ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Plataforma para realización de rutinas de vuelo autónomo sobre cuadricóptero
(Platform to perform autonomous flight routines on quadcopter)

Para acceder al Título de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Ramón González Ruiz

Julio - 2016

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Ramón González Ruiz
Director del TFG: Eugenio Villar Bonet
Título: "Plataforma para realización de rutinas de vuelo autónomo sobre cuadricóptero"
Title: "Platform to perform autonomous flight routines on quadcopter"

Presentado a examen el día:

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente: Villar Bonet, Eugenio
Secretario: Pascual Gutiérrez, Juan Pablo
Vocal: Sánchez Espeso, Pablo Pedro

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente	Fdo.: El Secretario
Fdo.: El Vocal	Fdo.: El Director del TFG (sólo si es distinto del Secretario)
Vº Bº del Subdirector	Trabajo Fin de Grado Nº (a asignar por Secretaría)

Agradecimientos

Me gustaría reconocer en este trabajo el apoyo y la confianza de todos aquellos que me han acompañado estos años y también de los que lo harán en el futuro.

A todas las personas del laboratorio de sistemas embebidos por su ayuda y compañía durante los meses en los que se ha desarrollado este trabajo.

Y sobre todo a la familia y amigos, aquellos que con su apoyo y compañía hacen que uno tenga motivos para seguir trabajando.

Resumen:

El objetivo de este trabajo es convertir un cuadricóptero diseñado para ser dirigido con un mando de radiofrecuencia en una plataforma de experimentación y docencia capaz de realizar rutinas de vuelo autónomas, tomar información del exterior y procesarla para tomar decisiones.

La motivación principal para querer contar con una plataforma de este tipo es el auge del mercado de los sistemas ciberfísicos y en concreto el de los UAV y la conveniencia de la incorporación de este tipo de sistemas a la docencia y a la investigación universitaria.

Esta plataforma permite el desarrollo, sobre un sistema ciberfísico, de aplicaciones que requieran integrar software y hardware en el mismo sistema, proporcionando un alto nivel de prestaciones, flexibilidad y escalabilidad, al poder dividir la funcionalidad con total libertad entre la lógica programable y el software según los requisitos y especificaciones de cada aplicación.

Abstract

The objective of this project is to transform a quadcopter designed to be flight with a radio frequency transmitter into an experimentation and teaching platform able to perform autonomous flight routines, taking information from the environment and processing it to take decisions.

The main reason to desire a platform of this kind is the rise in the market of cyber physical systems and UAV particularly and the convenience of the incorporation of this type of systems into teaching and research in the University.

This platform allows the development, on a cyber physic system, of applications that demand integrating software and hardware under the same system, providing a high level of performance, flexibility and scalability by allowing to divide the functionality with total freedom between programmable logic and software according to the requirements and specifications of each application.

Índice

Introducción:

Motivación

El rápido desarrollo de la tecnología que se ha producido en el campo de los UAV en los últimos años representa una gran oportunidad de mercado.

Los vehículos aéreos no tripulados ya están fuertemente implantados en la industria militar y en los últimos años ha comenzado su implantación en campos de la vida civil. En los próximos años se espera un fuerte crecimiento del peso de este tipo de sistema ciberfísico en la economía.

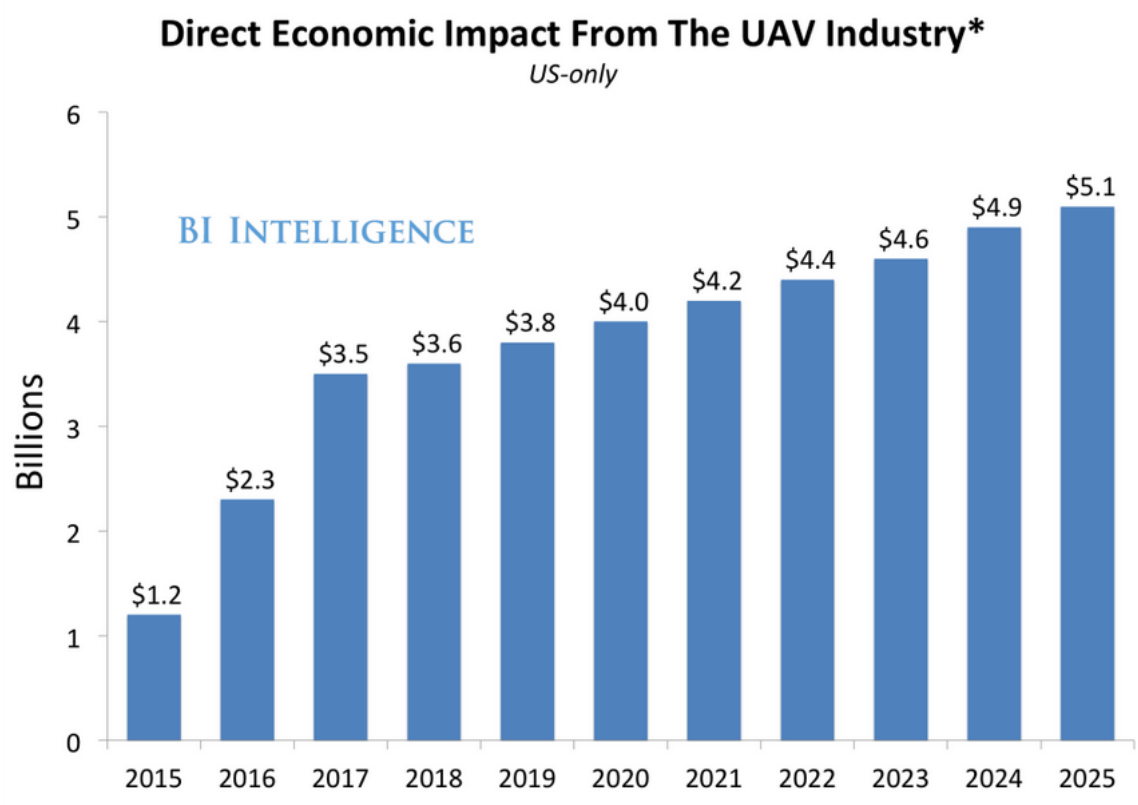
Según un artículo de la revista Business Insider[i] el impacto directo de este tipo de vehículos en la economía es de 1.200 millones de dólares, solamente en EEUU, y se espera que su peso en la economía aumente considerablemente en los próximos años, como puede observarse en la ***Figura 1***.

Según el informe de esta revista el mercado comercial de drones formará parte de soluciones en varios tipos de aplicaciones: agricultura, energía, minería, construcción, noticias y rodaje de películas. La mayor parte del crecimiento será en la parte civil, con una Tasa de Crecimiento Anual Compuesto (CAGR) del 19% entre 2015 y 2020, mientras que en la parte militar será del 5%.

También señala que el comercio electrónico y los envíos no serán a corto plazo el eje principal de la industria de los drones y que los fabricantes que ya existen centrados en clientes militares no tienen por qué tener necesariamente ventaja en el terreno civil, abriéndose por tanto una gran oportunidad para que nuevas empresas se introduzcan en este mercado.

A nivel global en el campo de la seguridad y defensa se podría esperar una Tasa de Crecimiento Anual Compuesto del 4.6% para 2022, según el estudio de Strategic Defence Intelligence [ii], esto es, un incremento de 68.6 miles de millones de dólares.

En el campo de los UAV de pequeñas dimensiones, un artículo de Research and Markets[iii] publicado en la revista Bussiness Wire el 12 de febrero de 2016, sitúa la previsión para la Tasa de Crecimiento Anual Compuesto en un 6.52% en el periodo 2016-2020.



iv

Figura 1 – Expectativas de la Evolución del Impacto Económico de la Industria de los UAV en EEUU según el artículo de Bussiness Insider

Estructura

Este trabajo se divide en tres bloques fundamentales: la introducción, el desarrollo y descripción del contenido y finalmente las conclusiones.

En el primer apartado se trata el concepto de los sistemas ciberfísicos, y se introducen las características de la plataforma sobre la cual se desarrolla el trabajo, la especificación propuesta y una descripción superficial de los componentes del sistema y su funcionalidad.

En el segundo se trata en mayor profundidad el proceso de diseño de cada uno de los distintos bloques que componen el proyecto, describiendo con detalle su funcionamiento y características.

En la tercera se propone una pequeña guía para el uso y desarrollo de aplicaciones sobre la plataforma, describiendo el modo de uso de cada una de las partes que componen el sistema.

En la cuarta se comentan los resultados finales y se definen las futuras líneas de trabajo basadas en esta plataforma por las que se podría apostar en el futuro.

Sistemas ciberfísicos:

El término cyber-physical systems (CPS)[v] hace referencia a una nueva generación de sistemas con capacidades computacionales y físicas integradas que pueden interactuar con los humanos de varias formas.

