

Ingeniería Electrónica

Trabajo Final de Grado Desarrollo de una red de distribución de vídeo RADAR en protocolo ASTERIX CAT240 sobre Ethernet Gigabit

Autor/es Tec. Agustin Emanuel Allende Tec. Juan José Miguel Aguirre

> Director o Tutor Mg. Guillermo Friedrich Ing. Adrián Laiuppa

Bahía Blanca | 15 de Septiembre de 2023

Índice de contenidos

Lista de abreviaciones	3
Lista de figuras	6
Lista de tablas	10
1. Introducción	11
2. Descripción del proyecto	12
2.1. Propuesta de proyecto	12
2.2. Puesta en marcha y configuración del kit DE1-SoC	14
2.3. Software (sin la implementación del handshake entre la FPGA y el HPS)	32
2.3.1. Parte Qsys	33
2.3.2. Parte QtCreator	33
2.4. Ensayos (sin la implementación del handshake entre la FPGA y el HPS)	40
2.4.1. Transmisión y recepción de paquetes UDP utilizando el comando iperf	40
2.4.2. Medición de tiempos de lectura y de transmisión con datos fijos	43
2.4.3. Medición de tiempos de lectura y de transmisión con datos fijos	
empaquetados en protocolo ASTERIX CAT240	45
2.4.3.1. Tamaño de trama de 1056 [bytes]	46
2.4.3.2. Tamaño de trama de 2080 [bytes]	48
2.4.3.3. Tamaño de trama de 4128 [bytes]	51
2.4.3.4. Modo debug vs. Modo release	55
2.4.3.5. Capacidad del canal	55
2.4.3.6. Modificación de la frecuencia del clock del PLL h2f_axi_clk	56
2.4.3.7. Medición de la velocidad de lectura de la memoria compartida entre FPGA y el HPS	la 59
2.5. Software (con la implementación del handshake entre la FPGA y el HPS)	61
2.5.1. Parte Qsys	61
2.5.2. Parte QtCreator	63
2.6. Ensayos (con la implementación del handshake entre la FPGA y el HPS)	71
2.6.1. Medición de tiempos de lectura y de transmisión con datos empaquetados	en
protocolo ASTERIX CAT240	71
2.6.1.1. Tamaño de trama variable	73
3. Conclusiones	80
Anexo A: Placa DE1-SoC de Terasic	83
1. Placa DE1-SoC de Terasic	83
1.1. DE1-SoC de Terasic. Diagrama en bloque	83
1.2. DE1-SoC de Terasic. Layout	84
2. Cyclone V. Descripción general	85
2.1. Introducción al HPS de Cyclone V	85
2.2. Características del HPS	86
2.3. Interfaces HPS-FPGA	87
2.4. Proceso de booteo del HPS	87

3. Uso del Cyclone V. Información general	88
3.1. Proceso de booteo del HPS	88
3.2. Estructura del proyecto	89
Anexo B: Protocolo ASTERIX CAT240	90
Anexo C: Modelo OSI	94
1. Capa física	94
1.1. Capa de enlace de datos: Ethernet II	94
1.2. Capa de red: Trama IPv4	96
1.3. Capa de transporte	97
1.3.1. Trama UDP	97
1.3.1.1. Tamaño máximo del payload en del mensaje UDP	98
Agradecimientos	99
4. Referencias	100

Resumen

En este informe se describe el desarrollo e implementación de una red de distribución de vídeo RADAR en protocolo ASTERIX CAT240 sobre Ethernet Gigabit.

Este proyecto se realiza en el marco del proyecto Desarrollo de una consola de operaciones que pueda vincularse mediante una interfase de comunicación bidireccional al Sistema de Comando y Control de unidades de la Armada Argentina llevado adelante por el grupo Soluciones Embebidas Aplicadas de la Escuela de Oficiales de la Armada y del Servicio de Análisis Operativos, Armas y Guerra Electrónica de la Armada Argentina.

Palabras Clave: FPGA, HPS, RADAR, ASTERIX CAT240, handshake, Ethernet, Gigabit, UDP.

Lista de abreviaciones

ADC: Analog-to-Digital Converter. **ARM**: Advanced RISC Machine. **ASTERIX**: All purpose STructured Eurocontrol suRveillance Information eXchange. **API**: Application Programming Interface. AWM: Appliance Wiring Material. AXI: Advanced eXtensible Interfaces. **BI**: Bearing Increment. CAT8: CATegory 8. CAT240: CATegory 240. CPU: Central Processing Unit. CRC: Cyclic Redundancy Check. CS: Class Selector. DHCP: Dynamic Host Configuration Protocol. DMA: Direct Memory Access. **DSCP**: Differentiated Services Code Point. **ECN**: Explicit Congestion Notification. FCS: Frame Check Sequence. FPGA: Field-Programmable Gate Array. FSPEC: Field SPECification. FTP: Foiled Twisted Pair. GPIO: General Purpose Input/Output. GUI: Graphic User Interface. HM: Heading Marker. HPS: Hard Processor System. **IHL**: Internet Header Length. **IP**: Internet Protocol. LEN: LENght indicator. MAC: Media Access Control. MBPS: MegaBits Per Second. MF: More Fragments. MTU: Maximun Transmission Unit/Maximun Transfer Unit. **OS**: Operating System. **OSI**: Open Systems Interconnection. PC: Personal Computer. PHY: PHYsical layer. PLL: Phase-Locked Loop. **RADAR**: Radio Detection And Ranging. RAM: Random Access Memory. **RGMII**: Reduced Gigabit Media-Independent Interface. **RISC**: Reduced Instruction Set Computer. ROM: Read Only Memory. SAC: System Area Code. SIC: System Identification Code. **SD**: Secure Digital. SFD: Start Frame Delimiter. **SoC**: System on Chip. SRAM: Static Random Access Memory. **SSH**: Secure SHell.

Proyecto Final de Carrera Ingeniería Electrónica Allende, Agustin Emanuel Miguel Aguirre, Juan José

TCP: Transfer Control Protocol. TOS: Type Of Service. TTL: Time To Live. UDP: User Datagram Protocol.

Lista de figuras

Figura 1. Diagrama en bloques del sistema propuesto.	12
Figura 2. Diagrama en bloques del sistema propuesto. Partes del proyecto.	13
Figura 3. Diagrama en bloques del sistema propuesto. Parte del proyecto realizada por nosotros.	13
Figura 4. Diagrama de conexión generalizado. Dispositivos que forman parte de la red.	14
Figura 5. Creación del archivo ejecutable que se guardará en la tarjeta SD. Comandos empleados.	14
Figura 6 Vinculación de usuarios	15
Figura 7 Script create linux system sh Modificación de parámetros	16
Figura 8 Script create linux system sh Modificación de parámetros	16
Figura 9 Script create linux system sh Modificación de parámetros	16
Figura 10. Comando Isblk.	16
Figura 11. Creación del archivo ejecutable que se guardará en la tarjeta SD. Comandos	17
Eigura 12 Script create linux system sh Sacción u boot	1 /
Figura 12. Script create linux_system.sh. Seccion u-ooot.	10
Figura 13. Script create linux_system.sh. Archivos generados	19
Figura 14. Script create_initix_system.sn. Arcinvos generados.	19 20
Figura 16. Accesso a la EPGA utilizando SSH. Dirección IP asignada por DHCP	20
Figura 10. Acceso a la FFOA utilizado 5511. Dirección il asignada por Direr.	21
Figura 18 Kit de desarrollo DE1-SoC. Acceso a la /etc/network/interfaces	21
Figura 19. Accesso a la EPGA utilizando SSH. Dirección IP asignada por DHCP	22
Figura 20 Kit de desarrollo DE1-SoC Acceso a la /etc/ant/sources list Repositorios	
universe.	23
Figura 21. Software QtCreator. Configuración.	24
Figura 22. Software QtCreator. Configuración.	24
Figura 23. Software QtCreator. Configuración.	25
Figura 24. Software QtCreator. Configuración.	25
Figura 25. Software QtCreator. Configuración.	26
Figura 26. Software QtCreator. Configuración.	26
Figura 27. Software QtCreator. Configuración.	27
Figura 28. Software QtCreator. Configuración.	27
Figura 29. Software QtCreator. Configuración.	28
Figura 30. Software QtCreator. Configuración.	28
Figura 31. Software QtCreator. Configuración.	29
Figura 32. Software QtCreator. Configuración.	29
Figura 33. Ejecución del comando quartus &.	30
Figura 34. Compilación del proyecto en la herramienta Qsys del software Quartus. Erro	r
arrojado.	31
Figura 35. Partición de la tarjeta SD. Ubicación del preloader.	32

Figura 36. Grabado del preloader en la ruta de acceso correspondiente.	32
Figura 37. Qsys. Componentes instanciados.	33
Figura 38. QtCreator. Instanciación del header generado en Qsys.	34
Figura 39. QtCreator. Definición de todas las variables y funciones.	34
Figura 40. QtCreator. ID del archivo /dev/mem.	35
Figura 41. QtCreator. Mapeo de direcciones físicas a direcciones virtuales.	35
Figura 42. QtCreator. Librerías dedicadas al socket UDP.	35
Figura 43. QtCreator. Creación del socket.	35
Figura 44. QtCreator. Seteo de parámetros del socket creado.	36
Figura 45. QtCreator. Configuración para recibir.	36
Figura 46. QtCreator. Asociación del socket a una dirección o puerto específico.	36
Figura 47. QtCreator. Configuración para enviar o transmitir.	36
Figura 48. QtCreator. Manejo de archivos. Copiado de la trama ASTERIX CAT240 para acceder a la misma desde el HPS.	ι 37
Figura 49. QtCreator. Creación de uniones. Declaración de variables pertenecientes a la	
trama ASTERIX CAT240.	37
Figura 50. QtCreator. Declaración de variables.	38
Figura 51. QtCreator. Copiado de la trama ASTERIX CAT240 y posterior transmisión de	e
la misma por Ethernet a traves del socket UDP.	38
Figura 52. QtCreator. Calculo del data rate.	39
Figura 53. QtCreator. Copiado del contenido al que apunta sram_ptr al arreglo data mediante la instrucción memcpy().	39
Figura 54. QtCreator. Transferencia de parámetros específicos de la trama ASTERIX CAT240.	39
Figura 55. QtCreator. Envío de paquetes a través del socket.	39
Figura 56. QtCreator. Incremento de las variables.	39
Figura 57. QtCreator. Manejo de archivos. Cierre del archivo.	40
Figura 58. Comando iperf. Sintaxis y uso.	41
Figura 59. Consola de Linux. Comando iperf. Cliente.	42
Figura 60. Consola de Linux. Comando iperf. Servidor.	43
Figura 61. Consola de Linux. Ejecución código dma01. Tiempos de lectura y de transmisión.	44
Figura 62. Software Wireshark. Captura de tráfico.	45
Figura 63. Consola de Linux. Ejecución código dma01. Tiempos de transmisión.	46
Figura 64. Software Wireshark. Captura de tráfico.	47
Figura 65. Software Wireshark. Captura de tráfico.	48
Figura 66. Consola de Linux. Ejecución código dma01. Tiempos de transmisión.	48
Figura 67. Software RadarView. Visualización de paquetes transmitidos en protocolo ASTERIX CAT240.	49
Figura 68. Software Wireshark. Captura de tráfico.	50
Figura 69. Software Wireshark. Captura de tráfico. Recortado y reensamblado de paquet 50	tes.

Figura 70. Software Wireshark. Captura de tráfico. Recortado y reensamblado de paquetes.

51

51	
Figura 71. Consola de Linux. Ejecución código dma01. Tiempos de transmisión.	52
Figura 72. Software Wireshark. Captura de tráfico.	53
Figura 73. Software Wireshark. Captura de tráfico. Recortado y reensamblado de paquet 53	tes.
Figura 74. Software Wireshark. Captura de tráfico. Recortado y reensamblado de paquet 54	tes.
Figura 75. Software Wireshark. Captura de tráfico. Recortado y reensamblado de paquet 54	tes.
Figura 76. Consola de Linux. Ejecución código dma01. Modo debug vs modo release. Tiempos de transmisión	55
Figura 77 Consola de Linux Ejecución código dma01 Tiempos de transmisión	56
Figura 78 Bridges HPS-FPGA	57
Figura 79. Consola de Linux. Ejecución código dma01. Tiempos de transmisión.	57
Figura 80. Consola de Linux. Ejecución código dma01. Tiempos de transmisión.	58
Figura 81. Consola de Linux. Ejecución código dma01. Tiempos de lectura.	59
Figura 82. Osvs. Componentes instanciados.	62
Figura 83. QtCreator. Instanciación del header generado en Qsys.	63
Figura 84. QtCreator. Definición de todas las variables y funciones.	64
Figura 85. QtCreator. ID del archivo /dev/mem.	64
Figura 86. QtCreator. Definición de todas las variables necesarias para realizar el mapeo memoria.	o de 64
Figura 87. QtCreator. Mapeo de direcciones físicas a direcciones virtuales. Función	
mapping.	65
Figura 88. QtCreator. Librerías dedicadas al socket UDP.	66
Figura 89. QtCreator. Definición de todas las variables necesarias para la realización del socket UDP.	l 66
Figura 90. QtCreator. Creación del socket UDP. Función init_socket.	66
Figura 91. Llamado a funciones init_socket() y mapping(). Declaración de variables.	67
Figura 92. QtCreator. Lectura del Registro 0, adquisición de la trama ASTERIX CAT24 verificación de pérdida de paquetes, copiado y posterior transmisión de la trama por	0,
Ethernet a través del socket UDP y cálculo de la tasa de transferencia.	70
Figura 93. QtCreator. Copiado del contenido almacenado en memoria mem[num_mem] tamaño payload size al arreglo data mediante la instrucción memcpy().	de 71
Figura 94. QtCreator. Envío de datos a través del socket.	71
Figura 95. Consola de Linux. Ejecución código dma01. Verificación de pérdida de	
paquetes.	73
Figura 96. Consola de Linux. Ejecución código dma01. Tiempos de transmisión.	74
Figura 97. Software NetPerSec. Tasa de recepción de datos UDP en tiempo real.	75
Figura 98. Administrador de Tareas. Rendimiento. Tasa de recepción de datos UDP en tiempo real.	75
Figura 99. Software RadarView. Visualización de paquetes transmitidos en protocolo ASTERIX CAT240.	76
Figura 100. Software RadarView. Manual de Usuario. Parámetro	

ChanAPimMaxBytesPerSample.	76
Figura 101. Software RadarView. Modificación del parámetro	
ChanAPimMaxBytesPerSample en el archivo .rpi.	77
Figura 102. Software Wireshark. Captura de tráfico.	77
Figura 103. Software Wireshark. Captura de tráfico.	78
Figura 104. Software Wireshark. Tasa de recepción de datos UDP en tiempo real.	78
Figura 105. Software AsterixInspector. Análisis de paquete o trama empaquetada en protocolo ASTERIX CAT240.	79
Figura A.1. Kit de desarrollo DE1-SoC de Terasic. Diagrama en bloque.	83
Figura A.2. Kit de desarrollo DE1-SoC de Terasic. Layout. Parte posterior.	84
Figura A.3. Kit de desarrollo DE1-SoC de Terasic. Layout. Parte superior.	85
Figura A.4. Chip SoC Cyclone V 5CSEMA5F31C6. Porción FPGA y porción HPS.	86
Figura A.5. HPS. Diagrama en bloque.	87
Figura A.6. HPS. Flujos de arranque o booteo.	88
Figura A.7. Estructura del proyecto DE1_SoC_demo.	89
Figura B.1. Correspondencia entre el paquete ASTERIX CAT240 y su representación	
gráfica.	90
Figura B.2. Señales enviadas por el radar.	92
Figura B.3. Representación gráfica de los datos en base a las señales provenientes del	
radar.	93
Figura C.1. Capas del modelo OSI.	94
Figura C.2. Campos que conforman la trama Ethernet II.	95
Figura C.3. Campos que conforman la trama IPv4.	96
Figura C.4. DSCP y ECN.	96
Figura C.5. Campos que conforman la trama UDP.	98

Lista de tablas

Tabla 1. Tasa de transferencia obtenida para diferentes tamaños de trama o paquete.	60
Tabla 2. Capacidad del canal.	61
Tabla 3. Velocidad de lectura de la memoria compartida entre la FPGA y el HPS.	61
Tabla 4. Comparativa en la tasa de transferencia. Sin Handshake vs. Con handshake.	79
Tabla B.1. Protocolo ASTERIX CAT240. Conformación de la trama o paquete.	91

1. Introducción

La finalidad del proyecto consiste en diseñar e implementar una red para la distribución de video RADAR digitalizado y empaquetado en protocolo ASTERIX CAT240 a velocidades de Gigabit, o lo más cercano a ésta, utilizando UDP.

Para la implementación del sistema propuesto, luego de sortear diferentes alternativas, se optó por utilizar un SoC contenido dentro de una FPGA ya que es, en igual proporción, simple y económica además de ir acorde a los tiempos de desarrollo del proyecto en cuestión. La herramienta elegida es el kit DE1-SoC, el cual contiene la FPGA Intel Cyclone V 5CSEMA5F31.

Luego, de manera extensa y detallada, se describe la investigación realizada y el paso a paso seguido en el desarrollo para la correcta configuración y puesta en marcha del kit mencionado para comenzar así con la aplicación requerida.

Posteriormente, se detalla el software realizado para la comunicación entre la FPGA y el HPS, el cual se divide fundamentalmente en dos partes: una realizada en *Qsys*, en la cual se instancia todo el hardware necesario para la comunicación entre la FPGA y el HPS, y otra parte realizada en *QtCreator*, la cual se utiliza para programar el HPS.

En pocas palabras, el software realizado se encarga, en primera instancia, de mapear los puertos para lectura de memoria de la FPGA a memorias virtuales para que las mismas puedan ser accedidas desde el lado del HPS. Dicha memoria es una RAM dual port, que es la memoria compartida entre la FPGA y el HPS. Luego, se lee dicha memoria desde el HPS, la cual previamente fue cargada con datos empaquetados en protocolo ASTERIX CAT240 desde el lado de la FPGA, para, finalmente, transferir el contenido de la misma a un arreglo y poder así enviar dichos datos por Ethernet a través del socket UDP.

Por último, se detallan los ensayos realizados los cuales son pruebas de medición de tiempos. Dichas pruebas son realizadas con el fin de conocer los tiempos de lectura y de transmisión de datos empaquetados en protocolo ASTERIX CAT240 y poder así descubrir el tiempo máximo que el HPS necesita para leer un dato en memoria, copiarlo y transmitirlo por Ethernet a través de un socket UDP dando como resultado una velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real.

Las diferentes pruebas realizadas están relacionadas entre sí, es decir, siguen una línea temporal dado que la segunda es predecesora de la primera, y así sucesivamente con las diferentes pruebas llevadas a cabo.

Vale destacar que se realizaron dos tipos de software y dos tipos de ensayos: los primeros fueron sin la lógica de sincronización entre la FPGA y el HPS, es decir, sin el *handshake* mientras que los segundos fueron con la realización del *handshake* entre la escritura y lectura de la memoria compartida, donde la FPGA se encarga de escribir y el HPS de leer.

2. Descripción del proyecto

2.1. Propuesta de proyecto

La implementación elegida para la realización del sistema propuesto es un SoC contenido dentro de una FPGA. La herramienta por la que se optó es el kit de desarrollo DE1-SoC, el cual contiene la FPGA Intel Cyclone V 5CSEMA5F31C6.

A continuación, se observa un diagrama en bloques del sistema propuesto.



Figura 1. Diagrama en bloques del sistema propuesto.

Tal como se observa en la figura *ut supra*, la FPGA se compone de dos partes: una FPGA propiamente dicha y un HPS embebido en la FPGA.

La idea básica es, una vez que son capturadas las muestras de video RADAR crudo mediante un conversor AD conectado por GPIO a la FPGA y posteriormente empaquetadas en protocolo ASTERIX CAT240, realizar el *handshake* para que dichas muestras pasen de la FPGA al HPS y así poder realizar un empaquetamiento de las mismas en protocolo UDP para, finalmente, enviar el bloque a través de la red Gigabit Ethernet utilizando un bridge de comunicación entre la FPGA y el HPS.

En el Anexo B podrá encontrarse una breve reseña del protocolo ASTERIX CAT240. Por su parte, en el Anexo C podrá encontrarse un pequeño resumen del modelo OSI.

La ventaja principal es que el HPS ejecuta un OS dentro del mismo, un Linux (Ubuntu), el cual ya resuelve en su totalidad la parte de los protocolos de red.

A continuación, puede observarse en forma de diagrama las partes del proyecto en cuestión.



Figura 2. Diagrama en bloques del sistema propuesto. Partes del proyecto.

Más en detalle, se muestra en forma de diagrama la parte realizada por nosotros.



Figura 3. Diagrama en bloques del sistema propuesto. Parte del proyecto realizada por nosotros.

En resumen, las partes a resolver serían dos principalmente:

- Realización del *handshake* para la posterior adquisición de las muestras empaquetadas en protocolo ASTERIX CAT240.
- Empaquetado de dichas muestras en protocolo UDP dentro la FPGA, a su vez de implementar un bridge que se comunique con el HPS para así enviar dicho bloque a través de la red Gigabit Ethernet.

Tal como se mencionó anteriormente, el SoC, el cual está contenido dentro de una FPGA, formará parte de una red en la cual, dicho dispositivo, recibirá muestras de video RADAR crudo. Además, dicho dispositivo transmitirá muestras de video RADAR digitalizadas y empaquetadas en protocolo ASTERIX CAT240 a velocidades de Gigabit, o lo más cercana a ésta, utilizando UDP, a los dispositivos que las hayan solicitado.

Se detalla a continuación un diagrama de conexión generalizado de los dispositivos que forman parte de la red.



Figura 4. Diagrama de conexión generalizado. Dispositivos que forman parte de la red.

2.2. Puesta en marcha y configuración del kit DE1-SoC

Haciendo uso de la consola de Linux, se creó el archivo ejecutable, el cual se guardará en una tarjeta SD que luego será insertada en el kit de desarrollo DE1-SoC, con la finalidad de que el kit ejecute dicho archivo y pueda bootear así un OS Linux.

En el Anexo A, además de brindar una descripción general del kit de desarrollo mencionado, se describen las características del HPS, el proceso de booteo y la estructura del proyecto utilizada.

En la Figura 5 a continuación, se detallan los pasos seguidos, además de los comandos empleados en la consola de Linux, para realizar lo comentado previamente.



Figura 5. Creación del archivo ejecutable que se guardará en la tarjeta SD. Comandos empleados.

En primera instancia, se ejecutó el comando ./embedded_command_shell.sh, el cual se encarga de buscar todos los archivos ejecutables relacionados a Altera. Un detalle importante a tener en cuenta es que todas las instrucciones provistas en la guía de referencia[1], las cuales además se comentan y describen en la presente sección del informe, deben de ser ejecutadas posterior a ejecutar en la consola de Linux el ejecutable *Altera Embedded Comman Shell*, el cual se encuentra en el directorio *altera_install_directory/version/embedded/embedded_command_shell.sh*.

Una vez ejecutado el script, haciendo uso del comando ./create_linux_system.sh, se tuvieron dos errores relacionados a *yylloc*. La solución encontrada[2] radica en cambiar la línea *YYLTYPE yylloc* por la línea *extern YYLTYPE yylloc* en los archivos cuyos directorios se observan a continuación.

 $DE1_SoC_Demo \rightarrow sw \rightarrow hps \rightarrow linux \rightarrow source \rightarrow scripts \rightarrow dtc \rightarrow dtc-lexer.lex.c$ $DE1_SoC_Demo \rightarrow sw \rightarrow hps \rightarrow linux \rightarrow source \rightarrow scripts \rightarrow dtc \rightarrow dtc-lexer.lex.c_shipped$

Una vez corregidos dichos archivos, los errores relacionados a *yylloc* fueron solucionados por lo que pudo compilarse de manera exitosa. El error que continuaba apareciendo no era un error de compilación en sí, tal como el que se tenía previamente, sino que era un error debido a que la tarjeta SD no estaba insertada.

Hecho esto, fue necesario vincular el usuario a grupos pertenecientes a Debian 11 con el fin de, posteriormente, poder ejecutar el script. Los comandos empleados para tal fin son los observados en la Figura 6 a continuación.



Figura 6. Vinculación de usuarios.

El comando *id* es utilizado para ver a qué grupos pertenece el usuario. Por otra parte, el comando *cat /etc/group* es utilizado para ver qué grupos tiene asociados el usuario y su contenido. El grupo 20 (*dialout*), por ejemplo, es empleado para utilizar los periféricos de la FPGA a través de los puertos de la PC.

Posteriormente, fue necesario desloguearse y volver a loguearse para que los cambios realizados tengan efecto.

En el script *create_linux_system.sh* se modificó el parámetro *sdcard_partition_size_linux*, el cual se encuentra en la línea 66, y se lo igualó a *4096M*.



Figura 7. Script create_linux_system.sh. Modificación de parámetros.

Además, se debe corroborar que el parámetro *fpga_device_part_number*, el cual se encuentra en la línea 19, coincida con el nombre de la FPGA que se esté utilizando. En nuestro caso, y por default, es la FPGA 5CSEMA5F31C6.

```
19 fpga_device_part_number="5CSEMA5F31C6" # 5CSEMA4U23C6
```

Figura 8. Script create_linux_system.sh. Modificación de parámetros.

Se comentaron, además, las líneas que van desde la 495 a la 499 ya que la compilación del Linux se realiza por única vez. Será necesario descomentar las líneas 495 y 496 a medida que se realicen cambios en el proyecto de *Quartus*.

495	<pre>#compile_quartus_project</pre>
496	<pre>#compile_preloader</pre>
497	#compile_uboot
498	#compile_linux
499	<pre>#create_rootfs</pre>

Figura 9. Script create_linux_system.sh. Modificación de parámetros.

En la Figura 10 a continuación puede observarse la ejecución del comando *lsblk*, el cual muestra las particiones que se tienen en el dispositivo.

>								DE1_SoC_demo : bash — Konsole
Archivo	Editar Ve	er I	Marcadores	P	referenc	ias Ayuda		
siag00:	1@debian	-sia	ag001:~s	; 1:	sblk			
NAME	MAJ:MIN	RM	SIZE	R0	TYPE	MOUNTPOINT		
sda	8:0	0	894,3G	0	disk			
—sda1	8:1	0	243M	0	part	/boot/efi		
—sda2	8:2	0	343M	0	part	/boot		
—sda3	8:3	0	1K	0	part			
—sda5	8:5	0	22,4G	0	part	[SWAP]		
—sda6	8:6	0	33,2G	0	part	/		
_sda7	8:7	0	838,2G	0	part	/home		
sdb	8:16	1	14,4G	0	disk			
_sdb1	8:17	1	14,4G	0	part			
_sdb1	8:17	1	14,4G	0	part			

Figura 10. Comando lsblk.

La partición que nos interesa es *sdb*, que es la tarjeta SD que será insertada en el kit de desarrollo DE1-SoC, la cual almacenará el archivo ejecutable del OS Linux generado.

En resumen, los comandos necesarios para realizar en su totalidad el procedimiento descrito, es decir, crear el ejecutable del OS Linux y guardarlo en la tarjeta SD, son los observados en la Figura 11 a continuación.

DE1_SoC_demo : bash — Konsole
Archivo Editar Ver Marcadores Preferencias Ayuda
siag001@debian-siag001:~\$ cd altera siag001@debian-siag001:~/altera\$ cd 15.1 siag001@debian-siag001:~/altera/15.1\$ cd embedded siag001@debian-siag001:~/altera/15.1/embedded\$./embedded_command_shell.sh
Altera Embedded Command Shell
Version 15.1 [Build 185]
stag001@debian-siag001:~/altera/15.1/embedded\$ cd siag001@debian-siag001:~\$ cd altera siag001@debian-siag001:~/altera\$ cd DE1_SoC_demo siag001@debian-siag001:~/altera/DE1_SoC_demo\$ lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sda 8:0 0 894,36 0 disk
sda5 8:5 0 22.46 0 part [SWAP]
└─sda7 8:7 0 838,2G 0 part /home
sdb 8:16 1 14,46 0 disk
- cdb1 8:1/ 1 32M 0 part
Sold 8:19 1 10 0 part
<pre>siag001@debian-siag001:~/altera/DE1_SoC_demo\$./create_linux_system.sh /dev/sdb</pre>

Figura 11. Creación del archivo ejecutable que se guardará en la tarjeta SD. Comandos empleados.

El script /*create_linux_system.sh* realiza, de forma automática y tal como se observa en la Figura 12, las siguientes tareas:

- Compila el proyecto de *Quartus Prime*.
- Genera, configura y compila el *preloader*.
- Descarga, configura y compila el *u-boot*.
- Descarga, configura y compila el *Linux*.
- Descarga y configura el sistema de archivos raíz del Ubuntu Core.
- Particiona la tarjeta SD.
- Escribe la tarjeta SD.

Tal como puede observarse en la figura *ut supra*, se detalla todo el procedimiento seguido desde la ejecución del comando ./*embedded_command_shell.sh*, el cual busca todos los ejecutables relacionados a Altera, hasta la ejecución del comando ./*create_linux_system.sh*, script necesario para crear el ejecutable del OS Linux y guardarlo en la tarjeta SD que luego será insertada en la FPGA.

Algo a destacar es que el comando /*dev/sdb*, comando concatenado al comando ./*create_linux_system.sh*, es utilizado para guardar en la partición *sdb* el resultado de ejecutar el comando ./*create_linux_system.sh*, que sería el archivo ejecutable del OS Linux.

En este punto, se tuvo un error en la compilación del script *create_linux_system.sh* y no pudo crearse el *u-boot*. Por lo tanto, para solucionarlo, se debió entrar al script *create_linux_system.sh* e ingresar los comandos enunciados a continuación a través de la consola de Linux.

gvim create_linux_system.sh cd sw/hps/u-boot/ make distclean git checkout "b104b3dc1dd90cdbf67ccf3c51b06e4f1592fe91" less Makefile make "socfpga_cyclone5_config" less Makefile

Tal como se observa en la Figura 12 a continuación, el script *create_linux_system.sh* debería de hacerlo de forma automática pero, por alguna razón que se desconoce, no lo hace. De todas maneras, el haber ingresado los comandos mediante la consola de Linux soluciona el problema.



Figura 12. Script create_linux_system.sh. Sección u-boot.

Además, haciendo uso de la consola de Linux y empleando el comando *sudo apt install u-boot-tools*, se instalaron las herramienta necesarias para que la PC, una vez ejecutado el script *create_linux_system.sh*, cree y guarde en la tarjeta SD los archivos necesarios para que el kit DE1-SoC pueda bootear el OS.

Una vez realizado lo comentado anteriormente, se ejecutó nuevamente el script *create_linux_system.sh* mediante el comando *./create_linux_system.sh* /*dev/sdb* para que los archivos generados, los cuales se observan en la Figura 13 y en la Figura 14, se guarden en la tarjeta SD.

<u> </u>		a2		^ _ ¤ X
File Edit View Go	Help			
く > ^ 企	/home/siag001/altera/DE1_SoC_demo/sdcard/a2/			G
Places	Name	▼ Size	Туре	Date Modified
📕 Computer	preloader-mkpimage.bin	256.0 KiB	unknown	Today
🏫 siag001				
🛄 Desktop				
蒏 Trash				
🛅 Documents				
la Music				
🛅 Pictures				
lo Videos				
🛅 Downloads				
Devices				
🔜 File System				
Network				
🟪 Browse Netw				
	1 file: 256.0 KiB (262,144 bytes), Free space: 733.5 GiB			

Figura 13. Script create_linux_system.sh. Archivos generados.

	fat32				^ _ ¤ X			
File Edit View Go	Help							
く > ^ 企 🖿	home/siag001/altera/DE1_SoC_demo/sdcard/fat32/				G			
Places	Name	▼ S	ize	Туре	Date Modified			
📕 Computer	socfpga.dtb		16.8 KiB	unknown	Today			
🏫 siag001	socfpga.rbf		6.7 MiB	plain text document	Today			
🛄 Desktop	u-boot.img		436.2 KiB	Raw disk image	Today			
📷 Trash	u-boot.scr		1.7 KiB	unknown	Today			
Documents	in zimage		3.4 MiB	unknown	Today			
a Music								
Pictures								
Downloads								
Devices								
File System								
Network								
Browse Netw								
0628								
	2 LIGE: TO'2 WIR (TO'3AA'055 DÁGES)' HEG EBDEG: 123'2 PIR							

Figura 14. Script create_linux_system.sh. Archivos generados.

La ejecución del script y posterior creación de archivos, tal como se observa de las figuras *ut supra*, fue realizada con éxito.

Por lo tanto, se extrajo la tarjeta SD y posteriormente se la insertó en la FPGA.

Acto seguido, se encendió la FPGA y se la conectó a la PC mediante una interfaz serial, haciendo uso del terminal *CuteCom*, para poder así comunicarse con ésta. El usuario (*username*) empleado es *sahand* mientras que la contraseña (*password*) es *1234*, tal como indica la guía de referencia utilizada[1].

