GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

MEDIDA DE LA CAPACIDAD DE CÓMPUTO DE LA TARJETA ULTRASCALE DE XILINX

Autor: Manuel Álvarez-Requejo Heredero

Director: Samuel Aparicio Castrillo

Madrid

Agosto de 2019

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

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

El autor D. MANUEL ÁLVAREZ-REQUEJO HEREDERO

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: MEDIDA DE LA CAPACIDAD DE CÓMPUTO DE LA TARJETA ULTRASCALE DE XILINX que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

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

3º. Condiciones de la cesión y acceso

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

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

4º. Derechos del autor.

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

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

5º. Deberes del autor.

El autor se compromete a:

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

intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

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

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

Madrid, a 20 de Agosto de 2019

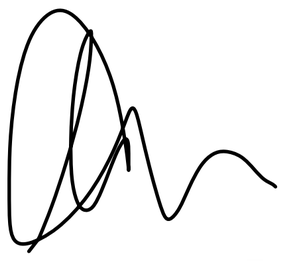
ACEPTA

Fdo.....

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

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título Medida de la capacidad de cómputo de la tarjeta Ultrascale de Xilinx en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2018/2019 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Manuel Álvarez-Requejo Heredero Fecha: 20 / 08 / 19



Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Samuel Aparicio Castrillo Fecha: 20 / 08 / 19

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

MEDIDA DE LA CAPACIDAD DE CÓMPUTO DE LA TARJETA ULTRASCALE DE XILINX

Autor: Manuel Álvarez-Requejo Heredero

Director: Samuel Aparicio Castrillo

Madrid

Agosto de 2019

MEDIDA DE LA CAPACIDAD DE CÓMPUTO DE LA TARJETA ULTRASCALE DE XILINX

Autor: Álvarez-Requejo Heredero, Manuel.

Director: Aparicio Castrillo, Samuel.

Entidad Colaboradora: SENER Aeroespacial.

RESUMEN DEL PROYECTO

Introducción

Desde la aparición de los ordenadores y con el desarrollo del mundo informático, la tecnología tiende a la velocidad. Las soluciones de procesado que ofrece el mercado son más rápidas año tras año. Cada generación busca un procesador más potente y rápido que el anterior para no quedarse atrás en la competición por el mercado de los procesadores.

Dado el extenso uso de soluciones de procesado, el mercado ofrece infinidad de opciones. Desde los electrodomésticos más básicos hasta los ordenadores o sistemas más complejos llevan algún tipo de herramienta de procesado de datos. Es por esto que se pueden encontrar procesadores muy básicos a precios bajísimos o auténticas bestias mucho más caras pero capaces de procesar millones de datos por segundo.

Ante este panorama, encontrar la solución ideal, ajustando precios, tamaños y potencias puede ser realmente complicado. Este tema es excesivamente amplio, por lo que se repasará los tipos de soluciones de procesado que se ofrecen hoy día por encima, deteniéndose un poco más en las piezas de hardware (HW) que son más de interés para la realización del proyecto. Estos procesadores son, desde un nivel de abstracción absoluto, en el que se trabaja con varios niveles de abstracción y el software se encarga de realizar todas las operaciones a nivel bit, hasta el nivel de abstracción nulo en el que el propio fabricante del procesador trabaja las vías por las que circularan los bits. Los tipos de procesadores son los siguientes: procesadores de uso general (GPP), procesadores gráficos (GPU), procesadores digitales de señales (DSP), *Field-Programmable Gate-Array* o FPGA y *Application-Specific Integrated Circuit* o ASIC. Los dos primeros son software puro, con muchos niveles de abstracción mientras que los dos últimos trabajan sobre el hardware a nivel bit.

GPP. Estos se encuentran en prácticamente todo lo que nos rodea. Su nivel de abstracción es absoluto ya que requieren de un sistema operativo para trabajar, pero esto facilita mucho la programación del mismo. Los lenguajes utilizados para este tipo de procesadores serían C, C++, Java, Python etc. Algunos de los ejemplos más famosos de este tipo de procesadores serían los Intel o los AMD.

GPU. Estas unidades, al igual que las anteriores, son relativamente sencillas de programar y trabajan a niveles de abstracción altos, aunque menos que los GPP. Se

caracterizan por tener miles de núcleos capaces de realizar tareas en paralelo, esto es, que pueden hacer varias tareas simultáneamente (NVIDIA, s.f.). En el procesado de imágenes, por ejemplo, en el que hay que procesar cada píxel individualmente, este tipo de solución es el idóneo. También se usa para procesar datos vectoriales, aunque no sean aplicaciones gráficas. Estas tarjetas se programan con lenguajes como OpenCL y algunos ejemplos serían las NVIDIA, AMD o ARM.

En tercer lugar, los DSP son procesadores muy sencillos que se encargan sobre todo de realizar operaciones a señales que serían demasiado complejas para hacerlas a nivel hardware, y por tanto una FPGA (que es la herramienta que más se suele utilizar para el procesado de señales) sería demasiado complicada de programar. Estos procesadores no se suelen utilizar a nivel individual, de hecho, las FPGA suelen incluir algún bloque DSP.

Hasta aquí, se trabajaba con algún tipo de abstracción, a nivel de software, con programación relativamente sencilla. Los dos elementos siguientes eliminan la abstracción prácticamente en su totalidad y trabajan a nivel bit, por lo que para utilizarlos no solo es necesario saber programarlos, sino además entender algo de electrónica digital. Esto complica mucho su programación.

Las FPGAs, son tarjetas compuestas por una matriz de bloques conectados entre sí. Estos bloques incluyen Flip Flops y puertas lógicas que realizaran alguna operación con la entrada para dar una salida específica. Al programar estas tarjetas, se abren algunas de las vías que hay entre los diferentes bloques de tal forma que a la entrada se le hacen las operaciones que el programador decida y así, obtener la salida deseada. Estas tarjetas se programan con lenguaje descriptivo, llamado RTL, como VHDL o Verilog y los mayores fabricantes del mundo son Altera y Xilinx.

Por último, los ASIC son el nivel de HW más puro. Ni siquiera utilizan software para la programación, sino que directamente se sueldan los elementos sobre la placa para

Figura 1. Procesadores de Intel. (Fuente: Difference Between)

Figura 2. GPU de Nvidia. (Fuente: NVIDIA)

Figura 3. DSP de ARM. (Fuente: Difference Between)

Figura 4. FPGA de Xilinx. (Fuente: Xilinx)

Figura 5. Circuito integrado. (Fuente: NewsReck)

obtener los resultados deseados. Los circuitos integrados tienen muy poca flexibilidad ya que no pueden ser reprogramados y están hechos para realizar una función específica.

Como se menciona más arriba, los tres primeros elementos son elementos que funcionan con software. Esto significa que requieren de niveles de abstracción que realizarán las tareas por ellos y se encargaran de manejar la parte física del circuito. Esto, por supuesto, facilita mucho las cosas ya que para programarlos no se requiere de conocimientos de electrónica, únicamente de programación. Estos procesadores son tremendamente flexibles dada su facilidad de programación, y pueden dedicarse a tareas muy variadas. Dada la baja especificación de los elementos, también ven reducidos sus costes respecto a los dos últimos. Sin embargo, en termas de eficiencia no trabajan tan bien. El consumo de energía requerido por el sistema operativo es mucho más grande que el que se requeriría para trabajar a nivel bit, y al no estar especializados en alguna tarea, pueden realizar la que sea pero consumiendo más recursos.

Por otro lado, las dos últimas trabajan sobre el HW. Esto complica mucho su programación, y cuanto más compleja sea la aplicación del circuito, más aumenta la dificultad de su programación. Es por esto que estos chips se utilizan para aplicaciones muy específicas. Esto tiene sus ventajas y sus desventajas. Si bien la flexibilidad se ve reducida de forma muy grande, la eficiencia de estas tarjetas es mucho más altas que la de las que trabajan con otros niveles de abstracción. Por último, siendo estos elementos tan complejos, son mucho más caros que los procesadores de uso genérico. En la figura 6, hay una gráfica que describe perfectamente la eficiencia frente a la flexibilidad de cada uno de estos procesadores.

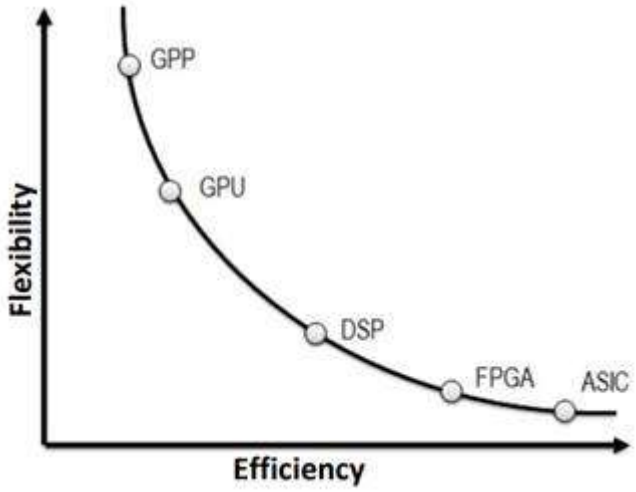


Figura 6. Flexibilidad frente a eficiencia de los diferentes procesadores. (Fuente: propia).

Por tanto, los GPP son mucho más sencillos de programar, ya que se trabaja directamente en SW, mientras que en FPGA se complica mucho la programación, ya que se necesitan conocimientos de electrónica digital y se trabaja a nivel bit. Además, las CPUs trabajan de forma secuencial mientras que las FPGAs trabajan en paralelo. Es por esto que las FPGAs ahorran mucho tiempo de ejecución, aunque este tiempo hay que invertirlo luego en programación. Otra ventaja de las FPGAs es que permiten trabajar a frecuencias inferiores, del orden de los MHz.

Por último, existe otro tipo de HW llamado System on a chip (SoC). Éstos combinan una CPU, generalmente con RAM o algún otro elemento, y acceso a ciertos periféricos dentro del mismo chip, encapsulado todo de forma muy comprimida. Este tipo de chips tienen la ventaja de que son más baratos de fabricar, debido a que no hay que hacer un ensamblaje sino que los elementos se sueldan directamente a la placa, y además la reducida distancia de los elementos implica una reducción de los tiempos de ejecución, ya que la distancia que tienen que recorrer los datos es mucho más baja. Los principales inconvenientes de estos elementos son que si se rompe aunque sea solo uno de sus componentes, éste no se puede cambiar y hay que cambiar el SoC entero. Además, el reducido espacio limita el tamaño de las piezas de HW que se le pueden introducir.

El proyecto tiene dos SoCs, fabricados por Xilinx. Éstos tienen en el propio chip la CPU, memoria RAM (aunque admite memoria externa) y acceso de alta velocidad a la FPGA entre otras cosas. Esta comunicación tan rápida con la FPGA permite realizar operaciones de HW y de SW del mismo programa sin sufrir grandes incrementos del tiempo empleado en ejecutar el programa en cuestión. De ahora en adelante, cuando se hable de "Processing System" (PS), es la parte que se ejecuta en el GPP, y la "Programmable Logic" (PL), la parte que se ejecuta en la FPGA. Más en concreto, el HW que se utiliza será:

- Xilinx Zynq RFSoC UltraScale+, tarjeta ZCU111.
- Xilinx Zynq 7000Series, tarjeta 7020.

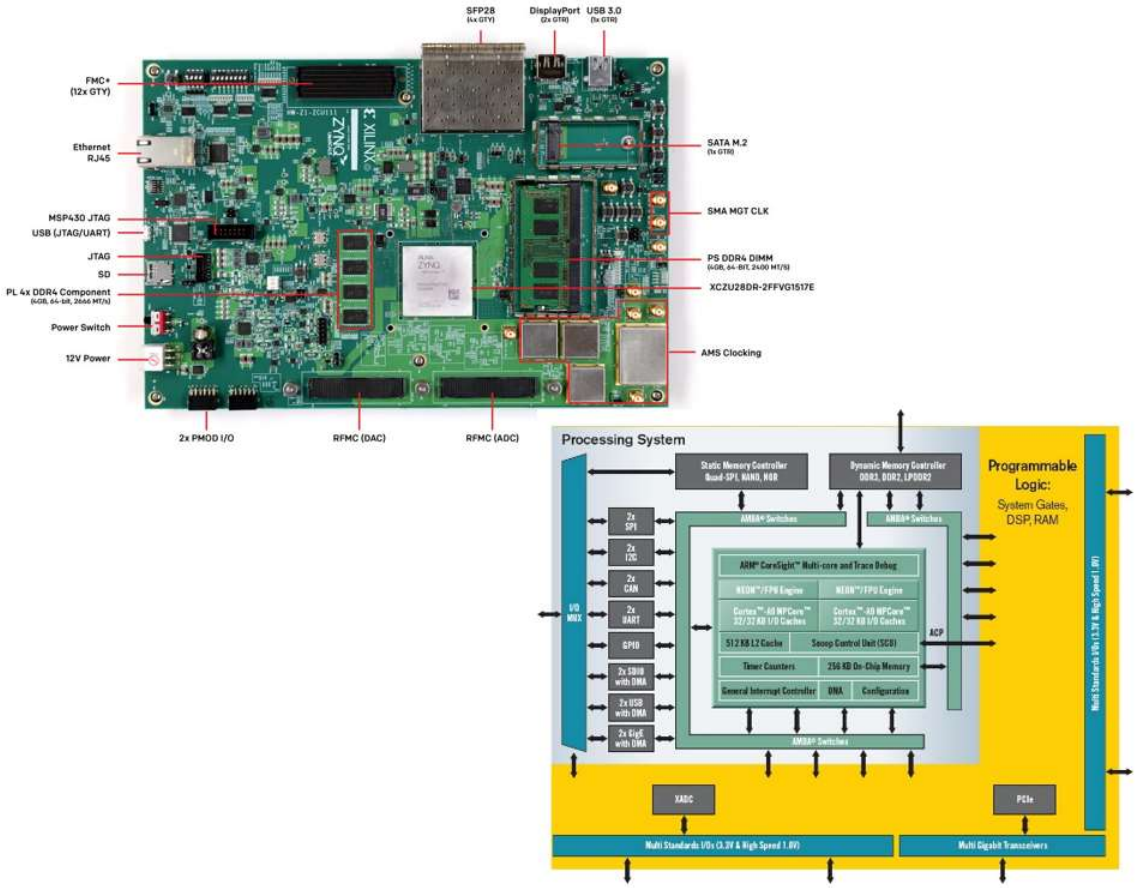


Figura 6. A la izquierda, Zynq US+ RFSoC, a la derecha, esquema de la familia Zynq 7000Series (Fuente: Xilinx)

La primera tarjeta consta, por la parte de SW, de cuatro procesadores ARM Cortex-A53 y dos procesadores en tiempo real ARM R5. La segunda, únicamente tiene dos procesadores ARM Cortex-A9.

Metodología

Para probar la tarjeta en cuestión, se ha ejecutado un programa en la misma. Dicho programa ejecuta dos funciones: la primera genera un enventanado y la segunda genera los coeficientes de un filtro de tipo FIR con el enventanado de la función anterior.

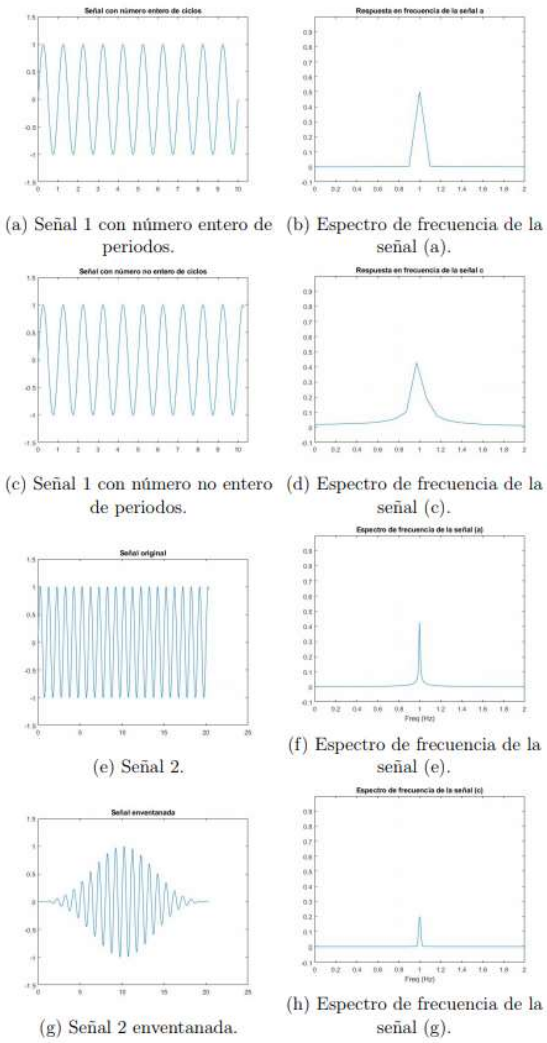
Para la programación de estos programas se ha utilizado una librería de procesamiento digital de señales llamada *liquid-dsp* creada por Joseph D. Gaeddert. Para su ejecución se ha generado la imagen de un kernel de Linux que se ha instalado en la tarjeta. El programa se ha realizado mediante el IDE de Eclipse y ejecutado con el compilador cruzado aarch64-linux-gnu-gcc. Por último, la comunicación con la tarjeta ha sido Ethernet y se ha hecho mediante Linaro.

