MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIONES

TRABAJO FIN DE MÁSTER

IMPLEMENTACIÓN DE UN SNIFFER DE PRIME CON BITSCOPE

Autor: Javier de la Paz Garcillán

Director: Javier Matanza Domingo

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Implementación de un sniffer de PRIME con BitScope

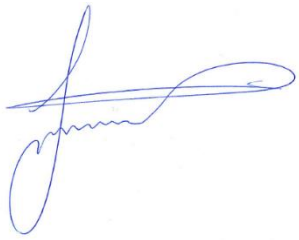
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2018/19 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.



Fdo.: Javier de la Paz Garcillán Fecha: 14 / 06 / 2019

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Javier Matanza Domingo Fecha: 14 / 06 / 2019

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Javier de la Paz Garcillán DECLARA ser el titular de los derechos de propiedad intelectual de la obra: Implementación de un sniffer de PRIME con BitScope, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente).*

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.
d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

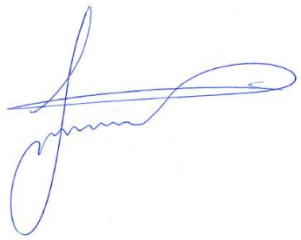
6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 14 de junio de 2019

ACEPTA



Fdo. Javier de la Paz Garcillán

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIONES

TRABAJO FIN DE MÁSTER

IMPLEMENTACIÓN DE UN SNIFFER DE PRIME CON BITSCOPE

Autor: Javier de la Paz Garcillán

Director: Javier Matanza Domingo

Madrid

Agradecimientos

Quiero seguir aprovechando esta sección, y al igual que hice en mi trabajo fin de grado, agradecer a todas aquellas personas que han sido tan importantes para mí en estos dos años de máster.

Siempre empezar por los más importantes, mis padres. Gracias por todo el esfuerzo que hacéis para ayudarme a alcanzar el mejor futuro posible en mi vida, por preocuparos tanto por mí, y ayudarme a que todo en mi vida sea más fácil. Valoro mucho todo lo que hacéis por mí. Prometo devolvéroslo como he hecho hasta ahora, trabajando muy duro e intentando que os sintáis muy orgullosos de mí.

Agradecer a ICAI la oportunidad de estudiar donde yo siempre quise, en la mejor universidad de ingeniería de España, inculcándome un lema que siempre podré en valor en cada ámbito de mi vida “El valor de la excelencia”.

A todos mis compañeros, desde los que empezaron conmigo en primero de grado, hasta los que acaban conmigo ahora segundo de máster, y todos aquellos que he conocido durante estos seis inolvidables años en esta universidad … muchas gracias. Sé que solo, este camino hubiese sido mucho más difícil, y gracias a la ayuda de todos, hemos conseguido superar todas las dificultades (que no han sido pocas) que nos hemos ido encontrando. Me alegra decir que a muchos de vosotros ya os puedo llamar amigos.

A todos los profesores de ICAI, de cada uno de vosotros me llevo algo que me hará mejor no solo como profesional, si no como persona. Y en especial, agradecer un año más a mi director de TFM Javier Matanza. Por haber vuelto a darme la oportunidad de desarrollar un proyecto bajo tu dirección, confiar en mí y ayudarme siempre.

Por último, quiero aprovechar este apartado para despedirme de una etapa de mi vida, que por seguro echaré mucho de menos. Estoy seguro que estos 6 años en lo que tantas cosas han pasado han merecido la pena, no los voy a olvidar nunca.

A todos.

Gracias.

IMPLEMENTACIÓN DE UN SNIFFER DE PRIME CON BITSCOPE

Autor: de la Paz Garcillán, Javier.
Director: Matanza Domingo, Javier
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Este proyecto busca ser una solución barata y open-source de un receptor PRIME. Se busca además ampliar la funcionalidad de las soluciones que hay actualmente en el mercado, obteniendo la salida de cada bloque del receptor, de modo que además pueda ser utilizado con fines docentes y académicos. Será capaz de hacer de sniffer en un canal de comunicación, detectar una señal PRIME, muestrearla, almacenarla, pasar por el receptor e interpretarla.

Palabras clave: PRIME, Smart Grids, BitScope, PLC, sniffer

1. Introducción

PRIME (PoweRline Intelligent Metering Evolution) es una especificación para la comunicación de la línea eléctrica de banda estrecha. La comunicación por línea eléctrica utiliza líneas eléctricas como medios de transmisión. Es un estándar maduro, consolidado y aceptado en todo el mundo para las comunicaciones por línea eléctrica basadas en OFDM, con un enfoque en la prestación de servicios de redes inteligentes a través de redes de distribución eléctrica. [2]

Actualmente nos encontramos ante una situación en la cual las soluciones PRIME existentes no son open-source, y además las que se pueden adquirir en el mercado tienen un alto precio. Por otro lado, son softwares limitados en funcionalidad, ya que tan solo nos proporcionan el mensaje interpretado a la salida del sistema (o en su defecto, los bits a la salida del mismo), por lo que no permite tener acceso a la salida de cada uno de los bloques de la cadena del receptor, si no tan solo al resultado final.

Es por esto, que, a través de este proyecto, se busca dar con una solución que permita resolver todas estas cuestiones.

2. Definición del proyecto

En este proyecto se implanta una solución que permite escuchar en un canal de transmisión de mensajes PRIME, e interceptar la transmisión, leerla, y poder obtener los bits a la salida de cada uno de los bloques de la cadena del receptor que se desee, hasta la interpretación final del mensaje.

Es una alternativa a algunos sistemas ya existentes, pero de una manera open-source, y con una alternativa hardware mucho más económica como es el procesador digital de señal y muestreador BitScope.

De este modo, se pretende obtener un sistema que permita obtener mucha más información a la que hasta ahora no se tenía acceso con ninguna otra solución en el mercado, y con un precio mucho más bajo al de los sistemas ya existentes.

Además, el hecho de conseguir resultados intermedios a la salida de los diferentes bloques que componen el receptor permite que este sistema pueda ser muy útil en fines como pueden ser docentes o académicos. Se podrá observar cómo funciona paso a paso un sistema de comunicación basado en PRIME.

3. Descripción del modelo/sistema/herramienta

El sistema se compondrá de tres bloques principales como se muestra en la Ilustración 1 del diagrama de comunicación del sistema (en un caso real, la placa de desarrollo PRIME sería un canal de comunicación PRIME).

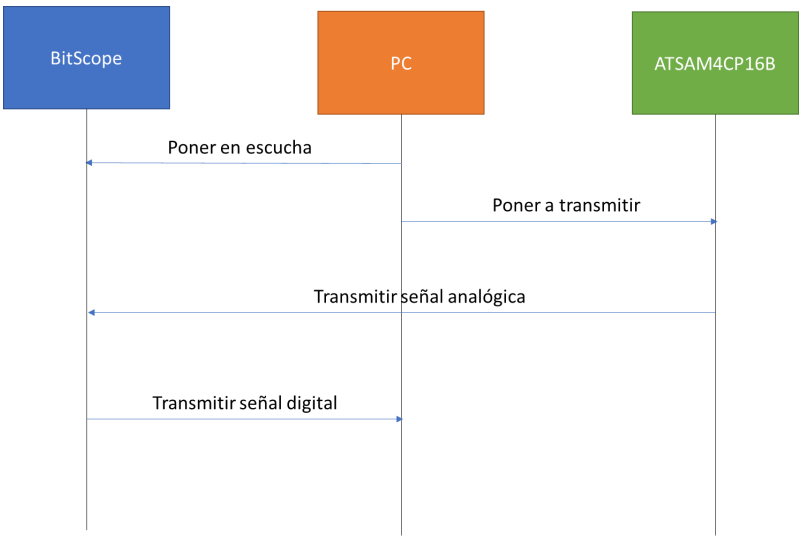


Ilustración 1. Diagrama de comunicación del sistema

- ATSAM4CP16B Evaluation Kit: placa de desarrollo PRIME. Genera un mensaje que transmite con la especificación PRIME. La señal se genera mediante un software llamado Atmel PLC PHY Tester Tool. Dicha señal se compondrá de preámbulo, cabecera y payload. Sirve como simulador de un canal de transmisión PRIME.
    - Filtro ATPLCOUP001v1: filtro hardware paso banda en la banda de frecuencias de PRIME (41 -89 kHz).
    - La señal será medida a la salida del filtro con una sonda de osciloscopio conectada al BitScope a través de un conector BNC.
- BitScope: muestreador programable. Gracias al trigger, empieza a muestrear (a 250 kHz) a partir de un umbral de tensión establecido (trigger). Captura el tamaño máximo de mensaje y almacena esas muestras en un buffer. Una vez tiene todas las muestras, se transmiten a través de un cable microUSB-USB, a un ordenador.
- PC: en el ordenador se estarán ejecutando dos programas Python. El primero capturará las muestras de la señal almacenada en el buffer y las guardará en un fichero de texto. El segundo leerá dicho fichero de texto y pasará las muestras por cada uno de los bloques que componen el receptor PRIME. Por último, se traducirá el mensaje a ASCII y se guardará dicho output en un fichero de texto como se muestra en Ilustración 2.

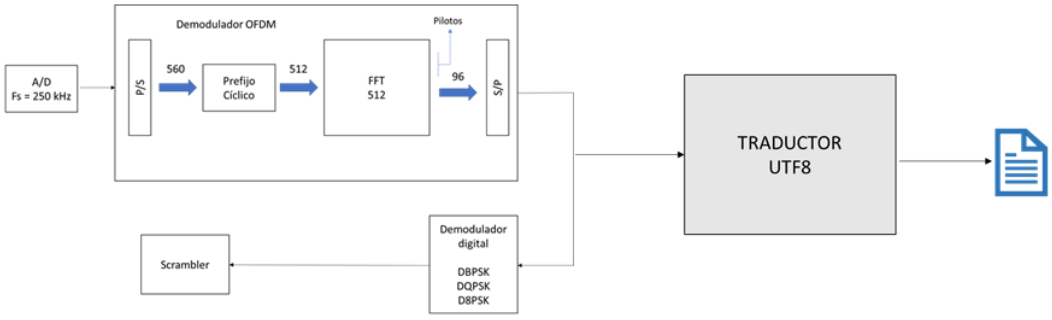


Ilustración 2. Esquema del sistema

4. Resultados

Como resultado final de este proyecto se ha logrado un sistema capaz de hacer la recepción y traducción a caracteres ASCII de mensajes PRIME.

Se han logrado los objetivos establecidos inicialmente, con las siguientes tecnologías:

- Detectar y leer una señal PRIME
  - Hacer un muestreo digital de la señal (A/D) → BITSCOPE y BITLIB ✓
  - Guardar la señal. → TEXT FILE ✓
  - Tratar la señal. → PYTHON ✓
- Hacer la función de receptor PRIME
  - Estimar el canal. → PYTHON ✓
  - Devolver la salida de cada uno de los bloques del receptor. → PYTHON y NUMPY – Almacena las imágenes resultantes de la salida de cada bloque en una carpeta de imágenes según se van generando. ✓
  - Representación gráfica de los bits a su paso por los diferentes bloques. → PYTHON y MATPLOTLIB – Almacena las imágenes resultantes de la salida de cada bloque en una carpeta de imágenes según se van generando. ✓
  - Interpretar (traducir) el mensaje detectado. → PYTHON ✓
  - Almacenar texto ASCII de salida → FILE TEXT ✓

El sistema final resultante sería el mostrado en la Ilustración 3:

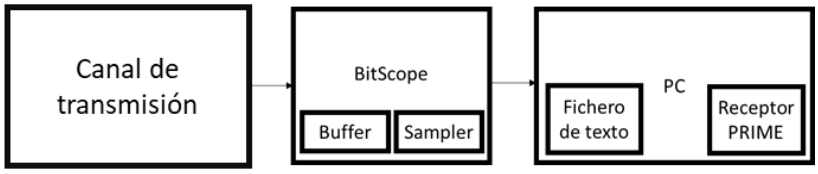


Ilustración 3. Bloques del sistema real

5. Conclusiones

Se puede concluir que se ha conseguido la implementación de un sniffer PRIME con BitScope, como dicta el título del proyecto.

Además, se ha logrado un sistema barato (178€), open source, que proporciona información de los bits a lo largo de todo el sistema, que genera gráficas a la salida de cada bloque y sencilla de implementar (tanto en cuanto a complejidad electrónica se refiere).

Sin embargo, nos hemos topado con una limitación técnica con respecto al muestreador. El BitScope, entre disparo y disparo, desde volver a hacer las fases de Trace y Acquire, lo que supone un tiempo aproximado de 100 ms entre capturas.

Esto limita la función de "sniffer" deseada, ya no es capaz de hacer una escucha permanente, ya que durante un periodo se encuentra ocupado.

Se propone como trabajos futuros, utilizar varios BitScope para ir haciendo capturas secuenciales de modo que sea capaz el sistema de muestrear más cantidad de símbolos OFDM. De este modo se acabaría con la limitación técnica aparecida, logrando con esto una sniffer con todas las funcionalidades del mismo.

Además, este proyecto podría dar lugar a otro tales como:

- Utilización de esta herramienta para el análisis de la seguridad de las redes PLC.
- Análisis del rendimiento de diferentes decodificadores convolucionales.
- Análisis del rendimiento de diferentes estrategias de ecualización para sistemas OFDM.

6. Referencias

[1] Recommendation ITU-T G.9904 - Narrowband orthogonal frequency division multiplexing power line communication transceivers for PRIME networks. 10/2012

[2] PRIME (PLC). En Wikipedia. Recuperado el 24 de marzo de 2019 de https://en.wikipedia.org/wiki/PRIME_(PLC)

PRIME SNIFFER IMPLEMENTATION WITH BITSCOPE

Author: de la Paz Garcillán, Javier.
Supervisor: Matanza Domingo, Javier.
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This project seeks to be an inexpensive and open-source solution for a PRIME receiver. It also seeks to expand the functionality of solutions currently on the market, obtaining the output of each block of the receiver, so that it can also be used for teaching and academic purposes. It will be able to sniffer in a communication channel, detect a PRIME signal, sample it, store it, pass through the receiver and interpret it.

Keywords: PRIME, Smart Grids, BitScope, PLC, sniffer

1. Introduction

PRIME (PoweRline Intelligent Metering Evolution) is a specification for narrow band electrical line communication. The communication by electric line uses electrical lines as means of transmission. It is a mature, consolidated and accepted worldwide standard for OFDM-based power line communications, with a focus on the provision of smart grid services through power distribution networks. [2]

We are currently facing a situation where existing PRIME solutions are not open-source, and those that can be purchased on the market are highly priced. On the other hand, they are softwares limited in functionality, since they only provide us with the interpreted message at the output of the system (or failing, the bits at the output of the it), so it does not allow us to have access to the output of each one of the blocks of the chain of the receiver, only to the final result.

It is for this reason that, through this project, the aim is to find a solution that will allow all these issues to be resolved.

2. Project definition

This project implements a solution that allows listening to a PRIME message transmission channel, intercepting the transmission, reading it, and being able to obtain the bits at the output of each of the blocks of the desired receiver chain, until the final interpretation of the message.

It is an alternative to some existing systems, but in an open-source way, and with a much cheaper hardware alternative such as the BitScope, a digital signal processor and sampler.

In this way, the aim is to obtain a system that allows much more information to be obtained than until now was not available with any other solution on the market, and at a much lower price than the existing systems.

In addition, the fact of getting intermediate results at the output of the different blocks that make up the receiver allows this system to be very useful in terms such as teachers or academics. You can see how a communication system based on PRIME works step by step.

3. System description

The system will consist of three main blocks as shown in Figure 1 of the system communication diagram (in a real case, the PRIME development board would be a PRIME communication channel).

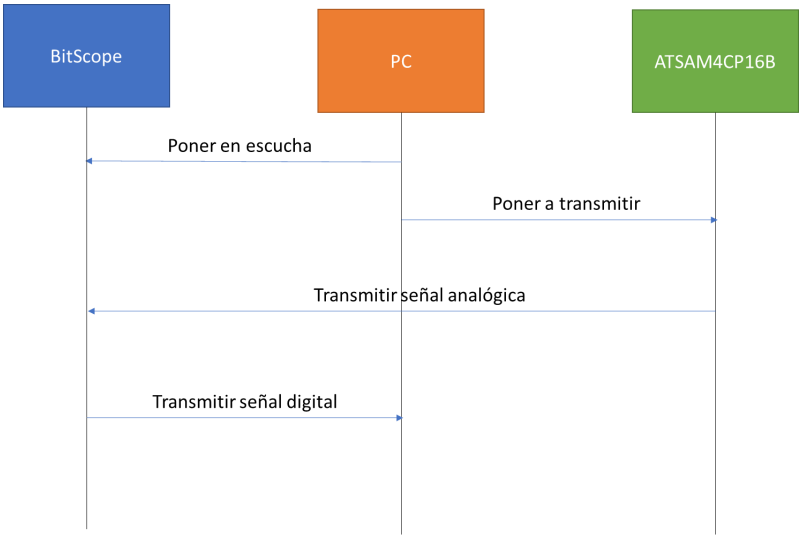


Illustration 1. System communication diagram

- ATSAM4CP16B Evaluation Kit: PRIME development board. It generates a message that transmits with the PRIME specification. The signal is generated by software called Atmel PLC PHY Tester Tool. This signal will consist of preamble, header and payload. It serves as a simulator of a PRIME transmission channel.
    - Filter ATPLCOUP001v1: passband hardware filter in the PRIME frequency band (41 -89 kHz).
    - The signal will be measured at the filter output with an oscilloscope sounding line connected to the BitScope through a BNC connector.

- BitScope: programmable sampler. Thanks to the trigger, it starts sampling (at 250 kHz) from a set voltage threshold (trigger). It captures the maximum message size and stores those samples in a buffer. Once you have all the samples, they are transmitted through a microUSB-USB cable, to a computer.

- PC: two Python programs will be running on the computer. The first one will capture the samples of the signal stored in the buffer and store them in a text file. The second one will read said text file and pass the samples through each of the blocks that make up the PRIME receiver. Finally, the message will be translated into ASCII and this output will be saved in a text file as shown in Figure 2.

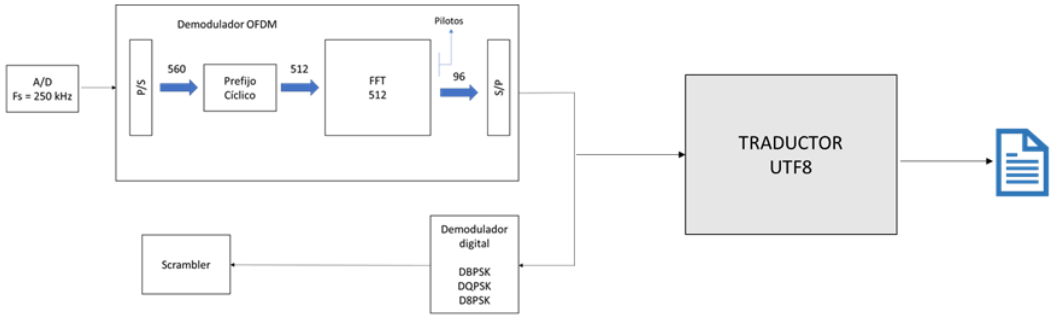


Illustration 2. System schematic

4. Results

The final result of this project is a system capable of receiving and translating PRIME messages into ASCII characters.

The initially established objectives have been achieved, with the following technologies:

- Detect and read a PRIME signal
  - Digital sampling of the signal (A/D) →BITSCOPE and BITLIB ✓
  - Save the signal. → TEXT FILE ✓
  - Processing the signal. → PYTHON ✓

- Make receiver PRIME function
  - Estimate the channel. → PYTHON ✓
  - Return the output of each of the receiver blocks. → PYTHON and NUMPY - Stores the images resulting from the output of each block in a folder of images as they are generated. ✓
  - Graphic representation of the bits passing through the different blocks. → PYTHON and MATPLOTLIB - Stores the images resulting from the output of each block in a folder of images as they are generated. ✓
  - Interpreting (translating) the detected message. → PYTHON ✓
  - Store ASCII output text →FILE TEXT ✓

The resulting final system would be as shown in Illustration 3:

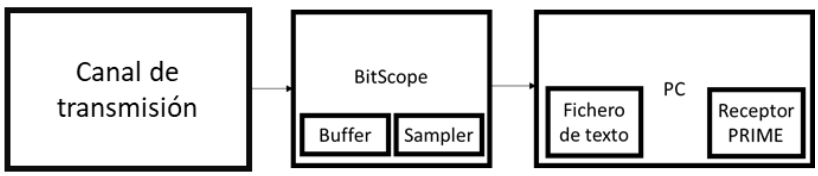


Illustration 3. Blocks of the real system

5. Conclusions

It can be concluded that the implementation of a PRIME sniffer with BitScope has been achieved, as dictated by the title of the project.