	CuteCom - Default			\vee \diamond \otimes
Sessions Help				
Close Device: /dev/ttyUSB0 ~				Settings
ш				
k				
clear				
logout				
CIS caband				
1234				
Input		LF \lor Char delay:	0 ms 🗘 Send file	Plain ~
<pre>>> 12112:56:023 DE1-Soc login: sahand*> 12112:56:023 DE1-Soc login: sahand*> 12113:05:67] Welcome to Ubuntu 1.40:45:UTS (GNU/Linux 4.6:0-rc2-dirty armv7l)*> 12113:05:608 *> 12113:05:608 *> 12113:05:608 *> 12113:05:602 sahand@DE1-Soc:-\$ </pre>				
Clear Hex output Logging to:				
and all successing and all appendix and all a				

Figura 15. Conexión mediante interfaz serial entre la PC y la FPGA.

Tal como puede observarse en la Figura 15, el OS Linux Ubuntu fue booteado con éxito en la FPGA.

Luego, utilizando cables de red, se conecta tanto el kit DE1-SoC como la PC al switch para así poder tener acceso a la red de trabajo que, en nuestro caso, es *SIAG_INGE*.

Un detalle a tener en cuenta es que la conexión de dispositivos a la red de trabajo, en este caso la PC con la que se realiza toda la configuración, debe ser cableada ya que, si la misma es realizada de manera inalámbrica, la latencia es muy alta lo que dificulta en demasía la configuración por terminal.

Utilizando el terminal *CuteCom*, se instaló el *SSH Server* en el kit mediante el comando *sudo apt install openssh-server*. Posteriormente, se pide una contraseña, la cual es *1234*. Con el servidor SSH se accede de manera remota al kit DE1-SoC desde la PC.

Por la manera en la que está configurado el kit de desarrollo, una vez que el mismo se conecta a la red de trabajo, se le asigna una dirección IP por DHCP. Por lo tanto, haciendo uso de la terminal *CuteCom*, empleando el comando *ip addr*, puede conocerse la dirección IP asignada al kit de desarrollo. En nuestro caso, la misma es 192.168.15.170. Luego, desde la PC, se le da click derecho al ícono de red que se encuentra en la barra de tareas y se accede a *Edit Connections* Hecho esto, en la sección *Ethernet*, se ingresa a *Wired Connection* y en la sección *IPv4 Settings* se agrega una dirección IP estática. En nuestro caso, la misma es 192.168.7.20. El poder asignarle dos direcciones IP a un mismo dispositivo es una gran ventaja en Linux.

Una vez que fue instalado con éxito el *SSH Server* se debe ingresar a la consola de Linux y, por medio del comando *ssh sahand@192.168.15.170*, acceder al kit DE1-SoC.

Un detalle a aclarar es que la primera vez que se realiza este procedimiento, tal como se observa en la Figura 16 a continuación, se pregunta si se desea realizar la conexión. En caso afirmativo, aceptar ingresando por consola la palabra *yes* (no ingresar *y*, como suele ser habitualmente). Posteriormente, se solicitará una contraseña, la cual es *1234*.



Figura 16. Acceso a la FPGA utilizando SSH. Dirección IP asignada por DHCP.

El objetivo de la presente configuración es asignarle al kit DE1-SoC una dirección IP mediante DHCP y otra dirección IP de manera estática, de igual manera que se realizó con la PC. La dirección IP asignada por DHCP es la utilizada para tener acceso a Internet en ambos dispositivos y realizar así las descargas y actualizaciones necesarias mientras que la dirección IP fija o estática es la utilizada para la comunicación entre dispositivos a la hora de realizar las configuraciones.

Por lo tanto, haciendo uso de la consola de Linux y empleando el comando *sudo nano* /*etc/network/interfaces*, se configuran las interfaces del kit DE1-SoC.



Figura 17. Kit de desarrollo DE1-SoC. Acceso a la /etc/network/interfaces.



Figura 18. Kit de desarrollo DE1-SoC. Acceso a la /etc/network/interfaces.

Finalmente, se deben reiniciar los dispositivos para que la configuración realizada tenga efecto. Para ello, se emplea el comando *sudo reboot* a través de la consola de Linux.

En este punto, ya puede accederse al kit DE1-SoC mediante SSH utilizando la dirección IP fija o estática, tal como se observa en la Figura 19 a continuación.



Figura 19. Acceso a la FPGA utilizando SSH. Dirección IP asignada por DHCP.

Empleando la consola de Linux, mediante el comando *sudo apt install mc htop build-essential*, se instalaron las siguientes herramientas en el kit de desarrollo DE1-SoC:

- *mc*, el cual es un administrador de archivos.
- *htop*, el cual es un administrador de procesos.

 build-essential, las cuales son todas las herramientas de compilación necesarias. En pocas palabras, instala lo básico para poder comenzar a escribir y así programar en Lenguaje C.

Una vez comenzada la instalación de dichas herramientas, la misma es interrumpida por el error *Unable to locate*. Para solventar dicho inconveniente, en primera instancia, se empleó el comando *sudo apt update* por medio de la consola de Linux, el cual es utilizado para corroborar si existe alguna actualización disponible para así instalarla. Hecho esto, se intentó nuevamente instalar dicha herramientas pero el error subsistía. Luego, por medio del comando *sudo nano /etc/apt/sources.list*, se habilitaron los repositorios para que el sistema pueda buscar y descargar las actualizaciones requeridas.



Figura 20. Kit de desarrollo DE1-SoC. Acceso a la /etc/apt/sources.list. Repositorios universe.

Tal como puede observarse de la figura *ut supra*, se habilitaron (se descomentaron eliminando el #) los últimos 4 repositorios *universe*.

Hecho esto, se volvió a ejecutar el comando *sudo apt update* seguido del comando *sudo apt install mc htop build-essential* para instalar así las herramientas requeridas con éxito y sin error alguno. Además, por medio del comando *sudo apt install rsync*, se instaló una herramienta para transferencias de archivos, la cual es muy utilizada para realizar back-up.

Posteriormente, volviendo a hacer uso de la consola de Linux, se realizó la instalación de dos software sobre la PC: el primero, a través del comando *sudo apt install cmake*, fue *CMake*. Dicho software genera los archivos de compilación de otro sistema. En nuestro caso, será utilizado en conjunto con el software *QtCreator*. El segundo software instalado fue *QtCreator*, haciendo uso del comando *sudo apt install qtcreator*. Una vez instalado, tuvo que ser configurado para así poder vincular y ejecutar los archivos en el kit DE1-SoC. Para ello, se instalaron el compilador y el depurador haciendo uso de la consola Linux y de los comandos *sudo apt install g++-9-arm-linux-gnueabihf* y *sudo apt install gbd-multiarch* respectivamente. El compilador es propio para el procesador que se tiene en el kit de desarrollo DE1-SoC, el cual crea los archivos binarios mediante el software *QtCreator* y los manda al kit. Por su parte, el depurador del software *QtCreator* se conecta por red al

depurador del kit. Por otro lado, mediante el comando *sudo apt install gdbserver*, se instaló en el kit un servidor que se ejecuta con el archivo *.bin* y el número de puerto. Dicho servidor espera y ejecuta comandos de una PC. La ventaja que se tiene con esto es que se pueden mandar comandos por consola y ejecutarlos desde el kit de desarrollo DE1-SoC.

Posteriormente, una vez realizada las instalaciones, mediante el comando *sudo reboot*, reiniciamos el OS para que los cambios tengan efecto.

Un detalle importante a mencionar es que se debe habilitar el usuario *root* en *SSH*[3][4] debido a que, una vez que se ejecute el programa posterior a haber realizado toda la configuración necesaria explicada a continuación, se pedirá acceso a lugares con privilegios como, por ej. a la memoria DMA. Es por esta razón que es necesario brindar acceso *root*. Esta configuración debe realizarse del lado del servidor, es decir, debe hacerse del lado del kit DE1-SoC.

C	Options — Qt Creator		~ >
Filter	Devices		
🖬 Kits	Devices Android QNX SSH		
Environment	Device: Local PC (default for Desktop)	•	Add
Text Editor			
🔣 FakeVim	General		Set As Default
Ø Help	Name: Local PC		
{} C++	Type: Desktop		Show Running Processes
🖈 Qt Quick	Auto-detected: Yes (Id Is "Desktop Device")		
➤ Build & Run			
🖽 Qbs	Type Specific		
Debugger			
🖍 Designer			
Python			
E Analyzer			
Version Control			
Devices	v		
		v	Apply 😫 Cancel 🖉 OK

Una vez iniciado el software *QtCreator*, nos dirigimos a *Tools* \rightarrow *Options*.

Figura 21. Software QtCreator. Configuración.

En la parte de *Devices*, en la viñeta *Devices*, se debe de agregar un dispositivo mediante la opción *Add*. El tipo de dispositivo que se debe seleccionar es *Generic Linux Device*.

Devices		
Devices Android QNX SSH		
Device: Local PC (default for Desktop)	C Device Configuration Wizard Selection — Q 🔺 X	- <u>A</u> dd
General Name: Local PC Type: Desktop Auto-detected: Yes (id is "Desktop D Current state: Unknown	Available device types: Generic Linux Device QNX Device	Remove Set As Default Show Running Processes
	Cancel Start Wizard	

Figura 22. Software QtCreator. Configuración.

A continuación, se le debe colocar un nombre al dispositivo. En nuestro caso, se le asignó el nombre *DE1-SoC*. Luego, le colocamos la dirección IP fija o estática asignada anteriormente que, es nuestro caso, es 192.168.7.40 y, posteriormente, un nombre de usuario para loguearse al dispositivo. En nuestro caso, el nombre de usuario empleado es *root*.

QC	New Generic Linux Device Configurat	ion Setup — Qt Creator	^ X
Connection			
Connection Key Deployment Summary	The name to identify this configuration: The device's host name or IP address: The username to log into the device:	DE1_SoC 192.168.7.40 root	
			<u>N</u> ext > Cancel

Figura 23. Software QtCreator. Configuración.

Hecho esto, se debe crear una *key*. Para ello, se debe clickear en *Create New Key Pair*. Al hacerlo, se despliega una ventana, en la cual debemos seleccionar *Generate And Save Key Pair*.

Q	SSH Key Configuration — Qt Creator	^	×
9	Options		
	Key algorithm: <u>R</u>SA ECDSA 		
	Key <u>s</u> ize: 1024 -		
	Private key file: /home/siag001/.ssh/qtc_id Browse		
	Public key file: /home/siag001/.ssh/qtc_id.pub		
	<u>G</u> enerate And Save Key Pair <u>C</u> ancel		

Figura 24. Software QtCreator. Configuración.

Posteriormente, dicha ventana se cierra y, en la ventana en la que se estaba previamente, se debe seleccionar *Deply Public Key*.

QC	New Generic Linux Device Configuration Setup — Qt Creator	^ X
Key Deployment		
Connection Key Deployment Summary	We recommend that you log into your device using public key authentication. If your device is already set up for this, you do not have to do anything here. Otherwise, please deploy the public key for the private key with which to connect in the f If you do not have a private key yet, you can also create one here. Private key file: siag001/.ssh/qtc_id Browse Create New Key Pair Deploy Public Key	uture.
	< <u>B</u> ack <u>N</u> ext >	Cancel

Figura 25. Software QtCreator. Configuración.

Al hacerlo, aparece una ventana emergente en la cual se solicita la contraseña del nombre de usuario empleado, la cual es *1234*.

qtc-askpass	^	×
Password required for SSH log sahand@192.168.7.40's passv	in. vord	:
••••		
<mark>⊗</mark> <u>C</u> ancel	<u>0</u> K	

Figura 26. Software QtCreator. Configuración.

Si todo el procedimiento descrito anteriormente fue realizado con éxito, aparecerá una ventana al respecto. Para ir finalizando, se debe clickear en *Next* seguido de *Finish*. Luego, de forma automática, se realiza un test de conectividad.

QC	Device Test — Qt Creator	^ X
Connecting to host Checking kernel ve Linux 4.6.0-rc2-dirt	i ersion ty armv7l	
Checking if specifie All specified ports	ed ports are available are available.	
Checking whether SFTP service availa	an SFTP connection can be set up able.	
Checking whether rsync is functional.	rsync works	
Device test finis	ned successfully.	
		Close

Figura 27. Software QtCreator. Configuración.

Finalmente, clickeamos en Apply.

6		Options — Qt Creator		^
Filter	Devices			
🖬 Kits	Devices Android QN	NX SSH		
Environment	Device: DE1 SoC (defau	It for Generic Linux)	•	Add
Text Editor	General		A	Bomovo
🔣 FakeVim		-		Remove
Ø Help	Name: DEI_S			Set As Default
{} C++	Type: Generi	c Linux		Test
🖈 Qt Quick	Current state: Unknow	Current state: Unknown S		
➤ Build & Run	Type Specific	Type Specific		
🖽 Qbs	Machine type:	Physical Device		Open Remote Shell
Debugger	Authentication type:	O Default		
✓ Designer	Host name:	192 168 7 40 SSH port: 22 A V Check host key		
🕏 Python	<u>n</u> oschame.			
Analyzer	Free ports:	10000-10100 Timeout: 10s 📮		
Version Control	<u>U</u> sername:	sahand		
Devices	Private key file:	siag001/.ssh/qtc_id Browse Create New	v	
Devices	 Private key file: 	siag001/.ssh/qtc_id Browse Create New		

Figura 28. Software QtCreator. Configuración.

En la parte de *Kits*, en la viñeta *Debuggers*, se debe agregar un debugger mediante la opción *Add*.

			Options — Qt Creator		
ter	Kits				
Kits	Kits Qt Version	s Compilers Debug	jers CMake		
) Environment	Name	Location	Туре	Ar	Add
Text Editor	 Auto-detected System GDF 	B at /usr/bip/gdb /usr/bip	adb GDB	Cic	one
FakeVim	System GDE • Manual	B at /bin/gdb /bin/gdb	GDB	Rem	nov
C++	gdb-multiar	ch /bin/gdł	-multiarch GDB		
Qt Quick					
Build & Run	Name:	gdb-multiarch			
Qbs	Path:	/bin/gdb-multiarch		Browse	
Debugger	Type:				
Designer	ABIs:	x86-linux-generic-elf			
Python	Version:				
Analyzer	Working directo			Browse	
Version Control	working directo			Diowse	

Figura 29. Software QtCreator. Configuración.

En la parte de *Kits*, en la viñeta *Compilers*, debe de aparecer el compilador que fue instalado previamente.

QC	Options — Qt Creator		~ X
Filter	Kits		
🖾 Kits 📤	Kits Qt Versions Compilers Debuggers CMake		
🖵 Environment	Name	Type 🔺	- Add
Text Editor	 Auto-detected C 		Clone
🔏 FakeVim	GCC (C, x86 64bit in /usr/bin)	GCC	Remove
Ø Help	GCC (C, arm 32bit in /usr/bin)	GCC	Remove All
{} C++	GCC 9 (C, arm 32bit in /usr/bin)	GCC	Re-detect
🖪 Qt Quick	GCC (C++, x86 64bit in /usr/bin)	GCC 👻	Auto detection Settings
➤ Build & Run			Auto-detection Settings
🖽 Qbs	Name	î	
Debugger			
/ Designer	Compiler path: //usi/bin/arm-inux-gnueabim-gcc-9		
🕏 Python	Platform codegen flags:		
E Analyzer	Platform linker flags:		
Version Control	ABI: arm-linux-generic-elf-32t * arm * - linux * - generic * - elf	▼ - 32bit ▼ ▼	
Y		v	Apply Ocancel

Figura 30. Software QtCreator. Configuración.

Una vez realizada la configuración comentada, en la parte de *Kits*, en la viñeta *Kits*, se debe asociar al *device* creado inicialmente con el compilador y el debugger.

<u>i</u> G		Options – Qt Creator		^
Filter	Kits			
🖾 Kits	Kits Qt Versions Comp	ilers Debuggers CMake		
Environment	Name		Ad	d
Text Editor	Auto-detected ▼ Manual		Clo	ne
🐰 FakeVim	DE1_SoC Desktop (default)		Rem	ove
Help			Make D	efault
{} C++			Settings	Filter
🖈 Qt Quick		Defi	ault Sett	ings Filter
Build & Run	Name:	DE1_SoC		
🖽 Qbs	File system name:			
Debugger	Device type:	Generic Linux Device		
/ Designer	Device:	DE1 SoC (default for Generic Linux)	-	Manage
Python	Systoot			Browse
E Analyzer	5,5,0000	C: GCC 9 (C arm 32bit in /usr/bin)		bronoeni
Version Control	Compiler:	C++: GCC 9 (C++ arm 32bit in /usr/bin)		Manage
Devices	Environmont:			Chango
	Debugger	adh multiarah		Change
Tanguage Client	Debugger:		•	Manage
iz lesting	Qt version:	None	•	Manage
	Qt mkspec:			
	Additional Qbs Profile Setting	js:		Change
	CMake Tool:	System CMake at /usr/bin/cmake	*	Manage
	CMake generator:	<none> - Unix Makefiles, Platform: <none>, Toolset: <none></none></none></none>		Change
	CMake Configuration:	CMAKE_CXX_COMPILER:STRING=%{Compiler:Executable:Cxx}; CMAKE_C_COMPILER:STRING=%{Compiler:Executable:Cxx}; CMAKE_C_C_COMPILER:STRING=%{{Compiler:Executable:Cxx}; CMAKE_C_C_COMPILER:STRING=%{{Compiler:Executable:Cxx}; CMAKE_C_C_COMPILER:STRING=%{{Compiler:Executable:Cxx}; CMAKE_C_C_COMPILER:STRING=%{{Compiler:Executable:Cxx}; CMAKE_C_C_COMPILER:STRING=%{{Cmpiler:Executable:Cxx}; CMAKE_C_C_COMPILER:STRING=%{{CmpileR:Cxx}; CMAKE_C_C_C_COMPILER:STRING=%{{CmpileR:Cxx}; CMAKE_C_C_C_C_C_C_C_C_C; CMAKE_C_C_C_C_C_C_C_C_C_C_C_C_C_C_C_C_C_C_C	able:	Change 👻

Figura 31. Software QtCreator. Configuración.

Posteriormente, para que el debugger pueda ser ejecutado, se realiza la configuración mostrada a continuación.

		inanie (annos finasce). 4t ereater
<u>File</u> <u>E</u> dit	<u>V</u> iew <u>B</u> uild <u>D</u> ebug <u>A</u> nalyze	Iools <u>W</u> indow <u>H</u> elp
Welcome	Manage Kits	Run Settings
Edit A	ctive Project	Deployment Method: Deploy to Remote Linux Host Add Remove Rename
Design	Import Existing Build	Files to deploy: Override deployment data from build system
^{bebug} B	uild & Run	Local File Path Remote Directory Add
ojects	■ DE1_SoC	
lelp	Run G Desktop	Install into temporary host directory Details *
Р	roject Settings	Check for free disk space Details * Kill current application instance
	Editor Code Style	Deploy files via rsync Details *
	Dependencies Environment Clang Code Model Clang Tools	Add Deploy Step ~ Run
na01	lesting	Run configuration: dma01 (on DE1_SoC) Add Remove Rename Clone
		Executable on device:
Debug		Alternate executable on device: /root/source/dma01 Use this command instead

Figura 32. Software QtCreator. Configuración.

Una vez abierto el proyecto *dma01*, utilizando el software *QtCreator*, se accede a *CMakeLists.txt* para setear la ruta de acceso en donde, una vez compilado y realizado el *deploy* del proyecto, se guarda el ejecutable del mismo en el kit DE1-SoC. Una vez que se abre el *QtCreator*, el archivo *CMakeLists.txt* se ejecuta automáticamente, definiendo así la ruta mencionada.

Luego, se debe ingresar a la consola de Linux y por medio del comando *ssh* root@192.168.7.40 se accede al kit DE1-SoC. Una vez ahí, mediante el comando *mkdir*, se

crea la carpeta o directorio *source* que es en donde se guarda el ejecutable creado por *QtCreator* luego de haber compilado el proyecto y realizado el *deploy* del mismo.

Para comenzar con la configuración del HPS, se ejecutó el *Qsys*. El mismo es abierto haciendo uso de la consola Linux, mediante el comando *quartus* &, tal como se observa en la Figura 33 a continuación.



Figura 33. Ejecución del comando quartus &.

En pocas palabras, el *Qsys* es una herramienta gráfica que se utiliza para el diseño digital de hardware. Dicha herramienta contiene procesadores, memorias, interfaces de I/O, timers, etc. La herramienta *Qsys* le permite al usuario, mediante una GUI, elegir los componentes deseados y automáticamente genera el hardware para conectar dichos componentes.

En este punto, se tuvo un error al momento de compilar el proyecto, tal como se observa en la Figura 34 a continuación.

<u></u>	Generate Completed	~
AI 😫 🔺 🕕		
Progress: Building connections		
Progress: Parameterizing connections		
O Progress: Validating		
Progress: Done reading input file		
Info: soc_system.hps_0: HPS Main PLL count	er settings: n = 0 m = 73	
Info: soc_system.hps_0: HPS peripherial PLL	. counter settings: n = 0 m = 39	
A Warning: soc_system.hps_0: "Configuration	HPS-to-FPGA user 0 clock frequency" (desired_cfg_cik_mhz) requested 100.0 MHz, but only achieved 97.368421 MHz	
A Warning: soc_system.hps_0: 1 or more output	It clock frequencies cannot be achieved precisely, consider revising desired output clock frequencies.	
Info: soc_system.pio_0: PIO inputs are not ha	rdwired in test bench. Undefined values will be read from PIO inputs during simulation.	
Info: soc_system.sysid: System ID is not assi	gned automatically. Edit the System ID parameter to provide a unique ID	
Info: soc_system.sysid: Time stamp will be a	utomatically updated when this component is generated.	
Info: soc_system.system_pll: The legal refere	ince clock frequency is 5.0 MHz. 800.0 MHz	
Info: soc_system.system_pll: Able to implem	ant PLL with user settings	
A Warning: soc_system.dma_0: Interrupt sende	vr dma_0.irq is not connected to an interrupt receiver	
A Warning: soc_system.sysid: sysid.control_sl	ave must be connected to an Avalon-MM master	
Info: soc_system: Generating soc_system "s	ac_system" for QUARTUS_SYNTH	
Info: Interconnect is inserted between master	nps_0.h2f_axi_master and slave onchip_memory2_0.s2 because the master is of type axi and the slave is of type avalon.	
Info: Interconnect is inserted between master	Jma_0.read_master and slave hps_0.12h_axi_slave because the master is of type avalon and the slave is of type axi.	
Info: Interconnect is inserted between master	Jma_0.write_master and slave onchip_memory2_0.s1 because the master has address signal 18 bit wide, but the slave is 14 bit wide.	
Info: Interconnect is inserted between master	Jma_0.write_master and slave onchip_memory2_0.s1 because the master has waitrequest signal 1 bit wide, but the slave is 0 bit wide.	
Info: dma_0: softresetEnable = 1		
Info: dma_0: Starting RTL generation for module	ile 'soc_system_dma_0'	
Info: dma_0: Generation command is [exec /h	sme/siag001/altera/15.1/quartus/linux64//per/l/bin/per/l-i/home/siag001/altera/15.1/quartus/linux64//per/l/bib-l/home/siag001/altera/15.1/quartus/sopc_builder/bin/europa-l/home/siag001/altera/15.1/quartus/linux64//per/l/bib-l/per/l/bib-) -I /home
Info: dma_0: # 2023.02.13 11:26:13 (*) soc_sy	stem_dma_0: allowing these transactions: word, byte_access	
Info: dma_0: perl: warning: Setting locale faile	i.	
Info: dma_0: perl: warning: Please check that	your locale settings:	
Info: dma_0: LANGUAGE = "es_AR:es",		=
Info: dma_0: LC_ALL = (unset),		
Info: dma_0: LC_CTYPE = "en_US.UTF-8",		
Info: dma_0: LANG = "C"		
Info: dma_0: are supported and installed on years	Jur system.	
Info: dma_0: perl: warning: Falling back to the	standard locale ("C").	
😢 Error: dma_0: Failed to generate module soc_	system_dma_0	
Info: dma_0: Done RTL generation for module	'soc_system_dma_0'	
Info: dma_0: "soc_system" instantiated altera	_avalon_dma *dma_0"	
😢 Error: Generation stopped, 16 or more module	is remaining	
Info: soc_system: Done "soc_system" with 1:	3 modules, 5 files	
S Error: qsys-generate failed with exit code 1: 2 E	Errors, 4 Warnings	
🕕 Info: Finished: Create HDL design files for syn	rthesis	¥
A		•
😢 Generate: completed with errors and warnings		
	Stop	Close

Figura 34. Compilación del proyecto en la herramienta Qsys del software Quartus. Error arrojado.

Tal como puede observarse de la figura *ut supra*, los errores son debidos al *locate*, es decir, al idioma (el OS de la PC estaba en Español y el *Qsys* lo requería en Inglés). Por lo tanto, para solventar dicho inconveniente, haciendo uso de la consola de Linux, se cambió el idioma[5] del OS Debian 11. Posteriormente, se reinició el mismo para que los cambios surtan efecto. Una vez reiniciado el equipo, la interfaz gráfica del OS no arrancaba. Luego de realizados varios intentos para solucionar la problemática enfrentada, se optó por cambiar de entorno gráfico[6]. En este punto, volvió a reiniciarse el OS. Una vez iniciado el mismo, se observó el cambio en el idioma. Luego, se generó el subsistema (*SoC subsystem*) en *Qsys* y, posteriormente, se lo instanció en el proyecto.

Hecho esto, se volvió a compilar de manera exitosa el *Qsys* por lo que, en este punto, ya se puede empezar a trabajar sobre el HPS.

Como se necesita al HPS para poder acceder a los periféricos que forman parte de la estructura de la FPGA, se debe de generar un *header* el cual, posteriormente, debe ser instanciado en el proyecto de *Qsys*. Para ello, se necesita entrar a la carpeta *cd/altera/DE1-SoC_demo/hw/quartus* y ejecutar, mediante la consola de Linux, el comando *sopc-create-header-files soc_system.sopcinfo --single hps_soc_system.h --module hps_0*. Tal como puede observarse, el nombre del *header* generado es *hps_soc_system.h*.

Un detalle importantísimo a tener en cuenta, previo a realizar cualquier ensayo, es que se debe de poner el switch MSEL, el cual se encuentra en la parte inferior del kit de desarrollo DE1-SoC, en "00000"[1].

Además, cada vez que se desea realizar un ensayo sobre el kit de desarrollo DE1-SoC, se debe grabar la tarjeta SD con el *preloader* y el archivo *.rbf*. Por su parte, el archivo *.rbf* es el archivo con el que el HPS programa la FPGA. El software *Quartus* genera el *SRAM Object File* (*.sof*), archivo necesario para programar la FPGA. El archivo *.sof* emplea el *header*, el cual debe volver a generarse solo sí se realizó algún cambio en el *Qsys*. Como nosotros trabajamos con el HPS, necesitamos el archivo *RAW Binary File* (*.rbf*) para programar la FPGA. La conversión del archivo *.sof* a *.rbf* la realiza de forma automática,

una vez que es ejecutado, el script *create_linux_system.sh* pero debe de grabarse a mano en la tarjeta SD.

Tal como se observa en la figura *ut infra*, la partición de la tarjeta SD que contiene el *preloader* es la *sdb3*. Puede notarse que la misma tiene el identificador a2 y el tipo es *unknown*.



Figura 35. Partición de la tarjeta SD. Ubicación del preloader.

Por su parte, en la Figura 36 a continuación se observa la ruta en donde debe de grabarse el *preloader* además del comando utilizado para hacerlo, el cual es *sudo dd if=preloader-mkpimage.bin of=/dev/sdb3 bs=64k seek=0*, seguido del comando *sync* que es utilizado para transferencia de archivos.



Figura 36. Grabado del preloader en la ruta de acceso correspondiente.

2.3. Software (sin la implementación del handshake entre la FPGA y el HPS)

En esta sección del informe se comenta en detalle el software realizado para la comunicación entre la FPGA y el HPS, el cual se divide fundamentalmente en dos partes: una realizada en *Qsys*, en la cual se instancia todo el hardware necesario para la comunicación entre la FPGA y el HPS, y otra parte realizada en *QtCreator*, la cual se utiliza para programar el HPS.

A grandes rasgos, el software realizado se encarga, en primera instancia, de mapear las direcciones de la FPGA a una memoria virtual para que las mismas puedan ser accedidas desde el lado del HPS. Luego, se escribe la memoria de la FPGA en su totalidad (8192 [bytes]) con datos fijos empaquetados en protocolo ASTERIX CAT240 y posteriormente se la lee desde el HPS. Luego, se transfiere el contenido de la memoria a un arreglo para, finalmente, enviar dichos datos por Ethernet a través del socket UDP.

2.3.1. Parte Qsys

El *Qsys* es una herramienta gráfica que se utiliza para el diseño digital de hardware. Dicha herramienta contiene procesadores, memorias, interfaces de I/O, timers, etc. La herramienta *Qsys* le permite al usuario, mediante una GUI, elegir los componentes deseados y automáticamente genera el hardware para conectarlos.

La herramienta *Qsys* permite diseñar el sistema y la comunicación entre el HPS y la FPGA. El HPS y la FPGA están conectadas a través de una serie de *AXI Bridges*. En este caso, nosotros trabajaremos desde el lado del HPS por lo que, para la comunicación entre ellos, se utilizan dos *bridges*:

- HPS-to-FPGA bridges (*h2f*).
- Lightweight HPS-to-FPGA bridge (*lwh2f*).

*				Qsys - soc_system.qsys (/ho	ome/siag001/altera/DE1_SoC_d	emo/hw/quartus/s	soc_system.qsys)				^ _ ¤ >
Eile B	dit S	/stem <u>G</u> enerate	⊻iew <u>T</u> ools <u>H</u> elp								
) 📇 IF	P Catalı	og 🛏 🗗 📜 👬 N	Wessages ⊨ Ø								
‡⊡ s	ystem	Contents 🛛									- 5 0
[System: soc_	system Path: clk_0								
-	Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode Nat
	V		⊟ clk_0	Clock Source							
			clk_in_reset clk_in_reset clk_ clk_reset	Clock Input Reset Input Clock Output Reset Output	cik reset Double-click to export Double-click to export	exported clk_0					
			tel 92 monoy memory h2_reset h2_rad_clock h2_rad_clock 12h_ad_clock 12h_ad_clock 12h_ad_clock 12h_ad_slave h2_rUw_ad_clock h2_Uw_ad_master	Arna VU-Volone V Hara Processor Conduit Reset Output Clock Input AVI Master Clock Input AVI Slave Clock Input AVI Slave Clock Input AVI Slave	hps_0_ddr hps_0_io Double-click to export Double-click to export Double-click to export Double-click to export Double-click to export Double-click to export	clk_0 (h2(,axi_clock) clk_0 [f2h_axi_clock] clk_0 (h2(,bw_axi_clock]	aî.				
	×		hps_fpga_leds clk reset s1 external_connection	PIO (Parallel VO) Clock Input Reset Input Avalon Memory Mapped Slave Conduit	Double-click to export Double-click to export Double-click to export hps_fpga_leds_external_connection	cik_0 [cik] [cik]	● 0x0001_0030	0x0001_003f			
	 ✓ ✓ ✓ 	Current f	E onchip_memory2_0 cikit s1 reset1 s2 cik2 reset2 iker:	On-Chip Memory (RAM of ROM) Citock Input Avaton Memory Mapped Slave Reset Input Avaton Memory Mapped Slave Citock Input Reset Input	onchip_memory2_0_clk1 onchip_memory2_0_s1 Double-click to export Double-click to export Double-click to export Double-click to export Bable-click to export	exported [clk1] [clk1] [clk2] clk2 [clk2] [clk2]	● ⊮ 0x0000_0000	0x0000_1fff			
0 Erro	ors. 4 W	arnings								Genera	te HDL Finish

Figura 37. Qsys. Componentes instanciados.

Una vez que todo el hardware ha sido configurado correctamente, la comunicación entre el HPS y la FPGA se programa a través de una aplicación en C mapeada en memoria. Dicho mapeo de memoria permite que la CPU vea y acceda al espacio de direcciones de la FPGA, la cual contiene los componentes, para poder así leer/escribir información según sea necesario.

La aplicación en C desarrollada utiliza una API para enviar, o recibir, datos de escritura a, y desde, direcciones de memoria especificadas.

2.3.2. Parte QtCreator

Una vez configurado el hardware para la comunicación entre el HPS y la FPGA a través de la herramienta gráfica de diseño digital de hardware *Qsys*, se procede a programar el HPS.

Como se necesita que el HPS acceda a los periféricos que forman parte de la estructura de la FPGA, lo primero que se debe de realizar es la instanciación en el proyecto del *header*, el cual es generado una vez realizada la configuración en *Qsys*, mediante *#include "hps_soc_system.h"*.

16 #include "hps soc system.h"

Figura 38. QtCreator. Instanciación del header generado en Qsys.