En primer lugar, se va a hablar del enventanado de una señal. Cuando se realizar el procesamiento digital de una señal, esta señal debe estar acotada en el tiempo y tener un número finito de puntos, es decir, estar muestreada. Si fuera posible coger siempre un número entero de periodos de la señal en todas sus componentes en frecuencia, el problema acabaría aquí, pero por desgracia eso casi nunca es posible. Un programa que realice procesamiento de señales, lo que hace es conectar los dos extremos de la señal en un tiempo finito, formando una circunferencia. Si en el punto de unión no coinciden los valores, es decir, si el primer punto del dominio del tiempo muestreado es diferente que el último punto del dominio del tiempo muestreado, aparece una discontinuidad en esa señal que se traduce en un espectro de frecuencia de la señal falso. Esta discontinuidad lo que provoca es que aparezcan frecuencias que la señal realmente no lleva consigo. Las ventanas funcionan atenuando los bordes de la señal en cuestión, de tal forma que estas frecuencias virtuales tengan la menor relevancia posible. (National Instruments, 2019).

En la figura 7, se puede apreciar como es cambia la respuesta en frecuencia de una señal con un número entero de periodos y un número no entero de periodos, y abajo se puede ver las diferencias de su espectro de frecuencia si se le aplica una ventana. Como se puede apreciar, la respuesta en frecuencia de la figura 7-b es mucho más nítida que la de la figura 7-d, a pesar de que ambas son la misma señal. En cuanto a la señal 2, las figuras f y h no se aprecia demasiado bien el valor del enventanado, sin embargo sí que se puede ver como la 7-h tiene una amplitud menor aunque el punto en el que pasa a valer cero está mucho más claro que en la figura 7-f. Si la señal tuviera más muestras y más frecuencias el uso de la ventana sería más claro y necesario, pero en este caso era solo para poner un ejemplo.

Hay muchos tipos de ventanas útiles en situaciones diferentes. Si la señal tiene muchas frecuencias que están juntas, por ejemplo, se utilizarán ventanas con lóbulos centrales muy estrechos mientras que si es más importante conocer la amplitud de la señal que la posición exacta de las frecuencias donde se encuentra dicha amplitud, se usará una

ventana con un lóbulo principal más ancho. Si las frecuencias de interés están muy alejadas entre sí, habrá que coger ventanas con lóbulos laterales altos mientras que si aparece mucho ruido en los extremos, se utilizarán lóbulos laterales más bajos.



(a) Señal 1 con número entero de periodos.
(b) Espectro de frecuencia de la señal (a).
(c) Señal 1 con número no entero de periodos.
(d) Espectro de frecuencia de la señal (c).
(e) Señal 2.
(f) Espectro de frecuencia de la señal (e).
(g) Señal 2 enventanada.
(h) Espectro de frecuencia de la señal (g).

Figura 7. Distintas señales con sus respectivas respuestas en frecuencia. (Fuente: Propia).

En la figura 8, se pueden ver algunas de las ventanas más utilizadas.

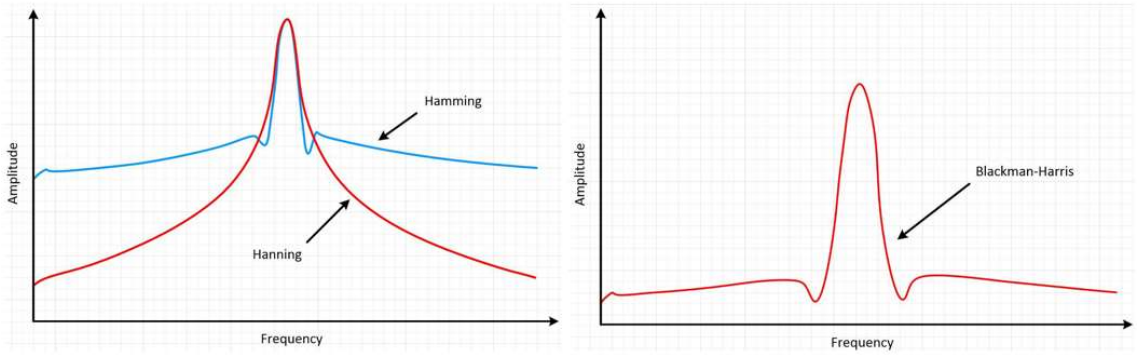


Figura 8. Ventanas en el dominio de la frecuencia. (Fuente: National Instruments)

En este proyecto, las ventanas se han formado a partir de dos fuentes. En primer lugar la propia librería que s eha utilizado para el procesamiento digital de señales, *liquid-dsp* de Joseph D. Gaeddart, contiene funciones para la generación de algunas ventanas. El resto de las ventanas se han generado atendiendo a la ecuación que las define, que se ha obtenido de Matlab.

Los filtros FIR son un tipo de filtros digitales. Son filtros de respuesta al impulso finita (Finite Impulse Response en inglés) y se caracterizan por no tener realimentación y, por tanto, siempre alcanzan el reposo en un periodo finito de tiempo y siempre son estables (en contraposición con los Infinite Impulse Response IIR). La gran mayoría de herramientas de procesamiento de señales traen un filtro FIR, ya que es muy sencillo de programar e implementar. Otra ventaja de estos filtros es que permiten obtener una fase perfectamente lineal, cosa que solo se puede conseguir con aproximaciones en otros filtros y además complica el cálculo del mismo en gran medida. La mayor desventaja que ofrece este tipo de filtros, es que por no tener realimentación, los tiempos de ejecución y los requerimientos del sistema aumentan mucho con filtros de órdenes altos. (Oshana, 2012).

Para el proyecto, este filtro se ha aplicado igual que la función fir1 de Matlab, de la toolbox de procesamiento de señales. En primer lugar, se genera el diagrama de bode que debería tener la respuesta del filtro una vez montado. A este bode se le aplica la transformada inversa de Fourier (IFFT) y de estos datos se obtendrán los coeficientes del filtro, que, multiplicados por el valor de la ventana correspondiente deben dar el coeficiente por el que se debe multiplicar la señal para obtener el resultado.

La puesta en marcha de la tarjeta es uno de los mayores retos de este proyecto. En primer lugar, se tuvo que generar una imagen de un kernel de Linux que iría instalado en la tarjeta. Este kernel se generó mediante el uso de una herramienta aportada por Xilinx para trabajar con sus productos: *Petalinux*. Petalinux es una herramienta de trabajo para personalizar, construir y ejecutar soluciones embebidas sobre sistemas de Xilinx. (Xilinx, s.f.). Para programar la tarjeta se ha utilizado lenguaje C sobre el IDE de Eclipse en una máquina virtual con sistema operativo Ubuntu 64. Se ha hecho compilación cruzada del software con la herramienta de Linux aarch64-linux-gnu-gcc, sobre la que se ha tenido que instalar la librería que se ha mencionado antes, *liquid-dsp* de Joseph D. Geaddert. La comunicación con la tarjeta se ha hecho mediante Ethernet utilizando la herramienta Linaro.

Resultados

Los resultados de este proyecto, hablando en términos cualitativos, se pueden intentar predecir. La tarjeta Zynq 7020 es una tarjeta con procesadores peores, y solo tiene dos mientras que la UltraScale+ tiene cuatro procesadores de más potentes además de dos procesadores en tiempo real, por esto cabe esperar que sea mucho más rápida la segunda a la hora de ejecutar cualquier programa. A estas dos les debería ganar el ordenador industrial. Este último no se sirve de un sistema embebido con un SoC sino que utiliza una CPU real con un procesador mucho más potente, por tanto este será mucho más rápido.

Los resultados se pueden observar en las tablas y el gráfico que hay a continuación:

Tarjeta		Zynq RFSoC											
Número de eventos por itercaión		100											
Orden del filtro		512				1024				4096			
Tipo de ventana		Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev
Tiempo total	Media	6,7573	6,7583	6,7671	6,7596	6,7781	6,7784	6,8001	6,8377	6,8944	6,9011	6,9963	7,3585
Tiempo por evento	Media	67,5607	67,5595	67,6732	67,6379	67,8127	67,7821	67,9787	68,3878	68,9470	68,9730	69,9433	73,5499

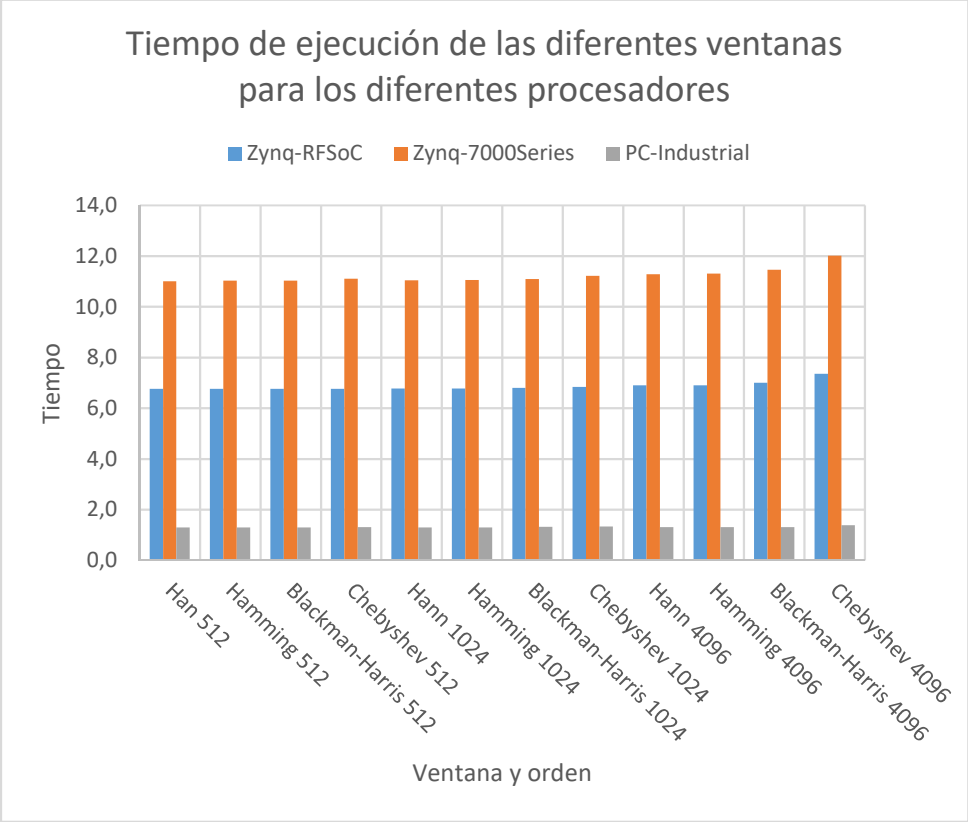
Tabla 1. Resultado de tiempos de la Zynq UltraScale+ RFSoC (ZCU111)

Tarjeta		Zynq 7000-Series											
Número de eventos por itercaión		100											
Orden del filtro		512				1024				4096			
Tipo de ventana		Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev
Tiempo total	Media	11,0060	11,0249	11,0330	11,1064	11,0462	11,0571	11,0939	11,2244	11,2804	11,3125	11,4598	12,0093
Tiempo por evento	Media	110,0555	110,3255	110,2559	111,0425	110,4421	110,5379	110,9475	112,2694	112,8110	113,0156	114,5423	120,0858

Tabla 2. Resultado de tiempos de la Zynq 7000Series (7020)

Tarjeta		Ordenador industrial - Intel core i7											
Número de eventos por itercaión		100											
Orden del filtro		512				1024				4096			
Tipo de ventana		Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev
Tiempo total	Media	1,2981	1,2947	1,2980	1,3067	1,2901	1,2928	1,3133	1,3251	1,3042	1,3087	1,3076	1,3874
Tiempo por evento	Media	13,0162	12,9349	12,9911	13,0549	12,9124	12,9553	13,1412	13,2209	13,0445	13,0887	13,1192	13,8947

Tabla 3. Resultado de tiempos del ordenador industrial (Intel Core i7 3770)



Gráfica 1. Gráfica de tiempos de ejecución de las tres tarjetas.

De media, la velocidad a la que procesa datos la Zynq RFSoC es del 61,5% de la velocidad de la 7020 y unas cinco veces más lenta que un ordenador industrial, con unos tiempos de ejecución entre los 6,75 y 7,35 segundos para ejecutar 100 iteraciones del programa, a unos 67,5 o 73,5 milisegundos por iteración.

La Zynq 7020, como era de esperar, es la más lenta de todas, siendo prácticamente el doble de lenta que la RFSoC y diez veces más que el ordenador industrial. Los tiempos de ejecución están entre los 11 y los 12 segundos para cien iteraciones, a unos 110 o 120 milisegundos por iteración.

Conclusiones

Efectivamente, los resultados no resultan sorprendentes. Como se había predicho anteriormente, la más lenta de todas las tarjetas es la Zynq 7020, ya que los procesadores que lleva son menos y menos potentes, mientras que la RFSoC lleva cuatro ARM Cortex-A53 además de dos procesadores en tiempo real ARM R-5, la 7020 lleva solo dos ARM Cortex-A9.

Los tiempos de ejecución del ordenador son mucho más rápidos que los de cualquiera de las tarjetas por tener un procesador mucho más potente. Mientras que los otros sistemas tienen sistemas embebidos, que buscan ocupar poco espacio, el ordenador no pretende economizar espacio sino costes. Sin límites de tamaño se pueden conseguir procesadores mucho más potentes como es el Intel Core i7 que lleva el ordenador en el que se han ejecutado los datos.

El futuro de los procesadores está en la velocidad. Ser capaz de procesar muchos datos cada vez en menos tiempo es la meta de cualquier sistema informático, y por eso todas las evoluciones de los procesadores se hacen con ese fin: reducir tiempos, costes y tamaños. De cara al programador, conocer la velocidad y la capacidad que tiene la herramienta con la que trabaja es esencial. Igual que un soldador debe conocer los metales que va a unir, un programador debe saber qué capacidad tiene su procesador. En este estudio se ha visto que los tiempos de ejecución son adecuados para el proyecto en cuestión, y se seguirá adelante con la tarjeta UltraScale+ de Xilinx.

Referencias

Gomar, J. (30 de Septiembre de 2018). Profesional Review. Obtenido de https://www.profesionalreview.com/2018/09/30/que-es-un-soc/

National Instruments. (24 de Mayo de 2019). National Instruments. Obtenido de https://www.ni.com/es-es/innovations/white-papers/06/understanding-ffts-and-windowing.html

NVIDIA. (s.f.). NVIDIA. Obtenido de nvidia.com: https://la.nvidia.com/object/what-is-gpu-computing-la.html

Oshana, R. (2012). DSP For Embedded and Real-Time Systems. Newnes.

Xilinx. (s.f.). Petalinux Tools. Obtenido de https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html

XILINX ULTRASCALE CARD PROCESSING CAPACITY MEASURMENT

Introduction

Since the apparition of the first computers and with the computing world development, technology tends to go faster. All processing solutions present in the market are quicker every year. Each generation looks for a faster and more powerful processor to keep the race with the competition for the processing market.

There are many solutions for processing nowadays. From the most basic electrical appliances to the computers and most complex systems, all of them have some kind of processing tool. This is why there can be found very basic processors with not too good specifications for a cheap price or real processing beasts able to process millions of data per second at much higher prices.

Against that background, finding an ideal solution, adjusting price, size and power can be really complex. This subject is excessively extensive, so in this project the different types of processing solutions will be summarized, deepening more in the hardware gadgets that are of a bigger interest for it.

These processors are from an absolute abstraction level, where there are several abstraction layers and the software is in charge of making all the bit-level operations, to a null level of abstraction, where there are no layers and the programmer himself must work on the copper or the silicon to create a physical program where the electrons will move and obtain the wanted result. The different types of processors are the ones that follow: General Purpose Processor (GPP), Graphic Processing Unit (GPU), Digital Signal Processor (DSP), Field-Programmable Gate-Array (FPGA) and Application-Specific Integrated Circuit (ASIC). The first two are pure software, whit many abstraction layers and very easily programmed, while the last two work with hardware at bit level, harder to program and a lot more expensive.

GPP. These can be found in almost every gadget you may think of, as they are very flexible, cheap and relatively easy to program. Their abstraction level is very big, they require an operating system to work which will translate the programing language to hardware functionalities, this makes them very flexible and easy to program. Some of the programing languages they use are C, C++, Python, Java etc. And some of the most famous examples of these processors are Intel or AMD.

GPU. These units, just like the ones before, are relatively easy to program and work at very high abstraction levels, even though less than the GPPs. Their most important feature is that, while GPPs may do four, even eight tasks at once, depending on the number of cores they have, and work in a sequential way, GPUs have many cores and are able to work in parallel, developing many tasks at the same time (NVIDIA, n.d.). In image processing, for example, images are treated like a giant matrix where every pixel is processed individually. GPUs are very useful in these situations. That is why computers that work a lot with imaging like videogames or cinema producers need very powerful GPUs. They are also used for some non-graphical processes such as vector data

processing. These cards are programmed with languages like OpenCL and some examples could be NVIDIA, AMD or ARM.

In third place, DPSs are very simple processors that mostly make operations with signals that would be too hard to operate and program at hardware level (FPGA, which is the most common tool for signal processing). These processors are not usually used on their own, but integrated in bigger systems with better processors. In fact, FPGAs usually have some DSP blocks.

These three processors are digital, and work with some kind of abstraction at software level and they are relatively easy to program. The next two are hardware, they don´t have abstraction, or they have very little, and work at bit-level. To use them, not only do you need to know programing but also digital electronics, so they are much harder to program.

FPGAs are cards made up by a matrix of connected logic blocks. These block include Flip Flops and logic gates that will make some operation with the entrance to obtain the desired exit. Programing one of these cards consists on opening or closing the connections between the different blocks, so that the entrance to the circuit goes through some different blocks that will make the pertinent operations to obtain the exit that the programmer needs. These cards are programmed using hardware description language (HDL), called RTL, such as VHDL or Verilog and the biggest producers of these products are Xilinx and Altera.

