

Universidade de Vigo

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN



PROYECTO FIN DE CARRERA

Ingeniería de Telecomunicación

SISTEMA DE GESTIÓN REMOTA POR ETHERNET PARA PLATAFORMA DE INSTRUMENTACIÓN VIRTUAL

Autor: Pablo Ríos Costas

Tutor: Enrique Soto Campos

Curso: 2014 - 2015

SISTEMA DE GESTIÓN REMOTA POR ETHERNET PARA PLATAFORMA DE INSTRUMENTACIÓN VIRTUAL

Autor: Pablo Ríos Costas

Tutor: Enrique Soto Campos

RESUMEN

La finalidad de este proyecto es realizar un sistema de control remoto por Ethernet mediante el envío y recepción de datos (comandos, datos de control y medidas) en tiempo real que permita una comunicación completa entre un cliente UDP y un servidor UDP.

El cliente UDP es un ordenador donde reside la aplicación de usuario. Se trata de una aplicación multiplataforma basada en una interfaz gráfica realizada en Python y Qt. El objetivo de dicha aplicación es controlar los instrumentos disponibles en el servidor.

Por otra parte, el servidor UDP es la placa de desarrollo, que está basada en FPGA. En ella se encuentran los instrumentos, todos ellos controlados por un procesador embebido.

El sistema, tanto en el plano de la aplicación de usuario, como en el plano de los instrumentos, está realizado con el objetivo de que sea fácilmente ampliable a otro número de instrumentos, y no se restrinja a la plataforma de instrumentos planificados en este proyecto (un frecuencímetro y un generador de funciones).

Palabras Clave: Configuración remota, Ethernet, UDP, Python, instrumentación virtual.

Índice de contenidos

CAPÍTULO 1: INTRODUCCIÓN Y OBJETIVOS.....	11
1.1.- INTRODUCCIÓN.....	11
1.2.- MEDIOS MATERIALES	12
1.3.- OBJETIVOS.....	12
1.4.- PLAN DE DESARROLLO	13
CAPÍTULO 2: PROTOCOLOS DE RED.....	15
2.1.- ETHERNET.....	15
2.1.1.- Introducción	15
2.1.2.- CSMA/CD.....	17
2.1.3.- Paquete Ethernet.....	18
2.1.4.- Auto-negociación.....	19
2.2.- IP	19
2.2.1.- Introducción	19
2.2.2.- Direccionamiento IPv4	20
2.2.3.- Encaminamiento IP.....	21
2.2.3.- Paquete IPv4.....	21
2.3.- ARP.....	24
2.3.1.- Introducción	24
2.3.2.- ARP Request/ ARP Reply.....	24
2.4.- UDP	26
2.4.1.- Introducción	26
2.4.2.- Paquete UDP.....	27
CAPÍTULO 3: SERVIDOR UDP – PLACA DE DESARROLLO	29
3.1.- PLACA DE DESARROLLO DE2.....	29
3.2.- FPGA CYCLONE II DE ALTERA	30
3.2.1.- Introducción	30
3.2.2.- Diagrama de bloques y características.....	31
3.2.3.- Configuración hardware.....	32
3.3.- MICROPROCESADOR NIOS II.....	34
3.3.1.- Introducción	34
3.3.2.- Características.....	35
3.3.3.- Excepciones	36
3.3.4.- Acceso a memoria y periféricos.....	38
3.3.5.- Sistema Operativo uC/OSII	39
3.3.6.- Programación del Nios II.....	40
3.4.- UNIDAD DE COMPARACIÓN Y CAPTURA	40
3.4.1.- Funcionamiento	40
3.4.2.- Frecuencímetro.....	41
3.4.3.- Generador de funciones.....	42
3.4.4.- Configuración.....	42
3.5.- CONTROLADOR ETHERNET DAVICOM DM9000A	44
3.6.- DISEÑO DEL HARDWARE	45
3.6.1.- Quartus II 13.0	45
3.6.2.- Qsys.....	47
3.6.3.- Integración de componentes del procesador con Qsys	50

3.6.3.1.- Reloj	50
3.6.3.2.- PLL	51
3.6.3.3.- CPU	55
3.6.3.4.- Controlador SDRAM	58
3.6.3.5.- JTAG UART	60
3.6.3.6.- Temporizadores	60
3.6.3.7.- Pulsadores	61
3.6.3.8.- Commutadores	62
3.6.3.9.- Leds Rojos	63
3.6.3.10.- Leds Verdes	64
3.6.3.11.- Elementos de control de la UCC	65
3.6.3.12.- Controlador de LCD	67
3.6.3.13.- Identificador del sistema (<i>system_id</i>)	67
3.6.3.14.- Controlador Ethernet DM9000A	67
3.6.3.15.- Finalización y generación del procesador	69
3.6.4.- <i>Creación de un esquemático</i>	72
3.6.5.- <i>Inserción de bloques al esquemático principal</i>	73
3.6.5.1.- Procesador	74
3.6.5.2.- Unidad de comparación y captura	74
3.6.5.3.- Circuito de reset	76
3.6.6.- <i>Interconexión de componentes</i>	78
3.6.7.- <i>Creación y asignación de pines</i>	79
3.6.8.- <i>Compilación y configuración del proyecto en la FPGA</i>	81
3.7.- DISEÑO DEL SOFTWARE	82
3.7.1.- <i>Crear un proyecto con Eclipse</i>	83
3.7.2.- <i>Protocolo de aplicación: VI Protocol</i>	86
3.7.2.1.- Descripción y funcionalidades	86
3.7.2.2.- Comandos del cliente	87
3.7.2.3.- Comandos del servidor	88
3.7.3.- <i>Diagrama de funcionamiento</i>	89
3.7.4.- <i>Instrumentos de medida</i>	90
3.7.4.1.- Frecuencímetro	90
3.7.4.2.- Generador de funciones	92
3.7.5.- <i>Explicación del código</i>	92
3.7.5.1.- Protocolo VI (<i>vi_protocol.h</i>)	93
3.7.5.2.- Driver DM9000A (*. <i>h</i> y *. <i>c</i>)	93
3.7.5.3.- Cabecera del programa principal (<i>frec_gen.h</i>)	94
3.7.5.4.- Programa principal (<i>frec_gen.c</i>)	96
CAPÍTULO 4: CLIENTE UDP – APLICACIÓN DE USUARIO	101
4.1.- PYTHON	101
4.1.1.- <i>Instalación de Python</i>	101
4.1.2.- <i>Python (x, y)</i>	101
4.2.- QT Y PYQT	104
4.3.- CREACIÓN DE LA INTERFAZ GRÁFICA CON <i>QT DESIGNER</i>	104
4.3.1.- <i>Crear un nuevo proyecto</i>	104
4.3.2.- <i>Inclusión de widgets</i>	106
4.4.- FUNCIONAMIENTO DE LA INTERFAZ GRÁFICA	115
4.4.1.- <i>Funcionamiento</i>	115
4.4.2.- <i>Ficheros de señal no cuadrada</i>	119
4.5.- DISEÑO DE LA APLICACIÓN DE USUARIO	121
4.5.1.- <i>Diagrama de funcionamiento</i>	121

4.5.2.- <i>Explicación del código</i>	121
4.5.2.1.- <i>window.py</i>	122
4.5.2.2.- <i>vi_protocol.py</i>	122
4.5.2.3.- <i>main.pyw</i>	123
CAPÍTULO 5: VERIFICACIÓN DE FUNCIONAMIENTO	127
5.1.- HERRAMIENTA <i>WIRESHARK</i>	127
5.2.- FUNCIONAMIENTO	128
5.2.1.- <i>Servidor</i>	128
5.2.1.1.- Frecuencímetro	128
5.2.1.2.- Generador de funciones	129
5.2.2.- <i>Cliente</i>	130
5.2.2.1.- Frecuencímetro	131
5.2.2.2.- Generador de funciones cuadradas	132
5.2.2.3.- Generador de funciones no cuadradas	135
5.3.- RESULTADOS	137
5.3.1.- <i>Frecuencímetro</i>	137
5.3.2.- <i>Generador de funciones cuadradas</i>	141
5.3.3.- <i>Generador de funciones no cuadradas</i>	142
CAPÍTULO 6: CONCLUSIONES	145
6.1.- CONCLUSIONES	145
6.2.- PROPUESTAS DE TRABAJO FUTURO	146
CAPÍTULO 7: MANUAL DE USUARIO	147
7.1.- SERVIDOR	147
7.2.- CLIENTE	147
APÉNDICES	149
APÉNDICE 1: CÓDIGO FUENTE DE LA PLACA DE DESARROLLO	149
Apéndice 1A: <i>VI Protocol (vi_protocol.h)</i>	149
Apéndice 1B: <i>Cabecera del Controlador Ethernet (DM9000A.h)</i>	150
Apéndice 1C: <i>Controlador Ethernet (DM9000A.c)</i>	152
Apéndice 1D: <i>Cabecera del programa principal (freq_gen.h)</i>	156
Apéndice 1E: <i>Programa principal (freq_gen.c)</i>	162
APÉNDICE 2: CÓDIGO FUENTE DE LA APLICACIÓN DE USUARIO	183
Apéndice 2A: <i>Interfaz gráfica (window.py)</i>	183
Apéndice 2B: <i>Protocolo VI (vi_protocol.py)</i>	189
Apéndice 2C: <i>Aplicación de usuario (main.pyw)</i>	191
BIBLIOGRAFÍA	199

Índice de Figuras

Figura 1: Conexión cliente-servidor UDP a través de la red	11
Figura 2: Red Ethernet	15
Figura 3: Ethernet en el modelo OSI.....	16
Figura 4: Conector RJ45 usado en Ethernet.....	16
Figura 5: Red IP	20
Figura 6: Encaminamiento a nivel IP	21
Figura 7: Paquete IPv4	22
Figura 8: Protocolo ARP.....	24
Figura 9: Paquete ARP	25
Figura 10: Esquema de funcionamiento del protocolo UDP	27
Figura 11: Paquete UDP	27
Figura 12: Placa DE2 de Altera	29
Figura 13: Diagrama de bloques de la FPGA Cyclone II 2C35	31
Figura 14: Configuración JTAG	33
Figura 15: Configuración AS	34
Figura 16: Arquitectura del sistema operativo MicroC/OSII.....	39
Figura 17: Esquema del controlador Ethernet Davicom DM9000A	45
Figura 18: Entorno Quartus II 13.0.....	46
Figura 19: Quartus II nuevo proyecto (1)	46
Figura 20: Quartus II nuevo proyecto (2)	47
Figura 21: Abrir Qsys en Quartus	48
Figura 22: Herramienta Qsys	48
Figura 23: Configuración de la FPGA en Qsys	50
Figura 24: Qsys - reloj.....	51
Figura 25: Qsys - PLL (1).....	52
Figura 26: Qsys - PLL (2).....	52
Figura 27: Qsys - PLL (3).....	53
Figura 28: Qsys - PLL (4)	53
Figura 29: Qsys - PLL (5)	54
Figura 30: Qsys - PLL (6)	54
Figura 31: Qsys - conexión del PLL.....	55
Figura 32: Qsys configuración de cpu (1)	56
Figura 33: Qsys configuración de cpu (2)	56
Figura 34: Qsys configuración de cpu (3)	57
Figura 35: Qsys configuración de cpu (4)	57
Figura 36: Qsys conexión de cpu	58
Figura 37: Qsys controlador SDRAM (1)	58
Figura 38: Qsys controlador SDRAM (2)	59
Figura 39: Qsys conexión del controlador SDRAM	59
Figura 40: Qsys finalización de la configuración de la CPU	60
Figura 41: Qsys conexión JTAG UART.....	60
Figura 42: Qsys temporizador	61
Figura 43: Qsys conexión de los temporizadores.....	61
Figura 44: Qsys pulsadores.....	62

Figura 45: Qsys conexión pulsadores	62
Figura 46: Qsys conmutadores	63
Figura 47: Qsys conexión de los conmutadores	63
Figura 48: Qsys LEDs rojos.....	64
Figura 49: Qsys conexión LEDs rojos.....	64
Figura 50: Qsys LEDs verdes	65
Figura 51: Qsys conexión LEDs verdes	65
Figura 52: Qsys configuración PIO.....	66
Figura 53: Qsys conexión LCD.....	67
Figura 54: Qsys conexión sistem_id.....	67
Figura 55: Incorporación de los IP Cores al directorio	68
Figura 56: vista del DM9000A en Qsys	68
Figura 57: Qsys controlador Ethernet DM9000A.....	69
Figura 58: Qsys conexión global de componentes	71
Figura 59: Qsys generation	72
Figura 60: Generación de esquemático en Quartus	72
Figura 61: Incorporación de un símbolo al esquemático.....	73
Figura 62: Símbolo del procesador en el esquemático.....	74
Figura 63: Símbolo de la unidad de captura en el esquemático	75
Figura 64: Creación de un símbolo a partir de HDL	76
Figura 65: Símbolo del circuito de reset.....	77
Figura 66: Interconexión de componentes en el esquemático	79
Figura 67: Propiedades de asignación de pines	79
Figura 68: Pin Planner.....	80
Figura 69: Importar asignaciones	81
Figura 70: Compilación del proyecto.....	81
Figura 71: Configuración de la FPGA	82
Figura 72: Selección del workspace al iniciar Eclipse	83
Figura 73: Ventana principal de Eclipse	83
Figura 74: Asociación del hardware en Eclipse.....	84
Figura 75: Vista del Project Explorer de Eclipse.....	85
Figura 76: Ejecución de tareas.....	89
Figura 77: Ruido de detección de flanco (1)	91
Figura 78: Ruido de detección de flanco (2)	91
Figura 79: Distribución de pesos ponderada de forma lineal (con 20 nodos)	92
Figura 80: Distribución de pesos exponencial (con 25 nodos)	92
Figura 81: Terminal de Python en la consola del sistema	102
Figura 82: Python (x, y)	102
Figura 83: Ventana principal de <i>Spider</i>	103
Figura 84: Ejecución del binding con PyQt4	104
Figura 85: Nuevo proyecto en Qt Designer	105
Figura 86: Ventana principal de Qt Designer	105
Figura 87: Interfaz gráfica al introducir los Dock Widgets.....	107
Figura 88: Ventana Connection	108
Figura 89: Páginas del Stacked Widget	108
Figura 90: Configuración del Table Widget.....	109
Figura 91: Configuración de QcomboBox 1.....	109

Figura 92: Ventana Virtual Instrument (1)	110
Figura 93 Configuración de QcomboBox 2	110
Figura 94: Ventana Virtual Instrument (2)	111
Figura 95: Configuración de QcomboBox 3.....	112
Figura 96: Ventana Virtual Instrument (3)	112
Figura 97: Barra de menú	113
Figura 98: Asociación de signals/slots (1)	114
Figura 99: Asociación de signas/slots (2).....	114
Figura 100: Aplicación final	115
Figura 101: Configuración con señal cuadrada	117
Figura 102: Configuración con señal no cuadrada.....	118
Figura 103: Cargar señal no cuadrada.....	119
Figura 104: Guardar tabla en formato csv.....	119
Figura 105: Fichero de señal no cuadrada.....	120
Figura 106: Hilos de la aplicación de usuario.....	121
Figura 107: Ventana principal de WireShark.....	127
Figura 108: Conexión de cables con el frecuencímetro	128
Figura 109: Estado de la placa con el frecuencímetro activado	129
Figura 110: Conexión de cables con el generador de funciones activado.....	129
Figura 111: Estado de la placa con el generador de funciones (señal cuadrada y señal no cuadrada).....	130
Figura 112: Generador de funciones externo con señal cuadrada de 10 KHz	131
Figura 113: 10 Auto Request con 10 muestras y 0.2 segundos de intervalo entre muestras ..	131
Figura 114: Tres solicitudes de frecuencia sin Auto Request	132
Figura 115: Señal cuadrada de 6 KHz y 90% de Duty Cycle	132
Figura 116: Señal cuadrada de 5 KHz y 50% de Duty Cycle	133
Figura 117: Señal cuadrada de 1 KHz y 80% de Duty Cycle	134
Figura 118: Señal cuadrada de 6 KHz y 10% de Duty Cycle	134
Figura 119: Señal senoidal de 1 KHz.....	135
Figura 120: Señal tipo Sinc de 2 KHz	136
Figura 121: Señal de tipo triangular de 3KHz	137
Figura 122: Comparativa de la desviación típica osciloscopio-FPGA	139
Figura 123: Comparativa de la desviación típica entre los tres tipos de promediado	140
Figura 124: Señal cuadrada 10 KHz	141
Figura 125: Señal cuadrada 2 MHz	141
Figura 126: Señal cuadrada 7 MHz	142
Figura 127: Señal cuadrada 10 MHz.....	142
Figura 128: Señal triangular en la aplicación	143
Figura 129: Señal triangular de 150 Hz	143
Figura 130: Señal triangular de 9 KHz	143
Figura 131: Señal triangular de 10 KHz.....	144
Figura 132: Señal triangular de 15 KHz.....	144

CAPÍTULO 1: INTRODUCCIÓN Y OBJETIVOS

1.1.- Introducción

Este proyecto trata sobre la realización de un sistema de gestión remota sobre Ethernet de tipo cliente-servidor UDP, que permita interactuar con una plataforma de instrumentos implementados sobre una FPGA (*Field Programable Gate Array*).

Las FPGAs son dispositivos semiconductores cuya funcionalidad no está predeterminada de antemano, sino que es el diseñador del sistema el que, mediante un lenguaje de descripción hardware (VHDL, Verilog...), programa la funcionalidad deseada en el dispositivo. Además, dicha funcionalidad o configuración puede ser modificada, es decir, se puede reprogramar, lo que le otorga una gran ventaja frente a, por ejemplo, los sistemas de tipo ASIC. Esta capacidad de reprogramación es muy útil en el prototipado de sistemas, puesto que permite tener el sistema implementado físicamente y completamente funcional (no solo simulado) con un coste mucho más bajo que su equivalente ASIC. Tanto las FPGAs como los ASICs se utilizan en aplicaciones similares, sin embargo las FPGAs son más lentas, tienen un mayor consumo, y están más limitadas en cuanto a la complejidad del sistema a diseñar. Por ello, generalmente se suele diseñar el prototipo mediante una FPGA, y posteriormente, una vez verificado el sistema, se fabrica el producto final en tecnología de tipo ASIC.

La siguiente figura muestra una visión general y simplificada del presente proyecto.

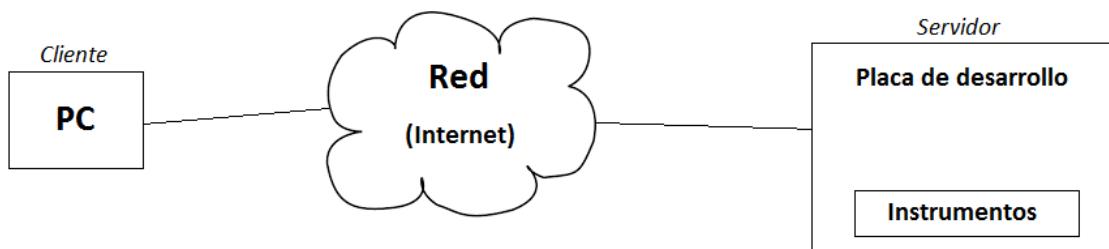


FIGURA 1: CONEXIÓN CLIENTE-SERVIDOR UDP A TRAVÉS DE LA RED

El cliente es un ordenador de aplicación general (PC), que a través de una interfaz gráfica nos permite interactuar con los instrumentos que se encuentran en el

servidor. Podemos tanto configurar los instrumentos, como obtener datos (medidas) de los mismos. Gracias a la utilización de Python y Qt, la aplicación será multiplataforma, lo que hará posible su ejecución en cualquier PC, independientemente del sistema operativo utilizado.

Por otro lado, el servidor es la placa de desarrollo basada en FPGA. En dicha placa se implementan los instrumentos, que en nuestro caso son dos: un frecuencímetro y un generador de funciones. Ambos instrumentos están conectados a un procesador embebido, el Nios II de Altera, que permitirá el control interno de dichos instrumentos. El Nios II es un procesador configurable (se puede adaptar a las necesidades del sistema) de arquitectura de 32 bits que está diseñado de forma específica para la familia de FPGAs de Altera¹. Como sistema operativo de dicho procesador se utiliza el MicroC OSII.

1.2.- Medios materiales

Para la realización de este proyecto contaremos con los siguientes medios materiales:

- Placa de desarrollo Altera DE2 (basada en la FPGA Cyclone II)
- Osciloscopio digital Tektronix TekScope THS710
- Generador de funciones Philips PM5192
- Cable Ethernet categoría 5E
- Dispositivos discretos (resistencias y condensadores) y cables de conexión
- PC con Windows 7 64 bits
- Documentación especificada en la bibliografía

1.3.- Objetivos

El objetivo del proyecto es realizar una comunicación efectiva en tiempo real entre dos entidades, un ordenador y una placa de desarrollo haciendo uso de una conexión Ethernet. Dicha comunicación permite enviar y recibir datos encapsulados en protocolos de uso general en Internet - como son tanto el protocolo IP y el protocolo UDP - con el objetivo de configurar remotamente instrumentos de medida implementados en una FPGA.

¹ Xilinx, el principal competidor de Altera, también dispone de un procesador de similares características adaptado para sus FPGAs denominado MicroBlaze.

1.4.- Plan de desarrollo

El proyecto se dividirá en dos partes claramente diferenciadas: el cliente (PC) y el servidor (placa de desarrollo DE2). La propuesta del autor es comenzar desarrollando primero la parte del servidor, y posteriormente implementar la aplicación del cliente. Por ello, se procederá mediante el siguiente plan de desarrollo:

- Estudio del software Quartus II 13.0 sp1 y de la herramienta Qsys, ambos proporcionados por Altera.
- Estudio de los protocolos Ethernet, IP y UDP.
- Estudio del controlador Ethernet DM9000A disponible en la placa de desarrollo.
- Implementación del microprocesador Nios II y asociación de los periféricos necesarios (memorias, pulsadores, controlador Ethernet, etc) en la FPGA Cyclone II mediante la herramienta Qsys.
- Integración de la unidad de comparación y captura y el procesador Nios II, y asignación de pines de entrada y salida de la FPGA.
- Estudio del sistema operativo uC OSII en el procesador Nios II.
- Desarrollo del código (en C y ensamblador) correspondiente para la realización de un frecuencímetro, un generador de funciones, y el control del envío y recepción de datos por medio del controlador Ethernet.
- Familiarización con Python y Qt. Utilizaremos un entorno de herramientas llamado Python(x, y) que integran múltiples herramientas y librerías para el desarrollo de programas en Python. Concretamente nos familiarizaremos con el editor *Spider*, que permite desarrollar el código Python, y con *Qt Designer*, que simplifica el diseño de interfaces gráficas.
- Desarrollo de la interfaz gráfica y el código Python asociado para la realización de la aplicación de usuario.
- Verificación del sistema completo.

- Pruebas, resultados y conclusiones obtenidas.

CAPÍTULO 2: PROTOCOLOS DE RED

En este capítulo describiremos los protocolos de red implementados en el proyecto, que son: Ethernet, IP, UDP y ARP.

2.1.- Ethernet

2.1.1.- Introducción

El protocolo Ethernet es un protocolo de red de área local (LAN, *Local Area Network*) que fue diseñado por *Xerox Corporation, DEC e Intel* en 1976. Ethernet utiliza una topología en bus o estrella, soportando altas tasas de transmisión². Actualmente, la topología más utilizada es la de tipo estrella, puesto que proporciona un mayor rendimiento. A continuación se muestra una figura genérica de una red local actual compuesta por cuatro estaciones (*hosts*) y un *switch*.

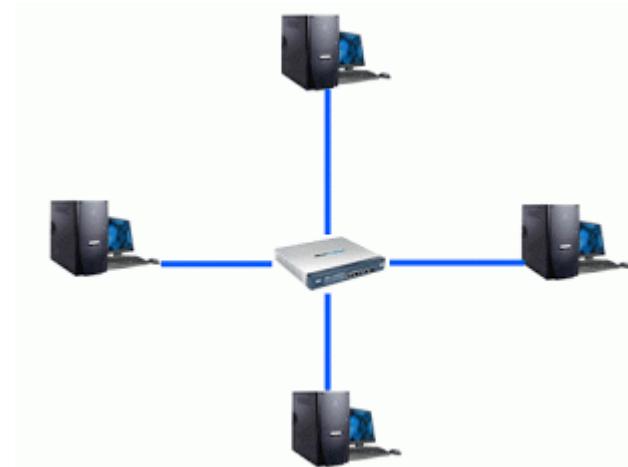


FIGURA 2: RED ETHERNET

² Lo habitual actualmente es encontrar controladores Ethernet que permiten tasas de transmisión de 10/100/1000/10000 Mbps (*Mega bits per second*), y ya se están probando conexiones a 1 Tbps (*Terabit Ethernet*).

La especificación Ethernet sirvió como base para la creación del estándar IEEE 802.3, el cual especifica la capa física y la capa software más baja.

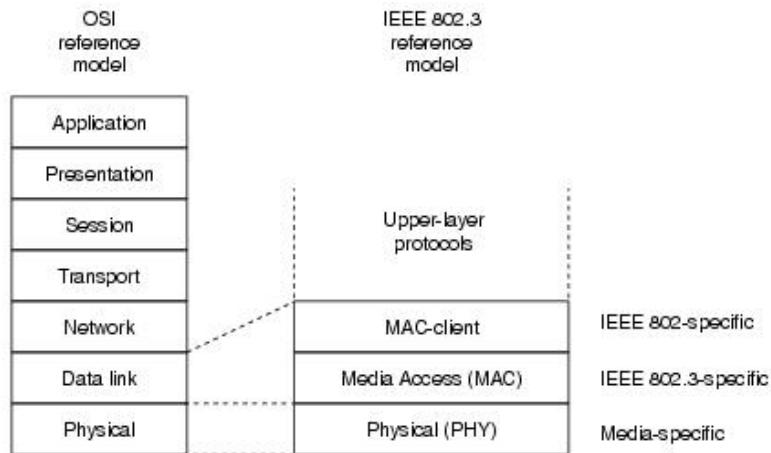


FIGURA 3: ETHERNET EN EL MODELO OSI

IEEE 802.3 es el grupo de trabajo y el conjunto de estándares IEEE producidos por el grupo de trabajo que define la capa física y el control de acceso al medio (MAC, *Media Access Control*) de Ethernet.

Para realizar la transmisión, en las versiones de 10 Mbps y 100 Mbps ("Fast Ethernet") se utiliza un cable UTP (*Unshielded Twisted Pair*) de categoría 5, o un cable de fibra óptica. Como conector se utiliza un RJ45 como el mostrado en la siguiente figura.



FIGURA 4: CONECTOR RJ45 USADO EN ETHERNET

2.1.2.- CSMA/CD

Ethernet utiliza CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) como método de acceso al medio con el objetivo de poder gestionar varias peticiones de transmisión simultáneas. CSMA/CD es uno de los métodos de control de acceso al medio más utilizados, principalmente en redes de área local. Usa un esquema de identificación de portadora, de forma que ante una colisión³, se ejecuta un algoritmo que tras su finalización, permite que la transmisión se realice de forma fiable.

El procedimiento de transmisión es el siguiente:

- 1) ¿Está la trama de datos lista para ser transmitida? En caso afirmativo se salta al siguiente punto.
- 2) ¿Está el medio de transmisión preparado (estado *idle*)? Si no los estamos esperando hasta que lo esté.
- 3) Empezamos a transmitir.
- 4) ¿Ha ocurrido una colisión? En caso afirmativo se procede a solucionarla mediante el “procedimiento de colisión detectada”.
- 5) Se reinician los contadores de retransmisión y la transmisión de fin de trama.

El “procedimiento de colisión detectada” realiza las siguientes funciones:

- 1) Se continúa transmitiendo una señal específica (“*jam signal*”) en lugar de la trama de datos (cabecera, datos, redundancia cíclica) hasta que se alcanza el mínimo tiempo de transmisión de trama para estar seguro de que todos los receptores de la red han detectado la colisión.
- 2) Se incrementa el contador de retransmisión.
- 3) Si se ha alcanzado el máximo número de retransmisiones especificado a priori, se aborta la transmisión.
- 4) Se calcula el tiempo de espera de retransmisión, basado en una estrategia de tipo *backoff*, en función del número de colisiones.
- 5) Se vuelve a intentar el procedimiento comenzando en el punto 1.

³ Colisión se entiende como el intento de transmisión simultánea de dos o más estaciones (nodos) transmisores.

2.1.3.- Paquete Ethernet

El paquete Ethernet está precedido por un preámbulo y un delimitador de comienzo de trama (SFD, *Start-of-Frame Delimeter*). Consta de tres campos principales:

- Cabecera. En ella se incorporan aquellos campos destinados al encaminamiento del paquete por la red, como es el caso de las direcciones MAC⁴.
- El campo de datos. En este campo se introducen los datos que se desean transmitir, incluyendo cabeceras de otros protocolos de capas superiores (como por ejemplo el protocolo IP) encapsuladas en dicho campo.
- Redundancia (FCS, *Frame Check Sequence*). Se trata de un campo de 4 bytes de redundancia cíclica usado para detectar la posible modificación de datos durante la transmisión del paquete por el canal de comunicación.

Los datos transmitidos tienen la siguiente estructura:

Preamble	SFD	Destination MAC Address	Source MAC Address	Ethertype/ Length	Data	FCS
7 bytes	1 byte	6 bytes	6 bytes	2 bytes	46 – 1500 bytes	4 bytes

- *Preamble* y SFD: indican el comienzo de un nuevo paquete Ethernet.
- *Destination MAC Address*: dirección MAC del receptor del paquete. Puede ser *unicast* o *broadcast*.
- *Source MAC Address*: dirección MAC del origen del paquete (de tipo *unicast*).
- *Ethertype/length*: indica el tipo de datos (protocolo encapsulado dentro del paquete Ethernet). Cada protocolo que se puede encapsular tiene un identificador único. Los más relevantes para este proyecto son dos: el protocolo IPv4 (IP versión 4) y el protocolo ARP.

⁴ La dirección MAC es un identificador único de 8 octetos asignado a la interfaz de red de cada dispositivo. Por ejemplo, a0:80:17:c7:14:e6 es una dirección MAC *unicast*. La dirección *broadcast* es FF:FF:FF:FF:FF:FF, que permite transmitir un paquete Ethernet a todos los hosts vecinos del transmisor.

Protocolo	Ethertype/length
IPv4	0x0800
ARP	0x0806

- *Data*: datos a transmitir. Este campo es variable, con un mínimo de 46 bytes y un máximo de 1500.
- *FCS*: cuatro octetos de redundancia cíclica que permiten identificar errores de transmisión por el canal.

2.1.4.- Auto-negociación

La auto-negociación es el procedimiento por el cual dos dispositivos conectados entre sí eligen unos parámetros de transmisión comunes, por ejemplo, la velocidad de transmisión (10 Mbps, 100 Mbps, etc) y el modo de transmisión (simplex, half dúplex, full dúplex...). Esto garantiza la compatibilidad hacia atrás de los dispositivos de red, ya que permite que, tanto la velocidad como el modo de transmisión se adecúen al dispositivo más lento.

2.2.- IP

2.2.1.- Introducción

El protocolo IP (*Internet Protocol*) es el principal protocolo de la capa de Internet, que especifica el formato de los paquetes y el esquema de direccionamiento en una red de conmutación de paquetes. Tiene dos funciones: identificar los hosts, y proporcionar una localización lógica de los mismos.

IP tiene como cometido principal la tarea de transportar paquetes desde un host origen a un host destino, valiéndose para ello de las direcciones IP que se encuentran en la cabecera de dichos paquetes.

Actualmente conviven dos versiones, la primera versión y más dominante denominada versión 4 (IPv4) y su sucesora, la versión 6 (IPv6). A lo largo del presente proyecto utilizaremos el protocolo IP para referirnos exclusivamente a la versión IPv4, que es la que utilizaremos en la implementación posterior.

2.2.2.- Direccionamiento IPv4

IPv4 utiliza direcciones de 4 octetos, que son representadas generalmente en una notación decimal con puntos, por ejemplo 172.19.5.91. La dirección 255.255.255.255 es la dirección IP *broadcast*.

Una subred es el conjunto lógico de una red IP. Los dispositivos conectados dentro de una misma subred tienen asociadas unas direcciones IP muy similares. Concretamente se pueden diferenciar dos campos característicos: un prefijo de subred (obtenido realizando AND lógico bit a bit entre la IP del host y la máscara de subred), y el resto, que es el identificador de host dentro de la susodicha subred.

La subred está caracterizada por la máscara de subred. Dicha máscara es también una dirección de 4 octetos, y su propósito es identificar el tamaño de la subred.

Se distinguen principalmente dos tipos de direcciones IP:

- IP pública: pueden ser accedidas desde cualquier subred.
- IP privada: solo pueden ser accedidas desde la misma subred. Además, un conjunto de subredes privadas pueden estar asociadas a una IP pública a través de un router NAT.

Las IPs privadas son del estilo: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16 (el símbolo '/x' indica el prefijo, en bits, de la máscara de subred. Por ejemplo, 10.0.0.0/8 indica que la máscara de subred es 255.0.0.0).

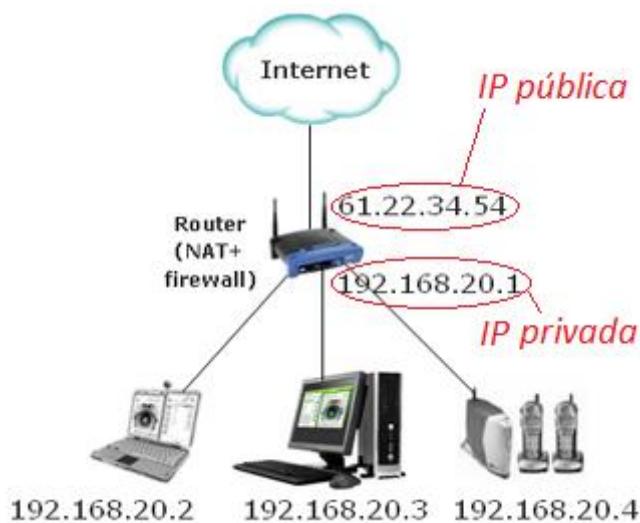


FIGURA 5: RED IP

2.2.3.- Encaminamiento IP

El encaminamiento de paquetes IP, denominado en inglés *IP forwarding* o *IP routing* es el procedimiento que se lleva a cabo para determinar el enlace de salida en el que se debe situar un paquete para que pueda llegar al destino solicitado por la fuente.

Generalmente las redes están separadas unas de otras a través de *routers*, de forma que los paquetes son encaminados entre los *routers* hasta llegar al destino especificado en la dirección destino del paquete IP.

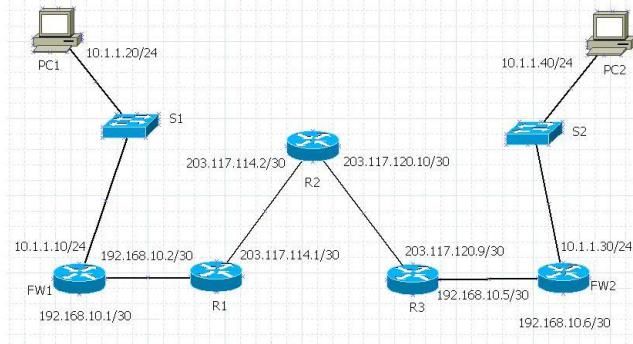


FIGURA 6: ENCAMINAMIENTO A NIVEL IP

Es importante tener en cuenta que cada *router* solamente tiene que conocer el “siguiente salto” del paquete en la red, es decir, el enlace de salida en función de la dirección destino.

2.2.3.- Paquete IPv4

Cada paquete consta de dos partes: la cabecera y el campo de datos (este último tiene un longitud variable). En la siguiente figura se muestra la distribución de los campos de la cabecera.

Offsets Octet		0				1				2				3																													
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31										
0	0	Version	IHL	DSCP			ECN			Total Length																																	
4	32	Identification										Flags			Fragment Offset																												
8	64	Time To Live				Protocol				Header Checksum																																	
12	96	Source IP Address																																									
16	128	Destination IP Address																																									
20	160	Options (if IHL > 5)																																									

FIGURA 7: PAQUETE IPv4

- *Version*: es el primer campo de la cabecera, que tiene como objetivo identificar la versión del protocolo IP utilizada. Para IPv4 tendrá el valor 4.
- *IHL (Internet Header Length)*: indica el número de palabras de 32 bits que tiene la cabecera. Esto es debido a que la longitud de la cabecera puede ser variable (campo “*Options*”). El mínimo valor es 5, que está asociado a una longitud de cabecera de $32 \times 5 = 160$ bits = 20 bytes.
- *DSCP (Differentiated Services Code Point)*: denominado previamente como *ToS (Type of Service)*. Se trata de un campo orientado a proporcionar servicios diferenciados, principalmente relacionados con servicios de comunicación en tiempo real, como es el caso de VoIP (*Voice over IP*). Es un mecanismo opcional, de forma que si no se quiere usar simplemente se introduce un cero en dicho campo.
- *ECN (Explicit Congestion Notification)*: originalmente formaba parte del campo *ToS*. Se utiliza para proporcionar una notificación de congestión de la red sin necesidad de eliminar paquetes. Al igual que DSCP también se trata de un mecanismo opcional.
- *Total Length*: indica, en número de bytes, la longitud total del paquete IP, incluyendo la cabecera y los datos. Algunas subredes imponen restricciones al tamaño de los paquetes, en cuyo caso son fragmentados.
- *ID (Identification)*: usado para identificar al grupo de fragmentos pertenecientes al mismo datagrama IP.
- *Flags*: campo de tres bits que se utiliza para llevar a cabo el control y/o identificación de fragmentos. En función del bit activado se indica: bit0

(reservado), bit1 (no fragmentar el paquete), bit2 (quedan más fragmentos del paquete).

- *Fragment Offset*: indica, en bytes, el offset del paquete dentro del datagrama IPv4 original.
- *TTL (Time To Live)*: indica el máximo número de saltos que el datagrama puede dar por la red. Cada *router* de la red decremente este campo en una unidad, de forma que si un *router* se encuentra con el campo *TTL* de un paquete a cero, el paquete es automáticamente descartado y se elimina de la red. Esto permite evitar la explosión de paquetes en la red, lo que podría provocar una situación de congestión (provocado principalmente por aquellos paquetes con dirección IP destino *broadcast*).
- *Protocol*: indica el protocolo de transporte que se está encapsulado en el campo de datos del paquete IP. Los más utilizados son los siguientes:

Número del protocolo	Nombre del protocolo
1	ICMP
2	IGMP
6	TCP
17	UDP
41	ENCAP
89	OSPF
132	SCTP

- *Header Checksum*: redundancia que controla los posibles errores de transmisión en la cabecera (no se incluye el campo de datos). Como cada vez que llega un paquete al *router* estos decrementan el valor de TTL, el *router* deberá calcular el nuevo checksum asociado.
- *Source Address*: Identifica al emisor del paquete. Cabe decir que dicha dirección puede ser modificada por ejemplo si el host está detrás de un *router* NAT⁵.
- *Destination Address*: identifica al receptor del paquete. Del mismo modo que la dirección origen, ésta también puede ser modificada.

⁵ Un router NAT (Network Address Translation) permite asociar una subred privada (conjunto de IPs privadas) a una única dirección IP pública teniendo en cuenta los puertos de cada conexión.

- *Options*: campos adicionales que se pueden incorporar a la cabecera si el valor del campo IHL es mayor a cinco.

2.3.- ARP

2.3.1.- Introducción

El protocolo ARP (*Address Resolution Protocol*) se utiliza para, dentro de una misma subred, asociar una dirección IP con una dirección física (dirección MAC). Hay que tener en cuenta que las direcciones IP pueden cambiar, no así las direcciones MAC, que son únicas.

El protocolo se encapsula dentro de los paquetes Ethernet indicando en el campo *Ethertype* el código 0x0806 en hexadecimal.

2.3.2.- ARP Request/ ARP Reply

El funcionamiento del protocolo es muy simple ya que se trata de un protocolo de dos mensajes: *Request* y *Reply* (o *Response*). Cuando un host o *router* no conoce la dirección MAC del host destino requerido, simplemente se envía un paquete *ARP Request* de tipo broadcast indicando la IP de la cual se quiere conocer su MAC asociada. Gracias a esto, la solicitud ARP Request llegará a todas las estaciones de la subred. El host cuya dirección MAC sea la requerida enviará un paquete de respuesta ARP Reply (de tipo *unicast*) indicando su MAC a la fuente de la solicitud. De esta forma el host o router original ya sabrá la MAC destino buscada y ya podrá enviar paquetes a ese otro host de la subred.

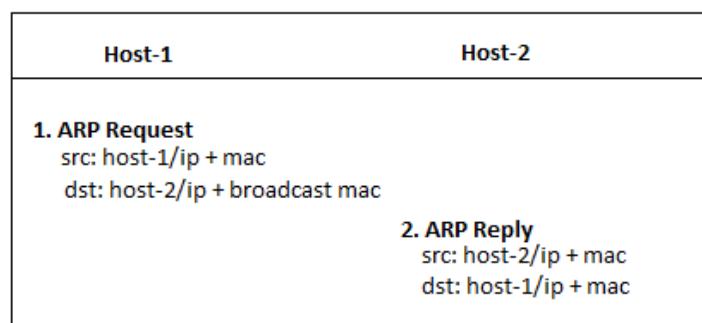


FIGURA 8: PROTOCOLO ARP

La estructura de los paquetes ARP *Request* y ARP *Reply* tiene exactamente la misma longitud y los mismos campos. Se describe a continuación:

Internet Protocol (IPv4) over Ethernet ARP packet		
octet offset	0	1
0	Hardware type (HTYPE)	
2	Protocol type (PTYPE)	
4	Hardware address length (HLEN)	Protocol address length (PLEN)
6	Operation (OPER)	
8	Sender hardware address (SHA) (first 2 bytes)	
10	(next 2 bytes)	
12	(last 2 bytes)	
14	Sender protocol address (SPA) (first 2 bytes)	
16	(last 2 bytes)	
18	Target hardware address (THA) (first 2 bytes)	
20	(next 2 bytes)	
22	(last 2 bytes)	
24	Target protocol address (TPA) (first 2 bytes)	
26	(last 2 bytes)	

FIGURA 9: PAQUETE ARP

- *Hardware type*: identifica el protocolo de red superior utilizado. En el caso de utilizar Ethernet este campo tendrá el valor 0x0001.
- *Protocol type*: especifica el protocolo de internet para el cual se solicita el ARP Request. Si se utiliza IPv4 este campo tiene el valor 0x0800.
- *Hardware length*: longitud en bytes de la dirección hardware. Si se usa Ethernet la longitud de las direcciones MAC es de 6 bytes.
- *Protocol length*: longitud en bytes de las direcciones utilizadas del protocolo especificado en *Protocol type*. En el caso de IPv4 la longitud es de 4 bytes.
- *Operation/Opcode*: indica si el paquete es Request (1) o Reply (2).
- *Sender hardware address*: si el paquete es una petición ARP *Request* este campo indica la dirección MAC del host que ha enviado la petición. Por el

contrario, en un paquete ARP *Reply* el campo indica la dirección MAC del host solicitado con ARP *Request*.

- *Sender protocol address*: dirección de red (IPv4) del transmisor.
- *Target hardware address*: dirección MAC del dispositivo solicitado. Cuando se envía ARP *Request* este campo se envía a cero. En cambio, cuando un host envía ARP *Reply* introduce la MAC del host que ha enviado el ARP *Request* previo.
- *Target protocol address*: dirección de red del receptor esperado.

2.4.- UDP

2.4.1.- Introducción

El protocolo UDP (*User Datagram Protocol*) es un protocolo de comunicaciones (perteneciente al nivel de transporte del modelo OSI) usado de forma encapsulada en el campo de datos de los paquetes IP. Se utiliza en comunicaciones en tiempo real y en aquellas situaciones donde se intercambian poca cantidad de datos.

UDP introduce dos servicios no disponibles en la capa IP. Proporciona la posibilidad (opcional) de incluir redundancia para el control de errores, y también los números de puerto UDP para poder distinguir paquetes (también denominados datagramas) de distintas aplicaciones que se están ejecutando en la misma máquina.

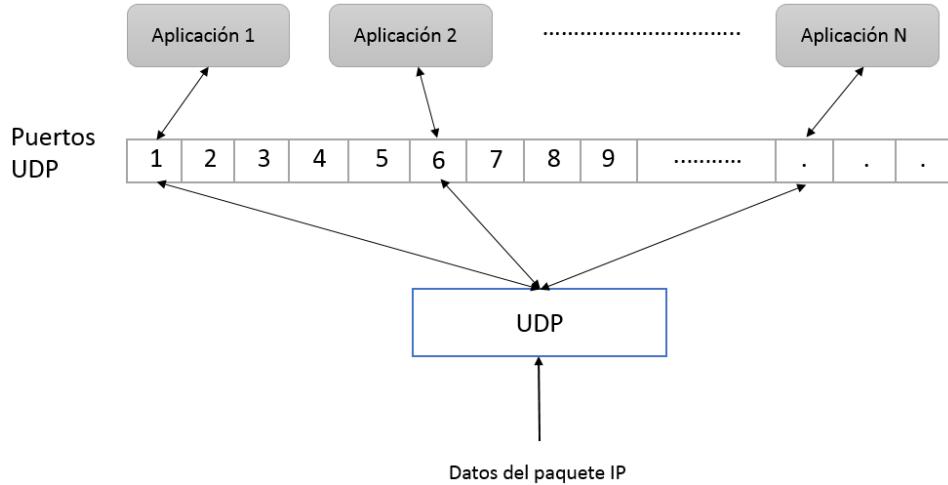


FIGURA 10: ESQUEMA DE FUNCIONAMIENTO DEL PROTOCOLO UDP

2.4.2.- Paquete UDP

La estructura del paquete UDP es muy sencilla como se puede ver a continuación, pues consta únicamente de cuatro campos: puertos fuente y destino, longitud y redundancia. La longitud total de la cabecera es de 8 octetos (4 campos de 16 bits cada uno).

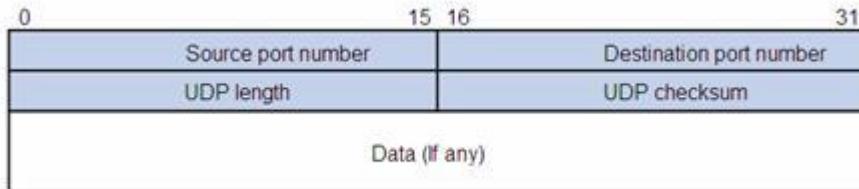


FIGURA 11: PAQUETE UDP

- *Source port number*: campo que identifica al puerto emisor. Es el puerto que utiliza la aplicación asociada para enviar y recibir información.
- *Destination port number*: puerto UDP del receptor. Se trata del puerto que utiliza la aplicación receptora para recibir/enviar paquetes. Este puerto debe ser conocido por la entidad que inicia la comunicación. Por ejemplo,

los servidores HTTP utilizan generalmente el puerto 80 (o 8080), TFTP usa el puerto 69, etc.

- *Length*: longitud en bytes del datagrama UDP. Incluye la cabecera y los datos.
- *Checksum*: campo de redundancia opcional que permite identificar errores de transmisión. Para calcular el *checksum* se incluye una pseudocabecera IP que consta de varios de los campos de la cabecera IP. Es un servicio opcional, de forma que si no se quiere utilizar tendrá el valor 0x0000.

CAPÍTULO 3: SERVIDOR UDP – PLACA DE DESARROLLO

3.1.- Placa de desarrollo DE2

La placa de desarrollo DE2 de Altera es un dispositivo ideal para aprender sobre lógica digital, FPGAs y organización computacional, permitiendo desde el desarrollo de sistemas lógicos básicos hasta complejos SOC (*System On Chip*).

Integra una FPGA Cyclone II 2C35 con un encapsulado de 672 pines. Todos los componentes y periféricos de la placa están directamente conectados con la FPGA por medio de sus pines, lo que nos garantiza el acceso y control de todos y cada uno de los componentes de la placa.

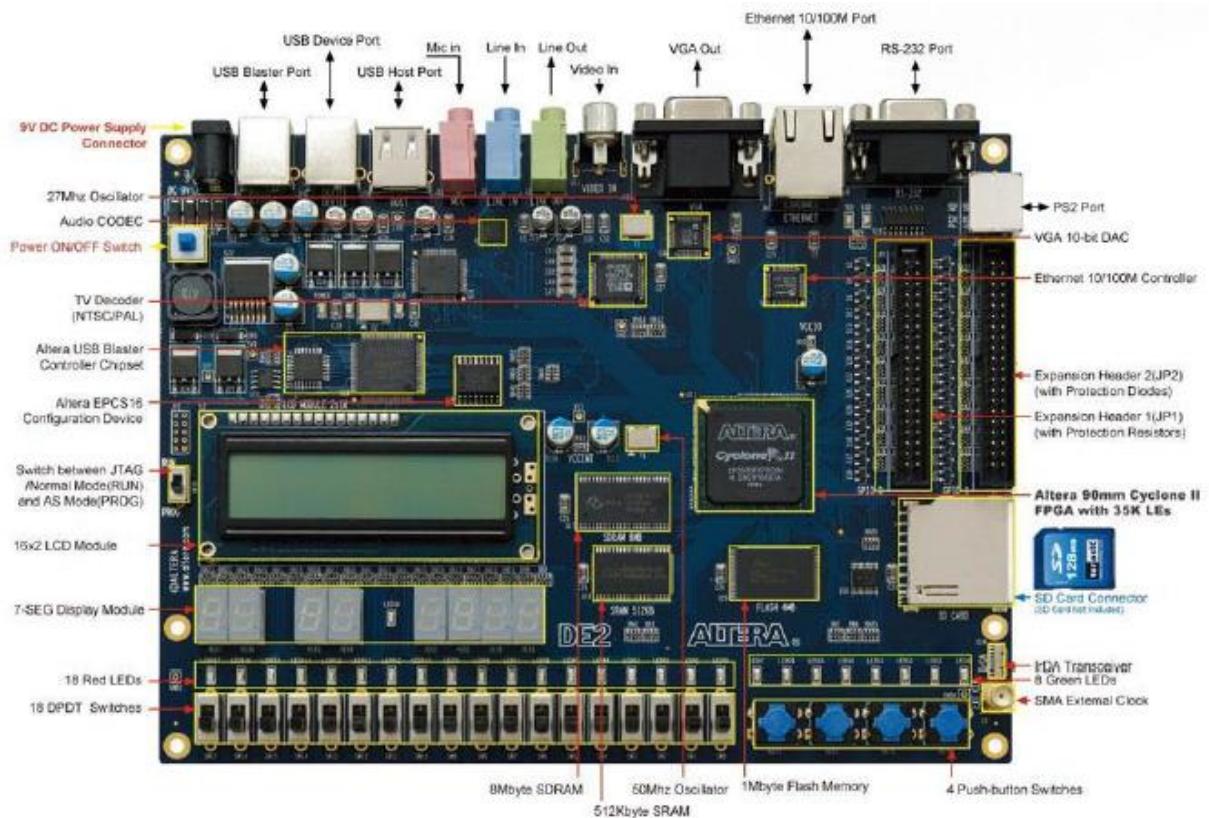


FIGURA 12: PLACA DE2 DE ALTERA

La placa proporciona los siguientes componentes:

- FPGA Altera Cyclone II 2C35
- Dispositivo de configuración serie EPICS16
- USB Blaster para programación. Disponibles los modos de programación: JTAG y Active Serial (AS).
- SRAM de 512-Kbyte
- SDRAM de 8-Mbyte
- Memoria Flash de 4-Mbyte
- Slot para tarjetas SD
- 4 pulsadores con antirrebotes incorporado
- 18 conmutadores
- 18 LEDs rojos
- 9 LEDs verdes
- 8 visualizadores de 7 segmentos
- Visualizador LCD de 16x2
- Osciladores de 50-MHz y 27-MHz (del decodificador de televisión) para funcionar como reloj
- Entrada de reloj externa
- CODEC de audio de 24-bits con calidad CD con líneas de entrada/salida y entrada de micrófono
- Salida de vídeo VGA con DAC (de 10-bits)
- Decodificador de televisión (NTSC/PAL) con su correspondiente conector de entrada
- Controlador y conector Ethernet (10/100 Mbps)
- Controlador USB maestro (*host*) y esclavo (*slave*) con conectores USB de tipo A y B
- Transceptor RS-232 con su correspondiente pin DB9 (9-pines)
- Conector de ratón y teclado PS/2
- Transmisor y receptor IrDA de infrarrojo
- Dos buses de expansión (GPIO-0 y GPIO-1) de 40 pines cada uno con protección basada en diodo

3.2.- FPGA Cyclone II de Altera

3.2.1.- Introducción

La FPGA Cyclone II 2C35 disponible en la placa DE2 permite la utilización de elementos de propiedad intelectual (IP cores) y kits de desarrollo para facilitar el desarrollo rápido de soluciones. Además, soporta el procesador embebido Nios II,

que permite la implementación de sistemas embebidos a medida, tanto simples, como complejos (por ejemplo varios procesadores Nios II funcionando en paralelo). El uso de Cyclone II junto con el Nios II permiten la obtención de sistemas de procesado embebido de alto rendimiento y a un bajo coste.

Esta FPGA está indicada para un amplio espectro de aplicaciones, como pueden ser la automoción, comunicaciones, electrónica de consumo, procesado de vídeo, test y medida.

3.2.2.- Diagrama de bloques y características

Con el objetivo de garantizar la máxima flexibilidad al usuario, todas las conexiones de la placa han sido realizadas a través de la FPGA, de forma que el usuario pueda configurarla para implementar cualquier sistema.

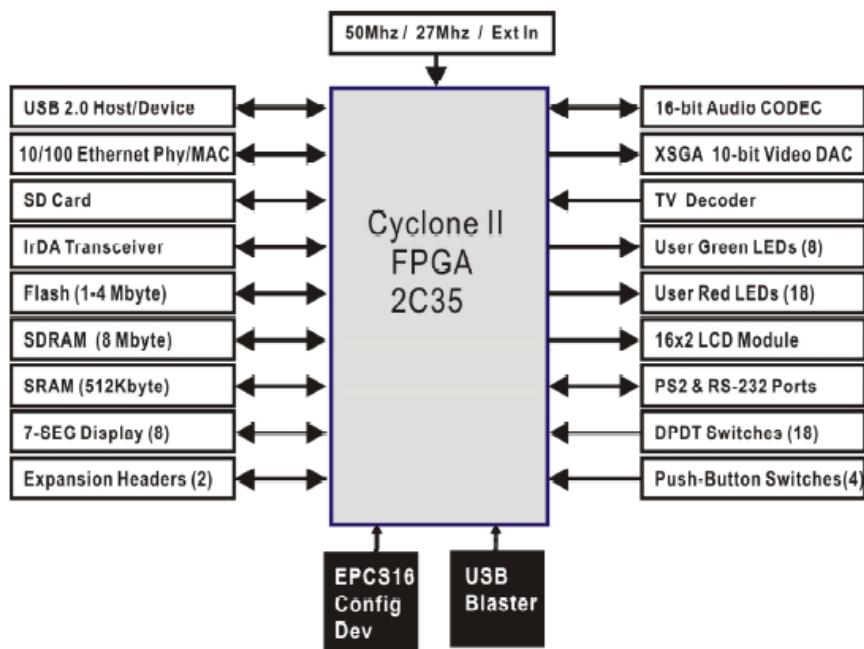


FIGURA 13: DIAGRAMA DE BLOQUES DE LA FPGA CYCLONE II 2C35

Las características principales de la FPGA Cyclone II son las siguientes:

- 33216 elementos lógicos
- 105 M4K bloques de memoria RAM
- 483840 bits de RAM

- 35 multiplicadores embebidos
- 4 PLLs
- 475 pines de entrada/salida disponibles para el usuario
- Encapsulado de tipo BGA (*Ball-Grid Array*) de 672 pines

3.2.3.- Configuración hardware

La placa DE2 contiene una memoria E²PROM denominada EPICS16, que se encarga de almacenar la configuración de la FPGA Cyclone II. Durante el proceso de encendido la configuración se carga de forma automática en la FPGA. Además, utilizando el software Quartus II se puede reprogramar directamente la FPGA en cualquier momento, siendo también posible cambiar el contenido que se almacena en el chip EPICS16. Para realizar la programación de la placa el fabricante nos proporciona un cable denominado USB Blaster, que nos permitirá enviar la información de configuración desde el ordenador a la placa. Dicho cable USB no es más que un cable USB con un extremo de tipo A y el otro de tipo B.

Se dispone de dos métodos de programación: *JTAG* y *Active Serial*.

- *JTAG*: este método de programación consiste en el envío de los datos de configuración de forma directa a la FPGA. La configuración se mantendrá en la FPGA mientras la placa mantenga la alimentación, y se perderá por tanto, cuando se desconecte de la misma. Para utilizar este método de programación se deben seguir los siguientes pasos:
 - a) Conectar la alimentación y encender la placa (botón rojo)
 - b) Conectar el cable USB Blaster al conector USB de tipo B denominado "blaster" en la placa, y el otro extremo a uno de los puertos USB del ordenador
 - c) Seleccionar la opción RUN en el conmutador de la parte izquierda de la placa para establecer el método de programación JTAG
 - d) Programar la FPGA usando el módulo Quartus II *Programmer*, por medio de un fichero de configuración con extensión *.sof

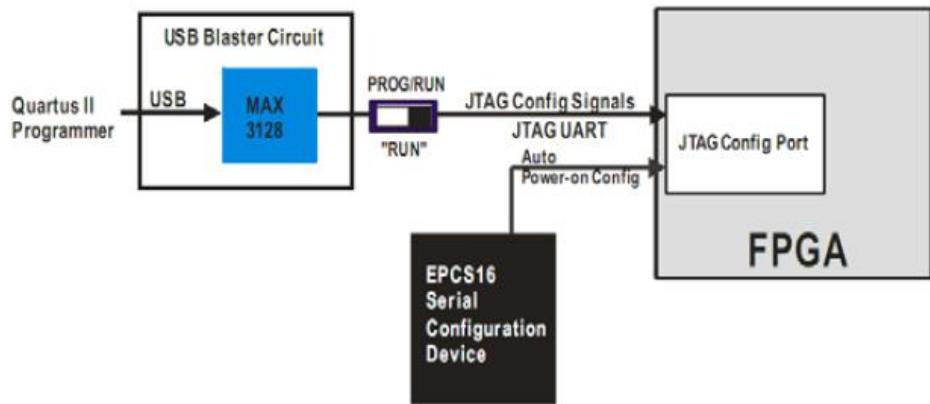


FIGURA 14: CONFIGURACIÓN JTAG

- AS: en este método, denominado *Active Serial*, los datos de configuración se envían al chip de memoria EPCS16. Al tratarse de una memoria de tipo E²PROM se garantiza el almacenamiento no volátil de los datos. Cuando se enciende la placa, la configuración del chip EPCS16 se carga automáticamente en la FPGA Cyclone II. El contenido del chip EPCS16 se puede modificar siguiendo el siguiente procedimiento:
 - a) Conectar la alimentación y encender la placa (botón rojo)
 - b) Conectar el cable USB Blaster al conector USB de tipo B denominado “blaster” en la placa, y el otro extremo a uno de los puertos USB del ordenador
 - c) Seleccionar la opción PROG en el conmutador de la parte izquierda de la placa para establecer el método de programación AS
 - d) Programar el chip EPCS16 utilizando el módulo *Programmer* de Quartus II, por medio de un fichero de configuración con extensión *.pof
 - e) Una vez finalizada la programación seleccionar la opción RUN del conmutador y realizar un reset de la placa (apagar y volver a encender). Esta acción provocará que la nueva configuración presente en el chip EPCS16 se cargue en la FPGA.

