Universidad de Oviedo

Memoria del Trabajo Fin de Máster realizado por

Mario Roos Hoefgeest Toribio

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

Tutor: Ignacio Álvarez García

SISTEMA DE ADQUISICIÓN DE DATOS 3D BASADO EN SISTEMA EMBEBIDO TIPO SOC PARA SENSOR CONOPOINT

FEBRERO DE 2019

ÍNDICE

ÍNDICE DE LA MEMORIA

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

MEMORIA

1. INTRODUCCIÓN

Las técnicas de medición sin contacto suponen un paso más allá respecto a los métodos de medida tradicionales. Hasta hace no mucho tiempo, incluso en muchos procesos de la actualidad, los controles de calidad y sistemas de inspección de un proceso productivo se basaban en la inspección manual de algunas muestras aleatorias.

El principal problema de las técnicas de medidas por contacto es la limitada velocidad de medida, ya que velocidades grandes supondrían impactos entre el elemento de medida y la pieza evaluada, pudiendo provocar defectos en ambas partes. Esto hace muy complicado integrar estas técnicas en línea, ya que la capacidad de producción se vería muy reducida.

Las técnicas de medición sin contacto vienen a solucionar este problema, ya que, al evitarse la posibilidad de impactos, la única limitación de velocidad será la inherente a la técnica escogida. Como, por lo general, estas técnicas permiten la inspección a velocidades muy superiores, es posible evaluar prácticamente por completo la producción sin tener un impacto significativo en la velocidad del proceso. Estos sensores pueden basarse en técnicas tan dispares como la visión artificial convencional, ultrasonidos, técnicas interferométricas…

1.1 OBJETIVOS

El presente trabajo fin de máster se realiza en la empresa Desarrollo Soluciones Integrales Plus (DSIplus), con el fin de ampliar las capacidades actuales de un sensor Conopoint [1], un sensor para mediciones de distancia sin contacto basado en holografía conoscópica. DSIplus es una empresa con amplia experiencia en el sector. Lleva desde 2007 integrando este tipo de sensores en proyectos de inspección dimensional o detección de defectos en línea.

El sensor Conopoint [1] permite obtener mediciones puntuales de alta precisión sin contacto utilizando técnicas de interferometría por holografía conoscópica. Sin embargo, en su versión actual, la adquisición de datos para su procesamiento y la sincronización con los movimientos necesarios para obtener una

escena 3D son limitados: conexión únicamente con PC, límite de 3 encóders, sin posibilidad de generar movimientos y trayectorias.

El objetivo del trabajo propuesto es realizar un diseño y programación basados en una tarjeta electrónica con procesador System on Chip (SoC), que incluye un procesador ARM y una FPGA, de forma que se mejoren ambas capacidades en la adquisición y tratamiento de datos de sensores ConoPoint.

El trabajo consistirá en el diseño hardware y programación de una tarjeta madre para Microzed [2], con procesador SoC Xilinx Zynq 7010, que permita la adquisición de datos de un sensor de distancias Conopoint, mediante protocolo UDP, y su sincronización con otros sensores y/o acciones de movimiento. Será necesario adaptar el SDK de comunicación con el sensor Conopoint, que actualmente funciona en Linux o Windows, para que se ejecute en un SoC que funcionará en modo *baremetal*, sin sistema operativo.

Actualmente, el sensor de holografía conoscópica permite sincronizar en tiempo real la medida de profundidad dada con el láser con hasta tres señales de encóder, entregando la información en la misma trama UDP. De esta manera el sensor se puede colocar en un sistema de tres ejes como se representa en la figura.

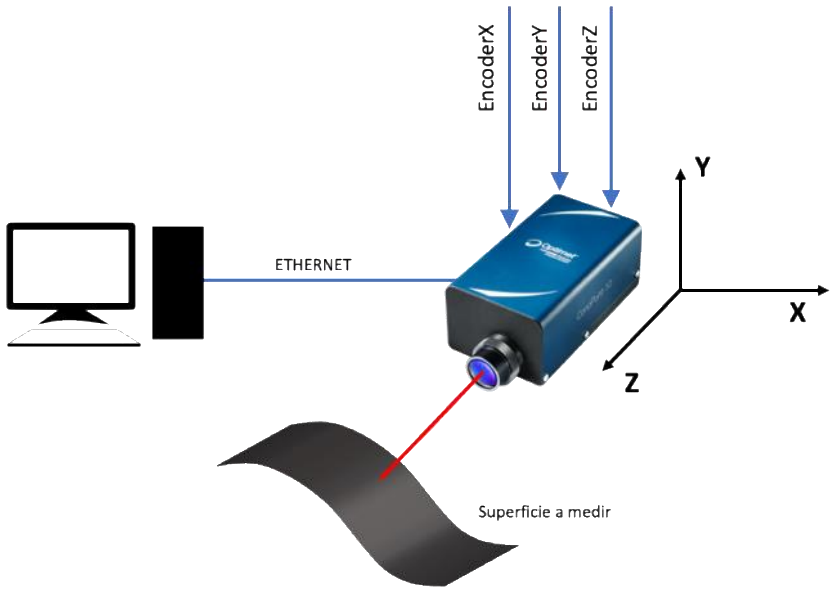

Figura 1. Estado actual.

El sistema desarrollado tomará la información de distancia del sensor de holografía, y permitirá sincronizar la adquisición con hasta seis señales de encóder que se conectarán directamente a la tarjeta electrónica. De esta forma se podrá saber con precisión la posición y orientación exacta del sensor cuando este esté colocado en un sistema de hasta seis ejes, lo que permitirá reconstruir con más precisión la escena 3D. Además, se podrán generar señales analógicas que permitan controlar el movimiento tanto de motores como de espejos que permitan dirigir el haz láser. Esto permitirá una configuración cómo la de la siguiente figura.

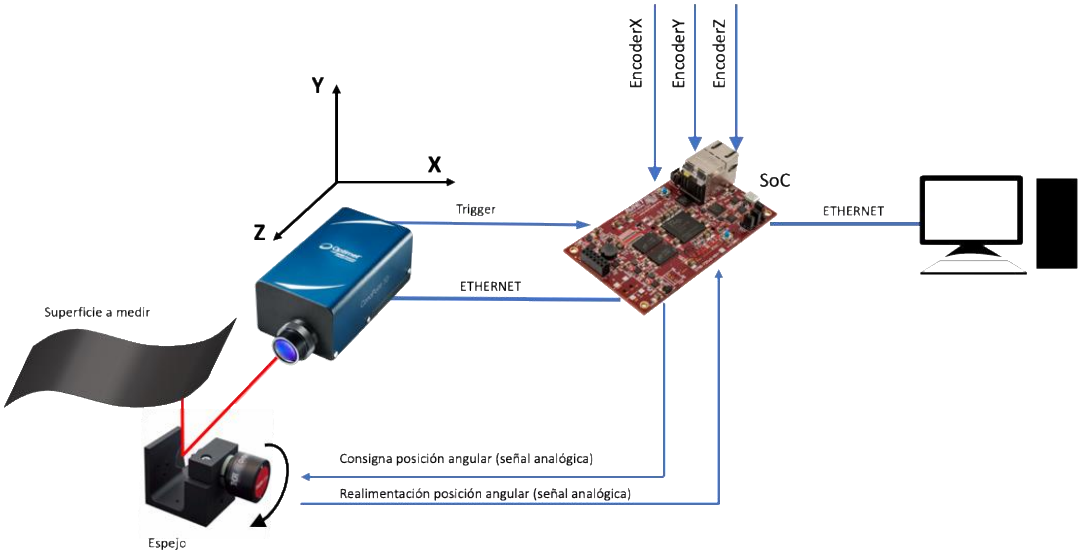

Figura 2. Objetivo deseado.

Es interesante el uso de galvos o espejos combinados con la tecnología de holografía conoscópica, ya que el haz láser puede redirigirse para lograr medidas complejas, como es el caso de superficies interiores usando un periscopio. En este caso, el uso de un espejo móvil permitirá que el sensor pueda medir una línea, y no solo un punto, para una posición fija del sensor.

1.2 ESTADO DE LAS TÉCNICAS DE MEDICIÓN

1.2.1 Técnicas de medición con contacto.

En la industria, lo más habitual es encontrar sistemas de medición con contacto, con las limitaciones en la velocidad y posicionamiento de la pieza que esto implicca. Dentro de estas técnicas se pueden encontrar:

1.2.1.1 Reloj comparador

Un reloj comparador se utiliza para comparar cotas mediante la medición del desplazamiento de una punta de contacto esférica cuando el instrumento está fijo sobre un soporte.

Este instrumento permite realizar controles dimensionales de manera sencilla llegando a resoluciones de hasta milésimas de milímetro.

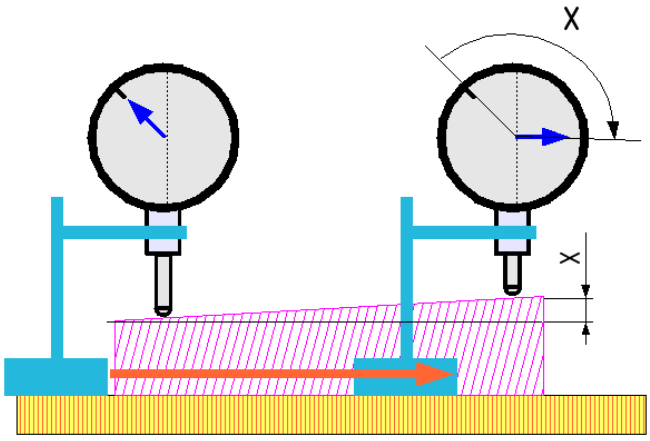

Figura 3. Ejemplo de uso de un reloj comparador para medidas de planitud.

1.2.1.2 Máquinas de medición por coordenadas.

Son instrumentos de medida de precisión capaces de reconstruir en el espacio la posición de distintos puntos de una pieza, de forma que se puedan medir los parámetros que interesen sobre el modelo reconstruido. Estas máquinas se basan en un palpador mecánico colocado en el extremo de un sistema robótico de múltiples ejes, de manera que se pueda saber con precisión la posición del palpador en todo momento.

Estas máquinas pueden llegar a resoluciones inferiores a la micra, pero presentan limitaciones en cuanto a velocidad y posicionamiento de la pieza.

Figura 4. Máquina de medición por coordenadas.

1.2.2 Técnicas de medición sin contacto

Las técnicas de medición sin contacto vienen a solucionar los problemas que presentaban los anteriores, aunque puedan presentar otros problemas asociados a cada tecnología.

1.2.2.1 Visión artificial

Las técnicas de visión artificial pueden dividirse en varias categorías:

- Monoculares:

En esta categoría se engloban las técnicas de visión artificial que buscan obtener información 3D del entorno utilizando una sola cámara. Algunos ejemplos son los basados en análisis de textura; *shape from shading*, que obtiene información tridimensional analizando las sombras; o *dynamic focusing*, que consiste en ir moviendo el foco de la lente para analizar que partes de la imagen quedan enfocadas.

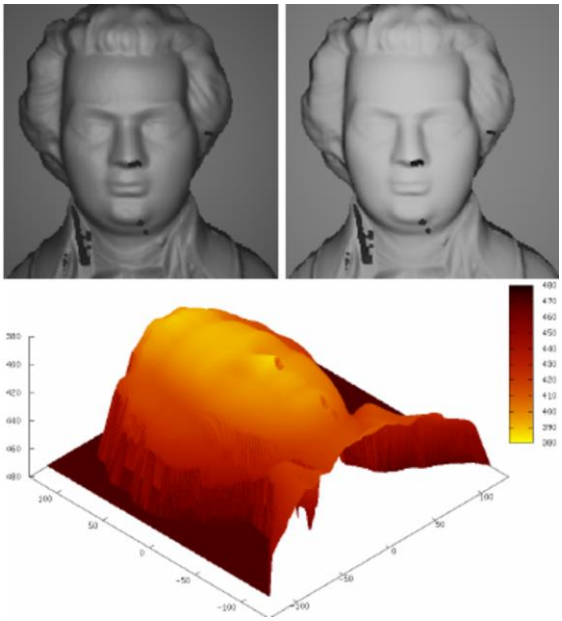

Figura 5. Ejemplo de Shape of Shading utilizando la cara de Mozart. [3]

- Estereoscópicas

Estas técnicas utilizan dos o más cámaras para extraer la información tridimensional de una escena. Aquí se engloban las basadas en fotogrametría [4], que pueden ser utilizadas tanto para construir modelos desde pequeños objetos hasta grandes escenas, como una ciudad. Un ejemplo concreto es la correlación de imagen digital (DIC), que se utiliza para medir contornos, deformaciones, vibraciones...

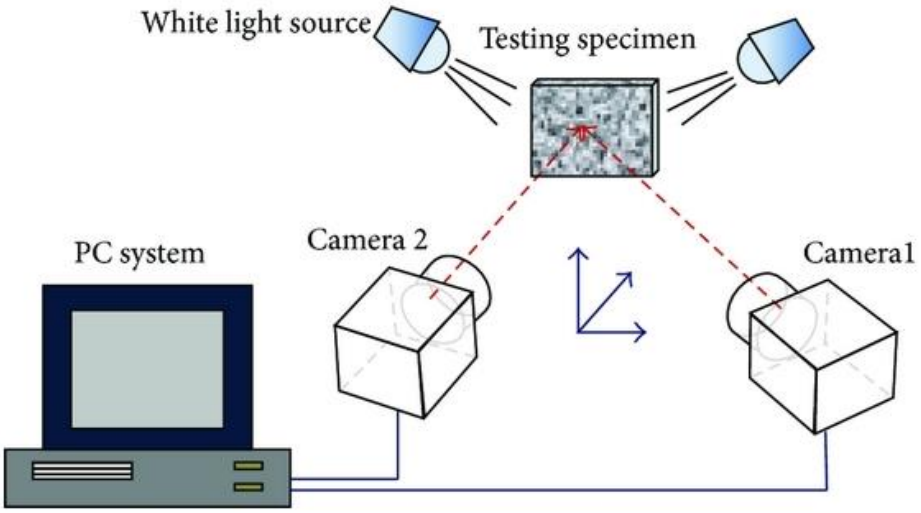

Figura 6. Correlación de imagen digital.

- Triangulación laser.

Para reconstruir con precisión puntos o perfiles, aparecen tecnologías como la triangulación láser. Se basan en proyectar un punto o una línea sobre un objeto y, observando con una cámara desde una posición distinta ver como se deforma el patrón proyectado. En función de el ángulo entre el emisor láser y la cámara varía la resolución en la medida. Con grandes ángulos pueden llegarse a resoluciones por debajo de las micras.

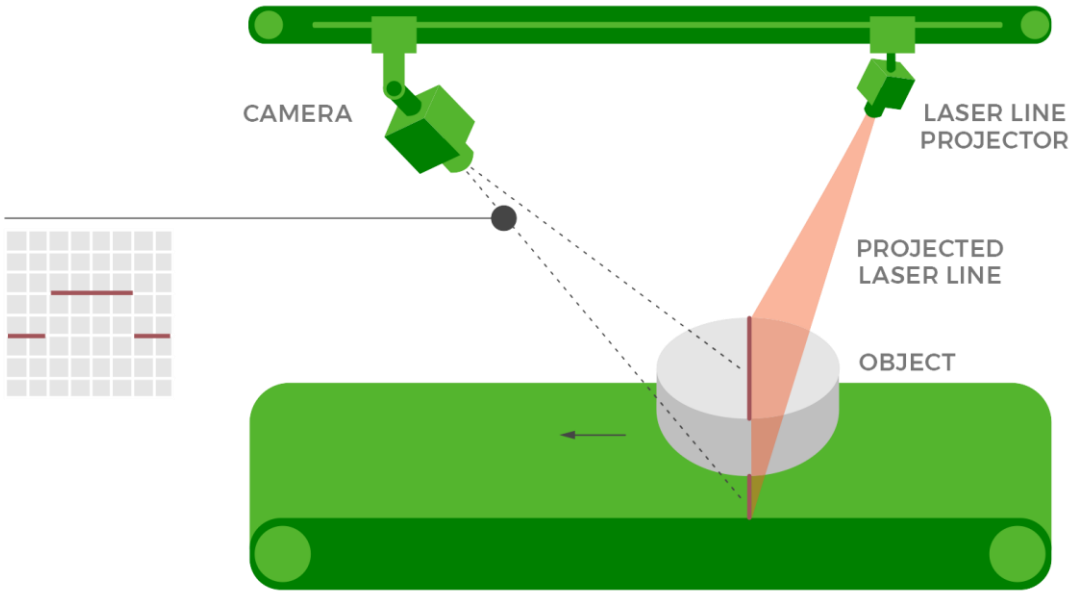

Figura 7. Triangulación láser.

- Técnicas de luz estructurada.

Se basan en proyectar un patrón; por ejemplo, franjas o puntos; sobre una escena. Analizando cómo se deforma el patrón es posible construir un modelo de la escena [5]. Algunos ejemplos conocidos que ese basan en estas técnicas son la *Kinect*, de Microsoft, o la *TrueDepth Camera* del iPhone.

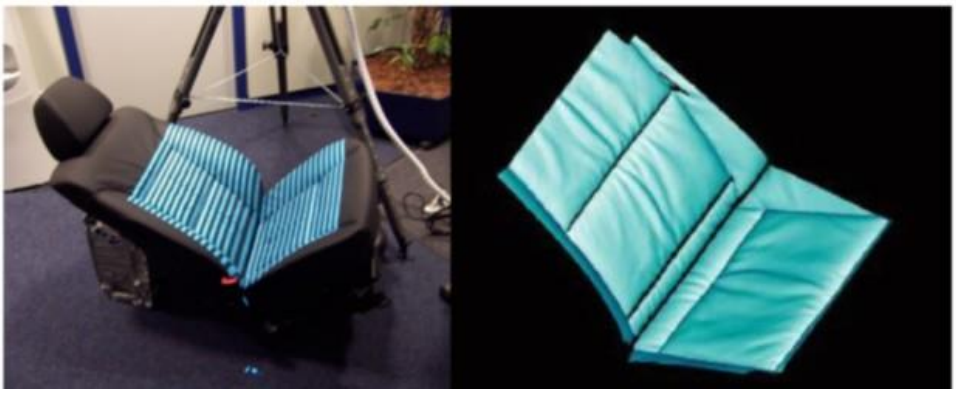

Figura 8. Reconstrucción de un sillón con proyección de luz estructurada.

1.2.2.2 Tiempo de vuelo

Los sensores de tiempo de vuelo se basan en emitir una señal y detectarla tras su rebote con una superficie. Jugando con la modulación de la señal emitida, se puede medir el desfase con la onda reflejada, aumentando la precisión del sistema de medida. Basados en estas técnicas existen sensores basados en láser, microondas o ultrasonidos. Los sensores láser pueden llegar a resoluciones de décimas de milímetro.

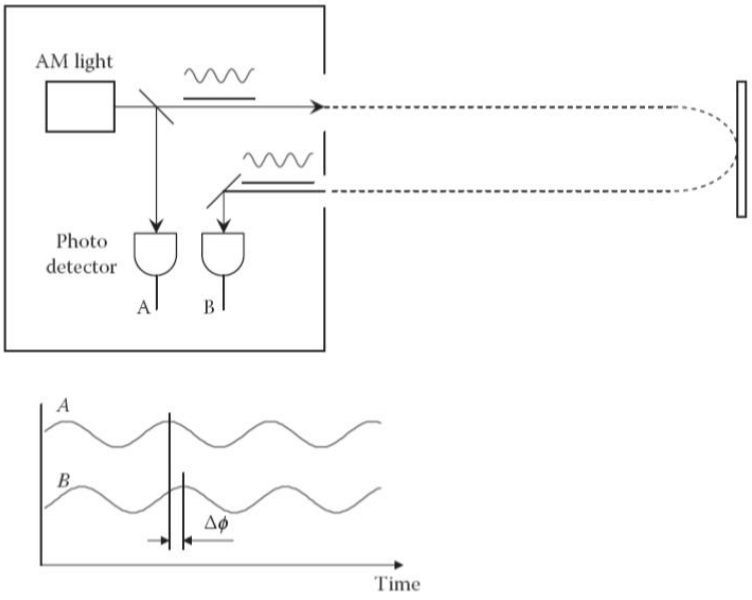

Figura 9. Ilustración un sistema de medida de tiempo de vuelo basada en modulación de una señal láser.

1.2.2.3 Barrera

Los sensores de barrera se basan en un emisor que emite una línea láser enfrentado a un receptor. Si un objeto se encuentra entre el receptor y el emisor habrá

oclusión. Según la cantidad de luz que llega al receptor pueden realizarse distintos tipos de mediciones. Estos sensores pueden llegar a resoluciones de centésimas de milímetro.

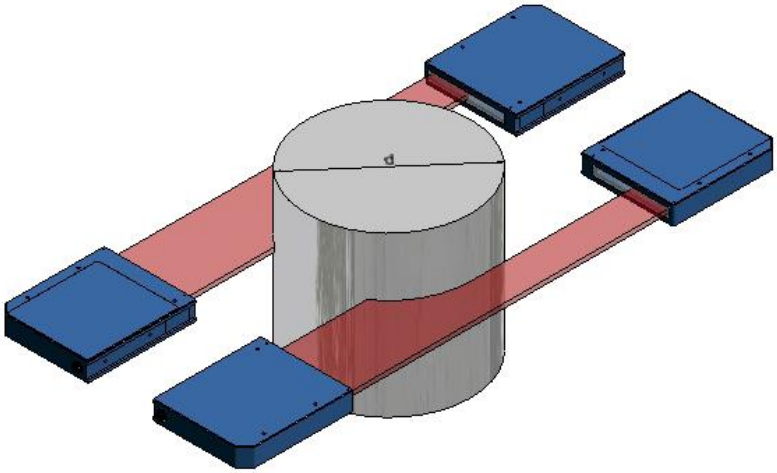

Figura 10. Sensor de barrera midiendo el diámetro de un tubo.

