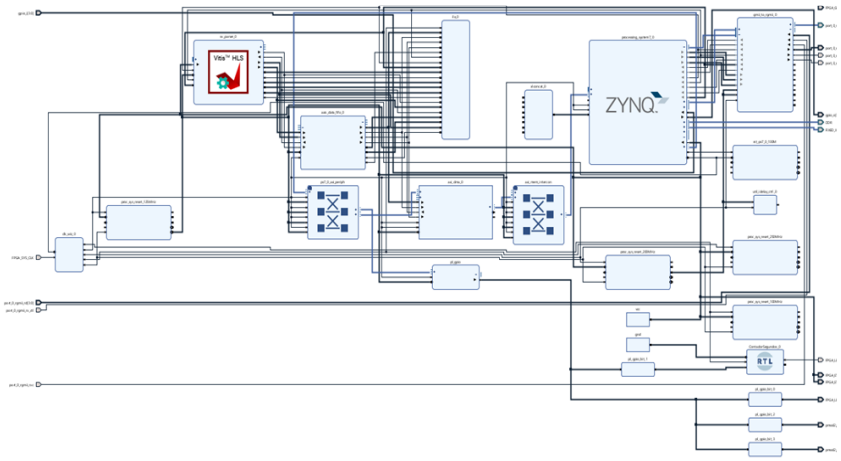
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIONES

TRABAJO FIN DE MASTER

DESARROLLO DE UN SISTEMA DE COMUNICACIONES PARA SMART GRIDS



Estudiante: Santaolalla Sánchez, Iñigo

Director/Directora: Aranguren Aramendía, Gerardo
Codirector/Codirectora:

Curso: 2023-2024

Fecha: Bilbao, 19, febrero, 2024

Resumen Castellano

En el presente trabajo se muestra una propuesta de un nuevo sistema de comunicaciones para que funcione en una tarjeta electrónica de comunicaciones. La finalidad de esta propuesta es la mejora de los equipos de protección de red, llamados GPG (Generic Power Gateway), que se encuentran en las Smart Grids españolas. Para ello, se han utilizado diversas herramientas de descripción de hardware tales como Vivado, Vitis o Vitis HLS para diseñar el sistema de comunicaciones.

Palabras clave: IOT, Industria 4.0, TIC, Smart Grids y SoC

Resumen Euskera

Lan honetan komunikazio-sistema berri baten proposamen bat erakusten da, komunikazio-txartel elektroniko batean funtziona dezan. Proposamen honen helburua da Espainiako Smart Grids-etan dauden GPG (Generic Power Gateway) izeneko sarea babesteko ekipoak hobetzea. Horretarako, hardwarea deskribatzeko hainbat tresna erabili dira, hala nola Vivado, Vitis edo Vitis HLS komunikazio-sistema diseinatzeko.

Hitz gakoak: IOT, Industria 4.0, TIC, Smart Grids eta I+D

Resumen Inglés

This work presents a proposal of a new communications system that works on an electronic communications card. The purpose of this proposal is to improve the network protection equipment, called GPG (Generic Power Gateway), which is found in Spanish Smart Grids. To this end, various hardware description tools such as Vivado, Vitis or Vitis HLS have been used to design the communications system.

Key words: IOT, Industria 4.0, TIC, Smart Grids and I+D

Tabla de contenido

Lista de tablas

Lista de Figuras

1. Memoria

1.1 Introducción

Cada vez más, gracias a la IoT, la tecnología se está transformando en un proceso de manejo de gran cantidad de información de una forma rápida y accesible. Un caso de aplicación de la filosofía IoT o Industria 4.0 en las redes eléctricas son las ya conocidas Smart Grids. Las Smart Grids o redes inteligentes, son la aplicación por un lado de los fundamentos de la electricidad, y por otro, de las tecnologías de la información (TIC) para obtener datos de las propias redes.

Los datos obtenidos, pueden servir para realizar tareas de monitorización de la red eléctrica, y consecuentemente, corregir faltas que se produzcan en la misma. En este escenario es muy importante llevar a cabo una importante labor de estudio para que se aplique de forma correcta la tecnología, por lo que la I+D tendrá un papel importante. Para ello, será necesario realizar una labor de programación de una tarjeta con un chip SoC.

Por un lado, llevaré a cabo una introducción de donde se sitúa este trabajo dentro de la empresa en que se ha desarrollado. Y, por otro lado, realizaré una descripción de la solución llevada a cabo para cumplir con la funcionalidad de mi sistema. Para ello, nos hemos basado en el uso de diferentes herramientas:

Primeramente, he empleado el programa Vivado (perteneciente a Xilinx) para realizar el diseño del hardware (HW) del sistema. Y, por otro lado, he utilizado la herramienta Vitis, también de Xilinx, de diseño software, para realizar el programa que correrá el procesador que tenga nuestra tarjeta de comunicaciones.

Y, por último, con todo montado se procede a realizar las pruebas hasta llegar a una solución concluyente.

1.2 Contexto

Este TFM se ha desarrollado en la fábrica que tiene la empresa General Electric en Zamudio, Figura 1.

Figura 1. Imagen exterior de la fábrica

En particular, este trabajo se ha realizado en el departamento denominado AAA (Advanced Automation Application). Este departamento de ingeniería se centra en el desarrollo de IEDs (Inteligent Electronic Devices).

En la *Figura 2* se muestra la localización de los IED en la red eléctrica, estos se sitúan entre las centrales de generación y la red de transporte.

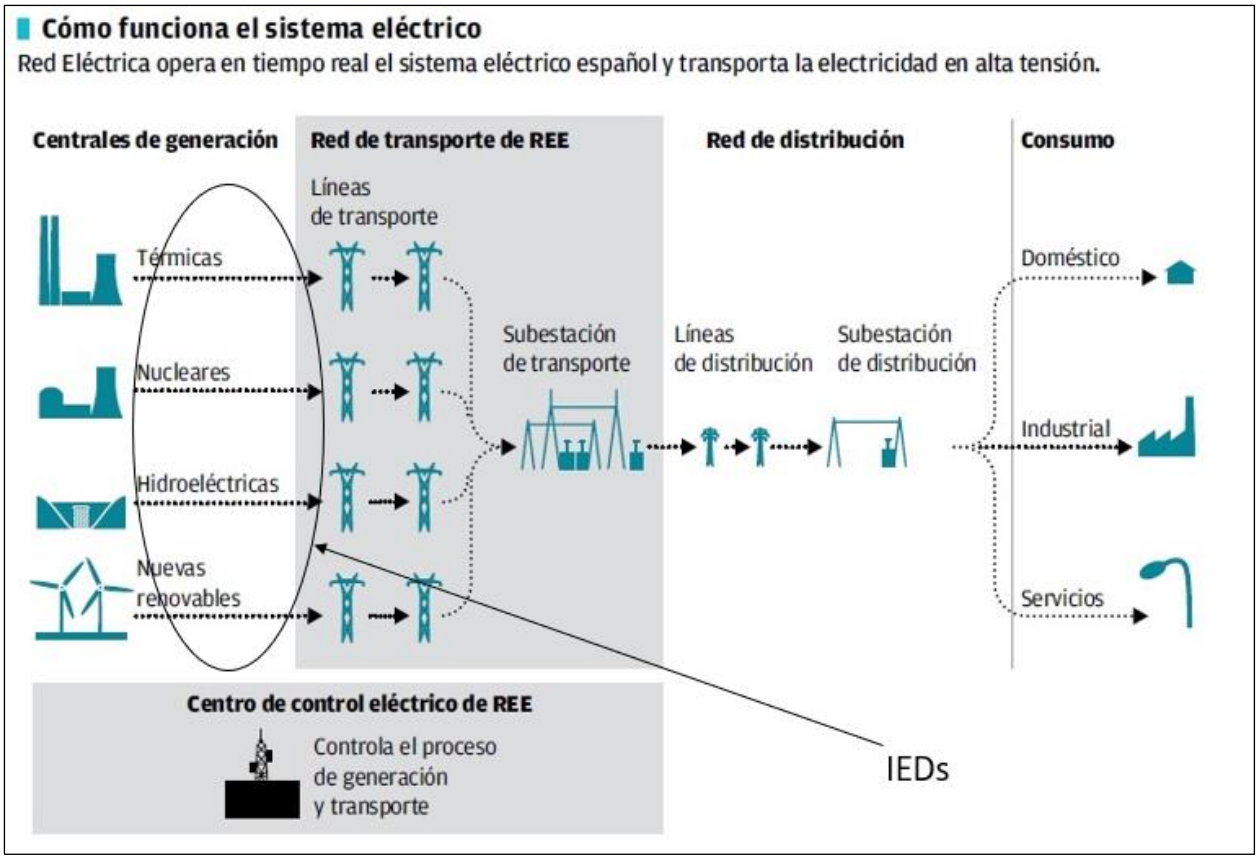


Figura 2. Estructura genérica de red eléctrica [1]

En la *Figura 3* se puede ver el emplazamiento de los IED en la línea que une las centrales de generación con la red de transporte de REE, además se puede apreciar la existencia de otros equipos de protección de red. A continuación, se realiza una breve descripción de la funcionalidad de cada uno de ellos:

1. MU (Merging Unit): Equipo encargado de tomar medidas de corriente y tensión de las 4 fases (R, S, T y N). Estas medidas las manda al IED en formato IEC61850.

2. Breaker IED: Relés encargados de abrir y cerrar la línea de tensión en caso de que se produzca una falta en la red eléctrica.

3. IED: Equipo encargado de analizar tramas que llegan de los diferentes MU a los que esté conectado, activando o desactivando los breaker IED en caso de que se produzca una sobretensión o una sobrecorriente.

4. Centro de Control: Sitio donde se presenta el estado actual de la red gracias a la información que llega de los IED. Realiza tareas tanto de supervisión como de gestión de la propia red.