Algunos ejemplos de desarrollo en este campo son las nuevas generaciones de vehículos aéreos y aeroespaciales, automóviles híbridos, automóviles de conducción autónoma…

El mayor reto para el desarrollo de estos sistemas es el desarrollo de métodos asequibles para diseñar, analizar y verificar componentes a varios niveles de abstracción, analizar y comprender las interacciones entre el sistema de control del vehículo y los subsistemas físicos del mismo (frenos, motores, transmisión…) y asegurar la fiabilidad y estabilidad del vehículo manteniendo el coste lo más bajo posible.

El hardware y software deben ser altamente fiables, reconfigurables, y cuando sea requerido, certificable, desde el nivel de componente al del sistema integrado completo. La certificación es uno de los factores más importantes en este tipo de sistemas, se estima que ésta consume más del 50% de los recursos requeridos para desarrollar nuevos sistemas con fiabilidad crítica en el caso de la industria de la aviación.

Especificación:

El proyecto consistirá en el diseño de un sistema que permita el manejo de un cuadricóptero a través de software sobre una plataforma Linux. El sistema estará basado en la placa de desarrollo ZedBoard que contiene dos cores ARM Cortex A9 y lógica programable. El objetivo es que el programador pueda definir rutinas de vuelo a través de software de alto nivel además de obtener información de distintos sensores que permitan conocer el estado del cuadricóptero en cada momento. El esquema del sistema que se deberá desarrollar se presenta en la ***Figura 2.***

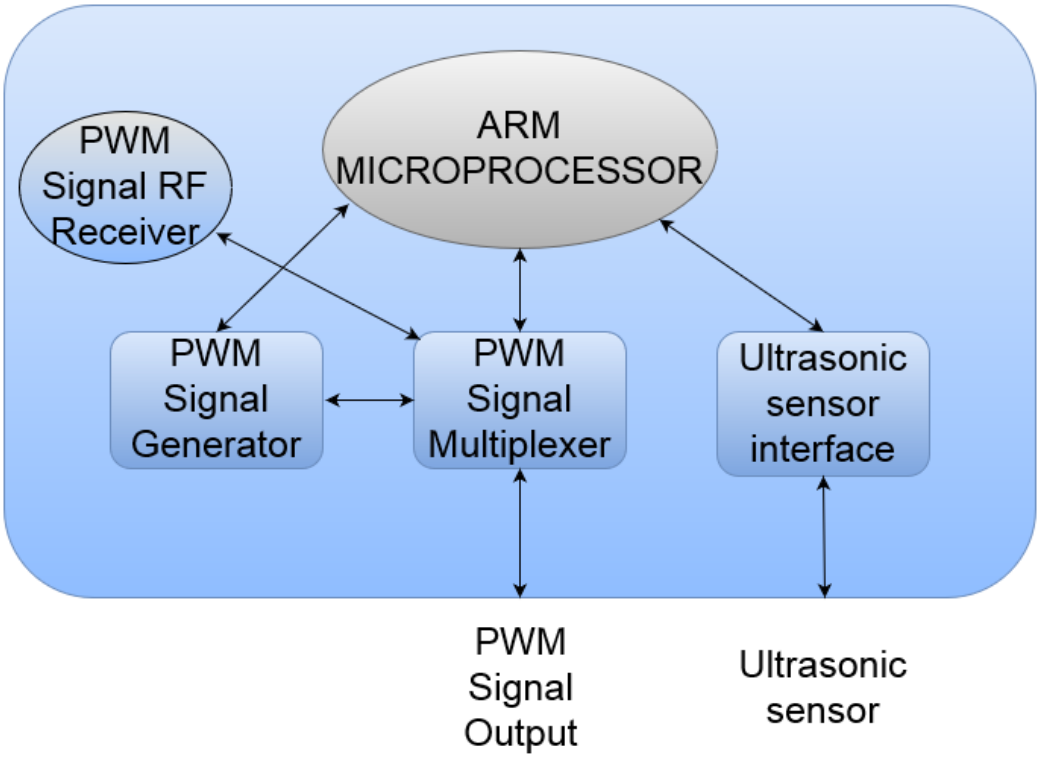


Figura 2 - Diagrama de bloques del sistema

1- El sistema deberá funcionar sobre un SO Linux. El soporte de drivers para cámaras será el mayor factor a tener en cuenta en cuanto a la elección del tipo de Linux en concreto.

2- El sistema recibirá las señales PWM del mando del cuadricóptero y las enviará cuando éste esté siendo usado, en lugar de las generadas por el software del sistema.

3- El control de los módulos hardware que direccionan y generan las señales PWM necesarias para realizar el vuelo programado se realizará mediante funciones software de alto nivel HDS-3 en la plataforma Linux.

4- Contará con un módulo hardware capaz de generar las señales PWM de control de los motores dirigido mediante software por las funciones de alto nivel

HDS-3[1].

5- Contará con un módulo multiplexor encargado de enviar a la salida las señales generadas a través del software o las señales del mando cuando se esté utilizando. El funcionamiento del módulo será programable mediante funciones de alto nivel HDS-3 y se podrán filtrar y enviar sólo determinadas señales del mando (ej.: sólo transmitir las señales del control horizontal).

6- El sistema deberá contar con distintos sensores: altímetro, cámara, ultrasonidos y GPS. Estos sensores proporcionarán información al software para el control del cuadricóptero. Los sensores de ultrasonidos irán orientados a la detección de objetos cercanos y a la corrección de la trayectoria en caso de que sean detectados.

a. Altímetro: MPL3115A2+ Placa de Montaje
   Conexión: I2C
   Tensión: 3.3V
   Datasheet
   Enlace Placa

b. Cámara: Logitech c110
   Conexión: USB
   Página del producto

c. Ultrasonidos: HC-SR04
   Conexión: 2 Puertos GPIO
   Tensión: 5V
   Datasheet
   Enlace Sensor

d. GPS: Adafruit Industries LLC 746
   Conexión: UART
   Tensión: 3.3V
   Enlace GPS
   Enlace Datasheet

7- La alimentación del sistema requerirá 12V y al menos 5W (> 0.42 A) para la placa y 3.3V y 5V para los sensores a través de las tomas disponibles en la placa. Se empleará la batería del cuadricóptero (14.8V) y se empleará un conversor DC-DC de tipo buck para obtener 12V. El conversor empleado será un Murata Power Solutions Inc. UWE-12/6-Q12P-C, que proporciona una corriente máxima de 2.5A, más que suficiente para alimentar la placa. Datasheet.

[1] Hardware Dependent Software

8- La placa se colocará atornillada bajo una lámina de metacrilato anclada al cuadricóptero. Las dimensiones aproximadas de la lámina serán 17.5 cm x 14 cm.

Plataforma de trabajo

El proyecto se desarrolla sobre un vehículo aéreo no tripulado con cuatro motores llamado Mikrokopter Quadro XL desarrollado por la empresa alemana HiSystems GmbH cuya imagen puede verse en la ***Figura 3***.

Figura 3 - Mikrokopter Quadro XL

Un cuadricóptero es un vehículo aéreo que cuenta con cuatro hélices en lugar de las dos con las que cuenta un helicóptero tradicional. Para lograr mantener la estabilidad dos de las hélices deben rotar en un sentido y las otras dos en otro. Esta arquitectura permite una mayor capacidad de maniobra que la de un helicóptero tradicional al poder controlar los distintos posibles movimientos alterando la velocidad relativa de los rotores.

El Mikrokopter Quadro dispone de dos tipos de placas con electrónica:
La placa Flight Ctrl V2.1, a la vista en la ***Figura 4***, que cuenta con la lógica para transformar las señales que provienen del receptor de radiofrecuencia en impulsos para los motores, así como manejar los procesos de calibrado, de toma de datos para mantener la estabilidad mediante la sensórica que incluye y la posibilidad de compartir datos con el mando del UAV a través del receptor de radiofrecuencia.

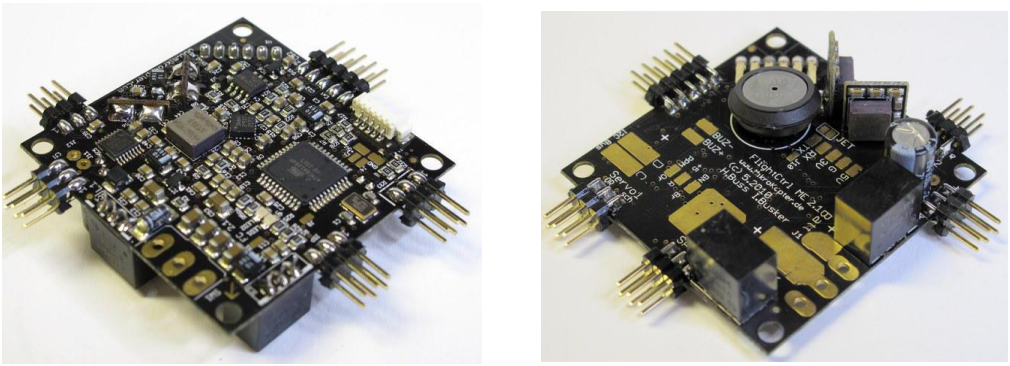


Figura 4 - Flight Ctrl 2.1

Cuatro placas BL-Ctrl V2.0, representada en la ***Figura 5***, que se encargan de transmitir la potencia a los cuatro motores según las instrucciones recibidas de la Flight Ctrl a través de una señal I2C.