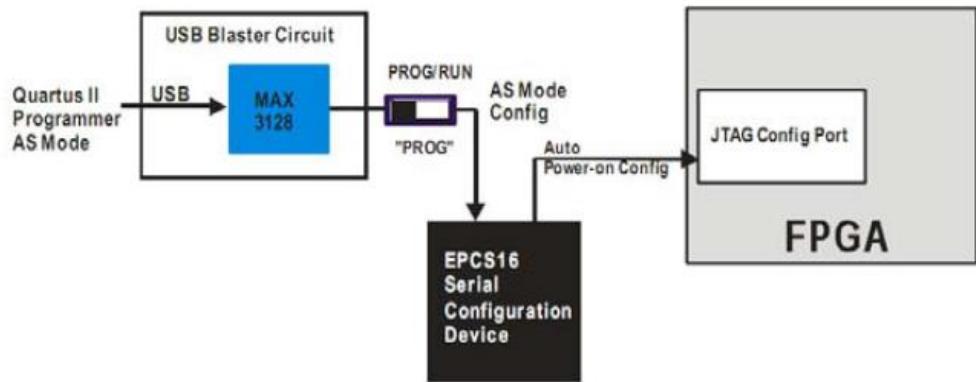


FIGURA 15: CONFIGURACIÓN AS

3.3.- Microprocesador NIOS II

3.3.1.- Introducción

En este apartado describiremos las principales funcionalidades que nos ofrece el procesador embebido Nios II, que es el que usamos en este proyecto.

El fabricante Altera proporciona un conjunto de herramientas para diseñar microprocesadores embebidos a medida según las necesidades y requisitos del diseñador, mediante la configuración de diversos componentes configurables sobre sus FPGAs. Para ello, nos proporciona una herramienta de desarrollo específico, Qsys, que nos facilita la configuración del microprocesador Nios II, y que gracias al entorno Quartus II, puede ser implementado de forma directa sobre una de las FPGAs del fabricante.

El microprocesador es muy flexible. Se trata de un núcleo configurable que nos permite adecuar sus características a las necesidades específicas de cada sistema, incorporando, o no, ciertas funcionalidades. Además el diseñador puede decidir si una característica debe ser implementada en hardware, o por el contrario deba ser realizada en software, permitiendo ajustar los costes y reducir los tiempos de desarrollo.

3.3.2.- Características

El Nios II es un microprocesador con una arquitectura de 32 bits de tipo Harvard. En este tipo de arquitectura se disponen de buses de datos e instrucciones independientes, lo contrario que en la arquitectura Von Newman que solo dispone de un bus compartido. El procesador tiene las siguientes características principales:

- Palabras de 32 bits
- Direccionamiento de 32 bits
- 32 registros de propósito general de 32 bits (R0 a R31)
- 6 registros de control de 32 bits (CTL0 a CTL5)
- 32 fuentes de interrupción externa (IRQ0 a IRQ31)
- Operaciones de multiplicación y división de 32 bits por hardware
- Instrucciones específicas de multiplicación de 64 y 128 bits
- Instrucciones para operaciones en coma flotante en precisión simple
- Acceso a los periféricos mediante *mapping* de memoria

El microprocesador está formado por un conjunto de unidades funcionales, dependiendo de la versión implementada. Se proporcionan tres versiones distintas según se busque maximizar el rendimiento del procesador o minimizar la cantidad de recursos lógicos utilizados:

- Nios II *economy*: versión que requiere menos recursos lógicos de la FPGA. No dispone de “pipeline”, y está muy limitada, ya que por ejemplo, no es posible de multiplicación y división hardware.
- Nios II *standard*: versión con “pipeline” de cinco etapas dotada de unidad aritmética y lógica (ALU, *Arithmetic Logic Unit*). Tiene como objetivo llegar a un compromiso entre rendimiento y consumo de recursos.
- Nios II *fast*: versión orientada al alto rendimiento, que dispone de un “pipeline” de seis etapas y proporciona opciones que permiten aumentar el rendimiento, como memorias caché de datos e instrucciones, controlador de interrupciones vectorizado, unidad de gestión de memoria (MMU, *Memory Management Unit*).

Para realizar un sistema embebido completo basado en el Nios II, cada una de dichas versiones se complementa con otros elementos como pueden ser memorias y periféricos, por medio de su conexión a través de un bus de sistema denominado “*Avalon Switch Fabric*”.

3.3.3.- Excepciones

El controlador de excepciones es el circuito que se encarga de las tareas relacionadas con la atención de excepciones. Las excepciones son aquellas situaciones anómalas que interrumpen el flujo normal de ejecución de un programa, y que requieren la atención del procesador. Pueden ser de dos tipos: hardware y software.

- Excepciones hardware:
 - Interrupción hardware (*"hardware interrupt"*): se da cuando se activa una de las 32 entradas (IRQ0 a IRQ31) de las que dispone el microprocesador al utilizar el controlador de interrupciones interno (IIC, *Internal Interrupt Controller*), lo que permite el acople asíncrono entre el microprocesador y un periférico.
- Excepciones software:
 - Instrucción *trap* (*"software trap"*): esta excepción salta cuando se ejecuta la instrucción *trap* en un programa. Esto permite transferir la ejecución de forma instantánea a otro programa, generalmente, al sistema operativo.
 - Instrucción no implementada (*"unimplemented instruction"*): se produce si el procesador intenta ejecutar una instrucción válida que no está implementada en hardware. Este es el caso de las instrucciones de multiplicar y dividir al utilizar el Nios II con la versión *economy*.

Al producirse una excepción, el procesador transfiere la ejecución al controlador de excepciones. La gestión de las excepciones puede ser de dos tipos: vectorizada (por hardware), o basada en una única dirección (por software mediante sondeo). Si es vectorizada, cuando se produce la excepción se transfiere la ejecución a la dirección de memoria indicada por el vector de excepción, puesto que la excepción está completamente identificada. Sin embargo, si se gestiona por medio de una única dirección, es la rutina de atención quien debe determinar qué tipo de excepción se ha producido, empeorando por tanto la latencia.

El procesador Nios II incluye en sus tres versiones un controlador interno de interrupciones, y ofrece la posibilidad en caso de necesidad por parte del diseñador, de incorporar un controlador de interrupciones externo de tipo vectorizado (este último en la versión *fastest* del procesador).

- IIC (*"Internal Interrupt Controller"*): gestiona 32 entradas de interrupción (IRQ0 a IRQ31) por medio de un único vector de interrupción, con lo que la identificación y resolución de prioridad debe ser resuelta por software mediante sondeo (*"polling"*).
- EIC (*"External Interrupt Controller"*): incorpora un control de interrupciones de tipo vectorizado (VIC, *Vectored Interrupt Controller*) que permite la identificación de la excepción de forma directa por hardware, en base a unos niveles de prioridad determinados a priori.

Cuando se produce una petición de interrupción (IRQ, *Interrupt ReQuest*), se realiza el siguiente número de tareas de forma secuencial:

- 1) La ejecución de la instrucción en curso es cancelada.
- 2) Se guarda el estado de la CPU, el valor del contador de programa y la información de estado. El objetivo de guardar el contexto es, una vez finalizada la ISR de la excepción, poder volver al punto de ejecución previo en el que ha saltado la interrupción.
- 3) Deshabilitación de las interrupciones externas del procesador.
- 4) Cesión del control de ejecución al gestor de interrupciones para que identifique la fuente de interrupción y la active en el registro de control de interrupción, que es el registro donde se muestran las interrupciones pendientes de ser atendidas.
- 5) La ejecución salta a la dirección donde se encuentra la ISR y se ejecuta. Una vez ejecutada se borra el *flag* de petición de interrupción correspondiente a dicha interrupción.
- 6) Se carga el contexto (estado de la CPU, contador de programa...) previamente guardado, y se reanuda la ejecución del programa.

Para que se pueda generar una interrupción se deberán cumplir una serie de condiciones:

- I. Que el bit PIE del registro de estado se encuentre activado. Este bit es el que habilita las interrupciones de forma global.
- II. Que se produzca una interrupción en una de las líneas de interrupción IRQ.
- III. Que el bit correspondiente a la línea de interrupción esté activado en el registro de control de interrupciones (IENABLE).

3.3.4.- Acceso a memoria y periféricos

El procesador Nios II tiene buses independientes para instrucciones y datos, lo que permite leer de forma simultánea una instrucción y un dato. Con este tipo de conexiónado, el bus de instrucciones se utiliza para leer la secuencia de instrucciones (el programa) almacenado en memoria, que serán ejecutadas por el procesador. Por otra parte, el bus de datos es el encargado de acceder tanto a las distintas posiciones de memoria donde se almacenan los operandos, como también a los periféricos, que se encuentran mapeados en memoria.

En el procesador Nios II las memorias de datos y los periféricos comparten el bus de datos de 32 bits. De este modo, el acceso de entrada y salida se realiza por medio de un mapa de memoria, de la misma forma que el acceso a cualquiera otra dirección de memoria, haciendo uso de las mismas instrucciones de lectura y escritura.

Puesto que tanto la memoria como el conjunto de los periféricos son configurables, el mapa de direcciones de memoria depende totalmente del sistema diseñado. Dicho mapa de direcciones, al igual que la dirección de reset y las prioridades de interrupción, se establecerán en el momento de generar el sistema.

Los periféricos se pueden clasificar en dos tipos: estándar y a medida:

- Estándar: son aquellos periféricos proporcionados por Altera que permiten realizar tareas comunes, tales como entradas y salidas de propósito general, temporizadores, controladores de memoria SDRAM, controladores de memoria Flash, interfaces de comunicación serie, entre otros.
- A medida: en los periféricos a medida es donde se muestra la potencia de los sistemas embebidos configurables sobre FPGA, ya que es el diseñador el que decide qué partes se implementarán en hardware y cuáles en software. De esta forma, todos aquellos procedimientos o algoritmos cuyo rendimiento sea crítico serán implementados en hardware a través de un periférico totalmente a medida. Así, se obtienen dos beneficios. Por una parte se consigue un mayor rendimiento del periférico gracias al carácter concurrente de la implementación por hardware, y por otra parte se libera al procesador de dicha tarea.

3.3.5.- Sistema Operativo uC/OSII

El sistema operativo (OS, *Operative System*) utilizado en este proyecto es el MicroC/OSII. Es un kernel producido por Micrium Inc. Además de estar soportado para el NiosII también lo está para otras arquitecturas de procesadores orientadas a aplicaciones de diversos ámbitos. Proporciona los siguientes servicios:

- Tareas (hilos de ejecución)
- *Flags* para captar eventos
- Sistemas de colas para comunicar hilos
- Semáforos
- Gestión de tiempo del procesador (reparto de tiempo entre tareas)

MicroC/OSII opera sobre una capa de abstracción de hardware (HAL, *Hardware Abstraction Layer*). Debido a esto, el desarrollo de aplicaciones sobre el NiosII tiene las siguientes ventajas:

- Los programas son portables hacia otros sistemas basados en NiosII.
- Los programas seguirán funcionando a pesar de hacer cambios en el hardware subyacente, ya que los programas solo “ven” la capa de abstracción HAL.
- Facilitad a la hora de implementar las interrupciones.

La arquitectura del sistema operativo sigue el siguiente esquema:

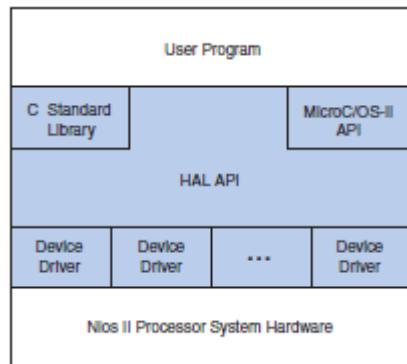


FIGURA 16: ARQUITECTURA DEL SISTEMA OPERATIVO MICROC/OSII

3.3.6.- Programación del Nios II

Para llevar a cabo la programación, es decir, la inclusión del software que permita controlar el procesador, disponemos de un *plugin* para el entorno *Eclipse* con el que Altera nos facilita el desarrollo de aplicaciones para sus procesadores.

La interacción con los elementos hardware se realiza mediante la HAL, lo que nos permite abstraernos del hardware implementado haciendo más sencillo el desarrollo de aplicaciones.

3.4.- Unidad de comparación y captura

3.4.1.- Funcionamiento

La unidad de comparación y captura (UCC) utilizada en el presente proyecto y una versión básica de los instrumentos (frecuencímetro y generador de funciones) ha sido desarrollada por [9]. Dispone de los siguientes modos de funcionamiento:

- **Modo captura:** permite la medida del tiempo de duración de un evento. En nuestro caso es usado para implementar un frecuencímetro, de forma que medirá el tiempo entre dos flancos de subida de una señal cuadrada externa. Se disponen de cinco registros de captura (captura0, captura1, captura2, captura3 y captura4).
- **Modo comparación:** compara valores contenidos en dos registros de comparación, siendo uno de ellos un temporizador. Sirve para sincronizar sistemas externos. Para dicho propósito también se disponen de cinco registros (compara0, captura1, captura2, captura3 y captura4). En nuestro proyecto no lo usaremos.
- **Modo PWM (*Pulse Width Modulation*):** permite obtener una salida cuadrada, de la cual podemos variar la frecuencia y el *duty cycle*⁶. Diferenciamos este modo del siguiente porque en este modo la salida PWM no tiene que pasar por un filtro, y por dicho motivo el rango de frecuencias de salida es mayor.
- **Modo salida no cuadrada:** en este modo podremos generar una función partir de los valores presentes en una RAM de 36 posiciones. Dividimos el

⁶ Duty cycle: porcentaje de tiempo que la señal está a nivel alto dentro de un periodo.

periodo de la señal en grados (entre 0 y 360), y tomamos muestras de la señal cada 10 grados. De esta forma disponemos de 36 valores de señal que definen un periodo completo de la misma. Dichos valores pueden ser reconfigurados por software con el objetivo de que se pueda representar cualquier función en la salida.

A continuación se muestra una figura de la misma con las entradas (a la izquierda) y las salidas (a la derecha).

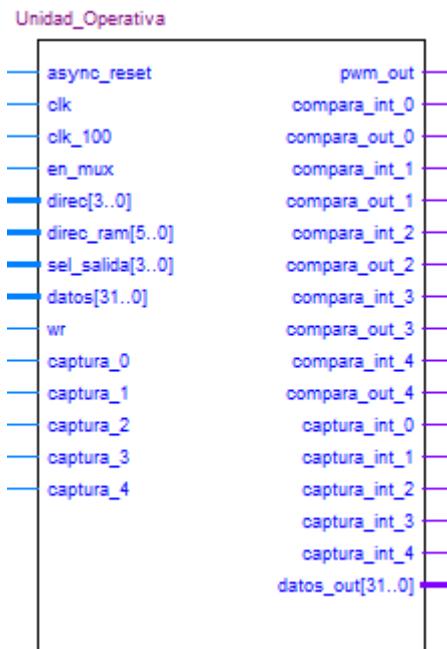


FIGURA 3.5: SÍMBOLO DE LA UNIDAD DE COMPARACIÓN Y CAPTURA

3.4.2.- Frecuencímetro

La señal cuadrada a la que queremos calcular su frecuencia se conecta a una de las entradas “captura_int_x”. Se capturan dos valores (uno por flanco de entrada) del temporizador interno de la unidad de captura y se almacenan en dos registros (*captura* y *capturaant*). Cuando se tienen los dos valores se genera una interrupción al procesador, y este, en la rutina de atención a la interrupción lee sendos valores y calcula la frecuencia asociada a dicho periodo.

3.4.3.- Generador de funciones

El generador de funciones tiene dos modos de funcionamiento diferenciados: señal de salida cuadrada, y señal de salida no cuadrada.

En el caso de la señal cuadrada la salida es una señal PWM a la cual se le puede modificar el *duty cycle* y la frecuencia.

Por otra parte, si queremos obtener una señal de tipo no cuadrada, utilizamos el generador PWM1 para generar una señal cuadrada 36 veces superior a la deseada. Esta señal es la que activa la RAM donde se almacenan los valores de la señal de salida. Posteriormente se tendrá que filtrar dicha señal para eliminar la PWM residual que se genera, de forma que podamos obtener la señal de salida deseada.

3.4.4.- Configuración

La configuración de la unidad de captura se almacena en el denominado “Registro de Configuración”. Se trata de un registro de 16 bits que es, naturalmente, accesible desde el exterior del periférico.

Bit 2	Bit 1	Bit 0	Funcionalidad
-	0	0	-
-	0	1	Modo Captura
-	1	0	Modo Comparación
0	1	1	Modo generador señal cuadrada
1	1	1	Modo generador señal NO cuadrada

Bit 4	Bit 3	Funcionalidad
-	0	Los registros de captura están activados
-	1	Se detienen las capturas
0	-	La salida del módulo de captura es “captura”
1	-	La salida del módulo de captura es “capturaant”

Bit 5	Funcionalidad
0	<i>Reload</i> del temporizador desactivado
1	<i>Reload</i> del temporizador activado

Bit	Flag de interrupción
Bit 6	Registro captura0
Bit 7	Registro captura1
Bit 8	Registro captura2
Bit 9	Registro captura3
Bit 10	Registro captura4
Bit 11	Registro compara0
Bit 12	Registro compara1
Bit 13	Registro compara2
Bit 14	Registro compara3
Bit 15	Registro compara4

Para escribir y leer en el “Registro de Configuración” y en otros bloques del periférico se utilizan dos multiplexores, uno de entrada (para escribir) y otro de salida (para leer).

Multiplexor de entrada (escritura)	
Dirección (4 bits)	Actúa sobre...
0000	Registro de configuración
0001	Precarga del <i>timer</i>
0010	Periodo del PWM
0011	Ton del PWM
0100	Comparador 0
0101	Comparador 1
0110	Comparador 2
0111	Comparador 3
1000	Comparador 4
1001	Periodo del PWM1
1010	Ton del PWM1
1011	Memoria RAM de la unidad (señal no cuadrada)

Multiplexor de salida (lectura)	
Dirección (4 bits)	Actúa sobre...
0000	Registro de configuración
0001	Valor del <i>timer</i>
0010	Periodo del PWM

0011	Ton del PWM
0100	Comparador 0
0101	Comparador 1
0110	Comparador 2
0111	Comparador 3
1000	Comparador 4
1001	Captura 0
1010	Captura 1
1011	Captura 2
1100	Captura 3
1101	Captura 4
1110	Memoria RAM de la unidad (señal no cuadrada)

Para más información sobre la unidad de comparación y captura se remite al lector a la bibliografía [9].

3.5.- Controlador Ethernet Davicom DM9000A

El chip DM9000A es un controlador de *Fast-Ethernet* (10/100 Mbps) totalmente integrado, de bajo coste y con un número reducido de patillas (48 en formato LQFP). Consta de una interfaz simple con el objetivo de conectarlo a un procesador de propósito general. Además tiene una capa física que permite la transmisión a 10 y 100 Mbps, y dispone de una memoria SRAM interna de 4 Kbytes. Una característica fundamental es la auto-negociación integrada, que de forma automática facilitará la configuración del enlace Ethernet, tal y como se explicó en [2.1.4.- Auto-negociación](#). También permite realizar por hardware el *checksum* de varios de los protocolos más utilizados (IP, TCP y UDP).

Para más información se remite a la hoja de características del controlador, disponible en la bibliografía [8].

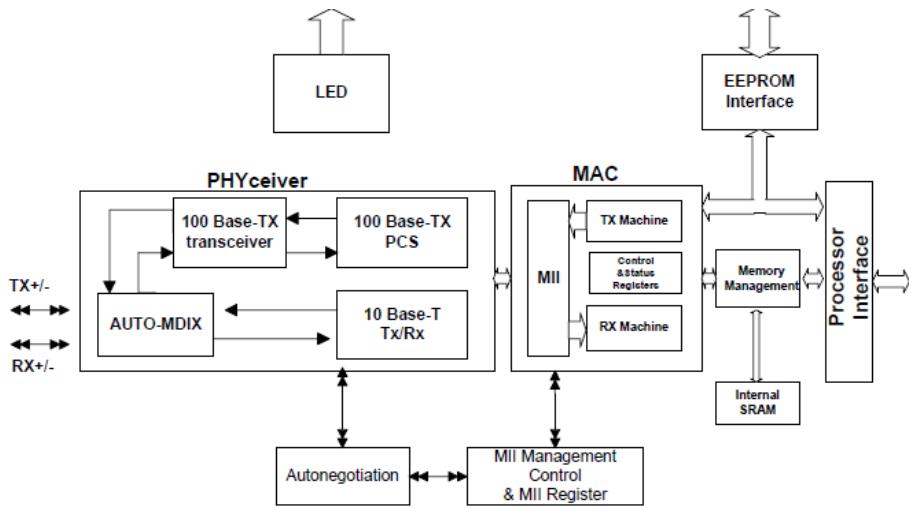


FIGURA 17: ESQUEMA DEL CONTROLADOR ETHERNET DAVICOM DM9000A

3.6.- Diseño del hardware

Para llevar a cabo el diseño hardware, Altera nos proporciona dos herramientas software fundamentales, Quartus II y Qsys.

3.6.1.- Quartus II 13.0

Quartus II es el entorno de desarrollo hardware de Altera. Una vez iniciado el programa se mostrará lo siguiente:

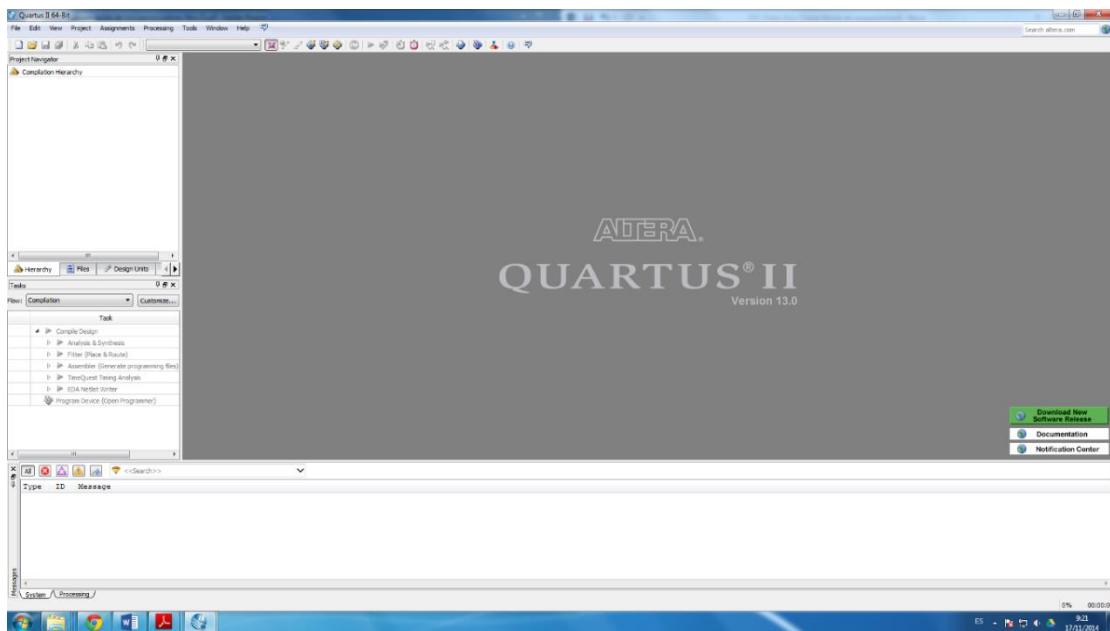


FIGURA 18: ENTORNO QUARTUS II 13.0

Para crear un proyecto, hacer click en **File -> New Quartus II Project**. Se abrirá un asistente que nos facilitará su creación. Podremos añadir ficheros externos y asignar al proyecto la FPGA utilizada. Los pasos a seguir son los siguientes:

1. Asignar un nombre y una ubicación al proyecto. Es importante que el nombre no contenga espacios y la ubicación no sea demasiado larga. Una vez completados los campos pulsar *Next*.

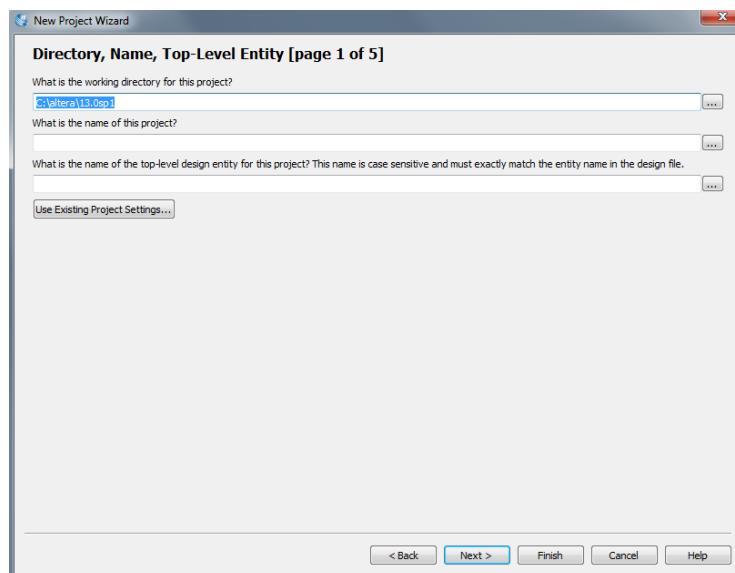


FIGURA 19: QUARTUS II NUEVO PROYECTO (1)

2. Añadir archivos al proyecto. En este momento no añadiremos ningún archivo, por tanto pulsamos *Next*.
3. Seleccionamos la FPGA con la que vamos a trabajar. En nuestro caso es la Cyclone II EPC235F672C6. La seleccionamos y pulsamos directamente *Next*.

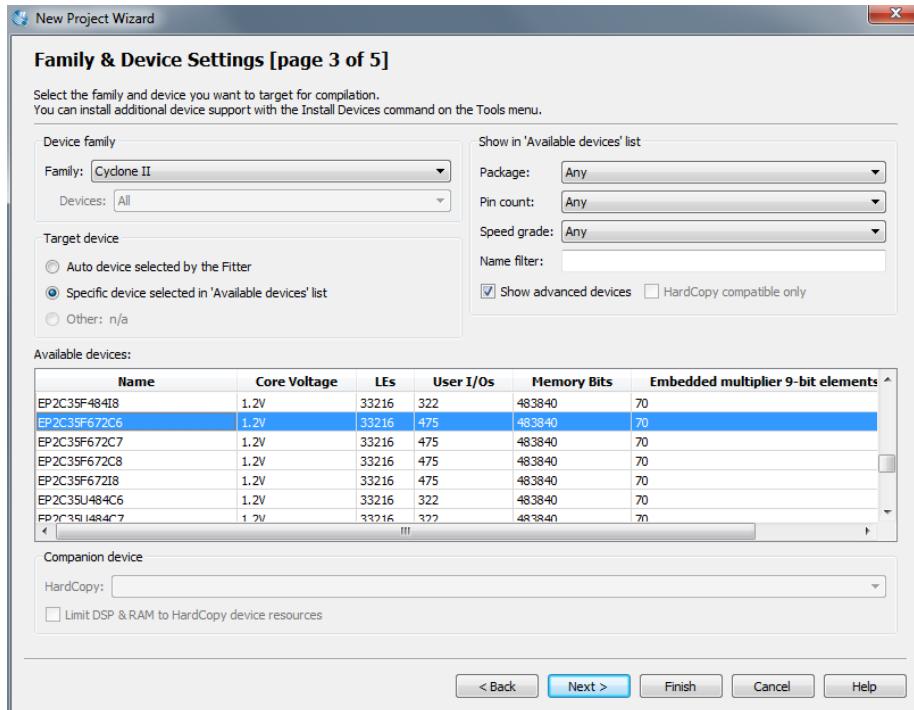


FIGURA 20: QUARTUS II NUEVO PROYECTO (2)

4. En el siguiente cuadro aparece el *EDA Tool Settings*. En nuestro caso no utilizaremos ninguna de las herramientas indicadas y por tanto pulsamos *Next* sin seleccionar ninguna.
5. En este último punto vemos un resumen de las características previamente seleccionadas, que las aceptaremos pulsando *Finish*.

3.6.2.- Qsys

Qsys permite generar el sistema hardware del procesador embebido Nios II de una forma muy simple y gráfica. Para utilizar Qsys basta con ir añadiendo elementos y realizando su interconexión.

Iniciamos la herramienta Qsys haciendo clic en **Tools -> Qsys** en el entorno Quartus II.

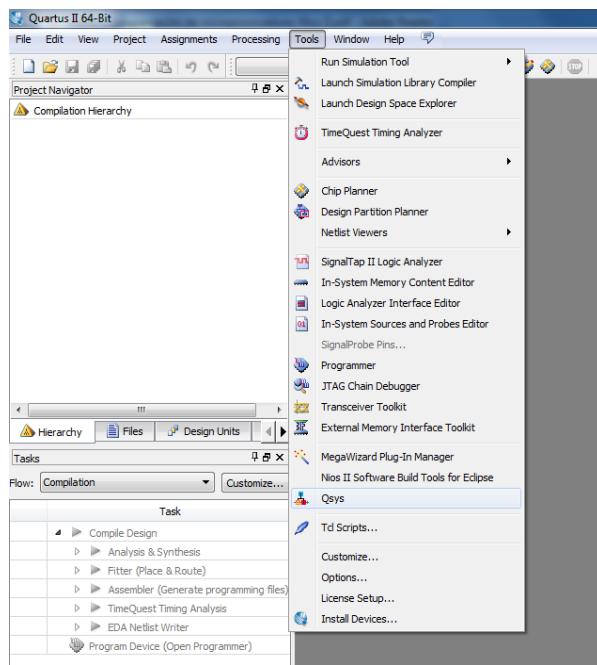


FIGURA 21: ABRIR QSYS EN QUARTUS

Una vez inicializada la herramienta veremos la apariencia de Qsys, que es la siguiente:

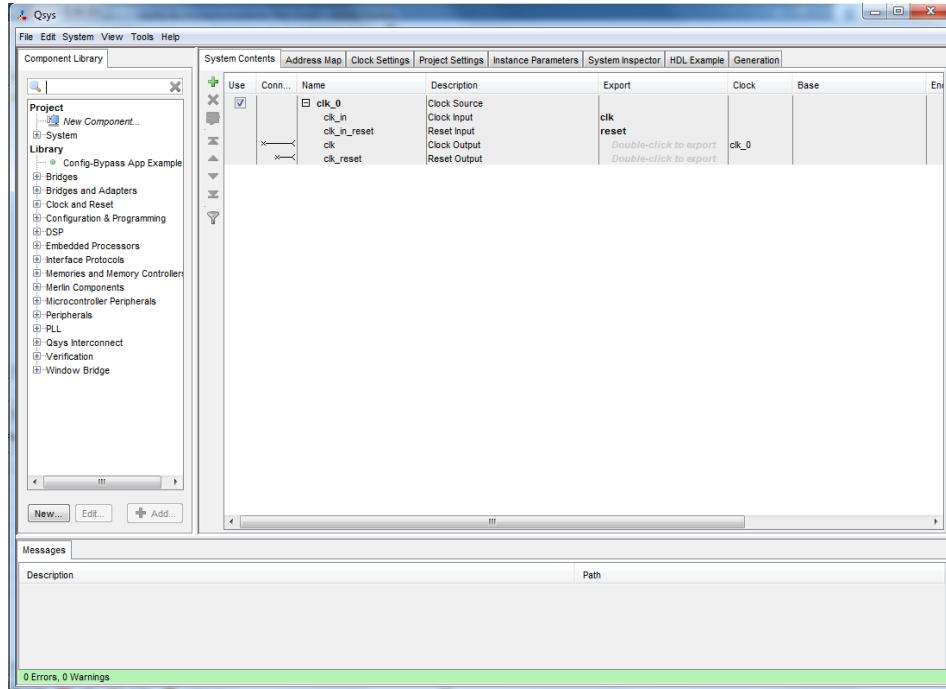


FIGURA 22: HERRAMIENTA QSYS

Una vez cargado el entorno se diferencian tres partes diferenciadas: en la parte izquierda vemos la biblioteca de componentes (*Component Library*), donde buscaremos los componentes que añadiremos al proyecto. En la parte central vemos el editor, en el cual, por defecto se muestra la pestaña *System Contents*. Por último, en la parte inferior se muestra un cuadro, el cual nos indicará los errores y advertencias (*warnings*) que se den en la integración de los componentes.

En el editor se muestran distintas pestañas:

- *System Contents*: nos permite editar cada componente y realizar el interconexión entre componentes. Las columnas que aparecen en esta pestaña tienen el siguiente cometido: *Use* indica si el componente se está usando o no en el diseño, *Connections* permite realizar las conexiones entre los componentes, *Name* indica el nombre del componente y el de sus interfaces de entrada y de salida, *Description* indica el tipo de componente y el tipo de entrada y salida, *Export* permite exportar una señal de forma que sea visible desde el exterior del sistema que se está generando en Qsys, *Clock* nos indica el reloj o relojes utilizados en cada componente, *Base* y *End* nos indica el comienzo y final de la dirección de memoria donde reside el mapeado de memoria de dicho componente, e *IRQ* permite ver y editar el nivel de prioridad de interrupción del componente asociado.
- *Address Map*: se muestra el mapa de memoria actual.
- *Clock Settings*: veremos los relojes que tenemos configurados (tanto el reloj básico como las derivaciones producidas por PLLs).
- *Project Settings*: nos permite indicar la FPGA con la que estamos trabajando.
- *Instance Parameters*: permite definir parámetros para configurar posibles instanciaciones de componentes. Nosotros no lo usaremos.
- *System Inspector*: podemos ver el conjunto de componentes e interconexiones que componen el sistema completo que hemos diseñado en la herramienta.
- *HDL Example*: nos muestra la declaración en HDL (Verilog o VHDL) del sistema.
- *Generation*: nos permite generar los ficheros del sistema para posteriormente integrarlo en un esquemático de Quartus II.

Lo primero de todo es configurar el sistema en base a nuestro hardware físico, que es la FPGA Cyclone II EP2C35F672C6. Para ello vamos a la pestaña *Project Settings* y lo configuramos de la siguiente forma:

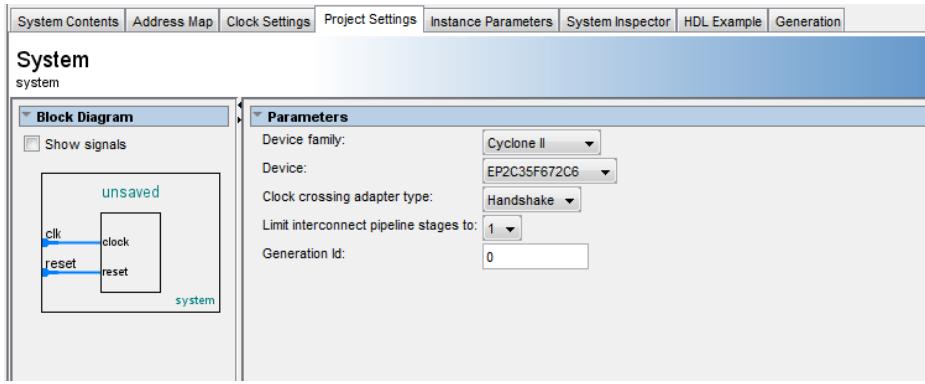


FIGURA 23: CONFIGURACIÓN DE LA FPGA EN QSYS

Una vez hecho esto pasamos a instanciar los componentes necesarios para nuestro proyecto.

3.6.3.- Integración de componentes del procesador con Qsys

En este apartado explicaremos qué componentes se deben instanciar y cuál debe ser su configuración (parametrización) para que el sistema funcione de acuerdo con nuestras necesidades. Se deben realizar dos pasos: primero buscar el componente en *Component Library* e instanciarlo haciendo doble clic. Posteriormente, hacer doble clic en el componente instanciado y configurar sus opciones en una ventana desplegable.

Los componentes que se introducirán serán los siguientes: reloj, PLL, CPU, controlador SDRAM, JTAG UART, temporizadores, entradas/salidas para controlar la unidad de comparación y captura, conmutadores, leds rojos y verdes, controlador de LCD, identificador del sistema, controlador Ethernet DM9000A.

3.6.3.1.- Reloj

El reloj ya se encuentra instanciado por defecto. Seleccionando el componente y haciendo **click derecho -> Rename** podemos cambiarle el nombre. Le llamamos *clk*. Haciendo doble clic sobre el componente accedemos a su configuración a través de una ventana emergente. En este caso se configura asignándole una frecuencia de 50 MHz. La siguiente figura es autoexplicativa:

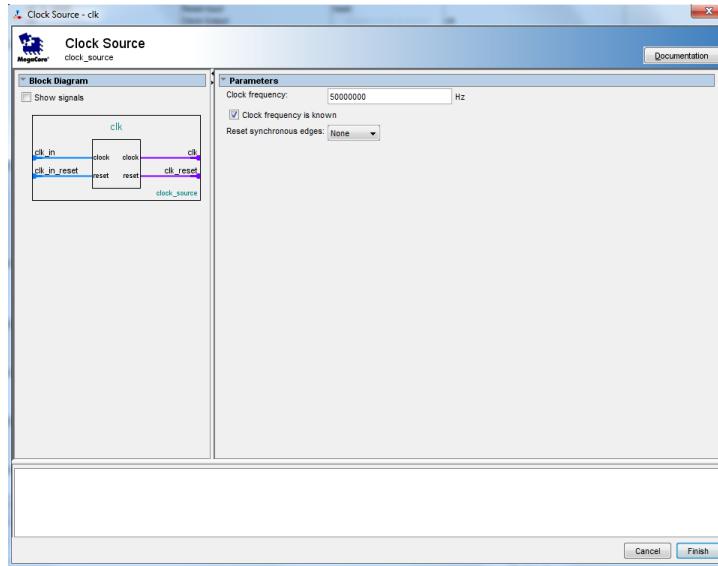
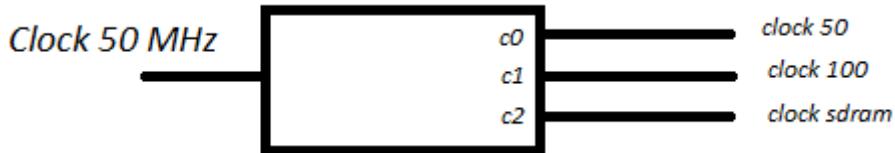


FIGURA 24: QSYS - RELOJ

3.6.3.2.- PLL

En la ventana *Component Library* seleccionamos **PLL -> Avalon ALTPLL**. Cada bloque PLL nos permite generar tres relojes independientes. En nuestro caso generamos tres señales de reloj que nos permitirán controlar distintos componentes:



- *Clock 50*: reloj de 50 MHz que se conectará a los elementos que necesiten dicha frecuencia de funcionamiento.
- *Clock 100*: reloj de 100 MHz que se conectará a la unidad de comparación y captura.
- *Clock sram*: reloj de 50 MHz con un desfase de -3ns con respecto al reloj clock 50, que se conectará al controlador de memoria SDRAM.

Una vez instanciado hacemos doble clic y lo configuramos de la siguiente forma:

1. En **Paramenter Settings -> Gerenal/Modes** introducir la frecuencia del reloj de entrada, en nuestro caso 50 MHz. Hacer clic en *Next*.

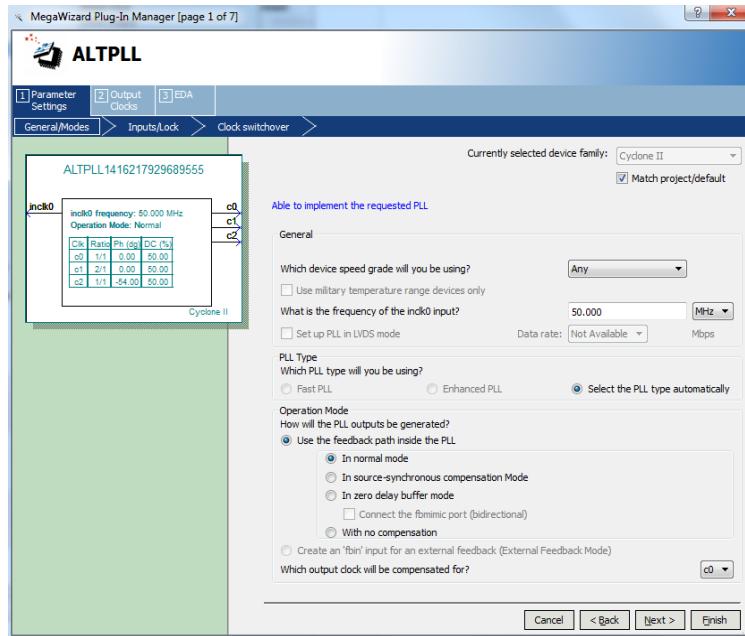


FIGURA 25: QSYS - PLL (1)

2. En **Parameter Settings -> Inputs/Lock:**

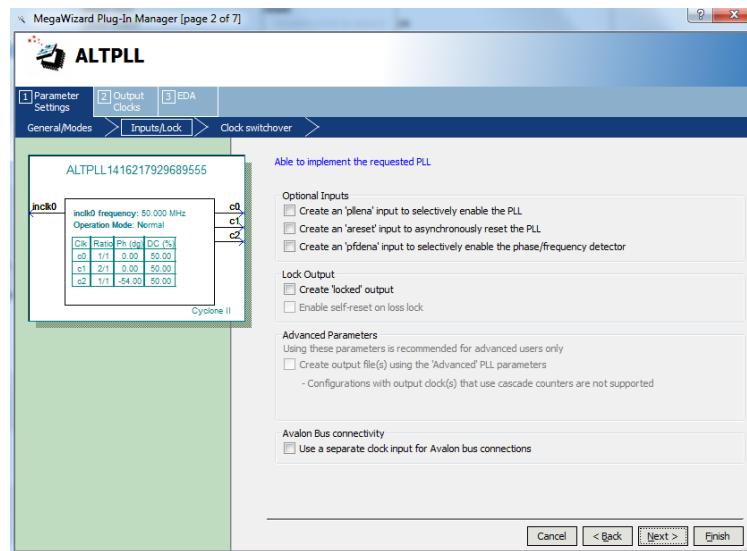


FIGURA 26: QSYS - PLL (2)

3. Parameter Settings -> Clock switchover:

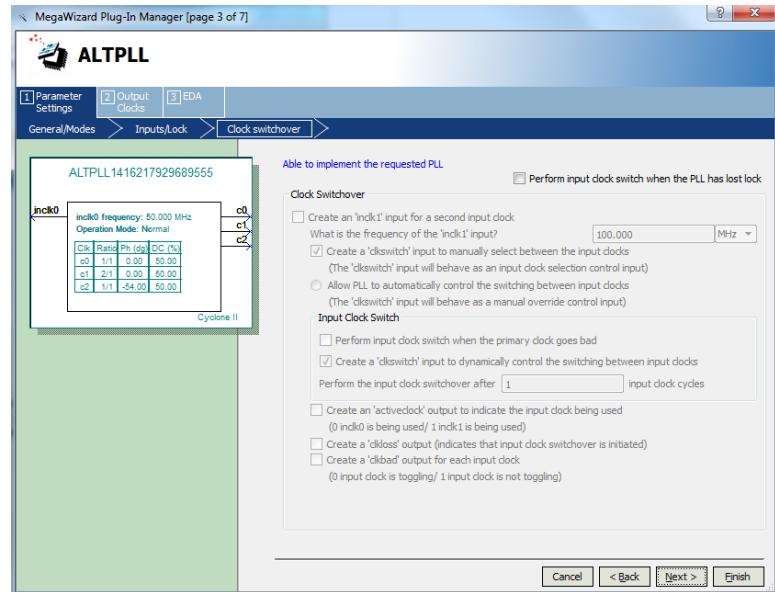


FIGURA 27: QSYS - PLL (3)

4. Output Clocks -> clk c0:

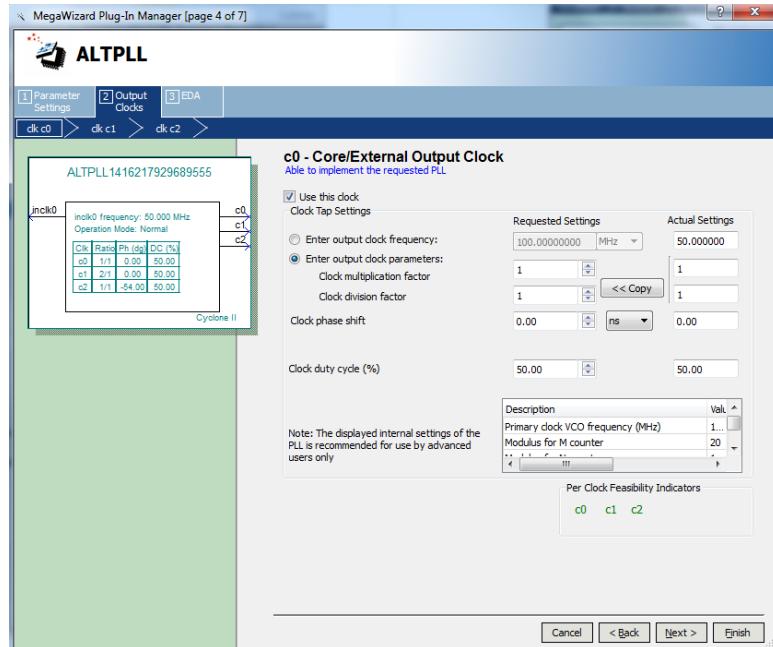


FIGURA 28: QSYS - PLL (4)

5. Output Clocks -> clk c1:

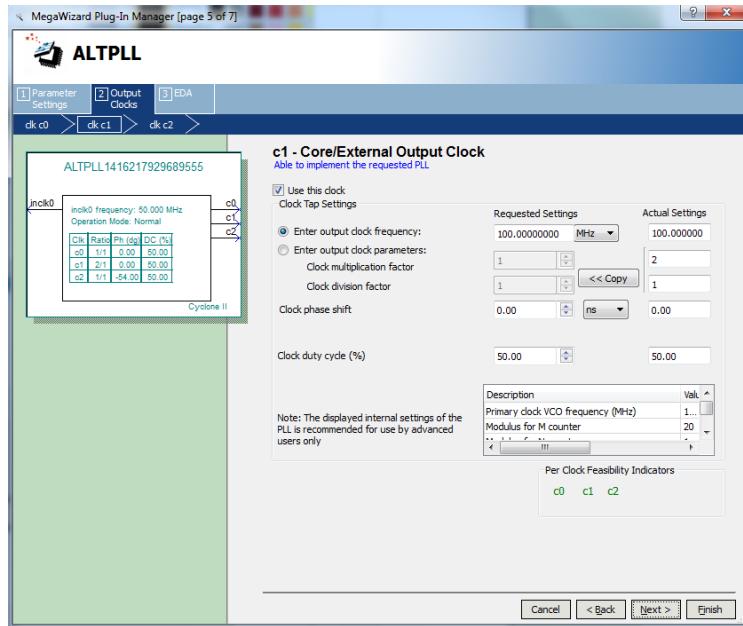


FIGURA 29: QSYS - PLL (5)

6. Output Clocks -> clk c2. Una vez configurado según la siguiente figura hacer clic en *Finish*.

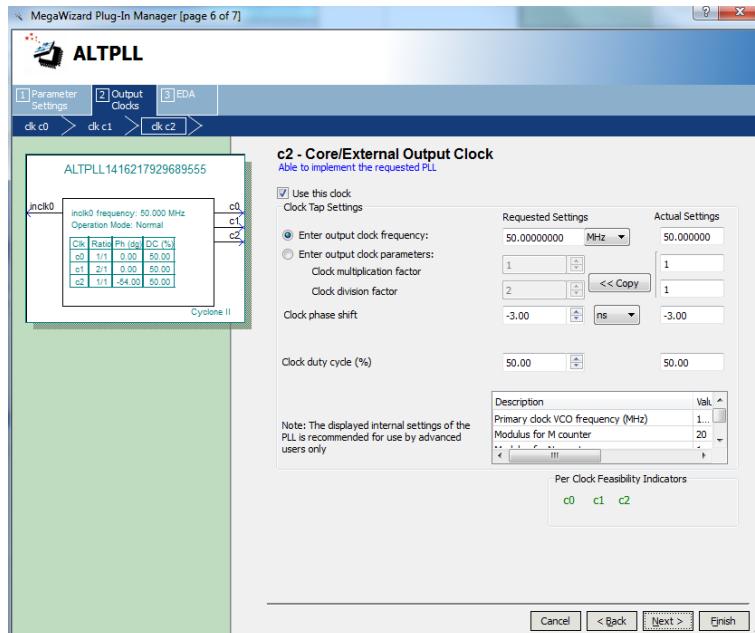


FIGURA 30: QSYS - PLL (6)

Una vez configurado podremos cambiarle el nombre al componente. Le denominamos *pll*.

La conexión entre el PLL y el reloj se debe realizar de la siguiente forma: se deben conectar las señales *clk* y *clk_reset* del reloj a las entradas *inclk_interface* y *inclk_interface_reset* del PLL. Además se deberán exportar dos señales: la señal *c1* se exporta con el nombre *clock_100*, y la señal *c2* se exporta con el nombre *sram_clock*. Para exportar una señal basta con hacer doble clic en la fila correspondiente de la columna *Export* y asignarle un nombre identificativo de la señal a exportar.

El reloj *c0* se conectará con todos aquellos elementos que necesiten un reloj de 50 MHz.

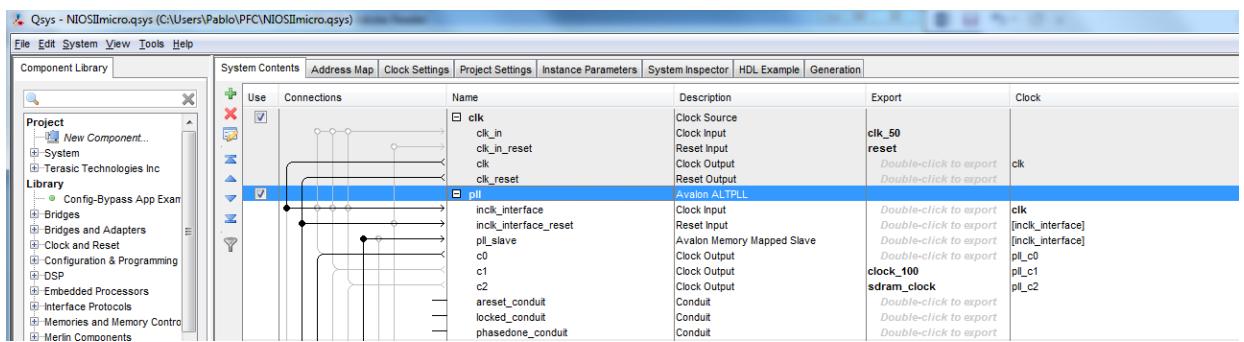


FIGURA 31: QSYS - CONEXIÓN DEL PLL

3.6.3.3.- CPU

La CPU es el núcleo del sistema. Para incorporarlo hacer clic en **Library -> Embedded Processors** en la biblioteca de componentes y hacer doble clic sobre *Nios II Processor*. Se configurará con las siguientes características:

1. En la pestaña *Core Nios II* seleccionar la versión *fastest* (Nios II/f) y habilitar la multiplicación hardware (*Embedded Multipliers*). No preocuparse si aparecen errores relacionados con el vector de reset y el vector de excepción. Serán solucionados cuando integremos la memoria SDRAM.

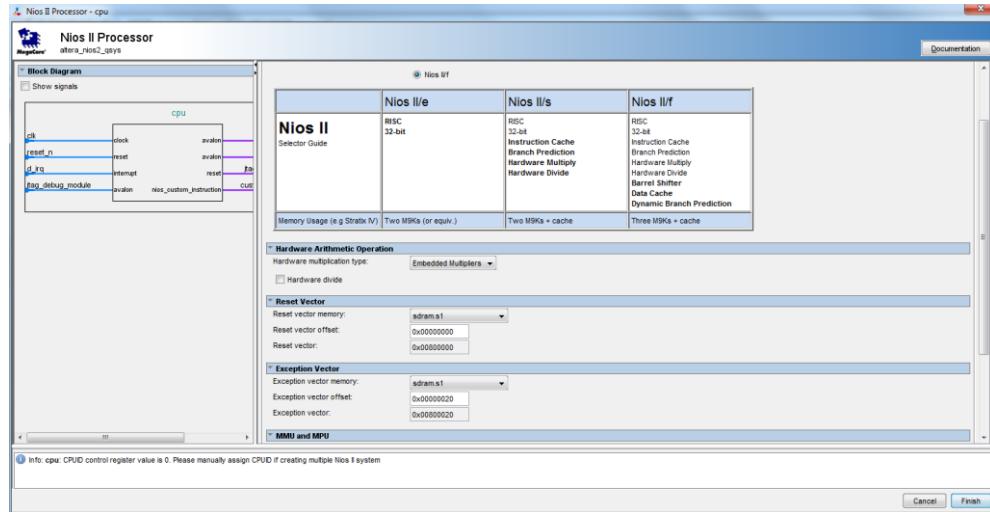


FIGURA 32: QSYS CONFIGURACIÓN DE CPU (1)

2. En la pestaña *Caches and Memory Interfaces* introduciremos dos memorias caché, una de instrucciones de 4 Kbytes y otra de datos de 2 Kbytes:

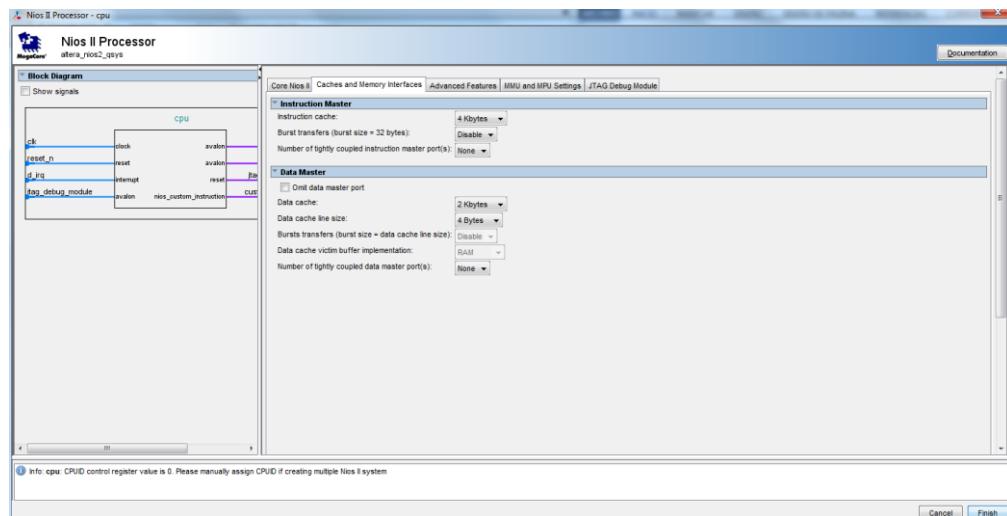


FIGURA 33: QSYS CONFIGURACIÓN DE CPU (2)

3. En la pestaña *Advance Features* seleccionaremos el tipo de controlador de interrupciones, en nuestro caso será interno.

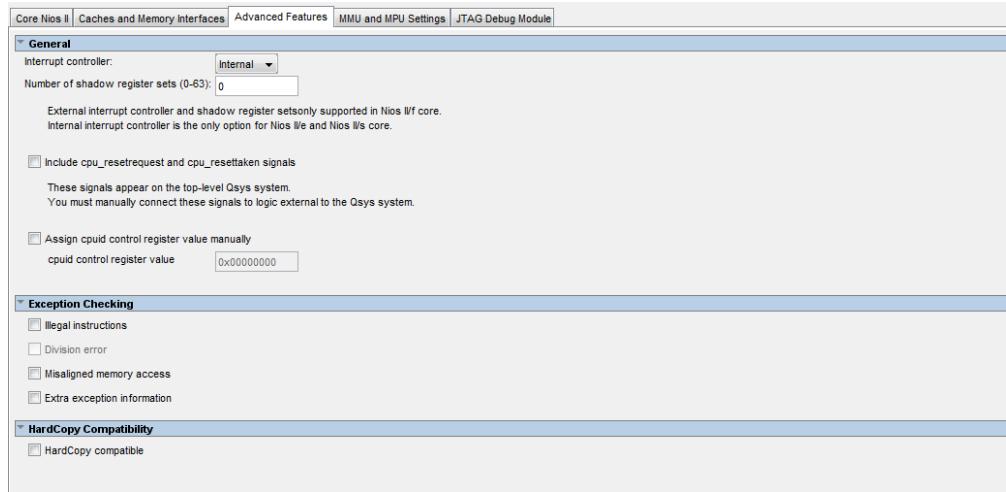


FIGURA 34: QSYS CONFIGURACIÓN DE CPU (3)

4. *MMU and MPU Settings* lo dejaremos como está.
5. *JTAG Debug Module*. Activaremos el nivel 1 de depuración.

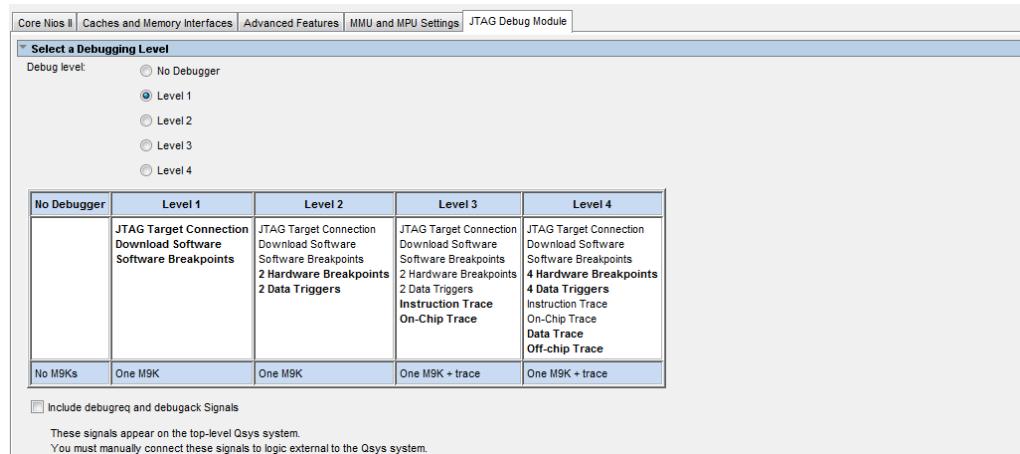


FIGURA 35: QSYS CONFIGURACIÓN DE CPU (4)

La conexión con el resto de componentes se muestra en la siguiente figura. Se introducen las señales *clk* y *reset* que vienen del reloj del sistema, y salen de la CPU las señales *data_master* (bus de datos) y *instruction_master* (bus de instrucciones).

