UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

SISTEMA DE VERIFICACIÓN E IDENTIFICACIÓN DE INTERLOCUTOR

Autor: José Manuel Villapún Sánchez

Director: Rafael Gadea Gironés

Fecha de comienzo: 01/09/2016

INDICE

1. Introducción

En la historia actual de la ciencia, uno de los sectores que más ha avanzado en los últimos años ha sido el de las tecnologías de información y de la comunicación, y dentro de este campo nos encontramos con el de la electrónica. Este último ha avanzado a pasos agigantados movido principalmente por la electrónica de consumo, donde se intenta cada vez el desarrollo de dispositivos más pequeños y portables.

Actualmente esta tecnología ha evolucionado hasta el punto de que en los circuitos integrados de última generación nos encontramos tecnología de 10 nanómetros (tamaño de los transistores), y se espera que en los próximos años se llegue hasta los 5 nanómetros, lo que demuestra que se trata de un campo en constante evolución.

De todo esto sacamos la importancia de estar en constante aprendizaje en lo que se refiere a este campo, ya que cada vez tenemos dispositivos con mayores prestaciones y potencia, lo que los hace más complejos y nos obliga a aprender nuevos métodos y herramientas de trabajo para poder utilizarlos en nuestros diseños.

En este trabajo se hace uso de estas tecnologías de última generación, en donde se hace uso de una FPGA con tecnología de 28 nanómetros con un procesador con arquitectura ARM de dos núcleos embebido, y en donde se utilizan las últimas herramientas disponibles para el desarrollo sobre este dispositivo. Lo que nos da todos los elementos necesarios para el diseño final que se busca.

Por otro lado, en los últimos años se ha escuchado hablar mucho de la inteligencia artificial (Redes neuronales artificiales) y de sus avances (siempre ligados a los avances en la potencia de computación de los equipos electrónicos) y aplicaciones en los distintos campos de la ciencia.

Las Redes Neuronales Artificiales, ANN (Artificial Neural Networks) están inspiradas en las redes neuronales biológicas del cerebro humano. Están constituidas por elementos que se comportan de forma similar a la neurona biológica en sus funciones más comunes. Estos elementos están organizados de una forma parecida a la que presenta el cerebro humano.

Las ANN al margen de "parecerse" al cerebro presentan una serie de características propias del cerebro. Por ejemplo, las ANN aprenden de la experiencia, generalizan de ejemplos previos a ejemplos nuevos y abstraen las características principales de una serie de datos.

Este trabajo hace uso de redes neuronales para resolver el objetivo final del trabajo, el cual plantea el uso de estas para el reconocimiento de fonemas y de interlocutores.

En la siguiente imagen (Figura 1-1) se puede ver el sistema final.

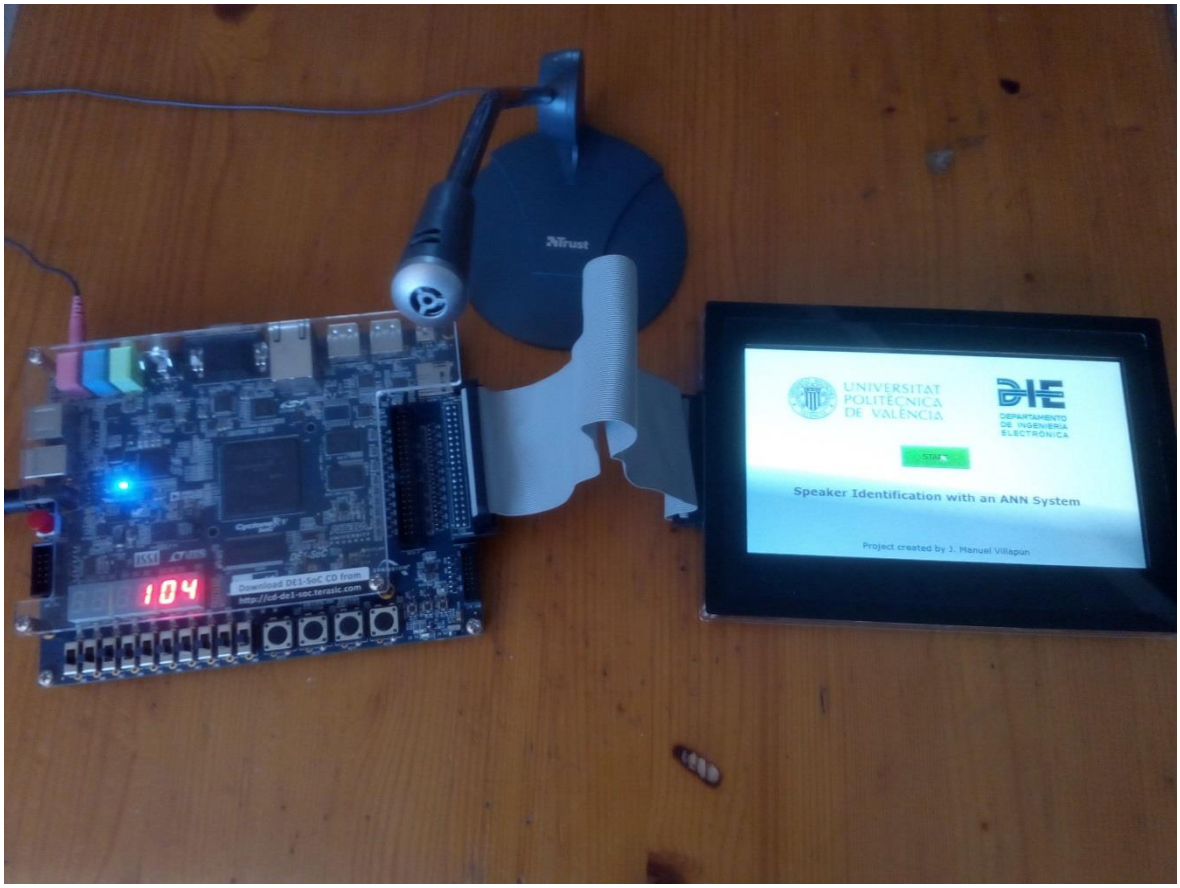


Figura 1-1 – Sistema de desarrrollo

1.1.Motivación

La principal motivación que me ha impulsado a elegir este tema de trabajo para su desarrollo ha sido que se trata de un sistema bastante completo, en donde se tocan diferentes campos de la electrónica digital, como son el desarrollo sobre sistemas procesador utilizando Linux embebido, sobre sistemas microprocesador y el desarrollo hardware sobre FPGAs y de procesamiento de señal.

Al elegir un desarrollo en donde se abordan diferentes campos de la electrónica digital, creo que el aprendizaje de los mismos va a ser de utilidad en mi futura vida laboral, ya que se trata de campos en constante evolución y en los que hay que estar siempre en constante aprendizaje, pero siempre teniendo en cuenta las base sobre las que han evolucionado.

Por otro lado, también me ha motiva el uso de redes neuronales ya que últimamente se encuentran en auge por su demostrada eficacia en diferentes campos de la ciencia, por ello me he propuesto aprender sobre su uso en este trabajo.

1.2. Objetivos

En este trabajo se han marcado diferentes objetivos, todos ellos dirigidos al aprendizaje sobre el uso de sistemas de electrónica digital de última generación. A continuación, se detallan los principales objetivos:

El principal objetivo de este trabajo es realizar el diseño de un sistema electrónico que nos permita mediante un estímulo de entrada, que en este caso serán fonemas, identificar y verificar uno o diferentes interlocutores, en donde se dispondrá de una interface de usuario de fácil manejo y todo lo necesario para poder hacer todas las pruebas experimentales necesarias.

Otro de los objetivos es conseguir que el sistema funcione en tiempo real, para ello se hará uso de redes neuronales y de procesamiento de señal sobre FPGA para conseguir la mayor rapidez posible en la identificación y verificación.

Por otro lado, se plantea el objetivo del uso de la última generación de FGPA's que incorporan un procesador embebido dentro de la propia FPGA y haciendo uso de una distribución de Linux Embebido sobre este procesador. De esta forma se plantea como objetivo realizar el control e interface de usuario sobre Linux y teniendo como interface hacia fuera una pantalla LCD multitouch.

El objetivo final será comprobar que el sistema funciona y se comporta según lo especificado, para ello se harán todas las pruebas de testeo necesarias para verificar el correcto funcionamiento. Estas pruebas contendrán el testeo de la red neuronal, mediante su entrenamiento y posterior prueba de identificación y verificación, realizando una estadística sobre el correcto funcionamiento. Otro caso de testeo, aunque más sencillo y rápido, sería la prueba de la interface de usuario, donde se tendrá que probar que todas las opciones funcionan correctamente.

1.3. Estructura del trabajo

Este trabajo, como se refleja en las diferentes secciones de este informe, se ha estructurado en diferentes unidades o partes, para de este modo facilitar el desarrollo y pruebas del mismo. A continuación, se exponen las diferentes partes de las que está compuesto el trabajo:

- La primera parte de este trabajo se compone del hardware y herramientas que se han utilizado para el desarrollo del mismo. Esta parte se expone en el apartado 2 de este informe, en donde se profundiza sobre la placa de desarrollo utilizada y las diferentes herramientas utilizadas.
- La segunda parte de este trabajo se corresponde con el diseño y desarrollo hardware. Esta parte se expone en el apartado 3 de este informe.
- La tercera parte, que se refleja en el apartado 4 del informa, se corresponde con el diseño y desarrollo software que se ha realizado.
- Por último, se expone una cuarta parte que se corresponde con la pruebas y resultados experimentales, en donde se explican las diferentes pruebas realizadas para validar el correcto funcionamiento del sistema, y que podemos encontrar en el apartado 5 de este informe.

2. Entorno de Trabajo

En esta sección se expone el entorno de trabajo utilizado para el desarrollo de este proyecto, haciendo mención al sistema hardware utilizado para probar las diferentes partes y a las herramientas utilizadas para su diseño y desarrollo.

2.1. Entorno Hardware

En esta parte se va hacer una introducción de la placa de desarrollo DE1-SoC de Terasic, en donde se profundizará en las diferentes partes de las que se ha hecho uso en este trabajo, así como la FPGA Cyclone V y su hardprocessor compuesto por dos núcleos ARM CORTEX-A9, su módulo de adquisición de audio que utiliza un ADC/DAC y sus diferentes partes de pulsadores, interruptores e indicadores. También se hará mención a la pantalla Touch-Panel utilizada para la interface de usuario.

2.1.1. Placa de desarrollo DE1-SoC

El kit de desarrollo DE1-SoC está compuesta por una robusta plataforma de diseño de hardware construida sobre un sistema FPGA-SoC (System on chip) de Altera, que combina los últimos núcleos embebidos Cortex-A9 de doble núcleo con una lógica programable líder en la industria y que permite gran flexibilidad en el diseño. El SoC de Altera integra un Hard-processor basado en ARM (HPS) que consiste en procesador, periféricos e interfaces de memoria interconectados perfectamente con la FPGA usando un bus de interconexión de alto ancho de banda. La placa de desarrollo DE1-SoC está equipada con memoria DDR3 de alta velocidad, capacidades de vídeo y audio, Ethernet, etc.

La parte FPGA de la placa está compuesta por los siguientes componentes:

- Altera Cyclone® V SE 5CSEMA5F31C6Ndevice
- Altera serial configuration device –EPCQ256
- USB-Blaster II on board for programming; JTAG Mode
- 64MB SDRAM (16-bit data bus)
- 4 push-buttons
- 10slide switches
- 10red user LEDs
- Six 7-segment displays
- Four 50MHz clock sources from the clock generator
- 24-bit CD-quality audio CODEC with line-in, line-out, and microphone-in jacks
- VGA DAC (8-bit high-speed triple DACs) with VGA-out connector
- TV decoder (NTSC/PAL/SECAM) and TV-in connector
- PS/2 mouse/keyboard connector
- IR receiverand IR emitter
- Two40-pin expansion header with diode protection
- A/D converter, 4-pin SPI interface with FPGA

La parte del HPS (Hard Processor System) de la placa está compuesta por los siguientes componentes:

- 800MHz Dual-core ARM Cortex-A9 MPCore processor
- 1GB DDR3 SDRAM (32-bit data bus)
- 1Gigabit Ethernet PHY with RJ45 connector
- 2-port USB Host, normal Type-A USB connector
- Micro SD card socket
- Accelerometer (I2Cinterface + interrupt)
- UART to USB, USB Mini-B connector
- Warm reset button and cold reset button
- One user button and one user LED
- LTC 2x7 expansion header

En la siguiente imagen (Figura 2-1) se puede ver la placa DE1-SoC utilizada en este trabajo, con las indicaciones de los componentes por los que está compuesta:

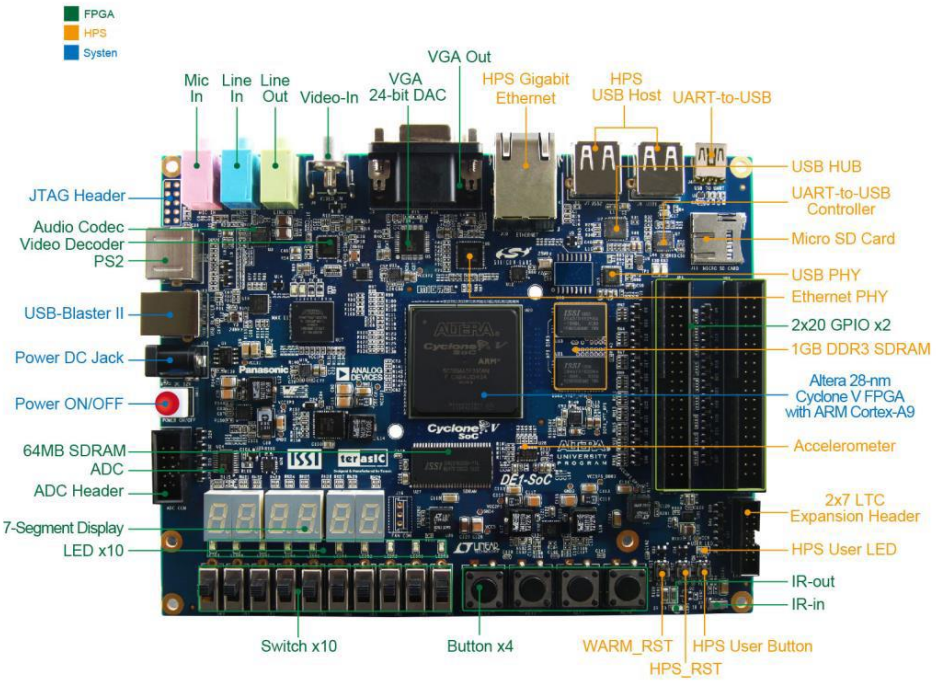


Figura 2-1 – Placa de desarrollo DE1-SoC

2.1.2. Especificación FPGA – HPS

- Cyclone V SoC5CSEMA5F31Device
- Dual-core ARM Cortex-A9 (HPS)

- 85K programmable logic elements
- 4,450Kbits embedded memory
- 6 fractional PLLs
- 2 hard memory controllers

Las FPGA, con su capacidad de paralelismo y su maleabilidad, son dispositivos idóneos para operaciones que requieren altas velocidades o grandes cantidades de datos. Su programación, sin embargo, es complicada por lo que el tiempo requerido para diseñar incluso el programa de control más simple es mucho más elevado que en un microcontrolador.

Para solventar este problema sin necesidad de requerir hardware externo apareció la idea de los soft-cores, sistemas procesadores diseñados aprovechando la misma lógica configurable de las FPGA. Estas CPU, sin embargo, sufren en velocidad comparadas con los procesadores hard, aquellos diseñados directamente en silicio.

Por otro lado, desde hace un tiempo los distintos fabricantes de FPGA han ido añadiendo diversos elementos hard a las mismas, como memorias, multiplicadores o núcleos DSP, que permiten una ejecución mucho más rápida que la realizada con lógica programable. Es, por tanto, una evolución natural de ambas tendencias la aparición de CPU integradas, llamadas hard-cores, junto a una FPGA en el mismo chip.

Este es el caso de las Cyclone-V de Altera o las Zynq-7000 de Xilinx. Ambas series disponen de un procesador ARM Cortex-A9 de doble núcleo capaz de funcionar a 800MHz, no muy diferente de los disponibles en smartphones o tablets de gama media-alta. Estos procesadores son capaces, de hecho, de ejecutar sistemas operativos complejos, siendo Linux común en esta clase de dispositivos.

La coexistencia con una FPGA, por su parte, permite una gran flexibilidad, al hacer posible el diseño de aceleradores hardware, la utilización de transductores de alta velocidad o cualquiera de las muchas opciones que la lógica programable ofrece, junto a la simplicidad de programación de una CPU estándar sin las limitaciones de velocidad que ofrezcan los soft-cores.

Es importante notar que esta clase de dispositivos incorporan el HPS y la FPGA de manera prácticamente separada, cada sección con sus propios pines independientes en el integrado. Un diagrama de bloques de una Cyclone V puede verse en la figura 2-2. Las posibilidades de esta combinación vienen, pues, limitadas por las posibilidades de interconexión entre el hard-core y la lógica programable.

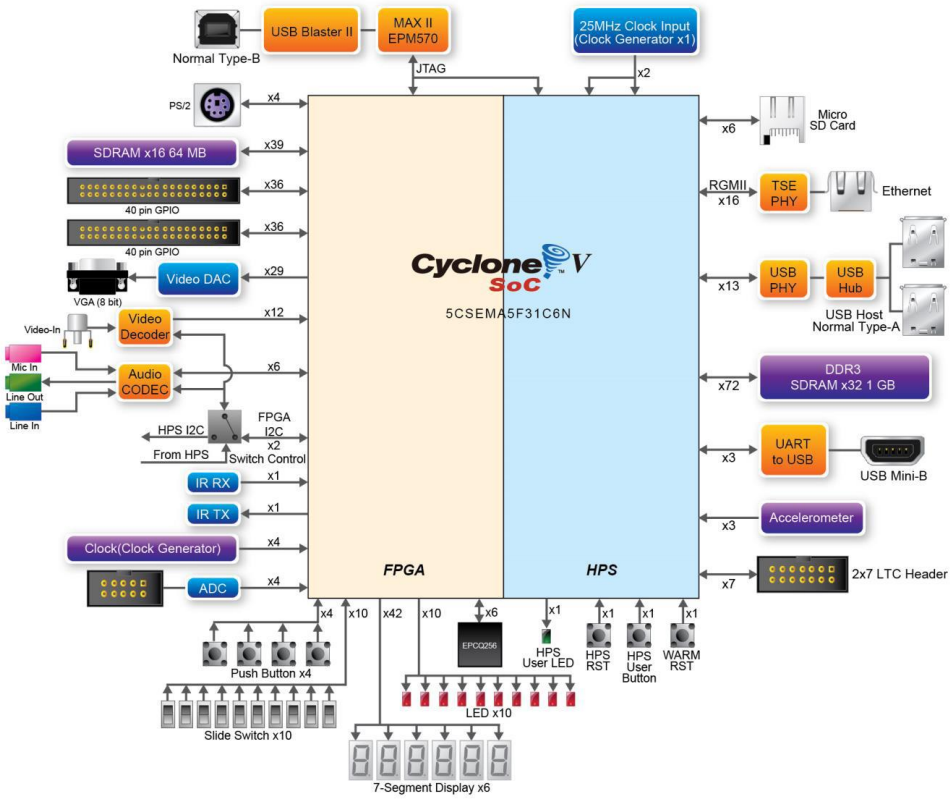


Figura 2-2– Diagrama de bloques Cyclone V

2.1.3. Interconexión HPS-FPGA

En Altera SoC FPGA, la lógica HPS y el tejido FPGA se conectan a través del puente AXI (Advanced eXtensible Interface). Para que la lógica de HPS se comunique con el tejido FPGA, la herramienta de integración de sistemas Altera Qsys se debe usar para diseñar el sistema. El sistema debe incluir el componente Altera HPS. Desde el puerto maestro AXI del componente HPS, HPS puede acceder a aquellos componentes Qsys cuyos puertos esclavos asignados a memoria están conectados al puerto maestro.

Los interfaces AXI/Avalon-MM son probablemente los que más frecuentemente se vayan a utilizar en un diseño mixto FPGA-HPS. Estos permiten su utilización con todos los componentes Avalon-MM ya disponibles en Qsys, así como con componentes diseñados para el bus AXI de ARM. Estos interfaces se conectan directamente con el bus AXI del ARM, formando parte del mapa de memoria del mismo.

Cada uno de estos interfaces posee una entrada de reloj que, obviamente, ha de compartir con todos los demás periféricos conectados a él. Esto permite que cada uno puede funcionar en dominios temporales distintos, realizándose internamente las traducciones necesarias.

El HPS contiene los siguientes puentes HPS-FPGA AXI:

- Puente FPGA-HPS
- Puente HPS a FPGA
- Puente ligero de HPS a FPGA

La Figura 2-3 muestra un diagrama de bloques de los puentes AXI en el contexto del tejido FPGA y la interconexión L3 al HPS. Cada interfaz maestra (M) y esclava (S) se muestra con su ancho (s) de datos.

El dominio del reloj para cada interconexión se muestra entre paréntesis.

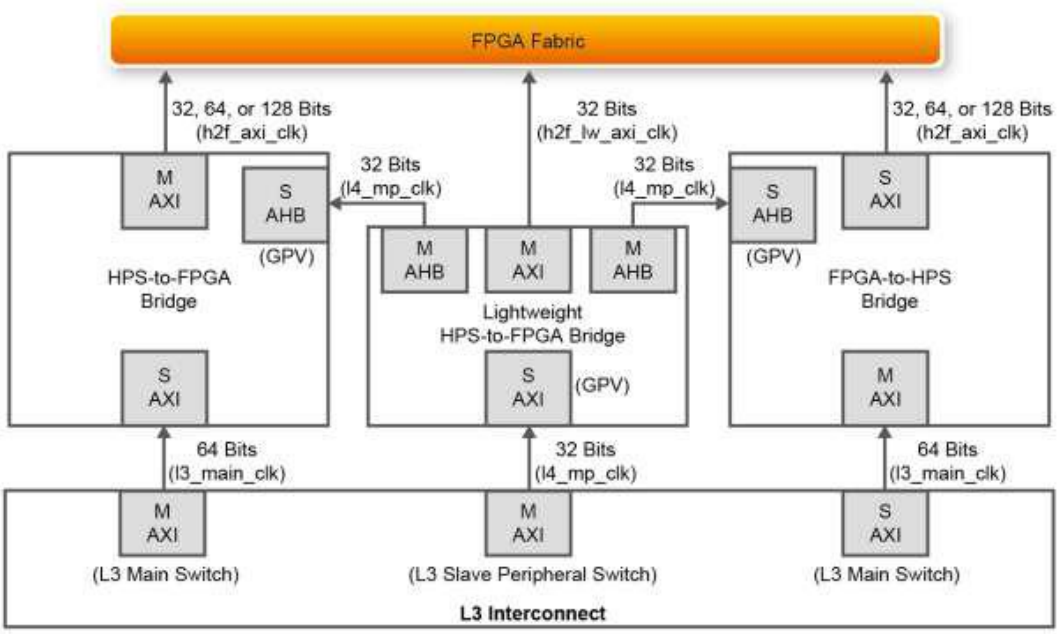


Figura 2-3 – Diagrama de bloques AXI

El puente HPS a FPGA es dominado por el interruptor principal de nivel 3 (L3) y el puente ligero de HPS a FPGA es dominado por el conmutador periférico esclavo L3. En el Cuarto de esta demostración, puente HPS-FPGA se utiliza para ARM / HSP para controlar los LED conectados a FPGA.

