1}$$

La definición anterior es formal en el sentido de que la integral que aparece en dicha definición no tiene porque existir para una f cualquiera.

### *V-D. Transformada de Fourier discreta*

Para tiempos discretos se tiene la transformada discreta de Fourier o DFT, esta esta definida de la siguiente manera.

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-\frac{2\pi j}{N} kn} \tag{2}$$

Donde N es el numero de muestras de x[n], x[n] es una señal de prueba discreta, X[k] Espectro en función de la frecuencia discreta.[17]

### *V-E. Series de Fourier*

Las series de Fourier son series de términos coseno y seno, usadas para representar funciones periódicas generales. Tienen una aplicación muy importante a la hora de resolver problemas que involucran ecuaciones diferenciales ordinarias y parciales. Estas son mas universales en cierta medida que las series de Taylor, ya que constituyen la herramienta mas importante en la solución de problemas con valores en la frontera.[16] La serie se define de la siguiente forma:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n cosnx + b_n sinnx) \tag{3}$$

De esta formula podemos definir las componentes armónicas, cuando n=0 se conoce como valor DC e indica si la señal tiene un nivel de offset, cuando n=1 se conoce como componente fundamental siendo esta la que da la frecuencia general de la señal y es la que tiene mas energía entre todos los armónicos.[16].

### *V-F. Algoritmo de FFT*

La Transformada Rápida de Fourier es un algoritmo para el cálculo de la Transformada Discreta de Fourier basado en la división del tiempo, eliminando así gran parte de los cálculos repetitivos que hay que llevar a cabo si se desea resolver la TFD de forma directa. Si hacemos una comparación del costo de los dos métodos, el cálculo directo de la TFD y la FFT, podemos observar el factor de mejora que brinda la FFT [18]

| $N$ | $N°$ de operaciones usando cálculo directo ($N^2$) | $N°$ de operaciones usando FFT ($N.log_2N$) | Factor de Mejora |
|---|---|---|---|
| 4 | 8 | 4 | 2,0 |
| 8 | 64 | 12 | 5,3 |
| 16 | 256 | 32 | 8,0 |
| 32 | 1.024 | 80 | 12,8 |
| 64 | 4.096 | 192 | 21,3 |
| 28 | 16.384 | 448 | 36,6 |
| 256 | 65.536 | 1.024 | 64,0 |
| 512 | 262.144 | 2.304 | 113,8 |
| 1.024 | 1.048.576 | 5.120 | 204,8 |
| $2^{30}$ | $2^{60}$ | $30 \times 2^{30}$ | 35.791.394,1 |

Figura 1: Tabla Comparativa que muestra la cantidad de operaciones a realizar con Calculo Directo y con FFT para diversos valores de N.[18]

### *V-G. Relación entre FFT y series de Fourier*

La relación que existe entre estos valores es la siguiente, Si el la muestra que se tomo coincide con un ciclo del periodo de la señal, la componente DC sera la primera posición de la FFT sobre la cantidad de datos de la muestra, para las demás componentes sera el valor de la FFT sobre la mitad de la cantidad de datos de la muestra.

### *V-H. Medición de variables en la red eléctrica por medio de la serie de Fourier*

Para utilizar este algoritmo necesitamos las componentes RMS de los armónicos de la serie de Fourier, por lo que primero multiplicamos todos los términos por $\frac{1}{\sqrt{2}}$ dado que todos los armónicos son señales senos o cosenos.[7]

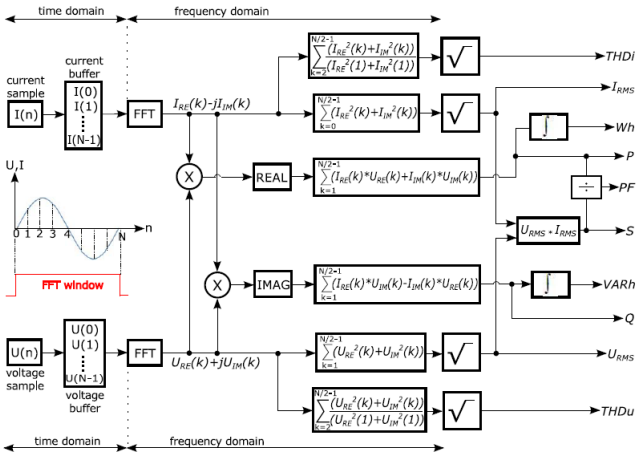



Figura 2: Ecuaciones para el calculo de las variables[7]

## VI. Metodología Utilizada

Con la revisión bibliográfica se selecciona el microcontrolador de la familia ARM, específicamente STM32 (Anexo 2) como el mejor para el trabajo. Se programa el microcontrolador STM32 CubeMX con el ambiente de desarrollo Atollic True STUDIO for STM32 IDLE. Se procede a Implementar diferentes algoritmos para realizar el trabajo y probar la velocidad de ejecución de los códigos, se implementa y verifica el funcionamiento. Con el análisis comparativo con un patrón de medición, se mide la distorsión armónica en redes eléctricas

## VII. Implementación

Se escoge el microcontrolador STM32F446RET dada su velocidad de operación, cuenta con una unidad de punto flotante que le permite realizar operaciones con floats en un solo flanco de reloj.

Para programar este microcontrolador se utilizan 2 programas, el primero es STM32CubeMX que permite configurar los puertos y periféricos de forma rápida y eficiente, El segundo se llama Atollic TrueSTUDIO for STM32, esta IDLE permite escribir códigos, depurar código y programar la tarjeta.

### *VII-A. Funcion de FFT*

Para comenzar con el codigo toca que tener en cuenta como se utiliza la FFT en el microcontrolador, en este chip existe una libreria optimizada llamada “arm_math.h", para poder utilizarla se siguen los siguientes pasos.

Al crear el proyecto en STM32CubeMX nos vamos a la seccion Proyect Manajer


Figura 3: Seleccionar Proyect Manajer

En pestaña de Code Generator seleccionar las siguientes opciones:

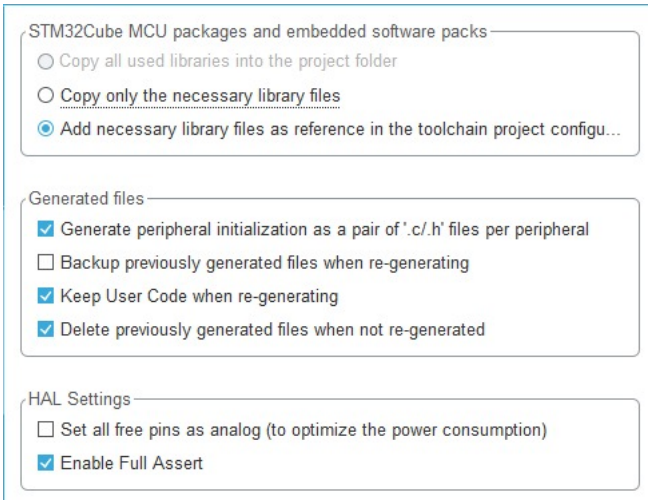



Figura 4: Configuración de Code Generator

En la pestaña de Project se coloca que el proyecto sera para TrueSTUDIO.


Figura 5: Configuración de Project

Después se genera el codigo y pasamos a Atollic TrueSTUDIO, En el se realizan las siguientes configuraciones. En la pestaña de Project seleccionar Properties:

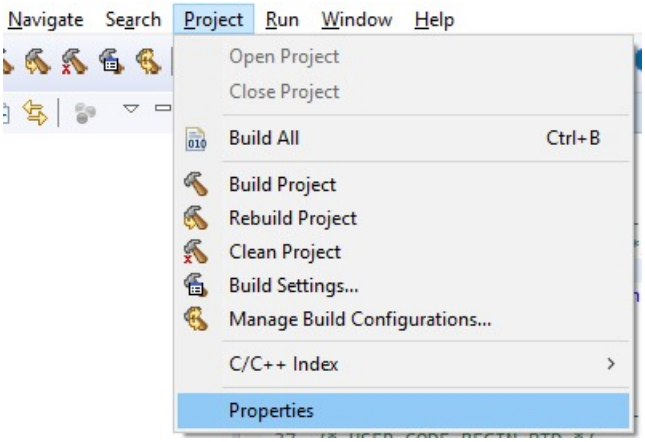



Figura 6: Pestaña Project

Desplegar el menú de C/C++ Build y seleccionar Settings:

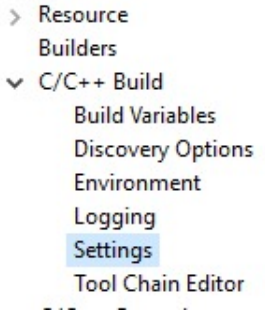



Figura 7: Menú de C/C++ Build

Desplegar el menú de C Compiler y seleccionar Target:

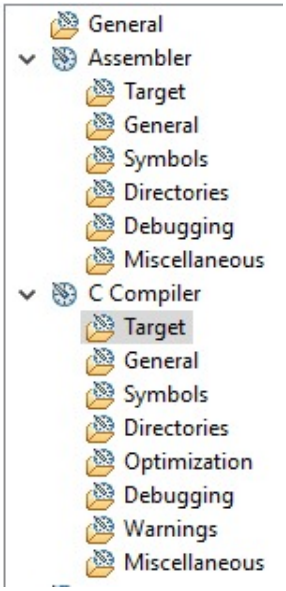



Figura 8: Menú de Tool Settings

En esta pestaña se colocan estas configuraciones:

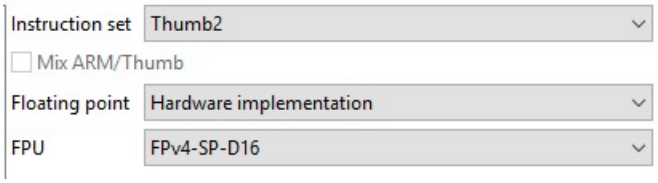



Figura 9: Configuraciones en Target

En el menú de C Compiler seleccionar Symbols y añadir ARM_MATH_CM4 en la lista.