Lastly, ASIC are the purest hardware level you can program in. They don’t even use software, like FPGAs used HDL, but the program is hardware wielded onto the card to manipulate the entrance as required and obtain an exit. Integrated circuits have very little flexibility, as once wielded, the function of the circuit cannot be changed, reprogramed, they are made to carry out a specific task. They are very hard to program,

Figure 1. Intel Processors. (Source: Difference Between)

Figure 2. Nvidia GPU. (Source: NVIDIA)

Figure 3. ARM DSP. (Source: Difference Between)

Figure 4. Xilinx FPGA. (Source: Xilinx)

Figure 5. Integrated Circuit. (Source: NewsReck)

big functions require a lot of space and ability, but being so specific, they are optimized and consume very little time.

As mentioned previously, the first three elements are elements that work with software. This means that, to work, they require different levels of abstraction that will realize the tasks for them and will be in charge of managing the physical part of the circuit. This makes things considerably easy, as the programmer does not have to know anything about electronics, only learn a language that the operating system will translate to electronics later on. They can be programed to develop many tasks in an easy way, which makes them very flexible and being as unspecific as their elements are, the costs are reduced considerably, in programing time and ability and in money. However, in terms of efficiency they are not that good. Having such a high level of abstraction requires a lot of power, these processors consume a lot, and the operating system needs its time to make all the "translations" from the programing language to the "electronics" language, so they are not as fast in execution either.

On the other hand, the las two work on hardware. This makes its programing much harder and it gets a lot harder as the complexity of the application for the circuit increases. This happens because to program these processors, not only you need to know about programing but also about digital electronics. This is why these chips are used only in very specific applications. This has some advantages and disadvantages. This cards will be very little flexible, for their complexity at the time of programing and some may not even be reprogrammable. On the other hand, they are a lot more efficient, as they don't have all these abstraction levels. Their running times are a lot lower than the running times of the chips mentioned previously and they consume a lot less power. Lastly, being as complex as they are, this chips are a lot more expensive than the ones with more general uses. In figure 6 there is a graphic that describes how flexibility and efficiency are related with these processors:

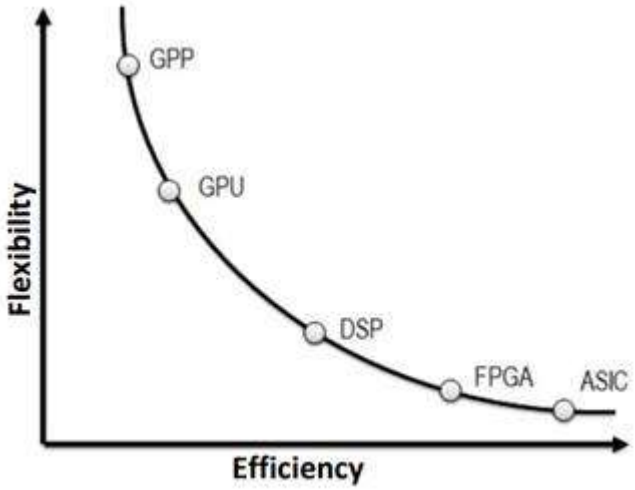


Figure 6. Flexibility versus efficiency of the different types of processors. (Own source).

GPPs are much easier to program, as the programing is directly on software and an abstraction layer is in charge of doing the hard things, the electronics. On the other side, FPGAs are much harder to program as the programmer needs to know about electronics and works at bit level. Furthermore, CPUs work sequentially while FPGAs work in parallel. For these reasons, FPGAs work a lot faster than GPPs, but that saved

time will have to be invested in programing it. In addition, FPGAs may work at lower frequencies, in the megahertz.

Finally, there is another type of hardware called System on Chip (SoC). SoCs combine, normally, a CPU with a RAM or other memory elements and access to some kinds of peripherals within the same chip, encapsulated in a very space-efficient way. These chips have the advantage that they are very cheap to make, as they don't need assembly in a fabrication process, but they are directly wielded on the board that is going to be used. Moreover, one of the processes that take most time is the transportation of information from one part of the card to another, these systems reduce the distance between elements, decreasing the communication time within the same chip. On the other hand, being all the elements wielded together as they are, if one of them breaks, it cannot be substituted directly but a much more expensive reparation is needed or changing the whole piece. Furthermore, as the space is very limited, not every processor can be added to these chips, and usually, the most powerful CPUs are the biggest, so the space limitation may considerably limit the processing speed.

This project will use two SoCs made by Xilinx. They include in the SoC the CPU, RAM memory (even though it accepts external memory) and high-speed access to the FPGA. This fast communication with the FPGA allows making hardware and software operations without having to spend big amounts of time on it within the same program. Xilinx calls the Processing System (PS) the software part (GPP) and the Programmable Logic (PL) the hardware part (FPGA). The communication between the PS and the PL es of about 16Gbps, and depending on the card, there are more or less connections. The used hardware is:

- Xilinx Zynq RFSoC UltraScale+, tarjeta ZCU111.

- Xilinx Zynq 7000Series, tarjeta 7020.

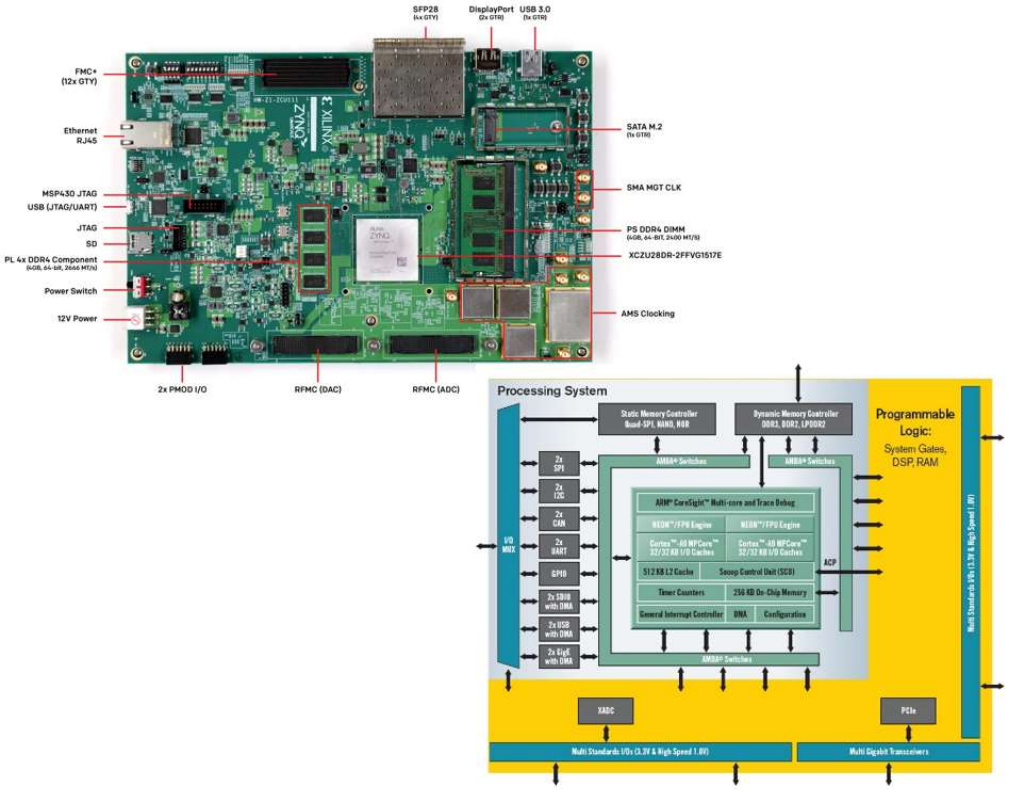


Figura 6. In the left, Zynq US+ RFSoC, in the right, block diagram of the Zynq 7000Series (Source: Xilinx)

In the first card, in the processing system there are four ARM Cortex-A53 processors and two real time ARM R5 processors. The second one has only two ARM Cortex-A9 processors. It can be predicted from this information that the UltraScale+ will be faster, as it has more processors than the 7000Series and the Cortex-A53 are more powerful than the Cortex-A9.

Methodology

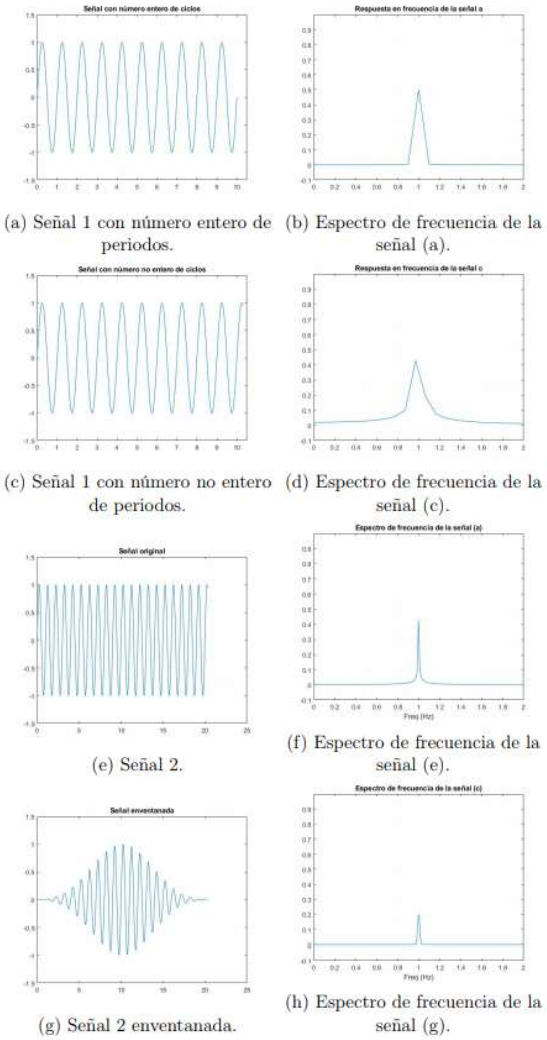
To test the objective card, a program has been executed in it. This program will run two different functions: the first one will obtain a window selected by the user and the second one will generate the coefficients for a FIR type digital filter, using the window generated in the previous function.

For the programming of these functions, a digital signal processing library has been used. This library, created by Joseph D. Geaddert, is called *liquid-dsp*, and includes some useful code lines to do mathematical operations to signals, like the Fast Fourier Transform. To run this program, a Linux kernel has been generated and installed in the card. The program has been written using the Eclipse IDE and executed using the cross-compiler aarch64-linux-gnu-gcc. Lastly, the communication with the card has been via Ethernet and using the tool Linaro.

In first place, the windowing of a signal will be explained. When a digital signal processing is going to take place, this signal must be during a finite period of time and it has to have a finite number of points, this means, it must be sampled. Were it always

possible to get an integer number of periods of the signal in all its frequency components, there would be no problem, but this case is usually impossible. A program that works processing signals, gets the part of the signal to be processed and joins its ends, making a circle. If this circle is not closed, this means, the first and the last point of the signal are different because there is not an integer number of periods, a discontinuity appears which may lead into the appearance of frequencies that are not really in the signal. To attenuate this problem, windows are used. Windows are coefficients that multiply every point of the signal, attenuating the ends so that the virtual frequencies have the least possible importance. (National Instruments, 2019).

In the figure 7, it can be seen how the frequency response of a signal changes when there is a whole number of periods and a decimal number of periods, and below the differences in their frequency spectrum if windowing is applied. In figure 7-b, the frequency response is much clearer than in figure 7-d, even though they are both the same signal. Concerning signal 2, in figures f and h look very alike, but kind of different. While h has only got one point above zero, the transition in f takes a little bit longer. When using the window, the gain is lower, but the main frequency is more clear. Where this a more complex example, it would not be so hard to appreciate, but using only a few periods and one frequency, the value of the window is harder to see.



(a) Señal 1 con número entero de periodos.
(b) Espectro de frecuencia de la señal (a).
(c) Señal 1 con número no entero de periodos.
(d) Espectro de frecuencia de la señal (c).
(e) Señal 2.
(f) Espectro de frecuencia de la señal (e).
(g) Señal 2 enventanada.
(h) Espectro de frecuencia de la señal (g).

Figure 7. Different signals with their respective frequency response. (Source: own).

There are many different types of useful windows in different situations. If the signal has many frequencies that are close together, for example, windows with very narrow central lobes will be used, whereas if it is more important to know the exact signal amplitude rather than the position of the frequencies where this amplitude is found, a window with a wider central lobe will be used. If the frequencies of interest are very far away one from another, windows with high lateral lobes will be needed whereas if there is a lot of noise in the ends, lower lateral lobes will be needed.

In figure 8, three of the most used windows can be seen, Hamming, Hann and Blackman-Harris windows. In the picture, the differences in their central and lateral lobes can be appreciated.

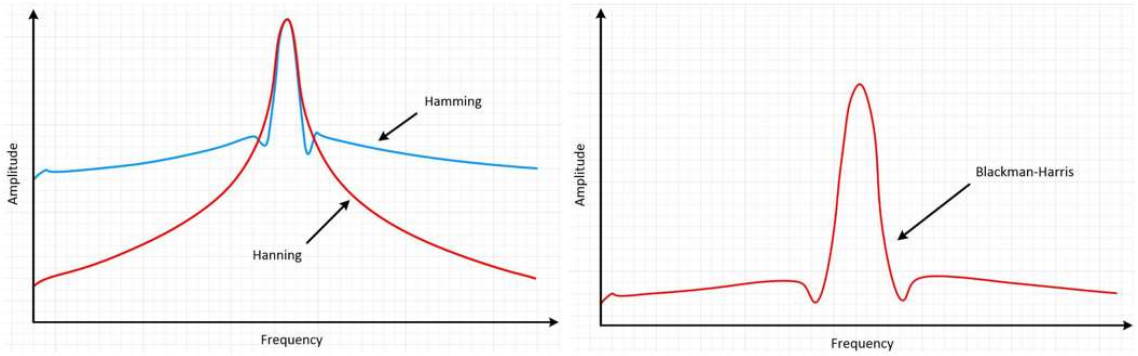


Figure 9. Windows in the frequency domain. (Source: National Instruments).

For the realization of this project, the windows will be obtained from two sources: in first place, the library *liquid-dsp* by Joseph d. Gaeddert includes some of the windows that will be used. On the other hand, windows not included in that library or those whose equation in the library is not the required from the project, will be obtained from Matlab. The idea is getting the same result from a C program that would be obtained from a Matlab program.

FIR filters are a type of digital filters. They are Finite Impulse Response filters and they're basic characteristic is that they don´t have feedback, so they always reach a steady state in a finite period of time and they are always stable (unlike the Infinite Impulse Response IIR). A large variety of signal processing signals have FIR filters pre-installed, as they are very easily programmed and implemented. These filters are also able to obtain a perfectly linear phase diagram, which in other filters can only be made by approximation and with very complex calculations. On the other hand, the biggest disadvantage of these filters is that, not having feedback, execution times and system requirements increase rapidly as the order of the filter grows. (Oshana, 2012).

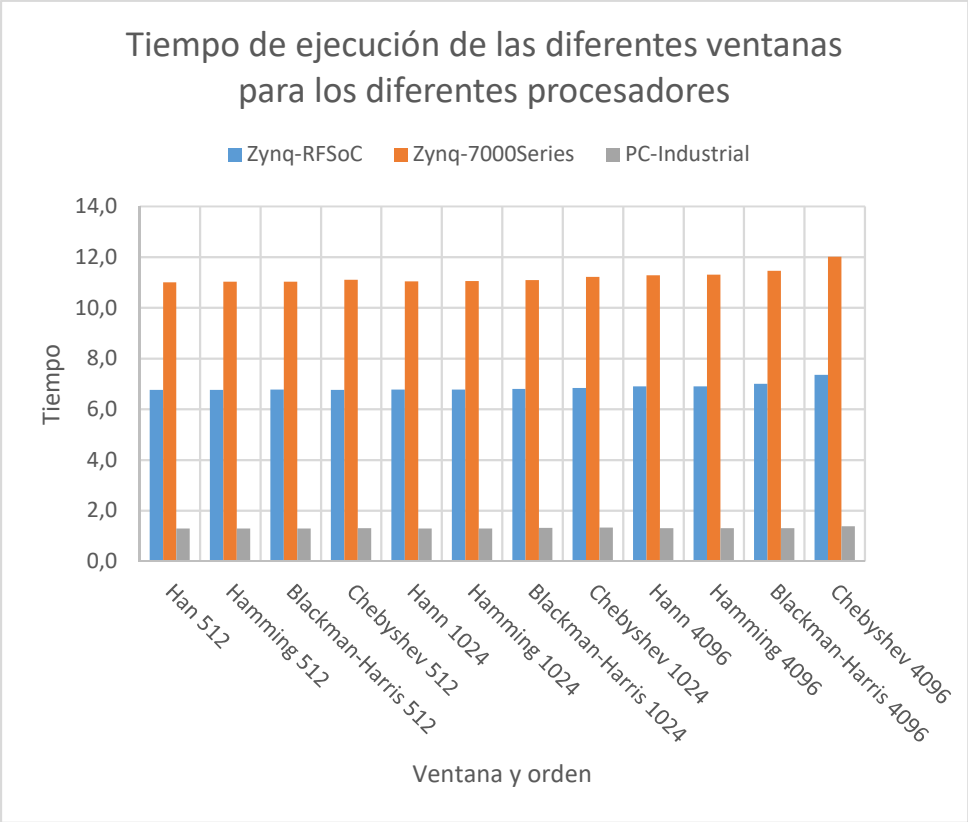
For the realization of tis project, this filter has been implemented like the function fir1 in Matlab, from the signal processing toolbox. In first place, the Bode diagram that the filter should present once finished is drawn manually. To this Bode diagram, the inverse fast Fourier transform (IFFT) is applied, to get the points in the time domain instead of the frequency domain, and the coefficients are obtained from this vector. Multiplying this vector by the window will give the parameters by which the signal must be multiplied to obtain the desired exit.