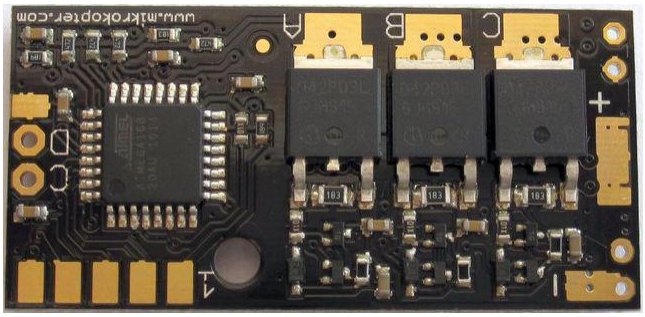


Figura 5 - Bl-Ctrl 2.0

El sistema recibe las instrucciones de un mando de tipo Graupner MC-20 HoTT:
El mando cuenta con dos sticks principales para modificar los valores de la potencia (gas) y el ángulo de navegación (guiñada/yaw, cabezeo/pitch y alabeo/roll), como puede verse en la ***Figura 7***. En la ***Figura 6*** se representa gráficamente cada uno de los ángulos de navegación.

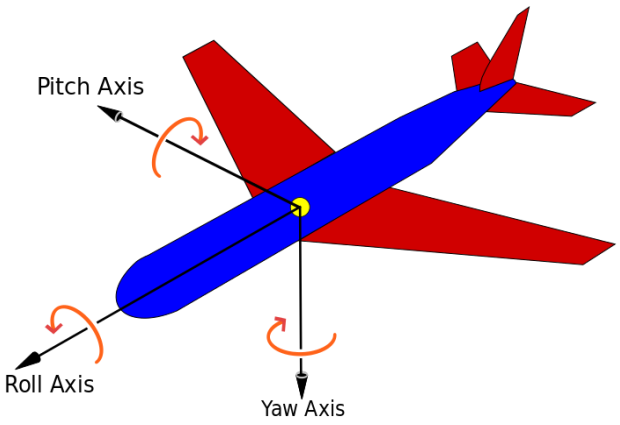


Figura 6 - Esquema de los ángulos de navegación[vi]

Figura 7 - Relación de controles del mando y ángulos de navegación

Es posible programar el mando para modificar la forma en la que actúan los controles. Por defecto los interruptores adicionales sirven para controlar distintas señales de telemetría. En el caso de este proyecto se ha empleado el interruptor 9 que controla el canal 8 de la señal emitida para decidir si el cuadricóptero funciona en modo autónomo o controlado por el mando.

El dispositivo encargado de recibir la señal de radiofrecuencia y transmitirla al Flight Ctrl es el receptor Graupner GR-16 HOTT, representado en la ***Figura 8*** que proporciona una señal de salida PPM[2] Sum12[3] a la placa.

El receptor es conecta a la placa a través del conector del canal 8 y envía la señal procedente del mando. A través del canal T recibe señales de telemetría y de confirmación para el mando desde la placa de control de vuelo.

Figura 8 - Receptor de radiofrecuencia Graupner GR-16

El sistema que permitirá que el cuadricóptero pueda realizar rutinas de vuelo de forma autónoma estará implementado en la placa de prototipado ZedBoard, a la vista en la ***Figura 9***. Esta placa está basada en la plataforma Zynq All Programmable SoC de Xilinx, que combina en el mismo chip lógica programable (FPGA), microprocesador y funcionalidad de E/S.

Este tipo de plataforma permite distribuir la funcionalidad entre hardware y software de forma que las funciones que requieran mayor peso en procesado de datos, fiabilidad o requisitos de tiempo real puedan implementarse en la parte de lógica programable mientras que la funcionalidad que requiera mayor flexibilidad o mayor nivel de abstracción puedan implementarse mediante código software.

En el caso concreto de este proyecto esta plataforma permite realizar las funciones más críticas y de tiempo real, que requieren una mayor fiabilidad, en la parte de lógica programable: generación de la señal para la placa de control de vuelo, detección del modo de vuelo autónomo o dirigido por mando y lectura de los sensores de distancia. En la parte software que corre sobre un sistema Linux se realiza la gestión de los algoritmos de vuelo analizando los valores recibidos de la parte hardware.

[2] Pulse-position modulation
[3] Señal de tipo PPM con 12 canales de información

Figura 9 - Zedboard

La herramienta de Diseño Electrónico por Computadora (EDA) empleada es Vivado, diseñada y distribuida por Xilinx. Esta herramienta permite emplear lenguajes de descripción hardware (HDL) para realizar diseños electrónicos, permitiendo la simulación de los mismos y el diseño mediante la integración de bloques de propiedad intelectual (IP), así como funciones como la descripción de bloques de hardware mediante código C o C++, Síntesis de Alto Nivel (HLS).

El medio que comunica la parte de la lógica programable con el microprocesador y el software que se ejecuta en él es el bus AXI[4]. Este bus es parte ARM AMBA, una familia de buses para microcontroladores introducida por primera vez en 1996. La primera versión de AXI fue incluida por primera vez en AMBA 3.0 en 2003. La versión que emplea este sistema, AXI4 fue incluida en la versión 4.0 de AMBA lanzada en 2010[vii].

Todo el sistema se alimenta mediante una batería de polímero de litio de 4 celdas Vislero 6600, con una tensión de 3,7V por celda, es decir, una tensión de 14,8V a la salida, y una capacidad de 6600 mAh. La ***Figura 10*** muestra una vista de la batería Vislero 6600.

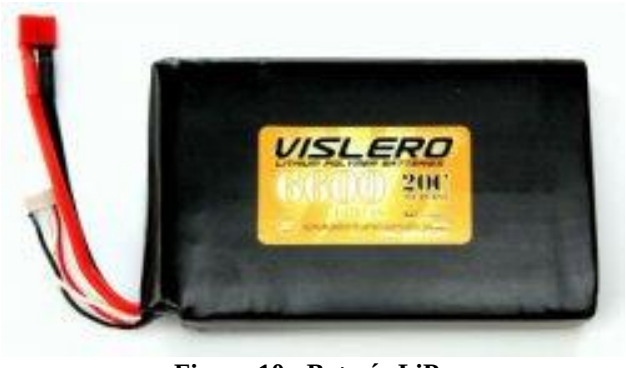


Figura 10 - Batería LiPo

[4] Advanced eXtensible Interface

Descripción del sistema electrónico diseñado:

El sistema está dividido fundamentalmente en dos partes: la parte hardware y la parte software.

Sobre la parte hardware, cuyo diagrama de bloques IP se muestra en la ***Figura 11*** se realizan las tareas que podrían requerir una mayor carga de procesado si se implementasen sobre software y que fuesen susceptibles de bloquear el microprocesador, así como las funciones críticas para la seguridad de la plataforma de vuelo, el multiplexado de las señales que recibe la placa de control de vuelo y la generación de la señal resultante de la rutina de vuelo. También se ha introducido en la parte hardware una serie de periféricos para la lectura de sensores de distancia mediante polling y evitar que este proceso ocupe tiempo de procesador.

Cada componente está dividido en dos partes: un IP descrito en VHDL para tener el mayor control posible y controlar ciclo a ciclo el funcionamiento del módulo y un IP descrito con síntesis de alto nivel (HLS) para comunicar el módulo VHDL con el bus de una forma sencilla.