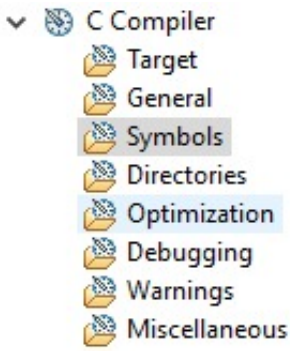



Figura 10: Menú de Tool Settings

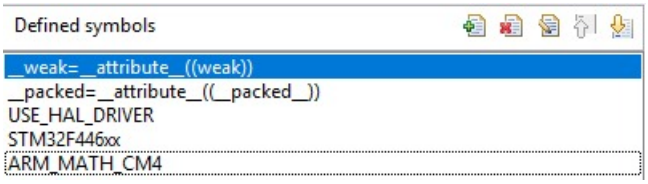



Figura 11: Símbolos definidos por el sistema

Una vez se configura de esta forma atollic, se abre en el programa el main y se coloca en la sección de Private includes estas dos librerías

```
/* Private includes
----------------------------------*
/* USER CODE BEGIN Includes */
#include "arm_math.h"
#include "arm_const_structs.h"
```

La recomendación para el uso de estas dos librerias es buscarlas en la carpeta

C:/Users/User/STM32Cube/Repository/STM32Cube _FW_F4_V1.24.1/Drivers/CMSIS/DSP

y copiar los siguientes archivos que se encuentran en la carpeta local de nuestro proyecto.

De la carpeta /DSP/Include se copian estos 3 archivos a la carpeta del proyecto /Core/Inc

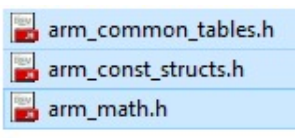



Figura 12: Archivos de /DSP/Include

De la carpeta /DSP/Source/CommonTables se copian estos 2 archivos a la carpeta del proyecto /Core/Src

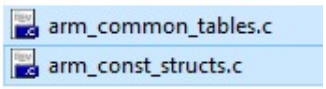



Figura 13: Archivos de /DSP/Source/CommonTables

De la carpeta /DSP/Source/TransformFunctions se copian estos 4 archivos a la carpeta del proyecto /Core/Src

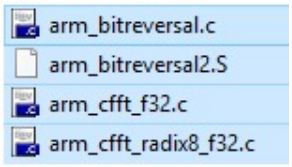



Figura 14: Archivos de /DSP/Source/TransformFunctions

Para finalizar y comprobar que el compilador esta reconociendo las librerias se hace ctrl+ clic en la libreria de “arm_math.h", una vez en la libreria se busca la siguiente linea de codigo

```
#elif defined (ARM_MATH_CM4)
  #include "core_cm4.h"
  #define ARM_MATH_DSP
```

Se entra en la libreria core y se busca la siguiente linea

```
#elif defined ( __GNUC__ )
  #if defined (__VFP_FP__) && !defined(__SOFTFP__)
    #if defined (__FPU_PRESENT) && (__FPU_PRESENT
        == 1U)
      #define __FPU_USED       1U
    #else
      #error "Compiler generates FPU instructions
          for a device without an FPU (check
          __FPU_PRESENT)"
      #define __FPU_USED       0U
    #endif
  #else
    #define __FPU_USED         0U
  #endif
```

Se va a modificar estas lineas de codigo de tal forma que se defina __FPU_PRESENT como 1U, para esto se comentan las siguientes lineas de codigo.

```
#elif defined ( __GNUC__ )
//  #if defined (__VFP_FP__) && !defined(
    __SOFTFP__)
//    #if defined (__FPU_PRESENT) && (
    __FPU_PRESENT == 1U)
      #define __FPU_USED       1U
//    #else
//      #error "Compiler generates FPU
    instructions for a device without an FPU (
    check __FPU_PRESENT)"
//      #define __FPU_USED       0U
//    #endif
//  #else
//    #define __FPU_USED         0U
//  #endif
```

Con esto ya se tiene la funcionando las funciones de FFT en el microcontrolador. Para el uso de las funciones de FFT se usara el tipo de variable llamado float32_t, este es un float de 32 bits.

Para el uso de función FFT compleja tenemos que definir el tamaño del vector que se va a usar, en este caso se usa un tamaño de 1024 datos. Se tiene que crear un vector del doble de tamaño en el que se manejaran como números complejos los datos, esto es de la forma que todas las posiciones pares son la parte real, y las posiciones impares son la parte imaginaria. Para guardar la señal de entrada, esta es una señal real por lo que se coloca en todas las posiciones impares 0, La función necesita otro parámetro de entrada que indica el numero de datos y unos apuntadores a una serie de tablas para operar. Hay que tener en cuenta que la función reescribe el vector de datos de entrada dejando un solo vector para su uso. un ejemplo de su uso es:

```
float32_t v1[2048];
v1[0]=119.99774103391213;
v1[1]=0;
v1[2]=119.99096422069734;
v1[3]=0;
.
.
.
v1[2046]=120.0;
v1[2047]=0;
arm_cfft_f32(&arm_cfft_sR_f32_len1024,v1,0,1);
```

### *VII-B. Algoritmo usado*

Para solucionar el problema de tener que tomar datos cada cierto tiempo y a su vez procesar la información se utilizan 2 vectores de datos, donde mientras uno se esta llenando el otro esta siendo procesado. Para lograr este funcionamiento se utiliza el ADC conectado con un timer donde al terminar una conversión se dispara una interrupción que permite procesar de forma preliminar la información y guardarla correctamente en su debido vector.

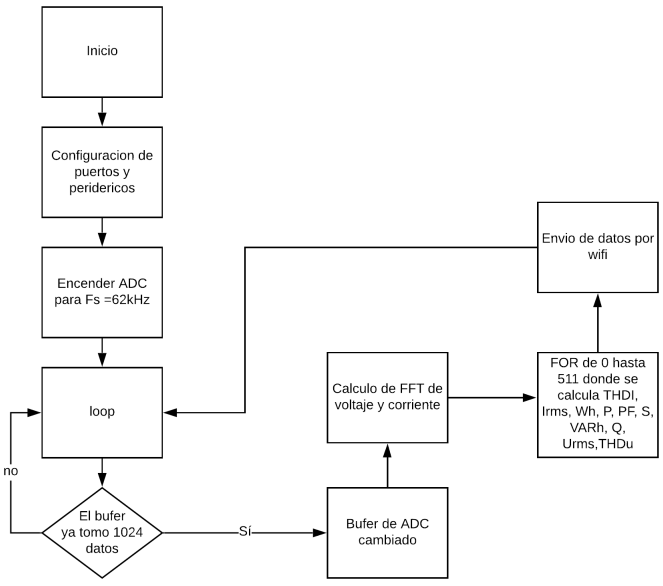



Figura 15: Diagrama de flujo

La configuración de los periféricos y el código completo implementado se encuentra en los anexos.

### *VII-C. Medición de voltaje y corriente*

La medición del voltaje se realiza mediante un divisor resistivo de gran impedancia, resistencia de 1 mega ohm, y un amplificador de instrumentación AD620 para referenciarlo con la tierra del circuito. La medición de corriente se realiza mediante el sensor de efecto hall ACS756 y un amplificador operacional lf347, dado que la señal que nos entrega es muy pequeña. Dado que el microcontrolador no puede recibir voltajes negativos se implementa un circuito de rectificación de precisión en las dos señales y un circuito con un comparador para determinar en que punto la señal es negativa y cuando positiva.