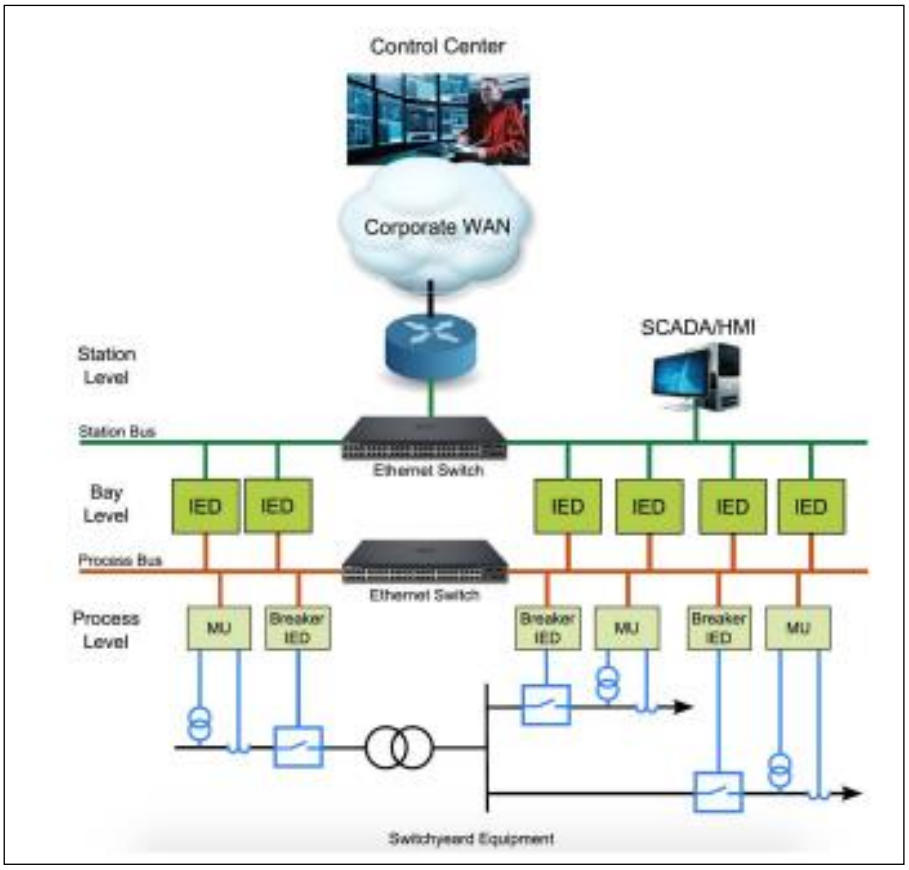


Figura 3. Diagrama de red de una parte de la red eléctrica [2]

El nombre comercial de los IED es GPG (GE Power Gateway). En la *Figura 4* viene representado el aspecto externo de un GPG.

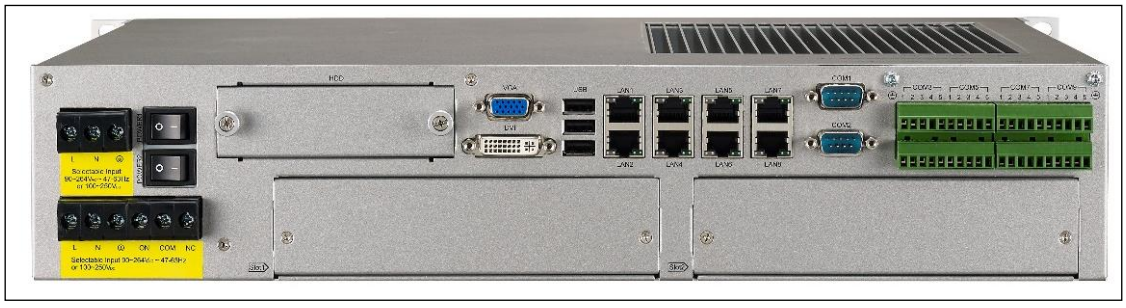


Figura 4. Vista frontal de un GPG

La arquitectura interna del GPG viene representada según la *Figura 5*.

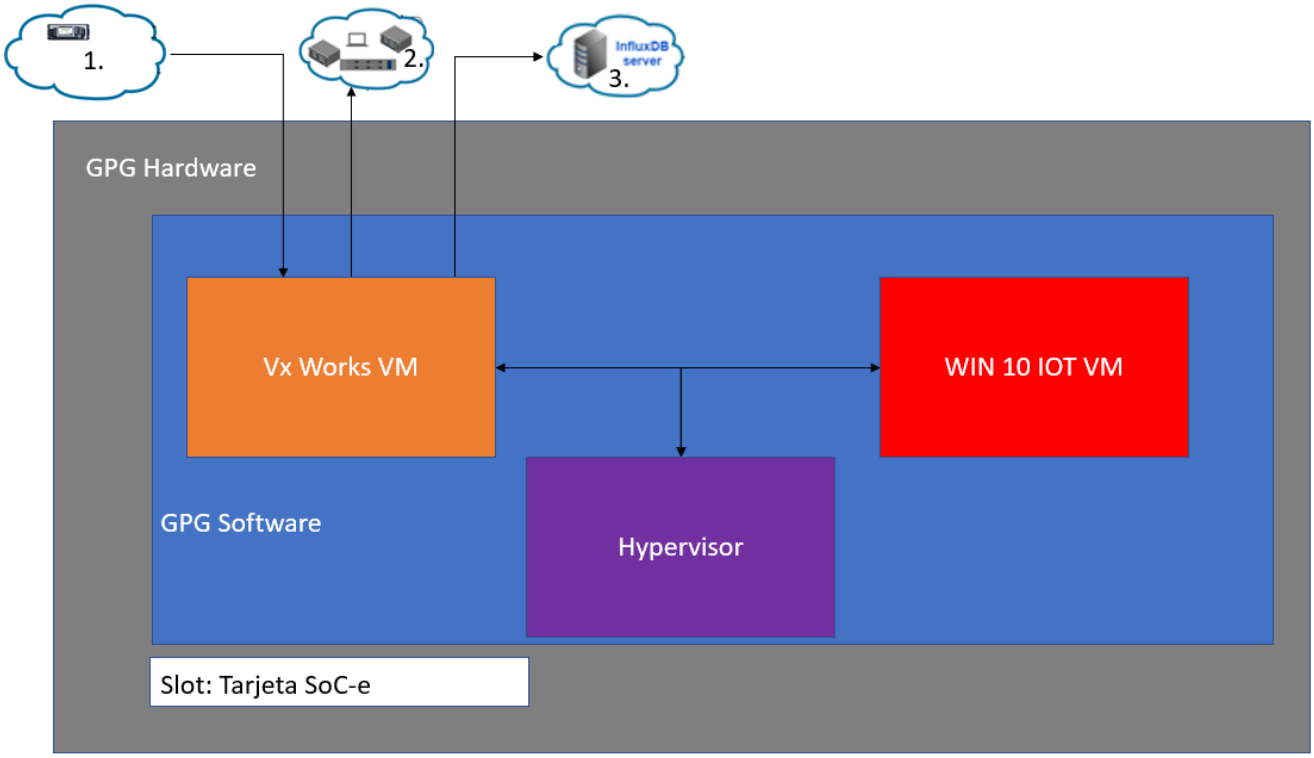


Figura 5. Arquitectura interna del GPG

Se pueden destacar principalmente 4 partes:

1. VxWorks VM. Máquina virtual que se encarga de realizar 3 tareas:
    i. Recibir muestras de corriente y tensión de los MU (1) y decodificarlas.
    ii. Mandarla a una base de datos (2) para almacenar estadísticos.
    iii. Procesar las muestras recibidas para que en caso de fallo en la red eléctrica se actúe sobre los relés (3).
2. WIN 10 IOT VM. Máquina virtual de GE con los diferentes dispositivos hardware virtualizados. Se encarga de almacenar información lógica (direcciones IP, MAC …) de los breaker IED, del IED y de los Merging Unit.
3. Hipervisor. Permite tener más de una máquina virtual dentro del mismo equipo físico.
4. GPG Hardware. Es la electrónica de los IEDs, pertenece a la empresa Advantech y el modelo es el ECU-4784. En la *Figura 6* se aprecia una descripción técnica del mismo.

Hardware	
CPU	6th Gen. Intel® Core™ i5-6300U 2.4GHz Dual-Core
Memory	2 x DDR4 SO-DIMM Slot
Dual Power Input	Supported
Power Input	100~240 VAC, 100~240 VDC
LAN	8 x 10/100/1000 Mbps
Ethernet	Intel® i210-IT GbE, 802.1Qav, IEEE1588/802.1AS, 802.3az
I/O	2 x RS-232, 8 x RS-232/422/485
Storage	3 x 2.5" SSD/HDD Bay, 1 x M.2 M key
Hardware Security	TPM 2.0
Watchdog Timer	Programmable
OS Support	Microsoft® Windows 10 LTSB

Figura 6. Características del GPG [3]

1.3 Objetivos y alcance del trabajo

El objetivo del presente trabajo es diseñar un sistema de comunicaciones para el GPG. La función de este sistema es recibir tramas de tipo IEC61850 y obtener a partir de ellas las muestras de corriente y tensión. En la *Figura 7* viene representado el flujo de una trama IEC61850 de un MU a un IED, la parte de comunicaciones del IED se podría definir en 3 apartados:

1. Físico: Esta es la electrónica de la tarjeta. Próximamente, se realizará una breve descripción de la misma.

2. Enlace: En esta parte se define la dirección MAC, así como el protocolo de comunicaciones (en este caso Ethernet).

3. Aplicación: Programa encargado de hacer la decodificación de las tramas IEC61850.

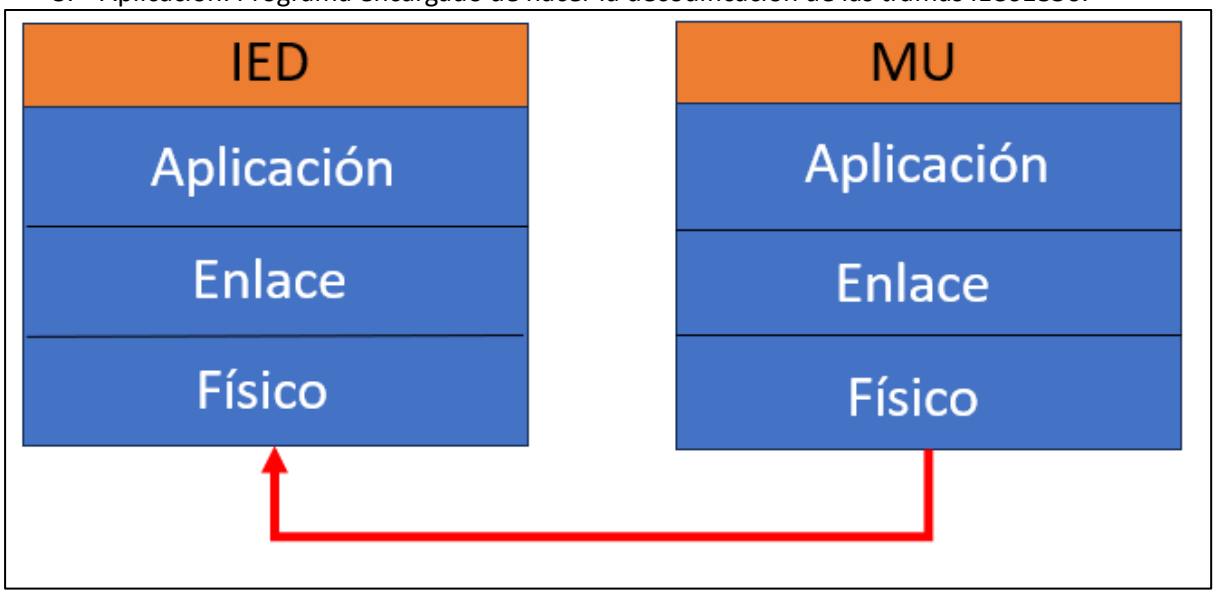


Figura 7. Diagrama de comunicaciones entre un IED y un MU

La idea inicial del presente TFM es el de lograr que este sistema de comunicaciones sea implementado en una tarjeta de SoC-e, *Figura 8*, con éxito. Por ello, una vez se haya realizado la implementación, se inserte dicha tarjeta en un slot del IED, *Figura 4*.