Luego, se definen todas las variables y funciones necesarias para realizar los *bridges* entre el HPS y la FPGA.

```
18
       #define H2F AXI MASTER BASE 0xC0000000
19
       // main bus; scratch RAM, conectada a h2f axi master
20
      #define FPGA_ONCHIP_BASE ONCHIP_MEMORY2_0_BASE
#define FPGA_ONCHIP_SPAN ONCHIP_MEMORY2_0_SPAN
21
22
23
      // h2f bus
24
      // RAM FPGA port s2
      // main bus addess 0x0800 0000
25
26
      void *h2f axi master virtual base;
27
      volatile unsigned int * sram ptr = NULL ;
28
       void *sram virtual base;
29
       // =======
      // lw bus;
30
      // h2f lw axi master -> control port
31
      // read_master -> f2h_axi_slave, puedo leer cualquier
// periferico que este en este bus,
// write_master -> onchip_memory2_0.sl = 0x00020000
32
33
34
      #define H2F LW AXI MASTER BASE 0xff200000
35
      #define HW REGS_SPAN 0x00005000
36
     // the h2f light weight bus base
37
38
      void *h2p lw virtual base;
      // -----
39
      // HPS onchip memory base/span
40
       // 2^16 bytes at the top of memory
41
42
43
      #define HPS_ONCHIP_BASE 0xfff0000
#define HPS_ONCHIP_SPAN 0x00010000
44
      // HPS onchip memory (HPS side!)
45
46
      volatile unsigned int * hps onchip ptr = NULL ;
       void *hps onchip virtual base;
```

Figura 39. QtCreator. Definición de todas las variables y funciones.

Puede observarse que cada uno de los componentes poseen una dirección base. Dichas direcciones son utilizadas para acceder, controlar y enviar datos desde y hacia el SoC.

Para mapear las direcciones físicas a direcciones virtuales, lo primero que se realiza es una llamada abierta al sistema para abrir el controlador o *driver* del dispositivo de memoria *"/dev/mem"* seguido de la llamada al sistema *mmap*, la cual es utilizada para asignar la dirección física del HPS a una dirección virtual representado por el puntero $h2f_axi_master_virtual_base$.



Figura 40. QtCreator. ID del archivo /dev/mem.

71			// === get FPGA addresses =================
72			// Open /dev/mem
73	F	1	if((fd = open("/dev/mem", (O RDWR O SYNC))) == -1) {
74			<pre>printf("ERROR: could not open \"/dev/mem\"\n");</pre>
75			return(1):
76		_	
77			
78			//=====================================
79			// get virtual addr that mans to physical
80			// for light weight bus
81			htp://wirtual.base = mmap(NULL, HW REGS SPAN, (PROT READ PROT WRITE), MAP SHARED, fd. H2F LW AXI MASTER BASE);
82	F	5	if (b2p lw virtual base == MAP FAILED) {
83		Г	printf("ERROR: mmanl() failed, n"):
84			close(fd).
85			return (1):
86			
87			
88			
80			// I2_0
90			volutile ansigned into plojo public - (ansigned into ")(http://w_viitedal_buble + fito_bably;
91			//=
92			
93			// DIM FDG1 parameter addr
94			// MAILING PARAMETER AND
95			eram prr = (urginged int \$) (b); avi meter virtual base + ONCHED MEMORY O. BASE).
96			Stampor = (anorgane inc.)(her_arr_masorr_viroaar_base (okohir_himokrz_j_base))
97			//
98			
00			// HDS onchin ram
1100			// HIS CHORED FAM
101			hps_menip_viredai_base = mmap(web, ms_owemi_staw, (iver_ktaw + iver_waite), iar_stawawb, ia, ms_owemi_base),
102	F	5	if/ bos opchin virtual base == MAP FATLED) (
103	_	Г	<pre>nrintf/ "FDROR mman3() failed \n");</pre>
104			close (fd.) -
105			return(1):
106			
107) // Gat the address that mans to the HDS yam
108			// occ the dedress that maps of the his fam

Figura 41. QtCreator. Mapeo de direcciones físicas a direcciones virtuales.

La dirección virtual de AXI_MASTER_BASE está representada por $h2f_axi_master_virtual_base$, que es un puntero con el cual se podrá acceder directamente a los registros en el controlador.

Esto se realiza con el fin de poder acceder a los periféricos que forman parte de la estructura de la FPGA, lo cual se logra mapeando las direcciones de la FPGA a una memoria virtual para que las mismas puedan ser accedidas desde el lado del HPS.

Para la transmisión de datos empaquetados en UDP a través de Ethernet, se realiza un *socket UDP*[7]. Para ello, en primera instancia, se deben agregar las librerías #include <sys/socket.h>, #include <unistd.h> y #include <arpa/inet.h>.

13#include <sys/socket.h>14#include <unistd.h>15#include <arpa/inet.h>

Figura 42. QtCreator. Librerías dedicadas al socket UDP.

Posteriormente, se crea el *socket* mediante la función *socket()*[8]. Para ello, debe definirse el descriptor de archivo de socket que, en este caso, es *sfd* (*socket file descriptor*).

110	<pre>int sfd = socket(AF_INET, SOCK_DGRAM, 0);</pre>
111	if (sfd < 0) {
112	<pre>printf("Socket failed\n");</pre>
113	exit(1);
114	- }
115	<pre>printf("socket open\n");</pre>

Figura 43. QtCreator. Creación del socket.
Luego, mediante la función *setsockopt()*[9], se establecen o setean los parámetros del *socket* creado.

117
118
int optval=1;
setsockopt(sfd, SOL_SOCKET, SO_REUSEPORT, &optval, sizeof(optval));
Figura 44. OtCreator. Seteo de parámetros del socket creado.

Además, se debe definir un puerto, que en nuestro caso será el *port=7000*, y una estructura *sockaddr in*, la cual será empleada para inicializar el *socket* creado.

Además, se debe inicializar la dirección del socket fuente.

120	<pre>int port=7000;</pre>
121	<pre>struct sockaddr_in address, dst_address;</pre>
122	address.sin_family = AF_INET;
123	address.sin_addr.s_addr = hton1(INADDR_ANY);
124	<pre>address.sin_port = htons(port);</pre>

Figura 45. QtCreator. Configuración para recibir.

Luego, por medio de la función *bind()*[10], se asocia el *socket* creado a una dirección particular o a un puerto específico.



Figura 46. QtCreator. Asociación del socket a una dirección o puerto específico.

Finalmente, se debe inicializar el socket de destino.

133	dst_address.sin_family = AF_INET;
134	<pre>inet_pton(AF_INET, "192.168.7.20", &dst_address.sin_addr);</pre>
135	<pre>dst_address.sin_port = htons(port);</pre>

Figura 47. QtCreator. Configuración para enviar o transmitir.

En este punto, se generó una trama en protocolo ASTERIX CAT240 mediante un software diseñado por el Ing. Diego Martinez (perteneciente al SIAG). La trama generada es almacenada en un archivo *.bin* el cual, mediante el software *Okteta*, es convertido a *.hex*. Este archivo *.hex* es el que se carga en la memoria RAM dual port, memoria compartida entre la FPGA y el HPS, desde la herramienta gráfica *Qsys* para simular así que la misma es escrita o cargada con datos empaquetados en protocolo ASTERIX CAT240 desde el lado de la FPGA.

Un detalle a tener en cuenta es que la trama empaquetada en protocolo ASTERIX CAT240 está compuesta de datos fijos. En otras palabras, tenemos "basura" empaquetada en protocolo ASTERIX CAT240. No hay que perder de vista que el objetivo principal del software diseñado es leer la memoria RAM dual port, copiar su contenido y empaquetarlo en protocolo UDP para su posterior transmisión por Ethernet a través de un socket UDP por lo que el contenido de la trama ASTERIX CAT240, a priori, no es de interés. De todas

formas, el software diseñado está pensado para trabajar con tramas empaquetadas en protocolo ASTERIX CAT240 compuesta de datos variables, que sería el caso real.

Una vez generada la trama ASTERIX CAT240, se utilizan las herramientas de manejo de archivos[11] para abrir el archivo y copiar el contenido del mismo, que se encuentra cargado en la memoria RAM dual port, al puntero *sram_ptr* con la finalidad de acceder a los datos desde el lado del HPS.



Figura 48. QtCreator. Manejo de archivos. Copiado de la trama ASTERIX CAT240 para acceder a la misma desde el HPS.

Posteriormente, fueron creadas dos *uniones*[12]: *w32* y *w16*. Una vez creadas, se declaran tres variables del tipo *union*: *msg_index*, *start_az* y *end_az*. Dichas variables son empleadas para manejar la parte de interés de la trama ASTERIX CAT240[13].

155	Ē.	union w32{
156		<pre>uint32_t dataattribute((aligned(4)));</pre>
157		char c[4];
158	-	37
159		union w16{
160		<pre>uintl6_t dataattribute((aligned(4)));</pre>
161		char c[2];
162	-	- } / · · · · · · · · · · · · · · · · · ·
163		
164		union w32 msg_index;
165		<pre>union wl6 start_az, end_az;</pre>
166		
167		<pre>msg_index.data = 0;</pre>
168		<pre>start_az.data = 0;</pre>
169		end_az.data = 0;

Figura 49. QtCreator. Creación de uniones. Declaración de variables pertenecientes a la trama ASTERIX CAT240.

Tal como puede observarse en la figura *ut supra*, al declarar las uniones w32 y w16, se le especificó a la variable *data* un atributo especial del tipo *aligned*[14] con la finalidad de optimizar el código. Al utilizar el atributo especial *aligned*, se busca que el compilador realice menos operaciones lo que mejora considerablemente la eficiencia del tiempo de ejecución.

Luego, fueron creadas las variables *data*, que es un arreglo de datos utilizado para cargar o copiar el contenido al que apunta el puntero *sram_ptr*, que es nada más ni nada menos que el archivo que contiene la trama ASTERIX CAT240 generada cargado en la memoria SRAM dual port, y la variable *dr*, que es la variable utilizada para calcular el data rate o la tasa de transferencia de datos.

```
      172
      int data[16384] __attribute__((aligned(4)));

      173
      int dr __attribute__((aligned(4)));
```

Figura 50. QtCreator. Declaración de variables.

Al igual que para el caso de las uniones *w32* y *w16*, a las variables *data* y *dr* también se le especificaron un atributo especial del tipo *aligned*.

Posteriormente, se ejecuta un ciclo *while* en el cual se realiza, a grosso modo, el copiado del contenido al que apunta *sram_ptr* al arreglo *data*, el cargado de los datos específicos pertenecientes a la trama ASTERIX CAT240, los cuales son *msg_index*, *start_az* y *end_az*, para posteriormente poder graficar, el envío o transmisión del arreglo *data* a través del socket UDP y el cálculo de la tasa de transferencia.

```
while(1){
176
                   if(start_az.data == 0){
177
                       gettimeofday(&tl, NULL);
178
       þ
179
                   if(start az.data == 65535) {
                       gettimeofday(&t2, NULL);
                        elapsedTime = (t2.tv_sec - tl.tv_sec) * 1000000.0;
elapsedTime += (t2.tv_usec - tl.tv_usec) ; // us to
181
182
                                                                                           // sec to us
                       dr = (start_az.data*elem / (uint32_t)elapsedTime) * 8;
printf("sendto() scan T=%.2f uSec rate=%d MBits/Sec\n\r", elapsedTime, dr);
183
184
185
                   ł
186
187
                   memcpy((void*)data, (const void*)sram ptr, elem);
188
189
                   *((uint8 t *)data + 11) = msg index.c[0];
190
                   *((uint8_t *)data + 10) = msg_index.c[1];
191
                   *((uint8_t *)data + 9) = msg_index.c[2];
                   *((uint8_t *)data + 8) = msg_index.c[3];
192
                   *((uint8_t *)data + 13) = start_az.c[0];
*((uint8_t *)data + 12) = start_az.c[1];
193
194
195
                   *((uint8 t *)data + 15) = end az.c[0];
                   *((uint8_t *)data + 14) = end_az.c[1];
196
197
198
                   sendto(sfd, (void *)data, elem, 0, (struct sockaddr *)&dst address, sizeof(dst address));
199
                   start_az.data += 1;
201
                   end_az.data += 1;
202
                   msg index.data += 1;
203
              } // end while(1)
204
```

Figura 51. QtCreator. Copiado de la trama ASTERIX CAT240 y posterior transmisión de la misma por Ethernet a través del socket UDP.

Desglosando lo comentado previamente, al comienzo del ciclo *while* se tienen dos sentencias *if*, las cuales son utilizadas para marcar el comienzo y final, respectivamente, del barrido o vuelta completa al radar al momento de graficar. Además, una vez completado el barrido, haciendo uso de las funciones *gettimeofday()*[15] y *elapsedTime*, la cual retorna el tiempo transcurrido entre dos valores de tiempo, calculados previamente con la función *gettimeofday()*, se calcula el data rate o tasa de transferencia de datos.

I	175	Ē.	while(1){
I	176	Ē.	<pre>if(start_az.data == 0) {</pre>
I	177		<pre>gettimeofday(&tl, NULL);</pre>
I	178	-	}
I	179	Ē	if(start_az.data == 65535) {
I	180		<pre>gettimeofday(&t2, NULL);</pre>
I	181		<pre>elapsedTime = (t2.tv_sec - t1.tv_sec) * 1000000.0; // sec to us</pre>
I	182		<pre>elapsedTime += (t2.tv_usec - t1.tv_usec) ; // us to</pre>
I	183		<pre>dr = (start_az.data*elem / (uint32_t)elapsedTime) * 8;</pre>
1	184		<pre>printf("sendto() scan T=%.2f uSec rate=%d MBits/Sec\n\r", elapsedTime, dr);</pre>
1	185	-	}

Figura 52. QtCreator. Cálculo del data rate.

Haciendo uso de la función *memcpy()*[16], se transfiere el contenido al que apunta el puntero *sram_ptr* al arreglo *data*, que es el archivo que contiene la trama ASTERIX CAT240 generada cargado en la memoria SRAM dual port, memoria compartida entre la FPGA y el HPS.

187 memcpy((void*)data, (const void*)sram_ptr, elem);

Figura 53. QtCreator. Copiado del contenido al que apunta sram_ptr al arreglo data mediante la instrucción memcpy().

Una vez copiado el contenido de *sram_ptr* al arreglo *data*, se realiza el copiado de los parámetros específicos de la trama ASTERIX CAT240 con la finalidad de poder realizar la gráfica.

El cómo o la forma en que se realiza la transferencia de datos es de acuerdo a la arquitectura del microprocesador empleado[17]. Hay que tener en cuenta que la PC es *little-endian* mientras que el protocolo ASTERIX CAT240 es *big-endian*.

189	<pre>*((uint8_t *)data + 11) = msg_index.c[0];</pre>
190	<pre>*((uint8_t *)data + 10) = msg_index.c[1];</pre>
191	<pre>*((uint8_t *)data + 9) = msg_index.c[2];</pre>
192	<pre>*((uint8_t *)data + 8) = msg_index.c[3];</pre>
193	<pre>*((uint8_t *)data + 13) = start_az.c[0];</pre>
194	<pre>*((uint8_t *)data + 12) = start_az.c[1];</pre>
195	<pre>*((uint8_t *)data + 15) = end_az.c[0];</pre>
196	<pre>*((uint8 t *)data + 14) = end_az.c[1];</pre>

Figura 54. QtCreator. Transferencia de parámetros específicos de la trama ASTERIX CAT240.

Finalmente, por medio de la función *sendto()*[18], se envían los paquetes o datos empaquetados en UDP, previamente empaquetados en protocolo ASTERIX CAT240, a través de Ethernet por medio del *socket* creado inicialmente.

197 sendto(sfd, (void *)data, elem, 0, (struct sockaddr *)&dst_address, sizeof(dst_address));
Figura 55. QtCreator. Envío de paquetes a través del socket.

Además, es importante incrementar los valores de las variables *msg_index*, *start_az* y *end_az* ya que son las utilizadas para graficar y para evaluar las sentencias *if*.

200	<pre>start_az.data += 1;</pre>
201	<pre>end_az.data += 1;</pre>
202	<pre>msg_index.data += 1;</pre>

Figura 56. QtCreator. Incremento de las variables.

Por último, se cierra el archivo.

206 fclose(asterix_file);

Figura 57. QtCreator. Manejo de archivos. Cierre del archivo.

2.4. Ensayos (sin la implementación del handshake entre la FPGA y el HPS)

En la presente sección del documento se detallan los ensayos realizados empleando el kit de desarrollo DE1-SoC, utilizando el software desarrollado y explicado en la sección previa del presente informe.

El primer ensayo a realizar consiste en una simple prueba de rendimiento en la cual, haciendo uso del comando *iperf*, se realiza la transmisión y recepción de paquetes UDP con el fin de verificar el correcto funcionamiento del OS Linux Ubuntu embebido dentro del HPS, el cual se encarga de realizar toda la parte de red, ya que se requiere que la velocidad (*throughput*) o tasa de transmisión y/o recepción efectiva de datos UDP en tiempo real de la aplicación a diseñar sea del orden de los 1000 Mbps/1 Gbps, o lo más cercano posible a ésta.

El segundo ensayo a realizar consiste en una prueba de medición de tiempos con el fin de conocer los tiempos de lectura y de transmisión de datos fijos, los cuales fueron cargados en una memoria RAM dual port, que es la memoria compartida entre la FPGA y el HPS, y así descubrir el tiempo máximo que el HPS necesita para leer un dato en memoria, copiarlo y transmitirlo por Ethernet a través de un socket UDP.

El tercer y último ensayo a realizar es un símil del segundo con la salvedad de que, en este caso, los datos fijos cargados en la memoria RAM dual port están empaquetados en protocolo ASTERIX CAT240. Además, se realizará una prueba de visualización de los datos transmitidos por Ethernet haciendo uso del software *RadarView*, de Cambridge Pixel. Mediante este ensayo verificaremos de manera fehaciente si la línea de trabajo adoptada cumple o no con las expectativas y requerimientos solicitados.

2.4.1. Transmisión y recepción de paquetes UDP utilizando el comando *iperf*

Haciendo uso del switch administrable L2+ JetStream TL-SG3210, de una PC y del kit de desarrollo DE1-SoC, se llevó a cabo la primera prueba de rendimiento mediante la cual se medirán diferentes parámetros tal como el ancho de banda (*bandwidth*), la velocidad (*throughput*), el jitter, la pérdida de paquetes (*packet loss*) y la latencia (*latency*). Para ello, se conecta la PC al puerto Ethernet 1 y el kit de desarrollo DE1-SoC al puerto Ethernet 2 del switch mediante cables/patchcords directos/derechos (straight-through) RJ45 de la marca Vention (CAT8 FTP PATCH CABLE 4PAIRS AWM PVC 75°C EIA/TIA 568B).

Las pruebas de rendimiento se llevan a cabo utilizando la consola de Linux, donde la PC deberá de fungir como servidor mientras que el kit de desarrollo DE1-SoC de cliente, y haciendo uso del comando específico *iperf*, el cual será empleado para medir los parámetros mencionados anteriormente. La herramienta *iperf* es muy utilizada para el diagnóstico de red.

iperf [-s|-c host] [options] iperf [-h|--help] [-v|--version] Server: bandwidth #[kmgKNG | pps] bandwidth to send at in bits/sec or packets per second enhancedreports use enhanced reporting giving more tcp/udp and traffic information format [kmgKNG] format to report: Kbits, KBytes, NBytes einterval # seconds between periodic bandwidth reports len #[kmKN] length of buffer in bytes to read or write (Defaults: TCP=128K, v4 UDP=1470, v6 UDP=1450) print_mss print_TCP maximum segment size (NTU - TCP/IP header) output cflemame> output the report or error message to this specified file port # server port to listen on/connect to use UDP rather than TCP window #[KN] TCP window size (socket buffer size) realtime request realtime scheduler bind <host>[:<port]!(*dev=) bind to <host>, ip addr (including multicast address) and optional port and device compatibility for use with older versions does not sent extra msgs mss # set TCP moximum segment size (NTU - 4 bytes) -modelay set TCP no delay, disabling Nagle's Algorithm -tos # set the socket's IP_TOS (byte) field snecific: run in server mode time in seconds to listen for new connections as well as to receive traffic (default not set) enable UDP latency histogram(s) with bin width and count, e.g. 1,1000=1(ms),1000(bins) bind to multicast address and optional device set the SSM source, use with -B for (S,G) run in single threaded UDP mode run the server as a daemon Enable IPv6 reception by setting the domain and socket to AF_INET6 (Can receive on both IPv4 and IPv6) nistogram #,# bind <ip>[%<dev>] ssm-host <ip> single_udp daemon lgv6 daemon ipv6_domain ecific: lient <host> run in client mode, connecting to <host> ualtest Do a bidirectional test simultaneously (multiple sockets) allduplex run fullduplexectional test over same socket (full duplex mode) pg set the the interpacket gap (milliseconds) for packets within an isochronous frame sochronous <frames-per-seconds:<meany-sciddev> send traffic in bursts (frames - emulate video traffic) ncr-dstip Increment the destination ip with parallel (-P) traffic threads un #[kngKMG] number of bytes to transmit (instead of -t) radeoff Do a fullduplexectional test individually ine # time in seconds to transmit for (default 10 secs) ind [<ip> | <ip:port>] bind ip (and optional port) from which to source traffic lieinput <name> input the data to be transmitted from a file tdin input the data to be transmitted from stdin istemport # port to receive fullduplexectional tests back on arallel # number of parallel client threads to run everse reverse the test (client receives, server sends) tl # time-to-live, for multicast (default 1) pv6_domain Set the domain to IPv6 (send packets over IPv6) eer-detect perform server version detection and version exchange inux-congestion <algo> set TCP congestion control algorithm (Linux only) eeous: client <host> dualtest fullduplex ipg isochronous <frames incr-dstip #[kmgKMG] ipv6_domain linux-congestion <algo> Atiscellaneous: -x, --reportexclude [CDMSV] exclude C(connection) D(data) N(multicast) S(settings) V(server) reports -y, --reportstyle C report as a Comma-Separated Values -h, --help print this message and quit -v, --version print version information and quit kmgKMG] Indicates options that support a k,m,g,K,N or G suffix owercase format characters are 10^3 based and uppercase are 2^ e.g. 1k = 1000, 1K = 1024, 1m = 1,000,000 and 1M = 1,048,576) 'n based TCP window size option can be set by the environment variable WINDOW_SIZE. Most other options can be set by an environment variable E_clong option name>, such as IPERF_BANDWIDTH. rce at <http://sourceforge.net/projects/iperf2/>
ort bugs to <iperf-users@lists.sourceforge.net>

Figura 58. Comando iperf. Sintaxis y uso.

Por lo tanto, los comandos a utilizar son los siguientes:

• Cliente (PC con IP 192.168.7.40)

iperf3.exe -c 192.168.7.20 -u -i 1 -b 1000M -t 60

Donde -*c* hace referencia a que es cliente, 192.168.7.20 es la dirección IP del servidor, -*u* es referido a UDP, -*i* 1 hace referencia al intervalo en segundos entre reportes (1 es que los intervalos entre reportes son cada 1 segundos), -*b* 1000M es el ancho de banda (-*b* es de bandwidth y 1000M es de 1000 Mbits/s) y -*t* 60 es la duración de la prueba a realizar (-*t* es de tiempo y 60 es la duración en segundos).

ootgi	DE1-SoC:-#	ipert	f - c 1	192.168.	7.20	u -i 1 -b	1000M -t 60		
lient	connecti	na to	192.	168.7.20	UDP	nort 5991			
Sendir	a 1470 by	te dat	tagrar	15	, 001	pore 5661			
JDP bu	ffer size:	: 160	KBy1	te (defai	ult)				
31	local 192	.168.7	.40	port 403	19 cor	nnected wi	th 192.168.7.2	20 port 50	301
ID	Interval		Trans	MButon	Bandy	width Mhite/coo			
31	1.0- 2.0	sec	94.0	MBytes	790	Mbits/sec			
31	2.0-3.0	sec	94.1	MBytes	790	Mbits/sec			
3]	3.0- 4.0	sec	94.1	MBytes	789	Mbits/sec			
3]	4.0- 5.0	sec	94.1	MBytes	790	Mbits/sec			
3]	5.0- 6.0	sec	94.3	MBytes	791	Mbits/sec			
31	6.0-7.0	sec	94.1	MBytes	798	Mbits/sec			
31	8.6-9.6	Sec	94.2	MBytes	791	Mbits/sec			
31	9.0-10.0	sec	94.4	MBytes	792	Mbits/sec			
31	10.0-11.0	sec	94.4	MBytes	792	Mbits/sec			
3]	11.0-12.0	sec	94.4	MBytes	792	Mbits/sec			
3]	12.0-13.0	sec	94.4	MBytes	792	Mbits/sec			
31	13.0-14.0	sec	94.5	MBytes	793	Mbits/sec			
31	14.0-15.0	sec	94.0	MBytes	794	Mbits/sec			
31	16.0-17.0	Sec	94.2	MRvtes	790	Mhits/sec			
31	17.0-18.0	sec	94.3	MBytes	791	Mbits/sec			
3]	18.0-19.0	sec	94.5	MBytes	792	Mbits/sec			
3]	19.0-20.0	sec	94.4	MBytes	792	Mbits/sec			
3]	20.0-21.0	sec	94.2	MBytes	790	Mbits/sec			
31	21.0-22.0	sec	94.2	MBytes	791	Mbits/sec			
31	22.0-23.0	sec	94.0	MBytes	789	Mbits/sec			
31	24.0-25.0	sec	94.2	MBytes	790	Mbits/sec			
31	25.0-26.0	sec	94.1	MBytes	789	Mbits/sec			
3]	26.0-27.0	sec	94.2	MBytes	790	Mbits/sec			
3]	27.0-28.0	sec	94.2	MBytes	790	Mbits/sec			
3]	28.0-29.0	sec	94.0	MBytes	789	Mbits/sec			
31	29.0-30.0	sec	94.2	MBytes	790	Mbits/sec			
31	30.0-31.0	sec	94.1	MBytes	790	Mbits/sec			
31	32.0-33.0	sec	94.1	MBytes	790	Nbits/sec			
3]	33.0-34.0	sec	94.1	MBytes	789	Mbits/sec			
3]	34.0-35.0	sec	94.1	MBytes	790	Mbits/sec			
3]	35.0-36.0	sec	94.2	MBytes	790	Mbits/sec			
31	36.0-37.0	sec	94.1	MBytes	790	Mbits/sec			
31	37.0-38.0	sec	94.2	MBytes	798	Mbits/sec			
31	39.0-40.0	sec	94.0	MBytes	788	Mbits/sec			
31	40.0-41.0	sec	94.0	MBytes	788	Mbits/sec			
31	41.0-42.0	sec	94.1	MBytes	789	Mbits/sec			
3]	42.0-43.0	sec	94.0	MBytes	788	Mbits/sec			
3]	43.0-44.0	sec	94.2	MBytes	790	Mbits/sec			
31	44.0-45.0	sec	94.1	MBytes	790	Mbits/sec			
- 31	45.0-40.0	sec	94.2	MBytes	790	Mbite/sec			
31	47.0-48.0	Sec	94.0	MRvtes	789	Mhits/sec			
31	48.0-49.0	sec	93.9	MBytes	788	Mbits/sec			
3]	49.0-50.0	sec	93.8	MBytes	787	Mbits/sec			
3]	50.0-51.0	sec	94.0	MBytes	788	Mbits/sec			
3]	51.0-52.0	sec	93.9	MBytes	788	Mbits/sec			
31	52.0-53.0	sec	94.0	MBytes	789	Mbits/sec			
31	54.0-54.0	sec	94.0	MButes	788	Mbits/sec			
31	55.0.56.0	sec	94.0	MBytes	788	Nhits/sec			
31	56.0-57.0	sec	93.9	MBytes	788	Mbits/sec			
31	57.0-58.0	sec	93.9	MBytes	788	Mbits/sec			
[3]	58.0-59.0	sec	93.9	MBytes	788	Mbits/sec			
3]	0.0-60.0	sec	5.52	GBytes	790	Mbits/sec			
3]	Sent 40287	773 da	atagra	ams					

Figura 59. Consola de Linux. Comando iperf. Cliente.

• Servidor (PC con IP 192.168.7.20)

iperf3.exe -s -u -i 1 -b 1000M -t 60

Donde -s hace referencia a que es servidor, -u es referido a UDP, -i 1 hace referencia al intervalo en segundos entre reportes (1 es que los intervalos entre reportes son cada 1 segundos), -b 1000M es el ancho de banda (-b es de bandwidth y 1000M es de 1000 Mbits/s) y -t 60 es la duración de la prueba a realizar (-t es de tiempo y 60 es la duración en segundos).

siag001@debian-siag001:~\$ iperf -s -u -i 1 -b 1000M -t 60 Server listening on UDP port 5001 with pid 2501 Read buffer size: 1.44 KByte (Dist bin width= 183 Byte) UDP buffer size: 208 KByte (default)

Figura 60. Consola de Linux. Comando iperf. Servidor.

La velocidad (*throughput*) o tasa de transmisión y/o recepción efectiva de datos UDP en tiempo real se calculó de la siguiente manera:

 $\frac{Transfer[GBytes] \cdot 8[bits]}{Segundos} = Velocidad[Gbps]$

Por lo tanto, tenemos que:

 $\frac{5.52[GBytes] \cdot 8[bits]}{60 \ segundos} = 0.736[Gbps] = 736[Mbps]$

Mediante el presente ensayo se verificó el correcto funcionamiento del OS Linux Ubuntu embebido dentro del HPS, el cual se encarga de realizar toda la parte de red.

2.4.2. Medición de tiempos de lectura y de transmisión con datos fijos

Haciendo uso de una PC y del kit de desarrollo DE1-SoC, se realizará una prueba de medición de tiempos. Dicho ensayo consiste en la escritura y posterior lectura de datos fijos en una memoria RAM dual port, que es la memoria compartida entre la FPGA y el HPS, para medir así los tiempos de lectura y de transmisión con el fin de conocer el tiempo máximo que el HPS necesita para leer un dato en memoria, copiarlo y transmitirlo por Ethernet a través de un socket UDP. Vale destacar que la escritura de datos es realizada desde el lado de la FPGA mientras que la lectura de los mismos es realizada desde el lado del HPS.

La prueba de medición de tiempos se lleva a cabo utilizando la consola de Linux, donde se ejecuta el código *dma01* desde el lado del HPS. El código empleado para realizar el ensayo en cuestión está basado en un código encontrado en Internet[19][20]. En pocas palabras, el código se encarga, en primera instancia, de mapear las direcciones de la FPGA a una memoria virtual para que las mismas puedan ser accedidas desde el lado del HPS. Luego, se escribe la memoria de la FPGA en su totalidad (8192 [bytes]) con datos fijos y posteriormente se la lee, haciendo uso de un puntero denominado *sram_ptr*, desde el HPS. A continuación, haciendo uso de la función *memcpy()*[16], se transfiere el contenido del puntero *sram_ptr* a un arreglo *data* para, finalmente, enviar dichos datos por Ethernet a través del socket UDP empleando la función *sendto()*[18]. Para calcular los tiempos, se emplean las funciones *gettimeofday()*[15] y *elapsedTime*. Esta última retorna el tiempo transcurrido entre dos valores de tiempo, calculados previamente con la función *gettimeofday()*.

Conociendo a grosso modo la función del código *dma01*, se procede a realizar las pruebas comentadas. Para ello, desde la consola de Linux ejecutamos dicho código, tal y como se observa en la imagen a continuación.



Figura 61. Consola de Linux. Ejecución código dma01. Tiempos de lectura y de transmisión.

La velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real se calculó de la siguiente manera:

 $\frac{Tamaño \ de \ la \ memoria[Bytes] \cdot 8[bits]}{Tiempo \ de \ transmisión[segundos]} = Velocidad[Mbps]$

Por lo tanto, tenemos que:

 $\frac{8192[Bytes] \cdot 8[bits]}{107 \cdot 10^{-6} [segundos]} \approx 600 [Mbps]$

Luego, mediante el software *Wireshark*, realizamos la captura del tráfico de red. Con esto, nos aseguramos que el socket UDP realizado funciona correctamente. Vale destacar que los datos son transmitidos desde el HPS, cuya dirección IP es 192.168.7.40, y son recibidos por la PC, cuya dirección IP es 192.168.7.20.