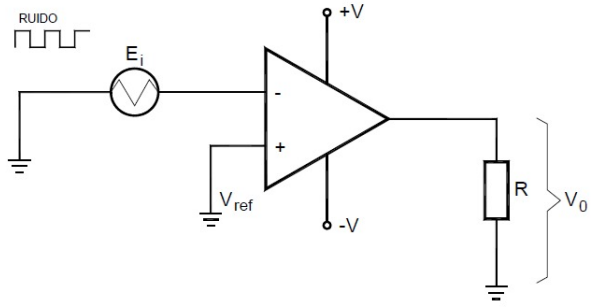



Figura 16: Comparador de cruce por 0

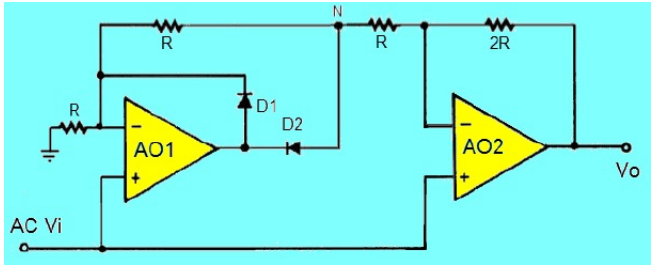



Figura 17: Rectificador de precisión

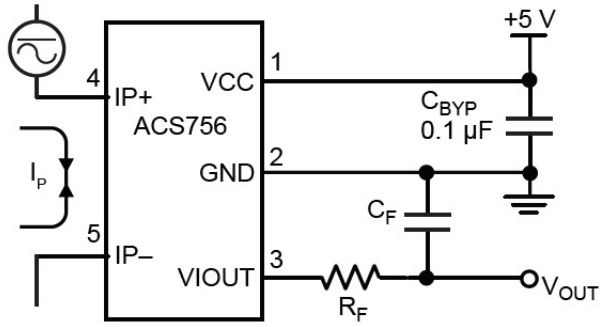



Figura 18: Sensor ACS756

Las señales que ingresan al microcontrolador son 4, 2 señales digitales que dan la información del signo de la señal, y 2 señales análogas que son la rectificación de las ondas de entrada, estas señales pueden estar en el rango de 0 a 3.3 V

## VIII. Resultados

Para comprobar el correcto funcionamiento del dispositivo se realizaron diferentes tomas de datos con diferentes cargas, estas fueron un generador de señales, una estación de soldadura y un computador. Tenemos que tener en cuenta que la relación para el voltaje es de 1 V a la entrada del microcontrolador equivale a 101.86 V en la red eléctrica. La relación de corriente es 1 V en la entrada del microcontrolador equivalen a 2.8114 amperios en la red eléctrica.

### *VIII-A. Estación de soldadura*

La estación de soldadura tiene 2 modos de funcionamiento, uno es cuando se esta calentando y consume mas corriente, el otro es cuando se encuentra en *Sleep* que consume una corriente mucho menor. Se van a mostrar las señales que se reciben de los sensores y la FFT de la señal de corriente, que es la señal que se distorsiona mas, también se mostraran los resultados que genera el microcontrolador y los datos de la FFT de la corriente que ve el microcontrolador, La señal de voltaje se vera en el canal 1 .ªmarilloz la de corriente en el canal 2 .ªzul".

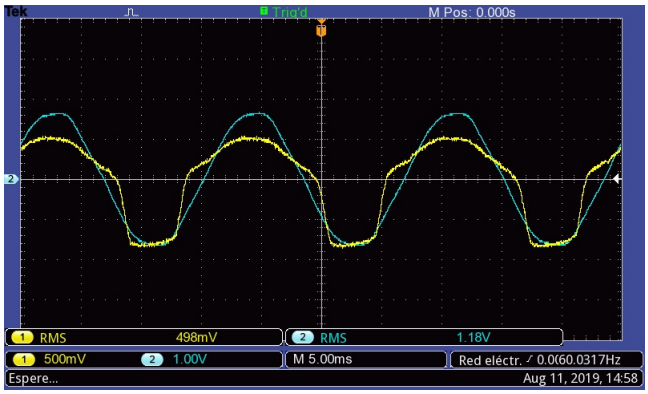



Figura 19: Señales de estación de soldadura en estado activo

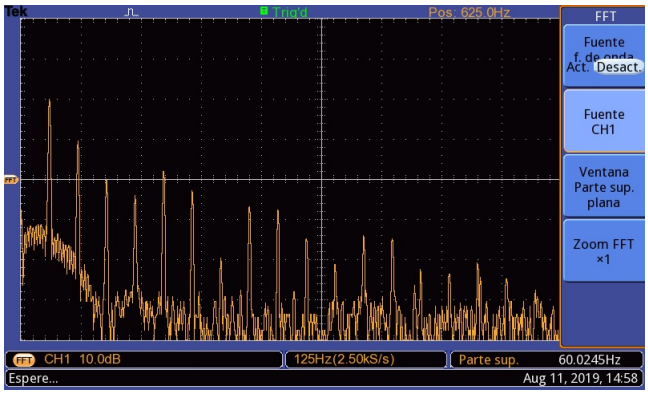



Figura 20: FFT en dB de corriente de estación de soldadura en estado activo

Los resultados mostrados de la FFT del osciloscopio estan en dB por lo que si queremos los valores de los armónicos tendremos que usar la formula :

$$a(n) = 10^{a_{dB}(n)/20} \tag{4}$$

Escalando los valores de los sensores a requeridos nos queda esta formula.

$$a(n) = 10^{a_{dB}(n)/20} * 2{,}8114 \tag{5}$$

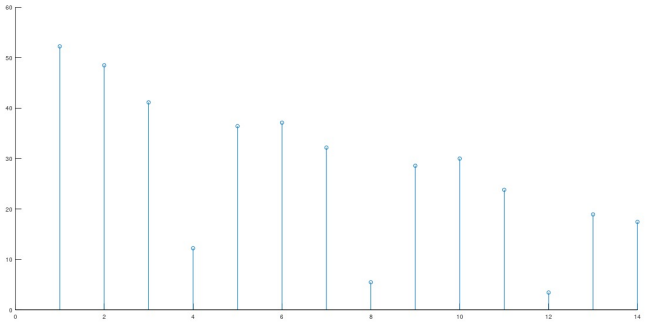


Figura 21: FFT en dB de corriente de estación de soldadura en estado activo

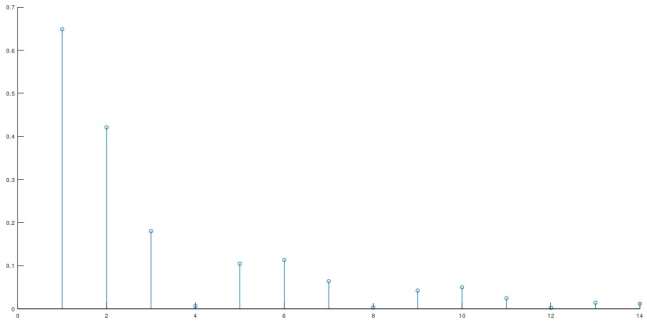


Figura 22: serie de Fourier de corriente de estación de soldadura en estado activo

En este estado los resultados del programa fueron los siguientes.

1 VRMS:115.52811
2 IRMS:1.3698407
3 THDu:0.0770416
4 THDi:0.71474224
5 Factor de potencia:0.64840657
6 Potencia activa:101.6146
7 Potencia reactiva:7.2371507
8 Potencia aparente:156.71432

El siguiente caso que se analizara sera cuando la estacion de soldadura se encuentra en estado de *Sleep* nuevamente se mostrara la señal de voltaje que se vera en el canal 1 .ªmarilloz la de corriente en el canal 2 .ªzul"

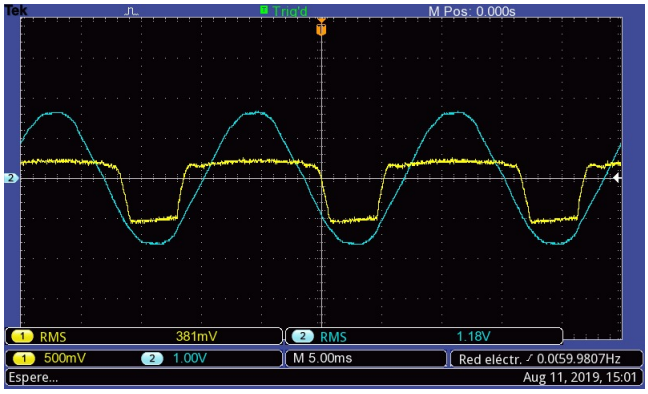



Figura 23: Señales de estación de soldadura en estado sleep

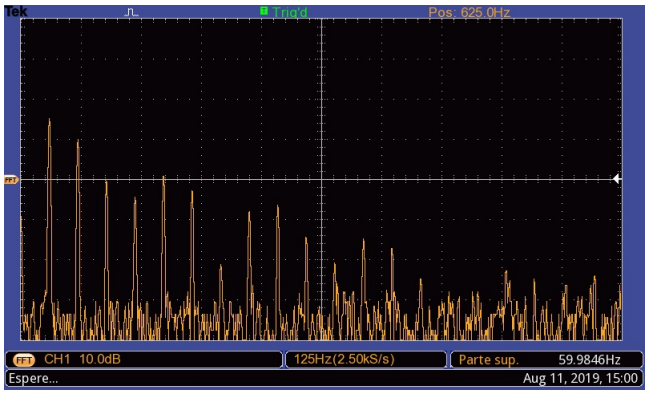



Figura 24: FFT en dB de corriente de estación de soldadura en estado sleep

Los resultados del microcontrolador :