In addition, a cheap system (€ 178), open source, has been achieved, which provides bit information throughout the system, which generates graphics at the output of each block and is simple to implement (both in terms of electronic complexity it means).

However, we have encountered a technical limitation with respect to the sampler. The BitScope, between trigger and trigger, has to retrace the Trace and Acquire phases, which means an approximate time of 100 ms between captures.

This limits the desired "sniffer" function, it is no longer able to make a permanent listening, since during a period it is busy.

It is proposed as future works, to use several BitScope to make sequential captures so that the system is able to sample more OFDM symbols. This would finish with the the technical limitation that appeared, thus achieving a sniffer with all the functionalities of it.

In addition, this project could lead to anothers such as:

- Use of this tool to analyse the security of PLC networks.
- Analysis of the performance of different convolutional decoders.
- Analysis of the performance of different equalization strategies for OFDM systems.

6. References

[1] Recommendation ITU-T G.9904 - Narrowband orthogonal frequency division multiplexing power line communication transceivers for PRIME networks. 10/2012

[2] PRIME (PLC). In Wikipedia. Recovered the 24 of March, 2019 from https://en.wikipedia.org/wiki/PRIME_(PLC)

Índice de la memoria

Índice de Ilustraciones

Índice de tablas

Índice de Acrónimos

A/D – Analógico / Digital

API - Application Programming Interface

BPF – Band Pass Filter

D8PSK - Differential Eight-Phase Shift Keying

DBPSK - Differential Binary Phase Shift Keying

DFT - Discrete Fourier Transform

DQPSK - Differential Quaternary Phase Shift Keying

FEC - Forward Error Correction

FFT - Fast Fourier Transform

IFFT - Inverse Fast Fourier Transform

MSB - Most Significant Bit

OFDM - Orthogonal Frequency Division Multiplexing

PDS – Procesado Digital de Señal

PLC - Power Line Communications

PRIME - PoweRline Intelligent Metering Evolution

OS - Operating System

Capítulo 1. INTRODUCCIÓN

PRIME (PoweRline Intelligent Metering Evolution) es una especificación para la comunicación de la línea eléctrica de banda estrecha. La comunicación por línea eléctrica utiliza líneas eléctricas como medios de transmisión. ITU G.9904

PRIME fue desarrollado en 2007. Las primeras publicaciones se remontan a 2008. En 2009, se demostró la interoperabilidad de múltiples proveedores y se lanzó la PRIME Alliance. PRIME Alliance tiene pruebas de interoperabilidad implementadas, que se llevan a cabo en múltiples laboratorios de prueba acreditados. Actualmente, las pruebas han pasado más de 40 productos. [6]

Es un estándar maduro, consolidado y aceptado en todo el mundo para las comunicaciones por línea eléctrica basadas en OFDM, con un enfoque en la prestación de servicios de redes inteligentes a través de redes de distribución eléctrica.

Es utilizada para aplicaciones de redes inteligentes como la medición inteligente, iluminación y automatización industrial, automatización del hogar, iluminación de calles, energía solar y estaciones de carga.

Actualmente nos encontramos ante una situación en la cual las soluciones PRIME existentes no son open-source, y además las que se pueden adquirir en el mercado tienen un alto precio.

Algunos ejemplos de ello son:

- Contadores inteligentes.
- Productos de monitorización y medida con capacidades de comunicaciones que utilizan PLC basado en PRIME

Por otro lado, son softwares limitados en funcionalidad, ya que tan solo nos proporcionan el mensaje interpretado a la salida del sistema (o en su defecto, los bits a la salida del mismo),

por lo que no permite tener acceso a la salida de cada uno de los bloques de la cadena del receptor, si no tan solo al resultado final.

Es por esto, que, a través de este proyecto, se busca dar con una solución que permita resolver todas estas cuestiones.

De este modo, la principal motivación de este proyecto es implantar una solución que permita interceptar una señal PRIME (a modo de sniffer), leerla, y poder obtener los bits a la salida de cada uno de los bloques de la cadena del receptor que se desee, hasta la interpretación final del mensaje.

La idea del proyecto es lograr una solución capaz de, una vez conectada a un canal de transmisión de mensajes bajo la especificación PRIME, sea capaz de:

- Leer una señal PRIME
- Muestrear la señal
- Hacer la función de receptor PRIME

Además, el aspecto docente o académico es doblemente importante debido a que: permitirá mostrar a los alumnos las aplicaciones prácticas a conceptos teóricos aprendidos; y podrá ser utilizado en la asignatura de Sistemas de Comunicación I como ejemplo real de cómo funciona el sistema de comunicación.

El proyecto ha sido planificado con una duración total de 10 meses, de los cuales 3 de ellos han sido empleados en investigación acerca de la materia y los medios a utilizar, 1 para las señales PRIME, y los 6 restantes para la programación del Receptor.

Los recursos empleados para este proyecto son los que se enumeran a continuación:

Hardware:

- BitScope
- Sonda de osciloscopio
- Cable USB-microUSB

- Cable USB-miniUSB
- Dual BNC Adapter
- Filtro ATPLCOUP001v1
- ATSAM4CP16B Evaluation Kit
- PC

Software:

- Python 2.7
- Bitlib
- Atmel PLC PHY Tester Tool
- USB Drivers

Además, y por último, sería óptimo que este proyecto, pudiera servir en un futuro, además de para fines comentados como objetivos, para otros proyectos relacionados con análisis de rendimiento y análisis de seguridad de redes PLC.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

2.1 PROCESAMIENTO DIGITAL DE SEÑAL

Manipulación matemática de una señal de información para modificarla o mejorarla. Se caracteriza por la representación en el dominio discreto (de tiempo o frecuencia) de señales por medio de una secuencia de números o símbolos y el procesado de esas señales.

Se utiliza para tratar y modificar una señal, de modo que se pueda trabajar con ella, y poder ser analizada.

Para este fin, se utilizan sistemas basados en un procesador o microprocesador con instrucciones, hardware y software optimizados para aplicaciones que requieran operaciones numéricas de alta velocidad.

El propósito de procesar una señal puede ser disminuir el nivel de ruido o mejorar determinados matices.

La aplicación de la computación digital al procesamiento de señales permite muchas ventajas sobre el procesamiento analógico en gran número de aplicaciones, tales como la detección y corrección de errores en la transmisión y la compresión de datos. PDS es aplicable tanto a los datos de transmisión como a datos estáticos (almacenados). [8]

2.1.1 MUESTREO DIGITAL

Fase del proceso de digitalización de una señal [8]. Éste consiste en tomar muestras de una señal analógica a una frecuencia de muestreo constante (como se ve en la Ilustración 1). Se reduce una señal continua en el tiempo a señal discreta. [9]

- Una muestra es un valor o serie de valores en un punto en tiempo y/o espacio.

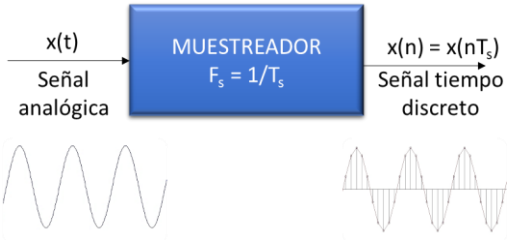


Ilustración 4. Muestreo digital

- Un muestreador es un subsistema que extrae muestras de una señal continua.

Para analizar y manipular digitalmente una señal analógica, se debe digitalizar con un convertidor de analógico a digital. El muestreo se realiza en dos etapas, discretización y cuantificación, como se muestra en la Ilustración 2.

- Discretización: la señal se divide en intervalos iguales de tiempo, y cada intervalo se representa mediante una única medida de amplitud.
- Cuantización: cada medida de amplitud se aproxima por un valor de un conjunto finito. Redondear números reales a enteros es un ejemplo.

El teorema de muestreo de Nyquist-Shannon establece que una señal puede reconstruirse exactamente a partir de sus muestras si la frecuencia de muestreo es mayor que el doble del componente de frecuencia más alta en la señal.

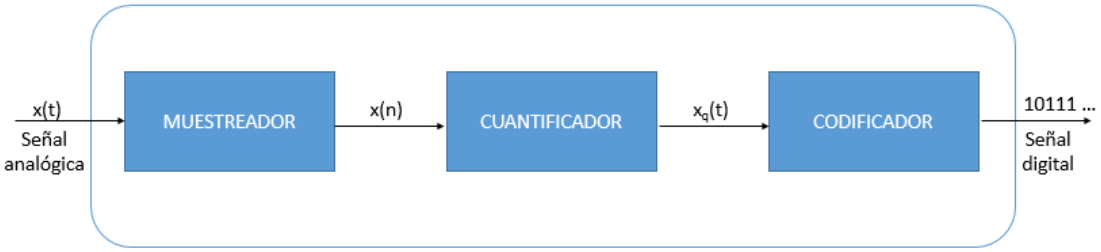


Ilustración 5. Diagrama de bloques de Conversor A/D

Es útil en la digitalización de señales.

2.1.2 FFT

Un algoritmo de transformada rápida de Fourier (FFT), calcula la transformada discreta de Fourier (DFT) de una secuencia, o su inversa (IFFT).

El análisis de Fourier convierte una señal de su dominio original (a menudo tiempo) en una representación en el dominio de la frecuencia y viceversa (por ejemplo, permite pasar de la representación de la izquierda de la Ilustración 3, a la de la derecha con una FFT y viceversa con una IFFT). Una FFT calcula rápidamente tales transformaciones por factorización de la matriz DFT en un producto de escasos factores. [10]

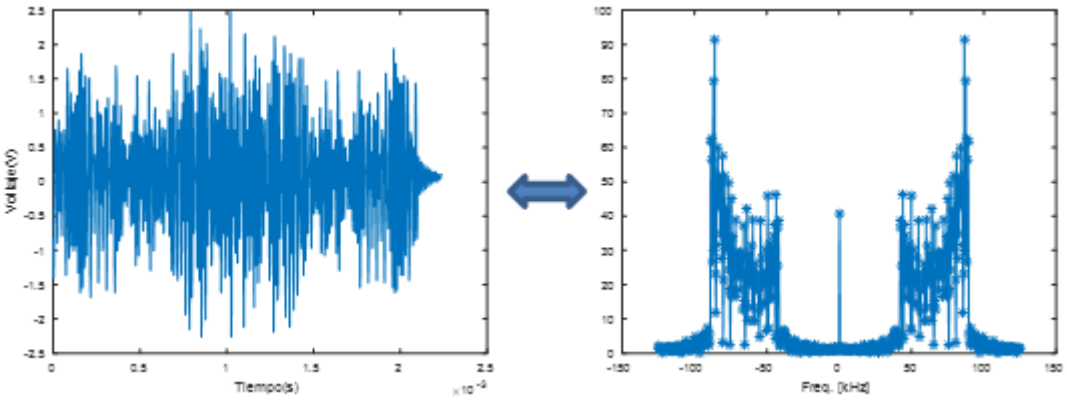


Ilustración 6. Señal en tiempo y frecuencia

2.2 MODULACIÓN

Técnicas utilizada para transportar información sobre una onda portadora. Permite un mejor aprovechamiento del canal de comunicación lo que posibilita transmitir más información de forma simultánea además de mejorar la resistencia contra ruidos e interferencias.

La modulación consiste en hacer que un parámetro de la onda portadora cambie de valor de acuerdo con las variaciones de la señal moduladora, que es la información que se quiere transmitir. [11]

2.2.1 MODULACIÓN DIGITAL

Proceso mediante el cual se trasforman los símbolos digitales en forma de onda adecuadas para la transmisión sobre un canal de comunicación. [11]

La modulación digital tiene por objeto transmitir una señal digital a través de un canal limitado en banda.

2.2.1.1 Modulación por desplazamiento diferencial de fase (DPSK)

Es una forma de modulación digital, donde la información binaria de la entrada está compuesta en la diferencia entre las fases de dos elementos sucesivos de señalización, y no en la fase absoluta.

En los sistemas DPSK, el flujo digital de entrada es codificado de forma diferencial y luego es modulado mediante PSK. [11]

2.2.2 OFDM

La multiplexación por división de frecuencias ortogonales es una técnica de transmisión que consiste en la multiplexación de un conjunto de ondas portadoras de diferentes frecuencias, donde cada una transporta información, la cual es modulada en QAM o en PSK (o DPSK). [12]

2.3 PYTHON

Es un lenguaje de programación de alto nivel. Es un lenguaje interpretado, usa tipado dinámico, es multiplataforma y con una sintaxis que favorece el código legible.

Además, se trata de un lenguaje de programación multiparadigma: soporta orientación a objetos, programación imperativa y funcional. [13]

2.4 CODIFICACIÓN UTF-8

UTF-8 es una codificación de caracteres de ancho variable capaz de codificar los 1112064 puntos de código válidos en Unicode utilizando de uno a cuatro bytes de 8 bits.

La codificación está definida por el estándar Unicode.

UTF-8 divide los caracteres Unicode en varios grupos, en función del número de bytes necesarios para codificarlos. El número de bytes depende exclusivamente del código de carácter asignado por Unicode y del número de bytes necesario para representarlo. [14]

En la Ilustración 4, se muestran las equivalencias necesarias para la traducción UTF8-ASCII en este proyecto (letras y números).

0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	DEL 007F

Ilustración 7. Diccionario UTF-8

2.5 ATMEL PLC PHY TESTER TOOL

Es un software, que ha sido desarrollado para permitir probar las características básicas de las capas físicas de los productos de Microchip PLC.

Esta herramienta es capaz de configurar las diferentes capas físicas de cada producto con parámetros básicos tales como esquemas de modulación, potencia de transmisión, velocidad en baudios, etc. para hacer posible el intercambio de mensajes PLC básicos.

La herramienta está estructurada de manera similar a un asistente, donde cada paso permite configurar algunos parámetros relacionados con un aspecto del test, lo cual permite al configurar la prueba que se realizará. [18]

2.6 BITSCOPE

Son osciloscopios programables basados en PC, analizadores lógicos, analizadores de espectro, generadores de formas de onda y sistemas de adquisición de datos para Windows, Mac OS X, Linux y Raspberry Pi. Ofrecen soluciones integrales en pruebas, medidas, monitoreo y control para ingenieros en la industria, educación, I+D y servicio.

Económicos y basados en la BitScope Virtual Machine. La BitScope Virtual Machine permite actualizar y personalizar dinámicamente el software.

Son compatibles con el software más actualizado.

Son capaces de capturar señales digitales y analógicas simultáneamente. Tienen un ancho de banda de 100 MHz y hasta 40 MS/s de captura digital. Incorporan generadores de onda y reloj, decodificadores de protocolo, I/O digitales y son capaces de conducir circuitos externos de baja potencia.

Los BitScopes son completamente programables por el usuario para poder ser utilizados en aplicaciones altamente personalizadas o incluso integrados en productos de terceros y sistemas de software en OEM. Pueden programarse directamente a nivel de máquina virtual utilizando guías de usuario de publicación o mediante la BitScope Library, que pone a disposición todas las funciones de adquisición y generación de datos de señal mixta de BitScope mediante una API de llamada de función fácil de usar. [16]

2.6.1 BITLIB

Librería de programación de BitScope. Es una librería potente para poder crear una aplicación BitScope personalizada.

Aprovecha la potente mixed mode waveform y la captura de datos lógicos de BitScope y admite diferentes lenguajes de programación (C++, Python) y herramientas de análisis numérico.

También admite la generación de ondas arbitrarias con capacidad de carga de datos completa y enlaces de comunicaciones en red para aplicaciones de adquisición de datos remotos desde un solo PC. [19]

BitScope implementa una arquitectura de VM que se programa a través de scripts de comandos cortos que operan en registros. [16]

2.7 ATSAM4CP16B EVALUATION KIT

Es un kit de evaluación del dispositivo SAM4CP16B para comunicación por línea de alimentación de Atmel.

Tiene una arquitectura flexible, compuesta de aceleradores de hardware y coprocesadores, y el conjunto de periféricos incluye un transceptor de comunicación de línea de alimentación certificado PRIME.

Incluye un stack de comunicaciones PRIME completamente operativa. [4]

2.7.1 SAM4CP16B

Es un microcontrolador Flash basado en un procesador dual ARM Cortex-M4 RISC de 32 bits y alto rendimiento que incorpora un transceptor PRIME para la comunicación por línea de alimentación que implementa los perfiles PRIME CENELEC-A, FCC y ARIB.

Esta placa de desarrollo proporciona una plataforma con todas las funciones para desarrollar un sistema de comunicaciones completo a través de la tecnología Powerline Communication (PLC). [4]

2.7.2 ATPLCOUP001v1

Es una placa de acoplamiento PLC de aislamiento de tensión de red especialmente optimizada para comunicarse en la banda CENELEC-A (la marcada con una "A" en la Ilustración 5), especialmente en la banda PRIME de 41 a 89 kHz (canal 1 de PRIME).

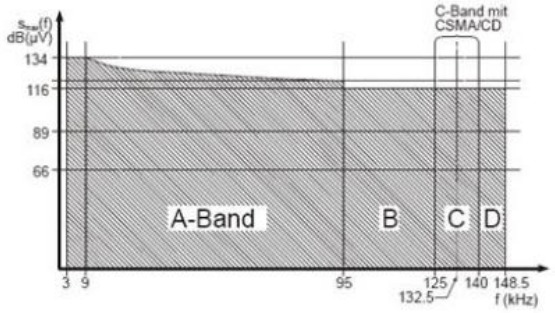


Ilustración 8. Bandas CENELEC

Es un diseño de aislamiento galvánico que proporciona un acoplamiento PLC optimizado en términos de nivel de señal de salida en un amplio rango de valores de impedancia de carga. Admite la banda de frecuencia entre 41 y 89 kHz de la banda CENELEC-A. Está compuesto de una sola rama de transmisión cuya etapa de filtrado tiene una respuesta de paso de banda plana con impedancias de campo típicas. [5]

Nombre	*Banda de Frecuencia*	*Rama*	*Aislamiento Eléctrico*	*Canal PRIME*	*Banda CENELEC*	*ARIB*	*FCC*
ATPLCOUP001	41-89 kHz	Única	SI	1	A	-	-

Tabla 1. Características filtro ATPLCOUP001V1

2.8 OTROS

Además de las tecnologías, softwares y hardware descritas arriba, en este proyecto se ha hecho uso de otras, no menos importantes, pero que se ha considerado que no requieren de una descripción explicativa de los mismos.

Hardware:

- Sonda de osciloscopio
- Jumpers
- Cable USB-microUSB
- Cable USB-miniUSB
- Dual BNC Adapter
- PC / Raspberry-pi

Software:

- USB Drivers
- Sublime Text
- MATLAB

Otras tecnologías:

- Buffer
- Puntero

Capítulo 3. ESTADO DE LA CUESTIÓN

En el mercado actual, podemos encontrar algunas soluciones que tienen una parte de la funcionalidad que se busca obtener en este proyecto, pero con diferencias apreciables.

Algunos ejemplos de esto son algunos contadores digitales de PRIME como pueden ser los siguientes:

ZIV:

Tiene equipos y soluciones, interoperables y desarrolladas sobre estándares internacionales abiertos como PRIME. [23]

- 5CTM- Contador Monofásico Inteligente | PLC PRIME
- 5CTD- Contador Trifásico Inteligente | PLC PRIME
    - Comunicación bidireccional sobre PLC (modelo PRIME y G3)
    - Lectura de datos, cambios de configuración

5CTM y 5CTD proporcionan soluciones de lectura de medidores automatizadas y robustas para las empresas de distribución.

Las capacidades de comunicación local y remota permiten la lectura de datos, los cambios de configuración, la sincronización de fecha y el funcionamiento del interruptor incorporado.

Comunicación bidireccional mediante tecnología PRIME. Los medidores inteligentes 5CTM y 5CTD integran un nodo de servicio PLC PRIME que se identifica automáticamente en la red del PLC (plug & play).

Estos medidores implementan la tecnología propia de ZIV para los estándares abiertos PRIME.

ORMAZABAL

Utilizan datos de medidores y mediciones de bajo voltaje para construir un conjunto de aplicaciones de redes inteligentes que facilitan la gestión de activos.