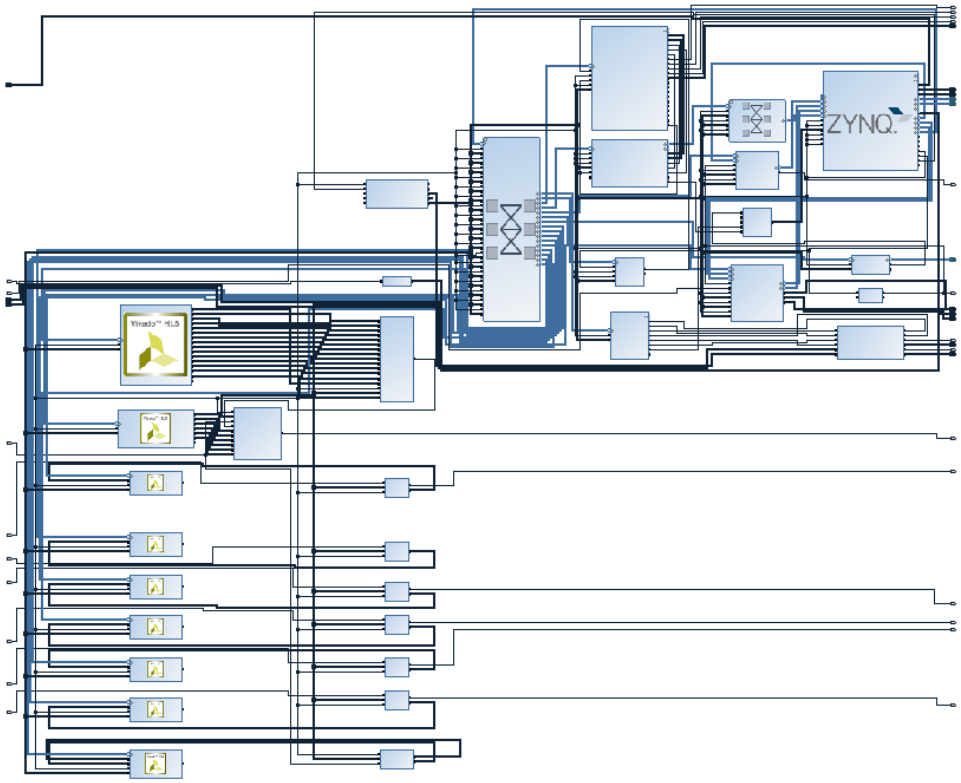


Figura 11 - Diagrama de bloques en Vivado

Además de la parte de lógica programable el sistema cuenta con otros componentes hardware discretos:

Cuatro sensores de tipo HC SR04, a la vista en la ***Figura 12***, para detección de obstáculos por ultrasonidos:

Figura 12 – Sensor de ultrasonidos HC SR04

Este sensor permite la detección y medición de la distancia a obstáculos situados hasta a 3 metros de distancia. Para liberar carga al procesador se ha diseñado un módulo hardware que envía la señal de disparo y obtiene el ancho de pulso con el que responde el sensor sin necesidad de bloquear el procesador.

- Un sensor de tipo MaxSonar Ez1, representado en la ***Figura 13***, para detección de obstáculos por ultrasonidos con una distancia de detección de hasta 6 metros:

-

Figura 13 - Sensor de ultrasonidos MaxSonar Ez1

Para la lectura de este dispositivo se ha modificado el hardware que realiza las lecturas de sensor HC SR04 adaptándolo a las características del MaxSonar Ez1.

Un conversor DC-DC Murata-Power-Solutions-Inc. UWE-12-6-Q12P-C, a la vista en la ***Figura 14***, para adaptar la salida de 14.8 V de la batería del cuadricóptero a las 12 V que necesita la ZedBoard para funcionar.

-

Figura 14 - Conversor DC-DC UWE-12-6-Q12P-C de Murata

La parte software del sistema corre sobre un Linux basado en Ubuntu llamado Linaro preparado para plataformas basadas en ARM. Sobre este sistema se ejecuta el código que permite configurar y dirigir el funcionamiento de los periféricos de la parte hardware, a través de una API compuesta por funciones software Hardware Dependent Software que se comunican a través del bus con el hardware.

Esta API está programada en C/C++ y realiza la comunicación con el bus y la parte hardware del sistema a través de la técnica de mapeo en memoria, solicitando al sistema operativo una dirección virtual que apunte a la dirección física solicitada. Cuenta con funciones para la calibración del cuadricóptero, la lectura de los valores que están recibiendo los motores, la lectura de los valores de los canales recibidos desde el mando, el control y la lectura de la potencia de los motores y la lectura de la sensórica incorporada en el sistema.

Al sistema operativo se le han añadido los módulos necesarios para poder hacer uso de una cámara webcam, manejar el bus I2C, así como la ejecución del programa de la rutina de vuelo automáticamente al arrancar el sistema.

Desarrollo:

Parte Hardware

La parte hardware del sistema está integrada por varios bloques IP conectados al bus AXI o a los distintos pines de la FPGA. Gran parte de estos módulos son periféricos para el sistema operativo proporcionados por Xilinx en una plantilla para el uso del puerto HDMI con la plataforma ZedBoard. Entre estos módulos se encuentran los que permiten el manejo de los buses I2C, I2S, HDMI y el módulo de distribución de la señal de reloj.

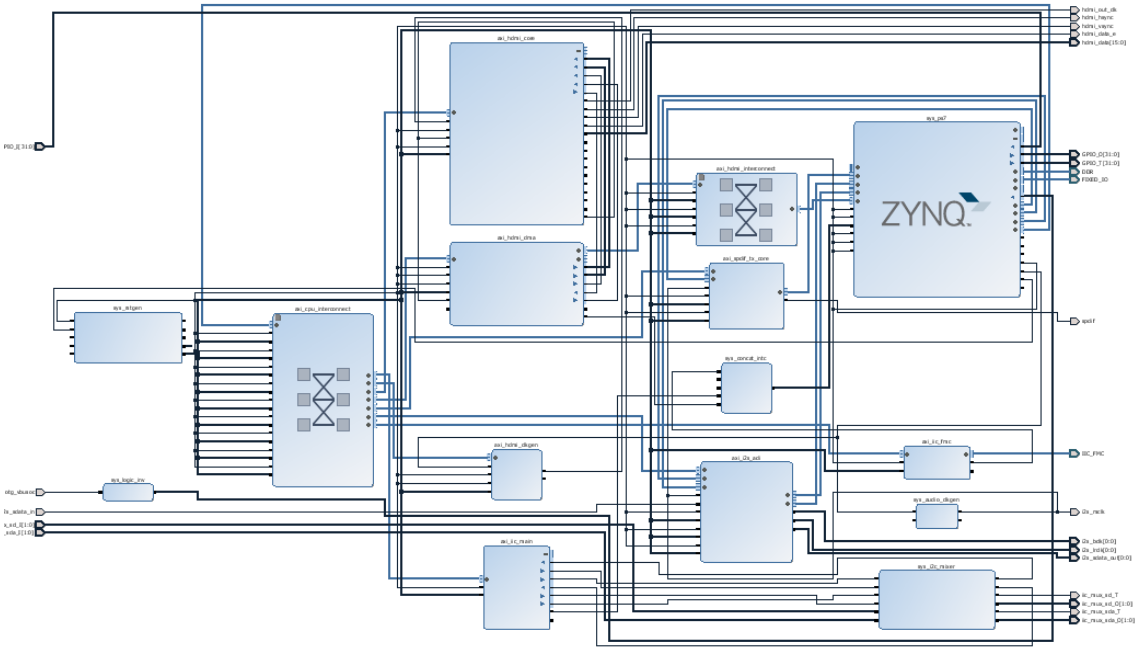


Figura 15 - Vista de los bloques IP de la plantilla de Xilinx en Vivado

Por otro lado están los módulos que han sido diseñados específicamente para este proyecto. En general cada bloque con una única función está dividido en dos partes: un módulo descrito en VHDL para manejar con total control el reloj y la salida y un módulo descrito con HLS en lenguaje C para comunicar el módulo anterior con el bus AXI de una forma sencilla. La ***Figura 15*** proporciona una vista de estos bloques y su interconexión.

Generador de señales PPM:

En primer lugar se encuentra el generador de señal que recibirá la placa de control de vuelo si la plataforma se encuentra en modo de vuelo autónomo.

Este módulo se encarga de generar una señal con las características similares a las que produce el receptor de radiofrecuencia GR-16, de forma que la placa de control de vuelo sea capaz de entender las instrucciones recibidas.

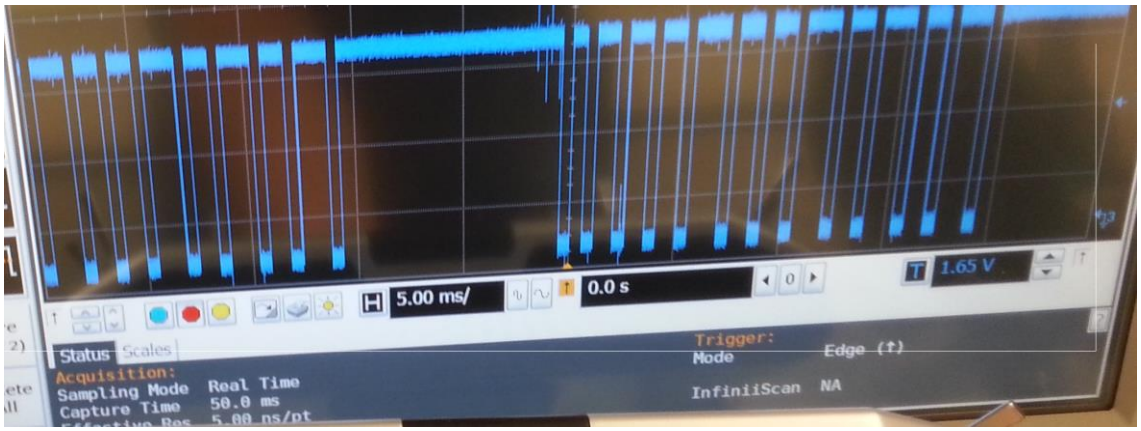


Figura 16 - Señal recibida del receptor GR-16

Esta señal está compuesta por una trama de 30 ms de duración compuesta por 13 pulsos con distinto ancho a valor '1' a 3,3 V. Cada pulso representa un canal y el ancho indica el valor recibido, así el primer pulso muestra el valor del parámetro "Gas", el segundo el valor del parámetro "Roll", el tercero el valor del parámetro "Pitch" y el cuarto el valor del parámetro "Yaw". Un ancho de pulso de aproximadamente 0.6 ms representa el valor mínimo para ese canal, mientras que el valor máximo corresponde a un ancho de 1.4 ms. El pulso 13 es de sincronismo y su ancho es el valor necesario para que la trama tenga una duración total de 30 ms. La ***Figura 16*** muestra la forma de este tipo de señal en el osciloscopio.