El puente FPGA-a-HPS controla el interruptor principal L3, permitiendo que cualquier maestro implementado en el tejido FPGA acceda a la mayoría de los esclavos en el HPS. Por ejemplo, el puente FPGA-HPS puede acceder a la coherencia del acelerador.

Los tres puentes contienen el registro GPV global de la vista del programador. El registro GPV controla el comportamiento del puente. El acceso a los registros GPV de los tres puentes se proporciona a través del puente ligero de HPS a FPGA.

Al formar parte del mapa de memoria, para acceder desde el HPS a dispositivos esclavos conectados a cualquiera de los interfaces maestros del mismo no hay más que sumar a la dirección configurada en Qsys la dirección base del bus y acceder a un puntero apuntando al resultado. Por supuesto, esto se refiere a direcciones físicas. El acceso se complica si se está utilizando el MMU o un sistema con memoria virtual o memoria paginada, teniendo que tener en cuenta las traducciones hechas o utilizar funciones de sistema operativo que realicen la traducción necesaria.

Los tres buses que se describen a continuación, que disponen además de registros de control que permiten configurar diversas opciones de bajo nivel, así como comprobar el estado de los interfaces.

- Puente HPS-FPGA.

El primer interfaz es el que permite el control desde el HPS de elementos en la lógica programable. Este es un bus de 32, 64 o 128 bits diseñado para ofrecer altas prestaciones, especialmente indicado para su conexión a memorias u otros esclavos que se beneficien de un gran ancho de banda. Es el equivalente de un interfaz Avalon-MM Máster, pudiendo conectarse a cualquier Avalon-MM Slave.

Este interfaz puede acceder hasta 960MB, siendo la dirección base del mismo 0xC0000000.

- HPS-FPGA Lightweight

Este segundo interfaz, de 32 bit únicamente, también permite el acceso a esclavos en la FPGA desde el HPS. A diferencia del anterior, sin embargo, el interfaz lightweight posee mucho menor rendimiento estando especialmente indicado para la lectura y escritura en registros de estado y control, transacciones que requieren poco ancho de banda y que podrán suponer una carga innecesaria para un bus de alta velocidad que requiera acceder a grandes cantidades de datos.

Este bus se encuentra situado en la dirección 0xFF200000 y posee un rango de acceso de 2MB.

- FPGA-HPS

El último bus disponible es el inverso a los anteriores. Este es un bus esclavo que permite el acceso al mapa de memoria del HPS desde un maestro situado en la FPGA. Este bus puede configurarse con 32, 64 o 128 bits y, al igual que el bus HPS-FPGA, permite una comunicación de altas prestaciones.

Es importante darse cuenta de que este bus accede a los 4GB de direcciones disponibles en el HPS. Su tamaño, pues, ocupa la totalidad de los 32bits de direcciones que posee el bus Avalon-MM. En un sistema complejo en el que un máster tuviese que acceder a varios dispositivos junto a regiones concretas de memoria del HPS sería necesario limitar el rango accesible. Para ello Altera recomienda usar el componente Address Span Extender que provee de un slave de tamaño configurable puenteado, con un offset, a un máster de mayor tamaño por lo que permite, en la práctica, “recortar" el rango disponible.

En la siguiente imagen (Figura 2-4) podemos ver el mapa de memoria en el que trabajan estos tres buses dentro del HPS.

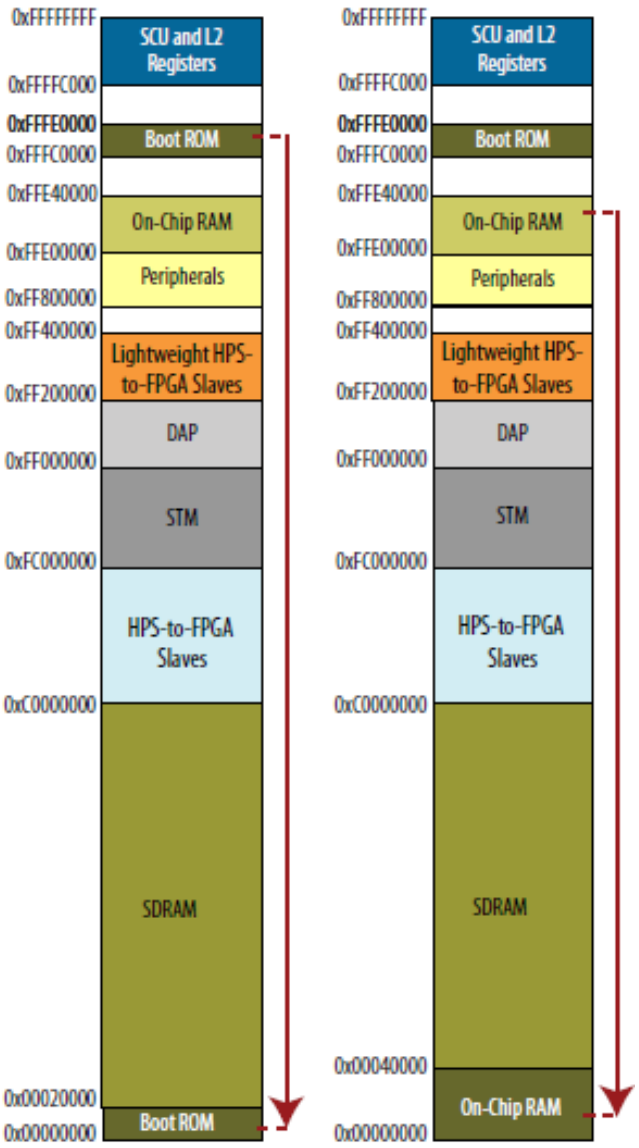


Figura 2-4 – Mapa de memoria HPS

2.1.4. Microprocesador NIOS II

El Nios II es un microprocesador configurable proporcionado por Altera para ser utilizado en sus FPGAs [9]. Se trata de un procesador RISC de 32 bits de propósito general, basado en una arquitectura tipo Harvard con buses separados para instrucciones y para datos. El núcleo se puede configurar en cualquiera de las tres versiones disponibles según se busque minimizar los recursos de la FPGA o maximiza el rendimiento. Estas tres versiones son: Nios II/f "fast": Procesador diseñado para alto rendimiento optimizado en velocidad, el cual presenta el máximo número de opciones en su configuración (MMU, MPU, Pipeline, etc.) Nios II/s "standard": Procesador diseñado para ocupar pocos recursos mientras mantiene un rendimiento moderado. Este procesador tiene un pipeline de 5 etapas. Nios II/e "economic": Procesador diseñado para ocupar el menor espacio posible y con un rendimiento muy limitado. Este procesador necesita 6 ciclos de reloj para ejecutar una instrucción, no tiene pipeline y no es necesario tener licencia. En nuestro caso hemos utilizado el procesador Nios II/e ya que es no nos hace falta licencia. Este hecho nos limita mucho las características de nuestro proyecto, ya que para llenar nuestro buffer de memoria nos generamos una interrupción cada 10us con un temporizador. Este tiempo puede ser menor si utilizamos un procesador Nios II/f, dos procesadores y/o un el core Floating Point Hardware. En una segunda versión del trabajo se podría optimizar para conseguir un periodo de muestreo mucho más bajo.

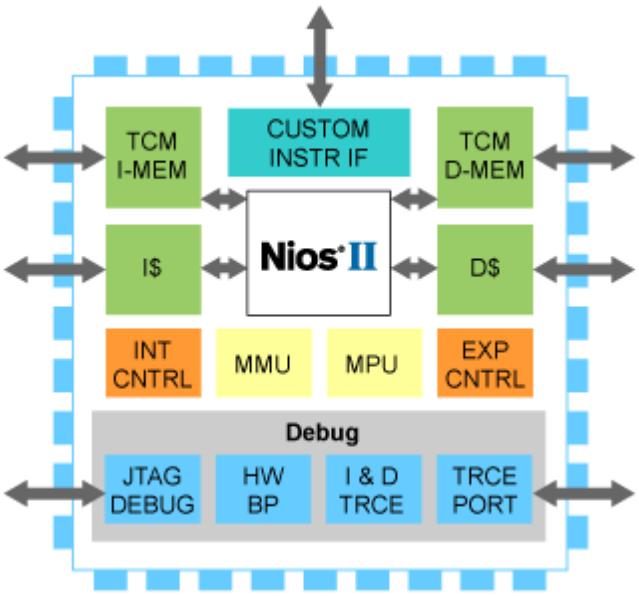


Figura 2-5 – Diagrama de bloques Nios II

Más definiciones de los elementos del procesador Nios II como los buses Avalon Slave y Master, gestión de excepciones e interrupciones, conexión y acceso a memoria y periféricos se mencionan en la bibliografía. [15].

2.1.5. Audio CODEC

La placa de desarrollo DE1-SoC contiene una contiene un audio CODEC WM8731 del fabricante WOLFSON Microelectronics con las siguientes características:

- Controlador de auriculares altamente eficiente.
- Rendimiento de audio.
  - ADC SNR 90dB ('A' ponderado) a 3.3V, 85dB a 1.8V.
  - DAC SNR 100dB ('A' ponderado) a 3.3V, 95dB a 1.8V.
- Baja potencia.
  - Reproducción sólo 22mW, 8mW ('L' Variante).
  - Paso analógico a través de 12mW, 3.5mW ('L' variante).
  - Operación de suministro digital de 1.42 - 3.6V.
  - Operación de alimentación analógica de 2.7 – 3.6V.
  - 1.8 - 3.6V de funcionamiento de la alimentación analógica ('L' Variant).
- ADC y DAC con frecuencia de muestreo: 8kHz - 96kHz.
- Filtro de paso alto ADC seleccionable.
- Interfaz de control serie MPU de 2 ó 3 líneas.
- Modos de interfaz de datos de audio programables.
  - I2S, izquierda, derecha justificada o DSP.
  - Longitud de palabra de 16/20/24/32 bits.
  - Modo de reloj Master o Slave.
- Entrada de micrófono y Electret Bias con mezclador de tono lateral.
- Disponible en paquete SSOP de 28 derivaciones o en QFN de 28 derivaciones.

En la siguiente imagen (Figura 2-6) podemos ver el diagrama de bloques del integrado:

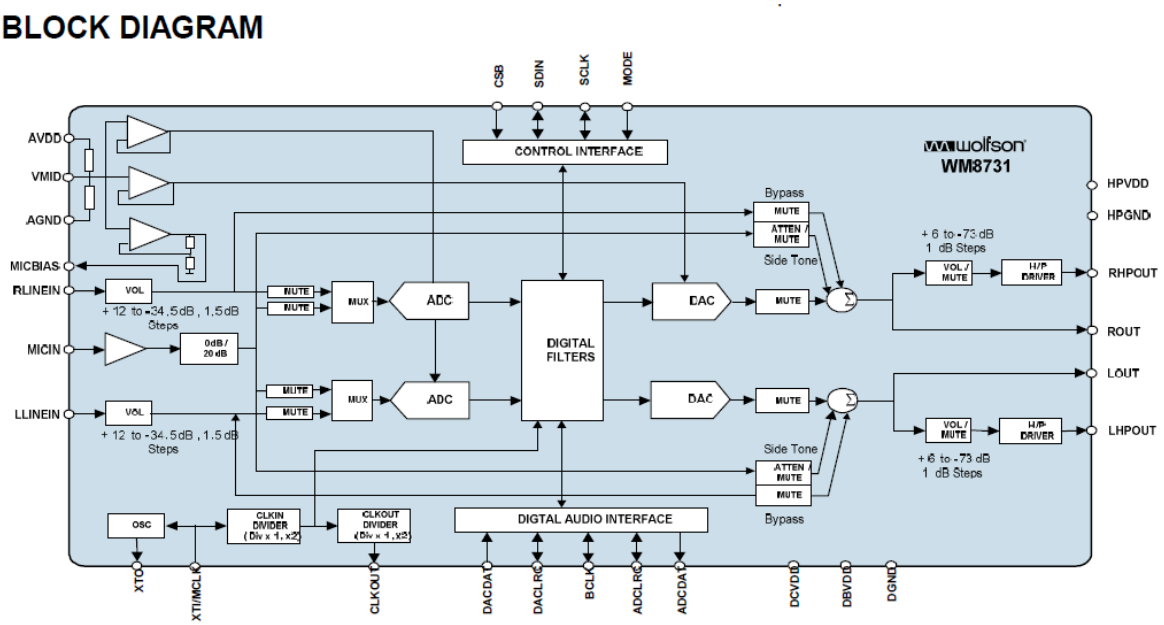


Figura 2-6 – Diagrama de bloques del Audio CODEC

El WM8731 o es un CODECs de baja potencia con un controlador de auriculares integrado. TheWM8731 está diseñado específicamente para reproductores de audio y voz MP3 portátiles y grabadoras. El WM8731 también es ideal para las máquinas MD, CD-RW y grabadoras DAT.

Se suministran entradas de audio de nivel de línea estéreo y micrófono mono, junto con una función de silenciamiento, control de volumen programable de nivel de línea y una salida de tensión de polarización adecuada para un micrófono tipo electret.

Los ADCs y los DACs sigma delta de múltiples bits con 24 bits estéreo se utilizan con interpolación digital de sobremuestreo y filtros de decimación. Se admiten las longitudes de palabra de entrada de audio digital de 16-32 bits y velocidades de muestreo de 8kHz a 96kHz.

Las salidas de audio estéreo se almacenan en un búffer para la conducción de auriculares desde un control de volumen programable, las salidas de nivel de línea también se proporcionan junto con el silencio anti-golpe y los circuitos de encendido / apagado.

El dispositivo se controla a través de una interfaz serie de 2 o 3 líneas. La interfaz proporciona acceso a todas las funciones, incluyendo controles de volumen, silenciadores, de-énfasis y extensas instalaciones de administración de energía.

En la sección de configuración hardware de este trabajo se detalla la configuración y el funcionamiento del sistema con este dispositivo.

2.1.6. Terasic Multi-Touch

Para facilitar el control final del sistema diseñado en este trabajo, se ha decidido incorporar una pantalla táctil sobre la que poder operar de forma fácil e intuitiva. Para ello se ha utilizado el dispositivo Terasic Multi-Touch que proporciona el mismo fabricante que la placa DE1-SoC, lo que ha sido de bastante ayudo, ya que debido a ser del mismo fabricante de placas de desarrollo Altera, se proporcionan diseños de ejemplo de uso de esta pantalla, de los que se ha podido aprovechar los módulos de control que están implementados en la FPGA. También al ser un dispositivo del mismo fabricante, este proporciona todo lo necesario para la conexión con la placa de desarrollo sin la necesidad de realizar ningún montaje.

Este dispositivo LCD táctil va a tener las siguientes características.

- LCD
    - 7-inch panel
    - 800x480 pixels
    - 24-bits color depth
    - TTL interface

- Multi-touch

  - Capacitive Touch Screen
  - Multi-touch Gestures
  - Single Touch Support

- Interface
  - IDE Connector
  - 2x20 GPIO connector with ITG (IDE to GPIO) adapter

- Gesture Table
  - One Point
  - Two Point

En la siguiente imagen (Figura 2-7) podemos ver la interconexión entre FPGA-Pantalla, donde la parte de la Touch Screen se comunica con la FPGA a través del estándar I2C y la parte de escritura en la pantalla LCD a través de líneas discretas de datos.

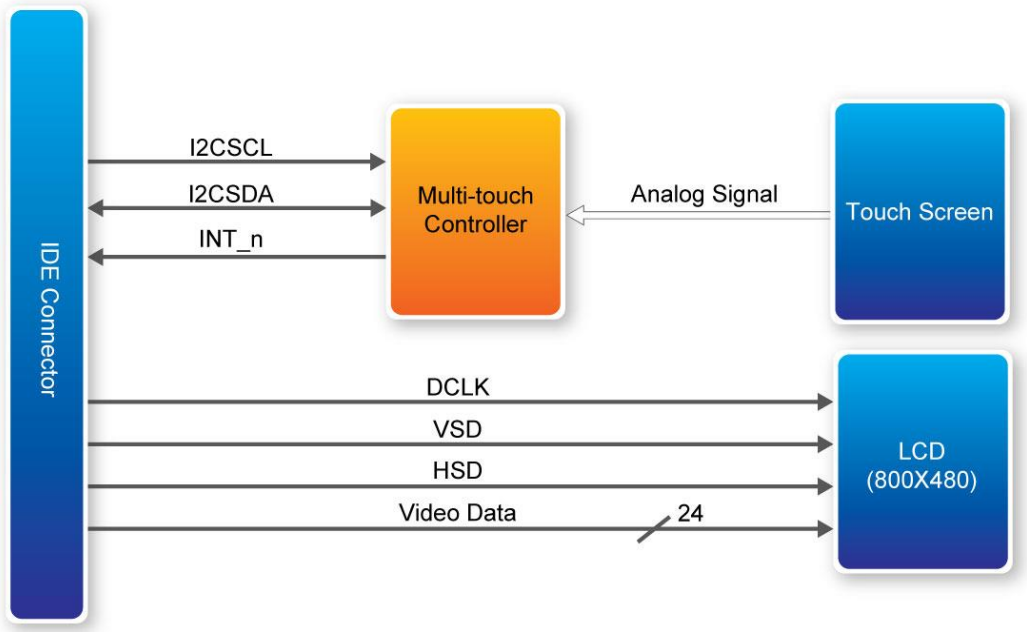


Figura 2-7 – Interconexión FPGA con la pantalla LCD

2.2. Herramientas de desarrollo Software, Firmware-Hardware

A la hora de desarrollar con las herramientas descritas en los siguientes apartados, se ha decidido trabajar en nuestro PC con Windows para el Quartus II, Ecliqse y EDS, y sobre una máquina virtual Linux Ubuntu 14.04 sobre Windows para el entorno QT.

2.2.1. Quartus II

Es una aplicación software desarrollada por Altera para el análisis y síntesis de diseño de sistemas digitales realizados en HDL (Hardware Description Language) e implementados sobre FPGAs. Este entorno integra todo lo necesario para el desarrollo hardware sobre la FPGA, en donde nos proporciona potentes herramientas como QSYS para la instanciación de bloques IP predefinidos e interconexión entre ellos, como pueden ser el microprocesador NIOS II de Altera, timers, controladores de memoria, controladores de ethernet, etc. Este entorno de trabajo también nos permite instanciar bloques IP predefinidos por Altera para sus FPGAs, como pueden ser FFTs (Fast Fourier Transform), mediante el plugin de MegaWizard Pluging Manager.

Además de todas estas herramientas integradas en el entorno de Quartus II, también se encuentran herramientas para simulación y análisis del diseño creado, como son el Modelsim-Altera.

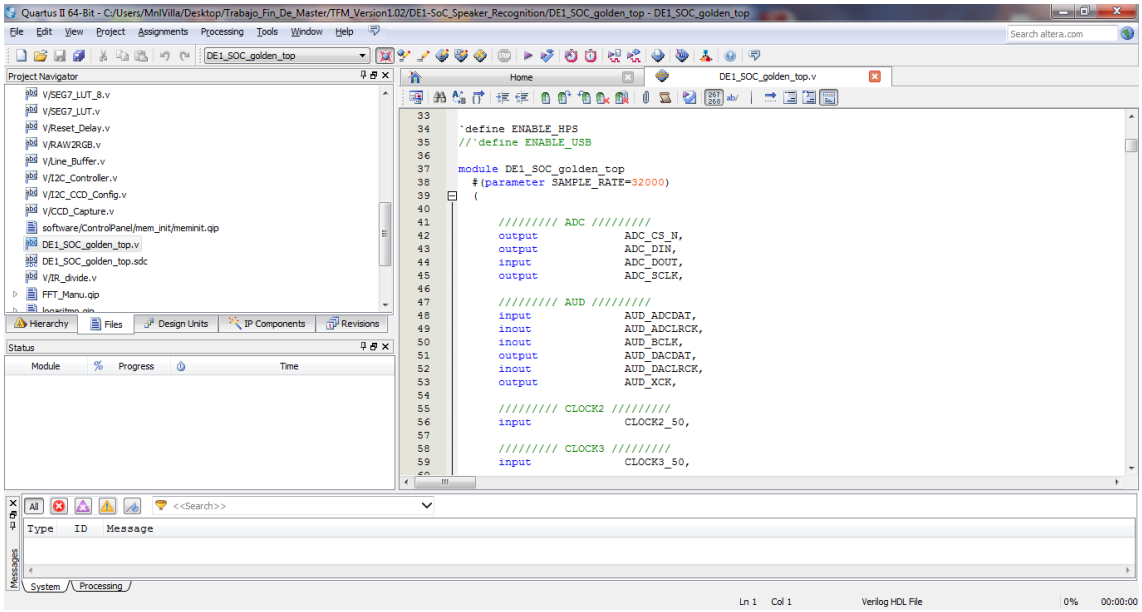


Figura 2-8 – Quartus II

En este trabajo se utilizará la versión Web Edition de Quartus II 13.1, la cual es gratuita y soporta FPGAs de bajo coste como la nuestra.

2.2.2. NIOS II, Software Build Tool for Eclipse

Se utilizará la aplicación Eclipse que viene integrada en el entorno Quartus II para crear y construir aplicaciones C/C++ en el procesador NIOS II. Para ello utilizará el archivo creado en QSYS .SOPCinfo que describe el hardware del sistema. Esta herramienta la integra el paquete de instalación de Quartus II.