Tienen productos de medida (MV-BPL), de medida (IDC, DIOM) y de monitorización (LV InsitE, Rogowski sensor). Y software: OGN, OGM y OGD, de cada uno de los tipos respectivamente.

Tienen concentradores de datos inteligentes certificados por PRIME que utilizan datos de medidores y mediciones de bajo voltaje para construir un conjunto de aplicaciones de redes inteligentes que facilitan la gestión de activos, la evaluación comparativa del rendimiento y, finalmente, permiten decisiones sobre la optimización de la red.

- IDC - es una solución de infraestructura que combina las capacidades de comunicaciones que utilizan el PLC basado en PRIME con un software integrado avanzado para proporcionar la plataforma de recopilación, informes y análisis para la implementación de infraestructura de medición avanzada (AMI). El DC ofrece recopilación de medidores, priorización y administración de órdenes de trabajo, y proporciona un amplio soporte de administración remota de dispositivos. [24]

MICROCHIP

También tiene kits de evaluación como el siguiente:

- ATPL230A - Módem de banda base de comunicaciones de línea eléctrica, compatible con la capa PHY de la especificación PRIME (Evolución de medición inteligente de línea eléctrica). Puede operar en bandas de transmisión seleccionables independientemente. Ha sido creado para ser incluido con un microchip MCU o MPU externo. Microchip proporciona una biblioteca de capas PRIME PHY que es utilizada por la MCU / MPU externa para tomar el control del dispositivo de capa ATPL230A PHY. [25]

Sin embargo, estas soluciones tienen los inconvenientes de no ser open-source y de tener un coste mucho más elevado que Bitscope (alternativa utilizada en este proyecto). Además, estas soluciones comerciales solo proporcionan los bits de salida, no hay acceso a los diferentes puntos de la cadena del receptor o transmisor (demodulador OFDM, demodulador digital, scrambler, etc), ni tan poco nos da la posibilidad de trabajar con la señal, analizarla o representarla en bloques intermedios. El trabajo de receptor PRIME es totalmente transparente y no permite observar como funciona. Esto, unido a sus altos precios, dificulta su uso en ámbitos académicos.

Con respecto a trabajos de investigación, no se ha encontrado ninguno que haya aportado los resultados que se quieren alcanzar en éste, ni ningún otro con un alcance u objetivo similar.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

Este proyecto busca implantar una solución que permita interceptar una señal PRIME (a modo de sniffer), leerla, y poder obtener los bits a la salida de cada uno de los bloques de la cadena del receptor que se desee, hasta la interpretación final del mensaje.

La principal motivación es encontrar una alternativa a algunos sistemas ya existentes en la recepción y traducción de señales PRIME, pero de una manera open-source, y con una alternativa hardware mucho más económico como es el procesador digital de señal BitScope.

Se intenta demostrar que se puede conseguir un sistema que sea económico, con la misma funcionalidad que los productos actualmente existentes en el mercado (incluso proporcionando mayor información), no compleja y de código abierto.

Además, el hecho de conseguir resultados intermedios a la salida de los diferentes bloques que componen el receptor permite que este sistema pueda ser muy útil en la otra principal motivación que sirve como justificación a este proyecto, y no es otra que para fines docentes o académicos.

El aspecto docente o académico es doblemente importante debido a que:

- Permitirá mostrar a alumnos las aplicaciones prácticas de conceptos teóricos aprendidos durante el grado y el máster de telecomunicación, de manera sencilla, práctica, visual y económica.
- Podrá ser utilizado en la asignatura de Sistemas de Comunicación I como ejemplo real de cómo funciona un sistema de comunicación. Funciona con una señal PRIME real, un muestro en tiempo real y una recepción y traducción del mensaje casi instantánea. Además, al ser de código abierto y el sistema no es de un alto precio, podrían trabajar, modificar y aprender con él.

4.2 OBJETIVOS

El objetivo del proyecto es lograr una solución capaz de, una vez conectada a un canal de transmisión de mensajes bajo la especificación PRIME, sea capaz de, a modo de sniffer:

- Detectar y leer una señal PRIME
    - Hacer un muestreo digital de la señal (A/D).
    - Almacenarla temporalmente en un buffer.
    - Guardar la señal.
    - Tratar la señal.
        - Eliminar muestras no pertenecientes a la señal
        - Separar preámbulo, cabecera y payload
- Hacer la función de receptor PRIME
    - Estimar el canal.
    - Devolver la salida de cada uno de los bloques del receptor.
        - Demodulador OFDM
        - Demodulador digital
            - DBPSK
            - DQPSK
            - D8PSK
        - Scrambler
    - Representación gráfica de los bits a su paso por los diferentes bloques.
    - Interpretar (traducir) el mensaje detectado.
    - Almacenar texto ASCII de salida

Para ello, como requisito inicial, tendremos que ser capaces de generar mensajes bajo la especificación PRIME, para simular el canal de transmisión de mensajes al que conectar el sniffer.

Además de esto, se requiere que la solución sea:

- Barata (en comparación con los sistemas del mercado).

- Open-source.
- Proporcione información de los bits a la entrada y salida de cada bloque del receptor PRIME.
- Proporcione información gráfica de cada etapa.
- Sencilla (que la herramienta desarrollada no suponga una complejidad electrónica excesiva).

4.3 METODOLOGÍA

En este apartado, tan solo se mencionará con que tecnología se abordará cada punto de los objetivos.

- Generación de mensajes PRIME → ATSAM4CP16B Evaluation Kit
- Comunicación entre PC y ATSAM4CP16B Evaluation Kit → Atmel PLC PHY Tester Tool
- Filtrado en banda de frecuencias → Filtro ATPLCOUP001v1
- Detección y conversión A/D de las señales PRIME entrantes → BitScope
- Almacenamiento temporal de muestras digitalizadas → Buffer de BitScope
- Comunicación con el BitScope → Python y Bitlib
- Almacenamiento de la señal PRIME para lectura → fichero de texto
- Receptor PRIME → Python

Se entrará en más detalle de cómo funciona cada parte, en el apartado 5 del documento, que es donde corresponde la descripción en profundidad del diseño.

4.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA

4.4.1 PLANIFICACIÓN

Con el propósito de organizar el proyecto, se ha intentado seguir un calendario de hitos y objetivos (que se muestra en el cronograma de la Ilustración 6 donde se muestran las diferentes tareas a llevar a cabo en el proyecto). No ha sido una guía estricta, pero si orientativa para una buena planificación y organización del proyecto.

Nombre de tarea	Duración	Comienzo	Fin
Investigación	**68 días**	**lun 10/09/18**	**mié 12/12/18**
Documentación PRIME	40 días	lun 10/09/18	vie 02/11/18
Investigar ATSAM4CP16B Evaluation Kit	22 días	lun 05/11/18	mar 04/12/18
Instalar software	6 días	mié 05/12/18	mié 12/12/18
Testing	**42 días**	**jue 13/12/18**	**vie 08/02/19**
Generar señales PRIME	5 días	jue 13/12/18	mié 19/12/18
Detectar señales	10 días	jue 20/12/18	mié 02/01/19
Muestrear y almacenar señales (BitScope)	27 días	jue 03/01/19	vie 08/02/19
Programar solucion	**101 días**	**lun 11/02/19**	**lun 01/07/19**
Limpiar y separar señal	10 días	lun 11/02/19	vie 22/02/19
Programar Bloques	**53 días**	**lun 25/02/19**	**mié 08/05/19**
Demodulador OFDM	30 días	lun 25/02/19	vie 05/04/19
Demodulador Digital	**13 días**	**lun 08/04/19**	**mié 24/04/19**
Scrambler	10 días	jue 25/04/19	mié 08/05/19
Adaptar código para multiples símbolos Payload	5 días	jue 09/05/19	mié 15/05/19
Traducir mensaje recibido	7 días	jue 16/05/19	vie 24/05/19
Memoria final	13 días	lun 27/05/19	mié 12/06/19

Ilustración 9. Planificación del proyecto

Este proyecto comenzó a principios del mes de septiembre del año 2018 y ha finalizado con su documentación y presentación ante tribunal el 19 de junio de 2019 (como se observa en el diagrama de Gantt de la Ilustración 7).

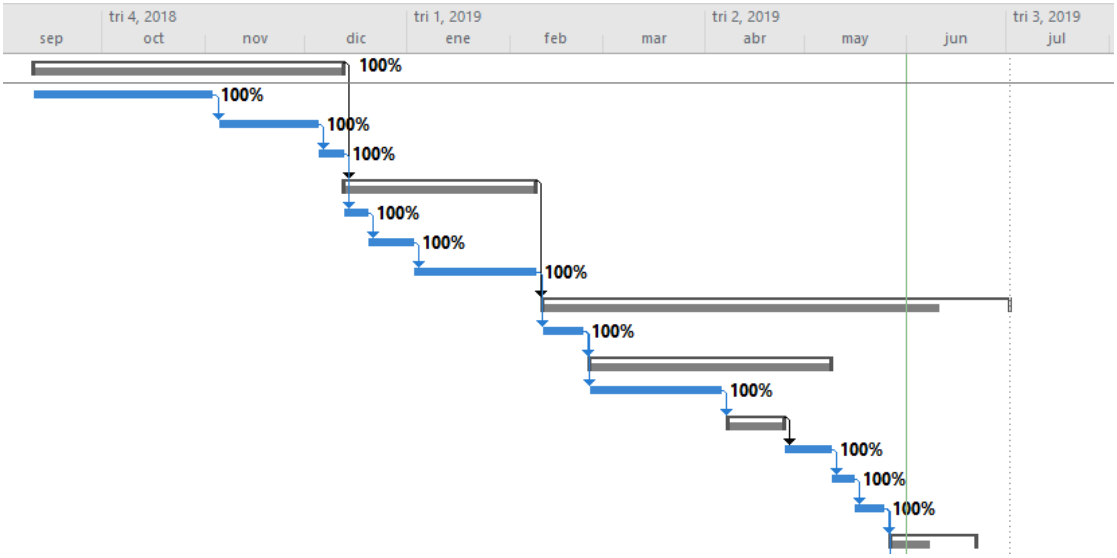


Ilustración 10. Diagrama de Gantt del proyecto

El proyecto se divide en tres bloques estructurales tanto por el contenido del trabajo como por el ritmo de trabajo llevado a cabo:

- De septiembre a diciembre de 2018. Inicio del proyecto. Periodo de documentación, investigación, aprendizaje, selección de herramientas, definición del proyecto e instalación y testing del software y hardware a utilizar. Ritmo de trabajo bajo.
    - Se lleva a cabo una profunda investigación y documentación acerca de la temática del proyecto a desarrollar: PRIME.
    - Una vez se tiene el conocimiento necesario para comenzar el proyecto, se buscan soluciones software y hardware para llevar a cabo los objetivos.
    - Fase de documentación acerca de los softwares y hardwares elegidos para el proyecto y testeo de las mismas:
        - ATSAM4CP16B Evaluation Kit (hw) y Atmel PLC PHY Tester Tool (sw) para generar mensajes bajo la especificación PRIME
        - BitScope (hw) y su librería BitLib (sw) para la detección, el muestreo y almacenamiento de señales PRIME.
- De enero a junio de 2019. Fase de implementación. Ritmo de trabajo alto.
    - Programar la solución.
        - Bloques del receptor

- Traducir mensaje

- Junio 2019. Memoria final del proyecto.
  - Una vez finalizado el proyecto, se documenta de manera detalla y explicativa.

4.4.2 ESTIMACIÓN ECONÓMICA

En la estimación económica, se sacarán 2 cifras:

- El coste total para llevar a cabo el proyecto.
- El coste del sistema en sí, es decir, el coste del proyecto sin las herramientas necesarias para simular una situación real (herramientas de testeo).

A continuación, en la Tabla 2, viene un listado y descripción del hardware que se ha necesitado para llevar a cabo este proyecto.

Componente	*Precio Unitario*	*Cantidad*	*Link*
BitScope	98,00 €	1	http://es.farnell.com/bitscope/bitscope-micro/oscilloscope-2-6ch-20mhz-40msps/dp/2432906
Adaptador BNC	24,00 €	1	http://es.farnell.com/bitscope/mp01a/bnc-adapter-bitscope-micro-oscilloscope/dp/2455505?ost=bitscope&categoryId=700000022505
Sonda de osciloscopio	56,00 €	1	https://es.rs-online.com/web/p/sondas-para-osciloscopios/7158733/
Jumpers	2,50 €	10	https://es.rs-online.com/web/p/jumpers-y-derivadores/0217851/

USB A macho a Micro USB B macho	2,37 €	1	https://es.rs-online.com/web/p/cables-usb/1213251/
ATSAM4CP16B Evaluation Kit	444,97 €	1	https://www.microchipdirect.com/product/search/all/ATBASENODE-EK

Tabla 2. Coste de componentes hardware

Además del hardware incluido en la tabla, se ha utilizado, y es parte imprescindible del sistema, un ordenador. Sin embargo, cualquier ordenador con capacidad para instalar el software necesario y correr Python, es válido. Como hay ordenadores que cumplen estas características desde 200€ a 5000 €, no se hablará de un precio concreto para el PC. También se puede utilizar una Raspberry-pi, mucho más económica (33€), compatible y capaz de cumplir con la misma funcionalidad que confiere el PC.

Por tanto, y concluyendo con el apartado de estimación económica, en la siguiente tabla (Tabla 3), se mostrarán los costes totales:

Coste del sistema	*Coste total*
178,00 € + ordenador (o Raspberry-pi)	627,84 € + ordenador (o Raspberry-pi)

Tabla 3. Costes totales del proyecto.

Capítulo 5. SISTEMA/MODELO DESARROLLADO

5.1 ANÁLISIS DEL SISTEMA

El sistema que se ha seguido es el descrito en el esquema de la Ilustración 9 y la comunicación entre distintos sistemas se muestra en la Ilustración 8.

En este subapartado, por tanto, se hará una descripción del sistema a muy alto nivel.

Ya en el apartado 5.2 de diseño, se profundizará en los componentes que se han escogido, las decisiones que se han tomado y el porqué de las mismas.

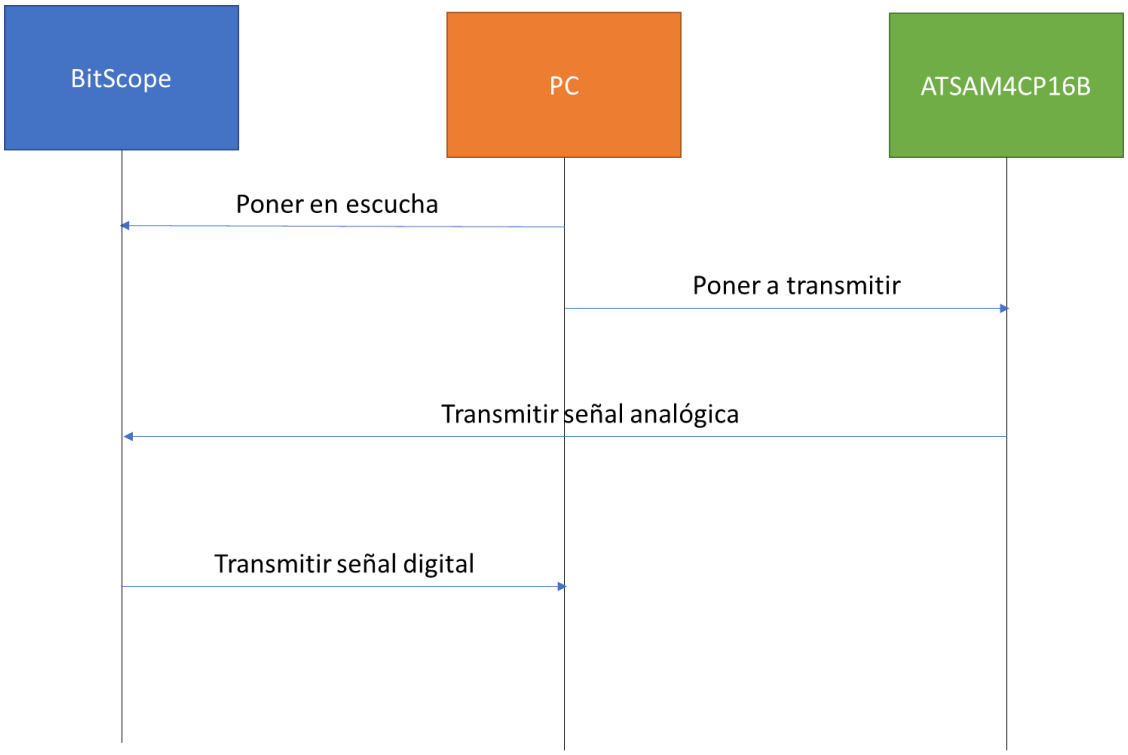


Ilustración 11. Comunicación entre sistemas

En la Ilustración 8 se observa cómo lo primero que se tiene que llevar a cabo es una doble comunicación del PC con:

- Primero con BitScope. Gracias a un programa Python y con la ayuda de la librería BitLib que permite controlar el hardware. Se pone en escucha el muestreador, se establece un trigger, y el muestreador disparará y empezará a muestrear cuando detecte una señal (incremento de voltaje).
    - Conexión PC-BitScope → mediante cable USB-miniUSB
- Segundo con ATSAM4CP16B Evaluation Kit. Mediante el software ATMEL PLC PHY TESTER TOOL. Se le comunica al hardware el mensaje que debe transmitir y la modulación digital que se prefiere.
    - Conexión PC- ATSAM4CP16B Evaluation Kit → mediante cable USB-microUSB

La transmisión de la señal analógica del ATSAM4CP16B Evaluation Kit al BitScope y la transmisión de la señal digital resultante de la salida del sampler al PC, se describen más detalladamente en la Ilustración 9.

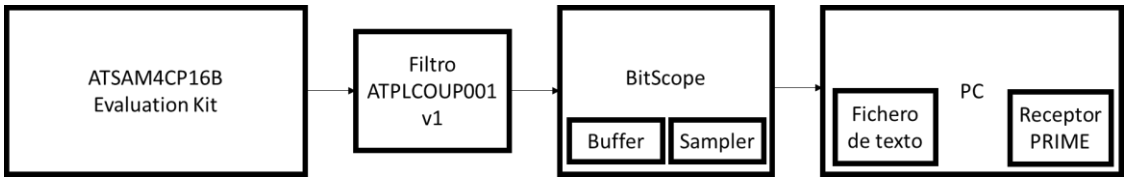


Ilustración 12. Esquema del sistema

El filtro ATPLCOUP001V1 se conecta a la placa ATSAM4CP16B para filtrar todas aquellas frecuencia que no son las de la banda de paso.

El BitScope se conecta mediante una sonda de osciloscopio al pin L5 del filtro ATPLCOUP001V1.

Una vez se empieza a transmitir, la señal pasa a través de la sonda y de un adaptador dual BNC hasta llegar al muestreador BitScope.

El sampler debe muestrear con un frecuencia de muetreo de 250 kHz.

Una vez se hacer la conversión A/D, dichas muestras se almacenan en el buffer el BitScope hasta llegar a la capacidad máxima del mismo (12000 muestras).

Las muestras se almacenan en el buffer, hasta que, desde el PC, se hace una llamada al BitScope para solicitar las muestras.

Las muestras van del BitScope al PC a través de un cable miniUSB-USB.

Una vez en el PC, se guarda en un text file y con un programa Python se ejecutan las funciones de un Receptor PRIME y finalmente se traduce el mensaje.

5.2 DISEÑO

Este apartado se va a utilizar para describir cómo se han llevado a cabo las decisiones técnicas de diseño, y el porqué de estas decisiones.

Los elementos hardware (con su correspondiente parte software) elegidos para este proyecto, y los motivos de su elección, son los siguientes:

- BitScope: para la conversión de señales PRIME de analógico a digital, se ha decidido utilizar el muestreador BitScope. Este hardware plantea un entorno totalmente funcional donde la digitalización de la señal de entrada (en este caso el mensaje PRIME), se encuentra totalmente resuelta, por lo que el muestreo de las señales es transparente al usuario. Además, es completamente programable, lo cual permite personalizarlo.

  Para su puesta en marcha, tan solo se debe llevar a cabo un proceso de inicialización y configuración sencillo con las condiciones deseadas al activar el dispositivo hardware (entre ellas, el valor umbral del trigger para el disparo, la frecuencia de muestro, o la cantidad de muestras a guardar en el buffer) y tras esto, permite centrar el trabajo únicamente en las muestras (digitales) que devuelve de la señal PRIME al PC.