The start-up of the card is one of the biggest challenges in this project. In the first place, an image of a Linux based kernel that would be installed in the card had to be generated. This kernel was generated using a tool distributed by Xilinx to work with their products called *Petalinux.* Petalinux is a tool which gives the user ways to build, customize and deploy embedded Linux solutions on Xilinx processing systems (Xilinx, s.f.). To program the card, C language has been used on the Eclipse IDE in a virtual machine with operating system Ubuntu 64. The cross compilation of the software has been done with the Linux cross compiler aarch64-linux-gnu-gcc. This compiler has used the mentioned library *liquid-dsp* created by Joseph D. Gaeddart. The communication with the card has been via Ethernet using the tool Linaro.

Results

The results for this project, talking in qualitative terms, could have been easily predicted beforehand. The 7000Series card is a card with worse and les processors, it only has two, whereas the UltraScale+ has four processors that are more powerful and two real time processors. Therefore, it is expected that the RFSoC will be much faster than the Zynq 7020. These two should be widely surpassed by the industrial computer. This last system is only to get a reference to compare, but it has an Intel Core i7 processor, much bigger than a CPU installed in a SoC, and much more powerful.

The results can be observed in the tables and the graphic shown below:



Graphic 1. Execution times of the three processors.

Tarjeta		Zynq RFSoC											
Número de eventos por itercaión		100											
Orden del filtro		512				1024				4096			
Tipo de ventana		Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev
Tiempo total	Media	6,7573	6,7583	6,7671	6,7596	6,7781	6,7784	6,8001	6,8377	6,8944	6,9011	6,9963	7,3585
Tiempo por evento	Media	67,5607	67,5595	67,6732	67,6379	67,8127	67,7821	67,9787	68,3878	68,9470	68,9730	69,9433	73,5499

Tabla 4. Zynq UltraScale+ RFSoC (ZCU111) time results-

Tarjeta		Zynq 7000-Series											
Número de eventos por itercaión		100											
Orden del filtro		512				1024				4096			
Tipo de ventana		Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev
Tiempo total	Media	11,0060	11,0249	11,0330	11,1064	11,0462	11,0571	11,0939	11,2244	11,2804	11,3125	11,4598	12,0093
Tiempo por evento	Media	110,0555	110,3255	110,2559	111,0425	110,4421	110,5379	110,9475	112,2694	112,8110	113,0156	114,5423	120,0858

Tabla 5. Zynq 7000Series (7020) time results

Tarjeta		Ordenador industrial - Intel core i7											
Número de eventos por itercaión		100											
Orden del filtro		512				1024				4096			
Tipo de ventana		Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev
Tiempo total	Media	1,2981	1,2947	1,2980	1,3067	1,2901	1,2928	1,3133	1,3251	1,3042	1,3087	1,3076	1,3874
Tiempo por evento	Media	13,0162	12,9349	12,9911	13,0549	12,9124	12,9553	13,1412	13,2209	13,0445	13,0887	13,1192	13,8947

Tabla 6. Industrial computer (Intel Core i7 3770) time results

On average, the processing speed of the Zynq RFSoC UltraScale+ is a 61.5% of the speed of the 7020 and about five times slower than an industrial computer, with execution times ranging between 6.75 and 7.35 seconds to execute 100 iterations of the program, about 67.5 or 73.5 milliseconds per iteration.

The card Zynq 7020, as expected. Is the slowest of all, almost twice as slow as the UltraScale+ and ten times slower than the industrial computer. The execution times range between 11 and 12 seconds for one hundred iterations, about 110 or 120 milliseconds per iteration.

Conclusions

The results of this project are not any different than expected. As it had been predicted, the slowest of the cards is the Zynq 7020, because it has installed less processors, and they are less powerful. While the UltraScale+ has four processors ARM Cortex-A53 and two real time processors ARM R-5, the 7020 has only two processors ARM Cortex A-9.

The execution times of the computer are much faster than those of the cards, as a PC has a much more powerful processor. Embedded systems require small sized, which limit the hardware that can be installed and, obviously, the processing speed, but computers don't have a space limitation, so they can have much more powerful processors. The computer where the executions have been carried out has an Intel Core i7.

The future of the processors resides in speed. Being able to process a huge amount of data in smaller and smaller periods of time is the goal of any informatic system, and that is why all the evolutions in processors tend to this end: decrease time, cost and size. Regarding the programmer, knowing the speed and capacity in your processing system is essential. Just like a wielder needs to know the metals he is going to wield, temperatures, resistances, positions etc., a programmer needs to know the capacity of their processing tool. In this research, the card UltraScale+ of Xilinx has proved to be the tool needed for the company's project.

Referencies

Gomar, J. (30 de Septiembre de 2018). *Profesional Review*. Obtenido de https://www.profesionalreview.com/2018/09/30/que-es-un-soc/

National Instruments. (24 de Mayo de 2019). *National Instruments.* Obtenido de https://www.ni.com/es-es/innovations/white-papers/06/understanding-ffts-and-windowing.html

NVIDIA. (s.f.). *NVIDIA*. Obtenido de nvidia.com: https://la.nvidia.com/object/what-is-gpu-computing-la.html

Oshana, R. (2012). DSP For Embedded and Real-Time Systems. Newnes.

Xilinx. (s.f.). *Petalinux Tools*. Obtenido de https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html

Contenido

Tabla de Contenido: Figuras

Tabla de contenido: Tablas

Tabla de contenido: Gráficas

Capítulo 1. Introducción y planteamiento del proyecto

El panorama actual de las herramientas de procesado es muy complejo. Desde procesadores sencillos que solo mueven una batidora o un microondas hasta los más complejos de los mejores superordenadores capaces de procesar billones de bits en un segundo, todas estas herramientas se pueden conseguir en el mercado.

Ante una variedad tan amplia y tantas opciones por las que decantarse, es imprescindible para cualquier diseñador que se precie entender qué herramientas está utilizando para el desarrollo de un proyecto. Es necesario saber qué capacidad tienen los procesadores, datos de tiempos, temperatura etc. Intentar hacer un proyecto escogiendo un procesador sin saber es como intentar construir una casa sin saber qué materiales usar. La trascendencia de saber decidir incurre en tiempos de reacción, programación o ejecución, capacidades de procesador e incluso costes.

Un desarrollador de software que entiende como elegir el hardware que va a usar es realmente valioso. El principal objetivo de este proyecto es precisamente ese: poner a prueba una tarjeta desarrollada por Xilinx y ejecutar en ella un par de funciones sencillas que den datos fiables de los tiempos de computación. Estos tiempos serán comparados con los tiempos de ejecución de otra tarjeta un poco menos potente y con los de un ordenador industrial. Al final del proyecto, se debería saber si la tarjeta que se está poniendo a prueba es suficiente para ser utilizada en otro proyecto de la empresa en el que estas dos sencillas funciones desarrollan un papel.

Como parte del proyecto se considera el desarrollo de las funciones en cuestión, una de ellas genera el enventanado de una señal y la otra los coeficientes de un filtro de tipo FIR, y la generación de un kernel basado en Linux que se instalará en la tarjeta como sistema operativo para ejecutar el programa. Todo esto se realizará sobre una tarjeta UltraScale de Xilinx, sin embargo, en la otra tarjeta con la que se va a comparar, la Zynq 7000Series, solamente se ejecutarán los programas ya que ya está puesta en marcha.

1.1.- Estado del arte

Como se ha mencionado antes, actualmente se ofrecen cientos de soluciones de procesado en el mercado. Desde los procesadores más sencillos, con capacidades muy limitadas y funciones muy básicas, hasta los procesadores más complejos, capaces de procesar muchos datos por segundo y precios más altos.

Obviamente, no todos los procesadores están desarrollados para realizar las mismas funciones, y no todos ellos funcionan o se programan de la misma forma. Se podría decir, que hay cinco grandes grupos de tipos de procesadores que se diferencian por la forma en la que procesan los datos, según los niveles de abstracción que necesitan para realizar operaciones. Ordenados de más niveles de abstracción a menos niveles de abstracción, estos serían: procesadores de uso genérico (GPP siglas en inglés), unidades de procesado gráfico (GPU, siglas en inglés), procesador digital de señales (DSP, siglas en inglés), *Field Programmable Gate Array* o FPGA y *Application Specific Integrated Circuit* o ASIC.

Mientras que las dos primeras son herramientas para trabajar a nivel de sistema operativo, de forma digital en Software y con varios niveles de abstracción, las dos últimas tienen un nivel de abstracción, o ni eso, trabajan a nivel bit y requieren de conocimientos de electrónica digital además de software para ser capaz de programarlos.

- GPP: *General Purpose Procesors* en inglés. Estos procesadores se encuentran en cualquier sistema informático. Su sencillez de programación y bajo precio los hacen ideales para realizar cualquier tarea de programación, desde lo más sencillo hasta lo más complicado. Estos procesadores son los que se encontrarían, por ejemplo, en un ordenador normal y corriente como son los Intel, AMD etc. Su programación se realiza mediante lenguajes muy familiares, sencillos de usar, como son Python, C, C++ o Java entre muchos otros. Estos lenguajes son traducidos por compiladores y más tarde por sistemas operativos para que el procesador pueda realizar las operaciones a nivel bit. Todas estas traducciones convierten a estos procesadores en herramientas realmente lentas (comparadas con otras herramientas de procesado) y con un alto consumo, pero muy sencillas de programar incluso para la aplicación más complicada.

Figura 1. GPP de Intel (Fuente: Intel)

- GPU: *Graphics Processing Unit* en inglés. Son procesadores muy parecidos a los anteriores en temas de cómo funcionan y cómo se programan. Sin embargo, un GPP tiene entre 1 y 4 núcleos, 8 los más potentes. Esto implica que los procesadores mencionados arriba realizan las funciones de forma secuencial, una detrás de otra, y en paralelo no pueden realizar más de 4 (u 8) funciones. Las GPU se caracterizan por tener miles de núcleos menos potentes que los núcleos de una GPP. Por tanto, las GPU pueden realizar funciones en paralelo, esto es, ejecutar todas al mismo tiempo. Esto es muy útil., entre otras cosas, para el procesado de imágenes. Habiendo miles de píxeles en cada imagen, una GPU los puede tratar independientemente, dándole más agilidad al ordenador en uso. Estas herramientas se utilizan en aplicaciones no gráficas también, como es el procesado de grandes vectores (NVIDIA, s.f.). Las tarjetas gráficas se programan con lenguajes como OpenCL y algunos ejemplos serían NVIDIA, AMD o ARM.

Figura 2. GPU de NVIDIA (Fuente: NVIDIA)

- DSP: *Digital Signal Processor* en inglés. Estos procesadores son unidades muy sencillas que se utilizan para procesar señales de forma digital. En general, las señales se tratan utilizando FPGA, sin embargo, si la operación a realizar es muy compleja, programar una FPGA es más complicado que un sistema digital, por lo que se utilizan estos chips. Estos elementos no se

suelen utilizar de forma independiente, sino que forman parte de sistemas mucho más grandes con otros procesadores que realizan el trabajo real. Sin embargo, se pueden utilizar en aplicaciones de sonido o vídeo de forma independiente. De hecho, muchas FPGA hoy día traen bloques que incluyen algún DSP, para no tener que cambiar de dispositivo al ejecutar un programa.

Figura 3. DSP de Texas Instruments. (Fuente: Arquitectura de los computadores)

- FPGA: *Field Programmable Gate Array*. Estas tarjetas consisten en una matriz de bloques lógicos conectados entre sí. Cada bloque tiene un número de entradas y salidas y, dentro, puertas lógicas o Flip Flops. De esta forma, cada bloque realiza una operación sobre los bits de entrada para obtener la salida deseada. Programar esta tarjeta implica abrir ciertas conexiones entre los bloques, para conseguir que los bits sean sometidos a las operaciones que el programador desee. Estas tarjetas se programan con lenguaje RTL, o lenguaje descriptivo, como son VHDL o Verilog, y los mayores fabricantes del mundo son Xilinx y Altera.

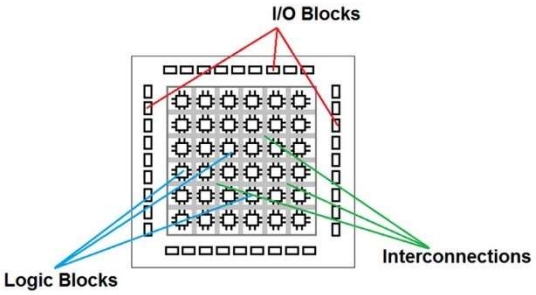


Figura 4. Esquema de una FPGA. (Fuente: Stemmer Imaging)

- ASIC: *Application Specific Integrated Circuit*. Estos chips son el nivel de hardware más puro. Consisten en soldar los elementos electrónicos directamente sobre el circuito para manipular la entrada y conseguir la salida deseada. Los ASIC son la forma de procesado de datos más específica y menos flexible que existe. Actualmente, está muy de moda el uso de estos chips para minar criptomoneda.

Figura 5. ASIC de Samsung para minería de criptomoneda (Fuente: bitcoinist)

En primer lugar, los elementos de software son los más sencillos de utilizar. Los GPP y GPU tienen la gran ventaja de su flexibilidad. Al usar un sistema operativo, un compilador y más capas de abstracción que traduzcan un código de programación al

código que entienden los ordenadores, código binario, se hace mucho más sencilla su programación, ya que estos lenguajes de programación representan en funciones sencillas, procesos complicados a nivel hardware. Precisamente por eso, intentar programar la misma función en un GPP o un GPU es mucho más sencillo que hacerlo en una FPGA o un ASIC.

Las dos últimas, trabajan a nivel bit, sobre el propio silicio. No hay una capa de abstracción que sirva de intérprete para lo que se escribe en pantalla, sino que directamente se debe hablar con el sistema, que habla un lenguaje mucho más complicado. Para programar estas tarjetas son necesarios conocimientos de electrónica digital. Es por esto que, los programas más complicados, suelen ir programados en sistemas de software, no de hardware, ya que se ahorra en costes de tiempo de programación y de un especialista que sepa programarlo.

En segundo lugar, está la flexibilidad. Como se ha mencionado antes, un equipo de software es muy sencillo de programar. Con un GPP o GPU se pueden programar infinidad de programas de forma relativamente fácil. Estas herramientas permiten realizar funciones tan complejas que se puede decir que son realmente flexibles. Además, estas tarjetas son reprogramables. En cualquier momento se puede eliminar un programa e instalar uno nuevo. Sin embargo, las FPGA se pueden reprogramar en algunos casos, dependiendo de la tarjeta, y subiendo el precio de la misma, y los ASIC ni siquiera se pueden reprogramar, ya que no tienen elementos de software, sólo de hardware.

Está claro que las FPGA y los ASIC deberían tener alguna ventaja, para que tenga sentido su existencia. Sus mayores ventajas son la enorme eficiencia y la reducción en costes de tiempo. Estas tarjetas trabajan directamente sobre el silicio, sin pasar por apenas capas de abstracción. Esto implica que, cuando se ejecuta un programa, no es necesario un intérprete que traduzca el lenguaje del ordenador al lenguaje de la tarjeta, y eso acelera mucho los procesos. Además, si no se debe alimentar a ese interprete, los costes de energía son mucho más bajos, simplemente hay que gastar lo que cuesta alimentar a la tarjeta. Por otro lado, los sistemas de software tienen varias capas de abstracción que facilitan en gran medida la programación, y, por tanto, alimentar esas capas de abstracción no es gratis, alguien tiene que cargar con ellas, que es el propio procesador. Es por esto, que estos sistemas tienen consumos de tiempos de ejecución y energía mucho más altos.

Por último, los costes monetarios de cada tarjeta. Los elementos de software son elementos bastante sencillos de crear, ya que el trabajo lo hacen sistemas virtuales, las capas de abstracción como el sistema operativo, por tanto, son procesadores mucho más baratos. A medida que se avanza hacia sistemas de hardware, más específicos, estos son más complicados de fabricar y el precio sube de forma bastante rápida. Por tanto, las FPGA son tarjetas muy caras ya, y los ASIC más todavía.

En conclusión, para proyectos que requieran tareas más complejas y tiempos de ejecución relativamente bajos, se utilizan sistemas de software, que además son más baratos. Sin embargo, para tareas más sencillas pero que requieran tiempos de ejecución muy bajos y para los que el presupuesto sea más alto, es mucho mejor utilizar FPGA.

En la figura que se presenta a continuación, se puede ver perfectamente el equilibrio entre flexibilidad y eficiencia de cada uno de los tipos de procesador:

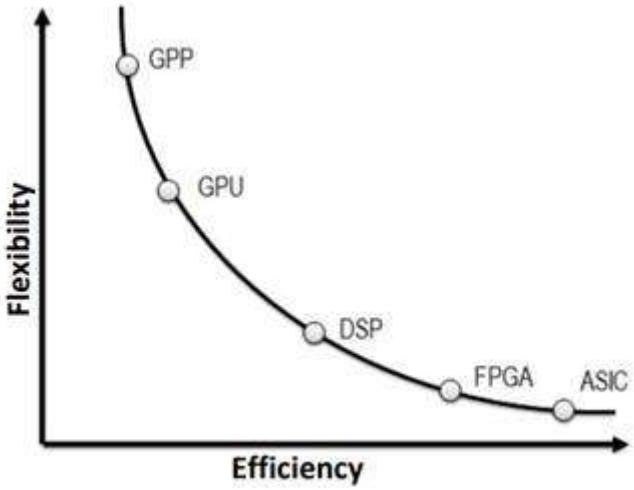


Figura 6. Eficiencia frente a flexibilidad de los diferentes tipos de procesadores. (Fuente: Propia).