Para la realización de esta función se ha decidido que el módulo hardware que genere la señal reciba el valor del ancho de cada canal en forma de número de ciclos en lugar de en forma de tiempo para reducir la complejidad del hardware, hacer más fácil la depuración y aumentar las posibilidades de modificación del sistema, ya que la obtención del ancho de pulso en ciclos a partir del tiempo se realiza mediante software con una única multiplicación y no supone overhead para el procesador.

Para el diseño y descripción del bloque se ha utilizado la siguiente FSM[5] presentada en la ***Figura 17***:

[5] Máquina de Estados Finitos (Finite State Machine)

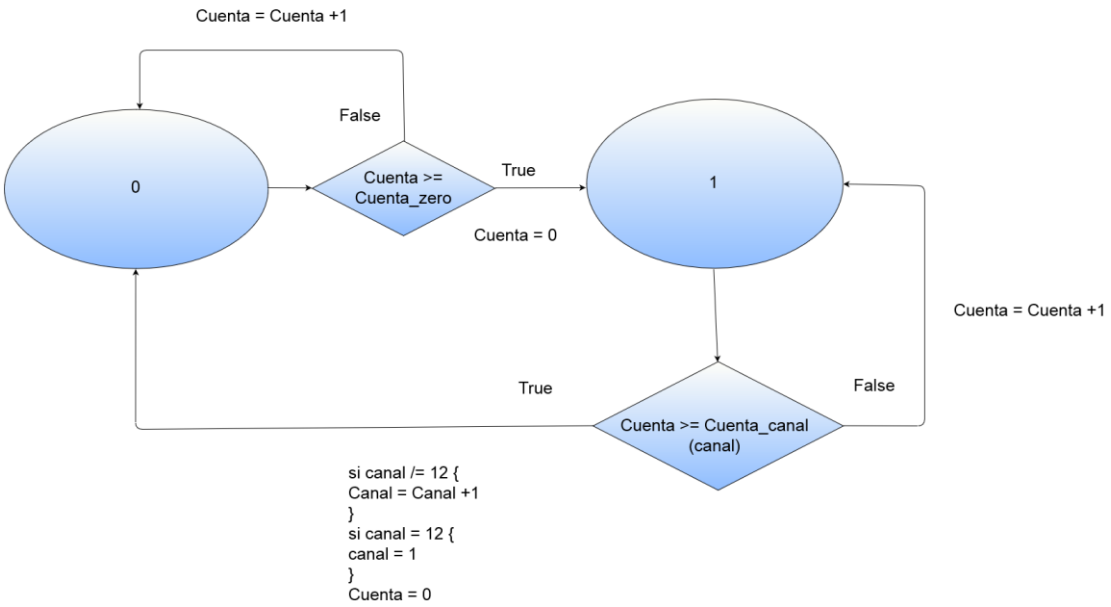


Figura 17 - Diagrama ASM del generador de señal

El módulo recibe como entradas los valores de los anchos de los 13 canales y el tiempo que la señal debe permanecer a '0' entre los canales.

A esta descripción se le debe añadir una señal "valid" añadida posteriormente para sincronizar a la vez todos los valores recibidos desde el software.

Una vez descrito el código VHDL del componente su funcionamiento es comprobado a través de un *testbench* con la herramienta ISim. A través de varias pruebas con distintos valores se certifica que el módulo genera la señal esperada, como puede observarse en la ***Figura 18***.

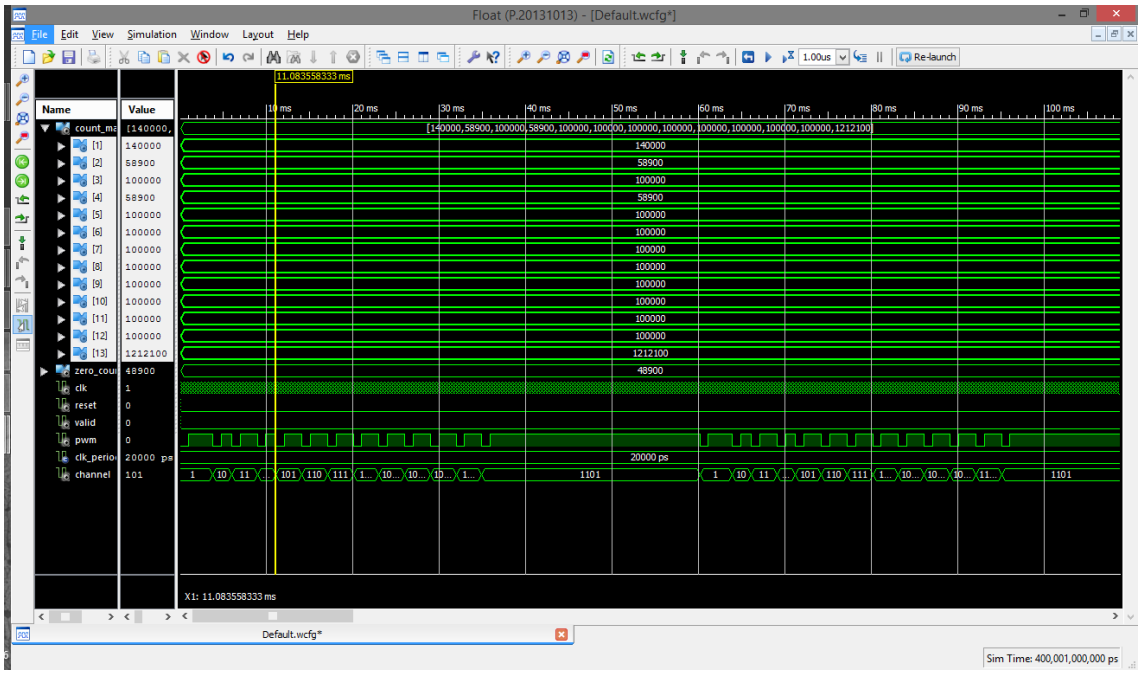


Figura 18 - Simulación sobre un testbench del funcionamiento del generador de señal

Cuando se comprueba que el hardware descrito funciona como se espera se procede a su empaquetado en un bloque IP, para realizar pruebas del funcionamiento del sistema completo con el módulo integrado en él.

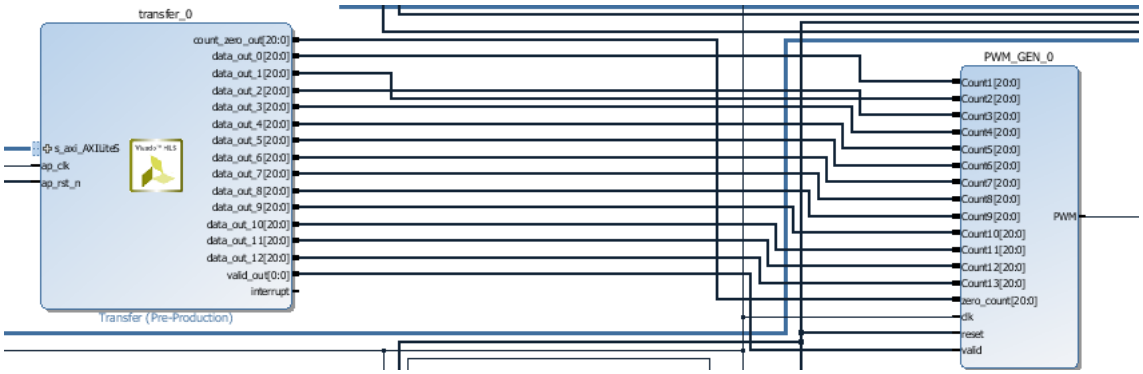


Figura 19 - Vista de los dos bloques IP que componen el generador de señal en Vivado

Junto a este bloque en el diseño se coloca un módulo descrito en HLS que carga las señales del bus AXI enviadas desde la parte software del sistema y las transfiere al bloque generador de señales, de forma que este puede leerlas cuando se sitúa la señal "valid" a '1'. La interconexión de los dos bloques se muestra en la ***Figura 19***.

