Technical Report

Revista elektron, Vol. 7, No. 2, pp. 48-52 (2023)
https://doi.org/10.37537/rev.elektron.7.2.183.2023

Ecosistema de Interoperabilidad basado en C++/MicroPython sobre plataformas Raspberry y ESP32

Interoperability ecosystem based on C++/Micropython on Raspberry and ESP32 Platforms

Jorge Niño[1], Marcos Politi, Maximiliano Gulfo, Camilo Quiroga, Lucien Lucangioli, Hector Laiz
Dep. De Energías Renovables, Instituto Nacional de Tecnología Industrial
General Paz 5445, San Martín, Prov. de Bs. As, Argentina
[1]jnino@inti.gob.ar

Recibido: 29/09/23; Aceptado: 04/12/23

***Resumen* — El presente trabajo describe una solución de bajo costo basada en el procesador ESP32 y un controlador LoRa SX1276 para control de inversores fotovoltaicos con interfaz WiFi, tanto en lenguaje C++ como MicroPython. También se presenta una solución en Python para Raspberry. Para la prueba del firmware se empleó la plataforma HELTEC. Se desarrollaron además el hardware del ecosistema con diversas funcionalidades y un firmware configurable para manejo de múltiples interfaces con los inversores FV.**

Palabras Clave—LoRa, Raspberry, HTTP, JSON, ESP32, WiFi, Interoperabilidad, MicroPython, Python, IoE, IoT.

***Abstract* —This work describes a low cost solution based on ESP32 processor and the LoRa controller SX1276 for interoperability of photovoltaic inverters with WiFi interfaces. The system firmware is developed in C++ and MicroPython. Also, this work describes a solution on Python for Raspberry. For the test of firmware solution, we use a HELTEC platform. This work present also, a development of Hardware solution on PCB with more functionalities for different kind of interfaces in PV inverters.**

Keywords—LoRa, Raspberry, HTTP, JSON, ESP32, WiFi, Interoperability, MicroPython, Python, IoE, IoT.

I. INTRODUCCIÓN

Los inversores on-grid están diseñados para operar interconectados a la red eléctrica. Cuando hay energía solar disponible, el sistema alimenta esta energía a la instalación eléctrica que la requiera, y cuando la energía solar generada no es suficiente para satisfacer la demanda, el sistema dirige la energía de la red eléctrica. Si se genera más energía solar que la necesaria, la energía se inyectaría a la red. En determinados modelos de inversores fotovoltaicos se pueden encontrar tanto interfaces MODBUS como WiFi [1], que permiten acceder tanto a los datos de operación del inversor como a los parámetros de configuración y operación del mismo.

Se desarrollaron una serie de equipos capaces de interactuar con el inversor, en una red local dentro del rango del propio inversor. Los sistemas desarrollados están basados en un microcontrolador de 32 bits de ESPRESSIF, el ESP32, con el cual se realiza la comunicación con los inversores fotovoltaicos y la plataforma Raspberry Pi modelo 3B+. Se implementaron dos versiones del firmware, en MicroPython y C++. Esta estrategia permite diversificar los equipos y la potencia de procesamiento, que da flexibilidad al despliegue de gran cantidad de equipos, en redes complejas. Para la interfaz entre nodos de una red distribuida y el concentrador o Gateway [2], el sistema de comunicación escogido es LoRa, basándose en un controlador SX1276. Para el propio Gateway se emplea tanto el ESP32, programado bajo el firmware en C++, como una Raspberry programada en Python conectada a un controlador SX1276.

II. ESQUEMA GENERAL

A. *Topología de dispositivos*

La topología de la red, descripta en la Fig. 1, muestra al inversor conectado a la red local, mientras que el concentrador o Gateway accederá al servidor AWS donde se encuentra la plataforma de smartgrid. Por otro lado, se mantiene la conexión del equipo MODBUS-ETH [3].