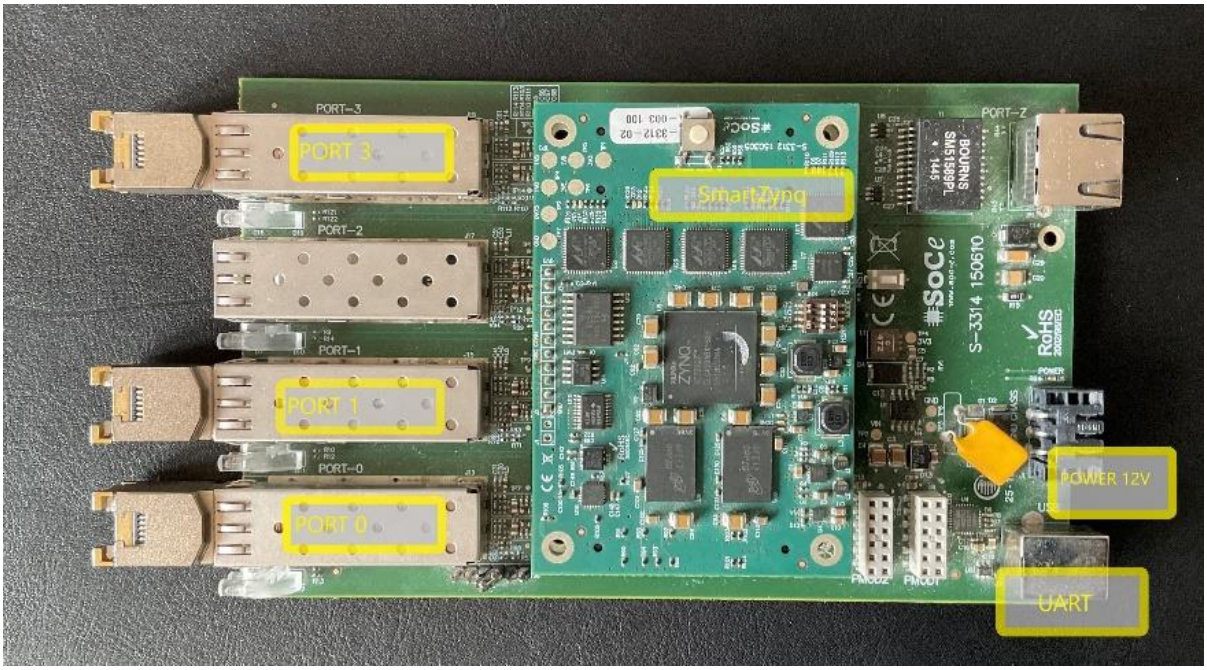


Figura 8. Tarjeta SoC-e

Esta tarjeta, se compone principalmente por 3 partes: Comunicaciones, alimentación y procesamiento. Las comunicaciones están formadas por 5 canales ethernet y 2 de UART. Tiene una alimentación de 12V y 0.4 A. El procesamiento corre a cargo del chip Zynq 7000 (distribuido por Xilinx) que está dividido en 2 partes tal y como se muestra en la *Figura 9.*

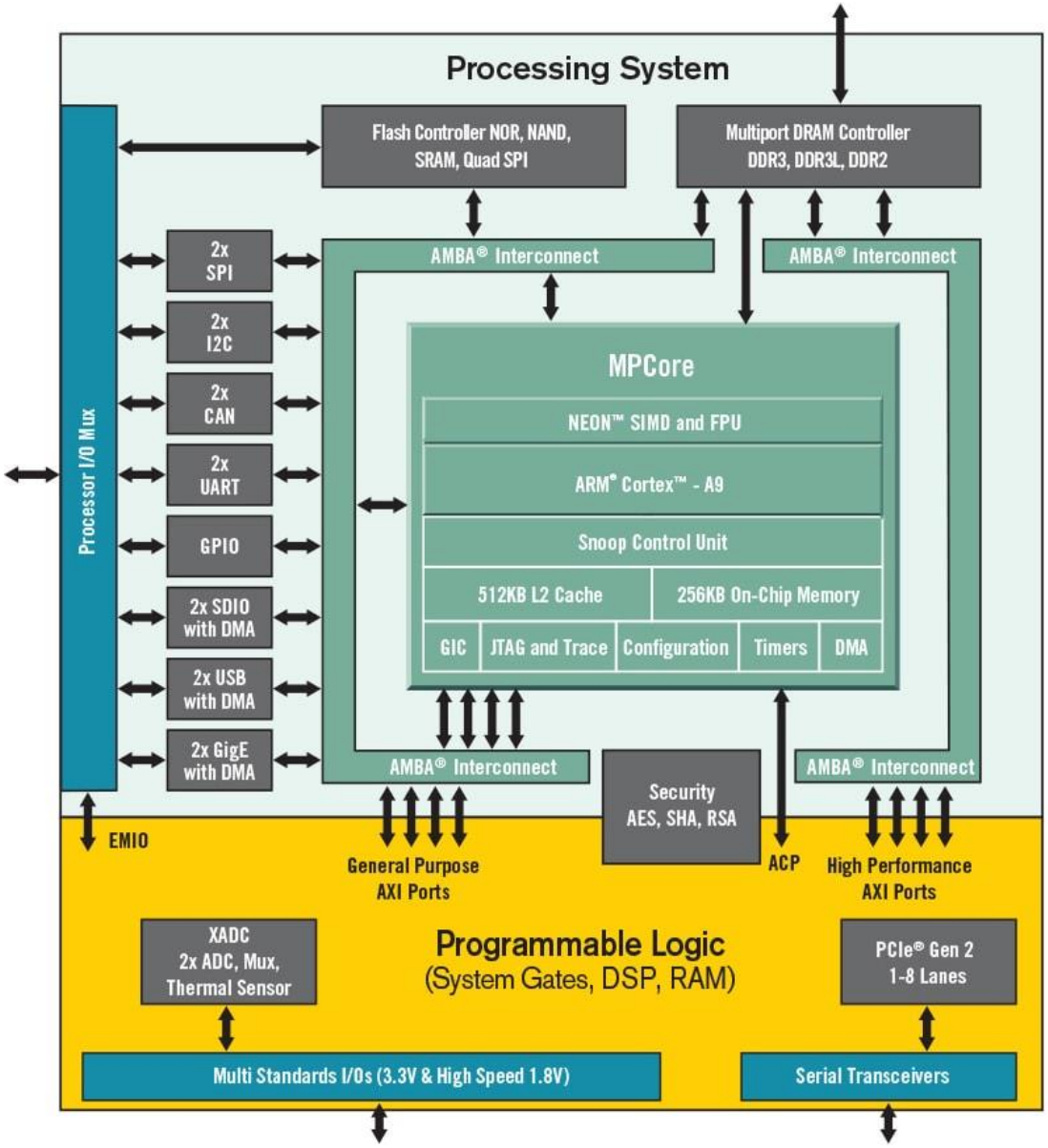


Figura 9. Arquitectura interna de un Zynq 7000

La primera parte es el PS (Processing System), en cuyo núcleo hay 2 microprocesadores ARM CORTEX-A9 que conectan con los diferentes periféricos (I2C, CAN, UART, General Purpose AXI ...) a través de los buses AMBA.

La segunda parte es el PL (Programable Logic), que es principalmente un bloque de la FPGA. Se comunica con el PS a través de los buses de interconexión AXI.

1.4 Beneficios que aporta el trabajo

Como ya se ha mencionado anteriormente, la mejora que se introducirá en los GPG (diseñados por la empresa) es una tarjeta de SoC-e cuya característica principal es que contiene un SoC Zynq 7000. La ventaja principal de este SoC respecto a cualquier procesador del mercado es que posee una FPGA, lo que permite automatizar electronicamente funciones.

En nuestro caso la función a automatizar será la de obtener las muestras de corriente y tensión a partir de las tramas recibidas, lo que consecuentemente, repercutirá en los tiempos de procesamiento, reduciéndolos respecto a un procesador comercial.

1.5 Análisis del estado del arte

Como ya se ha comentado anteriormente, las muestras de corriente y tensión vienen encapsuladas en tramas con el formato definido por la norma IEC61850. Lo importante para este trabajo es conocer la composición de las tramas que intercambian los sistemas MU e IED, ya que se van a modificar para añadir las funciones. Por tanto, a continuación, se describen estas tramas.

Las tramas están definidas según la norma IEC61850 que esta subdividida en 10 tipos diferentes, en el presente trabajo me centraré en lo definido en el apartado IEC61850-9. Esta sub-norma define el formato en el que se envía los Sampled Values entre los MUs y los IEDs. En la *Figura 10* viene representado los campos que componen una trama Ethernet, a continuación, se realiza una descripción de cada uno de ellos:

1. Preámbulo: Indica el inicio de la trama y tienen el objetivo de que el dispositivo que lo recibe se sincronice con los bits a recibir.

2. Delimitador de inicio de trama: Indica que la trama empieza a partir de él.

3. Dirección destino: Dirección MAC del receptor de la trama Ethernet.

4. Dirección origen: Dirección MAC del transmisor de la trama Ethernet.

5. Priority Tagged: Campo que define características específicas de la trama Ethernet.

6. Ethertype-PDU [4]:

    a. El campo Ethertype: Indica con qué protocolo están encapsulados los datos que contiene la trama, siendo este valor de 0x88BA para el caso de una trama IEC61850-9.

    b. El APPID: Identificador de aplicación, tiene un valor de 0x4000 para tramas IEC6850-9.

    c. El campo Length: Indica el número de octetos que incluye la cabecera Ethertype PDU. Después de estos hay 4 bytes que son reservados y, por último, está el APDU, que se explicará más adelante.