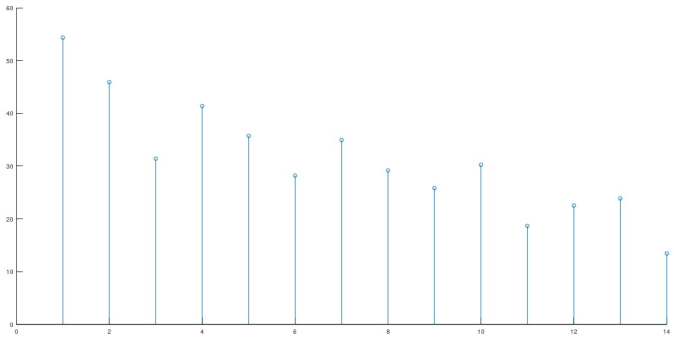


Figura 25: FFT en dB de corriente de estación de soldadura en estado sleep

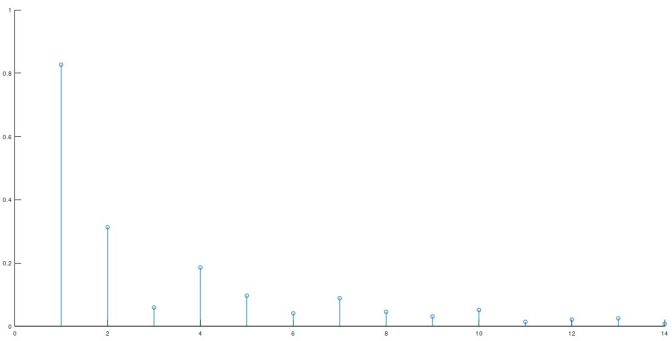


Figura 26: serie de Fourier de corriente de estación de soldadura en estado sleep

En este estado los resultados del programa fueron los siguientes.

1 VRMS:114.52873
2 IRMS:1.2680219
3 THDu:0.12613717
4 THDi:0.44038194
5 Factor de potencia:0.57968497
6 Potencia activa:84.184715
7 Potencia reactiva:3.5918787
8 Potencia aparente:145.22495

*VIII-B. Generador de señales*

El siguiente caso que se analizara sera un generador de señales de dos canales, nuevamente se mostrara la señal de voltaje que se vera en el canal 1 .ªmarilloz la de corriente en el canal 2 .ªzul"

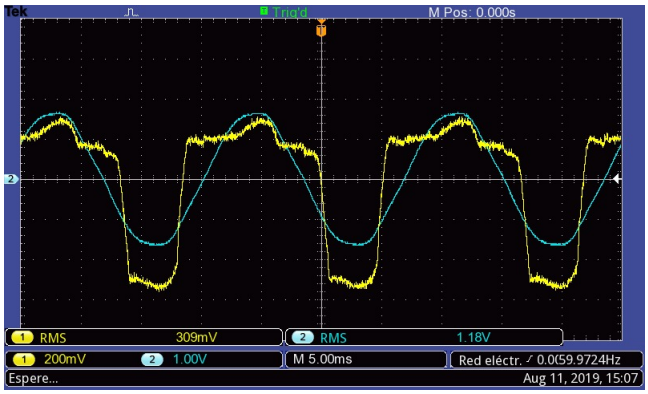



Figura 27: Señales de generador de señales

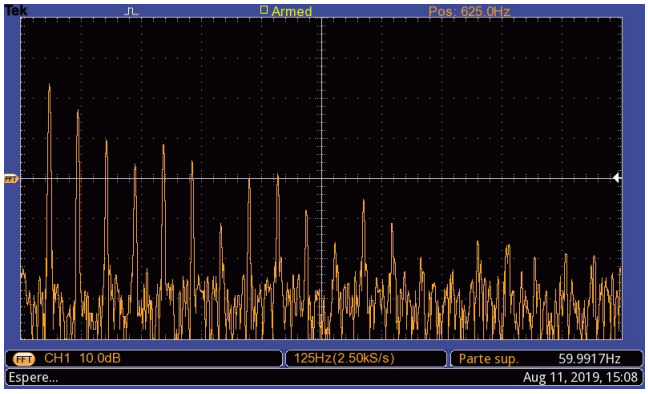



Figura 28: FFT en dB de corriente de generador de señales

Los resultados del microcontrolador :

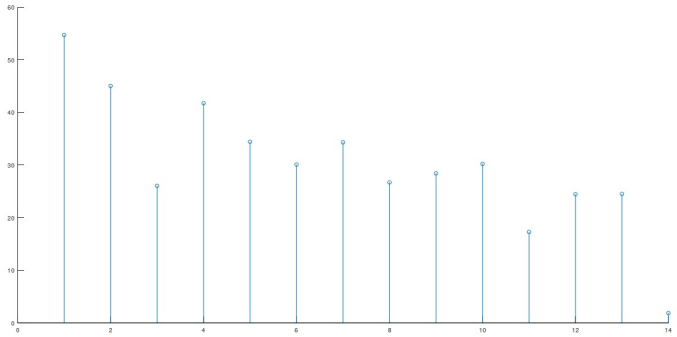


Figura 29: FFT en dB de corriente de generador de señales

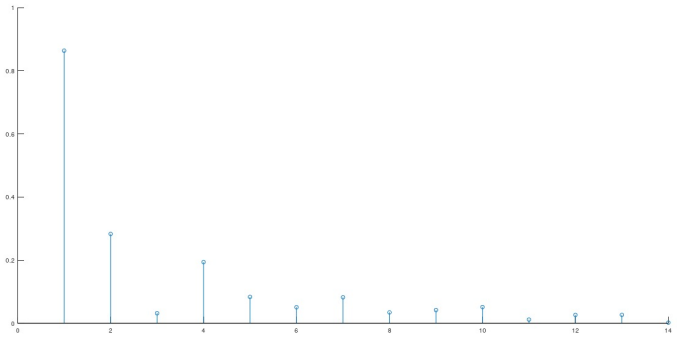


Figura 30: serie de Fourier de corriente de generador de señales

En este estado los resultados del programa fueron los siguientes.

1 VRMS:117.09824
2 IRMS:1.3778865
3 THDu:0.080704
4 THDi:0.62068677
5 Factor de potencia:0.65815949
6 Potencia activa:106.19278
7 Potencia reactiva:3.9295881
8 Potencia aparente:160.02969

### *VIII-C. Computador*

El siguiente caso que se analizara sera un computador portátil, nuevamente se mostrara la señal de voltaje que se vera en el canal 1 .ªmarilloz la de corriente en el canal 2 .ªzul"

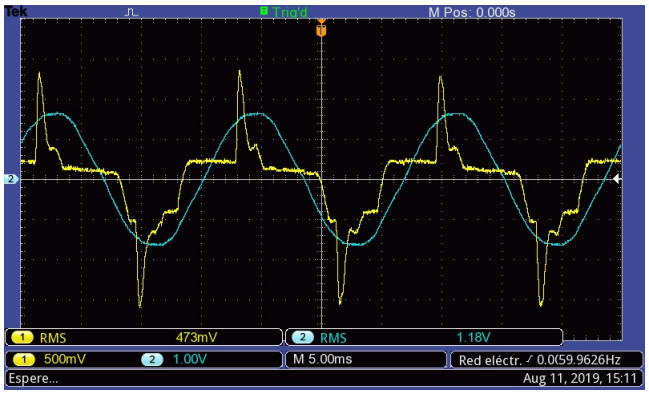



Figura 31: Señales del computador

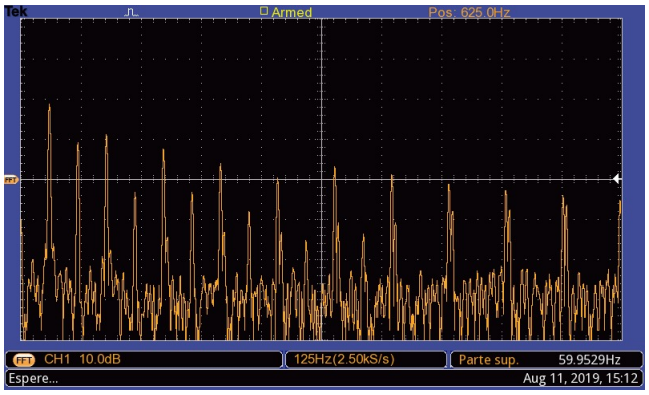



Figura 32: FFT en dB de corriente del computador

Los resultados del microcontrolador :

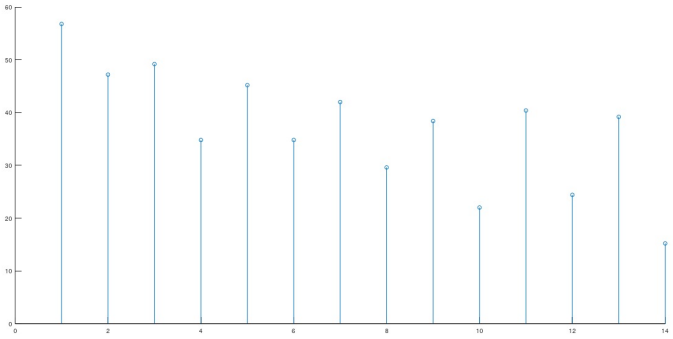


Figura 33: FFT en dB de corriente del computador

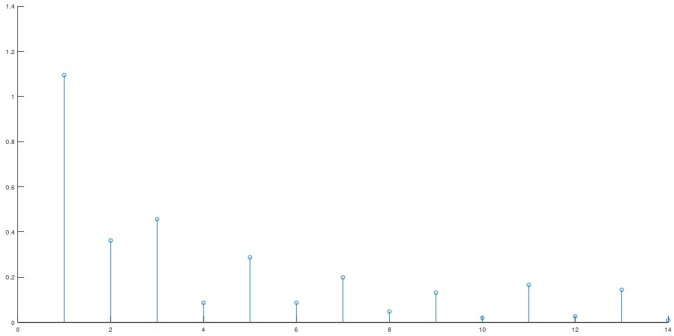


Figura 34: serie de Fourier de corriente del computador

En este estado los resultados del programa fueron los siguientes.