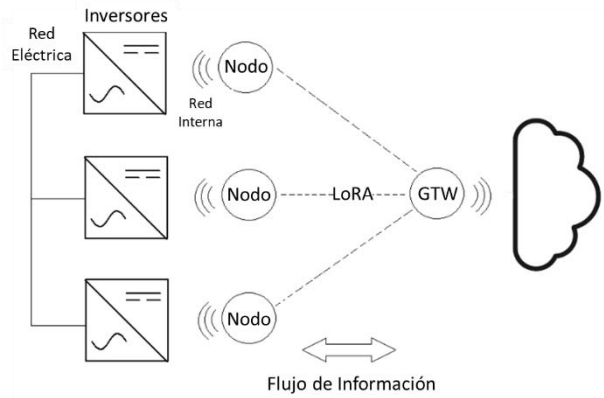

Fig. 1: Sistema de operación de inversores mediante Gateway-Nodo.

B. *Reversebilidad y flexibilidad*

La implementación de los nodos y Gateway WiFi-LoRa se realizó de tal manera que se puedan configurar sus funciones mediante una serie de macros, obteniendo flexibilidad y

sencillez en su puesta a punto, además de independizar el código del hardware en cierta medida, ya que la plataforma en ambos dispositivos puede ser la misma.

III. HARDWARE DEL EQUIPO.

El hardware del dispositivo fue diseñado en base a un ESP32 como procesador central, un módulo LoRa RFM95 que integra el controlador SX1276. A continuación, se detallan los componentes de cada parte del circuito.

A. Fuente de alimentación

Se empleó para la fuente de alimentación un regulador LM2576 [4], configurado para entregar 5 V, con una fuente de 12 VDC externa. Se utilizan tres reguladores LM1117 para obtener alimentación de 3.3 V y así poder suministrar suficiente corriente para todo el sistema.

B. Microcontrolador y Chip LoRa

Se conectó al módulo ESP32-WROOM, una serie de periféricos (conectores para sensores) y la interfaz SPI para el controlador LoRa.

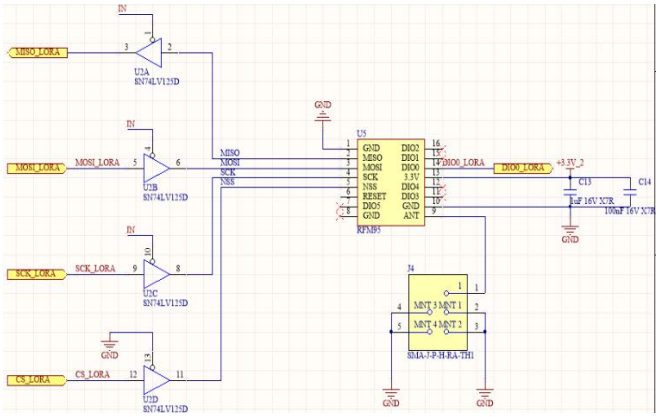

Fig. 2: Sub-circuito LoRa

El hardware cambia en función de la modalidad del nodo o GTW (Fig. 2) en el caso de tratarse de un Hardware para Nodo, se añaden más salidas para control de sistemas de operación (Relés, drivers de potencia), mientras que en los Gateway solo interesan las distintas conectividades (LoRa, WiFi, ETH, GSM/GPRS) aunque se disponen de algunas salidas para control.