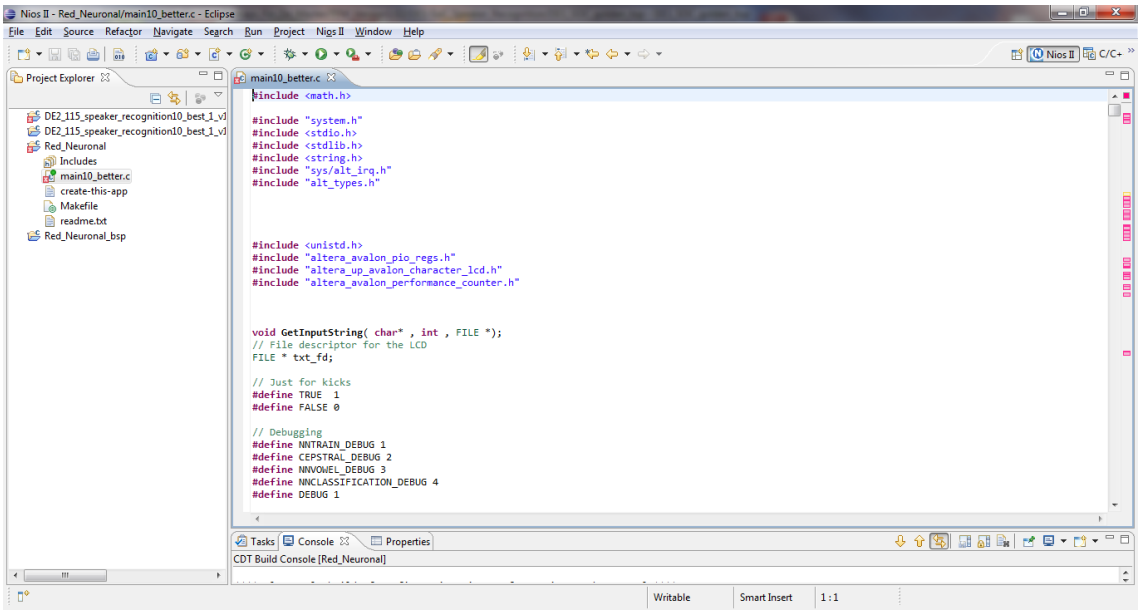


Figura 2-9 – Software Built Tool for Eclipse

2.2.3. QT Creator

Para el desarrollo de aplicaciones sobre Linux se ha decidido utilizar el entorno de desarrollo QT, ya que nos proporciona la parte de creación de aplicaciones GUI utilizando librerías de QT para Linux embebido. También proporciona un simulador/depurador en nativo, en donde podemos ver y depurar la interface de usuario en nuestro PC de trabajo sin necesidad de cargar el ejecutable en la tarjeta y depurarlo con un compilador cruzado. Sin embargo, también está la opción de configurar un compilador cruzado con el que podemos depurar sobre la tarjeta, necesario para depurar la interface con los periféricos y la FPGA.

Para poder compilar nuestra aplicación, así como el kernel de Linux se ha instalado en la máquina virtual de Linux Ubuntu el compilador “gcc-linaro” [18] con el que podremos

compilar nuestras aplicaciones para ARM, hacer una compilación cruzada y depurar nuestra aplicación sobre la tarjeta.

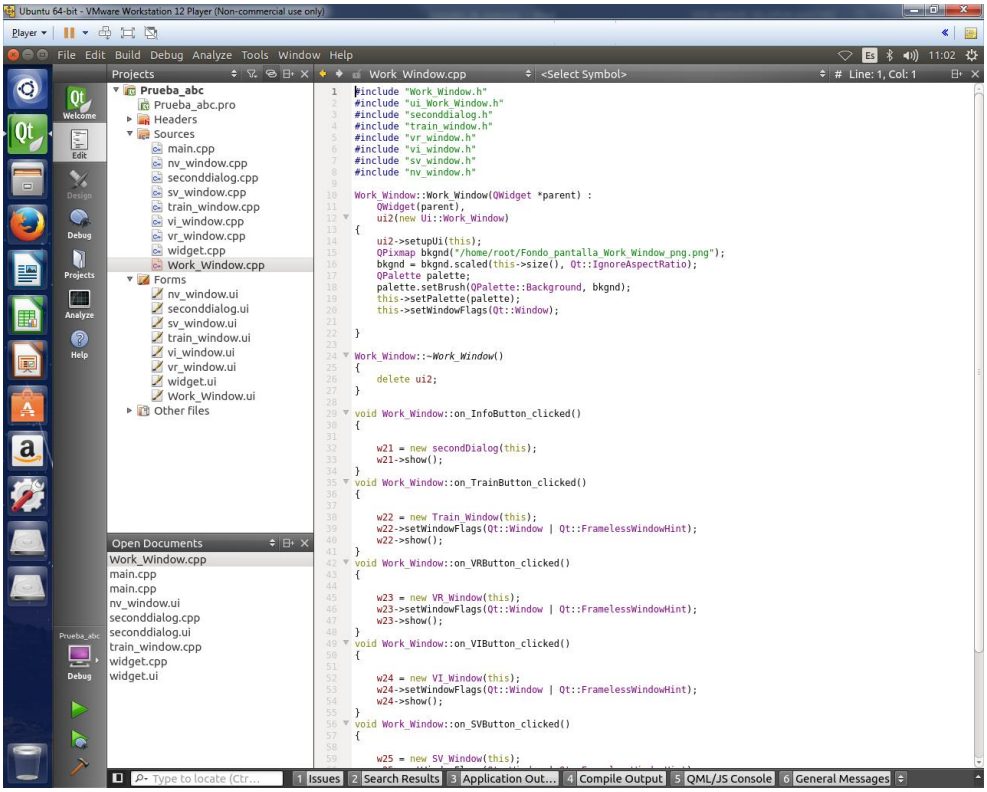


Figura 2-10 – QT Creator

Para este trabajo se ha utilizado la versión Qt Creator 3.0.1 basada en la Qt 5.2.1 (GCC 4.8.2, 64 bit) para Linux, para trabajar sobre sistema operativo Linux Ubuntu.

2.2.4. SoC Embedded Design Suite (EDS)

También se ha hecho uso del entorno de desarrollo que proporciona Altera para sus FPGAs SoC, en donde tenemos disponibles una serie de herramientas para el desarrollo sobre Linux embebido y con arquitectura ARM, como son:

- Depurador DS-5.
- Compilador ARM 5.03u2 para código embebido y bare-metal.
- Linaro GCC Toolchain 2013.03 para aplicaciones Linux y kernel.
- ARM Streamline Performance Analyzer para tarjetas de Linux y Android
- Eclipse IDE, editor de código fuente y gestor de proyectos

• Plataformas virtuales fijas (FVP) para procesadores Cortex-A8 y quad-core Cortex-A9

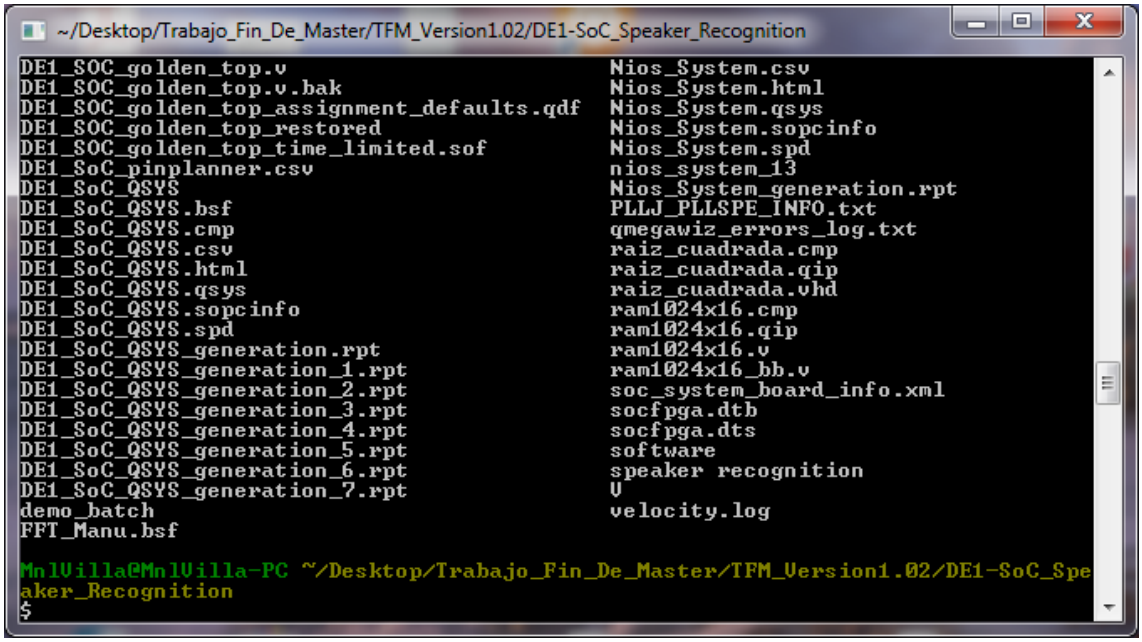


Figura 2-11 – Command Shell del EDS

En este trabajo se ha utilizado la versión 13.1 de EDS, en donde se ha utilizado este entorno de desarrollo para la creación del preloader de Linux y el Device Tree necesarios para el arranque y configuración del sistema Linux Embedded. En las siguientes secciones de este documento se abordará este tema con más profundidad.

3. Diseño y desarrollo Hardware

En esta sección se realiza una descripción del diseño hardware realizado sobre la FPGA, así como la configuración de los diferentes circuitos integrados utilizados. Este diseño se ha realizado sobre la placa de desarrollo utilizada en el apartado 2 de este documento, por lo que teniendo en cuenta los componentes utilizados y su interconexión, más el diseño hardware utilizado, se podría realizar el montaje en un nuevo PCB, aunque para la finalidad que se quería conseguir en este trabajo, con la placa de desarrollo comercial utilizada se ha conseguido realizar el prototipo perfectamente.

Profundizando en el diseño hardware de la FPGA, de un inicio se ha planteado el diseño que se observa en la siguiente imagen (Figura 3-1):

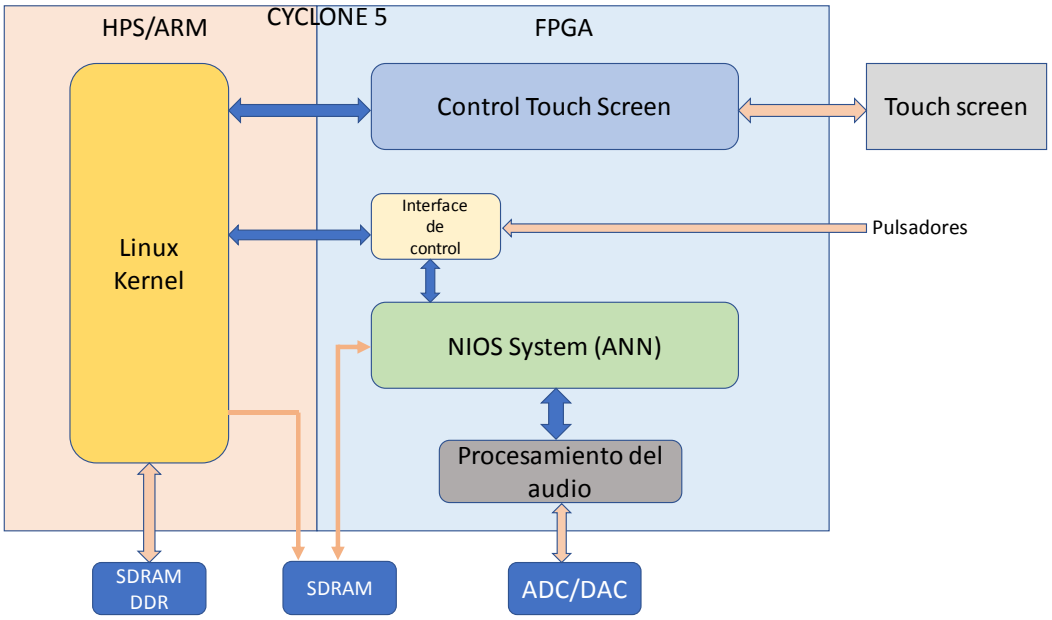


Figura 3-1 – Diagrama de bloques del diseño HW 1

En este diseño se diferencian dos partes en la FPGA, la parte HPS/ARM y la parte de la FPGA de lógica programable. En la primera es donde se configura todo lo necesario para que funcione una distribución de Linux, sobre la que se ejecutará la interface de usuario. Sobre el segundo se configuran todos los módulos necesarios para el funcionamiento, que serán los siguientes y se explicarán con más detalle en los siguientes subapartados:

- Módulo “Control Touch Screen”.
- Módulo “Interface de Control”.
- Módulo “Nios II System”.
- Módulo “Procesamiento del audio”.

Con este primer diseño se ha conseguido realizar perfectamente el objetivo principal de este trabajo, pero a la hora de añadir mejoras al diseño, se ha visto que se planteaban diferentes problemas, como puede ser el almacenamiento de los coeficientes de la red neuronal entrenada o el uso de diferentes algoritmos en la red neuronal, por lo que se ha optado por un nuevo diseño en el que se elimina el uso del sistema NIOS II implementado y se traspasa toda su función a la parte del kernel de Linux, donde se simplifica de forma importante el desarrollo de estas nuevas mejoras ya que trabajamos sobre un sistema operativo Linux.

La siguiente imagen (Figura 3-2) muestra un diagrama de bloques de este segundo diseño implementado:

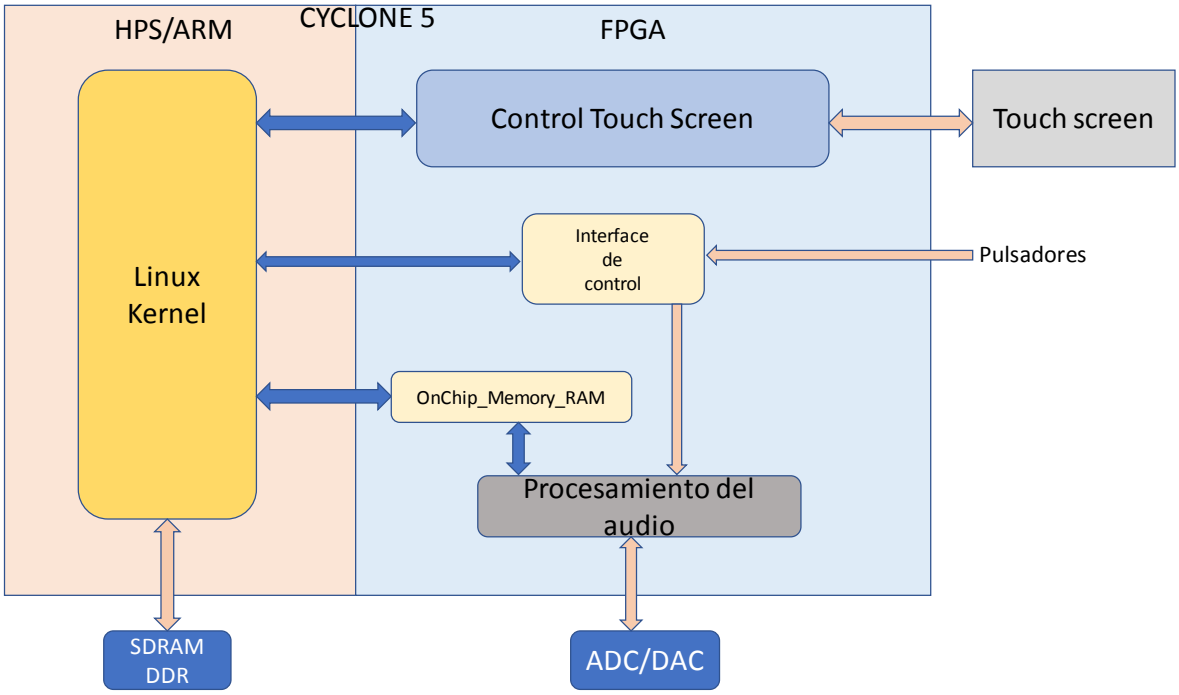


Figura 3-2 – Diagrama de bloques del diseño HW 2

En este segundo diseño se mantienen los módulos de “Control Touch Screen” y el de “Procesamiento del audio” que no cambian. Al eliminar el módulo del NIOS II system, se ha tenido que adaptar el módulo de “Interface de Control” y aparte reconectar el módulo “OnChip_Memory_RAM” (ya existente en el primer diseño donde estaba englobado en el módulo del NIOS II) para conectarlo al bus LH2F y que de esta forma se pueda mapear en memoria del kernel de Linux.

En este trabajo se entregan los dos diseños implementados, ya que el primero es el especificado de un inicio y el segundo se trata de un sistema mejorado, con el que se puede seguir trabajando en un futuro.

3.1. Configuración Qsys del HPS (Hard Proccessor system)

Como ya se ha explicado en la sección anterior, la FPGA utilizada para la realización de este trabajo contiene dos núcleos ARM Cortex A9 embebidos y que se denomina HPS (Hard Proccessor System). Para que la parte HPS de la FPGA se pueda comunicar a través de los diferentes puentes, salidas GPIO, etc., con la parte de lógica programable de esta, se debe instanciar y configurar en el sistema Qsys el bloque de "Hard Proccessor System", con el cual podremos comunicar el resto de partes implementadas en lógica programable de este diseño.

En la Figura 3-3 se puede ver la instanciación de este bloque en el sistema común, así como los elementos con los que se comunica y que son los siguientes:

1. Alt_vip_vfr_0: Este módulo se corresponde con el control y visualización de la pantalla multi touch y se conecta a la parte HPS por el puente HPS-FPGA Lightweight. En los siguientes apartados se explica el funcionamiento de este módulo.
2. Alt_vip_itc_0: Este módulo se corresponde con el control y visualización de la pantalla multi touch y se conecta a la parte HPS por el puente HPS-FPGA Lightweight. En los siguientes apartados se explica el funcionamiento de este módulo.
3. Terasic_multi_touch_0: Este módulo se corresponde con el control y visualización de la pantalla multi touch y se conecta a la parte HPS por el puente HPS-FPGA Lightweight. En los siguientes apartados se explica el funcionamiento de este módulo.
4. SDRAM_Nios: Este módulo se corresponde con un SDRAM Controller que se utilizará para cargar en una memoria externa el programa Nios II y se conectará al puente HPS-FPGA que abarca una sección de memoria de 1Gbyte, suficiente para poder mapear los 64MBytes de esta memoria externa. En los siguientes apartados se explica el uso de este módulo.
5. Control_Reg; Este módulo se corresponde con un PIO (Parallel input output) que se utilizará en la interface de control HPS-FPGA y se conecta al puente HPS-FPGA Lightweight. En los siguientes apartados se explica el uso de este módulo.
6. Indicator_Reg: Este módulo se corresponde con un PIO (Parallel input output) que se utilizará en la interface de control HPS-FPGA y se conecta al puente HPS-FPGA Lightweight. En los siguientes apartados se explica el uso de este módulo.
7. Indicator_Reg: Este módulo se corresponde con un PIO (Parallel input output) que se utilizará en la interface de control HPS-FPGA y se conecta al puente HPS-FPGA Lightweight. En los siguientes apartados se explica el uso de este módulo.

A parte de los módulos conectados, se han conectados las señales de reloj con las que va a funcionar cada puente. Al puente HPS to FPGA se ha conectado la señal de reloj de 100MHz con la que trabaja el sistema Nios II debido a que comparten el módulo SDRAM_Nios.

Al puente FPGA to HPS se conecta un reloj de 150MHz que se saca a través de un PLL a partir de la señal de reloj de 50 MHz conectada a la FPGA. Se configura con este valor de

frecuencia debido a que el módulo que se conecta a este puente, el "alt_vip_itc_0" trabaja a esta frecuencia

Por último, al puente HPS-FPGA Lightweight se conecta la señal de reloj de 50MHz, ya que los módulos conectados a este puente son de uso exclusivo del sistema HPS y no necesitan más frecuencia para su funcionamiento.

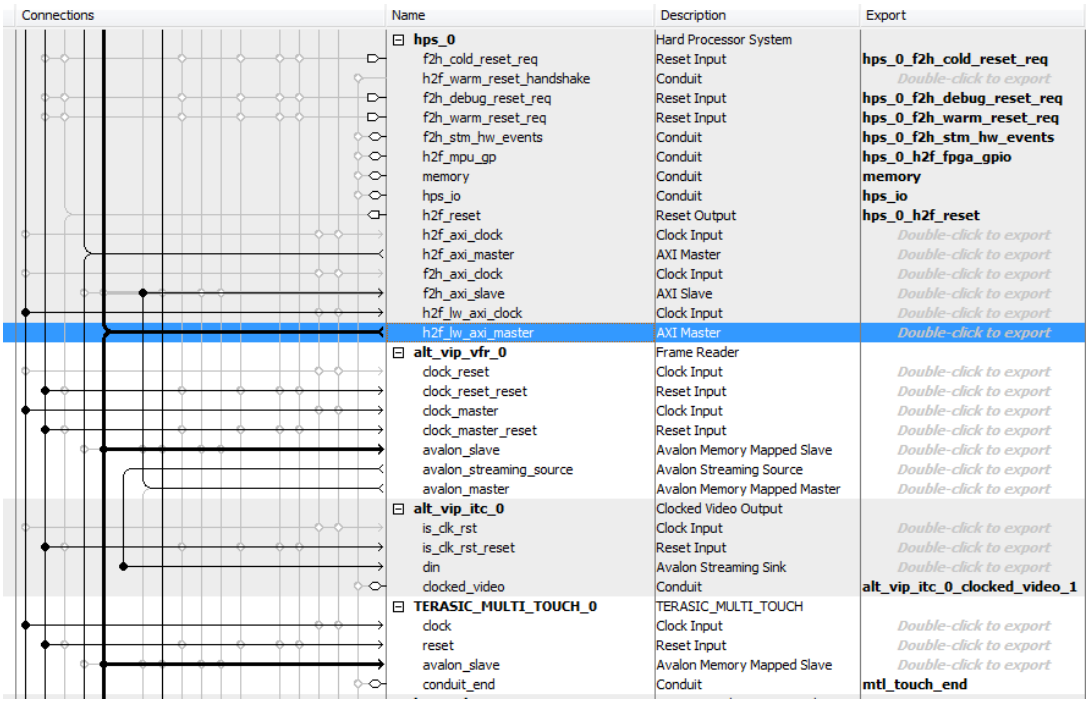


Figura 3-3 – Sistema Qsys del HPS

A parte de eso se configura para que saque hacia la parte de lógica de la FPGA las líneas de GPIO y se configura el ancho de los puentes "HPS to FPGA", "FPGA to HPS" y "HPS-FPGA Lightweight". En la siguiente imagen (Figura 3-4) podemos estas configuraciones en el sistema Qsys.

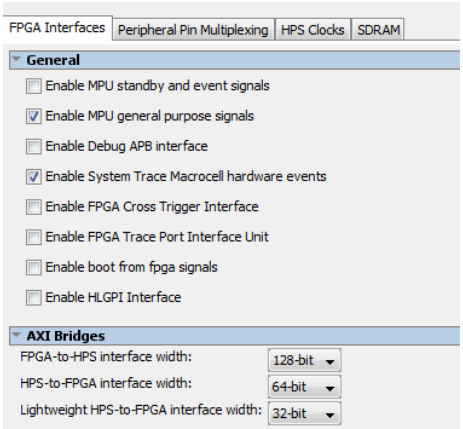


Figura 3-4 – Configuración general del bloque HPS

Otra de las configuraciones que se ha hecho es la de habilitar los diferentes reset e interrupciones desde la FPGA hacia la parte HPS y que podemos ver en la siguiente imagen (Figura 3-5).