  BitScope cumple con los requisitos necesarios para este proyecto de poder digitalizar la señal PRIME entrante para su posterior almacenamiento, paso por el receptor y traducción, por lo que todo lo relacionado con la captura de la señal y su conversión A/D queda resulto con BitScope.

Para comunicarse, configurar y programar el BitScope, se utilizará BitLib, la librería propia que nos proporciona, y el lenguaje utilizado será Python (ya que es uno de los lenguajes compatibles con BitLib, y el menos complejo).

- ATSAM4CP16B: hemos elegido este kit de evaluación ya que permite la generación de señales de PRIME de manera sencilla, sin complicaciones electrónicas, tiene un stack de comunicaciones PRIME completamente operativa, y proporciona un software para pruebas Atmel PLC PHY Tester Tool, que permite configurar el mensaje a enviar y las características del mismo (frecuencia de envío, modulación digital, contenido del mensaje, etc) con una interfaz gráfica de muy alto nivel, con total facilidad (más allá de la instalación del software y resolver las compatibilidades con el ordenador).

  Además, esta placa de desarrollo proporciona una plataforma con todas las funciones para desarrollar un sistema de comunicaciones completo a través de PLC.
  - ATPLCOUP001V1: se ha decidido utilizar un filtro paso banda plana con impedancias de campo típicas, de modo que permita el paso a las frecuencias en la banda de frecuencias PRIME (41 KHz - 89KHz), y elimine el resto, ayudando así a limpiar la señal y evitar interferencias de otras frecuencias.

- PC (o Raspberry-pi): el ordenador tendrá una doble función en la simulación del sistema (que aplicado a un caso de sniffing real, será tan solo una). Por un lado, la de comunicarse con la placa ATSAM4CP16B y transmitirle la información necesaria para transmitir, y, por otro lado, la de comunicarse con el BitScope y recibir y guardar la señal una vez muestreada. Para ello, se necesitaba un dispositivo que pudiera:
  - Tener Python instalado, y compatibilidad con la librería BitLib de BitScope de modo que se puedan ejecutar los programas que:
    - Programa 1: capturan y muestrean la señal analógica PRIME
    - Programa 2: hacen de Receptor PRIME, traducen el mensaje a texto, y almacenan la salida en un fichero.

Esto es necesario tanto para el sistema en sí, como para la simulación del mismo.

Dispositivos y OS que permiten correr ambas tecnologías, son: ordenador con Windows, Linux o MAC, o una Raspberry-Pi.

- Tener capacidad y compatibilidad para instalar Atmel PLC PHY Tester Tool. Necesario tan solo en la simulación, para simular un canal de transmisión de mensajes bajo la especificación PRIME. Sirve para comunicarse con la placa ATSAM4CP16B y proporcionarle la información necesaria para transmitir. Solo es compatible con ordenadores con sistema operativo Windows.

5.3 IMPLEMENTACIÓN

5.3.1 SIMULACIÓN DEL CANAL DE TRANSMISIÓN

Como se ha explicado, se ha decidido utilizar una placa de desarrollo con las funciones necesarias para desarrollar un sistema de comunicaciones completo, usando tecnología PLC y bajo la especificación de PRIME, ATSAM4CP16B EVALUATION KIT.

De este modo, con esta herramienta, se tratará de simular una comunicación de mensajes PRIME.

5.3.1.1 Software ATMEL PLC PHY TESTER TOOL

Para controlar el funcionamiento de la placa, se dispone de un software llamado ATMEL PLC PHY TESTER TOOL.

Este software permite indicar el mensaje a transmitir por la placa y las características deseadas en la transmisión.

A continuación, se detallará su funcionamiento a través de sus pantallas de configuración:

Una vez instalado, se ejecuta la aplicación, se muestra la pantalla de la Ilustración 10.

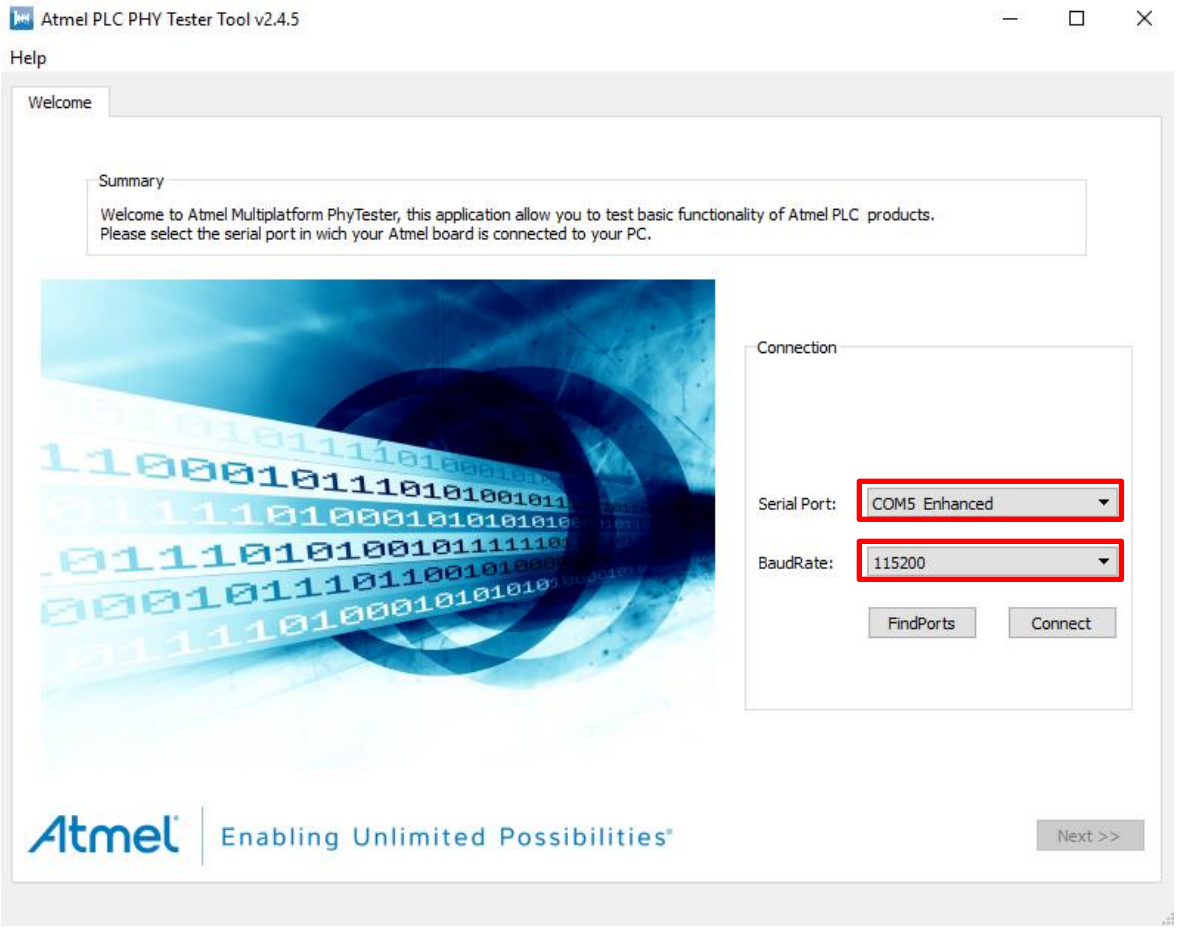


Ilustración 13. Pantalla Configuración Tester Tool 1

La pantalla mostrada en la Ilustración 10 es la pantalla de comienzo. En esta pantalla se debe buscar el puerto al que vamos a conectar la placa. La placa se puede conectar a la red eléctrica, o como en este caso, vale con conectar la placa a un ordenador a través del conector microUSB de la misma y a un puerto USB del ordenador. Una vez identificado el puerto, debe ser seleccionado en la lista desplegable de Serial Port.

Además del puerto, se debe seleccionar la velocidad de transmisión adecuada en Baudios. Para este proyecto se ha seleccionado que la velocidad de transmisión debe ser de 115200 Baudios.

Una vez seleccionadas ambas características de conexión, se pulsa el botón “Connect”. Esto desencadena un proceso de identificación de la placa; y después de unos segundos, el texto

del botón cambia a "Disconnect", lo que significa que el proceso de identificación ha finalizado correctamente y ya hay conexión entre el PC y la placa. Se pulsa “Next”.

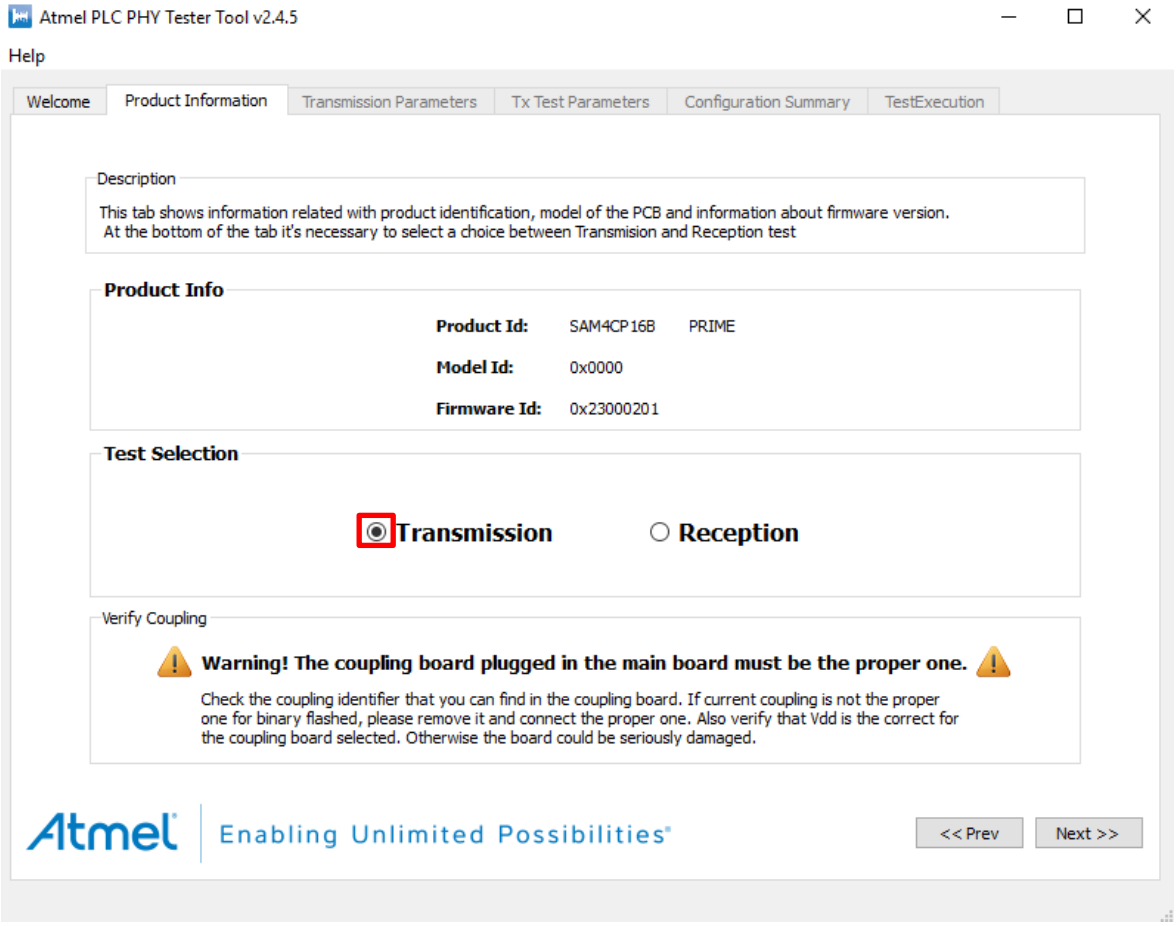


Ilustración 14. Pantalla Configuración Tester Tool 2

Aparece una nueva pantalla de configuración en la Ilustración 11, con información del producto y si se desea Transmitir o Recibir. En este caso, como se quiere simular un canal de transmisión de mensajes PRIME, se selecciona “Transmission”.

Se pulsa “Next”.

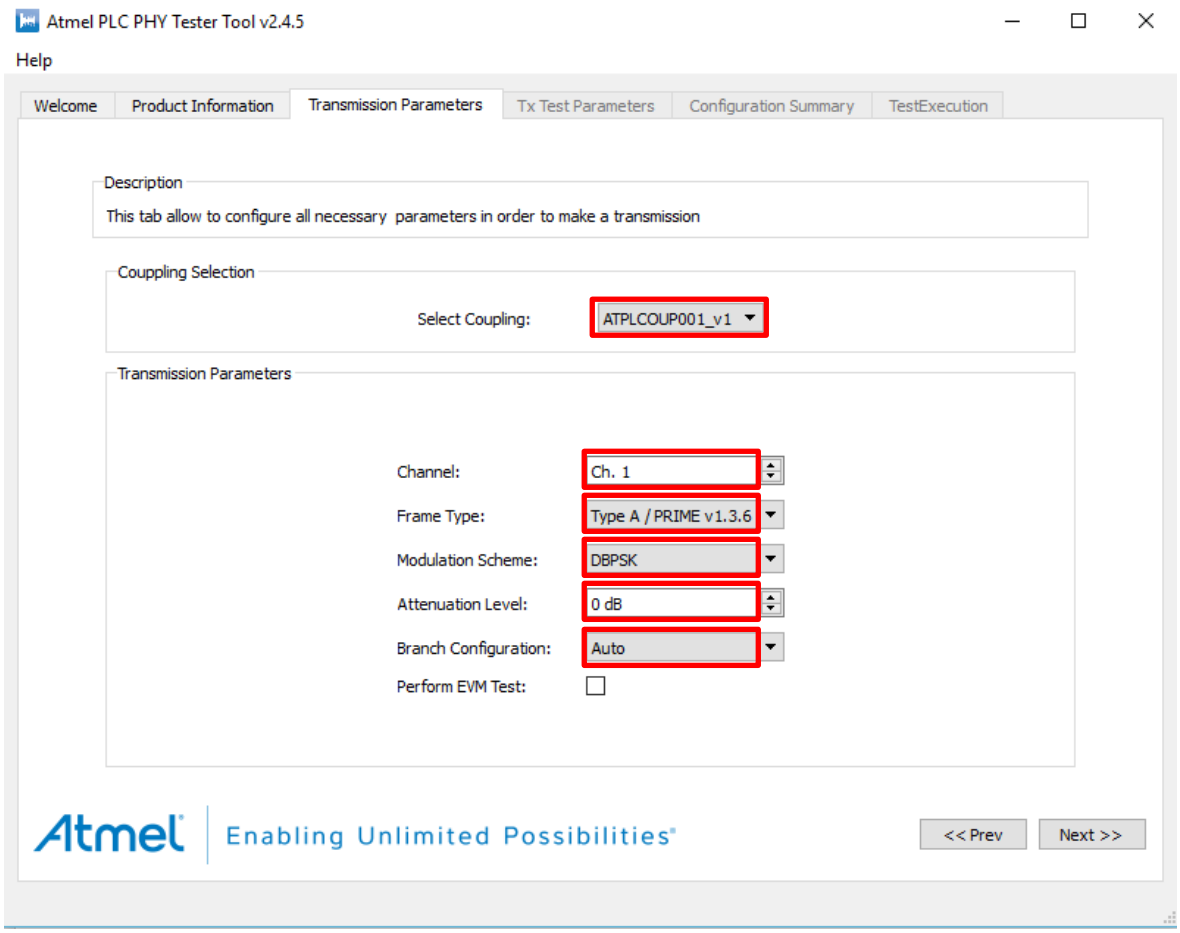


Ilustración 15. Pantalla Configuración Tester Tool 3

Si se ha elegido "Transmission", la siguiente pantalla será la mostrada en la Ilustración 12.

Se debe elegir:

- *Select Coupling*: según el acoplamiento utilizado. En este caso, se está utilizando el filtro ATPLCOUP001v1.
- *Channel*: permite seleccionar en qué canal se transmitirán las tramas. Dependiendo del acoplamiento conectado a la placa, pueden estar disponibles diferentes canales. Por defecto se selecciona el Ch. 1.
- *Frame type*: configura la placa para transmitir diferentes tipos de tramas (Tipo A / B / BC) siguiendo la especificación PRIME 1.4. En este proyecto se trabaja con tramas Tipo A.

- *Modulation Scheme*: configura la modulación de las tramas. En este proyecto se podrán tratar las 3 modulaciones digitales posibles, tanto DBPSK, DQPSK o D8PSK.
- *Attenuation Level*: permite atenuar la señal transmitida. En este caso no será necesario atenuar la señal, por lo que se podrá 0 dB.
- *Branch Configuration*: configura la etapa de salida dependiendo de la impedancia de línea vista por la placa. Por defecto en Auto.
- *Perform EVM test*: al seleccionar esta opción, se cambia el mensaje y el intervalo de transmisión para realizar una prueba que evalúe el rendimiento de la capa PHY. No se considera necesario, por lo que se deja "no checked".

Se pulsa "Next".

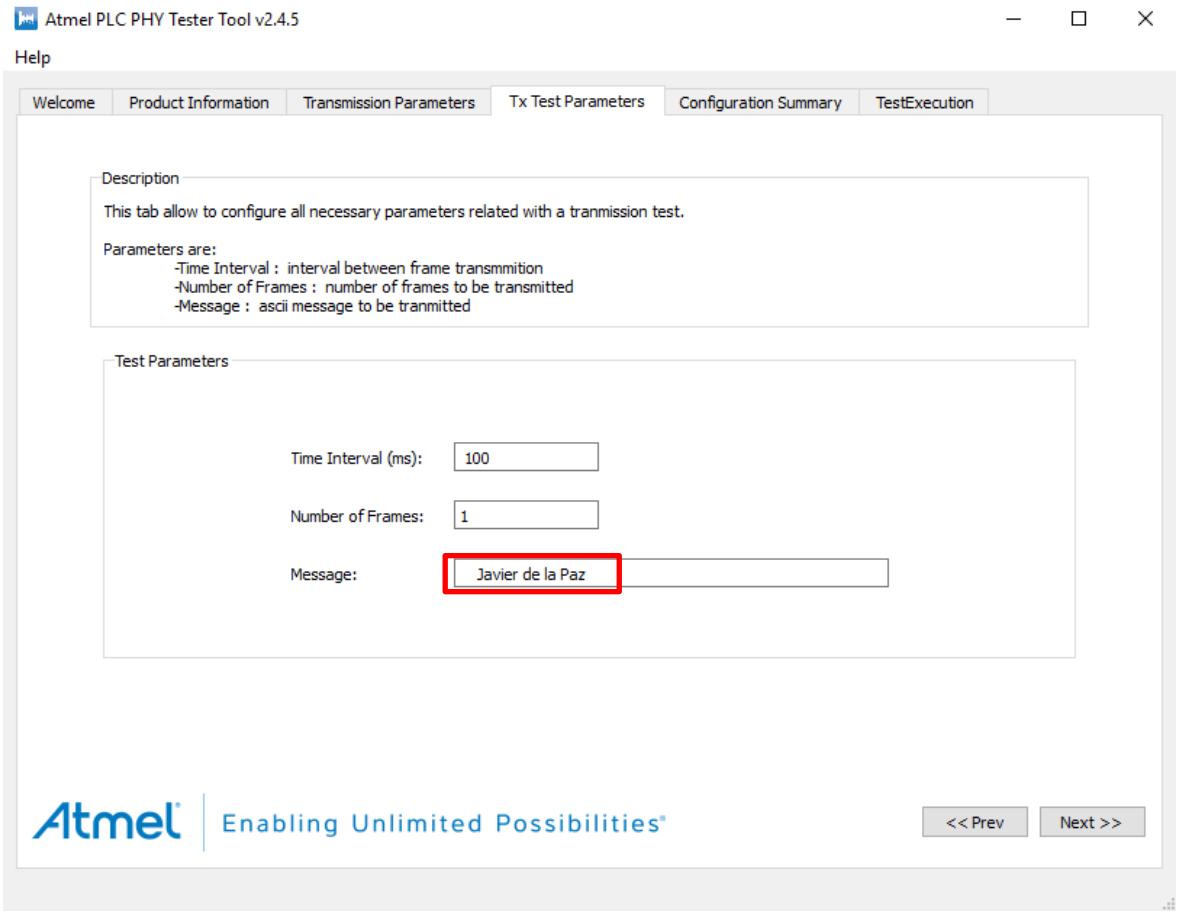


Ilustración 16. Pantalla Configuración Tester Tool 4

En la pantalla mostrada en la Ilustración 13, se deben seleccionar:

- *Time Interval:* cada cuanto transmitir cada copia del mensaje (en ms).
- *Number of frames*: número de copias del mensaje a transmitir.
- *Message*: mensaje a transmitir.