1 VRMS:117.09824
2 IRMS:1.74578
3 THDu:0.080704
4 THDi:0.897643
5 Factor de potencia:0.5134914
6 Potencia activa:181.31512
7 Potencia reactiva:15.15214412
8 Potencia aparente:210.02969

## IX. Conclusiones

1 Cuando tenemos cargas no lineales la potencias de la red no se pueden calcular de forma tradicional y se confirma que la diferencia entre los métodos es muy diferente.
2 Se logra la implementación del algoritmo utilizando el motor DSP que tiene el microcontrolador, con esto se logra que el procesamiento y transmisión de la señal se realice en 15.22 milisegundos.
3 El sistema permite el análisis de la distorsión de sistemas eléctricos no lineales. Es más confiable en la medición de las variables de la distorsión armónica en: potencia compleja, valores RMS y factor de potencia de una carga monofásica. para ser monitorizada e integrar en una red loT.

## Referencias

[1] Universidad Politecnica Salesiana, «Armonicos en las redes electricas».
[2] J. D. Arcila, «ARMÓNICOS EN SISTEMAS ELÉCTRICOS», p. 26.
[3] STMicroelectronics, «Digital signal processing for STM32 microcontrollers using CMSIS», Application note AN4841, 2018.
[4] «Digital signal processing for STM32 microcontrollers using CMSIS», p. 25.
[5] F. H. Martínez, «EL fENÓMENO DE DISTORSIÓN ARMÓNICA EN REDES ELÉCTRICAS», p. 9.
[6] O. Karpis, «FFT on ARM-Based Low-Power Microcontrollers», International Journal of Engineering Research and Development, vol. 9, pp. 67-72, abr. 2013.
[7] L. Slosarcik, «FFT-Based Algorithm for Metering Applications», Application Note AN4255, jul. 2015.
[8] R. M. y U. B., «IOT based Energy Meter Monitoring using ARM Cortex M4 with Android Application», IJCA, vol. 150, n.º 1, pp. 22-27, sep. 2016.
[9] P. Talwar y S. B. Kulkarni, «IOT Based Energy Meter Reading», International Journal of Recent Trend in Engineering Research, vol. 02, n.º 06, p. 6, 2016.
[10] R. Wade, «La transformada de Fourier: propiedades y aplicaciones», p. 45.

[11] STMicroelectronics, «Smart grid distribution and smart meters.pdf», 2015.
[12] «Smart grid distribution and smart meters.pdf». .
[13] «T025 LOS ARMONICOS EN LAS INSTALACIONES ELECTRICAS.pdf». .
[14] «Tema 2. Series y transformadas de Fourier.pdf». .
[15] «transformada_fourier_almira.pdf».
[16] J. Duoandikoetxea, «LECCIONES SOBRE LAS SERIES Y TRANSFORMADAS DE FOURIER», p. 181.
[17] J.-P. Caceres, «Transformada de Fourier», p. 20.
[18] A. Lucía, «FFT: Transformada Rápida de Fourier», p. 3.