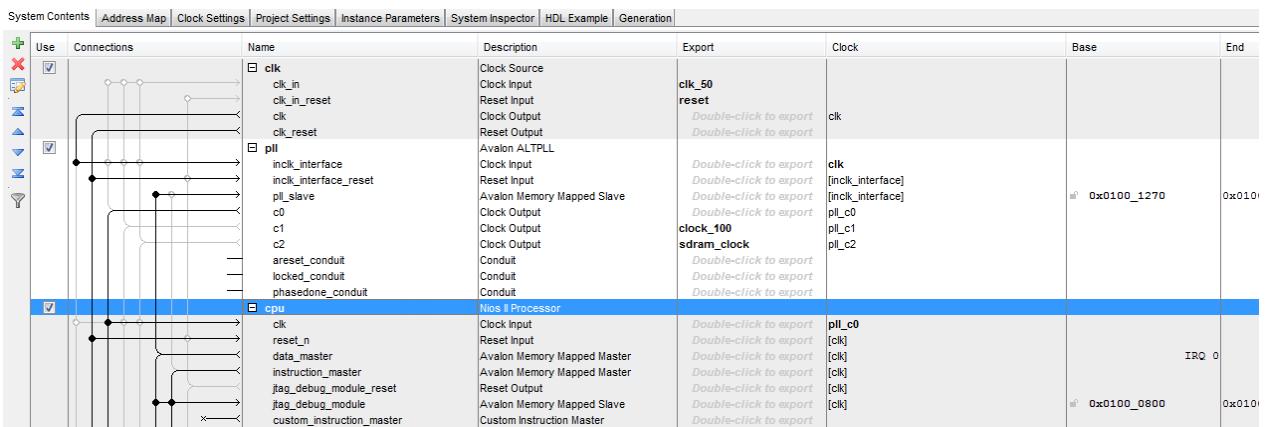


FIGURA 36: QSYS CONEXIÓN DE CPU

3.6.3.4.- Controlador SDRAM

Para controlar la memoria SDRAM de 8 Mbytes disponible en la placa de desarrollo necesitamos un controlador que nos simplifique la interfaz. Incorporar el componente haciendo doble clic en **Memories and Memory Controllers -> External Memory Interfaces -> SDRAM Interfaces -> SDRAM Controller**. Configurarlo de la siguiente forma:

1. Pestaña *Memory Profile*:

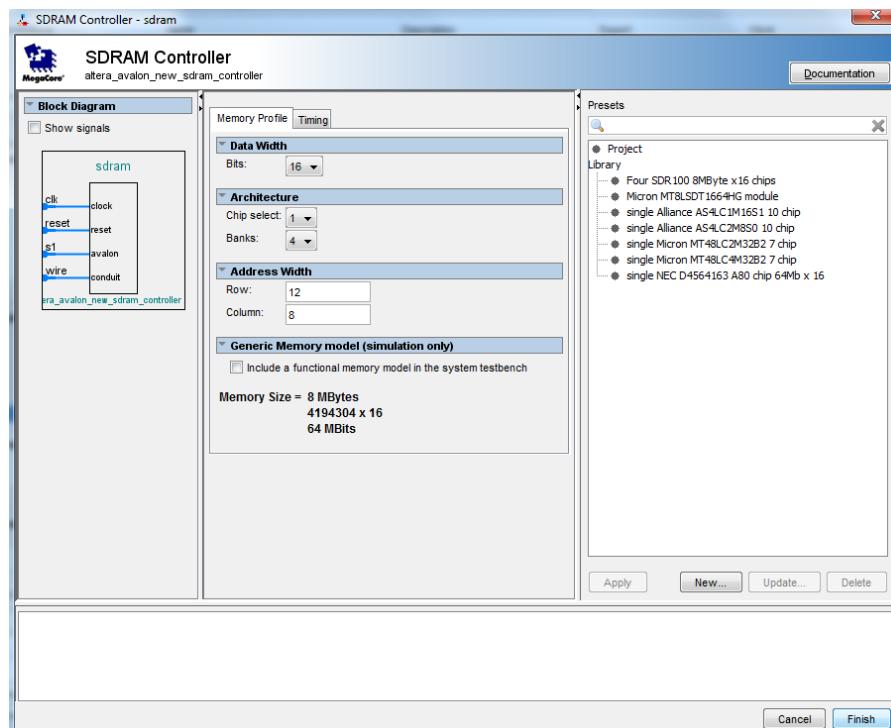


FIGURA 37: QSYS CONTROLADOR SDRAM (1)

2. Pestaña Timing:

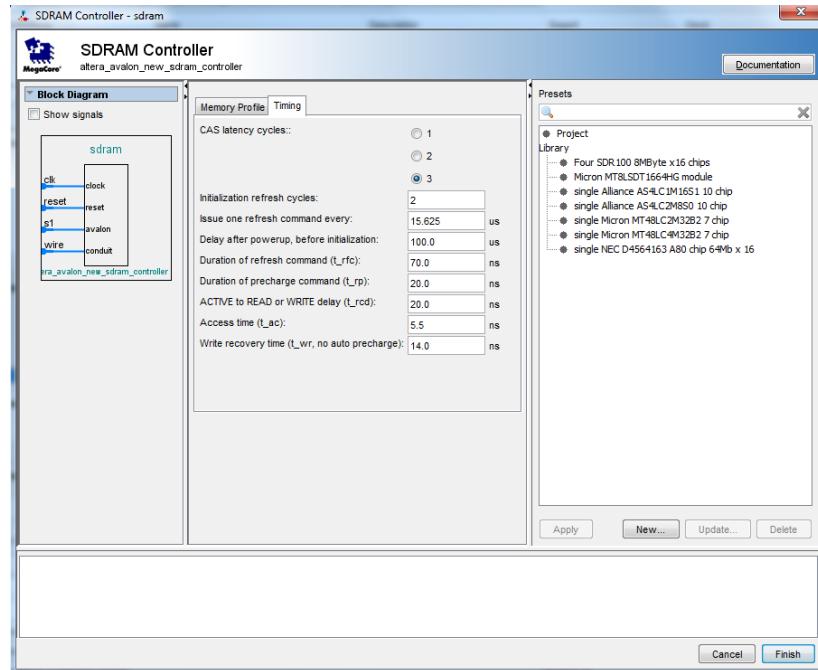


FIGURA 38: QSYS CONTROLADOR SDRAM (2)

Una vez configurado el controlador asignarle el nombre *sram*, y realizar el conexionado con la CPU. Se deberá introducir las señales *c0*, *reset*, *instruction_master*, y *data_master*. Además se debe exportar la señal *wire* con el nombre *sram_controller*.

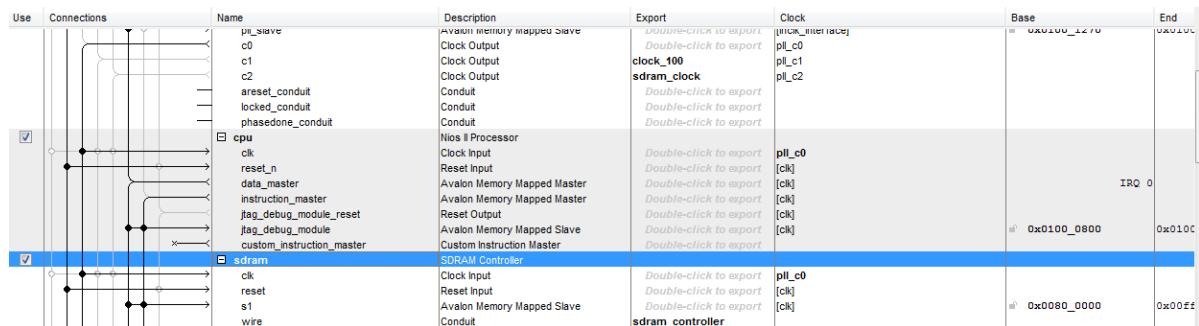


FIGURA 39: QSYS CONEXIÓN DEL CONTROLADOR SDRAM

Una vez añadido el controlador de SDRAM ya podemos finalizar la configuración de la CPU estableciendo el vector de reset y el vector de excepción. Para ello seleccionar la *cpu*, acceder a su configuración y configurarla del siguiente modo:

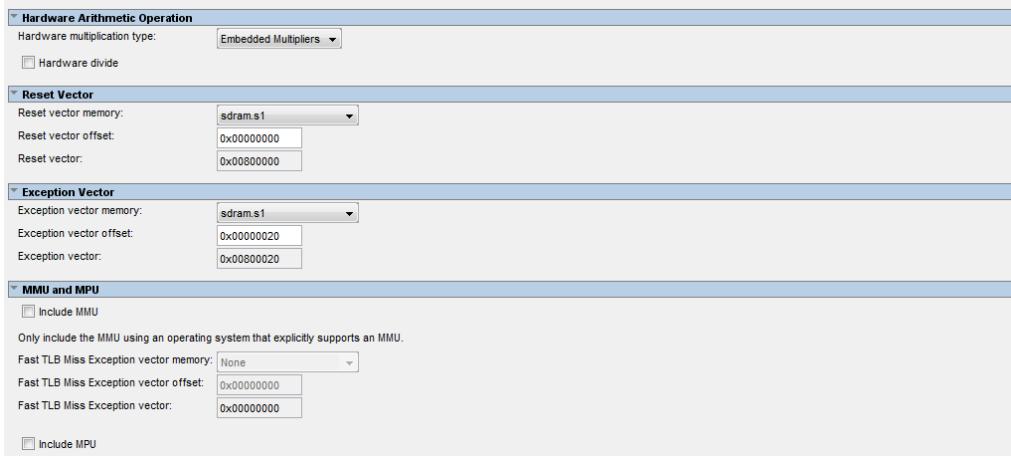


FIGURA 40: QSYS FINALIZACIÓN DE LA CONFIGURACIÓN DE LA CPU

3.6.3.5.- JTAG UART

La interfaz JTAG permite comunicar al procesador Nios II con el PC a través de series de caracteres. Lo usaremos para depurar el programa software.

En la biblioteca de componentes seleccionar **Interface Protocols -> Serial -> JTAG UART**. Dejar la configuración por defecto, renombrarla como *uart*, y conectar las señales de c0, reset y data_master.



FIGURA 41: QSYS CONEXIÓN JTAG UART

3.6.3.6.- Temporizadores

Se introducirán dos temporizadores, *timer_0* y *timer_1*. En la biblioteca de componentes seleccionar **Peripherals -> Microcontroller Peripherals -> Interval Timer**.

El *timer_0* tendrá un periodo de 100 us y el *timer_1* de 1 ms, que serán los tiempos que tarden en desbordarse. El *timer_0* es el temporizador del sistema. Es el que se encargará del cambio de ejecución entre unas tareas y otras.

La configuración de ambos temporizadores será idéntica según la siguiente figura, a excepción del periodo:

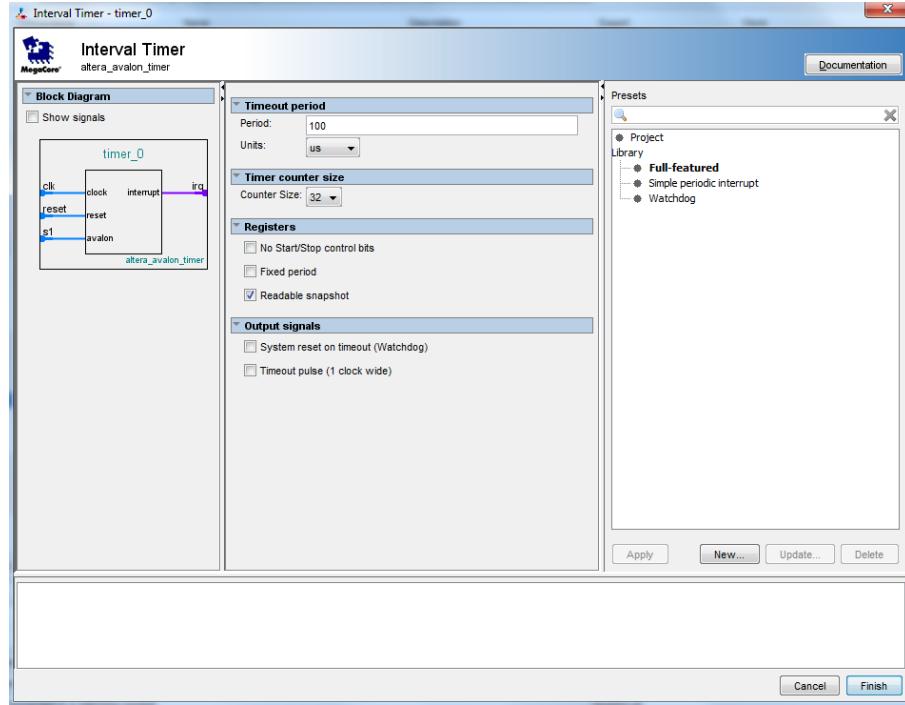


FIGURA 42: QSYS TEMPORIZADOR

Las señales que se utilizan para conectar los temporizadores son: *c0*, *reset*, y *data_master*. La conexión queda del siguiente modo:

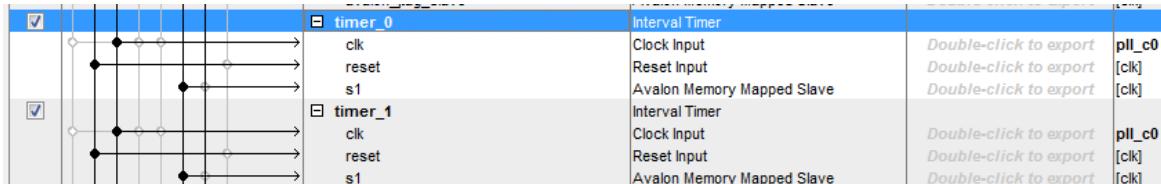


FIGURA 43: QSYS CONEXIÓN DE LOS TEMPORIZADORES

3.6.3.7.- Pulsadores

Este bloque está compuesto por tres pulsadores de la placa (KEY1, KEY2 y KEY3). Se configura como bloque de entrada/salida (PIO). En la biblioteca de componentes seleccionar **Peripherals -> Microcontroller Peripherals -> PIO**.

Se renombra como *pio_in_key_edge* y se configura de la siguiente forma:

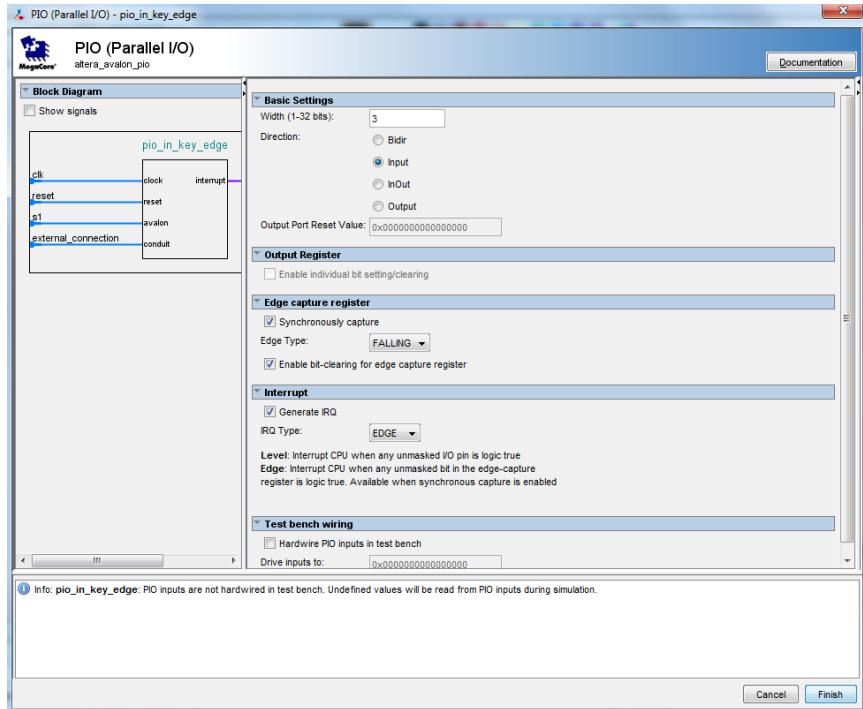


FIGURA 44: QSYS PULSADORES

Se conectan las señales *c0*, *reset*, y *data_master*, y exporta la señal “*external connection*” con el nombre *pio_in_key_edge*.

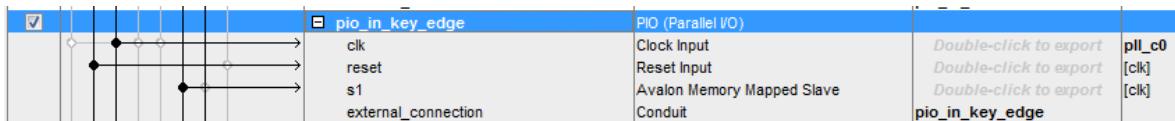


FIGURA 45: QSYS CONEXIÓN PULSADORES

3.6.3.8.- Comutadores

Se introducen los 18 conmutadores que dispone la placa. No serán usados en el sistema final, pero sí serán útiles para generar el sistema desde sus inicios. Incluimos un bloque PIO configurado de la siguiente forma:

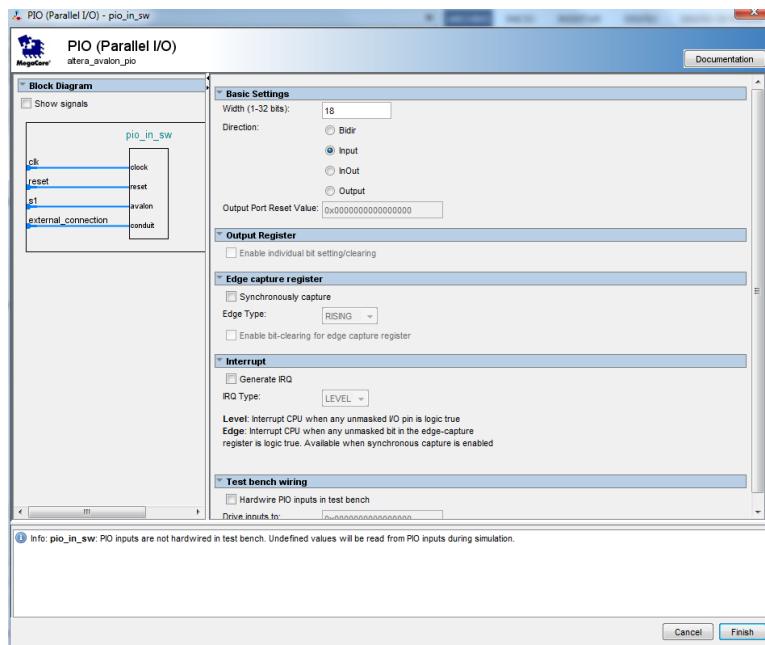


FIGURA 46: QSYS CONMUTADORES

Se conectan las señales *c0*, *reset*, *data_master*, y se exporta la señal “*external connection*” como *pio_in_sw*.

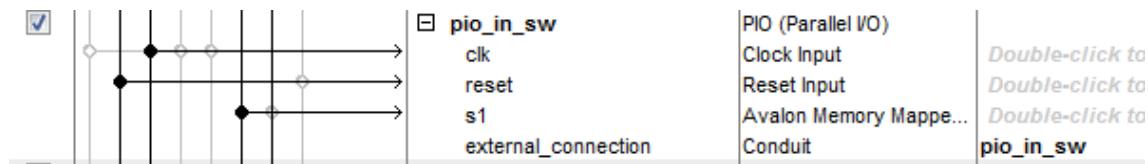


FIGURA 47: QSYS CONEXIÓN DE LOS CONMUTADORES

3.6.3.9.- Leds Rojos

Componente PIO que renombramos con el nombre *pio_out_red* y configuramos de la siguiente manera:

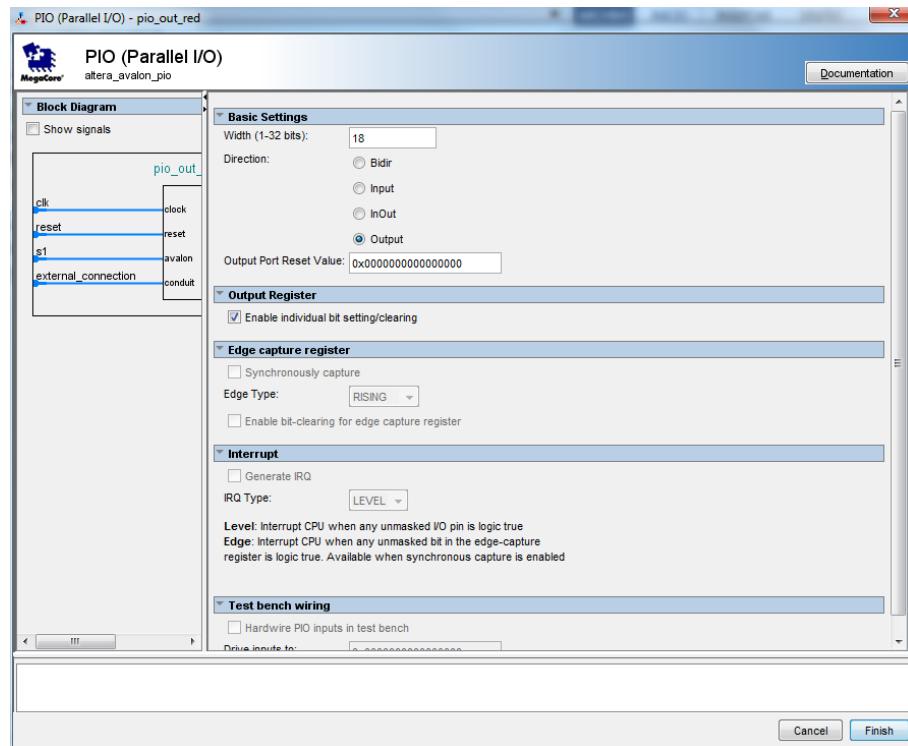


FIGURA 48: QSYS LEDS ROJOS

Se conectan las señales *c0*, *reset*, y *data_master*, y se exporta la señal “*external_connection*” con el nombre *pio_out_red*.



FIGURA 49: QSYS CONEXIÓN LEDS ROJOS

3.6.3.10.- Leds Verdes

Componente PIO que renombramos con el nombre *pio_out_green* y configuramos de la siguiente manera:

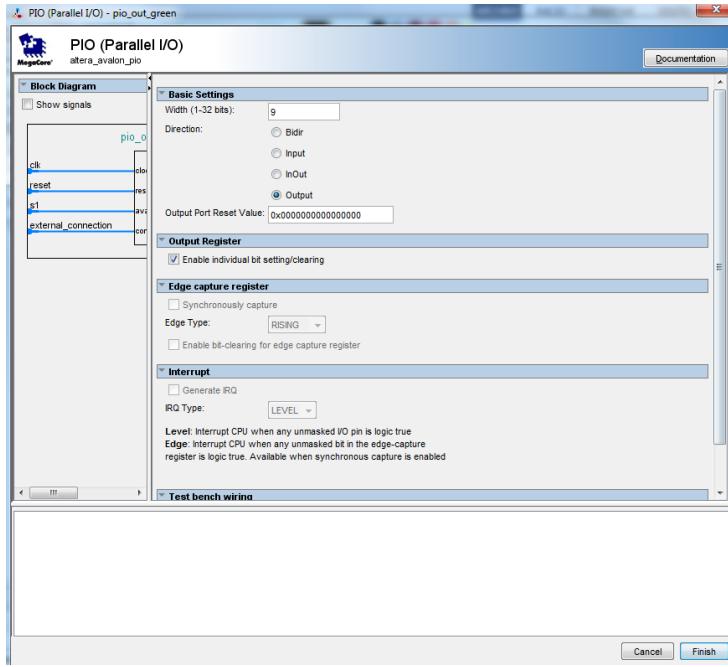


FIGURA 50: QSYS LEDS VERDES

Se conectan las señales *c0*, *reset*, y *data_master*, y exporta la señal “*external connection*” con el nombre *pio_out_green*.

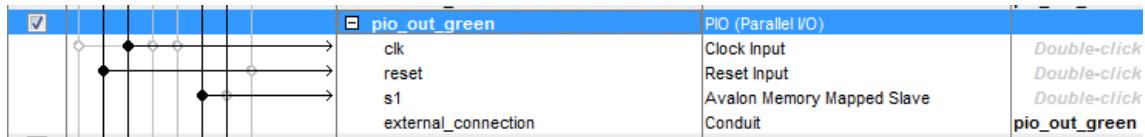


FIGURA 51: QSYS CONEXIÓN LEDS VERDES

3.6.3.11.- Elementos de control de la UCC

A continuación añadiremos como entradas/salidas PIO las líneas de conexión que nos permitirán conectar la unidad de comparación y captura. La configuración de estos bloques se realiza de forma idéntica a como se han configurado los anteriores bloques de PIOs. Una vez configurado los bloques se conectan a cada uno de ellos las señales *c0*, *reset*, *data_master*, y se exporta la señal “*external connection*” con el nombre de dicho bloque.

La siguiente tabla resume dicha configuración:

Bloque (Nombre)	Salida/Entrada	Número de bits
Pio_in_ext	Entrada	8
Pio_out_ext	Salida	1
Pio_async_reset	Salida	1
Pio_en_mux	Salida	1
Pio_direc	Salida	4
Pio_direc_ram	Salida	6
Pio_wr	Salida	1
Pio_datos	Salida	32
Pio_compara_int_0	Salida	1
Pio_compara_int_1	Salida	1
Pio_compara_int_2	Salida	1
Pio_compara_int_3	Salida	1
Pio_compara_int_4	Salida	1
Pio_captura_0	Entrada	1
Pio_captura_1	Entrada	1
Pio_captura_2	Entrada	1
Pio_captura_3	Entrada	1
Pio_captura_4	Entrada	1
Pio_in_32	Entrada	32

Las entradas de interrupción *pio_compara_int_x* y *pio_captura_x* deben ser configuradas como entradas con la siguiente configuración:

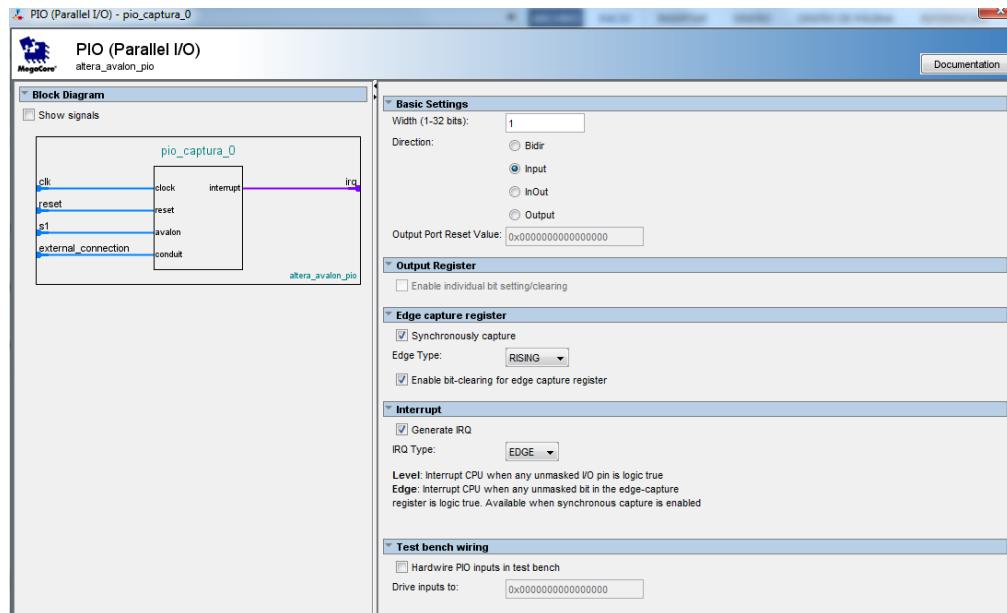


FIGURA 52: QSYS CONFIGURACIÓN PIO

3.6.3.12.- Controlador de LCD

Seleccionar en la librería de componentes **Peripherals -> Display -> Character LCD**. Nos permitirá mostrar información en modo de caracteres en la pantalla LCD que incorpora la placa de desarrollo. Una vez instanciado, renombrar como *lcd*. Este componente no necesita ser configurado.

Conectar las señales *c0*, *reset* y *data_master*, y exportar la señal “*external*” con el nombre “*lcd*”.

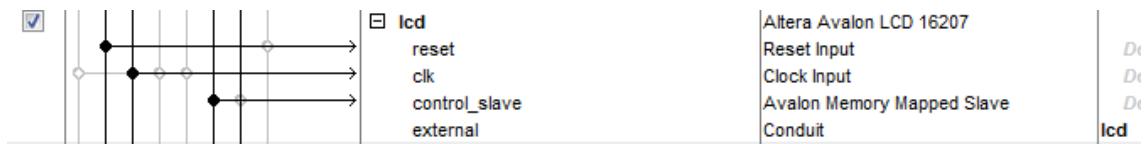


FIGURA 53: QSYS CONEXIÓN LCD

3.6.3.13.- Identificador del sistema (*system_id*)

Este componente permite identificar que el software y el hardware con el que se programa y configura el sistema son compatibles. En la biblioteca de componentes seleccionar **Peripherals -> Debug and Performance -> System ID Peripheral**. Mantener la configuración por defecto.

Conectar las señales *c0*, *reset*, *data_master* e *instruction_master*.



FIGURA 54: QSYS CONEXIÓN SISTEM_ID

3.6.3.14.- Controlador Ethernet DM9000A

El controlador Ethernet nos permitirá interactuar con el chip Ethernet para enviar y recibir datos por la red. Es un componente IP externo⁷ que se incorpora a Qsys de forma automática si guardamos la carpeta del IP *core* en el directorio del proyecto. En el directorio guardamos el conjunto de IP *cores* que necesitemos en una carpeta denominada “*ip*”:

⁷ El IP *core* del controlador Ethernet DM9000A se puede encontrar en el ejemplo DE2_NET que viene en el CD de la placa de desarrollo o en la página web de Altera.

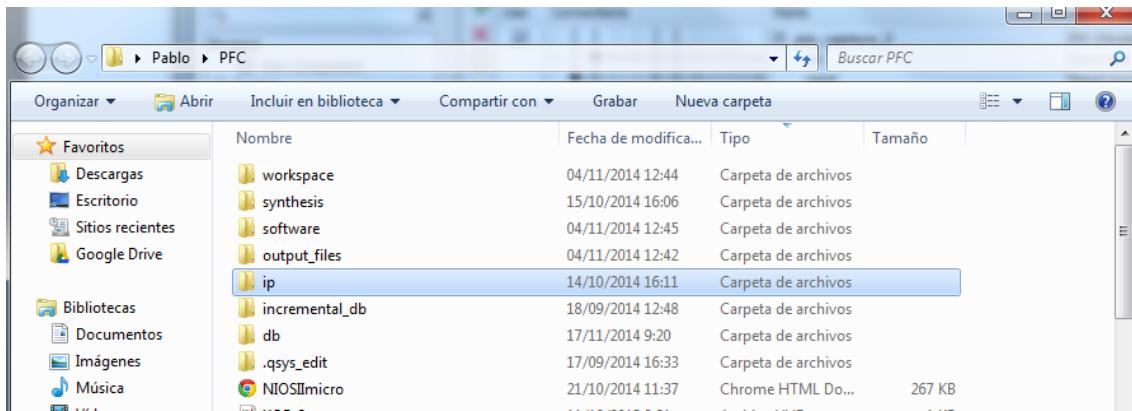


FIGURA 55: INCORPORACIÓN DE LOS IP CORES AL DIRECTORIO

Dentro de dicha carpeta tenemos nuestros *IP cores*. Tenemos disponibles dos *IP cores*: un controlador de USB maestro/esclavo (ISP1362) y el controlador Ethernet DM9000A. Este último es el que utilizaremos en el presente proyecto.

Una vez copiado el IP core a la carpeta indicada, ya debe aparecer el componente en Qsys. (si no aparece reiniciar la herramienta Qsys).

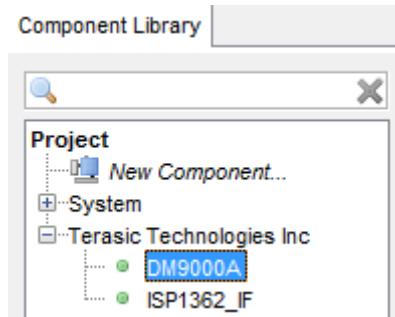


FIGURA 56: VISTA DEL DM9000A EN QSYS

Hacemos doble clic para incorporarlo al proyecto, y lo renombramos como DM9000A. No es necesario parametrizarlo.

Se conectan las señales *c0*, *reset*, y *data_master*, y se exporta la señal "*avalon_slave_0_export*" con el nombre "*dm9000a*".

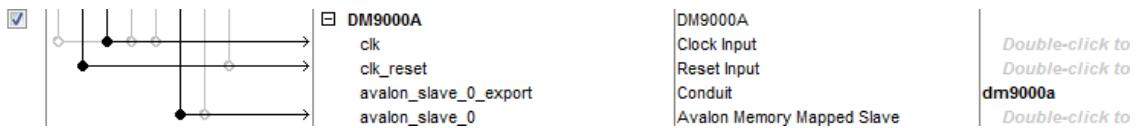


FIGURA 57: QSYS CONTROLADOR ETHERNET DM9000A

3.6.3.15.- Finalización y generación del procesador

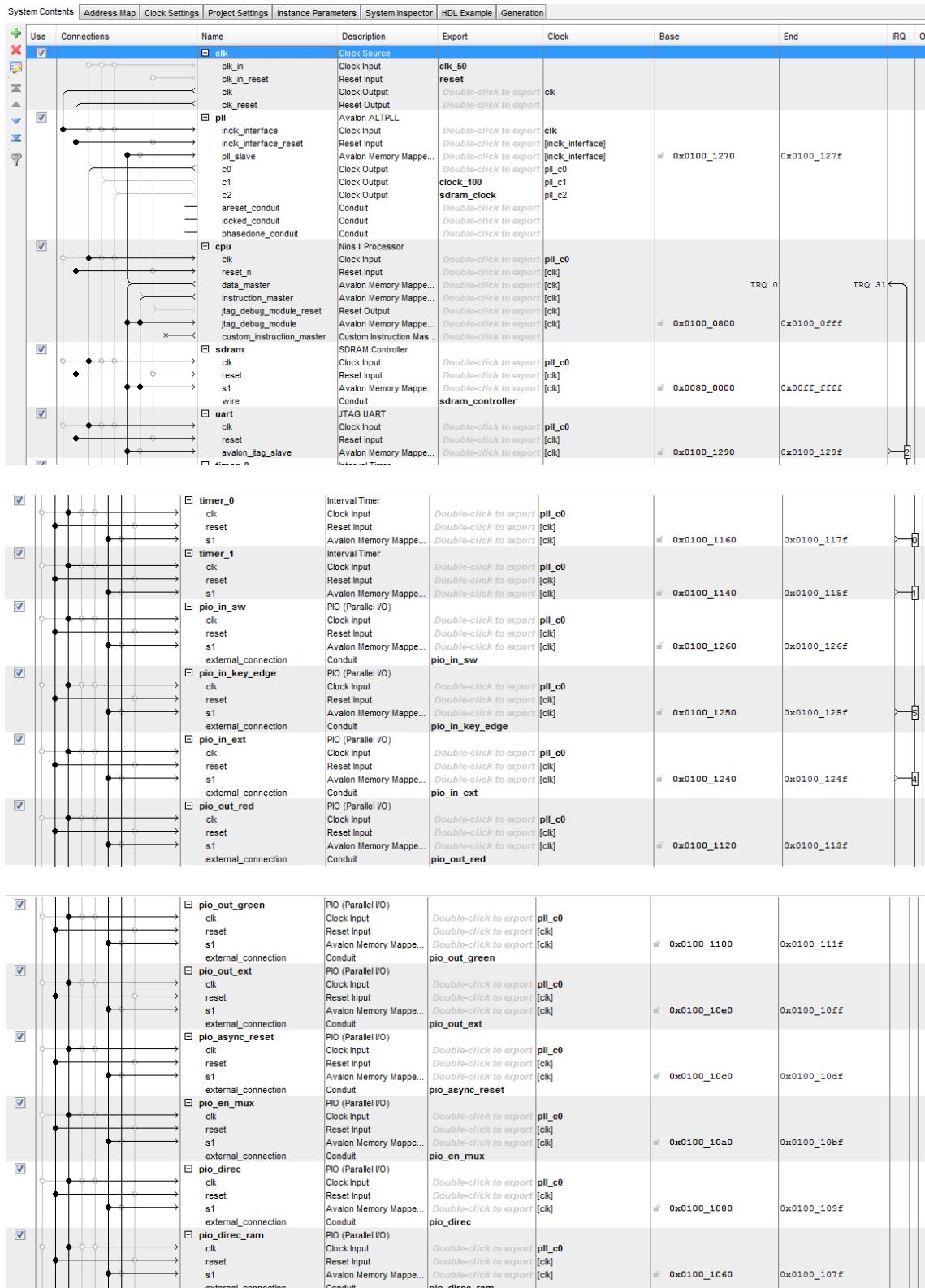
Una vez incorporados todos los componentes necesarios para nuestro sistema debemos realizar la asignación de direcciones de memoria (*mapping*) y establecer la prioridad de las interrupciones.

La asignación de direcciones base para el mapeado de memoria se realiza de forma automática seleccionando **System -> Assign Base Address**. Una vez realizada la asignación automática, en las columnas *Base* y *End* se puede ver dicha asignación. También se puede ver en la pestaña *Address Map*.

Por último debemos asignar la prioridad de interrupción en la columna *IRQ*, que será la siguiente (los elementos mostrados en gris no van a ser utilizados en el proyecto pero los vamos a incluir porque permiten conectar al procesador todas las entradas y salidas de la unidad de captura, lo que facilitará su comprensión):

Componente	Prioridad IRQ
Timer_0	0
Timer_1	1
Uart	2
Pio_captura_0	3
Pio_in_ext	4
Pio_in_key_edge	5
DM9000A	6
Pio_compara_int_1	7
Pio_compara_int_2	8
Pio_compara_int_3	9
Pio_compara_int_4	10
Pio_compara_int_0	11
Pio_captura_1	12
Pio_captura_2	13
Pio_captura_3	14
Pio_captura_4	15

Una vez asignado las prioridades de interrupción de acuerdo a la tabla anterior, tenemos el sistema completo siguiente donde se muestran todos los componentes instanciados, con su respectiva conexión, número de interrupción, etc:



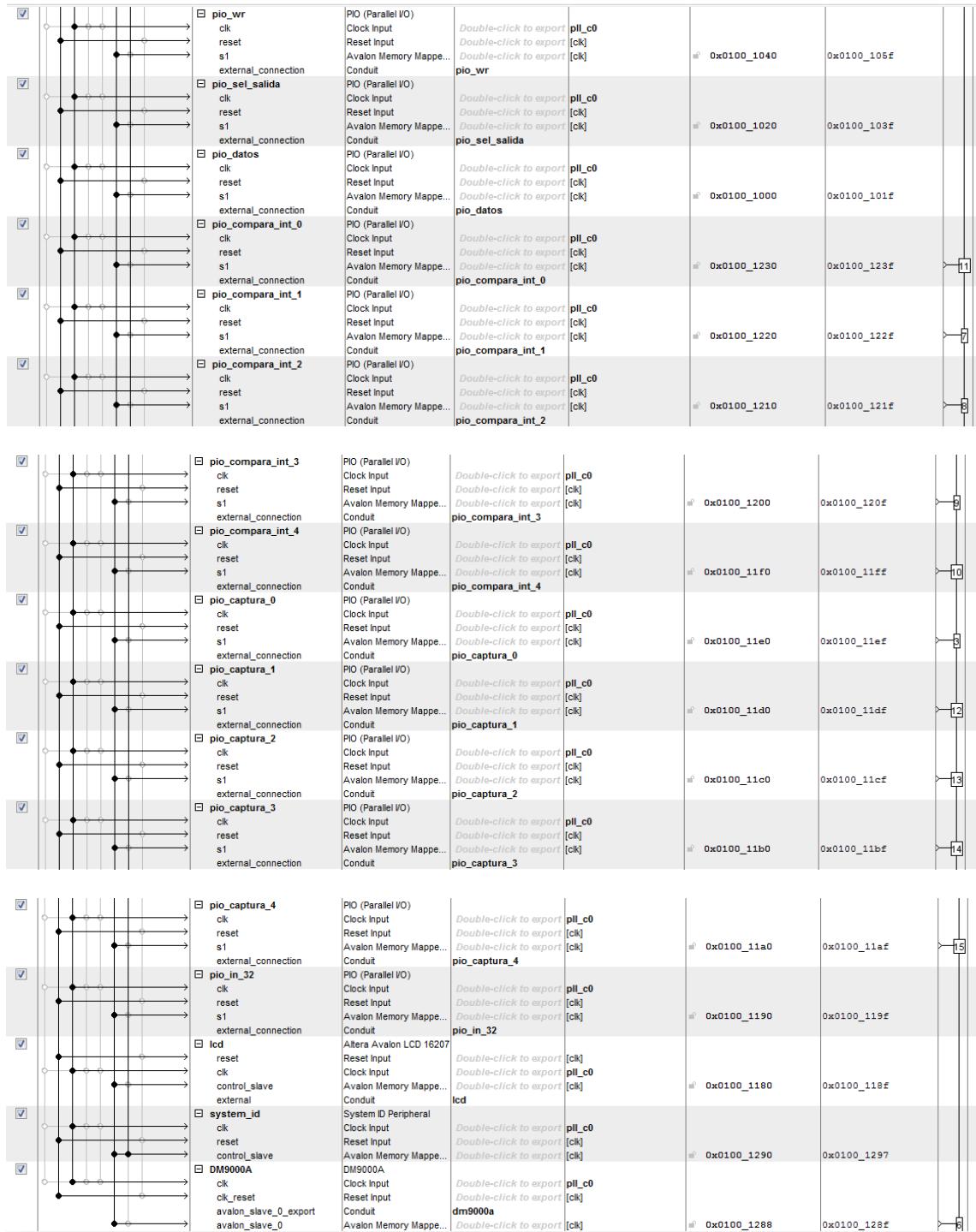


FIGURA 58: QSYS CONEXIÓN GLOBAL DE COMPONENTES

Una vez terminado el sistema anterior debemos guardarlo. Para ello hacer clic en **File -> Save** y guardarlo en el directorio de trabajo con el nombre **NIOSIIImicro.qsys**.

Ahora debemos generar el bloque procesador, que contendrá todos los bloques instanciados en Qsys. Para ello vamos a la pestaña de Qsys *Generation* y seleccionamos *Generate*.

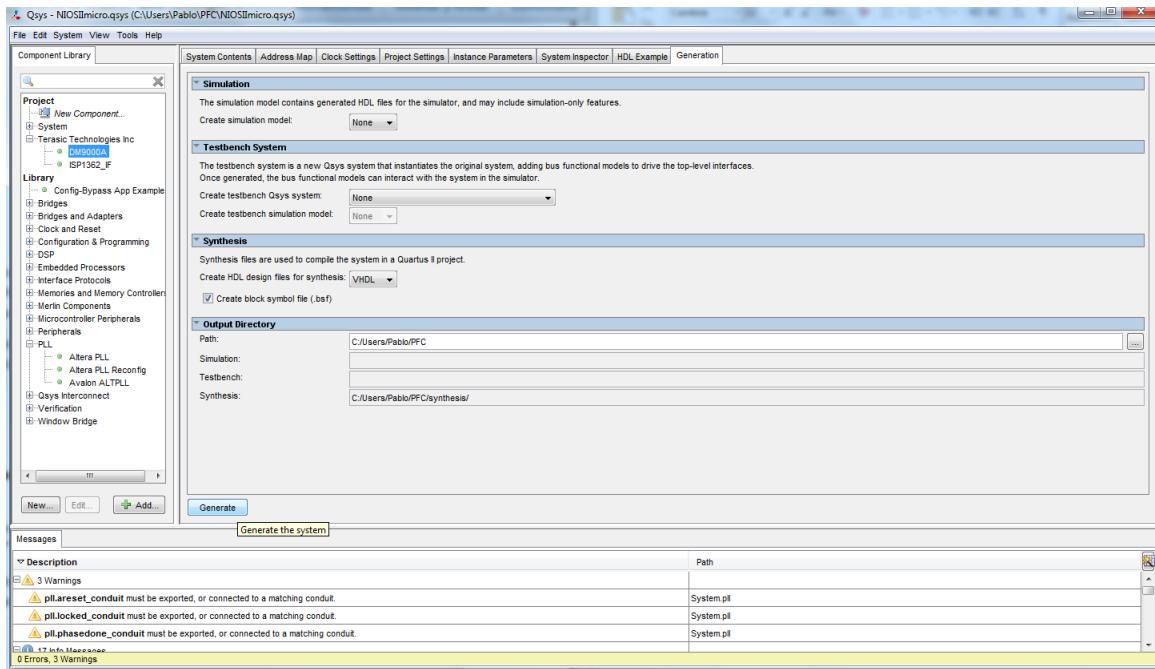


FIGURA 59: QSYS GENERATION

3.6.4.- Creación de un esquemático

Para realizar la interconexión entre el procesador y la unidad de captura es más sencillo hacerlo de una forma gráfica, para lo cual hacemos uso de los esquemáticos.

Para añadir un nuevo esquemático seleccionamos **File -> New -> Block Diagram/Schematic File**.

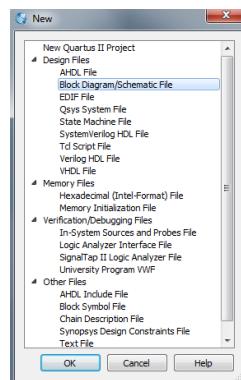


FIGURA 60: GENERACIÓN DE ESQUEMÁTICO EN QUARTUS

Para enlazar el proyecto con el esquemático simplemente debemos hacer clic en **File -> Save As** y guardar el archivo con extensión ***.bsf** en el directorio de trabajo. Se debe guardar con el mismo nombre que el proyecto al que pertenece, ya que dicho esquemático será la entidad superior del proyecto. El nombre propuesto es **NIOSIIproject.bsf**.

3.6.5.- Inserción de bloques al esquemático principal

En este apartado indicaremos cómo introducir bloques en el esquemático. Introduciremos tres bloques en total: procesador, unidad de comparación y captura, y un circuito de *reset*.

Para incluir un bloque se debe proceder de la siguiente forma:

1. Hacer doble clic en el esquemático **NIOSIIproject.bdf** que está situado en la ventana *Project Navigator* (a la izquierda de la pantalla).
2. En el esquemático que se muestra, hacer clic con el botón derecho y seleccionar **Insert -> Symbol**.

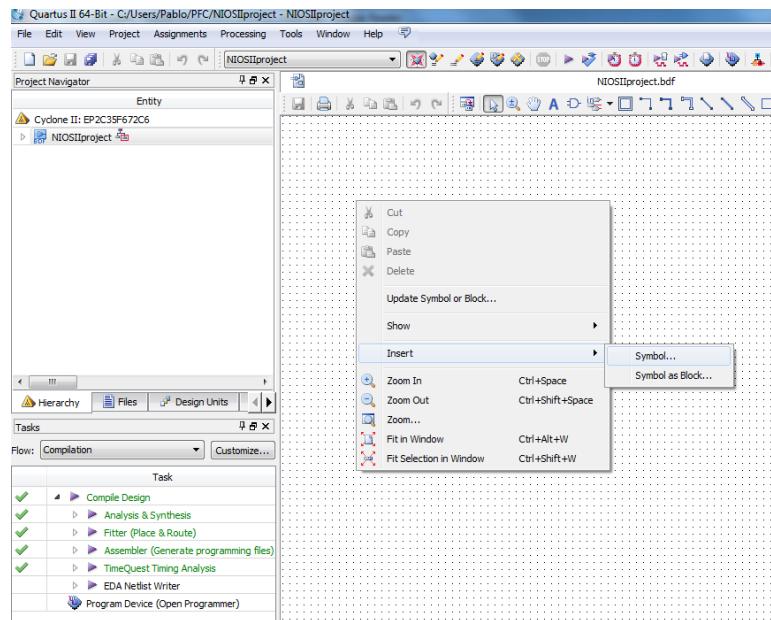


FIGURA 61: INCORPORACIÓN DE UN SÍMBOLO AL ESQUEMÁTICO

3. Por último seleccionar el símbolo requerido en cada caso.

3.6.5.1.- Procesador

Seleccionamos el símbolo del procesador, que tiene como nombre *NIOSII micro*, y lo introducimos en el esquemático.

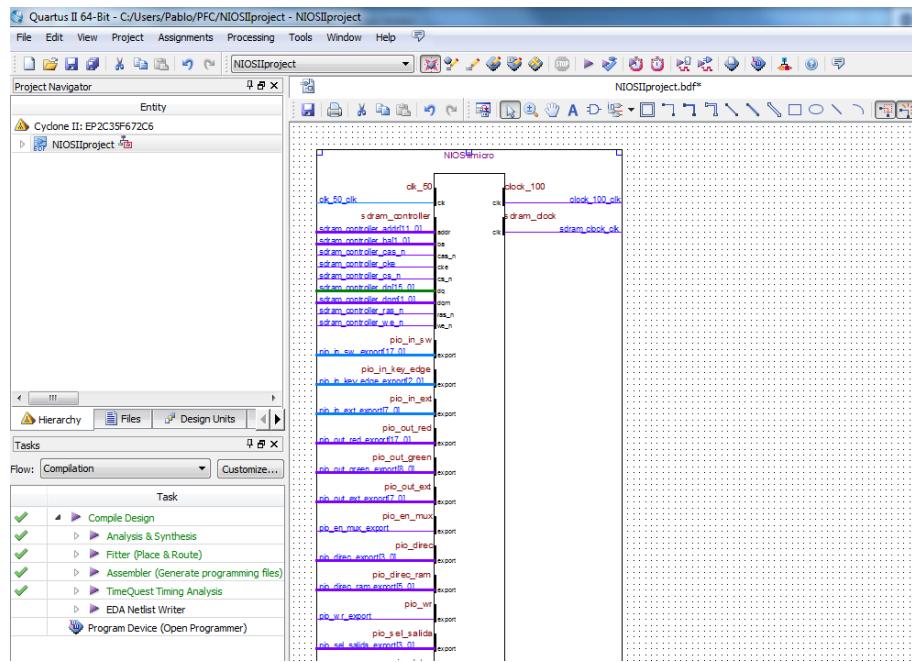


FIGURA 62: SÍMBOLO DEL PROCESADOR EN EL ESQUEMÁTICO

3.6.5.2.- Unidad de comparación y captura

Primero debemos incorporar los ficheros asociados a la unidad de comparación y captura. Para ello seleccionamos **Project -> Add/Remove Files in Project**, y seleccionamos todos los ficheros asociados a la unidad.

El procedimiento es el mismo que el caso anterior, salvo que debemos seleccionar el símbolo “*Unidad Operativa*”.

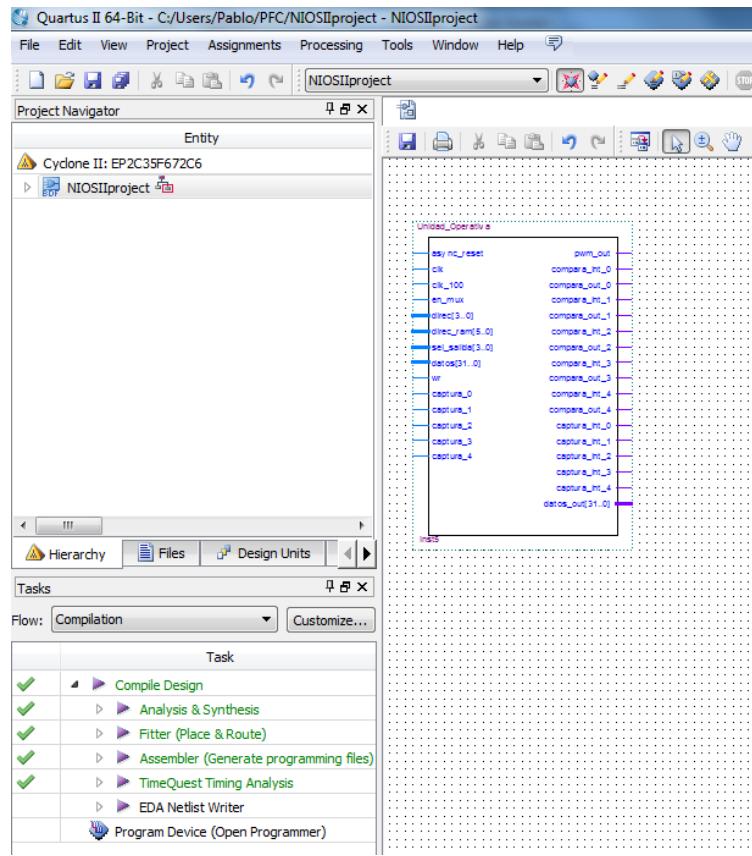


FIGURA 63: SÍMBOLO DE LA UNIDAD DE CAPTURA EN EL ESQUEMÁTICO

En el caso de que no tengamos el símbolo, pero sí los elementos que lo componen en VHDL o Verilog, lo que debemos hacer es lo siguiente:

1. Ir a la ventana *Project Navigator* y seleccionar la pestaña *Files*.
2. Hacer doble clic en el elemento de jerarquía más alta. En este caso es la *Unidad_Operativa*
3. Ir a **File -> Create/Update -> Create Symbol Files for Current File**

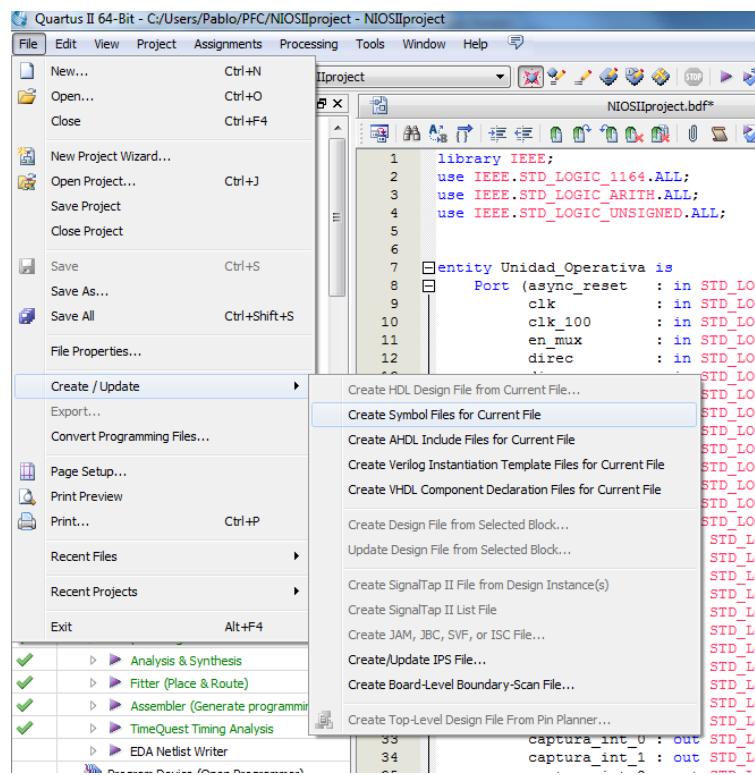


FIGURA 64: CREACIÓN DE UN SÍMBOLO A PARTIR DE HDL

4. Ahora ya podemos incluir el símbolo mediante el procedimiento descrito en 3.6.5.

3.6.5.3.- Circuito de reset

Incluimos un circuito de reset para que, cuando se introduzca el software del procesador en la placa de desarrollo se realice una reinicialización del sistema.

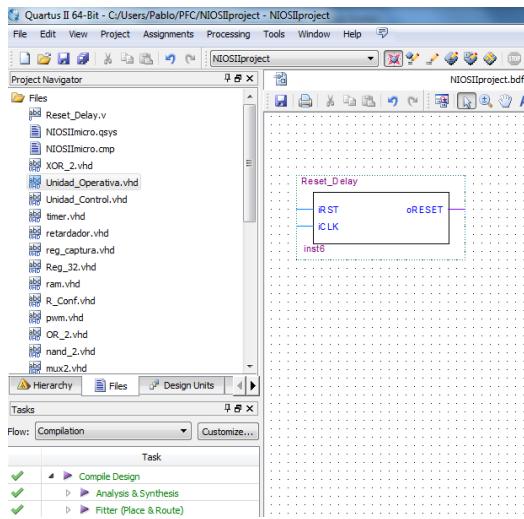


FIGURA 65: SÍMBOLO DEL CIRCUITO DE RESET

El código asociado al símbolo es el siguiente:

```

module Reset_Delay(iRST,iCLK,oRESET);
input iCLK;
input iRST;
output reg oRESET;
reg [23:0] Cont;

always@(posedge iCLK or negedge iRST)
begin
    if(!iRST)
    begin
        oRESET <= 1'b0;
        Cont <= 24'h0000000;
    end
    else
    begin
        if(Cont!=24'hFFFFFF)
        begin
            Cont <= Cont+1;
            oRESET <= 1'b0;
        end
        else
            oRESET <= 1'b1;
    end
end
endmodule

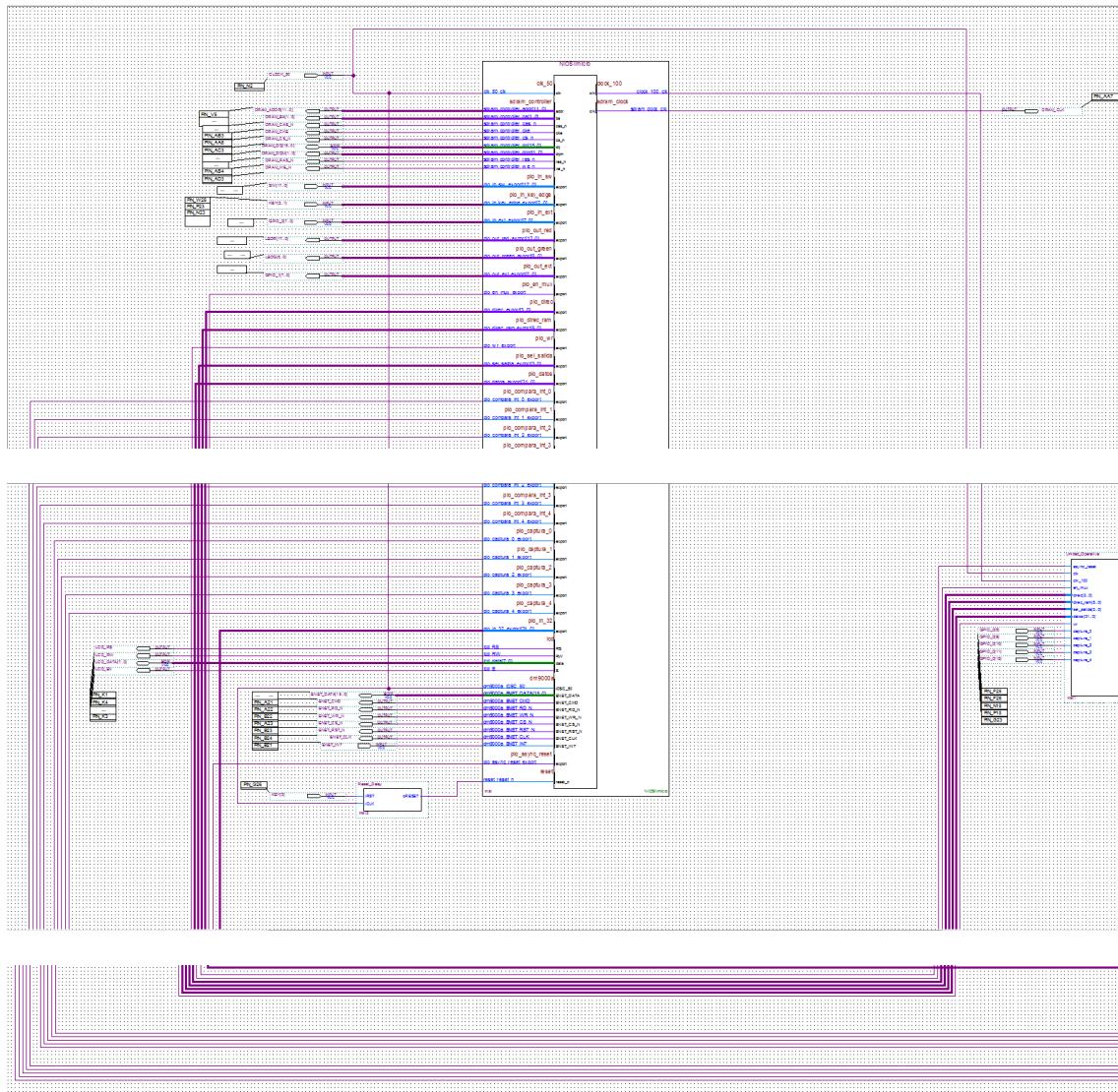
```

La entrada *iRST* irá conectada al pulsador “KEY0” de la placa, la señal *iCLK* al reloj de 50 MHz, y la señal de salida *oRESET* irá conectada a la entrada *reset_n* del procesador.