IMPORTANTE: se ha descubierto una limitación en el software no contemplada en la documentación. Los cuatro primeros caracteres del mensaje no se envían, por lo que lo que se quiera transmitir debe estar a partir del quinto.

Se pulsa "Next".

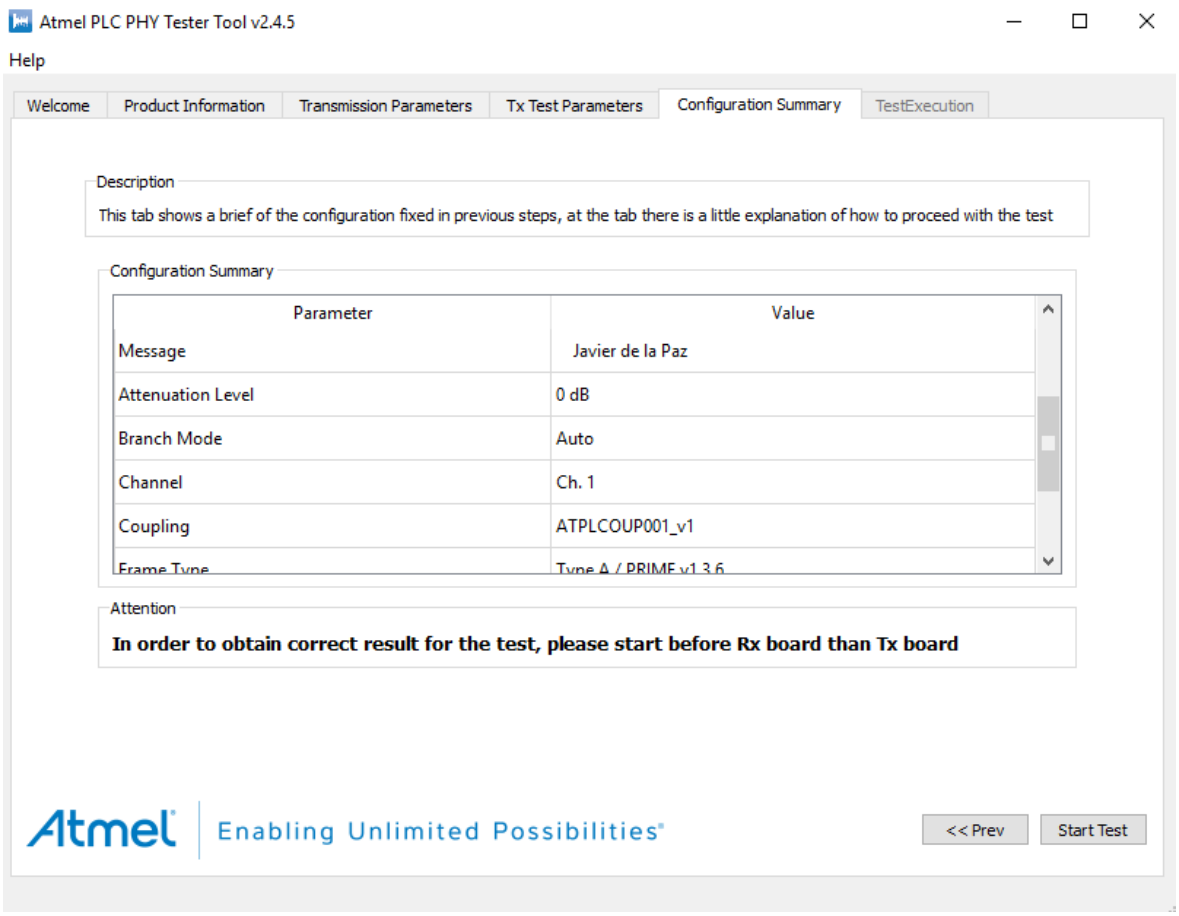


Ilustración 17. Pantalla Configuración Tester Tool 4

En la siguiente pantalla que se muestra en la Ilustración 14, aparece un resumen con la configuración establecida para la transmisión del mensaje.

Si se está de acuerdo, se podrá pulsar el botón "Start test" y comenzará a transmitir.

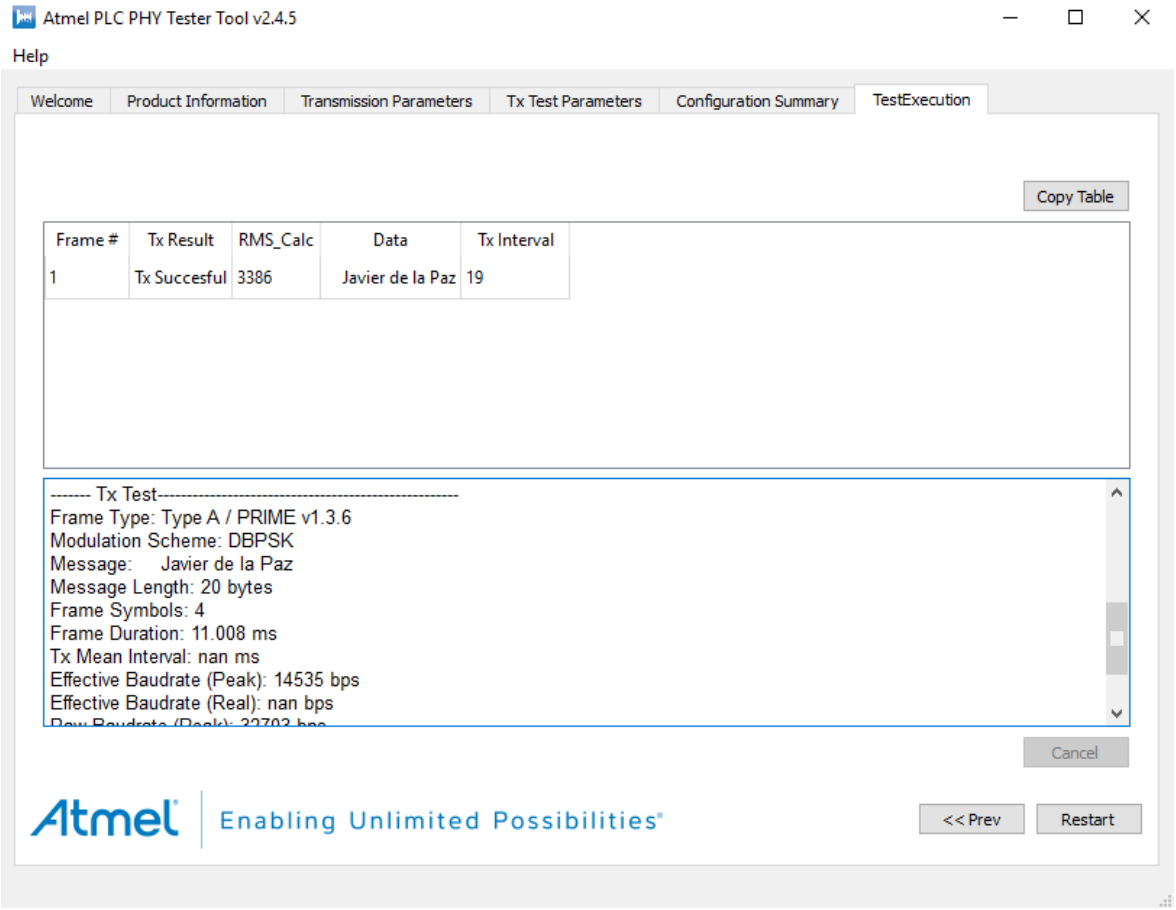


Ilustración 18. Pantalla Configuración Tester Tool 5

En la pantalla de la Ilustración 15 se muestra una tabla a la cual se le agrega una fila a la parte superior por cada trama transmitida. La tabla contiene cuatro columnas que muestran:

- *Frame #*: indica el número de frames transmitidos.
- *Tx Result*: indica el resultado de la transmisión. Si se produce un error, aparecerá un texto descriptivo.
- *RMS_Calc*: este número está relacionado con la impedancia detectada en la línea eléctrica. Es utilizado por la capa PHY para determinar el modo de transmisión.

- *Data*: muestra el mensaje recibido en formato ASCII.
- *Tx Interval*: representa el tiempo medido entre el frame actual y el frame anterior.

Una vez que se hayan transmitido todos los frames, aparecerá un cuadro de texto con información sobre la prueba en la parte inferior de la pantalla. Si se desea, se puede detener la transmisión el momento que se considere oportuno.

5.3.1.2 Hardware ATSAM4CP16B

La implementación del hardware ATSAM4CP16B, es sencilla.

- Por una parte, la placa debe estar conectada con el PC en el que se vaya a correr el software Atmel PLC PHY Tester Tool, mediante un cable microUSB – USB. De esta forma se llevará a cabo:
  - La alimentación de la placa.
  - La transmisión de mensajes PRIME.
- Por otra parte, se conectará el filtro paso banda ATPLCOUP001 a la placa siguiendo la Ilustración 16. Este filtro es una placa de acoplamiento PLC de aislamiento de tensión de red, y está optimizada para comunicaciones en la banda PRIME de 41 kHz a 89 kHz. Esto permitirá eliminar ruido e interferencias de otras frecuencias a la salida del mismo.

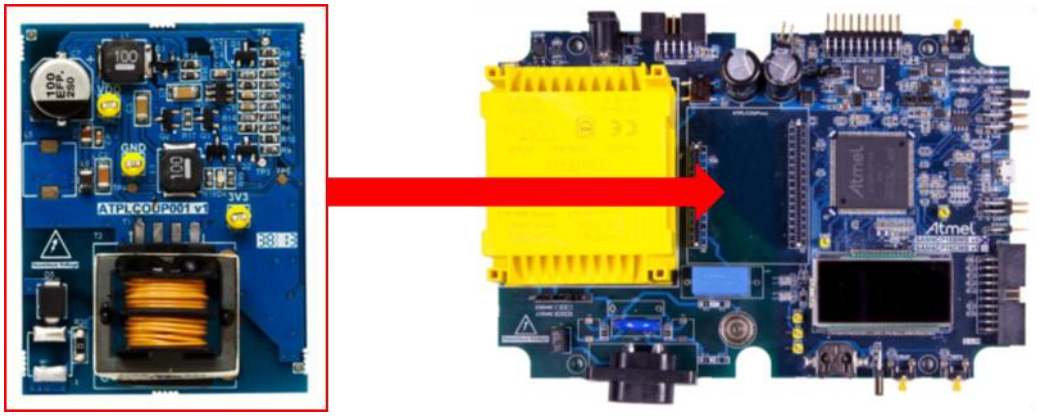


Ilustración 19. Conexión ATPLCOUP001 Coupling board.

- Por último, se conectará la sonda del osciloscopio al pin inferior L5 del filtro ATPLCOUP001 (el que se muestra en la Ilustración 17 como "Output") y la tierra de la sonda al GND del filtro (utilizaremos la tierra del filtro y no la de la placa).

  La sonda del osciloscopio irá conectada a la entrada de un adaptador BNC conectado al BitScope de modo que así puede recibir la señal el muestreador.

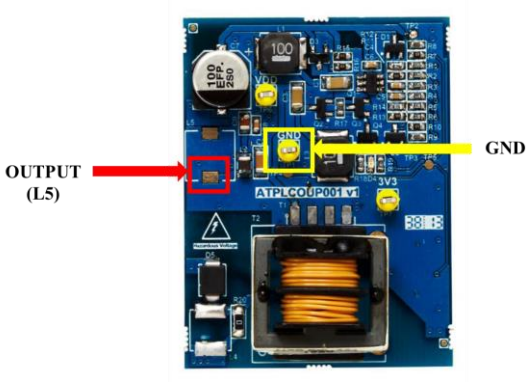


Ilustración 20. Pines ATPLCOUP001

5.3.2 SISTEMA

El sistema estará formado por tres bloques que se muestran en la Ilustración 18 y que se explicarán con detalle en este apartado:

- Receptor PRIME
- Traductor UTF8
- Almacenamiento de salida

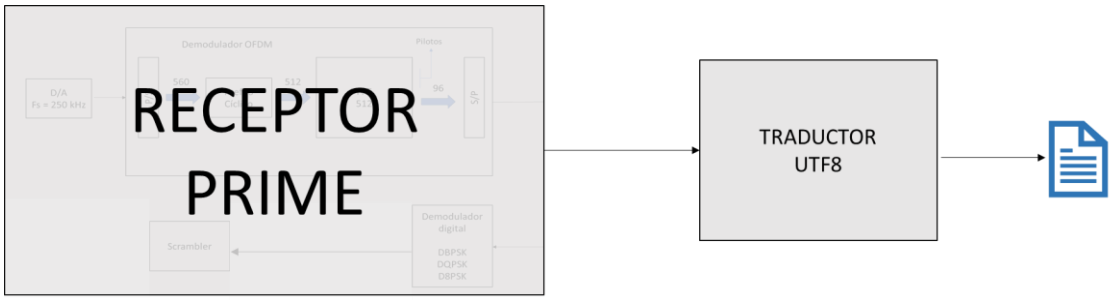


Ilustración 21. Recepción, traducción y almacenamiento de la salida

5.3.2.1 Receptor PRIME

El receptor PRIME está compuesto por los bloques que se muestran en la ilustración 19. Y son los siguientes:

- Conversor A/D
- Demodulador OFDM
- Demodulador digital (3 posibilidades de modulación)
    - DBPSK
    - DQPSK
    - D8PSK
- Scrambler

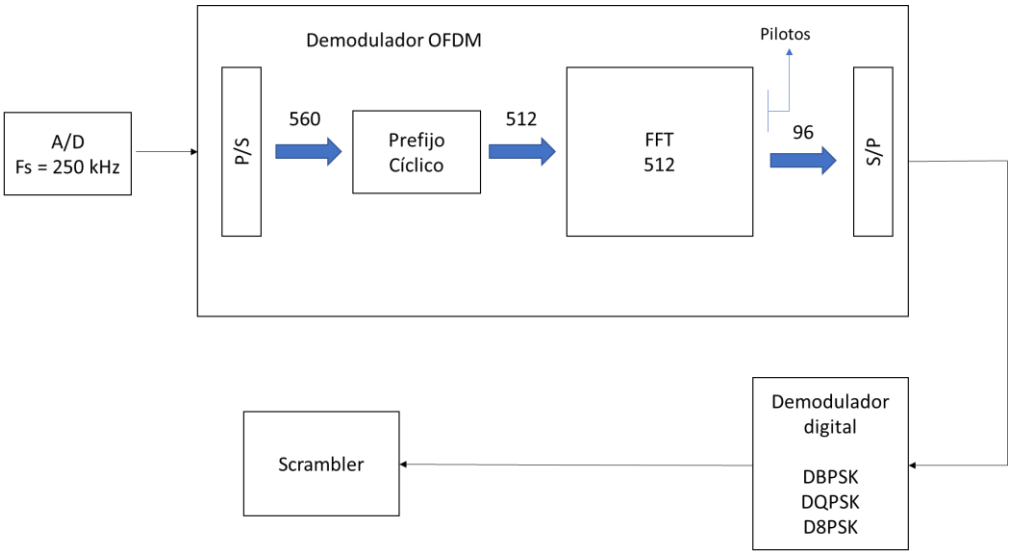


Ilustración 22. Receptor PRIME

5.3.2.1.1 Muestreo

El muestreo de la señal se llevará a cabo con BitScope. BitScope es un muestreador completamente programable, que nos permitirá detectar la señal PRIME, capturarla, convertirla a digital y almacenarla.

BitScope es un hardware y se controlará a través de Python y una librería propia de BitScope, BitLib.

Para el muestreo se ha creado una clase en Python que se ha decidido llamar *Sampler* a la que se llamará a través de un puntero desde el programa principal, y que será la encargada de la conexión y comunicación con el BitScope.

Para trabajar con el BitScope, siempre se debe seguir la siguiente secuencia:

(1) Initialize → (2) Setup → (3) Trace → (4) Acquire → (5) Close

Lo primero es inicializar y hacer el setup. Se va a explicar como llevar a cabo las cinco fases para la utilización del BitScope a partir de su código, detallando a qué fase pertenece cada bloque, qué se puede hacer en cada fase, y qué se ha hecho y elegido en la configuración y ejecución para este proyecto.

(1) Inicialización

```
BL_Initialize() # Inicializa la librería.
BL_Open("", 1) #Abre el dispositivo.
```

Inicializamos y abrimos el dispositivo.

(2) Setup

```
BL_Count(BL_COUNT_ANALOG) # Detecta el número de canales analógicos
BL_Count(BL_COUNT_LOGIC) # Detecta el número de canales lógicos

BL_Select(BL_SELECT_DEVICE, MY_DEVICE=0) # Seleccionar dispositivo
BL_Select(BL_SELECT_CHANNEL, MY_CHANNEL=0) # Seleccionar canal
BL_Select(BL_SELECT_SOURCE, BL_SOURCE_POD) # Seleccionar dispositivo

BL_Mode(MY_MODE=BL_MODE_FAST # Modo de trace
BL_Range(BL_Count(BL_COUNT_RANGE)) # Selecciona el rango de canales
BL_Offset(BL_ZERO) # Asigna el offset del canal
BL_Enable(TRUE) # Cambia el estado del canal a habilitado.
```

Se selecciona el dispositivo 0 (el correspondiente al BitScope), el canal 0 (correspondiente al CHA, a través del cual estará conectada la sonda de osciloscopio con la que medir), el offset inicial será 0 y por último se habilitará el canal.

```
BL_Rate(MY_RATE) # Frecuencia de muestreo (Fs)
BL_Size(MY_SIZE) # Tamaño de muestras a almacenar en buffer
```

Se establece la frecuencia de muestreo a 500 kHz (la Fs de PRIME es 250 kHz, pero más adelante se explicará por qué utilizando la placa ATSAM4CP16B, no se puede muestrear a 250 kHz).
Y se elige el tamaño máximo a almacenar en buffer, que en BitScope son 12000 muestras.
Ambos parámetros se le pasarán desde el programa principal al Sampler.

- Rate = 500000
- Size = 12000

Los pasos 1 y 2 se hacen con la llamada a la función *scope_acquire()* del Sampler, desde el programa principal.

(3) Traza
(4) Adquisición

Las etapas 3 y 4 van siempre juntas y se deben repetir cada vez que se quiera hacer una nueva captura de señal.

Para dichas etapas, se ha creado una función llamada *scope_acquire()*, que lleva a cabo las llamas a la las funciones de la librería BitLib necesarias.

```
BL_Trigger(self.TRIGGER_VALUE,BL_TRIG_RISE) # Establece el nivel del trigger
BL_Trace(BL_TRACE_FOREVER, False) # Comienza el trace y captura la señal
self.ch1_data = BL_Acquire() # Adquiere datos
```

Se establecerá un trigger de 0.05V de modo que sea el suficiente para no leer ruido de bajo nivel que se pudiera introducir en el muestreador, pero que en cuanto se detectará una subida apreciable de tensión (esperablemente de una señal PRIME entrante), se disparará.

Se ha puesto el dispositivo en espera indefinida, de modo que permanerá en ese estado hasta la llegada de una señal.

Cuando se capture la señal y sea muestreada, se guardará en el buffer de BitScope.

Por último, con la función *getLastSample*, se cogerá la señal ya muestreada y se pasará desde el buffer del BitScope al PC.

(5) Cerrar

```
BL_Close() # Cierra todos los dispositivos abiertos
```

Se cierra el dispositivo una vez finalizado todo el proceso.

Una vez tenemos la señal en el ordenador, se almacena en un fichero, de modo que se pueda leer, procesar y modificar en estático.

Tras pasar el muestreo de la señal, llegan el resto de los bloques del Receptor PRIME que se van a ejecutar en un nuevo fichero Python.

IMPORTANTE: como se ha comentado anteriormente, debido a una limitación del hardware del filtro ATPLCOUP001V1, no es posible muestrear la señal PRIME de salida de la placa a una frecuencia de muestreo de 250 kHz.

Para solucionar este inconveniente, se ha decidido muestrear al doble de frencuencia (a 500 kHz) y luego diezmar en un factor de 2 como se muestra en la Ilustración 20.