# X. Anexos

## X-A. *Código del controlador*

```
/* USER CODE BEGIN Header */
/**
  ******************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2019
     STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST
     under BSD 3-Clause license,
  * the "License"; You may not use this file
     except in compliance with the
  * License. You may obtain a copy of the License
     at:
  *                        opensource.org/licenses
     /BSD-3-Clause
  *
  ******************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------*/
#include "main.h"
#include "adc.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------*/
/* USER CODE BEGIN Includes */
#include "arm_math.h"
#include "arm_const_structs.h"
#include <stdio.h>
#include "datos_def.h"
/* USER CODE END Includes */

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

/* USER CODE END PTD */

/* Private define ------------------------------*/
/* USER CODE BEGIN PD */
//#define _datosin
#define detec_errores
//#define datos_largos
#define SIGNOS GPIOC
#define Svoltage GPIO_PIN_2
#define Scourrent GPIO_PIN_3
#define voltaje_conv   8.20676436335403e-2
#define corriente_conv 2.26505660974121e-3
/* USER CODE END PD */

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

/* USER CODE END PM */

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

/* USER CODE BEGIN PV */
float32_t v1[2048],v2[2048],i1[2048],i2[2048];
volatile char buffok=0;
int num_errores=1;
char envfft=0,ant=0;
float32_t Wh=0,VARh=0,const_fac;
float32_t THDi=0,IRMS=0,P=0,PF=0,Ss=0,Q=0,URMS=0,
    THDu=0;
```

```c
/* USER CODE END PV */

/* Private function prototypes -----------*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */
#ifdef __GNUC__
        #define PUTCHAR_PROTOTYPE int __io_putchar
            (int ch)
#else
        #define PUTCHAR_PROTOTYPE int fputc(int ch
            , FILE *f)
#endif

/* USER CODE END PFP */

/* Private user code ---------------------------*/
/* USER CODE BEGIN 0 */
PUTCHAR_PROTOTYPE{
        HAL_UART_Transmit(&huart3,(uint8_t *)&ch
            ,1,0xffff);
        return ch;
}
/* 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_ADC1_Init();
  MX_ADC2_Init();
  MX_TIM2_Init();
  MX_USART3_UART_Init();
  /* USER CODE BEGIN 2 */
  float32_t ire1;
  float32_t iim1;
  float32_t vre1;
  float32_t vim1;
  float32_t ifrac;
  float32_t vfrac;
  float32_t irek;
  float32_t     iimk;
  float32_t vrek;
  float32_t     vimk;
  char buf[32]={0};
  const_fac=512*sqrt(2);
#ifdef _datosin
iniciardatos(v1,i1);
#else
  HAL_TIM_Base_Start(&htim2);
  HAL_ADC_Start_IT(&hadc1);
  HAL_ADC_Start_IT(&hadc2);
#endif
  printf("Hello STM32 \n\n");
  /* USER CODE END 2 */

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

    /* USER CODE BEGIN 3 */
        int cc=0;
#ifndef _datosin
        while(buffok==0);
#endif
        //inicia la fft para v1 y i1
        if(HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_13)
            ==1&&ant==0){
                envfft=1;
                ant=1;
        }
        if(HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_13)
            ==0&&ant==1){
                        envfft=0;
                        ant=0;
        }
        arm_cfft_f32(&arm_cfft_sR_f32_len1024,v1
            ,0,1);
        arm_cfft_f32(&arm_cfft_sR_f32_len1024,i1
            ,0,1);
        v1[0]=v1[0]/1024;
        v1[1]=v1[1]/1024;
        v1[2]=v1[2]/const_fac;
        v1[3]=v1[3]/const_fac;
        i1[0]=i1[0]/1024;
        i1[1]=i1[1]/1024;
        i1[2]=i1[2]/const_fac;
        i1[3]=i1[3]/const_fac;
        ire1=i1[2]*i1[2];
        iim1=i1[3]*i1[3];
        vre1=v1[2]*v1[2];
        vim1=v1[3]*v1[3];
        ifrac=1/(ire1+iim1);
        vfrac=1/(vre1+vim1);
        IRMS=i1[0]*i1[0]+i1[1]*i1[1];
        P=i1[0]*v1[0]+i1[1]*v1[1];
        Q=i1[0]*v1[1]-i1[1]*v1[0];
        THDi=0;
        IRMS+=ire1+iim1;
        P+=i1[2]*v1[2]+i1[3]*v1[3];
        Q+=i1[2]*v1[3]-i1[3]*v1[2];
        URMS=vre1+vim1+v1[0]*v1[0]+v1[1]*v1[1];
        THDu=0;
        if(envfft==1){
                printf("\n");
                float32_t auxxx;
                arm_sqrt_f32(i1[0]*i1[0]+i1[1]*
                    i1[1],&auxxx);
                gcvt(auxxx,15,buf);
                printf(buf);
                printf("\n");
                arm_sqrt_f32(ire1+iim1,&auxxx);
                gcvt(auxxx,15,buf);
                printf(buf);
        }
        for(cc=4;cc<1022;cc=cc+2){
                v1[cc]=v1[cc]/const_fac;
                v1[cc+1]=v1[cc+1]/const_fac;
                i1[cc]=i1[cc]/const_fac;
                i1[cc+1]=i1[cc+1]/const_fac;
                irek=i1[cc]*i1[cc];
                iimk=i1[cc+1]*i1[cc+1];
                vrek=v1[cc]*v1[cc];
                vimk=v1[cc+1]*v1[cc+1];
                THDi+=(irek+iimk)*ifrac;
                IRMS+=irek+iimk;
```

```
                P+=i1[cc]*v1[cc]+i1[cc+1]*v1[cc
                    +1];
                Q+=i1[cc]*v1[cc+1]-i1[cc+1]*v1[
                    cc];
                URMS+=vrek+vimk;
                THDu+=(vrek+vimk)*vfrac;
                if(envfft==1){
                        printf("\n");
                        float32_t auxxx;
                        arm_sqrt_f32(irek+iimk,&
                            auxxx);
                        gcvt(auxxx,15,buf);
                        printf(buf);
                }
        }
        arm_sqrt_f32(THDi,&THDi);
        arm_sqrt_f32(IRMS,&IRMS);
        arm_sqrt_f32(URMS,&URMS);
        arm_sqrt_f32(THDu,&THDu);
        Wh+=P*1.0/60.0*num_errores;
        VARh+=Q*1.0/60.0*num_errores;

        Ss=URMS*IRMS;
        PF=P/Ss;
        envfft=0;
        // envio de datoss
#ifdef datos_largos
        printf("Voltaje RMS = ");
        gcvt(URMS,15,buf);
        printf(buf);
        printf(" V\n");
        printf("Corriente RMS = ");
        gcvt(IRMS,15,buf);
        printf(buf);
        printf(" A\n");
        printf("Distorcion armonica en voltaje =
             ");
        gcvt(THDu,15,buf);
        printf(buf);
        printf("\n");
        printf("Distorcion armonica en corriente
             = ");
        gcvt(THDi,15,buf);
        printf(buf);
        printf("\n");
        printf("Factor de potencia = ");
        gcvt(PF,15,buf);
        printf(buf);
        printf("\n");
        printf("Potencia activa = ");
        gcvt(P,15,buf);
        printf(buf);
        printf(" W\n");
        printf("Potencia reactiva RMS = ");
        gcvt(Q,15,buf);
        printf(buf);
        printf(" Var\n");
        printf("Potencia aparente RMS = ");
        gcvt(Ss,15,buf);
        printf(buf);
        printf(" Va\n");
        printf("Potencia consumida RMS = ");
        gcvt(Wh,15,buf);
        printf(buf);
        printf(" Wh\n");
        printf("Potencia reactiva consumida RMS
            = ");
        gcvt(VARh,15,buf);
        printf(buf);
        printf(" Varh\n");
#else
        printf("VRMS:");
        gcvt(URMS,8,buf);
        printf(buf);
        printf(" IRMS:");
        gcvt(IRMS,8,buf);
        printf(buf);
        printf(" THDu:");
        gcvt(THDu,8,buf);
        printf(buf);
        printf(" THDi:");
        gcvt(THDi,8,buf);
        printf(buf);
        printf(" Fp:");
        gcvt(PF,8,buf);
        printf(buf);
        printf(" P:");
        gcvt(P,8,buf);
        printf(buf);
        printf(" Q:");
        gcvt(Q,8,buf);
        printf(buf);
        printf(" S:");
        gcvt(Ss,8,buf);
        printf(buf);
        printf(" Wh:");
        gcvt(Wh,8,buf);
        printf(buf);
        printf(" Varh:");
        gcvt(VARh,8,buf);
        printf(buf);
        printf("\n");
#endif
        num_errores=1;
#ifndef _datosin
        while(buffok==1);
#endif
        //inicia la fft para v2 y i2

        arm_cfft_f32(&arm_cfft_sR_f32_len1024,
            v2, 0, 1);
        arm_cfft_f32(&arm_cfft_sR_f32_len1024,
            i2, 0, 1);
        v2[0]=v2[0]/1024;
        v2[1]=v2[1]/1024;
        v2[2]=v2[2]/const_fac;
        v2[3]=v2[3]/const_fac;
        i2[0]=i2[0]/1024;
        i2[1]=i2[1]/1024;
        i2[2]=i2[2]/const_fac;
        i2[3]=i2[3]/const_fac;
        ire1=i2[2]*i2[2];
        iim1=i2[3]*i2[3];
        vre1=v2[2]*v2[2];
        vim1=v2[3]*v2[3];
        ifrac=1/(ire1+iim1);
        vfrac=1/(vre1+vim1);
        IRMS=i2[0]*i2[0]+i2[1]*i2[1];
        P=i2[0]*v2[0]+i2[1]*v2[1];
        Q=i2[0]*v2[1]-i2[1]*v2[0];
        THDi=0;
        IRMS+=ire1+iim1;
        P+=i2[2]*v2[2]+i2[3]*v2[3];
        Q+=i2[2]*v2[3]-i2[3]*v2[2];
        URMS=vre1+vim1+v2[0]*v2[0]+v2[1]*v2[1];
        THDu=0;
        for(cc=4;cc<1022;cc=cc+2){
                v2[cc]=v2[cc]/const_fac;
                v2[cc+1]=v2[cc+1]/const_fac;
                i2[cc]=i2[cc]/const_fac;
                i2[cc+1]=i2[cc+1]/const_fac;
                irek=i2[cc]*i2[cc];
                iimk=i2[cc+1]*i2[cc+1];
                vrek=v2[cc]*v2[cc];
                vimk=v2[cc+1]*v2[cc+1];
                THDi+=(irek+iimk)*ifrac;
                IRMS+=irek+iimk;
                P+=i2[cc]*v2[cc]+i2[cc+1]*v2[cc
                    +1];
                Q+=i2[cc]*v2[cc+1]-i2[cc+1]*v2[
                    cc];
                URMS+=vrek+vimk;
                THDu+=(vrek+vimk)*vfrac;
        }
```

```
            arm_sqrt_f32(THDi,&THDi);
            arm_sqrt_f32(IRMS,&IRMS);
            arm_sqrt_f32(URMS,&URMS);
            arm_sqrt_f32(THDu,&THDu);
            Wh+=P*1.0/60.0*num_errores;
            VARh+=Q*1.0/60.0*num_errores;
            Ss=URMS*IRMS;
            PF=P/Ss;

            // envio de datoss
#ifdef datos_largos
            printf("Voltaje RMS = ");
            gcvt(URMS,15,buf);
            printf(buf);
            printf(" V\n");
            printf("Corriente RMS = ");
            gcvt(IRMS,15,buf);
            printf(buf);
            printf(" A\n");
            printf("Distorcion armonica en voltaje =
                 ");
            gcvt(THDu,15,buf);
            printf(buf);
            printf("\n");
            printf("Distorcion armonica en corriente
                = ");
            gcvt(THDi,15,buf);
            printf(buf);
            printf("\n");
            printf("Factor de potencia = ");
            gcvt(PF,15,buf);
            printf(buf);
            printf("\n");
            printf("Potencia activa = ");
            gcvt(P,15,buf);
            printf(buf);
            printf(" W\n");
            printf("Potencia reactiva RMS = ");
            gcvt(Q,15,buf);
            printf(buf);
            printf(" Var\n");
            printf("Potencia aparente RMS = ");
            gcvt(Ss,15,buf);
            printf(buf);
            printf(" Va\n");
            printf("Potencia consumida RMS = ");
            gcvt(Wh,15,buf);
            printf(buf);
            printf(" Wh\n");
            printf("Potencia reactiva consumida RMS
                = ");
            gcvt(VARh,15,buf);
            printf(buf);
            printf(" Varh\n");
#else
            printf("VRMS:");
            gcvt(URMS,8,buf);
            printf(buf);
            printf(" IRMS:");
            gcvt(IRMS,8,buf);
            printf(buf);
            printf(" THDu:");
            gcvt(THDu,8,buf);
            printf(buf);
            printf(" THDi:");
            gcvt(THDi,8,buf);
            printf(buf);
            printf(" Fp:");
            gcvt(PF,8,buf);
            printf(buf);
            printf(" P:");
            gcvt(P,8,buf);
            printf(buf);
            printf(" Q:");
            gcvt(Q,8,buf);
            printf(buf);
            printf(" S:");
            gcvt(Ss,8,buf);
            printf(buf);
            printf(" Wh:");
            gcvt(Wh,8,buf);
            printf(buf);
            printf(" Varh:");
            gcvt(VARh,8,buf);
            printf(buf);
            printf("\n");
#endif
            num_errores=1;
  }
  /* USER CODE END 3 */
}

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

  /** Configure the main internal regulator output
       voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(
      PWR_REGULATOR_VOLTAGE_SCALE3);
  /** Initializes the CPU, AHB and APB busses
      clocks
  */
  RCC_OscInitStruct.OscillatorType =
      RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue =
      RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource =
      RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 10;
  RCC_OscInitStruct.PLL.PLLN = 192;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 5;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) !=
      HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB busses
      clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK
      |RCC_CLOCKTYPE_SYSCLK
                               |RCC_CLOCKTYPE_PCLK1
                                   |
                                   RCC_CLOCKTYPE_PCLK2
                                   ;
  RCC_ClkInitStruct.SYSCLKSource =
      RCC_SYSCLKSOURCE_PLLRCLK;
  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();
  }
}

/* USER CODE BEGIN 4 */
```

```
#ifndef _datosin
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef*
    hadc)
{
  /* Prevent unused argument(s) compilation
      warning */
  UNUSED(hadc);
  /* NOTE : This function Should not be modified,
      when the callback is needed,
            the HAL_ADC_ConvCpltCallback could be
                implemented in the user file
   */
  static int count=0;
  int a1,a2;
  a1=HAL_ADC_GetValue(&hadc1);
  a2=HAL_ADC_GetValue(&hadc2);
  if(buffok==0){
          //para equipo 1
          if(HAL_GPIO_ReadPin(SIGNOS,Svoltage)==1)
              {
                  v1[count]=a1*voltaje_conv;
          }
          else{
                  v1[count]=(-1.0)*a1*voltaje_conv
                      ;
          }
          if(HAL_GPIO_ReadPin(SIGNOS,Scourrent)
              ==1){
                  i1[count]=a2*corriente_conv;
          }
          else{
                  i1[count]=(-1.0)*a2*
                      corriente_conv;
          }
          count++;
          v1[count]=0;
          i1[count]=0;

  }
  else{
          if(HAL_GPIO_ReadPin(SIGNOS,Svoltage)==1)
              {
                  v2[count]=a1*voltaje_conv;
          }
          else{
                  v2[count]=(-1.0)*a1*voltaje_conv
                      ;
          }
          if(HAL_GPIO_ReadPin(SIGNOS,Scourrent)
              ==1){
                  i2[count]=a2*corriente_conv;
          }
          else{
                  i2[count]=(-1.0)*a2*
                      corriente_conv;
          }
          count++;
          v2[count]=0;
          i2[count]=0;
  }
  count++;
  if(count==2048){
        count=0;
#ifdef detec_errores
        if(buffok==0){
                if(fabs(v1[0]-v1[2046])<0.5){
                        buffok=1;
                }
                else{
                        num_errores++;
                }
        }
        else{
                if(fabs(v2[0]-v2[2046])<0.5){
                        buffok=0;
                }
                else{
                        num_errores++;
                }
        }
#else
        if(buffok==0){
                buffok=1;
        }
        else{
                buffok=0;
        }
#endif
  }
}
#endif
/* 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 can add his own implementation to report
       the HAL error return state */

  /* 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,
     tex: printf("Wrong parameters value: file %s
         on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT
    STMicroelectronics *****END OF FILE****/
```