Además de los mencionados anteriormente, existe otro tipo de HW llamado System on a chip (SoC). Éstos combinan, generalmente, una CPU con RAM o algún otro elemento, y acceso a ciertos periféricos dentro del mismo chip, encapsulado todo de forma muy comprimida. Este tipo de chips tienen la ventaja de que son más baratos de fabricar, debido a que no hay que hacer un ensamblaje, sino que los elementos se sueldan directamente a la placa, y además la reducida distancia de los elementos implica una reducción de los tiempos de ejecución, ya que la distancia que tienen que recorrer los datos es mucho más baja. Los principales inconvenientes de estos elementos son que si se rompe aunque sea solo uno de sus componentes, éste no se puede cambiar y hay que cambiar el SoC entero. Además, el reducido espacio limita el tamaño de las piezas de HW que se le pueden introducir.

1.2.- Motivación y objetivos

Como ingeniero, siempre es necesario conocer con qué herramientas se trabaja. La capacidad que tienen estas herramientas y si responden a los requerimientos del proyecto en el que se esté trabajando.

Mediante la realización de este proyecto, se pretende dar con datos concretos de ejecución de ciertos programas para comprender si el material que se va a utilizar durante el proyecto es adecuado a las necesidades de la empresa. Por otro lado, se comparará con otros dispositivos para obtener de forma cualitativa una idea de cuál es el peso real del programa y cómo cambiaría el proyecto si se utilizaran otras herramientas diferentes.

Además, durante el desarrollo del trabajo se programarán unas funciones que realmente pueden ser utilizadas en un proyecto real de la empresa, y se ejecutarán sobre

una tarjeta que será necesaria para el mismo proyecto. Como parte del trabajo también se pondrá en marcha la tarjeta mediante la generación de un kernel basado en Linux y se realizará la conexión con la tarjeta.

1.3 Metodología

Para la realización de este proyecto, se deben tomar medidas de tiempos de ejecución de un programa en diferentes plataformas. Por tanto, el proceso de realización del proyecto se puede dividir en dos grandes bloques: los programas que se van a ejecutar y la puesta en marcha de las plataformas para la ejecución de los programas.

En primer lugar, la programación del software que pondrá a prueba las tarjetas. El programa que se va a ejecutar consta de dos sencillas funciones que pretenden estresar la CPU de las tarjetas para ver a qué velocidad son capaces de ejecutarlo. Estos programas serán escritos en lenguaje de programación C y se utilizarán funciones de la librería *liquid-dsp* de Joseph D. Gaeddert.

La primera función consiste en un generador de coeficientes para un enventanado. Las entradas de esta función son el orden de la ventana a generar y el tipo de ventana, que viene codificada con un número del uno al ocho, y la salida es un vector del tamaño del orden con la ventana. La función utilizará en parte algunos de los comandos que ofrece la librería mencionada arriba, pero para las ventanas que no están presentes en dicha librería, se ha simulado el mismo código que utiliza Matlab para generarlas.

La segunda función es otro generador de coeficientes, esta vez de un filtro de tipo FIR, con el final enventanado. En la función deben entrar el orden del filtro, la frecuencia de corte y la frecuencia de muestreo, y la ventana del tamaño del orden del filtro. La salida será un vector de tamaño una unidad superior al orden del filtro, y estará codificado en 16 bits con signo. El código de esta función es generado a partir del código de la función fir1.m de Matlab.

La última función es la más sencilla de todas, para medir el tiempo. Con la librería *time.h* se obtuvieron la hora de inicio de ejecución y la hora de fin de ejecución de las funciones de interés, y posteriormente se restaron las horas, obteniendo una buena aproximación de los tiempos de ejecución.

En segundo lugar, la puesta a punto de las diferentes plataformas. Esta ha sido la parte más complicada del proyecto. Por un lado, hubo que generar desde un ordenador, una imagen de un kernel de Linux que posteriormente se instalaría en la tarjeta en cuestión. Para ello, se utilizó una herramienta aportada por la propia empresa Xilinx llamada *Petalinux*. Esta herramienta permite trabajar con sistemas embebidos de Xilinx y, aprender a utilizarla, supuso uno de los mayores retos del proyecto. Afortunadamente, solo hubo que utilizarla en una de las tarjetas Zynq, en la RFSoC, ya que la 7020 estaba ya configurada para su uso. Por último, en el ordenador industrial se utilizó una máquina con Ubuntu 64 para ejecutar las funciones y obtener unos tiempos de referencia. Para la comunicación con la tarjeta, se utilizó conexión Ethernet y una herramienta llamada *Linaro.*

Por último, la toma de datos. Para obtener resultados más fiables y homogéneos, en lugar de medir el tiempo de una ejecución del programa, se ha ejecutado el mismo varias veces, en concreto 100, y se obtiene el tiempo medio de ejecución. De esta forma las medidas son más fiables. Esto se ha hecho 20 veces por cada medida, y a esas 20 medidas se les ha hecho la media.

Capítulo 2. Descripción de las tecnologías

Durante el proyecto, se utilizarán dos SoCs, fabricados por Xilinx. Éstos tienen en el propio chip la CPU, memoria RAM (aunque admite memoria externa) y acceso de alta velocidad a la FPGA entre otras cosas. Esta comunicación tan rápida con la FPGA permite realizar operaciones de HW y de SW del mismo programa sin sufrir grandes incrementos del tiempo empleado en ejecutar el programa en cuestión. De ahora en adelante, cuando se hable de "Processing System" (PS), es la parte que se ejecuta en el GPP, y la "Programmable Logic" (PL), la parte que se ejecuta en la FPGA. Más en concreto, el HW que se utiliza será:

- Xilinx Zynq RFSoC UltraScale+, tarjeta ZCU111.
- Xilinx Zynq 7000 Series, tarjeta 7020.

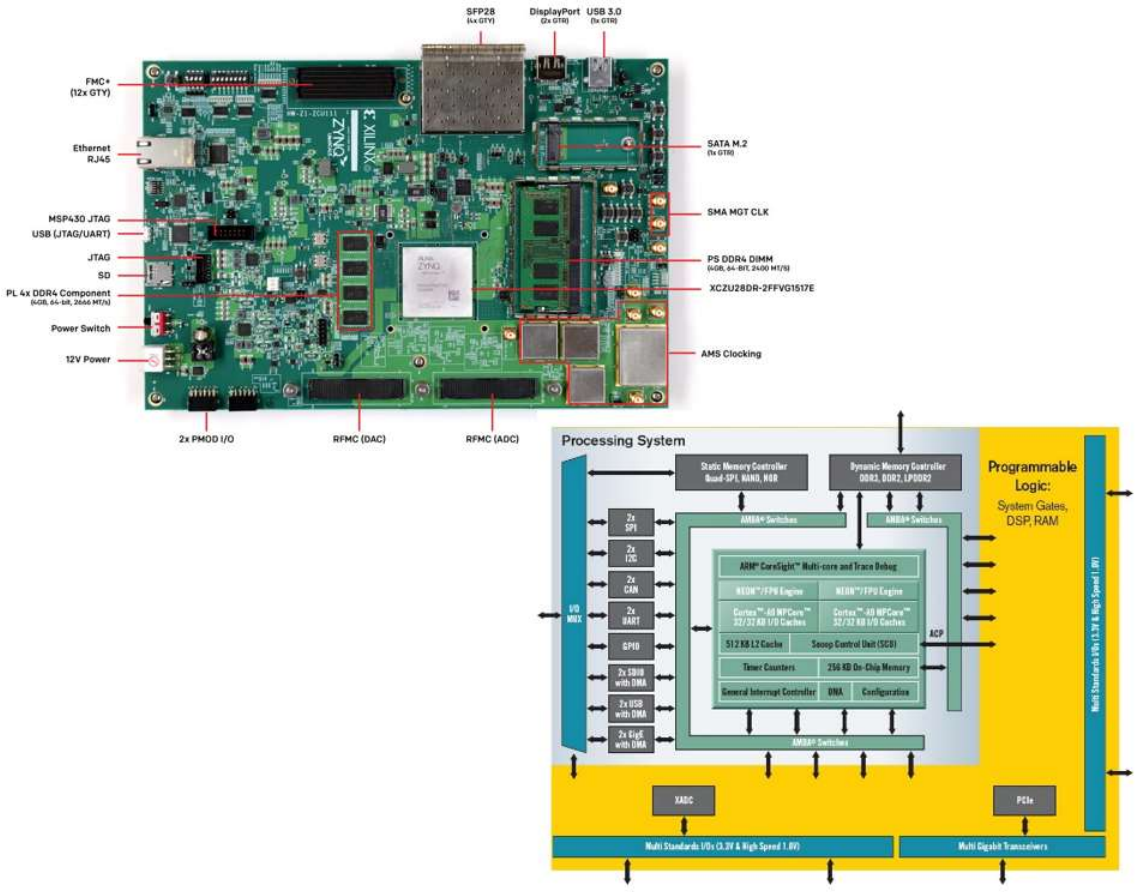


Figura 7. A la izquierda, Zynq US+ RFSoC, a la derecha, esquema de la familia Zynq 7000Series (Fuente: Xilinx)

A priori, y viendo el hardware de cada uno de los elementos, se puede empezar a suponer cual debería ser la tarjeta más rápida. La UltraScale+ tiene cuatro procesadores ARM Cortex-A53 y dos procesadores en tiempo real ARM R5, mientras que la

7000Series tiene únicamente dos procesadores ARM Cortex-A9. Esto es el PS de las tarjetas. En la figura 8, se puede ver una comparación de los rendimientos de un solo hilo de los distintos procesadores Cortex-A de ARM (en 2012). Los Cortex-A53 incluyen la nueva arquitectura ARMv8, frente a la ARMv7 del Cortex-A9, que es de 64 bits en lugar de 32 (Jeff, 2013).

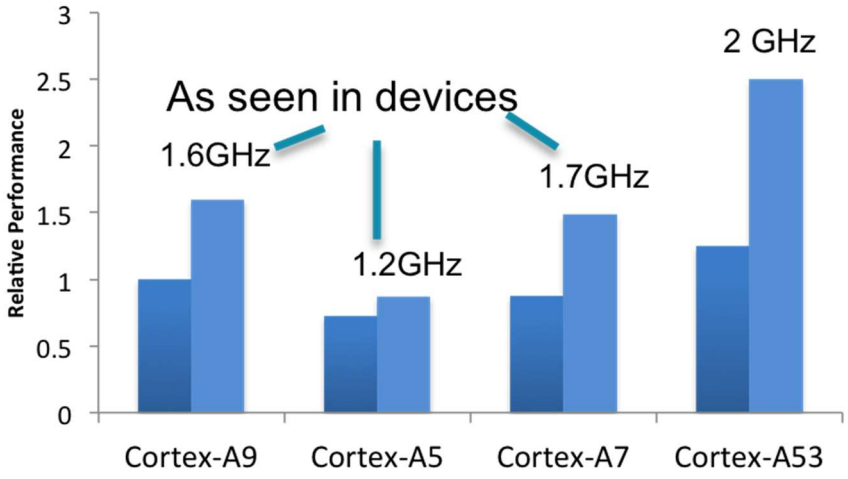
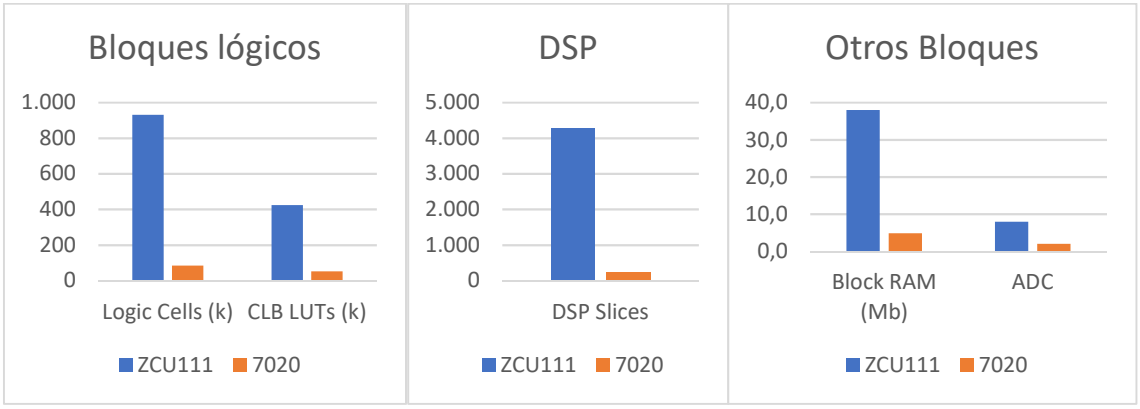


Figura 8. Comparacion de los procesadores Cortex-A. (Fuente: ARM Community)

En cuanto al PL, la UltraScale+ tiene varios bloques mucho más potentes. Como referencia de comparación se utilizarán algunos de los parámetros más representativos de las FPGA.

	ZCU111	7020
Logic Cells (k)	930	85
CLB LUTs (k)	425	53
Block RAM (Mb)	38,0	4,9
DSP Slices	4.272	220
ADC	8	2

Tabla 1. Comparación de los parámetros más importantes de los PLs. (Fuente: Propia)



Gráfica 1. Comparación de los parámetros más importantes de los PLs. (Fuente: Propia)

Logic cells es simplemente una medida de tamaño que utiliza Xilinx para comparar sus FPGA. Un *logic cell* se compone de 4 *look up tables* (LUT) y un flip flop. Los CLB LUTs son los *configurable logic block* con *look up tables*. Un CLB es la unidad básica de una FPGA. Cada bloque que realiza unas operaciones es un CLB y cada CLB contiene un LUT un flip flop y un multiplexor en la mayoría de tarjetas de este fabricante. Un LUT es una pequeña memoria que puede realizar las operaciones de Boole que se le asignen a cada bloque (Eastland, 2015). En la siguiente imagen se puede ver cómo está estructurada una FPGA:

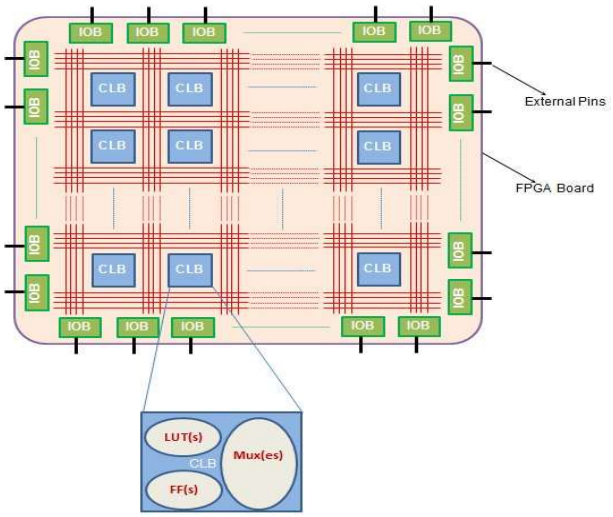


Figura 9. Esquema de una FPGA. (Fuente: All about circuits).

Como se puede apreciar, una FPGA tiene muchos bloques de entrada y salida y consiste en una matriz de CLB, que son la unidad lógica básica que contiene las operaciones lógicas. Cada CLB es diferente, puede contener un número variable de LUTs, flip flops o muxes. Las *logic cells* son simplemente una forma de medir la tarjeta, no son bloques que existen de verdad dentro de la tarjeta, sino que es una unidad de medida en

la que, por cada cuatro LUTs y un flip flop, aunque no esté en el mismo CLB, hay una *logic cell*.

Block RAM son unos pequeños bloques de RAM que hay distribuidos por la tarjeta para tener la memoria dinámica más cerca de los bloques, y acelerar los procesos. Este dato indica la cantidad de memoria block RAM total, pero no dice cuántos bloques de RAM tiene cada tarjeta. En este caso, son bloques de 36Kb, por tanto, la UltraScale+ tiene 1080 bloques de RAM mientras que la 7000Series tiene 140.

Por último, los DSP slices son pequeños procesadores digitales de señales como los que se han descrito anteriormente, y los ADC son simples convertidores analógico/digital. En el caso de la 7000Series, son de 12 bits, mientras que en la RFSoC funcionan a 12 y a 14 bits.

Con estos datos, queda claro que la FPGA de la US+ es más grande que la de la Zynq 7020, lógico dado que tiene un PS más potente y, por tanto, puede enviar más datos. Se debe tener en cuenta que se han comparado sobre todo tamaños de FPGA, no velocidades de procesado de datos, que no tiene porqué ser lo mismo.

Por último, como referencia, se utilizará un ordenador un ordenador prefabricado. Este ordenador tiene un procesador Intel Core i7 3770 con 8GB de RAM. Por supuesto, esto no es un sistema embebido, sino que es un procesador grande con mucha potencia, por lo que debería ser mucho más rápido que cualquiera de las otras dos tarjetas.

Capítulo 3. Desarrollo del proyecto

Como se ha explicado anteriormente, para el desarrollo del proyecto hay dos puntos clave: la programación de las funciones y la puesta en marcha de las plataformas de ejecución.