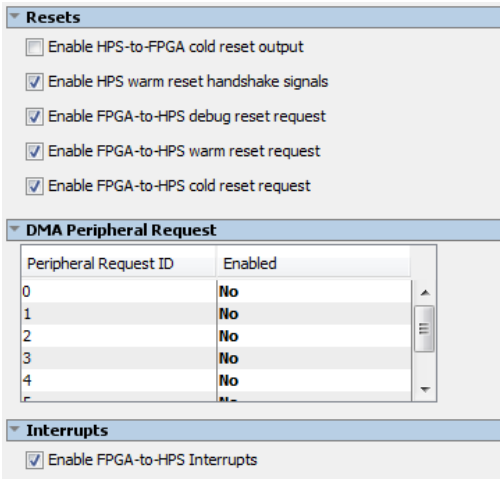


Figura 3-5 – Configuración de reset e interrupciones del HPS

Por último, otro de los ajustes que se ha realizado es el mapeo de todos los módulos que se conectan al sistema HPS, teniendo en cuenta las especificaciones de esta FPGA sobre el tamaño de las zonas de memoria para cada puente HPS-FPGA. Los offsets configurados en el mapa de memoria serán los que se utilicen a la hora de configurar el sistema Linux que corre sobre los procesadores ARM y acceder a las zonas respectivas de memoria de cada módulo. La siguiente imagen (Figura 3-6) muestra el mapa de memoria resultante:

	hps_0.h2f_axi_master	hps_0.h2f_lw_axi_master	...	hps_only_master.master	fpga_only_master.master
timer.s1		0x0000_0000 - 0x0000_001f			
onchip_memory2.s1	0x0004_0000 - 0x0006_70ff				0x0004_0000 - 0x0006_70ff
jtag_uart.avalon_jtag_slave		0x0000_0020 - 0x0000_0027			0x0000_0020 - 0x0000_0027
sysid.control_slave		0x0000_0028 - 0x0000_002f			0x0000_0028 - 0x0000_002f
hps_0.f2h_axi_slave			. .	0x0000_0000 - 0xffff_ffff	
alt_vip_vfr_0.avalon_slave		0x0000_0100 - 0x0000_017f			
TERASIC_MULTI_TOUCH_0.avalon_slave		0x0000_0200 - 0x0000_027f			
intr_capturer_0.avalon_slave_0					0x0003_0000 - 0x0003_0007
sdram_nios.s1	0x0400_0000 - 0x07ff_ffff				
Control_Reg.s1		0x0000_0040 - 0x0000_005f			
Indicator_Reg.s1		0x0000_0060 - 0x0000_006f			
Indicator_Reg2.s1		0x0000_0080 - 0x0000_008f			
cpu_0.jtag_debug_module					
jtag_uart_0.avalon_jtag_slave					
FFTStart.s1					
FFTDone.s1					
FFTAddr.s1					
FFTPower.s1					
FFTExp.s1					
AppSwitches.s1					
Train.s1					
VowelID.s1					
VowelLEDs.s1					
RedLED.s1					
GreenLED.s1					
onchip_memory2_0.s1					
onchip_memory2_0.s2					
performance_counter_0.control_slave					
master_start.s1					
coeficientes.s1					
nios_load.s1					
hps2fpga_On.s1	0x0800_0100 - 0x0800_010f				

Figura 3-6 – Mapa de memoria de los módulos conectados al HPS

3.2. Configuración Qsys del Nios II(diseño1)

Como ya se introdujo en la primera parte de esta sección, en este trabajo se han realizado 2 diseños. En el caso del primero se decidió usar un microcontrolador "soft-macro" NIOS II instanciado en la FGPA, el cual albergará la codificación de la red neuronal.

Para ello, utilizando la herramienta Qsys integrada en el Quartus II, se añade el elemento "Nios II Processor", el cual se configurará en su modo "fast" y usando la opción de división de hardware y multiplicadores embebidos, para de esta forma conseguir la máxima potencia en la implementación del mismo.

Como veremos con más detalle en los siguientes apartados de esta sección, este procesador se configura para que use como memoria de programa una SDRAM externa, por lo que se ha instanciado y configurado/conectado al mismo un elemento SDRAM Controller en el árbol de Qsys.

Otras de las configuraciones importantes a la hora de instanciar el Nios II es la elección de la cache de instrucciones de 4Kbytes y la cache de datos de 2Kbytes con 32Bytes por línea. También se configura para asignar de forma manual el registro de control "cpuid" y se selecciona un nivel de debug 1 para hacerlo más eficiente en el uso de memoria.

Por otro lado, al procesador Nios II, se le conectan a través del bus Avalon Memory Mapped Master de datos los siguientes elementos:

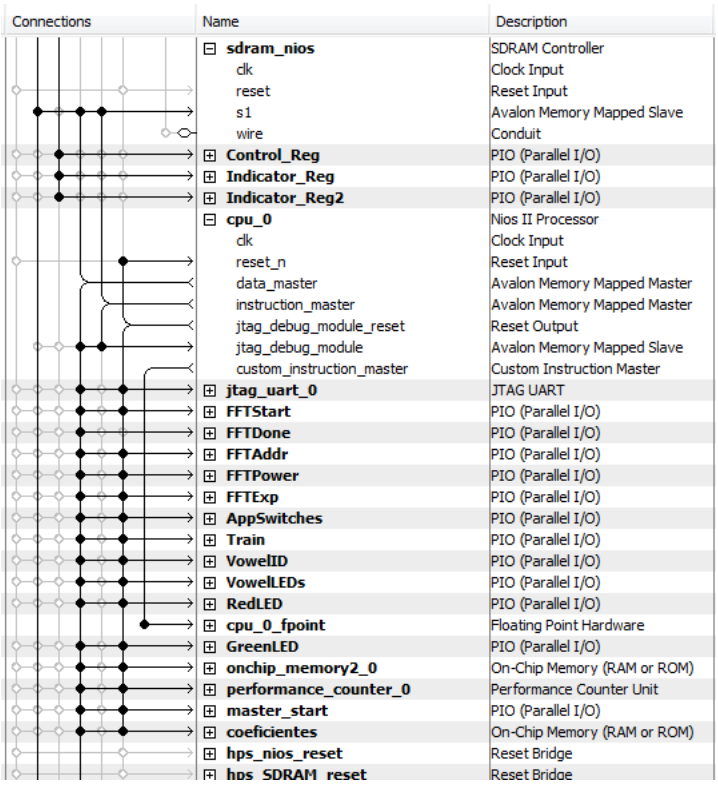


Figura 3-7 – Sistema Qsys del Nios II (para el diseño1)

En la siguiente imagen (Figura 3-8) podemos ver el mapa de memoria asociado al procesador Nios II de todos los elementos interconectados que utiliza:

	cpu_0.data_master	cpu_0.instruction_master
timer.s1		
onchip_memory2.s1		
jtag_uart.avalon_jtag_slave		
sysid.control_slave		
hps_0.f2h_axi_slave		
alt_vip_vfr_0.avalon_slave		
TERASIC_MULTI_TOUCH_0.avalon_slave		
intr_capturer_0.avalon_slave_0		
sdram_nios.s1	0x0400_0000 - 0x07ff_ffff	0x0400_0000 - 0x07ff_ffff
Control_Reg.s1		
Indicator_Reg.s1		
Indicator_Reg2.s1		
cpu_0.jtag_debug_module	0x0000_0800 - 0x0000_0fff	0x0000_0800 - 0x0000_0fff
jtag_uart_0.avalon_jtag_slave	0x0000_10b0 - 0x0000_10b7	
FFTStart.s1	0x0000_1000 - 0x0000_100f	
FFTDone.s1	0x0000_1010 - 0x0000_101f	
FFTAddr.s1	0x0000_1020 - 0x0000_102f	
FFTPower.s1	0x0000_1030 - 0x0000_103f	
FFTExp.s1	0x0000_1040 - 0x0000_104f	
AppSwitches.s1	0x0000_1050 - 0x0000_105f	
Train.s1	0x0000_1060 - 0x0000_106f	
VowelID.s1	0x0000_1070 - 0x0000_107f	
VowelLEDs.s1	0x0000_1080 - 0x0000_108f	
RedLED.s1	0x0000_1090 - 0x0000_109f	
GreenLED.s1	0x0000_10a0 - 0x0000_10af	
onchip_memory2_0.s1	0x0000_0000 - 0x0000_03ff	
onchip_memory2_0.s2		
performance_counter_0.control_slave	0x0000_0400 - 0x0000_047f	
master_start.s1	0x0000_2000 - 0x0000_200f	
coeficientes.s1	0x0000_8000 - 0x0000_8fff	
nios_load.s1	0x0000_1100 - 0x0000_110f	
hps2fpga_On.s1		

Figura 3-8 – Mapa de memoria del sistema Nios II

3.3. Carga del NIOS II desde sistema Linux(diseño1)

A la hora de hacer este trabajo, uno de los objetivos que se quería conseguir es que el sistema se inicializase automáticamente sin tener que cargar/lanzar desde un ordenador el programa, tanto en la parte de Linux como en la parte de Nios II. En esta última se ha presentado el mayor problema.

Para arrancar de forma autónoma la parte Nios II se ha decidido hacerlo desde la parte de Linux, en donde para ello se utiliza la memoria SDRAM que contiene la placa de desarrollo DE1-SoC como memoria de programa del Nios II. Esta memoria se configura en el sistema a través del elemento del Qsys SDRAM Controller que se configura con 16 bits de datos repartidos en 13 filas y 10 columnas.

Una vez configurado el SDRAM Controller se conecta a la parte HPS a través del puente AXI Master HPS to FPGA y a la parte Nios II por el bus data master e instruction master.

En la siguiente imagen (Figura 3-9) podemos ver la conexión en el sistema Qsys:

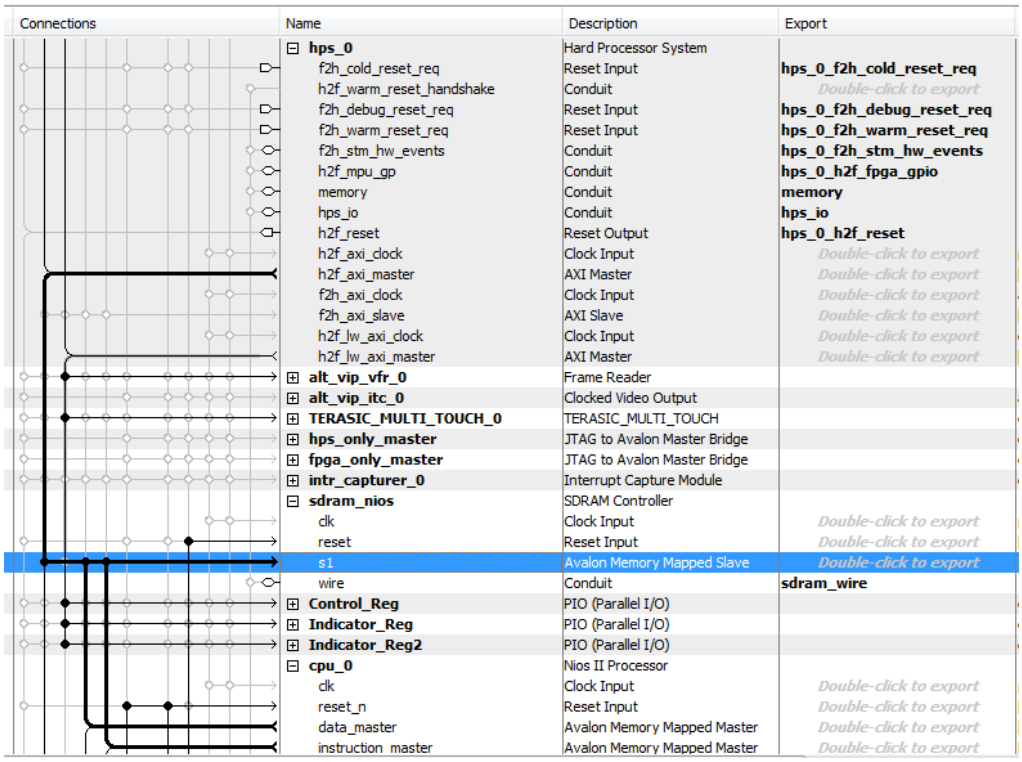


Figura 3-9 – Conexión con la SDRAM Controller (diseño 1)

Una vez que tenemos que tenemos todo configurado, el procedimiento a seguir es primero poner en reset todo el sistema Nios II y la SDRAM. Para ello se configuran en el sistema Qsys dos puentes de reset que estarán conectados por lógica programable al puerto GPIO del HPS, exactamente a los bits 0 y 1 de salida.

- Hps_nios_reset (Reset Bridge).
- Hps_SDRAM_reset (Reset Bridge).

Después se levanta la línea de reset de la SDRAM y se hace la carga del programa del Nios II en la misma desde el sistema Linux. Una vez que la carga finaliza se levanta la línea de reset del sistema Nios II y de esta forma se pone arranca.

A la hora de hacer esta operación hay que tener varias cosas en cuenta. Una de ellas es que a la hora de conectar el reloj de la SDRAM tenemos que provocarle un desfase negativo de 90 grados para que funcione. Para ello se utiliza un PLL al cual le ponemos en la entrada el reloj de 100MHz que utiliza el Nios II y el puente del HPS y se configura para tener una salida desfasada de la misma frecuencia que conectaremos a la SDRAM.

Otra puntualización es a la hora de generar el binario del programa Nios II, el cual tenemos que generar con las secciones de programa sin que contenga el total de los 64Mbytes de la SDRAM, para que de esta forma sea fácilmente cargable desde Linux. Para este propósito se ha utilizado el Command Shell que proporciona la herramienta Eclipse de Nios. Con la siguiente línea de comandos generamos el binario únicamente con las secciones que queremos:

“nios2-elf-objcopy -j .bss -j .entry -j .exceptions -j .heap -j .rodata -j .rwdata -j .stack -j .text -O binary Red_neuronal.elf Red_neuronal.bin”

3.4. Interface de control HPS – FPGA

Para el primer diseño desarrollado en este trabajo se ha implementado una interface de control en la FGPA para la comunicación entre la parte Linux del HPS y la parte implementada en el Nios II, con la que las ordenes que el usuario ejecuta en la interface gráfica de Linux se traspasan al sistema Nios II para su control y a la inversa para la monitorización de los resultados en el IHM.

Esta interface de control y monitorización se hace a nivel de registros, en donde se mapean de forma independiente en la parte HPS y en la parte Nios II diversos registros, y luego se interconectan los diferentes bits de los registros HPS con los bits de los registros del Nios II.

En la Figura 3-10 podemos ver los registros creados en la parte HPS, en donde se utiliza el elemento PIO (Parallel I/O) de Qsys para conectar a la parte del puente HPS-FPGA Lightweight y mapearlo en su espacio de memoria con un determinado offset configurado como en la figura. De este elemento se exporta a través de un "conduit" las líneas a la parte de lógica programable, con lo que de esta forma podemos conectar estos registros con la parte Nios II.

- **Control_Reg** (registro de 18 bits de salida).
- **Indicator_Reg** (registro de 9 bits de entrada).
- **Indicator_Reg2** (registro de 18 bits de entrada).

⊟ **Control_Reg**	PIO (Parallel I/O)					
clk	Clock Input	*Double-click to export*	**clk_50**			
reset	Reset Input	*Double-click to export*	[clk]			
s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0000_0040	0x0000_005f	
external_connection	Conduit	**control_reg**				
⊟ **Indicator_Reg**	PIO (Parallel I/O)					
clk	Clock Input	*Double-click to export*	**clk_50**			
reset	Reset Input	*Double-click to export*	[clk]			
s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0000_0060	0x0000_006f	
external_connection	Conduit	**indicator_reg**				
⊟ **Indicator_Reg2**	PIO (Parallel I/O)					
clk	Clock Input	*Double-click to export*	**clk_50**			
reset	Reset Input	*Double-click to export*	[clk]			
s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0000_0080	0x0000_008f	
external_connection	Conduit	**indicator_reg2**				

Figura 3-10 – Interface de control de la parte HPS

De la misma forma que en la parte HPS, en la parte Nios II se instancian los siguientes registros de control e indicación:

- **AppSwitches** (registro de 2 bits de entrada). Los bits de este registro sirven para configurar el tipo de reconocimiento o identificación que se desea realizar con la red neuronal. Estos bits se conectan con los bits 0 y 1 del Control_Reg del HPS.
- **Train** (registro de 1 bit de entrada). Este bit sirve para configurar la red neuronal en modo entrenamiento o en modo normal y se conecta con el bit 2 del Control_Reg del HPS.

- **VowelID** (registro de 15 bits de entrada). Con estos bits configuramos cuando estamos en modo entrenamiento que fonema estamos entrenando. Estos se conectar a los bits 3 al 17 del Control_Reg del HPS.
- **VowelLEDs** (registro de 15 bits de salida). Con estos bits la red neuronal nos devuelve el resultado de la identificación resultante. Estos se conectan con los bits del 3 al 17 del Indicator_Reg2 del HPS.
- **RedLED** (registro de 1 bit de salida). Con este bit la red neuronal refleja el éxito de la identificación/reconocimiento, donde si está a 1 el resultado ha sido erróneo. Este bit se conectará al bit 0 del indicator_Reg2 del HPS.
- **GreenLED** (registro de 1 bit de salida). Con este bit la red neuronal refleja el éxito de la identificación/reconocimiento, donde si está a 1 el resultado ha sido correcto. Este bit se conectará al bit 7 del indicator_Reg del HPS.

La Figura 3-11 muestra los registros instanciados en el sistema Qsys:

AppSwitches	PIO (Parallel I/O)				
clk	Clock Input	Double-click to export	**pll_sys_...**		
reset	Reset Input	Double-click to export	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_1050	0x0000_105f
external_connection	Conduit	**appswitches_external_connec...**			
Train	PIO (Parallel I/O)				
clk	Clock Input	Double-click to export	**pll_sys_...**		
reset	Reset Input	Double-click to export	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_1060	0x0000_106f
external_connection	Conduit	**train_external_connection**			
VowelID	PIO (Parallel I/O)				
clk	Clock Input	Double-click to export	**pll_sys_...**		
reset	Reset Input	Double-click to export	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_1070	0x0000_107f
external_connection	Conduit	**vowelid_external_connection**			
VowelLEDs	PIO (Parallel I/O)				
clk	Clock Input	Double-click to export	**pll_sys_...**		
reset	Reset Input	Double-click to export	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_1080	0x0000_108f
external_connection	Conduit	**vowelleds_external_connection**			
RedLED	PIO (Parallel I/O)				
clk	Clock Input	Double-click to export	**pll_sys_...**		
reset	Reset Input	Double-click to export	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_1090	0x0000_109f
external_connection	Conduit	**redled_external_connection**			

Figura 3-11 – Interface de control de la parte Nios II

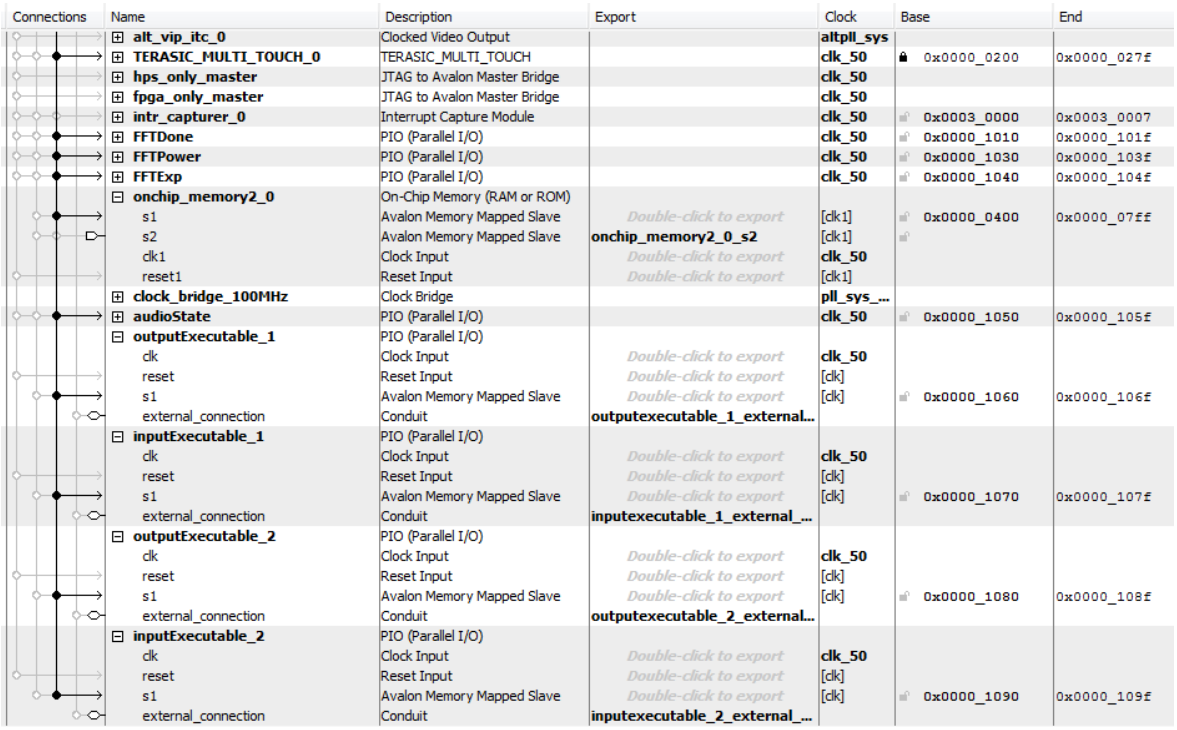
En el segundo diseño realizado en este trabajo se ha tenido que cambiar el módulo de interface de control implementado en la FPGA para adaptarlo a las nuevas necesidades.

Como en este nuevo diseño ya no se utiliza el procesador Nios II, se han eliminado todos los registros de comunicación entre la parte Linux y la parte Nios, y se ha dejado únicamente parte de pulsadores de control de inicio y fin de grabación del audio. Luego, como ya se ha comentado en el apartado anterior, se ha reconectado la parte de la FFT al sistema Linux.

Por otro lado, se han añadido nuevos registros mapeados y conectados en la zona del puente HPS-FPGA Lightweight que son los mostrados en la siguiente imagen (Figura 3-12):

- outputExecutable_1 (registros de 32bits).
- inputExecutable_1 (registros de 32bits).
- outputExecutable_2 (registros de 32bits).