1.2.2.4 Interferometría

Esta familia de técnicas se basa en la interferencia que se produce cuando se superponen dos ondas electromagnéticas. Cuando dos ondas electromagnéticas de la misma frecuencia interfieren se genera un patrón que depende de la diferencia de fase entre las dos ondas. Si ambas dos ondas están en fase, se genera una interferencia constructiva, mientras que si están en oposición de fase la interferencia es destructiva. El patrón de intensidades se produce cuando la diferencia de fase está en una situación intermedia entre los dos casos. Analizando el patrón se puede conocer el desfase entre las dos ondas, que está directamente relacionado con la diferencia entre los caminos ópticos de las dos ondas. Lo que se puede utilizar para medir distancias incluyendo el espécimen a medir en el montaje óptico.

Este principio se utiliza en diversas técnicas, como son la holografía [6], interferometría láser [7], interferometría speckle [8] o la holografía conoscópica [9], técnica que utiliza el sensor utilizado en el proyecto.

La holografía conoscópica puede llegar a resoluciones similares a la triangulación láser, pero con las ventajas de ser colineal. Esto es, el punto láser proyectado viaja en la misma línea que el eje óptico de la cámara, por lo que es

posible utilizarlo para medir en cualquier tipo de hueco o, incluso, redirigir la luz con periscopios o espejos. De esta manera se evitan los problemas de oclusión inherentes a una triangulación.

Su principio de funcionamiento será explicado en el apartado siguiente.

2. MARCO TEÓRICO

2.1 HOLOGRAFÍA CONOSCÓPICA

La holografía conoscópica es una técnica interferométrica que se basa en la propiedad doble refracción de la luz en cristales uniaxiales. Esta propiedad fue descubierta por Sirat y Psaltis en 1985 en el Instituto Tecnológico de California [9].

En la figura se puede observar el principio de funcionamiento de la holografía conoscópica [10].

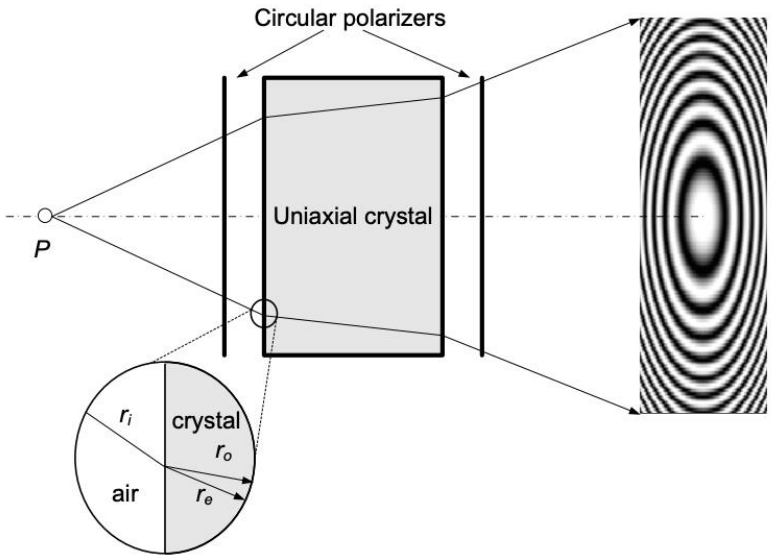

Figura 11. Principio de funcionamiento de la holografía conoscópica.

Cuando un haz de luz monocromática polarizada entra en un cristal uniáxico, se divide en dos rayos con polarizaciones ortogonales, llamado rayo ordinario y extraordinario. El rayo ordinario presenta una velocidad constante, mientras que la del extraordinario varía en función del ángulo de incidencia, directamente relacionado con la distancia al punto observado. A la salida del cristal, los rayos estarán desfasados en función de la diferencia de velocidades. Al colocar un polarizador, los dos rayos podrán interferir entre sí produciendo un patrón de interferencia. Analizando este patrón es posible conocer con precisión la distancia entre el punto P y el sensor.

2.2 MICROZED

Los SoC (*system on chip*) son dispositivos que incorporan, en un único circuito integrado, gran parte de los módulos que constituyen un sistema informático. Estos sistemas constituyen la tendencia actual del mercado debido a la potencial reducción de tamaño de los distintos equipos y el ahorro energético que supone que los distintos componentes se integren en el mismo circuito.

En este proyecto se utiliza la tarjeta de desarrollo MicroZed 7010 [2], basada en el SoC Zynq-7010 [11] de Xilinx. La placa incluye, además 100 pines de entrada/salida, 1GB de memoria DDR2 SDRAM, 128Mb de memoria flash QSPI, puertos Ethernet Gigabit, USB 2.0 y USB-UART.

Figura 12. Placa MicroZed.

La principal ventaja que ofrece el SoC Zynq-7010 [11] es que integra un procesador basado en ARM Cortex-A9 [12] con una FPGA. Esto supone un gran incremento en la capacidad de ejecutar tareas de tiempo real, liberando la carga del procesador. Además, aporta una gran versatilidad al sistema al poder programar en *hardware* tareas como gestión de entradas y salidas o generación de señales, de forma que se ejecuten de forma paralela.

La tarjeta MicroZed se puede dividir en dos grandes bloques: el PS, con sus dos procesadores ARM, y el PL, que constituye la lógica programada. A grandes rasgos, el sistema se ilustra en la siguiente figura:

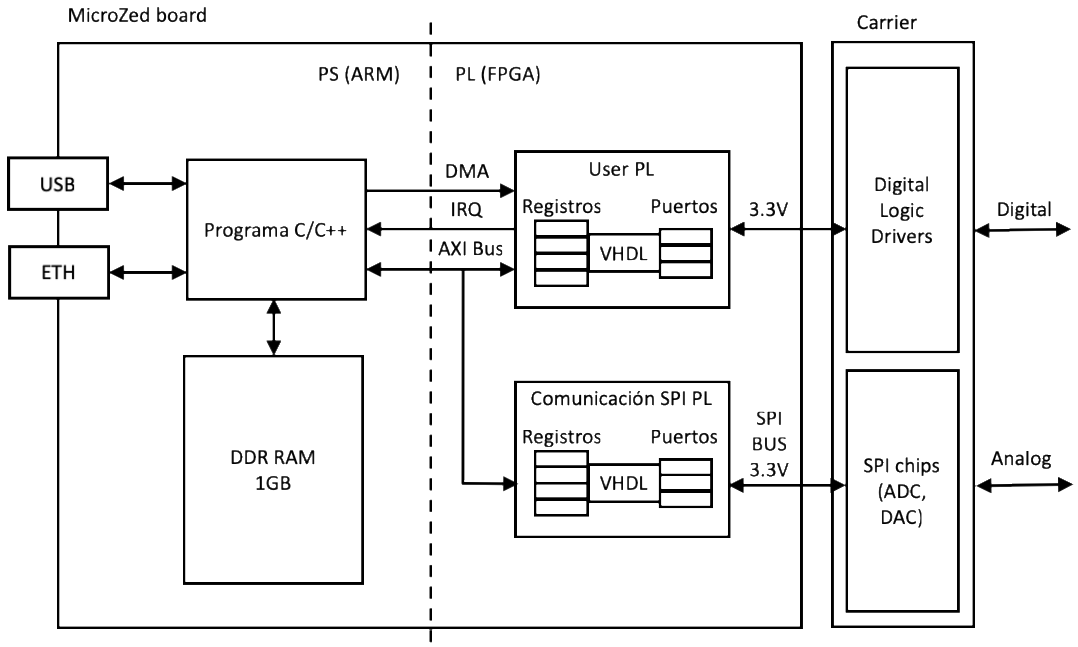

Figura 13. Diagrama de bloques simplificado de la placa MicroZed.

El PS y el PL constituyen dos partes diferenciadas que se podrán comunicar a través de tres vías, que serán explicadas con detalle en los próximos apartados, un bus AXI, DMA e interrupciones.

En el PS se programarán las tareas de más alto nivel, como es la comunicación con el sensor y el PC, a través del puerto ethernet, dejando las tareas con restricciones temporales para el PL. En el PL se implementarán la generación de señales y la comunicación SPI con los sensores AD y DA. De esta manera será posible sincronizar la señal analógica leía por el conversor AD SPI, con la señal de disparo del sensor de holografía conoscópica y con la trama que envía el sensor por ethernet y lee el PS.

Una vez dada una visión general del sistema, es conveniente explicar más en detalle las partes más relevantes de la arquitectura del SoC Zynq-7000, que se describe en la siguiente figura.

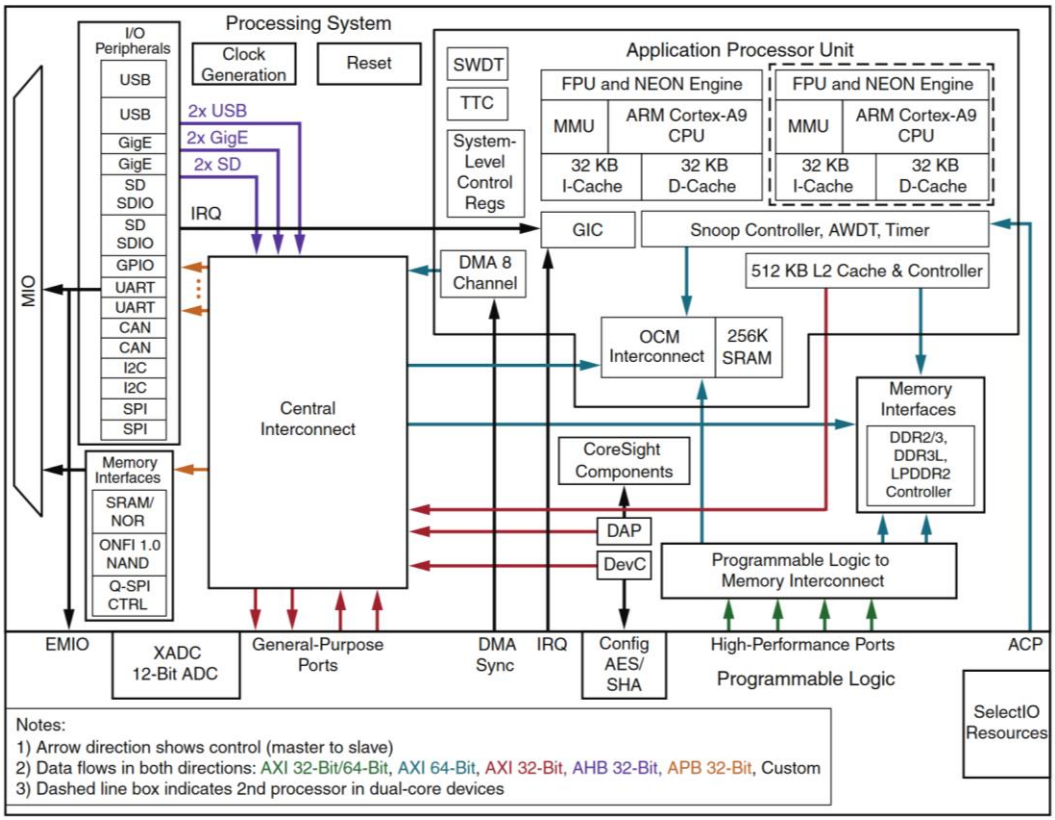

Figura 14. Arquitectura Zynq-7000 SoC. Imagen obtenida de Xilinx [11].

La figura ilustra cómo el SoC se divide en dos partes principales, el PS (*Processing System*) y el PL (*Programmable Logic*). El PS incluye el microprocesador ARM de dos núcleos, junto a las interfaces y puertos de entrada/salida; mientras que el PL representa la FPGA. Ambas partes pueden comunicarse entre sí a través de distintos puertos basándose en el protocolo AXI, que se explicará más detalladamente en el apartado 2.2.3.

2.2.1 Puertos de entrada/salida

Como puede verse en la parte izquierda de la Figura 14, Zynq permite la conexión del procesador con múltiples interfaces estándar a través del bloque MIO (*Multipurpose IO*).

El MIO es fundamental para la conexión con periféricos debido al limitado número de pines. Mediante software se puede programar la conexión de las señales de entrada a los 54 pines del MIO, que pueden configurase como:

- *SRAM / NOR Flash memory interface*.

- *NAND Flash memory interface.*
- Dos *10/100/1000 Ethernet MACs.*
- Dos interfaces USB 2.0 OTG.
- Dos SD Card interfaces.
- Dos UARTs.
- Dos intefaces maestro-esclavo I2C.
- Dos interfaces *full-duplex* SPI.
- Dos interfaces CAN 2.0B.
- PJTAG y TRACE *debug interfaces*.
- *Triple timer/counter* (TTC).
- *System watchdog timer*.

El MIO se divide en dos bancos, MIO0 y MIO1, que pueden configurarse para funcionar a distinta tensión. El MIO0 comprende los pines del 0 al 15, mientras que el MIO1 del 16 al 53. El banco 0 incluye pines de configuración, que son leídos al encender el sistema. Los pines del 2 al 8 son utilizados para definir el modo de arranque, uso de PLLs y la tensión de trabajo de los bancos 0 y 1.

Además del MIO, se tiene un bloque EMIO (*Extended Multipurpose IO*), que permite extender el bloque MIO hacia la FPGA, añadiendo otros 64 pines de entrada/salida adicionales.

El sistema de entrada/salida puede resumirse en el siguiente diagrama. Para una información más detallada, ver el capítulo 2 del manual de referencia del SoC Zynq-7000 [13, Cap. 2].

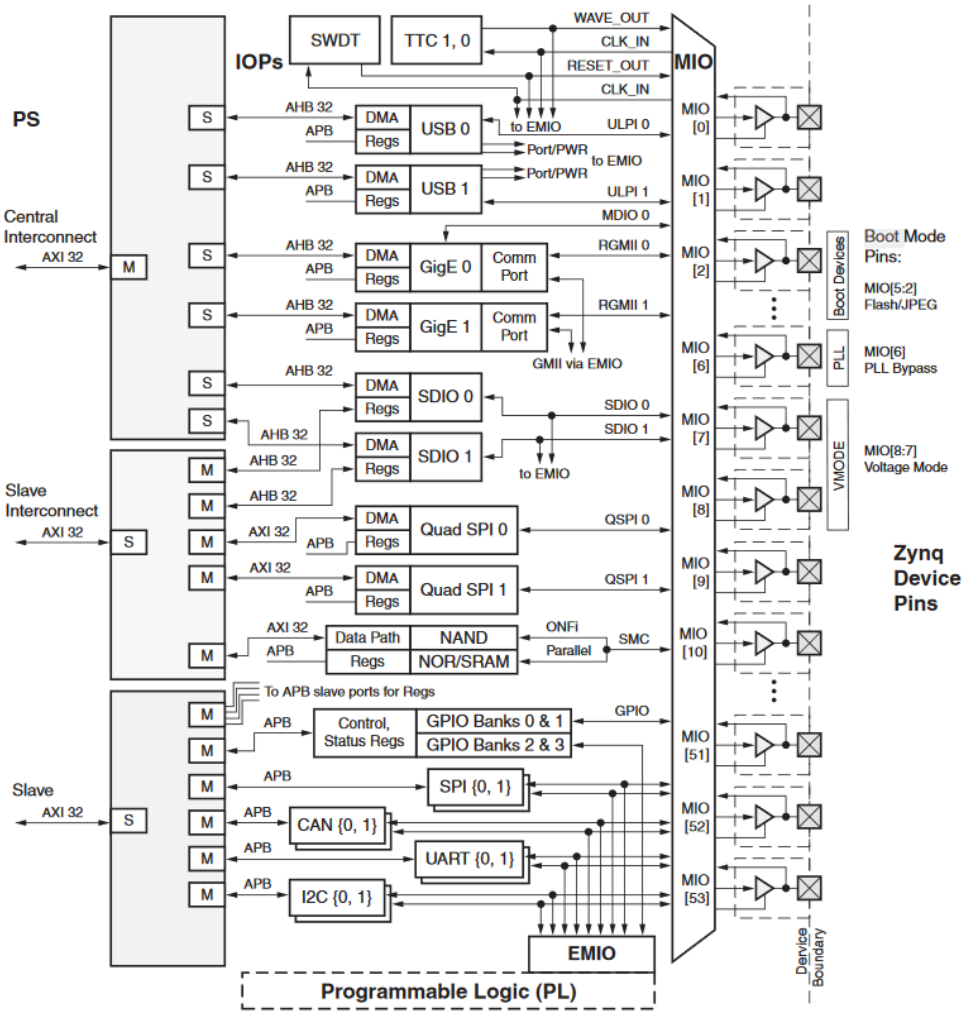

Figura 15. *I/O Peripherals System Diagram*.

Los pines de entrada y salida pueden ser controlados tanto desde el PL como desde el PS, de forma bastante sencilla. Cuando dirige el procesador ARM, se utilizarán unas macros definidas en `xgpiops.h` [14], que se incluye por defecto al crear un proyecto en modo *standalone*. Desde la lógica programada las conexiones con los pines de entrada y salida se definen mediante las denominadas *constraints.*

2.2.2 Constraints

Las *constraints*, o restricciones, constituyen un área relevante en el diseño de una aplicación en la lógica programada. Se utilizan para definir desde la frecuencia de operación de los relojes del sistema, hasta es uso de los pines de entrada y salida.

Para la programación de la FPGA se utilizan las *Xilinx® Design Constraints* (XDC) [15]. Constituyen una mezcla del estándar SDC (*Synopsys Design Constraints*)

y restricciones físicas propiedad de Xilinx. Las restricciones se interpretan como si fueran comandos Tcl [16].

Las restricciones pueden dividirse en varios grupos:

- Restricciones temporales: se utilizan para definir las frecuencias de los relojes y *delays* máximos entre distintos bloques. Se ilustra dicho retardo en la Figura 16.
- Excepciones temporales: definen excepciones a las restricciones anteriores.
- Restricciones físicas: pueden definir conexiones con pines de entrada/salida, ubicación física del diseño en celdas concretas o rutas fijas entre celdas.

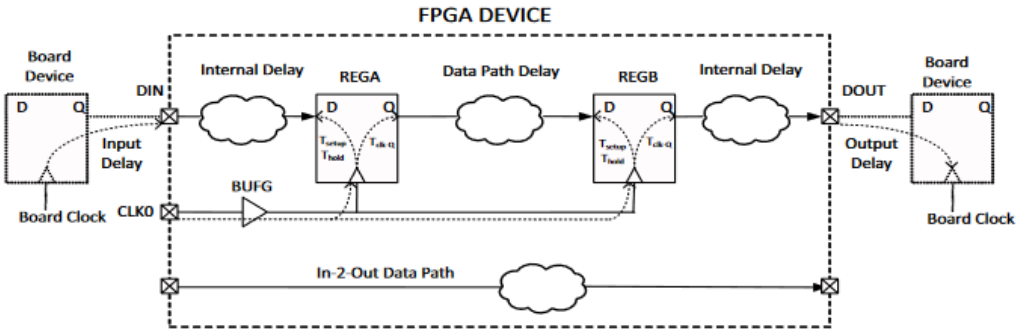

Figura 16. Ejemplo de *timing path*. Representa el retraso que se produce desde que se genera un pulso de reloj hasta que tiene lugar el correspondiente efecto en DOUT.

2.2.3 AXI4

AXI es el protocolo escogido por Xilinx para establecer comunicaciones eficientes entre IP cores y facilitar enormemente la tarea de diseño, al descargar al desarrollador del diseño este tipo de soluciones. Un IP Core (*intelectual property core*) es cada uno de los bloques lógicos usados en la programación de FPGAs.

AXI forma parte de ARM AMBA [17], un estándar de comunicación *On-Chip* que se remonta a 1996. AMBA 4.0, lanzada en 2010, incluye la última revisión del protocolo AXI, AXI4.

AXI4 presenta tres tipos de interfaces:

- AXI4: para transferencias de alto rendimiento mapeadas en memoria.
- AXI4-Lite: versión simplificada que permite transferencias sencillas sin grandes requerimientos de rendimiento, como registros de estado o control.
- AXI4-Stream: pensada para transferir secuencias de datos a gran velocidad.

Las especificaciones describen una interfaz entre dos bloques, maestro y esclavo, que representan dos *IP cores* intercambiando información. Para la conexión de más bloques entre sí, Xilinx proporciona distintos *IP cores*, como *Xilinx AXI Ineterconect IP* [18] o AXI *SmartConnectIP* [19].

Las interfaces AXI4 y AXI4-Lite presentan cinco canales diferentes: lectura y escritura de datos, lectura y escritura de direcciones, y uno de respuesta tras escritura para indicar si se ha completado con éxito.

Los datos pueden moverse simultáneamente de maestro a esclavo y viceversa. La diferencia fundamental entre AXI4 y AXI4-Lite es que la última permite solo la transferencia de un dato por transacción, mientras que AXI4 hasta 256.

En las dos siguientes figuras puede observarse de manera simplificada cómo se llevan a cabo las transferencias de datos, tanto en operaciones de lectura como escritura.

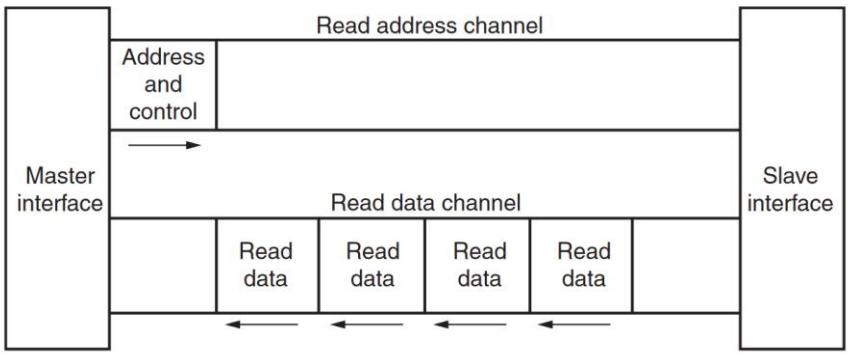

Figura 17. Secuencia de lectura de datos.