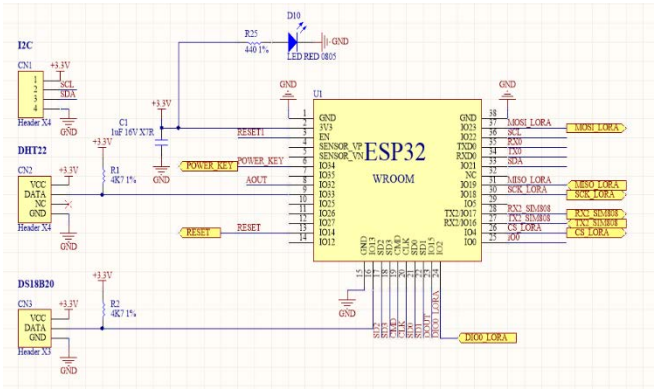

Fig. 3: Sub-circuito principal ESP32

C. Ethernet y otros perifericos

Se desarrollaron varias versiones de PCB (En la Fig. 3 se observa la versión para SIM808), entre ellas, se implementó una versión ETH-LoRa y otra con GPS y LoRa, dejando un conector libre para conectar un módulo Ethernet. En el primer caso, el circuito de Ethernet consiste en el adaptador SPI-ETH W5100 [5].

Una de las versiones de PCB incluye un chip SIM808, que incorpora un controlador GSM/GPRS con un periferico GPS y Bluetooth lo cual dota al sistema de la capacidad de emplear conexiones satelitales y celulares. En esta versión se incluye asimismo, el circuito para la operación de la memoria SD necesaria para el SIM808 que emplea la memoria SIM para la operación celular, así como las conexiones Serial y SPI para comunicarse con el microcontrolador central (ESP32). Por otro lado, una vez terminada, la primera versión del sistema se implementó como GTW (Fig. 4).

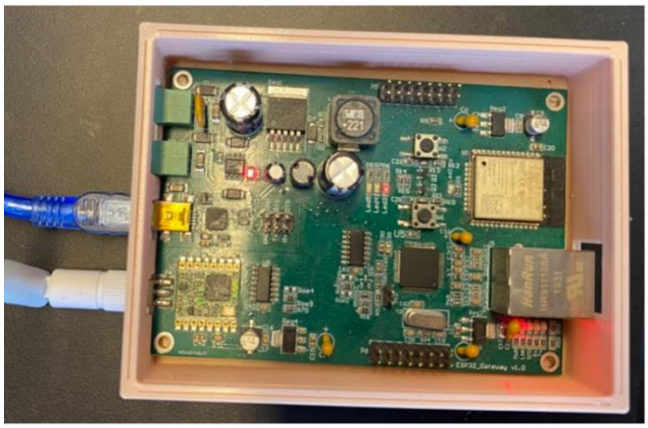

Fig. 4: PCB del equipo operando como GTW por WiFi.

D. Ecosistema en detalle.

A continuación, se detallan en la Tabla 1 los equipos con el procesador de base, el lenguaje empleado para la programación y su característica de operación:

Tabla 1.
Tabla de características por equipo.

Característica	Procesador	Lenguaje
Nodo/Gateway	ESP32	C++
Nodo	ESP32	µPython
Gateway	Broadcom-Raspberry	µPython
Gateway	ESP32	C++

IV. IMPLEMENTACIÓN DEL CÓDIGO

Para la implementación del código se emplearon las bibliotecas estándar de LoRa para ESP32, tanto en su versión para HELTEC como para módulos WROOM [6] [7]. Se incluyó una biblioteca desarrollada para inversores SMA, pero solo para la lectura través de JSON/HTTP [8], la cual se modificó intencionalmente para operar los parámetros del mismo. Para el sistema basado en MicroPython, se emplearon una serie de bibliotecas estándar que están ampliamente probadas, entre ellas la “u-lora” [9], que provee todas las funciones necesarias para operar en Python el controlador SX1276. Se implementó una máquina de estados general y un

método de macros y condicionales, para poder configurar fácilmente características del hardware en el cual se programa el firmware para C++. El mismo modelo de máquina de estados fue empleado para realizar el núcleo operativo del sistema sobre MicroPython.

A. Modificación de la bilbioteca SMA-Reader

Para leer el inversor SMA [10], se empleó la biblioteca SMA-Reader. Sin embargo, dicha biblioteca no tiene implementadas las funciones de operación sobre parámetros SMA. Por lo tanto, se implementaron una serie de funciones que permiten acceder a las "Keys" del SMA que contienen los parámetros del SMA que son de escritura/lectura. A continuación, se muestran los prototipos de las funciones implementadas:

1.- getAllParams()

2.- setParams(const String* key, uint32_t values)

La primera función permite extraer los parámetros sean cuales sean y sus correspondientes "claves" (direcciones). La segunda permite escribir en la "Key" correspondiente, un dato o una cadena de símbolos según sea necesario.