- inputExecutable_2 (registros de 32bits).



Connections	Name	Description	Export	Clock	Base	End
	alt_vip_itc_0	Clocked Video Output		**altpll_sys**		
→	**TERASIC_MULTI_TOUCH_0**	TERASIC_MULTI_TOUCH		**clk_50**	0x0000_0200	0x0000_027f
	hps_only_master	JTAG to Avalon Master Bridge		**clk_50**		
	fpga_only_master	JTAG to Avalon Master Bridge		**clk_50**		
	intr_capturer_0	Interrupt Capture Module		**clk_50**	0x0003_0000	0x0003_0007
→	**FFTDone**	PIO (Parallel I/O)		**clk_50**	0x0000_1010	0x0000_101f
→	**FFTPower**	PIO (Parallel I/O)		**clk_50**	0x0000_1030	0x0000_103f
→	**FFTExp**	PIO (Parallel I/O)		**clk_50**	0x0000_1040	0x0000_104f
	onchip_memory2_0	On-Chip Memory (RAM or ROM)				
→	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]	0x0000_0400	0x0000_07ff
▷	s2	Avalon Memory Mapped Slave	**onchip_memory2_0_s2**	[clk1]		
	clk1	Clock Input	*Double-click to export*	**clk_50**		
	reset1	Reset Input	*Double-click to export*	[clk1]		
	clock_bridge_100MHz	Clock Bridge		**pll_sys_...**		
→	**audioState**	PIO (Parallel I/O)		**clk_50**	0x0000_1050	0x0000_105f
	outputExecutable_1	PIO (Parallel I/O)				
	clk	Clock Input	*Double-click to export*	**clk_50**		
	reset	Reset Input	*Double-click to export*	[clk]		
→	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0000_1060	0x0000_106f
○	external_connection	Conduit	**outputexecutable_1_external...**			
	inputExecutable_1	PIO (Parallel I/O)				
	clk	Clock Input	*Double-click to export*	**clk_50**		
	reset	Reset Input	*Double-click to export*	[clk]		
→	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0000_1070	0x0000_107f
○	external_connection	Conduit	**inputexecutable_1_external_...**			
	outputExecutable_2	PIO (Parallel I/O)				
	clk	Clock Input	*Double-click to export*	**clk_50**		
	reset	Reset Input	*Double-click to export*	[clk]		
→	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0000_1080	0x0000_108f
○	external_connection	Conduit	**outputexecutable_2_external...**			
	inputExecutable_2	PIO (Parallel I/O)				
	clk	Clock Input	*Double-click to export*	**clk_50**		
	reset	Reset Input	*Double-click to export*	[clk]		
→	s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0000_1090	0x0000_109f
○	external_connection	Conduit	**inputexecutable_2_external_...**			

Figura 3-12 – Interface de control para el diseño 2

Estos registros se usan para el traspaso de información de forma segura entre los dos programas/ejecutables que corren en Linux y que se explicarán en la siguiente sección (programa de usuario y ejecutable de la red neuronal).

3.5. Módulo de adquisición del audio

El módulo de adquisición del audio es el encargado de tomar todas las muestras del audio y guardarlas en una memoria mediante el control del usuario, lo que quiere decir que es el usuario quien dice cuando empezar y finalizar la adquisición. Para ello este módulo se encarga de configurar por bus serie I2C el circuito integrado Audio CODEC que se encarga de digitalizar la señal analógica.

La configuración utilizada es la siguiente:

- Lline in y Rline in: Configurados con alta ganancia.
- Lphone out y Rphone out: Configurados con ganancia unitaria.
- Se configura la línea ADC, el DAC a ON, no hacer bypass ni sidetone.
- Frecuencia de muestreo de 32KHz.
- Se configura el Power ON.
- Se configura con los bits MSB primero, justificación a la izquierda y modo esclavo en la comunicación.
- Se configura con modo normal de funcionamiento.
- Se activa el módulo.

Una vez que el Audio CODEC está configurado, este está continuamente tomando muestras a la frecuencia de muestreo configurada y enviándolas al módulo "Audio DAC ADC" implementado en la FPGA por la línea de datos serie "AUD_ADCDAT". Aunque este módulo está diseñado para usar varios canales de entrada (haciendo uso del secuenciador del ADC), el canal derecho y el canal izquierdo del micrófono, en este diseño sólo se utiliza el canal izquierdo del micrófono, que nos sirve perfectamente para el propósito final.

Las muestras del canal izquierdo del micrófono, con una resolución de 16 bits, se encaminan hacia el módulo de "Audio_RAM" donde serán almacenadas.

La forma de almacenar las muestras provenientes del módulo "Audio DAC ADC" es mediante una ventana que marca el usuario a través de los pulsadores.

Una vez que las muestras van entrando en este módulo, donde llegarán con una frecuencia de muestreo de 32KHz, se baja la frecuencia de muestreo a 16KHz, que nos sigue llegando para no tener aliasing, ya que la voz humana se encuentra entorno a los 8KHz y según la ley de Nyquist que nos dice que la frecuencia de muestreo debe de ser según la siguiente relación:

$$Fs \geq 2 * F$$

Las muestras que van entrando se procesan con un filtro preénfasis de coeficiente 0.97 y se hace un downsample.

La memoria de almacenamiento esta implementada en lógica programable de la FPGA con un tamaño de 1024 posiciones por 16 bits por dato. En esta memoria se guardarán un total

de 600 muestras de entrada, momento en el que el módulo indicará que la carga está completa y no guardará más muestras.

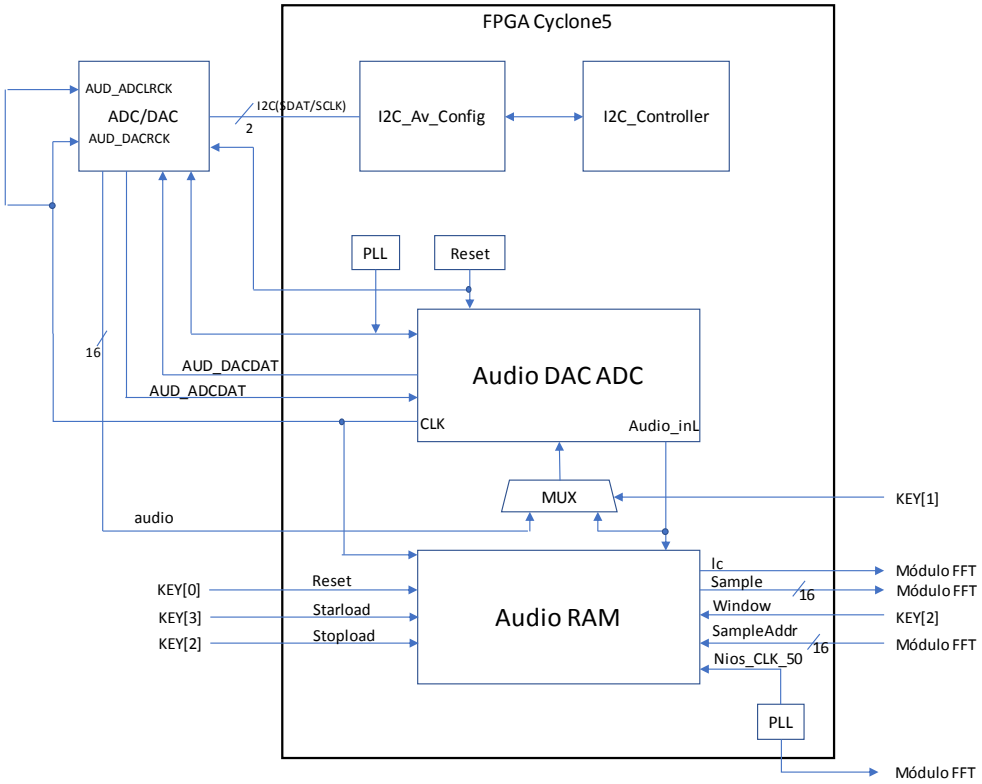


Figura 3-13 – Diagrama de bloques de la adquisición del audio

3.6. Módulo de procesamiento de señal.

Una vez que las muestras de audio están guardadas en el módulo "Audio RAM", momento en el que este módulo nos indicará mediante una línea el estado de la carga como completo, es el momento en el que empieza a trabajar el módulo de procesamiento del audio para finalmente conseguir tener la representación espectral de la ventana de muestras mediante la FFT y posterior filtrado de las mismas.

En la siguiente imagen (Figura 3-14) podemos ver el esquema de bloques implementado en la FPGA para realizar el procesamiento de señal, donde primero se cogen de la AudioRAM y finalmente se guarda el resultado en una memoria On_Chip de doble puerto, para que sea accesible por parte del sistema Nios II en el caso del primer diseño y por parte del sistema Linux en el caso del segundo diseño. En la imagen observamos tres salidas hacia el sistema Nios II, en el caso del segundo diseño estas imágenes se enviarían al sistema Linux.

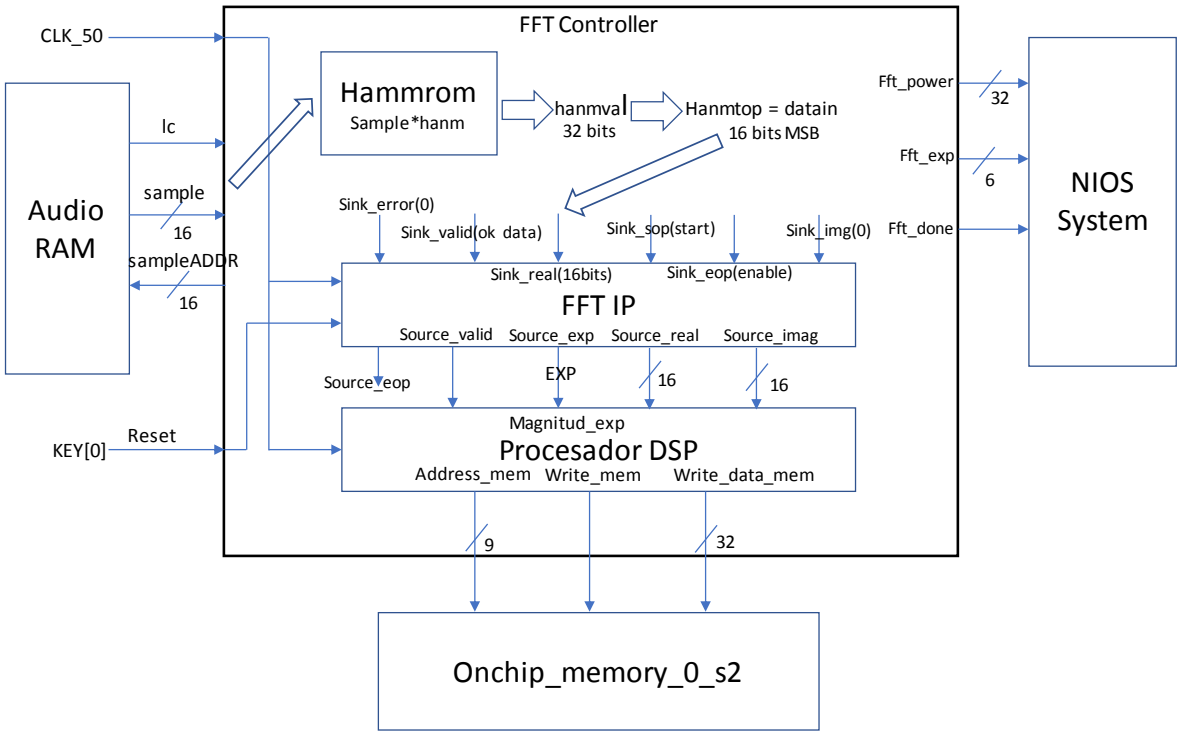


Figura 3-14 – Diagrama de bloques del procesamiento de señal

La forma de hacer el procesado de estas muestras es ir leyendo las muestras almacenadas una por una, donde primero se pasarán por un enventanado de tipo Hamming, que consiste en hacer la convolución de las muestras con una ventana de tipo Hamming que previamente se calcula con la herramienta Matlab. Al hacer la convolución de las muestras, como estas son de 16 bits y se multiplican por los valores de la ventana que también serán de 16bits, el resultado va a ser muestras de 32 bits.

Una vez que tenemos la convolución de las muestras con la ventana hamming, estas se envían al módulo FFT, que tendrá una entrada real de 16 bits, por lo que se cogen los 16 bits MSB.

El módulo FFT se corresponde con uno de los módulos predefinidos por altera en la herramienta Quartus y que podemos instanciar mediante el MegaWizard Plug-in Manager. En la siguiente imagen (Figura 3-15) podemos ver la configuración del módulo FFT instanciado, donde se configura con 512 puntos y una precisión de datos de 16 bits.

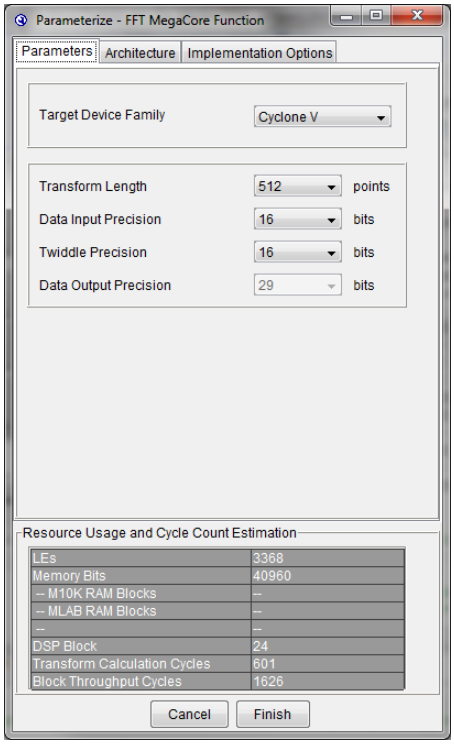


Figura 3-15 – Configuración del bloque FFT

Una vez que las muestras van saliendo del módulo FFT que nos da la componente espectral de las mismas, estas se envían al módulo procesador DSP, el cual se encarga de hacer el filtrado y de esta forma mover el espectro dentro de la escala MEL, para posteriormente realizar la transformada del coseno del espectro MEL y así obtener los 16 coeficientes utilizados en la red neuronal como entrada para su excitación.

Este banco de filtros se realiza utilizando los módulos predefinidos de Altera, como son el módulo Logaritmo (ALTFP_LOG), el módulo de multiplicación (ALTFP_MULT), el módulo raíz cuadrada (ALTFP_SQRT) y el módulo de suma/resta (ALTFP_ADD_SUB).

En las siguientes imágenes se puede ver la configuración elegida a la hora de instanciar los módulos del logaritmo, multiplicador, raíz cuadrada y restador:

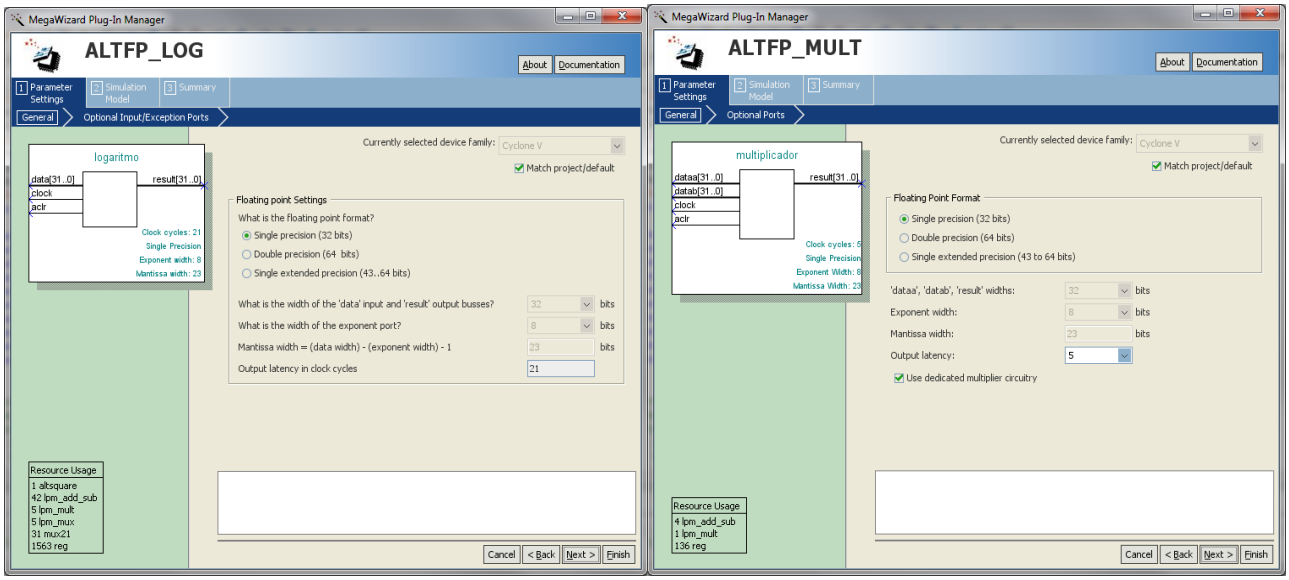


Figura 3-16 – Configuración de los bloques logaritmo y multiplicador

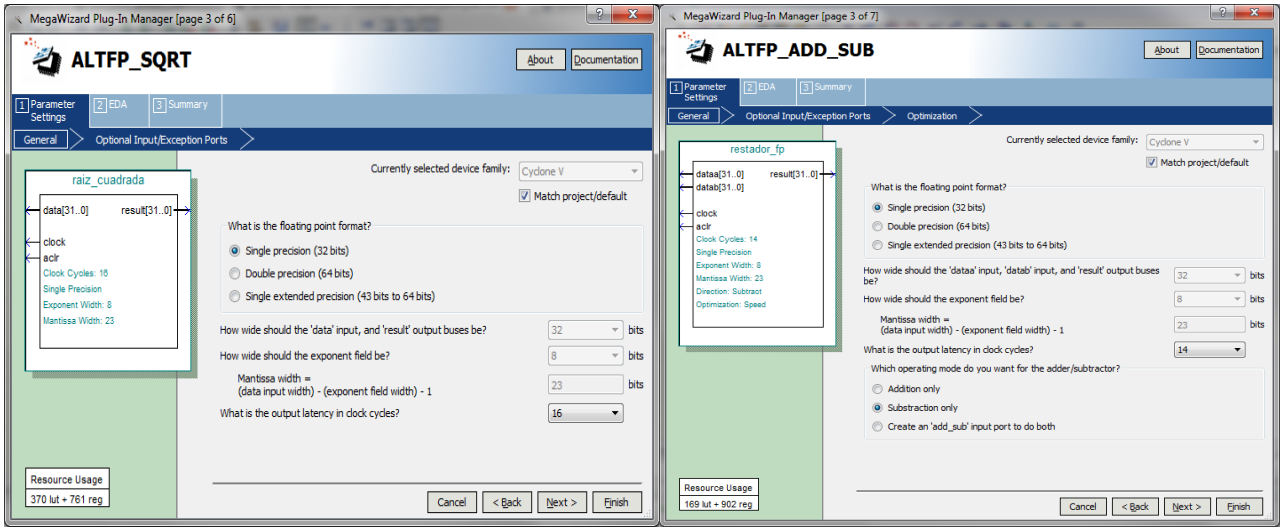


Figura 3-17 – Configuración de los bloques raíz cuadrada y restador

La configuración que se sigue en este banco de filtros para obtener la escala MEL sigue la siguiente formula:

$$m = 1127{,}01048 * log_e(1 + \frac{f}{700})$$

En la figura 3-18 se puede ver la representación de la escala MEL con respecto a la frecuencia.

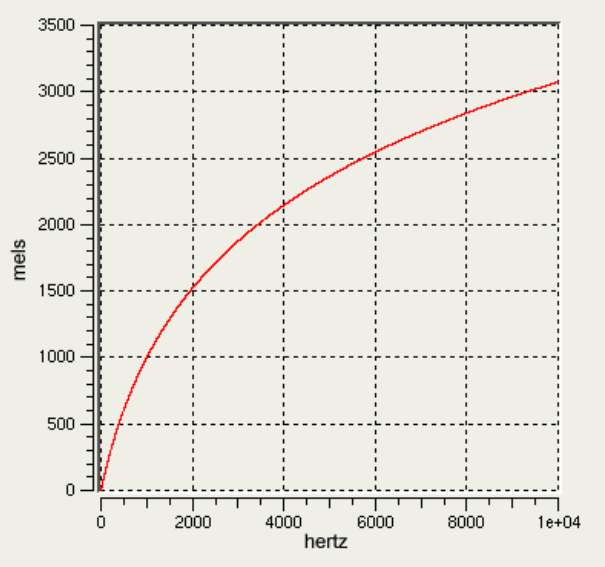


Figura 3-18 – Escala MEL

Para el caso de la transformada del coseno (DCT), se realiza la siguiente operación, con la que obtenemos únicamente la representación con números reales:

$$f_j = \left(\frac{1}{2}\right)(x_0 + (-1)^j x_{n-1}) + \sum_{k=1}^{n-2} x_k \cos[\frac{\pi}{n-1} k_j]$$

Estos coeficientes obtenidos se denominan MFCC (Mel Frequency Cepstral Coefficients) y comúnmente sirven para la representación del habla basados en la percepción auditiva humana. Estos surgen de la necesidad, en el área del reconocimiento de audio automático, de extraer características de las componentes de una señal de audio que sean adecuadas para la identificación de contenido relevante, así como obviar todas aquellas que posean información poco valiosa como el ruido de fondo, emociones, volumen, tono, etc. y que no aportan nada al proceso de reconocimiento, al contrario, lo empobrecen.

3.7. Módulo de control de la Multi Touch LCD

La siguiente imagen (Figura 31) muestra la arquitectura del sistema de DE1-SoC-MTL. Por el lado de FPGA, la suite de Altera VIP (vídeo y procesamiento de imágenes) se utiliza para diseñar el controlador de visualizador de LCD y el controlador de pantalla táctil Terasic se utiliza para controlar la pantalla táctil. En este kernel de Linux, un controlador de framebuffer está diseñado para pantalla LCD y el controlador de pantalla táctil está diseñado para panel de pantalla táctil.

Como ya se ha comentado en apartados anteriores, en el sistema Qsys tenemos instanciados los módulos (cores IP) "alt_vip_vfr", "alt_vip_itc" y "Terasic_Multi_Touch" que se utilizan para el procesado de video e imagen, además del control/indicación por I2C de los puntos presionados en la pantalla. El primero se corresponde con el core IP Frame Reader y que se usa para leer de un stream de video desde un frame de video que está almacenado en un buffer de memoria. El segundo se corresponde con un core IP Clocked Video Output y que se usa para el procesamiento de entrada/salida de video e imagen. El último se corresponde con un core IP diseña para el control e indicación a través de I2C de la pantalla LCD, como pueden ser las coordenadas de las pulsaciones en la pantalla.