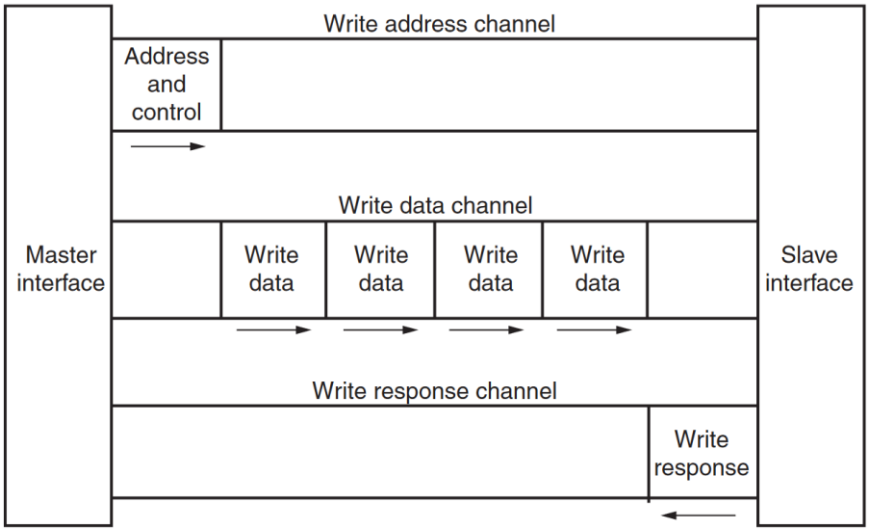

Figura 18. Secuencia de escritura de datos.

El protocolo AXI-Stream define un único canal de transmisión, equivalente al canal de escritura de datos de AXI. La ventaja de este protocolo es que puede transferir una cantidad ilimitada de datos por transferencia, sin la limitación de 256.

2.2.4 DMA

En el lado del PS se puede observar otro bloque relevante, el DMA (*Direct Memory Access*), destinado a transferir datos de manera eficiente entre distintas partes del sistema. El bloque DMA presenta ocho canales, cuatro de ellos dedicados al PL, que permiten cuatro tipos de transferencia de datos: memoria a memoria, memoria a PL, PL a memoria y *scatter-gather*; un modo que permite transferencias más eficientes de datos no alineados en memoria.

En la siguiente figura se muestra la secuencia de comunicación que debe seguir un periférico con el controlador DMA para establecer la comunicación. En el diagrama se usan los prefijos DR para indicar el bus de *request* del periférico, y DA para el bus de ACK del controlador.

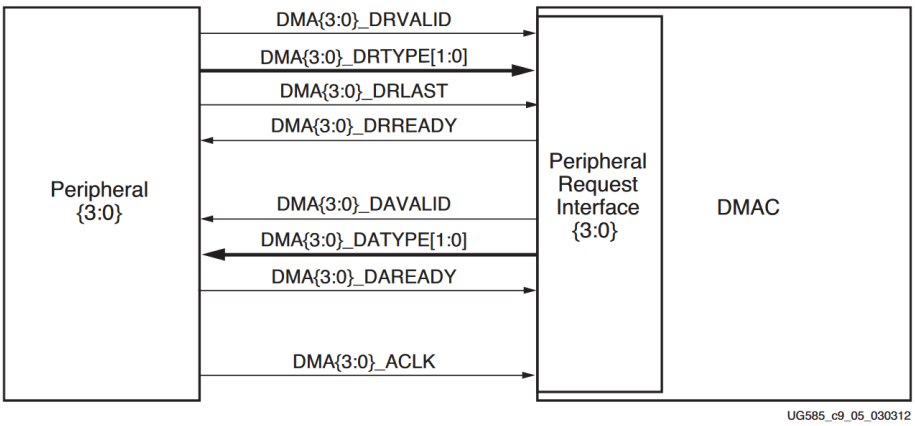

Figura 19. Señales *Request/Acknowledge* en la comunicacación con el DMAC.

Ambos buses usan el *valid-ready handsake* descrito en el protocolo AXI [20], en el que la señal *ready* indica cuándo el bus está preparado para la comunicación y la señal *valid* se mantiene a nivel alto durante la transferencia de datos.

El periférico utiliza los registros DMA{3:0}_DRTYPE[1:0] para seleccionar el tipo de comunicación, mientras que el DMAC utiliza el DMA{3:0}_DATYPE[1:0] para indicar cuándo se completa la transacción.

El registro DMA{3:0}_DRLAST sirve para indicar al DMAC el comienzo del último ciclo de reloj de la transferencia.

La secuencia de señales se describe en la figura siguiente:

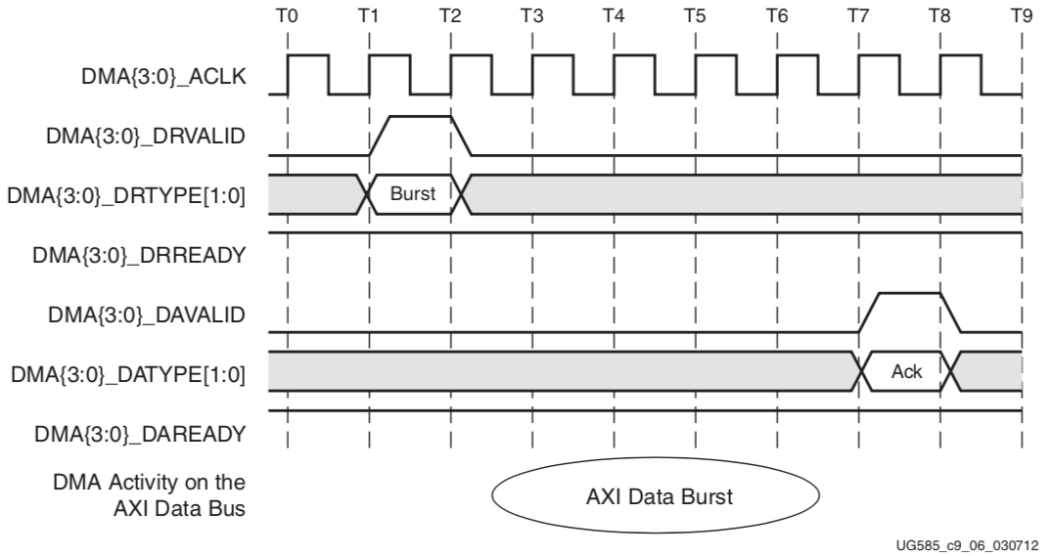

Figura 20. Cronograma de la comunicación entre el periférico y el DMAC.

En la figura se puede observar la secuencia antes descrita. En T1, el periférico solicita la transacción, que tiene lugar entre T2 y T7 en el bus de datos, momento el que el DMAC indica el fin de la comunicación.

2.2.5 Interrupciones

El SoC utiliza un *ARM Generic Interrupt Controller (GIC)* para procesar las interrupciones.

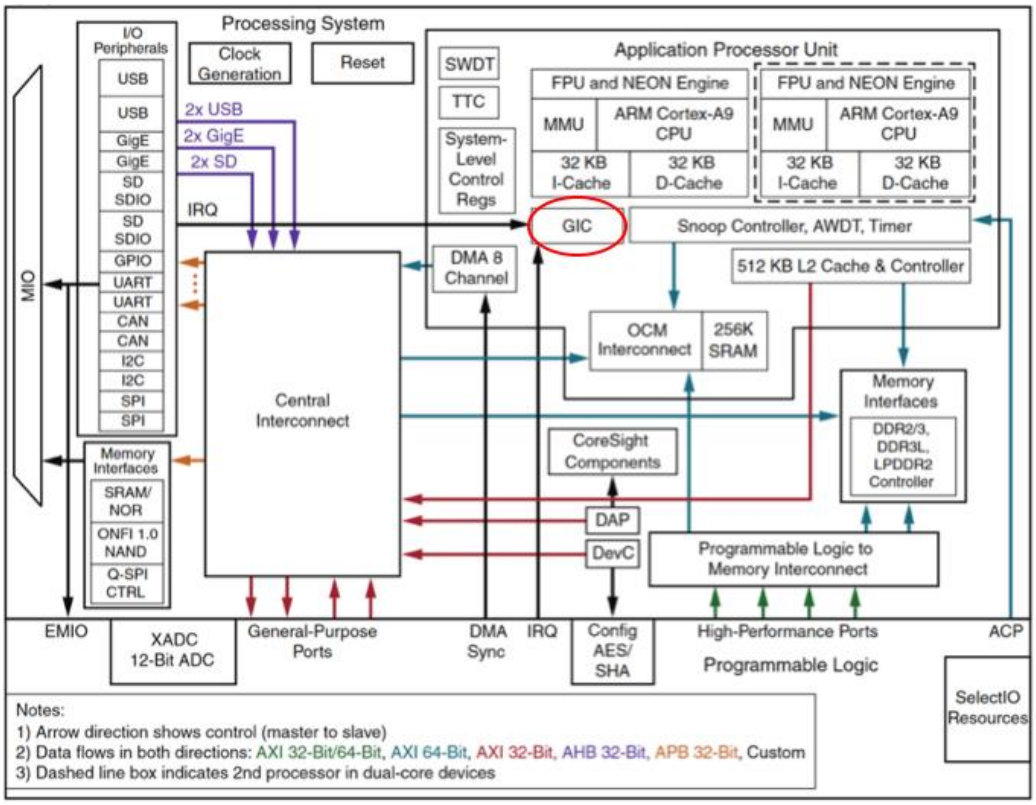

Figura 21. GIC rodeado en rojo. Nótese que está comunicado con el PL y con los periféricos de entrada/salida.

Hay tres tipos de interrupciones:

- Generadas por software: hasta 16 por procesador. Pueden interrumpir la ejecución del otro núcleo o del mismo que la ha generado.
- *Shared Peripheral Interrupts*: hasta 60 interrupciones. Su origen puede venir tanto del PS, 44 desde los periféricos de entrada y salida, y 16 de la lógica programable (PL). Son compartidas por ambos procesadores.

- *Private Peripheral Interrupts*: Son 5, privadas de cada procesador. Corresponden a los *timers* de las CPUs, los *Watchdogs* y una dedicada para interrupciones generadas en el PL.

Como para atender una interrupción se suspende la ejecución normal del programa, puede darse la situación en que durante la rutina de atención a la interrupción salte una interrupción diferente, o que salten varias simultáneamente. Para gestionar estas situaciones a cada interrupción se le asigna una prioridad, de manera que se despachen primeros las de prioridad más alta.

2.2.6 Ethernet

El *SoC Zynq* presenta *capacidades Ethernet Gigabit*, lo que permite implementar unas comunicaciones rápidas y flexibles en las aplicaciones desarrolladas.

El SoC incluye dos controladores *Ethernet Gigabit*, que pueden ser configurados independientemente. El controlador implementa el MAC, control de acceso al medio, y permitirá implementar distintos protocolos de comunicación, como TCP/IP [21] o UDP [22]. En la aplicación actual se implementará comunicación basada en el protocolo UDP, puesto que es el utilizado por el sensor de holografía conoscópica.

2.2.6.1 UDP

UDP (*User Datagram Protocol*) es un protocolo que implementa la capa de transporte del modelo OSI [23] de ISO. Es un protocolo no orientado a la conexión, que proporciona una interfaz mínima entre las capas de red y transporte.

Este protocolo no garantiza que un paquete llegue a su destino, aunque sí su integridad. Aunque a priori pueda suponer un problema que no se garantice la llegada el paquete, la probabilidad de que estos se pierdan es mínima, ya que no se usarán arquitecturas de red complejas. La comunicación será directa, utilizando, a lo sumo, un *switch* entre el sensor y la placa.

La composición de los paquetes es muy sencilla. Tiene cuatro campos de 16 bits: puertos de origen y destino, longitud del mensaje en *bytes* y *checksum*, para corroborar la integridad del mensaje, su cálculo incluye también una *pseudocabecera*

IP, que aporta información de las direcciones IP de origen y destino, así como la longitud total del paquete UDP. El paquete completo resulta como el indicado en la figura.

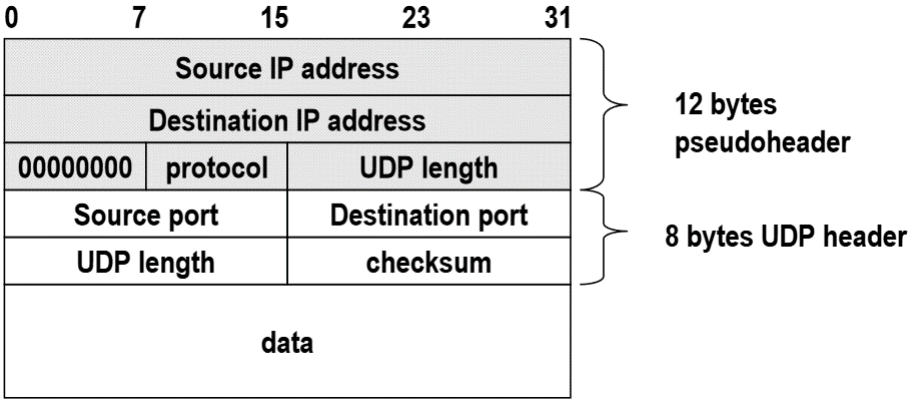

Figura 22. Paquete UDP.

3. DISEÑO IMPLEMENTADO

3.1 DISEÑO HARDWARE

El sistema se compone de cinco elementos diferentes dispuestos como en la Figura 23: la placa *microzed* [2], el sensor de holografía conoscópica Conopoint [1], un galvo y conversores AD y DA que se comunican por SPI [24].

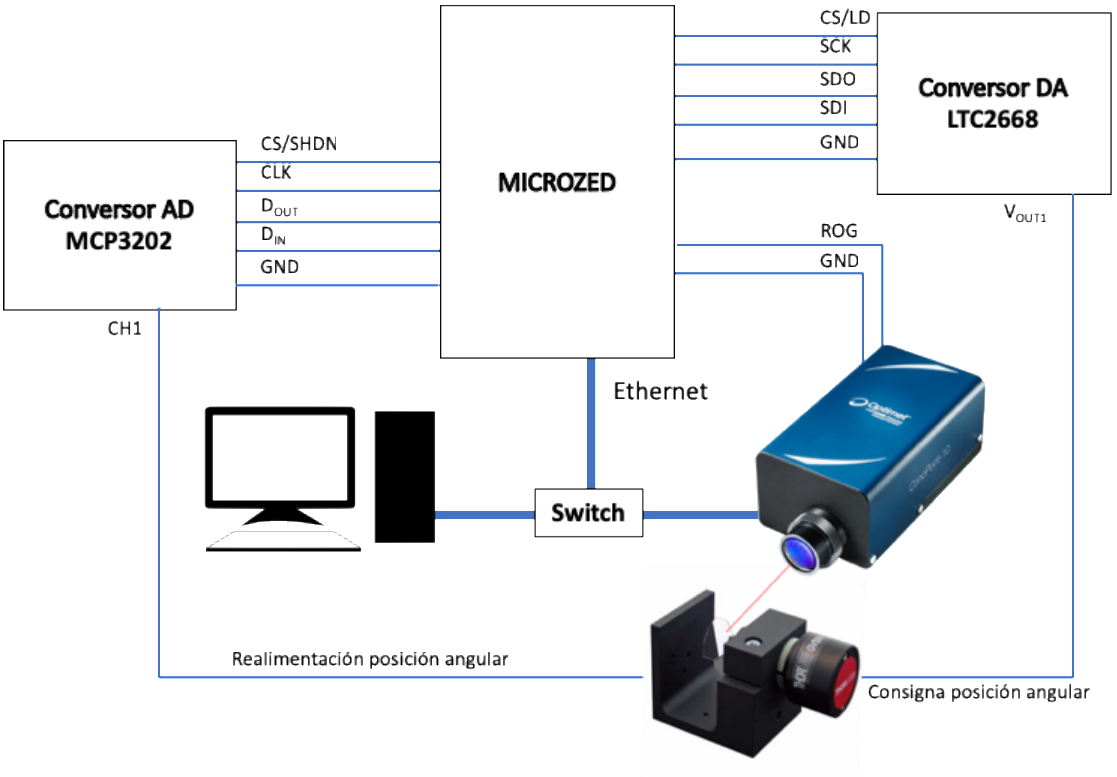

Figura 23. Esquema de conexión de los distintos componentes.

3.1.1 Sensor de holografía conoscópica Conopoint.

Se ha utilizado el sensor Conopoint 20 [1], de Optimet. Es un sensor para mediciones de distancia sin contacto basado en holografía conoscópica, como ya se ha explicado en el apartado 2.1. Presenta las siguientes características:

- Puede adaptarse a diferentes distancias y resoluciones en función de la lente que se coloque.

- El procesamiento para obtener la medida de distancia lo realiza el propio sensor.
- Puede usarse con modos de disparo interno y externo.
- Comunicación por UDP.
- Fácil integración en PC usando DLLs. Estas librerías son las que serán adaptadas para que permitir compatibilidad con la placa *microzed*.
- Medición de hasta 20000 puntos por segundo.
- Precisión submicrométrica con objetivos de baja distancia focal.
- Medición de superficies complejas, con ángulos de hasta 85º.
- Posibilidad de integración con distintos elementos ópticos, como espejos.
- Capacidad de sincronización con hasta tres señales de encóder. Lo que será ampliado a seis en el presente trabajo.

Figura 24. Senor Conopoint 20.

El sensor emite un rayo láser, enfocado mediante una lente según la distancia de trabajo requerida, que impacta sobre la superficie a medir. Parte de esta luz vuelve hacia el sensor, entrando en el conoscopio, que dividirá este haz en dos. Estos dos haces interferirán entre sí generando un patrón recogido en el sensor. Este patrón es procesado para obtener la distancia a la superficie examinada.

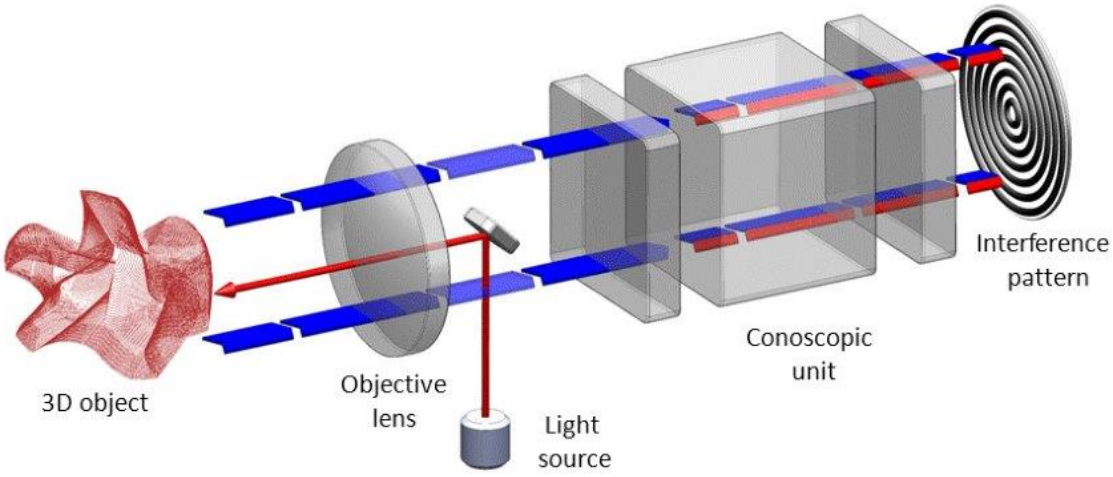

Figura 25. Camino óptico de la luz en el interior del sensor. Imagen obtenida de Optimet [25].

3.1.2 Galvo.

El galvo utilizado es el GVS311 [26], de Thorlabs. Consiste en un espejo ensamblado a un motor, junto al driver de control del motor. El galvo presenta un modo de control sencillo de utilizar, la posición se puede controlar con una señal analógica entre -10 y 10 voltios, realizando todo el control de posición la propia tarjeta de control. Se recomiendan frecuencias para el conversor DA utilizado de alrededor de 100ksps (*kilosamples per second*), y resolución de 16 bits para un funcionamiento óptimo.

Figura 26. Galvo y driver del motor.

El driver entrega en todo momento, en forma de valor analógico, la posición real del motor, que servirá de realimentación para el sistema.

3.1.3 Conversor AD.

Se utiliza el conversor AD MCP3202 [27], de microchip. Es un conversor AD de 12 bits con interfaz SPI [24]. Es un conversor AD de aproximaciones sucesivas que se puede programar para ofrecer una medida diferencial o dos entradas simples. El dispositivo es capaz de alcanzar hasta 100 ksps alimentado a 5V y 50 ksps a 2.7V.

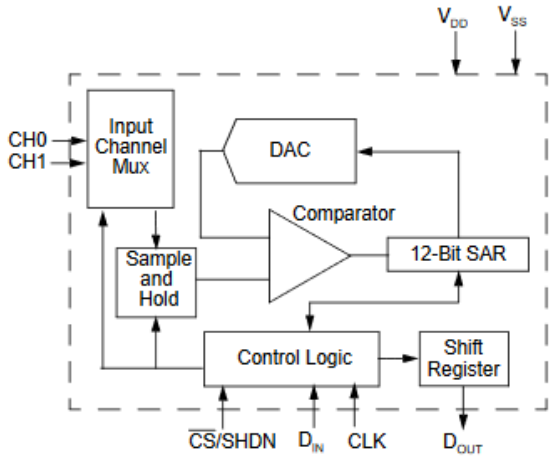

Figura 27. Diagrama de bloques del MCP3202.

3.1.4 Conversor DA.

El DAC utilizado es el LTC2668 [28]. Es un conversor digital/analógico de 16 bits, de 16 canales, con interfaz SPI entre 1.8 y 5V a frecuencias de hasta 50MHz. Puede funcionar en los rangos 0 a 5V, 0 a 10V, ±2.5V, ±5V y ±10V, seleccionables por software o hardware.