Las funciones que se han desarrollado para este proyecto se han programado en lenguaje C y son dos: la primera, genera el enventanado de una señal del tipo que el usuario especifique y la segunda obtiene los coeficientes de un filtro digital de tipo FIR. Para el desarrollo de la primera función, fue necesaria otra función que llevara la programación de la ventana de Chebyshev, ya que ésta era mucho más compleja de obtener que las demás. Por último, una sencilla función que se encargue de las medidas de tiempo, con el uso de la librería *time.h*, nada complicado.

La puesta en marcha de las plataformas consiste, sobre todo, en el uso de la herramienta de Xilinx *Petalinux*. Sin embargo, también es necesario establecer la conexión mediante Ethernet con una dirección IP concreta de la tarjeta con el ordenador.

3.1.- Funciones

La primera función que se desarrolló para el proyecto en cuestión fue el enventanado. En primer lugar, ¿qué es el enventanado?

Cuando una señal es tratada con sistemas digitales, esta señal debe ser muestreada en un periodo finito de tiempo. El número de muestras por periodo indicará la resolución de la señal y está limitada por el hardware que se utilice. El periodo finito de tiempo que se utilice también es importante, y aquí es donde el enventanado cumple una función imprescindible.

Dado que un sistema digital solo puede trabajar en periodos finitos de tiempo, es necesario cortar la señal en dos puntos para obtener una porción representativa de la misma. El sistema lo que hace es unir el último instante de tiempo con el primer instante de tiempo, creando un círculo que simula un periodo de tiempo infinito. El problema surge cuando, el valor de la muestra en el primer instante de tiempo es diferente del valor de la muestra en el último instante de tiempo. Cuando esto ocurre, aparece una discontinuidad en la señal que realmente no existe, por lo que el sistema interpreta que aparecen frecuencias que realmente no están en la señal muestreada. Este fenómeno se puede apreciar en la figura 10. (National Instruments, 2019).

La senoidal 1, representa una senoidal con un número entero de periodos. La senoidal 2 es una senoidal con un número no entero de periodos. En las respuestas en frecuencia, se puede apreciar cómo es mucho más nítida la de la senoidal 1 que la de la senoidal 2, que el descenso hasta 0 es mucho más progresivo. Está claro que en este ejemplo no parece muy útil, la señal es simple y está claro que tiene una frecuencia, pero a medida que las señales se hacen más complejas, esta falta de nitidez puede resultar realmente molesta.

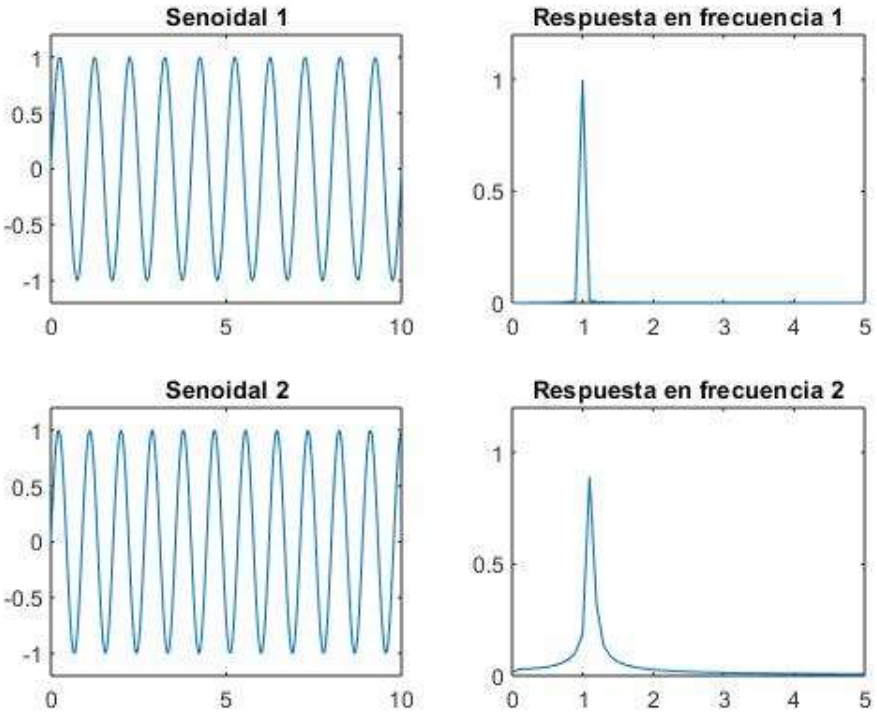


Figura 10. Señales acotadas y muestreadas con sus respuestas en frecuencia. (Fuente: Propia).

A continuación, se puede ver una señal sin enventanar y una señal a la que se le ha aplicado la ventana de Hann. De esta forma se ve perfectamente como afecta la ventana a los extremos de la señal:

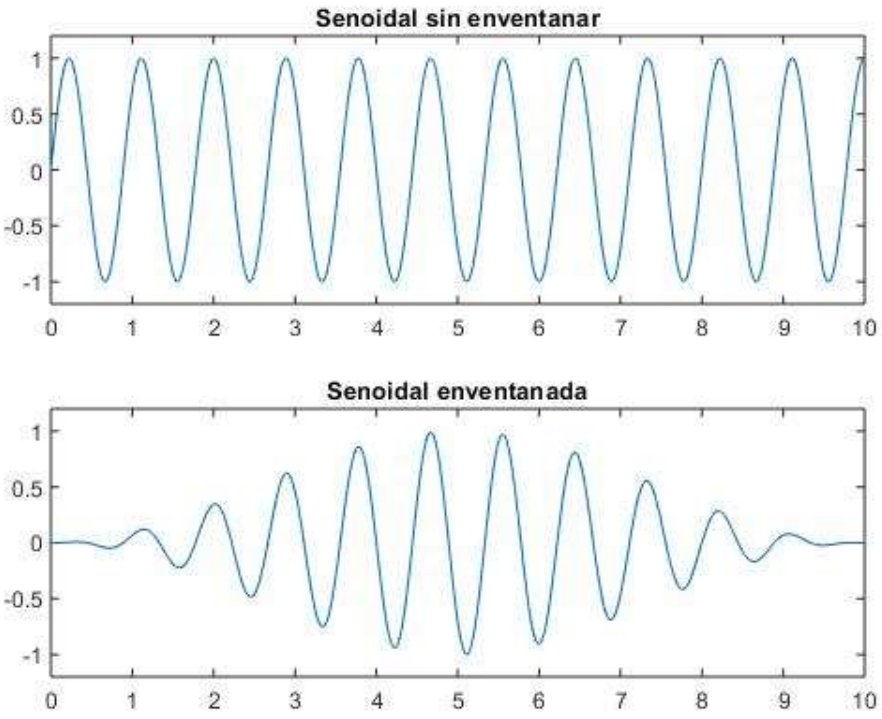


Figura 11. Senoidal enventanada y sin enventanar. (Fuente: Propia)

Hay ventanas de muchos tipos diferentes. En el espectro de frecuencia de las ventanas, se puede apreciar que algunas tienen lóbulos centrales más anchos o más estrechos, otras se diferencian en los lóbulos laterales, la atenuación en las diferentes zonas. Según la forma de la señal con la que se vaya a trabajar, la ventana que se utilizará tendrá una forma u otra que responda a los requisitos del usuario. Si se seleccionan ventanas con una caída muy alta de los lóbulos laterales, el ancho de banda del lóbulo central aumenta, aunque se reduce la fuga espectral (los picos de la respuesta en frecuencia son más estrechos).

Como se puede ver, seleccionar una ventana no es tarea fácil, aunque hay algunas pautas que pueden servir. Si la frecuencia de interés está lejos de las frecuencias de interferencia, se debe coger una ventana con fuerte caída lateral, mientras que si las interferencias están cerca del interés, se debe coger una ventana con lóbulos laterales bajos. Si la señal tiene varias frecuencias de interés cerca, se debe aumentar la resolución espectral cogiendo ventanas con lóbulo central estrecho. Si es más importante conocer con precisión la amplitud de una frecuencia de interés que su posición, el lóbulo central debe ser más ancho. La ventana de Hann es una ventana que funciona en la mayoría de los casos, por eso es la más utilizada de todas. (National Instruments, 2019).

A continuación, se muestran algunas ventanas en el dominio del tiempo y de la frecuencia.

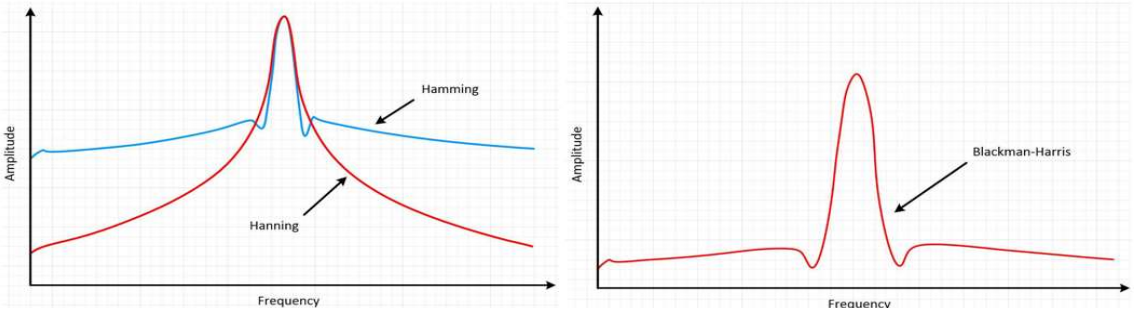


Figura 12. Ejemplos de ventanas en el dominio de la frecuecnia. (Fuente: National Intruments).

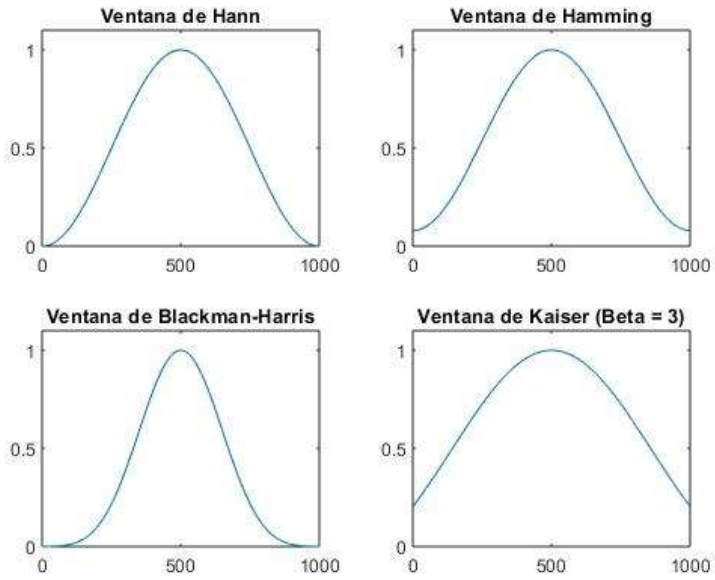


Figura 13. Ejemplos de ventanas en el dominio del tiempo (Fuente: Propia).

Las entradas de la función eran el orden de la ventana y el tipo de ventana que se quería generar, codificada por un número del 1 al 8. La programación de esta función se ha realizado con la ayuda de dos herramientas. Como prioridad para el proyecto de la empresa, el resultado de la función debía ser el mismo que daría una función de Matlab. Sin embargo, como prioridad, siempre se han utilizado las funciones incluidas en la librería *liquid-dsp* de Joseph D. Gaeddert. Si el resultado de la función no daba el mismo resultado que en Matlab o la función no existía en esa librería, se recurría a las funciones de Matlab y se intentaban emular en el código en C. Para la ventana de Chebyshev, que es más complicada, se tuvo que programar una nueva función diferente. En el Apéndice A, se puede ver el código de la función de enventanado, y en el

Apéndice B la función para la ventana de Chebyshev.

La segunda función es la programación de un filtro digital de tipo FIR (*Finite Impulse Response*). Un filtro FIR es un filtro digital que se caracteriza principalmente por no tener realimentación. La salida del filtro se calcula como la suma ponderada de los valores presentes y pasados de la entrada del filtro (University of Colorado). El modelo matemático que siguen estos filtros es el siguiente:

$$y[n] = \sum_{k=-M_1}^{M_2} b_k x[n-k]$$

Donde y es la salida, n el orden del filtro, M_1 y M_2 son los extremos de las muestras de las entradas, asignándole 1 a la primera y el número de muestras a la última, b_k es la ponderación de cada término y x el valor de la entrada. (Oshana, 2012).

Otra forma de ver el modelo matemático sería mediante el siguiente diagrama:

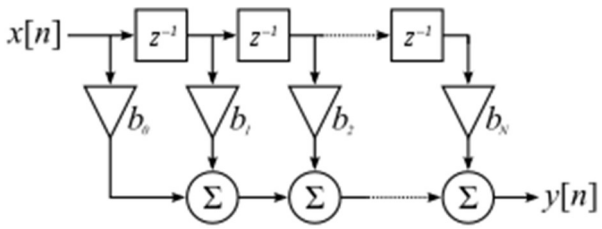


Figura 14. Diagrama de bloques de un filtro FIR. (Fuente: Wikipedia)

La salida no se realimenta, sino que la cada muestra de la entrada se va sumando a la anterior ponderada con un parámetro.

La principal ventaja de esto es que estos filtros son siempre, inevitablemente estables, ya que, al no tener realimentación, no hay forma de que no se llegue a un régimen permanente. Además, este tipo de filtros, en contraposición con los IIR (*Infinite Impulse Response*), tiene una fase lineal. Esto no ocurre con otros filtros, que solo se puede aproximar la fase lineal, y las aproximaciones complican el filtro en gran medida. Además, estos filtros son muy fáciles de programar, por lo que podemos encontrarlo en muchas aplicaciones en las que se requiera filtrado digital. Por otro lado, estos filtros requieren de órdenes muy altos para funcionar bien y, dado que no hay realimentación, los cálculos son mucho más pesados que los de un IIR. Por tanto, estos filtros pueden ser mucho más lentos realizando los cálculos si no se tiene un sistema lo suficientemente potente. En la figura 15, se aprecia muy bien la desventaja de los filtros FIR frente a los IIR en temas de orden necesario para la misma atenuación. (Siemens Phenom, 2018).

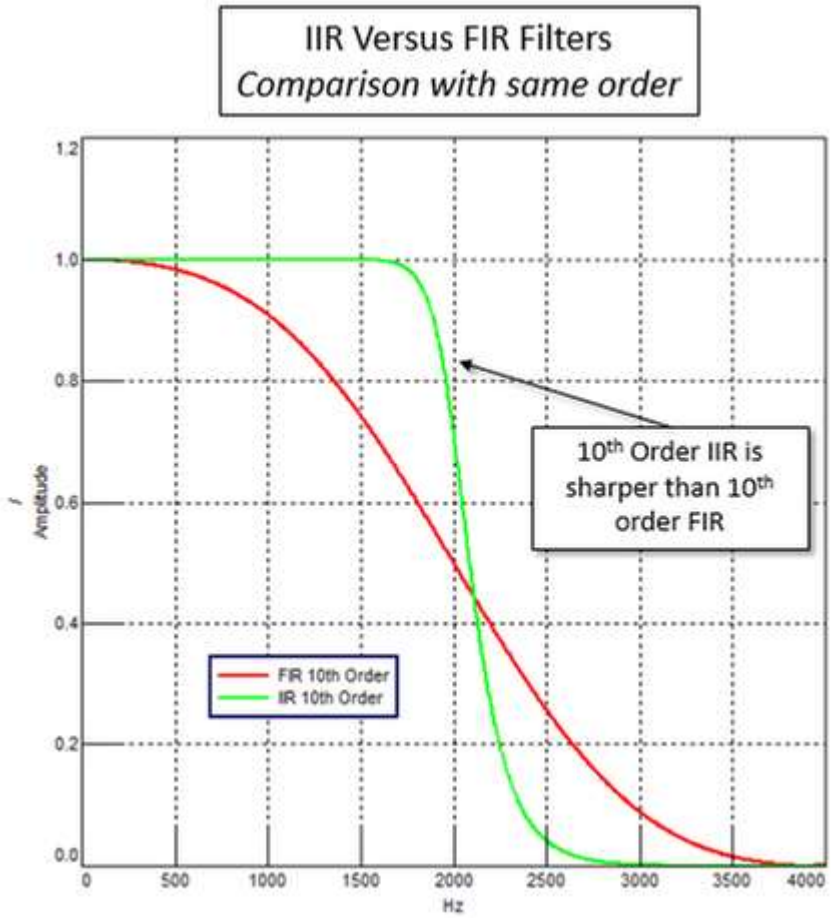


Figura 15. Comparación de un filtro IIR y un filtro FIR. (Fuente: Siemens)

La función que se ha hecho en el proyecto está inspirada en la función fir1.m de Matlab, aunque el filtro siempre es paso bajo. Las entradas de la función son el orden del filtro, la frecuencia de muestreo y de corte y la ventana. El programa comienza creando un diagrama de Bode ideal, el óptimo para el filtro que se busca. A continuación, con la ayuda de funciones de la librería *liquid-dsp*, se hace la transformada inversa de Fourier para conseguir los puntos en el dominio del tiempo. Tras eliminar la biteralidad y normalizar los parámetros, el resultado con los coeficientes de un filtro FIR. Estos coeficientes se multiplican por la ventana, para aplicarla directamente a la señal. El código de esta función se encuentra en el Apéndice C.