El procedimiento para incorporar dicho circuito es idéntico a los anteriores. Si el bloque no tiene símbolo asociado se debe proceder como se ha indicado en el apartado precedente.

3.6.6.- Interconexión de componentes

Este es el momento de interconectar los tres componentes. Es un procedimiento bastante tedioso, pero se recomienda hacerlo con cuidado pues un error de conexión hará que el funcionamiento posterior no sea el correcto. El conexionado quedará de la siguiente forma:



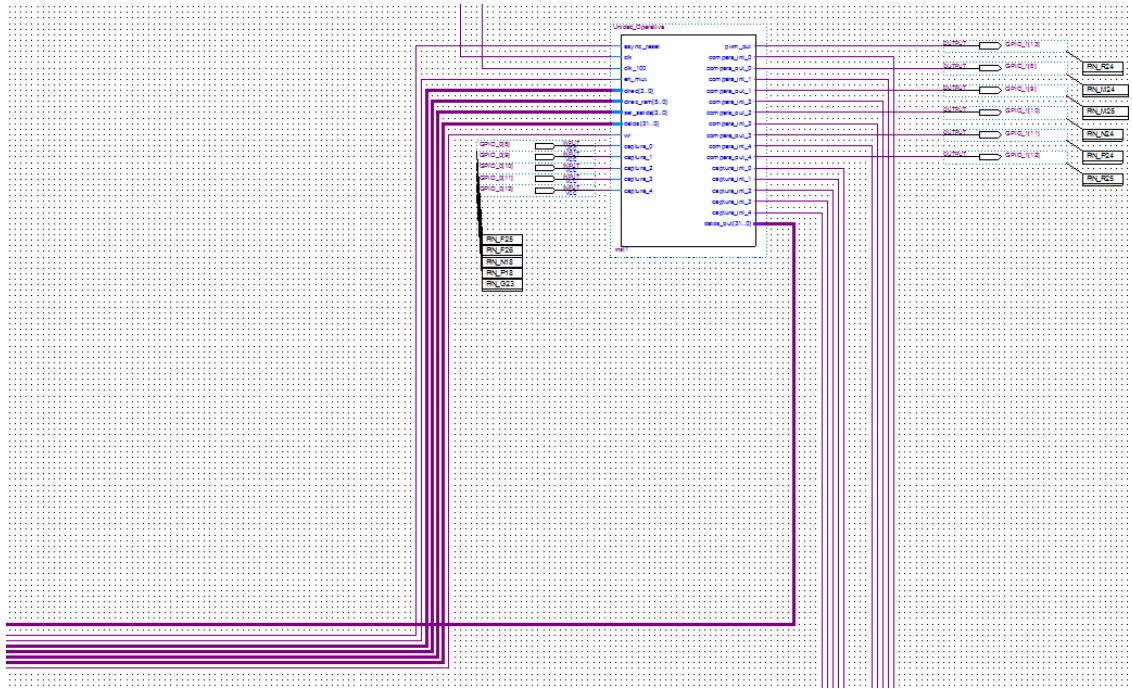


FIGURA 66: INTERCONEXIÓN DE COMPONENTES EN EL ESQUEMÁTICO

3.6.7.- Creación y asignación de pines

Ahora debemos asignar pines a aquellas entradas o salidas que han quedado libres (sin conexión) en el esquemático. En función de la señal se debe incorporar el pin adecuado (*in*, *out* o *inout*). Para saber ante qué tipo de señal se trata, hacer clic derecho sobre uno de los símbolos del esquemático y seleccionar *Properties*. Posteriormente, seleccionar la pestaña *Ports*, y ahí se podrá ver el tipo de puerto que tiene asignado cada línea.

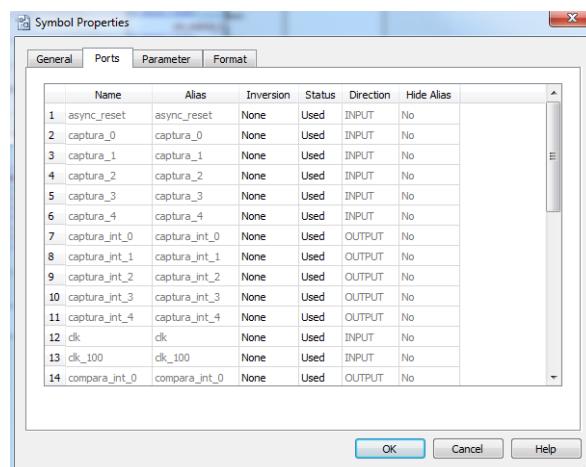


FIGURA 67: PROPIEDADES DE ASIGNACIÓN DE PINES

Se les debe asignar un nombre, que se recomienda que sea autoexplicativo de la señal a la que va conectado. Posteriormente se une el pin a la línea correspondiente.

Una vez que se tienen todos los pines con sus respectivos nombres asignados y colocados en sus respectivas posiciones, es la hora de asignar dichos pines a los pines físicos de la FPGA. Para ello hacemos clic en **Assignments -> Pin Planner**. Ahora tan solo debemos asignar cada pin a la salida física de la FPGA, para ello rellenamos la tabla de la parte inferior del *Pin Planner*. Para saber el nombre del pin físico acudimos al manual de la placa DE2 disponible en la bibliografía. Cada vez que asociemos un pin, éste se mostrará en rojo en el esquema de la zona superior.

Una vez finalizada la asignación podremos ver el siguiente resultado:

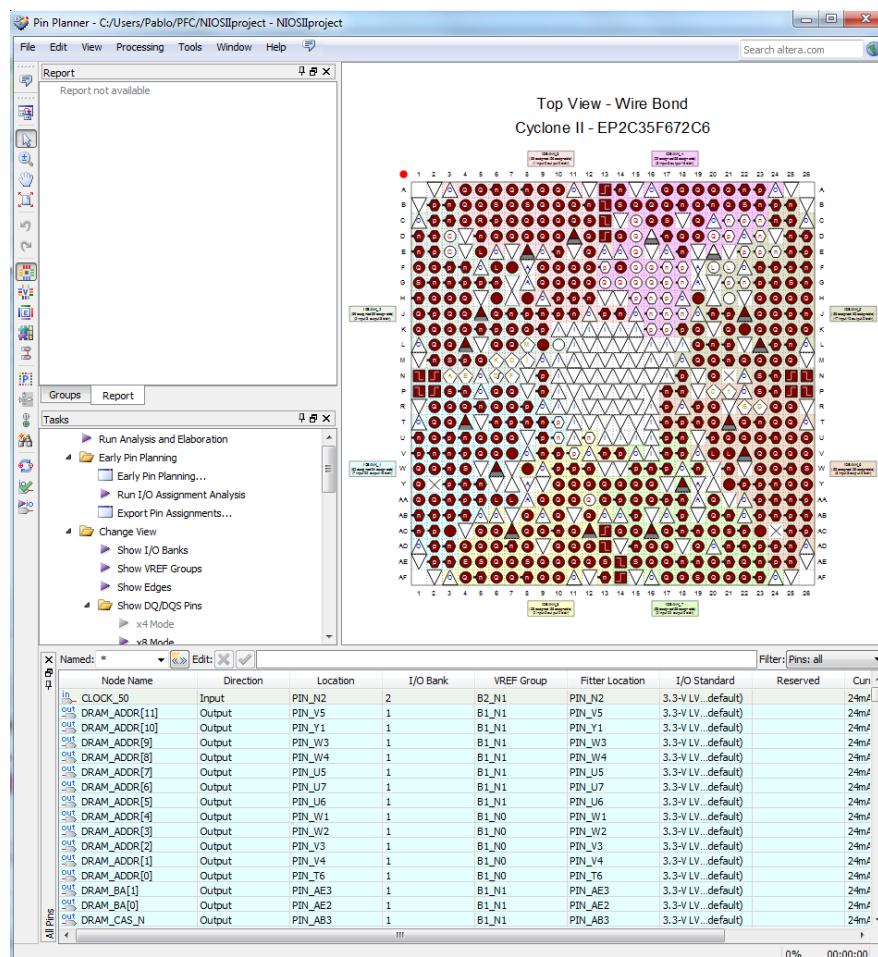


FIGURA 68: PIN PLANNER

Los pines de la placa que no se utilizan se establecerán como entradas de tercer estado. Para ello en Quartus II seleccionar **Assignments -> Device -> Device and**

Pin options -> unused pins, configurándolo como “As input tri-stated with weak pull-up”.

Si se dispone de los ficheros del presente proyecto, para agilizar todo este proceso se recomienda importar las asignaciones. Para ello, cerramos el *Pin Planner* y seleccionamos **Assignments -> Import Assignments**. En la ventana emergente que aparece seleccionamos el fichero del proyecto “PFC_PIN_assignments.csv” y hacemos clic en OK.

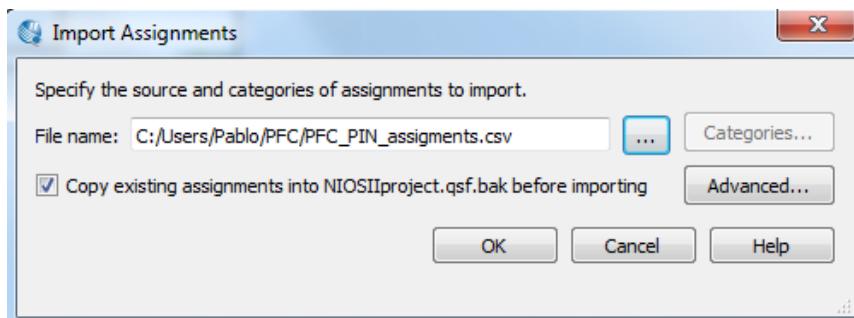


FIGURA 69: IMPORTAR ASIGNACIONES

3.6.8.- Compilación y configuración del proyecto en la FPGA

Ahora comprobaremos que no tenemos errores en el proyecto. Para tal fin hacemos doble clic (o clic derecho -> start) en *Compile Design*. Si no hay ningún error mostrará todo los progresos en color verde y al finalizar mostrará una ventana emergente indicando que la compilación ha resultado exitosa.

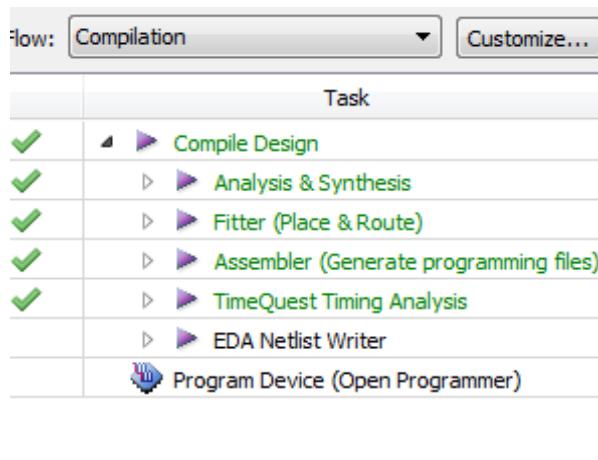


FIGURA 70: COMPILACIÓN DEL PROYECTO

Ahora es el momento de configurar la FPGA con el sistema creado. Para tal fin, se deben seguir los siguientes pasos:

1. Conectar la placa a la alimentación, y conectar el USB Blaster a uno de los puertos USB del PC.
2. Poner el conmutador de programación en la posición RUN.
3. Seleccionar **Tools -> Programmer**. Se abrirá el programador de la configuración hardware.

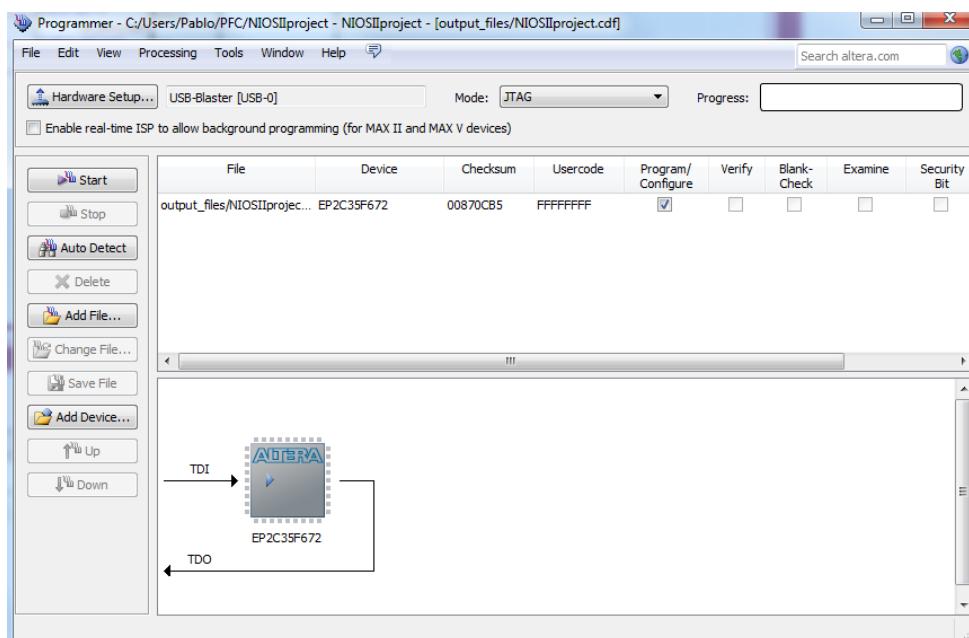


FIGURA 71: CONFIGURACIÓN DE LA FPGA

4. Seleccionar el fichero deseado (con extensión *.sof) y pulsar el botón *start*.

3.7.- Diseño del software

Una vez que se ha concluido la parte del diseño hardware es el momento de diseñar el software que permitirá controlar dicho hardware. En los siguientes apartados explicaremos cómo utilizar el entorno de programación y los pasos a seguir para generar el código, y posteriormente descargar dicho código en el procesador Nios II presente en la FPGA.

3.7.1.- Crear un proyecto con Eclipse

El software que introduciremos en el Nios II lo generaremos mediante el conocido entorno de programación Eclipse. Este entorno ya viene integrado en Quartus II con un *plugin* para el Nios II. Para arrancarlo debemos seleccionar **Tools -> Nios II Software Build for Eclipse**. Una vez seleccionado nos pedirá un directorio en donde se almacenará la información del software que generaremos:

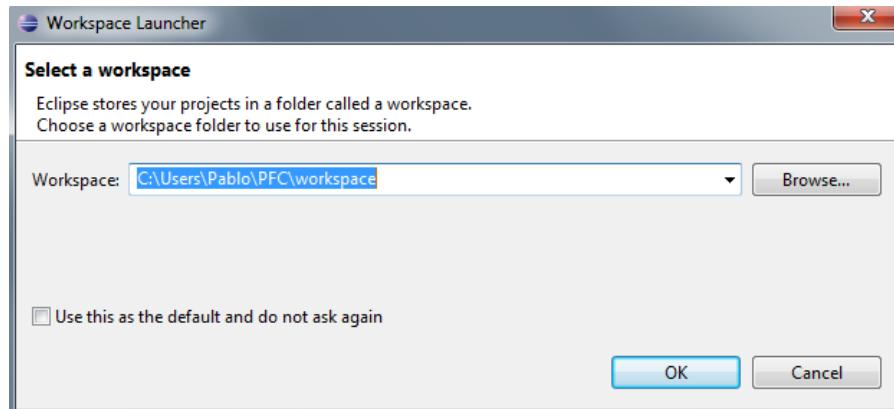


FIGURA 72: SELECCIÓN DEL WORKSPACE AL INICIAR ECLIPSE

Seleccionamos el directorio deseado (en nuestro caso creamos una carpeta de nombre *workspace* en el directorio principal de nuestro proyecto) y hacemos clic en OK. Ahora nos aparecerá la ventana principal de Eclipse.

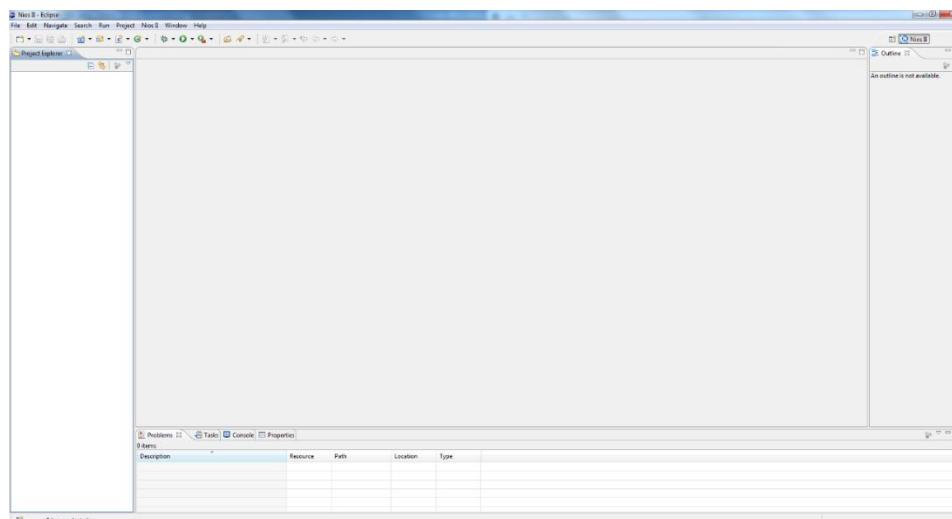


FIGURA 73: VENTANA PRINCIPAL DE ECLIPSE

Observamos cuatro ventanas. En la parte izquierda observamos una ventana denominada *Project Explorer*, donde se nos indica los ficheros que componen nuestro proyecto. En la parte derecha, vemos una ventana llamada *Outline*, en la que, una vez estemos desarrollando el software, nos mostrará un pequeño índice muy útil para acceder de forma rápida a las distintas partes del código. En la parte inferior vemos una ventana con distintas pestañas, que nos indicarán los errores, *warnings*, y otra información relevante del proyecto. Por último, la parte central es el editor de textos, que nos permitirá editar el código. El resto de opciones las iremos viendo a lo largo del siguiente apartado.

Para crear un nuevo proyecto seleccionamos **File > New -> Nios II Application and BSP for Template**. Se mostrará la siguiente ventana emergente.

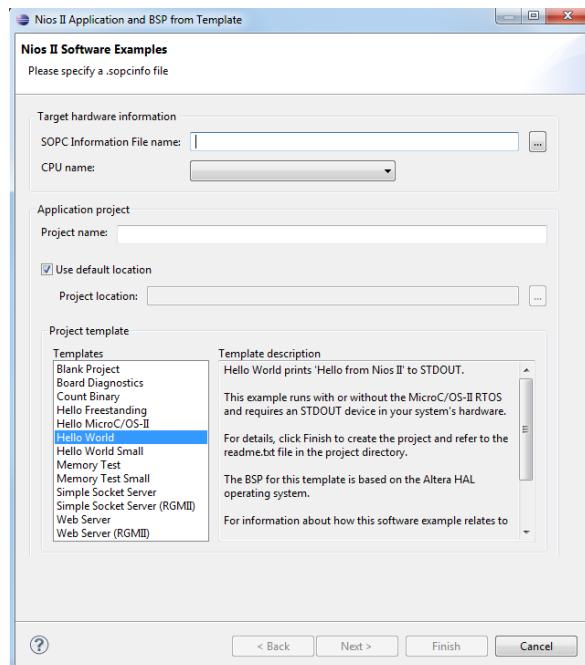


FIGURA 74: ASOCIACIÓN DEL HARDWARE EN ECLIPSE

Debemos realizar tres cosas en esta ventana:

1. En el apartado *SOPC Information File Name* se debe incorporar el fichero **.sopcinfo*, que incorpora la información necesaria sobre el hardware diseñado en Qsys.
2. En el apartado *Project Name* le asignamos el nombre *frec_gen*.

3. Por último, en *Project Template* seleccionamos *Hello Micro/OS-II*. Esto nos incluirá todas las librerías del sistema operativo para poder introducir y gestionar los hilos de ejecución.
4. Pulsar *Finish*.

Una vez hecho esto en la ventana *Project Explorer* se deben estar mostrando dos carpetas: una denominada *frec_gen* y otra *frec_gen_bsp*.

Si se expande la carpeta *frec_gen* se observan el fichero por defecto *hello_ucosii.c*, que es el código de la plantilla por defecto que hemos generado. Para verlo y editarlo hacer doble clic sobre el nombre. Se puede renombrar como *frec_gen.c* haciendo clic derecho y seleccionando *Rename* en el menú desplegable.

Si por el contrario se expande la carpeta *frec_gen_bsp*, encontraremos toda la información relevante al diseño hardware y sus librerías asociadas. Creamos los ficheros presentes en el **Apéndice 1: código fuente de la placa de desarrollo** y tendremos lo siguiente:

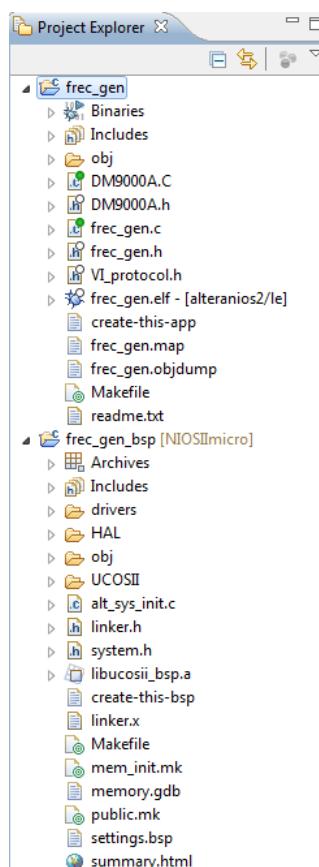


FIGURA 75: VISTA DEL PROJECT EXPLORER DE ECLIPSE

3.7.2.- Protocolo de aplicación: *VI Protocol*

3.7.2.1.- Descripción y funcionalidades

Se trata de un protocolo de aplicación hecho a medida para el proyecto que permite comunicar al PC con la placa de desarrollo e interactuar con los instrumentos disponibles en la misma. En nuestro caso, como ya hemos descrito, dichos instrumentos son dos: un frecuencímetro, y un generador de funciones.

Funciona sobre UDP, es decir, es un protocolo de aplicación que se encapsula dentro del campo de datos de los datagramas UDP. Concretamente utiliza el puerto UDP 80 como puerto en el servidor, y uno cualquiera en el cliente, que es proporcionado de forma aleatoria por el sistema operativo.

Permite realizar las siguientes tareas:

1) Acciones disponibles para el frecuencímetro:

- Activar el frecuencímetro
- Promediado frecuencial (simple, ponderado, exponencial)
- Número de nodos de promediado de frecuencia
- Solicitar la frecuencia actual

2) Acciones disponibles para el generador de funciones:

- Activar el generador de funciones
- Señal cuadrada/no cuadrada (activa PWM o PWM1+RAM)
- Modificar frecuencia (de forma automática también se modifica Ton)
- Modificar Ton (sin modificar la frecuencia)
- Modificar los 36 valores de la RAM para modificar la “señal no cuadrada”

Los campos del protocolo son los siguientes:

```
typedef struct VI_protocol
{
    int command;
    char data[36];
}VI_protocol_t;
```

Para simplificar todavía más el protocolo se ha decidido utilizar una longitud de datos fija de 36 bytes, que coincide con el tamaño de la RAM de la unidad de captura, y se corresponde con el comando de longitud más grande, que será el

comando que permita modificar los valores de la RAM. De esta forma los datagramas UDP tienen un campo de datos de 40 bytes (*command + data*). Esta decisión permite simplificar las operaciones que se realizan con los paquetes en la FPGA, ya que, por ejemplo, no hay que modificar los campos de longitud de los paquetes. Concretamente, los campos que sí se modifican en las cabeceras son los siguientes:

- **A nivel UDP:** *Source port, Destination Port, Checksum.*
- **A nivel IPv4:** *Time to live (TTL), Header Checksum, Source IP Address, Destination IP Address.*
- **A nivel Ethernet:** *Ethertype, Destination MAC Address, Source MAC Address.*

El resto de campos de las cabeceras permanecerán invariantes.

3.7.2.2.- Comandos del cliente

Son los comandos que envía el cliente UDP (el ordenador) para comunicarse con el servidor UDP (la placa de desarrollo):

Comando	Función
CMD_ACTIVATE_FREQ_METER	Habilita el frecuencímetro
CMD_FREQ_REQUEST	Solicita el envío de la frecuencia que se está midiendo
CMD_NFREQ_NODES	Cambia el número de nodos de promediado
CMD_AV_SIMPLE	Promediado frecuencial con media simple (media aritmética)
CMD_AV_WEIGHTED	Promediado frecuencial con media ponderada
CMD_AV_EXP	Promediado frecuencial con media exponencial
CMD_ACTIVATE_FUNC_GEN	Activa el generador de funciones
CMD_RAM_VALUES	Envía en el campo “data[36]” los 36 nuevos valores a introducir en la RAM

	de la unidad de comparación y captura
CMD_FG_SQUARE	Activa señal de salida cuadrada
CMD_FG_NON_SQUARE	Activa señal de salida no cuadrada; los valores de la señal de salida están en la RAM de la unidad de comparación y captura
CMD_FG_CHANGE_FREQ	Cambia la frecuencia del generador de funciones. La nueva frecuencia se envía en el campo "data[36]". De forma automática también se modifica el ciclo de trabajo.
CMD_FG_CHANGE_TON	Cambia el valor del ciclo de trabajo. El nuevo valor se envía en el campo "data[36]"
CMD_RESET	Realiza un reset. La placa queda en el estado que se ha considerado por defecto

3.7.2.3.- Comandos del servidor

Son los comandos que envía el servidor ante la recepción de un paquete que envía el cliente:

Comando	Función
CMD_FREQ_ANSWER	Si el cliente envía el comando CMD_FREQ_REQUEST el servidor le responde con este comando, junto con la frecuencia media actual en el campo "data[36]"

3.7.3.- Diagrama de funcionamiento

El siguiente diagrama muestra una imagen conceptual de cómo está diseñado el software.

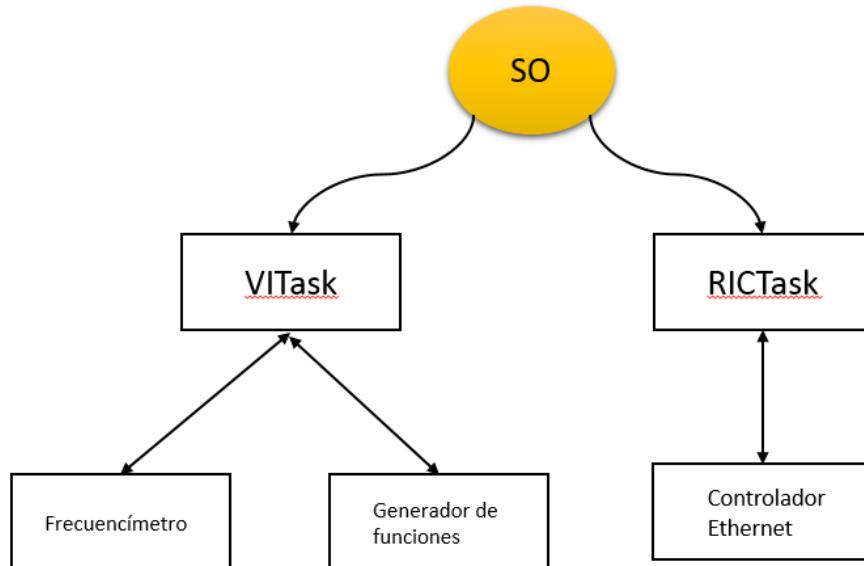


FIGURA 76: EJECUCIÓN DE TAREAS

Se diferencian dos tareas:

- **VITask** (“Virtual Instrument Task”): en función del instrumento configurado (frecuencímetro o generador de funciones en nuestro caso) se ejecuta la función asociada de cada instrumento.
- **RICTask** (“Refresh Instrument Configuration”): se actualiza la configuración del instrumento y se habilita/deshabilita el instrumento actual. En nuestro caso esta tarea controla la recepción de paquetes de la conexión Ethernet, que es a través de la interfaz por la cual se configuran los instrumentos de la placa.

Cabe decir que la tarea RICTask es más prioritaria que VITask.

3.7.4.- Instrumentos de medida

Las versiones previas del *software* que permite generar los instrumentos del proyecto (frecuencímetro y generador de funciones) han sido desarrolladas por [9]. En este apartado se describirán los cambios que se han introducido con el objetivo de mejorar sus prestaciones.

3.7.4.1.- Frecuencímetro

Se ha mejorado la precisión de este instrumento realizando los siguientes cambios:

- Retención de los valores capturados. En la versión previa, cada vez que la unidad de captura realizaba una captura completa (dos flancos de subida), se generaba la interrupción y se guardaban los valores capturados en las variables globales *captura* y *capturaant*. Posteriormente en una función dedicada al frecuencímetro se calculaba la frecuencia asociada a dicho periodo. Esta forma de proceder tiene un grave problema, y es que la interrupción que actualiza los valores *captura* y *capturaant* puede saltar en cualquier momento y puede modificar dichos valores de captura. Por este motivo hemos incluido una variable global (*captura_valida*) que funciona de forma similar a un semáforo software:
 - En la interrupción de la captura: si *captura_valida* es cero, capturamos nuevos valores y los almacenamos en *captura* y *capturaant*. Después ponemos *captura_valida* a uno.
 - Función del frecuencímetro: si *captura_valida* es uno procedemos a realizar las acciones oportunas (cálculo de la frecuencia) con los valores *captura* y *capturaant*, y al finalizar asignamos *captura_valida* a cero.
- Promediado. Se introducido un promediado de frecuencia de N nodos, siendo N configurable por el usuario, con el objetivo de reducir el error de frecuencia introducido por el generador de funciones externo. Dicho error de frecuencia se puede dar por un error puro de frecuencia, o debido al ruido aleatorio de detección del flanco (mayor a medida que aumenta la frecuencia). Las siguientes figuras muestran de una forma exagerada dicho error de detección.

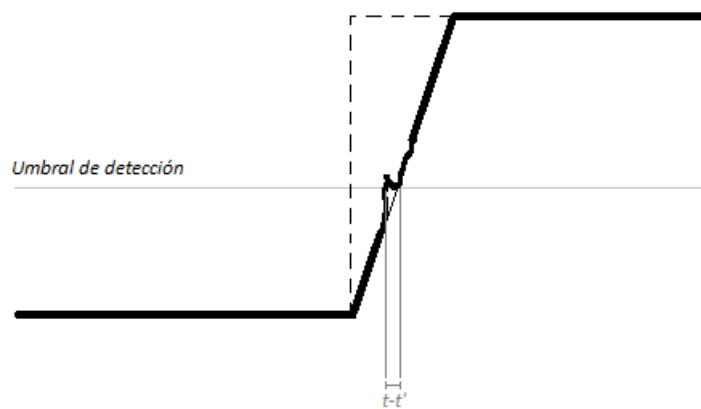


FIGURA 77: RUIDO DE DETECCIÓN DE FLANCO (1)

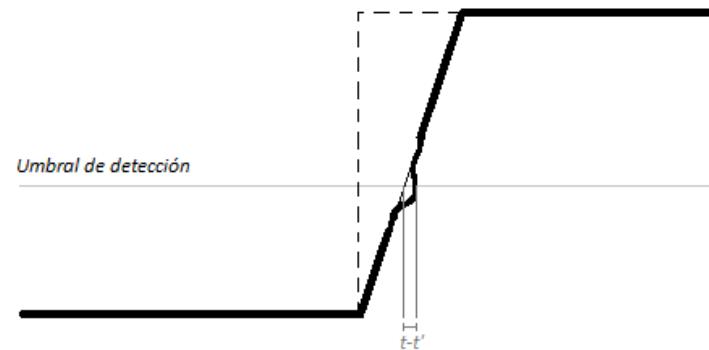


FIGURA 78: RUIDO DE DETECCIÓN DE FLANCO (2)

- Tipo de promediado. Damos la posibilidad de elegir entre tres tipos de promediado: simple, ponderado y exponencial.
 - *Simple:* media móvil simple de N nodos. Sigue una distribución uniforme y permite la reducción máxima del error de frecuencia.
 - *Ponderado:* la distribución de pesos la realizamos de una forma lineal.

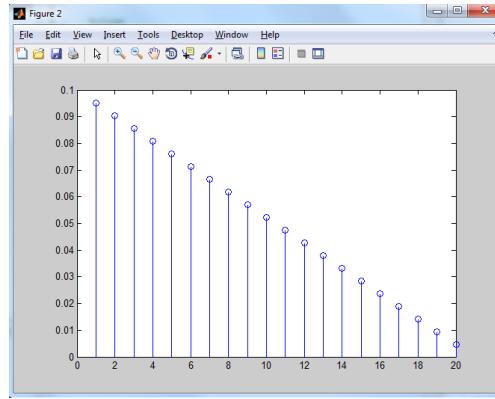


FIGURA 79: DISTRIBUCIÓN DE PESOS PONDERADA DE FORMA LINEAL (CON 20 NODOS)

- *Exponencial:* se trata de una particularización del modo ponderado donde los pesos siguen una distribución exponencial.

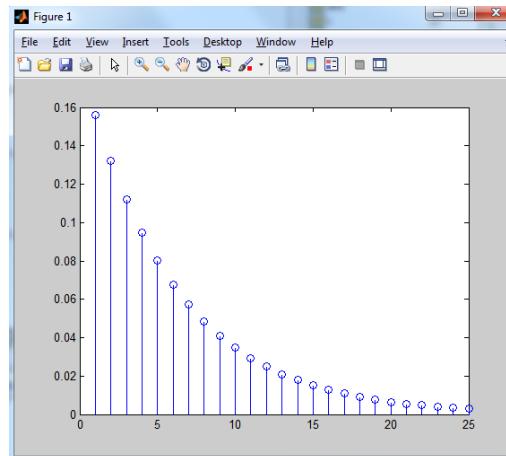


FIGURA 80: DISTRIBUCIÓN DE PESOS EXPONENCIAL (CON 25 NODOS)

3.7.4.2.- Generador de funciones

Se ha introducido la posibilidad de reconfiguración remota de los valores presentes en la memoria RAM de la unidad de comparación y captura con el objetivo de poder representar cualquier señal.

3.7.5.- Explicación del código

En este apartado explicaremos las distintas partes del código. El código se ha hecho con el objetivo de que sea lo suficientemente modular para que se pueda

ampliar a otro número de instrumentos de forma sencilla. El código es de por sí suficientemente autoexplicativo, de todas formas, en los apartados posteriores se explicará cada uno de los ficheros y se puntualizarán algunas peculiaridades que se crean oportunas.

Se divide en los siguientes ficheros: *vi_protocol.h*, *DM9000A.h*, *DM9000A.c*, *frec_gen.h*, *frec_gen.c*.

3.7.5.1.- Protocolo VI (*vi_protocol.h*)

Contiene los comandos del *VI Protocol* y la estructura de los paquetes. El código está disponible en el [Apéndice 1A: VI Protocol \(*vi_protocol.h*\)](#)

3.7.5.2.- Driver DM9000A (*.h y *.c)

Los ficheros *DM9000A.h* (cabecera) y *DM9000A.c* (código) son los ficheros que componen el driver del controlador Ethernet Davicom *DM9000A*.

El driver tiene asociadas seis funciones:

- *Iow*: rutina auxiliar de ensamblador para escribir en uno de los registros del controlador.
- *Ior*: rutina auxiliar de ensamblador para leer de uno de los registros del controlador.
- *Phy_write*: escritura en registro del controlador.
- *DM9000_init*: inicialización del controlador. Realiza la auto-negociación (configurara la velocidad y el modo de funcionamiento).
- *TransmitPacket*: transmite un paquete. Como parámetros se indican el paquete (** unsigned char*) y su respectiva longitud en bytes.
- *ReceivePacket*: se recibe un paquete. Como parámetro se indican el paquete (** unsigned char*) y su respectiva longitud en bytes. A esta función se accede en la interrupción que genera el controlador.

Los ficheros están disponibles en el [Apéndice 1B: Cabecera del Controlador Ethernet \(*DM9000A.h*\)](#) y en el [Apéndice 1C: Controlador Ethernet \(*DM9000A.c*\)](#).

3.7.5.3.- Cabecera del programa principal (*freq_gen.h*)

Se trata del fichero de cabecera de *freq_gen.c*. En él se definen estructuras, constantes, variables globales y funciones que son utilizadas en *freq_gen.c*.

El fichero completo se encuentra en el [Apéndice 1D: Cabecera del programa principal \(*freq_gen.h*\)](#). A continuación describiremos varias partes de dicho fichero para facilitar su comprensión.

Concretamente se definen:

- *Struct node*: se trata de un nodo de la lista de frecuencia. Utilizado para formar parte de una lista doblemente enlazada con un campo de datos de tipo *float* en el que se almacena una medida de frecuencia.

```
typedef struct node
{
    struct node *prev;
    float frequency;
    struct node *next;
}node_t;
```

- *Struct system_config*: es la estructura que define el comportamiento de los instrumentos en cada instante. Será modificada por la tarea *RICTask* y leída por la tarea *VITask*.

```
typedef struct system_config
{
    char activated_device;
    int num_freq_nodes;
    char average_type;
    unsigned long int per_pwm;
    unsigned long int ton_pwm;
    char square_activated;
}system_config_t;
```

- *Struct udp_header*: consta de los elementos que forman la cabecera UDP.

```
typedef struct udp_header
{
    char source_port_h;
    ..
    ..
    ..
    char data[UDP_DATA_LENGTH];

}udp_header_t;
```

- *Struct ip_header*: consta de los campos que forman la cabecera IP.

```

typedef struct ip_header
{
    char version_ihl;
    ..
    ..
    ..
    struct udp_header udp;

}ip_header_t;

```

- *Struct pkt_buffer*: estructura del paquete Ethernet con encapsulación IPv4.

```

typedef struct pkt_buffer
{
    char mac_dest5;
    ..
    ..
    ..
    struct ip_header ip;

}pkt_t;

```

- *Struct arp_header*: campos del paquete ARP (*Request y Reply*).

```

typedef struct arp_header
{
    char hardware_type_h;
    ..
    ..
    ..
    char iptarget0;

}arp_header_t;

```

- *Struct pkt_buffer_2*: estructura del paquete Ethernet con encapsulación ARP.

```

typedef struct pkt_buffer_2
{
    char mac_dest5;
    ..
    ..
    ..
    struct arp_header arp;

}pkt_buffer_2_t;

```

- Se definen todas las constantes referidas a los paquetes (Ethernet, IP, UDP, ARP), la mayor parte de las cuales son los valores por defecto (*DEFAULT_**), y las constantes referidas a los instrumentos. Estas últimas constantes van a ser utilizadas para modificar la estructura *system_config* previamente descrita.
- A continuación se definen las prioridades de las tareas. Como ya se ha dicho la tarea *RICTask* es prioritaria sobre *VITask*. En el sistema operativo MicroC/OSII una prioridad inferior es más prioritaria que una superior. Así, si una tarea tiene asignada una prioridad 7, dicha tarea será más prioritaria que otra tarea con prioridad 8, pero menos prioritaria que otra de prioridad 5.
- Por último se incluye el prototipo de todas las funciones utilizadas en *frec_gen.c*.

3.7.5.4.- Programa principal (*frec_gen.c*)

En este apartado se describirá la finalidad de cada función del código. Para un mayor detalle se remite al lector al fichero completo disponible en el [Apéndice 1E: Programa principal \(*frec_gen.c*\)](#).

- `int main`: se crean las dos tareas *VITask* y *RICTask* y se lanza el sistema operativo para que commute entre las dos tareas y se ejecuten ambas. Para ello se sigue una estrategia de reparto de tiempo de procesador basado en prioridades.
- `void VITask`: tarea que ejecuta los instrumentos en función de la configuración de la estructura *system_config*.
- `void RICTask`: tarea más prioritaria que habilita la recepción de paquetes por medio de la conexión Ethernet. La gestión de la recepción de paquetes se realiza por ráfagas, es decir:
 - 1) Se recibe una ráfaga de paquetes
 - 2) Se suspende la tarea para que la tarea *VITask* (que es menos prioritaria) se pueda ejecutar.
 - 3) Se vuelve al punto 1.

- `void lcd_init`: inicializa la pantalla de LCD de la placa de desarrollo.
- `void lcd_clear`: borra el contenido de la pantalla LCD
- `void lcd_write_first_line`: escribe en la primera línea de la LCD una cadena de caracteres que se pasa como parámetro.
- `void lcd_write_second_line`: escribe en la segunda línea de la LCD una cadena de caracteres que se pasa como parámetro.
- `struct node *create`: crea un nodo de la lista de frecuencias con el dato de frecuencia que se le pasa como parámetro.
- `void insert_beginning`: para introducir un nuevo nodo de frecuencias al principio de la lista. Se actualiza el puntero *first*, que indica el principio de la lista.
- `void insert_at_end`: para introducir un nuevo nodo de frecuencias al final de la lista. Se actualiza el puntero *last*, que indica el final de la lista.
- `void delete_beginning`: elimina el primer nodo de la lista y actualiza los punteros *first* y *last*.
- `void delete_at_end`: elimina el último nodo de la lista y actualiza los punteros *first* y *last*.
- `void delete_list`: elimina toda la lista de frecuencias y los punteros *first* y *last* se asignan a *NULL*.
- `void frequency_meter`: función que recoge los datos *captura* y *capturaant* de la interrupción que genera la unidad de comparación y captura. Si esos datos son válidos se procede del siguiente modo:
 - 1) Se calcula la frecuencia asociada a dicho período.
 - 2) Se introduce la frecuencia en la lista de frecuencias, y se calcula la media asociada en función de la distribución configurada por el usuario (simple, ponderada, o exponencial).
 - 3) Se muestra la frecuencia media en la segunda línea de la LCD.

- 4) Se reconfigura la unidad de captura para que vuelva a realizar una medida.
- `float calculate_new_freq`: calcula la frecuencia en base a los valores *captura* y *capturaant*. Dichos valores representan el período de la señal medida.
 - `float calculate_average_frequency`: introduce la frecuencia en la lista de frecuencias y se calcula la media asociada en función de la distribución configurada por el usuario:
 - Simple: la distribución de pesos sigue una distribución uniforme.
 - Ponderada: la distribución de pesos sigue una distribución cualquiera, que en nuestro caso será lineal.
 - Exponencial: la distribución de pesos sigue una distribución exponencial (es una particularización de la ponderada).
 - `void show_frequency_lcd`: se muestra la frecuencia medida en la segunda línea de la LCD.
 - `void function_generator`: si se configura el generador de funciones se ejecuta esta función, que permite configurar la unidad de comparación y captura en función de la configuración deseada por el usuario.
 - `void write_ram_function_generator`: guarda una nueva serie de valores en la RAM de la UCC. Dichos valores serán representados si se configura el generador de funciones como “señal no cuadrada”.
 - `void reset_ccp`: se provoca un reset de la unidad de comparación y captura.
 - `void configuration_ccp_captura_0`: se configura la UCC en modo captura 0 escribiendo en el registro de configuración de la UCC el valor 0x0001, y se reinicia la interrupción (se borra le *flag* de interrupción asociado).
 - `void configuration_function_generator_square`: configuración de la UCC en modo generación de señal cuadrada. Los valores de periodo (*per_pwm*) y duty cycle (*ton_pwm*) que han sido elegidos por el usuario, se guardan en los registros oportunos de la UCC.

- `void configuration_function_generator_non_square`: similar al anterior, salvo que en este caso se configura para generar una señal de tipo “no cuadrada”, de forma que a la salida se mostrarán los valores almacenados previamente en la memoria RAM del periférico.
- `void configuration_function_generator_ton`: cambia el valor del *duty cycle* de la “señal cuadrada” por el nuevo valor que ha sido configurado por el usuario (*ton_pwm*).
- `void ccp_activate_square`: habilita la señal de salida cuadrada.
- `void ccp_activate_non_square`: habilita la señal de salida no cuadrada.
- `void system_config_init`: inicialización de la estructura *system_config*. Por defecto está habilitado el frecuencímetro, la ponderación con media simple, y 20 nodos de promediado.
- `void assign_interruptions`: asigna las funciones de interrupción de los botones (*edge_button_interrupts*) y de la unidad de captura (*interrupt_captura_0*).
- `void interrupt_captura_0`: permite obtener los valores capturados (*captura* y *capturaant*), y borrar el *flag* de interrupción asociado.
- `void edge_button_interrupts`: en función del botón pulsado hacemos lo siguiente:
 - Botón 1 (KEY1): hacemos un reset de la unidad de comparación y captura.
 - Botón 2 (KEY2): mostramos la MAC de la placa en la segunda línea de la LCD.
 - Botón 3 (KEY3): mostramos la IP de la placa en la segunda línea de la LCD.
- `void ethernet_interrupts`: esta interrupción salta cada vez que recibimos un paquete. Se realiza lo siguiente:
 - Si es un paquete del protocolo *VI Protocol*, vemos el comando asociado y configuramos la placa en función de dicho comando. Si el comando implica el envío de datos al cliente enviamos un paquete de respuesta al cliente que nos ha hecho dicha petición.

- Si es un paquete ARP *Request* que solicita nuestra MAC, creamos un paquete ARP *Reply* y contestamos a dicha petición.
- `void assign_pkt_defaults`: asigna a un paquete los valores por defecto a todos los campos del paquete.
- `unsigned int calculate_ip_checksum`: calcula la redundancia de la cabecera IP y la incorpora al campo adecuado de dicha cabecera.
- `unsigned int calculate_udp_checksum`: calcula la redundancia UDP. Se trata de un cálculo opcional, y en nuestro caso no lo utilizamos.
- `int pkt_is_viprotocol`: determina si el paquete recibido es un paquete *VI Protocol*. Para determinarlo se verifican diversos campos: *Ethertype, MAC destino, IP destino, longitud, puerto UDP...*
- `int pkt_is_arprequest`: determina si el paquete recibido es un paquete ARP. Para determinarlo se verifican diversos campos: *Ethertype, MAC destino, longitud...*
- `void pkt_changes_to_send`: se realizan cambios en el paquete con el objetivo de preparar el paquete para ser enviado. Por ejemplo, se modifican: *MAC origen, MAC destino, IP origen, IP destino, puertos UDP...*
- `void pkt_arp_changes_to_send`: se realizan cambios en el paquete para prepararlo para ser enviado. Alguno de los cambios son: *MAC origen, MAC destino, código de operación ARP* (se cambia *Request* por *Reply*)...

CAPÍTULO 4: CLIENTE UDP – APLICACIÓN DE USUARIO

La aplicación de usuario permite la reconfiguración remota de los instrumentos disponibles en la placa de desarrollo. Está diseñada, al igual que la placa, con el objetivo de que sea fácilmente ampliable a otro número y tipo de instrumentos.

Se ha decidido implementarla en Python y Qt, lo que permite que sea una aplicación multiplataforma. Para desarrollar la aplicación haremos uso del entorno Python (x, y), que incluye, entre otros, los siguientes programas:

- *Spider*: es un editor de programación. Es muy útil ya que integra diversas funcionalidades (explicación de funciones, consola integrada, etc) que nos hará más sencillo el desarrollo de la aplicación.
- *Qt Designer*: esta herramienta nos permite diseñar la interfaz gráfica deseada.

4.1.- Python

4.1.1.- Instalación de Python

Python es un lenguaje de programación interpretado multi-paradigma que facilita el desarrollo rápido de aplicaciones. En este proyecto usaremos la versión 2.7.6, que se puede descargar de la página web oficial de Python⁸. Nosotros no la descargaremos de este modo puesto que ya la incluye el software que se describe a continuación.

4.1.2.- Python (x, y)

Para instalar Python (x, y) debemos acceder al siguiente sitio web: <https://code.google.com/p/pythonxy/wiki/Downloads> y descargar el programa con extensión *.exe que se indica en dicha página. Una vez finalizada la instalación ya podremos crear y ejecutar código Python.

Para abrir una consola Python basta con escribir “python” en la consola del sistema. Una vez hecho esto podemos probar a escribir código Python y ver su

⁸ <https://www.python.org/downloads/>

ejecución. Esto se denomina “consola en modo interactivo”, y será muy útil cuando no estemos seguros de cómo se ejecuta una pequeña porción de código. Dicha consola también la tendremos disponible de forma integrada en *Spider*.

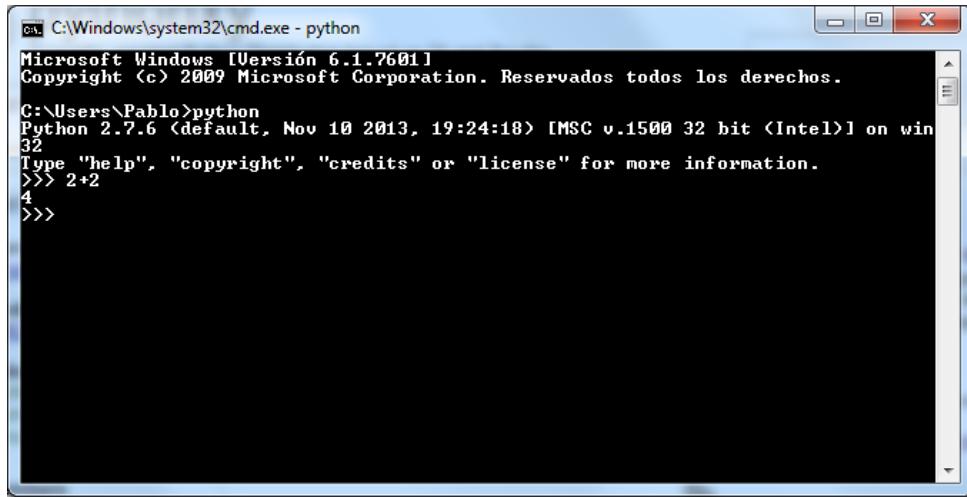


FIGURA 81: TERMINAL DE PYTHON EN LA CONSOLA DEL SISTEMA

Ahora iniciaremos el entorno Python (x, y). Para ello buscamos el programa en el menú de inicio y una vez seleccionado veremos lo siguiente:

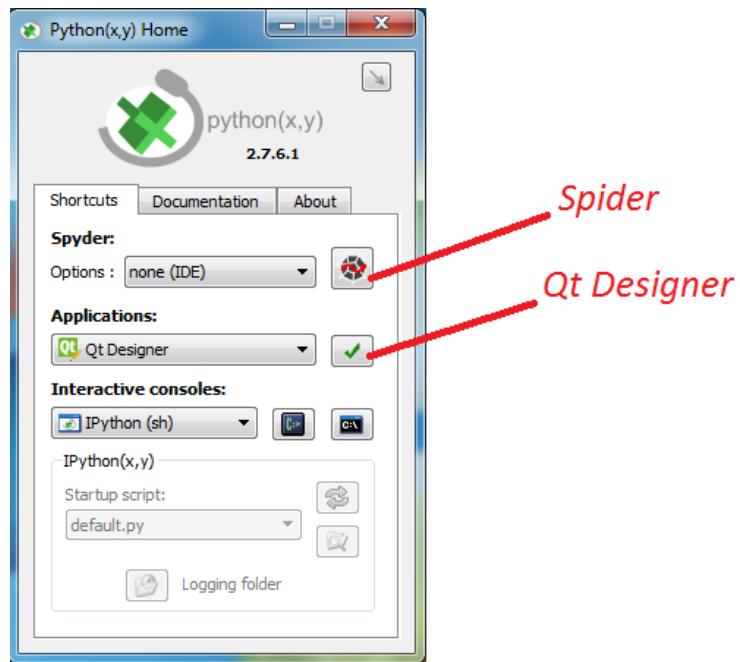


FIGURA 82: PYTHON (x, y)

Desde aquí podemos lanzar *Spider* y *Qt Designer*. Haciendo clic sobre los dos botones que se indican en la figura superior lanzamos ambas aplicaciones.

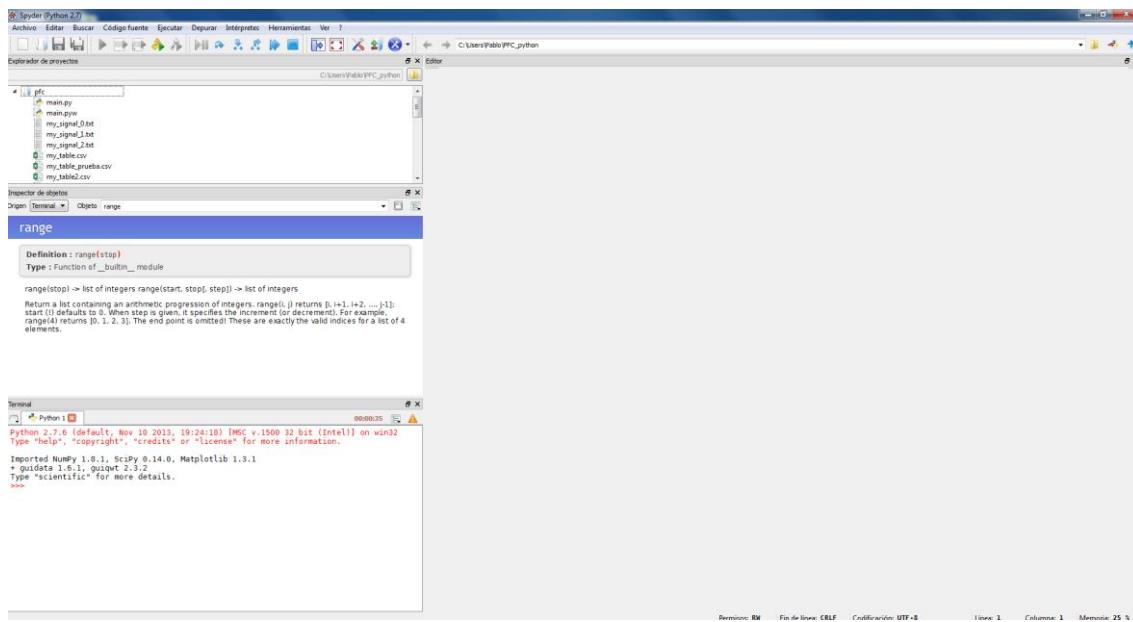


FIGURA 83: VENTANA PRINCIPAL DE SPIDER

La figura superior muestra el entorno *Spider*. Se distinguen cuatro ventanas:

- *Explorador de proyectos*: indica el directorio de trabajo y sus elementos (ficheros, carpetas...).
- *Inspector de objetos*: nos muestra información relevante de funciones que están implementadas. Como por ejemplo sus argumentos necesarios, datos de retorno, etc.
- *Terminal*: terminal de Python integrada.
- *Editor*: en la parte central de la ventana vemos el editor en el que escribiremos nuestro código Python.

Para crear un nuevo proyecto seleccionar **Archivo -> Nuevo Proyecto**, introducir un nombre y hacer clic en Aceptar. Ahora ya podemos desarrollar nuestros ficheros en Python. Estos ficheros de Python pueden tener dos tipos de extensión: ***.py**, si tiene la consola activada, y ***.pyw** si la consola no está activada.

Se recomienda, en caso de no estar familiarizado con el entorno, navegar por los distintos menús de la aplicación para comprobar las distintas opciones que hay disponibles.

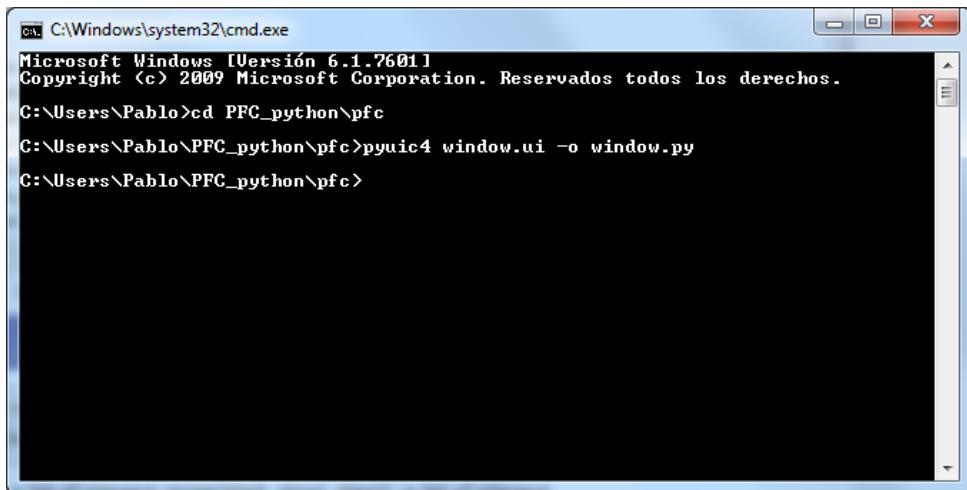
4.2.- Qt y PyQt

Qt Designer nos permite diseñar el entorno de ventanas de nuestra aplicación. Tiene la ventaja de ser multiplataforma y multilenguaje, de forma que la interfaz gráfica diseñada pueda ser ejecutada haciendo uso de muchos lenguajes de programación. La interfaz gráfica diseñada en *Qt Designer* se guarda en un fichero xml con extensión **.ui*, que posteriormente convertiremos a código Python mediante un *binding*. El *binding* utilizado en este proyecto es PyQt (versión 4), que está incluido en Python(x,y). También existen otros *bindings* como por ejemplo PySide, pero no disponen de tanto soporte ni librerías como PyQt, y por este motivo han sido descartados para realizar este proyecto.

Para realizar el *binding* abrimos una terminal del sistema, nos situamos en el directorio de trabajo y ejecutamos lo siguiente:

```
>> pyuic4 *.ui -o *.py
```

Donde el asterisco se deberá sustituir por el nombre del fichero xml.

A screenshot of a Windows Command Prompt window titled "cmd C:\Windows\system32\cmd.exe". The window shows the following text:

```
Microsoft Windows [Versión 6.1.7601]
Copyright © 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\Pablo>cd PFC_python\pfc
C:\Users\Pablo\PFC_python\pfc>pyuic4 window.ui -o window.py
C:\Users\Pablo\PFC_python\pfc>
```

The window has a standard Windows title bar with minimize, maximize, and close buttons. The text is in black on a white background.