# 1. Description

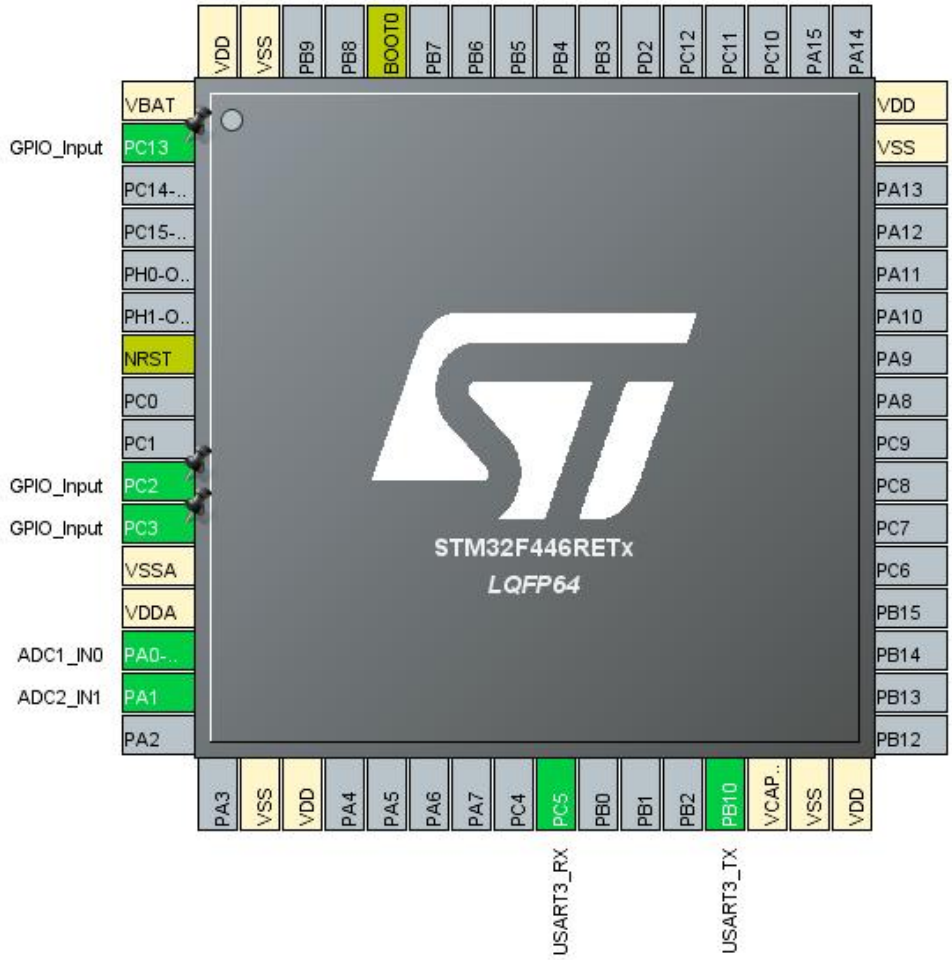
## 1.1. Project

| | |
|---|---|
| Project Name | intento7 |
| Board Name | NUCLEO-F446RE |
| Generated with: | STM32CubeMX 5.3.0 |
| Date | 12/11/2019 |

## 1.2. MCU

| | |
|---|---|
| MCU Series | STM32F4 |
| MCU Line | STM32F446 |
| MCU name | STM32F446RETx |
| MCU Package | LQFP64 |
| MCU Pin number | 64 |