7. Frame check sequence o CRC: Es un conjunto de bits adjuntados al final de la trama Ethernet, utilizado para verificar la integridad de la información recibida mediante una "secuencia" de verificación de trama incorrecta, también conocido como checksum. [5]

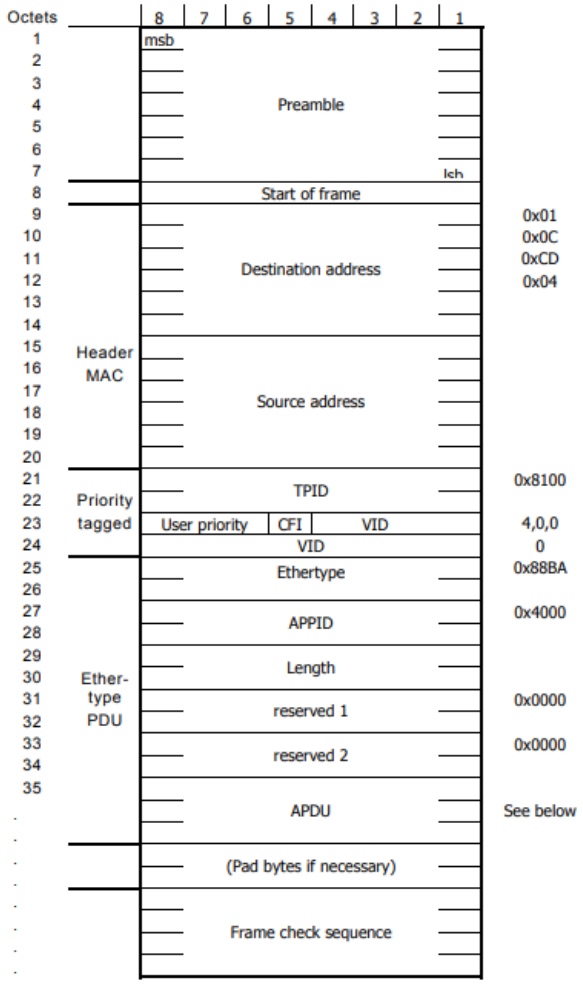


Figura 10. Formato de una trama Ethernet según la norma IEC61850-9 [6]

Los últimos bytes que componen el campo Ethertype-PDU de una trama IEC61850-9 está compuesta por el APDU (Aplication Data Unit), que son los valores medidos (tanto de tensión como de corriente) por el Merging Unit. La composición de esta parte de la trama está estructurada según lo marcado por la *Figura 11*.

El inicio del APDU está marcado por 3 campos que son:

1. savPDU: Tamaño del APDU
2. noASDU (Application Specific Data Unit): Número de ASDUs presentes en la APDU
3. Sequence of ASDU: Tamaño en bytes de los ASDUs

Después de estos 3 campos aparecen ya los ASDUs, y cada uno de estos está formado por 7 campos:

1. Sequence ASDU1: Número de bytes que ocupa su respectivo ASDU
2. svID: Identificador del Sample Value
3. smpCnt: Contador de Sampled Values recibidos
4. confRev: Señala el número de configuración para el ASDU

5. smpSynch: Define el método de sincronización del reloj para los Sampled Values

6. Sequence of Data: Tamaño en bytes del conjunto de muestras

7. DataSet: El conjunto de muestras de corriente y tensión

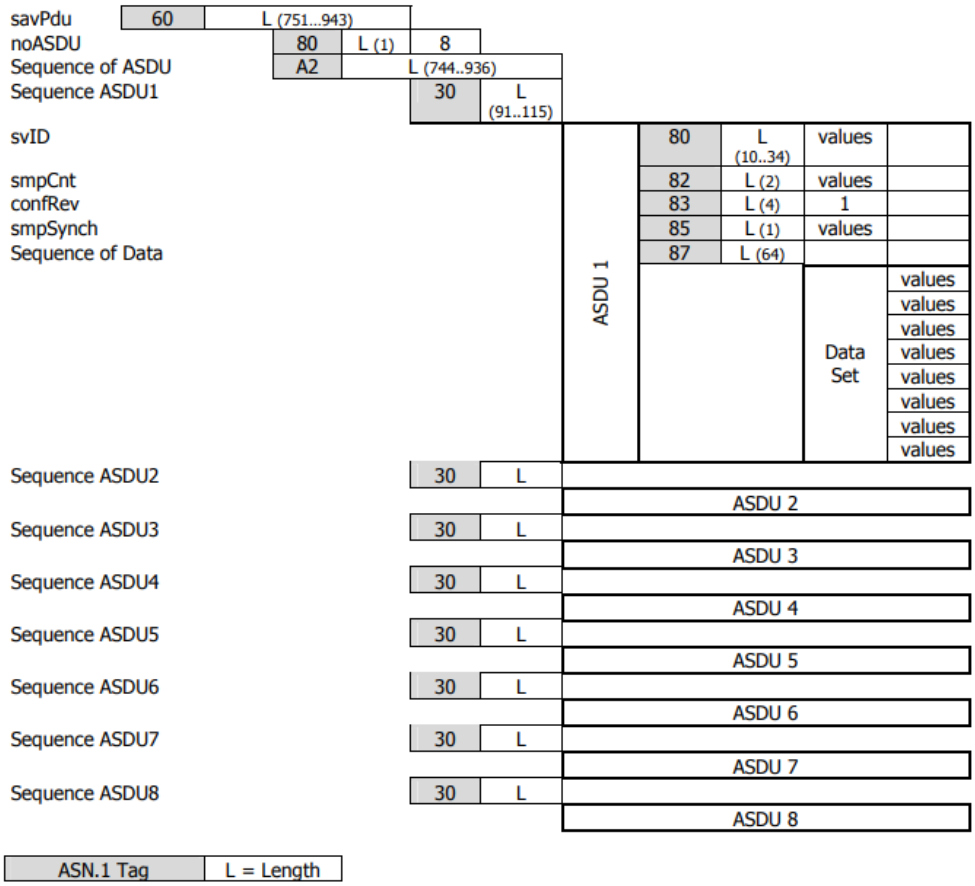


Figura 11. Contenido del campo APDU [6]

1.6 Selección/Descripción de la solución propuestas

En la *Figura 12* queda representada la funcionalidad del nuevo sistema de comunicaciones dentro de la tarjeta. Cabe destacar la presencia de 5 bloques:

1. SV Parser: Módulo que recibe las tramas de tipo IEC61850 y extrae de ellas muestras de corriente y tensión.

2. DMA (Direct Memory Access): Realiza el intercambio de información entre el PS y el PL guardando las muestras de corriente y tensión en una memoria DDR.

3. GMII2RGMII: Realiza la conversión de tramas de formato GMII a formato RGMII y viceversa. La principal diferencia entre estos 2 formatos es la forma de enviar los datos. En el GMII los datos se envían en bloques de 8 bits mientras que en el RGMII se envían 4 bits.

4. Marvell 88E1512: Es el driver hardware de la empresa Marvell, que transforma señales de RGMII a 1000 BaseTx y viceversa.

5. SFP (Small Form Factor Pluggable): Transceptor que está conectado con el driver Marvell 88E1512. A este transceptor se le conectará un cable Ethernet.

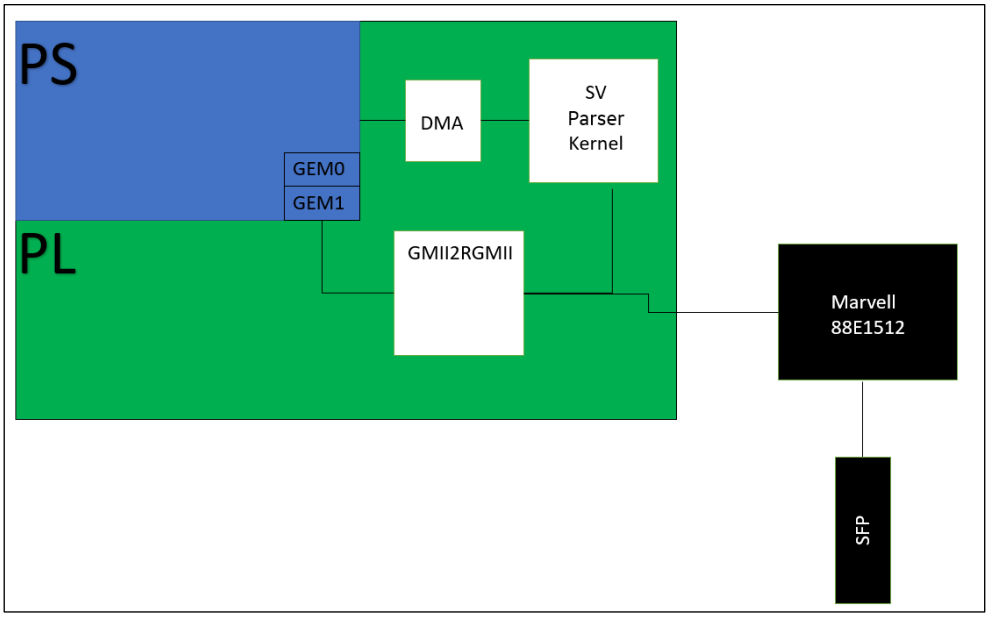


Figura 12. Esquema conceptual de la idea del trabajo

Una aproximación más genérica de la solución llevada a cabo se puede observar en la *Figura 13*, donde destacan principalmente 3 partes:

1. Software: Donde se realiza el programa base que se encargará de manejar el DMA y realizar el algoritmo correspondiente con las muestras.

2. PS: Donde se almacenará el programa base compilado para un Ubuntu en un SoC.

3. PL: Donde estará el rutado del diseño HW.

Figura 13. Esquema general de la solución

A continuación, realizaré una descripción detallada de cada una de las partes:

1.6.1 Primera Parte

En la *Figura 14* se puede apreciar el diseño HW de la FPGA, en este caso se ha empleado el programa de diseño software denominado Vivado. En ella aparecen los 3 primeros bloques, pertenecientes al PL, mencionados a partir de la *Figura 12*. Estos bloques son 1 el parseador, 2 la DMA [7], 3 el gmii a rgmii.

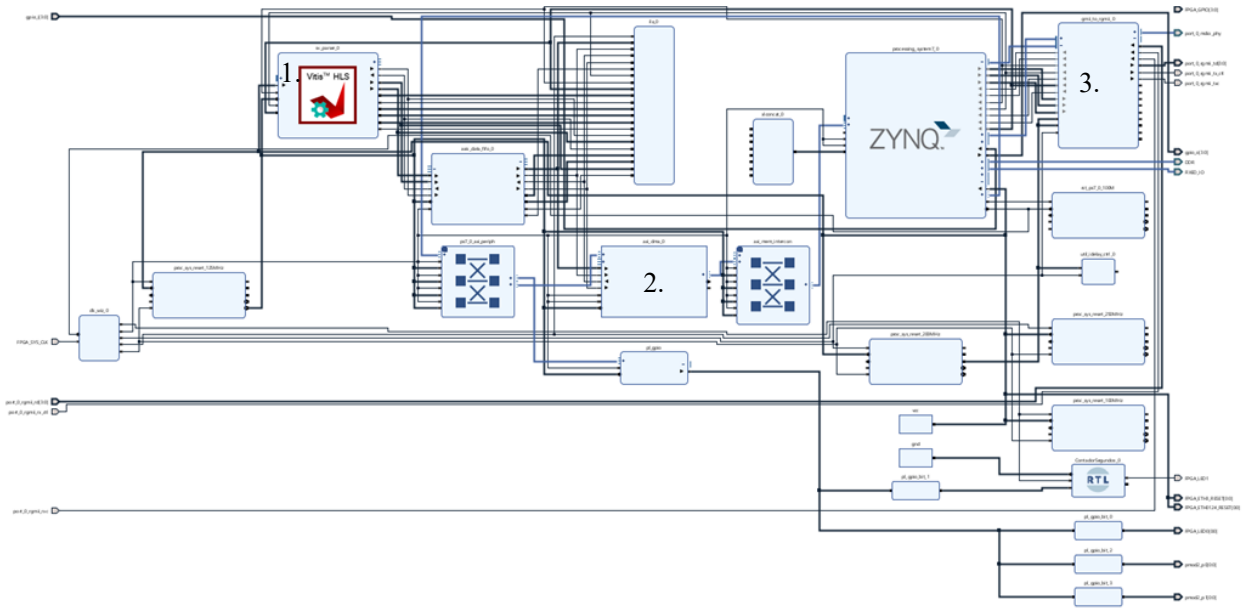


Figura 14. Diseño hardware de Vivado

Con el diseño de Vivado se procede a generar el bitstream (paso 3) correspondiente a este proyecto. Para ello será necesario realizar la síntesis del diseño (paso 1) (1. create HDL Wrapper) y la generación de los productos de síntesis, simulación, implementación y entrega hardware (paso 2) (2. Generate Output Product). Seguidamente, una vez generado el bitstream (paso 3) se exporta la plataforma con el bitstream obtenido (paso 4), el resultado de esta exportación será un fichero top_design.xsa que nos servirá luego para compilar el Sistema Operativo Ubuntu. En la *Figura 15* se aprecia los pasos llevados a cabo para conseguir exportar la plataforma.