FIGURA 84: EJECUCIÓN DEL BINDING CON PYQT4

4.3.- Creación de la interfaz gráfica con *Qt Designer*

4.3.1.- Crear un nuevo proyecto

Una vez abierto *Qt Designer* veremos la pantalla que se muestra a continuación. En ella, en la pestaña “*templates\forms*” seleccionamos *Main Window* y pulsamos *Create*. Una vez hecho esto guardamos el proyecto con el nombre *window.ui*.

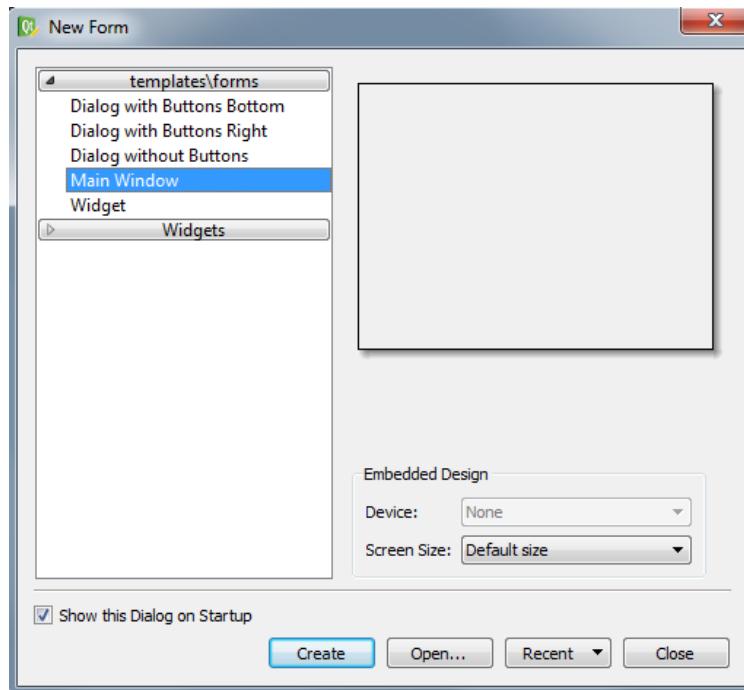


FIGURA 85: NUEVO PROYECTO EN QT DESIGNER

En este momento la pantalla de *Qt Designer* tendrá el siguiente aspecto:

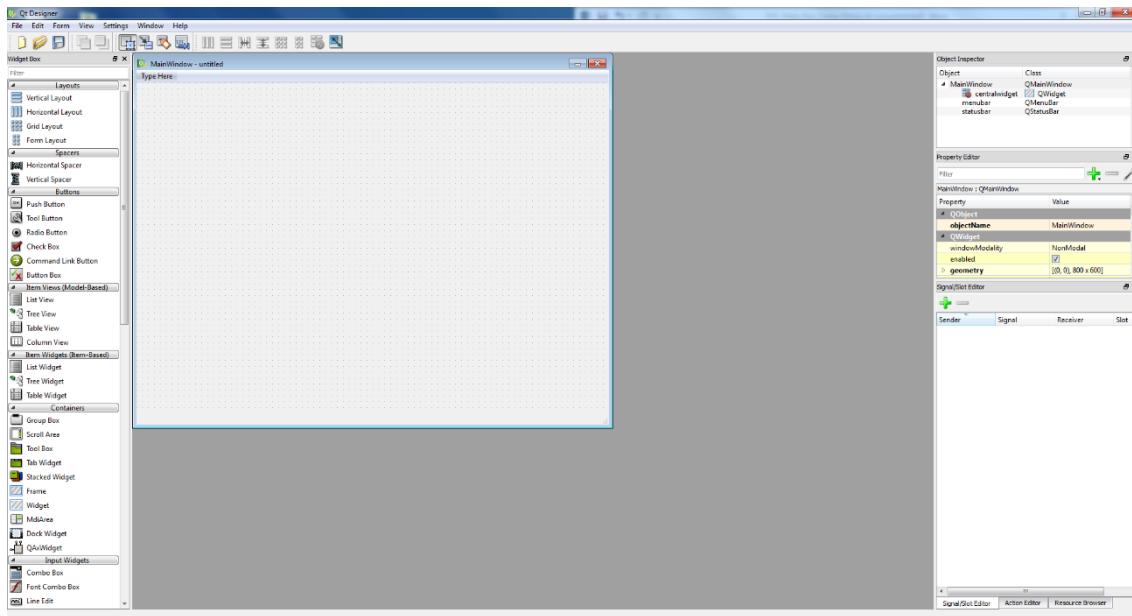


FIGURA 86: VENTANA PRINCIPAL DE QT DESIGNER

Se pueden observar las siguientes ventanas integradas:

- *Widget Box*: contiene los widgets que se pueden incorporar a la interfaz gráfica.

- *MainWindow*: será nuestra interfaz gráfica. En ella incorporaremos los widgets que conformarán la interfaz. Para mostrar la interfaz “real” que verá el usuario presionar CRTL + R.
- *Object Inspector*: nos muestra todos los objetos que hemos introducido en la interfaz.
- *Property Editor*: muestra las propiedades de cada objeto. Se pueden modificar para conseguir la funcionalidad deseada de cada widget.
- *Signal/Slot Editor*: nos mostrará las conexiones entre los componentes de la interfaz gráfica⁹.

Cabe destacar el uso de los siguientes botones disponibles en la barra de menú:



- *Edit Widgets*: permite introducir widgets y modificar sus propiedades.
- *Edit Signals/Slots*: permite describir acciones a realizar cuando el usuario interaccione con la interfaz.
- *Edit Buddies*: permite asociar elementos entre sí. Por ejemplo una etiqueta con un campo de datos.
- *Edit Tab Order*: nos permite indicar el orden de secuencia a seguir cuando el usuario pulse el tabulador del teclado.

Se recomienda al lector que navegue por los menús para identificar todas las opciones que están disponibles y se familiarice con el nombre de los widgets. También se recomienda encarecidamente que, si el lector tiene una carencia de conocimientos de Python y Qt, acuda a los libros presentes en la bibliografía, especialmente [16].

4.3.2.- Inclusión de widgets

Partimos de la interfaz en blanco como se mostraba en la figura anterior.

1. En la ventana *Property Editor* cambiar el nombre del objeto por *MainWindow*, y en la propiedad “window title” introducir *Virtual Instrument*.

⁹ *Signal*: evento específico producido por la interacción del usuario con algún elemento de la interfaz gráfica.

Slot: característica de un elemento que se ve modificada por la acción de una señal.

2. Incluir tres *Dock Widget* y cambiar el nombre por *Connection*, *Virtual Instrument* y *Error/Warning/Info*. En la ventana de propiedades deseleccionar la opción “DockWidgetClosable”.



FIGURA 87: INTERFAZ GRÁFICA AL INTRODUCIR LOS DOCK WIDGETS

3. Incluir el widget *Frame* a la parte central y aplicar un layout horizontal.
4. En la ventana *Connection* incluir los widgets:
 - *Label* y *line edit*. Renombrar el *line edit* como *textserverip*. En las propiedades establecer por defecto la IP “172.19.5.91”. Cambiar el contenido de la etiqueta por “UDP Server IP:”.
 - *Label* y *Spin Box*. Renombrar el *Spin Box* como *numserverport*. En las propiedades establecer por defecto el número 80. Cambiar el contenido de la etiqueta por “UDP Server Port:”.
 - Dos botones de tipo *Push Button*. Renombrarlos como *btnconnect* y *btndisconnect*, y cambiar el contenido de los botones por “Connect” y “Disconnect”.

A cada pareja aplicar un layout de tipo *Form*, y aplicar un layout global a la ventana *Connection* de tipo vertical. De esta forma obtendremos lo siguiente:

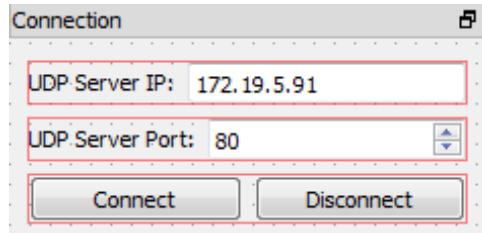


FIGURA 88: VENTANA CONNECTION

5. En la ventana *Error/Warning/Info* incluir:
 - *Text Browser*. Renombrarlo como *textbrowser*, y en las propiedades activar la barra de desplazamiento vertical.
 Aplicar un *layout* horizontal a dicha ventana.
6. En la ventana central incluir un *Stacked Widget* y aplicar un layout horizontal. Haciendo clic derecho podemos modificar el número de páginas asociadas al widget, que en este caso serán dos.

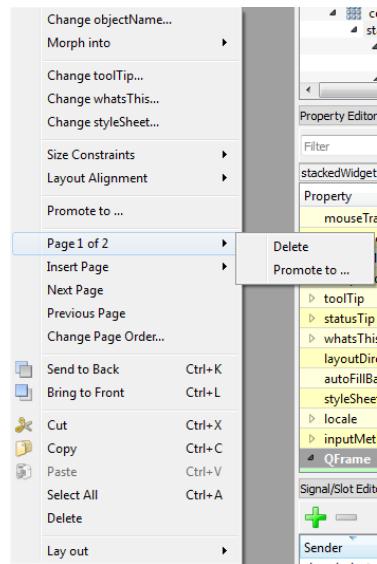


FIGURA 89: PÁGINAS DEL STACKED WIDGET

En la primera página incluimos un *Table Widget*. En las propiedades renombrarlo como *tablefrequency*, y activar la barra de desplazamiento vertical. Aplicar un layout horizontal y posteriormente hacer doble clic. Nos saldrá una tabla de propiedades que debemos llenar de la siguiente forma para indicar el nombre de cada una de las columnas.

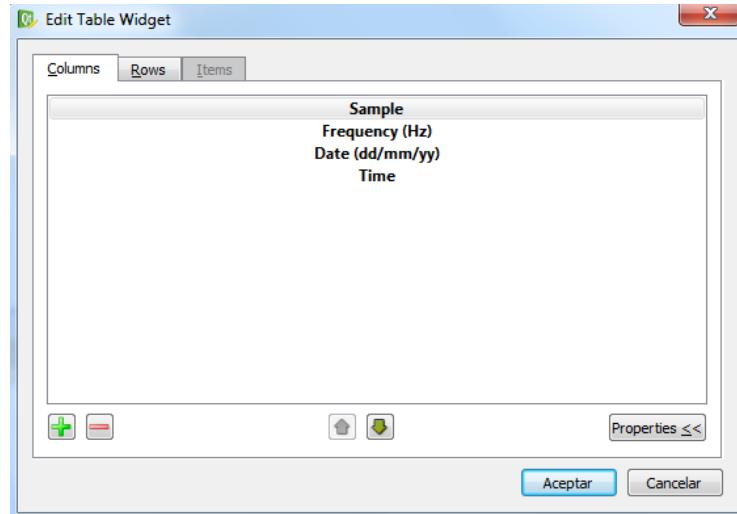


FIGURA 90: CONFIGURACIÓN DEL TABLE WIDGET

En la segunda página incluir un *QwtPlot* y renombrarlo como *graphsignal*.

7. En la ventana *Virtual Instrument* introducir:

- *Label* y *QComboBox*. Renombrar el *QComboBox* como *cboxviselect*, e introducir en la etiqueta el texto “Instrument:”. Asignar a esta pareja de elementos un *Form Layout*.
- Hacer doble clic en el *QComboBox* e introducir la siguiente configuración:

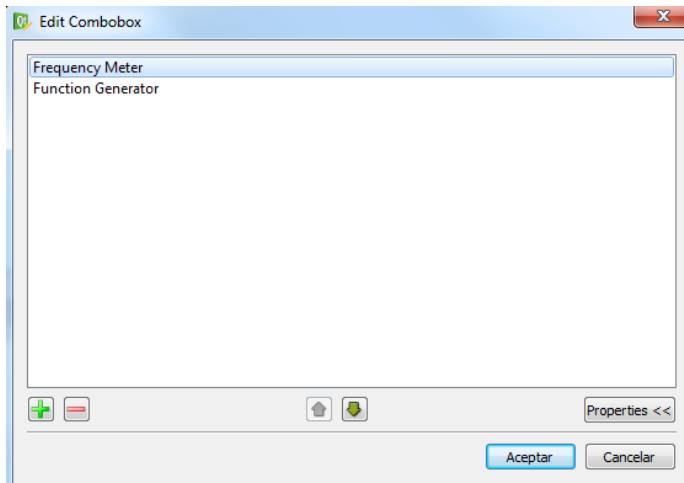


FIGURA 91: CONFIGURACIÓN DE QCOMBOBOX 1

- Debajo de lo anterior introducir un *Stacked Widget* con dos páginas.
- Debajo de lo anterior introducir un *verticalSpacer*.

Una vez realizado lo anterior, en la ventana *Virtual Instrument* debemos ver lo siguiente.

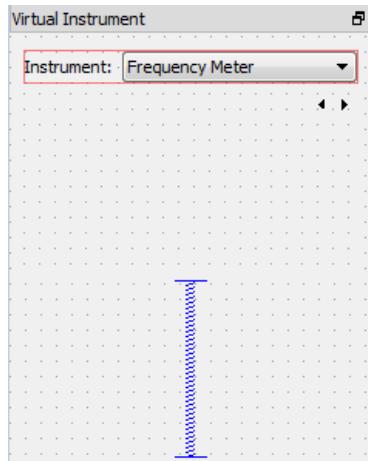


FIGURA 92: VENTANA VIRTUAL INSTRUMENT (1)

8. Introduciremos los siguientes elementos relacionados con el frecuencímetro en la primera página del *Stacked Widget* de la ventana *Virtual Instrument*:
 - *Label* y *Spin Box*. Renombrar el *Spin Box* como *numaveragenodes* y poner 20 como valor por defecto. Por último cambiar el contenido de la etiqueta por “Average Nodes:”.
 - *Label* y *QComboBox*. Cambiar el contenido de la etiqueta por “Average Type:” y renombrar el *QComboBox* por *cboxaveragetype*. Posteriormente hacer doble clic en el *QComboBox* e incluir la siguiente configuración:

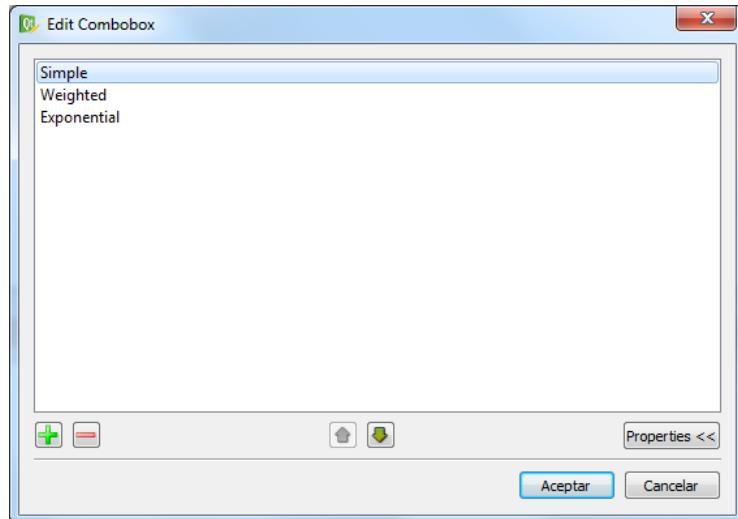


FIGURA 93 CONFIGURACIÓN DE QCOMBOBOX 2

- *Push Button*. Lo renombramos como *btnconfigfm*, y como contenido del botón introducimos “Configure Frequency Meter”.
- *QCheckBox*. Lo renombramos como *checkboxfrecreq*, e introducimos como texto “*Auto Request*”.
- *Label* y *Spin Box*. La etiqueta la renombramos como *label_samples* y el *Spin Box* como *numsamples*. En el texto de la etiqueta introducimos “*Samples:*”.
- *Label* y *Double Spin Box*. La etiqueta la renombramos como *label_interval* y el *Double Spin Box* como *dintervalsamples*. En las propiedades establecemos sus valores por defecto (10 y 0,200 respectivamente), el valor mínimo y el máximo. Por último en el texto de la etiqueta introducimos “*Interval (seconds):*”.
- *Push Button*. Lo renombramos como *btncfrequest*, y como contenido del botón introducimos “Frequency Request”.

Aplicar a cada pareja de etiqueta-elemento un *Form layout*. Una vez finalizado la configuración anterior debemos obtener lo siguiente:

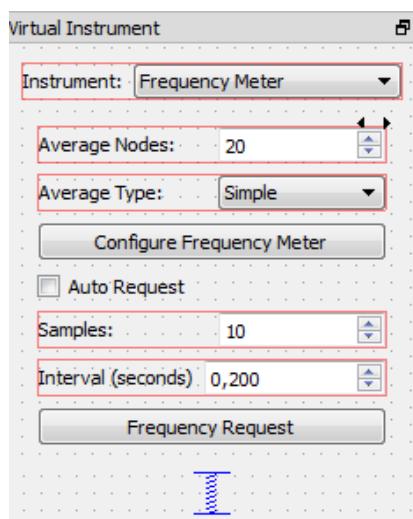


FIGURA 94: VENTANA VIRTUAL INSTRUMENT (2)

9. Introducimos los siguientes elementos relacionados con el generador de funciones en la segunda página del *Stacked Widget* de la ventana *Virtual Instrument*:
 - *Label* y *QComboBox*. Renombrar el *QComboBox* como *cboxsignaltype*. En la etiqueta introducimos el texto “*Signal Type:*”. Posteriormente hacemos doble clic en el *QComboBox* e introducimos la siguiente configuración:

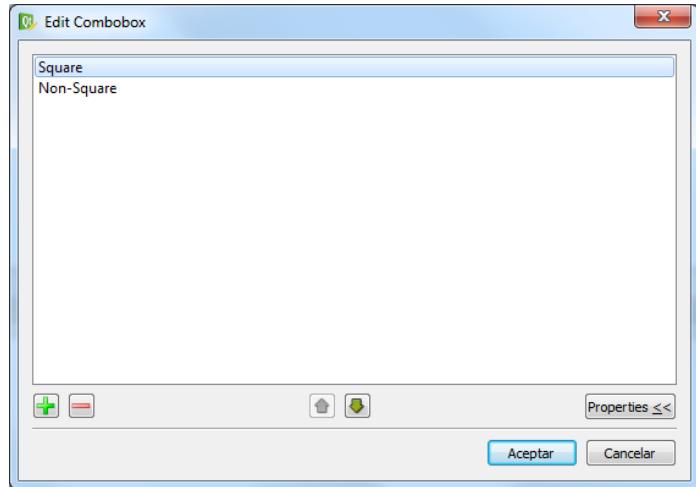


FIGURA 95: CONFIGURACIÓN DE QCOMBOBOX 3

- *Label y Double Spin Box.* Renombrar el *Double Spin Box* como *dnumfrequency*. Cambiamos el texto de la etiqueta por “*Frequency (Hz):*”.
- *Label y Spin Box.* Renombrar la etiqueta como *labeldutycycle* y el *Spin Box* como *numdutycycle*. Cambiamos el texto de la etiqueta por “*Duty cycle (%):*”.
- *Push Button.* Renombrarlo como *btnconfigfg*, y cambiar el contenido del botón por “*Configure Function Generator*”.

Aplicar a cada pareja de elementos un layout de tipo Form, y al *Stacked Widget* un layout de tipo vertical. Se debe obtener lo siguiente:

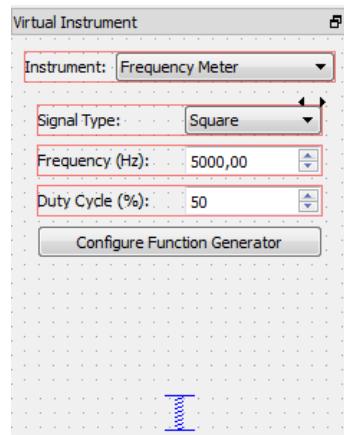


FIGURA 96: VENTANA VIRTUAL INSTRUMENT (3)

10. En la barra de menú introducimos las siguientes tres entradas: “*Load Signal*”, “*Save Table*” y “*Clear Table*”. Renombramos los objetos por *menufileloadsignal*, *menufilesavetable*, y *menufileclearable* respectivamente.

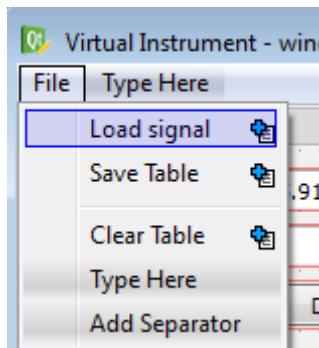


FIGURA 97: BARRA DE MENÚ

11. Nos vamos a la pestaña  y seleccionamos el orden del tabulador empezando por la parte de arriba y yendo hacia abajo.

12. Seleccionamos la pestaña  y asociamos las etiquetas a sus respectivos campos (*Spin Box*, *Double Spin Box*, *QComboBox*...).

13. Seleccionamos la pestaña *Signals/Slots*  y asignamos las siguientes señales:

- Pinchamos con el ratón en el *QComboBox* denominado *cboxviselect*, arrastramos la señal hasta coincidir con el *Stack Widget* de la ventana central, momento en el cual debemos soltar el ratón. Nos aparecerá la siguiente ventana. En la columna izquierda seleccionamos *currentIndexChanged(int)* y en la derecha seleccionamos *setCurrentIIndex(int)*.

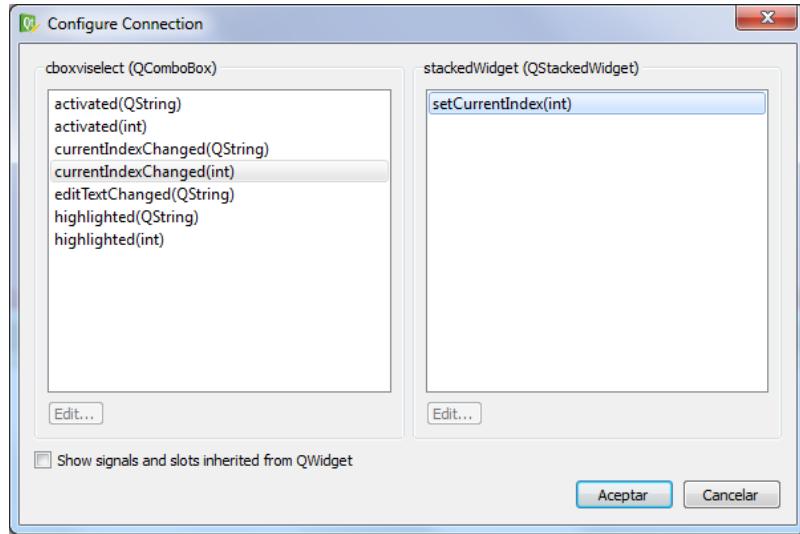


FIGURA 98: ASOCIACIÓN DE SIGNALS/SLOTS (1)

- Hacemos lo mismo que en el punto anterior pero de forma inversa, en el sentido *Stack Widget* -> *QComboBox*.
- Volvemos a repetir el mismo procedimiento, en este caso entre el *QComboBox* denominado *cboxviselect* y el *Stack Widget* de la ventana *Virtual Instrument*.

Una vez finalizado este proceso, en la ventana *Signal/Slot Editor* se puede ver lo siguiente:

Sender	Signal	Receiver	Slot
stackedWidget	current...ed(int)	cboxviselect	setCurrentIndex(int)
cboxviselect	currentl...ged(int)	stackedWidget	setCurrentIndex(int)
cboxviselect	currentl...ged(int)	stackedWidget_2	setCurrentIndex(int)
stackedWidget_2	current...ed(int)	cboxviselect	setCurrentIndex(int)

FIGURA 99: ASOCIACIÓN DE SIGNAS/SLOTS (2)

Una vez finalizada la interfaz gráfica aplicamos el *binding* de PyQt4 para obtener el código Python equivalente:

```
>> pyuic4 window.ui -o window.py
```

4.4.- Funcionamiento de la interfaz gráfica

4.4.1.- Funcionamiento

La interfaz gráfica definitiva es la que se muestra a continuación. La primera figura se corresponde al frecuencímetro y la segunda al generador de funciones. Para poder conmutar entre una pantalla y la otra, seleccionar “*Frequency Meter*” o “*Function Generator*” en el *QComboBox* cuya etiqueta asociada es *Instrument*.

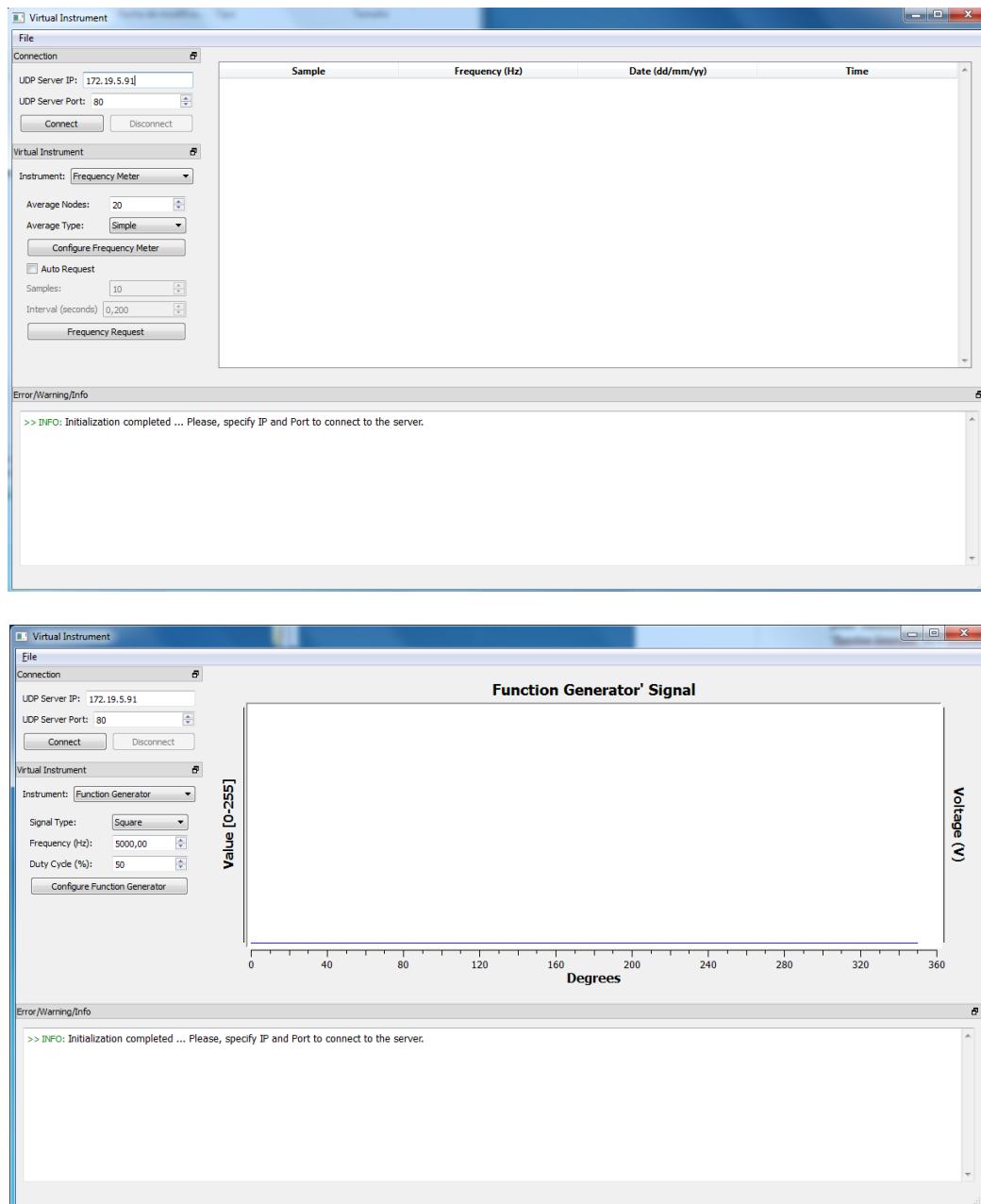
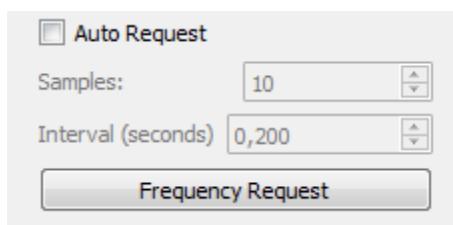


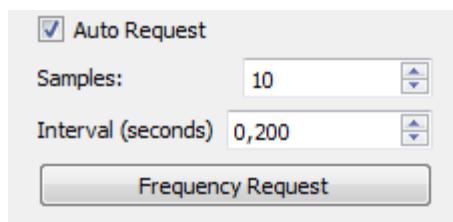
FIGURA 100: APLICACIÓN FINAL

La interfaz se compone de cuatro ventanas principales:

- *Connection*: donde se indica la IP y puerto del servidor, además de sendos botones para conectar y desconectar (se crea y destruye el socket respectivamente).
- *Virtual Instrument*: donde seleccionamos el instrumento deseado y su configuración. En función del instrumento disponemos de distintas opciones:
 - *Frequency Meter*: podemos cambiar el número de nodos de promediado y el tipo de promediado. Si pulsamos el botón “Configure Frequency Meter” se enviará al servidor la información de configuración correspondiente.
Para solicitar la frecuencia actual disponemos de dos modos de funcionamiento:
 1. Manual (*Auto Request* desactivado): cada vez que el usuario hace clic en el botón “Frequency Request” se envía una petición de frecuencia al servidor.



2. Automático (*Auto Request* activado): cuando el usuario hace un clic en “Frequency Request” se envían *numsamples* peticiones de frecuencia con un intervalo entre peticiones de *dintervalsamples* segundos.



- *Function Generator*: podemos elegir entre una señal de tipo cuadrada o una señal de tipo no cuadrada. Si seleccionamos una señal de tipo cuadrada podemos cambiar su frecuencia y su ciclo de trabajo. En la gráfica de la derecha se muestra la señal.

Signal Type:	Square
Frequency (Hz):	5000,00
Duty Cycle (%):	50
Configure Function Generator	

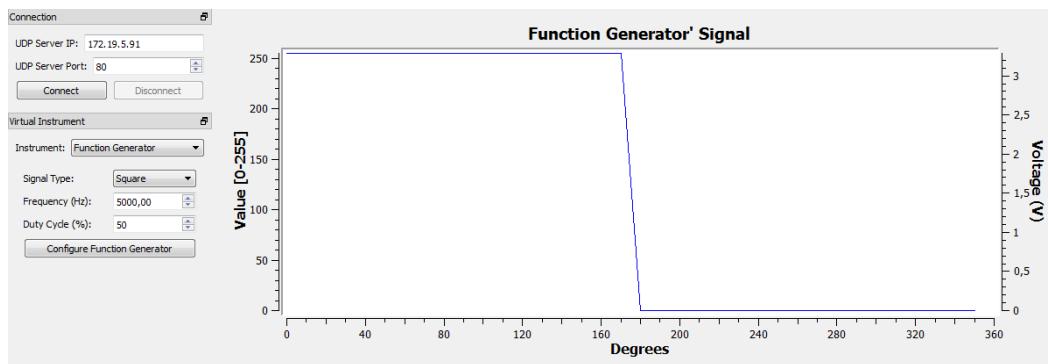


FIGURA 101: CONFIGURACIÓN CON SEÑAL CUADRADA

Por el contrario, si seleccionamos una señal de tipo no cuadrada únicamente podremos cambiar la frecuencia. En la gráfica de la derecha se muestra una señal no cuadrada.

Signal Type:	Non-Square
Frequency (Hz):	5000,00
Duty Cycle (%):	50
Configure Function Generator	

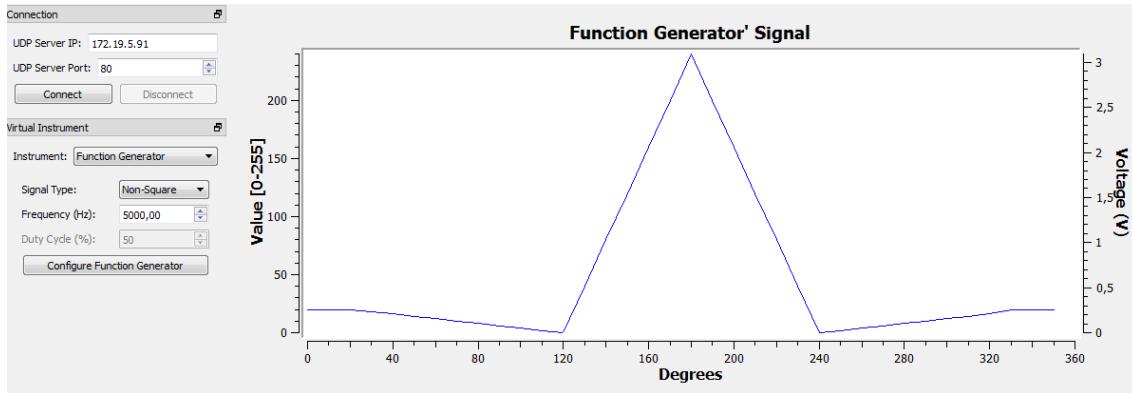


FIGURA 102: CONFIGURACIÓN CON SEÑAL NO CUADRADA

- *Error/Warning/Info:* se muestra información relevante al usuario.
- *Ventana central:* es una tabla en el caso de seleccionar el frecuencímetro (donde se mostrarán las muestras de frecuencia), y una gráfica si se selecciona el generador de funciones (donde se muestra la función que se envía al servidor).

Además de las ventanas también disponemos de la barra de menú, en la que incluimos tres opciones: *Load Signal*, *Save Table* y *Clear Table*.

- *Load Signal:* si estamos en modo “Function Generator” y señal de tipo no cuadrada (“Non-Square”) podremos cargar una señal de 36 coeficientes, donde cada coeficiente debe representar un valor entre 0 (0V) y 255 (3,3V). Haciendo clic en *Load Signal* aparece una ventana emergente donde podemos seleccionar la señal no cuadrada deseada.

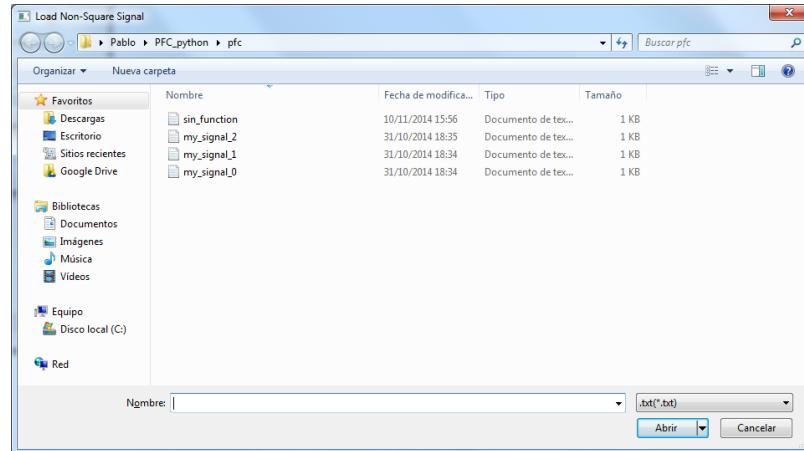


FIGURA 103: CARGAR SEÑAL NO CUADRADA

- *Save Table*: si se hace clic en este elemento saltará una ventana emergente donde podremos seleccionar el directorio y nombre con el que queremos guardar la tabla actual. El fichero tendrá una extensión *.csv, lo que facilitará el procesado posterior de los datos.

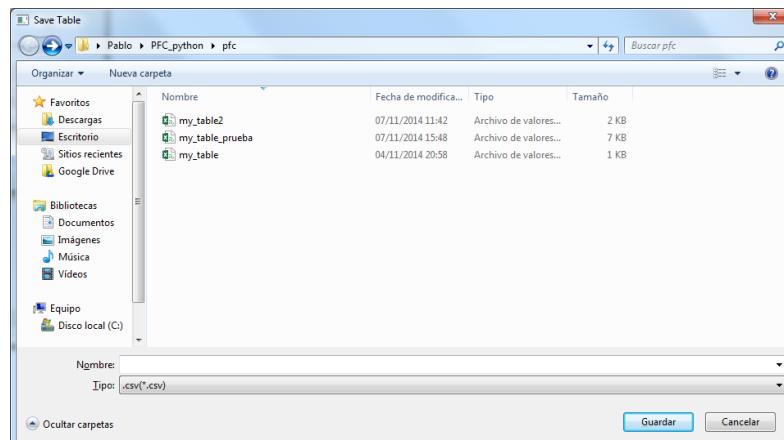


FIGURA 104: GUARDAR TABLA EN FORMATO CSV

- *Clear Table*: permite borrar la tabla actual de frecuencias.

4.4.2.- Ficheros de señal no cuadrada

El formato que se sigue para los ficheros de entrada de señal no cuadrada es el siguiente: la primera línea es un comentario que empieza con el símbolo '#', y

posteriormente se introducen 36 valores en dos columnas, que son separadas por dos tabulaciones. La primera columna es el índice (entre 0 y 35), y la segunda columna es el valor asociado a dicho índice (entre 0 y 255). Por ejemplo:

```
##### Values of the CCP RAM #####
0      20
1      20
2      20
3      18
4      16
5      14
6      12
7      10
8      8
9      6
10     4
11     2
12     0
13     40
14     80
15     120
16     160
17     200
18     240
19     200
20     160
21     120
22     80
23     40
24     0
25     2
26     4
27     6
28     8
29     10
30     12
31     14
32     16
33     20
34     20
35     20
```

FIGURA 105: FICHERO DE SEÑAL NO CUADRADA

4.5.- Diseño de la aplicación de usuario

4.5.1- Diagrama de funcionamiento

La aplicación de usuario consta de un hilo de ejecución primario y dos hilos secundarios:

- **Hilo principal:** es el encargado de gestionar los cambios que se producen en la interfaz gráfica y del envío de paquetes.
- **Hilo secundario 1:** se encarga de recibir paquetes.
- **Hilo secundario 2:** se encarga de enviar paquetes cuando está activada la opción *Auto Request* del frecuencímetro. Este envío de paquetes se realiza en un hilo de ejecución independiente porque el hilo se suspenderá durante un tiempo (el intervalo de tiempo entre paquetes seleccionado por el usuario). Con motivo de dicha suspensión, este tipo de transmisión de paquetes no puede ser realizado en el hilo principal, ya que, si se suspendiese, se estaría suspendiendo la interfaz gráfica, lo cual sería inaceptable.

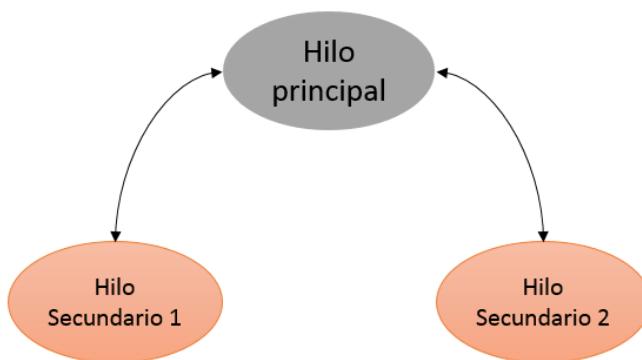


FIGURA 106: HILOS DE LA APLICACIÓN DE USUARIO

4.5.2.- Explicación del código

El código se divide en tres ficheros: *vi_protocol.py*, *window.py* y *main.pyw*. Todos los códigos han sido realizados para que sean completamente autoexplicativos, por lo

que a continuación únicamente se describirá de forma descriptiva el funcionamiento de cada función. Si el lector quiere acceder al código completo puede acudir al [**Apéndice 2: código fuente de la aplicación de usuario.**](#)

4.5.2.1.- window.py

Este archivo se genera, como se indicó anteriormente, de forma automática al hacer un *binding* de *window.ui* con PyQt4.

Si se observa dicho archivo se verá que consta de una clase con dos funciones: *setupUi* y *retranslateUi*.

```
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        .
        .
        .

    def retranslateUi(self, MainWindow):
```

No entraremos en profundidad en este código ya que nosotros únicamente lo usaremos de forma indirecta al interactuar con la aplicación. Si el lector quiere acceder al código completo, lo tiene disponible en el [Apéndice 2A: Interfaz gráfica \(window.py\)](#).

4.5.2.2.- vi_protocol.py

Se trata del código en el que está definido el *VI Protocol*. Concretamente se definen las constantes necesarias de funcionamiento (los comandos del cliente y del servidor) y dos funciones, que detallaremos a continuación:

- `def assemble_packet(fields):`: esta función permite ensamblar los campos de un paquete en una cadena de bytes, concretamente de 40 bytes de longitud, por ser esta la longitud de los paquetes del protocolo implementado. Los paquetes están formados por un campo *command* de 4 bytes, y un campo *data* de 36 bytes.

Dependiendo del comando a enviar, la integración del paquete se realiza de una forma o de otra debido a que los datos asociados al paquete pueden ser de varios tipos (*int*, *float*, *char*) y en consecuencia pueden ocupar más o menos bytes.

Por ejemplo, si en el paquete a enviar solo se envía el comando sin datos, dicho paquete se genera del siguiente modo:

```
packet = struct.pack("!i", fields[0]) + str(bytearray(36))
```

Lo que quiere decir la línea anterior es que se introduce en “packet” el comando contenido en “fields[0]” como un entero de 4 bytes (“i”) en formato de red (“!”), es decir, en formato *big endian*. Posteriormente rellenamos con 36 ceros el resto del paquete.

- `def disassemble_packet(packet):`: el procedimiento es similar, salvo que en lugar de utilizar la función *struct.pack()* utilizamos la función *struct.unpack()*.

El código completo de esta sección está disponible en [Apéndice 2B: Protocolo VI \(vi_protocol.py\)](#).

4.5.2.3.- main.pyw

Este es el código principal del cliente. En él capturamos las señales que genera el usuario en la interfaz gráfica y realizamos las acciones oportunas en cada caso. Como en los apartados anteriores el código ya es de por sí completamente auto-explicativo, de forma que a continuación únicamente se describirá de forma descriptiva. Para consultar el código completo el lector puede acudir a [Apéndice 2C: Aplicación de usuario \(main.pyw\)](#).

La estructura principal del código es la que se muestra a continuación. De forma general, lo que hacemos es lo siguiente: importamos las librerías necesarias, creamos una clase denominada *Form*¹⁰ en donde incluiremos las funciones, y posteriormente creamos y lanzamos la aplicación.

```
Import . . .

.

.

.

class Form(QMainWindow, Ui_MainWindow):

.

.

.

# Run the application
if __name__ == '__main__':
    app = QApplication(sys.argv)
    form = Form()
    form.show()
    sys.exit(app.exec_())
```

Las funciones que implementamos dentro de la clase *Form* son las que nos permiten darle una funcionalidad a las acciones que realiza el usuario con la interfaz gráfica. Son las siguientes:

- **def __init__(self):** función de inicialización de la clase *Form*. Crea e inicializa la interfaz gráfica.
- **def initUi(self):** función llamada por “*__init__(self)*”. Se habilitan y deshabilitan los botones oportunos de la interfaz, y se conectan las señales necesarias. Por ejemplo, cuando el usuario pulsa sobre el botón *btnconnect*, asociamos dicha acción a la función *handle_btnconnect* de la siguiente forma:

```
self.connect(self.btnconnect, SIGNAL("clicked()"), self.handle_btn
connect)
```

- **def graph_signal_init(self):** función llamada por “*__init__(self)*”. Se inicializa el gráfico del generador de funciones (ejes de la gráfica, nombres, etc).

¹⁰ En la clase *Form* aplicamos herencia sobre las clases *QMainWindow* y *Ui_MainWindow*, que son las clases relacionadas con la interfaz gráfica *window.py*.

- `def initTableFreq(self):` función llamada por “`__init__(self)`”. Inicializamos la tabla de frecuencia con cuatro columnas y cero filas.
- `def handle_btnconnect(self):` cuando el usuario pulsa `btnconnect` se crea un socket UDP (de tipo `SOCK_DGRAM` en Python) y lanzamos el hilo de recepción de paquetes.
- `def receive_pkt(self):` función que ejecuta el hilo que se creó con la función `handle_btnconnect`. Recibe paquetes en el socket y actualiza la tabla de frecuencias (el único paquete que se recibe en este proyecto es el paquete con comando `CMD_FREQ_ANWER` en el que tenemos la frecuencia asociada a una solicitud enviada previamente).
- `def handle_btndisconnect(self):` cuando el usuario pulsa el botón `btndisconnect` se cierra el socket.
- `def handle_load_signal(self):` cuando se pulsa **File -> Load Signal** se muestra una ventana emergente donde el usuario puede cargar una señal de tipo no cuadrada. Esta función obtiene los valores presentes en el fichero seleccionado y actualiza la gráfica con dichos valores.
- `def plot_axis(self):` actualiza el gráfico y dimensiona los ejes para adaptarse de forma óptima a la nueva señal introducida.
- `def handle_save_table(self):` cuando el usuario pulsa **File -> Save Table** salta una ventana emergente donde el usuario puede dar un nombre al fichero (con extensión `*.csv`) en donde se guardará la tabla.
- `def handle_clear_table(self):` si se pulsa **File -> Clear Table** se borra la tabla actual.
- `def handle_btnconfigfm(self):` cuando el usuario pulsa el botón `btnconfigfm` se envía la configuración del frecuencímetro al servidor.
- `def handle_btnfreqrequest(self):` cuando el usuario pulsa `btnfreqrequest` y `Auto Request` está desactivado enviamos un paquete al servidor solicitándole la frecuencia actual que está midiendo. Si Auto Request está activado se crea un nuevo hilo que ejecuta la función `sending_auto_request`.

- `def sending_auto_request(self, my_list, num, delay)`: función que se ocupa de enviar *num* paquetes con un intervalo entre paquetes de *delay* segundos. En el parámetro de tipo lista *my_list* se encuentran los campos del paquete a ser enviado.
- `def handle_checkboxfreqreq(self)`: habilita o deshabilita los campos *numsamples* y *dintervalsamples* en función de si se activa o desactiva *Auto Request*.
- `def handle_btnconfigfg(self)`: cuando se pulsa el botón *btnconfigfg* se envía la configuración del generador de funciones. Se envían unos u otros comandos en función si la señal es de tipo cuadrada o de tipo no cuadrada.
- `def handle_cboxsignaltypc(self)`: habilita o deshabilita el campo *duty cycle* en función de si se ha elegido una señal cuadrada o una señal no cuadrada.
- `def handle_numdutycycle(self)`: cuando se cambia el campo *numdutycycle* se actualiza la señal del gráfico para hacerla coincidir con dicho ciclo de trabajo.
- `def write_text_browser_error(self, error)`: escribe un error en la ventana *Error/Warning/Info*.
- `def write_text_browser_warning(self, warning)`: escribe un *warning* en la ventana *Error/Warning/Info*.
- `def write_text_browser_info(self, info)`: escribe una información en la ventana *Error/Warning/Info*.

CAPÍTULO 5: VERIFICACIÓN DE FUNCIONAMIENTO

5.1.- Herramienta *WireShark*

El programa *WireShark*¹¹ es un analizador de paquetes en el que se muestran los paquetes que se reciben y se envían a través de la interfaz de red del ordenador. Se trata de una herramienta muy útil a la hora de comprobar que los paquetes de nuestra aplicación se están enviando y recibiendo, y se ha utilizado a lo largo de desarrollo del proyecto. Para demostrar el funcionamiento de la aplicación no es necesario ejecutarlo (ni instalarlo) pero se ha decidido incluirlo en esta sección por haber sido utilizado de forma continua en el desarrollo del proyecto.

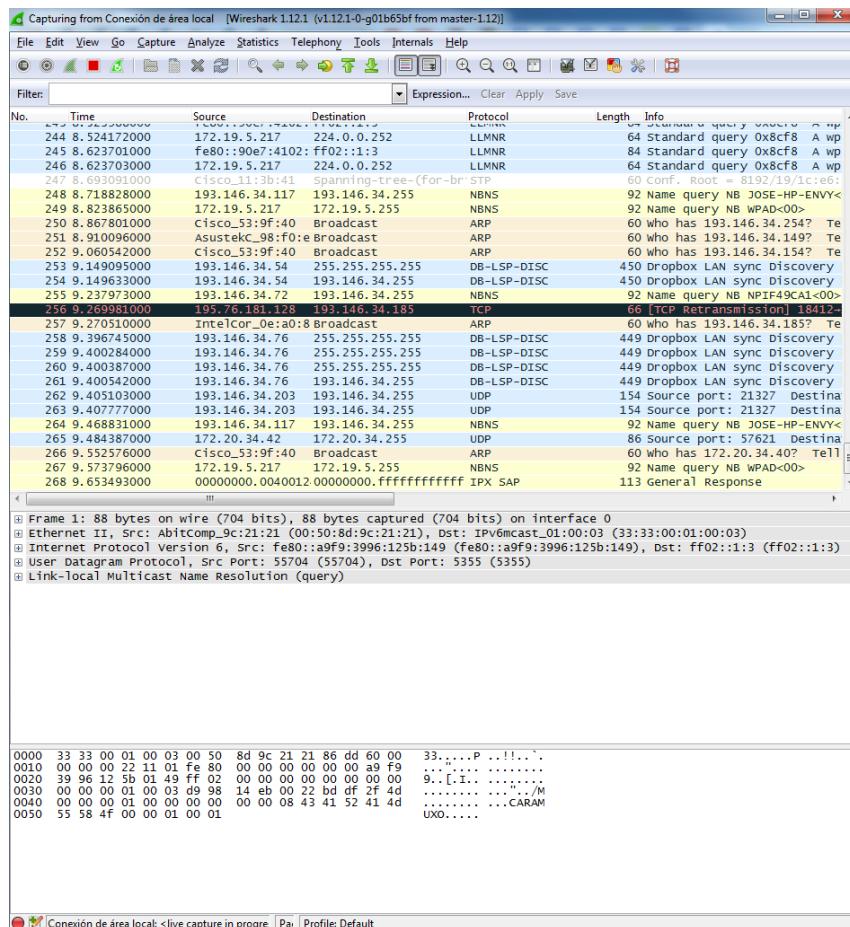


FIGURA 107: VENTANA PRINCIPAL DE WIRESHARK

¹¹ Se puede descargar desde <https://www.wireshark.org/download.html>

5.2.- Funcionamiento

5.2.1.- Servidor

Se deberá introducir la IP y MAC deseada modificando los ficheros *frec_gen.h* (campos *DEFAULT_IP_SOURCE*, *DEFAULT_SOURCE_MAC*) y *DM9000A.h* (campo *ether_addr*). A continuación configuramos (con *Quartus II Programmer*), y programamos (con *Eclipse*) la placa de desarrollo. Por último conectamos el cable Ethernet entre la placa y el *switch* o el *router* de la red.

5.2.1.1.- Frecuencímetro

Los cables de conexión entre el generador de funciones externo y la placa de desarrollo se conectan al bus de expansión GPIO_0. Concretamente la masa se conecta al sexto pin de la segunda columna y la señal al quinto pin de la primera columna. Ambos cables se conectan a un cable coaxial que será la salida del generador de funciones externo.

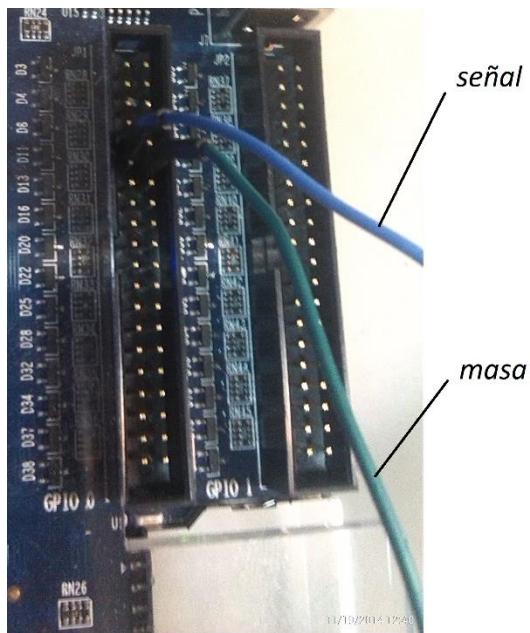


FIGURA 108: CONEXIÓN DE CABLES CON EL FRECUENCÍMETRO

Cuando el cliente active el frecuencímetro, en la pantalla de LCD se muestra "Frequency Meter", y se enciende el LEDG0.

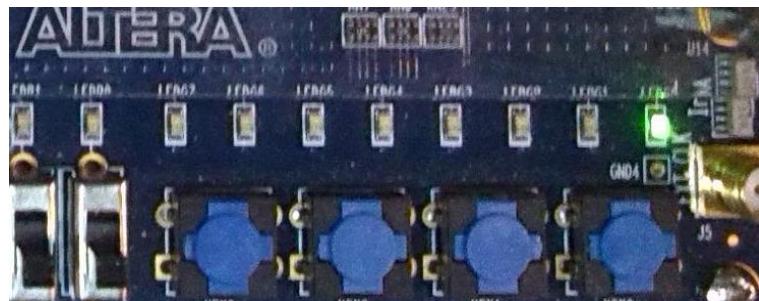
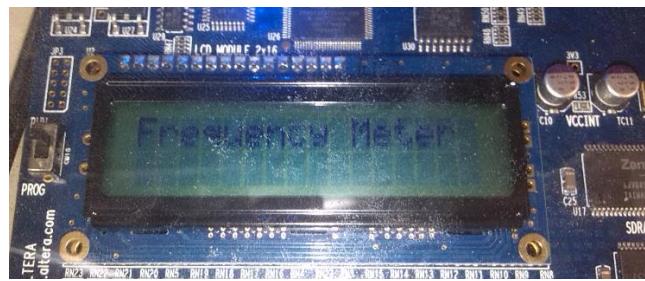


FIGURA 109: ESTADO DE LA PLACA CON EL FRECUENCÍMETRO ACTIVADO

5.2.1.2.- Generador de funciones

Los cables de conexión en modo generador de funciones se conectan al bus de expansión GPIO_1. Concretamente la masa se conecta al sexto pin de la segunda columna y la señal al octavo pin también de la segunda columna. Dichos cables se conectarán a un osciloscopio para observar la señal generada.

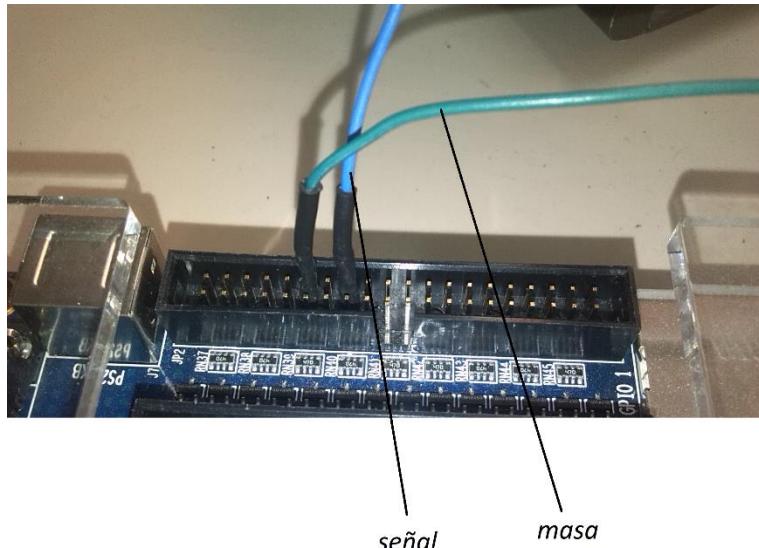


FIGURA 110: CONEXIÓN DE CABLES CON EL GENERADOR DE FUNCIONES ACTIVADO

Cuando el cliente active el generador de funciones, en la pantalla de LCD se muestra “F. Generator” y la frecuencia seleccionada, y también se enciende el LEDG1. Además, si la señal es de tipo cuadrada se ilumina el LEDR0 y si es de tipo no cuadrada se ilumina el LEDR1.

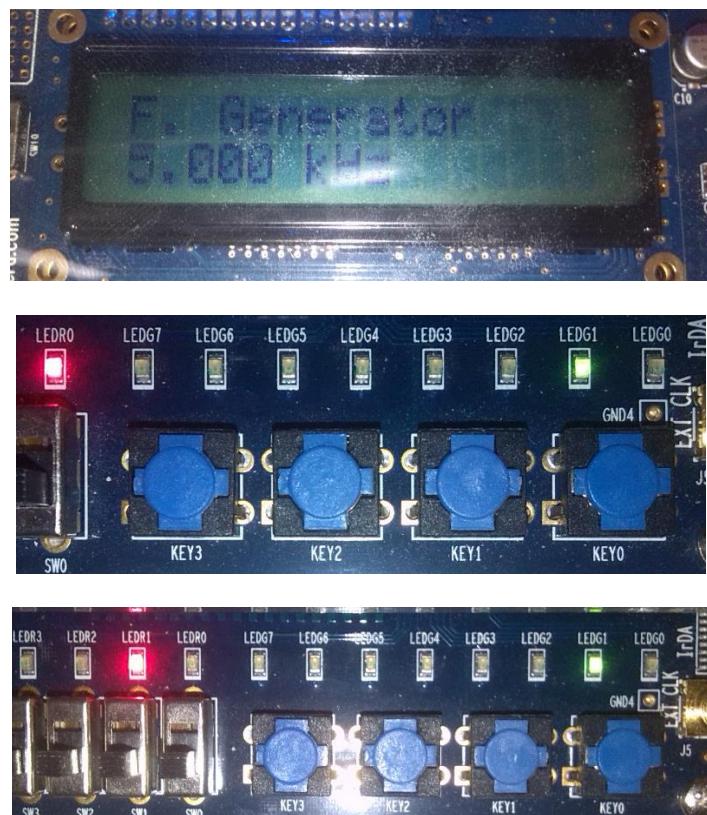


FIGURA 111: ESTADO DE LA PLACA CON EL GENERADOR DE FUNCIONES (SEÑAL CUADRADA Y SEÑAL NO CUADRADA)

5.2.2.- Cliente

En el ordenador hacemos doble clic sobre el fichero *main.pyw*. De esta forma ya tenemos la aplicación en pantalla. Indicamos la IP y puerto del servidor y hacemos clic en *Connect*. Ahora solo nos seleccionar el instrumento deseado y su configuración.

5.2.2.1.- Frecuencímetro

Seleccionamos el modo frecuencímetro, configuramos el número de nodos y el tipo de promediado deseado y hacemos clic en “Configure Frequency Meter”.

A continuación mostramos un ejemplo de medida: con el generador de funciones externo generamos una señal cuadrada de 10 KHz, que procedemos a medir con nuestra aplicación. Una vez configurada la placa en modo frecuencímetro ya podemos solicitar la frecuencia actual (con *Auto Request* activado o desactivado).