Por último, y con ayuda de la librería *time.h* de C, se hizo la medida de tiempos. No es una función complicada, simplemente obtiene la hora a la que se inician las funciones y la hora a la que terminan de ejecutarse. Al restarlas se consigue el tiempo transcurrido de ejecución. El código de esta función se puede ver en el Apéndice D. El *main* del programa, que ejecuta todo, junto con los *includes* está en el Apéndice E.

3.2.- Puesta en marcha

Para la puesta en marcha de las plataformas, hubo que preparar solo dos de ellas. La tarjeta Zynq 7020 ya estaba lista para ejecutar programas por lo que no hubo que realizar ningún tipo de puesta en marcha.

En primer lugar, lo más sencillo fue poner en marcha el ordenador industrial. El PC tenía instalado un sistema de Windows, sobre el que se puso una máquina virtual con un sistema operativo de Linux, más en concreto, Ubuntu de 64 bits. Para ello, se utilizó el programa VirtualBox de Oracle, que es una herramienta muy sencilla de utilizar para crear máquinas en un ordenador. La máquina virtual tenía instalado el IDE de Eclipse, desde el que se realizó la programación y compilación de las funciones a utilizar. Además, se tuvo que instalar el compilador cruzado *aarch64-linux-gnu-gcc*, que es una herramienta que permite compilar programas para ejecutarlos en el kernel de Linux que se iba a instalar posteriormente. Además, para que las librerías funcionaran, hubo que instalarlas para este compilador en concreto. Afortunadamente, las librerías básicas como *stdio.h, math.h, time.h* etcétera, ya venían instaladas. Sin embargo, la librería *liquid.h* tuvo que ser instalada de forma externa, lo que dio algún dolor de cabeza.

En segundo lugar, y la parte más complicada de todas, fue la puesta en marcha de la tarjeta Zynq UltraScale+ RFSoC de Xilinx. A pesar de que Xilinx ofrece una herramienta para trabajar con sus sistemas embebidos de forma sencilla, y muy útil, descifrarla no fue la parte más sencilla.

La herramienta en cuestión se llama *Petalinux* y ofrece utilidades para trabajar con sus sistemas embebidos. En el caso de este proyecto, se utilizó para generar una imagen de un kernel basado en Linux que, posteriormente, se instalaría en la tarjeta.

Lo primero que se necesita para generar el kernel es un archivo de configuración de hardware o .hdf. Este archivo .hdf se genera a partir de un proyecto de Vivado. Vivado es otro programa, también de Xilinx, que permite trabajar con lenguaje de descripción de hardware. Para generar el kernel de Linux, es necesario dar una descripción al sistema operativo de cómo es el sistema al que se ha conectado (la FPGA), por tanto este archivo incluye una estructura de cómo se van a comunicar la FPGA con la CPU, mediante DMA con una comunicación en serie etc. La configuración y creación de este archivo no forma parte del proyecto.

Por tanto, una vez tiene *Petalinux* una estructura del hardware con el que va a trabajar, se puede proceder a su configuración. En esta etapa, se le dice al sistema qué herramientas se van a utilizar y cómo. Se le configuran los componentes configurables y se instalan drivers para los componentes que requieran unos drivers.Una vez hecho esto, la imagen del kernel ya está generada. Se introduce en una tarjeta SD que será la tarjeta desde la que se inicie la UltraScale+.

Una vez conectada la tarjeta al ordenador, mediante un cable Ethernet y otro JTAG, se puede iniciar la conexión. Dado que la conexión con el ordenador será por Ethernet, es necesario cambiar las direcciones IP. Por un lado, al sistema operativo anfitrión, Windows, tendrá la IP 192.168.1.123, que será la IP que tiene como anfitrión el sistema operativo invitado, la máquina virtual de Ubuntu, con una dirección de 192.168.1.1. Utilizando Linaro, una herramienta de software libre que se ha utilizado para para comunicar la tarjeta con el anfitrión, se le asigna la dirección IP 192.168.1.125 a la RFSoC.

Una vez asignadas todas las direcciones IP, se arranca la tarjeta y se abre un terminal en Ubuntu. Se ejecuta la orden: *linaro @192.168.1.125*, y la comunicación se

establece. Desde este momento, se puede trabajar en la terminal de Ubuntu, pero ejecutando todos los comandos en la tarjeta.

Tras compilar el programa que ya estaba listo, se coge el ejecutable y se introduce en la carpeta del sistema que está compartida con la UltraScale+. Dado que ya se han conectado los dos sistemas, se puede ejecutar la orden *./nombredelejecutable*, y el programa correrá en la tarjeta. Desde este punto, sólo queda tomar los datos y hacer la media solicitada.

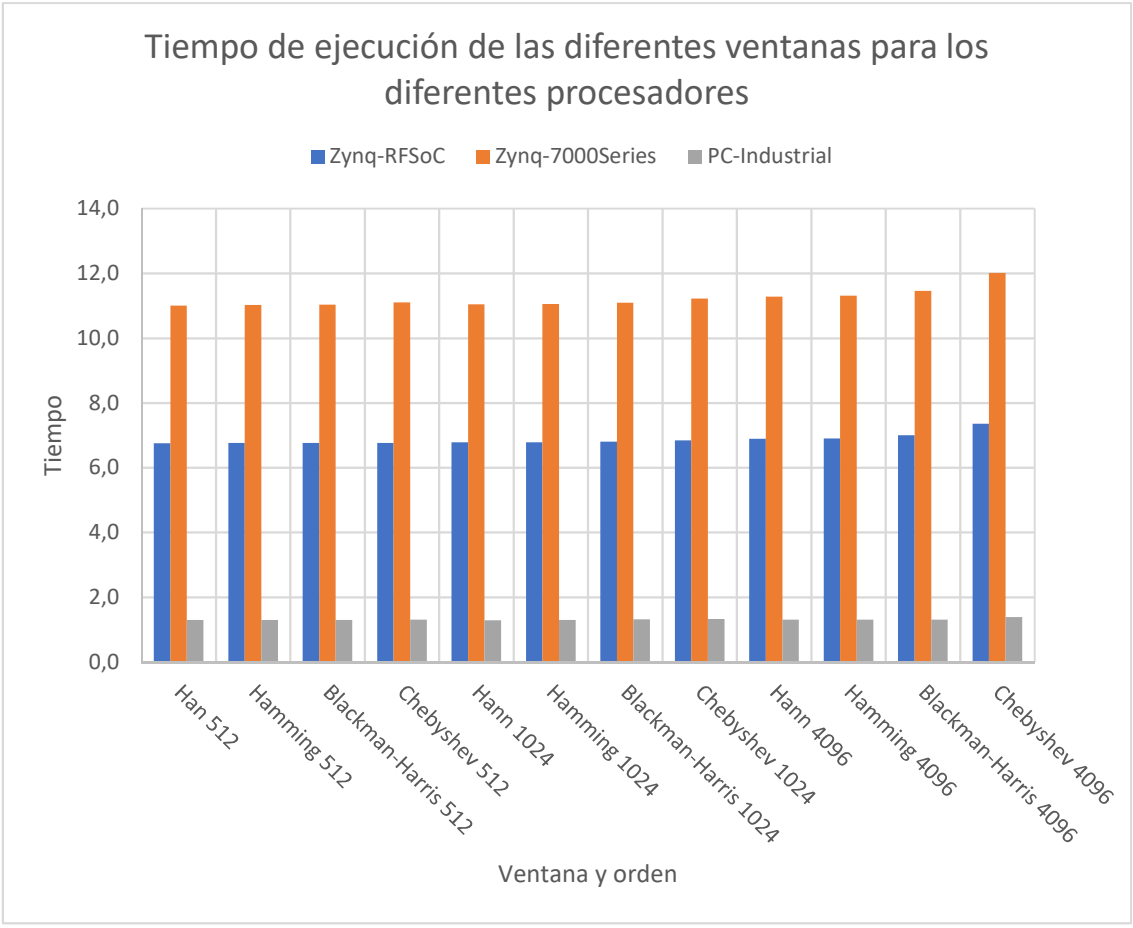
.

Figura 16. Logotipos de Linaro, Vivado y Petalinux. Herramientas básicas de este proyecto. (Fuente: Wikipedia).

Capítulo 4.- Análisis de resultados

El proceso de obtención de resultados se ha explicado anteriormente: simplemente, se ejecutó el programa cien veces seguidas y se midió el tiempo que tardaba en ejecutarlo esas cien veces. Este proceso se repitió veinte veces por cada prueba, y se hizo una media de estos veinte tiempos para sacar el valor final. Se hicieron doce pruebas, cada una ejecutaba una de entre cuatro ventanas para uno de entre tres valores de orden del filtro. Las ventanas eran: ventana de Hann, Hamming, Blackman-Harris por ser muy representativas y la de Chebyshev con 80 dB de atenuación por ser la más pesada de ejecutar. Por último, los órdenes de filtro que se usaron fueron 512, 1024 y 4096, por ser los órdenes más representativos para el proyecto de la empresa.

A continuación, se presenta una gráfica que compara las tres tarjetas, junto con unas tablas que recogen los resultados de cada uno de los experimentos.



Gráfica 2. Comparación de los tiempos de ejecución de las tres plataformas objeto de estudio.

Tarjeta		Zynq RFSoC											
Número de eventos por itercaión		100											
Orden del filtro		512				1024				4096			
Tipo de ventana		Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev
Tiempo total (s)	1	6,7928	6,7496	6,8152	6,7600	6,7665	6,7738	6,7975	6,8402	6,8860	6,8986	6,9989	7,3587
	2	6,7462	6,7480	6,7621	6,7561	6,8139	6,7740	6,7969	6,8412	6,8814	6,8920	6,9936	7,3542
	3	6,7444	6,7473	6,7615	6,7580	6,7715	6,7721	6,7945	6,8395	6,8906	6,8872	6,9918	7,3535
	4	6,7425	6,7490	6,7599	6,7569	6,7687	6,7747	6,7976	6,8377	6,9597	6,9024	6,9942	7,3451
	5	6,7426	6,8220	6,7644	6,7581	6,7632	6,7701	6,8005	6,8326	6,8908	6,8956	7,0003	7,4007
	6	6,7453	6,7472	6,7707	6,8223	6,7732	6,7708	6,7973	6,8369	6,8844	6,9018	6,9948	7,3492
	7	6,7478	6,7456	6,7610	6,7582	6,8433	6,8249	6,7973	6,8360	6,8891	6,8952	6,9944	7,3427
	8	6,7458	6,7476	6,7639	6,7543	6,7733	6,7790	6,8008	6,8400	6,8821	6,8991	6,9915	7,3528
	9	6,8081	6,7523	6,7582	6,7574	6,7662	6,7692	6,7997	6,8428	6,8863	6,9017	6,9923	7,3485
	10	6,7453	6,7509	6,7564	6,7567	6,7728	6,7734	6,7966	6,8409	6,8967	6,8994	6,9916	7,3446
	11	6,7575	6,7542	6,7663	6,7541	6,7749	6,7775	6,7994	6,8361	6,8938	6,9061	6,9999	7,3591
	12	6,7687	6,7547	6,7663	6,7579	6,7787	6,7795	6,8021	6,8338	6,8899	6,9056	6,9921	7,3614
	13	6,7578	6,7551	6,7677	6,7566	6,7746	6,7801	6,7977	6,8385	6,8953	6,9064	6,9997	7,3614
	14	6,7596	6,7534	6,7698	6,7533	6,7716	6,7767	6,7988	6,8377	6,8951	6,9027	6,9955	7,3638
	15	6,7572	6,8042	6,7663	6,7574	6,7701	6,7785	6,8030	6,8334	6,8923	6,9040	7,0024	7,3631
	16	6,7584	6,7566	6,7685	6,7562	6,7737	6,7777	6,8020	6,8395	6,8946	6,9033	6,9947	7,3601
	17	6,7579	6,7566	6,7682	6,7508	6,7749	6,7802	6,8045	6,8340	6,8946	6,9077	6,9996	7,3627
	18	6,7558	6,7571	6,7642	6,7532	6,7751	6,7754	6,8055	6,8356	6,8949	6,9033	7,0001	7,3633
	19	6,7557	6,7595	6,7653	6,7575	6,7802	6,7803	6,8055	6,8373	6,8965	6,9045	6,9995	7,3632
	20	6,7562	6,7551	6,7669	6,7568	6,7752	6,7798	6,8044	6,8406	6,8931	6,9064	6,9984	7,3621
	Media	6,7573	6,7583	6,7671	6,7596	6,7781	6,7784	6,8001	6,8377	6,8944	6,9011	6,9963	7,3585
Tiempo por evento (ms/evento)	Media	67,5607	67,5595	67,6732	67,6379	67,8127	67,7821	67,9787	68,3878	68,9470	68,9730	69,9433	73,5499

Tabla 2. Tiempos de ejecución de la Zynq UltraScale+ RFSoC, tarjeta ZCU111.

Tarjeta		Zynq 7000-Series											
Número de eventos por itercaión		100											
Orden del filtro		512				1024				4096			
Tipo de ventana		Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev
Tiempo total (s)	1	11,0045	11,0174	11,0342	11,1038	11,0428	11,0565	11,0980	11,2303	11,2822	11,3323	11,4523	12,0087
	2	11,0043	11,0122	11,0293	11,1018	11,0416	11,0525	11,0953	11,2269	11,2799	11,3330	11,4501	12,0103
	3	11,0046	11,0373	11,0295	11,1016	11,0418	11,0504	11,0963	11,2271	11,2791	11,0019	11,4524	12,0036
	4	11,0056	11,0414	11,0292	11,1030	11,0412	11,0512	11,0960	11,2247	11,2808	11,3305	11,4546	12,0124
	5	11,0039	11,0365	11,0282	11,1000	11,0406	11,0506	11,0963	11,2259	11,2781	11,3310	11,4502	12,0083
	6	11,0036	11,0355	11,0292	11,1009	11,0400	11,0513	11,0982	11,2268	11,2795	11,3306	11,4498	12,0047
	7	11,0033	11,0352	11,0031	11,1008	11,0411	11,0509	11,0959	11,2263	11,2799	11,3313	11,4516	12,0091
	8	11,0031	11,0362	11,0286	11,1033	11,0419	11,0523	11,0951	11,2278	11,2786	11,3311	11,4509	12,0074
	9	11,0124	11,0364	11,0299	11,1241	11,0640	11,0687	11,0961	11,2352	11,2932	11,3686	11,4778	12,0166
	10	11,0103	11,0375	11,0146	11,1032	11,0472	11,0535	11,0803	11,2184	11,2797	11,3252	11,4527	12,0048
	11	11,0051	11,0173	11,0597	11,1105	11,0439	11,0625	11,0929	11,2272	11,2796	11,3308	11,4680	12,0133
	12	11,0060	11,0157	11,0577	11,1093	11,0412	11,0613	11,0979	11,2253	11,2778	11,3282	11,4657	12,0131
	13	11,0042	11,0150	11,0578	11,1077	11,0412	11,0577	11,0976	11,2253	11,2770	11,3276	11,4655	12,0036
	14	11,0048	11,0143	11,0574	11,1082	11,0419	11,0611	11,0979	11,2240	11,2768	11,3283	11,4671	12,0104
	15	11,0068	11,0157	11,0586	11,1074	11,0408	11,0618	11,1354	11,2247	11,2770	11,3272	11,4666	12,0111
	16	11,0091	11,0199	11,0237	11,1078	11,0567	11,0612	11,0835	11,2209	11,2821	11,3206	11,4651	12,0145
	17	11,0079	11,0197	11,0230	11,1074	11,0539	11,0622	11,0819	11,2178	11,2825	11,3183	11,4646	12,0089
	18	11,0070	11,0190	11,0223	11,1090	11,0538	11,0592	11,0808	11,2189	11,2808	11,3188	11,4629	12,0150
	19	11,0072	11,0172	11,0209	11,1095	11,0548	11,0577	11,0810	11,2167	11,2815	11,3174	11,4642	12,0018
	20	11,0066	11,0190	11,0222	11,1086	11,0531	11,0597	11,0813	11,2183	11,2821	11,3182	11,4631	12,0075
	Media	11,0060	11,0249	11,0330	11,1064	11,0462	11,0571	11,0939	11,2244	11,2804	11,3125	11,4598	12,0093
Tiempo por evento (ms/evento)	Media	110,0555	110,3255	110,2559	111,0425	110,4421	110,5379	110,9475	112,2694	112,8110	113,0156	114,5423	120,0858

Tabla 3. Tiempos de ejecución de la tarjeta Zynq 7000Series, 7020.