							Capturing f	rom enp1s0							^ _ ¤ X
<u>File</u>	dit <u>V</u> iew	<u>Go</u> <u>C</u> apture	e <u>A</u> nalyze	<u>Statistics</u>	s lelepho	on <u>y W</u> ireless	loois <u>H</u> eip								
	1 🙆			. 🔹 🔶	✤ I	a 🔳 🔳	• • • •								
📕 udp.	port == 7	000													¥ 🗆 🔹 🕈
No.	Time	Sou	urce		0	Destination	Protocol	Length Info							
t 2	90 4.6310	067310 192	2.168.7.4	0	1	192.168.7.20	RX RX	834 ACKALL	Seq: 50331	648 Call:	33554432	Source Port	t: 7000	Destinatio	n Port: 700
L 2	92 4.6312	242714 192	2.168.7.2	0	1	192.168.7.40	ICMP	590 Destina	tion unreac	hable (Por	rt unreacha	uble)			
▶ Ethe	ernet II,	Src: 1a:f5	:94:18:96	6:38 (1a:f	5:94:18:	96:38), Dst	: LCFCHeFe_38:d9:	06 (54:e1:ad:	38:d9:06)						
▶ Inte	ernet Pro	tocol Versi	on 4, Sro	: 192.168	3.7.40, D	ost: 192.168	.7.20								
+ User	Datagra	m Protocol,	Src Port	:: 7000, D	Ost Port:	7000									
De	estinatio	on Port: 706	90												
Le	ength: 82	200													
E E	Checksum	Status: Unv	/erified]												
Į.	Stream in	ndex: 11]													
→ [`	Timestamp	DS] od (8192 byt	-05)												_
► RX F	rotocol	au (0132 byt	-63)												_
0000	1h EQ 1h	E9 20 09 7	0 04 00	00 00 00	01 00 00		N.								¥
0010	02 00 00	00 03 00 0	0 00 04	00 00 00	05 00 00	00	y								
0020	06 00 00	00 07 00 0	00 00 08	00 00 00	09 00 00	9 00									
0030	0e 00 00	00000000000000000000000000000000000000	00 00 00 00	00 00 00	11 00 00	9 00									
0050	12 00 00	0 00 13 00 0	00 00 14	00 00 00	15 00 00	9 00									
0060	16 00 00	00 17 00 C	0 00 18	00 00 00	19 00 00 1d 00 00	9 00									
0080	1e 00 00	00 1f 00 0	00 00 20	00 00 00	21 00 00	9 00									
0090	22 00 00	00 23 00 0	0 00 24	00 00 00	25 00 00	9 00 "···#	\$ %								
0010	20 00 00	00 21 00 0	28	00 00 00	29 00 00	ουυ α····	()								Ŧ
Frame	e (834 byte	es) Reasse	mbled IPv4	4 (8200 byt	tes)										
0 🏾	Payload (udp.payload)	, 8,192 by	es						Packets	: 9162 · Dis	played: 2 (0.0	%)		Profile: Default

Figura 62. Software Wireshark. Captura de tráfico.

2.4.3. Medición de tiempos de lectura y de transmisión con datos fijos empaquetados en protocolo ASTERIX CAT240

Haciendo uso de una PC y del kit de desarrollo DE1-SoC, se realizará una prueba de medición de tiempos similar a la anterior con la salvedad de que, en este caso, los datos fijos cargados en la memoria RAM dual port están empaquetados en protocolo ASTERIX CAT240. Por lo tanto, la medición de los tiempos de lectura y de transmisión, es decir, el tiempo máximo que el HPS necesita para leer un dato en memoria, copiarlo y transmitirlo por Ethernet a través de un socket UDP, nos darán una velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real.

La prueba de medición de tiempos se lleva a cabo utilizando la consola de Linux, donde se ejecuta el código *dma01* desde el lado del HPS. El código empleado para realizar el ensayo en cuestión está basado en un código encontrado en Internet[19][20], el cual fue comentado de forma generalizada en el ensayo anterior¹. De todas maneras, el software realizado tuvo que ser modificado debido a que, para este caso, se debe de generar primero una trama en protocolo ASTERIX CAT240, la cual posteriormente es cargada en memoria, leída a través del HPS y enviada por Ethernet a través del socket UDP. La trama ASTERIX es generada por medio de un software, el cual fue diseñado por el Ing. Diego Martinez (perteneciente al SIAG). La misma es almacenada en un archivo *.bin* el cual, mediante el software *Okteta*, es convertido a *.hex*. Este archivo *.hex* es el que se carga en la memoria RAM dual port para simular así que la misma es escrita con datos empaquetados en protocolo ASTERIX CAT240 desde el lado de la FPGA. Posteriormente, se lee dicha memoria, se empaquetan los datos cargados previamente en la memoria en protocolo UDP para, finalmente, enviar los mismos por Ethernet a través de un socket UDP.

En el Anexo B podrá encontrarse una breve reseña del protocolo ASTERIX CAT240, además de una explicación sobre cómo se arma un paquete o trama empaquetado en dicho protocolo y la correspondencia entre el mismo y su representación gráfica, lo cual resultará

 $^{^{1}}$ ver ensayo 2.4.2.

útil conocer al momento de observar las capturas de tráfico de red realizadas en los ensayos detallados a continuación.

Como punto de partida, se debe asegurar una velocidad de transmisión efectiva de datos que sea igual a la velocidad a la que se cargan los datos en memoria, los cuales dijimos son de tamaño variable. Es decir, como mínimo se debe ser capaz de transmitir a la misma velocidad en que almacenan los datos en memoria para así, mientras se transmite la primera trama, poder leer la segunda y prepararla para la transmisión. Esta sería la peor condición que debemos cumplir o asegurar.

Sabemos que la velocidad de muestreo del ADC es de 16 [MS/s] y cada muestra es de 2 [bytes]. Por lo tanto, la velocidad a la que se cargan los datos en memoria se calcula como:

 $16[MS/s] \cdot 2[Bytes] = 32[MB/s]$

A lo largo de los ensayos descritos en la presente sección, se observará que se optimiza bajo ciertas condiciones. No hay que perder de vista que estos son los primeros ensayos que se realizan, los cuales brindarán un nuevo punto de partida para ensayos o pruebas futuras. A la fecha, nuestra peor condición a cumplir es el tiempo que se tarda en almacenar o cargar los datos en memoria, es decir, debemos ser capaces de superar una tasa de transferencia de datos de 32[MB/s] la cual está dada por la velocidad de muestreo del ADC.

Durante los ensayos, además, se notará que el tamaño del paquete o trama empaquetada en protocolo ASTERIX CAT240 es variable, es decir, comenzaremos utilizando tramas de 1056 [bytes], seguiremos con tramas de 2080 [bytes] y finalizaremos con tramas de 4128 [bytes].

2.4.3.1. Tamaño de trama de 1056 [bytes]

El primer ensayo realizado es para un tamaño de paquete o trama empaquetada en protocolo ASTERIX CAT240 de 1056 [bytes], donde los datos o información útil, es decir, sin contar encabezados y demás, son 1024 [bytes].

Por lo tanto, tenemos que:

oot@DE1-SoC:~/source# ./dma01									
ocket open									
bind success									
EOF en asterix file.									
Se leveron 1056 bytes									
endto() scan T=1379786.00 uSec rate=400 MBits/Sec									
endto() scan T=1380547.00 uSec rate=400 MBits/Sec									
endto() scan T=1379938.00 uSec rate=400 MBits/Sec									
endto() scan T=1381119.00 uSec rate=400 MBits/Sec									
endto() scan T=1382061.00 uSec rate=400 MBits/Sec									
endto() scan T=1381881.00 uSec rate=400 MBits/Sec									
endto() scan T=1380899.00 uSec rate=400 MBits/Sec									
endto() scan T=1382052.00 uSec rate=400 MBits/Sec									
endto() scan T=1381185.00 uSec rate=400 MBits/Sec									
endto() scan T=1382301.00 uSec rate=400 MBits/Sec									
endto() scan T=1380898.00 uSec rate=400 MBits/Sec									
endto() scan T=1381089.00 uSec rate=400 MBits/Sec									
endto() scan T=1381833.00 uSec rate=400 MBits/Sec									
endto() scan T=1380115.00 uSec rate=400 MBits/Sec									

Figura 63. Consola de Linux. Ejecución código dma01. Tiempos de transmisión.

Podemos observar que un barrido o una vuelta completa de radar tarda en completarse 1.38 segundos aproximadamente.

Por lo tanto, la tasa de transferencia de datos en tiempo real se calculó de la siguiente manera:

 $\frac{Cantidad \ de \ paquetes \cdot Tamaño \ de \ trama[Bytes]}{Tiempo \ de \ transmisión[segundos]} = Tasa \ de \ transferencia[MB/s]$

Considerando datos o información útil, tenemos que:

65536 paquetes ·1024[Bytes] 1.38[segundos] ≈48.63[MB/s]

La velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real se calculó de la siguiente manera:

65536 paquetes·1024[Bytes]·8[bits] 1.38[segundos] ≈389[Mbps]

Luego, mediante el software *Wireshark*, se realiza la captura del tráfico de red. Con esto, nos aseguramos que el socket UDP realizado funciona correctamente.

Vale destacar que los datos son transmitidos desde el HPS, cuya dirección IP es 192.168.7.40, y son recibidos por la PC, cuya dirección IP es 192.168.7.20.

<u></u>	Capturing	from enp1s0 ^ _ ¤ X
File Edit View Go Capture Analyze Statistics	Telephony Wireless Tools Help	
	🗲 🖌 💘 📃 📄 🗗 🖛 🕷	
udp.port == 7000		🛛 🗖 🔻
No. Time Source Des	stination Protocol Length Info	A
819435 32.185624245 192.168.7.40 192	2.168.7.20 UDP 1098 7000 →	7000 Len=1056
819436 32.185641589 192.168.7.40 192	2.168.7.20 UDP 1098 7000 →	7000 Len=1056
819437 32.185658357 192.168.7.40 192	2.168.7.20 UDP 1098 7000 → 7	7000 Len=1056
819438 32.185675545 192.168.7.40 192	2.168.7.20 UDP 1098 7000 →	7000 Len=1056
819439 32.185692844 192.168.7.40 192	1.168.7.20 UDP 1098 7000 →	/000 Len=1056
019440 32.105/10520 192.100.7.40 192 819441 22 185727628 192 168 7 49 192	2.168.7.20 UDP 1096.7000 → 1 2.168.7.20 UDP 1098.7000 → 1	7000 Len-1056
819442 32 185744078 192 168 7 40 192	2 168 7 20 UDP 1098 7000 → 1	2000 Len=1056
819443 32,185761189 192,168,7,40 192	2.168.7.20 UDP 1098 7000 -	2000 Len=1056
819444 32.185778608 192.168.7.40 192	2.168.7.20 UDP 1098 7000 →	7000 Len=1056
819445 32.185795518 192.168.7.40 192	2.168.7.20 UDP 1098 7000 →	7000 Len=1056
819446 32.185812717 192.168.7.40 192	2.168.7.20 UDP 1098 7000 →	7000 Len=1056
819447 32.185829375 192.168.7.40 192	2.168.7.20 UDP 1098 7000 → 7	7000 Len=1056
819448 32.185846557 192.168.7.40 192	2.168.7.20 UDP 1098 7000 → 7	7000 Len=1056
819449 32.185864279 192.168.7.40 192	2.168.7.20 UDP 1098 7000 →	/000 Len=1056
819450 32.185881362 192.168.7.40 192	1.168.7.20 UDP 1098.7000 → 1	7000 Len=1056
819452 32 185919806 192 168 7 40 192	2 168 7 20 UDP 1090 7000 → 1	7000 Len=1056
819453 32, 185933493 192, 168, 7, 40 192	2.168.7.20 UDP 1098 7000 → 1	7000 Len=1056
010/E4 22 100000120 102 100 7 40 102	160 7 30 UDD 1000 7000 '	▼
Time to Live: 64		A
Protocol: UDP (17)		
Header Checksum: 0x6668 [validation disa	abled]	
[Header checksum status: Unverified]		
Destination Address: 192.166.7.40		
Elser Datagram Protocol, Src Port: 7000, Ds	st Port: 7000	
 Data (1056 bytes) 		
Data: f00420e7a0010202000c7d497d497d4900	00000000bebc20000040400000400103c3c	3636
[Length: 1056]		*
0020 07 14 1b 58 1b 58 04 28 45 08 f0 04 2	20 07 20 01	
0030 02 02 00 0c 7d 49 7d 49 7d 49 00 00 0	00 00 0h eh	
0040 c2 00 00 04 04 00 00 04 00 10 3c 3c 3	3c 3c 3c 3a	
0050 2c 1d 1b 19 33 5c 71 72 7b 94 ab b2 b	b1 b1 af c1 ,3\qr {	
0060 e6 fa	fa fa fa fa ••••	
0070 fa fa f9 de b7 a3 a3 a0 9a 93 93 93 9	93 93 8f 89	
0000 07 87 86 83 81 81 80 79 71 6e 6f 6a 5	56 56 56	¥
💛 🗷 Data (data), 1,056 bytes		Packets: 829464 · Displayed: 818600 (98.7%) Profile: Default

Figura 64. Software Wireshark. Captura de tráfico.

					Capturing fro	m enp1s0		^ _ ¤ X
Eile	Edit View Go Captu	re <u>A</u> nalyze <u>S</u> ta	tistics Telephony	Wireless	[ools <u>H</u> elp			
	i 🗴 💿 🗎 🚺	X 6 🔍 🚸	۱ کې کې کې کې		e a 👖			
ip.	addr == 192,168,7,40							× •
N.	T	Courses	Destination	Destand	L e a atta la fa			
NO.	11me	50urce	Descination	Protocol	Length Into	0 Lon=10E6		
	810/20 22 1856028//	192.100.7.40	192.100.7.20	UDP	1098 7000 → 700	0 Len=1056		
	819440 32 185710526	192 168 7 40	192 168 7 20	UDP	1098 7000 - 700	0 Len=1050		
	819441 32 185727628	192.168.7.40	192.168.7.20	UDP	1098 7000 - 700	0 Len=1056		
	819442 32.185744078	192.168.7.40	192.168.7.20	UDP	1098 7000 → 700	0 Len=1056		
	819443 32.185761189	192.168.7.40	192.168.7.20	UDP	1098 7000 → 700	0 Len=1056		
	819444 32.185778608	192.168.7.40	192.168.7.20	UDP	1098 7000 → 700	0 Len=1056		
	819445 32.185795518	192.168.7.40	192.168.7.20	UDP	1098 7000 → 700	0 Len=1056		
	819446 32.185812717	192.168.7.40	192.168.7.20	UDP	$1098\ 7000 \rightarrow 700$	0 Len=1056		
	819447 32.185829375	192.168.7.40	192.168.7.20	UDP	1098 7000 → 700	0 Len=1056		
	819448 32.185846557	192.168.7.40	192.168.7.20	UDP	1098 7000 → 700	0 Len=1056		
	819449 32.185864279	192.168.7.40	192.168.7.20	UDP	1098 7000 → 700	0 Len=1056		
	819450 32.185881362	192.168.7.40	192.168.7.20	UDP	1098 7000 → 700	0 Len=1056		
	B10451 32,105099032	102 168 7 40	102 168 7 20	UDP	1090 7000 → 700	0 Len=1056		
	R19452 32.105919000	192.100.7.40	192.100.7.20	UDP	1098 7000 - 700	0 Len=1050		
	819454 32 185950129	192.168.7.40	192.168.7.20	UDP	1098 7000 - 700	0 Len=1056		
	819455 32.185966974	192.168.7.40	192.168.7.20	UDP	1098 7000 → 700	0 Len=1056		
	819456 32.185983733	192.168.7.40	192.168.7.20	UDP	1098 7000 → 700	0 Len=1056		
	0104E7 00 106001104	100 100 7 40	100 160 7 00	UDD	1009 7000 700	A Lon=10E6		•
	Time to Live: 64							<u>^</u>
	Protocol: UDP (17)							
	Header Checksum: Oxe	5668 [Validatio	n disabledj					
	Source Address: 192	168 7 40	uj					
	Destination Address	192 168 7 20						
► Us	er Datagram Protocol	. Src Port: 70	00. Dst Port: 70	00				
- Da	ta (1056 bytes)	.,						
	Data: f00420e7a00102	202000c7d497d49	7d490000000000beb	c20000040	400000400103c3c3c3	3c		
	[Length: 1056]							
0000	07 14 1b 50 15 50	04 29 40 00 4	0 04 20 07 00 04	V V	(1			
0020	07 14 10 58 10 58 02 02 00 0c 7d 49	7d 49 7d 49 0	0 04 20 07 a0 01 A AA AA AA AA AA AA	· · · X · X				<u>^</u>
0040	c2 00 00 04 04 00	00 04 00 10 3	c 3c 3c 3c 3c 3c 3a					
0050	2c 1d 1b 19 33 5c	71 72 7b 94 a	b b2 b1 b1 af c1	, 3	ar {			
0060	e6 fa fa fa fa fa							
0070	fa fa f9 de b7 a3							
0080	87 87 86 83 81 81	80 79 71 6e 6	f 6a 5e 56 56 56		∙y qnoj^VVV			Ψ
0 2	Data (data), 1,056 by	rtes					Packets: 830873 · Displayed: 818600 (98.5%)	Profile: Default

Figura 65. Software Wireshark. Captura de tráfico.

2.4.3.2. Tamaño de trama de 2080 [bytes]

El segundo ensayo realizado es para un tamaño de paquete o trama empaquetada en protocolo ASTERIX CAT240 de 2080 [bytes], donde los datos o información útil, es decir, sin contar encabezados y demás, son 2048 [bytes].

Por lo tanto, tenemos que:

Terminal - root@DE1.Soft - steaurce	•	- ×
The Fulk Many Tensolation Tensolation		
rie coli, view lettrinital labs neip		
root@DE1-SoC: ~/source × siag001@debian-siag001: ~		×
root@DE1-SoC:~/source# ./dma01		
socket open		
bind success		
EOF en asterix_file.		
Se leyeron 2080 bytes		
sendto() scan T=2664828.00 uSec rate=408 MBits/Sec		
sendto() scan T=2653951.00 uSec rate=408 MBits/Sec		
sendto() scan T=2660320.00 uSec rate=408 MBits/Sec		
sendto() scan T=2656627.00 uSec rate=408 MBits/Sec		
sendto() scan T=2662542.00 uSec rate=408 MBits/Sec		
sendto() scan T=2654698.00 uSec rate=408 MBits/Sec		
sendto() scan T=2663246.00 uSec rate=408 MBits/Sec		
sendto() scan T=2653688.00 uSec rate=408 MBits/Sec		
sendto() scan T=2659911.00 uSec rate=408 MBits/Sec		
sendto() scan T=2652442.00 uSec rate=408 MBits/Sec		
sendto() scan T=2661450.00 uSec rate=408 MBits/Sec		
sendto() scan T=2652028.00 uSec rate=408 MBits/Sec		
sendto() scan T=2658622.00 uSec rate=408 MBits/Sec		
sendto() scan T=2654499.00 uSec rate=408 MBits/Sec		
sendto() scan T=2660584.00 uSec rate=408 MBits/Sec		
sendto() scan T=2650724.00 uSec rate=408 MBits/Sec		
sendto() scan T=2658204.00 uSec rate=408 MBits/Sec		
sendto() scan T=2649689.00 uSec rate=408 MBits/Sec		

Figura 66. Consola de Linux. Ejecución código dma01. Tiempos de transmisión.

Podemos observar que un barrido o una vuelta completa de radar tarda en completarse 2.65 segundos aproximadamente.

Considerando datos o información útil, tenemos que:

La velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real se calculó de la siguiente manera:

<u>65536 paquetes·2048[Bytes]·8[bits]</u> ≈405.2[Mbps] 2.65[segundos]

Haciendo uso del software *RadarView*, de Cambridge Pixel, lo que hacemos es graficar los datos enviados por Ethernet a través del socket UDP.



Figura 67. Software RadarView. Visualización de paquetes transmitidos en protocolo ASTERIX CAT240.

Luego, mediante el software Wireshark, se realiza la captura del tráfico de red.

					*6	enp1s0	^ _ ¤ X
Ei	le <u>E</u> d	it <u>V</u> iew <u>G</u> o	Capture Analyze St	atistics Telephony <u>W</u> ireless <u>T</u> ools	<u>H</u> elp		
		201	i 🛅 🕱 🎑 🔍) 🔹 🥆 🍋 🕷 📃 🗖 🖉			
	udp.pd	ort == 7000					•
No		Time	Source	Destination	Protoco	Length Info	A
T.	6	4 0.45203918	31 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	6	6 0.45206849	96 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	6	8 0.45211609	94 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	7	0 0.45218209	96 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	7	2 0.45224312	23 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	7	4 0.45229669	98 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
+	7	6 0.45236520	06 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	7	8 0.45241900	04 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	8	0 0.45246519	95 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	8	2 0.45251747	79 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	8	4 0.45257692	25 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	8	6 0.45263761	L6 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	8	8 0.45270494	19 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	9	0 0.45277298	37 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	9	2 0.45281/1/	/8 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	9	4 0.45288932	22 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	9	6 0.45293634	15 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	9	8 0.45298793	32 192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	10	0 0.45304485	192.168.7.40	192.168.7.20	UDP	642 7000 → 7000 Len=2080	
	10	2 0.45310267	1 192.166.7.40	192.100.7.20	UDP	642 7000 → 7000 Len=2000	×
•	Frame	76: 642 by	tes on wire (5136 b	its), 642 bytes captured (5136 b	its) on	interface enpis0, id 0	
•	Ether	net II, Src	: 66:bf:aa:a6:f9:9a	(66:bf:aa:a6:f9:9a), Dst: LCFCH	eFe_38:d	9:06 (54:e1:ad:38:d9:06)	
•	Inter	net Protoco	L Version 4, Src: 1	92.168.7.40, Dst: 192.168.7.20			
•	User	Datagram Pro	otocol, Src Port: 7	000, DSt Port: 7000			
-	Data	(2080 bytes) Is and an an an an an a fact the			2-0-0-	
	Dat	a: 10082007	a001020200220D160D1	60b160000000000bebc2000004080000	180020303	3C3C3C	
	[Le	engrn: 2000]					
00	000 1	Lb 58 1b 58	08 28 a9 6e f0 08	20 e7 a0 01 02 02 ·X·X·(·n ··			A
00	010	00 22 0b f6	0b f6 0b f6 00 00	00 00 0b eb c2 00 ."			
00	020			3c 3c 3c 3c 3c 3c <<<	.<<<<		
00	030			19 1b 29 3c 51 64 <<;81) · · · ·	•) <qd< td=""><td></td><td></td></qd<>		
00	040	Se 73 73 73	78 80 8c 9b a6 af	b2 b1 b1 b1 b1 b1 nsssx			
00	050	0 af b8 ca	dc ef f9 fa fa fa	ta ta ta fa fa fa			
00	060	a ta ta ta	та та та та та та та	та та та та та та			-
F	rame (642 hytes)	Reassembled IPv/L (20	188 bytes)			

Figura 68. Software Wireshark. Captura de tráfico.

Puede observarse que la trama está definitivamente empaquetada en protocolo ASTERIX CAT240 debido a que se identifica el primer byte de la misma con f0.

Además, pudo observarse que los datos empaquetados en protocolo ASTERIX CAT240, al transmitirlos por UDP, son recortados y luego reensamblados para su posterior representación utilizando el software *RadarView*. Esta división de paquetes se debe a que se supera el *MTU* que, para el caso de Ethernet, es de 1500 [bytes].

									*enj	p1s0					^ _	• ×
Ei	le <u>E</u> dit <u>V</u> iew	<u>Go</u> Ca	pture <u>A</u> nal	lyze <u>S</u> t	atistics	Telephony	Wireless T	ools <u>H</u> elp								
) 🛍 [X 🖸	Q 🔮		• He H 🛛										
	ip.addr == 19	92.168.7.4	10													• •
No	. Time		Source			Desti	nation	Pro	otocol	Length Info						-
	7 0.451	1039412	192.168.	7.40		192.3	168.7.20	IP	v4	1514 Fragmented IP protoco	l (proto=UDP 17	', off=0,	ID=7bc8)	[Reassembled	in #8] 🖛	_
٠	8 0.451	1039621	192.168.	7.40		192.3	168.7.20	UD	Р	642 7000 → 7000 Len=2080						
	10 0.451	1039829	192.168.	7.40		192.3	168.7.20	IP	v4	1514 Fragmented IP protoco	l (proto=UDP 17	', off=0,	ID=7bc9)	[Reassembled	in #11]	
	11 0.451	1040048	192.168.	7.40		192.:	168.7.20	UD	Р	642 7000 → 7000 Len=2080						
	12 0.451	1040256	192.168.	7.40		192.3	168.7.20	IP	v4	1514 Fragmented IP protoco	l (proto=UDP 17	′, off=0,	ID=7bca)	[Reassembled	in #13]	
	13 0.451	1040464	192.168.	7.40		192.3	168.7.20	UD	Р	642 7000 → 7000 Len=2080						Ŧ
*	<pre>> Frame /: 1514 bytes on wire (12112 Dits), 1514 bytes captured (12112 Dits) on interface enpise, 1d 0 > Ethernet II, Src: 66:16:1a:a6:67:93 (a6:bf:na:a6:67:93), 155: LCFCHeFe_38:d9:06 (54:e1:ad:38:d9:06) > Internet Protocol Version 4, Src: 192.168.7.40, Dst: 192.168.7.20 0100 = Version: 4 0101 = Header Length: 20 bytes (5) > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 1500 Identification: 0x7bc8 (31680) > Flags: 0x20, More fragments Fragment Offset: 0 Time to Live: 64 Protocol: UDP (17) Header Checksum status: Unverified] [Header checksum status: Unverified] Source Address: 192.168.7.40</pre>															
	Reassemb	lon Addre oled IPv4	in frame	68.7.20 : 81	1											
-	Data (1480	bytes)														
	Data: 1b5	81b58082	8a9c8f008	20e7a00	10202002	220bd80bd8	0bd800000	000bebc20	000040	0800						
	[Length:	1480]														
00	020 07 14 1	b 58 1b	58 08 28	a9 c8	f0 08 20	0 e7 a0 01	••••וו	(•							*
00	930 02 02 0 940 c2 00 0	0 22 00	00 00 00	00 00	30 30 30	000000e0			2							
00	050 3c 3c 3	IC 3C 3b	38 31 29	20 1h	1a 1b 19) 1b 29 3c	<<<<:81		2							
00	060 51 64 6		73 78 80	8c 9b	a6 af b2	2 b1 b1 b1	Qdnsssx									
00	970 b1 b1 b				fa fa fa	a fa fa fa										
00	980 fa fa f	a fa fa	fa fa fa	fafa	fa fa fa	i fa fa fa	· · · · · · ·									
00		а та та 2 02 01	TA TA TA	TA TD	TC T4 e6	03 CO DO			1							
00	ao az a	is as al	al an ao	90 93	93 93 93	5 93 93 93										Ŧ
	Z Data (da	(etch etc	1 480 bytes							Packots: 13317	70 · Dicplayod· 43	3463 (100	0%) · Dror	ned: 0 (0 0%)	Profile: De	fault

Figura 69. Software Wireshark. Captura de tráfico. Recortado y reensamblado de paquetes.

				*е	enp1s0				^ _ ¤ X	
E	ile <u>E</u> dit <u>V</u> iew <u>G</u> o	<u>Capture</u> <u>A</u> nalyze <u>S</u>	tatistics Telephony <u>W</u> ire	ess <u>T</u> ools <u>H</u> elp						
	(🔳 🧟 🔘 🖿) 🗋 🖹 🏹 🖣	e 🔹 🏵 🕪 el 📃							
	ip.addr == 192.16	8.7.40							× +	
No	p. Time	Source	Destinatio	n Protoco	l Length Info				-	
	7 0.4510394	12 192.168.7.40	192.168.	.20 IPv4	1514 Fragmented IP protocol	(proto=UDP 17,	off=0, ID=7bc8)	[Reassembled in	#8]	
+	- 80.4510396	621 192.168.7.40	192.168.	.20 UDP	642 7000 → 7000 Len=2080					
	10 0.4510398	329 192.168.7.40	192.168.	.20 IPv4	1514 Fragmented IP protocol	(proto=UDP 17,	off=0, ID=7bc9)	[Reassembled in	#11]	
	11 0.4510400	048 192.168.7.40	192.168.	.20 UDP	642 7000 → 7000 Len=2080					
	12 0.4510402	256 192.168.7.40	192.168.	.20 IPv4	1514 Fragmented IP protocol	(proto=UDP 17,	off=0, ID=7bca)	[Reassembled in	#13]	
	13 0.4510404	164 192.168.7.40	192.168.	.20 UDP	642 /000 → /000 Len=2080					
* * *	<pre>> Frame 8: 642 bytes on wire (5136 bits), 642 bytes captured (5136 bits) on interface enpls0, id 0 > Ethernet II, Src: 66: bria:aa:67:930, DSt: LCFCHEFe_38:d9:06 (54:e1:ad:38:d9:06) > Internet Protocol Version 4, Src: 192.168.7.40, DSt: DSt: LCFCHEFe_38:d9:06 (54:e1:ad:38:d9:06) > 0100 = Version: 4</pre>									
	Destination A	ddress: 192.168.7.2	9							
	▶ [2 IPV4 Fragm User Datagram P	rotocol Src Port: 7	#/(1460), #8(608)]							
Ļ	Data (2080 byte	s)	000, DSL POIL. 7000							
	Data: f00820e [Length: 2080	7a001020200220bd80b	d80bd8000000000bebc200	0040800000800203c3	3c3c3c					
0 0 0 0 0 0	1b 58 1b 58 0010 00 22 0b 68 0020 00 04 08 06 0303 3c 3c 3b 36 0040 6e 73 73 73 0050 b0 af b8 ca 0060 fa fa fa fa	08 28 a9 c8 f0 08 0b d8 0b d8 00 </td <td>20 e7 a0 01 02 02 00 00 0b eb c2 00 3c 3c 3c 3c 3c 3c 19 1b 29 3c 51 64 52 b1 b1 b1 b1 b1 b1 fa fa fa fa fa fa fa fa fa fa fa fa</td> <td>-X.(</td> <td></td> <td></td> <td></td> <td></td> <td>×</td>	20 e7 a0 01 02 02 00 00 0b eb c2 00 3c 3c 3c 3c 3c 3c 19 1b 29 3c 51 64 52 b1 b1 b1 b1 b1 b1 fa fa fa fa fa fa fa fa fa fa fa fa	-X.(×	
F	Frame (642 bytes)	Reassembled IPv4 (2)	088 bytes)							

Figura 70. Software Wireshark. Captura de tráfico. Recortado y reensamblado de paquetes.

2.4.3.3. Tamaño de trama de 4128 [bytes]

Una forma de optimizar y, por consiguiente, aumentar la velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real, es aumentando el tamaño de trama.

Pudo observarse que con un tamaño de trama de 1024 [bytes], la tasa de transferencia obtenida fue de aproximadamente 48.63[MB/s]² mientras que con un tamaño de trama 2048 [bytes], la misma fue de 50.65[MB/s]³. Esto se debe al tiempo que se tarda, haciendo uso de la instrucción *memcpy*, en copiar en un buffer lo que se encuentra almacenado en memoria y transmitirlo. Este es un tiempo fijo de procesamiento que se pierde. Por lo tanto, a medida que se aumenta el tamaño de trama, dicho tiempo se hace cada vez menos considerable debido a que se está manejando un volumen mayor de datos pero si, en cambio, se trabaja con tamaño de tramas pequeños, dicho tiempo comienza a tomar importancia lo que impacta directamente en la velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real.

Entonces, si ahora el tamaño del paquete o trama empaquetada en protocolo ASTERIX CAT240 es de 4128 [bytes], donde los datos o información útil, es decir, sin contar encabezados y demás, son 4096 [bytes], tenemos que:

 $^{^{2}}$ ver ensayo 2.4.3.1.

 $^{^{3}}$ ver ensayo 2.4.3.2.

2			Terminal - root@DE1-9	SoC: ~/source	^ _ D >
File Edit View Te	erminal Tabs Help				
root@DE1-SoC: ~	-/source		×	siag001@debian-siag001: ~	×
root@DE1-SoC:~/	/source# ./dma01				
socket open					
bind success					
EOF en asterix	file.				
Se leyeron 4128	8 bytes	1 172 112 1			
sendto() scan 1	F=4523036.00 uSec	rate=472 MBits/Sec			
sendto() scan 1	F=4524695.00 uSec	rate=472 MBits/Sec			
sendto() scan i	F=4523598.00 uSec	rate=4/2 MBits/Sec			
sendto() scan i	F=4524161.00 uSec	rate=4/2 MBits/Sec			
sendto() scan	1=4525565.00 uSec	rate=4/2 MBits/Sec			
sendto() scan	1=4523232.00 uSec	rate=4/2 MBits/Sec			
sendto() scan	1=4525584.00 uSec	rate=4/2 MBIts/Sec			
sendto() scan	=4520583.00 uSec	rate=4/2 MBIts/Sec			
sendto() scan	=4520949.00 uSec	rate=4/2 MBits/Sec			
sendto() scan	=4522342.00 uSec	rate=4/2 MBits/Sec			
sendto() scan	=4531507.00 uSec	rate=4/2 MBits/Sec			
sendto() scan	=4521/38.00 uSec	rate=4/2 MBits/Sec			
sendto() scan i	=4522/49.00 usec	rate=4/2 MBits/Sec			

Figura 71. Consola de Linux. Ejecución código dma01. Tiempos de transmisión.

Podemos observar que, para este caso, un barrido o una vuelta completa de radar tarda en completarse 4.55 segundos aproximadamente.