Cabe destacar que el bloque parseador se ha realizado mediante la herramienta de Xilinx denominada Vitis HLS, que permite realizar bloques vhdl a partir de un lenguaje de programación de alto nivel. En este caso el Programa **1** que aparece en el Anexo se ve el código de dicho bloque, que está escrito en C++.

Mientras, que el resto de los bloques (Processing System, AXI Interconnect, AXI DMA …) se han ido añadiendo con los diseños que proporciona Vivado.

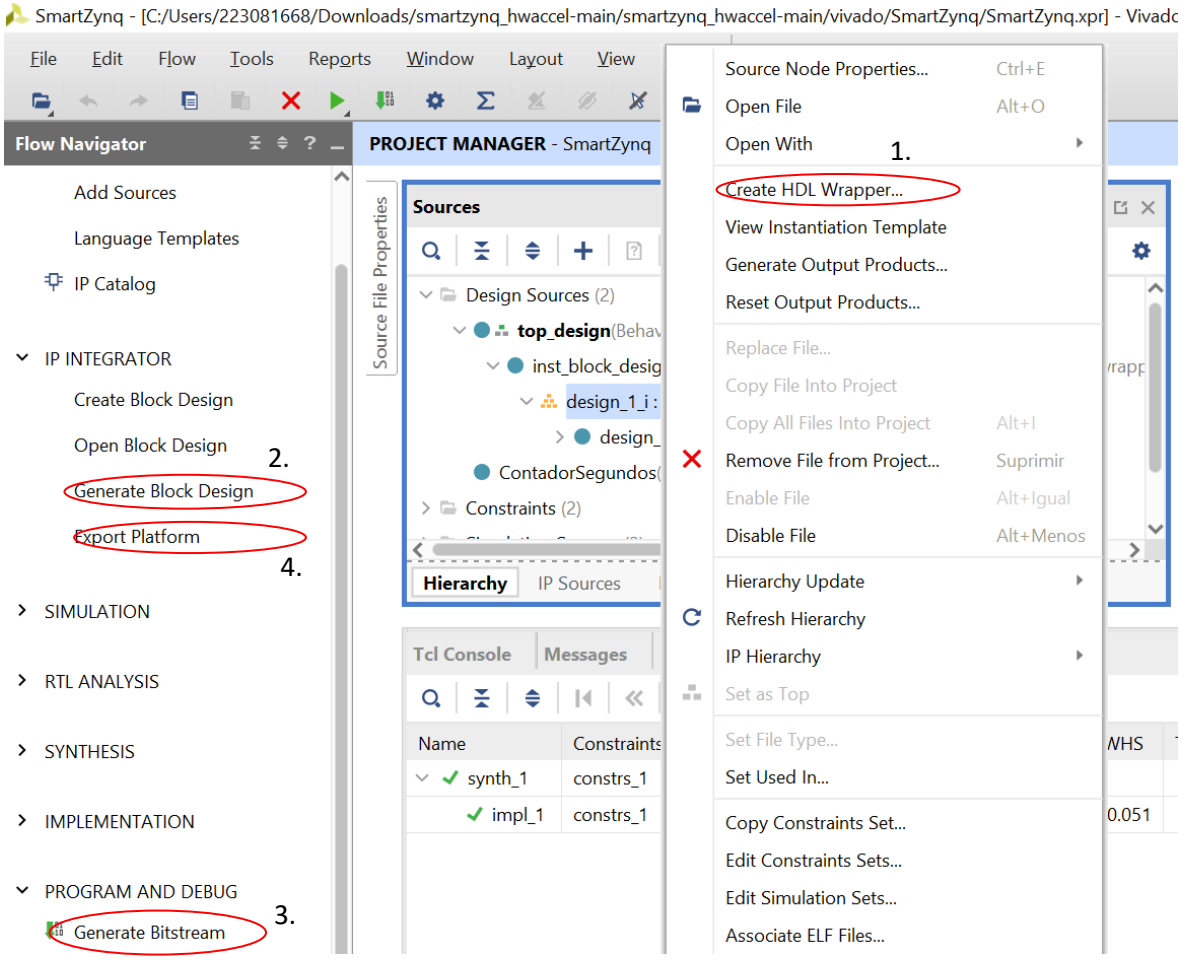


Figura 15. Procedimiento de diseño en Vivado

1.6.2 Segunda Parte

La tarjeta de SoC-e funcionará sobre el SO (Sistema Operativo) Ubuntu, la herramienta empleada para implementarlo es petalinux. En la *Figura 16* queda representado la secuencia de comandos llevada a cabo para obtener los ficheros necesarios del SO, con respecto al apartado anterior, es necesario el fichero top_design.xsa para poder compilar el Sistema Operativo.

```
inigoss@inigoss-VirtualBox:~/petalinux$ source settings.sh
inigoss@inigoss-VirtualBox:~/petalinux/ZYNQ$ petalinux-create --type project --template zynq --name ZYNQ
inigoss@inigoss-VirtualBox:~/petalinux$ cd ZYNQ/
inigoss@inigoss-VirtualBox:~/petalinux/ZYNQ$ export VIVADO_SDK_EXPORT_DIR=/home/inigoss/workspace/smartzynq_hwaccel-main/vivado/SmartZynq
inigoss@inigoss-VirtualBox:~/petalinux/ZYNQ$ petalinux-config --get-hw-description=$VIVADO_SDK_EXPORT_DIR
inigoss@inigoss-VirtualBox:~/petalinux/ZYNQ$ petalinux-config -c rootfs
inigoss@inigoss-VirtualBox:~/petalinux/ZYNQ$ petalinux-config -c kernel
inigoss@inigoss-VirtualBox:~/petalinux/ZYNQ$ petalinux-build
inigoss@inigoss-VirtualBox:~/petalinux/ZYNQ$ petalinux-build --sdk
```

Figura 16. Comandos empleados para generar los archivos necesarios [8]

En la *Figura 17,* se muestra los pasos realizados para el arranque del SO en la tarjeta, a continuación, pasaré a describir los siguientes archivos:

1. boot.scr: Primer archivo de arranque de SO, es la BIOS de Ubuntu.

2. BOOT.bin (zynq_fsbl.elf, top_design.bit y u-boot.elf): Carga el zynq_fsbl.elf (que carga el top_design.bit y luego pasa al u-boot.elf), el top_design.bit (es el diseño hardware del sistema) y el u-boot.elf es el archivo que carga el Sistema Operativo.

3. Kernel (uImage): Este ya es el programa del SO en sí, en el estará cargada la configuración gráfica, los programas que vengan ya instalados en el SO. Además, será capaz de comunicarse con los diferentes dispositivos HW a los que esté conectados gracias al device-tree.

4. Device-tree (system.dtb): Es el archivo donde está guardado la configuración de los diferentes dispositivos a los que está conectado el procesador, tales como conexiones USB, UART o componentes que estén en la FPGA.

5. Root Fs (rootfs.ext4): Con este fichero se cargará el sistema de archivos del SO, el conjunto de directorios que lo forman junto con las librerías que hallamos añadido en el proceso de compilación.

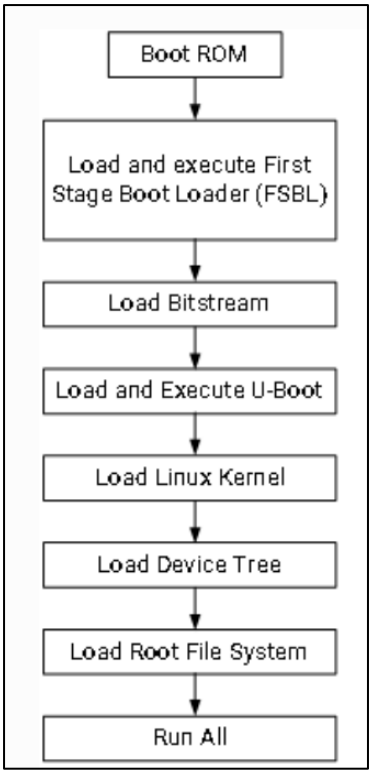


Figura 17. Esquema de arranque de Ubuntu en un SoC [9]

1.6.3 Tercera Parte

A continuación, paso a describir la última parte de la solución propuesta:

Para ello, utilizamos el software de Xilinx Vitis. Primero se crea la plataforma para generar el archivo de plataforma Xilinx ZYNQ_CON_DMA.xpfm, necesario para compilar nuestro posterior programa. En la *Figura 18* viene representado los archivos que pide Vitis para generar la plataforma denominada ZYNQ_CON_DMA. Estos archivos son los siguientes:

1. Carpeta Boot: Donde está el zynq_fsbl.elf, u-boot.elf y el system.dtb.

2. Carpeta sd_dir: Donde está el boot.scr y el system.dtb.

3. Rootfs.ext4

4. Directorio Sysroot: Donde se almacena el directorio del Sistema Operativo.

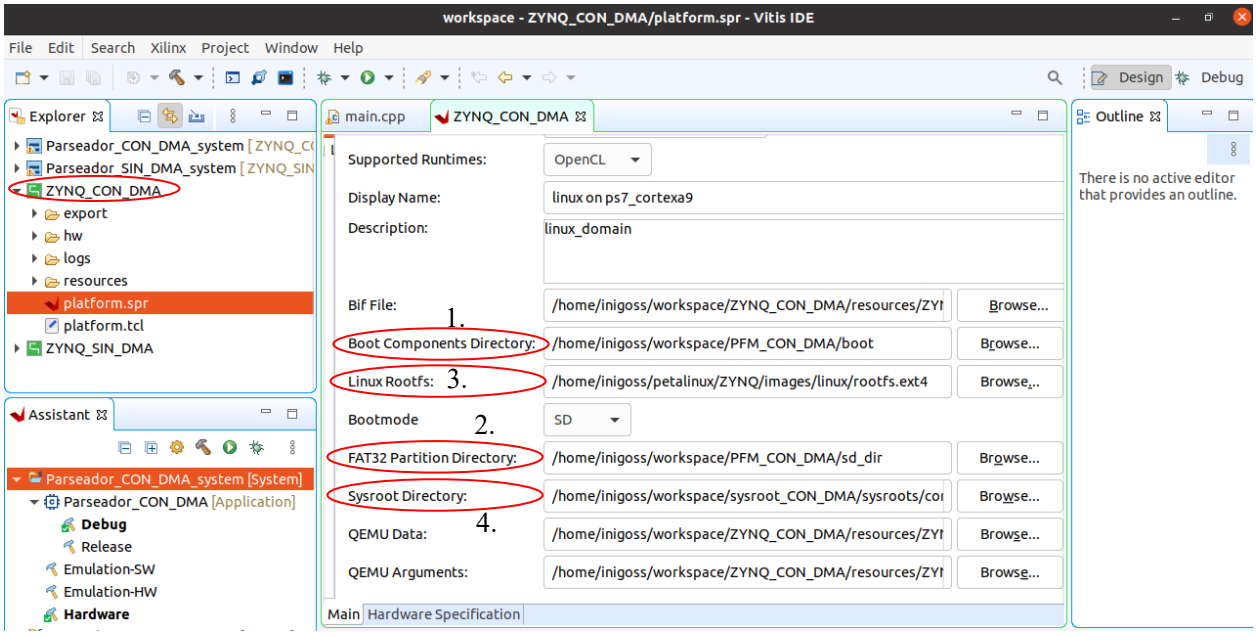


Figura 18. Creación de la plataforma

Segundo, iniciamos la programación de la aplicación que será implementada en la tarjeta electrónica. En la Figura 19 se muestra un esquema del código, en ella se distinguen 3 pasos.

1. La primera sirve para abrir un driver SW de la memoria DDR, así como para establecer un puntero a dicha memoria.

2. Se inicializan los registros de la DMA, y se espera a recibir la primera trama con los paquetes.

3. Por último, se entra en un bucle, donde, primero se resetea un registro de la DMA y posteriormente se recibe la próxima trama.

En el Anexo en el Programa **2** se incluye el código en C++ del programa, además, se incluyen otras 2 Figuras. En la *Figura 21*, aparecen las muestras de corriente y tensión en pantalla obtenidas de una

simulación. Mientras que en la *Figura 22*, se encuentra una trama IEC61850, además de una transferencia de las muestras de corriente y tensión al PS.

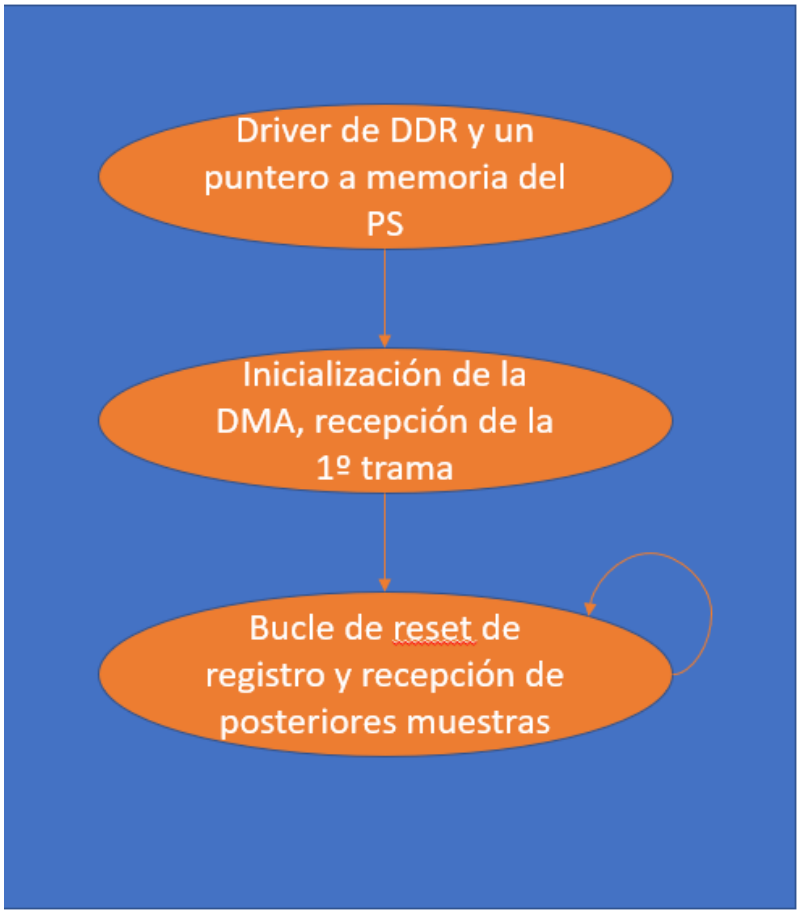


Figura 19. Diseño del código del sistema de comunicaciones

2. Metodología seguida en el desarrollo del trabajo

2.1 Descripción de tareas, fases, equipos y procedimientos

Descripción de tareas

1. Seguimiento del proyecto y documentación:

Esta tarea se ha centrado en la supervisión y documentación del trabajo, tanto de forma diaria en la empresa como de forma semanal en la Escuela.

Nombre	Horas
Tutor TFM	20
Iñigo Santaolalla	160

Tabla 1. Supervisión del proyecto

2. Definición del proyecto:

Establecimiento de los objetivos y especificaciones del trabajo.

Nombre	Horas
Empleado	15
Iñigo Santaolalla	40

Tabla 2. Definición del proyecto

3. Implementación del SO en la tarjeta electrónica:

Pasos realizados para la implementación del SO Ubuntu en la tarjeta de desarrollo.

Nombre	Horas
Empleado	30
Iñigo Santaolalla	200

Tabla 3. Implementación del SO

4. Pruebas realizadas:

Una vez se ha introducido Ubuntu en la tarjeta se procede a realizar una serie de pruebas para profundizar en el conocimiento de lo que se está trabajando.

Nombre	Horas
Empleado	20
Iñigo Santaolalla	160

Tabla 4. Realización de pruebas

5. Programación del módulo SVParser y testeo en la tarjeta:

Para acabar se introduce el módulo parseador en el chip y se toman medidas para verificar su correcto funcionamiento.

Nombre	Horas
Empleado	15
Iñigo Santaolalla	80

Tabla 5. Programación y testeo en la tarjeta

Descripción de equipos

El grupo de trabajo de este proyecto se compone de tres personas:

Nombre	Horas
Empleado+Tutor TFM	Supervisión del proyecto
Iñigo Santaolalla	Realización del proyecto

Tabla 6. Composición del grupo de trabajo

A continuación, se procede a nombrar los recursos materiales utilizados durante el trabajo.

Material	Cantidad
Ordenador Personal	1
Tarjeta electrónica + conectores	1

Tabla 7. Recursos materiales utilizados

2.2 Diagrama de Gantt

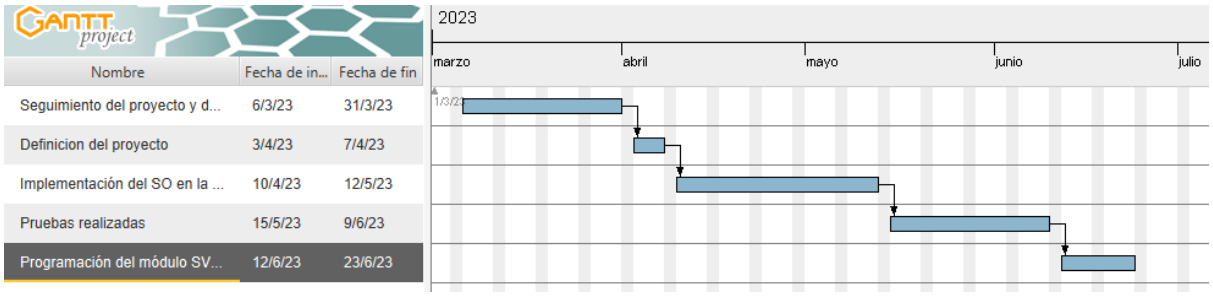


Figura 20. Diagrama de Gantt del proyecto

3. Aspectos económicos

3.1 Descripción del presupuesto ejecutado

Nombre	Horas (hrs)	Precio/horas (€/hrs)	Precio (€)
Empleado+Tutor TFM	100	70	7.000
Iñigo Santaolalla	640	20	12.800
Material	X	X	1.000
Costes Indirectos	X	X	2.123
Total	X	X	**22.923**

Tabla 8. Presupuesto total del trabajo

Conclusiones

En este TFM se ha descrito el hardware para un IED. En concreto, se ha descrito el hardware de comunicaciones para un GPG. Por un lado, se ha realizado una contextualización del GPG; explicando donde se ubica, que componentes lo forman y la funcionalidad de cada componente. Además, se ha realizado una breve introducción al tipo de tramas empleadas por estos IEDs, este tipo de trama está estructurada según el formato IEC61850.

Por otro lado, he explicado el proceso llevado a cabo para implementar un programa sencillo en el hardware. Primeramente, he realizado una breve introducción de la solución llevada a cabo y, dicha solución la he dividido en 3 apartados. El primero, cuenta como obtener el fichero top_design.xsa. El segundo, detalla el proceso llevado a cabo para obtener los ficheros que definen el Sistema Operativo. Y, por último, se describe el procedimiento para obtener el programa necesario de obtención de las muestras de corriente y tensión.

Este trabajo se ha realizado dentro de la empresa General Electric y se ha concluido realizando las pruebas funcionales. En estas pruebas se presentan 3 apartados. El primero, muestra el código desarrollado para presentar el programa que decodifique las muestras de corriente y tensión. El segundo, muestra una captura con la visualización de varias muestras de corriente y tensión. Y, por último, se muestra otra captura que presenta varias señales de cómo es el proceso de decodificación de una trama para obtener los valores de corriente y tensión.

En conclusión, en las simulaciones presentadas se puede ver el sistema de comunicaciones funcionando, a falta de alguna corrección en el diseño, este, serviría para ser implementado en un GPG.

Bibliografía

[1] «Red Eléctrica de España,» [En línea]. Available: https://unaautopistadetrasdelenchufe.wordpress.com/conoce-red-electrica-de-espana/.

[2] L. M. J. J. A. A. J. L. Le Sun, «High-Performance Computing Architecture for,» *IEEE,* p. 11, 2021.

[3] «Advantech,» [En línea]. Available: https://www.advantech.com/en/products/1-369nwl/ecu-4784/mod_18553282-e8f5-4b32-a64b-1083f7182d36.

[4] «Typhoon'hil,» [En línea]. Available: https://www.typhoon-hil.com /documentation/typhoon-hil-software-manual/References/iec_61850_sampled_values_protocol.html.

[5] «Wikipedia,» 21 Marzo 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Frame_Check_Sequence#:~:text=El%20Frame%20Check%20Sequence%20es%20un%20conjunto%20de,trama%20incorrecta%2C%20tambi%C3%A9n%20conocido%20como%20CRC%20o%20checksum..