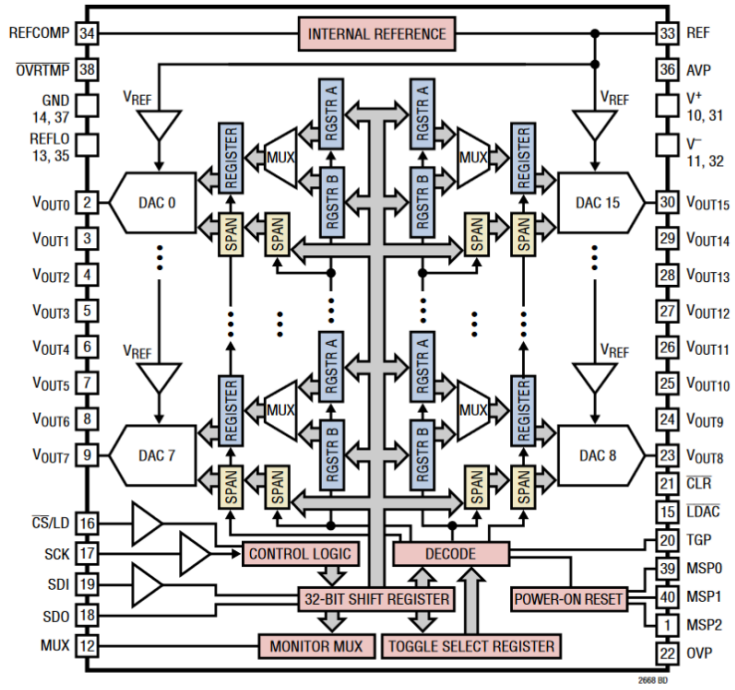

Figura 28. Diagrama de bloques del DAC LTC2668.

Los mensajes de comunicación deben construirse según la siguiente secuencia:

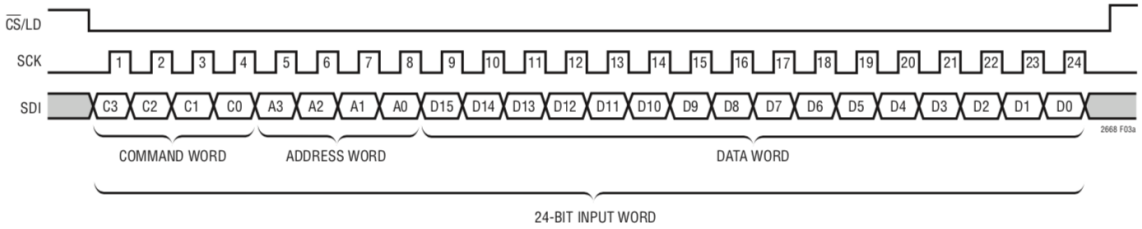

Figura 29. Secuencia de tamaño mínimo. Palabra de 24 bits.

El mensaje puede ocupar desde 24 a 32 bits. Se compone de una palabra de 12 bits, la denominada palabra de datos; y dos de cuatro bits, comando y dirección. Si se utilizan secuencias de 32 bits, los 8 restantes no se tienen en cuenta.

Existen 16 comandos de configuración posibles, según se quiera cambiar el valor de una salida, de todas, actualizar salidas de manera sincronizada, etc. Para más información ver la hoja de características [28].

En la dirección se indica a cuál de las 16 salidas hace referencia el mensaje. Y, finalmente, en los 12 bits de datos se sitúa el valor analógico deseado.

3.2 PROGRAMACIÓN MICROZED

La tarjeta Microzed albergará el programa de control del sistema. Como ya se ha explicado en el apartado 2.2, la plataforma Zynq puede dividirse en dos partes principales, la FPGA o PL (lógica programada) y el PS (procesador ARM). En el PL se programará la interacción con los conversores AD y DA y la generación de señales, mientras que el PS gestionará la comunicación con el sensor Conopoint, a través de UDP.

El funcionamiento del sistema puede describirse en el siguiente diagrama de bloques.

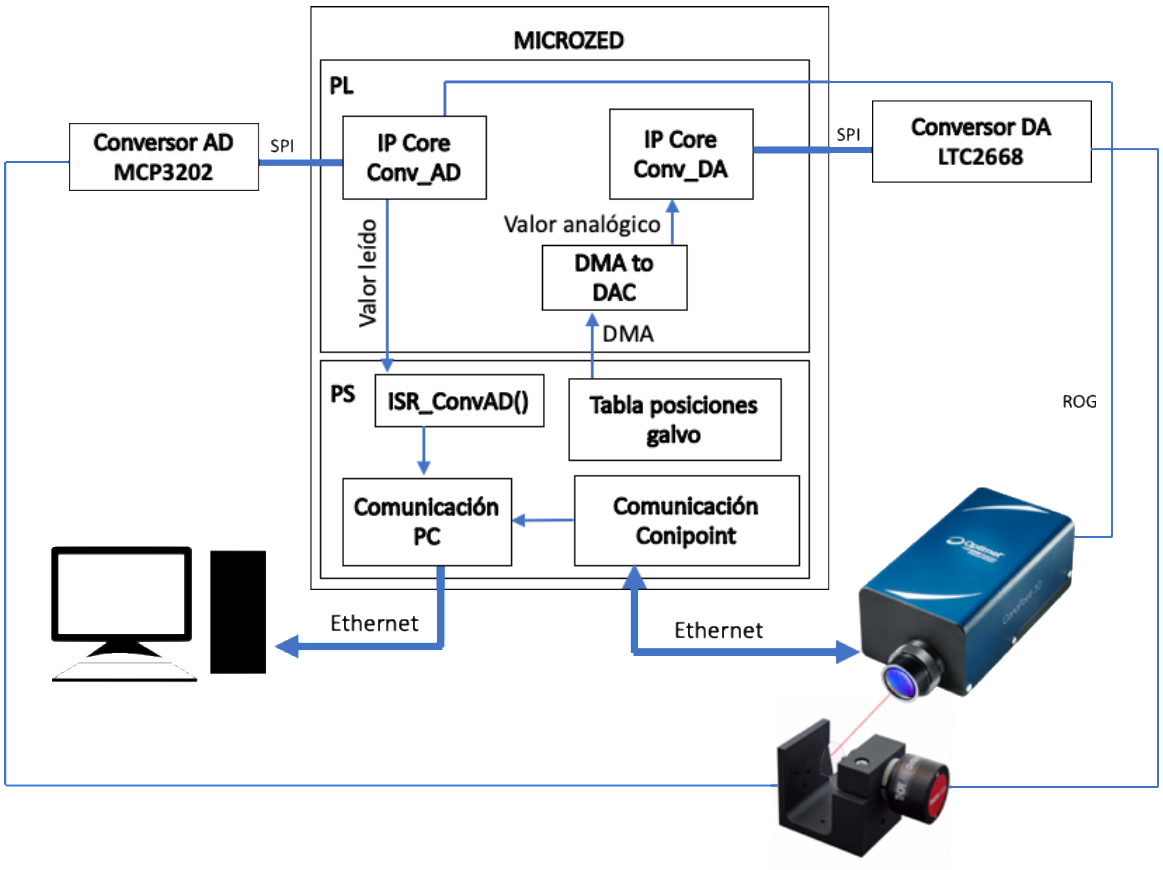

Figura 30. Diagrama de bloques del sistema.

Todo esto se programará en el entorno de desarrollo Vivado 2018 [29].

3.2.1 Programación hardware (FPGA)

Para programar el PL, se utiliza un diagrama de bloques, el que distintos IP Cores se relacionan entre sí. Muchos de estos IP cores son proporcionados por Xilinx en el entorno de desarrollo, mientras que otros son programados por el desarrollador, como veremos a continuación. En la Figura 31 puede verse el diagrama de bloques implementado.

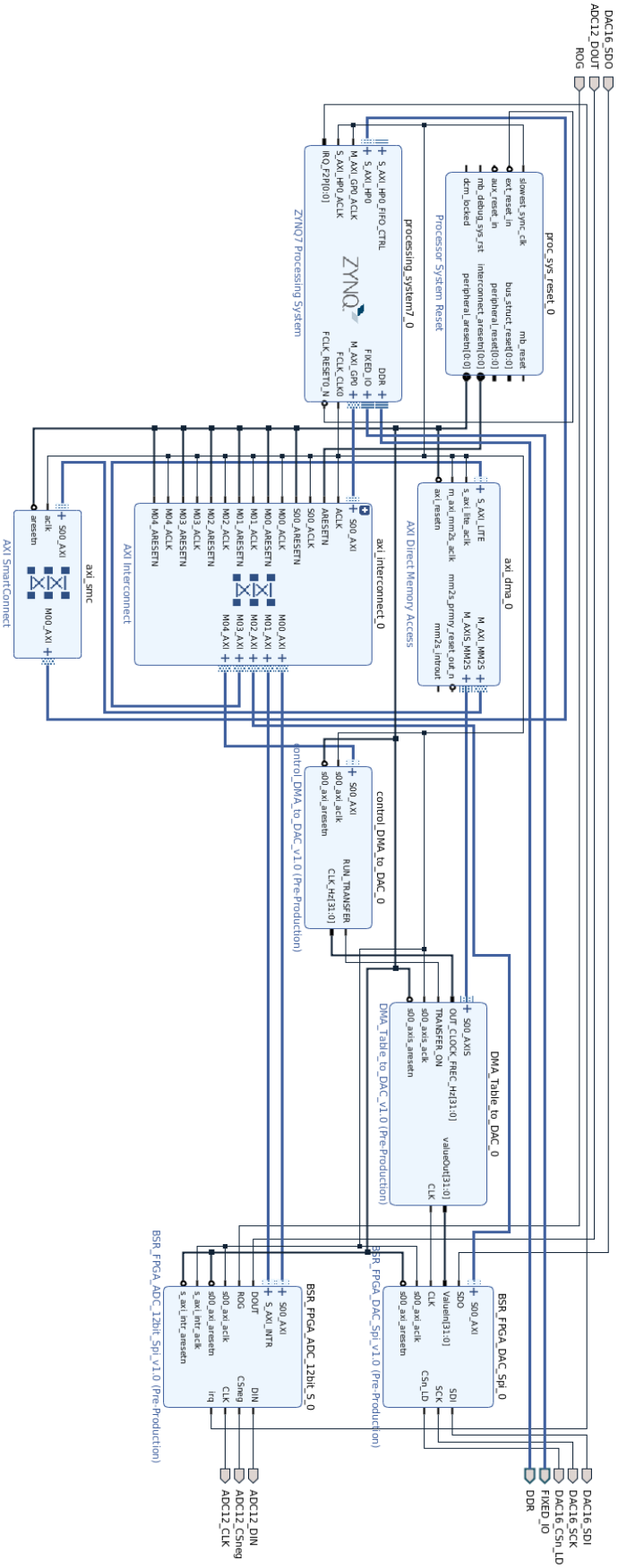

Figura 31. Diagrama de bloques del PL en Vivado.

Los primeros bloques que se observan son el *ZYNQ7 Processing System*[30] y el *Processor System Reset* [31]. Son dos bloques necesarios al trabajar con una plataforma basada en *Zynq*, en el bloque del procesador se encuentran las distintas configuraciones que definen el sistema a nivel de hardware, como la configuración de puertos de entrada/salida, relojes, puertos...

El bloque AXI Interconnect [18], permite la comunicación entre el procesador y los distintos IP Cores a través del protocolo AXI, como se ha explicado en el apartado 2.2.3.

Además de los ya mencionados, se encuentran los bloques encargados de la comunicación DMA, el conversor AD y el conversor DA. Estos ya no son bloques estándar, sino que son programados a medida.

3.2.1.1 Conversor AD.

Para el control del conversor AD, a través de comunicación SPI, se ha creado un IP Core personalizado. El bloque presenta las entradas y salidas que se pueden ver en la Figura 32.

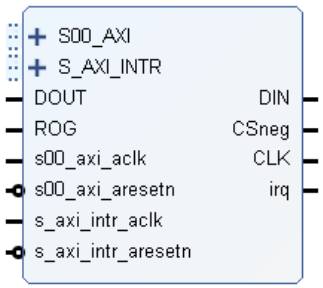

Figura 32. Bloque conversor AD.

El IP Core se ha creado con las funcionalidades relativas a comunicación AXI lite y capacidad para generar interrupciones, por lo que presenta el conjunto de entradas relativas a los buses AXI y sus correspondientes entradas de reloj y reset.

El resto de las entradas y salidas se componen de la señal de ROG, las relativas a la comunicación SPI y una destinada a lanzar una interrupción. Las dos entradas son: ROG y DOUT. La señal de ROG será la señal proveniente del sensor Conopoint que indica cuándo tiene lugar la adquisición de medidas, y se utiliza para la

sincronización de las medidas con la señal analógica medida. DOUT corresponde con el pin a través del cual el conversor AD emite los pulsos que indican el valor medido.

En lo referente a las salidas, DIN, CSneg y CLK se conectan a las respectivas entradas de conversor, mientras que IRQ se conecta al vector de interrupciones del procesador. Este pin se utilizará para generar una interrupción en el procesador cuando termine la medida.

Durante el tiempo que dura la adquisición de datos del sensor de holografía conoscópica, la señal de ROG se pone a “1”, durante este tiempo el conversor AD comenzará a tomar medidas. Una vez la señal de ROG vuelve a “0” se emitirá un pulso en IRQ, que disparará una interrupción en el procesador desde la que se podrán leer tanto el valor acumulado de las medidas como el número total de mediciones. De esta forma se tiene el valor medio de la señal analógica durante la adquisición de datos.

Para la implementación de esta funcionalidad, se programarán cuatro procesos concurrentes. El primero se encarga de la generación de la señal de reloj, una señal simétrica de 1.6 kHz. Cada pulso reloj en *s00_axi_clk*, se está usando un reloj de 100MHz, se ejecutará el siguiente proceso:

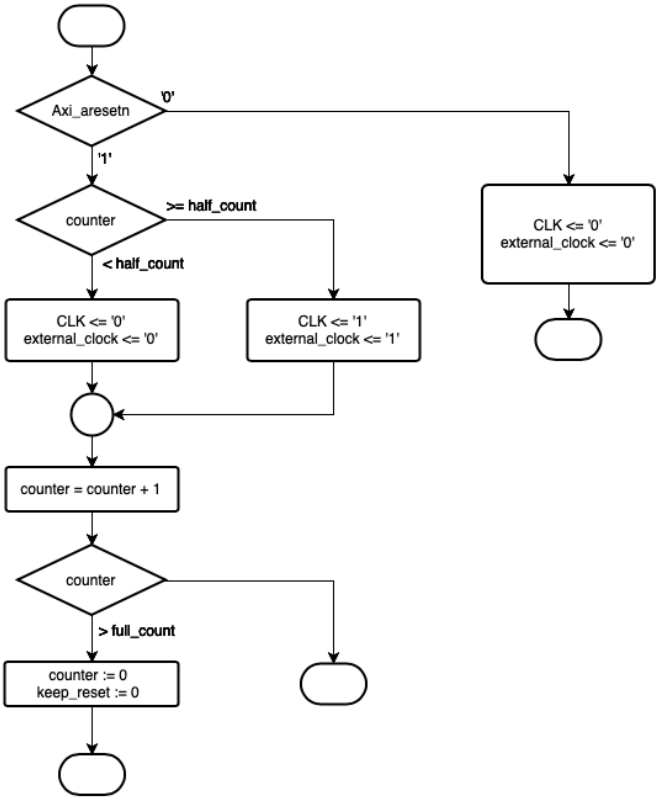

Figura 33. Diagrama de secuencia del proceso que gobierna el reloj.

Donde *full_count* es el cociente entre la frecuencia de reloj de la FPGA y la frecuencia del reloj SPI buscada. *Half_count* es la mitad del valor anterior, para generar una señal simétrica.

Dos procesos escribir en DIN y leer en DOUT. En los flancos de bajada del reloj SPI se escribe la salida DIN, de la siguiente manera:

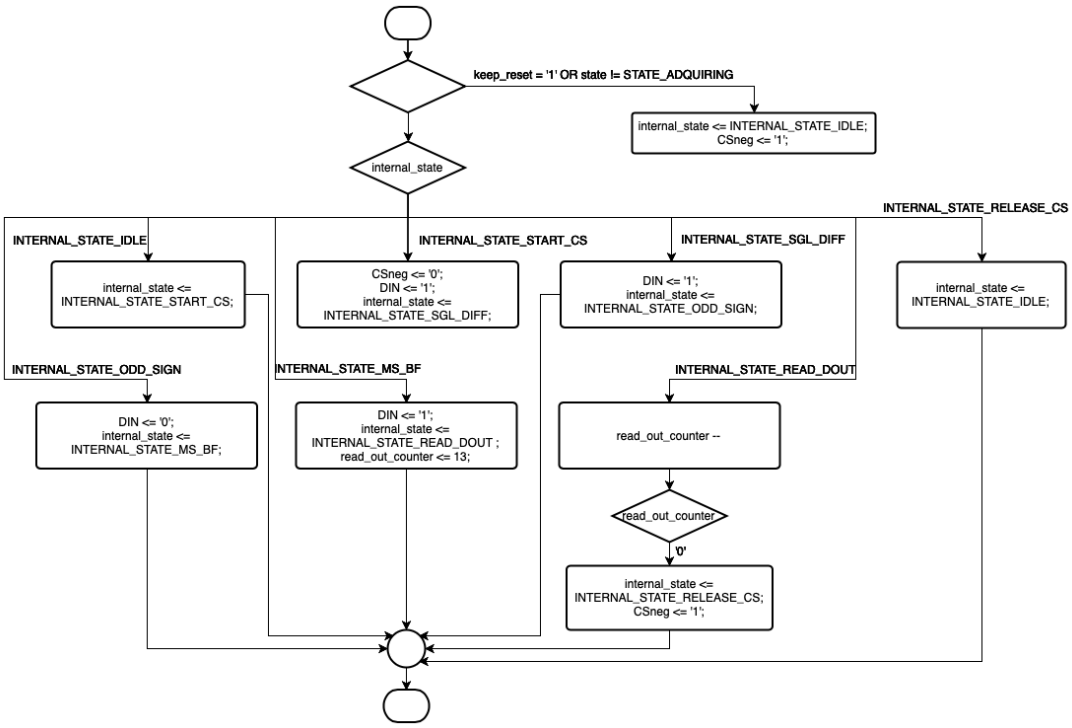

Figura 34. Diagrama de estados para lectura y escritura en DOUT y DIN.

Los flancos de bajada se utilizan para leer la entrada DOUT. Como el contador de bits y el estado interno se actualizan en el proceso anterior, este queda realmente sencillo: si se encuentra en el estado INTERNAL_STATE_READ_DOUT y el contador de bits está entre el 0 y el 11, se almacena el valor de DOUT en el bit correspondiente del registro que guarda el último valor leído.

El último proceso es el que se encarga del comportamiento a más alto nivel. Controla que el conversor realice las adquisiciones según dicte la señal de ROG, como puede verse en el siguiente diagrama.

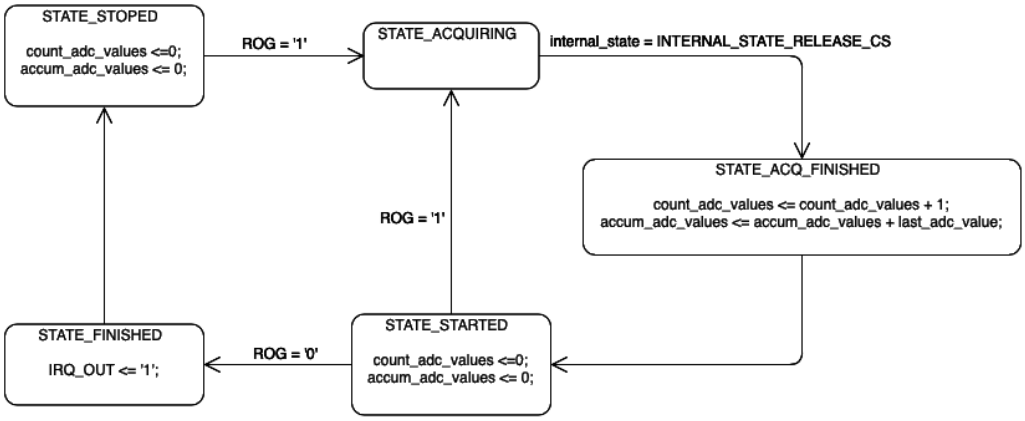

Figura 35. Diagrama de estados de alto nivel.

Una vez se genera el pulso en IRQ_OUT saltará una interrupción en el procesador. Desde esta interrupción se podrán leer los valores de los registros del valor acumulado y número de medidas para tomar el valor medio.

3.2.1.2 Conversor DA.

Al igual que el conversor AD, el DA también presenta comunicación SPI. Es IP Core contendrá la funcionalidad básica de la comunicación SPI. En la entrada VauleIN recibirá un *array* con los bits correspondientes al comando que se quiera escribir y, según los pulsos de entrada en CLK, se irán escribiendo secuencialmente. Al contrario que en el conversor AD, en este bloque no se ha implementado ningún comportamiento de más alto nivel, ya que se los valores analógicos que se pretenden escribir en el conversor serán transmitidos por DMA, como se explica más adelante. No se almacenarán los valores en registros de comunicación AXI, puesto que la señal analógica buscada ocupará una tabla de unos 500 valores diferentes. También se descarta utilizar un registro e ir actualizándolo, ya que el procesador tendría que encargarse de actualizarlo, lo que dificultaría enormemente el funcionamiento a tiempo real de manera simultánea con el resto de las tareas a las que se destinará.

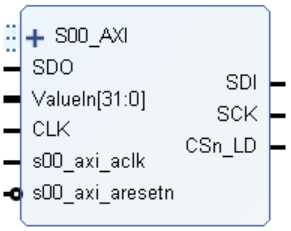

Figura 36. Bloque conversor DA.