Considerando datos o información útil, tenemos que:

65536 paquetes 4096[Bytes] 4.55[segundos] ≈59[MB/s]

La velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real se calculó de la siguiente manera:

Luego, mediante el software Wireshark, se realiza la captura del tráfico de red.

A		*enp1s0	^ _ ¤ X
File Edit View Go Captu	ure <u>A</u> nalyze <u>S</u> tatistics Telephor	y <u>W</u> ireless <u>T</u> ools <u>H</u> elp	
🥖 🔳 🖉 🕥 🗎 🗋	🖹 🙆 🔍 🚸 🔶 🍽 🕯	(📃 📃 o e a 🏦	
dp.port == 7000			₩ •
No. Time	Source Destination	Protocol Length Info	A
2027699 53.775359928	192.168.7.40 192.168.7.2	0 UDP 1210 7000 → 7000 Len=4128	
2027702 53.775416168	192.168.7.40 192.168.7.2	0 UDP 1210 7000 → 7000 Len=4128	
2027705 53.775473748	192.168.7.40 192.168.7.2	0 UDP 1210 7000 → 7000 Len=4128	
2027708 53.775529855	192.168.7.40 192.168.7.2	0 UDP 1210 7000 → 7000 Len=4128	
2027711 53.775586487	192.168.7.40 192.168.7.2	0 UDP 1210 7000 → 7000 Len=4128	
2027714 53.775641266	192.168.7.40 192.168.7.2	0 UDP 1210 7000 → 7000 Len=4128	
2027717 53.775695925	192.168.7.40 192.168.7.2	0 UDP 1210 7000 → 7000 Len=4128	
2027720 53.775751318	192.168.7.40 192.168.7.2	0 UDP 1210 7000 → 7000 Len=4128	
2027723 53.775807009	192.168.7.40 192.168.7.2	0 UDP 1210 7000 → 7000 Len=4128	
2027720 53.775018470	192.100.7.40 192.100.7.2	0 UDP 1210 7000 → 7000 Len=4120	
2027722 52 775074467	192.100.7.40 192.100.7.2	0 UDP 1210 7000 → 7000 Len=4120	
2027735 53 776029241	192 168 7 40 192 168 7 2	DP 1210 7000 - 7000 Len=4120	
2027738 53, 776086385	192.168.7.40 192.168.7.2	UDP 1210 7000 - 7000 Len=4128	
2027741 53.776141185	192.168.7.40 192.168.7.2	UDP 1210 7000 → 7000 Len=4128	
2027744 53.776198145	192.168.7.40 192.168.7.2	UDP 1210 7000 → 7000 Len=4128	
2027747 53.776254000	192.168.7.40 192.168.7.2	0 UDP 1210 7000 → 7000 Len=4128	
2027750 53.776310148	192.168.7.40 192.168.7.2	0 UDP 1210 7000 → 7000 Len=4128	
2027753 53.776364767	192.168.7.40 192.168.7.2	0 UDP 1210 7000 → 7000 Len=4128	
Frame 2027711: 1210 by	vtes on wire (9680 bits), 121	0 bytes captured (9680 bits) on interface enpls0, id 0	
▶ Ethernet II, Src: 8a:	4e:e8:a5:6f:cf (8a:4e:e8:a5:6	f:cf), Dst: LCFCHeFe 38:d9:06 (54:e1:ad:38:d9:06)	
Internet Protocol Vers	sion 4, Src: 192.168.7.40, Ds	t: 192.168.7.20	
User Datagram Protoco	l, Src Port: 7000, Dst Port:	7000	
- Data (4128 bytes)			
Data: f01020e7a0010	202000a4f5d4f5d4f5d000000000	ebc20000041000001000403c3c3c3c	
[Length: 4128]			
0000 1b 58 1b 58 10 28	e9 45 f0 10 20 e7 a0 01 02	02 ·X·X·(·E ·· ····	A
0010 00 0a 4f 5d 4f 5d	4f 5d 00 00 00 00 0b eb c2	000]0]0]	
0020 00 04 10 00 00 10	00 40 3c 3c 3c 3c 3c 3c 3c 3c	30	
	<u>30 30 30 30 30 30 30 30 39</u>	10 2 ****	
0050 25 2e 37 41 4b 57	60 67 6c 70 72 73 74 73 72	73 %.7AKW`q lprstsrs	
			٣
Frame (1210 bytes) Rea	ssembled IPv4 (4136 bytes)		

O Z Data (data.data), 4,128 bytes

Packets: 2027753 · Displayed: 675557 (33.3%) · Dropped: 0 (0.0%) Profile: Default

Figura 72. Software Wireshark. Captura de tráfico.

		*enp1s	0	^ _ ¤ X
File Edit View Go Capture Analy	ze <u>Statistics</u> Telephony	<u>W</u> ireless <u>T</u> ools <u>H</u> elp		
	🗟 🚸 🕸 🏵 🕪 🕷			
ip.addr == 192.168.7.40				*
No. Time Source	Destination	Protocol Length Info		
2027702 53.775416168 192.16	8.7.40 192.168.7.20	UDP 1210 7000 → 7000	Len=4128	
2027703 53.775453170 192.16	8.7.40 192.168.7.20	IPv4 1514 Fragmented	IP protocol (proto=UDP 17, off=0, ID=ea96) [Reassembled in #2027705]	
2027704 53.775465216 192.16	8.7.40 192.168.7.20	IPv4 1514 Fragmented	<pre>IP protocol (proto=UDP 17, off=1480, ID=ea96) [Reassembled in #2027705]</pre>	
2027705 53.775473748 192.16	8.7.40 192.168.7.20	UDP 1210 7000 → 7000	Len=4128	
2027706 53.775509276 192.16	8.7.40 192.168.7.20	IPv4 1514 Fragmented	IP protocol (proto=UDP 17, off=0, ID=ea97) [Reassembled in #2027708]	
2027/07 53.775521326 192.16	8.7.40 192.168.7.20	19V4 1514 Fragmented	IP protocol (proto=UDP 17, off=1480, ID=ea97) [Reassembled in #2027/08]	
2027700 53.775529055 192.10	8 7 40 102 168 7 20	UDP 1210 7000 → 7000	ILEN=4120	
2027710 52 775577911 192 16	8 7 40 192 168 7 20	TPv4 1514 Fragmented	IP protocol (proto-UDP 17, 011-6, 10-ea36) [Reassembled in #2027711]	
2027710 53.775586487 192.10 2027711 53 775586487 192.16	8 7 40 192 168 7 20	UDP 1210 7000 . 7000	len=4128	
2027712 53,775620465 192.16	8.7.40 192.168.7.20	TPv4 1514 Fragmented	IP protocol (proto=UDP 17, off=0, ID=ea99) [Reassembled in #2027714]	
2027713 53.775632650 192.16	8.7.40 192.168.7.20	IPv4 1514 Fragmented	IP protocol (proto=UDP 17, off=1480, ID=ea99) [Reassembled in #2027714]	
2027714 53.775641266 192.16	8.7.40 192.168.7.20	UDP 1210 7000 → 7000	Len=4128	
2027715 53.775674941 192.16	8.7.40 192.168.7.20	IPv4 1514 Fragmented	IP protocol (proto=UDP 17, off=0, ID=ea9a) [Reassembled in #2027717]	
2027716 53.775688585 192.16	8.7.40 192.168.7.20	IPv4 1514 Fragmented	IP protocol (proto=UDP 17, off=1480, ID=ea9a) [Reassembled in #2027717]	
2027717 53.775695925 192.16	8.7.40 192.168.7.20	UDP 1210 7000 → 7000	Len=4128	
2027718 53.775730086 192.16	8.7.40 192.168.7.20	IPv4 1514 Fragmented	IP protocol (proto=UDP 17, off=0, ID=ea9b) [Reassembled in #2027720]	
2027719 53.775742756 192.16	8.7.40 192.168.7.20	IPv4 1514 Fragmented	<pre>IP protocol (proto=UDP 17, off=1480, ID=ea9b) [Reassembled in #2027720]</pre>	
2027720 53.775751318 192.16	8.7.40 192.168.7.20	UDP 1210 7000 → 7000	Len=4128	
Time to Live: 64	0 7 /0 100 120 7 10	Thus 1613 Fragmontod	TD protocol (proto-UDB 17 off-0 TD-0000) [Boaccombled in #9097799]	
Protocol: UDP (17)				
Header Checksum: Oxdaeb [va	lidation disabled]			
[Header checksum status: Un	verified]			
Source Address: 192.168.7.4	0			
Destination Address: 192.16	8.7.20			
[Reassembled IPv4 in frame:	2027711]			
 Data (1480 bytes) 				
Data: 1b581b581028e945f0102	0e7a0010202000a4f5d4f5d4	4f5d000000000bebc2000004100	Ð	
[Length: 1480]				Ŧ
0000 54 e1 ad 38 d9 06 8a 4e	e8 a5 6f cf 08 00 45 00	T · · 8 · · · N · · · 0 · · · E ·		*
0010 05 dc ea 98 20 00 40 11 d	da eb c0 a8 07 28 c0 a8	· · · · · @· · · · · · (· ·		
0020 07 14 1b 58 1b 58 10 28 0	e9 45 f0 10 20 e7 a0 01	· · · X · X · (· E · · · · ·		
0030 02 02 00 0a 4T 50 4f 5d	41 50 00 00 00 00 0b eb			
	00 40 30 30 30 30 30 30 30 30 30	······································		
0060 39 36 33 28 28 27 22 18	1h 1a 1a 1a 1h 1h 19 19	963.*'".		
wireshark enp1s00D9W51.pca	apng		Packets: 2027753 · Displayed: 2026691 (99.9%) · Dropped: 0 (0.0%)	Profile: Default

Figura 73. Software Wireshark. Captura de tráfico. Recortado y reensamblado de paquetes.

						*enp1s0	^ _ ¤ ×
<u>F</u> i	le <u>E</u> dit <u>V</u> iew	<u>G</u> o <u>C</u> aptu	re <u>A</u> nalyze <u>S</u> tat	tistics Telephony	<u>W</u> ireless	<u>T</u> ools <u>H</u> elp	
	(🔳 🔏 🔘	🗎 🛅 🕻	8 🙆 🗟 🔹	* * * *		0 - 1 1	
	ip.addr == 19	2.168.7.40					× +
No	. Time	2	Source	Destination	Protocol	Length Info	
	2027702 53.	775416168	192.168.7.40	192.168.7.20	UDP	1210 7000 → 7000 Len=4128	
	2027703 53.	775453170	192.168.7.40	192.168.7.20	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=ea96) [Reassembled in #2027705]	
	2027704 53.	775465216	192.168.7.40	192.168.7.20	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=ea96) [Reassembled in #2027705]	
	2027705 53.	775473748	192.168.7.40	192.168.7.20	UDP	1210 7000 → 7000 Len=4128	
	2027706 53.	775509276	192.168.7.40	192.168.7.20	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=ea97) [Reassembled in #2027708]	
	2027707 53.	775521326	192.168.7.40	192.168.7.20	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=ea97) [Reassembled in #2027708]	
	2027708 53.	775529855	192.168.7.40	192.168.7.20	UDP	1210 7000 → 7000 Len=4128	
	2027709 53.	775565524	192.168.7.40	192.168.7.20	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=ea98) [Reassembled in #2027711]	
	2027710 53.	775577911	192.168.7.40	192.168.7.20	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=ea98) [Reassembled in #2027711]	
	2027711 53.	775586487	192.168.7.40	192.168.7.20	UDP	1210 7000 → 7000 Len=4128	
	2027712 53.	775620465	192.168.7.40	192.168.7.20	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=ea99) [Reassembled in #2027714]	
	2027713 53.	775632650	192.168.7.40	192.168.7.20	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=ea99) [Reassembled in #2027714]	
	2027714 53.	775641266	192.168.7.40	192.168.7.20	UDP	1210 7000 → 7000 Len=4128	
	2027715 53.	775674941	192.168.7.40	192.168.7.20	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=ea9a) [Reassembled in #2027717]	
	2027716 53.	775688585	192.168.7.40	192.168.7.20	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=ea9a) [Reassembled in #2027717]	
	2027717 53.	775695925	192.168.7.40	192.168.7.20	UDP	1210 7000 - 7000 Len=4128	
	2027718 53.	775730086	192.168.7.40	192.168.7.20	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=ea9b) [Reassembled in #2027720]	
	2027719 53.	775742756	192.168.7.40	192.168.7.20	IPv4	1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=ea9b) [Reassembled in #2027720]	
	2027720 53.	775751318	192.168.7.40	192.168.7.20	UDP	1210 7000 - 7000 Len-4128	
	Time to L	ive: 64	4107 460 7 344	4107 420 7 42	11874	1514 FRAMMARTA III ARAFAGAI / ARAFAEIUU 1/ AFTEM IIIBAUAI IUAAZAMMATA 18 WIMI/////	
	Protocol:	UDP (17)					
	Header Che	ecksum: 0xd	a32 [validatio	n disabledl			
	[Header ch	necksum sta	tus: Unverifie	d]			
	Source Add	ress: 192.	168.7.40	-,			
	Destinatio	on Address:	192.168.7.20				
	[Reassemb]	led IPv4 in	frame: 202771	1]			
*	Data (1480 b	vtes)					
	Data: 0000	000000000000	000000000000000000000000000000000000000	000000000000000000000000000000000000000	000000000	100000000000000000	
	[Length: :	L480]					*
00	000 54 e1 ar	AA 49 A6	8a 4e e8 a5 6	f cf 08 00 45 00	T 8	.N	
00	010 05 dc ea	98 20 b9	40 11 da 32 ci	0 a8 07 28 c0 a8			
00	020 07 14 00	00 00 00	00 00 00 00 00	0 00 00 00 00 00			
00	00 00 00 00	00 00 00	00 00 00 00 00	0 00 00 00 00 00			
00	00 00 00	00 00 00	00 00 00 00 00	0 00 00 00 00 00			
00	00 00 00	00 00 00	00 00 00 00 00	0 00 00 00 00 00			
00	00 00 00	00 00 00	00 00 00 00 00	0 00 00 00 00 00			w
	2 wirochark		W51 pcappg			Packets: 2027753 , Displayed: 2026691 (99.9%) , Dropped: 0 (0.0%) = Pro	file: Default

Figura 74. Software Wireshark. Captura de tráfico. Recortado y reensamblado de paquetes.

4	*enp1s0	^ _ ¤ X
File Edit View Go Capture Analyze Statistics Telephony M	/ireless Iools Help	
ip.addr == 192.168.7.40		A
No. Time Source Destination	Protocol Length Info	A
2027702 53.775416168 192.168.7.40 192.168.7.20	UDP 1210 7000 → 7000 Len=4128	
2027703 53.775453170 192.168.7.40 192.168.7.20	IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=ea96) [Reassembled in #2027705]	
2027704 53.775465216 192.168.7.40 192.168.7.20	IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=ea96) [Reassembled in #2027705]	
2027705 53.775473748 192.168.7.40 192.168.7.20	UDP 1210 7000 - 7000 Len=4128	
2027706 53.775509276 192.168.7.40 192.168.7.20	IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=ea97) [Reassembled in #2027708]	
2027707 53.775521326 192.168.7.40 192.168.7.20	IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=ea97) [Reassembled in #2027708]	
2027708 53.775529855 192.168.7.40 192.168.7.20	UDP 1210 7000 → 7000 Len=4128	
2027709 53.775565524 192.168.7.40 192.168.7.20	IPV4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=ea98) [Reassembled in #2027711]	
2027/10 53.7/557/911 192.168.7.40 192.168.7.20	1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=ea98) [Reassembled in #2027/11]	
	UDP 1210 7000 - 7000 Len=4128	
2027712 53.775020405 192.100.7.40 192.100.7.20	1914 Fragmented IP protocol (proto=UPP 17, off=6, ID=ea99) [Reassembled in #2027/14]	
2027714 52 775641266 102 168 7 40 102 168 7 20	1214 Fragmented IP protocol (proto-obp 17, 011-1406, ID-eass) [Reassembled II #2027714]	
2027715 52 775674041 102 168 7 40 102 168 7 20	$DDY = 1210,000 \rightarrow 000$ Lell-4120 DDV = 1514 Fragmented TD protocol (proto-UDD 17, off-0, TD-aa9a) [Paassambled in #2027717]	
2027716 52 775688585 102 168 7 40 102 168 7 20	1944 1514 Fragmented IP protocol (proto-opp 17, off-0, ID-edd) [Redssembled II #2027171]	
	100 1214 Fragmented IF protocol (proto-obr 17, 011-1400, 10-easa) [Reassembled II #2021117]	
2027718 53 775730086 192 168 7 40 192 168 7 20	TDv4 1514 Fragmented TP protocol (proto=UDP 17 off=0 TD=ea9b) [Reassembled in #2027720]	
2027719 53 775742756 192 168 7 40 192 168 7 20	1514 Engineented TP protocol (proto=UDP 17, off=1480 TD=ea9b) [Reassembled in #2027720]	
2027720 53, 775751318 192, 168, 7, 40 192, 168, 7, 20		
	Thus 1514 Fragmented TD pratecol (protection 17 offer TD-color) [Descrembled in #2007722]	
Protocol: UDP (17)		
Header Checksum: 0xfaa9 [validation disabled]		
[Header checksum status: Unverified]		
Source Address: 192.168.7.40		
Destination Address: 192.168.7.20		
 [3 IPv4 Fragments (4136 bytes): #2027709(1480), #202 	27710(1480), #2027711(1176)]	
> User Datagram Protocol, Src Port: 7000, Dst Port: 7000		
▼ Data (4128 bytes)		
Data: f01020e7a0010202000a4f5d4f5d4f5d000000000bebc2	20000041000001000403c3c3c3c	
[Length: 4128]		w
0000 1b 58 1b 58 10 28 e9 45 f0 10 20 e7 a0 01 02 02	-X-X-(-E	
0010 00 0a 4f 5d 4f 5d 4f 5d 00 00 00 00 0b eb c2 00		
0020 00 04 10 00 00 10 00 40 3c 3c 3c 3c 3c 3c 3c 3c 3c	·······@ <<<<<<	
0030 3c	<<<<<<< > >>>>>>>>>>>>>>>>>>>>>>>>>>>>>	
0040 33 2e 2a 27 22 1e 1b 1a 1a 1a 1b 1b 19 19 1a 1e	3.*************************************	v
Frame (1210 bytes) Reassembled IPv4 (4136 bytes)		
🔾 🍸 Data (data.data), 4,128 bytes	Packets: 2027753 · Displayed: 2026691 (99.9%) · Dropped: 0 (0.0%) Pro	ofile: Default

Figura 75. Software Wireshark. Captura de tráfico. Recortado y reensamblado de paquetes.

Al igual que en el ensayo anterior⁴, pudo observarse que los paquetes empaquetados en protocolo ASTERIX CAT240, al transmitirlos por UDP, son recortados y luego reensamblados para su posterior representación utilizando el software *RadarView*, debido a que se supera el *MTU* que, para el caso de Ethernet, es de 1500 [bytes]. La única diferencia con el ensayo anterior es que la fragmentación es mayor debido a que el tamaño del paquete o trama es mayor.

 $^{^{4}}$ ver ensayo 2.4.3.2.

2.4.3.4. Modo debug vs. Modo release

Con la finalidad de optimizar el código, se llevó a cabo otra prueba la cual consiste en pasar el código de *debug* a *release*.

```
root@DE0-Nano-SoC:~/source# ./dma01
socket open
bind success
EOF en asterix file.
Se leveron 4128 bytes
sendto() scan T=7094229.00 uSec rate=305 MBits/Sec
sendto() scan T=7077369.00 uSec rate=305 MBits/Sec
sendto() scan T=7077798.00 uSec rate=305 MBits/Sec
^C
root@DE0-Nano-SoC:~/source# ./dma01
socket open
bind success
EOF en asterix file.
Se leyeron 4128 bytes
sendto() scan T=4561710.00 uSec rate=474 MBits/Sec
sendto() scan T=4544897.00 uSec rate=476 MBits/Sec
sendto() scan T=4532886.00 uSec rate=477 MBits/Sec
sendto() scan T=4533208.00 uSec rate=477 MBits/Sec
```

Figura 76. Consola de Linux. Ejecución código dma01. Modo debug vs modo release. Tiempos de transmisión.

Puede observarse que la tasa de velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real es menor con el código optimizado (modo *release*) respecto al código no optimizado (modo *debug*). Esto ocurre debido a que la instrucción *memcpy* es más óptima cuando el código no está optimizado (modo *debug*)[19].

2.4.3.5.Capacidad del canal

Posteriormente, se realizó otro ensayo en el cual se cambió de lugar la instrucción *memcpy* en el código realizado.

El código empleado está realizado para tramas o paquetes variables pero como, en este caso, estamos trabajando con tramas o paquetes fijos, este cambio es realizado con el fin de obtener la capacidad del canal.

La capacidad de canal es una medida de la máxima cantidad de información que puede transmitirse de forma fiable a través de un canal de comunicaciones.

Por lo tanto:

2	Terminal - root@DE1-	SoC: ~/source	^ _ = >
File Edit View Terminal Tabs Help			
root@DE1-SoC: ~/source	×	siag001@debian-siag001: ~	×
root@DE1-SoC:~/source# ./dma01			
socket open			
bind success			
EUF en asterix tile.			
Se leyeron 4128 bytes			
sendto() scan 1=2431269.00 uSec Fate=888 MBits/Sec			
sendto() scan 1=2425/31.00 uSec Fate=888 MBits/Sec			
sendto() scan 1=2426807.00 uSec Fate=888 MB1ts/Sec			
sendto() scan 1=2425500.00 uSec rate=888 MBIts/Sec			
Sendio() Scan 1=2427046.00 USec Fale=888 MBILS/Sec			
sendto() scan T=2423923.00 uSec rate=888 MBits/Sec			
sendto() scan T=2423276.00 USec rate=866 MBILS/Sec			
condto() scan T=2374738.00 uSec rate=904 PBIts/Sec			
sendto() scan T=2374095.00 usec rate=904 PBIts/sec			
sendto() scan T=2370416.00 usec rate=904 PBits/sec			
sendto() scan T=2222027 00 uSec rate=904 MBits/Sec			
sendto() scan T=2201277 00 uSec rate=904 MBits/Sec			
sendto() scan $T=2380354$ A0 uSec rate=904 MBits/Sec			
sendto() scan T=2381286 AA uSec rate=904 MBits/Sec			
sendto() scan $T=2381233$ A0 uSec rate=904 MBits/Sec			
sendto() scan T=2386718 AA uSec rate=904 MBits/Sec			
sendto() scan $T=2389044$ A0 uSec rate=904 MBits/Sec			
sendto() scan T=2388714.00 uSec rate=904 MBits/Sec			

Figura 77. Consola de Linux. Ejecución código dma01. Tiempos de transmisión.

En pocas palabras, a diferencia de los ensayos anteriores⁵, la copia de los datos o trama empaquetada en protocolo ASTERIX CAT240 se realiza por única vez por lo que, dentro del ciclo, solo se realiza el incremento del *azimut* para, posteriormente, poder graficar.

Considerando datos o información útil, tenemos que:

$$\frac{65536 \text{ paquetes} \cdot 4096[Bytes]}{2.4[\text{segundos}]} \approx 112[MB/s]$$

La velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real se calculó de la siguiente manera:

De los resultados obtenidos, puede observarse que el canal permite alcanzar velocidades de hasta casi 900 [Mbps].

2.4.3.6. Modificación de la frecuencia del clock del PLL h2f_axi_clk

Se modificó la frecuencia del clock del PLL $h2f_axi_clk$, que es el clock encargado de manejar los clocks de los bridges entre la FPGA y el HPS.

La finalidad de modificar la frecuencia de dicho clock es observar si existe un incremento o una mejoría en la velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real.

Hasta el momento, todos los ensayos realizados fueron para una frecuencia del $h2f_axi_clk$ de 100 [MHz].

⁵ ver ensayos 2.4.3.1, 2.4.3.2, 2.4.3.3 y 2.4.3.4.



Figura 78. Bridges HPS-FPGA.

Para el caso de $h2f_axi_clk = 200 MHz$ y un tamaño de paquete o trama empaquetada en protocolo ASTERIX CAT240 de 1056 [bytes], donde los datos o información útil, es decir, sin contar encabezados y demás, son 1024 [bytes], tenemos que:

root@DE1	-SoC:-	/source# ./	dma01						
socket op	pen								
bind suce	cess								
EOF en as	steri>	file.							
Se leyer	on 105	6 bytes							
sendto()	scan	T=1226224.0	0 uSec	rate=448	MBits/Sec				
sendto()	scan	T=1228217.0	0 uSec	rate=448	MBits/Sec				
sendto()	scan	T=1227971.0	0 uSec	rate=448	MBits/Sec				
sendto()	scan	T=1227722.0	0 uSec	rate=448	MBits/Sec				
sendto()	scan	T=1227289.0	0 uSec	rate=448	MBits/Sec				
sendto()	scan	T=1227867.0	0 uSec	rate=448	MBits/Sec				
sendto()	scan	T=1226744.0	0 uSec	rate=448	MBits/Sec				
sendto()	scan	T=1228278.0	0 uSec	rate=448	MBits/Sec				
sendto()	scan	T=1230175.0	0 uSec	rate=448	MBits/Sec				
sendto()	scan	T=1227631.0	0 uSec	rate=448	MBits/Sec				
sendto()	scan	T=1227121.0	0 uSec	rate=448	MBits/Sec				
sendto()	scan	T=1228706.0	0 uSec	rate=448	MBits/Sec				

Figura 79. Consola de Linux. Ejecución código dma01. Tiempos de transmisión.

Considerando datos o información útil, tenemos que:

Por lo que la velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real se calculó de la siguiente manera:

65536 paquetes·1024[Bytes]·8[bits] 1.23[segundos] ≈436. 48[Mbps]

En cambio, para un tamaño de paquete o trama empaquetada en protocolo ASTERIX CAT240 de 4128 [bytes], donde los datos o información útil, es decir, sin contar encabezados y demás, son 4096 [bytes], tenemos que la velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real obtenida para diferentes frecuencias del clock del PLL *h2f axi clk*.

- Para el caso de *h2f_axi_clk* = 110 MHz, se obtuvo una tasa de transmisión efectiva de datos UDP de aproximadamente 494 MHz.
- Para el caso de *h2f_axi_clk = 125 MHz*, se obtuvo una tasa de transmisión efectiva de datos UDP de aproximadamente 509 MHz.
- Para el caso de *h2f_axi_clk* = 150 *MHz*, se obtuvo una tasa de transmisión efectiva de datos UDP de aproximadamente 520 MHz.

Para el caso de $h2f_axi_clk = 200 \text{ MHz}$, manteniendo el tamaño de paquete o trama empaquetada en protocolo ASTERIX CAT240 de 4128 [bytes], donde los datos o información útil son 4096 [bytes], tenemos que:

>_					Ter	minal ·	root@DE1-	SoC: ~/source	<u>~ _</u>		×
File	Edit	View	Terminal	Tabs	s He	elp					
si	ag001(@debia	n-siag00	1:~/a	ltera	a/DE1_	SoC ×	root@DE1-SoC: ~/source		×	
root	@DE1	SoC:	~/sour	ce# .	/dm	a01					
SOC	ket op	ben									
FOF	en av	sterix	x file.								
Se l	.eyer	on 412	28 byte	es							
senc	lto()	scan	T=3911	L679.	00	uSec	rate=552	MBits/Sec			
senc	lto()	scan	T=3908	3551.	00	uSec	rate=552	MBits/Sec			
senc	lto()	scan	T=3907	7390.	00	uSec	rate=552	MBits/Sec			
senc	Ito()	scan	T=3904	+321. 1272	00	usec	rate=552	MBits/Sec			
senc	Ito()	scan	T=3904	1321	00	uSec	rate=552	MBits/Sec			
senc	lto()	scan	T=3903	8685.	00	uSec	rate=552	MBits/Sec			
senc	lto()	scan	T=3902	2873.	00	uSec	rate=552	MBits/Sec			
senc	lto()	scan	T=3907	7538.	00	uSec	rate=552	MBits/Sec			
											1

Figura 80. Consola de Linux. Ejecución código dma01. Tiempos de transmisión.

Considerando datos o información útil, tenemos que:

$$\frac{65536 \text{ paquetes} \cdot 4096[Bytes]}{3.9[\text{segundos}]} \approx 69[MB/s]$$

Por lo que la velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real se calculó de la siguiente manera:

65536 paquetes·4096[Bytes]·8[bits] 3.9[segundos] ≈550[Mbps]

De los resultados obtenidos, puede observarse que un incremento en la frecuencia del clock del PLL $h2f_axi_clk$ conlleva a un aumento en la velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real.

2.4.3.7. Medición de la velocidad de lectura de la memoria compartida entre la FPGA y el HPS

Otro ensayo realizado consiste en realizar la medición de los tiempos de lectura de datos desde la memoria compartida entre la FPGA y el HPS, cuyo tamaño es de 8192 [bytes].

Además, se realizan tres lecturas de memoria por ejecución de programa (se ejecuta la instrucción *memcpy* tres veces consecutivas) y se realiza un promedio de los tiempos de lectura arrojados con la finalidad de obtener un tiempo fidedigno.



Figura 81. Consola de Linux. Ejecución código dma01. Tiempos de lectura.

Entonces, tenemos que:

 $\frac{\text{Lectura de la memoria[veces]} \cdot \text{Tamaño de la memoria[Bytes]}}{\text{Tiempo de lectura[segundos]}} = \frac{3 \cdot 8192[Bytes]}{141 \cdot 10^{-6}[segundos]} \approx 166[MB/s]$

Por lo que la velocidad de lectura de la memoria en la FPGA desde el HPS se calculó de la siguiente manera:

Velocidad de lectura de la memoria $\left[\frac{MB}{s}\right] \cdot 8[bits] = 166\left[\frac{MB}{s}\right] \cdot 8[bits] \approx 1330[Mbps]$

A continuación, en forma de resumen, se realizan tablas comparativas con la finalidad de observar de manera más sencilla los resultados obtenidos en los ensayos realizados.

Tamaño de trama	Tasa de tra	nsferencia		Frecuencia del clock
[bytes]	[MB/s]	[Mbps]	NIOdo	del PLL <i>n2<u>j</u>_axi_cik</i> [MHz]
1056 (1024 bytes útiles)	~ 48.63	~ 389	Debug	100
1056 (1024 bytes útiles)	~ 54.56	~ 436.48	Debug	200
2080 (2048 bytes útiles)	~ 50.65	~ 405.2	Debug	100
4128 (4096 bytes útiles)	~ 59	~ 472	Debug	100
4128 (4096 bytes útiles)	~ 37.81	~ 302.46	Release	100
4128 (4096 bytes útiles)	~ 61.75	~ 495	Debug	110
4128 (4096 bytes útiles)	~ 63.63	~ 510	Debug	125
4128 (4096 bytes útiles)	~ 65	~ 520	Debug	150
4128 (4096 bytes útiles)	~ 69	~ 550	Debug	200

Tabla 1. Tasa de transferencia obtenida para diferentes tamaños de trama o paquete.⁶

⁶ ver ensayos 2.4.3.1, 2.4.3.2, 2.4.3.3, 2.4.3.4 y 2.4.3.6.

Tamaño de trama	Tasa de transferencia			Frecuencia del clock	
[bytes]	[MB/s]	[Mbps]	NIOdo	[MHz]	
4128 (4096 bytes útiles)	~ 112	~ 895	Debug	100	

Tabla 2. Capacidad del canal.⁷

Tamaño de la memoria [bytes]	Tasa de transferencia			Frecuencia del clock	
	[MB/s]	[Mbps]	Modo	del PLL <i>n2j_axi_cik</i> [MHz]	
8192	~ 166	~ 1330	Debug	200	

Tabla 3. Velocidad de lectura de la memoria compartida entre la FPGA y el HPS.⁸

2.5. Software (con la implementación del handshake entre la FPGA y el HPS)

En esta sección del informe se comenta en detalle el software realizado para la comunicación entre la FPGA y el HPS, el cual se divide fundamentalmente en dos partes: una realizada en *Qsys*, en la cual se instancia todo el hardware necesario para la comunicación entre la FPGA y el HPS, y otra parte realizada en *QtCreator*, la cual se utiliza para programar el HPS.

El software realizado se encarga, en primera instancia, de mapear los puertos para lectura de memoria de la FPGA a memorias virtuales para que las mismas puedan ser accedidas desde el lado del HPS. Además, se agrega un registro extra con el cual se realiza el *handshake* entre la FPGA y el HPS. En pocas palabras, a través de dicho registro, la FPGA le dice al HPS qué memoria está lista para leer y qué cantidad de datos tiene almacenados la misma. Las memorias, las cuales tienen un tamaño máximo de 2048 [bytes], se van llenando con datos empaquetados en protocolo ASTERIX CAT240 desde el lado de la FPGA y, una vez que éstas estén listas, el HPS las va leyendo. Posteriormente, se transfiere el contenido de dichas memorias a un arreglo para, finalmente, enviar dichos datos por Ethernet a través del socket UDP.