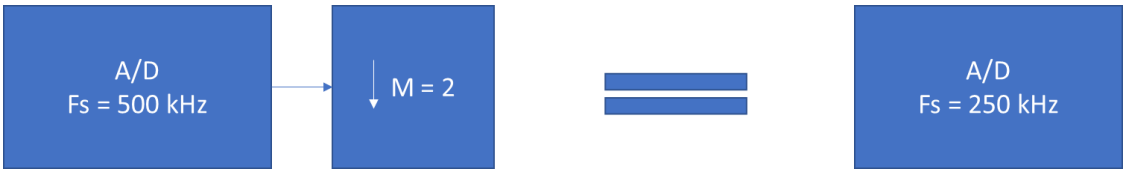


Ilustración 23. Diezmado

El diezmado de una señal consiste en modificar la frecuencia original de muestreo (f_{s1}) reduciéndola por un factor entero de M, de tal forma que:

$$f_{s2} = \frac{f_{s1}}{M}$$

5.3.2.1.2 Pre-procesado de la señal

Antes de empezar a trabajar con la señal, se tiene que llevar a cabo dos tareas:

- Limpiar la señal: eliminar todas aquellas muestras iniciales con valor cercano a 0 que no pertenecen a la señal.
- Identificar y separar las distintas partes de la señal:
    - Preámbulo (2.048 ms)
    - Cabecera (4.48 ms)
    - Carga útil (2.24ms x M símbolos)

  Es importante ya que cada parte será analizada y procesada por separado.

El preámbulo se utiliza al principio de cada PPDU para fines de sincronización. Para proporcionar un máximo de energía, se utiliza una señal de envolvente constante en lugar de símbolos OFDM.

La cabecera PHY se compone de dos símbolos OFDM que siempre se envían utilizando la modulación DBPSK. Sin embargo, el payload tiene modulación es DBPSK, DQPSK o D8PSK, según la configuración de la capa MAC.

Los dos primeros símbolos OFDM en la PPDU correspondientes al encabezado de la PHY están compuestos por 84 subportadoras de datos y 13 subportadoras piloto. Después del encabezado PHY, cada símbolo OFDM en la carga útil lleva 96 subportadoras de datos y una subportadora piloto. Cada subportadora de datos lleva 1, 2 o 3 bits en función de la modulación utilizada.

Por tanto, la estructura del frame PHY es la que se muestra en la Ilustración 21. Cada frame PHY comienza con un preámbulo con una duración de 2.048 ms, seguido de una serie de símbolos OFDM, cada uno con una duración de 2.24 ms. Los dos primeros símbolos OFDM llevan el cabecera de trama PHY. El cabecera PHY también se genera como se describe en la cláusula 7.4. El valor de M se señala en el encabezado PHY, y es como máximo igual a 63.

En la Ilustración 21 se muestra la estructura por duración de las partes del símbolo ODFM.

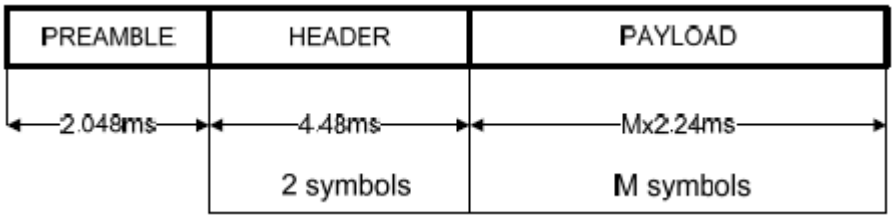


Ilustración 24. Formato PHY de frame PRIME [1]

En la Tabla 4, se muestra la velocidad de datos de PHY durante la transmisión de la carga útil y la longitud máxima de MSDU para diferentes combinaciones de modulación y codificación.

	DBPSK	*DQPSK*	*D8PSK*
Bits de información por subportadora, NBPSC	1	2	3
Bits de información por símbolo OFDM, NBPS	96	192	288
Velocidad de datos sin procesar (kbit/s aprox.)	42.9	85.7	128.6
Longitud máxima de MSDU (en bits) con 63 símbolos	6048	12096	18144
Longitud máxima de MSDU (en bytes) con 63 símbolos	756	1512	2268

Tabla 4. Data rates PHY para cada modulación.

5.3.2.1.3 Canal

Se considera el canal como el medio de transmisión por el que viajan las señales portadoras de información, en este caso la señal PRIME, entre emisor y receptor.

Se buscará su respuesta en frecuencia y cómo afectará a la señal. Para ello se estimará el canal gracias a las muestras del preámbulo de la señal PRIME.

5.3.2.1.4 Demodulador OFDM

Para llevar a cabo la demodulación OFDM, se debe realizar símbolo a símbolo.

Por cado símbolo OFDM, se debe:

- Eliminar las ultimas 48 muestras de cada bloque, es decir, las correspondientes al prefijo cíclico (N_{CP}), y quedarse con las 512 iniciales.
- Hacer la FFT de modo que se consiga el mapeo inverso de subportadoras mostrado en la ilustración 22. Se busca revertir el mapeo llevado a cabo con la IFFT.

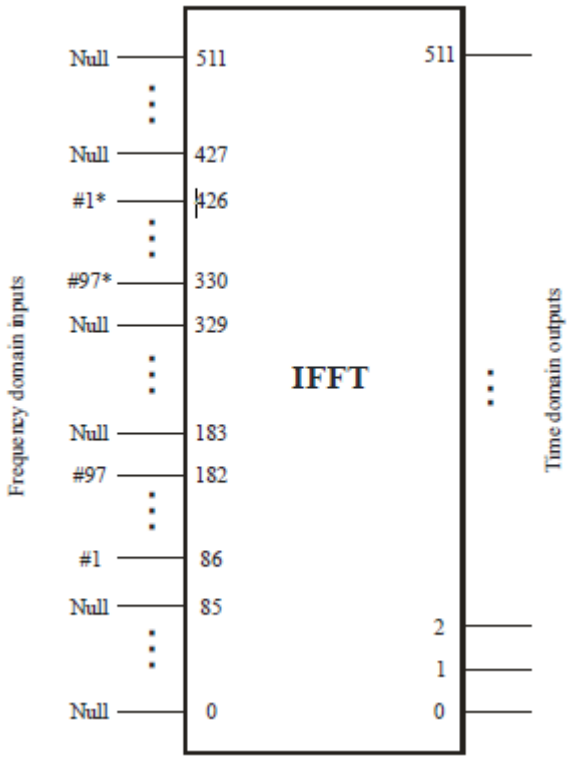


Ilustración 25. Mapeo de subportadoras [1]

Una vez realizada la demodulación OFDM se debe separar de cada símbolo digital resultante de la FFT, el piloto (primera portadoras) del resto de subportadoras (las siguientes y restantes 96). Es necesario este paso para poder realizar la demodulación digital DPSK posterior.

5.3.2.1.5 Demodulador digital DPSK

La salida del demodulador ODFM son símbolos digitales, por lo que los siguientes pasos serán:

- Deshacer el "efecto" del piloto
- Demodulación digital de los símbolos

Como se ha comentado con anterioridad, el payload de PPDU ha sido modulado como una señal de codificación de cambio de fase diferencial multiportadora con una subportadora piloto y otras 96 subportadoras de datos que comprenden 96 (DBPSK), 192 (DQPSK) o 288 (D8PSK) bits por símbolo.

El proceso de demodulación consiste en reconstruir los bits modulados a partir de los símbolos de la constelación recibidos. En una modulación por desplazamiento de fase, la discriminación de esos símbolos se hace observando fase de los mismos, y al ser diferencial, se lleva a cabo comparando la fase de cada símbolo con el anterior.

Los símbolos están modulados diferencialmente en el dominio de la frecuencia, en alguna de las tres posibles modulaciones diferenciales DPSK como se muestra en la ilustración 23:

- DBPSK
- DQPSK
- D8PSK

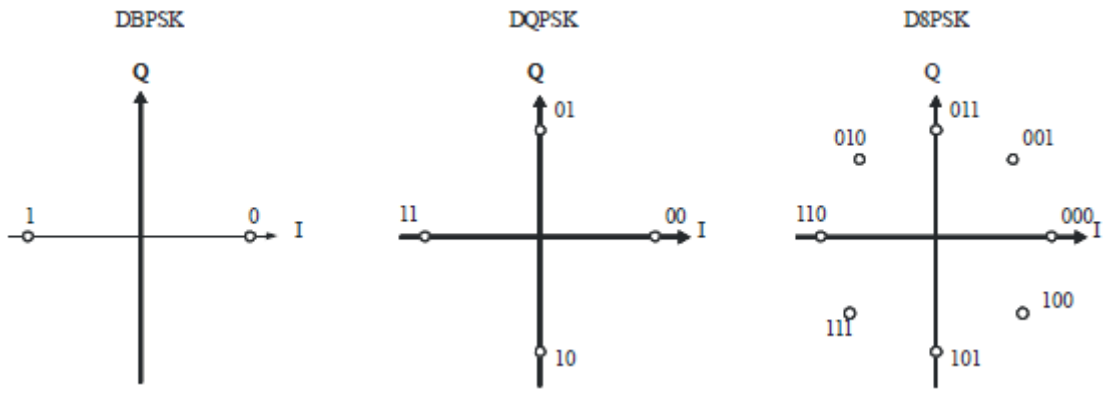


Ilustración 26. Modulaciones digitales DPSK [1]

La subportadora piloto proporciona una referencia de fase para la demodulación DPSK en el dominio de la frecuencia. Es por esto por lo que se debe revertir el "efecto" del piloto en cada símbolo digital antes de demodularlo. Para ello, basta con multiplicar cada símbolo por el ángulo de la subportadora piloto.

La constelación DPSK de M fases, viene definida como:

$$S_k = Ae^{(j\theta_k)}$$

Donde:

- k es el índice de frecuencia que representa la subportadora k-th en un símbolo OFDM.
- k = 1 corresponde a la subportadora piloto de referencia de fase.
- k > 1 corresponde al payload.
- S_k es la salida del modulador (un número complejo) para la subportadora k-th dada.
- θ_k representa la fase absoluta de la señal modulada
- M = 2, 4 u 8 en el caso de DBPSK, DQPSK o D8PSK, respectivamente

Se habla de una modulación m-ary cuando el mensaje binario se fragmenta para formar símbolos de "m" bits cada uno.

Cuando se transmite la cabecera, el piloto asignado en la subportadora k-th se utiliza como una referencia de fase para los datos asignados en la subportadora k + 1-th.

Al ser la demodulación, se convertirán los símbolos de la constelación en 1, 2 o 3 bits según sea la modulación DBPSK, DQPSK o D8PSK.

Efecto de ecualización

Además, como punto adicional, se mostrará los diferentes resultados obtenidos entre un sistema con ecualización y otro sin ella.

A priori, al no ser un canal con mucho ruido, y utilizando una modulación diferencial, no debería haber problemas para sacar el mensaje limpiamente sin necesidad de ecualización, pero esto se comprueba en el proyecto de manera real consiguiendo corregir las pequeñas diferencias de fase entre símbolos de la constelación pertenecientes al mismo código (bits).

5.3.2.1.6 Scrambler

Aleatoriza el flujo de bits para que reduzca el factor de pico (cresta) en la salida del IFFT.

Se utiliza para evitar la aparición de secuencias largas de bits idénticos.

Realiza un XOR del flujo de bits de entrada utilizando una secuencia de pseudo ruido pn obtenida por una extensión cíclica de la secuencia de 127 elementos dada por:

$Pref_{0..126}$=
{0,0,0,0,1,1,1,0,1,1,1,1,0,0,1,0,1,1,0,0,1,0,0,1,0,0,0,0,0,0,1,0,0,0,1,0,0,1,1,0,0,0,1,0,1,1,1,0,
1,0,1,1,0,1,1,0,0,0,0,0,1,1,0,0,1,1,0,1,0,1,0,0,1,1,1,0,0,1,1,1,1,0,1,1,0,1,0,0,0,0,1,0,1,0,1,0,1
,1,1,1,1 0,1,0,0,1,0,1,0,0,0,1,1,0,1,1,1,0,0,0,1,1,1,1,1,1,1}

La secuencia PRBS puede ser generada por el scrambler de la Ilustración 24 cuando se usa el estado inicial de todos-unos.

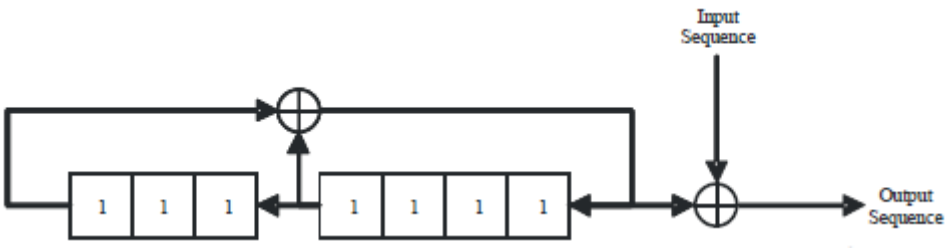


Ilustración 27. Scrambler [1]

Si el número de bits de entrada al scrambler es superior a 127, se repite la cadena de bits al final de esta de manera secuencial.

La carga de la secuencia pn se iniciará al comienzo de cada PPDU, justo después del preámbulo.

5.3.2.1.7 Traductor UTF8

Una vez se obtiene la salida de bits del scrambler, se tiene el output del Receptor PRIME.

Sin embargo, los bits de salida están codificados en UTF-8.

UTF8 es un formato de codificación de caracteres Unicode e ISO 10646 que utilizan símbolos de longitud variable.

Para su traducción bastará con hacer una doble transformación:

- Bit → int
- int → char

De este modo se logra el número asociado a los bits de la secuencia, y de ahí sacar su equivalencia en carácter ASCII.

La traducción se debe hacer de 8 en 8 bits, ya que cada 8 bits se forma una letra del mensaje transmitido

5.3.2.1.8 Almacenamiento de salida

A la salida se almacenará:

- Por un lado, el texto contenido en el mensaje PRIME, en un fichero de texto ya traducido.
- Por otro lado, se generará una carpeta con gráficas con los bits a la salida de cada bloque del Receptor PRIME

Capítulo 6. ANÁLISIS DE RESULTADOS

Para llevar a cabo el análisis de los resultados del proyecto, se mostrarán los resultados ante un mismo mensaje recibido, en las 3 posibles modulaciones digitales.

Se mostrará el resultado gráfico de cada bloque, si es que lo tiene, y de no ser así, se hará un explicación de lo ocurrido en dicha fase.

Para ello, se configura la placa ATSAM4CP16B con el ATMEL PLC PHY TESTER TOOL:

- Modo: transmisión
- Modulación - se van a analizar los resultados de las 3 modulaciones digitales:
  - DBPSK
  - DQPSK
  - D8PSK
- Mensaje – se va a transmitir un mensaje personalizado y largo, de modo que sea único y que tenga varios símbolos OFDM el payload, para mostrar cómo reacciona el Receptor. El mensaje será: “Me llamo Javier de la Paz y este es mi TFM”.

Una vez configurada la placa, se corre el programa Python *SignalCapture*, y se deja al programa Python en modo de escucha.

Se conecta la sonda de osciloscopio a la placa ATSAM4CP16B y se pulsa el botón de transmitir.

Cuando la placa transmite, el BitScope lo detecta, dispara, captura la señal, la muestrea y al almacena en el buffer. Las muestras pasan al PC, que las guarda en un fichero de texto para poder ser procesadas, y se muestra una representación de la señal muestreada.

A partir de este punto, se mostrarán los resultados obtenidos de las 3 ejecuciones.

Todas las gráficas que se muestreen son salidas del propio proceso que las genera.

Se procederá a hacer una representación de las tres modulaciones, de modo que se pueda apreciar y valorar, las diferencias apreciables entre ellas donde se considere necesario apreciar la diferenciación en la salida entre las tres modulaciones.

El orden siempre será de M menos a mayor, es decir M=2, M=4, M=8.

El resultado del muestreo con BitScope de la señal PRIME detectada es el que se representará es la que se observa en la Ilustración 25.

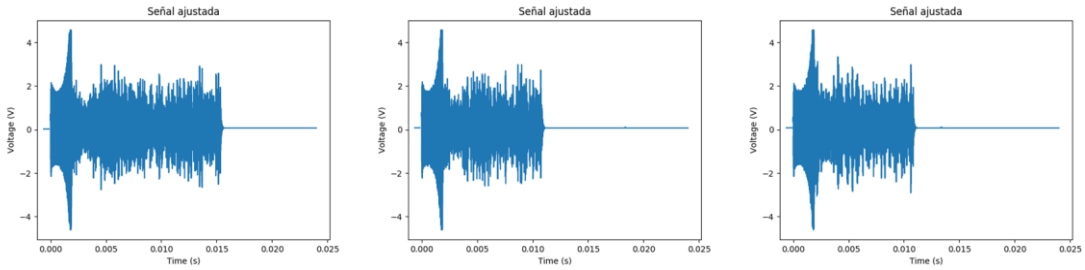


Ilustración 28. Señal PRIME

Se observa como la señal más larga para el mismo mensaje es la modulada con BPSK, esto es debido a que cada símbolo se corresponde con tan solo un bit, por que la modulación no optimiza la compresión en el mensaje, pero a cambio al estar los símbolos más alejados, mejora en sensibilidad frente a ruido.

La señal, a la vez, queda almacenada en un fichero de texto donde se guarda para poder ser utilizada, procesada y modificada tantas veces como se desee.

En la Ilustración 26 se pueden observar los ficheros de texto de cada una de las 3 señales (cada una de una modulación).

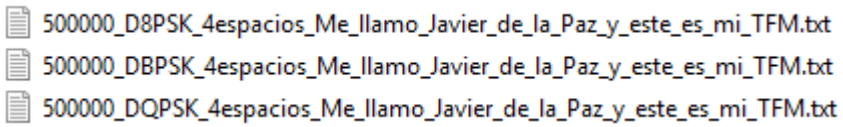


Ilustración 29. Ficheros de texto de las muestras almacenadas

Las muestras quedan guardadas en el formato que se observa en la Ilustración 27.

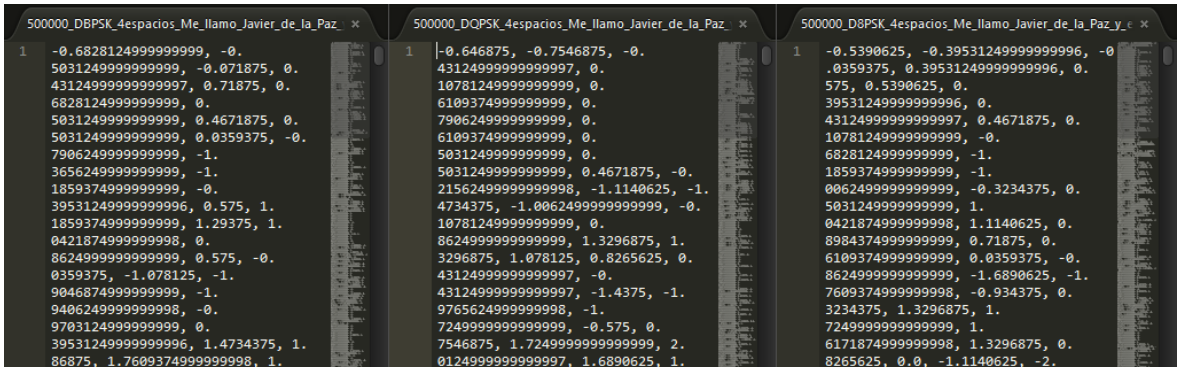


Ilustración 30. Formato de almacenamiento de las muestras

Una vez muestreada, se pasa a ejecutar el programa Python *ReceptorPRIME*. Y el resultado es el siguiente.

Lo primero que se hace es limpiar la señal. Para ello, se eliminan las primeras muestras vacías que no pertenecen a la señal, y queda como se muestra en la Ilustración 28.