La función "setParams" genera un "postText" que se pasa a la función estándar del "postSMA", y el Json debe tener el siguiente formato:

"{\"destDev\":[],\"values\":[{Key(string):Dato(Tipo adecuado)}]}"

B. Implementación de Maquina de estados general

Se implementó una máquina de estados como en la Fig. 5. La misma permite realizar una secuencia de operaciones para realizar la comunicación con los periféricos de inversores y mediante LoRa, en ambos casos bidireccionales.

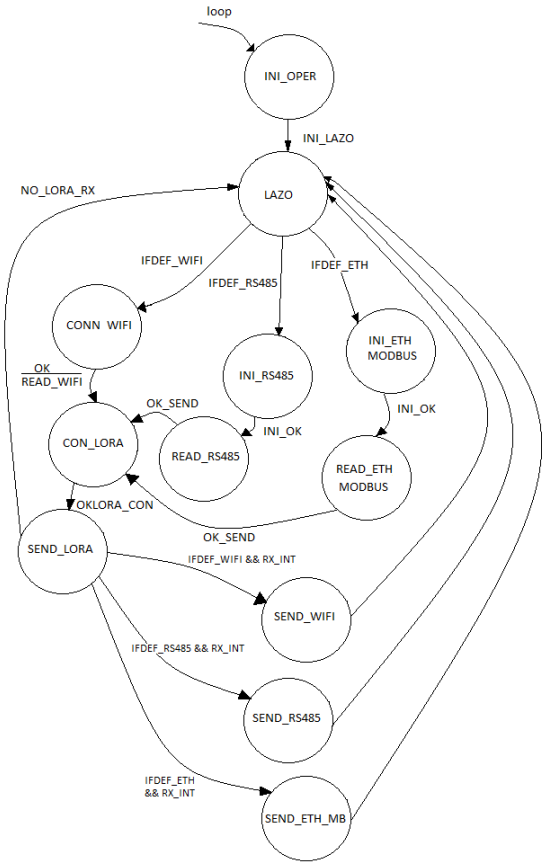

Fig. 5: Diagrama de Flujo de MdE principal.

Mediante una serie de macros se configuraron directivas para que el firmware pueda operar según la plataforma de hardware en la cual se está grabando. Este sistema de macros permite configurar el hardware sin tener que modificar la máquina de estados.

Esta implementación se realizó también en Python (Fig. 6) empleando las bibliotecas correspondientes y compilando la biblioteca SMA para poder ser usada por Pyhton.

```
def MDE_ini():
    global estado_ini
    switch = {0:ini_serie,1:ini_eth,2:ini_rs485,3:ini_wifi,4:ini_lora,5:ini_end}
    func=switch.get(estado_ini)
    return func(estado_ini)
def MDE_oper():
    global estado
    switch = {0:ini_oper,1:lazo,2:conect_lora,3:conect_wifi,4:conect_rs485,5:conect_eth,6:conect_med}
    func=switch.get(estado)
    return func(estado)
```

Fig. 6: Implementación de MdE en µPython.

Finalmente, se implementó el Sistema Nodo-Gateway sobre el inversor SMA. El Gateway fue ubicado dentro de una oficina a 300 metros del inversor y, como la red local de la institución es accesible desde todos los puntos, se aprovechó para colocar el nodo, primero cerca del inversor, pero luego en la misma oficina del Gateway (Fig. 7)

Fig. 7: Nodos recibiendo datos del inversor por la Red Inalámbrica a distancia.