2.5.1. Parte Qsys

El *Qsys* es una herramienta gráfica que se utiliza para el diseño digital de hardware. Dicha herramienta contiene procesadores, memorias, interfaces de I/O, timers, etc. La herramienta *Qsys* le permite al usuario, mediante una GUI, elegir los componentes deseados y automáticamente genera el hardware para conectarlos.

La herramienta *Qsys* permite diseñar el sistema y la comunicación entre el HPS y la FPGA. El HPS y la FPGA están conectadas a través de una serie de *AXI Bridges*. En este caso, nosotros trabajaremos desde el lado del HPS por lo que, para la comunicación entre ellos, se utilizan dos *bridges*:

⁷ ver ensayo 2.4.3.5.

⁸ ver ensayo 2.4.3.7.

- HPS-to-FPGA bridges (*h2f*).
- Lightweight HPS-to-FPGA bridge (*lwh2f*).

Use	Connections	Name	Description	Export	Clock	Base
V		□ clk_0	Clock Source			
	Ŷ <u></u> Ŷ	≻ clk_in	Clock Input	clk	exported	
	· · · · · · · · · · · · · · · · · · ·	≻ clk_in_reset	ResetInput	reset		
		≺ clk	Clock Output	Double-click to export	clk_0	
		≺ _clk_reset	Reset Output	Double-click to export		
~		曰 틸 hps_0	Arria V/Cyclone V Hard Processor			
	<u>}</u>	> memory	Conduit	hps_0_ddr		
		nps_io	Conduit	nps_u_io		
		> h2t_reset	Reset Output	Double-click to export	-11- 0	
		→ h2t_axi_clock	Clock input	Double-click to export	CIK_U	
		The avi clock	Clock Input	Double-click to export	[n21_axi_clock]	
		→ f2h_avi_slave	AVI Slave	Double-click to export	ff2h avi clockl	-
		→ h2f lw avi clock	Clock Input	Double-click to export		-
		≺ h2f lw_axi_crock	AXI Master	Double-click to export	[h2f lw_axi_clock]	
V		□ hps fpga leds	PIO (Parallel I/O)		fund-and-analy	
	• • • • • • • • • • • • • • • • • • • •	→ clk	Clock Input	Double-click to export	clk 0	
	• •	→ reset	ResetInput	Double-click to export	[clk]	
		→ s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_0030
	↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	external_connection	Conduit	hps_fpga_leds_external_connection		
V		🗆 pio_0	PIO (Parallel I/O)			
	•	→ clk	Clock Input	Double-click to export	clk_0	
		→ reset	ResetInput	Double-click to export	[clk]	
		→ s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	
	Q-	- external_connection	Conduit	pio_0_external_connection		
		E pll_0	Altera PLL	Dauble allaliste averat		
			Clock input	Double-click to export	CIK_U	
		 reset → outelk0 	Clock Output	Double-click to export	nll 0. outcik0	
		E asterix 0	asterix	Louine-cach to export	pii_0_outerio	
-		→ clock sink	Clock Input	Double-click to export	pli 0 outcik0	
	↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	→ clock avalon port	Clock Input	Double-click to export	pll 0 outclk0	
		→ s_Global	Avaion Memory Mapped Slave	Double-click to export	[clock_avalon_port]	
	┃┃┃┃┃┃ ≻++++++++++	- clock_source	Clock Output	clock_for_adc	asterix_0_clock_source	
	│ ◆◆ │	→ reset_sink	Reset Input	Double-click to export	[clock_sink]	
	│ │ │ │ │ │ │ │ 〉 	✓ avalon_master_1	Avalon Memory Mapped Master	Double-click to export	[clock_sink]	
		≺ avaion_master_2	Avalon Memory Mapped Master	Double-click to export	[clock_sink]	
		≺ avalon_master_3	Avaion Memory Mapped Master	Double-click to export	[clock_sink]	
		✓ avalon_master_4	Avalon Memory Mapped Master	Double-click to export	[clock_sink]	
		✓ avalon_master_5	Avalon Memory Mapped Master	Double-click to export	[clock_sink]	
		✓ avalon_master_6	Avalon Memory Mapped Master	Double-click to export	[clock_sink]	
		≺ avalon_master_7	Avalon Memory Mapped Master	Double-click to export	[clock_sink]	
		≺ avalon_master_8	Avalon Memory Mapped Master	Double-click to export	[clock_sink]	
		- conduit_BI	Conduit	DI	[CIOCK_SINK]	
		- conduit_HM	Conduit	nm	[CIOCK_SINK]	
		conduit_video	Conduit	Mdeo	[CIOCK_SINK]	
	· · · · · · · · · · · · · · · · · · ·	anchin momen/3 0	On Chip Mamony (PAM or POM)	unggen	[CIUCK_SITK]	
	•	→ clk1	Clock Input	Double-click to export	nll 0. outclk0	
		→ e1	Avalon Memory Manned Slave	Double-click to export	[c]k1]	● 0x0000
		→ reset1	Reset Innut	Double-click to export	[clk1]	- 000000
		→ s2	Avalon Memory Mapped Slave	Double-click to export	[clk2]	▲ 0x0000 0000
	• • • • • • • • • • • • • • • • • • • •	→ clk2	Clock Input	Double-click to export	clk 0	
		→ reset2	Reset Input	Double-click to export	[c]k2]	
V		onchip_memory2_1	On-Chip Memory (RAM or ROM)			
	• • •	→ clk1	Clock Input	Double-click to export	pll_0_outclk0	
		→ s1	Avaion Memory Mapped Slave	Double-click to export	[clk1]	0x0000
	+ +	→ reset1	ResetInput	Double-click to export	[clk1]	
		→ s2	Avalon Memory Mapped Slave	Double-click to export	[clk2]	0x0000_0800
	•	→ clk2	Clock Input	Double-click to export	clk_0	
		→ reset2	Reset Input	Double-click to export	[clk2]	
		E onchip_memoryz_z	On-Chip Memory (RAM or ROM)	Daubla alialata aumant		
		→ CIK1	Clock Input	Double-click to export		a 0
		- SI	Avaion Memory Mapped Slave	Double-click to export	[CIK1]	 0x3000
		- ieseti	Avalon Momony Monnod Slavo	Double click to export	[CIK1] [cIK1]	A 010000 1000
	• • • • • • • • • • • • • • • • • • •	→ clk2	Clock Input	Double-click to export	clk 0	
		→ reset2	ResetInput	Double-click to export	[c]k2]	
K		onchip_memory2 3	On-Chip Memory (RAM or ROM)		,	
_		→ clk1	Clock Input	Double-click to export	pll_0_outclk0	
	││ │ │ │ 	→ s1	Avaion Memory Mapped Slave	Double-click to export	[clk1]	● 0x2800
	│ ◆◆ │ │ ↓ │ │ │ │ │ │ │ │ │ │ │ │ │ │ │ │ │ │ │	→ reset1	Reset Input	Double-click to export	[clk1]	
	+ • • • • • • + + • • •	→ s2	Avalon Memory Mapped Slave	Double-click to export	[clk2]	0x0000_1800
	• <u>1 1 • • • </u>	→ clk2	Clock Input	Double-click to export	clk_0	
		reset2	Reset Input	Double-click to export	[CIK2]	
		in onchip_memory2_4	Cleak Input	Double elistete summer	nll 0 outol:0	
			Avalon Memory Monned Clave	Double-click to export	pii_0_outciku	A 0x2000
		+ reset1	Reset Innut	Double-click to export	[sist] [c]k1]	- 0/2000
		→ s2	Avaion Memory Manned Slave	Double-click to export	[c]k2]	● 0x0000 2000
	↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓	→ clk2	Clock Input	Double-click to export	clk 0	0.0000_2000
		→ reset2	Reset Input	Double-click to export	[clk2]	
K		onchip_memory2 5	On-Chip Memory (RAM or ROM)		,	
_	↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	→ clk1	Clock Input	Double-click to export	pll_0_outclk0	
		→ s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	▲ 0x1800
	│ │ ♦ ♦ │ → │ → │ → │ → │ │ │	→ reset1	ResetInput	Double-click to export	[clk1]	
		→ s2	Avalon Memory Mapped Slave	Double-click to export	[clk2]	0x0000_2800
	• • • • • • • • • • • • • • • • • • •	→ clk2	Clock Input	Double-click to export	clk_0	
		→ reset2	ResetInput	Double-click to export	[clk2]	
M		⊟ onchip_memory2_6	On-Chip Memory (RAM or ROM)			
		cik1	Clock Input	Double-click to export	pii_0_outcik0	0.0.1005
		- ST	Avaion Memory Mapped Slave	Double-circk to export	[CIK1]	 0X1000
		- resett	Avalan Mamon Manual Olar	Double-circk to export	[CIK1]	A 0×0000 0000
		32	Clock Input	Double click to export	clk 0	- 0x0000_3000
		reset?	Reset Innut	Double-click to export	[clk2]	
		E onchin memory? 7	On-Chin Memory (RAM or ROM)	Louinte-carch to export	[enc2]	
		→ clk1	Clock Input	Double-click to evoort	pli 0 outcik0	
		→ s1	Avaion Memory Manned Slave	Double-click to export	[c]k1]	
	│ ♦ 	→ reset1	Reset Input	Double-click to export	[c]k1]	
		→ s2	Avaion Memory Mapped Slave	Double-click to export	[clk2]	● 0x0000 3800
	•	→ clk2	Clock Input	Double-click to export	clk 0	0.0000_0000
	* *	→ reset2	Reset Input	Double-click to export	[clk2]	

Figura 82. Qsys. Componentes instanciados.

Tal como puede observarse en la imagen *ut supra*, se instancian 8 memorias, denominadas *onchip_memory2_X*, con *X* de 0 a 7. Por cada memoria se almacena un paquete o trama empaquetadas en protocolo ASTERIX CAT240.

Además, se instancia un componente denominado *asterix_0*, el cual se encarga de recibir las señales de sincronismo, de empaquetar los datos en protocolo ASTERIX CAT240, de realizar el *handshake* entre la FPGA y el HPS, de grabar las memorias con los paquetes o tramas empaquetadas en protocolo ASTERIX CAT240, entre otras funciones. Dicho componente es parte del desarrollo realizado por el Dr. Ricardo Cayssials y el alumno Mariano Valdez en el marco del proyecto.

Una vez que todo el hardware ha sido configurado correctamente, la comunicación entre el HPS y la FPGA se programa a través de una aplicación en C mapeada en memoria. Dicho mapeo de memoria permite que la CPU vea y acceda al espacio de direcciones de la FPGA, la cual contiene los componentes, para poder así leer/escribir información según sea necesario.

La aplicación en C desarrollada utiliza una API para enviar, o recibir, datos de escritura a, y desde, direcciones de memoria especificadas.

2.5.2. Parte QtCreator

Una vez configurado el hardware para la comunicación entre el HPS y la FPGA a través de la herramienta gráfica de diseño digital de hardware *Qsys*, se procede a programar el HPS.

Como se necesita que el HPS acceda a los periféricos que forman parte de la estructura de la FPGA, lo primero que se debe de realizar es la instanciación en el proyecto del *header*, el cual es generado una vez realizada la configuración en *Qsys*, mediante *#include "hps soc system.h"*.

16 #include "hps soc system.h"

Figura 83. QtCreator. Instanciación del header generado en Qsys.

Luego, se definen todas las variables y funciones necesarias para realizar los *bridges* entre el HPS y la FPGA.

```
19
       // ========
20
       #define H2F AXI MASTER BASE 0xC0000000
21
22
       // main bus; scratch RAM, conectada a h2f_axi_master
23
       // h2f bus
24
       // RAM FPGA port s2
25
       // main bus addess 0x0800 0000
26
27
       void *h2f_axi_master_virtual_base;
28
29
       // ==
30
       // lw bus;
       // h2f_lw_axi_master -> control port
31
       // read master -> f2h_axi_slave, puedo leer cualquier periferico que este en este bus,
// write_master -> onchip_memory2_0.sl = 0x00020000
33
       #define H2F_LW_AXI_MASTER_BASE
                                         0xff200000
0x00005000
34
      #define HW REGS SPAN
35
36
       // the h2f light weight bus base
37
       void *h2p_lw_virtual_base;
38
39
       // ==
40
41
42
       // HPS onchip memory base/span
43
       // 2^16 bytes at the top of memory
44
       #define HPS_ONCHIP_BASE 0xffff0000
                                    0x00010000
45
       #define HPS ONCHIP SPAN
       // HPS onchip memory (HPS side!)
46
       volatile unsigned int * hps_onchip_ptr = NULL ;
47
48
       void *hps onchip virtual base;
49
50
```

Figura 84. QtCreator. Definición de todas las variables y funciones.

Puede observarse que cada uno de los componentes poseen una dirección base. Dichas direcciones son utilizadas para acceder, controlar y enviar datos desde y hacia el SoC.

Para mapear las direcciones físicas a direcciones virtuales, lo primero que se realiza es una llamada abierta al sistema para abrir el controlador o *driver* del dispositivo de memoria *"/dev/mem"* seguido de la llamada al sistema *mmap*, la cual es utilizada para asignar la dirección física del HPS a una dirección virtual representado por el puntero $h2f_axi_master_virtual_base$.

```
54 // /dev/mem file id
55 int fd;
```

Figura 85. QtCreator. ID del archivo /dev/mem.

```
71
       //map variables
72
73
       volatile unsigned *global;
74
       volatile unsigned *meml;
75
       volatile unsigned *mem2;
       volatile unsigned *mem3;
76
77
       volatile unsigned *mem4;
78
       volatile unsigned *mem5;
79
      volatile unsigned *mem6;
       volatile unsigned *mem7;
80
       volatile unsigned *mem8;
81
```

Figura 86. QtCreator. Definición de todas las variables necesarias para realizar el mapeo de memoria.

La variable *global* es la que será utilizada para la realización del *handshake*, entre otras cosas, entre la FPGA y el HPS, tal como se explicará más adelante en la presente sección del informe.

113	D uint8_t mapping (void) {	
114		
115	// Declare volatile pointers to I/O registers (volatile	
116	// means that IO load and store instructions will be used	
117	// to access these pointer locations,	
118	// instead of regular memory loads and stores)	
119		
120	// get FPGA addresses	
121	// Open /dev/mem	
122	<pre>If((fd = open("/dev/mem", (O_RDWR O_SYNC))) == -1) {</pre>	
123	<pre>printf("ERROR: could not open \"/dev/mem\"\n");</pre>	
124	return(1);	
125	- 1	
126		
127	//	
128	// get virtual addr that maps to physical	
129	// for light weight bus	
130	// DMA status register	
131	hzp_iw_virtual_base = mmap(NULL, HW_REGS_SPAN, (PROT_READ PROT_WRITE), MAP_SHARED, fd, H2P_LW_AXI_MASTER_BAS	ж);
132	<pre>L] II(nzp iw virtual base == MAP_FAILED) {</pre>	
133	<pre>print("suuus: mmapl() falled\n"); </pre>	
1.34	close [Id])	
1.35	return(1);	
1.36		
1.37		
130	//	
140	// Manee de puertos para lectura de memoria y Bandebake	
141	77 Bargero de poetros para fectoria de manaria y nandantase b26 avi mostor vistor bardo e menor MULL de 20000000 / DEOF DEAD I DEOF MEDTE \ MAD GUADED 64 U2E AVI MAGEED	DACK N.
142	alphal = (victoria victoria and the victoria to the victoria transmission of the second transmission of the victoria to the vi	DAGE 17
143	granal = (volatile unsigned -)((ulter -)nrt_mater_virtual_naw + virtual_naw + volatile (not))))	
144	mont = (velatile unsigned -)((uinto_t +))2f avi master virtual hasertewilt = Mawney 1 = BASE) :	
145	mem3 = (inclutibe unsigned *) ((inint8 + *)b2f avi master virtual base+OMCHTP MEMORY2 2 BASE);	
146	mend = (volatile unsigned +)((vint# + +))2f avi master virtual hase+(NVMTP_MEM/NY2 3 BASE):	
147	mem5 = (volatile ungined +)((vint# t +))2f avi magter virtual hage+(NVCHIP MEM/BV2 4 RASE);	
148	mem6 = (volatile unsigned *) ((vint# t *))2f avi magter virtual hase+ONCHIP MEMORY2 5 RASE)	
149	mem7 = (volatile unsigned *) ((vint# t *)h2f axi master virtual base+ONCHIP MEMORY2 6 BASE);	
150	mem8 = (volatile unsigned *) ((vint8 t *)h2f axi master virtual base+ONCHIP MEMORY2 7 BASE);	
151		
152	//	
153		
154	// HPS onchip ram	
155	hps onchip virtual base = mmap(NULL, HPS ONCHIP SPAN, (PROT READ PROT WRITE), MAP SHARED, fd, HPS ONCHIP BAS	3E);
156		
157	1 1f (hps onchip virtual base == MAP FAILED) (
158	<pre>printf("ERROR: mmap3() failed\n");</pre>	
159	close(fd);	
160	return(1);	
161		
162	// Get the address that maps to the HPS ram	
163	<pre>hps_onchip_ptr =(unsigned int *)(hps_onchip_virtual_base);</pre>	
164		
165	//	
166		
167	return (0);	
168		

Figura 87. QtCreator. Mapeo de direcciones físicas a direcciones virtuales. Función mapping.

La dirección virtual de AXI_MASTER_BASE está representada por $h2f_axi_master_virtual_base$, que es un puntero con el cual se podrá acceder directamente a los registros en el controlador.

Esto se realiza con el fin de poder acceder a los periféricos que forman parte de la estructura de la FPGA, lo cual se logra mapeando las direcciones de la FPGA a una memoria virtual para que las mismas puedan ser accedidas desde el lado del HPS.

Para la transmisión de datos empaquetados en UDP a través de Ethernet, se realiza un *socket UDP*[1]. Para ello, en primera instancia, se deben agregar las librerías *#include* <sys/socket.h>, *#include* <unistd.h>, *#include* <arpa/inet.h> y las variables necesarias.

```
13 #include <sys/socket.h>
14 #include <unistd.h>
15 #include <arpa/inet.h>
```

Figura 88. QtCreator. Librerías dedicadas al socket UDP.

```
61 //-----
62
63 //socket variables
64 int sfd;
65 int port = 8000;
66 struct sockaddr_in address, dst_address;
67
68 //-----
```

Figura 89. QtCreator. Definición de todas las variables necesarias para la realización del socket UDP.

```
//-----
 82
 83
 84
      -void init socket(void){
 85
 86
           sfd = socket (AF INET, SOCK DGRAM, 0);
 87
      Ė
 88
           if(sfd < 0){
            printf("Socket failed\n");
 89
 90
              exit(1);
 91
           }
 92
           printf("socket open\n");
 93
 94
          int optval=1;
           setsockopt(sfd, SOL SOCKET, SO REUSEPORT, &optval, sizeof(optval));
 95
 96
 97
          address.sin family = AF INET;
 98
          address.sin_addr.s_addr = htonl(INADDR_ANY);
 99
           address.sin port = htons(port);
100
101
           int r = 0;
102
           r = bind(sfd, (struct sockaddr*) &address, sizeof(address));
103
     if(r < 0){
104
              printf("bind failed");
105
           }
106
           printf("bind success\n");
107
108
          dst address.sin family = AF INET;
109
          inet_pton(AF_INET, "192.168.7.21", &dst_address.sin_addr);
110
           dst_address.sin_port = htons(port);
      L
111
112 //-----
```

Figura 90. QtCreator. Creación del socket UDP. Función init_socket.

Tal como puede observarse, la creación del *socket* se realiza mediante la función *socket()*[8]. Para ello, debe definirse el descriptor de archivo de socket que, en este caso, es *sfd* (*socket file descriptor*).

Mediante la función *setsockopt()*[9], se establecen o setean los parámetros del *socket* creado.

Además, se debe definir un puerto, que en nuestro caso será el *port=8000*, y una estructura *sockaddr*, la cual será empleada para inicializar el *socket* creado.

Por lo tanto, se debe inicializar la dirección del *socket* fuente, denominado *in_address*, por medio de la función *bind()*[10]. Dicha función asocia el *socket* creado a una dirección particular o a un puerto específico. Lo mismo debe de realizarse para inicializar el *socket* de destino, denominado *dst_address*.

Posteriormente, dentro del *main()*, se realiza el llamado a las funciones *init_socket()* y *mapping()*, funciones comentadas anteriormente, además de la declaración de variables, cuyo uso o función se comentará a continuación.

169 // ====== 170 171 int main(void) 172 173 174 init socket(); 175 176 177 178 uint8_t a; 179 180 a = mapping(); 181 182 if(a != 0) return(EXIT FAILURE); 183 184 185 unsigned int data[1024] __attribute__((aligned(4))); 186 187 unsigned int reg0 __attribute__((aligned(4))); 188 unsigned int num_mem __attribute__((aligned(4))); 189 unsigned int payload_size __attribute__((aligned(4))); unsigned int dr __attribute__((aligned(4))); 190 volatile unsigned *mem[] = {mem1, mem2, mem3, mem4, mem5, mem6, mem7, mem8}; 191 192 193 int old buff attribute ((aligned(4))); 194 int resta attribute ((aligned(4))); 195 196 num mem = 0;197 payload size = 0; 198 dr = 0;199 old buff = 0; 200 global[2] = 0x3E8;201 202 global[3] = 0x3;global[7] = 0x10;203

Figura 91. Llamado a funciones init_socket() y mapping(). Declaración de variables.

La variable *data* es un arreglo de datos utilizado para cargar o copiar el contenido que se tiene cargado en la memoria SRAM dual port, que es nada más ni nada menos que el paquete o trama empaquetada en protocolo ASTERIX CAT240 desde el lado de la FPGA. La variable reg0 es utilizada para saber si hay o no datos almacenados o disponibles en alguna de las memorias.

La variable *num_mem* es utilizada para guardar el número de memoria a leer, es decir, guardar el número de la memoria que esté disponible con datos.

La variable *payload_size* es utilizada para guardar el tamaño del paquete o trama empaquetado en protocolo ASTERIX CAT240 que se encuentra disponible en memoria.

La variable dr es la variable utilizada para calcular el data rate o la tasa de transferencia de datos.

Las variables *old_buff* y *resta*, además de la ya mencionada variable *num_mem*, son las variables utilizadas para la verificación de pérdida de paquetes.

Tal como puede observarse en la figura *ut supra*, se les especificó a las variables comentadas previamente un atributo especial del tipo *aligned*[14] con la finalidad de optimizar el código. Al utilizar el atributo especial *aligned*, se busca que el compilador realice menos operaciones lo que mejora considerablemente la eficiencia del tiempo de ejecución.

Tal como se comentó inicialmente, para la realización del *handshake* entre la FPGA y el HPS, se agregó un registro extra el cual se conecta al HPS para que éste pueda acceder a los registros internos del módulo *asterix_0* de la FPGA. Dicho registro es mapeado a través del puntero *global*.

- Registro 0 (*global[0]*). Es de solo lectura y entrega 32 [bits].

 - Si hay mensaje para enviar desde el lado de la FPGA, entrega:
 - En los 16 [bits] menos significativos, el número de memoria en dónde se encuentran los datos almacenados. Como son 8 memorias, su valor va de 0 a 7.
 - En los 16 [bits] más significativos, el tamaño de trama o paquete (*payload* + encabezado) empaquetado en protocolo ASTERIX CAT240. Este tamaño de trama o paquete es coincidente con el campo *LEN* del paquete o trama empaquetada en protocolo ASTERIX CAT240.
- Registro 1 (*global[1]*). Es de lectura y escritura. Entrega el valor del encoder del RADAR, el cual por defecto es 4096. Básicamente, genera el *HM* cuando se generan internamente las señales de vídeo RADAR (autogeneración de vídeo).
- Registro 2 (*global[2]*). Es de lectura y escritura. Entrega el número de muestras. Por lo tanto, el tamaño de la trama dependerá del ancho de la muestra, es decir, dependerá de si se considera vídeo RADAR de 8 [bits] o de 16 [bits]. Un detalle a tener en cuenta es que el tamaño de paquete o trama no puede ser mayor a 1500 [bytes] ya que, al momento de transmitir por Ethernet UDP, se fragmenta ni tampoco puede ser mayor a 2048 [bytes], que es el tamaño de la memoria. Recordar que se almacena una trama o paquete por memoria. Otro detalle a tener en cuenta es que la trama o paquete se arma dependiendo del *Trigger* y/o del *BI* por lo que su tamaño es variable.
- Registro 3 (*global[3]*). Es de lectura y escritura. Tiene 2 [bits].
 - Bit[0] = 0, toma las señales de vídeo RADAR externas.
 - \circ *Bit[0]* = 1, genera internamente las señales de vídeo RADAR (autogeneración de vídeo).
 - Bit[1] = 0, considera vídeo RADAR de 8 [bits].
 - Bit[1] = 1, considera vídeo RADAR de 16 [bits].
- Registro 4 (*global[4]*). Solo válido en autogeneración de vídeo. Entrega la cantidad de períodos de reloj de ADC (16 [MHz]) entre *Triggers*. Por defecto está en 16000. Recordar que cada vuelta de RADAR son 32 [MS], es decir, 2 segundos por vuelta.

- Registro 5 (*global[5]*). Solo válido en autogeneración de vídeo. Entrega la cantidad de períodos de reloj de ADC (16 [MHz]) entre *BI*. Por defecto está en 7812. Recordar que cada vuelta de RADAR son 32 [MS], es decir, 2 segundos por vuelta, y se desea tener 4096 *BI* por vuelta.
- Registro 6 (*global[6]*). Entrega el valor de *CELL_DUR* en femtosegundos. Por defecto está en 62500000 [femtosegundos] para 16 [MHz] de clock de conversor AD.
- Registro 7 (*global*[7]). Solo válido en autogeneración de vídeo. Entrega la cantidad de *BI* entre generación de *HM*. En otras palabras, entrega el incremento de ángulo por cada *BI* en resolución $360^\circ = 2^{16} = 65536$. Por defecto está en 16 ya que *azimut*/*BI* = 65536/4096 = 16.

Por lo tanto, se le asignaron valores a algunos de los registros internos comentados anteriormente, tal como puede observarse en la figura *ut supra*.

Se le asignó el valor decimal 1000 (0x3E8 en hexadecimal) a global[2], que es el registro que se utiliza para modificar el tamaño del paquete o trama empaquetada en protocolo ASTERIX CAT240. Un detalle a tener en cuenta es que a global/2], como máximo, se le puede asignar el valor decimal 1440 (0x5A0 en hexadecimal) para que, una vez que los datos son encapsulados primero en protocolo ASTERIX CAT240 y luego en protocolo UDP para ser enviados por Ethernet a la red, no se fragmente debido a que se supera el MTU. Cabe recordar que el MTU es de 1500 [bytes], el encabezado IP es de 20 [bytes], el encabezado UDP es de 8 [bytes] y el encabezado ASTERIX CAT240 es de 32 [bytes] dando como resultado un payload máximo de 1440 [bytes]9. Por otro lado, se le asignó el valor decimal 3 (0x3 en hexadecimal) a global[3], que es el registro que se utiliza para la generación interna o toma externa de las señales de vídeo RADAR. Al ser su valor 3, significa que el Bit[0] = 1 y Bit[1] = 1, por lo que genera internamente las señales de vídeo RADAR (autogeneración de vídeo) y además considera vídeo RADAR de 16 [bits]. Finalmente, se le asignó el valor decimal 16 (0x10 en hexadecimal) a global[7], que es el registro que se utiliza para modificar la relación entre el incremento de ángulo (azimut) y el BI.

Tal como puede observarse en la figura *ut infra*, se ejecuta un ciclo *while* en el cual se realiza, a grosso modo, un ciclo *do* en el cual se lee el *Registro 0* con el fin de verificar si hay datos disponibles. Si es así, se lee en qué memoria se encuentra y el tamaño del mismo. Luego, se realiza una verificación de pérdida de paquetes. Posteriormente, se realiza el copiado del contenido almacenado en la memoria al arreglo *data* para realizar el envío o transmisión del mismo a través del socket UDP. Finalmente, se realiza el cálculo de la tasa de transferencia.

⁹ ver Anexo B y Anexo C.

Proyecto Final de Carrera Ingeniería Electrónica

```
234
         while(1){
235
236
                gettimeofday(&t1, NULL);
237
238
         白
                do{
239
240
                       reg0 = global[0];
241
242
                       if(reg0 == 0xFFFFFFF) continue;
243
244
         白
                        else{
245
                                num_mem = (reg0 & 0xFFFF);
246
                               payload_sise = (reg0 & 0xFFFF0000) >> 16;
247
                                resta = num mem - old buff:
248
249
250
         白
                                if( !(resta == 1 || resta == -7) ){
251
                                    printf("Error: %d\n", resta);
                                    printf("Actual: %d Anterior: %d\n", num_mem, old_buff);
252
253
254
                                }
255
256
                                el se
                                   printf("-");
257
258
259
                                break;
260
261
262
                      hwhile(1):
263
264
                   memcpy((void*)data, (const void*)mem[num_mem], payload_sise);
265
266
                    sendto(sfd, (void *)data, payload_size, 0, (struct sockaddr *)&dst_address, sizeof(dst_address));
267
268
                    old buff = num mem;
269
270
                    gettimeofday(&t2, NULL);
271
272
                    elapsedTime = (t2.tv_sec - t1.tv_sec) * 1000000.0;
273
                   elapsedTime += (t2.tv_usec - t1.tv_usec);
274
275
                    dr = (payload_size / (uint32_t)elapsedTime) * 8;
276
                   printf("data rate = %d MBits/Sec\n\r", dr);
277
278
279
          L
280
```

Figura 92. QtCreator. Lectura del Registro 0, adquisición de la trama ASTERIX CAT240, verificación de pérdida de paquetes, copiado y posterior transmisión de la trama por Ethernet a través del socket UDP y cálculo de la tasa de transferencia.

Desglosando lo comentado previamente, al comienzo del ciclo *while* se tiene un ciclo *do*, en el cual se lee el valor del *Registro 0* y se lo compara con el valor hexadecimal 0xFFFFFFFF. Si es verdadero, significa que no hay datos almacenados en memoria. En cambio, si es falso, significa que sí hay datos almacenados en memoria por lo que se leen los 16 [bits] menos significativos para saber en qué memoria (número de memoria) se encuentran y los 16 [bits] más significativos para conocer el tamaño.

Posteriormente, por medio de dos sentencias *if*, se realiza una simple verificación de pérdida de paquetes. Cabe recordar que se almacena un paquete o trama empaquetada en protocolo ASTERIX CAT240 por buffer o memoria por lo que la idea es, al momento de leer la memoria compartida entre la FPGA y el HPS desde el lado de éste último, chequear de alguna manera que no se está omitiendo ninguna memoria o, dicho de otra manera, perdiendo paquetes. Por lo tanto, en el caso de que se llegasen a perder paquetes, se imprime un error en el cual se informa el valor de la variable *resta* con el que se sabe qué cantidad de buffer o memoria fueron omitidas mientras que si se imprime el carácter "-" significa que no se perdió ningún paquete o trama.

Haciendo uso de las funciones *gettimeofday()*[15] y *elapsedTime*, la cual retorna el tiempo transcurrido entre dos valores de tiempo, calculados previamente con la función *gettimeofday()*, se calcula la tasa de transferencia de datos.

Puede observarse que el cálculo de la tasa de transferencia de datos contempla los tiempos muertos y no solo la adquisición y posterior transmisión de los datos a través de Ethernet por medio del *socket* UDP.

Haciendo uso de la función *memcpy()*[16], se transfiere el contenido almacenado en la memoria *mem[num_mem]* de tamaño *payload_size* al arreglo *data*.

 264
 memcpy((void*)data, (const void*)mem[num_mem], payload_size);

 Figura 93. QtCreator. Copiado del contenido almacenado en memoria mem[num_mem] de tamaño payload size al arreglo data mediante la instrucción memcpy().

Finalmente, por medio de la función *sendto()*[18], se envían los datos empaquetados en UDP, previamente empaquetados en protocolo ASTERIX CAT240, a través de Ethernet por medio del *socket* creado inicialmente.

266 sendto(sfd, (void *)data, payload_size, 0, (struct sockaddr *)&dst_address, sizeof(dst_address)); Figura 94. QtCreator. Envío de datos a través del socket.

2.6. Ensayos (con la implementación del handshake entre la FPGA y el HPS)

En la presente sección del documento se detallan los ensayos realizados empleando el kit de desarrollo DE1-SoC, utilizando el software desarrollado y explicado en la sección previa del presente informe.

El ensayo a realizar consiste en una prueba de medición de tiempos con el fin de conocer los tiempos de lectura y de transmisión de datos empaquetados en protocolo ASTERIX CAT240. Dichos datos son generados, empaquetados en protocolo ASTERIX CAT240 y posteriormente cargados en memoria RAM dual port, que es la memoria compartida entre la FPGA y el HPS, desde el lado de la FPGA. Se implementa, además, un *handshake* entre la FPGA y el HPS con el fin de que exista una sincronización entre ambos. Por lo tanto, por medio del mismo, la FPGA le comunica al HPS cuando el dato está listo para que éste lo lea, lo copie y posteriormente lo transmita por Ethernet a través de un socket UDP dando como resultado una velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real. Además, se realizará una prueba de visualización de los datos transmitidos por Ethernet haciendo uso del software *RadarView*, de Cambridge Pixel.