[6] G. L. F. L. F. S. Cristoph Brunner, «Implementation guideline for digital interface to intrument trans formers using IEC 61850-9,» *UCA International Users Group,* p. 31, 2004.

[7] W. Knitter, «Hackster.io,» [En línea]. Available: https://www.hackster.io/whitney-knitter/introduction-to-using-axi-dma-in-embedded-linux-5264ec. [Último acceso: Enero 2024].

[8] Xilinx. [En línea]. Available: https://xilinx.github.io/Vitis-Tutorials/2021-2/build/html/docs/Vitis_Platform_Creation/Introduction/02-Edge-AI-ZCU104/step2.html.

[9] Xilinx. [En línea]. Available: https://xilinx.github.io/Embedded-Design-Tutorials/docs/2020.2/build /html/docs/Introduction/Zynq7000-EDT/7-linux-booting-debug.html.

Anexo: Código y resultados

Programa 1. Bloque del código del parseador

Este código hace referencia al bloque parseador que se encarga de obtener las muestras de corriente y tensión de las tramas IEC61850.

```
#include <iostream>
#include <stdlib.h>
#include <hls_stream.h>
#include <assert.h>
#include <ap_axi_sdata.h>

// Posicion de variables
#define TIPOETHER_BYTE0 136
#define TIPOETHER_BYTE1 186
#define TIPOETHER_POS0 21
#define TIPOETHER_POS1 22
#define MUESTRAS_POS0 74
#define MUESTRAS_POS1 138

void Parsear(
      bool RX_DV,
      unsigned char *FIN,
      unsigned char DATO_RDO,
      unsigned char *posicion,
      unsigned char *contador1,
      unsigned char *datos,
      unsigned char *ps_salida,
      unsigned char *ps_entrada,
      hls::stream< ap_axis<8,0,0,0>> &muestras
      )
{

#pragma HLS INTERFACE s_axilite port = ps_entrada clock=AXI_CLK bundle =
DATOS_PS // Definicion AXI LITE entrada
#pragma HLS INTERFACE s_axilite port = ps_salida clock=AXI_CLK bundle =
DATOS_PS // Definicion AXI LITE salida
#pragma HLS INTERFACE axis port = muestras // Definicion AXI STREAM

// Declaración de variables intermedias
#pragma HLS interface ap_none port=contador1
#pragma HLS interface ap_none port=datos
#pragma HLS interface ap_none port=posicion
#pragma HLS interface ap_none port=FIN

#pragma HLS interface ap_ctrl_none port = return

      static unsigned char arreglo[68], seq_datos[68], pos = 0, pos1 = 0, n
= 0, contador3 = 0;
      static bool trama_IEC = 0, TIPOETHER_enable = 0, LEZ = 0, FIN_TRAMA =
0;

#pragma HLS PIPELINE
#pragma HLS DEPENDENCE variable=arreglo inter WAR false
#pragma HLS RESOURCE variable=arreglo core=RAM_2P

      ap_axis<8,0,0,0> tmp;
```

```
	if(FIN_TRAMA == 1){ // Si se ha acabado la trama
		if(pos < 68){ // Mientras la posicion sea menor a 68
			tmp.data = seq_datos[pos] ; // Escribimos valor en tmp e
			tmp.keep = 1;               // incrementamos la pos
			if(pos == 67){
				tmp.last = 1;
			}else{
				tmp.last = 0;
			}
			pos += 1;
			muestras.write(tmp);
		}else{ // De lo contrario reseteamos los valores
			pos = 0;
			pos1 = 0;
			contador3 = 0;
			FIN_TRAMA = 0;
		}
	}

	if(RX_DV == 1){ // Deberes para cuando llegue una trama
		contador3 += 1; // Incrementar el contador
 // Asignarlo a 1 si se cumplen la condiciones
		if(contador3 == TIPOETHER_POS0 && DATO_RDO == TIPOETHER_BYTE0){
		// 88 en hex
			TIPOETHER_enable = 1;		}else if(contador3 ==
TIPOETHER_POS1 && DATO_RDO == TIPOETHER_BYTE1 && TIPOETHER_enable == 1){//
ba en hex
			LEZ = 1;
			trama_IEC = 1;
// Rellenar el array seq_datos
		}else if(trama_IEC == 1 && (contador3 == 59 or contador3 ==
60)){
			seq_datos[pos1] = 0;
			pos1 += 1;
		}else if(contador3 == 61 or contador3 == 62){
							// Guardar el SmpCnt
			seq_datos[pos1] = DATO_RDO;
			pos1 += 1;
		}else if(contador3 >= MUESTRAS_POS0 && contador3 <
MUESTRAS_POS1 && trama_IEC == 1){	// Datos corriente y tension
			seq_datos[pos1] = DATO_RDO;
			pos1 += 1;
// Establecer a 1 el FIN_TRAMA
		}else if(contador3 == MUESTRAS_POS1 && trama_IEC == 1){
			FIN_TRAMA = 1;
		}
	}else if(RX_DV == 0){ // Deberes para cuando se termine la trama
		contador3 = 0;
		TIPOETHER_enable = 0;
		trama_IEC = 0;
		LEZ = 0;
		pos1 = 0;
	}
// Deberes para cuando se termine la trama

	*posicion = pos;
	*datos = tmp.data;
	*FIN = FIN_TRAMA;
	*contador1 = contador3;
```

Programa 2. Código del main.cpp

Este código, escrito en C++, pertenece al PS, en el que queda demostrado lo representado en la *Figura 19*.

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <termios.h>
#include <sys/mman.h>
#include <time.h>
#include <math.h>
#include <stdint.h>
#include <inttypes.h>

// Variables de registro
#define MM2S_CONTROL_REGISTER       0x00
#define MM2S_STATUS_REGISTER        0x04
#define MM2S_SRC_ADDRESS_REGISTER   0x18
#define MM2S_TRNSFR_LENGTH_REGISTER 0x28

#define S2MM_CONTROL_REGISTER       0x30
#define S2MM_STATUS_REGISTER        0x34
#define S2MM_DST_ADDRESS_REGISTER   0x48
#define S2MM_BUFF_LENGTH_REGISTER   0x58

// Variables de flags
#define IOC_IRQ_FLAG                1<<12
#define IDLE_FLAG                   1<<1

// Variables del estado
#define STATUS_HALTED               0x00000001
#define STATUS_IDLE                 0x00000002
#define STATUS_SG_INCLDED           0x00000008
#define STATUS_DMA_INTERNAL_ERR     0x00000010
#define STATUS_DMA_SLAVE_ERR        0x00000020
#define STATUS_DMA_DECODE_ERR       0x00000040
#define STATUS_SG_INTERNAL_ERR      0x00000100
#define STATUS_SG_SLAVE_ERR         0x00000200
#define STATUS_SG_DECODE_ERR        0x00000400
#define STATUS_IOC_IRQ              0x00001000
#define STATUS_DELAY_IRQ            0x00002000
#define STATUS_ERR_IRQ              0x00004000

// Variables para la DMA
#define HALT_DMA                    0x00000000
#define RUN_DMA                     0x00000001
#define RESET_DMA                   0x00000004
#define ENABLE_IOC_IRQ              0x00001000
#define ENABLE_DELAY_IRQ            0x00002000
#define ENABLE_ERR_IRQ              0x00004000
#define ENABLE_ALL_IRQ              0x00007000

// Variables del AXI
#define XPARSEAR_DATOS_PS_ADDR_PS_SALIDA_DATA  0x10
#define XPARSEAR_DATOS_PS_BITS_PS_SALIDA_DATA  64
#define XPARSEAR_DATOS_PS_ADDR_PS_ENTRADA_DATA 0x18
#define XPARSEAR_DATOS_PS_BITS_PS_ENTRADA_DATA 64
```

```
unsigned int write_dma(unsigned int *virtual_addr, int offset, unsigned int
value)
{
    virtual_addr[offset>>2] = value; // Escritura de un valor

    return 0;
}

unsigned int read_dma(unsigned int *virtual_addr, int offset)
{
    return virtual_addr[offset>>2]; // Lectura de un valor
}

void dma_s2mm_status(unsigned int *virtual_addr)
{
    unsigned int status = read_dma(virtual_addr, S2MM_STATUS_REGISTER);
    // Lectura del valor del status
    printf("Stream to memory-mapped status (0x%08x@0x%02x):", status,
S2MM_STATUS_REGISTER);

    if (status & STATUS_HALTED) {
            printf("Se ha quedado en Halted.\n");
      } else {
            printf(" Running.\n");  // Estado running

      }

    if (status & STATUS_IDLE) {
            printf(" Idle.\n"); // Estado Idle
      }

    if (status & STATUS_SG_INCLDED) {
            printf(" SG is included.\n");
      }

    if (status & STATUS_DMA_INTERNAL_ERR) {
            printf(" DMA internal error.\n");  // Estado error interno DMA
      }

    if (status & STATUS_DMA_SLAVE_ERR) {
            printf(" DMA slave error.\n");  // Estador error DMA
      }

    if (status & STATUS_DMA_DECODE_ERR) {
            printf(" DMA decode error.\n");  // Estado error de
    // decodificación DMA
      }

    if (status & STATUS_SG_INTERNAL_ERR) {
            printf(" SG internal error.\n");
      }

    if (status & STATUS_SG_SLAVE_ERR) {
            printf(" SG slave error.\n");
      }

    if (status & STATUS_SG_DECODE_ERR) {
            printf(" SG decode error.\n");
      }

    if (status & STATUS_IOC_IRQ) {
```

```
            printf(" IOC interrupt occurred.\n"); // Estado Interrupción
    // generada
      }

    if (status & STATUS_DELAY_IRQ) {
            printf(" Interrupt on delay occurred.\n"); // Estado
    // interrupcion con retraso

      }

    if (status & STATUS_ERR_IRQ) {
            printf(" Error interrupt occurred.\n");
      }
}

int dma_s2mm_sync(unsigned int *virtual_addr)
{
    unsigned int s2mm_status = read_dma(virtual_addr,
S2MM_STATUS_REGISTER);

      // sit in this while loop as long as the status does not read back
0x00001002 (4098)
      // 0x00001002 = IOC interrupt has occured and DMA is idle

      while(!(s2mm_status & IOC_IRQ_FLAG) || !(s2mm_status & IDLE_FLAG))
      {
        dma_s2mm_status(virtual_addr);
        //dma_mm2s_status(virtual_addr);

        s2mm_status = read_dma(virtual_addr, S2MM_STATUS_REGISTER);
      }


      return 0;
}

void print_mem(void *virtual_address, int byte_count)
{
      char *data_ptr = (char *)virtual_address;

      for(int i=0;i<byte_count;i++){
            printf("%02X", data_ptr[i]);