El comportamiento del IP Core se describe mediante el siguiente diagrama. Cada pulso de un reloj interno de 25MHz, generado como en la Figura 33, se ejecuta la siguiente secuencia.

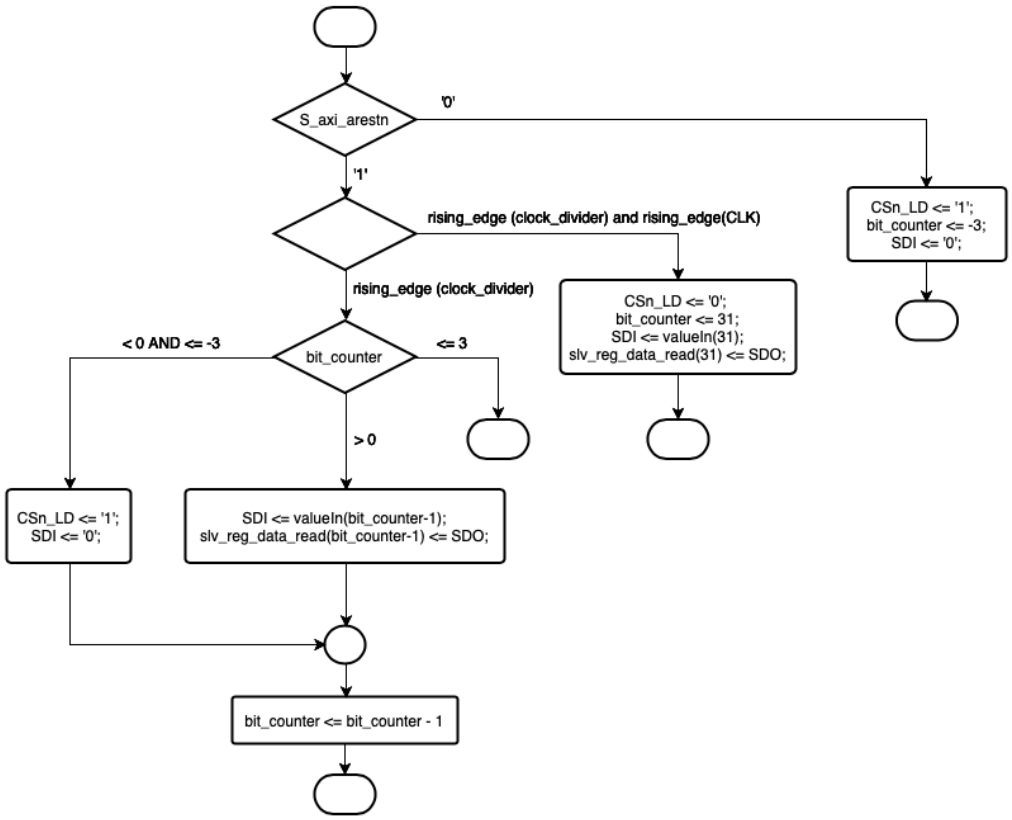

Figura 37. Diagrama de funcionamiento del IP Core del conversor DA.

El reloj externo, en la entrada CLK, indica cuando comenzar la transmisión de un nuevo dato. De este modo el reloj interno se ocupa de la frecuencia de la comunicación SPI, y el externo de la frecuencia de refresco del conversor.

3.2.1.3 Generación de señal analógica.

Para realizar los barridos con el láser se busca que el galvo gire siguiendo un movimiento sinusoidal, evitando así cambios bruscos de velocidad en los extremos del movimiento.

Para que el sistema diseñado en hardware, en la FPGA, sea versátil y sirva para generar señales periódicas sea cual sea su forma, los valores correspondientes a un período se generarán en el procesador ARM.

La señal generada tendrá 500 valores por período, por lo que no es factible realizar la comunicación entre el PS y el PL a través de AXI-4 Lite como en los IP Cores anteriores, ya que se necesitarían 500 registros. Además, la transferencia no sería muy eficiente, ya que este protocolo solo permite el intercambio de un dato por mensaje, como se ha explicado en el apartado 2.2.3. La tabla de valores se transferirá al PL utilizando el DMA, que permite la transferencia de un número ilimitado de datos en forma de AXI Stream.

Como se puede ver en la Figura 38, para configurar una transferencia DMA se necesitan dos IP Cores estándar: un *AXI SmartConnect* [19], conectado a la interfaz S_AXI_HP0 para comunicaciones de alto rendimiento; y un *AXI Direct Memory Access* [32], del que sale el *AXI Stream* con los datos que se quieren transferir.

El *stream* de datos lo recibe el *IP Core* DMA_Table_to_DAC, este ya creado a medida. Como la interfaz de comunicación es S00_Axis para poder transmitir el *stream* de datos, no es posible conectar este bloque al bus AXI4_Lite para habilitar registros de control a los que acceder desde el PS. Para solventar esto, se crea un nuevo IP Core auxiliar (Control_DMA_to_DAC), que servirá para generar las señales de control desde el procesador.

La función del bloque Control_DMA_to_DAC es muy simple, simplemente lee el valor de dos registros y lo vuelca a la salida correspondiente. El primer registro es un vector de 32 bits con la frecuencia, en hercios, deseada para la señal. Y el segundo registro es una señal binaria que servirá para parar o arrancar la transferencia de nuevos valores analógicos al conversor DA.

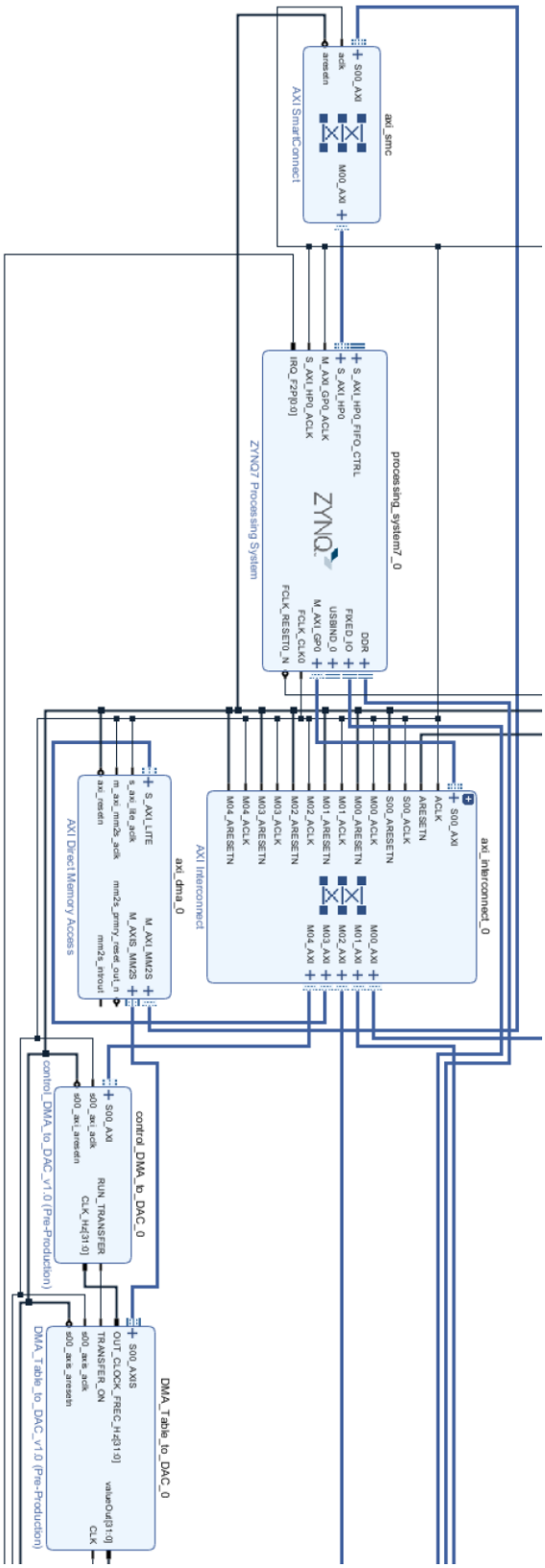

Figura 38. Bloques desde para la transferencia por DMA de una tabla de valores hasta el control de la frecuencia de la señal generada.

El bloque DMA_Table_to_DAC tiene tres funciones principales, leer el AXI_Stream procedente de la comunicación DMA para almacenarlo en un array, volcar secuencialmente los valores del array para generar la señal de la frecuencia deseada y generar la señal de reloj que indica el inicio de la escritura de un nuevo valor en el conversor AD.

La generación de la señal de reloj se hace como en los casos anteriores. La única diferencia es que en este caso el valor de frecuencia de salida depende de una señal de entrada, que indica el valor de la frecuencia de la señal de salida. Teniendo en cuenta que el reloj generado indica el cambio de valor analógico no hay que olvidar que un período de la señal tendrá 500 valores. Con esto, la secuencia será la misma que en la Figura 33.

Para volcar los datos del *stream* en un array se debe atender a la señal S_AXIS_TVALID, cuando esta está a uno significa que se ha transferido un valor. A partir del flanco de subida en S_AXIS_TVALID se lee el valor de S_AXIS_TDATA en cada ciclo de reloj hasta completar los 500 valores. La secuencia es la siguiente:

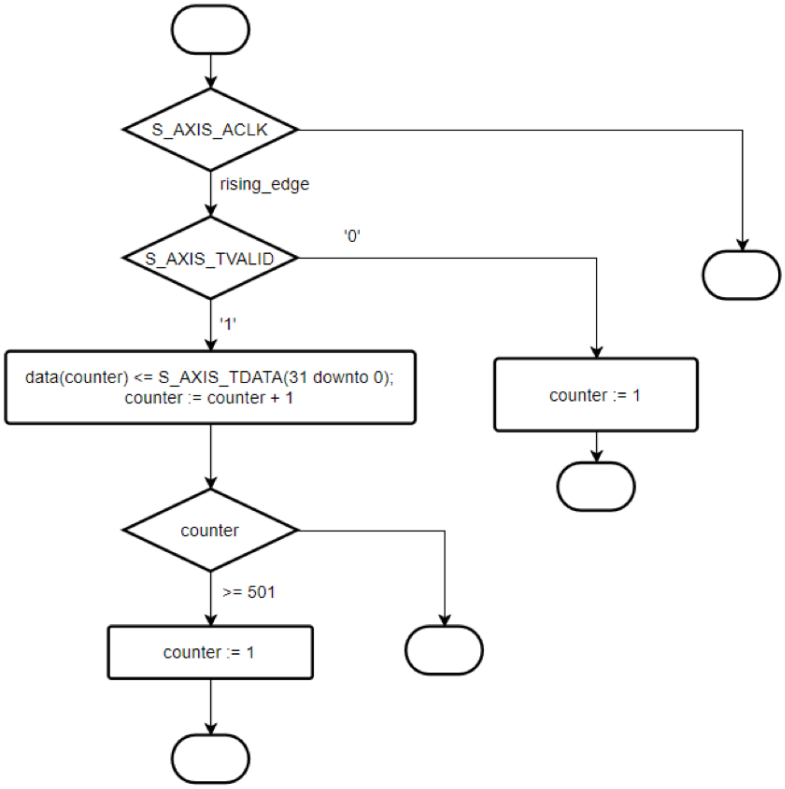

Figura 39. Almacenamiento de valores provenientes de la transferencia DMA en una tabla.

El último proceso es el que coloca en la salida secuencialmente el valor correspondiente de la tabla, para que lo lea el bloque del conversor AD. Este proceso se ejecuta en cada flanco de subida del reloj generado en el primer proceso, a una frecuencia 500 veces superior a la de la señal buscada.

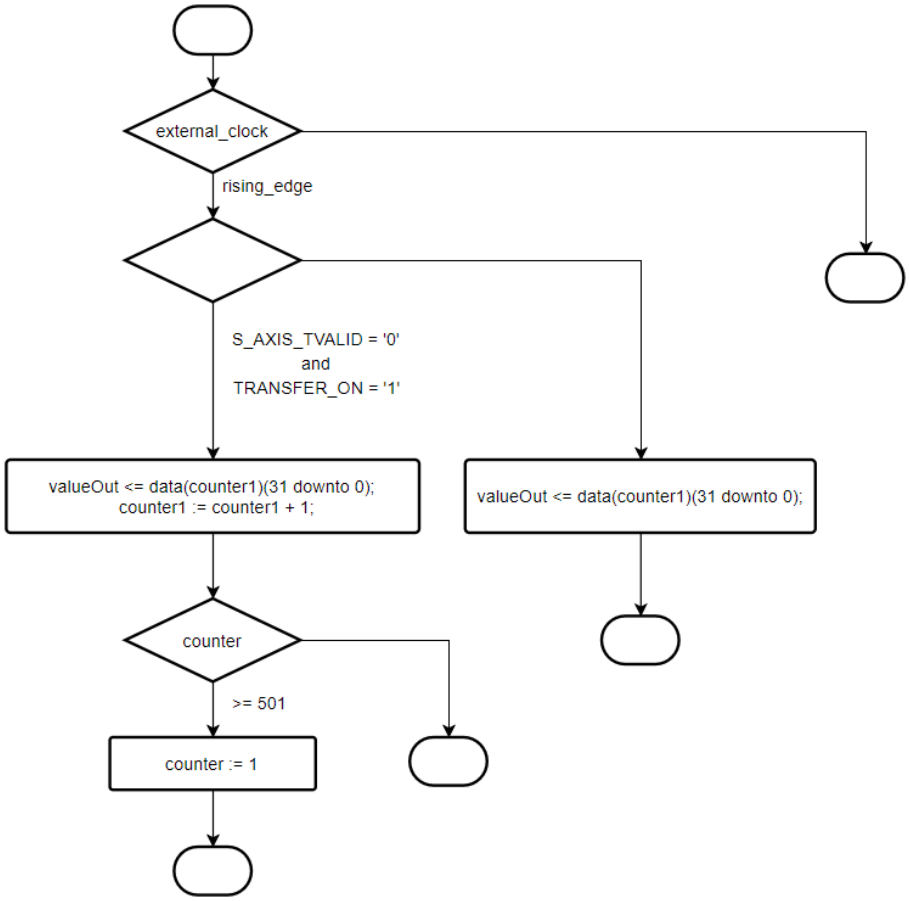

Figura 40. Secuencia de valores de salida para la generación de la señal.

3.3 PROGRAMACIÓN SOFTWARE (ARM)

Para el desarrollo del programa que se ejecutará en el PS, se ha tomado la decisión de trabajar en modo *standalone*, es decir, sin sistema operativo. Este modo proporciona una ligera capa de software que permite interactuar con el hardware del dispositivo de una manera relativamente sencilla. La principal ventaja de este modo de funcionamiento, frente al uso de un sistema Linux, es que permite tener un control total de todo lo que se ejecuta, sin todo el procesamiento extra que llevaría la ejecución de un sistema operativo.

3.3.1.1 Standalone

En ausencia de sistema operativo, *standalone* es una capa de software de bajo nivel. Proporciona acceso a características básicas del procesador, como cachés, interrupciones, excepciones o a las interfaces de entrada/salida; de manera que no haya que lidiar con ello a nivel de registros.

La gestión de interrupciones es especialmente relevante, puesto que el programa desarrollado dependerá fuertemente de ellas, tanto para la obtención de lecturas del conversor AD, como para la comunicación con el sensor de holografía conoscópica.

La gestión de interrupciones se encapsula en la siguiente clase:

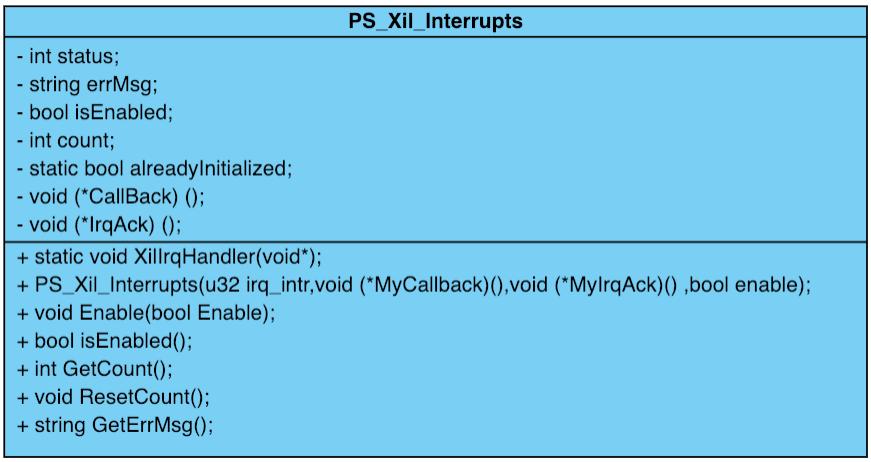

Figura 41. Clase para gestionar interrupciones.

Esta clase permitirá llevar un registro del número de veces que salta una interrupción. Ante una interrupción se ejecutará la función `CallBack`, indicada por el puntero correspondiente y, posteriormente, la indicada por el puntero `IrqAck`, que bajará el *flag* correspondiente a la interrupción.

A bajo nivel, la gestión de interrupciones se encuentra implementada en la librería `xscugic`. Para habilitar una interrupción es necesario llevar a cabo cuatro pasos:

- Crear la estructura `XScuGic_Config` que define el controlador del GIC (*Generic Interrupt Controller*).

- Inicializar el GIC.
- Conectar la interrupción hardware con la lógica de atención a la interrupción del procesador.
- Habilitar la máscara correspondiente a la interrupción.

Un ejemplo detallado puede verse en [33].

3.3.1.1 Conversor AD

Como se ha explicado en apartados anteriores, la comunicación con el conversor AD está implementada en un IP Core, para que la FPGA se encargue del grueso del procesamiento.

Como cualquier otro IP Core que se comunica por AXI4 con el PS, se tienen unos drivers, a nivel de registros, para interactuar desde el PS. En este caso se implementan cinco funciones para habilitar y deshabilitar las interrupciones; habilitar y deshabilitar la conversión; y obtener el valor analógico, a partir de la lectura de los registros de número de medidas y valor acumulado.

Sobre estos drivers, que operan a bajo nivel, se diseña una clase que encapsula tanto su funcionalidad como la gestión y configuración de la interrupción asociada.

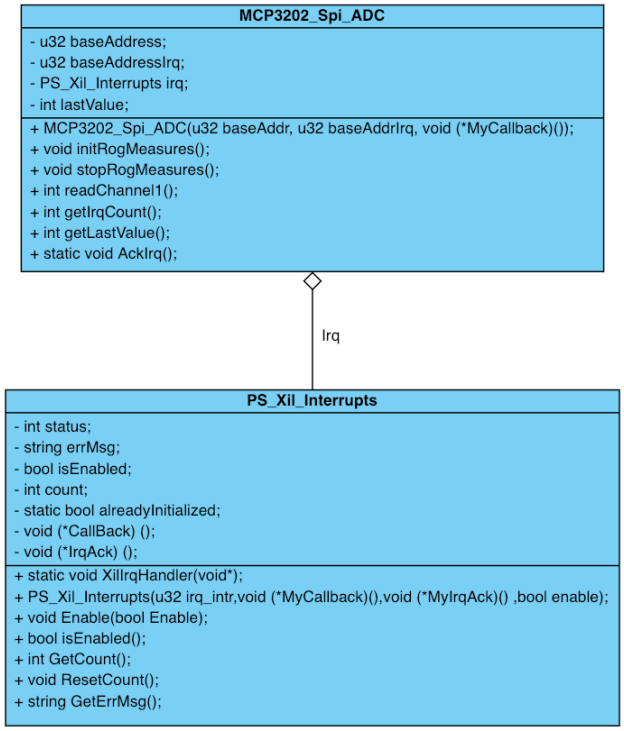

Figura 42. Clase que encapsula la funcionalidad del conversor AD.

La única información que necesitará el objeto para gestionar la adquisición de datos analógicos serán las direcciones de memoria de los registros asociados a la comunicación AXI y la función `CallBack` a ejecutar cuando indique la señal de ROG.

3.3.1.2 Galvo

En el caso del control del galvo, aparecen involucrados varios IP Cores distintos, cada uno con sus drivers. Se tiene el encargado de la comunicación con el conversor AD, con funciones de habilitación e inhabilitación o selección de canales del conversor; el responsable de la comunicación DMA; y el de control del IP Core que genera lee el *stream* transferido por DMA y guarda una tabla de valores en la FPGA.