2.6.1. Medición de tiempos de lectura y de transmisión con datos empaquetados en protocolo ASTERIX CAT240

Haciendo uso de una PC y del kit de desarrollo DE1-SoC, se realizará una prueba de medición de tiempos. Dicho ensayo consiste en la lectura de datos empaquetados en protocolo ASTERIX CAT240, los cuales fueron previamente generados, empaquetados en dicho protocolo y cargados en memoria RAM dual port, que es la memoria compartida entre la FPGA y el HPS, desde el lado de la FPGA. Además, se realiza la implementación de un *handshake* entre la FPGA y el HPS con el fin de que exista una sincronización entre ambos. Así, la FPGA le comunicará al HPS cuando el dato está listo para que éste lo lea, lo
copie y posteriormente lo transmita por Ethernet a través de un socket UDP. Por lo tanto, la medición de los tiempos de lectura y de transmisión, es decir, el tiempo máximo que el HPS necesita para leer un dato en memoria, copiarlo y transmitirlo por Ethernet a través de un socket UDP, darán como resultado una velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real.

La prueba de medición de tiempos se lleva a cabo utilizando la consola de Linux, donde se ejecuta el código dma01 desde el lado del HPS. El código empleado para realizar el ensayo en cuestión está basado en un código encontrado en Internet[19][20]. En pocas palabras, el código se encarga, en primera instancia, de mapear las direcciones de la FPGA a una memoria virtual para que las mismas puedan ser accedidas desde el lado del HPS. Luego, a través de un registro, se realiza el handshake entre la FPGA y el HPS. En pocas palabras, por medio de dicho registro, la FPGA le dice al HPS qué memoria está lista para leer (num mem) y qué cantidad de datos tiene almacenados la misma (pavload size). Las memorias, las cuales tienen un tamaño máximo de 2048 [bytes], se van llenando con datos empaquetados en protocolo ASTERIX CAT240 desde el lado de la FPGA y, una vez que éstas estén listas, el HPS las va levendo haciendo uso de un puntero denominado mem[num mem]. A continuación, haciendo uso de la función memcpy()[16], se transfiere el contenido del puntero mem[num mem] a un arreglo data para, finalmente, enviar dichos datos por Ethernet a través del socket UDP empleando la función sendto()[18]. Para calcular los tiempos, se emplean las funciones gettimeofday()[15] y elapsedTime. Esta última retorna el tiempo transcurrido entre dos valores de tiempo, calculados previamente con la función gettimeofday().

Tal como se mencionó en la sección anterior del presente informe, para la realización del *handshake* entre la FPGA y el HPS, se agregó un registro el cual es mapeado a través del puntero *global*. A través de dicho puntero, se podrán realizar modificaciones a ciertos registros¹⁰.

En el Anexo B podrá encontrarse una breve reseña del protocolo ASTERIX CAT240, además de una explicación sobre cómo se arma un paquete o trama empaquetado en dicho protocolo y la correspondencia entre el mismo y su representación gráfica, lo cual resultará útil conocer al momento de observar las capturas de tráfico de red realizadas en los ensayos detallados a continuación.

Como punto de partida, se debe asegurar una velocidad de transmisión efectiva de datos que sea igual a la velocidad a la que se cargan los datos en memoria, los cuales dijimos son de tamaño variable. Es decir, como mínimo se debe ser capaz de transmitir a la misma velocidad en que almacenan los datos en memoria para así, mientras se transmite la primera trama, poder leer la segunda y prepararla para la transmisión. Esta sería la peor condición que debemos cumplir o asegurar.

Sabemos que la velocidad de muestreo del ADC es de 16 [MS/s] y cada muestra es de 2 [bytes]. Por lo tanto, la velocidad a la que se cargan los datos en memoria se calcula como:

 $16[MS/s] \cdot 2[Bytes] = 32[MB/s]$

¹⁰ ver software 2.5.

Por lo tanto, nuestra peor condición a cumplir es el tiempo que se tarda en almacenar o cargar los datos en memoria, es decir, debemos ser capaces de superar una tasa de transferencia de datos de 32[MB/s] la cual está dada por la velocidad de muestreo del ADC.

2.6.1.1. Tamaño de trama variable

El ensayo realizado es para un tamaño de paquete o trama empaquetada en protocolo ASTERIX CAT240 variable.

Tal como se explica en el Anexo B, la trama o paquete se arma con un *trigger* o con un *BI* o, dicho de otra manera, por cada señal de *trigger* o *BI* nueva, se finaliza una trama y se comienza una nueva.

De todas maneras, a través de *global[2]*, que es el registro que se utiliza para modificar el tamaño del paquete o trama empaquetada en protocolo ASTERIX CAT240, se asignó el valor decimal 1000 (0x3E8 en hexadecimal) por lo que la trama tendrá como máximo un valor de 1000 [bytes] fuera de que puede adoptar valores fijados por las señales de *trigger* o *BI* menores o iguales a 1000 [bytes].

Además, se le asignó el valor decimal 3 (0x3 en hexadecimal) a global[3], que es el registro que se utiliza para la generación interna o toma externa de las señales de vídeo RADAR. Al ser su valor 3, significa que el Bit[0] = 1 y Bit[1] = 1, por lo que genera internamente las señales de vídeo RADAR (autogeneración de vídeo) y además considera vídeo RADAR de 16 [bits].

Finalmente, se le asignó el valor decimal 16 (0x10 en hexadecimal) a global[7], que es el registro que se utiliza para modificar la relación entre el incremento de ángulo (*azimut*) y el *BI*.

Terminal - root@DE1	SoC: ~/source	_ ¤ ×
File Edit View Terminal Tabs Help		
siag001@debian-siag001: ~/altera/15.1/embedded \times	root@DE1-SoC: ~/source	×
		• • • • • •
		•••••

Por lo tanto, tenemos que:

Figura 95. Consola de Linux. Ejecución código dma01. Verificación de pérdida de paquetes.

Podemos observar que, al momento de ejecutar el código, no se omite ningún buffer o memoria por lo que no hay pérdida de paquetes.

響 root@DE1-SoC: ~/source	-	×
data rate = 274 MBits/Sec		^
data rate = 274 MBits/Sec		
data rate = 267 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MB1t5/Sec		
data rate = 274 MBits/Sec		
adda fatt - 2/4 mbits/Sec		
data tabe = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 267 MBits/Sec		
data rate = 271 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MB1ts/Sec		
data rate = 267 MBits/Sec		
data rate = 2/1 MB15/Sec		
data rate = 2/4 mbits/Sec		
data tabe = 274 Mitta/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 267 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 274 MBits/Sec		
data rate = 2/4 MBits/Sec		\checkmark

Figura 96. Consola de Linux. Ejecución código dma01. Tiempos de transmisión.

Además, podemos observar que la velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real es variable debido a que el tamaño del paquete o trama empaquetada en protocolo ASTERIX CAT240 es variable.

De todas maneras, podemos apreciar que la máxima tasa de transmisión efectiva de datos UDP en tiempo real lograda es del orden de los 270 [Mbps].

Con el fin de constatar la tasa de transmisión efectiva de datos UDP en tiempo real lograda, la cual fue calculada por software¹¹, se utilizaron los software *NetPerSec* y el *Administrador de Tareas*. Por medio de los mismos, se midió la tasa de recepción de datos UDP en tiempo real, tal como se observa de las figuras *ut infra*.

¹¹ ver software 2.5.

NetPerSec	_		\times
Graph Options Display About			
Received: 48.679 Gbit Current 290.70 Mbit/s Average 291.58 Mbit/s Max 305.88 Mbit/s Auto scale	400.(200.(0 bit)	00 Mbit/s 00 Mbit/s /s	
Sent: 1.91 Mbit Current O bit/s Average 77.49 Kbit/s Max 768.06 Kbit/s ✓ Auto scale	1.00 500.(0 bit/	Mbit/s 00 Kbit/s ⁄s	
Display ✓ Current	it/s) (B/s)	Reset data	1
Aceptar Car	ncelar	Ауис	la

Figura 97. Software NetPerSec. Tasa de recepción de datos UDP en tiempo real.



Figura 98. Administrador de Tareas. Rendimiento. Tasa de recepción de datos UDP en tiempo real.

Puede decirse entonces que, a efectos prácticos, ambas tasas (de transmisión y de recepción) son coincidentes. Se estima que la leve discrepancia existente entre ambas radica en la forma en la que se realiza el cálculo de la tasa de transmisión por software ya que la arrojada tanto por el software *NetPerSec* y el *Administrador de Tareas* son muy similares.

Haciendo uso del software *RadarView*, de Cambridge Pixel, lo que hacemos es graficar los datos enviados por Ethernet a través del socket UDP.



Figura 99. Software RadarView. Visualización de paquetes transmitidos en protocolo ASTERIX CAT240.

En este punto, es necesario realizar una aclaración. Debido a que el conversor AD utilizado es de 14 [bits], la resolución de datos es de 16 [bits] (14 [bits] del conversor AD utilizado + 2 [bits] de padding) por lo que es necesario, para poder visualizar los datos enviados por Ethernet a través del socket UDP en el software *RadarView*, modificar manualmente el parámetro *ChanAPimMaxBytesPerSample* en el archivo de configuración *SPxRadarView-240.rpi*.

Tal como se observa en la figura *ut infra*, dicho parámetro viene con el valor 1 por defecto, es decir, viene dado para una resolución de datos de 8 [bits].

Parameter Name	Start- up?	Туре	Description	Default Value
ChanAPimMaxBytesE ChanBPimMaxBytesE	PerSample PerSample			
	Yes	INT	Maximum number of bytes per sample that can be received from an input source. Set to either 1 or 2. If set to 2 then RadarView can receive 2-byte samples from a source; however at present the samples are truncated to 1 byte for processing and display. Enabling support for 2-byte samples increase the memory resources required by RadarView at start-up.	1

Figura 100. Software RadarView. Manual de Usuario. Parámetro ChanAPimMaxBytesPerSample.

Por lo tanto, fue necesario modificar el valor del parámetro *ChanAPimMaxBytesPerSample* de 1 a 2 para así poder visualizar los datos enviados por

Ethernet a través del socket UDP en el software *RadarView* de manera correcta. Para ello, utilizando el *Block de Notas*, se debe abrir el archivo *SPxRadarView-240.rpi* y modificar manualmente dicho parámetro tal y como se observa en la figura a continuación.

SPxRadarView-240.rpi: Bloc de notas Archivo Edición Formato Ver Ayuda						- 1	<
######################################							^
<pre># General Parameters ChanAName = "Channel-A" ChanAMastangeMetres = 20000.000 ChanARadarVMetres = 0.000 ChanANorthOfsetDeg = 0.000 ChanAVideoStatusTimeoutSecs = 3 ChanATurningStatusTimeoutSecs = 3 ChanAPrfIndicatorEnabled = 1 ChanAPeriodIndicatorEnabled = 1</pre>							
# Polar Store Parameters ChanAPimBangeMode = 2 ChanAPiaScamMode = 3 ChanARJSScamMode = 0 ChanARJSSreKB = 4096							ł
ChanAPimMaxBytesPerSample = 2 # Para 8 bits = 1 ; Para 8 ###################################	######################################						
ChanAPImRangeSamples = 2048 ChanAPimAzimuths = 2048 ChanAPimFocessingNode = 0 ChanAPimFocessingNode = 0 ChanAPimAziInputRef = 0 ChanAPimAziIFillEnabled = 1 ChanAPimAziClearLimit = 10 ChanAPimAziClearLimit = 32 ChanAPimAziConstantLimit = 0		-					
# Source Parameters ChanASrcType = 1							~
s		L	ínea 1, columna 1	100%	Windows (CRLF)	UTF-8	•

Figura 101. Software RadarView. Modificación del parámetro ChanAPimMaxBytesPerSample en el archivo .rpi.

📕 Captura	_Tramas_Ext_16bit.p	capng				-	- 0	×
Archivo	Edición Visualiza	ción Ir Captura	Analizar Estadística	as Telefo	ía Wireless Herramientas Ayuda			
	🛞 📙 🗅 🗙	🔓 🍳 👄 👄 🕾	I 🕂 🕹 📃 🔳		. 拜			
udp.port	: == 8000						\times \rightarrow	- +
No.	Time	Source	Destination	Protocol	Lengt Info			^
32508	3 1.538880575	192.168.7.40	192.168.7.21	UDP	1098 8000 + 8000 Len=1056			
32509	9 1.538911841	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32516	0 1.539036704	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32511	L 1.539073917	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32512	2 1.539095542	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32513	3 1.539115271	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32514	1.539137006	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32515	5 1.539137050	192.168.7.40	192.168.7.21	UDP	394 8000 → 8000 Len=352			
32516	5 1.539167281	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32517	7 1.539188332	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32518	3 1.539207432	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32519	9 1.539228095	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32526	0 1.539247936	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32521	L 1.539257665	192.168.7.40	192.168.7.21	UDP	778 8000 → 8000 Len=736			
32522	2 1.539286431	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32523	3 1.539317292	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32524	1.539348940	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32525	5 1.539381344	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32526	5 1.539412762	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32527	7 1.539443402	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			×
> Frame	32518: 1098 byt	tes on wire (8784	bits), 1098 byt	es captu	ed (8784 bits) on interface enpls0, id 0			
> Ethern	et II, Src: ea:	:73:1b:ae:ba:99 (ea:73:1b:ae:ba:9	9), Dst:	PcsCompu_3b:41:b6 (08:00:27:3b:41:b6)			
> Intern	et Protocol Ver	rsion 4, Src: 192	.168.7.40, Dst:	192.168.	.21			
> User D	atagram Protoco	ol, Src Port: 800	0, Dst Port: 800	0				
✓ Data (1056 bytes)							
Dat	a: f00420e7a001	L0202014a7550f900	f9000000181a03b9	aca00005	40000200101b181c18			
[Le	ngth: 1056]							
0000 07	15 16 40 16 40	04 00 00 74 50	04 00 -7 -0 01					
0020 07	02 01 40 1T 40	104 20 00 70 10	04 20 27 30 01 00 18 15 03 b0					^
0040 ac	a0 00 05 04 00	00 02 00 10 1b	18 1c 18 1d 18					
0050 le	18 1f 18 20 18	21 18 22 18 23	18 24 18 25 18		- "·#-\$-%-			
0060 26	18 27 18 28 18	29 18 2a 18 2b	18 2c 18 2d 18	&·'·(·	• • • • • • • • • • • • • • • • • • •			
0070 <mark>2e</mark>	18 2f 18 30 18	31 18 32 18 33	18 34 18 35 18	. • / • 0 • :	· 2·3·4·5·			
0080 36	18 37 18 38 18	39 18 3a 18 3b	18 3c 18 3d 18	6.7.8.9	· 1·2·2·<			
0090 <u>3e</u>	18 3T 18 40 18	41 18 42 18 43	18 44 18 45 18	>	- B-C-D-E-			
0080 45 00h0 40	18 4f 18 50 18	51 18 52 18 53	18 54 18 55 18	N-O-P-	- B-S-T-II-			~
	Data (data data) 1.0	56 byte(c)	10 51 10 55 10		Paqueter: 61934 : Montrado: 61721 (00.99	()	Per	61: Dafa It

Luego, mediante el software Wireshark, se realiza la captura del tráfico de red.

Figura 102. Software Wireshark. Captura de tráfico.

🚄 Captura_Tramas_Ext_16bit.p	ocapng				-	٥	×
Archivo Edición Visualiza	ción Ir Captura	Analizar Estadística	s Telefor	ía Wireless Herramientas Ayuda			
📶 🔳 🔬 💿 📙 🛅 🗙	🔓 🔍 👄 🔿 🖻	i Ŧ 🕹 📃 📃	ଇ୍ଇ୍∈	. 班			
udp.port == 8000						×I⇒	• +
No. Time	Source	Destination	Protocol	Lengt Info			^
32608 1.541911824	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32609 1.541943066	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32610 1.541974329	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32611 1.542007980	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32612 1.542008032	192.168.7.40	192.168.7.21	UDP	138 8000 → 8000 Len=96			
32613 1.542041370	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32614 1.542070992	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32615 1.542101622	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32616 1.542133053	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32617 1.542164344	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056		-	_
32618 1.542194971	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32619 1.542227169	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32620 1.542254541	192.168.7.40	192.168.7.21	UDP	1034 8000 → 8000 Len=992			
32621 1.542286764	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32622 1.542317481	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32623 1.542349665	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32624 1.542380532	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32625 1.542411956	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32626 1.542443610	192.168.7.40	192.168.7.21	UDP	1098 8000 → 8000 Len=1056			
32627 1.542474883	192.168.7.40	192.168.7.21	UDP	1098 8000 + 8000 Len=1056			Y
> Frame 32612: 138 byte	es on wire (1104	bits), 138 bytes	capture	(1104 bits) on interface enp1s0, id 0			
> Ethernet II, Src: ea	:73:1b:ae:ba:99 (ea:73:1b:ae:ba:9	9), Dst:	PcsCompu_3b:41:b6 (08:00:27:3b:41:b6)			
> Internet Protocol Ver	rsion 4, Src: 192	168.7.40, Dst:	192.168.	.21			
> User Datagram Protoco	ol, Src Port: 800	0, Dst Port: 800	9				
✓ Data (96 bytes)							
Data: f00060e7a001	10202014a75aef920	f92000000fa803b9a	aca000050	0400002001a90faa0f			
[Length: 96]							
0000 08 00 27 3b 41 be	iea 73 1bae ba	99 08 00 45 00	···';A··	s · · · · · E·			
0010 00 7c 2a 29 40 00	0 40 11 80 ba <u>c</u> 0	a8 07 28 c0 a8	· *)@·@	· ····(··			
0020 07 15 1f 40 1f 40	00 68 e2 aa f0	00 60 e7 a0 01	···0·0·	h			
0030 02 02 01 4a 75 ac	≘ f9 20 f9 20 00	00 0f a8 03 b9	···· Ju ··				
0040 ac a0 00 05 00 40	5 00 00 20 01 a9 Faf be ef b1	of ba of ba of					
0050 b4 of b5 of b6 of	f b7 0f b8 0f b1	0f ba 0f bb_0f					
0070 bc 0f bd 0f be 0f	fbf0fc00fc1	0f c2 0f c3 0f					
0080 c4 0f c5 0f c6 0f	fc70fc80f						
O Z Data (data data) 96	huta(e)			Danuatar: 61824 : Mortrado: 61721 (00.9%)		Parfil	Default
	oyac(o)			Paquetes: 01037 (9060 a00; 01/21 (95,6%)		1 Perns	Derduit

Figura 103. Software Wireshark. Captura de tráfico.

Vale destacar que los datos son transmitidos desde el HPS, cuya dirección IP es 192.168.7.40, y son recibidos por la PC, cuya dirección IP es 192.168.7.21.

Puede observarse que la trama está definitivamente empaquetada en protocolo ASTERIX CAT240 debido a que se identifica el primer byte de la misma con f0.

Además, ingresando a la viñeta *Estadísticas -> Propiedades de archivo de captura* del software *Wireshark*, puede realizarse un análisis más minucioso del tráfico de red capturado. En la figura *ut supra* puede observarse la tasa de recepción de datos UDP en tiempo real, tasa prácticamente coincidente con la arrojada por el software *NetPerSec* y el *Administrador de Tareas*, constatando así la tasa de transmisión efectiva de datos UDP en tiempo real lograda.

Datalas								
Archivo								
Nombre: Longitud: Hash (SFA256): Hash (SFEMD160): Hash (SFA1): Formato: Encapsulado:	C: Lisers \Agustin \AppData \Local\Temp 5305MB b09493565410413860d8e21c96aeb2bi 1819477767665335253777681abe991 136bd7a04776af8c5fd9996af42f47564 Wireshark\ opcpng Ethernet	(wireshark_Ethernet 2FAXB91.pcapng bde77814694266432ccbd058d998ba1 3764848 427bfab4	7a					
Intervalo								
Primer paquete: Último paquete: Transcurrido:	2023-08-14 11:52:52 2023-08-14 11:55:19 00:02:27							
Captura								
Hardware: SO: Aplicación:	Intel(R) Core(TM) i7-7500U CPU @ 2.7 64-bit Windows 10 (2009), build 19045 Dumpcap (Wireshark) 3.4.4 (v3.4.4-0-	0GHz (with SSE4.2) gc33f6306cbb2)						
Interfaces								
<u>Interfaz</u> Ethernet 2	Paquetes perdidos 9732 (0.2%)	<u>Filtro de captura</u> ninguno		<u>Tipo de enlace</u> Ethernet		<u>Límite de tamaño de paquete</u> 262144 bytes		
Estadísticas								
<u>Medida</u> Pagetas Espaco de tiempo, s Promedio de tamaño de paquete, 8 Bytes Promedio de bytes/s Promedio de bytes/s	Cabburatio 5825068 147.016 39622.0 8777 5107431370 344 27714		<u>Mostrado</u> 5825037 (100.0%) 140.634 41419.9 877 5107428098 (100.0%) 36M 290M		Marcado — — — — 0 —			

Figura 104. Software Wireshark. Tasa de recepción de datos UDP en tiempo real.

Haciendo uso del software *AsterixInspector*, se puede realizar un análisis minucioso de un paquete o trama empaquetada en protocolo ASTERIX CAT240 capturado.

Block Offset	Length	Category			Hex		đ
DIOCKOTISEC	Length	category			00000000: F00420E7 A0010202 02030E73 A5B0A5E	0 <u>8</u> . ç	r°¥°
xu	1050	240			00000010: 000009C9 03B9ACA0 00050400 0002001	0 ···É·*- ····	
					00000020: CA09CB09 CC09CD09 CE09CF09 D009D10	9 E-E-I-I-I-I-D 0 0 0 0 0 0 0 0	N· U
					00000040: DA09DB09 DC09DD09 DE09DF09 E009E10	9 Ú-Ô-Ô-Ô-Â	á
					00000050: E209E309 E409E509 E609E709 E809E90	9 â · ã · à · à · æ · ç · è	e-é
					00000060: EA09EB09 EC09ED09 EE09EF09 F009F10	9 ê.ê.ì.î.î.î.ð	١٠ñ
					00000070: F209F309 F409F509 F609F709 F809F90	9 ò ·ó ·ô ·õ ·ö · + •ø	s ∙ù
					00000080: FA09FB09 FC09FD09 FE09FF09 000A010	A ú∘û∘ü∘ý∘þ∘ÿ∘∘	
					00000090: 020A030A 040A050A 060A070A 080A090	A	
Record Offset	Time	Length	#Items		000000000: 0A0A0B0A 0C0A0D0A 0E0A0F0A 100A110	A	
					000000000 12081308 14081308 10081708 100817	A	
					I UUUUUUUCU: IAUAIDUA ILUAIDUA ILUAIPUA ZUUAZIU		
0x3					0000000d0: 220A230A 240A250A 260A270A 280A290	 A. ".≢.s.≋.s.!.((.)
0x3		1053	7		000000d0: 220A230A 240A250A 260A270A 280A29 000000e0: 220A230A 240A250A 260A270A 280A29 000000e0: 2A0A280A 2C0A2D0A 2E0A2F0A 300A310	 A "-≢-\$-\$-&-'-(A *-+-, -=/-0	(\cdot) (\cdot)
0x3		1053	7		00000000000000000000000000000000000000	A " · ∉ · \$ · \$ · \$ · • · (A * · + · , · − · . · / ·0 A 2 · 3 · 4 · 5 · 6 · 7 · 8	(·))·1)·9
0x3		1053	7		000000001 IA0AID0A ICONID0A ICONID0A 200A211 00000040: 220A230A 240A250A 260A270A 280A294 00000060: 2A0A2B0A 2COA2D0A 2E0A2F0A 300A311 00000100: 3A0A3B0A 3COA3D0A 3E0A3F0A 400A410	A "-#-\$-\$-\$'-(A *-+-,/-0 A 2-3-4-5-6-7-8 A :-;-<=->-?-@	(·))·1]·9 [·A
0x3		1053	7		00000000 1001001 001001 001001 001000 00000000	A ".#.\$.\$.6.'.(A *.+.,/.0 A 2.3.4.5.6.7.8 A :.;.<.=.>.?.0 A B.C.D.E.F.G.H	(·))·1)·9)·A [·I
0x3		1053	7		00000000: 12002100 12002100 12002100 0000000: 22002200 24002500 24002500 24002500 2002 00000000: 22002200 24002500 24002500 2500250 00000100: 32002500 34003500 34003500 34003500 34003500 00000110: 42004500 4000340 44004500 44004700 4500470 00000110: 42004500 44004500 44004500 44004700 45004500 00000120 45004500 44004500 44004500 45004700 550051 00000120 45004500 44004500 45004700 45004500	A ". #. \$. \$. \$	(·) · 1 · 9 · A · 1 · 1 · 2
0x3		1053	7		000000001 2002000 20000000 00000000 2002000 2000000 00000000	A "+f+.5+.5+.5+.7+.7 A *++,/0 A 2.3+4.5+6.7+8 A :; -<=>? @ A B-C-D-E-F-G-H A J-K-L-M-N-0-P A R-S-T-U-V-W-X Z-(-)-1X	(·) ·1 ·9 ·A ·1 ·1 ·9 ·1 ·9 ·1 ·9 ·1 ·9 ·1 ·9 ·1 ·9 ·1 ·9 ·1 ·9 ·1 ·9 ·1 ·9 ·1 ·1 ·9 ·1 ·1 ·1 ·1 ·1 ·1 ·1 ·1 ·1 ·1
0x3		1053	7		000000001 10001000 1000100 1000100 1000100 00000000	A "+ f - f + f + f + f + f + f + f + f + f	(·) · 1 · 9 · A · 1 · 1 · 2 · 2 · 4 · 4 · 1 · 1 · 1 · 1 · 1 · 1 · 9 · 1 · 1 · 9 · 1 · 1 · 1 · 1 · 1 · 1 · 1 · 1 · 1 · 1
0x3		1053	7		000000001 2002500, 20	A " # .5 .8 . 5 . ' . (A * + . , / .0 A 2.3 . 4.5 . 6 .7 .8 A : ; <	(·) · 1 · 9 · A · 1 · 2 · 2 · 2 · 4 · 2 · 3 · 4 · 1 · 4 · 1 · 4 · 4 · 4 · 1 · 4 · 4 · 4 · 4 · 4 · 4 · 4 · 4
0x3		1053	7		000000003 200302 200305 20050 0000006 20030 200305 20050 0000006 200300 20050 20050 0000006 200300 20050 20050 0000100 200300 20050 0000100 20050 20050 0000000000000000000000000	A ". #.G. %.G.'.(A A *.+.,/O A 2.3.4.5.6.7.8 A 1::< -=>.7.6 A B.C.D.E.F.G.H A J.K.L.M.N.O.P A R.S.T.U.V.W.X A Z.[.\.].^X A D.C.d.e.f.g.h A b.C.d.e.f.g.h A j.k.l.m.n.o.p A j.s.t.u.v.w.x	()) 1) 9 A I Q Y A I Q Y A Y Y Y
0x3 Item Offset	Length	1053 FRN	7 Field Type	Description	000000001 20002000 20002000 20002000 20002000	$\begin{array}{cccccccccccccccccccccccccccccccccccc$	(·) ·1 ·A ·A ·Q ·Q ·Q ·Q ·Q ·Q ·Q ·Q ·Q ·Q
0x3 Item Offset 0x5	Length 2	1053 FRN 1	7 Field Type 1240/010	Description Data Source Identifier	0000000001 20002000 2000200 000000000 2000200 2000200 00000000	" # G	(·) ·1 ·2 ·2 ·4 ·4 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2
0x3 Item Offset 0x5	Length 2	1053 FRN 1	7 Field Type 1240/010	Description Data Source Identifier	000000003 200320 2400350 24003	A	(·) ·1 ·1 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2
0x3 Item Offset 0x5 0x7	Length 2	1053 FRN 1 2	7 Field Type 1240/010 1240/000	Description Data Source Identifier Message Type	00000000000000000000000000000000000000	<pre></pre>	(.) .)
0x3 Item Offset 0x5 0x7 0.0	Length 2	1053 FRN 1 2	7 Field Type 1240/010 1240/000	Description Data Source Identifier Message Type	000000003 200302 2400350 24003	A " # G % 5 C · (/ 0 A 2 · 4 / 0 A 2 · 4 / 0 A 2 · 4 / 0 A 2 · 5 / 0 A 3 · 5 / 0 A 4 · 5 / 0 A 5 · 5	(·)· ·1 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2
0x3 Item Offset 0x5 0x7 0x8	Length 2 1 4	1053 FRN 1 2 3	7 Field Type 1240/010 1240/000 1240/020	Description Data Source Identifier Message Type Video Record Header	00000000000000000000000000000000000000		(·) ·1 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2
0x3 Item Offset 0x5 0x7 0x8 0xc	Length 2 1 4 12	1053 FRN 1 2 3 6	7 Field Type 1240/010 1240/000 1240/020 1240/020 1240/021	Description Data Source Identifier Message Type Video Record Header Video Header Femto	000000000 20020300 20020300 000000000 20020300 20020300 20020300 000000000 20020300 20020300 20020300 000000000 20020300 20020300 20020300 000000000 20020300 20020300 20020300 000000000 20020300 20020300 20020300 000001000 20020300 20020300 20020300 000001200 20020300 20020300 20020300 000001200 20020300 20020300 20020300 000001200 20020300 20020300 20020300 000001200 20020300 20020300 20020300 000001200 20020300 20020300 20020300 20020300 000001200 20020300 20020300 20020300 20020300 20020300 000001200 20020300 20020300 20020300 20020300 20020300 20020300 000001200 20020300 20020300 20020300 20020300 20020300	1 4 4 4 4 1 1 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5	(·) 0·1 8·9 8·A 1·I ·Q ·Q ·Q ·Q ·Q ·Q ·Q ·Q ·Q ·Q
bx3 Item Offset 0x5 0x7 0x8 0xc 0x18	Length 2 1 4 12 2	1053 FRN 1 2 3 6 7	7 Field Type 1240/010 1240/000 1240/020 1240/020 1240/041 1240/048	Description Data Source Identifier Message Type Video Record Header Video Header Femto Video Cells Resolution & Data Compression Indicator	000000000 20042004 20042104 20042104 000000000 20042004 20042104 20042104 000000000 20042004 20042104 20042104 000000000 20042004 20042104 20042104 000001200 20042104 20042104 20042104 000001200 20042104 20042104 20042104 000001200 20042104 20042104 20042104 000001200 20042104 20042104 20042104 000001200 20042104 20042104 20042104 000001200 20042104 20042104 20042104 000001200 20042104 20042104 20042104 000001200 20042104 20042104 20042104 000001200 20042104 20042104 20042104 000001200 20042104 20042104 20042104 000001200 20042104 20042104 20042104 000001200 20042104 20042104 20042104 0000012000 20042104 <td>A " 4 - 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5</td> <td>(·) ·1 ·1 ·1 ·2 ·4 ·1 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2</td>	A " 4 - 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5 + 5	(·) ·1 ·1 ·1 ·2 ·4 ·1 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2
0x3 Item Offset 0x5 0x7 0x8 0xc 0x18 0x1a	Length 2 1 4 2 2 2 5	1053 FRN 1 2 3 6 6 7 8	7 Field Type 1240/010 1240/000 1240/020 1240/020 1240/041 1240/048 1048/049	Description Data Source Identifier Message Type Video Record Header Video Header Femto Video Cells Resolution & Data Compression Indicator Video Octets & Video Cells Counters	000000000 200021000 200021000 200021000 0000000000 200021000 200021000 200021000 200021000 0000000000 2000210000 20002100000000000 20002100000000000000000000000000000000	1 4 5 4 5 6 5 6 6 0 7 6 6 5 6 5 6 5 7 6 7 5 7	(·) ·1 ·2 ·4 ·1 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2 ·2

Figura 105. Software AsterixInspector. Análisis de paquete o trama empaquetada en protocolo ASTERIX CAT240.

Mediante dicho software, pueden observarse los diferentes campos del protocolo ASTERIX CAT240 tal y como se detallan en el Anexo B^{12} .

A continuación, se realiza una tabla comparativa con la finalidad de observar de manera más sencilla los resultados obtenidos en los ensayos realizados con y sin el *handshake* entre la FPGA y el HPS.

Tamaño de trama	Tasa de tra Sin Han	ansferencia dshake ¹³	Tasa de tra Con Har	ansferencia 1dshake ¹⁴	Frecuencia del clock del PLL
[bytes]	[MB/s]	[Mbps]	[MB/s]	[Mbps]	h2f_axi_clk [MHz]
1056 (1024 bytes útiles)	~ 49	~ 389	~ 36	~ 290	100

Tabla 4. Comparativa en la tasa de transferencia. Sin Handshake vs. Con handshake.

¹² ver Tabla B.1.

¹³ ver ensayo 2.4.3.1.

¹⁴ ver ensayo 2.6.1.1.