En la referencia [2] podemos ver la documentación asociada a estos módulos.

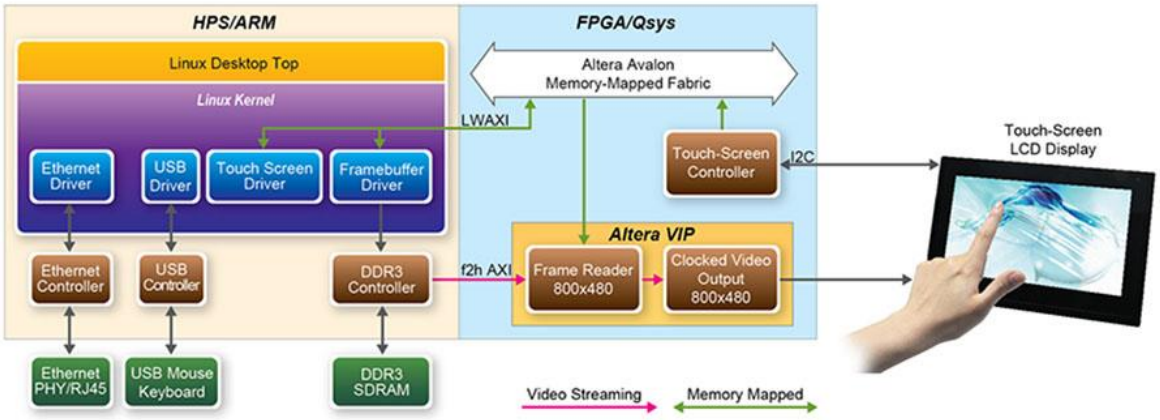


Figura 3-19 – Diagrama de bloques del control de la pantalla LCD

4. Diseño y desarrollo Software

En este apartado se va abordar el diseño software sobre el sistema operativo Linux, así como toda la configuración necesaria para tener el S.O. corriendo sobre el HPS, y también la aplicación software que va a correr sobre el microprocesador Nios II y sus configuraciones.

4.1. Diseño Software sobre Linux

En esta parte vamos a empezar con un pequeño resumen de que es Linux y como se compone toda la cadena necesaria para que el sistema operativo arranque y se ponga a funcionar, para luego poder hacer correr sobre él, nuestra aplicación de usuario.

Linux es un núcleo de libre distribución y mayormente libre semejante al núcleo de Unix.4. Linux es uno de los principales ejemplos de software libre y de código abierto que está licenciado bajo la GPL v2.

Linux no es un sistema operativo en sí mismo. Es un núcleo compatible con el de Unix. Inicialmente se publicó como un núcleo semejante a Minix4 y a día de hoy sigue incluyendo el mismo tipo de software: controladores, planificadores, gestores de memoria virtual y más funcionalidad habitual en un núcleo de sistema operativo. Linux se ejecuta en el nivel con más privilegios del modo protegido, el que corresponde al núcleo del sistema operativo. La parte de un sistema operativo que se ejecuta sin privilegios o en espacio de usuario es la biblioteca del lenguaje C, que provee el entorno de tiempo de ejecución, y una serie de programas o herramientas que permiten la administración y uso del núcleo y proveer servicios al resto de programas en espacio de usuario, formando junto con el núcleo el sistema operativo.

Las distribuciones son una recopilación de software que incluyen el núcleo Linux y el resto de programas necesarios para completar un sistema operativo.

Para que el sistema Linux arranque primero necesitamos un proceso de “boot” que empezará cuando la CPU en la MPU sale del estado de Reset. Cuando la CPU sale del Reset, comienza a ejecutar código en la dirección de excepción del Reset. En funcionamiento normal, la BootROM de arranque esta mapeada en la dirección de excepción para que el código comience a ejecutarse en esta. Esta BootROM está embebida en la FPGA y viene configurada de fábrica, pero también existe la opción de mapear una on chip RAM o SDRAM en la dirección de excepción de Reset para arrancar desde estas, aunque para este trabajo no se ha utilizado por no ser necesario.

Normalmente, las principales responsabilidades de la ROM de arranque son:

• Detectar la fuente de inicio seleccionada

• Realizar una inicialización HPS mínima

• Cargue la próxima etapa de arranque (normalmente el Preloader) de Flash a OCRAM y salte a ella

El comportamiento de la ROM de arranque está influenciado por las opciones BSEL y CSEL, que son los registros en donde se indica la fuente de arranque(en donde se encuentra el preloader) y de reloj. En nuestro caso, al estar utilizando una tarjeta de desarrollo, estos registros se pueden configurar físicamente en un switch.

La siguiente imagen (Figura 4-1) muestra los Switches MSEL en la tarjeta en donde tendremos que configurar todos a "0" para hacer un arranque desde la memoria SD/MMC.

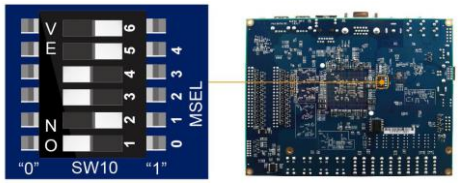


Figura 4-1 – Configuración switches DE1-SoC

Las siguientes etapas del arranque de Linux, preloader, bootloader, sistema operativo y aplicación se detallan en los siguientes apartados. En la siguiente imagen (Figura 4-2) podemos ver como se compone la cadena completa de todo el proceso.

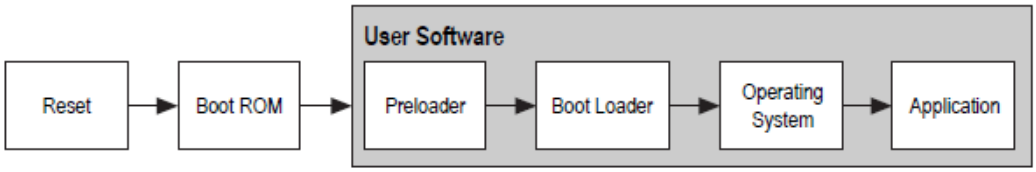


Figura 4-2 – Etapas de arranque sistema Linux

4.1.1. Generación del Preloader.

La generación del Preloader es la primera etapa para componer el sistema Linux sobre el que vamos a trabajar, donde la Boot ROM se encargará de cargar su imagen desde la memoria FLASH dentro de la on-chip RAM y pasar el control a este. Antes de esto la Boot ROM va a chequear que la imagen del Preloader es válida verificando el CRC.

La imagen del Preloader va a tener un tamaño máximo de 60KB, que es limitado por el tamaño de 64KB de la on-chip-RAM.

Profundizando un poco en la función del Preloader, este va a cumplir las siguientes funciones necesarias para la correcta configuración y carga del Kernel de Linux, como son:

- Inicializar la interface de la SDRAM.
- Configurar el registro "remap" de la on-chip RAM a la dirección 0x0, por lo que las excepciones van a ser manejadas por este.
- Configurar las entradas y salidas del sistema HPS.
- Configurar el multiplexado de pines.
- Configurar los relojes del HPS.

- Inicializar la FLASH Controller.
- Cargar el BootLoader dentro de la SDRAM y pasarle el control.

A continuación, podemos ver el diagrama de flujo de los pasos que sigue el Preloader (Figura 4-3).

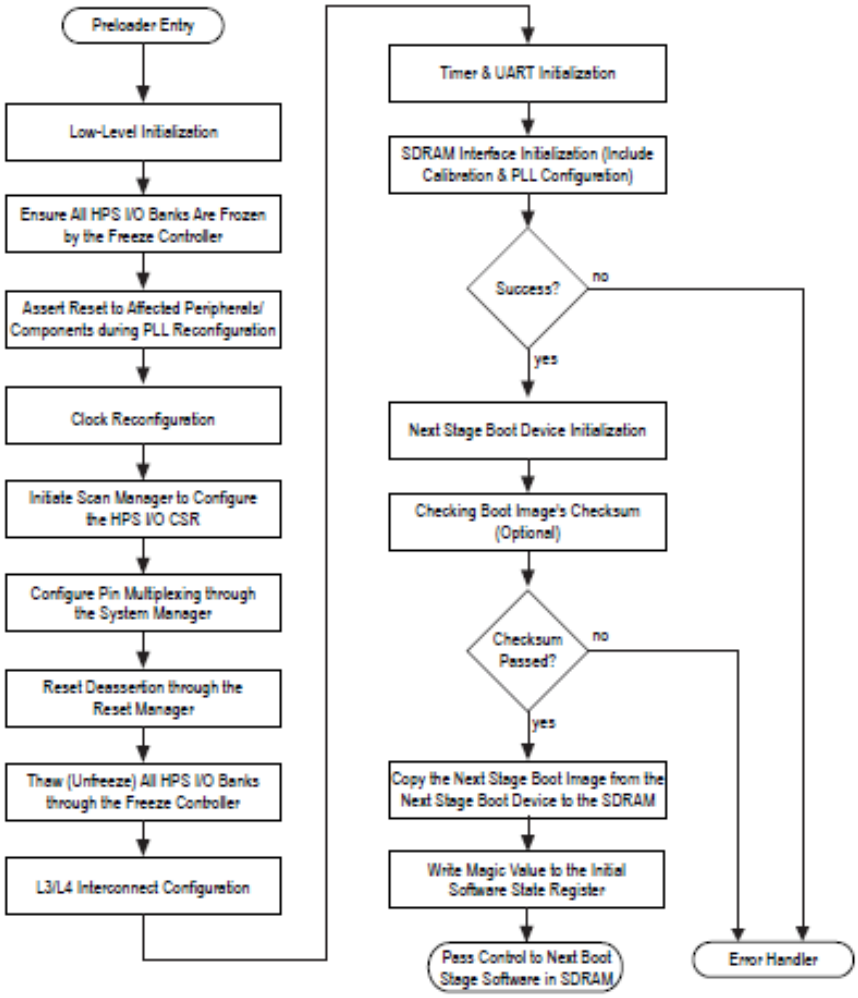


Figura 4-3 – Proceso del preloader

Para la configuración y generación del Preloader, se ha utilizado la herramienta que proporciona Altera en su plataforma SoC EDS, para ello, haciendo uso del command Shell, se ejecuta la línea de comandos "*bsp-editor*", que nos abrirá una interface de usuario con la que podremos realizar todas las configuraciones necesarias y generar el archivo descriptor.

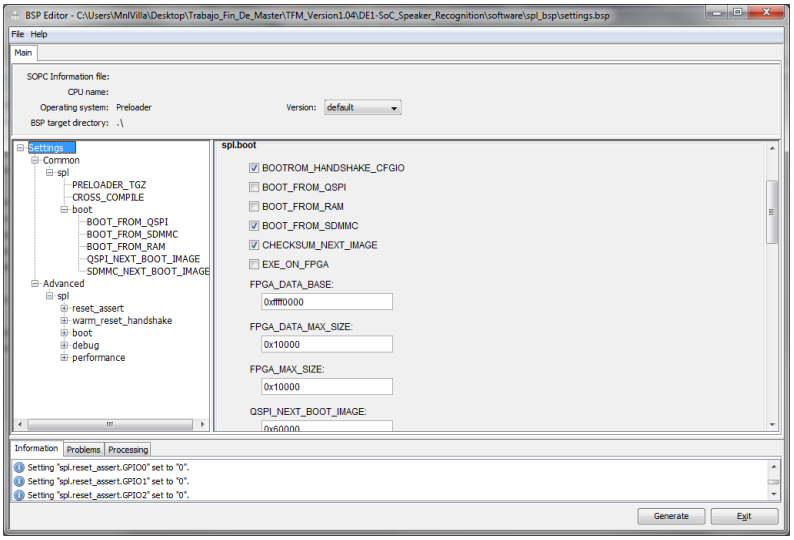


Figura 4-4 – Bsp-Editor del SoC EDS

En esta herramienta se crea una nueva configuración de BSP, donde seleccionamos el arranque desde la SDMMC y configuramos todas direcciones de arranque.

Una vez generado el descriptor con esta herramienta, tendremos que compilar los ficheros generados con el command Shell, simplemente situándonos en el directorio donde generamos los descriptores y ejecutando el comando "make". Con esto generaremos el binario del preloader, el cual se cargará en la tarjeta SD en la primera partición. La forma de configurar la tarjeta SD se explica en los siguientes apartados.

4.1.2. Generación del Bootloader y kernel de Linux.

La siguiente etapa del arranque del sistema Linux se corresponde con el bootloader. Este tiene responsabilidades típicas que son similares a las del preloader Preloader, excepto que no es necesario que arranque la SDRAM. Debido a que el Bootloader ya está residiendo en SDRAM, no está limitado por el tamaño del OCRAM. Por lo tanto, puede proporcionar una gran cantidad de características, como soporte de comunicaciones.

En la configuración de este trabajo, el bootloader va a cumplir las siguientes tareas:

- Configuración del entorno del sistema operativo
- Recuperar la imagen del sistema operativo desde la SD/MMC.
- Almacenar la imagen de arranque en SDRAM y pasar el control al cargador de arranque siguiente, es decir, arrancar el Kernel.
- Modificar el Device Tree Blob.
- Cargar el binario de la FPGA.

Para generar la imagen del bootlader necesitamos tener el código fuente U-Boot. Este código fuente es de código abierto y soporta muchas configuraciones de diferentes arquitecturas,

como PPC, ARM, MIPS, etc. En nuestro caso Altera nos proporciona el código fuente configurado para sus sistemas HPS, el cual se ha compilado desde el command Shell de la herramienta SoC EDS. Para este trabajo no ha sido necesario hacer modificaciones en las configuraciones del código fuente y se ha compilado por defecto.

La siguiente imagen (Figura 4-5) muestra el diagrama de bloques de la generación de la imagen U-Boot:

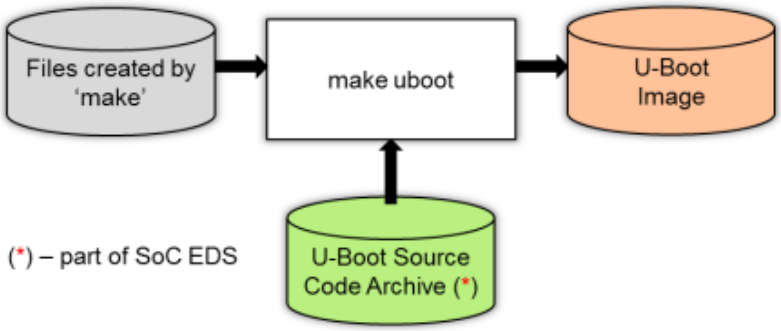


Figura 4-5 – Diagrama de bloques para la generación del U-Boot

Adentrándonos en el código fuente del Bootloader, destacar los siguientes ficheros de configuración:

- include/configs/socfpga_cyclone5.h
- include/configs/socfpga_common.h

El primero se corresponde con el fichero de configuración de la FPGA Cyclone 5. El segundo es un fichero de configuración común para las FPGAs Cyclone 5 y Arria 5. Como nota importante, en este último podemos encontrar las configuraciones de carga del binario de la FPGA y del Device Tree, por eso a la hora de cargar en la partición el binario de la FPGA y el Device Tree Blob los debemos llamar con los nombres "socfpga.rbf" y "soc_system.dtb" respectivamente para que los encuentre y los cargue.

Con el bootloader generado, el siguiente paso es generar el Kernel de Linux que va a ser cargado y el sistema de directorios de este (root filesystem).

Para generar primero el Kernel de Linux, para este trabajo se ha obtenido un Kernel que proporciona el fabricante de la placa de desarrollo, Terasic, el cual viene configurado para trabajar sobre su placa de desarrollo DE1-SoC, configurado para que funciona sobre la FPGA Cyclone 5 y su sistema HPS. También contiene los drivers de Linux para el uso de la pantalla MultiTouch que también proporciona este fabricante.

A la hora de generar la imagen del Kernel de Linux, se ha compilado sobre la máquina virtual de Ubuntu con la que se ha trabajado en este desarrollo, usando el compilador de "gcc-linaro". El proceso seguido no tiene complicación ninguna, donde se han ejecutado los

siguientes comandos situándonos sobre el directorio donde se ha descomprimido el código fuente del kernel y lanzando los siguientes comandos sobre el terminal de ubuntu:

```
> make CROSS_COMPILE=arm-linux-gnueabihf- ARCH=arm socfpga_defconfig
> alias armmake='make CROSS_COMPILE=arm-linux-gnueabihf- ARCH=arm'
>armmake -j4 uImage LOADADDR=0x8000
```

Una vez que la compilación ha terminado, en el directorio de la carpeta de trabajo "arch/arm/boot/" podemos encontrar los binarios generados.

El último paso para que el sistema Linux funcione es la generación del "root filesystem", es decir, es sistema de directorios de Linux, que se instalarán sobre la partición EXT Linux. Aunque existen diferentes herramientas para la generación del "root filesystem" con la configuración que necesitemos, en el caso de este trabajo se ha optado por extraerlo de una imagen de un ejemplo proporcionado por Terasic y Altera.

4.1.3. Configuración del Device Tree.

Una parte importante de la configuración del sistema Linux embebido es el Device Tree, que se compone de una estructura de datos que describen el hardware presente en el sistema operativo. Al pasar esta estructura de datos al Kernel del sistema operativo, un solo binario del sistema operativo puede ser capaz de soportar muchas variaciones de hardware. Esta flexibilidad es particularmente importante cuando el hardware incluye una FPGA.

La herramienta Generador de Device Tree forma parte de Altera SoC EDS y se utiliza para crear Device Tree's para sistemas SoC que contienen diseños FPGA creados con Qsys. El Device Tree generado describe los periféricos HPS, FPGA Soft IP seleccionados y también periféricos dependientes de la placa.

Mediante el uso de un Device Tree para configurar un kernel en tiempo de ejecución, en lugar de un código compilado, se puede utilizar un único kernel binario en una amplia variedad de tablas y diseños de FPGA.

Un árbol de Device Tree contiene algunos o todos los elementos siguientes:

- Número de CPUs.
- Tamaño y ubicación de varias RAMs.
- Buses y puentes.
- Conexiones de los dispositivos periféricos
- Controladores de interrupciones y conexiones de línea IRQ.
- Configuración específica del controlador de dispositivo, como:
    - Dirección MAC de Ethernet
    - Reloj de entrada del periférico

Para usar los Device Tree, hay que hacer una representación textual llamada Device Tree Source (DTS). El DTS se compila entonces en una representación binaria llamada Device Tree Blob (DTB). El DTB se entrega al kernel en el arranque.

El compilador del Device Tree (dtc) se construye como parte de la compilación del kernel de Linux, y también está disponible como parte del SoC EDS.

El uso de Device Tree reduce la necesidad de recompilar el kernel, pero no elimina completamente la necesidad de recompilar un kernel cuando cambia un diseño FPGA. Específicamente, si se agrega un nuevo componente que requiere un nuevo controlador y / o una nueva configuración del kernel, El kernel de Linux tendrá que ser recompilado. Esto mismo nos pasa en este trabajo, donde es necesario compilar el kernel de Linux con los drivers para el uso del Frame Buffer y de la TouchScreende Linux para configurarlos para el uso del hardware implementado en la FPGA, ya que estos se empiezan a usar en el arranque del sistema, por lo que no se puede cargar dinámicamente, lo que provocaría una parada del sistema.

Para módulos que no necesitan ser arrancados justo al inicio, se pueden evitar las recompilaciones completas del kernel si los controladores se construyen como módulos cargables dinámicamente. No obstante, los Device Tree no se pueden utilizar para módulos cargados dinámicamente.

La primera parte, como se comenta anteriormente, es generar el Device Tree Souerce (.dts), para el cual necesitamos que este generado el fichero ".sopcinfo" que nos da la herramienta Qsys. También es necesario tener los ficheros con extensión ".xml", los cuales nos dan la información de los periféricos de la placa de desarrollo DE1-SoC y de los relojes que están conectados al sistema HPS. Estos se han obtenido de los ejemplos de ayuda que proporciona Terasic con la placa de desarrollo.

La siguiente imagen (Figura 4-6) muestra el diagrama de bloques del proceso para generar el Device Tree.

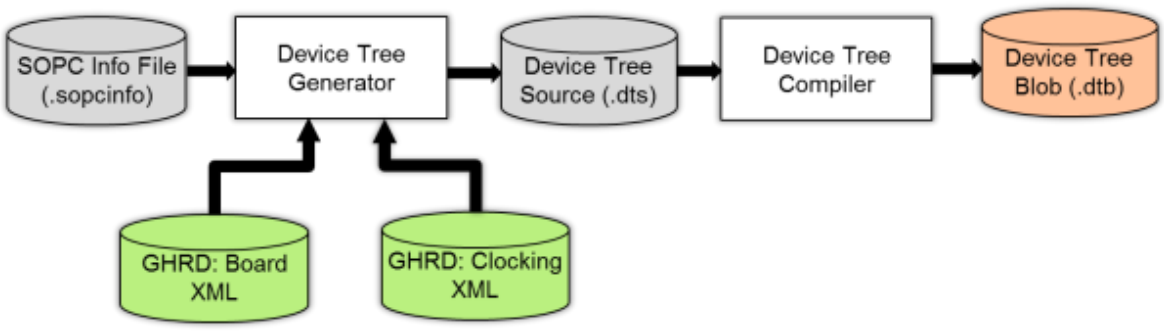


Figura 4-6 – Diagrama de bloques de la generación del Device Tree Blob

Una vez que tenemos los ficheros descritos anteriormente, se puede generar el fichero ".dts". Para ello se hace uso del command Shell que proporciona el entorno SoC EDS, en el cual ejecutaremos el siguiente comando:

```
sopc2dts --input soc_system.sopcinfo --output socfpga.dts --board soc_system_board_info.xml --board
hps_clock_info.xml
```

En este punto, para este trabajo ha sido necesario tocar a mano una serie de configuraciones para adaptar el fichero ".dts" generado a la composición del sistema, como es la configuración del módulo de la TouchScreen y Video Frame Buffer implementados en la parte de la FPGA. También se ha tocado la inicialización de la comunicación ethernet, para que esté configurada en el arranque y poder acceder al terminal de Linux del sistema a través del protocolo "ssh".