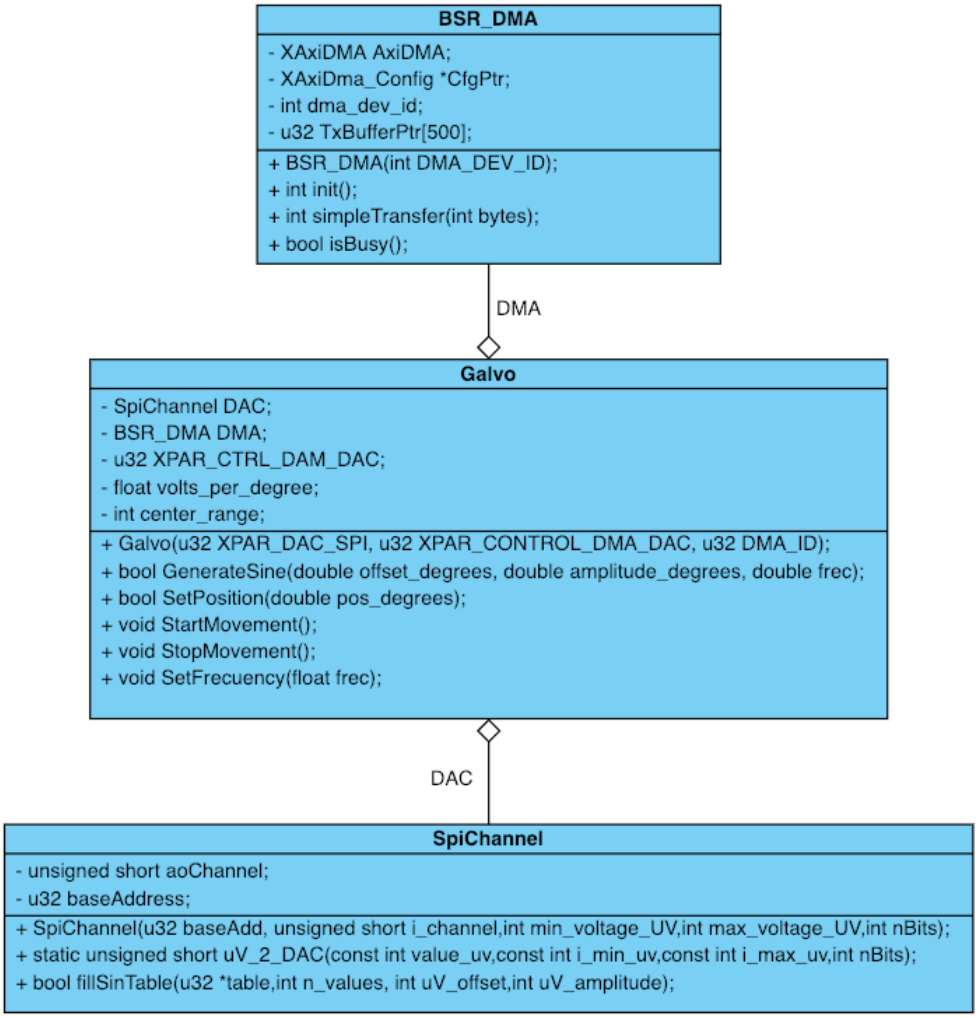

Figura 43. Diagrama de clases que ilustra el control del galvo.

La clase `galvo` encapsula el control del galvo a más alto nivel, de manera que para su uso no haya que lidiar con operaciones como la configuración de la transferencia *Axi DMA*, implementada en la clase `BSR_DMA`; o la construcción de los mensajes a enviar al DAC, implementado en la clase `SpiChannel`.

Aquí se implementa la generación de la señal analógica, en este caso con la función `GenerateSine()`, que rellena un vector con un período de la señal y lo envía como un *AXI Stream* a la FPGA, para que lo recoja el *IP Core* correspondiente. Esta implementación otorga gran flexibilidad, puesto que en caso de que fuese necesaria una señal periódica de cualquier otra forma, bastaría con rellenar el vector con los valores que fuera menester.

3.3.1.3 Portar Conoprobe a baremetal sin QT

El protocolo de comunicación con sensores Conopoint anterior estaba programado en una librería compatible con sistemas Windows y Linux. En la tarea de portar la librería al sistema *baremetal* que corre en el procesador aparecen dos inconvenientes principales: la librería está basada en Qt, no disponible para esta plataforma; y la comunicación UDP basada en sockets, una herramienta propia de un sistema operativo.

Propias de Qt se utilizan distintas estructuras de datos, como pueden ser *Qvector* o *QString*. Aunque laborioso, este es un problema menor, ya que al final todas estas clases tienen su equivalente en C++ estándar por el que se pueden sustituir.

Sustituir el uso de sockets ya es más complejo. En el siguiente diagrama de clases puede verse, de manera simplificada, la estructura de la librería de comunicación.

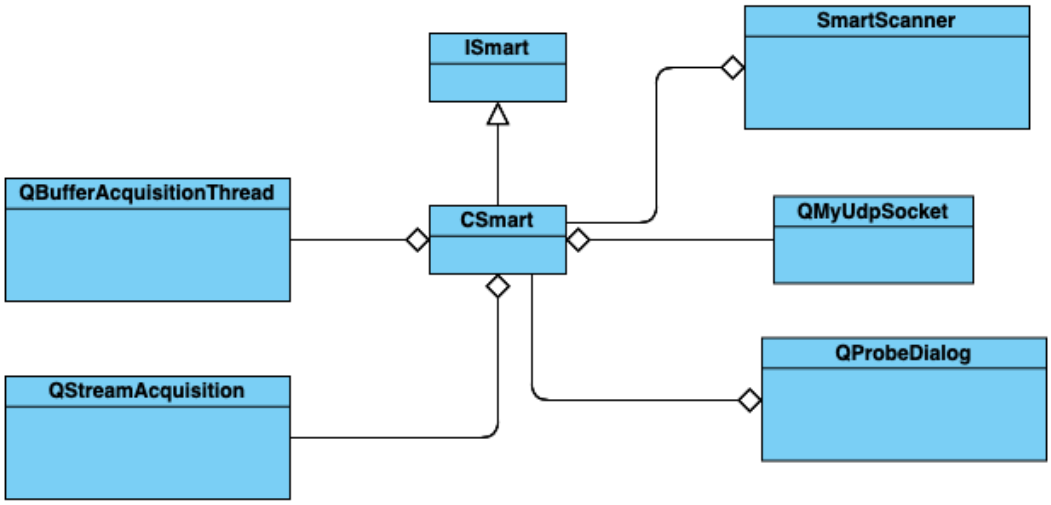

Figura 44. Diagrama de clases simplificado de la librería.

La clase `ISmart` es la clase principal, en ella se definen como métodos estáticos los referentes a buscar sensores en la red. Y como funciones virtuales puras todas las funcionalidades disponibles en el sensor. De esta manera esta clase sirve como interfaz de la que derivar la clase `CSmart`, en la que se implementan realmente la comunicación con el sensor y todas las funciones de las que dispone el sensor Conopoint.

`CSmart` incluye instancias de las clases `QBufferAcquisitionThread` y `QStreamAcquisition`. Estas clases implementan la captura de una secuencia de mediciones. La clase `QBufferAcquisitionThread` crea un nuevo hilo en el que va recibiendo las medidas y las almacena en un *buffer*. Puesto que no se está utilizando un sistema operativo que permita distintos flujos de ejecución simultáneos, esta funcionalidad no se implementará. Se portará solamente la clase `QStreamAcquisition`, equivalente a la anterior, pero la lectura de las medidas se realiza en el hilo principal.

`QProbeDialog` implementa una interfaz gráfica para realizar configuraciones y medidas desde la pantalla. Tampoco será necesaria en el sistema implementado, ya que no se ejecutará en un sistema de escritorio con pantalla, teclado y ratón.

`SmartScanner` permite realizar medidas con varios sensores simultáneamente. Como el sistema de medida implementado solo puede comunicarse con un sensor tampoco será necesario.

Por último, queda la clase `QUdpSocket`, en la que se implementa la comunicación a través del protocolo UDP utilizando QSockets, la manera en la que Qt implementa los sockets, ya sean de Linux o Windows. Aquí es donde estará la mayor enjundia, puesto que hay que implementar un sistema equivalente a los sockets con lwIP [34], una librería para implementar distintos tipos de comunicación ethernet pensada para microcontroladores. En el apartado 3.3.1.4 se explica detalladamente la implementación.

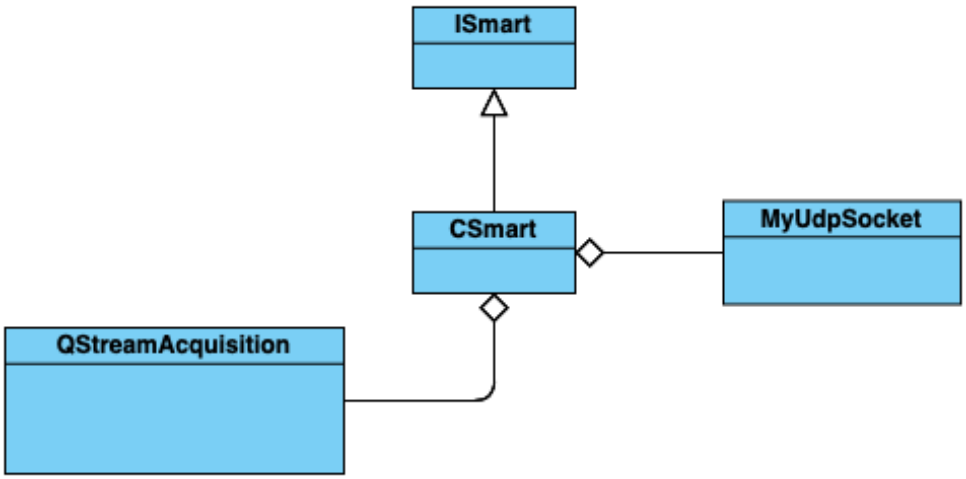

Figura 45. Diagrama de clases implementado.

En la Figura 45 se ve el diagrama de clases final. Solo se muestra lo referente a la comunicación con el sensor. `CSmart` también incluye la posibilidad de mover un espejo, tal como se ha explicado en el apartado 3.3.1.2.

Por último, es necesario modificar el formato de la trama en el que llegan las medidas, ya que el sensor Conopoint20 permite la sincronización de hasta tres señales de encoder, frente a las seis que debe permitir el nuevo sistema.

El sensor devuelve las medidas en una estructura de 128 bytes con el siguiente formato.

Tabla 1. Estructura de datos provenientes del sensor.

UINT16	Tag
UINT16	Unused
float	Distance
float	Frecuency

UINT32	Reserved
UINT16	Total
UINT16	Snr
UINT16	SerialNumber
UINT16	Bitset
INT32	EncoderX
INT32	EncoderY
INT32	EncoderZ
BYTE	Reserved2[86]

Afortunadamente, la estructura del mensaje presenta 86 bytes libres al final, reservados para que el software pueda ser compatible con el resto de los sensores de la marca. Incrustar los nuevos valores de encoder en la trama será tan sencillo como rellenar los 12 primeros bytes reservados antes de enviar la medida al ordenador correspondiente. La trama final tendrá la siguiente forma:

Tabla 2. Estructura de datos que se enviará al PC.

UINT16	Tag
UINT16	Unused
float	Distance
float	Frecuency
UINT32	Reserved
UINT16	Total
UINT16	Snr
UINT16	SerialNumber
UINT16	Bitset
INT32	Encoder1
INT32	Encoder2
INT32	Encoder3
INT32	Encoder4
INT32	Encoder5
INT32	Encoder6
BYTE	Reserved2[74]

3.3.1.4 Comunicación UDP

En este punto se puede encontrar la principal desventaja de no usar un sistema Linux, no poder utilizar los mecanismos de comunicación que nos proporciona. Como no se tiene acceso a los sockets de Linux, es necesario implementar una alternativa.

Para implementar la comunicación UDP se ha usado la librería *LightWeight IP* (lwIP) [34]. Esta librería es una implementación ligera y *open-source* de la pila TCP/IP, pensada para su inclusión en sistemas embebidos. lwIP fue desarrollada originalmente por Adam Dunkels, en el Instituto Sueco de Ciencias de la Computación (SICS). Actualmente se mantiene su desarrollo gracias las aportaciones de programadores de todo el mundo.

La implementación realizada utiliza la denominada *Raw API*, una API dirigida por eventos diseñada para ser usada sin sistema operativo. Es la única API disponible sin un sistema operativo. Las aplicaciones que utilizan este sistema se implementan como un conjunto de *callbacks* que son invocados por el núcleo lwIP cuando ocurren eventos como la entrada de datos, salida de datos, pérdida de conexión...

Utilizando esta librería se reescribirá la clase MyUdpSocket, mostrada en la Figura 45, de manera que sea ejecutable sin sistema operativo. Esta clase será sustituida por la clase lwipUdp, que emulará un sistema basado en sockets utilizando la librería lwIP, con la siguiente forma:

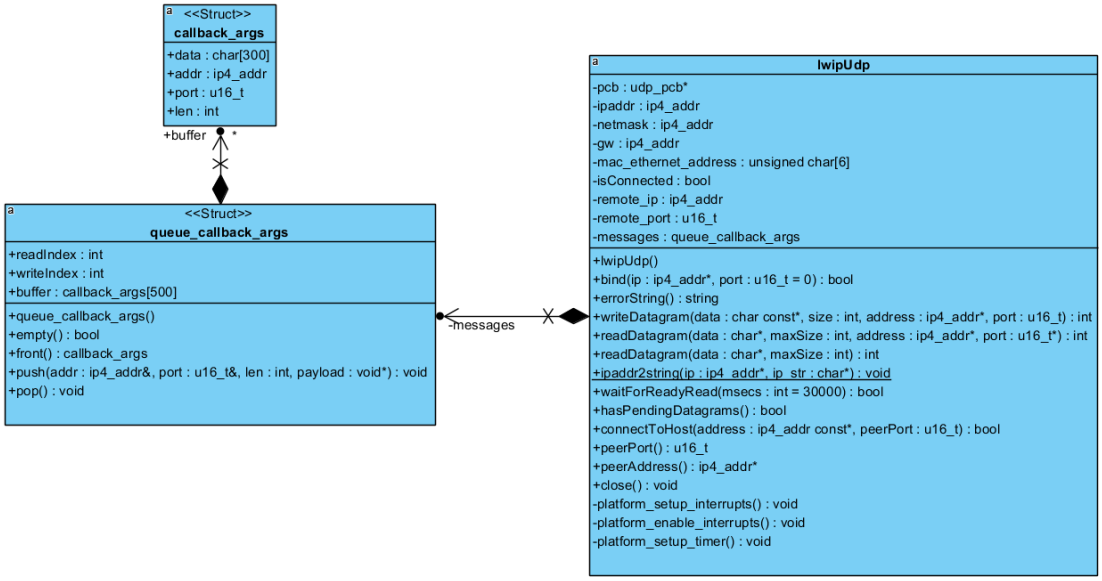

Figura 46. Diagrama de clases que emulan el funcionamiento de los sockets utilizando la Raw Api de lwIP.

La principal diferencia entre los sockets y la RawApi de lwIP es que los primeros se basan en un *file descriptor* [35], en el que el sistema operativo va recogiendo los mensajes recibidos como si fuera una cola, mientras que lwIP utiliza un sistema de *callbacks*, en el que al llegar un nuevo mensaje salta una interrupción.

Una vez habilitadas las interrupciones, se almacenarán los mensajes recibidos en una cola, de manera que se pueda replicar el comportamiento anterior. El flujo de trabajo para establecer una comunicación es el siguiente:

Antes de nada, es necesario definir una interfaz de red, representada por la estructura `netif`, que incluye la información básica para establecer la comunicación. En este caso la dirección ip del dispositivo, la máscara de red y la dirección MAC del dispositivo.

Tras inicializar la parte hardware, se crea una nueva *PCB*, una estructura de datos que alberga los distintos parámetros de la comunicación. La estructura tiene los siguientes campos.

Tabla 3. Campos de la estructura `udp_pcb`.

Next	Puntero al siguiente elemento de la cadena de PCBs, si lo hubiera.
Local_IP	Dirección IP local.

Dest_IP	Dirección IP de destino.
Local_port	Puerto de recepción de mensajes.
Dest_port	Puerto de destino en el envío de mensajes.
Flags	Flags para configuración de chequeos.
Checksum	Suma de verificación para garantizar la integridad de los datos.
Recv	Puntero a la función que se ejecutará al recibir un paquete.
Recv_arg	Puntero void pasado como argumento a la función de interrupción.

El PCB se mantiene inactivo hasta que se haga un *bind* a una dirección local. Esto es, asociar al PCB una dirección local, que puede ser IP_ADDR_ANY para indicar que se escuche cualquier dirección local, y un puerto por el que recibirá los mensajes.

Tras hacer el *bind*, se asocia una función callback a la llegada de un nuevo paquete con la función `udp_receive()`. Para una información más detallada de las funciones Raw/UDP, ver [36].

Para que los paquetes se reciban con normalidad queda un último detalle, específico de la adaptación que Xilinx hace de la librería. Es necesario llamar de manera periódica a la función `xemacif_input()`, para mover lo paquetes recibidos desde los servicios de atención a la interrupción gestionados por la capa *standalone* al *lwIP stack*. Esto se realizará en una interrupción asociada a un *timer*.

Con el fin de implementar las funciones que se ven en la Figura 46, en la clase lwipUdp, para mantener la compatibilidad con el resto de la librería, se implementa una cola, `queue_callback_args`, en la que almacenar los mensajes cuando se de una interrupción por la llegada de uno nuevo. Para realizar una implementación eficiente de la cola, se ha realizado con un *array* de tamaño predefinido y dos punteros, uno que apunta a la dirección de lectura y otro de escritura. El funcionamiento es muy sencillo, se ilustra en la siguiente figura.

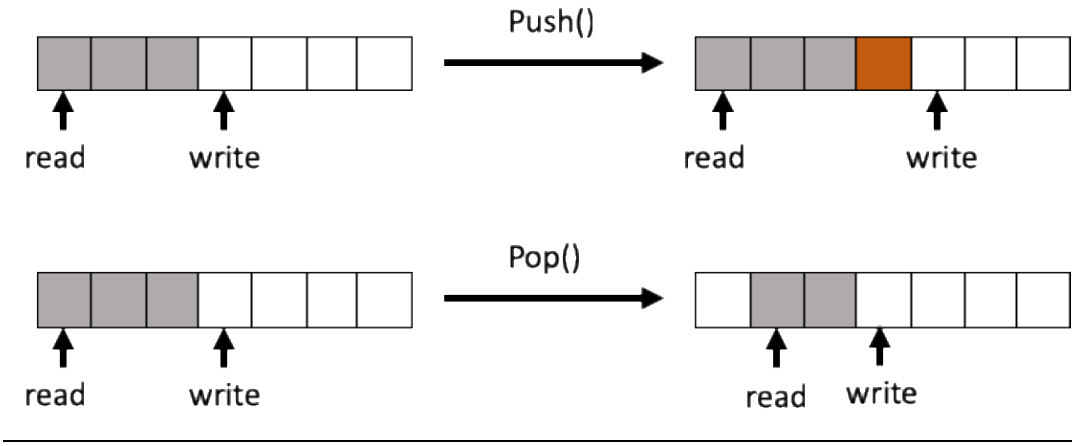

Figura 47. Implementación de una cola utilizando un *array*. En gris los espacios con datos útiles.

Cuando se añade un nuevo elemento a la cola con la función `Push()` basta con mover el puntero *write*, de manera que apunte a la primera dirección vacía, en caso de llegar al final del *array* se vuelve a la primera dirección. Para eliminar un elemento de la cola, con la función `Pop()`, no hace falta más que desplazar un valor el puntero *read*, que indica la posición del primer elemento.

Una vez la cola de mensajes funciona correctamente, reimplementar las funciones de lectura de mensajes es trivial, ya que solo habrá que mirar si hay mensajes pendientes y extraerlos.

Respecto al envío de mensajes, la cosa es más sencilla, ya que la API de lwIP es similar a los sockets. Basta con utilizar la función `udp_sendto()` [36], indicando el mensaje, la dirección y el puerto de destino.

En esta librería, los paquetes son constituidos a partir de una estructura de datos denominada `pbuf` (*packet buffer*). Estas estructuras permiten gestión dinámica de memoria y, para optimizar la velocidad en la reserva de memoria para paquetes entrantes, la librería utiliza las denominadas `pbuf_pool`. Esto es, bloques de memoria asignados previamente que se dividen en bloques de tamaño definido, donde alojar los mensajes entrantes sin el tiempo de ejecución añadido que implicaría un `malloc`.

La estructura de datos presenta los siguientes campos:

Tabla 4. Estructura de datos `pbuf`.

Next	Puntero al siguiente elemento de la cadena de pbufs, si lo hubiera.
payload	Puntero a los datos del buffer.
tot_len	Tamaño del buffer actual y todos los buffers siguientes de la cadena.
len	Tamaño del buffer actual.
type	Tipo de paquete.
flags	Flags para configuraciones.
ref	Cuenta del número de punteros que apuntan a este buffer.

Cabe destacar que, aunque los `pbuf` estén preparados para formar cadenas, de forma que un mensaje pueda estar compuesto por varios `pbufs`, dicha funcionalidad no está implementada aún en la librería. Por tanto, todos los campos que hacen referencia a esto carecen de utilidad real.

Una vez implementadas todas las partes, el comportamiento del programa será el siguiente:

- En el arranque, el procesador realizará las inicializaciones necesarias para comunicarse con el PL y habilitará la comunicación ethernet.
- Se buscarán los sensores conectados a la misma red local.
- Se creará el objeto iSmart correspondiente a la comunicación con el sensor con el número de serie preestablecido por software. De esta manera se puede escoger a priori el sensor con el que comunicarse, facilitando la convivencia de más de un sistema de adquisición en la misma red local.
- Se entra en un bucle de ejecución continua a la espera de recibir una trama UDP que indique cuándo comenzar una adquisición. La trama incluye la frecuencia de adquisición, número de medidas, frecuencia del galvo, amplitud del movimiento del galvo y offset del galvo.

4. EXPERIMENTOS Y RESULTADOS

Para poner a prueba el sistema de medida se realiza el montaje de la Figura 48. En la fotografía puede verse sobre la mesa una línea láser. Al vibrar el galvo a gran velocidad, la cámara no es capaz de tomar en una imagen la posición instantánea del punto. Con esto se puede ilustrar el rango de medida ajustado en esta situación, unos 2º de amplitud en el movimiento del galvo.