En la ***Figura 20*** se muestra la señal generada por el generador de señal durante una prueba realizada.

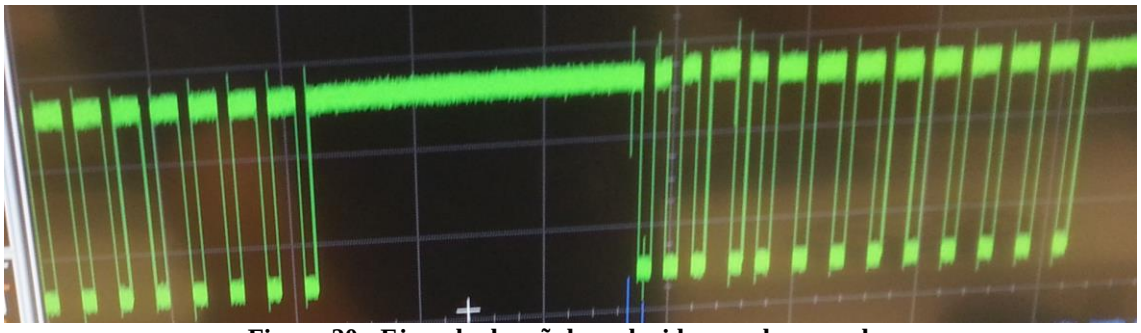


Figura 20 - Ejemplo de señal producida por el generador

Multiplexor de señales PPM:

El siguiente módulo diseñado para la plataforma es un bloque que permita enviar a la salida la señal generada por el vuelo programado o bien la señal recibida proveniente del mando en función del valor de uno de los interruptores del mando.

Para la realización de esta función se ha escogido el interruptor 9 del mando MC-20, destacado en la ***Figura 21***, que controla el valor del canal 8. Cuando el interruptor está en la posición superior la plataforma se encuentra en el modo controlado por mando y el valor del canal 8 es de 1 ms de ancho. Cuando el interruptor está en la posición inferior el sistema pasa al modo de vuelo autónomo y el valor del canal 8 es de 1.4 ms de ancho.

Figura 21 - Interruptor nº 9 para el control de modo de funcionamiento

Este módulo recibe como entradas el número del canal donde se determina el modo de funcionamiento, de forma que este podría configurarse mediante software en otro lugar del mando, el valor con el que se cambia el modo de funcionamiento, el tiempo a partir del cual considera que la emisora MC-20 está apagada y transmite la señal procedente del vuelo programado, y el ancho de pulso a partir del cual se considera que el canal que se ha recibido es de sincronismo (13), con un ancho mucho mayor que el del resto.

El motivo para que estos parámetros sean entradas configurables desde el software es facilitar la depuración y las pruebas sobre el sistema y permitir que la plataforma pueda seguir funcionando si se requiriesen cambios en la configuración del mando, así como optimizar los tiempos de respuesta al cambiar entre modos de funcionamiento. En la ***Figura 22*** se muestra el diagrama ASM en el que se basa el hardware diseñado.

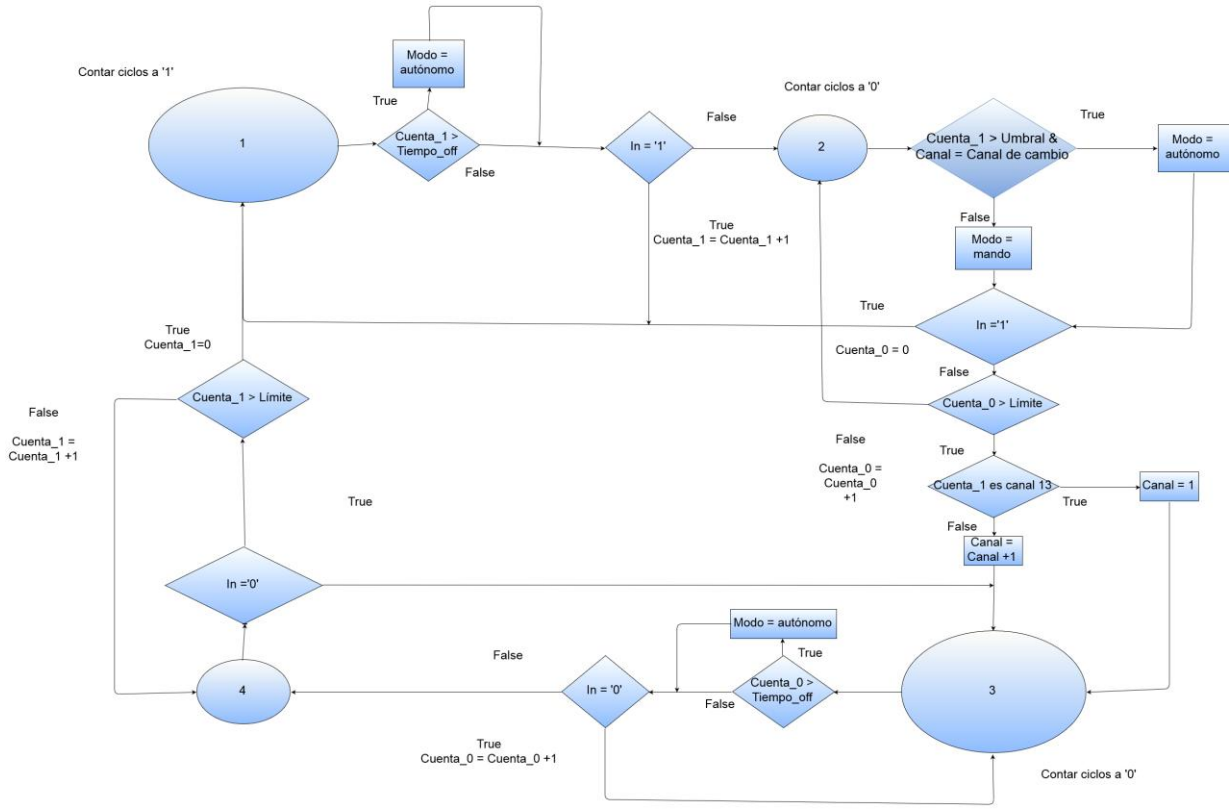


Figura 22 - Diagrama ASM del multiplexor de señales

El funcionamiento del módulo se basa en contar el ancho de pulso del canal que se está recibiendo y comprobar si es el canal del interruptor de cambio y si es así, su valor. En caso de que se supere el ancho de pulso especificado por el software se cambia a modo autónomo, si el canal es el de cambio pero el valor no supera el especificado se fija el modo con mando.

En la transición estado 2 al 3 se aumenta el contador de canal a no ser que se haya detectado un ancho que corresponda al canal 13, en cuyo caso se resetea la cuenta y se vuelve a empezar por el canal 1.

También se tiene en cuenta que el mando puede estar apagado, por ello se verifica en el estado 1 que si se supera el tiempo a partir del cual se haya definido mediante el software que el mando se considera apagado, se cambia a modo autónomo.

Los estados 2 y 4 se añadieron al observar que el módulo en la práctica presentaba un comportamiento erróneo en la cuenta, y que podía deberse a la presencia de ruido en la señal. Para evitar este mal funcionamiento sólo se considera una transición si el valor de la señal que ha cambiado se mantiene durante más de 0.2 ms, sino simplemente se ignora y se continúa con la cuenta. El resultado es que con esta corrección el periférico se comporta de forma adecuada.

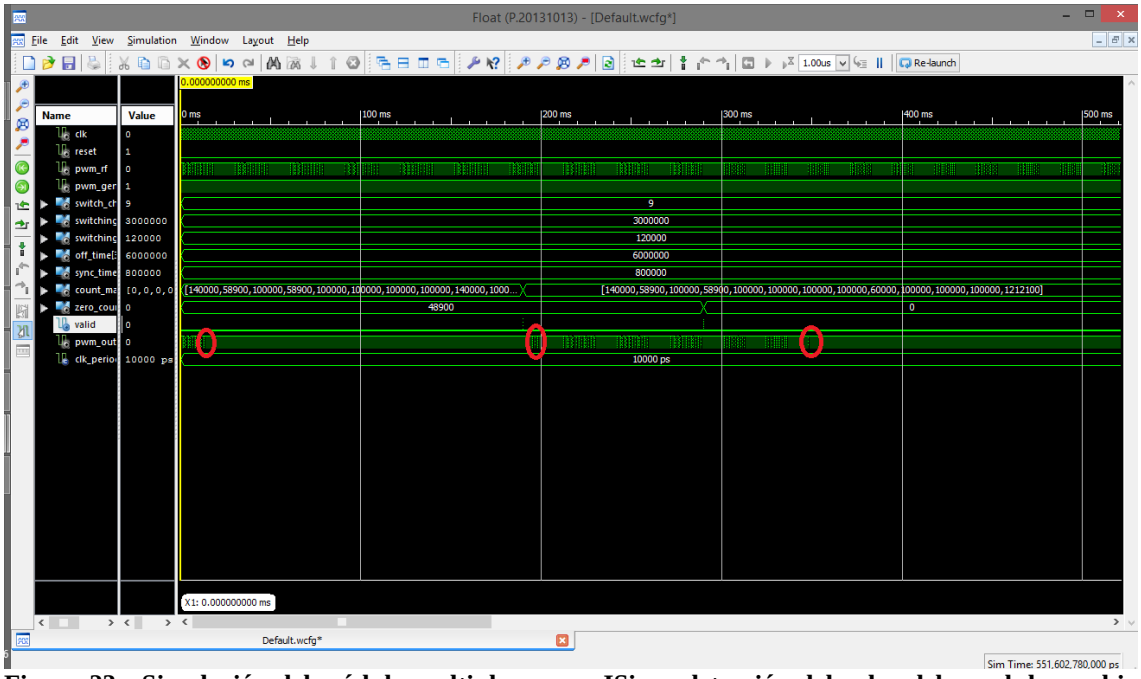


Figura 23 - Simulación del módulo multiplexor con ISim - detección del valor del canal de cambio de modo

En primer lugar se comprueba que el módulo separado del sistema es capaz de realiza las funciones deseadas. Para ello se ha empleado un *testbench* VHDL y la herramienta ISim, como puede verse en la ***Figura 23***.