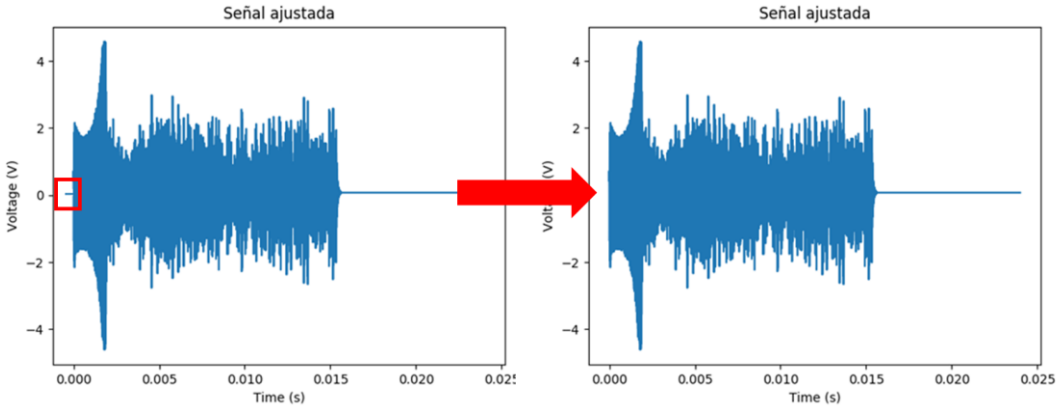


Ilustración 31. Señal original vs Señal ajustada

Tras limpiar la señal, el siguiente paso en el pre-procesado de la señal es identificar y separar las distintas partes de la señal.

Una vez llevado a cabo, se guarda el resultado que aparece en la Ilustración 29, en el cual se ven claramente las muestras pertenecientes a cada parte de la señal.

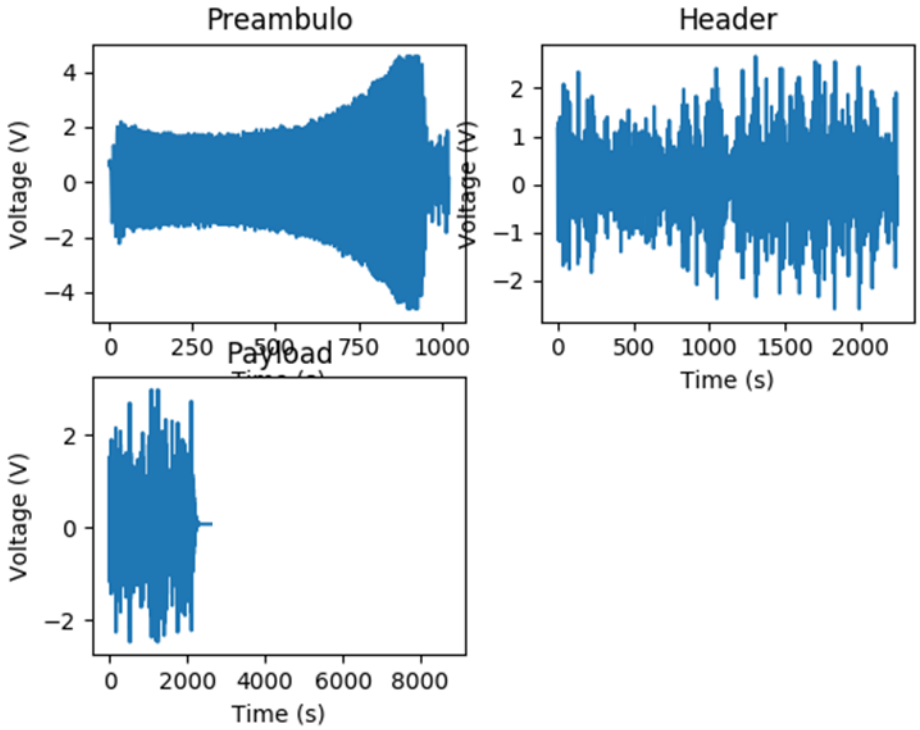


Ilustración 32. Preámbulo, header y payload

Se pasa por tanto ahora a la estimación del canal de transmisión y, como se ha comentado en el apartado de la Implementación, para ello, se ha utilizado el preámbulo de la señal PRIME.

El programa es capaz de generar una representación en frecuencia del mismo. Se puede observar, por tanto, que el canal estimado tiene la respuesta en frecuencia que se muestra en la ilustración 30.

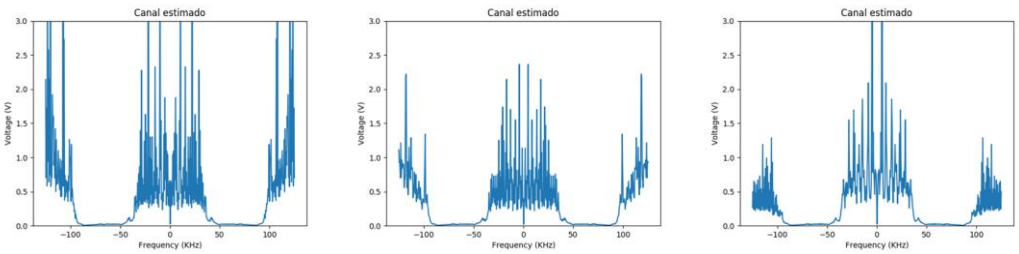


Ilustración 33. Canal estimado

Como se observa, es plano en la banda de frecuencias PRIME (41 – 89 kHz), como era de esperar.

El primer bloque es el modulador OFDM. Se va a analizar la salida, tanto en módulo como en fase gracias a la Ilustración 31.

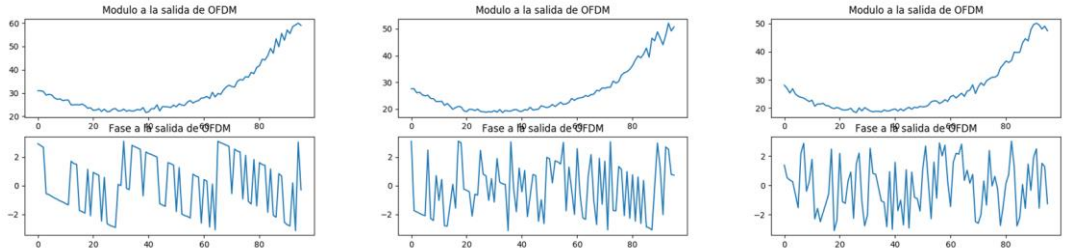


Ilustración 34. Modulo y fase a la salida de demodulador OFDM

Se puede ver como como en módulo, apenas se aprecian diferencias entre las tres modulaciones, mientras que en las diferencias de fase: en DBPSK hay 2 niveles, en DQPSK 4 y en D8PSK 8. Cada una de estas diferencias de fase se corresponden con un símbolo de la constelación que se observa en la Ilustración 32.

Como se comentaba, si representamos las subportadoras a la salida del demodulador OFDM, obtenemos la constelación gracias al scatterplot de la Ilustración 32.

También, gracias a la Ilustración 33, se puede observar la representación en frecuencia de las subportadoras, y como efectivamente coinciden los niveles altos de señal en la banda de frecuencias PRIME (de 41 – 89 kHz). En el resto de las frecuencias apenas se observa señal, con un mínimo ruido apreciable. Sí cabe resaltar un alto valor medio.

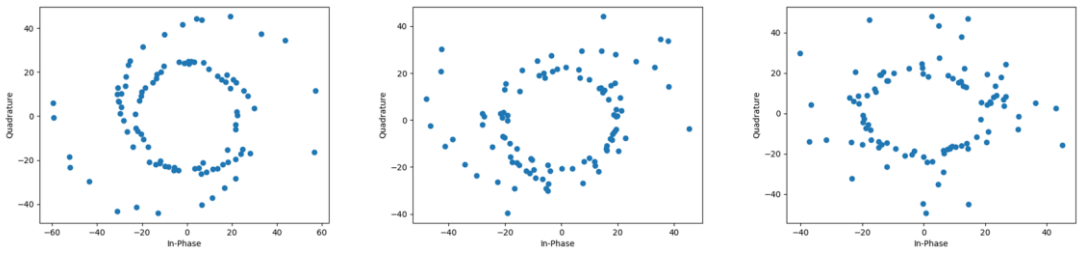


Ilustración 35. Constelación a la salida de la FFT

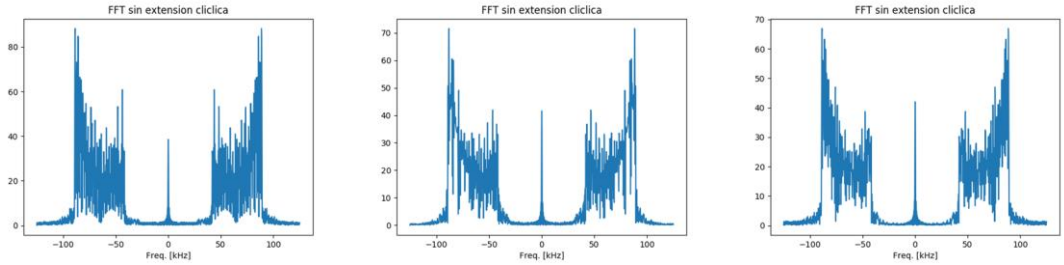


Ilustración 36. Respuesta en frecuencia a la salida de la FFT

Apenas se aprecian diferencias considerables en la respuesta en frecuencia de las tres modulaciones, sin embargo, se observa claramente qué modulación ha sido aplicada, si atendemos a la constelación resultante de la demodulación, como figura en la Ilustración 34.

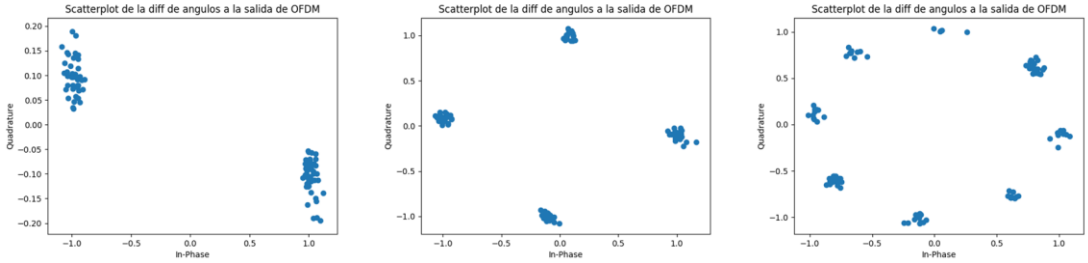


Ilustración 37. Constelación a la salida del demodulador digital

Se ve como en DBPSK solo hay dos símbolos en la constelación (desplazados debido a la falta de ecualización de la señal, aunque como el umbral para diferenciarlo está puesto en el eje imaginario, es decir, en si la parte real es positiva o negativa, esto no afectará a la correcta diferenciación entre símbolo), cuatro en DQPSK y ocho en D8PSK.

Si se observa la Ilustración 34, resultado real del sistema, y la Ilustración 23, ejemplo de un caso ideal, se aprecian los parecidos esperados de una señal correctamente demodulada con M=2, M=4, y M=8.

Para demostrar que en este caso la ecualización o no de la señal, no altera los resultados finales, se ha realizado una comparativa entre las diferencias de fase de un símbolo ecualizado y otro sin ecualizar de modulación DBPSK. Los resultados son los que se muestran en la Ilustración 35.

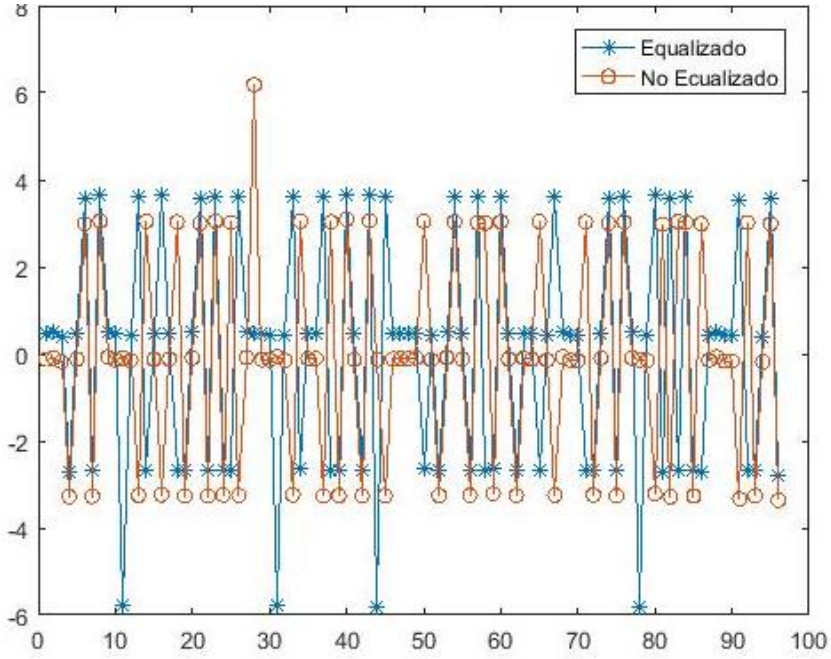


Ilustración 38. Ecualizado vs Sin Ecualizar

Se observa cómo son iguales en muchos valores, y en lo que no lo son, la diferencia de fase es la misma, ya que entre ellos hay una diferencia de 2π.

Por lo tanto, el resultado, en este caso mostrado, entre ecualizar y no ecualizar, no debería variar en absoluto.

Una vez hecha la demodulación digital, el siguiente paso es pasar por el srambler para eliminar la aleatorización llevada a cabo en la transmisión del mensaje PRIME. Sin embargo, el resultado de dicho proceso no es ilustrativo por lo que se ha decidido no añadir gráfica a la salida de dicho bloque del receptor.

Esto concluye por tanto el bloque principal "Receptor PRIME". Ya tenemos, por tanto, la salida en bits del mensaje transmitido.

Los dos siguientes pasos, son, de esta forma:

- Traducción UTF-8 a ASCII. De la forma explicada en el apartado de implementación, pasamos primero a número int y de aquí, al carácter asociado en el diccionario de UTF-8.
- Una vez se tiene el mensaje traducido, se almacena en un fichero de texto, con nombre "output", que contiene el mensaje enviado.

  En la Ilustración 36, se muestra el output del mensaje transmitido como ejemplo y que ha servido para realizar todo el apartado de Análisis de resultados.

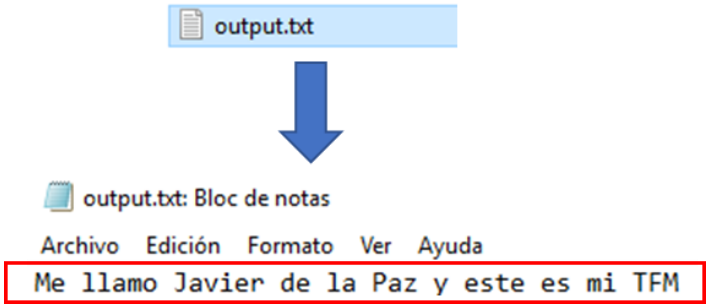


Ilustración 39. Output Traductor UTF-8

Coincide exactamente con el mensaje transmitido, por lo que se puede concluir que se ha realizado una recepción PRIME correcta y satisfactoria.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

7.1 CONCLUSIONES

Como resultado final de este proyecto se ha logrado un sistema capaz de hacer la recepción y traducción a caracteres ASCII de mensajes PRIME.

Se han logrado los objetivos establecidos inicialmente, con las siguientes tecnologías:

- Detectar y leer una señal PRIME
  - Hacer un muestreo digital de la señal (A/D) → BITSCOPE y BITLIB ✓
  - Guardar la señal. → TEXT FILE ✓
  - Tratar la señal. → PYTHON ✓
- Hacer la función de receptor PRIME
  - Estimar el canal. → PYTHON ✓
  - Devolver la salida de cada uno de los bloques del receptor. → PYTHON y NUMPY – Almacena las imágenes resultantes de la salida de cada bloque en una carpeta de imágenes según se van generando. ✓
  - Representación gráfica de los bits a su paso por los diferentes bloques. → PYTHON y MATPLOTLIB – Almacena las imágenes resultantes de la salida de cada bloque en una carpeta de imágenes según se van generando. ✓
  - Interpretar (traducir) el mensaje detectado. → PYTHON ✓
  - Almacenar texto ASCII de salida → FILE TEXT ✓

Por tanto, se puede concluir que se ha conseguido la implementación de un sniffer PRIME con BitScope, como dicta el título del proyecto.

Sin embargo, una vez llevado a cabo el proyecto, nos hemos topado con una limitación técnica con respecto al muestreador.

El BitScope, entre disparo y disparo, desde volver a hacer las fases ya explicadas de Trace y Acquire, lo que supone un tiempo aproximado de 100 ms entre capturas.

Esto, limita el funcionamiento del sistema, ya que, aunque es capaz de leer y traducir 12000 muestras cada 100 ms, no es capaz de hacer una escucha permanente, ya que durante un periodo se encuentra ocupado. Esto limita la función de sniffer, aunque se va a proponer una solución a dicho problema en trabajos futuros.

Además, se ha logrado un sistema barato (178€), open source, que proporciona información de los bits a lo largo de todo el sistema, que genera gráficas a la salida de cada bloque y sencilla de implementar (tanto en cuanto a complejidad electrónica se refiere).

7.2 TRABAJOS FUTUROS

Por otro lado, se busca que este proyecto no sea solo un fin en sí mismo, si n que ayude a fututos proyectos e investigaciones que puedan utilizar el sistema y los conocimientos de este trabajo como base o ayuda

Algunos posibles trabajos futuros derivados de este proyecto podrían ser:

- Utilizar varios BitScope para ir haciendo capturas secuenciales para ser capaces de muestrear más cantidad de símbolos OFDM.
  Debido a la limitación técnica de BitScope que impide muestrear de manera continua, se plantea la solución de hacer un sistema de varios BitScope de manera que, si muestrean de manera secuencial, se pueda llegar a resolver esta limitación técnica y llegar al objetivo de funcionar como realmente se esperaría de un sniffer en un canal de comunicación.
  Gracias a que la solución hardware de BitScope es muy económica, el sistema resultante seguiría con un precio muy inferior a los de mercado.

De este modo se acabaría con la limitación técnica aparecida, logrando con esto una sniffer con todas las funcionalidades del mismo.

- Utilización de esta herramienta para el análisis de la seguridad de las redes PLC.

También se pueden plantear proyectos de análisis, tanto de:

- Análisis del rendimiento de diferentes decodificadores convolucionales.
- Análisis del rendimiento de diferentes estrategias de ecualización para sistemas OFDM.

Capítulo 8. BIBLIOGRAFÍA

[1] Recommendation ITU-T G.9904 - Narrowband orthogonal frequency division multiplexing power line communication transceivers for PRIME networks. 10/2012

[2] de la Paz Garcillán, Javier (2017). *Implementación de un procesador digital de señal con BitScope.* Universidad Pontificia Comillas ICAI, Madrid, España

[3] PRIME Specification: PRIME-Spec_R1.3.6.1, 2014.

[4] SAM4CP16B Datasheet, 2015.

[5] PLC Coupling Reference Designs, 2015.

REFERENCIAS WIKIPEDIA

[6] PRIME (PLC). En Wikipedia. Recuperado el 24 de marzo de 2019 de https://en.wikipedia.org/wiki/PRIME_(PLC)

[7] Conversión analógica - digital. (s.f.). En Wikipedia. Recuperado el 01 de junio de 2019 de https://es.wikipedia.org/wiki/Conversi%C3%B3n_anal%C3%B3gica-digital

[8] Procesamiento digital de señales. (s.f.). En Wikipedia. Recuperado el 01 de junio de 2019 de https://es.wikipedia.org/wiki/Procesamiento_digital_de_se%C3%B1ales

[9] Muestreo digital. (s.f.). En Wikipedia. Recuperado el 01 de junio de 2019 de https://es.wikipedia.org/wiki/Muestreo_digital

[10] FFT. (s.f.). En Wikipedia. Recuperado el 01 de junio de 2019 de https://es.wikipedia.org/wiki/Transformada_r%C3%A1pida_de_Fourier

[11] Modulación (telecomunicación) (s.f.). En Wikipedia. Recuperado el 02 de junio de 2019 de https://es.wikipedia.org/wiki/Modulaci%C3%B3n_(telecomunicaci%C3%B3n)

[12] Multiplexación por división de frecuencias ortogonales (s.f.). En Wikipedia. Recuperado el 02 de junio de 2019 de https://es.wikipedia.org/wiki/Multiplexaci%C3%B3n_por_divisi%C3%B3n_de_frecuencias_ortogonales

[13] Python (s.f.). En Wikipedia. Recuperado el 09 de junio de 2019 de https://es.wikipedia.org/wiki/Python

[14] UTF-8 (s.f.). En Wikipedia. Recuperado el 13 de junio de 2019 de https://es.wikipedia.org/wiki/UTF-8

GUIAS DE USO

[15] Atmel PRIME Firmware Stack User Guide, 2015.