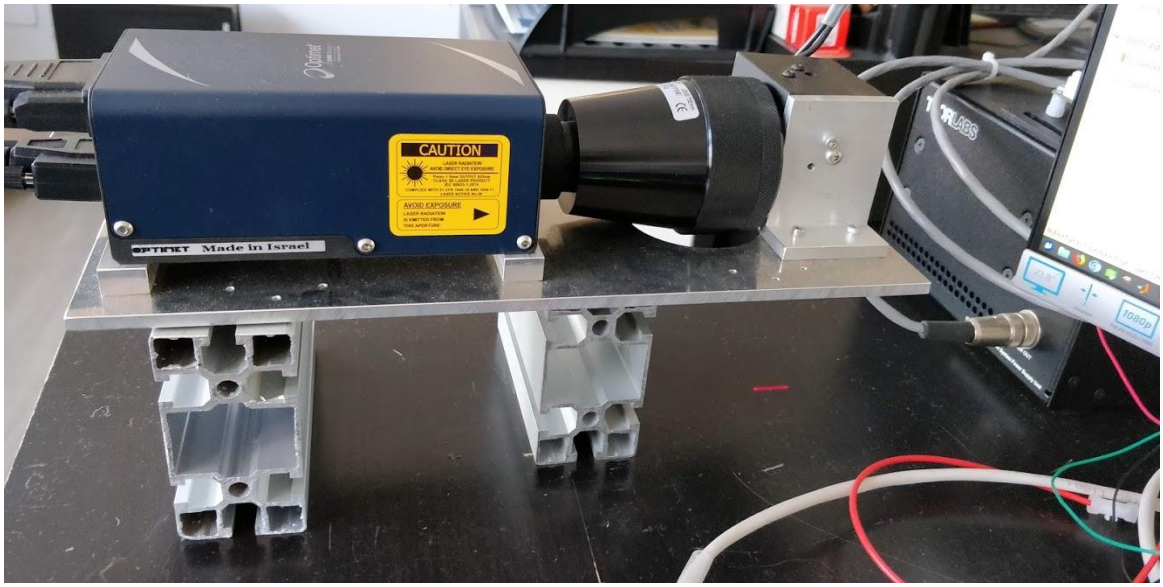

Figura 48. Montaje de un galvo redirigiendo el haz láser del sensor Conopoint.

Este montaje permitirá medir un perfil completo de una pieza sin necesidad de mover el sensor de holografía conoscópica.

Se utiliza una lente de distancia focal 75mm, que permite un rango de trabajo de 18 mm, centrado a 70 mm del sensor, con una resolución de 10µm.

Para la recepción de los datos en un ordenador se ha modificado ligeramente, la librería ConoprobeUDP, explicada en el apartado 3.3.1.3, de manera que para comenzar una adquisición se envíe a la tarjeta MicroZed una trama por UDP con la configuración deseada, y no el mensaje estándar al sensor Conopoint.

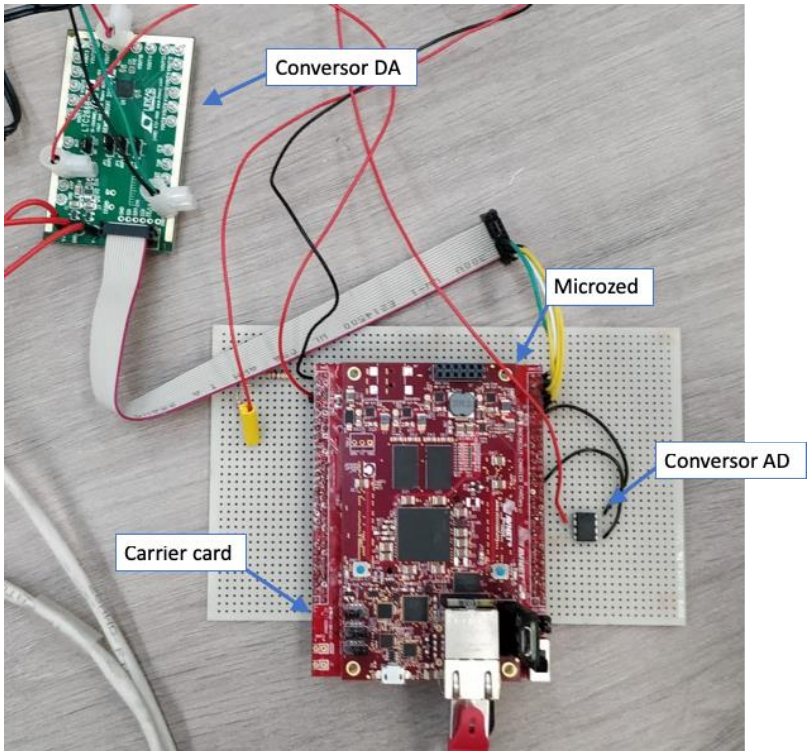

Figura 49. Montaje del prototipo electrónico en una protoboard.

En la medida de una línea de una pieza de chapa doblada por embutición se obtienen los siguientes datos. Los resultados mostrados son para perfiles tomados a una frecuencia de 2000 puntos por segundo.

En primer lugar, la medida de distancia del punto medido al sensor de la cámara, en milímetros. Aquí se puede apreciar la forma del perfil, a priori presenta un poco de ruido, pero con un poco de procesamiento servirá para ver defectos.

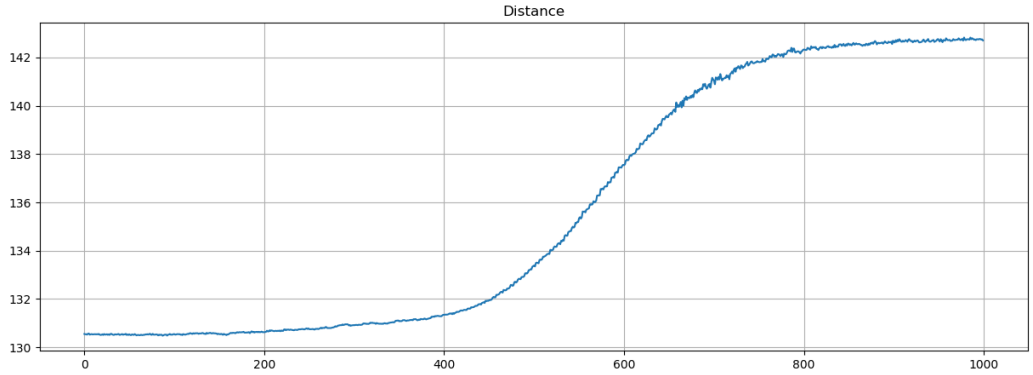

Figura 50. Medida de distancia al sensor.

Con un filtro de Savitzky–Golay [37] de segundo orden y unas pocas muestras ya se obtienen gráficos más limpios.

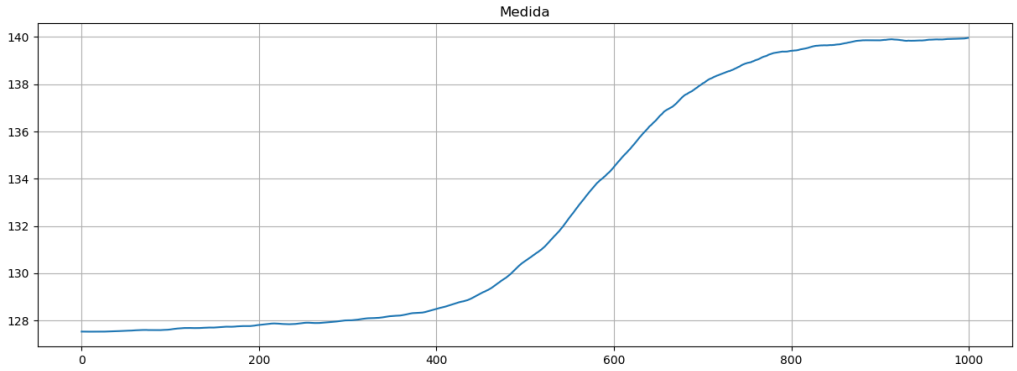

Figura 51. Medida de distancia al sensor filtrada con un filtro de Savitzky-Golay de segundo orden y 25 muestras.

La realimentación de posición del galvo, medida por el conversor AD, con un pequeño filtrado, da los siguientes resultados.

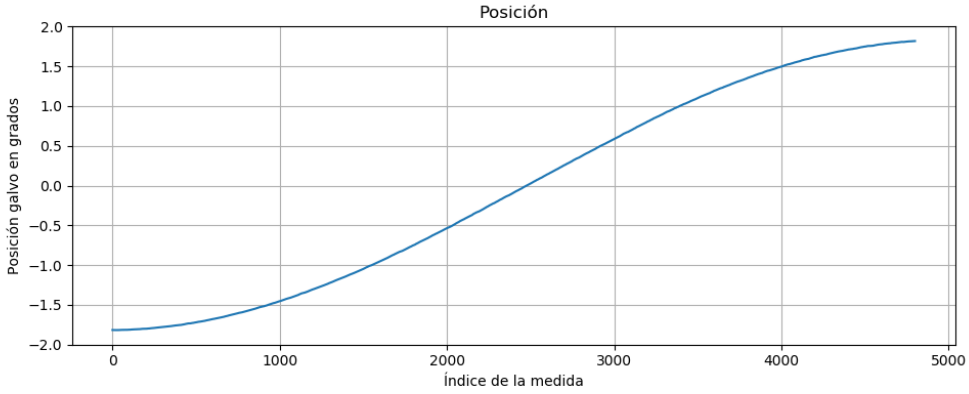

Figura 52. Posición angular del galvo en grados.

En esta figura se ve como el galvo sigue la referencia de posición sinusoidal generada.

El sensor Conopoint entrega también valor de SNR (*Signal to Noise Ratio*), que da una idea de la calidad de la señal.

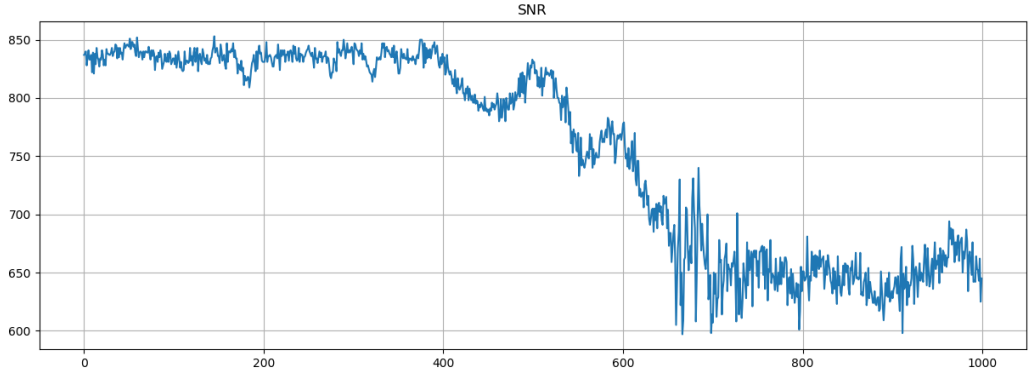

Figura 53. Medida del SNR de la señal.

Comparando esta gráfica con la de la Figura 51, se aprecia que la calidad de la señal disminuye cuando aumenta la distancia del punto al sensor. Esto se debe a que la distancia está acercándose al límite del rango de trabajo.

Con el fin de comprobar el comportamiento del sistema de cara a la detección de defectos se han hecho distintas pruebas. A continuación, se mostrarán dos experimentos con muestras de chapa doblada por embutición. Al doblar la chapa, esta sufre un estiramiento que, si es demasiado severo, puede provocar hasta la rotura del material.

Este primer ejemplo muestra una pieza que ha llegado a romper, por lo que es un defecto fácil de encontrar. Se observa a simple vista.

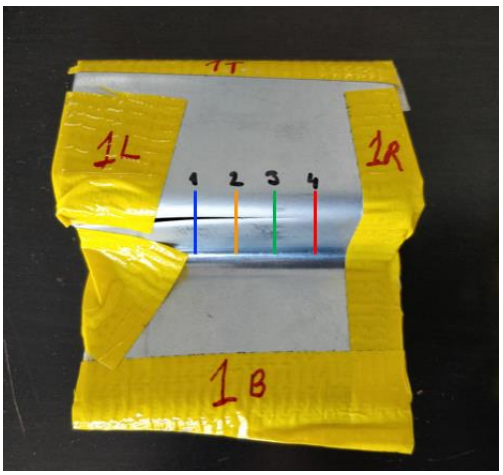

Figura 54. Pieza que presenta una rotura. Están marcados con líneas de colores los perfiles examinados con el sensor.

Se han examinado cuatro líneas diferentes de la pieza, con el fin de comparar zonas con y sin rotura. Los resultados obtenidos son los siguientes:

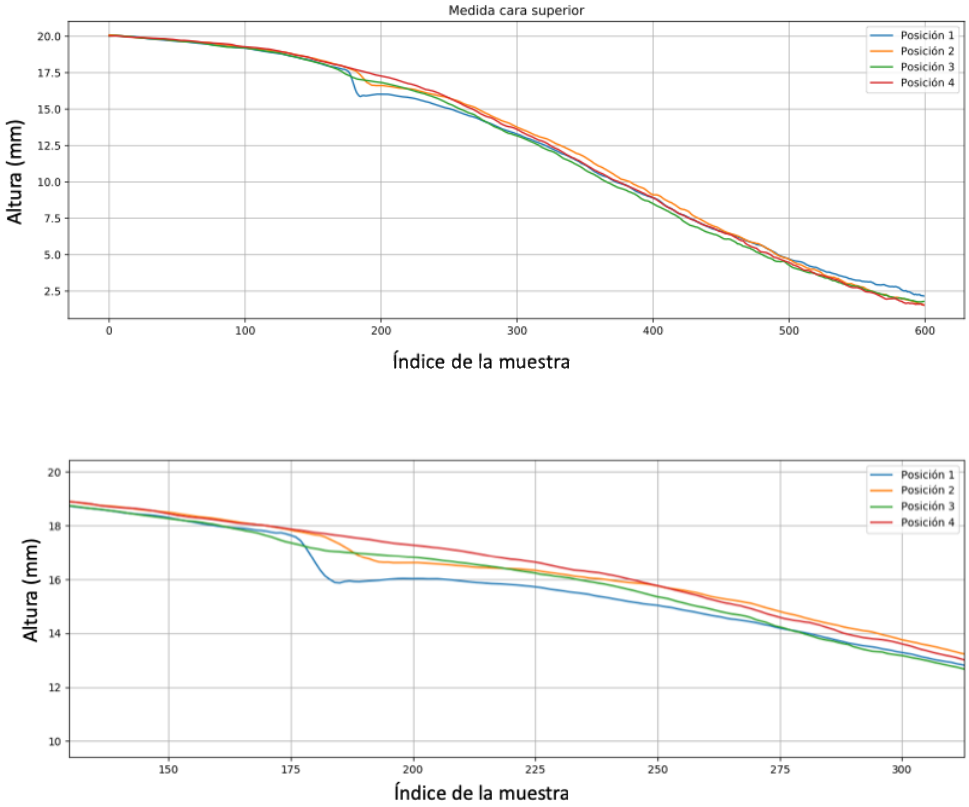

Figura 55. Arriba los perfiles obtenidos. Abajo ampliada la zona del defecto. En el eje Y se representa la altura de la pieza en milímetros, en el X el índice de la muestra.

Esta pieza presenta una rotura visible a simple vista, lo que queda resaltado en forma de abruptos saltos. En la zona de rotura, la señal adquirida presenta discontinuidades, aunque en la figura se presenten suavizadas por el filtrado, por lo que estos son defectos sencillos de localizar.

La siguiente pieza presenta algo más de complejidad, ya que su defecto es mucho más sutil. En este caso no hay rotura, solo un adelgazamiento excesivo.

La pieza es la siguiente:

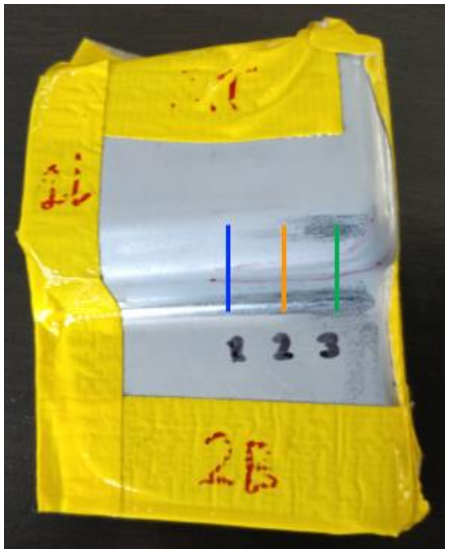

Figura 56. Pieza que presenta un adelgazamiento excesivo. Están marcados con líneas de colores los perfiles examinados con el sensor.

Los tres perfiles examinados dan los siguientes resultados:

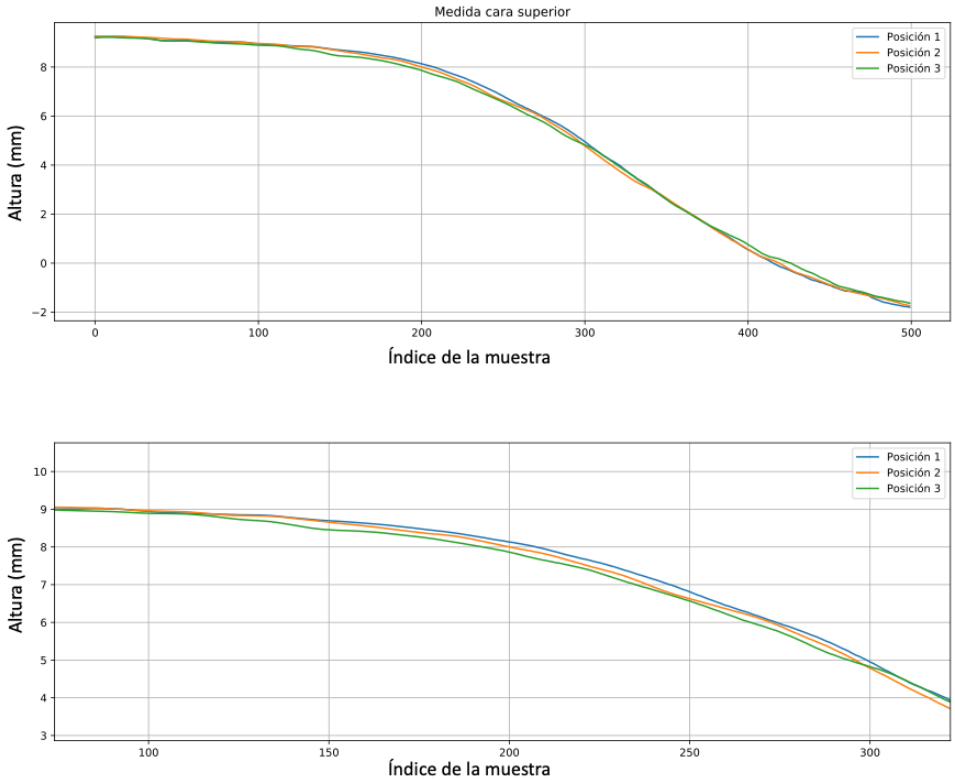

Figura 57. Arriba los perfiles obtenidos. Abajo ampliada la zona del estiramiento.

En esta pieza no hay rotura, por lo que el defecto no se observa tan fácilmente. En la vista en detalle se intuye que puede haber defecto en el tercer perfil, que presenta una pequeña reducción de espesor alrededor de la muestra 150.

Acercándose más a la zona y eliminando la tendencia de la curva puede apreciarse mejor.

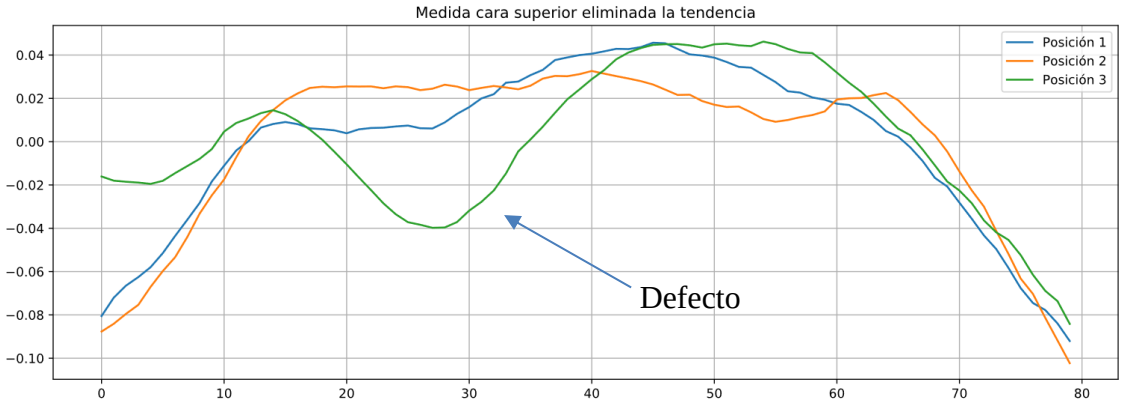

Figura 58. Detalle del defecto. Obsérvese que los perfiles presentan cambios abruptos en la curvatura.

En esta figura, el defecto asociado al estiramiento aparece con más claridad. Si pieza fuera ideal, las tres curvas no presentarían cambios de tendencia, sería una curva suave. Sin embargo, las tres curvas presentan el defecto en mayor o menor medida. El caso más claro es el del perfil número 3, que presenta una clara depresión en la zona del estiramiento, de alrededor de una décima de milímetro.

Una vez comprobado el funcionamiento de un sensor de manera individual, se ha realizado un montaje de dos sistemas de adquisición en un robot, que en el futuro servirá para medir espesores de chapa.