En el caso de los módulos implementados en la FPGA, se modifican para adaptar las direcciones y offsets donde están mapeados en el sistema Qsys de la siguiente forma:

```
alt_vip_vfr_0: vip@0x100000100 {
        compatible = "ALTR,vip-frame-reader-13.1", "ALTR,vip-frame-reader-9.1";
        reg = < 0x00000001 0x00000100 0x00000080 >;
        max-width = < 800 >;   /* MAX_IMAGE_WIDTH type NUMBER */
        max-height = < 480 >;   /* MAX_IMAGE_HEIGHT type NUMBER */
        bits-per-color = < 8 >;   /* BITS_PER_PIXEL_PER_COLOR_PLANE type NUMBER */
        colors-per-beat = < 4 >;   /* NUMBER_OF_CHANNELS_IN_PARALLEL type NUMBER */
        beats-per-pixel = < 1 >;   /* NUMBER_OF_CHANNELS_IN_SEQUENCE type NUMBER */
       mem-word-width = < 256 >;   /* MEM_PORT_WIDTH type NUMBER */
        }; //end vip@0x100000100 (alt_vip_vfr_0)

terasic_mtl_touch: tsc@0x100000200 {
        compatible = "terasic,mlt_touch_screen";
        reg = < 0x00000001 0x00000200 0x00000080 >;
        width_pixel = < 800 >;
        height_pixel = < 480 >;
        interrupt-parent = < &hps_0_arm_gic_0 >;
       interrupts = <0 40 4>;
       };  //end tsc@0x1000000200
```

En el caso de la inicialización del ethernet se añaden al final del fichero las siguientes líneas:

```
chosen {
     bootargs = "console=ttyS0,115200";
     }; //end chosen

aliases {
    /* this allow the ethaddr uboot environmnet variable contents
    * to be added to the gmac1 device tree blob.
    */
    ethernet0 = &hps_0_gmac1;
    }
```

Estos cambios se han hecho han mano directamente con un editor de textos, pero existe la opción de configurar estos componentes en el sistema Qsys para que generen en el fichero ".sopcinfo" la información para la posterior generación del fichero ".dts" de forma correcta.

Una vez que se tiene el Device Tree Source generado y configurado se puede generar el fichero Device Tree Blob (".dtb") que se cargará en la tarjeta SD para que haga uso de el Bootloader de Linux en su arranque.

Para la generación del ".dtb" se ha utilizado el commnad Shell del entorno de desarrollo SoC EDS, donde se lanza la siguiente línea de comandos para la generación:

```
dtc -I dts -O dtb -o socfpga.dtb socfpga.dts
```

Antes de cargar en la partición correspondiente de la tarjeta SD, se ha cambiado el nombre de este archivo, debido a que el kernel de Linux esta configurado para buscar en esa partición el Device Tree con el siguiente nombre:

- soc_system.dtb

4.1.4. Configuración de la tarjeta microSD.

Una vez que tenemos todos los ficheros y binarios que componen el sistema Linux, se cargan en la tarjeta SD. Para realizar este proceso se utiliza un script proporcionado por Altera para este fin, el cual realiza las particiones necesarias y cargar los binarios en la tarjeta.

```
$ /opt/altera-linux/bin/make_sdimage.sh -h
usage: make_sdimage.sh [-h] [-k f1,f2] [[-p preloader |-rp preloader] -b bootloader] [-r rfs_dir [-m
merge_dir]] [-o image] [-g size] [-t tool]
-k f1,f2,...   comma separated list of files to be copied into FAT partition, i.e. OS files (zImage, dtb) or
FPGA image (rbf)
-p preloader   preloader file with mkpimage header (i.e preloader-mkpimage.bin). Mutually exclusive
with -rp.
-rp preloader  raw preloader file (i.e u-boot-spl.bin). Specify your preloader header tool with -t option.
Mutually exclusive with -p.
-b bootloader  bootloader file (i.e. u-boot.img)
-r rfs_dir     location of the root file system files.
-o image       name of generated image file. Default filename is image_blk_demo.bin
-m merge_dir   copy files located in dir/ to rfs.
-g size[K/M/G] size of generated image, in unit KB, MB or GB (i.e 2000M). Default size is 2GB
-t tool        tool to geneate preloader with header (i.e /mkpimage). Use this with -rp option.
-h             this message
```

Lanzando este script, la tarjeta quedaría configurada con las siguientes particiones (Figura 4-7):

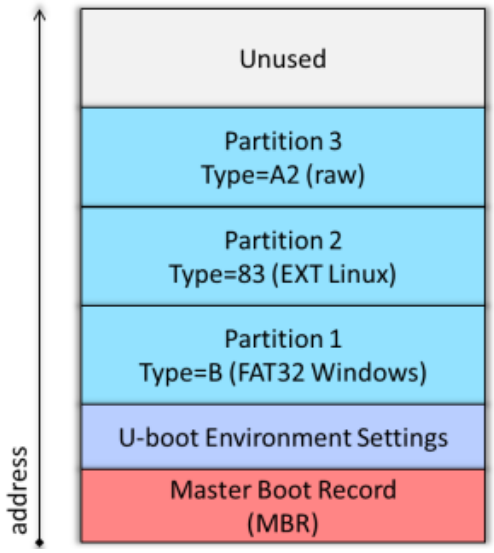


Figura 4-7 – Esquema particiones tarjeta SD

En la siguiente tabla podemos ver en que partición se instalan los diferentes ficheros, binarios e imágenes generadas para la composición del sistema Linux.

Localización	Nombre del fichero	Descripción
Partición 1	socfpga.rbf	Binario de la FPGA
	soc_system.dtb	Device Tree Blob
	zImage	Imagen comprimida del kernel
Partición 2	varios	Linux Root Filesystem
Partición 3	n/a	Imagen del Preloader
	n/a	Imagen del U-Boot

Aunque no es estrictamente necesario, es recomendable, también se copia la “zImage" así como el archivo “vmlinux" situado en la raíz del kernel en la carpeta “/boot" de la partición ext3 (root filesystem) de la SD, renombradas con la versión como sufijo, por ejemplo “zImage-3.12.0" y “vmlinux-3.12.0", al tiempo que se debe crear o modificar en dicha carpeta un enlace dinámico llamado “zImage" que enlace al fichero real.

Ya con la tarjeta SD configurada y cargada, con solamente introducirla en la placa de desarrollo DE1-SoC y encender el sistema, el sistema Linux se arranca y podemos acceder al terminal de Linux por puerto serie o a través de Ethernet con el protocolo SSH.

4.1.5. Generación de las librerías QT para interface gráfica.

Para poder realizar el diseño haciendo uso de los diferentes elementos prediseñados para interfaces de usuario que nos proporciona el entorno QT, como son widgets, botones, cuadros de texto, etc., con los que podremos crear de forma más o menos sencilla nuestra interface de usuario, necesitas generar una librería QT que se ajuste a nuestro sistema.

Con esta librería, donde configuraremos la herramienta QT Creator para que haga uso de ella, se pueden instanciar en el código fuente de forma sencilla las clases que usan los elementos de interface de usuario disponibles en la herramienta. La siguiente imagen (Figura 4-8) muestra la zona de trabajo dentro de la herramienta QT Creator donde se instancian estos elementos.

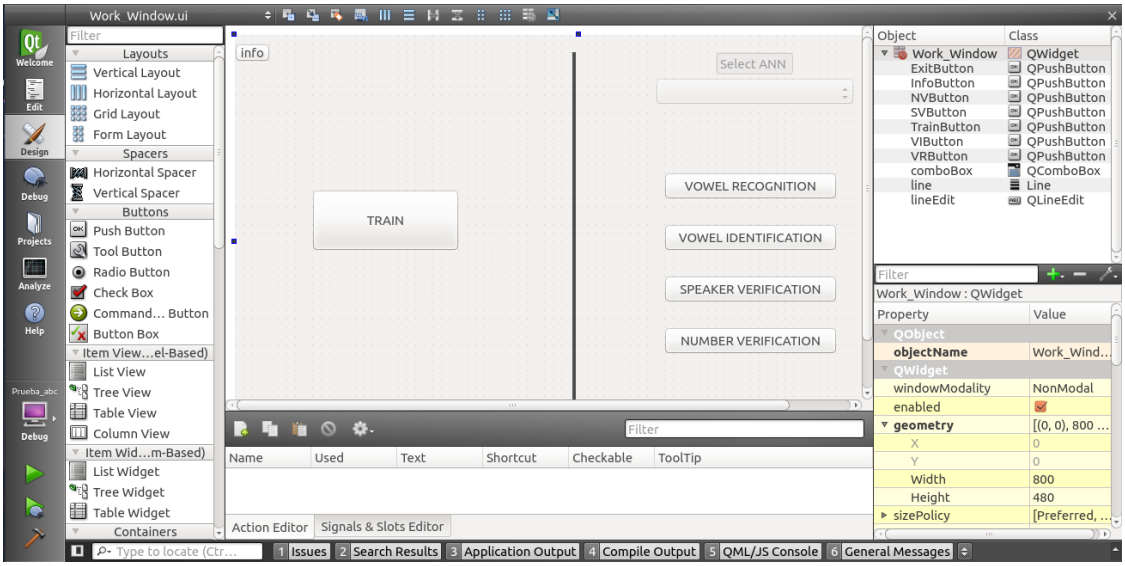


Figura 4-8 – Editor de interfaces de QT

Para generar esta librería, primero se ha obtenido del directorio web de QT (https://download.qt.io/archive/qt/4.8/4.8.6/, opensource) la librería para sistemas de Linux Embebido con la que podremos trabajar. Una vez que se ha obtenido la librería, se ha compilado con el mismo compilador con el que se ha compilado el kernel de Linux (gcc-linaro) realizando las configuraciones necesarias para que pueda funcionar en este sistema.

Para realizar las configuraciones necesarias, se ha hecho uso del terminal de Ubuntu (máquina virtual sobre la que se trabaja), donde primero nos situaremos en el directorio donde se han descomprimido los fuentes de la librería, luego, lanzando la línea de comandos que se puede ver en el siguiente cuadro, vamos a configurar la plataforma sobre la que trabajamos, la plataforma destino, el uso del ratón, etc.

```
./configure -v -qvfb -opensource -confirm-license -prefix /opt/qt2 -embedded arm -platform qws/linux-x86_64-g++ -xplatform qws/linux-altera-g++ -depths 16,24,32 -no-mmx -no-3dnow -no-sse -no-sse2 -no-cups -no-largefile -no-accessibility -no-openssl -no-gtkstyle -qt-mouse-pc -qt-mouse-linuxtp -qt-mouse-linuxinput -qt-mouse-tslib -plugin-mouse-linuxtp -plugin-mouse-pc -plugin-mouse-tslib -fast -little-endian -host-big-endian -no-pch -no-sql-ibase -no-sql-mysql -no-sql-odbc -no-sql-psql -no-sql-sqlite -no-sql-sqlite2 -no-webkit -no-qt3support -nomake examples -nomake demos -nomake docs -nomake translations
```

Una vez configurado sólo es necesario lanzar el comando "*make*" y se generará la librería en el directorio configurado, para que posteriormente la herramienta QT haga uso de ella.

Antes de compilar la librería de QT, se ha descubierto que era necesario configurar diferentes ficheros de la librería para poder trabajar sobre la plataforma de Altera, donde en el directorio de los fuentes ("*mkspecs/qws/linux-altera-g++*") se configuran los ficheros de configuración "*qmake.conf*" y "*qplatformdefs.h*" para que se use el compilador de "*gcc-*

linaro" y que esta librería haga uso de la librería TSLIB que se explica a continuación. También es necesario configurar el fichero "*makefile*" para que haga uso de este directorio.

Aparte de la librería de QT, también se ha tenido que generar una librería para poder trabajar con la TouchScreen y habilitar los eventos de entrada del ratón. Para ello se ha obtenido del directorio web de TSLIB esta librería (www.tslib.org, versión 1.0), la cual se configura y compila de igual modo que la librería QT. El siguiente cuadro muestra la línea de comandos usada para configurar esta librería:

```
./autogen-clean.sh
./autogen.sh
./configure CC=/opt/altera-linux/linaro/gcc-linaro-arm-linux-gnueabihf-4.7-2012.11-20121123_linux/bin/arm-linux-gnueabihf-gcc CXX=/opt/altera-linux/linaro/gcc-linaro-arm-linux-gnueabihf-4.7-2012.11-20121123_linux/bin/arm-linux-gnueabihf-g++ -host=arm-none-linux-gnueabi target=arm-none-linux-gnueabi -enable-static=yes -enable-shared=yes -prefix=/opt/qt/tslib ac_cv_func_malloc_0_nonnull=yes
```

Comentar también que al compilar la librería de TSLIB, esta genera un ejecutable para poder calibrar la pantalla TouchScreen ("ts_calibrate"), el cual podemos lanzar en el sistema de la DE1-SoC para calibrar las coordenadas que nos da la pantalla. Este ejecutable lanza una simple aplicación de usuario, la cual nos da varios puntos sobre los que tenemos que pulsar y de esta forma configurar el fichero "etc/ts.conf" que se encargará de calibrar las coordenadas de la pantalla en el arranque de Linux.

Para lanzar este ejecutable es necesario generar un script con las siguientes configuraciones antes de lanzarlo.

```
export LD_LIBRARY_PATH=/usr/lib
export QTDIR=/opt/qt2
export TSLIB_ROOT=/usr/lib
export TSLIB_CONFFILE=/etc/ts.conf
export TSLIB_PLUGINDIR=/usr/lib/ts
export TSLIB_CALIBFILE=/etc/pointercal
export TSLIB_TSDEVICE=/dev/input/event0
export TSTS_INFO_FILE=/sys/devices/soc.0/ff200000.bridge/ff200200.tsc/input/input0/uevent
export QWS_MOUSE_PROTO=Tslib:/dev/input/event0
export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_CONSOLEDEVICE=none
export TSLIB_TSEVENTTYPE=INPUT
export PATH=$PATH:/usr/bin
ts_calibrate
```

4.1.6. Aplicación de usuario.

La aplicación de usuario desarrollada se trata de un ejecutable desarrollado sobre C++ y que hace uso de las librerías de QT mencionadas en los anteriores apartados para la creación de la interface gráfica de usuario que vamos a poder ver y controlar sobre la pantalla LCD.

Esta aplicación se ha planteado en diferentes pantallas o "widgets", donde a través de diferentes botones y controles se podrá controlar la red neuronal. A continuación, se describen las pantallas que contiene la aplicación de usuario:

- Pantalla de Inicio: Esta es la pantalla que se mostrará en el arranque del sistema, en la cual podremos ver una portada del trabajo y un botón de inicio para ir a la pantalla de trabajo.

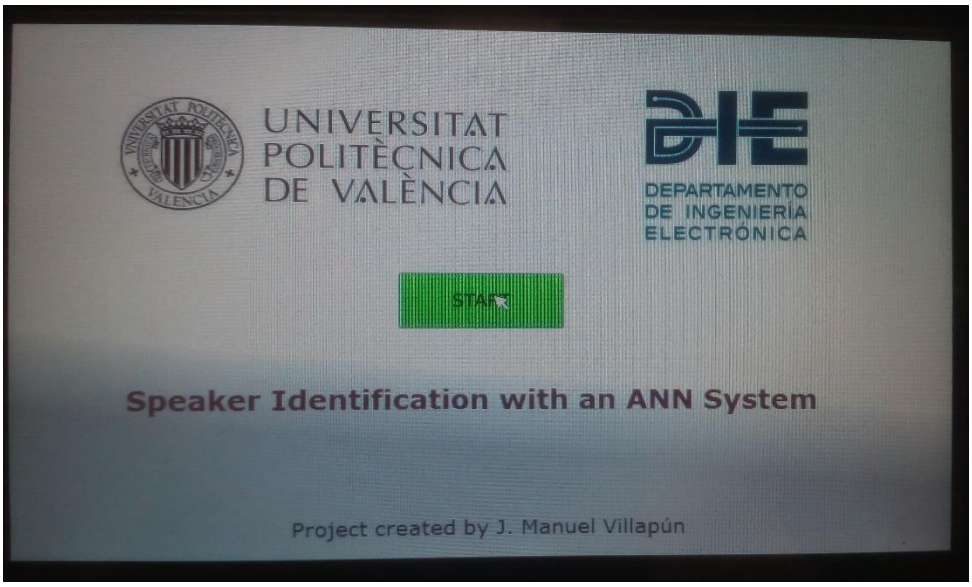


Figura 4-9 – Pantalla de inicio del sistema

- Pantalla de trabajo: Esta pantalla está separada en dos secciones, por un lado, para acceder a la pantalla de entrenamiento de la red neuronal, y en la otra sección todas las opciones de prueba de la misma, que nos llevarán a otra pantalla de prueba y resultados.

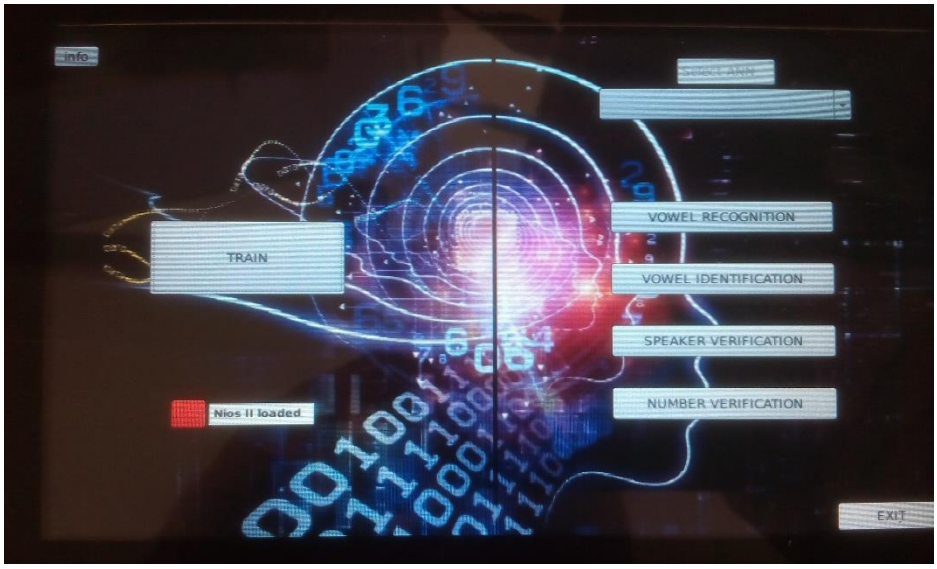


Figura 4-10 – Pantalla de trabajo del sistema

- Pantalla de entrenamiento: Sobre esta pantalla tenemos todas las opciones de configuración para el entrenamiento de la red neuronal, como son la selección del fonema a entrenar.

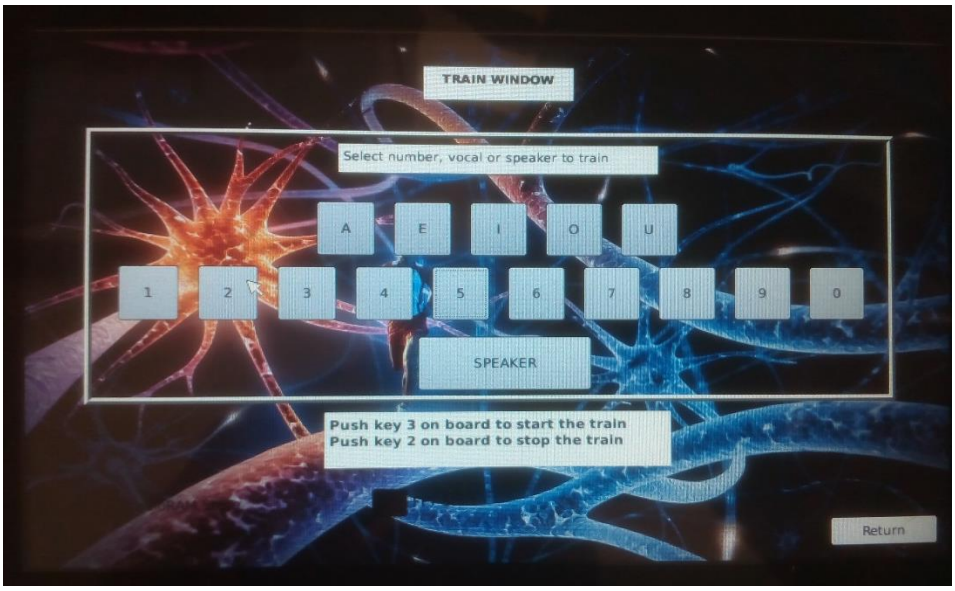


Figura 4-11 – Pantalla de entrenamiento del sistema.

- Pantallas de prueba: Sobre esta pantalla se representan los resultados al excitar la red neuronal entrenada, tanto en el caso de la identificación como en la verificación.

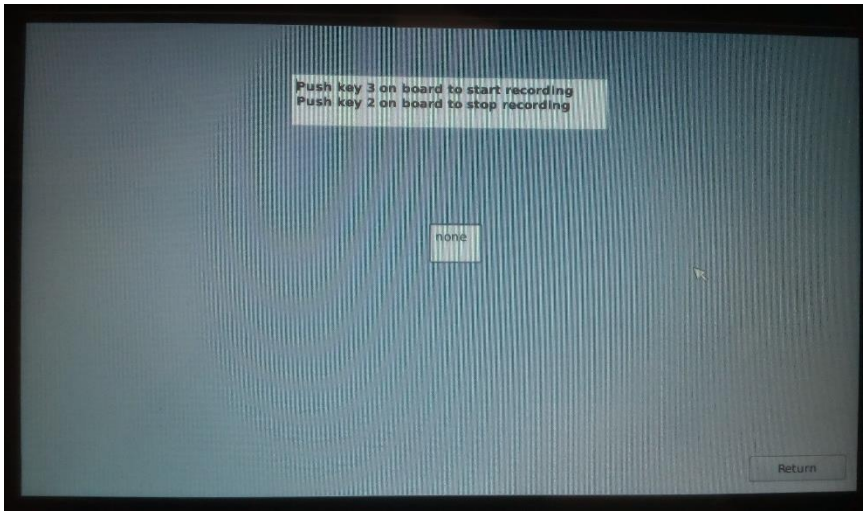


Figura 4-12 – Pantalla de verificación/identificación del sistema

- Ventana de ayuda: Esta ventana se abre en la pantalla de trabajo y muestra toda la información necesaria sobre el funcionamiento del sistema.