3. Conclusiones

El proyecto aquí presentado constituye varias etapas que se fueron llevando a cabo a lo largo del tiempo, dando como resultado distintas conclusiones para cada una:

La primera etapa de investigación y desarrollo, en donde se evaluaron diferentes alternativas a través de una extensa investigación y se eligió la implementación utilizada, la cual permitió absorber nuevos conocimientos y permitió el desarrollo de un juicio crítico para elegir un camino que sea acorde a los tiempos de desarrollo del proyecto.

La segunda etapa, en donde se realizó toda la configuración necesaria para poner el marcha el kit utilizado en el desarrollo, permitió la introducción en un nuevo campo de estudio y también aprender nuevos conceptos en materia de sistemas embebidos.

La configuración realizada es compleja por lo que debe hacerse de manera minuciosa ya que cualquier detalle omitido puede dar como resultado una mala configuración y posterior errónea puesta en marcha del kit de desarrollo DE1-SoC.

La tercera etapa consistió en ensayos de la implementación sobre el HPS, para conocer la capacidad del canal y la tasa de transmisión de datos, a su vez la velocidad de lectura del mismo sobre una memoria compartida entre él y la FPGA.

Una vez realizados los ensayos o pruebas, se puede decir que efectivamente se logró el objetivo propuesto, el cual es leer de una memoria compartida entre la FPGA y el HPS datos o tramas empaquetadas en protocolo ASTERIX CAT240, copiarlas y transmitirlas por Ethernet a una velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real mayor a 32[MB/s], que es la velocidad a la que se escribe la memoria desde el lado de la FPGA.

De los ensayos realizados pudo observarse que, a medida que se aumenta el tamaño del paquete o trama empaquetada en protocolo ASTERIX CAT240, la velocidad (throughput) o tasa de transmisión efectiva de datos UDP en tiempo real también lo hace aunque, durante la transmisión de los mismos por Ethernet, dichos paquetes debian de ser fragmentados y luego reensamblados debido a que se supera el MTU que, para el caso de Ethernet, es de 1500 [bytes]. Desde otra perspectiva, mientras más pequeño sea el tamaño de la trama, menor velocidad (throughput) o tasa de transmisión efectiva de datos UDP en tiempo real se logra fuera de que, para estos casos, no existía fragmentación de paquetes. Esta baja en el rendimiento se debe a la instrucción *memcpy*, la cual agrega un tiempo de procesamiento fijo que se pierde, que es el tiempo que se necesita para copiar en un buffer lo que se encuentra almacenado en memoria, cualquiera sea el tamaño de trama o paquete. De los ensayos realizados se observó que, a medida que se aumenta el tamaño de trama, dicho tiempo se hace cada vez menos considerable debido a que se está manejando un volumen mayor de datos pero si, en cambio, se trabaja con tamaño de tramas pequeños, dicho tiempo comienza a tomar importancia lo que impacta directamente en la velocidad (throughput) o tasa de transmisión efectiva de datos UDP en tiempo real. Con esto, queda demostrado que la fragmentación de paquetes no impacta negativamente en el rendimiento sino que el causante de la baja en la velocidad (throughput) o tasa de transmisión efectiva de datos UDP en tiempo real se debe al tiempo de procesamiento fijo que se pierde al emplear la instrucción *memcpy* para tamaños pequeños de tramas o paquetes.

Pudo observarse además que el limitante que se tiene en la velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real no es la lectura de los datos en la FPGA desde el HPS sino que es la transmisión de los mismos por Ethernet, es decir, el cuello de botella se encuentra en la capacidad del canal. De los ensayos realizados, pudimos observar que la lectura de los datos se hizo a una velocidad de 166 [MB/s]¹⁵ mientras que la capacidad máxima del canal fue de 112 [MB/s]¹⁶.

Cada ensayo brindó individualmente aportes y conclusiones, las cuales fueron necesarias para el desarrollo del ensayo o prueba posterior. Por lo tanto, puede concluirse que la línea de trabajo adoptada cumple con las expectativas y requerimientos solicitados.

Todas las pruebas expuestas hasta este punto fueron realizadas sin la lógica de sincronización entre la FPGA y el HPS, es decir, sin el *handshake* entre la escritura y lectura de la memoria compartida, donde la FPGA se encarga de escribir y el HPS de leer.

La cuarta etapa conlleva la unificación de dos partes: una implementación en FPGA, la cual recibe muestras de video RADAR además de generar internamente las señales *trigger*, *BI* y *HM*, y genera paquetes de tamaño variable en el protocolo ASTERIX CAT240, que son grabados en buffers compartidos por ambas partes (FPGA y HPS). La segunda parte es un software que se ejecuta sobre un OS montado en un HPS en donde se genera una sincronización entre ambas partes para enviar el contenido de dichos buffers por Ethernet hacia una computadora que ejecute un software de graficación de video RADAR en el protocolo mencionado anteriormente.

Una vez realizado el ensayo o prueba, se pudo concluir que efectivamente se logró el objetivo propuesto, el cual es leer de una memoria compartida entre la FPGA y el HPS datos o tramas empaquetadas en protocolo ASTERIX CAT240, copiarlas y transmitirlas por Ethernet a una velocidad (*throughput*) o tasa de transmisión efectiva de datos UDP en tiempo real mayor a 32[MB/s], que es la velocidad a la que se escribe la memoria desde el lado de la FPGA. De esta manera, puede concluirse que la línea de trabajo adoptada cumple con las expectativas y requerimientos solicitados.

La tasa de transmisión obtenida en la cuarta etapa fue del orden de los 290 [Mbps]¹⁷. La realización del *handshake* tiene un efecto negativo en la tasa de transferencia debido al tiempo muerto que aparece al estar el HPS esperando a que la FPGA llene un buffer.

Aumentar la frecuencia del bus master del HPS, la cual se optó en dejar a 100 [MHz], no mostró mejoras significativas en esta etapa. De hecho, una frecuencia mayor a 150 [MHz] provocó que el sistema se congelara.

La tasa de transmisión puede mejorarse, pero para eso es necesario aumentar el tamaño máximo de muestras modificando el registro pertinente por software, lo cual, en base a los ensayos realizados, si el tamaño de muestras es cercano al tamaño máximo del buffer, se generan desincronizaciones y pérdidas de paquetes. Por tanto, luego de los ensayos realizados, se determinó que tramas de hasta 1440 [bytes]¹⁸ son ideales para evitar cualquier tipo de retraso y desincronización en ambas partes, conllevando la pérdida de

¹⁵ ver ensayo 2.4.3.7.

¹⁶ ver ensayo 2.4.3.5.

¹⁷ ver ensayo 2.6.1.1.

¹⁸ ver Anexo C.

paquetes, efecto para nada deseado. Aún así, es posible aumentar por hardware el tamaño de los buffers, de tal manera de aumentar el tamaño de muestras de los paquetes y que no se produzca pérdida de los mismos, pero esto no se consideró necesario por el momento. Además, considerar un tamaño de trama mayor a 1440 [bytes] conlleva a que se supere el *MTU* permitido por Ethernet, generando una fragmentación de paquetes. Cabe destacar que el valor de *MTU* está dado para obtener la mayor eficiencia en la red ya que fragmentar implica perder eficiencia debido a que se deben procesar más encabezados[32]. Aunque para UDP, que es el tipo de transmisión de datos por Ethernet empleada, la fragmentación es transparente (en realidad es el protocolo IP el que realiza este trabajo, maneja fragmentación en el origen y reensamblado en el destino) no existe ninguna garantía de que los paquetes lleguen o lo hagan en el mismo orden en que fueron enviados sin contar que no realiza verificación o control de pérdida de los mismos[33].

Anexo A: Placa DE1-SoC de Terasic

1. Placa DE1-SoC de Terasic

1.1. DE1-SoC de Terasic. Diagrama en bloque

En la Figura A.1 observada a continuación se detalla el diagrama en bloque del kit de desarrollo DE1-SoC de Terasic, el cual contiene la FPGA Intel Cyclone V 5CSEMA5F31C6.



Figura A.1. Kit de desarrollo DE1-SoC de Terasic. Diagrama en bloque.

1.2. DE1-SoC de Terasic. Layout

En la Figura A.2 y Figura A.3, observadas a continuación, se detalla el layout del kit de desarrollo DE1-SoC de Terasic.



Figura A.2. Kit de desarrollo DE1-SoC de Terasic. Layout. Parte posterior.



Figura A.3. Kit de desarrollo DE1-SoC de Terasic. Layout. Parte superior.

Para un análisis más extenso y detallado de las características y/o especificaciones del kit de desarrollo DE1-SoC, además de otra documentación relevante, referirse a la bibliografía correspondiente[1][21][22][23].

2. Cyclone V. Descripción general

2.1. Introducción al HPS de Cyclone V

Tal como se observa en la Figura A.4 a continuación, el chip SoC Cyclone V 5CSEMA5F31C6 consiste o se divide en dos partes: la porción HPS y la porción FPGA.

			Altera	a SoC F	PGA D	evice		
	HPS Portion					F	PGA Por	tion
Flash Controlle	ers	SDRAM Controller Subsystem				- Control Block	User I/O	HSSI Transceivers
Cortex- On-Chi Memori	-A9 M ip es	IPU Subsys Suppo Periphe	ort erals	HPS-I Interf	FPGA faces	(LUTs, F	FPGA AMs, Mul	Fabric tipliers & Routing)
PLLs	Int Peri	terface ipherals Debug				PLLs	Hard PCle	Hard Memory Controllers

Figura A.4. Chip SoC Cyclone V 5CSEMA5F31C6. Porción FPGA y porción HPS.

Cada una de las porciones del dispositivo, es decir, el HPS y la FPGA, tienen sus propios pines. Los pines no son compartidos libremente entre ambos. Los pines I/O de la FPGA son configurados a través del HPS o cualquier fuente externa soportada por el dispositivo. Los pines I/O del HPS, en cambio, son configurados por software ejecutado en el HPS. El software que configura los pines I/O del HPS es el *preloader*, tal como se explicará en la sección siguiente del presente informe.

2.2. Características del HPS

En la Figura A.5 a continuación se observa un diagrama en bloque de las características más relevantes del HPS.



Figura A.5. HPS. Diagrama en bloque.

2.3. Interfaces HPS-FPGA

Las interfaces HPS-FPGA proveen de una variedad de canales de comunicación entre el HPS y la FPGA. Dichas interfaces, las que utilizaremos son las que se enuncian a continuación:

- Bridge de FPGA a HPS.
- Bridge de HPS a FPGA.

2.4. Proceso de booteo del HPS

Bootear software en el HPS es un proceso multietapa. Cada etapa es responsable de cargar la siguiente. La primera etapa del software es la *ROM de arranque* cuyo código localiza y ejecuta la segunda etapa, denominada *preloader*, que localiza (sí está presente) la siguiente etapa. El *preloader* y las etapas de software subsecuentes son denominadas colectivamente como *software de usuario* (*software user*).



Figura A.6. HPS. Flujos de arranque o booteo.

El *preloader* es una de las etapas de arranque o de booteo más importantes. En realidad, es lo que se denomina como *source* o *fuente* debido a que todas las etapas anteriores no pueden ser modificadas.

En la Figura A.6 *ut supra* se muestran los flujos de arranque o de booteo disponibles del HPS. Las etapas de *reinicio* y de *ROM de arranque* son las únicas partes *fijas* del proceso de booteo o de arranque. Todo lo contenido dentro de las etapas del software de usuario (*software user*) puede ser *personalizado*.

Las etapas de *reinicio*, la *ROM de arranque* y el *preloader* siempre están presentes en el flujo de booteo o de arranque del HPS. Lo que viene después del *preloader* depende del tipo de aplicación que se desee ejecutar.

El HPS puede ejecutar dos tipos de aplicaciones:

- Aplicaciones *bare-metal*.
- Aplicaciones sobre un OS (Linux).

En nuestro caso, como veremos en la sección siguiente del presente informe, utilizaremos ésta última.

3. Uso del Cyclone V. Información general

Es posible utilizar el SoC Cyclone V 5CSEMA5F31C6 en 3 (tres) configuraciones diferentes:

- FPGA-only.
- HPS-only.
- HPS & FPGA.

La configuración que nos interesa a nosotros es la última, es decir, la configuración HPS & FPGA. Para ello, es necesario correr una aplicación sobre un OS (Linux). A continuación, ampliaremos.

3.1. Proceso de booteo del HPS

Ejecutar código sobre un OS Linux tiene varias ventajas. En primer lugar, el kernel libera la CPU1 del reinicio al arrancar por lo que todos los procesadores están disponibles. Además, el kernel se inicializa y hace que la mayoría, por no decir todos, los periféricos del HPS estén disponibles para su uso por el programador. Esto es posible ya que el kernel de Linux tiene acceso a una gran cantidad de drivers. El código multiproceso es mucho más fácil de escribir ya que el programador tiene acceso a las llamadas de las familias de subprocesos del sistema. Finalmente, el kernel de Linux no se limita a ejecutar programas compilados en C. De hecho, siempre se puede ejecutar código escrito en otro lenguaje de programación siempre y cuando se instale previamente el entorno de ejecución requerido (debe estar disponible para los procesadores ARM).

Sin embargo, ejecutar una aplicación embebida sobre un OS también tiene desventajas. Debido al sistema de memoria virtual establecido por el OS, un programa no puede acceder directamente a los periféricos del HPS a través de sus direcciones asignadas o mapeadas de memoria física. En su lugar, primero se necesita mapear las direcciones físicas de interés en el espacio de direcciones virtuales del programa en ejecución. Sólo entonces será posible acceder a los registros de un periférico. Idealmente, el programador debería escribir un driver de dispositivo para cada componente específico que esté diseñado para tener una interfaz limpia entre el código de usuario y los accesos al dispositivo.

Al final del día, las aplicaciones bare-metal y las aplicaciones que ejecutan código sobre Linux pueden hacer lo mismo. En términos generales, la programación sobre Linux es superior y mucho más fácil en comparación al código bare-metal ya que sus ventajas superan ampliamente sus desventajas.

3.2. Estructura del proyecto

El proceso de desarrollo crea muchos más archivos en comparación con un diseño FPGA-only. Se usará la estructura de carpetas que se observan en la Figura A.7 para organizar el proyecto. Se utilizará *DE1_SoC_demo* como nombre de proyecto.

- El directorio *hw* contiene todos los archivos relacionados con el hardware.
- El directorio *sw* contiene todos los archivos relacionados con el software.
- El directorio *sdcard* contiene todo lo necesario para crear una tarjeta SD válida desde la que el DE1-SoC pueda bootear o arrancar.



Figura A.7. Estructura del proyecto DE1_SoC_demo.

El diseño completo del tutorial puede encontrarse en *DE1_SoC_demo.zip*[24].

Anexo B: Protocolo ASTERIX CAT240

ASTERIX es un conjunto de protocolos estándar empleado para el intercambio de datos de información de RADAR entre sistemas, propuesto por EUROCONTROL.

Los estándares ASTERIX identifican una colección de tipos de mensajes, llamados categorías o CAT. En nuestro caso, es CAT240, es decir, RADAR Video Transmission, que se utiliza para transferir datos de videos RADAR o, dicho de otra manera, para distribución de video RADAR.

Después de su especificación en el año 2009, el protocolo ASTERIX ha sido adoptado como el estándar de vídeo en red[25][26].

En la Tabla B.1, puede observarse la composición o conformación del mensaje empaquetado en protocolo ASTERIX CAT240[13].

En la imagen *ut infra*, en cambio, puede observarse la correspondencia entre el paquete ASTERIX CAT240 y su representación gráfica[25]. Vemos que el protocolo ASTERIX maneja coordenadas polares (*azimut*).



Figura B.1. Correspondencia entre el paquete ASTERIX CAT240 y su representación gráfica.

Field	Length [bytes]	Type of data	Hexadecimal value/Description
ASTERIX category	1	fixed	f0
Length	2	variable	Total length in bytes of the current frame, including the CAT and LEN fields.
FSPEC	2	fixed	e7a0
Data source identifier	2	variable	Identification of the system from which the data are received.
			SIC and SAC fields.
Message type	1	fixed	02 (Video Message).
Video record header	4	variable	Incremental 32 bits number (<i>MSG_INDEX</i>).
Start azimuth	2	variable	Start azimuth of the cell group.
End azimuth	2	variable	End azimuth of the cell group.
Starting range	4	variable	Starting range in number of cells.
Cell duration	4	variable	Duration of a video cell in femto seconds.
Compression	1	fixed	00 (No Compression applied).
Resolution ¹⁹	1	fixed	<i>04</i> (High Resolution Coding Length in bits: 8).
			05 (Very High Resolution Coding Length in bits: 16).
Number of video bytes	2	variable	Number of video bytes (without compression).
Number of cells	3	variable	Total number of radar video cells.
Repetition	1	variable	Number of 64 bytes video blocks.
Medium video block	64 x Repetition	variable	Radar video data (cells).

Tabla B.1. Protocolo ASTERIX CAT240. Conformación de la trama o paquete.

Tanto de la Tabla B.1 como de la Figura B.1 puede notarse que el paquete encapsulado en protocolo ASTERIX consiste o está formado por un encabezado (*video header*) y un bloque de datos (*video data block*). El encabezado, el cual es de 32 [bytes] fijos de

¹⁹ Resolution = 0x04, ver ensayo 2.4.3.; Resolution = 0x05, ver ensayo 2.6.1.

información no útil, referencia la información que posteriormente es representada en un visualizador de video RADAR.

En el caso real, la trama o paquete se arma con un *trigger* o con un *BI*, que son señales enviadas por el radar. En otras palabras, por cada señal de *trigger* o *BI* nueva, se finaliza una trama y se comienza una nueva.

La frecuencia del trigger varía dependiendo del tipo de radar.

Por otro lado, para comenzar a graficar, siempre se espera el primer *trigger* luego del *HM*, que también es una señal enviada por el radar. El *HM* marca el final e inicio del barrido, el cual puede estar referenciado al norte (*north-up*), a proa (*head-up*) o al curso actual de la embarcación (*course-up*)[27]. Un detalle a tener en cuenta es que el protocolo ASTERIX define que el video RADAR debe estar referenciado al Norte[13].

En las imágenes observadas a continuación, se observa una representación gráfica para clarificar un poco lo comentado.



Figura B.2. Señales enviadas por el radar.



Figura B.3. Representación gráfica de los datos en base a las señales provenientes del radar.

De las imágenes observadas anteriormente, puede decirse que por cada pulso de *trigger* se retorna al origen o centro del radar mientras que por cada pulso de *BI* se concatenan los datos hasta el nuevo pulso de *trigger*, con el cual se vuelve al centro del radar, dando además el *rango* del radar. En pocas palabras, podemos decir que tanto el *trigger* como el *BI* referencian las muestras, las cuales son de tamaño variable debido a que no existe sincronización entre el *trigger* y el *BI*.

Además, por barrido o vuelta completa al radar, se tienen 65536 paquetes cuyo tamaño dependerá del tamaño del paquete o trama empaquetada en protocolo ASTERIX CAT240, el cual es variable. En otras palabras, se divide el barrido en 65536 partes. Por lo tanto, el *azimut* es una palabra de 16 bits ($2^{16} = 65536$). Esto está definido por el protocolo ASTERIX CAT240 empleado[13].

Anexo C: Modelo OSI

El modelo OSI corresponde a un estándar de los protocolos de red cuyo objetivo se basa en interconectar sistemas de distinta procedencia simplificando el intercambio de información. La implementación del presente trabajo se adhiere a los protocolos existentes para establecer conectividad con cualquier dispositivo estándar.

A continuación se describen las diferentes capas que componen el modelo OSI. Cabe destacar que en el desarrollo se implementan sólo las cuatro primeras capas del modelo: capa física, capa de enlace de datos, capa de red y capa de transporte.



Figura C.1. Capas del modelo OSI.

1. Capa física

La capa física ya está implementada por un PHY externo. Por lo tanto, no se requiere realizar ninguna tarea al respecto con la FPGA. La conexión de la capa física con la MAC es independiente del medio. Para el proyecto es necesario utilizar el protocolo RGMII.

1.1. Capa de enlace de datos: Ethernet II

La figura a continuación muestra los diferentes campos que forman la trama Ethernet II[28][29]. A continuación se detalla cada uno de ellos.

802.3 Ethernet packet and frame structure											
Layer	Preamble	Start of frame delimiter	MAC destination	MAC source	802.1Q tag (optional)	Ethertype (Ethernet II) or length (IEEE 802.3)	Payload	Frame check sequence (32-bit CRC)	Interpacket gap		
	7 octets	1 octet	6 octets	6 octets	(4 octets)	2 octets	46-1500 octets	4 octets	12 octets		
Layer 2 Ethernet frame											
Layer 1 Ethernet packet & IPG	← 72-1530 octets →										

Figura C.2. Campos que conforman la trama Ethernet II.

• <u>Header:</u>

- **Preamble:** (7 [bytes]) es una secuencia de bits usada por el medio físico para la estabilización y sincronización. Indica el comienzo de un paquete y permite al emisor y receptor establecer la sincronización. El patrón del preámbulo es el siguiente:

10101010 10101010 10101010 10101010 10101010 10101010 10101010

- **Start frame delimiter:** (1 [byte]) indica el final del preámbulo y el comienzo de la trama Ethernet. Los bits que integran el campo SFD son: *10101011*. La recepción de la secuencia anterior significa que los datos recibidos a continuación son válidos.
- **Destination MAC address:** (6 [bytes]) contiene la dirección MAC del dispositivo de destino.
- Source MAC address: (6 [bytes]) contiene la dirección MAC del dispositivo de origen.
- **Ethertype:** (2 [bytes]) valores menores a 1500 indican que el campo se utiliza para determinar el tamaño del paquete y valores mayores a 1536 indican el protocolo de encapsulamiento (EtherType). En el segundo caso, el largo de la trama se determina por la ubicación del espacio entre paquetes y FCS. El valor *0x0800* (2048 en decimal, por lo tanto, correspondiente a EtherType) indica que se utiliza el protocolo IPv4.
- **Data:** (mayor o igual a 46 [bytes]) incluye información útil (*payload*). En el caso de que la información sea menor a la cantidad mínima permitida (46 [bytes]) se realiza padding. Este consiste en agregar ceros hasta completar la cantidad de bytes mínima permitida.

• <u>Trailer:</u>

FCS: (4 [bytes]) corresponde al CRC. Permite la detección de datos corruptos una vez que son recibidos. El valor de FCS se calcula como una función de los campos protegidos de MAC: *dirección de origen* y *destino*, *Ethertype*, *datos* y *padding*. Se utiliza un polinomio específico para realizar la función CRC32[30]. El receptor

debe calcular un nuevo FCS cuando recibe los datos y comparar el FCS recibido con el calculado.

- Interpack Gap o Interframe Gap: es un tiempo de espera entre cada paquete de datos, tiene una longitud mínima de 12 [bytes] en cero.

1.2. Capa de red: Trama IPv4

En la figura a continuación se muestra el frame IPv4 y se detallan cada uno de sus campos:



Figura C.3. Campos que conforman la trama IPv4.

- Versión: (4 [bits]) para IPv4 es 4.
- **IHL:** (4 [bits]) es la longitud de la cabecera IPv4. Se mide cada 4 [bytes], por lo que una longitud de 5 se refiere a 5 x 4B = 20 [bytes]. El mínimo tamaño será 5 si la cabecera no posee opciones IP. Caso contrario, si posee IP, la mayor longitud podría ser 15.
- **DSCP y ECN:** (1 [byte]) estos campos indican el valor de precedencia, la probabilidad de pérdida de paquetes y el servicio de red utilizado. Los primeros 6 bits para colocar el DSCP y los últimos 2 bits para ECN.

7	6	5	4	3	2	1	0
		EC	CN				

Figura C.4. DSCP y ECN.

DSCP define un CS que corresponde al campo precedencia del antiguo TOS. Se reemplazó este campo por los actuales DSCP y ECN. Para tráfico normal: DSCP = 000000 y ECN = 00.

- **Total length:** (2 [bytes]) se refiere a la longitud total del paquete en [bytes] incluyendo tanto la cabecera como el *payload* (*data*).

- **Identification:** (2 [bytes]) es un identificador de grupo de segmentos de un mismo datagrama IP. Es decir, si un paquete es fragmentado durante la transmisión, todos los fragmentos contendrán el mismo número de identificación del paquete IP original al que pertenecen.
- Flags: (3 [bits]) se considera el bit más significativo primero:
 - Bit 0: debe ser 0, es un bit reservado.
 - Bit 1: se coloca en 1 si el paquete no debe ser fragmentado. Para la comunicación realizada debe ser 1 (sin fragmentar).
 - Bit 2: MF es 1 si el paquete está fragmentado, sino es 0. Para este caso debe ser 0.
- **Fragment Offset:** indica el lugar del datagrama donde pertenece un fragmento. Se mide en unidades de 8 [bytes]. El primer fragmento tiene un *fragment offset* igual a 0.
- **TTL:** (1 [byte]) en cada *salto*, por ejemplo cuando un mensaje se transmite entre routers, el tiempo de vida del paquete disminuye en una unidad y cuando llega a cero es descartado. El valor de este campo por defecto es 128.
- **Protocol:** (1 [byte]) define qué protocolo se usa en el campo *data*. El valor 17 corresponde a UDP mientras que el valor 6 corresponde a TCP.
- **Header Checksum:** (2 [bytes]) su función es detectar discrepancias por cambios accidentales en la información. Si el checksum calculado en el destino con el recibido difiere el paquete es rechazado. Se calcula teniendo en cuenta los campos de la cabecera de la trama IP. Se halla el valor sumando todos los campos de la cabecera (exceptuando el propio checksum) y realizando el complemento a uno.
- Source IP Address: (4 [bytes]) dirección IP del dispositivo de origen.
- Destination IP Address: (4 [bytes]) dirección IP del dispositivo de destino.
- **IP Options:** (variable entre 0 y 40 [bytes]) campo para variedad de propósitos que no es frecuentemente usado.
- **Padding:** (variable) garantiza que la cabecera IP finalice con una longitud múltiplo de 4 [bytes], agregando ceros de ser necesario.

1.3. Capa de transporte

1.3.1. Trama UDP

- **Source port:** (2 [bytes]) puerto de origen. Cada proceso tiene un puerto asignado. El puerto de origen puede ser cero si no se espera respuesta.
- **Destination port:** (2 [bytes]) puerto de destino.
- **Length:** (2 [bytes]) es la longitud en bytes del paquete incluyendo el *header* y los *datos*. El valor mínimo es 8 [bytes] (longitud del header).

- **Checksum:** (2 [bytes]) corresponde a la cabecera UDP, los datos, junto con los campos *length, protocol, source and destination address* de la trama IP. En IPv4 este campo es opcional, por lo tanto, puede completarse con ceros.



Figura C.5. Campos que conforman la trama UDP.

1.3.1.1. Tamaño máximo del payload en del mensaje UDP

Existe un límite máximo en la longitud de los mensajes que pueden transmitirse por Ethernet v2 correspondiente al MTU, el cual, para Ethernet es de 1500 [bytes] aunque el campo *length* del protocolo UDP indique que puede enviarse un *payload* de 65.535 [bytes], Ethernet es el limitante.

La cantidad máxima de bytes del *payload* que pueden transmitirse por Ethernet utilizando los protocolos IPv4 y UDP puede definirse por la siguiente ecuación[31]:

 $\begin{array}{l} Payload\ m\acute{a}x.\ Ethernet\ =\ 1500[bytes]\ (MTU)\ -\ 20[bytes]\ (IP\ Header)\\ -\ 8[bytes]\ (UDP\ Header) \end{array}$

 $Payload m \acute{a}x. Ethernet = 1472[bytes]$

Por lo tanto, el tamaño máximo del *payload* del mensaje UDP que puede transmitirse a través de Ethernet sin realizar fragmentación es de 1472 [bytes].

Agradecimientos

A la familia y amigos cercanos durante todo el transcurso de la carrera y el presente proyecto final, por su apoyo incondicional, su cariño y su infinita paciencia para con nosotros, aún en tiempos turbulentos.

Al Ing. Christian L. Galasso, investigador del SIAG y docente de la Universidad Tecnológica Nacional Facultad Regional Bahía Blanca, por confiar en nosotros y darnos la oportunidad de pertenecer al presente proyecto, además de su apoteósica paciencia y predisposición para resolver cada una de las inquietudes y conflictos que se nos fueron presentando a lo largo del tiempo.

Al abnegado Ing. Diego Martínez, personal civil del SIAG, por su inestimable colaboración, tanto al introducirnos con la implementación que se llevó a cabo en este proyecto como en el desarrollo del mismo.

Al abnegado Dr. Ricardo Cayssials y al alumno Mariano Valdez, por desarrollar la parte de diseño de hardware del proyecto (captura de muestras de video RADAR y posterior empaquetado en protocolo ASTERIX CAT240 de las mismas) y su inestimable predisposición a la hora de depurar los problemas que iban surgiendo en los ensayos, siempre en forma puntual y con la mejor voluntad.

A los docentes de la Universidad Tecnológica Nacional Facultad Regional Bahía Blanca, por los conocimientos y experiencia compartidos para con nosotros, siempre con el objetivo de formarnos como profesionales de la manera más óptima.

A todas aquellas personas que fueron parte, directa o indirectamente, de nuestro objetivo y que por motivos de desafortunado olvido no hemos mencionado.

A todos... ;Gracias!

4. Referencias

- [1] https://people.ece.cornell.edu/land/courses/ece5760/DE1_SOC/SoC-FPGA%20Des ign%20Guide_EPFL.pdf
- [2] https://github.com/BPI-SINOVOIP/BPI-M4-bsp/issues/4
- [3] https://www.sysadmit.com/2020/03/linux-habilitar-ssh-root.html
- [4] https://www.ochobitshacenunbyte.com/2015/04/30/habilitar-usuario-root-conexione s-ssh-debian/
- [5] <u>https://infoaleph.wordpress.com/2008/05/20/como-cambiar-el-idioma-del-ambiente</u> _grafico-de-gnulinux-debian-despues-de-la-instalacion/
- [6] https://ciksiti.com/es/chapters/8566-how-to-change-debian-desktop-environment
- [7] https://users.cs.jmu.edu/bernstdh/web/common/lectures/summary_unix_udp.php
- [8] https://man7.org/linux/man-pages/man2/socket.2.html
- [9] <u>https://man7.org/linux/man-pages/man2/setsockopt.2.html</u>
- [10] <u>https://man7.org/linux/man-pages/man2/bind.2.html</u>
- [11] <u>https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C/Manejo_de_archivo</u>
- [12] <u>https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C/Estructuras_y_Unio</u> <u>nes#Uniones</u>
- [13] <u>https://www.eurocontrol.int/sites/default/files/content/documents/nm/asterix/20</u> 150513-asterix-cat240-v1.3.pdf
- [14] <u>https://gcc.gnu.org/onlinedocs/gcc-3.3/gcc/Type-Attributes.html</u>
- [15] <u>https://linuxhint.com/gettimeofday_c_language/</u>
- [16] <u>https://www.tutorialspoint.com/c_standard_library/c_function_memcpy.htm</u>
- [17] <u>https://es.wikipedia.org/wiki/Endianness</u>
- [18] <u>https://man7.org/linux/man-pages/man2/sendto.2.html</u>
- [19] <u>https://people.ece.cornell.edu/land/courses/ece5760/DE1_SOC/HPS_peripherial</u> <u>s/FPGA_addr_index.html</u>
- [20] <u>https://people.ece.cornell.edu/land/courses/ece5760/DE1_SOC/HPS_FPGA/D</u> MA/HPS_to_FPGA/DMA_1.c
- [21] <u>https://www.terasic.com.tw/en/</u>
- [22] <u>https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&Catego</u> ryNo=165&No=836#contents
- [23] <u>https://www.intel.com/content/www/us/en/homepage.html</u>
- [24] <u>https://github.com/sahandKashani/SoC-FPGA-Design-Guide/blob/master/DE1</u> <u>SoC_demo.zip</u>
- [25] Giacomo Longo, Enrico Russo, Alessandro Armando and Alessio Merlo (2022). *Attacking (and defending) the Maritime Radar System.*
- [26] Dr. David G. Johnson and Mr. Richard Warren. Using ASTERIX CAT-240 for Radar Video Distribution - Practical Considerations from Deployed Applications.
- [27] Alan Bole, Alan Wall and Andy Norris. (2014). *RADAR AND ARPA MANUAL*. *Radar, AIS and Target Tracking for Marine Radar Users* (Third Edition). UK: Elsevier.
- [28] <u>https://es.wikipedia.org/wiki/Ethernet</u>
- [29] <u>https://www.geeksforgeeks.org/ethernet-frame-format/</u>
- [30] <u>https://docs.xilinx.com/v/u/en-US/xapp209</u>
- [31] <u>Processing efficiency : "The most reliable and efficient UDP packet size", Stack</u> <u>Overflow</u>
- [32] <u>https://netwgeeks.com/fragmentacion-ipv4/</u>

[33] <u>https://www.enmimaquinafunciona.com/pregunta/793/pueden-paquetes-tcp-y-u</u> <u>dp-ser-divididas-en-partes</u>