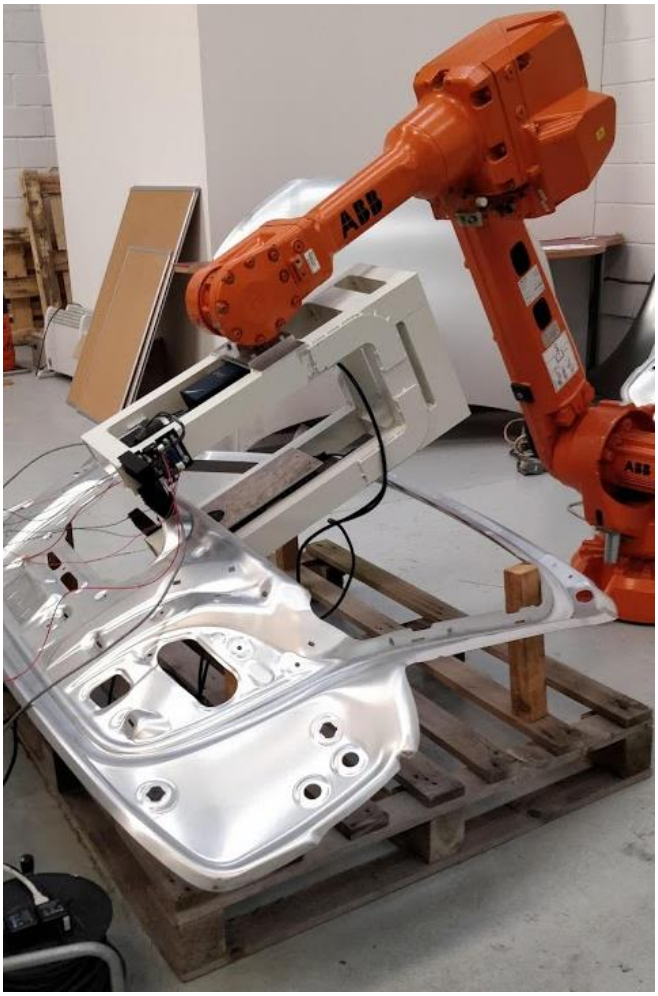

Figura 59. Montaje con dos sistemas de adquisición colocados para medir las dos caras de la chapa de una puerta de coche.

Con los dos sensores funcionando simultáneamente se obtienen los siguientes resultados en una chapa plana. Puede observarse una curvatura en el gráfico, puesto que aún no está modelado el aumento de distancia al sensor en función del ángulo del espejo.

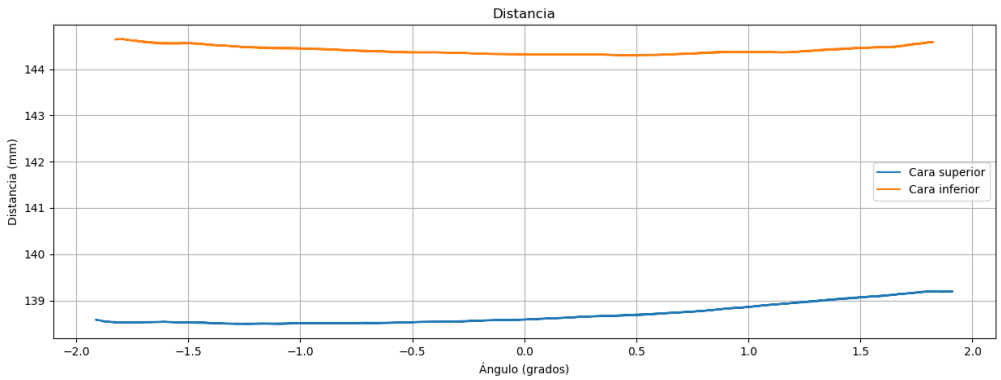

Figura 60. Medida de distancia en ambas caras.

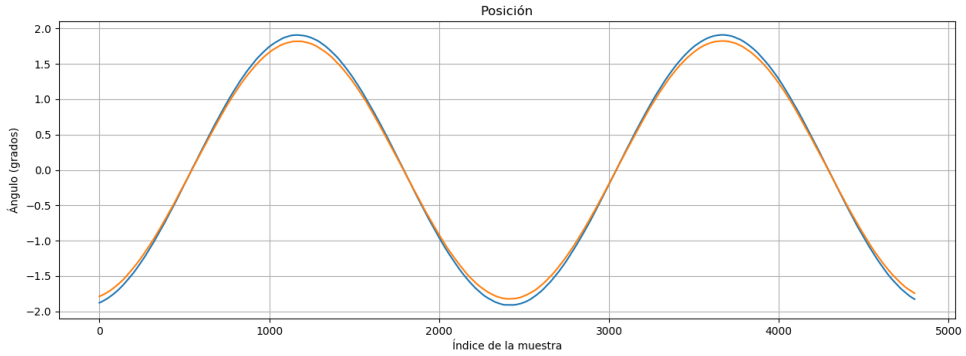

Figura 61. Posición de cada unos de los galvo durante la adquisición.

Puede observarse que uno de los galvos no llega a la misma amplitud que el otro. Esto se achaca a caídas de tensión debidas al montaje electrónico provisional, realizado en una protoboard y sin las conexiones entre cables y componentes más adecuadas.

5. DISCUSIÓN

Las líneas de trabajo para continuar este proyecto son claras. Por un lado, es necesario realizar el diseño físico definitivo, integrando los distintos componentes entre sí en una PCB, de forma que el sistema de adquisición pueda incluirse en un proyecto de inspección final, implantable dentro del proceso industrial.

Por otro lado, el sistema actual funciona con un solo sensor de holografía conoscópica por cada MicroZed. Como el sistema Zynq presenta dos núcleos ARM, y actualmente solo se está utilizando uno, sería conveniente habilitar el segundo núcleo para gestionar la comunicación con un segundo sensor, con el fin de comprobar si es posible gestionar todo el tráfico con un solo dispositivo. De esta forma se podría aprovechar de mejor manera las posibilidades del sistema Zynq.

Hasta el momento, el sistema se ha desarrollado en modo *standalone*, sin sistema operativo. Sería interesante comprobar si la utilización de un sistema operativo. Ya sea Linux o FreeRTOS, permite seguir atendiendo las distintas tareas de tiempo real y no añade una capa superior demasiado pesada limitando la optimización de las distintas tareas y la comunicación con los distintos IP cores de la FPGA. El uso de un sistema operativo aumentaría la versatilidad del sistema, al dar acceso a un sistema de archivos, uso simultáneo de los dos núcleos, planificación de tareas, uso de sockets...

Por último, sería conveniente modelar el aumento de distancia que se produce en la medida del sensor debido al movimiento del espejo, de manera que se puedan reconstruir de manera precisa perfiles de las piezas medidas.

6. CONCLUSIONES

En primer lugar, cabe concluir que los objetivos marcados al inicio del proyecto se han cumplido exitosamente, puesto que se ha logrado implementar de manera satisfactoria el sistema de adquisición de datos. A la conclusión de este trabajo final de Máster, el sistema es capaz de tomar medidas con el sensor Conopoint y sincronizar con estas la generación de una señal analógica y la lectura de otra, enviando los datos en otra trama UDP con idéntico formato al leído del sensor.

El sistema de adquisición es capaz de medir 20000 puntos por segundo, con una resolución de 10µm.

A modo de recapitulación, no está de más recuperar la figura mostrada con anterioridad, que refleja la arquitectura del sistema implementado. Se ha resuelto de manera satisfactoria la comunicación entre los distintos elementos del sistema, tanto con los elementos externos como internos.

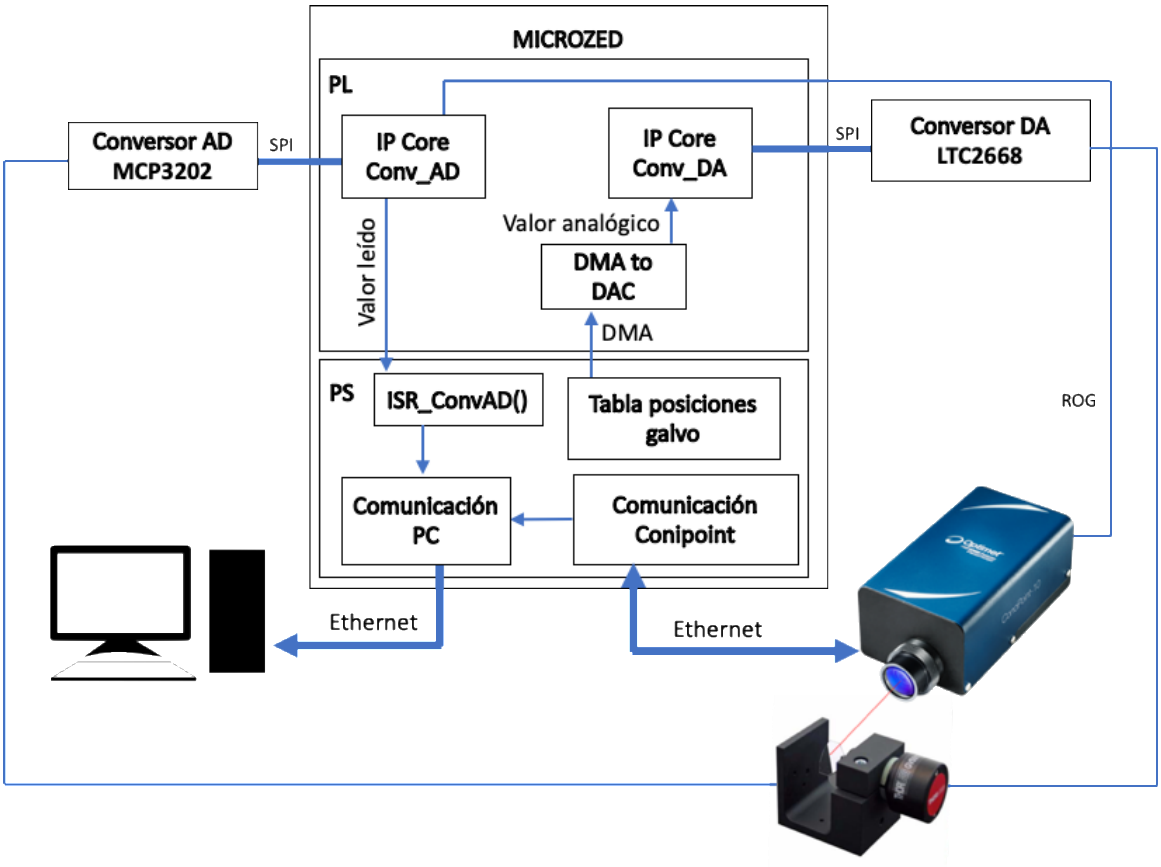

Figura 62. Diagrama de bloques del sistema implementado.

En cuanto a las competencias adquiridas y el desarrollo personal, este proyecto ha servido principalmente para aprender sobre temas muy dispares. En primer lugar, ha supuesto un primer contacto con sistemas de medición y detección de defectos basados en holografía conoscópica. También ha servido como primera toma de contacto con la programación no solo de FPGAs, sino de sistemas embebidos que integran FPGAs con microprocesadores, lo que permite diseñar sistemas muy versátiles y que cumplan con altos requerimientos temporales. En cuanto a ámbitos que resultaban algo más familiares, cabe destacar el aprendizaje relativo a comunicaciones, mayoritariamente mediante el protocolo UDP, aunque también la implementación a bajo nivel de comunicación SPI.

7. REFERENCIAS

[1] «ConoPoint-20, 3D non contact measurement – Optimet». [En línea]. Disponible en: https://www.optimet.com/conopoint-20.php. [Accedido: 28-nov-2018].

[2] «MicroZed | Zedboard». [En línea]. Disponible en: http://www.zedboard.org/product/microzed. [Accedido: 03-dic-2018].

[3] O. Vogel y E. Cristiani, «Numerical schemes for advanced reflectance models for Shape from Shading», 2011, pp. 5-8.

[4] «Photogrammetry», *Wikipedia*. 05-feb-2019.

[5] T. Bell, B. Li, y S. Zhang, «Structured Light Techniques and Applications», en *Wiley Encyclopedia of Electrical and Electronics Engineering*, Hoboken, NJ, USA: John Wiley & Sons, Inc., 2016, pp. 1-24.

[6] C. Roychoudhuri, *Fundamentals of Photonics*. 1000 20th Street, Bellingham, WA 98227-0010 USA: SPIE, 2008.

[7] R. Loughridge y D. Abramovitch, «A tutorial on laser interferometry for precision measurements», en *Proceedings of the American Control Conference*, 2013, pp. 3686-3703.

[8] P. Jacquot, «Speckle Interferometry: A Review of the Principal Methods in Use for Experimental Mechanics Applications: Methods of Speckle Interferometry», *Strain*, vol. 44, n.º 1, pp. 57-69, ene. 2008.

[9] G. Sirat y D. Psaltis, «Conoscopic holography», *Opt. Lett.*, vol. 10, n.º 1, p. 4, ene. 1985.

[10] I. Álvarez, J. M. Enguita, M. Frade, J. Marina, y G. Ojea, «On-Line Metrology with Conoscopic Holography: Beyond Triangulation», *Sensors*, vol. 9, n.º 9, pp. 7021-7037, sep. 2009.

[11] «Zynq-7000 SoC». [En línea]. Disponible en: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html. [Accedido: 03-dic-2018].

[12] A. Ltd, «Cortex-A9», *ARM Developer*. [En línea]. Disponible en: https://developer.arm.com/products/processors/cortex-a/cortex-a9. [Accedido: 03-dic-2018].

[13] «Zynq-7000 SoC Technical Reference Manual (UG585)», p. 1843, 2018.

[14] «9239-20459-xilinx_driver_gpiops_v1_02_a_xgpiops.pdf». [En línea]. Disponible en: http://xilinx.eetrend.com/files-eetrend-xilinx/forum/201509/9239-20459-xilinx_driver_gpiops_v1_02_a_xgpiops.pdf. [Accedido: 13-dic-2018].

[15] «ug903-vivado-using-constraints.pdf». [En línea]. Disponible en: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_1/ug903-vivado-using-constraints.pdf. [Accedido: 13-dic-2018].

[16] «Vivado Design Suite Tcl Command Reference Guide (UG835)», p. 1802, 2018.

[17] A. Ltd, «AMBA Specifications – Arm», *Arm | The Architecture for the Digital World*. [En línea]. Disponible en: https://www.arm.com/products/silicon-ip-system/embedded-system-design/amba-specifications. [Accedido: 04-dic-2018].

[18] «pg059-axi-interconnect.pdf». [En línea]. Disponible en: https://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v2_1/pg059-axi-interconnect.pdf. [Accedido: 04-dic-2018].

[19] «pg247-smartconnect.pdf». [En línea]. Disponible en: https://www.xilinx.com/support/documentation/ip_documentation/smartconnect/v1_0/pg247-smartconnect.pdf. [Accedido: 04-dic-2018].

[20] «ug1037-vivado-axi-reference-guide.pdf». [En línea]. Disponible en: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf. [Accedido: 04-dic-2018].

[21] «Familia de protocolos de internet», *Wikipedia, la enciclopedia libre*. 20-nov-2018.

[22] «Protocolo de datagramas de usuario», *Wikipedia, la enciclopedia libre*. 17-oct-2018.

[23] «Modelo OSI», *Wikipedia, la enciclopedia libre*. 05-dic-2018.

[24] «Serial Peripheral Interface», *Wikipedia, la enciclopedia libre*. 01-oct-2018.

[25] «Our Technology| Collinear Sensors | Optimet». [En línea]. Disponible en: https://www.optimet.com/our_technology.php. [Accedido: 30-ene-2019].

[26] «GVS311-Manual.pdf». [En línea]. Disponible en: https://www.thorlabs.com/drawings/aacd41de28ade1fb-F62FF058-DE68-91E0-A849240AD9BC96E0/GVS311-Manual.pdf. [Accedido: 04-ene-2019].

[27] «MCP3202». [En línea]. Disponible en: http://ww1.microchip.com/downloads/en/devicedoc/21034d.pdf. [Accedido: 04-ene-2019].

[28] «LTC2668 - 16-Channel 16-/12-Bit ±10V Vout SoftSpan DACs with 10ppm/°C Max Reference», p. 28.

[29] «Vivado Design Suite». [En línea]. Disponible en: https://www.xilinx.com/products/design-tools/vivado.html. [Accedido: 30-ene-2019].

[30] «pg082-processing-system7.pdf». 10-may-2017.

[31] «Processor System Reset Module v5.0 LogiCORE IP Product Guide (PG164)», p. 33, 2015.

[32] «pg021_axi_dma.pdf». [En línea]. Disponible en: https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf. [Accedido: 29-ene-2019].

[33] «Adam Taylor's MicroZed Chronicles Part 38 – Answering a question on Interrupts.», 23-jun-2014. [En línea]. Disponible en: https://forums.xilinx.com/t5/Xcell-Daily-Blog-Archived/Adam-Taylor-s-MicroZed-Chronicles-Part-38-Answering-a-question/ba-p/479978. [Accedido: 27-ene-2019].

[34] «lwIP: Overview». [En línea]. Disponible en: http://www.nongnu.org/lwip/2_1_x/index.html. [Accedido: 11-ene-2019].

[35] «File descriptor», *Wikipedia*. 03-oct-2018.

[36] «Raw/UDP», *lwIP Wiki*. [En línea]. Disponible en: http://lwip.wikia.com/wiki/Raw/UDP. [Accedido: 25-ene-2019].

[37] «Savitzky–Golay filter», *Wikipedia*. 07-ene-2019.

PRESUPUESTO

ÍNDICE DEL PRESUPUESTO

1. COSTE DE EJECUCIÓN MATERIAL

El coste de ejecución material incluye tres categorías, coste de equipos, coste de software y coste de mano de obra por el tiempo empleado en el proyecto.

1.1 COSTES DE EQUIPOS

Se supone un período de amortización del ordenador de cinco años. El proyecto se desarrolla a lo largo de siete meses.

CONCEPTO	PRECIO UNITARIO	CANTIDAD	SUBTOTAL
Dell Inspiron 15 7000 i7 16GB	1200	8/20	480
Sensor Conopoint 20	6000	2	12000
Lente 75Emm	1500	2	3000
Galvo GVS311	1600	2	3200
Fuente alimentación galvo	400	1	400
Conversor AD MCP3201-C	2,04	1	4,08
Conversor DA LTC2668-16	88,43	1	176,86
Otros componentes electrónicos	10,00	1	10,00
Xilinx MicroZed Evaluation Kit	162,14	1	324,28
MicroZed Breakout Carrier Card	59,00	1	118,00
Switch ethernet	20,94	1	20,94

Subtotal: …… 19734,16 €

1.2 COSTES DE SOFTWARE

Se estima un período de amortización de Matlab de 3 años.

CONCEPTO	PRECIO UNITARIO	CANTIDAD	SUBTOTAL
Xubuntu 18.04	0	1	0
Vivado webpack 2018.2	0	1	0
Microsoft Office 365	8,80 €/mes	8	70,40
Qt Creator	0 €	1	0
Matlab 17.04	2000	8/20	800

Subtotal: …… 870,40 €

1.3 COSTES DE MANO DE OBRA

Se estima un trabajo de 8 horas al día y 5 días a la semana, con un coste de 35€ la hora de ingeniería.

CONCEPTO	CANTIDAD (Días)	CANTIDAD (HORAS)	SUBTOTAL
Estudio plataforma microzed	20	160	5600
Implementación comunicación conversor AD	15	120	4200
Implementación conversor	20	160	5600

DA			
Implementación comunicación UDP	15	120	4200
Modificación librerías Conoprobe para compatibilidad con microzed	20	160	5600
Integración de las distintas partes	10	80	4200
Software adquisición PC	2	16	5600
Pruebas de funcionamiento	10	80	2800
Procesamiento medidas	10	80	2800
Redacción de documentos	10	80	2800

Subtotal: ……. 36960 €

1.4 COSTE TOTAL DEL PRESUPUESTO DE EJECUCIÓN MATERIAL

CONCEPTO	SUBTOTAL
Coste de equipos	19734,16 €
Coste de software	870,40 €
Coste de mano de obra	36960 €

Subtotal: ……. 57565,56 €

2. GASTOS GENERALES Y BENEFICIO INDUSTRIAL

Los gastos generales y beneficio industrial son los gastos obligados que se derivan de la utilización de las instalaciones de trabajo más el beneficio industrial. Se estima un porcentaje del 16 % sobre el coste de ejecución material

CONCEPTO	SUBTOTAL
Gastos generales y beneficio industrial	9210,33 €

3. IMPORTE TOTAL

CONCEPTO	SUBTOTAL
Coste total del presupuesto de ejecución material	**57564,56 €**
Gastos generales y beneficio industrial	**9210,33 €**

TOTAL: ……. 66774,89 €

IVA 21%: ……. 14022,73 €

TOTAL, IVA INCLUÍDO: …… 80797,62 €

El Importe Total del proyecto suma la cantidad de:

Ochenta mil setecientos noventa y siete euros con sesenta y dos céntimos.

PLANIFICACIÓN TEMPORAL

1. CRONOGRAMA

La planificación temporal del proyecto se describe en el siguiente cronograma.

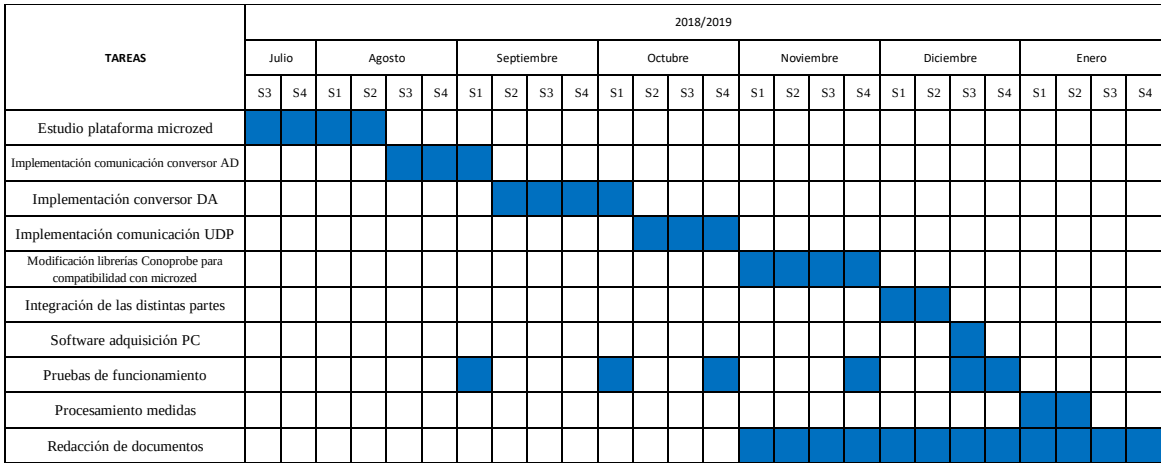

Figura 1. Planificación temporal del proyecto.