V. IMPLEMENTACIÓN DE DASHBOARD DE CONTROL

En esta implementación se utilizó una instancia del sistema operativo Ubuntu Server 18.04 LTS, que torga la posibilidad de desplegar la VM (Virtual Machine) de manera estándar, o bien configurar parámetros a elección. Se eligen en este caso, algunas especificaciones estándar.

Como resultado ha quedado, una instancia con 1 CPUt2. micro, con 1 GB de memoria RAM, y 10 GB de disco duro, con los puertos HTTPS 443, HTTP 80, MQTT 8080 8083, FTP 21, para poder acceder al servicio instalando un servidor Apache y de esta manera ofrecer el servicio de *frontend* deseado.

A su vez se habilitó también el puerto1880, que fue utilizado para poder acceder al servicio NodeRed [11] [12],

que es la plataforma desarrolladora de aplicaciones a través de programación que combina lo visual por medio de flujos, y que fue implementada en el presente trabajo para interactuar las bases de datos.

Una vez que la instancia fue desplegada a través de PuTTY, se accedió a la misma para instalar los paquetes mosquitto y NodeRed dado que utilizaremos este paquete para desarrollar la plataforma de visualización.

Luego se procedió a la instalación de los paquetes de seguridad, para ello debe se instaló el paquete de seguridad de nodered.

De esta forma, la aplicación y sus datos quedan protegidos por medio de usuario y contraseña. Posteriormente, para que los flujos de NodeRed queden corriendo constantemente (Fig. 8) se instalaron los siguientes paquetes *pm2*, luego *start node-red*.

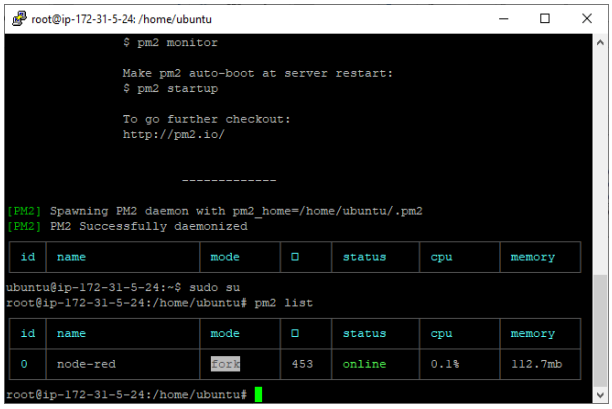

Figura 8: NodeRed iniciado.

Una vez desplegados los flujos e instalado el paquete Dashboard de NodeRed, se disponen los nodos y flujos de tal manera que podamos implementar la solución que deseamos.

Para poder guardar la información reportada de los endpoint se instaló MySQL.

Luego, se procedió a construir una base de datos y tablas para almacenamiento de datos, una para cada magnitud a visualizar.

VI. Interacción NodeRed MYSQL

En NodeRed fue necesario el despliegue de flujos que fuera consecuentes con la recopilación de datos a través del Broker de mosquitto, y la interacción con MySQL.

Finalmente fue posible concebir un sistema que interactúe con sistemas fotovoltaicos que permitan la integración de los mismos en una misma plataforma, con la posibilidad de visualizar en vivo, valores históricos y gestionarlos (Fig. 9)

Figura 9: Índice sitio smartgrid.ar

En el índice de dicha página se incluyó la posibilidad de visualización de datos en vivo, como así también datos históricos, y su descarga, tal como puede observarse en la figura 10.

Figura 10: Visualización de valores históricos de potencia.

Sobre el servidor AWS previamente funcional, se agregó la funcionalidad para operar un parámetro en particular: límite de potencia activa (Fig. 11).

Fig. 11: Dashboard mostrando el sistema de control del parámetro.

En la plataforma fue posible observar la evolución diaria del sistema interoperable (Fig. 12), y controlar el límite de potencia activa del inversor.