Tarjeta		Ordenador industrial - Intel core i7											
Número de eventos por itercaión		100											
Orden del filtro		512				1024				4096			
Tipo de ventana		Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev	Hann	Hamming	Blackman-Harris	Chebyshev
Tiempo total (s)	1	1,3030	1,3152	1,3036	1,3139	1,3136	1,3014	1,3252	1,3272	1,3248	1,3100	1,3185	1,4037
	2	1,3128	1,2842	1,3031	1,3119	1,2870	1,2890	1,3154	1,3205	1,3033	1,3105	1,3084	1,3942
	3	1,2979	1,2810	1,2961	1,3146	1,2907	1,2919	1,3015	1,3202	1,3158	1,3086	1,3063	1,3817
	4	1,3056	1,2883	1,2989	1,3164	1,2940	1,2965	1,3126	1,3298	1,2985	1,3077	1,3023	1,3798
	5	1,2996	1,3011	1,2994	1,3118	1,2815	1,2940	1,3231	1,3009	1,2890	1,3041	1,3165	1,4037
	6	1,3059	1,2926	1,2890	1,3065	1,2960	1,2871	1,3144	1,3117	1,3030	1,3043	1,3196	1,3783
	7	1,3081	1,2924	1,3033	1,2924	1,2924	1,2944	1,3040	1,3416	1,2998	1,3073	1,3138	1,4010
	8	1,2993	1,3016	1,2993	1,2970	1,2876	1,2873	1,3069	1,3226	1,3063	1,3064	1,3212	1,3844
	9	1,3023	1,2862	1,2981	1,2942	1,2858	1,2883	1,3140	1,3127	1,3080	1,3161	1,3129	1,3758
	10	1,2817	1,2924	1,3005	1,2962	1,2837	1,3256	1,3242	1,3337	1,2959	1,3138	1,2997	1,3921
	11	1,2884	1,2991	1,2989	1,3170	1,2778	1,2865	1,3122	1,3362	1,3118	1,3025	1,3129	1,3875
	12	1,2777	1,3112	1,3041	1,3103	1,2948	1,2858	1,3188	1,3573	1,2995	1,3086	1,3072	1,3762
	13	1,2926	1,2980	1,3007	1,3172	1,2856	1,3064	1,3145	1,3208	1,2987	1,3073	1,3029	1,3885
	14	1,2968	1,2935	1,2885	1,,29487	1,2847	1,2863	1,3357	1,3496	1,3030	1,3112	1,3001	1,3859
	15	1,2977	1,2946	1,2962	1,3005	1,2902	1,2861	1,3023	1,3205	1,3001	1,3012	1,2981	1,3848
	16	1,2992	1,2961	1,2988	1,3087	1,2860	1,2846	1,3035	1,3122	1,3042	1,3074	1,3089	1,4031
	17	1,2925	1,2876	1,2943	1,3031	1,2894	1,2896	1,3187	1,3308	1,3060	1,3076	1,3099	1,3758
	18	1,3044	1,2938	1,2944	1,3115	1,2853	1,2931	1,3047	1,3229	1,3051	1,3038	1,2911	1,3784
	19	1,3044	1,2944	1,2938	1,3051	1,2998	1,2815	1,3138	1,3252	1,2983	1,3095	1,3028	1,3882
	20	1,2931	1,2898	1,2985	1,2993	1,2964	1,3004	1,3014	1,3052	1,3121	1,3272	1,2993	1,3845
	Media	1,2981	1,2947	1,2980	1,3067	1,2901	1,2928	1,3133	1,3251	1,3042	1,3087	1,3076	1,3874
Tiempo por evento (ms/evento)	Media	13,0162	12,9349	12,9911	13,0549	12,9124	12,9553	13,1412	13,2209	13,0445	13,0887	13,1192	13,8947

Tabla 4. Tiempos de ejecución del ordenador industrial. Procesador Intel Core i7 3770.

De media, la velocidad a la que procesa datos la Zynq RFSoC es del 61,5% de la velocidad de la 7020 y unas cinco veces más lenta que un ordenador industrial, con unos tiempos de ejecución entre los 6,75 y 7,35 segundos para ejecutar 100 iteraciones del programa, a unos 67,5 o 73,5 milisegundos por iteración.

La Zynq 7020, como era de esperar, es la más lenta de todas, siendo prácticamente el doble de lenta que la RFSoC y diez veces más que el ordenador industrial. Los tiempos de ejecución están entre los 11 y los 12 segundos para cien iteraciones, a unos 110 o 120 milisegundos por iteración.

Capítulo 5.- Conclusiones

5.1.- Conclusiones de los resultados

Efectivamente, los resultados no resultan sorprendentes. Como se había predicho anteriormente, la más lenta de todas las tarjetas es la Zynq 7020, ya que los procesadores que lleva son menos y menos potentes, mientras que la RFSoC lleva cuatro ARM Cortex-A53 además de dos procesadores en tiempo real ARM R-5, la 7020 lleva solo dos ARM Cortex-A9.

Los tiempos de ejecución del ordenador son mucho más rápidos que los de cualquiera de las tarjetas por tener un procesador mucho más potente. Mientras que los otros sistemas tienen sistemas embebidos, que buscan ocupar poco espacio, el ordenador no pretende economizar espacio sino costes. Sin límites de tamaño se pueden conseguir procesadores mucho más potentes como es el Intel Core i7 que lleva el ordenador en el que se han ejecutado los datos.

El futuro de los procesadores está en la velocidad. Ser capaz de procesar muchos datos cada vez en menos tiempo es la meta de cualquier sistema informático, y por eso todas las evoluciones de los procesadores se hacen con ese fin: reducir tiempos, costes y tamaños. De cara al programador, conocer la velocidad y la capacidad que tiene la herramienta con la que trabaja es esencial. Igual que un soldador debe conocer los metales que va a unir, un programador debe saber qué capacidad tiene su procesador. En este estudio se ha visto que los tiempos de ejecución son adecuados para el proyecto en cuestión, y se seguirá adelante con la tarjeta UltraScale+ de Xilinx.

5.2.- Recomendaciones para futuros estudios

Para futuros estudios, sería recomendable ampliar la cantidad de objetos de estudio. En este caso se han probado únicamente dos tarjetas, ya que eran las tarjetas de interés para el proyecto de la empresa. Sin embargo, como herramienta de consulta para un ingeniero, esta información está demasiado limitada. En futuros estudios, sería recomendable añadir dos puntos a este estudio:

1. Utilizar, no solamente la CPU de la tarjeta, sino que se ponga también a prueba la FPGA. Este proyecto, incluye un estudio teórico de la potencia de las FPGA, pero sin un estudio práctico, todo son especulaciones y además de datos cualitativos. Con un estudio teórico de los tiempos de ejecución de las FPGA se podrían obtener datos mucho más concretos de la comparación de ambas plataformas.
2. Comparar más tarjetas. La cantidad de tarjetas que fabrica Xilinx es enorme, incluso dentro de este tipo de SoCs. No sería necesario compararlas absolutamente todas, pero utilizar de cada familia de tarjetas (MPSoC, RFSoC y 7000Series para este tipo de SoCs) por lo menos tres a diferentes niveles teóricos de potencia de ejecución aportaría datos mucho más sólidos al proyecto y lo convertiría en una fuente segura a la hora de

conocer qué hardware es más conveniente para un proyecto u otro. El problema es que no son piezas baratas de hardware, y conseguir los recursos para hacer esto no sería fácil.

Capítulo 6.- Bibliografía

Eastland, N. (31 de Julio de 2015). *Structure of an FPGA.* Obtenido de Digilent Blog: https://blog.digilentinc.com/structure-of-an-fpga/

Gomar, J. (30 de Septiembre de 2018). *Profesional Review*. Obtenido de https://www.profesionalreview.com/2018/09/30/que-es-un-soc/

Jeff, B. (10 de 28 de 2013). *The Top 5 Things to Know about Cortex-A53.* Obtenido de ARM Community: https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/the-top-5-things-to-know-about-cortex-a53

National Instruments. (24 de Mayo de 2019). *National Instruments.* Obtenido de https://www.ni.com/es-es/innovations/white-papers/06/understanding-ffts-and-windowing.html

NVIDIA. (s.f.). *NVIDIA*. Obtenido de nvidia.com: https://la.nvidia.com/object/what-is-gpu-computing-la.html

Oshana, R. (2012). DSP For Embedded and Real-Time Systems. Newnes.

Siemens Phenom. (29 de 08 de 2018). *Siemens.* Obtenido de Introduction to filters: FIR vs IIR: https://community.plm.automation.siemens.com/t5/Testing-Knowledge-Base/Introduction-to-Filters-FIR-versus-IIR/ta-p/520959

University of Colorado. (s.f.). *University of Colorado Colorado Springs.* Obtenido de uccs.edu/eas/: http://www.eas.uccs.edu/~mwickert/ece2610/lecture_notes/ece2610_chap5.pdf

Xilinx. (s.f.). *Petalinux Tools*. Obtenido de https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html

Capítulo 7.- Apéndices

Apéndice A

```
1. void Windowing(int Type, int N, float *out){
2.
3.         float w[N], x[N];
4.         unsigned int i;
5.
6.         switch (Type){
7.                 case 1:
8.                         //printf("Hann window.\n");
9.                         for(i = 0; i < N; i++){
10.                                 w[i] = hann(i, N);
11.                         }
12.                         break;
13.                 case 2:
14.                         //printf("Hamming window.\n");
15.                         for(i = 0; i < N; i++){
16.                                 w[i] = 0.54-
   0.46*cos(2.0f*(float)M_PI*(float)i/(float)(N-1));
17.                         }
18.                         break;
19.                 case 3:
20.                         //printf("Blackman window.\n");
21.                         for(i = 0; i < N; i++){
22.                                 if(i < N/2+1)
23.                                         w[i] = 0.42-
   0.5*cos(2*M_PI*(i+1)/(N-1))+0.08*cos(4*M_PI*(i+1)/(N-1));
24.                                 else
25.                                         w[i] = w[N-1-i];
26.                                 if(w[i] < 0.0f) w[i] = 0.0f
   ;
27.                         }
28.                         break;
29.                 case 4:
30.                         //printf("Kaiser window.\n");
31.                         for(i = 0; i < N; i++){
32.                                 w[i] = kaiser(i, N, 8, 0);
33.                         }
34.                         break;
35.                 case 5:
36.                         //printf("Blackman-Harris
   window.\n");
37.                         for(i = 0; i < N; i++){
38.                                 w[i] = blackmanharris(i, N)
   ;
39.                         }
40.                         break;
41.                 case 6:
42.                         //printf("Bohman window.\n");
43.                         for(i = 0; i < N; i++){
44.                                 x[i] = (float)(-
   1)+(float)(i*2.0f/(N-1));
45.                                 w[i] = (1.0f-
   (float)fabs(x[i]))*cos((float)M_PI*(float)fabs(x[i]))+1.0f/(floa
   t)M_PI*sin((float)M_PI*(float)fabs(x[i]));
```

```
46.                                         if(w[i] < 0.0f) w[i] = 0;
47.                                 }
48.                                 break;
49.                         case 7:
50.                                 //printf("Bartlett window.\n");
51.                                 for(i = 0; i < N; i++){
52.                                         if(i < N/2){
53.                                                 w[i] = 2.0f*(float)
   i/(float)(N-1);
54.                                         }else{
55.                                                 w[i] = 2.0f-
   2.0f*(float)i/(float)(N-1);
56.                                         }
57.                                 }
58.                                 break;
59.                         case 8:
60.                                 //printf("Dolph-Chebyshev
   window.\n");
61.                                 chebwin(w, N, attcheb);
62.                                 break;
63.                         default:
64.                                 //printf("Default: rectangular
   window.\n");
65.                                 for(i = 0; i < N; i++){
66.                                         w[i] = 1;
67.                                 }
68.                                 break;
69.                 }
70.         }
```

Apéndice B

```
1. void chebwin(float *w, int N, float atten){
2.         int i;
3.         int M = N - 1;
4.         float al;
5.         float A[M];
6.         float W[M];
7.         float max;
8.         int g = atten/20;
9.
10.
11.             al = cosh(acosh(pow(10.0f, g))/(float)M);
12.
13.             for(i = 0; i < M; i++){
14.                     A[i] = fabs(al*cos((float)M_PI*(float)i/(fl
   oat)M));
15.                     if(A[i] < 0)            A[i] = -A[i];
16.
17.                     if(A[i] > 1){
18.                             if(i % 2)       W[i] = -
   cosh((float)M*acosh(A[i]));
19.                             else            W[i] = cosh((float)
   M*acosh(A[i]));
20.                     }else{
21.                             if(i % 2)       W[i] = -
   cos((float)M*acos(A[i]));
22.                             else            W[i] = cos((float)M
   *acos(A[i]));
23.                     }
24.
25.             }
26.
27.             //ifft
28.             float complex * x = (float
   complex*) malloc(M*sizeof(float complex));
29.             float complex * y = (float
   complex*) malloc(M*sizeof(float complex));
30.
31.             for(i = 0; i < M; i++){
32.                     x[i] = W[i] + 0 * I;
33.             }
34.
35.             fftplan
   pb = fft_create_plan(M, x, y, LIQUID_FFT_BACKWARD, 0);
36.             fft_execute(pb);
37.             fft_destroy_plan(pb);
38.
39.             for(i = 0; i < M; i++){
40.                     y[i] = y[i]/(float)M;
41.                     w[i] = creal(y[i]);
42.             }
43.
44.             w[0] = w[0]/2;
45.             w[M] = w[0];
46.             for(i = 0; i < N; i++){
47.                     if(w[i] > max)  max = w[i];
48.             }
49.             for(i = 0; i < N; i++){
```

```
50.                         w[i] = w[i]/max;
51.                 }
52.                 return;
53.         }
```

Apéndice C

```
1. void fir(int n, int Fc, int Fs, float *win, int16_t *coefs){
2.         unsigned int i;
3.
4.         float f = (float)Fc/((float)Fs/2.0f);
5.         float b[n+1];
6.         float max;
7.         float sum;
8.
9.         //Generar una plantilla de puntos con la forma del bode
   del filtro ideal.
10.              int grid_n = 0x8000;                    //Resolució
   m. valor minimo: orden/2
11.              float grid[grid_n+1];
12.
13.              int ne = floor(f*(float)(grid_n+1));
14.              int nb = ne + 1;
15.              float gh[grid_n+1-ne];
16.
17.              for(i = 0; i < ne; i++){
18.                      grid[i] = 1;
19.              }
20.              ne = (float)grid_n+1;
21.
22.              for(i = 0; i < ne-nb; i++){
23.                      gh[i] = (float)i/(float)(ne-nb);
24.                      grid[i+nb-1] = 1-gh[i];
25.              }
26.
27.              //hacer la ifft para cambiar al dominio del tiempo
28.
29.              float complex * x = (float complex*) malloc((2*grid
   _n)*sizeof(float complex));
30.              float complex * y = (float complex*) malloc((2*grid
   _n)*sizeof(float complex));
31.              float fase;
32.              float dt = 0.5*n;
33.
34.              for(i = 0; i < 2*grid_n; i++){
35.                      if(i < grid_n+1){
36.                              fase = -
   dt*(float)M_PI*i/grid_n;                        //fase lineal
   con la frecuencia.
37.                              x[i] = grid[i]*(cos(fase) + I*sin(f
   ase));
38.                      }else{
39.                              x[i] = creal(x[2*grid_n - i]) - cim
   ag(x[2*grid_n - i])*I;       //plantilla final para conseguir
   los coeficientes
40.                      }
41.              }
42.
43.              fftplan
   pr = fft_create_plan(2*grid_n, x, y, LIQUID_FFT_BACKWARD, 0);
44.              fft_execute(pr);
45.              fft_destroy_plan(pr);
46.
47.              //normalizar salida
```

```
48.                 for(i = 0; i < 2*grid_n; i++){
49.                         y[i] = y[i]/(float)(2*grid_n);
50.                 }
51.
52.                 for(i = 0; i < n+1; i++){
53.                         b[i] = creal(y[i]);
                          //salida: parte real de los n+1 primeros
   elementos
54.                         //printf("b1: %f\n", b[i]);
55.                         b[i] = b[i]*win[i];
56.                 }
57.
58.                 /*for(i = 0; i < n+1; i++){
59.                         printf("#: %d\tb2: %f\n", i, b[i]);
60.                 }*/
61.
62.                 //Números de 16 bits
63.                 sum = 0.0f;
64.                 for(i = 0; i < n + 1; i++){
65.                         sum = b[i] + sum;
66.                 }
67.
68.                 for(i = 0; i < n + 1; i++){
69.                         b[i] = b[i]/sum;
70.                         if(max < b[i])  max = b[i];
71.                 }
72.
73.                 for(i = 0; i < n+1; i++){
74.                         if(b[i] >= 0){
75.                                 coefs[i] = round(b[i]/(max*2.0f) *
   (0xFFFF));
76.                                 if(coefs[i] < 0)        coefs[i] =
   0x7FFF;
77.                         }else{
78.                                 coefs[i] = round(b[i]/(max*2.0f) *
   (0xFFFF));
79.                         }
80.                 }
81.       }
```

Apéndice D

```
1. double timeval_diff(struct timeval *a, struct timeval *b){
2.   return
3.     (double)(a->tv_sec + (double)a->tv_usec/1000000) -
4.     (double)(b->tv_sec + (double)b->tv_usec/1000000);
5. }
```

Apéndice E

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <math.h>
#include <liquid.h>
#include <time.h>
#include <sys/time.h>

#define attcheb 80

	void Windowing(int Type, int N, float *out);
	void chebwin(float *w, int N, float atten);
	double timeval_diff(struct timeval *a, struct timeval *b);
	void fir(int n, int Fc, int Fs, float *win, int16_t *coefs)
;

	int main(){
		int N = 4096;
		int Type = 8;
		float w[N+1];
		int16_t o[N+1];
		//Variables para medir el tiempo
		struct timeval t_ini, t_fin;
		double secs;

		printf("Orden: %d\tVentana: %d\n",N,Type);

		gettimeofday(&t_ini, NULL);
		for(int i=0; i < 100; i++){
			//printf("Iteración número %d\n", i);
			Windowing(Type, N+1, w);
			fir(N, 25000, 102400000, w, o);
		}
		gettimeofday(&t_fin, NULL);

		secs = timeval_diff(&t_fin, &t_ini);
		printf("%.16g milisegundos\n", secs * 1000.0);

		/*
		printf("\n");
		for(int i = 0; i < N-1; i++)	printf("#: %d\tw:
%d\n", i, o[i]);
		FILE *f;
		f = fopen("win4096.txt", "w");
		for(unsigned int i = 0; i < N; i++){
			fprintf(f, "%d\n", o[i]);
		}
		fclose(f);*/
	}
```