REFERENCIAS DE PRODUCTOS OFICIALES

[16] Página Oficial de BitScope - http://www.bitscope.org/

[17] Página Oficial Python 2.7 - https://www.python.org/download/releases/2.7/

[18] Página Oficial PLC PHY Tester http://ww1.microchip.com/downloads/en//softwarelibrary/se_plc_phy_tester_tool/50002734.pdf

APIS

[19] BitLib - http://www.bitscope.com/software/library/API.html

[20] NumPy - https://docs.scipy.org/doc/numpy-1.12.0/reference/

[21] PyQt4 - http://pyqt.sourceforge.net/Docs/PyQt4/

[22] PyQtGraph - http://www.pyqtgraph.org/documentation/apireference.html

PROYECTOS RELACIONADOS

[23] ZIV - https://www.zivautomation.com/es/metering-solutions/

[24] ORMAZABAL – https://www.ormazabal.com/en/your-business/products/advanced-metering

[25] Microchip – https://www.microchip.com/wwwproducts/en/ATPL230A

ANEXO A. CÓDIGO FUENTE

SignalCapture.py

```
import sys
from bitlib import *
from sampler import Sampler
from PyQt4 import QtGui
import pyqtgraph as pg
import numpy as np
import fileinput

#Frecuencia de muestreo 500 KHz (luego diezmar 1/2)
MY_RATE = 500e3
#Coger máximo tamaño de buffer
MY_SIZE = 12000
#Puntero a Sampler
samp = Sampler(MY_SIZE,MY_RATE)
#Inicializar BitScope
samp.open_scope()

#Disparo
samp.scope_acquire()
ch1_data = list()
#Coger señal guardada en buffer
ch1_data = samp.getLastSample()

#Guardar señal en fichero de texto
with open("PRIMEsignal.txt","w") as output:
                    output.write(str(ch1_data))

# Eliminar los corchetes del fichero de texto para su posterior lectura de manera
correcta
with open('PRIMEsignal.txt', 'r') as file :
  filedata = file.read()

filedata = filedata.replace('[', '')
filedata = filedata.replace(']', '')

with open('PRIMEsignal.txt', 'w') as file:
  file.write(filedata)


#Representación gráfica de la señal capturada
app = QtGui.QApplication([])
win = pg.GraphicsWindow(title="Basic plotting examples")
win.resize(1000,600)
win.setWindowTitle('pyqtgraph example: Plotting')
```

```
pg.setConfigOptions(antialias=True)
p6 = win.addPlot(title="Updating plot")
curve = p6.plot(pen='b')
curve.setData(ch1_data)

sampler.close_scope()


if __name__ == '__main__':
    if (sys.flags.interactive != 1) or not hasattr(QtCore, 'PYQT_VERSION'):
        QtGui.QApplication.instance().exec_()
```

Sampler.py [2]

```
from bitlib import *
from PyQt4 import QtCore
import numpy as np

class Sampler(object):

    def __init__(self,MY_SIZE,MY_RATE):
        self.first = True
        self.ch1_data = np.zeros(MY_SIZE)
        self.MY_RATE = MY_RATE
        self.MY_SIZE = MY_SIZE
        self.TRIGGER_VALUE = 0.05

    #Abrir bitscope
    def open_scope(self):

        if BL_Open("", 1):

            MY_DEVICE = 0 # one open device only
            MY_CHANNEL = 0 # channel to capture and display
            MY_PROBE_FILE = "" # default probe file if unspecified
            MY_MODE = BL_MODE_FAST # preferred trace mode
            TRUE = 1

            MODES = ("FAST","DUAL","MIXED","LOGIC","STREAM")
            SOURCES = ("POD","BNC","X10","X20","X50","ALT","GND")

            print " Library: %s (%s)" % (
            BL_Version(BL_VERSION_LIBRARY),
            BL_Version(BL_VERSION_BINDING))

            BL_Select(BL_SELECT_DEVICE,MY_DEVICE)

            print "    Link: %s" % BL_Name(0)
            print "BitScope: %s (%s)" % (BL_Version(BL_VERSION_DEVICE),BL_ID())
            print "Channels: %d (%d analog + %d logic)" % (
```

```
            BL_Count(BL_COUNT_ANALOG)+BL_Count(BL_COUNT_LOGIC),
            BL_Count(BL_COUNT_ANALOG),BL_Count(BL_COUNT_LOGIC))

            print "   Modes:" + "".join(["%s" % (
                (" " + MODES[i]) if i == BL_Mode(i) else "") for i in
range(len(MODES))])

            BL_Mode(BL_MODE_LOGIC) == BL_MODE_LOGIC or BL_Mode(BL_MODE_FAST)

            BL_Range(BL_Count(BL_COUNT_RANGE));
            if BL_Offset(-1000) != BL_Offset(1000):
                print "  Offset: %+.4gV to %+.4gV" % (    BL_Offset(1000),
BL_Offset(-1000))

            for i in range(len(SOURCES)):
                if i == BL_Select(2,i):
                    print "     %s: " % SOURCES[i] + " ".join(["%5.2fV" %
BL_Range(n) for n in range(BL_Count(3)-1,-1,-1)])

            BL_Mode(MY_MODE) # prefered trace mode
            BL_Intro(BL_ZERO); # optional, default BL_ZERO
            BL_Delay(BL_ZERO); # optional, default BL_ZERO
            BL_Rate(self.MY_RATE); # optional, default BL_MAX_RATE
            BL_Size(self.MY_SIZE); # optional default BL_MAX_SIZE
            BL_Select(BL_SELECT_CHANNEL,MY_CHANNEL); # choose the channel
            BL_Trigger(self.TRIGGER_VALUE,BL_TRIG_RISE); # optional when
untriggered */
            BL_Select(BL_SELECT_SOURCE,BL_SOURCE_POD); # use the POD input */
            BL_Range(BL_Count(BL_COUNT_RANGE)); # maximum range
            BL_Offset(BL_ZERO); # optional, default 0
            BL_Enable(TRUE); # at least one channel must be initialised


        return


    #Cerrar bitscope
    def close_scope(self):

        BL_Close()

        return


    #Coger muestras
    def scope_acquire(self):
        BL_Trigger(self.TRIGGER_VALUE,BL_TRIG_RISE);
        BL_Trace(BL_TRACE_FOREVER, False)

        self.ch1_data = BL_Acquire()
```

```
    #Coger array de las ultimas muestras guardadas
    def getLastSample(self):
        return self.ch1_data

    #Puntero a UI
    def setUI(self,UIpointer):
        self.UIpointer = UIpointer

    #Cambiar numero de puntos
    def setSize(self,SIZE):
        self.MY_SIZE = SIZE

    #Cambiar frecuencia de muestreo
    def setRate(self,rate):
        self.MY_RATE = rate

    #Devuelve la frecuencia de muestreo
    def getRate(self):
        return self.MY_RATE

    #Cambiar nivel del trigger
    def setTiggerValue(self, triggerValue):
        self.TRIGGER_VALUE = float(triggerValue)

    #Devuelve el nivel del trigger
    def getTiggerValue(self):
        return self.TRIGGER_VALUE
```

ReceptorPRIME.py

```
#!/usr/bin/python
# coding: utf-8
import sys
import numpy as np
import matplotlib.pyplot as plt
import math
import numpy.matlib


m_ary = 4
k = math.log(m_ary,2)
N_sym_OFDM_cod = 1

NFFT = 512
Ncyc = 48
L = 2 # Oversampling
FsPRIME = 250e3
```

```
Fs = FsPRIME * L

x =
np.genfromtxt("500000_DQPSK_4espacios_Me_llamo_Javier_de_la_Paz_y_este_es_mi_TFM.
txt",delimiter=",") #Cargo la señal
t = np.linspace(0, (len(x)-1)/Fs, len(x), endpoint=True, retstep=False,
dtype=None) # Vector de tiempo - (n° de muestras)/(Fs) = segundos


ind = np.where(x>0.15) #Nos quedamos con los índices de los valores de señal con
Voltaje>0.15V de modo que detectemos cuando empieza a haber señal
ind = np.asarray(ind) #Nos quedamos con los valores de señal con Voltaje>0.15V
ind = ind[0] # Quitar los doble corchetes

x = x[ind[0]:len(x)] # x pasa a ser una nueva señal - la inicial sin la parte de
ceros del comienzo
t = np.linspace(0, (len(x)-1)/Fs, len(x), endpoint=True, retstep=False,
dtype=None) # Nuevo vector de tiempos para nuevo tamaño de señal


plt.figure()
plt.plot(t,x)
plt.xlabel('Time (s)')
plt.ylabel('Voltage (V)')
plt.title(u'Señal ajustada') #Señal sin muestras con O V al principio
plt.savefig('Images\cleared_signal.png')


preamb = x[1*L/2-1:1024*L/2]
header = x[1024*L/2:1024*L/2+1+(NFFT+Ncyc)*L*2-1]
N_symb = math.floor((len(x)-(1024*L/2+1+(NFFT+Ncyc)*L*2))/(560*L))
payload = x[1024*L/2+(NFFT+Ncyc)*L*2:len(x)]

plt.figure()
plt.subplot(2,2,1)
plt.plot(preamb)
plt.xlabel('Time (s)')
plt.ylabel('Voltage (V)')
plt.title(u'Preambulo')
plt.subplot(2,2,2)
plt.plot(header)
plt.xlabel('Time (s)')
plt.ylabel('Voltage (V)')
plt.title(u'Header')
plt.subplot(2,2,3)
plt.plot(payload)
plt.xlabel('Time (s)')
plt.ylabel('Voltage (V)')
plt.title(u'Payload')
plt.savefig('Images\pream_header_payload.png')
```

```
hope = np.linspace(0, len(preamb)-4,math.floor(len(preamb)/L)) # El hope nos
permite quitar el sobremuestreo
hope = hope.astype(int)
Preamb = np.fft.fftshift(np.fft.fft(preamb[hope]))

H_hat = np.ones(NFFT)/Preamb # Canal estimado
f = np.linspace(-0.5,0.5,NFFT)*FsPRIME # Línea de frecuencia

plt.figure()
plt.plot(f*1e-3,np.abs(H_hat)) # Representa el canal estimado
plt.xlabel('Frequency (KHz)')
plt.ylabel('Voltage (V)')
plt.title(u'Canal estimado')
plt.ylim(top=3)
plt.ylim(bottom=0)
plt.savefig('Images\canalEstimado.png')

f_diez = np.linspace(-.5,.5,(NFFT))*Fs/L

ind_data_pilots = np.where((f_diez>=42e3) & (f_diez<89.5e3))
ind_data_pilots = np.asarray(ind_data_pilots)
ind_data_pilots = ind_data_pilots[0]

matrix_payload = np.zeros((int(N_symb),96*k))

dig_symbols_diff = np.zeros((96*k),dtype=np.complex_)
iteracion = 0
iteracion2 = 1

while (iteracion2 < N_symb):

      dig_symbols_diff_iter = 1

      # DeModulación OFMD
      simbolo_i = payload[iteracion:560*L*iteracion2] # Simbolo i
      x_diez_cyc = simbolo_i[L-1:len(payload):L] # Diezmo para deshacer el
efecto del sobre-muestreo
      x_diez = x_diez_cyc[0:NFFT] # Quito extensió cíclica
      X_diez  = np.fft.fftshift(np.fft.fft(x_diez)) # FFT
      data_pilots = X_diez[ind_data_pilots[0]:len(X_diez)]
      data_pilots = data_pilots[0:97] # Portadoras
      dig_symbols = X_diez[ind_data_pilots[1:97]] # Piloto
      piloto = X_diez[ind_data_pilots[0]]

      if iteracion2 == 1:
            plt.figure()
            f_diez_3 = np.linspace(-.5,.5,(NFFT*3))*Fs/L
            plt.plot(f_diez_3*1e-
3,np.abs(np.fft.fftshift(np.fft.fft(np.append(x_diez,np.zeros(len(x_diez)*2))))))
            plt.xlabel('Freq. [kHz]')
            plt.title('FFT sin extension cliclica')
            plt.savefig('Images\salidaOFDM.png')
```

```
            plt.figure()
            plt.scatter(dig_symbols.real,dig_symbols.imag)
            plt.xlabel('In-Phase')
            plt.ylabel('Quadrature')
            plt.savefig('Images\FFT.png')

            plt.figure()
            plt.subplot(2,1,1)
            plt.plot(np.abs(dig_symbols))
            plt.title(u'Modulo a la salida de OFDM')
            plt.subplot(2,1,2)
            plt.plot(np.angle(dig_symbols))
            plt.title(u'Fase a la salida de OFDM')
            plt.savefig('Images\phase_module.png')

      dig_symbols = dig_symbols*np.exp(-1j*np.angle(piloto)) # Efecto del piloto

      rxBits2 = []
      lenSymb = len(dig_symbols)
      # DeModulación digital
      # DBPSK, DQPSK, D8PSK
      if m_ary==2:
            i=0
            angulo = np.angle(dig_symbols[i])
            if ((angulo < math.pi/2 and angulo> -math.pi/2)):
                  binaryWord = [0]
            else:
                  binaryWord = [1]
            rxBits2 = rxBits2+binaryWord
            while (i < lenSymb-1):
                  i = i+1
                  angulo_pre = np.angle(dig_symbols[i-1])
                  angulo = np.angle(dig_symbols[i])
                  angulo_diff = angulo_pre - angulo
                  dig_symbols_diff[dig_symbols_diff_iter] =
(dig_symbols[i])/(dig_symbols[i-1])
                  dig_symbols_diff_iter = dig_symbols_diff_iter +1
                  if angulo_diff>3.53:
                        angulo_diff = angulo_diff -2*math.pi
                  if angulo_diff<-3.53:
                        angulo_diff = angulo_diff +2*math.pi
                  if (angulo_diff<math.pi/4 and angulo_diff>-math.pi/4):
                        binaryWord = [0]
                  else:
                        binaryWord = [1]
                  rxBits2 = rxBits2+binaryWord
      elif m_ary==4:
            i=0
            angulo = np.angle(dig_symbols[i])
            if ((angulo < math.pi/4 and angulo>-math.pi/4)):
                  binaryWord = [0,0]
            elif (angulo < 3*math.pi/4 and angulo> math.pi/4):
                binaryWord = [0,1]
```

```
            elif (angulo > 3*math.pi/4 or angulo< -3*math.pi/4):
                binaryWord = [1,1]
            elif (angulo < -math.pi/4 and angulo> -3*math.pi/4):
                  binaryWord = [1,0]
            rxBits2 = rxBits2+binaryWord
            while (i < lenSymb-1):
                  i = i+1
                  angulo_pre = np.angle(dig_symbols[i-1])
                  angulo = np.angle(dig_symbols[i])
                  angulo_diff = angulo_pre - angulo
                  dig_symbols_diff[dig_symbols_diff_iter] =
(dig_symbols[i])/(dig_symbols[i-1])
                  dig_symbols_diff_iter = dig_symbols_diff_iter +1
                  if angulo_diff>3.53:
                        angulo_diff = angulo_diff -2*math.pi
                  if angulo_diff<-3.53:
                        angulo_diff = angulo_diff +2*math.pi
                  if ((angulo_diff < math.pi/4 and angulo_diff>-math.pi/4)):
                      binaryWord = [0,0]
                  elif ((angulo_diff > 3*math.pi/4 and angulo_diff<4) or
angulo_diff < -3*math.pi/4):
                      binaryWord = [1,1]
                  elif (angulo_diff < -math.pi/4 and angulo_diff > -
3*math.pi/4):
                      binaryWord = [0,1]
                  elif (angulo_diff >math.pi/4 and angulo_diff < 3*math.pi/4):
                        binaryWord = [1,0]
                  rxBits2 = rxBits2+binaryWord
      elif m_ary==8:
            i=0
            angulo = np.angle(dig_symbols[i])
            if (angulo < math.pi/8 and angulo>-math.pi/8):
                  binaryWord = [0,0,0]
            elif (angulo < 3*math.pi/8 and angulo> math.pi/8):
                binaryWord = [0,0,1]
            elif (angulo < 5*math.pi/8 and angulo> 3*math.pi/8):
                binaryWord = [0,1,1]
            elif (angulo < 7*math.pi/8 and angulo> 5*math.pi/8):
                  binaryWord = [0,1,0]
            elif (angulo < -7*math.pi/8 or angulo> 7*math.pi/8):
                binaryWord = [1,1,0]
            elif (angulo > -3*math.pi/8 or angulo < -math.pi/8):
                binaryWord = [1,0,0]
            elif (angulo < -3*math.pi/8 and angulo> -5*math.pi/8):
                  binaryWord = [1,0,1]
            elif (angulo < -5*math.pi/8 and angulo> -7*math.pi/8):
                  binaryWord = [1,1,1]
            rxBits2 = rxBits2+binaryWord
            while (i < lenSymb-1):
                  i = i+1
                  angulo_pre = np.angle(dig_symbols[i-1])
                  angulo = np.angle(dig_symbols[i])
                  angulo_diff = angulo_pre - angulo
```

```
                    dig_symbols_diff[dig_symbols_diff_iter] =
(dig_symbols[i])/(dig_symbols[i-1])
                    dig_symbols_diff_iter = dig_symbols_diff_iter +1
                    if angulo_diff>3.53:
                           angulo_diff = angulo_diff -2*math.pi
                    if angulo_diff<-3.53:
                           angulo_diff = angulo_diff +2*math.pi
                    if ((angulo_diff < math.pi/8 and angulo_diff>-math.pi/8)):
                        binaryWord = [0,0,0]
                    elif (angulo_diff > math.pi/8 and angulo_diff < 3*math.pi/8):
                        binaryWord = [1,0,0]
                    elif (angulo_diff > 3*math.pi/8 and angulo_diff <
5*math.pi/8):
                        binaryWord = [1,0,1]
                    elif (angulo_diff > 5*math.pi/8 and angulo_diff <
7*math.pi/8):
                           binaryWord = [1,1,1]
                    elif (angulo_diff > 7*math.pi/8 or angulo_diff < -
7*math.pi/8):
                        binaryWord = [1,1,0]
                    elif (angulo_diff < -math.pi/8 and angulo_diff > -
3*math.pi/8):
                        binaryWord = [0,0,1]
                    elif (angulo_diff < -3*math.pi/8 and angulo_diff > -
5*math.pi/8):
                           binaryWord = [0,1,1]
                    elif (angulo_diff < -5*math.pi/8 and angulo_diff > -
7*math.pi/8):
                           binaryWord = [0,1,0]
                    rxBits2 = rxBits2+binaryWord

      rxData=rxBits2
      matrix_payload[iteracion2-1,:] = rxData
      iteracion2 = iteracion2 +1
      iteracion=iteracion+1120
      if iteracion2 ==2:
            plt.figure()
            plt.scatter(dig_symbols_diff.real,dig_symbols_diff.imag)
            plt.xlabel('In-Phase')
            plt.ylabel('Quadrature')
            plt.title(u'Scatterplot de la diff de angulos a la salida de OFDM')
            plt.savefig('Images\diff_angle_'+str(iteracion2)+'.png')


rxData = np.reshape(matrix_payload,-1) # Pasar de paralelo a serie

#Scrambler
scr
=[0,0,0,0,1,1,1,0,1,1,1,1,0,0,1,0,1,1,0,0,1,0,0,1,0,0,0,0,0,0,1,0,0,0,1,0,0,1,1,0
,0,0,1,0,1,1,1,0,1,0,1,1,0,1,1,0,0,0,0,0,1,1,0,0,1,1,0,1,0,1,0,0,1,1,1,0,0,1,1,1,
1,0,1,1,0,1,0,0,0,0,1,0,1,0,1,0,1,1,1,1,1,0,1,0,0,1,0,1,0,0,0,1,1,0,1,1,1,0,0,0,1
,1,1,1,1,1,1]
scr = np.asarray(scr)
```

```
scr = np.append(scr,scr)
scr = np.append(scr,scr)
scr = np.append(scr,scr)
scr = np.append(scr,scr)
scr = np.append(scr,scr)
scr = np.append(scr,scr)
scr = np.append(scr,scr)
scr = np.append(scr,scr)
scr = np.append(scr,scr)

rxData = rxData.astype(int)
rxBits = np.bitwise_xor(rxData,scr[168:169+len(rxData)-1])


words = np.reshape(rxBits, (len(rxBits)/8,8))

# Traductor
iter = -1
string = ""
while iter < len(words)-1:
    iter = iter +1
    aux = str(words[iter,0:8])
    aux = aux[1]+aux[3]+aux[5]+aux[7]+aux[9]+aux[11]+aux[13]+aux[15]
    aux = int(aux,2)
    aux = chr(aux)
    string = string + aux

# Guardar en fichero de texto
with open("output.txt","w") as output:
            output.write(string)
```