            // print a space every 4 bytes (0 indexed)
            if(i%4==3){
                  printf(" ");
            }
      }

      printf("\n");
}

int main()
{
      int num_trama = 0; // Inicializado el valor num_trama
    printf("Hello World! - Running DMA transfer test application.\n");

      printf("Memory map the address of the DMA AXI IP via its AXI lite
control interface register block.\n");
    unsigned int *dma_virtual_addr = (unsigned int *) mmap(NULL, 65535,
PROT_READ | PROT_WRITE, MAP_SHARED, ddr_memory, 0x80400000);
```

```
    // Dir. destino de memoria DMA

      printf("Memory map the S2MM destination address register block.\n");
    unsigned int *virtual_dst_addr = (unsigned int *) mmap(NULL, 65535,
PROT_READ | PROT_WRITE, MAP_SHARED, ddr_memory, 0x0f000000);
    // Dir. destino de memoria DDR para el acceso del PS a la DMA

      printf("Clearing the destination register block...\n");
    memset(virtual_dst_addr, 0, 64);
    // Establecemos a 0 los valores de la dirección virtual de destino
    printf("Destination memory block data: ");
      print_mem(virtual_dst_addr, 64);

   // Printeamos el valor de virtual destination address

   struct timespec t_inicio1, t_final1;
   clock_gettime(CLOCK_REALTIME, &t_inicio1);

    printf("Reset the DMA.\n");
    write_dma(dma_virtual_addr, S2MM_CONTROL_REGISTER, RESET_DMA);
    // Escribimos 4 en el registro de control
    dma_s2mm_status(dma_virtual_addr, &reset);

      printf("Halt the DMA.\n");
    write_dma(dma_virtual_addr, S2MM_CONTROL_REGISTER, HALT_DMA);
    // Escribimos 0 en el registro de control
    dma_s2mm_status(dma_virtual_addr, &reset);

      printf("Enable all interrupts.\n");
    write_dma(dma_virtual_addr, S2MM_CONTROL_REGISTER, ENABLE_ALL_IRQ);
    // Escribimos 0x00007000 en el registro de control
    dma_s2mm_status(dma_virtual_addr, &reset);

    printf("Writing the destination address for the data from S2MM in
DDR...\n");
    write_dma(dma_virtual_addr, S2MM_DST_ADDRESS_REGISTER, 0x0f000000);
    // Escribimos 0xf000000 en el registro de buffer length
    dma_s2mm_status(dma_virtual_addr, &reset);

      printf("Run the S2MM channel.\n");
    write_dma(dma_virtual_addr, S2MM_CONTROL_REGISTER, RUN_DMA);
    // Escribimos 1 en el registro de control
    dma_s2mm_status(dma_virtual_addr, &reset);

    printf("Writing S2MM transfer length of 64 bytes...\n");
    write_dma(dma_virtual_addr, S2MM_BUFF_LENGTH_REGISTER, 64);
    // Escribimos 64 en el registro de buffer length
    dma_s2mm_status(dma_virtual_addr, &reset);

    clock_gettime(CLOCK_REALTIME, &t_final1);
    uint64_t diferencia_tiempo1 = (t_final1.tv_sec * pow(10,9) +
t_final1.tv_nsec) - (t_inicio1.tv_sec * pow(10,9) + t_inicio1.tv_nsec);
    // Sacamos el valor de la diferencia de tiempo
    printf("La diferencia de tiempo 1 es %d\n", diferencia_tiempo1);

    printf("Waiting for S2MM sychronization...\n");

    dma_s2mm_sync(dma_virtual_addr, &reset);
    dma_s2mm_status(dma_virtual_addr, &reset);

    printf("Destination memory block: ");
```

```
    print_mem(virtual_dst_addr, 64);
    num_trama += 1;                         // Numero de tramas leídas

    uint64_t diferencia_tiempo2;
      while(num_trama < 50){

            struct timespec t_inicio2, t_final2;
            clock_gettime(CLOCK_REALTIME, &t_inicio2);

            printf("Writing S2MM transfer length of 64 bytes...\n");
            write_dma(dma_virtual_addr, S2MM_BUFF_LENGTH_REGISTER, 64);
           // Escribimos 64 en el registro de buffer length
            dma_s2mm_status(dma_virtual_addr, &reset);

            printf("Waiting for S2MM sychronization...\n");

            dma_s2mm_sync(dma_virtual_addr);
            dma_s2mm_status(dma_virtual_addr);

            printf("Destination memory block: ");
            print_mem(virtual_dst_addr, 64);
            // Sacamos por pantalla los valores de la dirección virtual de
            // destino
            num_trama += 1;

            clock_gettime(CLOCK_REALTIME, &t_final2);
            diferencia_tiempo2 = (t_final2.tv_sec * pow(10,9) +
t_final2.tv_nsec) - (t_inicio2.tv_sec * pow(10,9) + t_inicio2.tv_nsec);
            // Obtenemos el valor de la variable diferencia de tiempo
            printf("La diferencia de tiempo 2 es %d\n",
diferencia_tiempo2);
            // Sacamos por pantalla el valor de número de tramas
            printf("El numero de tramas que has leido es:%d\n", num_trama);
    }

      printf("Se han leido %d tramas\n", num_trama);

    return 0;
}
```

```
root@SmartZynq_GE_plnx_2:~# ./Parseador_CON_DMA.elf
Hello World! - Running DMA transfer test application.
Opening a character device file of the Arty's DDR memeory...
Memory map the address of the DMA AXI IP via its AXI lite control interface register block
.
Memory map the S2MM destination address register block.
Clearing the destination register block...
Destination memory block data: 00000000 00000000 00000000 00000000 00000000 00000000 00000
000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Reset the DMA.
Stream to memory-mapped status (0x00000001@0x34): Halted.
Halt the DMA.
Stream to memory-mapped status (0x00000001@0x34): Halted.
Enable all interrupts.
Stream to memory-mapped status (0x00000001@0x34): Halted.
Writing the destination address for the data from S2MM in DDR...
Stream to memory-mapped status (0x00000001@0x34): Halted.
Run the S2MM channel.
Stream to memory-mapped status (0x00000000@0x34): Running.
Writing S2MM transfer length of 64 bytes...
Stream to memory-mapped status (0x00000000@0x34): Running.
La diferencia de tiempo 1 es 4268264
Waiting for S2MM sychronization...
Stream to memory-mapped status (0x00000000@0x34): Running.
Stream to memory-mapped status (0x00001002@0x34): Running.
 Idle.
 IOC interrupt occurred.
Destination memory block: 00000000 00000000 FFFFFB38 00000000 000004C8 00000000 00000000 0
0002000 00000000 00000000 FFFFE497 00000000 00001B69 00000000 00000000 00002000
Writing S2MM transfer length of 64 bytes...
Stream to memory-mapped status (0x00001000@0x34): Running.
 IOC interrupt occurred.
Waiting for S2MM sychronization...
Stream to memory-mapped status (0x00001002@0x34): Running.
 Idle.
 IOC interrupt occurred.
Destination memory block: 0000006E 00000000 FFFFFB04 00000000 0000048D 00000000 00000000 0
0002000 0000027B 00000000 FFFFE36E 00000000 00001A16 00000000 00000000 00002000
La diferencia de tiempo 2 es 4268364
Writing S2MM transfer length of 64 bytes...
Stream to memory-mapped status (0x00001000@0x34): Running.
 IOC interrupt occurred.
Waiting for S2MM sychronization...
Stream to memory-mapped status (0x00001002@0x34): Running.
 Idle.
 IOC interrupt occurred.
Destination memory block: 000000DD 00000000 FFFFFAD8 00000000 0000044B 00000000 00000000 0
0002000 000004F3 00000000 FFFFE273 00000000 00001899 00000000 00000000 00002000
La diferencia de tiempo 2 es 4268364
```

Figura 21. Muestras de corriente y tensión sacadas por pantalla. Captura de pantalla que muestra los valores de corriente y tensión obtenidos de varias decodificaciones

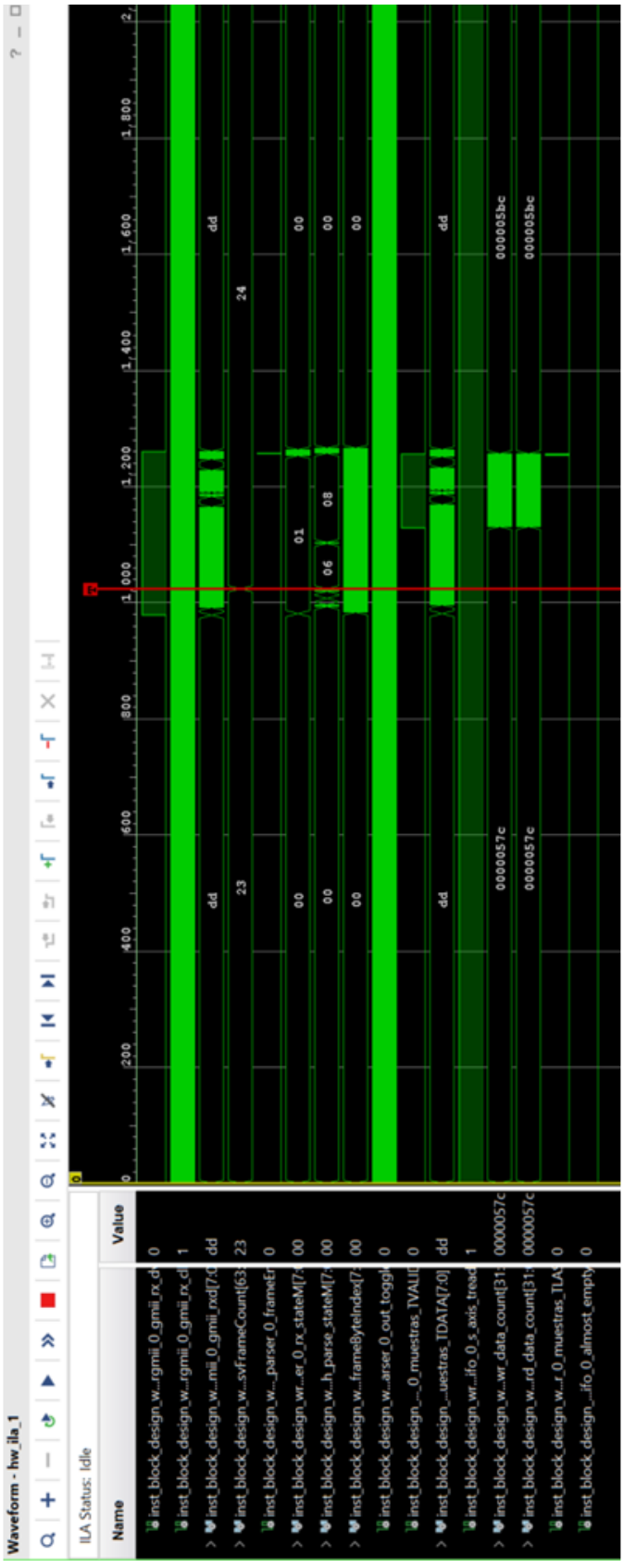


Figura 22. Imagen con recepción de tramas y transferencia de muestras. Captura de pantalla que muestra el proceso de decodificación de una trama para obtener las muestras de corriente y tensión