FIGURA 112: GENERADOR DE FUNCIONES EXTERNO CON SEÑAL CUADRADA DE 10 KHz

Sample	Frequency (Hz)	Date (dd/mm/yy)	Time
0	10000.0996094	19/11/2014	13:05:25
1	9999.59960938	19/11/2014	13:05:25
2	10000.2001953	19/11/2014	13:05:26
3	10000.0996094	19/11/2014	13:05:26
4	9999.79980469	19/11/2014	13:05:26
5	9999.70019531	19/11/2014	13:05:26
6	10000.2001953	19/11/2014	13:05:26
7	10000.0996094	19/11/2014	13:05:27
8	10000.0	19/11/2014	13:05:27
9	10000.0	19/11/2014	13:05:27

FIGURA 113: 10 AUTO REQUEST CON 10 MUESTRAS Y 0.2 SEGUNDOS DE INTERVALO ENTRE MUESTRAS

Sample	Frequency (Hz)	Date (dd/mm/yy)	Time
0	9999.90039062	19/11/2014	13:09:45
1	9999.90039062	19/11/2014	13:09:46
2	10000.0	19/11/2014	13:09:47

FIGURA 114: TRES SOLICITUDES DE FRECUENCIA SIN AUTO REQUEST

5.2.2.2.- Generador de funciones cuadradas

Seleccionamos el modo generador de funciones, la señal de tipo cuadrada, y configuramos la frecuencia y el ciclo de trabajo deseado. Por último hacemos clic en “Configure Function Generator”. A continuación se muestran diversas capturas de pantalla para distintas configuraciones.

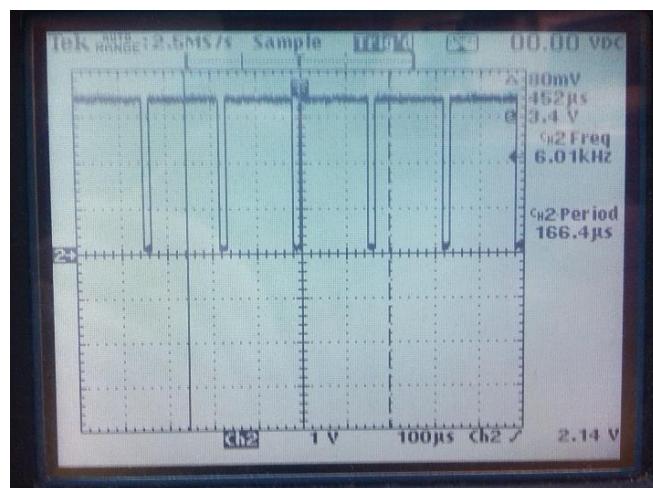
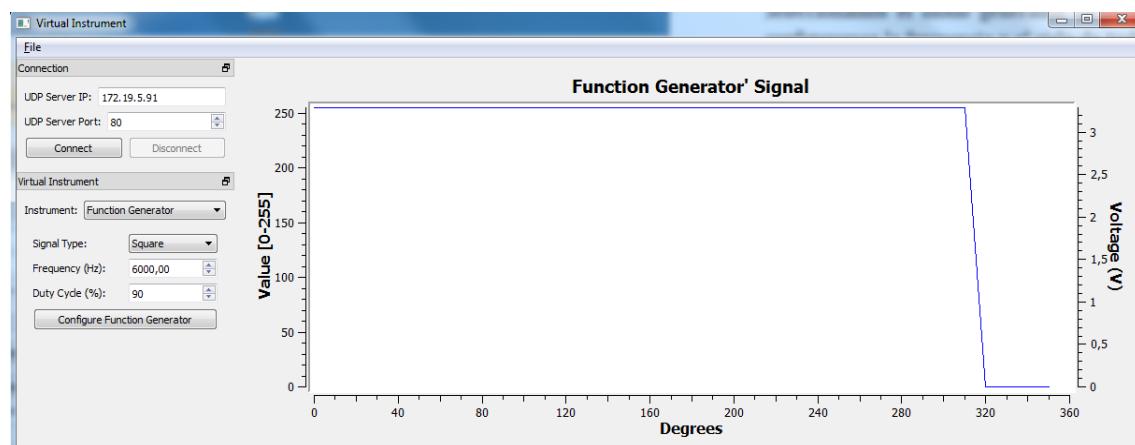


FIGURA 115: SEÑAL CUADRADA DE 6 KHZ Y 90% DE DUTY CYCLE

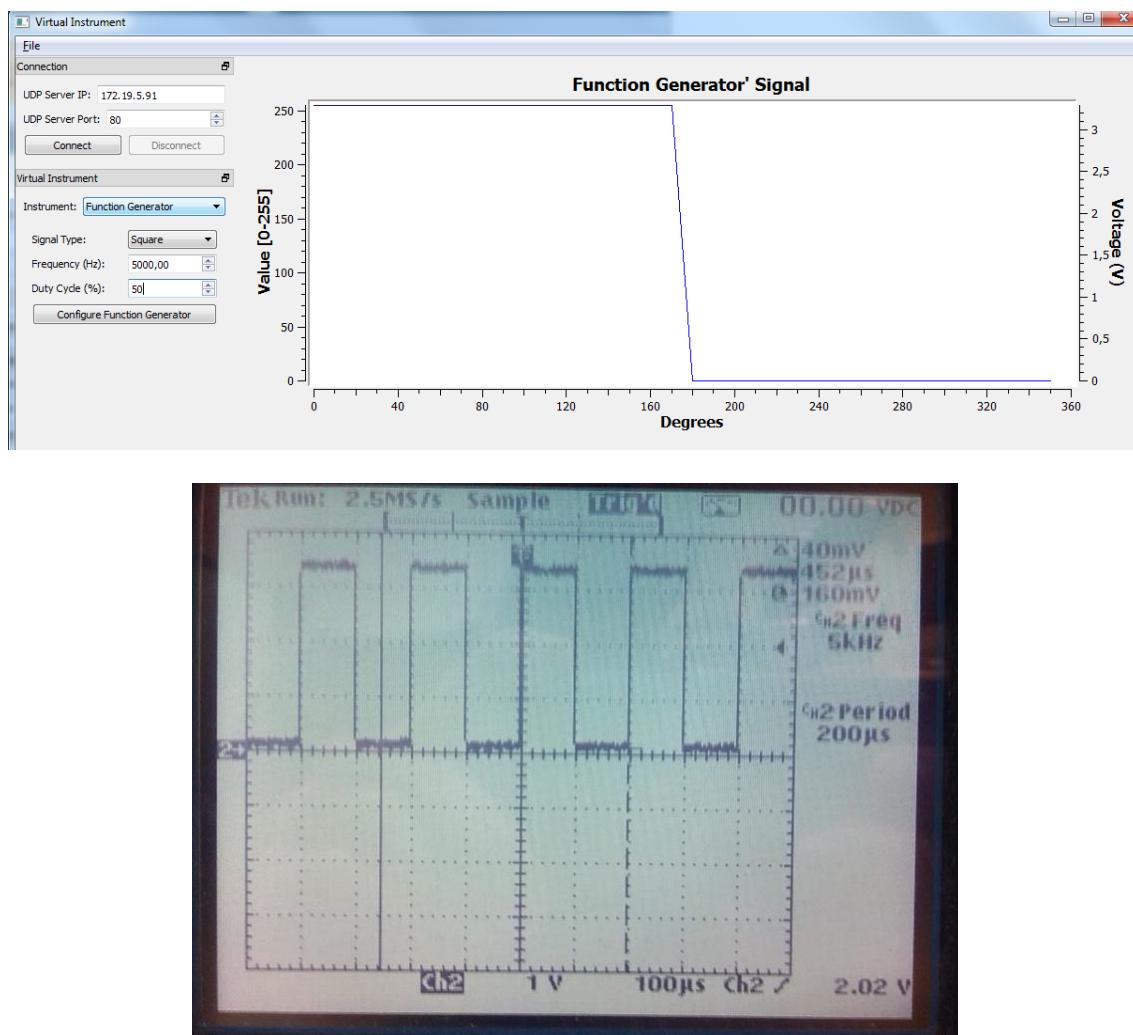
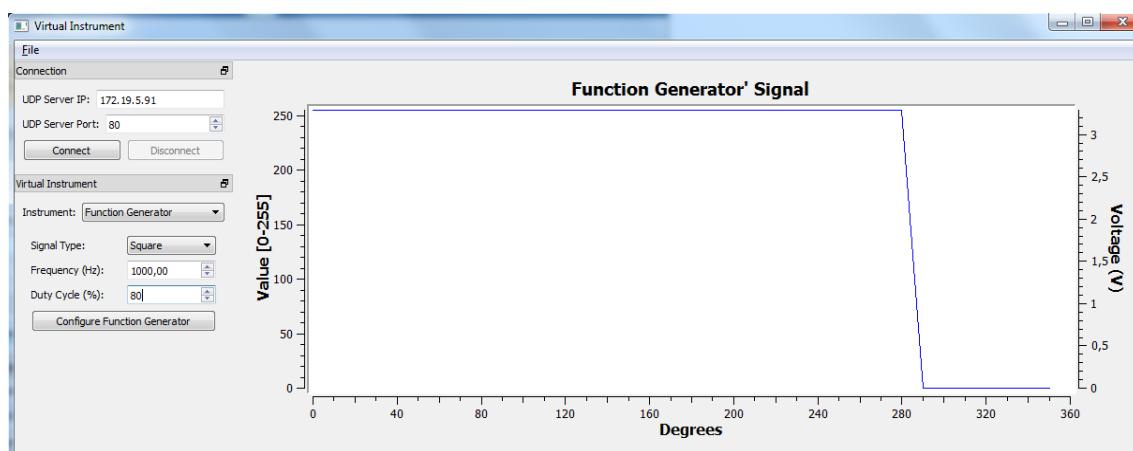


FIGURA 116: SEÑAL CUADRADA DE 5 KHZ Y 50% DE DUTY CYCLE



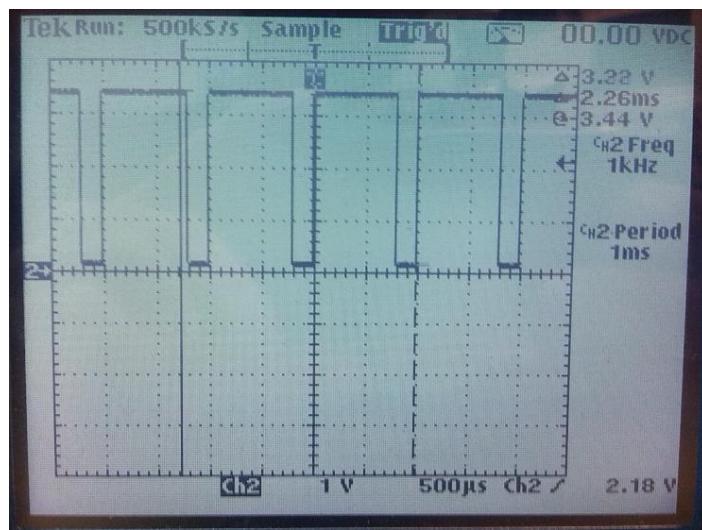


FIGURA 117: SEÑAL CUADRADA DE 1 KHZ Y 80% DE DUTY CYCLE

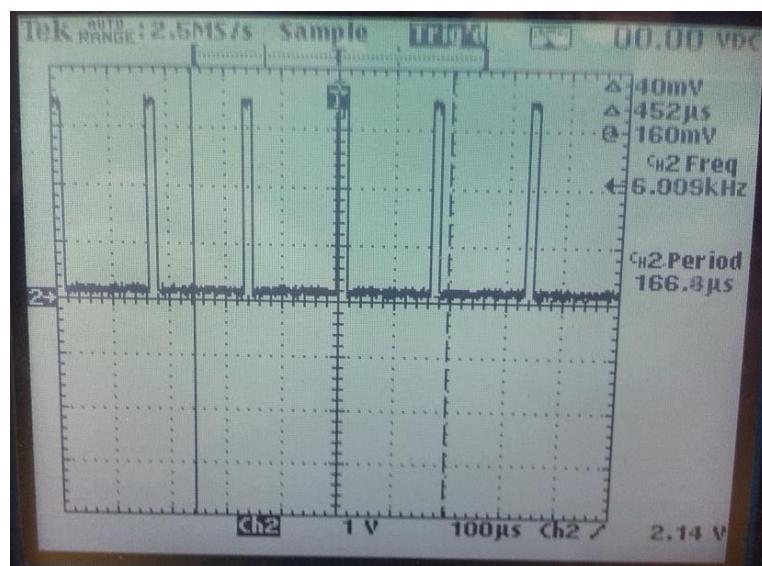
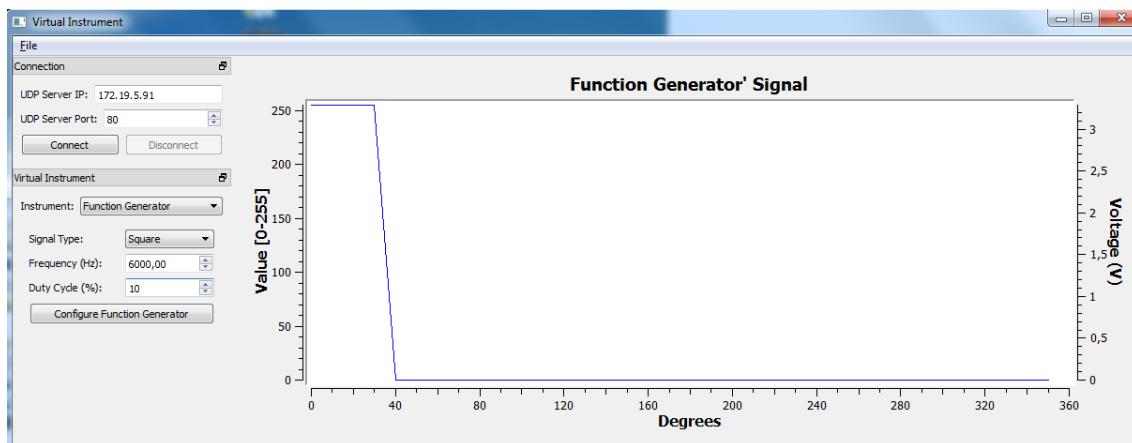


FIGURA 118: SEÑAL CUADRADA DE 6 KHZ Y 10% DE DUTY CYCLE

5.2.2.3.- Generador de funciones no cuadradas

Seleccionamos el modo generador de funciones, la señal de tipo no cuadrada, configuramos la frecuencia deseada y hacemos clic en "Configure Function Generator". A continuación se muestran diversas capturas de pantalla para distintas señales. Se recuerda que para cargar una señal hay que hacer clic en **File -> Load Signal** y posteriormente elegir la señal en la ventana emergente.

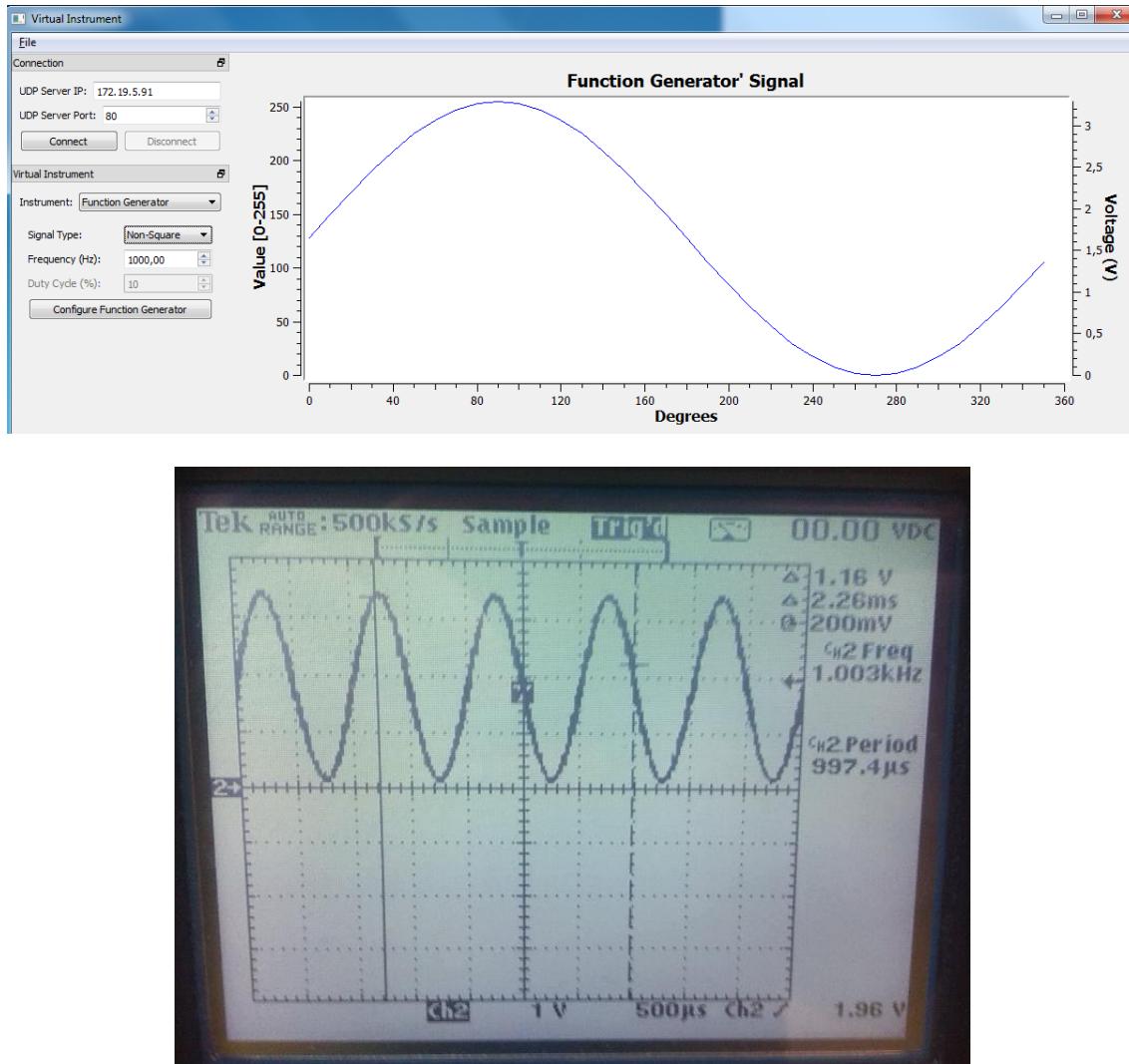


FIGURA 119: SEÑAL SENOIDAL DE 1 KHZ

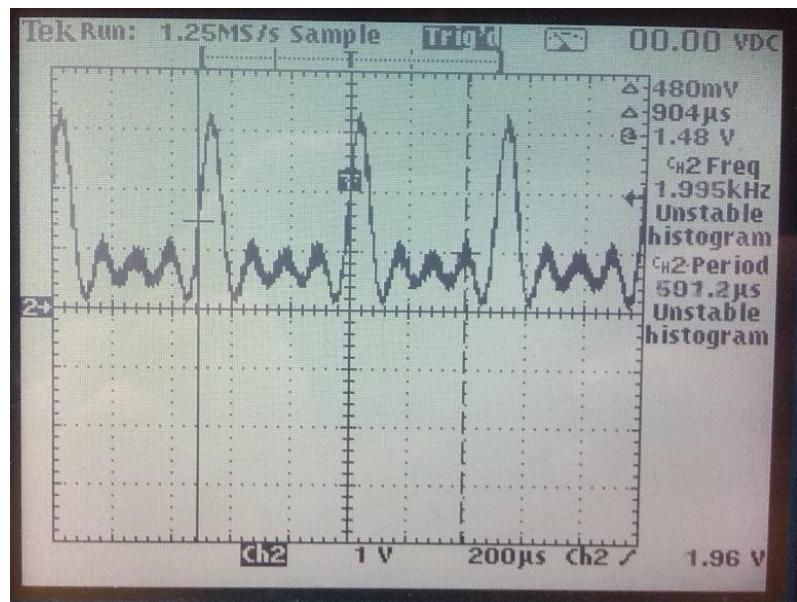
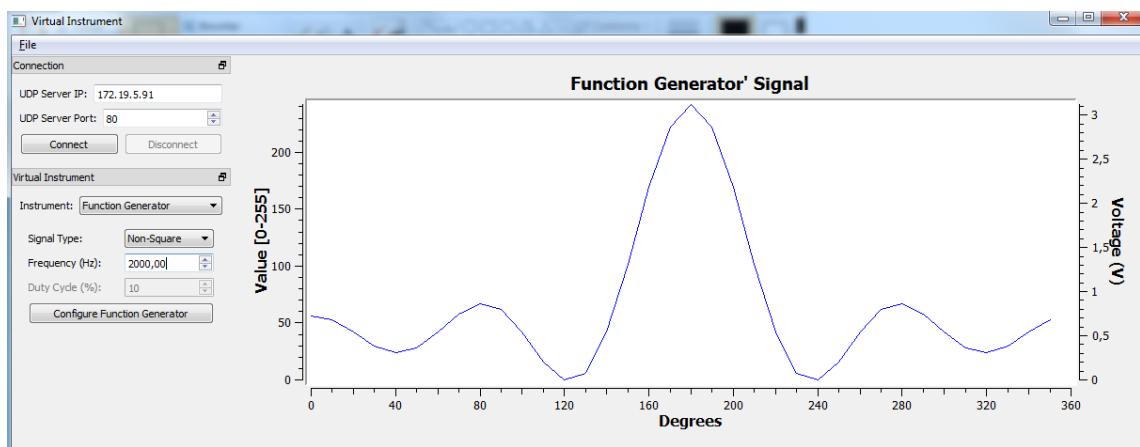
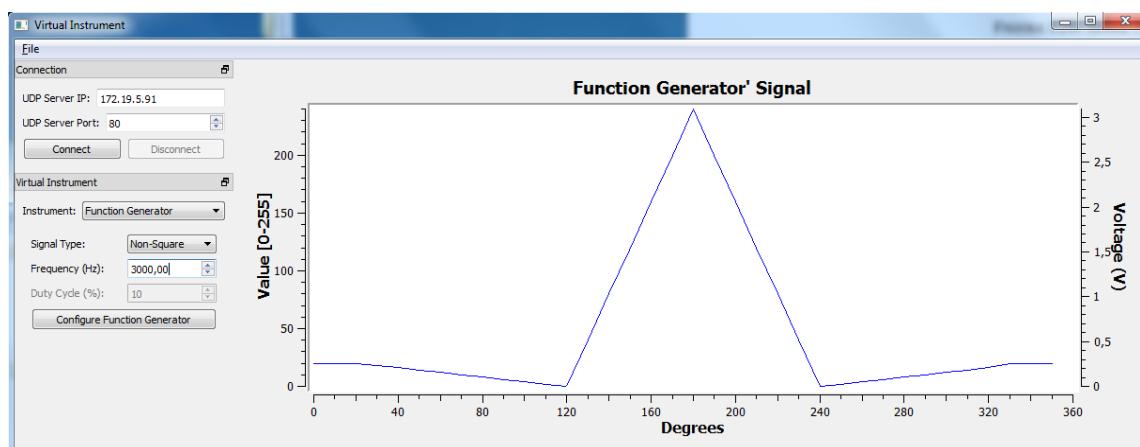


FIGURA 120: SEÑAL TIPO SINC DE 2 KHZ



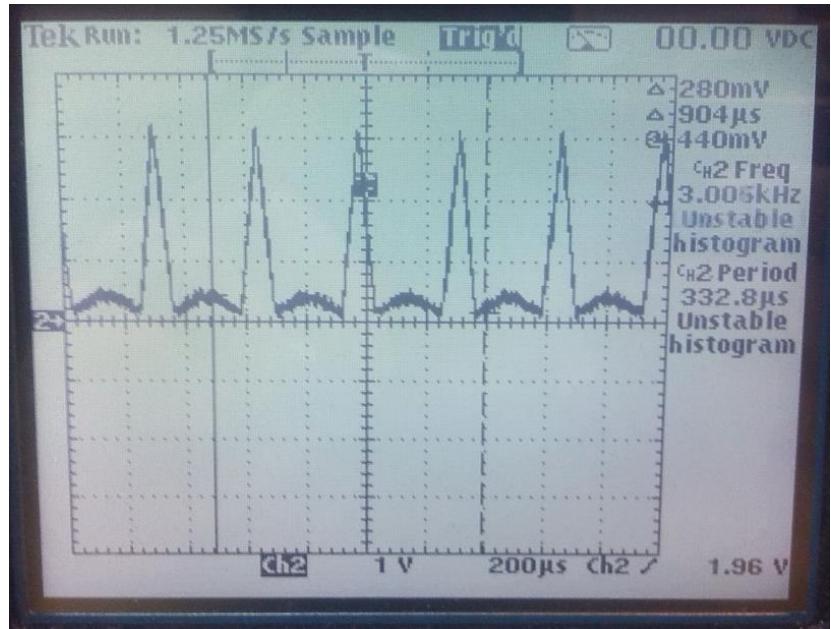


FIGURA 121: SEÑAL DE TIPO TRIANGULAR DE 3KHz

5.3.- Resultados

En los siguientes sub-apartados se comentan una serie de consideraciones referentes a cada uno de los instrumentos y sus respectivas características.

5.3.1.- Frecuencímetro

Para calcular la precisión se ha comparado la desviación típica de nuestro frecuencímetro con respecto a la del osciloscopio Tektronix THS710:

- Para cada frecuencia del generador de funciones PM5192 se han apuntado los dos valores extremos que mide el osciloscopio (“Osciloscopio (1)” y “Osciloscopio (2)”) y se ha calculado su desviación típica:

$$Desv. \text{t\'{i}pica} = \sqrt{(f_{max} - f)^2 + (f_{min} - f)^2}$$

Siendo f_{max} y f_{min} las frecuencias extremas del intervalo de frecuencias que mide el osciloscopio para una frecuencia dada f de salida del generador de funciones Philips PM5192.

- La desviación típica de la FPGA para cada una de las frecuencias de entrada se ha calculado de una forma similar al apartado anterior. Se han tomado,

para cada frecuencia f de salida del generador, 50 muestras de frecuencia con un intervalo entre muestras de 0,1 segundos. Posteriormente se identifican las frecuencias máxima (f_{max}) y mínima (f_{min}) de esas 50 muestras de frecuencia, y se calcula la desviación típica de la misma forma que en el caso anterior.

Generador (Hz)	Osciloscopio (1)	Osciloscopio (2)	Desv tipica (Hz)	Desv. Tipica FPGA
1	no mide	no mide	#¡VALOR!	2,8660E-06
12	11,99	12,02	0,02236068	5,1055E-05
50	50	50,1	0,1	3,0136E-04
148	147,9	148,1	0,141421356	7,2169E-04
1264	1263	1266	2,236067977	2,5280E+01
5784	5783	5787	3,16227766	9,0829E-01
10976	1,09E+04	1,10E+04	38,62641583	2,5086E+00
27941	2,79E+04	2,80E+04	21,9544984	1,0723E+01
85613	8,55E+04	8,57E+04	122,8739191	4,5633E+01
130456	1,30E+05	1,31E+05	71,21797526	7,7042E+01
548796	5,48E+05	5,49E+05	759,4945688	1,2101E+03
745841	7,46E+05	7,46E+05	432,3910267	1,2636E+03
987456	9,87E+05	9,89E+05	1466,516962	4,2265E+03
1254986	1,25E+06	1,26E+06	1414,352149	7,9920E+03
1590017	1,59E+06	1,59E+06	3600,913495	1,5893E+04
1789126	1,79E+06	1,79E+06	3574,8779	8,2286E+03
2187645	2,18E+06	2,19E+06	6477,812131	1,8917E+04
2812345	2,81E+06	2,82E+06	5968,085958	4,2913E+04
3124675	3,11E+06	3,14E+06	17607,84626	1,1539E+04
3978568	3,97E+06	3,99E+06	11330,19188	9,1657E+04
4265741	4,25E+06	4,28E+06	20534,17059	1,0918E+05
5132789	5,11E+06	5,14E+06	23620,22527	1,2485E+05
5875125	5,83E+06	5,90E+06	49681,79999	1,5908E+05
6548326	6,50E+06	6,56E+06	48301,93114	2,3986E+05
6978121	6,94E+06	7,03E+06	69009,90713	2,1120E+05
7283647	7,25E+06	7,31E+06	45954,07727	2,6376E+05
7777777	7,74E+06	7,83E+06	60474,07261	3,1306E+05
8140526	8,07E+06	8,20E+06	94305,29864	2,0145E+05
8500648	8,43E+06	8,55E+06	82831,49044	3,0352E+05
9123987	9,03E+06	9,19E+06	119537,3094	5,2736E+05
9762654	9,73E+06	9,82E+06	69588,35701	3,3820E+05
10125687	1,00E+07	1,01E+07	96751,55781	2,6348E+05
10597321	1,05E+07	1,07E+07	121464,456	5,5246E+05
11500686	1,14E+07	1,17E+07	185865,7074	5,7883E+05
11678781	1,15E+07	1,17E+07	160883,6596	5,6799E+05
12523699	1,24E+07	1,27E+07	219784,9522	5,0577E+05
13356890	1,31E+07	1,36E+07	341534,9824	1,1498E+06

13878822	1,37E+07	1,40E+07	240739,8084	9,1618E+05
14200327	1,40E+07	1,45E+07	374345,8479	1,3436E+06

Se observa que hasta aproximadamente los 12.5 MHz las desviaciones típicas tienen el mismo orden de magnitud. A partir de los 12.5 MHz la desviación típica de nuestro frecuencímetro crece de una forma muy brusca, y por tanto el error cometido a partir de esa frecuencia es alto.

La tabla anterior se muestra de una forma gráfica en la siguiente figura. La línea continua se corresponde con la FPGA y la discontinua con el osciloscopio Tektronix THS710.

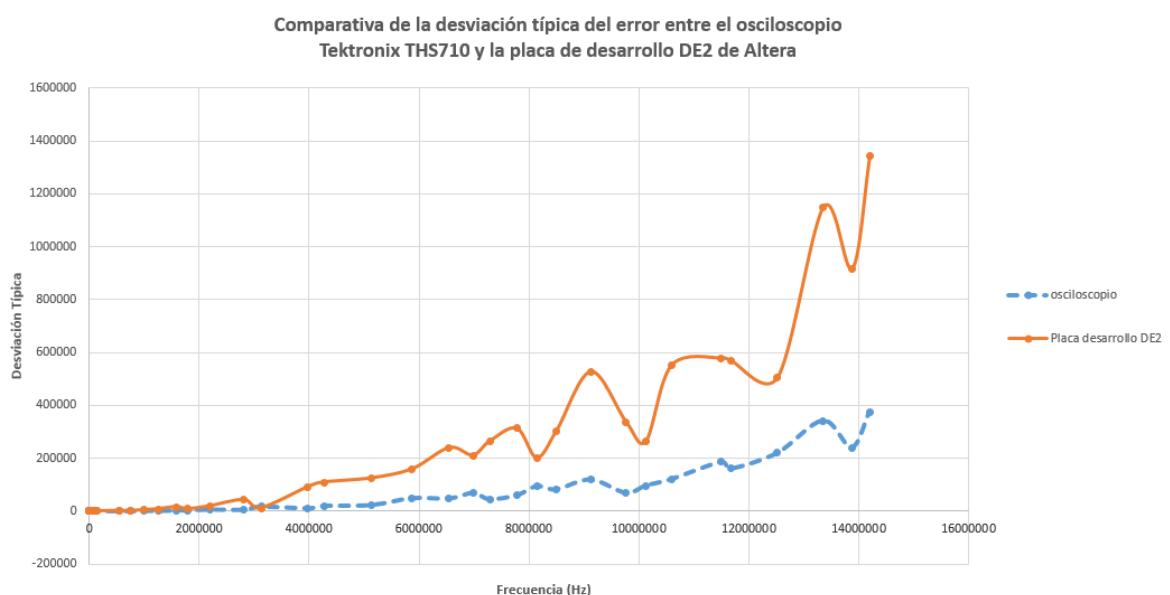


FIGURA 122: COMPARATIVA DE LA DESVIACIÓN TÍPICA OSCILOSCOPIO-FPGA

A continuación mostramos una comparativa de la desviación típica entre los tres modos de promediado que se ofrecen en el proyecto (simple, ponderado y exponencial).

Generador (Hz)	Simple	Ponderado	Exponencial
12	5,2666E-05	5,14E-05	4,60E-05
50	2,2498E+00	0,0001835	0,00022558
1264	2,5250E+01	0,0317047	0,0537377
5784	1,0559E+00	0,91325055	1,40947033
10976	3,1131E+00	2,83439168	2,51915041
27941	9,3861E+00	15,822241	14,1065641
85613	4,7811E+01	62,6397909	68,562751

130456	9,7136E+01	121,881226	83,1329414
745841	1,5719E+03	1797,32251	3114,19484
1789126	7,7991E+03	9714,75173	10947,2997
3978568	9,8031E+04	143534,64	154455,24
4265741	1,0014E+05	179510,189	173300,199
5132789	1,5004E+05	225863,382	236596,778
6548326	3,0212E+05	364902,657	361487,643
8500648	3,5118E+05	486906,747	465662,139
9123987	5,2905E+05	815850,54	784947,568
10125687	3,9473E+05	611283,311	612053,1

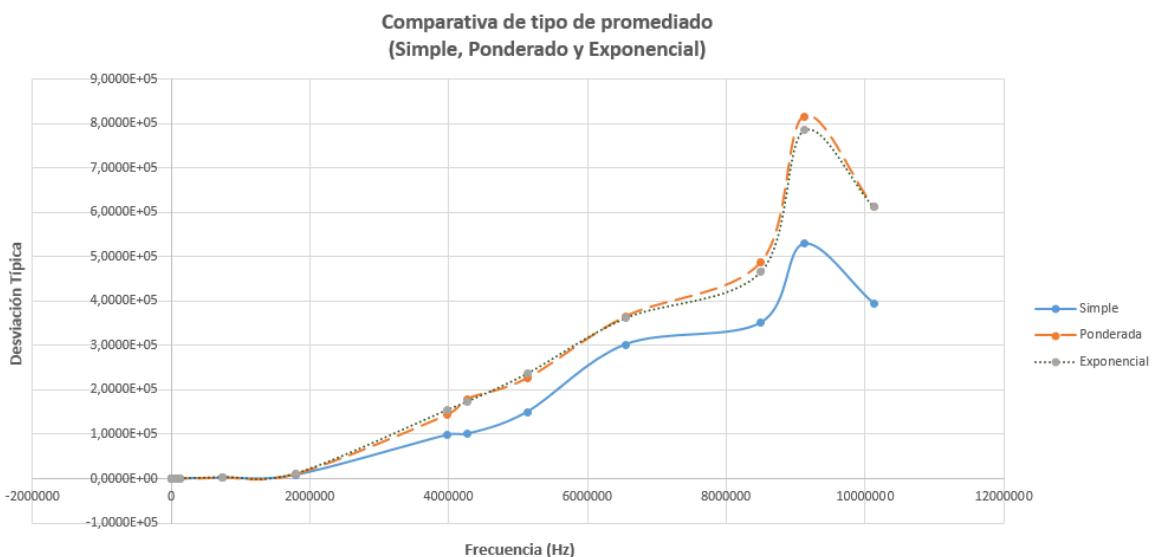


FIGURA 123: COMPARATIVA DE LA DESVIACIÓN TÍPICA ENTRE LOS TRES TIPOS DE PROMEDIADO

Como se observa en la gráfica anterior, el promediado simple (uniforme) genera la frecuencia media con un menor error que el promediado ponderado y que el promediado de tipo exponencial.

La utilidad de los promediados ponderado y exponencial se da cuando se quiere medir señales en baja frecuencia y la frecuencia varía. Si se utilizase un promediado uniforme, dicho promediado enmascararía ese cambio de frecuencia y por tanto no podríamos medir los cambios que se producen en esa señal.

5.3.2.- Generador de funciones cuadradas

Existe una limitación que viene dada por el rango de frecuencia que puede generar el instrumento. A medida que vamos incrementando la frecuencia la señal se va deformando. En las siguientes figuras observamos la deformación:

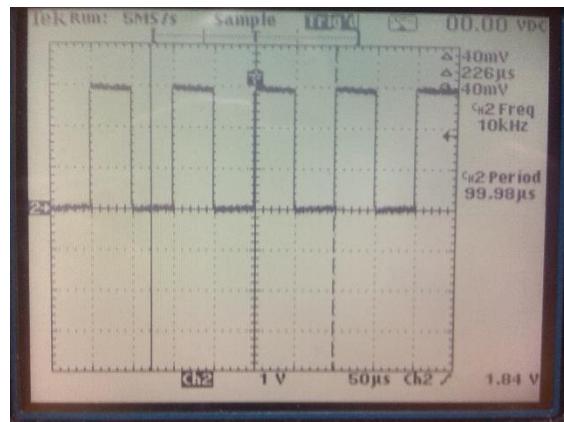


FIGURA 124: SEÑAL CUADRADA 10 KHz

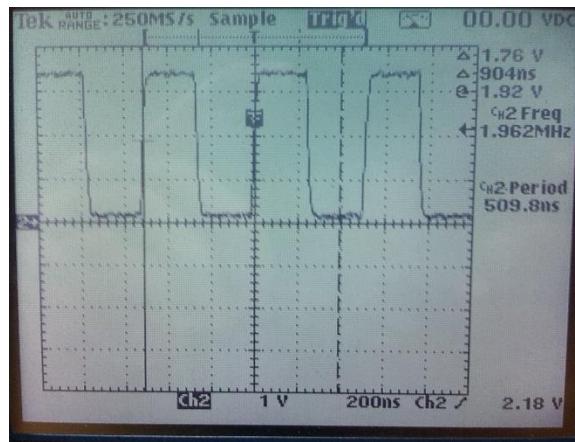


FIGURA 125: SEÑAL CUADRADA 2 MHz

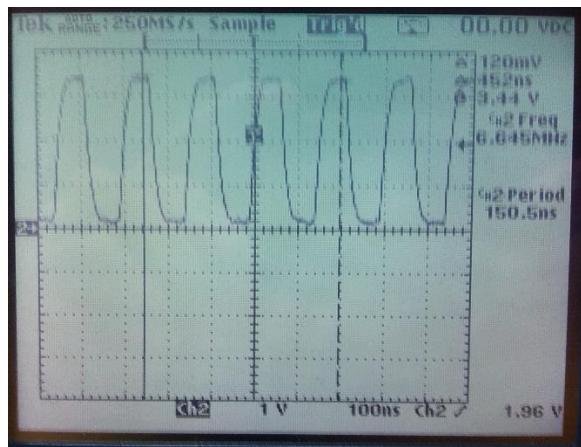


FIGURA 126: SEÑAL CUADRADA 7 MHZ

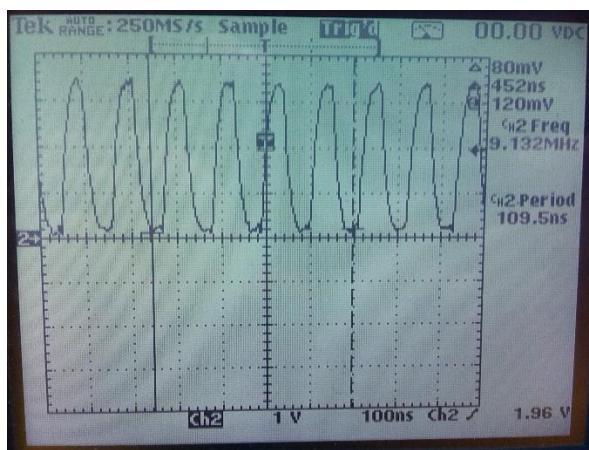


FIGURA 127: SEÑAL CUADRADA 10 MHZ

5.3.3.- Generador de funciones no cuadradas

En este caso se produce tanto deformación a baja frecuencia (causada por la discretización de valores) como a alta frecuencia. En este caso consideramos alta frecuencia a más de 10 KHz, que es la máxima frecuencia de salida que puede tener la señal no cuadrada. Si se supera dicha frecuencia no se estaría pasando por todos los valores de la RAM de la unidad de comparación y captura, y en consecuencia la deformación sería muy elevada. A continuación se muestra la deformación sobre una señal de tipo triangular.

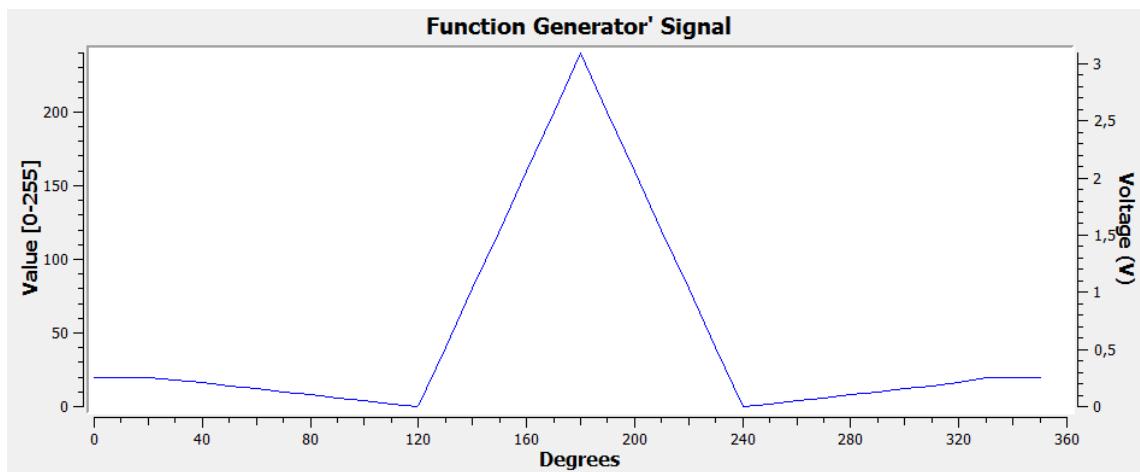


FIGURA 128: SEÑAL TRIANGULAR EN LA APLICACIÓN

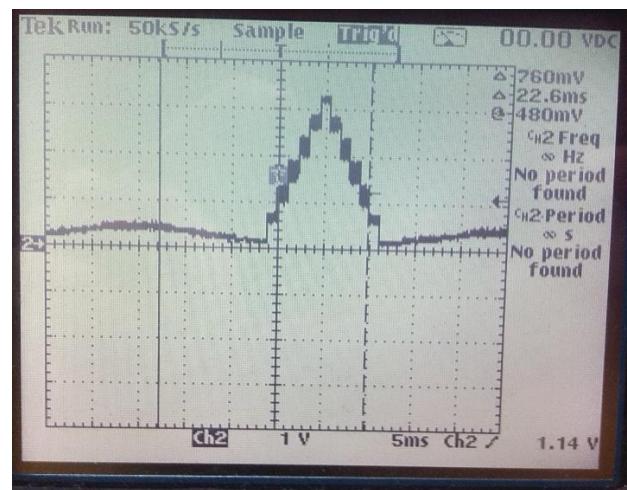


FIGURA 129: SEÑAL TRIANGULAR DE 150 Hz

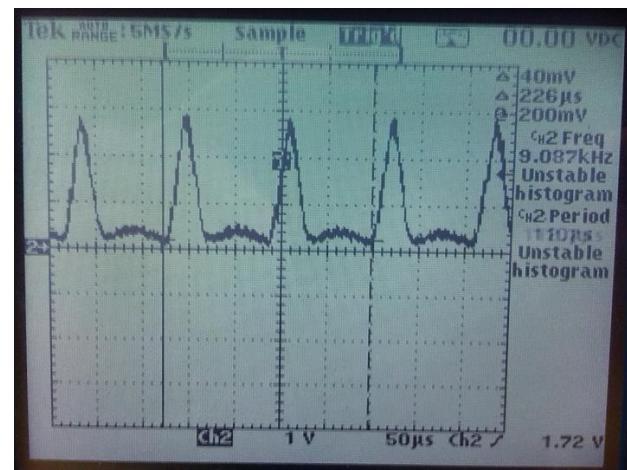


FIGURA 130: SEÑAL TRIANGULAR DE 9 KHz

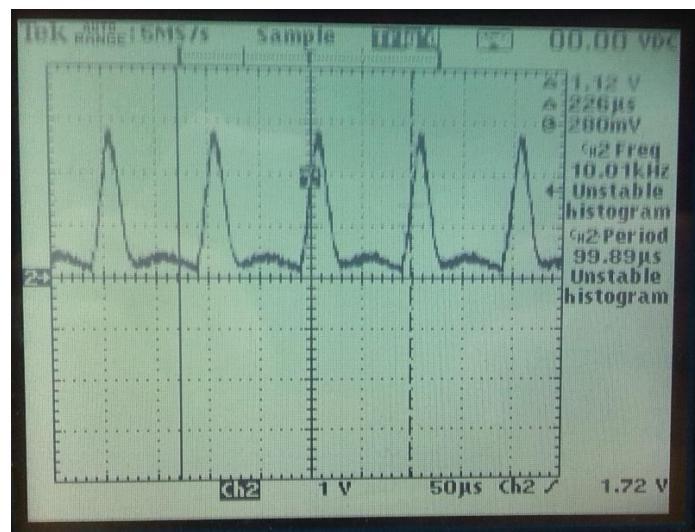


FIGURA 131: SEÑAL TRIANGULAR DE 10 KHz

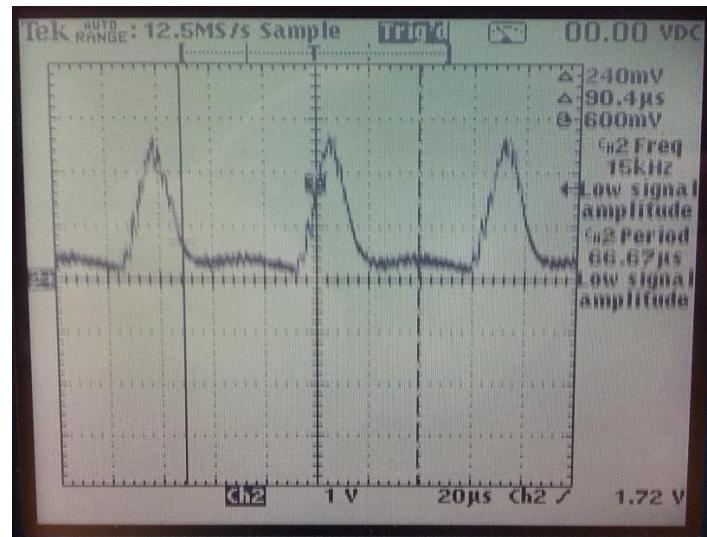


FIGURA 132: SEÑAL TRIANGULAR DE 15 KHz

En la señal de 15 KHz se observan escalones que son debidos a que, ante una frecuencia mayor de 10 KHz, no se está pasando por todos los valores de la RAM de la unidad de comparación y captura.

CAPÍTULO 6: CONCLUSIONES

6.1.- Conclusiones

En este proyecto se ha realizado un control remoto por Ethernet de una serie de instrumentos implementados en una FPGA. Para ello hemos creado una aplicación cliente-servidor UDP que nos permite comunicar en tiempo real al PC con la placa de desarrollo a través de la red.

Cabe destacar las siguientes conclusiones referentes a diferentes partes del proyecto:

- Instrumentos: hemos mejorado la precisión del frecuencímetro propuesto por [9] y hemos añadido más opciones de configuración tanto para el frecuencímetro como para el generador de funciones (ver apartado [3.7.4.- Instrumentos de medida](#))
- Sistema operativo (SO): se ha probado a eliminar el sistema operativo para el control de los instrumentos (sin la conexión Ethernet activada) y no se han encontrado diferencias de rendimiento. Por este motivo se ha decidido mantener el SO ya que facilita mucho la programación, principalmente la gestión de interrupciones, y sobre todo la gestión de tiempo del procesador una vez que incorporamos la tarea que controla la conexión Ethernet.
- Se ha creado una aplicación de tipo cliente servidor UDP (ver apartados [3.7.3.- Diagrama de funcionamiento](#) a [3.7.5.- Explicación del código](#), y [4.5.- Diseño de la aplicación de usuario](#)) que permite transmitir y recibir datos entre un ordenador y la placa de desarrollo, todo ello realizado el objetivo presente de la escalabilidad, es decir, que el proyecto se pueda ampliar a otras necesidades futuras.

El proyecto es muy completo ya que abarca conocimientos multidisciplinares dentro del ámbito de las telecomunicaciones. Desde el desarrollo hardware en bajo nivel en la placa de desarrollo, pasando por las comunicaciones de red, hasta el desarrollo software de alto nivel (Python).

Hemos adquirido, por tanto, conocimientos y habilidades de diversos ámbitos como: las ventajas y la versatilidad que ofrece trabajar con procesadores embebidos, la creación de sistemas embebidos a medida, el funcionamiento e

implementación de comunicaciones en red, y el desarrollo de aplicaciones de escritorio.

6.2.- Propuestas de trabajo futuro

El proyecto es muy versátil. Se ha creado para que sea posible ampliarlo, tanto desde el punto de vista de la placa de desarrollo, como desde el punto de vista de la aplicación de usuario. Por ello, se pueden incorporar nuevas funcionalidades orientadas a la mejora de distintos aspectos. El autor propone lo siguiente:

- Introducir un filtro (de tipo *notch*) reconfigurable de forma remota para la salida de la señal no cuadrada, con el objetivo de filtrar la señal de PWM de frecuencia 36 veces la señal deseada. De este modo se obtendría una señal más limpia.
- Tratar de introducir un nuevo sistema operativo (por ejemplo micro Linux) y analizar y comparar los posibles cambios de rendimiento.
- Introducir el protocolo DHCP para que la asignación de dirección IP se realice de forma dinámica por el *router* DHCP y no de forma estática por el usuario.
- Introducir nuevos instrumentos en la placa, y diseñar nuevos comandos adaptados a dichos instrumentos.

Capítulo 7: MANUAL DE USUARIO

Se ha decidido incluir este apartado para disponer de un documento de referencia rápida en donde se describa qué tiene que hacer usuario para poner en marcha el sistema, tanto en el servidor como en el cliente.

7.1.- Servidor

En los ficheros *frec_gen.h*, y *DM9000A.h* se deben introducir las direcciones IP y MAC deseadas. Luego se conecta el cable Ethernet entre la placa y el *switch* o el *router* de la red. Posteriormente se configura (*Quartus II Programmer*) y programa (*Eclipse*) la placa.

Se deben realizar el siguiente conexionado de cables de los instrumentos:

- Frecuencímetro: utilizamos el bus de expansión GPIO_0. La masa de la señal se conecta al sexto pin de la segunda columna, y la señal al quinto pin de la primera columna.
- Generador de funciones: utilizamos el bus de expansión GPIO_1. La masa se conecta al sexto pin de la segunda columna y la señal al octavo pin también de la segunda columna.

Desde ese momento el servidor estará funcionando (por defecto en modo frecuencímetro).

7.2.- Cliente

En el directorio donde tengamos guardados los ficheros *main.pyw*, *vi_protocol.py* y *window.py* debemos hacer doble clic en *main.pyw*. Se nos abrirá la aplicación y ya podemos empezar a interactuar con el servidor. Para ello se deben seguir los siguientes pasos:

1. Introducir la IP y puerto del servidor y hacer clic en el botón *Connect*.
2. Seleccionar el instrumento y su configuración:
 - Frecuencímetro: introducimos el número de nodos y tipo de promediado deseado.

- Generador de funciones: elegimos entre señal cuadrada o señal no cuadrada.
 - Señal cuadrada: introducimos la frecuencia y el ciclo de trabajo.
 - Señal no cuadrada: seleccionamos la señal deseada haciendo clic en **File -> Load Signal** e introducimos la frecuencia.
3. Pulsar el botón de configuración (“Configure Frecuency Meter”, o “Configure Function Generator” en función del instrumento elegido). Una vez hecho esto el servidor quedará configurado con dicha configuración.
 4. En el caso del frecuencímetro se puede solicitar la frecuencia actual pulsando “Frequency Request”. Existen dos variantes:
 - *Auto Request* activado: se envían ‘samples’ solicitudes de frecuencia al servidor con un intervalo entre peticiones de ‘Interval samples’ segundos.
 - *Auto Request* desactivado: se envía una petición cada vez que se pulsa “Frequency Request”.

Las medidas de frecuencia se almacenan en una tabla, que se puede guardar como un fichero *.csv seleccionando la opción disponible a tal efecto en la barra de menú **File -> Save Table**. Además, dicha tabla se puede ser borrada haciendo clic en **File -> Clear Table**.