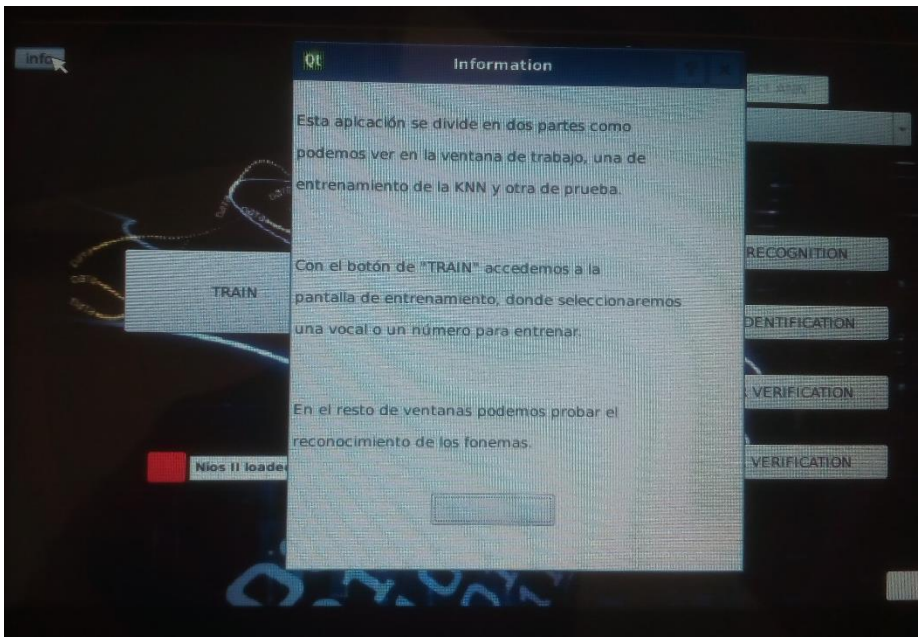


Figura 4-13 – Ventana de ayuda del sistema.

Para no tener que abrir un terminal de Linux y lanzar la aplicación, se ha configurado el fichero de configuración de Linux, "etc/inittab" para que lance este ejecutable en su arranque. También se configura para que, al cabo de 5 segundos de una caída del ejecutable, se relance con la opción "respawn".

A la hora de desarrollar el código fuente del ejecutable se ha hecho uso del dispositivo "/dev/mem" de Linux que provee acceso directo al mapa completo de memoria del HPS. Abriéndolo con la opción "O SYNC" de la orden "open" para evitar la caché y mediante la llamada al sistema "mmap" y las direcciones de los registros de la FPGA se pueden crear punteros para acceder a cualquier dispositivo. Sobra decir que esta operación es peligrosa, ya que el acceso a una zona de memoria incorrecta puede corromper el estado de ejecución del sistema operativo. Las llamadas a las APIs del HWLib, por su parte, están desaconsejadas desde un programa de usuario, aunque sea el superusuario.

```
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/fcntl.h>
#include <sys/stat.h>
#include <sys/ioctl.h>

fdDevMem = open( "/dev/mem", ( O_RDWR | O_SYNC ) )
virtual_base = mmap( NULL, HW_REGS_SPAN, ( PROT_READ | PROT_WRITE ), MAP_SHARED,
fdDevMem, HW_REGS_BASE );
munmap(virtual_base,HW_REGS_SPAN);
close(fdDevMem);
```

Por último, una función importante de este ejecutable es la carga del sistema Nios II, donde se realiza el proceso descrito en el apartado de descripción hardware, en donde primero se configuran los registros de las salidas GPIO y luego se carga en memoria el binario del Nios II.

4.1.7. Red neuronal (diseño 2)

En este segundo diseño, al eliminarse el sistema Nios II implementado en la parte hardware y que contiene la implementación en software del algoritmo de la red neuronal, tal y como se explica en la sección de diseño hardware, nace la necesidad de albergar en la parte HPS-Linux el algoritmo de la red neuronal.

De un principio se pensó y se empezó a desarrollar dentro del mismo ejecutable de la aplicación de usuario, donde se estarían ejecutando dos hilos dentro del mismo, es decir, dos tareas separadas. Para poder hacer esto, al no estar trabajando sobre un RTOS, se hace uso de la librería de QT, la cual nos da la opción de usar la case "QThread" que proporciona una plataforma independiente de hilos. Un QThread representa un hilo de control separado dentro del programa; Comparte datos con todos los otros subprocesos dentro del proceso, pero se ejecuta independientemente en la forma en que un programa independiente hace en un sistema operativo multitarea.

Finalmente se ha separado el diseño en dos ejecutables corriendo en el sistema operativo Linux, ya que, de esta forma, donde es el sistema operativo quien gestiona la planificación, resulta más sencillo conseguir el funcionamiento del sistema.

Por un lado, se ha mantenido el ejecutable del diseño 1 que ya se estaba ejecutando en el sistema Linux, sobre el cual corre la aplicación de usuario que gestiona la visualización de la pantalla LCD y el control por parte del usuario. Para poder mantener este ejecutable, solamente se han hecho unos pequeños cambios para adaptarlo a la nueva interface de control desarrollada en la parte hardware y que nos va a servir para la transferencia de información entre los dos ejecutables que corren sobre el sistema operativo de forma segura.

El segundo ejecutable diseñado en este segundo diseño, en principio se trata del mismo código implementado en la parte Nios II del diseño 1, pero se adapta para que corra sobre la parte Linux y se comunique con el primer ejecutable a través de la interface de control implementada en la parte hardware. En este caso no se hace uso de la librería de QT, ya que no es necesario, por lo que se va a ejecutar como un loop infinito.

4.2. Diseño Software sobre NIOS II

En este apartado se hace una introducción al funcionamiento del software implementado en la parte del Nios II, en el cual reside el algoritmo de la red neuronal. Se hace una breve explicación del tipo de algoritmo de entrenamiento utilizado y de su funcionamiento, así como el control de los diferentes modos en los que se puede trabajar.

4.2.1. Algoritmo de aprendizaje de la red neuronal.

El sistema de identificación y verificación del interlocutor estará basado en máquinas de aprendizaje de dos tipos diferentes:

- K -NN algorithm (como representante de Eager learning)

  Para el entrenamiento e identificación de vocales, se usa el algoritmo K-NN. Este es un método de clasificación supervisada que sirve para estimar la función de densidad F(x/Cj) de las predictoras "x" por cada clase "Cj".

  Este es un método de clasificación no paramétrico, que estima el valor de la función de densidad de probabilidad o directamente la probabilidad a posteriori de que un elemento "x" pertenezca a la clase "Cj" a partir de la información proporcionada por el conjunto de prototipos. En el proceso de aprendizaje no se hace ninguna suposición acerca de la distribución de las variables predictoras.

  La fase de entrenamiento del algoritmo consiste en almacenar los vectores característicos y las etiquetas de las clases de los ejemplos de entrenamiento. En la fase de clasificación, la evaluación del ejemplo (del que no se conoce su clase) es representada por un vector en el espacio característico. Se calcula la distancia entre los vectores almacenados y el nuevo vector, y se seleccionan losejemplos más cercanos. El nuevo ejemplo es clasificado con la clase que más se repite en los vectores seleccionados.

- Reilient Backpropagation (como representante de Lazy Learning).

  Para el caso del entrenamiento de números y su identificación se utiliza un algoritmo de red neuronal Backpropagation. También se hace uso de este algoritmo para la verificación de vocales.

  La unidad procesadora básica de la red Backpropagation se representa en la figura 4-14. Las entradas se muestran a la izquierda, y a la derecha se encuentran unidades que reciben la salida de la unidad procesadora situada en el centro de la figura. La unidad procesadora se caracteriza por realizar una suma ponderada de las entradas llamada S_j, presentar una salida a_j y tener un valor δ_j asociado que se utilizará en el

proceso de ajuste de los pesos. El peso asociado a la conexión desde la unidad i a la unidad j se representa por w_{ji}, y es modificado durante el proceso de aprendizaje.

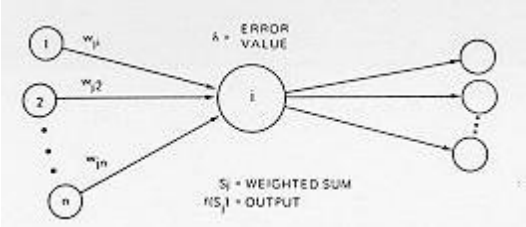


Figura 4-14 – Procesadora Backpropagation

En la siguiente imagen (Figura 4-15) se puede ver un ejemplo de una Backpropagation de tres capas. La capa inferior es la capa de entrada, y se caracteriza por ser la única capa cuyas unidades procesadoras reciben entradas desde el exterior. Sirven como puntos distribuidores, no realizan ninguna operación de cálculo. Las unidades procesadoras de las demás capas procesan las señales como se indica en la figura 45. La siguiente capa superior es la capa oculta, y todas sus unidades procesadoras están interconectadas con la capa inferior y con la capa superior. La capa superior es la capa de salida que presenta la respuesta de la red.

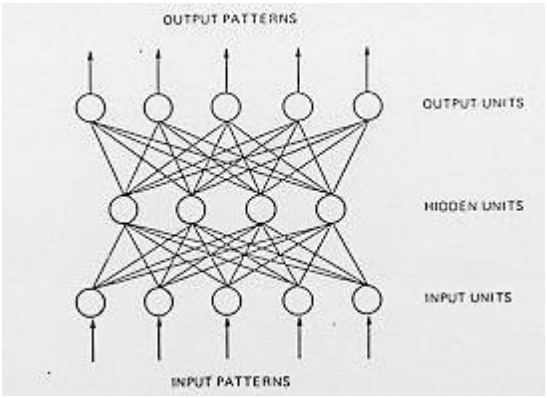


Figura 4-15 – Ejemplo capas backpropagation

Para este trabajo, cuando estamos en modo identificación de números, se utiliza una backprogation de 2 capas de conexión, 20 nodos de entrada y 10 de salida. Cuando se trata de verificación de vocales, el diseño cambia a 2 capas de conexión, con 20 nodos de entrada y 1 de salida.

Las redes Backpropagation tienen un método de entrenamiento supervisado. A la red se le presenta parejas de patrones, un patrón de entrada emparejado con un patrón de salida deseada. Por cada presentación los pesos son ajustados de forma que disminuya el error entre la salida deseada y la respuesta de la red.

El algoritmo de aprendizaje backpropagation conlleva una fase de propagación hacia adelante y otra fase de propagación hacia atrás. Ambas fases se realizan por cada patrón presentado en la sesión de entrenamiento.

4.2.2. Entrenamiento y prueba de la red neuronal.

Para el control y manejo de la red neuronal, como ya se explica en la sección del diseño hardware, se ha implementado una interface de control a través de registros. Estos registros son los que configuran el modo de funcionamiento de la red neuronal y en los que se representa el resultado de excitar a la misma.

Estos registros están mapeados en zona de memoria del Nios II, y se accede a ellos a través de la HAL generada a partir del fichero ".sopcinfo" en el entorno de desarrollo Eclipse para Nios II que proporciona Altera en la herramienta Quartus II.

En esta parte del software implementado, existen dos modos de funcionamiento, el modo de entrenamiento de la red neuronal y el modo de prueba, para ello se utiliza 1 registro, el cual es leído en cada ciclo de proceso que se lanza con cada detección de muestras procesadas.

- Registro Train: Este registro nos va a indicar si estamos en modo entrenamiento o en modo de reconocimiento/verificación.

Luego existe otro registro, con el cual se configura el tipo de entrenamiento o el el caso de estar en modo de prueba el tipo de reconocimiento/verificación

- Registro AppSwitches: Con este registro se va a configurar el tipo de entrenamiento o de reconocimiento/verificación de fonemas.
    - Reconocimiento de vocales.
    - Reconocimiento de números.
    - Verificación de número.
    - Verificación de vocal.

Otro registro de configuración es el siguiente:

- Registro VowelID, con el cual configuramos cuando estamos en modo entrenamiento el fonema que estamos entrenando.

Otra parte importante de la interface de control es la de indicación, con la cual se da el resultado de las operaciones realizadas sobre la red neuronal. Para ello se utilizan 3 registros, con los cuales se representa el resultado cuando estamos en modo prueba de excitar la red neuronal.

- Registro VowelLEDs: En este registro se codifica el resultado de excitar la red neuronal en modo prueba, el cual nos da el fonema identificado, tanto vocales como números.
- Registro RedLED: Este nos indica en pruebas de verificación un resultado erróneo al excitar la red neuronal.
- Registro GreenLED: Este nos indica en pruebas de verificación un resultado de éxito al excitar la red neuronal.

5. Resultados experimentales

En este apartado se presentan los resultados de las pruebas experimentales llevadas a cabo para validar el funcionamiento del sistema diseñado. Para ello se expone una tabla donde podemos ver la tasa de reconocimiento de la red neuronal.

Para llevar a cabo estas pruebas se ha hecho uso de tres interlocutores, suficientes para realizar las pruebas de concepto, pero insuficientes para poder sacar una estadística fiable sobre la tasa de reconocimiento.

El primer paso llevado a cabo ha sido el entrenamiento de la KNN para la identificación de vocales y de la Backpropagation para la identificación de números. Una vez entrenados, se procede a probar la tasa de reconocimiento con los siguientes resultados.

	KNN			Backpropagation		
Prueba Nº	Interlocutor 1	Interlocutor 2	Interlocutor 3	Interlocutor 1	Interlocutor 2	Interlocutor 3
1	√	X	√	X	X	√
2	√	X	X	√	√	√
3	√	√	√	√	√	√
4	X	√	X	√	√	√
5	√	√	√	√	√	√

Tasa de identificación de la KNN: 67%

Tasa de identificación de la Backpropagation: 86 %

6. Conclusiones

El principal objetivo de este trabajo, como ya se introduce al principio de este documento, era la creación de una interface de usuario de fácil uso en Linux para pruebas con redes neuronales. Para tal propósito se ha utilizado una FPGA con un procesar ARM embebido en el cual se configura un sistema operativo Linux embebido.

Esta tecnología usada es bastante reciente por lo cual el principal problema ha sido el aprendizaje de esta, teniendo en cuenta que la mayoría de la información sobre su uso está bastante dispersa y no existen muchos ejemplos de uso por el momento. Esto ha complicado bastante llegar a tener un sistema configurado con el que poder trabajar e implementar los programas de usuario que corren sobre Linux, sobre todo las primeras fases de arranque del sistema, como son el Preloader, Bootloader y Device Tree.

También comentar la dificultad encontrada para generar las librerías de QT para que funcionaran sobre el sistema ARM de Altera, de las que si que no hay información y he necesitado ver diferencias existentes entre otros sistemas como Rasperry Pi y Beaglebone Black para ver como configurarla y generarla. Una vez que he tenido la librería configura, si que puedo concluir que ahora puedo implementar una interface de usuario agradable de forma sencilla.

Todo este proceso de aprendizaje y configuración me ha ayudado a tener una mejor idea de cómo funciona un sistema con Linux embebido y así poder ver todas sus posibilidades y ventajas de uso.

A partir de aquí, puede decir, como conclusión del montaje del sistema Linux, que trabajar sobre estos sistemas facilita de forma considerada, tanto en complejidad como en tiempo, la implementación de interfaces de usuario y también el trabajar con el hardware disponible.

Como objetivo de este trabajo también estaba el uso de algoritmos de aprendizaje para el reconocimiento y verificación de fonemas, donde se necesita un previo procesamiento de señal para poder caracterizar los fonemas. En este caso se han encontrado varios problemas, por un lado, la parte implementada en la FPGA para procesar la señal, en concreto el banco de filtros logarítmicos, tenían un gran peso en la FPGA, por lo que a la hora de compilar esta se desbordaba (la Cyclone V utilizada es de gama baja). En esta parte he tenido que recibir ayuda de mi tutor para solventar el problema y conseguir una implementación menos pesada, por que como conclusión he visto que sería mejor intentar liberar la parte de lógica programable de estas implementaciones para hacerlas en procesadores potentes o en GPUs.

Otro de los problemas encontrados ha sido el estudio de los algoritmos de aprendizaje, donde me ha llevado un tiempo llegar a comprenderlos e integrarlos en el código, por lo que en esta parte aún necesitaría profundizar más en el tema para futuras mejoras.

Como conclusión del último objetivo marcado, puedo decir que lo más complicado ha sido hacer que las dos partes de este sistema, la parte implementada en la FPGA (lógica programable y sistema Nios II) y la parte diseño Linux en el HPS, funcionaran y se

comunicarán correctamente entre si, para de esta forma poder validar y probar el reconocimiento e identificación.

En cuanto a la prueba de la interface gráfica, una vez que el sistema Linux lo he tenido montado, su desarrollo y prueba ha sido sencilla, principalmente por que el entorno QT incorpora un compilador/depurados en nativo, con lo que pude probar el software de forma sencilla antes de cargarlo en la tarjeta.

Teniendo en cuenta todos los problemas vistos para conseguir que funcionara el sistema, puedo decir que he conseguido, sobre todo en la parte del sistema Linux, que he alcanzo un conocimiento aceptable, por lo que, a partir de aquí, puedo trazar un comienzo para futuros sistemas de bastante más complejidad usando redes neuronales y donde el objetivo sería su optimización al máximo haciendo uso de todo lo aprendido en el desarrollo de este trabajo y de nuevas tecnologías como las que expongo en la siguiente parte de futuras mejoras.

7. Futuras mejoras

Este trabajo se ha centrado principalmente en el aprendizaje del uso de sistemas SoC (System on Chip) sobre FPGAs utilizando los procesadores embebidos en estas y un sistema operativo Linux. Una vez que tenemos configurado todo lo necesario para trabajar sobre un sistema operativo Linux, se facilita de forma considerable el desarrollo de interfaces de usuario y por consiguiente el desarrollo de aplicaciones de laboratorio más manejables. También nos facilita el trabajar con mayor número de datos, el desarrollo de algoritmos en software más rápidos y eficientes, y el control del hardware disponible. Teniendo todo esto en cuenta, se puede trazar una línea de trabajo para futuras mejoras en donde el objetivo sería el desarrollo de un sistema completo de pruebas experimentales de diferentes algoritmos de aprendizaje.

Esta futura línea de trabajo se centraría en el desarrollo de un sistema de prueba de diferentes algoritmos de aprendizaje (Redes neuronales), donde se buscaría una mayor eficiencia en velocidad y en el trabajo con mayor número de datos. Para ello se plantean las siguientes mejoras que habría que añadir a este trabajo:

- Implementación de diferentes algoritmos de aprendizaje, pudiendo seleccionar desde la interface con el que se quiere trabajar.
- En el caso del reconocimiento de voz se plantea también el uso de coeficientes GTCC (Gammatone Cepstral Coefficients), ya que para este trabajo se han utilizado los coeficientes MFCC (Mel Frecuency Cepstral Coefficients).
- También se plantea el buscar otras áreas aparte del reconocimiento de voz con las que se pueda trabajar, como puede ser el reconocimiento de imágenes, el uso de estos algoritmos en sistemas de RF para su uso en filtros adaptativos, en umbrales adaptativos para filtros CFAR utilizados en sistemas RADAR, etc.
- Como elemento importante de mejora, sería el almacenamiento de la red neuronal entrenada (poder guardar los pesos de las diferentes capas del algoritmo), para de esta forma no tener que entrenar el sistema cada vez que se enciende. Al trabajar sobre un sistema operativo Linux, el almacenamiento de grandes cantidades de datos se puede realizar de forma sencilla.

Como última línea de trabajo para futuras mejoras, y sobre la que más interés pongo, sería el buscar acelerar estos algoritmos de aprendizaje. Aunque el trabajar sobre un sistema procesador ARM de dos núcleos ayuda bastante y no vamos a tener problema con pequeños algoritmos, si empezamos a trabajar con algoritmos más complicados y con mayor número de datos, habría que plantear soluciones hardware para acelerar estos algoritmos. La implementación de procesado de señal mediante módulos hardware implementados en la FPGA nos puede ayudar bastante, pero esta implementación se puede complicar bastante aparte de que si trabajamos con FPGAs de gama baja (de precio asumible) es fácil que se supere su capacidad en elementos lógicos disponibles.

Como solución para conseguir acelerar estos algoritmos con sistemas hardware se plantea el uso de aceleradores gráficos de los que existen diferentes integrados en el mercado diseñados

para tal propósito. Pero si ya tenemos una placa de desarrollo con una FPGA, la solución sería el uso de OpenCL sobre esta para la implementación de estos aceleradores sobre ella.

El estándar OpenCL [referencia] es el primer modelo de programación unificado, de libre uso, para acelerar algoritmos en sistemas heterogéneos. OpenCL permite el uso de un lenguaje de programación basado en C para desarrollar código a través de diferentes plataformas, tales como unidades de procesamiento central (CPUs), unidades de procesamiento gráfico (GPU), procesadores de señal (DSP) y FPGAs.

OpenCL es un modelo de programación para ingenieros de software y una metodología para arquitectos de sistemas. Se basa en la norma ANSI C (C99) con extensiones para extraer el paralelismo. OpenCL también incluye una interfaz de programa de aplicación (API) para que el host se comunique con el acelerador de hardware, tradicionalmente a través de PCI Express, o un kernel para comunicarse con otro sin interacción de host. Además de esto, ofrece, como extensión, una API de canal de E/S para transmitir datos en un kernel directamente desde una interfaz de E/S de transmisión, tal como 10Gb Ethernet. Un beneficio clave de OpenCL es que es un estándar portátil, abierto, de libre uso.

En el modelo OpenCL, el usuario programa las tareas a las colas de comandos, de las cuales hay al menos una para cada dispositivo. El tiempo de ejecución de OpenCL rompe las tareas paralelas de datos en pedazos y las envía a los elementos de procesamiento del dispositivo. Este es el método para que un host se comunique con cualquier acelerador de hardware.

Altera nos proporciona un entorno de trabajo, el Intel FPGA SDK para OpenCL que se ajusta al estándar OpenCL 1.0.

Por último, mencionar que el uso el OpenCL sobre FPGAs ha demostrado sus beneficios trabajando con redes neuronales, por lo que como principal línea de trabajo futura el aprender a utilizar este tipo de tecnología.

8. Referencias

[1] DE1-SoC User Manual [Documentación de Terasic].

[2] Multi-touch LCD Module [Documentación de Terasic].

[3] Introduction to Cyclone V Hard Processor System [Documentación de Altera 54001].

[4] Cyclone V Hard Processor System Technical Reference Manual [Documentación de Altera CV 5v4].

[5] Mapping HPS IP Peripheral Signals to the FPGA Interface [Documentación de Altera AN706].

[6] Floating-Point IP Cores User Guide [Documentación de Altera].

[7] HPS SoC Boot Guide - Cyclone V SoC Development Kit [Documentación de Altera AN709].

[8] Arria 10 SoC Boot User Guide [Documentación de Altera].

[9] Booting and Configuration Introduction [Documentación de Altera CV 5400a].

[10] http://www.denx.de/wiki/publish/U-Bootdoc/U-Bootdoc.pdf

[11] Linux Device Drivers, Third Edition, Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman.

[12] The Linux Kernel Module Programming Guide, Peter Jay Salzman.

[13] Video and Image Processing Suite User Guide [Documentación de Altera].

[14] SoC FPGA Embedded Development Suite User Guide [Documentación de Altera].

[15] NIOS II Processor Documentation. https://www.altera.com/products/processors/support.html

[16] http://doc.qt.io/qt-4.8/index.html

[17] https://rocketboards.org/foswiki/Documentation/WebHome

[18] https://releases.linaro.org/components/toolchain/binaries/latest-5/arm-linux-gnueabihf/