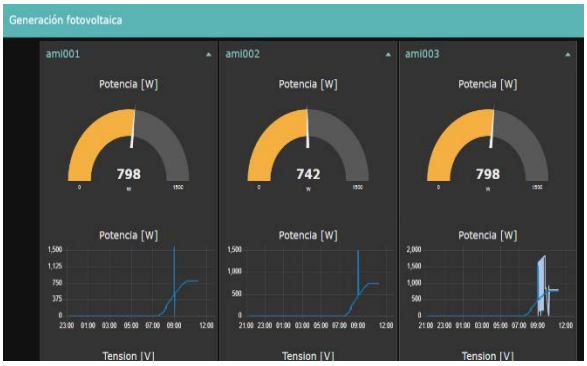

Fig. 12: Dashboard mostrando los datos de control del panel, AMI001 corresponde a un equipo de una versión previa basada en ATMega328 mientras que el AMI003 al sistema basado en ESP32.

Se obtuvieron también las curvas correspondientes al sistema bajo control con la limitación de potencia, que son almacenadas en la plataforma AWS (Fig. 13).

Fig. 13: Curva de día de operación del inversor con recorte de Potencia Activa.

VII. Resultados y conclusiones

La implementación del sistema WiFi-LoRa, permite una aproximación mucho más sencilla y nos permite acceder y controlar el inversor. La capacidad del sistema de configurar parámetros de operación del inversor, permite la implementación de estrategias de control en redes inteligentes. La solución implementada es de bajo costo (U$S 50) para el sistema ESP32-ESP32 mientras que el costo es significativamente mayor para el par ESP32-Raspberry (U$S200). Sin embargo, este último permite mayor flexibilidad y capacidad de procesamiento. Ambos sistemas al usar la misma máquina de estados y protocolo propio, pueden convivir en la red e intercambiar información, permitiendo generar un ecosistema flexible, intercambiable y reprogramable de fácil uso para sistemas de interoperabilidad. En futuros trabajos se implementará el sistema ya diseñado con herramientas de Machine Learning para predicción de demanda y oferta de energía.

Reconocimientos

Se agradece a la subgerencia de Energía y movilidad de INTI-Miguelete por el apoyo.

Referencias

[1] Politi, M., Niño, J. A., Gulfo, M. & Laiz, H. , “Equipo de Interoperabilidad para inverosres fotovoltaicos” En Actas del Congreso Argentino de Sistemas Embebidos (CASE 2021), 2021, pp. 175–177. ISBN 978-987-46297-8-4

[2] LoRa Alliance (2020, Junio), [Online]. Disponible: https://lora-alliance.org/about-lorawan

[3] Modbus Messaging on TCP/IP (2006). *MODBUS Messaging on TCP/IP Implementation Guide V1.0b*. Modbus Organization

[4] Switcher, Simple. “LM2576/LM2576HV Series SIMPLE SWITCHER ® 3A Step-Down Voltage Regulator.” (1999).

[5] “*Chip W5100*” (©2011 WIZnet Co., All Rights Reserved), [Online]. Disponible: https://www.wiznet.io/product-item/w5100/.

[6] HelTec Automation. “*Heltec_ESP32*” (2019), [Online]. Disponible: https://github.com/HelTecAutomation/Heltec_ESP32

[7] Sandeep Mistry. “arduino-LoRa” (2016), [Online]. Disponible: https://github.com/sandeepmistry/arduino-LoRa

[8] Pkoerboer. “*SMA-Reader*” (2020), [Online]. Disponible: https://github.com/pkoerber/SMA-SunnyBoy-Reader.

[9] Martyn Wheeler, “u-lora” (2021), [Online]. Disponible: https://github.com/martynwheeler/u-lora.

[10] SMA SunnyBoy FV On-Grid Inverter SB1.5-2.0-2.5 Documentación. [Online]. Disponible: https://www.sma.de/es/productos/inversor-fotovoltaico/sunny-boy-15-20-25

[11] Herramienta de programación de flujo NODE-RED [Online]. Disponible: https://nodered.org/

[12] Software abierto de para programación de broker MQTT[Online]. Disponible: https://mosquitto.org/