APÉNDICES

Apéndice 1: código fuente de la placa de desarrollo

Apéndice 1A: VI Protocol (*vi_protocol.h*)

```
#ifndef __VI_PROTOCOL_H__
#define __VI_PROTOCOL_H__


/*
 * File: VI_protocol.h
 * Date (last revision): 4/11/2014
 *
 * Description: header and definition of functions, structs, constants... of the
associated file VI_protocol.c
 *
 */
//


//-----//
// Protocol's constants


// Client (PC) Commands
#define CMD ACTIVATE FREQ METER 100
#define CMD FREQ REQUEST 101
#define CMD NFREQ NODES 102
#define CMD AV SIMPLE 103
#define CMD_AV_WEIGHTED 104
#define CMD_AV_EXP 105
#define CMD ACTIVATE FUNC GEN 106
#define CMD RAM VALUES 107
#define CMD_FG_SQUARE 108
#define CMD_FG_NON_SQUARE 109
#define CMD_FG_CHANGE_FREQ 110
#define CMD_FG_CHANGE_TON 111
#define CMD_RESET 112

// Server (board) Commands
#define CMD_FREQ_ANSWER 113

//-----//
// VI protocol (application protocol over UDP) to control the board's instrumentation
typedef struct VI_protocol
{
    int command;
    char data[36];
}VI_protocol_t;

#endif
```

Apéndice 1B: Cabecera del Controlador Ethernet (DM9000A.h)

```

#ifndef      DM9000A_H
#define      __DM9000A_H__


#define IO_addr      0
#define IO_data      1

#define NCR          0x00 /* Network Control Register REG. 00 */
#define NSR          0x01 /* Network Status Register REG. 01 */
#define TCR          0x02 /* Transmit Control Register REG. 02 */
#define RCR          0x05 /* Receive Control Register REG. 05 */
#define ETXCSR        0x30 /* TX early Control Register REG. 30 */
#define MRCMDX       0xF0 /* RX FIFO I/O port command READ for dummy
                           read a byte from RX SRAM */
#define MRCMD        0xF2 /* RX FIFO I/O port command READ from RX
                           SRAM */
#define MWCMD        0xF8 /* TX FIFO I/O port command WRITE into TX
                           FIFO */
#define ISR          0xFE /* NIC Interrupt Status Register REG. FEH */
#define IMR          0xFF /* NIC Interrupt Mask Register REG. FFH */

#define NCR_set       0x00
#define TCR_set       0x00
#define TX_REQUEST    0x01 /* TCR REG. 02 TXREQ Bit [0] = 1 polling
                           Transmit Request command */
#define TCR_long      0x40 /* packet disable TX Jabber Timer */
#define RCR_set       0x30 /* skip CRC_packet and skip LONG_packet */
#define RX_ENABLE     0x01 /* RCR REG. 05 RXEN Bit [0] = 1 to
                           enable RX machine */
#define RCR_long      0x40 /* packet disable RX Watchdog Timer */
#define PASS_MULTICAST 0x08 /* RCR REG. 05 PASS ALL MULTICAST Bit
                           [3] = 1: RCR set value ORed 0x08 */
#define BPTR_set      0x3F /* BPTR REG. 08 RX Back Pressure
                           Threshold: High Water Overflow
                           Threshold setting 3KB and
                           Jam_Pattern_Time = 600 us */
#define FCTR_set      0x5A /* FCTR REG. 09 High/ Low Water Overflow
                           Threshold setting 5KB/ 10KB */
#define RTFCR_set     0x29 /* RTFCR REG. 0AH RX/TX Flow Control
                           Register enable TXPEN + BKPM(TX_Half) +
                           FILCE(RX) */
#define ETXCSR_set    0x83 /* Early Transmit Bit [7] Enable and
                           Threshold 0~3: 12.5%, 25%, 50%,
                           75% */
#define INTR_set      0x81 /* IMR REG. FFH: PAR +PRM, or 0x83: PAR
                           + PRM + PTM */
#define PAR_set       0x80 /* IMR REG. FFH: PAR only, RX/TX FIFO
                           R/W Pointer Auto Return enable */

#define PHY_reset     0x8000 /* PHY reset: some registers back to
                           default value */
#define PHY_txab     0x05e1 /* set PHY TX advertised ability:
                           Full-capability + Flow-control (if
                           necessary) */
#define PHY_mode      0x3100 /* set PHY media mode: Auto
                           negotiation (AUTO sense) */

#define STD_DELAY     20 /* standard delay 20 us */

#define DMFE_SUCCESS  0
#define DMFE_FAIL     1

#define TRUE          1
#define FALSE         0

#define DM9000_PKT_READY 0x01 /* packets ready to receive */
#define PACKET_MIN_SIZE 0x40 /* Received packet min size */
#define MAX_PACKET_SIZE 1522 /* RX largest legal size packet with
                           fcs & QoS */
#define DM9000_PKT_MAX 3072 /* TX 1 packet max size without 4-byte
                           CRC */

//-----
unsigned char ether_addr[6]={ 0x00, 0x80, 0x2F, 0x14, 0x67, 0xA3 };

```

```
-----  
void      iow(unsigned int reg, unsigned int data);  
unsigned int ior(unsigned int reg);  
void      phy_write(unsigned int reg, unsigned int value);  
/* DM9000_init I/O routine */  
unsigned int DM9000_init (void);  
/* Transmit One Packet TX I/O routine */  
unsigned int TransmitPacket(unsigned char *data_ptr,unsigned int  
                           tx_len);  
/* Receive One Packet I/O routine */  
unsigned int ReceivePacket (unsigned char *data_ptr,unsigned int  
                           *rx_len);  
-----  
#endif
```

Apéndice 1C: Controlador Ethernet (DM9000A.c)

```

#include <stdio.h>
#include "DM9000A.H"
#include "system.h"
#include <io.h>
#include <stdio.h>

#define msleep(msec)                               usleep(1000*msec);
//-----
void iow(unsigned int reg, unsigned int data)
{
    IOWR(DM9000A_BASE,IO_addr,reg);
    usleep(STD_DELAY);
    IOWR(DM9000A_BASE,IO_data,data);
}
//-----
unsigned int ior(unsigned int reg)
{
    IOWR(DM9000A_BASE,IO_addr,reg);
    usleep(STD_DELAY);
    return IORD(DM9000A_BASE,IO_data);
}
//-----
void phy_write (unsigned int reg, unsigned int value)
{
    /* set PHY register address into EPAR REG. 0CH */
    iow(0x0C, reg | 0x40);                      /* PHY register address setting,
                                                and DM9000_PHY offset = 0x40 */

    /* fill PHY WRITE data into EPDR REG. 0EH & REG. 0DH */
    iow(0x0E, ((value >> 8) & 0xFF));        /* PHY data high_byte */
    iow(0x0D, value & 0xFF);                    /* PHY data low_byte */

    /* issue PHY + WRITE command = 0xa into EPCR REG. 0BH */
    iow(0x0B, 0x8);                            /* clear PHY command first */
    IOWR(DM9000A_BASE, IO_data, 0x0A);          /* issue PHY + WRITE command */
    usleep(STD_DELAY);
    IOWR(DM9000A_BASE, IO_data, 0x08);          /* clear PHY command again */
    usleep(50);                                /* wait 1~30 us (>20 us) for PHY + WRITE completion */
}
//-----
/* DM9000 init I/O routine */
unsigned int DM9000_init (void) /* initialize DM9000 LAN chip */
{
    unsigned int i;

    /* set the internal PHY power-on (GPIOs normal settings) */
    iow(0x1E, 0x01); /* GPCR REG. 1EH = 1 selected GPIO0 "output" port
                       for internal PHY */
    iow(0x1F, 0x00); /* GPR REG. 1FH GEPIO0 Bit [0] = 0 to activate
                       internal PHY */
    msleep(5);        /* wait > 2 ms for PHY power-up ready */

    /* software-RESET NIC */
    iow(NCR, 0x03); /* NCR REG. 00 RST Bit [0] = 1 reset on, and LBK
                      Bit [2:1] = 01b MAC loopback on */
    usleep(20);       /* wait > 10us for a software-RESET ok */
    iow(NCR, 0x00); /* normalize */
    iow(NCR, 0x03);
    usleep(20);
    iow(NCR, 0x00);

    /* set GPIO0=1 then GPIO0=0 to turn off and on the internal PHY */
    iow(0x1F, 0x01); /* GPR PHYPD Bit [0] = 1 turn-off PHY */
    iow(0x1F, 0x00); /* PHYPD Bit [0] = 0 activate phyxcer */
    msleep(10);       /* wait >4 ms for PHY power-up */

    /* set PHY operation mode */

    phy_write(0,PHY_reset); /* reset PHY: registers back to the
                           default states */
    usleep(50);            /* wait >30 us for PHY software-RESET ok*/
    phy_write(16, 0x404);  /* turn off PHY reduce-power-down mode

```

```

                only */
phy_write(4, PHY_txab); /* set PHY TX advertised ability: ALL +
                           Flow control */
phy_write(0, 0x1200); /* PHY auto-NEGO re-start enable
                        (RESTART_AUTO_NEGOTIATION +
                         AUTO_NEGOTIATION_ENABLE) to auto sense
                        and recovery PHY registers */
msleep(5); /* wait >2 ms for PHY auto-sense linking to
            partner */

/* store MAC address into NIC */
for (i = 0; i < 6; i++)
iow(16 + i, ether_addr[i]);

/* clear any pending interrupt */
iow(ISR, 0x3F); /* clear the ISR status: PRS, PTS, ROS, ROOS 4
                   bits, by RW/C1 */
iow(NSR, 0x2C); /* clear the TX status: TX1END, TX2END, WAKEUP 3
                   bits, by RW/C1 */

/* program operating registers~ */
iow(NCR, NCR_set); /* NCR REG. 00 enable the chip functions (and
                      disable this MAC loopback mode back to
                      normal) */
iow(0x08, BPTR_set); /* BPTR REG.08 (if necessary) RX Back
                      Pressure Threshold in Half duplex mode only:
                      High Water 3KB, 600 us */
iow(0x09, FCTR_set); /* FCTR REG.09 (if necessary) Flow Control
                      Threshold setting High/ Low Water Overflow
                      5KB/ 10KB */
iow(0x0A, RTFCR_set); /* RTFCR REG.0AH (if necessary) RX/TX Flow
                      Control Register enable TXPEN, BKPM
                      (TX_Half), FLCE (RX) */
iow(0x0F, 0x00); /* Clear the all Event */
iow(0x2D, 0x80); /* Switch LED to mode 1 */

/* set other registers depending on applications */
iow(ETXCSR, ETXCSR_set); /* Early Transmit 75% */

/* enable interrupts to activate DM9000 ~on */
iow(IMR, INTR_set); /* IMR REG. FFH PAR=1 only, or + PTM=1& PRM=1
                      enable RxTx interrupts */

/* enable RX (Broadcast/ ALL MULTICAST) ~go */
iow(RCR, RCR_set | RX_ENABLE | PASS_MULTICAST); /* RCR REG. 05
                                                       RXEN Bit [0] = 1 to enable the RX machine/ filter */

/* RETURN "DEVICE SUCCESS" back to upper layer */
return (ior(0x2D)==0x80) ? DMFE_SUCCESS : DMFE_FAIL;
}

//-----
/* Transmit one Packet TX I/O routine */
unsigned int TransmitPacket(unsigned char *data_ptr,unsigned int tx_len)
{
    unsigned int i;

    /* mask NIC interrupts IMR: PAR only */
    iow(IMR, PAR_set);

    /* issue TX packet's length into TXPLH REG. FDH & TXPLL REG. FCH */
    iow(0xFD, (tx_len >> 8) & 0xFF); /* TXPLH High_byte length */
    iow(0xFC, tx_len & 0xFF); /* TXPLL Low_byte length */

    /* write transmit data to chip SRAM */
    IOWR(DM9000A_BASE, IO_addr, MWCMD); /* set MWCMD REG. F8H TX I/O port ready */
    for (i = 0; i < tx_len; i += 2)
    {
        usleep(STD_DELAY);
        IOWR(DM9000A_BASE, IO_data, (data_ptr[i+1]<<8)|data_ptr[i] );
    }

    /* issue TX polling command activated */
    iow(TCR, TCR_set | TX_REQUEST); /* TXCR Bit [0] TXREQ auto clear
                                         after TX completed */

    /* wait TX transmit done */
    while(!(ior(NSR)&0x0C))
}

```

```

usleep(STD_DELAY);

/* clear the NSR Register */
iow(NSR,0x00);

/* re-enable NIC interrupts */
iow(IMR, INTR_set);

/* RETURN "TX SUCCESS" to upper layer */
return DMFE_SUCCESS;
}

//-----
/* Receive One Packet I/O routine */
unsigned int ReceivePacket (unsigned char *data_ptr,unsigned int *rx_len)
{
    unsigned char rx_READY,GoodPacket;
    unsigned int Tmp, RxStatus, i;

    RxStatus = rx_len[0] = 0;
    GoodPacket=FALSE;

    /* mask NIC interrupts IMR: PAR only */
    iow(IMR, PAR_set);

    /* dummy read a byte from MRCMDX REG. F0H */
    rx_READY = ior(MRCMDX);

    /* got most updated byte: rx_READY */
    rx_READY = IORD(DM9000A_BASE,IO_data)&0x03;
    usleep(STD_DELAY);

    /* check if (rx READY == 0x01): Received Packet READY? */
    if (rx_READY == DM9000_PKT_READY)
    {
        /* got RX_Status & RX_Length from RX SRAM */
        IOWR(DM9000A_BASE, IO_addr, MRCMD); /* set MRCMD REG. F2H RX I/O
                                                port ready */

        usleep(STD_DELAY);
        RxStatus = IORD(DM9000A_BASE,IO_data);
        usleep(STD_DELAY);
        rx_len[0] = IORD(DM9000A_BASE,IO_data);

        /* Check this packet status GOOD or BAD? */
        if ( !(RxStatus & 0xBF00) && (rx_len[0] < MAX_PACKET_SIZE) )
        {
            /* read 1 received packet from RX SRAM into RX buffer */
            for (i = 0; i < rx_len[0]; i += 2)
            {
                usleep(STD_DELAY);
                Tmp = IORD(DM9000A_BASE, IO_data);
                data_ptr[i] = Tmp&0xFF;
                data_ptr[i+1] = (Tmp>>8)&0xFF;
            }
            GoodPacket=TRUE;
        } /* end if (GoodPacket) */
        else
        {
            /* this packet is bad, dump it from RX SRAM */
            for (i = 0; i < rx_len[0]; i += 2)
            {
                usleep(STD_DELAY);
                Tmp = IORD(DM9000A_BASE, IO_data);
            }
            printf("\nError\n");
            rx_len[0] = 0;
        } /* end if (!GoodPacket) */
    } /* end if (rx_READY == DM9000_PKT_READY) ok */
    else if(rx_READY) /* status check first byte: rx_READY Bit[1:0] must
                      be "00"b or "01"b */
    {
        /* software-RESET NIC */
        iow(NCR, 0x03); /* NCR REG. 00 RST Bit [0] = 1 reset on, and LBK
                           Bit [2:1] = 01b MAC loopback on */
        usleep(20); /* wait > 10us for a software-RESET ok */
        iow(NCR, 0x00); /* normalize */
        iow(NCR, 0x03);
        usleep(20);
    }
}

```

```

iow(NCR, 0x00);
/* program operating registers~ */
iow(NCR, NCR_set); /* NCR REG. 00 enable the chip functions
                     (and disable this MAC loopback mode back
                     to normal) */
iow(0x08, BPTR_set); /* BPTR REG.08 (if necessary) RX Back
                     Pressure Threshold in Half duplex mode
                     only: High Water 3KB, 600 us */
iow(0x09, FCTR_set); /* FCTR REG.09 (if necessary) Flow Control
                     Threshold setting High/ Low Water
                     Overflow 5KB/ 10KB */
iow(0x0A, RTFCR_set); /* RTFCR REG.0AH (if necessary) RX/TX Flow
                     Control Register enable TXPEN, BKPM
                     (TX Half), FLCE (RX) */
iow(0x0F, 0x00); /* Clear the all Event */
iow(0x2D, 0x80); /* Switch LED to mode 1 */
/* set other registers depending on applications */
iow(ETXCSR, ETXCSR_set); /* Early Transmit 75% */
/* enable interrupts to activate DM9000 ~on */
iow(IMR, INTR_set); /* IMR REG. FFH PAR=1 only, or + PTM=1&
                     PRM=1 enable RxTx interrupts */
/* enable RX (Broadcast/ ALL MULTICAST) ~go */
iow(RCR, RCR_set | RX_ENABLE | PASS_MULTICAST); /* RCR REG. 05
                     RXEN Bit [0] = 1 to enable the RX machine/ filter */
} /* end NIC H/W system Data-Bus error */

    return GoodPacket ? DMFE_SUCCESS : DMFE_FAIL;
}
//-----

```

Apéndice 1D: Cabecera del programa principal (*freq_gen.h*)

```

// Destination UDP port
#define DEFAULT_DEST_PORT_H 0x00 // UDP 80 = 0x0050
#define DEFAULT_DEST_PORT_L 0x50

#define DEFAULT_UDP_LENGTH_H ((UDP_HEADER_LENGTH +  

                           UDP_DATA_LENGTH) >> 8)

#define DEFAULT_UDP_LENGTH_L ((UDP_HEADER_LENGTH +  

                           UDP_DATA_LENGTH) & 0x00FF)
0x00
0x00

// ----- IP header -----
#define IP_HEADER_LENGTH 0x14

#define DEFAULT_VERSION_IHL 0x45
#define DEFAULT_DSCP_ECN 0x00
#define DEFAULT_IP_LENGTH_H ((IP_HEADER_LENGTH +  

                           UDP_HEADER_LENGTH +  

                           UDP_DATA_LENGTH) >> 8)

#define DEFAULT_IP_LENGTH_L ((IP_HEADER_LENGTH +  

                           UDP_HEADER_LENGTH +  

                           UDP_DATA_LENGTH) & 0x00FF)
0x00
0x00
0x40 // Don't Fragment the packet

#define DEFAULT_FLAGS_OFFSET_H 0x00
#define DEFAULT_TTL 0x80
#define UDP_PROTOCOL 0x11
#define DEFAULT_PROTOCOL UDP_PROTOCOL
#define DEFAULT_IP_CRC_H 0x00
#define DEFAULT_IP_CRC_L 0x00
// Source (FPGA) IP Address default: 172.19.5.91 (Fixed assignment)
#define DEFAULT_IP_SOURCE_3 0xAC
#define DEFAULT_IP_SOURCE_2 0x13
#define DEFAULT_IP_SOURCE_1 0x05
#define DEFAULT_IP_SOURCE_0 0x5B
// Destination IP Address. If it's unknown -> broadcast address
#define DEFAULT_IP_DEST_3 0xAC
#define DEFAULT_IP_DEST_2 0x13
#define DEFAULT_IP_DEST_1 0x05
#define DEFAULT_IP_DEST_0 0x63

// ----- Ethernet header -----
#define ETHERNET_HEADER_LENGTH 0x0E // Ethernet header  

                                length = d'14 = h'0x0e
#define MAX_ETHERNET_LENGTH 1458
// Destination MAC. If it's unknown -> broadcast address
#define DEFAULT_DEST_MAC_5 0xff
#define DEFAULT_DEST_MAC_4 0xff
#define DEFAULT_DEST_MAC_3 0xff
#define DEFAULT_DEST_MAC_2 0xff
#define DEFAULT_DEST_MAC_1 0xff
#define DEFAULT_DEST_MAC_0 0xff
// Source (FPGA) MAC. By default, I consider 00:80:2f:14:67:a3
#define DEFAULT_SOURCE_MAC_5 0x00
#define DEFAULT_SOURCE_MAC_4 0x80
#define DEFAULT_SOURCE_MAC_3 0x2f
#define DEFAULT_SOURCE_MAC_2 0x14
#define DEFAULT_SOURCE_MAC_1 0x67
#define DEFAULT_SOURCE_MAC_0 0xa3
// Ethertype/length
#define DEFAULT_ETHERTYPE_H 0x08 // It means that there  

                                is an IPv4 ...
#define DEFAULT_ETHERTYPE_L 0x00 // ... packet  

                                encapsulated in  

                                the frame

#define PKT_LENGTH (ETHERNET_HEADER_LENGTH +  

               IP_HEADER_LENGTH + UDP_HEADER_LENGTH + UDP_DATA_LENGTH)

typedef struct udp_header
{

```

```

    char source_port_h;           // Source port
    char source_port_l;
    char dest_port_h;            // Destination port
    char dest_port_l;
    char udplenlength_h;         // Length
    char udplenlength_l;
    char crc_h;                  // Checksum
    char crc_l;
    char data[UDP_DATA_LENGTH];   // Data

}udp_header_t;

typedef struct ip_header
{
    char version_ihl;           // Version = 4bits ; IHL = 4bits
    char dscp_ecn;              // DSCP = 6bits ; ECN = 2bits
    char iplength_h;            // IP length
    char iplength_l;
    char id_h;
    char id_l;                  // Identification
    char flags_offset_h;         // Flags = 3bits ; Fragment offset=3bits
    char flags_offset_l;
    char ttl;                   // Time to live
    char protocol;
    char ipcrc_h;
    char ipcrc_l;               // Header checksum
    char ipsource3;             // IP source address
    char ipsource2;
    char ipsource1;
    char ipsource0;
    char ipdest3;               // IP destination address
    char ipdest2;
    char ipdest1;
    char ipdest0;

    struct udp_header udp; // Payload of IP packet
}ip_header_t;

typedef struct pkt_buffer
{
    char mac_dest5;             // Destination MAC
    char mac_dest4;
    char mac_dest3;
    char mac_dest2;
    char mac_dest1;
    char mac_dest0;
    char mac_source5;           // Source MAC
    char mac_source4;
    char mac_source3;
    char mac_source2;
    char mac_source1;
    char mac_source0;
    char ethertype_h;           // Ethernet type/length
    char ethertype_l;

    struct ip_header ip; // Payload of Ethernet packet
}pkt_t;

typedef struct arp_header
{
    char hardware_type_h;
    char hardware_type_l;
    char protocol_type_h;
    char protocol_type_l;
    char hardware_size;
    char protocol_size;
    char opcode_h;
    char opcode_l;
    char mac_sender5;
    char mac_sender4;
}

```

```

    char mac_sender3;
    char mac_sender2;
    char mac_sender1;
    char mac_sender0;
    char ipsender3;
    char ipsender2;
    char ipsender1;
    char ipsender0;
    char mac_target5;
    char mac_target4;
    char mac_target3;
    char mac_target2;
    char mac_target1;
    char mac_target0;
    char iptarget3;
    char iptarget2;
    char iptarget1;
    char iptarget0;

}arp_header_t;

typedef struct pkt_buffer_2
{
    char mac_dest5;           // Destination MAC
    char mac_dest4;
    char mac_dest3;
    char mac_dest2;
    char mac_dest1;
    char mac_dest0;
    char mac_source5;         // Source MAC
    char mac_source4;
    char mac_source3;
    char mac_source2;
    char mac_source1;
    char mac_source0;
    char ethertype_h;          // Ethernet type/length
    char ethertype_l;

    struct arp_header arp;// Payload of Ethernet packet
}pkt_buffer_2_t;

//-----// Constants
associated with the board instrumentation

#define FREQUENCY_METER          0      // Frequency meter
#define FUNCTION_GENERATOR        1      // Function generator
#define NUM_FREQ_NODES_DEFAULT    20     // Default number of
                                         frequency nodes (to
                                         do the frequency
                                         average)
#define SIMPLE_MOVING_AVERAGE    0      // Simple moving
                                         average
#define WEIGHTED_MOVING_AVERAGE  1      // Weighted moving
                                         average
#define EXPONENTIAL_MOVING_AVERAGE 2    // Exponential moving
                                         average
#define PERIOD_PWM_DEFAULT       0x00004E20 // Default period
                                         of the PWM signal
#define TON_PWM_DEFAULT           0x00002710 // Default ton of
                                         the PWM signal
#define SQUARE_SIGNAL_ACTIVATED   1      // Square signal
#define NON_SQUARE_SIGNAL_ACTIVATED 0    // Non-Square signal

#define RAM_PWM_SIZE              36     // Size of CCP's RAM
#define LCD_WR_COMMAND_REG        0      // LCD ...
#define LCD_WR_DATA_REG           2      // ... commands

//-----//Definition of Task Priorities
#define VI_TASK_PRIORITY          8
#define RIC_TASK_PRIORITY          7

```

```

//-----
//Definition of Global variables
unsigned int aaa,rx_len,packet_num; // Ethernet ...
unsigned char RXT[MAX_ETHERNET_LENGTH]; // ... variables

int captura_valida = 0; // Used to validate a capture
struct node *first = NULL, *last = NULL; // Pointers to the frequency
                                         list (to calculate the
                                         average)
struct system_config sc; // Node that stores the
                           global system configuration
unsigned long int captura = 1, capturaant = 0, sw = 0;

float frecuencia = 0.0;
int last_instrument = -1;
int num_nodes = 0; // Current number of frequency
                   nodes (used in the
                   initialization of the list)

int custom_signal[36]; // Signal to be stored in the CCP's RAM

//-----/*
* Definition of functions
*/
// Tasks (Threads)
void VITask(void* );
void RICTask(void* );

// LCD
void lcd_init( void );
void lcd_clear( void );
void lcd_write_first_line(char * );
void lcd_write_second_line(char * );

// Frequency list operation
struct node *create(float);
void insert_beginning(float);
void insert_at_end(float);
void delete_beginning( void );
void delete_at_end( void );
void delete_list( void );

// Frequency meter
void frequency_meter( void );
float calculate_new_freq ( void );
float calculate_average_frequency( float );
void show_frequency_lcd( float );

// Function generator
void function_generator( void );
void write_ram_function_generator(int *, int);

// CCP configuration
void reset_ccp( void );
void configuration_ccp_captura_0( void );
void configuration_function_generator_square( void );
void configuration_function_generator_non_square( void );
void configuration_function_generator_ton( void );
void ccp_activate_square( void );
void ccp_activate_non_square( void );

// Other functions related with the instruments
void system_config_init( void );
void assign_interruptions( void );
void decimal_to_binary(long , char *, int);

// Interrupts
void interrupt_captura_0(void* );
void edge_button_interrupts(void* );
void ethernet_interrupts();

// Network connexion
void assign_pkt_defaults( struct pkt_buffer *pkt );

```

```
unsigned int calculate_ip_checksum( struct pkt_buffer *pkt);
unsigned int calculate_udp_checksum( struct pkt_buffer *pkt);
int pkt_is_viprotocol(unsigned char *buffer, int rx_len);
int pkt_is_arprequest(unsigned char *buffer, int rx_len);
void pkt_changes_to_send(struct pkt_buffer *pkt);
void pkt_arp_changes_to_send(struct pkt_buffer_2 *pkt);

#endif /* __FREC_GEN_H__ */
```

Apéndice 1E: Programa principal (*frec_gen.c*)

```
/*
 *
 * File: freq_gen.c
 * Date (lastest version): 7/11/2014
 * Description: This program allows you to control a frequency meter and a function
 * generator that are integrated into the dev board. It can be controlled through commands
 * sent over a UDP connection between the board (server listening at port 80) an a PC
 * (client).
 *
 */
//-----
// Includes
#include "includes.h"
#include "system.h"

#include <unistd.h>
#include <errno.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#include "DM9000A.C"
#include "frec_gen.h"
#include "VI_protocol.h"

#include <alt_types.h>

#include <sys/alt_irq.h>
#include <altera_avalon_pio_regs.h>
#include <altera_avalon_lcd_16207.h>

//-----
//Definition of Task Stacks
#define TASK_STACKSIZE 2048
OS_STK VITask_stk[TASK_STACKSIZE];
OS_STK RICTask_stk[TASK_STACKSIZE];

/////////////////////////////// MAIN /////////////////////////////////
//int main()
{
    // Create and launch the threads
    OSTaskCreateExt(VITask,
                    NULL,
                    (void *)&VITask_stk[TASK_STACKSIZE-1],
                    VI_TASK_PRIORITY,
                    VI_TASK_PRIORITY,
                    VITask_stk,
                    TASK_STACKSIZE,
                    NULL,
                    0);

    OSTaskCreateExt(RICTask,
                    NULL,
                    (void *)&RICTask_stk[TASK_STACKSIZE-1],
                    RIC_TASK_PRIORITY,
                    RIC_TASK_PRIORITY,
                    RICTask_stk,
                    TASK_STACKSIZE,
                    NULL,
                    0);
}
```

```

OSStart(); // Operative System: to manage the threads

while(1); /*In normal execution this is never executed */
return(0);
}

////////////////////////////// THREADS CODE //////////////////
/* Thread 1: VITask */
/* We call the instrument's function that is activated in 'sc.activated_device' */
void VITask(void* pdata)
{
    first = last = NULL; // pointer to the list of frequency
    lcd_init();
    system_config_init();
    assign_interruptions(); //assign the interruptions to a specific
                           // ISR

    while(1) {

        if (sc.activated_device == FREQUENCY_METER){ // Frequency Meter activated
            frequency_meter();
            IOWR(PIO_OUT_GREEN_BASE, 0, 0x0001);
            // LEDG0 of the dev board is ON
            IOWR(PIO_OUT_RED_BASE, 0, 0x0000);
            // Clear Red leds (info of the Function Generator)

        } else if (sc.activated_device == FUNCTION_GENERATOR){
            // Function Generator activated
            function_generator();
            IOWR(PIO_OUT_GREEN_BASE, 0, 0x0002);
            // LEDG1 of the dec board isON
        }
    }
}

/* Thread 2: RICTask */
/* We receive packets via Ethernet connection */
void RICTask(void* pdata)
{
    unsigned int num_packets;

    while(DM9000_init() != DMFE_SUCCESS); //initialization of the
                                         // Ethernet controller

    // Register the ISR after the initialization
    alt_ic_isr_register(DM9000A_IRQ_INTERRUPT_CONTROLLER_ID, DM9000A_IRQ, (void
*)ethernet_interrupts , NULL, NULL);

    /*
     *      We receive 'num packets', then we suspend the task for
     *          100ms. Then we receive 'num_packets' ..., and so on.
     */
    num_packets = 25; // burst of packets to be read each time
    while (1) {

        while (num_packets != 0) {
            TransmitPacket(0,0x00); // clear the Ethernet Flag,
                                   // so we can receive other packet
            num_packets--;
        }
        num_packets = 25;
        OSTimeDlyHMSM(0, 0, 0, 100); // the task is suspended for
    }
}

```

```

        100ms, and we execute the
        VITask for that time.
    // As VITask has less
    priority than RICTask, we
    need to suspend RICTask
    in order to execute
    VITask.
}

///////////////////////////////////////////////////////////////////
//          LCD FUNCTIONS
///////////////////////////////////////////////////////////////////

/* To initialize the LCD */
void lcd_init( void )
{
    /* Set Function Code Four Times -- 8-bit, 2 line, 5x7 mode */
    IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x38 );
    usleep(4100); /* Wait 4.1 ms */
    IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x38 );
    usleep(100); /* Wait 100 us */
    IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x38 );
    usleep(5000); /* Wait 5.0 ms */
    IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x38 );
    usleep(100);

    /* Set Display to OFF */
    IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x08 );
    usleep(100);

    /* Set Display to ON */
    IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x0C );
    usleep(100);

    /* Set Entry Mode -- Cursor increment, display doesn't shift */
    IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x06 );
    usleep(100);

    /* Set the cursor to the home position */
    IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x02 );
    usleep(2000);

    /* Clear the display */
    IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x01 );
    usleep(2000);
}

/* To clear the LCD */
void lcd_clear( void )
{
    IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x02 ); //cursor to home
    usleep(2000);

    IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x01 ); //clear display
    usleep(2000);
}

/* Write a message in the first line of the LCD */
void lcd_write_first_line(char *message)
{
    char final_message[ALT_LCD_WIDTH]; //ALT_LCD_WIDTH = 16
    int i=0;
    IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x02 ); //cursor to home
    usleep(2000);

    if (strlen(message) > ALT_LCD_WIDTH) printf("ERROR: Input not valid on the first
line of the LCD");
    else {
        strncpy(final_message, message, strlen(message));
        // Rest of the line is filled with the ' ' space symbol
    }
}

```

```

        for (i = strlen(message); i <= ALT_LCD_WIDTH; i++) {
            final_message[i] = ' ';
        }
        for( i = 0; i < strlen(final_message) ; i++ ) { //show
            IOWR( LCD_BASE, LCD_WR_DATA_REG, final_message[i] );
            usleep(50);
        }
    }
}

/* Write a message in the second line of the LCD */
void lcd_write_second_line(char *message)
{
    char final_message[ALT_LCD_WIDTH]; //ALT_LCD_WIDTH = 16
    int i=0;

    //IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0x02 ); //cursor to home
    //usleep(2000);

    IOWR( LCD_BASE, LCD_WR_COMMAND_REG, 0xC0 ); // change of line:
                                                    to the second line
    usleep(100);

    if (strlen(message) > ALT_LCD_WIDTH) printf("ERROR: Input not valid on the second
line of the LCD");
    else {
        strncpy(final_message, message, strlen(message));

        // Rest of the line is filled with the ' ' space symbol
        for (i = strlen(message); i <= ALT_LCD_WIDTH; i++) {
            final_message[i] = ' ';
        }
        for( i = 0; i < strlen(final_message) ; i++ ) { //show
            IOWR( LCD_BASE, LCD_WR_DATA_REG, final_message[i] );
            usleep(50);
        }
    }
}

////////////////////////////////////////////////////////////////
//          FUNCTIONS TO OPERATE WITH THE FREQUENCY LIST      //
////////////////////////////////////////////////////////////////

/* TO create an empty node */
struct node *create(float data)
{
    struct node *temp;

    temp = (struct node *)malloc(sizeof(struct node));
    temp->prev = NULL;
    temp->next = NULL;
    temp->frequency = data;
    num_nodes++;
    return temp;
}

/* TO insert at beginning */
void insert_beginning(float data)
{
    struct node *temp;

    if (first == NULL) { //the list is empty
        first = create(data);
        last = first;
    } else { //the list is not empty
        temp = create(data);
        temp->next = first;
        first->prev = temp;
        first = temp;
    }
}

```

```

}

/* To insert at end */
void insert_at_end(float data)
{
    struct node *temp;

    if (first == NULL) { //the list is empty
        first = create(data);
        last = first;
    } else { //the list is not empty
        temp = create(data);
        last->next = temp;
        temp->prev = last;
        last = temp;
    }
}

/* To delete the first element */
void delete_beginning()
{
    struct node *temp;

    if(first == NULL) return; // the list is empty
    else if (first == last) { // the list has only one item
        free(first);
        first = NULL;
        last = NULL;
        num_nodes--;
    } else { // the list has more than one item
        temp = first->next;
        temp->prev = NULL;
        free(first);
        first = temp;
        num_nodes--;
    }
}

/* To delete the last element of the list*/
void delete_at_end()
{
    struct node *temp;

    if(first == NULL) return; // the list is empty
    else if (first == last) { // the list has only one item
        free(first);
        first = NULL;
        last = NULL;
        num_nodes--;
    } else { // the list has more than one item
        temp = last->prev;
        free(last);
        temp->next = NULL;
        last = temp;
        num_nodes--;
    }
}

/* To delete all the list */
void delete_list()
{
    struct node *temp;

    if(first == NULL) return; // the list is empty
    else if (first == last) { //the list has only one item
        free(first);
        first = NULL;
        last = NULL;
        num_nodes = 0;
    }
}

```

```

    } else { //the list has two items or more
        temp = last->prev;
        while (temp != first) {
            free(temp->next);
            temp = temp->prev;
        }
        free(temp->next);
        free(first);
        first = NULL;
        last = NULL;
        num_nodes = 0;
    }
}

/////////////////////////////////////////////////////////////////
//          FREQUENCY METER FUNCTIONS
/////////////////////////////////////////////////////////////////

/* Instrument: Frequency Meter */
void frequency_meter( void )
{
    float frecuencia_int = 0;

    if (last_instrument != FREQUENCY_METER){
        // When the frequency meter is chosen, we do a reset of the ccp
        // module ...
        // ... and configure the CCP in function generator mode

        last_instrument = FREQUENCY_METER;
        reset_ccp();           // reset of the CCP module
        OSTimeDlyHMSM(0, 0, 1, 0); //Delay(h,m,s,ms)
        lcd_write_first_line("Frequency Meter");
        configuration_ccp_capture_0();      // CCP configured in
                                            // mode capture_0
    }

    if (captura_valida == 1) {
        frecuencia_int = calculate_new_freq();
        // Calculates the frequency with the values of
        // capturaant and captura (global variables)...
        // ... that are modified by the ISR of captura_0
        frecuencia =
            calculate_average_frequency(frecuencia_int);

        //show_frequency_lcd(frecuencia);           // Shows the
                                                // frequency in the
                                                // second line of the
                                                // LCD
    }

    captura_valida = 0;
}

configuration_ccp_capture_0(); // Configuration in capture mode
                             (captura_0)
}

/* Calculate new frequency with capturaant and captura */
float calculate_new_freq ( void )
{
    float frequency;
    float tPer = 0.00000001; // period of a 100MHz signal

    if (captura > capturaant) {
        frequency = (1/((captura-capturaant)*tPer));
        return frequency;
    } else {
        return frecuencia;
    }
}

```

```

}

/* Calculate average frequency */
float calculate_average_frequency(float new_freq)
{
    float av_freq;
    struct node *temp;

    temp = first;
    av_freq = 0.0;

    // We operate with the frequency list to create/delete nodes
    if (num_nodes < sc.num_freq_nodes) { // insert a node at the
        beginning

        insert_beginning(new_freq);

    } else { // delete the last node and insert a new one

        delete_at_end();
        insert_beginning(new_freq);
    }

    switch (sc.average_type) {
        case SIMPLE_MOVING_AVERAGE: {

            // Calculates the average frequency
            int k = 1;
            temp = first;
            for(k = 1; k <= num_nodes; k++) {

                av_freq = av_freq + (temp->frequency);
                temp = temp->next;
            }
            av_freq = av_freq/num_nodes;
            break;
        }
        case WEIGHTED_MOVING_AVERAGE: {

            // Calculates the weights
            int sum = 0;
            int i = 0;

            for(i = 1; i <= num_nodes; i++) sum += i;
            // sum of the indexes of the nodes (1+2+3+ ...
            // +num_nodes)

            // Calculates the average frequency by using
            // the weights (i/sum)
            i = num_nodes;
            temp = first;
            for(i = num_nodes; i >= 1; i--) {

                av_freq = av_freq + ((temp-
                    >frequency)*(i*1.0/sum));
                temp = temp->next;
            }
            break;
        }
        case EXPONENTIAL_MOVING_AVERAGE: {

            // Calculates the exponential weights
            float alpha = 0, sum = 0;
            int i = 0;

            if (num_nodes == sc.num_freq_nodes) {

                alpha = 4.0/(num_nodes+1);
                for(i = 1; i <= num_nodes; i++) sum +=
                    (alpha*pow((1-alpha), (1-i)));
            }

            // Calculates the average frequency by
            // using the exponential weights
            (alpha*pow((1-alpha), (1-i))/sum)
            i = num_nodes;
        }
    }
}

```

```

        temp = first;
        for(i = num_nodes; i >= 1; i--) {
            av_freq = av_freq + ((temp->frequency)*((alpha*pow((1.0-alpha), (1.0-i)))/sum));
            temp = temp->next;
        }
    }
    break;
}
default: {
    printf("ERROR: sc->average_type not
            recognized\n");
}
break;
} //end switch
return av_freq;
}

/* Show frequency in the second line of the LCD display */
void show_frequency_lcd ( float frequency )
{
    char* s_freq;

    if (frequency < 10){
        sprintf(s_freq, "%.4f", frequency ); // It transform the
                                            // frequency in a character array
        strcat(s_freq, " Hz"); // Append the units
        lcd_write_second_line(s_freq);
    }else if ((frequency >= 10)&&(frequency < 1000)){
        sprintf(s_freq, "%.3f", frequency );
        strcat(s_freq, " Hz");
        lcd_write_second_line(s_freq);
    }else if ((frequency >= 1000)&&(frequency < 1000000)){
        frequency = frequency/1000;
        sprintf(s_freq, "%.3f", frequency );
        strcat(s_freq, " kHz");
        lcd_write_second_line(s_freq);
    }else if (frequency >= 1000000){
        frequency = frequency/1000000;
        sprintf(s_freq, "%.3f", frequency );
        strcat(s_freq, " MHz");
        lcd_write_second_line(s_freq);
    }
}

////////////////////////////////////////////////////////////////
//      FUNCTIONS ASSOCIATED WITH THE FUNCTION GENERATOR      //
////////////////////////////////////////////////////////////////

/* Instrument: Function generator */
void function_generator( void )
{
    float freq;

    freq = 1/(sc.per_pwm*0.00000001); // Frequency of the output
                                     // signal

    if (last_instrument != FUNCTION_GENERATOR) {
        // When the function generator is chosen, we do a reset of the CCP module ...
        // ... and configure the CCP in function generator mode
        last_instrument = FUNCTION_GENERATOR;
}

```

```

        lcd_write_first_line("F. Generator");
        reset_ccp();
    }

    if (sc.square_activated == SQUARE_SIGNAL_ACTIVATED) { // Squared
        signal

        configuration_function_generator_square();
        ccp_activate_square();
        IOWR(PIO_OUT_RED_BASE, 0, 0x0001); // LEDR0 of the dev
                                         board is ON

    } else { // Non-Square signal

        configuration_function_generator_non_square(); //it has to
                                                       be configure in a loop
        ccp_activate_non_square(); //it has to
                                   be configure in a loop
        IOWR(PIO_OUT_RED_BASE, 0, 0x0002); // LEDR0 of the dev
                                         board is ON

    }

    show_frequency_lcd(freq); // Shows the frequency in the second
                             line of the LCD
}

/* Write signal values in the CCP's RAM */
void write_ram_function_generator(int *function, int num)
{
    int i=0;

    IOWR(PIO_EN_MUX_BASE,0,1); // mux enabled
    IOWR(PIO_DIREC_BASE,0,0xB); // write in RAM

    for (i = 0; i < num ; i++) {
        IOWR(PIO_DIREC_RAM_BASE,0,i); // i position of the RAM
        IOWR(PIO_DATOS_BASE,0,function[i]&0x0ff); // value to
                                                     introduce
        IOWR(PIO_WR_BASE,0,1); // Edge to ...
        IOWR(PIO_WR_BASE,0,0); // ... write the value
    }
}

/////////////////////////////////////////////////////////////////
// CCP CONFIGURATION //
/////////////////////////////////////////////////////////////////

/* Reset of the CCP module */
void reset_ccp( void )
{
    IOWR(PIO_CAPTURA_0_BASE, 2, 0x0); // Disable interruption in
                                       captura 0
    IOWR(PIO_CAPTURA_0_BASE, 3, 0x1); // Erase the interruption flag
                                       of edgecapture register of
                                       captura 0,
                                       // that had been activated
                                       due to the input edge

    IOWR(PIO_ASYNC_RESET_BASE,0,1);
    IOWR(PIO_ASYNC_RESET_BASE,0,0); // Reset of the CP module

    IOWR(PIO_EN_MUX_BASE,0,1); // mux enabled
}

/* CCP configuration in mode captura 0 */
void configuration_ccp_captura_0( void )
{

```

```

IOWR(PIO_CAPTURA_0_BASE, 2, 0x01); // Enable interruption in
                                    captura_0
IORD(PIO_CAPTURA_0_BASE, 2);
IOWR(PIO_CAPTURA_0_BASE, 3, 0x01); // Erase the interruption
                                    flag of edgecapture
                                    register of captura_0,
                                    // that had been activated
                                    due to the input edge

IOWR(PIO_EN_MUX_BASE,0,1);           // mux enabled

IOWR(PIO_DIREC_BASE,0,0x00);        // Select to write in the RConf
                                    (Configuration register of
                                    the CCP)
IOWR(PIO_DATOS_BASE,0,0x0001);     // CAPTURA_0 activated y and
                                    flag erased
IOWR(PIO_WR_BASE,0,1);             // Edge ...
IOWR(PIO_WR_BASE,0,0);             // ... to write in the CCP

}

/* CCP configuration. To change the frequency in the function generator. SQUARE FUNCTION */
/*
// The parameter Ton is changed automatically depending to a specific
// freqcency
// The output frequency is fout = 100MHz / per_pwm
void configuration_function_generator_square( void )
{
    IOWR(PIO_DIREC_BASE,0,0x0002); // Select to write in the PWM
                                    period
    IOWR(PIO_DATOS_BASE,0, sc.per_pwm);
    IOWR(PIO_WR_BASE,0,1);
    IOWR(PIO_WR_BASE,0,0);          // Edge in wr to write
    IOWR(PIO_DIREC_BASE,0,0x0003); // Select to write in the Ton of
                                    the PWM
    IOWR(PIO_DATOS_BASE,0, (unsigned int)sc.ton_pwm);
    IOWR(PIO_WR_BASE,0,1);
    IOWR(PIO_WR_BASE,0,0);          // Edge in wr

    IOWR(PIO_DIREC_BASE,0,0x0009); // Select to write in the
                                    PWM1 period
    IOWR(PIO_DATOS_BASE,0, (unsigned int)(sc.per_pwm)/RAM_PWM_SIZE);
    IOWR(PIO_WR_BASE,0,1);
    IOWR(PIO_WR_BASE,0,0);          // Edge in wr
    IOWR(PIO_DIREC_BASE,0,0x000A); // Select to write in the Ton of
                                    the PWM1
    IOWR(PIO_DATOS_BASE,0, (unsigned
                           int)((sc.per_pwm)/RAM_PWM_SIZE)/2);
    IOWR(PIO_WR_BASE,0,1);
    IOWR(PIO_WR_BASE,0,0);          // Edge in wr
}

/* CCP configuration. To change the frequency in the function generator. NON SQUARE
FUNCTION */
void configuration_function_generator_non_square( void )
{
    IOWR(PIO_DIREC_BASE,0,0x0002); // Select to write in the
                                    PWM period
    IOWR(PIO_DATOS_BASE,0, 0x000000FF); // 255
    IOWR(PIO_WR_BASE,0,1);
    IOWR(PIO_WR_BASE,0,0);          // Edge in wr
    IOWR(PIO_DIREC_BASE,0,0x0003); // Select to write in the
                                    Ton of the PWM
    IOWR(PIO_DATOS_BASE,0, 0x0000007F); // 127 = 255/2
    IOWR(PIO_WR_BASE,0,1);
    IOWR(PIO_WR_BASE,0,0);          // Edge in wr
}

```

```

        IOWR(PIO_DIREC_BASE,0,0x0009);           // Select to write in the
                                                // PWM1 period
        IOWR(PIO_DATOS_BASE,0, (sc.per_pwm)/RAM_PWM_SIZE);
        IOWR(PIO_WR_BASE,0,1);
        IOWR(PIO_WR_BASE,0,0);                  // Edge in wr
        IOWR(PIO_DIREC_BASE,0,0x000A);         // Select to write in the
                                                Ton of the PWM1
        IOWR(PIO_DATOS_BASE,0, ((sc.per_pwm)/RAM_PWM_SIZE)/2);
        IOWR(PIO_WR_BASE,0,1);
        IOWR(PIO_WR_BASE,0,0);                  // Edge in wr
    }

/* CCP configuration. To change only Ton in the function generator */
void configuration_function_generator_ton( void )
{
    IOWR(PIO_DIREC_BASE,0,0x0003); // Select to write in the Ton of
                                    // the PWM
    IOWR(PIO_DATOS_BASE,0,sc.ton_pwm);
    IOWR(PIO_WR_BASE,0,1);
    IOWR(PIO_WR_BASE,0,0);          // Edge in wr
}

/* CCP configuration. PWM functionality is activated. Square signal */
void ccp_activate_square( void )
{
    IOWR(PIO_DIREC_BASE,0,0x0); // Select to write in the BDO
    IOWR(PIO_DATOS_BASE,0,0x0003); // and PWM
    IOWR(PIO_WR_BASE,0,1);
    IOWR(PIO_WR_BASE,0,0);          // Edge in wr
}

/* CCP configuration. PWM functionality is activated. NON Square signal */
void ccp_activate_non_square( void )
{
    IOWR(PIO_DIREC_BASE,0,0x0); // Select to write in the BDO
    IOWR(PIO_DATOS_BASE,0,0x0007); // RAM and PWM signal
    IOWR(PIO_WR_BASE,0,1);
    IOWR(PIO_WR_BASE,0,0);          // Edge in wr
}

/////////////////////////////////////////////////////////////////
// OTHER FUNCTIONS RELATED WITH THE INSTRUMENTS           //
/////////////////////////////////////////////////////////////////

/* Initialization of the system configuration parameters */
void system_config_init ( void )
{
    sc.activated_device = FREQUENCY_METER;
    sc.average_type = SIMPLE_MOVING_AVERAGE;
    sc.num_freq_nodes = NUM_FREQ_NODES_DEFAULT;
    sc.per_pwm = PERIOD_PWM_DEFAULT;
    sc.ton_pwm = TON_PWM_DEFAULT;
    sc.square_activated = SQUARE_SIGNAL_ACTIVATED;
}

/* Assigns the interruption functions */
void assign_interruptions( void )
{
    printf("assign_interruptions\n");

    // Enable the EDGE buttons
    IOWR(PIO_IN_KEY_EDGE_BASE, 3, 0x07); // Clear the edge
                                         // interruption flag
    IOWR(PIO_IN_KEY_EDGE_BASE, 2, 0x07); // Enable edge interruption
}

```

```

    // Assign interruptions
    alt_ic_isr_register(PIO_CAPTURA_0_IRQ_INTERRUPT_CONTROLLER_ID, PIO_CAPTURA_0_IRQ,
(void *)interrupt_captura_0, NULL, NULL);
    alt_ic_isr_register(PIO_IN_KEY_EDGE_IRQ_INTERRUPT_CONTROLLER_ID,
PIO_IN_KEY_EDGE_IRQ, (void *)edge_button_interrupts , NULL, NULL);

}

////////////////////////////// INTERRUPTS //////////////////

/* Interruption of the external signal to measure the frequency. Used by the frequency
meter */
void interrupt_captura_0(void* context)
{

    IOWR(PIO_EN_MUX_BASE,0,1);           // mux enabled
    IOWR(PIO_SEL_SALIDA_BASE,0,0x9); // Select to show captura_0 at
                                    // the output

    if (captura_valida == 0) { // If the VITask program does not
                                // finish to use the previous values
                                // of captura and capturaant,
                                // we do not modified them in order
                                // to not have any inconsistency with
                                // the data

        IOWR(PIO_DIREC_BASE,0,0x00); // Select to write in RConf
        IOWR(PIO_DATOS_BASE,0,0x0009); // Stop the captura
                                        // and continue in capture
                                        // mode ("1001" in RConf)
        IOWR(PIO_WR_BASE,0,1);
        IOWR(PIO_WR_BASE,0,0);          // Edge in wr

        captura = (IORD(PIO_IN_32_BASE, 0)); // Get the last
                                            // captured value and
                                            // stores it in 'captura'

        IOWR(PIO_DIREC_BASE,0,0x00); // Select to write in RConf
        IOWR(PIO_DATOS_BASE,0,0x0019); // Active lec to show the
                                        // previous capture
        IOWR(PIO_WR_BASE,0,1);
        IOWR(PIO_WR_BASE,0,0);          // Edge in wr

        capturaant = (IORD(PIO_IN_32_BASE, 0)); // Get the previous
                                                // of the last capture
                                                // value, and stores it in
                                                // 'capturaant'

        captura_valida = 1;
    }

    IOWR(PIO_CAPTURA_0_BASE, 2, 0x0); // Interruption is disabled
    IOWR(PIO_CAPTURA_0_BASE, 3, 0x1); // Clear the flag of the
                                    // edgecapture register
    IORD(PIO_CAPTURA_0_BASE, 3);      // Delay

    IOWR(PIO_DIREC_BASE,0,0x0);     // Select write in RConf
    IOWR(PIO_DATOS_BASE,0,0x0001); // Desactivates 'lec' and
                                    // continues in capture mode (RConf)
    IOWR(PIO_WR_BASE,0,1);          // Edge ...
    IOWR(PIO_WR_BASE,0,0);          // ... to write
}

/* Interruption function of the edge activated buttons */
void edge_button_interrupts(void* context)
{

    volatile int button;
    char* text;
    button = IORD(PIO_IN_KEY_EDGE_BASE, 3); //read the state of the

```

```

        edge capture register of the
        buttons

    if (button == 1) {

        reset_ccp();

    } else if (button == 2) {

        sprintf(text, "MAC-%.2x%.2x%.2x%.2x%.2x%.2x",
               (int)DEFAULT_SOURCE_MAC_5, (int)DEFAULT_SOURCE_MAC_4,
               (int)DEFAULT_SOURCE_MAC_3,
               (int)DEFAULT_SOURCE_MAC_2, (int)DEFAULT_SOURCE_MAC_1,
               (int)DEFAULT_SOURCE_MAC_0);
        lcd_write_second_line(text);
        //usleep(500000); // 0.5 seconds

    } else if (button == 4) {

        sprintf(text, "%d.%d.%d.%d", (int)DEFAULT_IP_SOURCE_3,
               (int)DEFAULT_IP_SOURCE_2, (int)DEFAULT_IP_SOURCE_1,
               (int)DEFAULT_IP_SOURCE_0);
        lcd_write_second_line(text);
        //usleep(500000); // 0.5 seconds

    }

    IOWR(PIO_IN_KEY_EDGE_BASE, 3, 0xF); // Clear the edge interruption flags
}

/* Ethernet interrupt. Is called when we received a packet from the network */
void ethernet_interrupts()
{
    packet_num++;
    aaa=ReceivePacket(RXT,&rx_len);
    if(!aaa)
    {
        if (pkt_is_viprotocol(RXT, rx_len)) {

            struct pkt_buffer *pkt;
            pkt = (struct pkt_buffer *)RXT;

            int command=0;
            // Extract the command from the packet
            command += pkt->ip.udp.data[3]&0x0ff;
            command += (pkt->ip.udp.data[2] << 8)&0x0ff00;
            command += (pkt->ip.udp.data[1] << 16)&0x0ff0000;
            command += (pkt->ip.udp.data[0] << 24)&0x0ff000000;

            switch (command) {
                // The server changes the instrument's parameters depending
                // on the received command //

                case CMD_ACTIVATE_FREQ_METER:
                {
                    // Activation of the frequency meter

                    sc.activated_device = FREQUENCY_METER;

                    //printf("\n CMD_ACTIVATE_FREQ_METER\n");
                    break;
                }
                case CMD_FREQ_REQUEST:
                {
                    // The client requests the current
                    // average frequency

                    pkt_changes_to_send(pkt);

                    // Send command CMD_FREQ_ANSWER to the
                    // destination
                    pkt->ip.udp.data[0] = (unsigned
                                           char)CMD_FREQ_ANSWER >> 24;
                    pkt->ip.udp.data[1] = (unsigned
                                           char)CMD_FREQ_ANSWER >> 16;
                    pkt->ip.udp.data[2] = (unsigned
                                           char)CMD_FREQ_ANSWER >> 8;
                }
            }
        }
    }
}

```

```

        char)CMD_FREQ_ANSWER >> 8;
pkt->ip.udp.data[3] = (unsigned
        char)CMD_FREQ_ANSWER;

// Pointer to char to extract the bytes of the
// float number and put them on
// the packet to send (the network is big
// endian)
char *s_freq;
s_freq = (char *)&frecuencia;
pkt->ip.udp.data[7] = s_freq[0]&0x0ff;
pkt->ip.udp.data[6] = s_freq[1]&0x0ff;
pkt->ip.udp.data[5] = s_freq[2]&0x0ff;
pkt->ip.udp.data[4] = s_freq[3]&0x0ff;

TransmitPacket(RXT,PKT_LENGTH);

//printf("\n CMD_FREQ_REQUEST\n");
        break;
}
case CMD_NFREQ_NODES:
{
// To change the number of nodes of the
// frequency average

int num=0;
delete_list();
num += pkt->ip.udp.data[7]&0x0ff;
    num += (pkt->ip.udp.data[6] <<
            8)&0xff00;
    num += (pkt->ip.udp.data[5] <<
            16)&0xffff0000;
    num += (pkt->ip.udp.data[4] <<
            24)&0xffffffff;

sc.num_freq_nodes = (int)num;

//printf("\n CMD_NFREQ_NODES, num = %d\n",
//        num);
        break;
}
case CMD_AV_SIMPLE:
{
// Simple average

sc.average_type = SIMPLE_MOVING_AVERAGE;

//printf("\n CMD_AV_SIMPLE\n");
        break;
}
case CMD_AV_WEIGHTED:
{
// Weighted average

sc.average_type =
        WEIGHTED_MOVING_AVERAGE;

//printf("\n CMD_AV_WEIGHTED\n");
        break;
}
case CMD_AV_EXP:
{
// Exponential average

sc.average_type =
        EXPONENTIAL_MOVING_AVERAGE;

//printf("\n CMD_AV_EXP\n");
        break;
}
case CMD_ACTIVATE_FUNC_GEN:
{
// Activation of the function generator

sc.activated_device = FUNCTION_GENERATOR;

//printf("\n CMD_ACTIVATE_FUNC_GEN\n");

```

```

        break;
    }
    case CMD_RAM_VALUES:
    {
        // Change the RAM values of the CCP
        int i=0;
        for(i = 0; i < RAM_PWM_SIZE; i++) {

            custom_signal[i] = (unsigned
                int)pkt->ip.udp.data[4+i];

        }

        write_ram_function_generator(custom_signal,
                                     RAM_PWM_SIZE);

        //printf("\n CMD_RAM_VALUES\n");
        break;
    }
    case CMD_FG_SQUARE:
    {
        // Square signal in the function
        // generator

        sc.square_activated =
            SQUARE_SIGNAL_ACTIVATED;

        //printf("\n CMD_FG_SQUARE\n");
        break;
    }
    case CMD_FG_NON_SQUARE:
    {
        // Non square signal in the function
        // generator

        sc.square_activated =
            NON_SQUARE_SIGNAL_ACTIVATED;

        //printf("\n CMD_FG_NON_SQUARE\n");
        break;
    }
    case CMD_FG_CHANGE_FREQ:
    {
        // Change the frequency (and also Ton) of
        // the function generator's signal
        // Fout = 100MHz/per_pwm

        int new_freq=0;
        // Extract the new frequency from the packet
        new_freq += pkt->ip.udp.data[7]&0x0ff;
        new_freq += (pkt->ip.udp.data[6] << 8)&0x0ff00;
        new_freq += (pkt->ip.udp.data[5] <<
                     16)&0x0ff0000;
        new_freq += (pkt->ip.udp.data[4] <<
                     24)&0x0ff000000;

        sc.per_pwm = (unsigned int)(100e6/new_freq);
        sc.ton_pwm = sc.per_pwm/2;

        //printf("\n CMD_FG_CHANGE_FREQ \n");
        break;
    }
    case CMD_FG_CHANGE_TON:
    {
        // Change the Ton of the function
        // generator's signal. We receive the
        // Duty cycle --> Ton =
        // per_pwm/(100/DC)

        int new_dc=0;
        // Extract the new duty cycle from the
        // packet to change Ton
        new_dc += pkt->ip.udp.data[7]&0x0ff;
        new_dc += (pkt->ip.udp.data[6] <<
                     8)&0x0ff00;
        new_dc += (pkt->ip.udp.data[5] <<
                     16)&0x0ff0000;

```

```

        new_dc += (pkt->ip.udp.data[4] <<
                    24) & 0x0ff000000;

        if ((new_dc > 0) && (new_dc <= 100)) {
            sc.ton_pwm = (unsigned
                           int)(sc.per_pwm/(100.0/new_dc));
        }

        //printf("\n CMD_FG_CHANGE_TON \n");
        break;
    }
    case CMD_RESET:
    {
        // Reset of the system

        reset_ccp();
        system_config_init();

        //printf("\n CMD_RESET\n");
        break;
    }
    default:
        //printf("WARNING: Unknown Command =
        %.2x%.2x%.2x%.2x ! -- We do not do
        anything!\n", pkt->ip.udp.data[0],
        pkt->ip.udp.data[1], pkt->ip.udp.data[2],
        pkt->ip.udp.data[3]);
        break;
    }
}

//end if(packet_is_for_me())
else if (pkt_is_arprequest(RXT, rx_len)) { // If we receive an
                                             ARP Request

    //printf("---- PACKET ARP FOR ME ----\n");
    struct pkt_buffer_2 *pkt;

    pkt = (struct pkt_buffer_2 *)RXT;
    int pkt_length;
    pkt_length = rx_len - 4; //do not count the Ethernet CRC

    // Now we change the content of the packet in order to send
    // an ARP Reply
    pkt_arp_changes_to_send(pkt);
    if (pkt_length == 42) pkt_length += 18; // 18 bytes of padding
                                              (any value)
    TransmitPacket(RXT, pkt_length); // the length of ARP Reply
                                     is always 60 bytes

}
}

////////////////////////////////////////////////////////////////
//          NETWORK CONNECTION                         //
////////////////////////////////////////////////////////////////

/* Assign default values to the fields of the pkt */
void assign_pkt_defaults( struct pkt_buffer *pkt )
{
    // Ethernet header
    pkt->mac_dest0 = DEFAULT_DEST_MAC_0;
    pkt->mac_dest1 = DEFAULT_DEST_MAC_1;
    pkt->mac_dest2 = DEFAULT_DEST_MAC_2;
    pkt->mac_dest3 = DEFAULT_DEST_MAC_3;
    pkt->mac_dest4 = DEFAULT_DEST_MAC_4;
    pkt->mac_dest5 = DEFAULT_DEST_MAC_5;
    pkt->mac_source0 = DEFAULT_SOURCE_MAC_0;
    pkt->mac_source1 = DEFAULT_SOURCE_MAC_1;
    pkt->mac_source2 = DEFAULT_SOURCE_MAC_2;
    pkt->mac_source3 = DEFAULT_SOURCE_MAC_3;
    pkt->mac_source4 = DEFAULT_SOURCE_MAC_4;
    pkt->mac_source5 = DEFAULT_SOURCE_MAC_5;
    pkt->ethertype_h = DEFAULT_ETHERTYPE_H;
}

```

```

    pkt->ethertype_l = DEFAULT_ETHERTYPE_L;

    // IP header
    pkt->ip.version_ihl = DEFAULT_VERSION_IHL;
    pkt->ip.dscp_ecn = DEFAULT_DSCP_ECN;
    pkt->ip.iplength_h = DEFAULT_IP_LENGTH_H;
    pkt->ip.iplength_l = DEFAULT_IP_LENGTH_L;
    pkt->ip.id_h = DEFAULT_ID_H;
    pkt->ip.id_l = DEFAULT_ID_L;
    pkt->ip.flags_offset_h = DEFAULT_FLAGS_OFFSET_H;
    pkt->ip.flags_offset_l = DEFAULT_FLAGS_OFFSET_L;
    pkt->ip.ttl = DEFAULT_TTL;
    pkt->ip.protocol = DEFAULT_PROTOCOL;
    pkt->ip.ipcrc_h = DEFAULT_IP_CRC_H;
    pkt->ip.ipcrc_l = DEFAULT_IP_CRC_L;
    pkt->ip.ipsource0 = DEFAULT_IP_SOURCE_0;
    pkt->ip.ipsource1 = DEFAULT_IP_SOURCE_1;
    pkt->ip.ipsource2 = DEFAULT_IP_SOURCE_2;
    pkt->ip.ipsource3 = DEFAULT_IP_SOURCE_3;
    pkt->ip.ipdest0 = DEFAULT_IP_DEST_0;
    pkt->ip.ipdest1 = DEFAULT_IP_DEST_1;
    pkt->ip.ipdest2 = DEFAULT_IP_DEST_2;
    pkt->ip.ipdest3 = DEFAULT_IP_DEST_3;

    // UDP header
    pkt->ip.udp.source_port_h = DEFAULT_SOURCE_PORT_H;
    pkt->ip.udp.source_port_l = DEFAULT_SOURCE_PORT_L;
    pkt->ip.udp.dest_port_h = DEFAULT_DEST_PORT_H;
    pkt->ip.udp.dest_port_l = DEFAULT_DEST_PORT_L;
    pkt->ip.udp.udplenlength_h = DEFAULT_UDP_LENGTH_H;
    pkt->ip.udp.udplenlength_l = DEFAULT_UDP_LENGTH_L;
    pkt->ip.udp.crc_h = DEFAULT_UDP_CRC_H;
    pkt->ip.udp.crc_l = DEFAULT_UDP_CRC_L;
}

/* Calculate the IP checksum */
unsigned int calculate_ip_checksum( struct pkt_buffer *pkt)
{
    char *ptr;
    int i;
    unsigned int sum = 0, sum_aux;

    ptr = (char *)pkt;
    ptr += ETHERNET_HEADER_LENGTH; //ptr pointing to the first field
                                    //of the IP header
    sum = 0xffff;

    for (i = 0; i < IP_HEADER_LENGTH/2; i++) {
        sum += ((*(ptr)&0xff)<<8) | (*(ptr+1)&0xff);
        ptr += 2;
        if (sum > 0xffff) sum -= 0xffff;
    }

    sum_aux = sum&0xffff;

    // return ones's complement
    return ~sum_aux;
}

/* Calculate the UDP checksum (it's optional) */
unsigned int calculate_udp_checksum( struct pkt_buffer *pkt)
{
    /*      It's optional. We do not use it. When it's not used
     *              the checksum value is 0x0000
     */

    return 0;
}

/* To determine if a received packet is for me or not */