## 2. Pinout Configuration

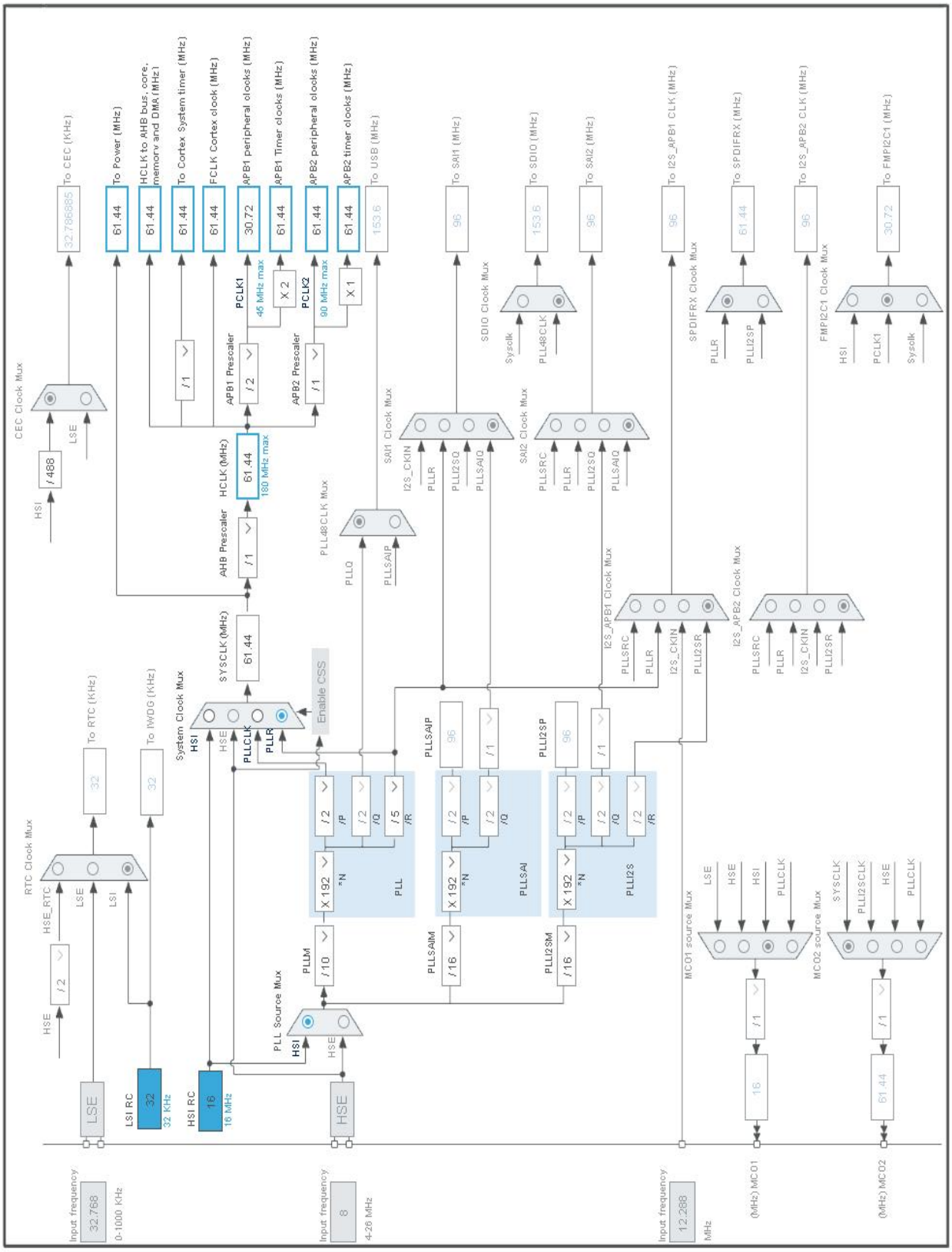


## 3. Pins Configuration

| Pin Number LQFP64 | Pin Name (function after reset) | Pin Type | Alternate Function(s) | Label |
|---|---|---|---|---|
| 1 | VBAT | Power | | |
| 2 | PC13 * | I/O | GPIO_Input | |
| 7 | NRST | Reset | | |
| 10 | PC2 * | I/O | GPIO_Input | |
| 11 | PC3 * | I/O | GPIO_Input | |
| 12 | VSSA | Power | | |
| 13 | VDDA | Power | | |
| 14 | PA0-WKUP | I/O | ADC1_IN0 | |
| 15 | PA1 | I/O | ADC2_IN1 | |
| 18 | VSS | Power | | |
| 19 | VDD | Power | | |
| 25 | PC5 | I/O | USART3_RX | |
| 29 | PB10 | I/O | USART3_TX | |
| 30 | VCAP_1 | Power | | |
| 31 | VSS | Power | | |
| 32 | VDD | Power | | |
| 47 | VSS | Power | | |
| 48 | VDD | Power | | |
| 60 | BOOT0 | Boot | | |
| 63 | VSS | Power | | |
| 64 | VDD | Power | | |

* The pin is affected with an I/O function

# 4. Clock Tree Configuration



# 5. Software Project

## 5.1. Project Settings

| Name | Value |
|---|---|
| Project Name | intento7 |
| Project Folder | C:\Users\Andres\Dropbox\2propuesta\intento7 |
| Toolchain / IDE | TrueSTUDIO |
| Firmware Package Name and Version | STM32Cube FW_F4 V1.24.1 |

## 5.2. Code Generation Settings

| Name | Value |
|---|---|
| STM32Cube MCU packages and embedded software | Add necessary library files as reference in the toolchain project configuration file |
| Generate peripheral initialization as a pair of '.c/.h' files | Yes |
| Backup previously generated files when re-generating | No |
| Delete previously generated files when not re-generated | Yes |
| Set all free pins as analog (to optimize the power consumption) | No |

# 6. Power Consumption Calculator report

6.1. Microcontroller Selection

| | |
|---|---|
| Series | STM32F4 |
| Line | STM32F446 |
| MCU | STM32F446RETx |
| Datasheet | 027107_Rev6 |

6.2. Parameter Selection

| | |
|---|---|
| Temperature | 25 |
| Vdd | 3.3 |

# 7. IPs and Middleware Configuration

## 7.1. ADC1

**mode: IN0**

### 7.1.1. Parameter Settings:

**ADCs_Common_Settings:**

| | |
|---|---|
| Mode | Independent mode |

**ADC_Settings:**

| | |
|---|---|
| Clock Prescaler | PCLK2 divided by 2 |
| Resolution | 12 bits (15 ADC Clock cycles) |
| Data Alignment | Right alignment |
| Scan Conversion Mode | **Enabled *** |
| Continuous Conversion Mode | Disabled |
| Discontinuous Conversion Mode | Disabled |
| DMA Continuous Requests | Disabled |
| End Of Conversion Selection | EOC flag at the end of single channel conversion |

**ADC_Regular_ConversionMode:**

| | |
|---|---|
| Number Of Conversion | 1 |
| External Trigger Conversion Source | **Timer 2 Trigger Out event *** |
| External Trigger Conversion Edge | Trigger detection on the rising edge |
| Rank | 1 |
| Channel | Channel 0 |
| Sampling Time | **84 Cycles *** |

**ADC_Injected_ConversionMode:**

| | |
|---|---|
| Number Of Conversions | 0 |

**WatchDog:**

| | |
|---|---|
| Enable Analog WatchDog Mode | false |

## 7.2. ADC2

**mode: IN1**

### 7.2.1. Parameter Settings:

**ADCs_Common_Settings:**

| | |
|---|---|
| Mode | Independent mode |

**ADC_Settings:**

| | |
|---|---|
| Clock Prescaler | PCLK2 divided by 2 |
| Resolution | 12 bits (15 ADC Clock cycles) |

| | |
|---|---|
| Data Alignment | Right alignment |
| Scan Conversion Mode | **Enabled *** |
| Continuous Conversion Mode | Disabled |
| Discontinuous Conversion Mode | Disabled |
| DMA Continuous Requests | Disabled |
| End Of Conversion Selection | EOC flag at the end of single channel conversion |

**ADC_Regular_ConversionMode:**

| | |
|---|---|
| Number Of Conversion | 1 |
| External Trigger Conversion Source | **Timer 2 Trigger Out event *** |
| External Trigger Conversion Edge | Trigger detection on the rising edge |
| Rank | 1 |
| Channel | Channel 1 |
| Sampling Time | **84 Cycles *** |

**ADC_Injected_ConversionMode:**

| | |
|---|---|
| Number Of Conversions | 0 |

**WatchDog:**

| | |
|---|---|
| Enable Analog WatchDog Mode | false |

## *7.3. RCC*

### 7.3.1. Parameter Settings:

**System Parameters:**

| | |
|---|---|
| VDD voltage (V) | 3.3 |
| Instruction Cache | Enabled |
| Prefetch Buffer | Enabled |
| Data Cache | Enabled |
| Flash Latency(WS) | 2 WS (3 CPU cycle) |

**RCC Parameters:**

| | |
|---|---|
| HSI Calibration Value | 16 |
| TIM Prescaler Selection | Disabled |
| HSE Startup Timout Value (ms) | 100 |
| LSE Startup Timout Value (ms) | 5000 |

**Power Parameters:**

| | |
|---|---|
| Power Regulatror Voltage Scale | Power Regulator Voltage Scale 3 |
| Power Over Drive | Disabled |

## 7.4. SYS

**Timebase Source: SysTick**

## 7.5. TIM2

**Clock Source : Internal Clock**

### 7.5.1. Parameter Settings:

**Counter Settings:**

| | |
|---|---|
| Prescaler (PSC - 16 bits value) | **10 *** |
| Counter Mode | Up |
| Counter Period (AutoReload Register - 32 bits value ) | **100 *** |
| Internal Clock Division (CKD) | No Division |
| auto-reload preload | Disable |

**Trigger Output (TRGO) Parameters:**

| | |
|---|---|
| Master/Slave Mode (MSM bit) | Disable (Trigger input effect not delayed) |
| Trigger Event Selection | **Update Event *** |

## 7.6. USART3

**Mode: Asynchronous**

### 7.6.1. Parameter Settings:

**Basic Parameters:**

| | |
|---|---|
| Baud Rate | 115200 |
| Word Length | 8 Bits (including Parity) |
| Parity | None |
| Stop Bits | 1 |

**Advanced Parameters:**

| | |
|---|---|
| Data Direction | Receive and Transmit |
| Over Sampling | 16 Samples |

*** User modified value**

# 8. System Configuration

## 8.1. GPIO configuration

| IP | Pin | Signal | GPIO mode | GPIO pull/up pull down | Max Speed | User Label |
|---|---|---|---|---|---|---|
| ADC1 | PA0-WKUP | ADC1_IN0 | Analog mode | No pull-up and no pull-down | n/a | |
| ADC2 | PA1 | ADC2_IN1 | Analog mode | No pull-up and no pull-down | n/a | |
| USART3 | PC5 | USART3_RX | Alternate Function Push Pull | Pull-up | **Very High *** | |
| | PB10 | USART3_TX | Alternate Function Push Pull | Pull-up | **Very High *** | |
| GPIO | PC13 | GPIO_Input | Input mode | No pull-up and no pull-down | n/a | |
| | PC2 | GPIO_Input | Input mode | No pull-up and no pull-down | n/a | |
| | PC3 | GPIO_Input | Input mode | No pull-up and no pull-down | n/a | |

## 8.2. DMA configuration

nothing configured in DMA service

## 8.3. NVIC configuration

| Interrupt Table | Enable | Preenmption Priority | SubPriority |
|---|---|---|---|
| Non maskable interrupt | true | 0 | 0 |
| Hard fault interrupt | true | 0 | 0 |
| Memory management fault | true | 0 | 0 |
| Pre-fetch fault, memory access fault | true | 0 | 0 |
| Undefined instruction or illegal state | true | 0 | 0 |
| System service call via SWI instruction | true | 0 | 0 |
| Debug monitor | true | 0 | 0 |
| Pendable request for system service | true | 0 | 0 |
| System tick timer | true | 0 | 0 |
| ADC1, ADC2 and ADC3 interrupts | true | 0 | 0 |
| PVD interrupt through EXTI line 16 | unused | | |
| Flash global interrupt | unused | | |
| RCC global interrupt | unused | | |
| TIM2 global interrupt | unused | | |
| USART3 global interrupt | unused | | |
| FPU global interrupt | unused | | |

*** User modified value**

# 9. Software Pack Report