```

```

int pkt_is_viprotocol(unsigned char *buffer, int rx_len)
{
    // Packet IS a vi protocol packet      --> return 1
    // Packet IS NOT a vi_protocol packet --> return 0

    int pkt_length = 0;
    struct pkt_buffer *pkt;
    pkt = (struct pkt_buffer *)buffer;

    pkt_length = rx_len - 4; // we don't count the four bytes of FCS
                           // (Ethernet checksum)

    // Check whether the packet length is the expected one
    if (pkt_length != PKT_LENGTH) return 0;

    // Check Ethernet header
    // MAC is broadcast
    if ( (pkt->mac_dest5 == (char)0xFF) && (pkt->mac_dest4 ==
        (char)0xFF) && (pkt->mac_dest3 == (char)0xFF)
        && (pkt->mac_dest2 == (char)0xFF) && (pkt->mac_dest1
        == (char)0xFF) && (pkt->mac_dest0 == (char)0xFF)) {
        pkt_length = rx_len - 4; //is just an assignment to
                               //complete the if statement
    } else { // IF it's not my MAC
        if (pkt->mac_dest5 != (char)DEFAULT_SOURCE_MAC_5) return 0;
        else if (pkt->mac_dest5 != (char)DEFAULT_SOURCE_MAC_5)
            return 0;
        else if (pkt->mac_dest4 != (char)DEFAULT_SOURCE_MAC_4)
            return 0;
        else if (pkt->mac_dest3 != (char)DEFAULT_SOURCE_MAC_3)
            return 0;
        else if (pkt->mac_dest2 != (char)DEFAULT_SOURCE_MAC_2)
            return 0;
        else if (pkt->mac_dest1 != (char)DEFAULT_SOURCE_MAC_1)
            return 0;
        else if (pkt->mac_dest0 != (char)DEFAULT_SOURCE_MAC_0)
            return 0;
    }

    // Check the IP header
    // If it's not my IP address
    if (pkt->ip.ipdest3 != (char)DEFAULT_IP_SOURCE_3) return 0;
    else if (pkt->ip.ipdest2 != (char)DEFAULT_IP_SOURCE_2) return 0;
    else if (pkt->ip.ipdest1 != (char)DEFAULT_IP_SOURCE_1) return 0;
    else if (pkt->ip.ipdest0 != (char)DEFAULT_IP_SOURCE_0) return 0;

    // Check the IP Length
    if (pkt->ip.iplength_h != (char)DEFAULT_IP_LENGTH_H) return 0;
    else if (pkt->ip.iplength_l != (char)DEFAULT_IP_LENGTH_L)
        return 0;

    // Check the UDP header
    // Destination port
    if (pkt->ip.udp.dest_port_h != (char)DEFAULT_SOURCE_PORT_H)
        return 0;
    else if (pkt->ip.udp.dest_port_l != (char)DEFAULT_SOURCE_PORT_L)
        return 0;

    // UDP Length
    if (pkt->ip.udp.udplength_h != (char)DEFAULT_UDP_LENGTH_H)
        return 0;
    else if (pkt->ip.udp.udplength_l != (char)DEFAULT_UDP_LENGTH_L)
        return 0;

    return 1; //the packet is a vi_protocol packet!!
}

/* To determine if a received packet is an ARP Request for me */
int pkt_is_arprequest(unsigned char *buffer, int rx_len)
{
    // Packet IS an ARP Request packet asking me      --> return 1
    // Packet IS NOT an ARP Request packet asking me --> return 0

```

```

int pkt_length = 0;
struct pkt_buffer_2 *pkt; // ARP buffer
pkt = (struct pkt_buffer_2 *)buffer;

pkt_length = rx_len - 4; // we don't count the four bytes of FCS
(Ethernet checksum)

// Ethertype ARP (0x0806)
if ((pkt->ethertype_h == 0x08) && (pkt->ethertype_l == 0x06)) {
    pkt_length = rx_len - 4; // is just an assignment to complete
                           the if statement
} else {
    return 0;
}

// Check whether the packet length is the expected one
if ((pkt_length == 42) || (pkt_length == 60)) {
    pkt_length = rx_len - 4; // it's just an assignment to
                           complete the if statement
} else {
    return 0;
}

// Check Ethernet header
// MAC is broadcast
if ( (pkt->mac_dest5 == (char)0xFF) && (pkt->mac_dest4 ==
      (char)0xFF) && (pkt->mac_dest3 == (char)0xFF)
      && (pkt->mac_dest2 == (char)0xFF) && (pkt->mac_dest1
      == (char)0xFF) && (pkt->mac_dest0 == (char)0xFF)) {
    pkt_length = rx_len - 4; // it's just an assignment to
                           complete the if statement
}

// Opcode ARP request
if (pkt->arp.opcode_l != 0x01) return 0;

// We verify if the ARP Request is requesting my MAC (we see
whether the target IP Address is my IP Address or not)
if ( (pkt->arp.iptarget3 == (char)DEFAULT_IP_SOURCE_3) && (pkt-
      >arp.iptarget2 == (char)DEFAULT_IP_SOURCE_2)
      && (pkt->arp.iptarget1 == (char)DEFAULT_IP_SOURCE_1)
      && (pkt->arp.iptarget0 == (char)DEFAULT_IP_SOURCE_0)) {
    pkt_length = rx_len - 4; // it's just an assignment to
                           complete the if statement
} else {
    return 0;
}

return 1;
}

/* Change header parameters in order to send the packet to the client */
void pkt_changes_to_send(struct pkt_buffer *pkt)
{
    // We need to change some parameters in the packet:
    //           Ethernet: Dest/Source MAC
    //           IP:      TTL, Source/Dest IP, Checksum
    //           UDP: Source/Dest Port, Checksum
    //

    unsigned int ip_checksum, udp_checksum;

    // Ethernet changes
    pkt->mac_dest0 = pkt->mac_source0;
    pkt->mac_dest1 = pkt->mac_source1;
    pkt->mac_dest2 = pkt->mac_source2;
    pkt->mac_dest3 = pkt->mac_source3;
    pkt->mac_dest4 = pkt->mac_source4;
    pkt->mac_dest5 = pkt->mac_source5;
    pkt->mac_source0 = DEFAULT_SOURCE_MAC_0;
    pkt->mac_source1 = DEFAULT_SOURCE_MAC_1;
    pkt->mac_source2 = DEFAULT_SOURCE_MAC_2;
    pkt->mac_source3 = DEFAULT_SOURCE_MAC_3;
    pkt->mac_source4 = DEFAULT_SOURCE_MAC_4;
    pkt->mac_source5 = DEFAULT_SOURCE_MAC_5;
}

```

```

// IP changes
pkt->ip.ttl = DEFAULT_TTL;
pkt->ip.ipdest0 = pkt->ip.ipsource0;
pkt->ip.ipdest1 = pkt->ip.ipsource1;
pkt->ip.ipdest2 = pkt->ip.ipsource2;
pkt->ip.ipdest3 = pkt->ip.ipsource3;
pkt->ip.ipsource0 = DEFAULT_IP_SOURCE_0;
pkt->ip.ipsource1 = DEFAULT_IP_SOURCE_1;
pkt->ip.ipsource2 = DEFAULT_IP_SOURCE_2;
pkt->ip.ipsource3 = DEFAULT_IP_SOURCE_3;

pkt->ip.ipcrc_l = 0x00;           // we clear it because the field is used
                                to ...
pkt->ip.ipcrc_h = 0x00;           // ... calculate the IP checksum
ip_checksum = calculate_ip_checksum(pkt);
pkt->ip.ipcrc_l = (char)ip_checksum&0xff;
pkt->ip.ipcrc_h = (char)(ip_checksum >> 8)&0xff;

// UDP changes
pkt->ip.udp.dest_port_h = pkt->ip.udp.source_port_h;
pkt->ip.udp.dest_port_l = pkt->ip.udp.source_port_l;
pkt->ip.udp.source_port_h = DEFAULT_SOURCE_PORT_H;
pkt->ip.udp.source_port_l = DEFAULT_SOURCE_PORT_L;

pkt->ip.udp.crc_h = 0x00; // we clear it because the field is
                           used to ...
pkt->ip.udp.crc_l = 0x00; // ... calculate the UDP checksum
udp_checksum = calculate_udp_checksum(pkt); // UDP checksum
pkt->ip.udp.crc_l = (char)udp_checksum&0xff;
pkt->ip.udp.crc_h = (char)(udp_checksum >> 8)&0xff;

}

/* Change ARP Request fields in order to send the ARP Reply */
void pkt_arp_changes_to_send(struct pkt_buffer_2 *pkt)
{

    // Ethernet Header
    // Changes: MAC dest/source
    pkt->mac_dest5 = pkt->arp.mac_sender5;
    pkt->mac_dest4 = pkt->arp.mac_sender4;
    pkt->mac_dest3 = pkt->arp.mac_sender3;
    pkt->mac_dest2 = pkt->arp.mac_sender2;
    pkt->mac_dest1 = pkt->arp.mac_sender1;
    pkt->mac_dest0 = pkt->arp.mac_sender0;

    pkt->mac_source5 = DEFAULT_SOURCE_MAC_5;
    pkt->mac_source4 = DEFAULT_SOURCE_MAC_4;
    pkt->mac_source3 = DEFAULT_SOURCE_MAC_3;
    pkt->mac_source2 = DEFAULT_SOURCE_MAC_2;
    pkt->mac_source1 = DEFAULT_SOURCE_MAC_1;
    pkt->mac_source0 = DEFAULT_SOURCE_MAC_0;

    // ARP Header
    // Changes: OPCode, Sender/Target MAC, Sender/Target IP
    pkt->arp.opcode_l = 0x02; // ARP Reply

    pkt->arp.mac_target5 = pkt->arp.mac_sender5;
    pkt->arp.mac_target4 = pkt->arp.mac_sender4;
    pkt->arp.mac_target3 = pkt->arp.mac_sender3;
    pkt->arp.mac_target2 = pkt->arp.mac_sender2;
    pkt->arp.mac_target1 = pkt->arp.mac_sender1;
    pkt->arp.mac_target0 = pkt->arp.mac_sender0;

    pkt->arp.iptarget3 = pkt->arp.ipsender3;
    pkt->arp.iptarget2 = pkt->arp.ipsender2;
    pkt->arp.iptarget1 = pkt->arp.ipsender1;
    pkt->arp.iptarget0 = pkt->arp.ipsender0;

    pkt->arp.mac_sender5 = DEFAULT_SOURCE_MAC_5;
    pkt->arp.mac_sender4 = DEFAULT_SOURCE_MAC_4;
    pkt->arp.mac_sender3 = DEFAULT_SOURCE_MAC_3;
    pkt->arp.mac_sender2 = DEFAULT_SOURCE_MAC_2;
    pkt->arp.mac_sender1 = DEFAULT_SOURCE_MAC_1;
    pkt->arp.mac_sender0 = DEFAULT_SOURCE_MAC_0;
}

```

```
    pkt->arp.ipsender3 = DEFAULT_IP_SOURCE_3;
    pkt->arp.ipsender2 = DEFAULT_IP_SOURCE_2;
    pkt->arp.ipsender1 = DEFAULT_IP_SOURCE_1;
    pkt->arp.ipsender0 = DEFAULT_IP_SOURCE_0;
}

////////////////////////////////////////////////////////////////////////
```

Apéndice 2: código fuente de la aplicación de usuario

Apéndice 2A: Interfaz gráfica (*window.py*)

```
from PyQt4 import QtCore, QtGui

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig, _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName(_fromUtf8("MainWindow"))
        MainWindow.resize(1205, 693)
        self.centralwidget = QtGui.QWidget(MainWindow)
        self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
        self.gridLayout = QtGui.QGridLayout(self.centralwidget)
        self.gridLayout.setObjectName(_fromUtf8("gridLayout"))
        self.stackedWidget = QtGui.QStackedWidget(self.centralwidget)
        self.stackedWidget.setObjectName(_fromUtf8("stackedWidget"))
        self.page = QtGui.QWidget()
        self.page.setObjectName(_fromUtf8("page"))
        self.verticalLayout_3 = QtGui.QVBoxLayout(self.page)
        self.verticalLayout_3.setObjectName(_fromUtf8("verticalLayout_3"))
        self.tablefrequency = QtGui.QTableWidget(self.page)
        self.tablefrequency.setAutoFillBackground(False)
        self.tablefrequency.setFrameShape(QtGui.QFrame.StyledPanel)
        self.tablefrequency.setFrameShadow(QtGui.QFrame.Sunken)
        self.tablefrequency.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)
        self.tablefrequency.setDragDropOverwriteMode(True)
        self.tablefrequency.setAlternatingRowColors(False)
        self.tablefrequency.setSelectionMode(QtGui.QAbstractItemView.NoSelection)
        self.tablefrequency.setCornerButtonEnabled(True)
        self.tablefrequency.setRowCount(0)
        self.tablefrequency.setObjectName(_fromUtf8("tablefrequency"))
        self.tablefrequency.setColumnCount(4)
        item = QtGui QTableWidgetItem()

        item.setTextAlignment(QtCore.Qt.AlignHCenter|QtCore.Qt.AlignVCenter|QtCore.Qt.AlignCenter)
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        item.setFont(font)
        self.tablefrequency.setHorizontalHeaderItem(0, item)
        item = QtGui QTableWidgetItem()

        item.setTextAlignment(QtCore.Qt.AlignHCenter|QtCore.Qt.AlignVCenter|QtCore.Qt.AlignCenter)
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        item.setFont(font)
        self.tablefrequency.setHorizontalHeaderItem(1, item)
        item = QtGui QTableWidgetItem()

        item.setTextAlignment(QtCore.Qt.AlignHCenter|QtCore.Qt.AlignVCenter|QtCore.Qt.AlignCenter)
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        item.setFont(font)
```

```

        self.tablefrequency.setHorizontalHeaderItem(2, item)
        item = QtGui.QTableWidgetItem()

item.setTextAlignment(QtCore.Qt.AlignHCenter|QtCore.Qt.AlignVCenter|QtCore.Qt.AlignCenter)
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        item.setFont(font)
        self.tablefrequency.setHorizontalHeaderItem(3, item)
        self.tablefrequency.horizontalHeader().setVisible(True)
        self.tablefrequency.horizontalHeader().setCascadingSectionResizes(False)
        self.tablefrequency.horizontalHeader().setDefaultSectionSize(222)
        self.tablefrequency.horizontalHeader().setHighlightSections(True)
        self.tablefrequency.horizontalHeader().setMinimumSectionSize(27)
        self.tablefrequency.horizontalHeader().setSortIndicatorShown(False)
        self.tablefrequency.horizontalHeader().setStretchLastSection(True)
        self.tablefrequency.verticalHeader().setVisible(False)
        self.tablefrequency.verticalHeader().setCascadingSectionResizes(False)
        self.verticalLayout_3.addWidget(self.tablefrequency)
        self.stackedWidget.addWidget(self.page)
        self.page_2 = QtGui.QWidget()
        self.page_2.setObjectName(_fromUtf8("page_2"))
        self.verticalLayout_4 = QtGui.QVBoxLayout(self.page_2)
        self.verticalLayout_4.setObjectName(_fromUtf8("verticalLayout_4"))
        self.graphsignal = Qwt5.QwtPlot(self.page_2)
        self.graphsignal.setObjectName(_fromUtf8("graphsignal"))
        self.verticalLayout_4.addWidget(self.graphsignal)
        self.stackedWidget.addWidget(self.page_2)
        self.gridLayout.addWidget(self.stackedWidget, 0, 0, 1, 1)
MainWindow.setCentralWidget(self.centralwidget)
self.menuubar = QtGui.QMenuBar(MainWindow)
self.menuubar.setGeometry(QtCore.QRect(0, 0, 1205, 21))
self.menuubar.setObjectName(_fromUtf8("menuubar"))
self.menu_File = QtGui.QMenu(self.menuubar)
self.menu_File.setObjectName(_fromUtf8("menu_File"))
MainWindow.setMenuBar(self.menuubar)
self.statusbar = QtGui.QStatusBar(MainWindow)
self.statusbar.setObjectName(_fromUtf8("statusbar"))
MainWindow.setStatusBar(self.statusbar)
        self.dockWidget = QtGui.QDockWidget(MainWindow)

self.dockWidget.setFeatures(QtGui.QDockWidget.DockWidgetFloatable|QtGui.QDockWidget.DockWidgetMovable)
        self.dockWidget.setObjectName(_fromUtf8("dockWidget"))
        self.dockWidgetContents = QtGui.QWidget()
        self.dockWidgetContents.setObjectName(_fromUtf8("dockWidgetContents"))
        self.formLayout_5 = QtGui.QFormLayout(self.dockWidgetContents)
        self.formLayout_5.setFieldGrowthPolicy(QtGui.QFormLayout.AllNonFixedFieldsGrow)
        self.formLayout_5.setObjectName(_fromUtf8("formLayout_5"))
        self.horizontalLayout_4 = QtGui.QHBoxLayout()
        self.horizontalLayout_4.setObjectName(_fromUtf8("horizontalLayout_4"))
        self.horizontalLayout_4.addWidget(self.label_4)
        self.label_4 = QtGui.QLabel(self.dockWidgetContents)
        self.label_4.setObjectName(_fromUtf8("label_4"))
        self.horizontalLayout_4.addWidget(self.label_4)
        self.textserverip = QtGui.QLineEdit(self.dockWidgetContents)
        self.textserverip.setObjectName(_fromUtf8("textserverip"))
        self.horizontalLayout_4.addWidget(self.textserverip)
        self.formLayout_5.setLayout(0, QtGui.QFormLayout.SpanningRole,
self.horizontalLayout_4)
        self.formLayout_2 = QtGui.QFormLayout()
        self.formLayout_2.setObjectName(_fromUtf8("formLayout_2"))
        self.label_5 = QtGui.QLabel(self.dockWidgetContents)
        self.label_5.setObjectName(_fromUtf8("label_5"))
        self.formLayout_2.setWidget(0, QtGui.QFormLayout.LabelRole, self.label_5)
        self.numserverport = QtGui.QSpinBox(self.dockWidgetContents)
        self.numserverport.setMinimum(65)
        self.numserverport.setMaximum(65535)
        self.numserverport.setProperty("value", 80)
        self.numserverport.setObjectName(_fromUtf8("numserverport"))
        self.formLayout_2.setWidget(0, QtGui.QFormLayout.FieldRole, self.numserverport)
        self.formLayout_5.setLayout(1, QtGui.QFormLayout.SpanningRole,
self.formLayout_2)
        self.horizontalLayout_6 = QtGui.QHBoxLayout()
        self.horizontalLayout_6.setObjectName(_fromUtf8("horizontalLayout_6"))
        self.btnconnect = QtGui.QPushButton(self.dockWidgetContents)
        self.btnconnect.setObjectName(_fromUtf8("btnconnect"))

```

```

        self.horizontalLayout_6.addWidget(self.btnconnect)
        self.btndisconnect = QtGui.QPushButton(self.dockWidgetContents)
        self.btndisconnect.setObjectName(_fromUtf8("btndisconnect"))
        self.horizontalLayout_6.addWidget(self.btndisconnect)
        self.formLayout_5.setLayout(2, QtGui.QFormLayout.SpanningRole,
self.horizontalLayout_6)
        self.dockWidget.setWidget(self.dockWidgetContents)
        MainWindow.addDockWidget(QtCore.Qt.DockWidgetArea(1), self.dockWidget)
        self.dockWidget_2 = QtGui.QDockWidget(MainWindow)

self.dockWidget_2.setFeatures(QtGui.QDockWidget.DockWidgetFloatable|QtGui.QDockWidget.DockWidgetMovable)
        self.dockWidget_2.setObjectName(_fromUtf8("dockWidget_2"))
        self.dockWidgetContents_2 = QtGui.QWidget()
        self.dockWidgetContents_2.setObjectName(_fromUtf8("dockWidgetContents_2"))
        self.verticalLayout = QtGui.QVBoxLayout(self.dockWidgetContents_2)
        self.verticalLayout.setObjectName(_fromUtf8("verticalLayout"))
        self.formLayout_4 = QtGui.QFormLayout()
        self.formLayout_4.setObjectName(_fromUtf8("formLayout_4"))
        self.label = QtGui.QLabel(self.dockWidgetContents_2)
        self.label.setObjectName(_fromUtf8("label"))
        self.formLayout_4.setWidget(0, QtGui.QFormLayout.LabelRole, self.label)
        self.cboxviselect = QtGui.QComboBox(self.dockWidgetContents_2)
        self.cboxviselect.setObjectName(_fromUtf8("cboxviselect"))
        self.cboxviselect.addItem(_fromUtf8(""))
        self.cboxviselect.addItem(_fromUtf8(""))
        self.formLayout_4.setWidget(0, QtGui.QFormLayout.FieldRole, self.cboxviselect)
        self.verticalLayout.setLayout(self.formLayout_4)
        self.stackedWidget_2 = QtGui.QStackedWidget(self.dockWidgetContents_2)
        self.stackedWidget_2.setObjectName(_fromUtf8("stackedWidget_2"))
        self.page_3 = QtGui.QWidget()
        self.page_3.setObjectName(_fromUtf8("page_3"))
        self.formLayout = QtGui.QFormLayout(self.page_3)
        self.formLayout.setObjectName(_fromUtf8("formLayout"))
        self.checkboxfreqreq = QtGui.QCheckBox(self.page_3)
        self.checkboxfreqreq.setObjectName(_fromUtf8("checkboxfreqreq"))
        self.formLayout.setWidget(3, QtGui.QFormLayout.LabelRole, self.checkboxfreqreq)
        self.horizontalLayout_5 = QtGui.QHBoxLayout()
        self.horizontalLayout_5.setObjectName(_fromUtf8("horizontalLayout_5"))
        self.label_interval = QtGui.QLabel(self.page_3)
        self.label_interval.setObjectName(_fromUtf8("label_interval"))
        self.horizontalLayout_5.addWidget(self.label_interval)
        self.dintervalsamples = QtGui.QDoubleSpinBox(self.page_3)
        self.dintervalsamples.setDecimals(3)
        self.dintervalsamples.setMinimum(0.001)
        self.dintervalsamples.setMaximum(1000000000.0)
        self.dintervalsamples.setSingleStep(0.1)
        self.dintervalsamples.setProperty("value", 0.2)
        self.dintervalsamples.setObjectName(_fromUtf8("dintervalsamples"))
        self.horizontalLayout_5.addWidget(self.dintervalsamples)
        self.formLayout.setLayout(5, QtGui.QFormLayout.SpanningRole,
self.horizontalLayout_5)
        self.horizontalLayout = QtGui.QHBoxLayout()
        self.horizontalLayout.setObjectName(_fromUtf8("horizontalLayout"))
        self.label_samples = QtGui.QLabel(self.page_3)
        self.label_samples.setObjectName(_fromUtf8("label_samples"))
        self.horizontalLayout.addWidget(self.label_samples)
        self.numsamples = QtGui.QSpinBox(self.page_3)
        self.numsamples.setMinimum(2)
        self.numsamples.setMaximum(100000000)
        self.numsamples.setProperty("value", 10)
        self.numsamples.setObjectName(_fromUtf8("numsamples"))
        self.horizontalLayout.addWidget(self.numsamples)
        self.formLayout.setLayout(4, QtGui.QFormLayout.SpanningRole,
self.horizontalLayout)
        self.btnconfigfm = QtGui.QPushButton(self.page_3)
        self.btnconfigfm.setObjectName(_fromUtf8("btnconfigfm"))
        self.formLayout.setWidget(2, QtGui.QFormLayout.SpanningRole, self.btnconfigfm)
        self.horizontalLayout_3 = QtGui.QHBoxLayout()
        self.horizontalLayout_3.setObjectName(_fromUtf8("horizontalLayout_3"))
        self.label_3 = QtGui.QLabel(self.page_3)
        self.label_3.setObjectName(_fromUtf8("label_3"))
        self.horizontalLayout_3.addWidget(self.label_3)
        self.cboxaveragetype = QtGui.QComboBox(self.page_3)
        self.cboxaveragetype.setObjectName(_fromUtf8("cboxaveragetype"))
        self.cboxaveragetype.addItem(_fromUtf8(""))
        self.cboxaveragetype.addItem(_fromUtf8(""))

```

```

        self.cboaveragetype.addItem(_fromUtf8(""))
        self.horizontalLayout_3.addWidget(self.cboaveragetype)
        self.formLayout.setLayout(1, QtGui.QFormLayout.SpanningRole,
self.horizontalLayout_3)
        self.horizontalLayout_2 = QtGui.QHBoxLayout()
        self.horizontalLayout_2.setObjectName(_fromUtf8("horizontalLayout_2"))
        self.label_2 = QtGui.QLabel(self.page_3)
        self.label_2.setObjectName(_fromUtf8("label_2"))
        self.horizontalLayout_2.addWidget(self.label_2)
        self.numaveragenodes = QtGui.QSpinBox(self.page_3)
        self.numaveragenodes.setMinimum(1)
        self.numaveragenodes.setMaximum(4000)
        self.numaveragenodes.setProperty("value", 20)
        self.numaveragenodes.setObjectName(_fromUtf8("numaveragenodes"))
        self.horizontalLayout_2.addWidget(self.numaveragenodes)
        self.formLayout.setLayout(0, QtGui.QFormLayout.SpanningRole,
self.horizontalLayout_2)
        self.btnfreqrequest = QtGui.QPushButton(self.page_3)
        self.btnfreqrequest.setObjectName(_fromUtf8("btnfreqrequest"))
        self.formLayout.setWidget(6, QtGui.QFormLayout.SpanningRole,
self.btnfreqrequest)
        self.stackedWidget_2.addWidget(self.page_3)
        self.page_4 = QtGui.QWidget()
        self.page_4.setObjectName(_fromUtf8("page_4"))
        self.formLayout_3 = QtGui.QFormLayout(self.page_4)
        self.formLayout_3.setFieldGrowthPolicy(QtGui.QFormLayout.AllNonFixedFieldsGrow)
        self.formLayout_3.setObjectName(_fromUtf8("formLayout_3"))
        self.horizontalLayout_7 = QtGui.QHBoxLayout()
        self.horizontalLayout_7.setObjectName(_fromUtf8("horizontalLayout_7"))
        self.label_6 = QtGui.QLabel(self.page_4)
        self.label_6.setObjectName(_fromUtf8("label_6"))
        self.horizontalLayout_7.addWidget(self.label_6)
        self.cboxsignaltypes = QtGui.QComboBox(self.page_4)
        self.cboxsignaltypes.setObjectName(_fromUtf8("cboxsignaltypes"))
        self.cboxsignaltypes.addItem(_fromUtf8(""))
        self.cboxsignaltypes.addItem(_fromUtf8(""))
        self.horizontalLayout_7.addWidget(self.cboxsignaltypes)
        self.formLayout_3.setLayout(0, QtGui.QFormLayout.SpanningRole,
self.horizontalLayout_7)
        self.horizontalLayout_8 = QtGui.QHBoxLayout()
        self.horizontalLayout_8.setObjectName(_fromUtf8("horizontalLayout_8"))
        self.label_7 = QtGui.QLabel(self.page_4)
        self.label_7.setObjectName(_fromUtf8("label_7"))
        self.horizontalLayout_8.addWidget(self.label_7)
        self.dnumfrequency = QtGui.QDoubleSpinBox(self.page_4)
        self.dnumfrequency.setMaximum(1000000000.0)
        self.dnumfrequency.setProperty("value", 5000.0)
        self.dnumfrequency.setObjectName(_fromUtf8("dnumfrequency"))
        self.horizontalLayout_8.addWidget(self.dnumfrequency)
        self.formLayout_3.setLayout(1, QtGui.QFormLayout.SpanningRole,
self.horizontalLayout_8)
        self.horizontalLayout_9 = QtGui.QHBoxLayout()
        self.horizontalLayout_9.setObjectName(_fromUtf8("horizontalLayout_9"))
        self.labeldutycycle = QtGui.QLabel(self.page_4)
        self.labeldutycycle.setObjectName(_fromUtf8("labeldutycycle"))
        self.horizontalLayout_9.addWidget(self.labeldutycycle)
        self.numdutycycle = QtGui.QSpinBox(self.page_4)
        self.numdutycycle.setMinimum(0)
        self.numdutycycle.setMaximum(100)
        self.numdutycycle.setProperty("value", 50)
        self.numdutycycle.setObjectName(_fromUtf8("numdutycycle"))
        self.horizontalLayout_9.addWidget(self.numdutycycle)
        self.formLayout_3.setLayout(2, QtGui.QFormLayout.SpanningRole,
self.horizontalLayout_9)
        self.btnconfigfg = QtGui.QPushButton(self.page_4)
        self.btnconfigfg.setObjectName(_fromUtf8("btnconfigfg"))
        self.formLayout_3.setWidget(3, QtGui.QFormLayout.SpanningRole, self.btnconfigfg)
        self.stackedWidget_2.addWidget(self.page_4)
        self.verticalLayout.addWidget(self.stackedWidget_2)
        spacerItem = QtGui.QSpacerItem(20, 40, QtGui.QSizePolicy.Minimum,
QtGui.QSizePolicy.Expanding)
        self.verticalLayout.addItem(spacerItem)
        self.dockWidget_2.setWidget(self.dockWidgetContents_2)
        MainWindow.addDockWidget(QtCore.Qt.DockWidgetArea(1), self.dockWidget_2)
        self.dockWidget_3 = QtGui.QDockWidget(MainWindow)

```

```

self.dockWidget_3.setFeatures(QtGui.QDockWidget.DockWidgetFloatable|QtGui.QDockWidget.DockWidgetMovable)
    self.dockWidget_3.setObjectName(_fromUtf8("dockWidget_3"))
    self.dockWidgetContents_3 = QtGui.QWidget()
    self.dockWidgetContents_3.setObjectName(_fromUtf8("dockWidgetContents_3"))
    self.horizontalLayout_10 = QtGui.QHBoxLayout(self.dockWidgetContents_3)
    self.horizontalLayout_10.setObjectName(_fromUtf8("horizontalLayout_10"))
    self.textbrowser = QtGui.QTextBrowser(self.dockWidgetContents_3)
    self.textbrowser.setFrameShape(QtGui.QFrame.StyledPanel)
    self.textbrowser.setFrameShadow(QtGui.QFrame.Sunken)
    self.textbrowser.setLineWidth(1)
    self.textbrowser.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)
    self.textbrowser.setObjectName(_fromUtf8("textbrowser"))
    self.horizontalLayout_10.addWidget(self.textbrowser)
    self.dockWidget_3.setWidget(self.dockWidgetContents_3)
MainWindow.addDockWidget(QtCore.Qt.DockWidgetArea(8), self.dockWidget_3)
self.menufileloadsignal = QtGui.QAction(MainWindow)
self.menufileloadsignal.setObjectName(_fromUtf8("menufileloadsignal"))
self.menufilesavetable = QtGui.QAction(MainWindow)
self.menufilesavetable.setObjectName(_fromUtf8("menufilesavetable"))
self.menufileclearable = QtGui.QAction(MainWindow)
self.menufileclearable.setObjectName(_fromUtf8("menufileclearable"))
self.menu_File.addAction(self.menufileloadsignal)
self.menu_File.addAction(self.menufilesavetable)
self.menu_File.addSeparator()
self.menu_File.addAction(self.menufileclearable)
self.menubar.addAction(self.menu_File.menuAction())
self.label_4.setBuddy(self.textserverip)
self.label_5.setBuddy(self.numserverport)
self.label.setBuddy(self.cboxviselect)
self.label_3.setBuddy(self.cboxaveragetype)
self.label_2.setBuddy(self.numaveragenodes)
self.label_6.setBuddy(self.cboxsignaltypes)
self.label_7.setBuddy(self.dnumfrequency)
self.labeldutycycle.setBuddy(self.numdutycycle)

self.retranslateUi(MainWindow)
self.stackWidget.setCurrentIndex(0)
self.stackWidget_2.setCurrentIndex(0)
QtCore.QObject.connect(self.cboxviselect,
QtCore.SIGNAL(_fromUtf8("currentIndexChanged(int)")),
self.stackWidget_2.setCurrentIndex)
    QtCore.QObject.connect(self.stackWidget_2,
QtCore.SIGNAL(_fromUtf8("currentChanged(int)")), self.cboxviselect.setCurrentIndex)
    QtCore.QObject.connect(self.stackWidget,
QtCore.SIGNAL(_fromUtf8("currentChanged(int)")), self.cboxviselect.setCurrentIndex)
    QtCore.QObject.connect(self.cboxviselect,
QtCore.SIGNAL(_fromUtf8("currentIndexChanged(int)")),
self.stackWidget.setCurrentIndex)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)
MainWindow.setTabOrder(self.textserverip, self.numserverport)
MainWindow.setTabOrder(self.numserverport, self.btnconnect)
MainWindow.setTabOrder(self.btnconnect, self.btdisconnect)
MainWindow.setTabOrder(self.btdisconnect, self.cboxviselect)
MainWindow.setTabOrder(self.cboxviselect, self.numaveragenodes)
MainWindow.setTabOrder(self.numaveragenodes, self.cboxaveragetype)
MainWindow.setTabOrder(self.cboxaveragetype, self.btnconfigfm)
MainWindow.setTabOrder(self.btnconfigfm, self.btnfreqrequest)
MainWindow.setTabOrder(self.btnfreqrequest, self.cboxsignaltypes)
MainWindow.setTabOrder(self.cboxsignaltypes, self.dnumfrequency)
MainWindow.setTabOrder(self.dnumfrequency, self.numdutycycle)
MainWindow.setTabOrder(self.numdutycycle, self.btnconfigfg)
MainWindow.setTabOrder(self.btnconfigfg, self.tablefrequency)

def retranslateUi(self, MainWindow):
    MainWindow.setWindowTitle(_translate("MainWindow", "Virtual Instrument", None))
    item = self.tablefrequency.horizontalHeaderItem(0)
    item.setText(_translate("MainWindow", "Sample", None))
    item = self.tablefrequency.horizontalHeaderItem(1)
    item.setText(_translate("MainWindow", "Frequency (Hz)", None))
    item = self.tablefrequency.horizontalHeaderItem(2)
    item.setText(_translate("MainWindow", "Date (dd/mm/yy)", None))
    item = self.tablefrequency.horizontalHeaderItem(3)
    item.setText(_translate("MainWindow", "Time", None))
    self.menu_File.setTitle(_translate("MainWindow", "&File", None))
    self.dockWidget.setWindowTitle(_translate("MainWindow", "Connection", None))

```

```

    self.label_4.setText(_translate("MainWindow", "UDP Server IP:", None))
    self.textserverip.setText(_translate("MainWindow", "172.19.5.91", None))
    self.label_5.setText(_translate("MainWindow", "UDP Server Port:", None))
    self.btnconnect.setText(_translate("MainWindow", "Connect", None))
    self.btndisconnect.setText(_translate("MainWindow", "Disconnect", None))
    self.dockWidget_2.setWindowTitle(_translate("MainWindow", "Virtual Instrument",
None))
    self.label.setText(_translate("MainWindow", "Instrument:", None))
    self.cboxviselect.setItemText(0, _translate("MainWindow", "Frequency Meter",
None))
    self.cboxviselect.setItemText(1, _translate("MainWindow", "Function Generator",
None))
    self.checkboxfreqreq.setText(_translate("MainWindow", "Auto Request", None))
    self.label_interval.setText(_translate("MainWindow", "Interval (seconds)",
None))
    self.label_samples.setText(_translate("MainWindow", "Samples:", None))
    self.btnconfigfm.setText(_translate("MainWindow", "Configure Frequency Meter",
None))
    self.label_3.setText(_translate("MainWindow", "Average Type:", None))
    self.cboxaveragetype.setItemText(0, _translate("MainWindow", "Simple", None))
    self.cboxaveragetype.setItemText(1, _translate("MainWindow", "Weighted", None))
    self.cboxaveragetype.setItemText(2, _translate("MainWindow", "Exponential",
None))
    self.label_2.setText(_translate("MainWindow", "Average Nodes:", None))
    self.btnfreqrequest.setText(_translate("MainWindow", "Frequency Request", None))
    self.label_6.setText(_translate("MainWindow", "Signal Type:", None))
    self.cboxsignaltypesetItemText(0, _translate("MainWindow", "Square", None))
    self.cboxsignaltypesetItemText(1, _translate("MainWindow", "Non-Square", None))
    self.label_7.setText(_translate("MainWindow", "Frequency (Hz):", None))
    self.labeldutycycle.setText(_translate("MainWindow", "Duty Cycle (%):", None))
    self.btnconfigfg.setText(_translate("MainWindow", "Configure Function
Generator", None))
    self.dockWidget_3.setWindowTitle(_translate("MainWindow", "Error/Warning/Info",
None))
    self.menufileloadsignal.setText(_translate("MainWindow", "Load signal", None))
    self.menufilesavetable.setText(_translate("MainWindow", "Save Table", None))
    self.menufileclearable.setText(_translate("MainWindow", "Clear Table", None))

from PyQt4 import Qwt5

```

Apéndice 2B: Protocolo VI (*vi_protocol.py*)

```

import struct

# Client (PC) commands
CMD_ACTIVATE_FREQ_METER = 100
CMD_FREQ_REQUEST = 101
CMD_NFREQ_NODES = 102
CMD_AV_SIMPLE = 103
CMD_AV_WEIGHTED = 104
CMD_AV_EXP = 105
CMD_ACTIVATE_FUNC_GEN = 106
CMD_RAM_VALUES = 107
CMD_FG_SQUARE = 108
CMD_FG_NON_SQUARE = 109
CMD_FG_CHANGE_FREQ = 110
CMD_FG_CHANGE_TON = 111
CMD_RESET = 112

# Server (board) commands
CMD_FREQ_ANSWER = 113

# Commands that are not associated to data in the data field of the protocol
command_empty = [CMD_ACTIVATE_FREQ_METER,
                  CMD_FREQ_REQUEST,
                  CMD_AV_SIMPLE,
                  CMD_AV_WEIGHTED,
                  CMD_AV_EXP,
                  CMD_ACTIVATE_FUNC_GEN,
                  CMD_FG_SQUARE,
                  CMD_FG_NON_SQUARE,
                  CMD_RESET]

# Commands that are associated to 1 data (4 bytes == int, float)
#   in the data field
command_1data = [CMD_NFREQ_NODES,
                  CMD_FG_CHANGE_FREQ,
                  CMD_FG_CHANGE_TON,
                  CMD_FREQ_ANSWER]

# Command associated to the 36 values of the CCP's RAM
command_moredata = [CMD_RAM_VALUES]

def assemble_packet(fields):
    """Assemble the packet:
    -----
    | command |          data          |
    -----
    command: command of the VI protocol
    data: int, float or anything

    return: the packet (str) with a length of 40 bytes:
            command -> 4 bytes
            data     -> 36 bytes
    """
    if not fields:
        raise 'NoArguments' # the list is empty

    # fields[0] is the command
    # fields[1] is the num
    # fields[1-37] are the RAM values if command is CMD_RAM_VALUES
    packet = ""
    if fields[0] in command_empty:

        # Command + 36 zeros to fill all the packet (40 bytes)
        packet = struct.pack("!i", fields[0]) + str(bytearray(36))
        #print repr(packet)

```

```

elif fields[0] in command_1data:

    # Command + num + 32 zeros to fill all the packet (40 bytes)
    #   num is sent as a 'unsigned long'== 'L'
    packet = struct.pack("!ii", fields[0], fields[1]) + str(bytarray(32))
    #print repr(packet)

elif fields[0] in command_moredata:

    if len(fields) != 37:
        raise 'InvalidVIData'

    # Command + 36 values of the CCP's RAM
    packet = struct.pack("!i", fields[0])
    for i in range(1, 37):
        packet = packet + struct.pack("!B", fields[i])
    #print repr(packet)

else:
    raise 'InvalidVICommand'

return packet

def disassemble_packet(packet):
    """Disassemble the received packet (40 bytes)

    It returns a list with its content
    """

    # The only packet received by the server is the command CMD_FREQ_ANSWER
    fields = []

    # Command + frequency + 32 zeros
    fields = struct.unpack("!if32c", packet)

return fields

```

Apéndice 2C: Aplicación de usuario (*main.pyw*)

```
import sys                      # System functions
import socket                   # Network connection
import struct                  # Assemble/disassemble packets
import time                     # To implement delays
import threading                # To implement threads
from PyQt4.QtCore import *      # PyQt ...
from PyQt4.QtGui import *       # ... modules
from PyQt4 import Qt, Qwt5       # QwtPlot (graphsignal)
from window import Ui_MainWindow # User Interface
from vi_protocol import *        # VI_protocol

class Form(QMainWindow, Ui_MainWindow):

    ##### Initialization functions #####
    ##

    def __init__(self):
        super(Form, self).__init__()
        self.setupUi(self)          # recreates what we designed in Qt Designer
        self.initUi()
        self.graph_signal_init()
        self.initTableFreq()

        self.write_text_browser_info("Initialization completed ... Please, "
                                     "specify IP and Port to connect to the "
                                     "server.")

    def initUi(self):
        """Initializes the user interface and connects the signals to the
        functions in order to handle the their changes.

        """

        # Disable buttons not enabled in the initialization
        self.btndisconnect.setEnabled(False)
        self.numsamples.setEnabled(False)
        self.dintervalsamples.setEnabled(False)
        self.label_samples.setEnabled(False)
        self.label_interval.setEnabled(False)

        # Connection of all the user interface signals we need
        self.connect(self.btnconnect, SIGNAL("clicked()"), self.handle_btnconnect)
        self.connect(self.btndisconnect, SIGNAL("clicked()"), self.handle_btndisconnect)
        self.connect(self.menufileloadsignal, SIGNAL("triggered()"),
                    self.handle_load_signal)
        self.connect(self.menufilesavetable, SIGNAL("triggered()"),
                    self.handle_save_table)
        self.connect(self.menufileclearable, SIGNAL("triggered()"),
                    self.handle_clear_table)
        self.connect(self.btnconfigfm, SIGNAL("clicked()"), self.handle_btnconfigfm)
        self.connect(self.btnfreqrequest, SIGNAL("clicked()"),
                    self.handle_btndfreqrequest)
        self.connect(self.checkboxfreqreq, SIGNAL("clicked()"),
                    self.handle_checkboxfreqreq)
        self.connect(self.btnconfigfg, SIGNAL("clicked()"), self.handle_btnconfigfg)
        self.connect(self.cboxsignaltyp, SIGNAL("currentIndexChanged(int)"),
                    self.handle_cboxsignaltyp)
        self.connect(self.numdutycycle, SIGNAL("valueChanged(int)"),
                    self.handle_numdutycycle)

    def graph_signal_init(self):
        """Initializes the graph of the Function Generator.

        """

        # lists of the x and y values to be plotted
        self.xaxis = list(range(36)) # list 0-350 (step of 10)
        self.yaxis = list(bytarray(36)) # list 36 zeros
```

```

# plot background and title
self.graphsignal.setCanvasBackground(Qt.Qt.white)
self.graphsignal.setTitle("Function Generator' Signal")

# show right axis
self.graphsignal.enableAxis(Qwt5.QwtPlot.yRight)

# titles of the axis
self.graphsignal.setAxisTitle(Qwt5.QwtPlot.xBottom, "Degrees")
self.graphsignal.setAxisTitle(Qwt5.QwtPlot.yRight, "Voltage (V)")
self.graphsignal.setAxisTitle(Qwt5.QwtPlot.yLeft, "Value [0-255]")

self.curve = Qwt5.QwtPlotCurve("Data") #Create a dataCurve to plot data (info
and printing format)
self.curve.attach(self.graphsignal)      #attach the wave to the plot
self.curve.setPen(QPen(Qt.Qt.blue))      #set the color of the wave

#Plot data and Replot
self.plot_axis()

def initTableFreq(self):
    """Initialize the parameters of the table of frequencies.

    """

    self.number_of_rows = 0
    self.tablefrequency.setColumnCount(4)
    self.tablefrequency.setRowCount(self.number_of_rows)

#####
## Functions to control the changes of the UI
#####
#####

def handle_btnconnect(self):
    """When the button 'Connect' is pressed we open a socket, and we launch
    the receive_pkt thread.

    """

    # Enable/Disable buttons of the connection
    self.btnconnect.setEnabled(False)
    self.btndisconnect.setEnabled(True)

    # Get data from the user interface
    self.serverip = str(self.textserverip.text())
    self.serverport = int(self.numserverport.value())
    self.remoteaddr = (self.serverip, self.serverport)

    try:
        # Open the socket
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sock.bind(('', 0)) # to get the host (IP, port) now, instead when a
        packet is sent

        #print self.sock.getsockname()

        #data = assemble_packet([CMD_RESET])
        #bytes sent = self.sock.sendto(data, self.remoteaddr)
        #print bytes_sent

        self.rcv_activated = True # it enables to create a reception socket in the
                                # reception thread

        # We launch the thread that allow us to receive packets from the server
        # It is a child thread of the main thread
        self.thread_receive_pkt = threading.Thread(target = self.receive_pkt)
        self.thread_receive_pkt.start()

        self.write_text_browser_info("Socket created ... ready to configure the "
                                    "board. Please, select the Virtual Instrument "
                                    "and its configuration.")
    
```

```

except socket.error:

    # If a socket exception occurs, we close the socket and show an error
    self.sock.close()
    self.write_text_browser_error("Failed to create the socket.")


def receive_pkt(self):
    """Function that is executed in a separated thread in order to receive
    packets from the server.

    """

    try:
        # Receive loop
        while (self.rcv_activated == True):

            packet, addr = self.sock.recvfrom(40)  # receives 40 bytes, which is the
            fixed length of                                     # VI_protocol packet

            fields = disassemble_packet(packet)

            if fields[0] == CMD_FREQ_ANSWER:

                # Put the frequency value received in a new row of the frequency
                table
                self.number_of_rows += 1
                self.tablefrequency.setRowCount(self.number_of_rows)
                self.tablefrequency.setItem(self.number_of_rows - 1, 0,
                QTableWidgetItem(str(self.number_of_rows - 1)))
                self.tablefrequency.setItem(self.number_of_rows - 1, 1,
                QTableWidgetItem(str(fields[1])))
                self.tablefrequency.setItem(self.number_of_rows - 1, 2,
                QTableWidgetItem(time.strftime("%d/%m/%Y")))
                self.tablefrequency.setItem(self.number_of_rows - 1, 3,
                QTableWidgetItem(time.strftime("%H:%M:%S")))

            else:

                self.write_text_browser_error("Command received unknown!")

    except socket.error: # The socket is closed

        self.sock.close()
        self.write_text_browser_error("The socket is closed! Please, connect again")


def handle_btndisconnect(self):
    """When the button 'Disconnect' is pressed we close the sockets that
    we had been created before.

    """

    # Close the socket
    self.sock.close()

    # Disable the receive
    self.rcv_activated = False

    # Enable/Disable buttons of the user interface
    self.btndisconnect.setEnabled(False)
    self.btnconnect.setEnabled(True)

    self.write_text_browser_warning("'Disconnect' pressed... socket closed.")


def handle_load_signal(self):
    """If the signa Non-Square is selected, we load a signal to the
    Function Generator. When the 'File->Load signal' is pressed, it
    shows a dialog where the user can select the signal's file to be loaded.

    """

    # Generates a dialog to get the function generator' signal to be loaded

```

```

filename = unicode(QFileDialog.getOpenFileName(self, 'Load Non-Square Signal',
'', ".txt(*.txt")))
if str(self.cboxsignaltypes.currentText()) == 'Non-Square':
    try:
        # Take the data from the file
        fp = file(filename, 'r')

        self.xaxis = [] # clear the xaxis list
        self.yaxis = [] # clear the yaxis list

        for line in fp:
            if line[0] == '#':
                continue
            else:
                x = line.split()
                self.xaxis.append(int(x[0]))
                self.yaxis.append(int(x[1]))

        fp.close()

        # Plot the signal with the new values of xaxis and yaxis
        self.plot_axis()

    except:
        self.write_text_browser_error("You have not selected a Non-Square
signal!")

def plot_axis(self):
    """Plot the signal (self.xaxis, self.yaxis) in the graph.

    """
    for i in range(0, 36):
        self.xaxis[i] = self.xaxis[i]*10

        # Scale the axis deppending on the data
        self.graphsignal.setAxisScale(Qwt5.QwtPlot.xBottom, 0, 360, 40.0)
        self.graphsignal.setAxisScale(Qwt5.QwtPlot.yRight, min(self.yaxis)*(3.3/256),
max(self.yaxis)*(3.3/256))
        self.graphsignal.setAxisScale(Qwt5.QwtPlot.yLeft, min(self.yaxis),
max(self.yaxis))

        # Update data
        self.curve.setData(self.xaxis, self.yaxis)
        self.graphsignal.replot()

def handle_save_table(self):
    """Saves the current table in a *.csv file.

    """
    # Generates a dialog to get the file's name
    filename = unicode(QFileDialog.getSaveFileName(self, 'Save Table', '',
'.csv(*.csv')))

    fcsv = file(filename, 'w')

    for currentRow in range(self.tablefrequency.rowCount()):
        a_0 = str(self.tablefrequency.item(currentRow, 0).text())
        a_1 = str(self.tablefrequency.item(currentRow, 1).text())
        a_2 = str(self.tablefrequency.item(currentRow, 2).text())
        a_3 = str(self.tablefrequency.item(currentRow, 3).text())

        fcsv.write("{0}, {1}, {2}, {3}\n".format(a_0, a_1, a_2, a_3))

    fcsv.close()

def handle_clear_table(self):
    """When 'File->Clear Table' is pressed, we clear the frequency table.

```

```

"""
self.number_of_rows = 0
self.tablefrequency.setColumnCount(4)
self.tablefrequency.setRowCount(self.number_of_rows)

def handle_btnconfigfm(self):
    """When the button 'Configure Frequency Meter' is pressed we send
    to the server the following commands:
        - CMD ACTIVATE FREQ METER
        - CMD AV ? (SIMPLE, WEIGHTED or EXPONENTIAL)
        - CMD_NFREQ_NODES

"""

# Get data from the user interface
average_nodes = int(self.numaveragenodes.value())
average_type = str(self.cboxaveragetype.currentText())

try:

    # Activate Frequency Meter
    data = assemble_packet([CMD_ACTIVATE_FREQ_METER])
    bytes_sent = self.sock.sendto(data, self.remoteaddr)
    #print bytes_sent

    # Send type of average
    if average_type == 'Simple':
        data = assemble_packet([CMD_AV_SIMPLE])

    elif average_type == 'Weighted':
        data = assemble_packet([CMD_AV_WEIGHTED])

    else: #average_type == 'Exponential'
        data = assemble_packet([CMD_AV_EXP])

    bytes_sent = self.sock.sendto(data, self.remoteaddr)
    #print bytes_sent

    # Send the number of average nodes
    data = assemble_packet([CMD_NFREQ_NODES, average_nodes])
    bytes_sent = self.sock.sendto(data, self.remoteaddr)
    #print bytes_sent

    self.write_text_browser_info("Configuration sent ... : "
                                "Frequency Meter, Average type = '%s', "
                                "Average nodes = %d" %(average_type,
                                average_nodes))

except socket.error: # The socket is closed
    self.write_text_browser_error("The socket is closed! Please, connect
again.")

def handle_bttnfreqrequest(self):
    """When 'Frequency Request' is pressed, we send a packet to
    the server in order to get the frequency that it's being measure by
    the frequency meter.

    If 'Auto Request' is checked, we send 'Sample' number of packets with
    'Interval' seconds interval between packets.

"""

if self.checkboxfreqreq.isChecked():      # Auto request enable

    # We call a child thread to send this packets because it will be
    # suspended (with the sleep function) and we do not want to
    # suspend the main thread (as it would suspend the main window)

```

```

        self.thread_auto_request = threading.Thread(target =
self.sending_auto_request,
                                                args = ([CMD_FREQ_REQUEST],
                                                       self.numsamples.value(),

self.dintervalsamples.value())))
        self.thread_auto_request.start()

    else:                                     # Auto request disable

        try:

            data = assemble_packet([CMD_FREQ_REQUEST])
            bytes_sent = self.sock.sendto(data, self.remoteaddr)
            #print bytes sent
            self.write_text_browser_info("Frequency request has been sent")

        except socket.error: # The socket is closed

            self.write_text_browser_error("The socket is closed! Please, connect
again.")

def sending_auto_request(self, my_list, num, delay):
    """Thread that is launch to send 'num' packets with a specific
'delay' between them.

    The data of the packet is in the list 'my_list'.

    """

    i = 0
    try:

        while i < num:

            data = assemble_packet(my_list)
            bytes_sent = self.sock.sendto(data, self.remoteaddr)
            i += 1
            #print bytes_sent
            self.write_text_browser_info("Frequency request (%d out of %d) has been
sent" % (i, num))
            time.sleep(delay)

    except socket.error: # The socket is closed

        self.write_text_browser_error("The socket is closed! Please, connect
again.")

def handle_checkboxfreqreq(self):
    """We enable or disable the auto request parameters deppending on the
state of the checkbox.

    """

    if self.checkboxfreqreq.isChecked():      # Auto request enable

        self.numsamples.setEnabled(True)
        self.dintervalsamples.setEnabled(True)
        self.label_samples.setEnabled(True)
        self.label_interval.setEnabled(True)

    else:                                     # Auto request disable

        self.numsamples.setEnabled(False)
        self.dintervalsamples.setEnabled(False)
        self.label_samples.setEnabled(False)
        self.label_interval.setEnabled(False)

def handle_btncfgfg(self):
    """When the button 'Configure Function Generator' is pressed, we send the
following packets to the server:

    - CMD_ACTIVATE_FUNCTION_GENERATOR

```

```

- CMD_FG ? (SQUARE or NON-SQUARE)
- CMD RAM VALUES (only if Non-Square signal is selected)
- CMD FG CHANGE FREQ
- CMD_CHANGE_TON (only if Square signal is selected)

"""

# Get data from the user interface
frequency = long(self.dnumfrequency.value())
dcycle = long(self.numdutycycle.value())
signal_type = str(self.cboxsignaltypes.currentText())

try:

    # Activate Function Generator
    data = assemble_packet([CMD_ACTIVATE_FUNC_GEN])
    bytes_sent = self.sock.sendto(data, self.remoteaddr)
    #print bytes_sent

    # Send frequency
    data = assemble_packet([CMD_FG_CHANGE_FREQ, frequency])
    bytes_sent = self.sock.sendto(data, self.remoteaddr)
    #print bytes_sent

    if signal_type == 'Square':

        # Send Square signal
        data = assemble_packet([CMD_FG_SQUARE])
        bytes_sent = self.sock.sendto(data, self.remoteaddr)
        #print bytes_sent

        # Send Ton
        data = assemble_packet([CMD_FG_CHANGE_TON, dcycle])
        bytes_sent = self.sock.sendto(data, self.remoteaddr)
        #print bytes_sent

        self.write_text_browser_info("Configuration sent ... : "
                                     "Function Generator, '%s' signal, "
                                     "Frequency = %s Hz, DCycle = %s%%"
                                     %(signal_type, str(frequency), str(dcycle)))

    elif signal_type == 'Non-Square':

        # Send Non-Square signal
        data = assemble_packet([CMD_FG_NON_SQUARE])
        bytes_sent = self.sock.sendto(data, self.remoteaddr)
        #print bytes_sent

        # Send the values of the Non-Square signal
        data = assemble_packet([CMD_RAM_VALUES] + self.yaxis)
        bytes_sent = self.sock.sendto(data, self.remoteaddr)
        #print bytes_sent

        self.write_text_browser_info("Configuration sent ... : "
                                     "Function Generator, '%s' signal, "
                                     "Frequency = %s Hz"
                                     %(signal_type, str(frequency)))

        self.write_text_browser_info("....."
                                     "RAM values = %s" % str(self.yaxis))

    else:
        pass
        #raise 'ErrorSignalType'

except socket.error: # The socket is closed

    self.write_text_browser_error("The socket is closed! Please, connect again.")

def handle_cboxsignaltypes(self):
    """Enable or disable the duty cycle options.

"""

    if self.cboxsignaltypes.currentText() == 'Square':

```

```

# Duty Cycle enabled
self.numdutycycle.setEnabled(True)
self.labeldutycycle.setEnabled(True)
self.handle_numdutycycle()

elif self.cboxsignaltyp.currentText() == 'Non-Square':

    # Duty Cycle disabled
    self.numdutycycle.setEnabled(False)
    self.labeldutycycle.setEnabled(False)

    # We clear the graph
    self.xaxis = list(range(36))      # list 0-35
    self.yaxis = list(bytarray(36))    # list 36 zeros
    self.plot_axis()

def handle_numdutycycle(self):
    """If the user varies the duty cycle we refresh the signal in the graph.

    """

    dc = int(self.numdutycycle.value())

    self.xaxis = list(range(36))      # list 0-36
    self.yaxis = list(bytarray(36))    # list 36 zeros

    n_zeros = int(round((36.0/(100.0/dc))))

    for i in range(0, n_zeros):
        self.yaxis[i] = 255

    # Refresh the graph
    self.plot_axis()

def write_text_browser_error(self, error):
    """Write ERROR in the text browser.

    """

    self.textbrowser.append("<font color=red>>> ERROR:</font> <font size=4
color=red> %s</font>" % error)

def write_text_browser_warning(self, warning):
    """Write WARNING in the text browser.

    """

    self.textbrowser.append("<font color=blue>>> WARNING:</font> <font size=4
color=black> %s</font>" % warning)

def write_text_browser_info(self, info):
    """Write INFO in the text browser.

    """

    self.textbrowser.append("<font color=green>>> INFO:</font> <font size=4
color=black> %s</font>" % info)

# Run the application
if __name__ == '__main__':
    app = QApplication(sys.argv)
    form = Form()
    form.show()
    sys.exit(app.exec_())

```

BIBLIOGRAFÍA

- [1].- Altera. (2011). *Embedded Design Handbook*. Obtenido de
http://www.altera.com/literature/hb/nios2/edh_ed_handbook.pdf?GSA_pos=2&WT.oss_r=1&WT.oss=embedded%20design%20handbook
- [2].- Altera. (2011). *Nios II Hardware Development Tutorial*. Obtenido de
http://www.altera.com/literature/tt/tt_nios2_hardware_tutorial.pdf?GSA_pos=3&WT.oss_r=1&WT.oss=nios%20ii%20hardware%20development%20tutorial
- [3].- Altera. (2012). *DE2 User Manual*. Obtenido de
ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf
- [4].- Altera. (2014). *Embedded Peripheral IP User Guide*. Obtenido de
http://www.altera.com/literature/ug/ug_embedded_ip.pdf?GSA_pos=1&WT.oss_r=1&WT.oss=embedded%20peripherals%20ip%20user%20guide
- [5].- Altera. (2014). *Nios II Processor Reference Handbook*. Obtenido de
http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf?GSA_pos=1&WT.oss_r=1&WT.oss=nios%20ii%20handbook
- [6].- Altera. (2014). *Nios II Software Developer's Handbook*. Obtenido de
http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf?GSA_pos=1&WT.oss_r=1&WT.oss=nios%20ii%20software%20handbook
- [7].- Álvarez Ruiz de Ojeda, L. (2004). *Diseño digital con lógica programable*. Santiago de Compostela: Tórculo.
- [8].- Davicom Semiconductor. (2006). *DM9000A Datasheet*. Obtenido de
<http://www.davicom.com.tw/userfile/24247/DM9000A-DS-F01-101906.pdf>
- [9].- González, V. (2013). *PFC: Instrumentación electrónica basada en procesador empotrado en lógica programable*.
- [10].- Hollenbeck, R. (2001). *The IEEE 802.3 Standard (Ethernet): An Overview of the Technology*.
- [11].- *RFC 768: User Datagram Protocol*. (1980). Obtenido de
<https://www.ietf.org/rfc/rfc768.txt>
- [12].- *RFC 791: Internet Protocol*. (1981). Obtenido de <https://www.ietf.org/rfc/rfc791.txt>
- [13].- Röst, H. e. (2012). *Python Programming*. Obtenido de
http://upload.wikimedia.org/wikipedia/commons/9/91/Python_Programming.pdf
- [14].- Semiconductor, D. (2006). Obtenido de DM9000A Data Sheet:
<http://www.davicom.com.tw/userfile/24247/DM9000A-DS-F01-101906.pdf>
- [15].- Spurgeon, C. (2000). *Ethernet: the definitive guide*. Sebastopol: O'Reilly.
- [16].- Summerfield, M. (2007). *Rapid GUI Programming with Python and Qt*. Prentice Hall.

[17].- van Rossum, G. (2014). *Python 2.7.8 Documentation*. Obtenido de
<https://docs.python.org/2/>