Alterando el valor del canal de cambio de modo se comprueba que el hardware responde adecuadamente. Por último se simula un apagado del mando haciendo que la señal recibida sea prácticamente todo el tiempo '1' y se observa que el sistema pasa a enviar a la salida la señal generada internamente.

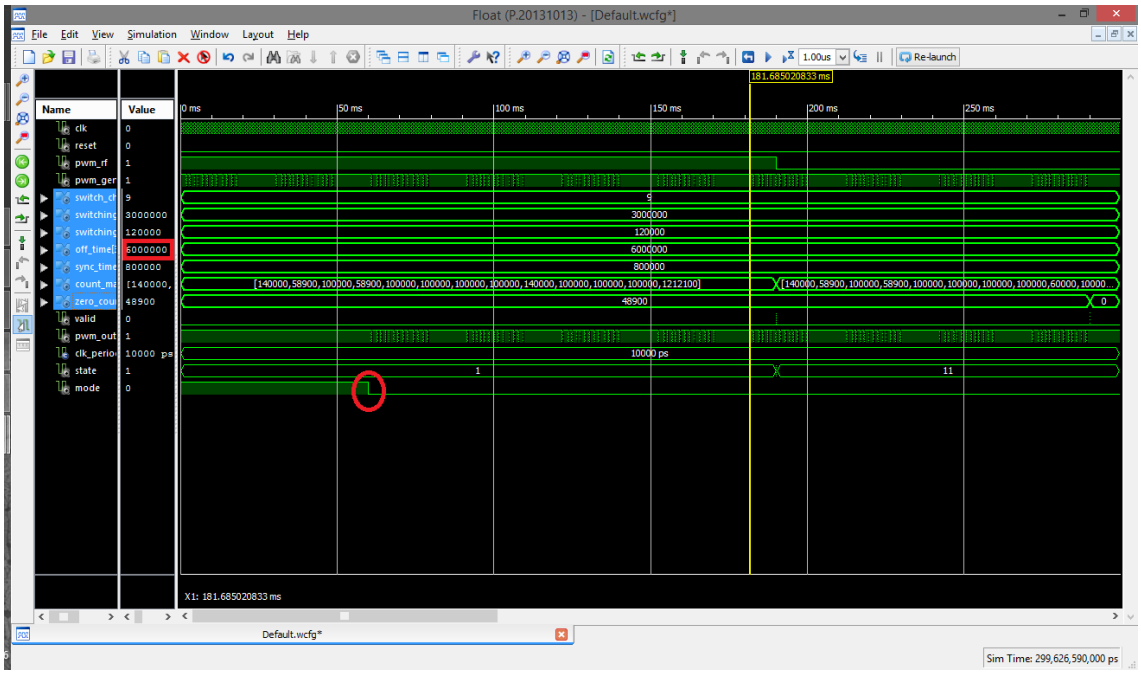


Figura 24 - Simulación del módulo multiplexor con ISim - detección de mando apagado

Por último, en la ***Figura 24***, se comprueba que el sistema detecta como mando apagado tanto una señal que permanece a '1' por un tiempo mayor que el especificado (60 ms en este caso) como una que permanece a '0' por ese mismo tiempo. Este último caso no corresponde a ningún estado contemplado del sistema, sin embargo se ha considerado para asegurar la fiabilidad del sistema.

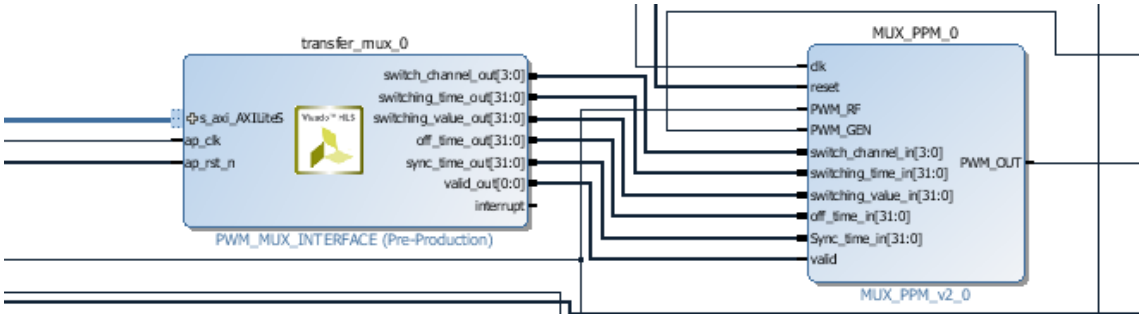


Figura 25 - IPs del multiplexor de señales en Vivado

De la misma forma que con el generador de señales, la comunicación con el software se realiza a través de un IP descrito mediante código C con HLS y que se encarga de transferir las señales del bus al módulo descrito en VHDL sin tener que manejar manualmente las señales del protocolo AXI, como se observa en la ***Figura 25***.

En la ***Figura 26*** se muestra como al funcionar en modo dirigido por mando (interruptor nº 9 hacia arriba) se puede observar que la señal de salida del sistema (en verde) coincide con la que se recibe del receptor GR-16 (azul).

Figura 26 - Sistema funcionando en modo conducido por mando

Al cambiar la posición del interruptor número 9 del mando se comprueba el ancho del canal 8 se sitúa en 1.4 ms y la señal de salida pasa a ser la generada internamente por el hardware diseñado, como se comprueba en la ***Figura 27***:

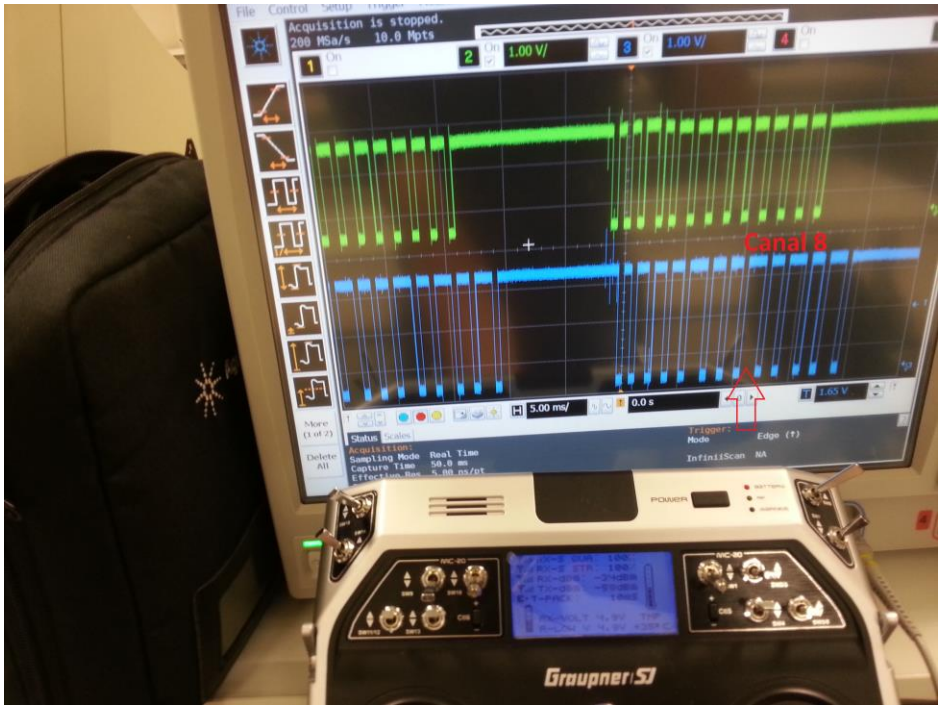


Figura 27 - Sistema funcionando en modo autónomo

Con el mando apagado el sistema envía a la salida la señal generada internamente por el hardware, como se puede ver en la ***Figura 28***:

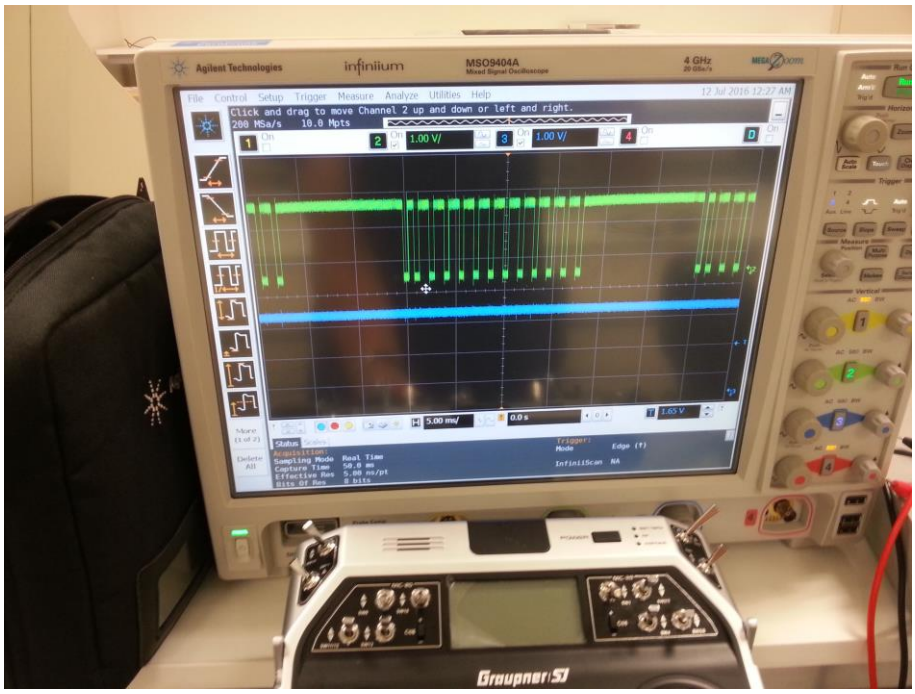


Figura 28 - Sistema con el mando apagado

Monitor de la señal recibida:

Este periférico fue diseñado con la intención de servir de medio para monitorizar la señal recibida en la ZedBoard desde el receptor GR-16 y depurar los posibles errores en el módulo multiplexor de señales.

No obstante este módulo también puede ser de gran utilidad para el programador, ya que puede servir para realizar funciones donde una rutina programada interactúe con un humano pilotando el cuadricóptero. Un ejemplo de aplicación donde este módulo hardware podría ser utilizado es el de un vuelo donde el piloto maneja el cuadricóptero pero el programa de vuelo se encarga de mantener el UAV en un determinado rango de alturas.

El esquema de funcionamiento del módulo, presentado en la ***Figura 29***, es similar al del módulo multiplexor, pero sin comprobar si el mando está encendido ni guardar el valor del registro que indica si se está operando en modo con mando o autónomo al llegar al canal que define el modo de vuelo.

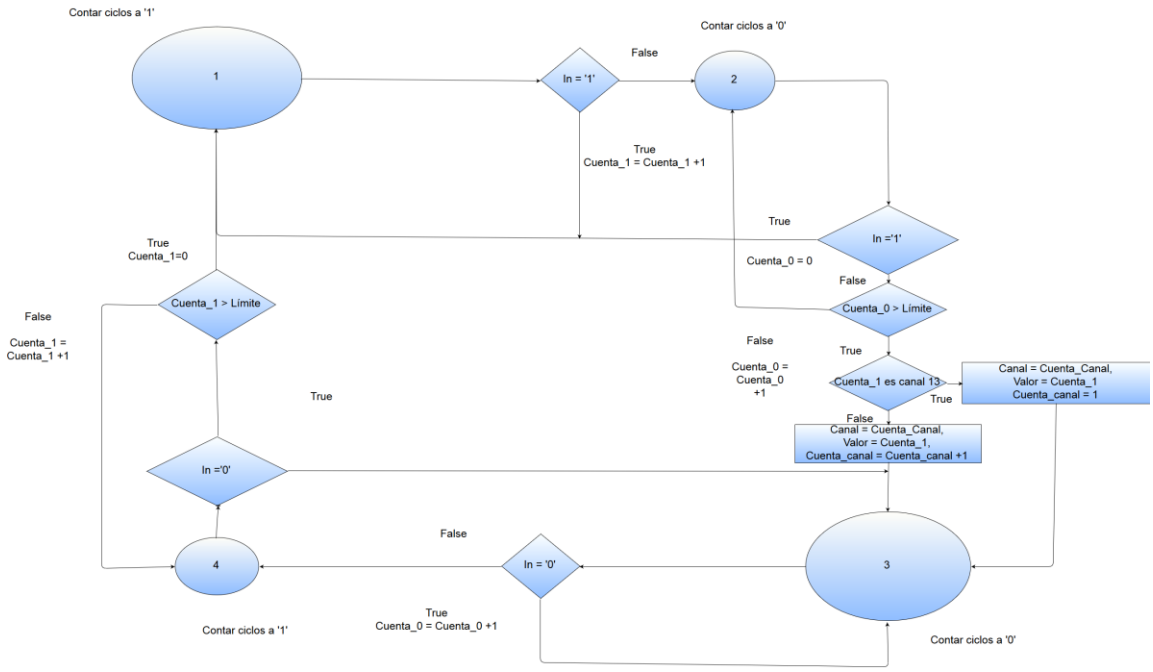


Figura 29 - Diagrama ASM del monitor de canales

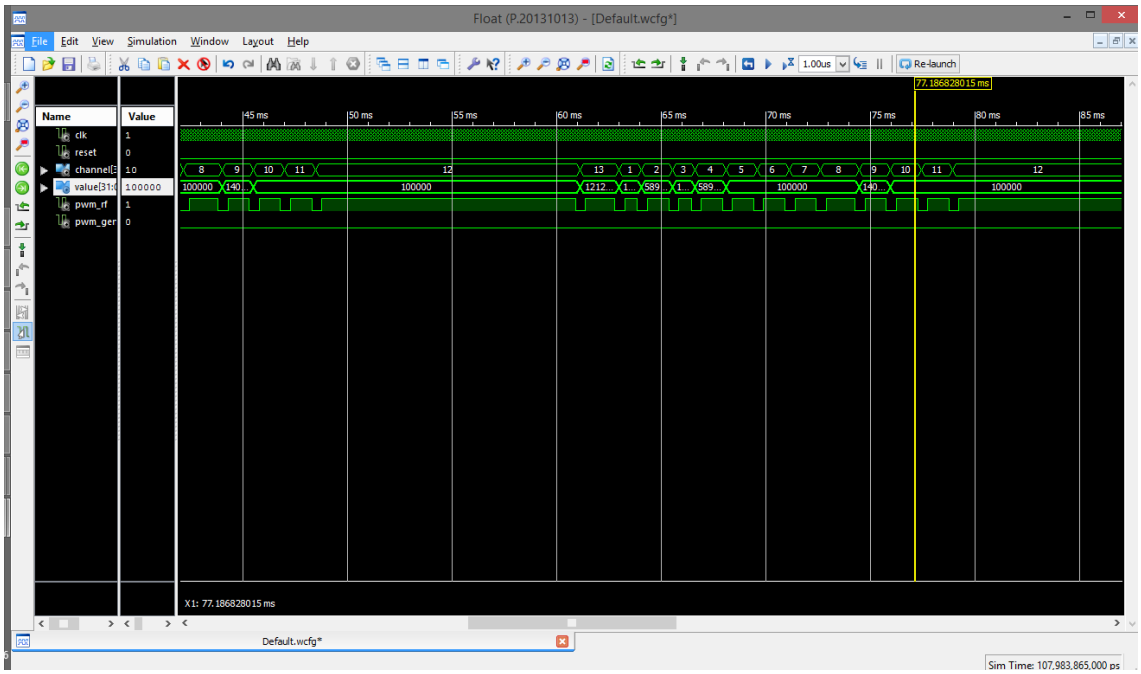


Figura 30 - Simulación del módulo monitor de canales con ISim

Se comprueba mediante un *testbench* en la herramienta ISim que el módulo proporciona el índice del último canal leído y el ancho de pulso recibido expresado en número de ciclos. La ***Figura 30*** muestra el resultado de la simulación.

Figura 31 - Vista de los bloques que componen el monitor de canales en Vivado

Se ha diseñado con HLS un módulo que sirve para guardar en un vector los valores leídos de los canales, de forma que utiliza la salida "channel" del periférico VHDL como puntero y almacena el valor "Value". La interconexión de los bloques se muestra en la ***Figura 31***.

Lector de distancia por ultrasonidos:

El principal medio con el que cuenta el UAV para conocer el entorno son 5 sensores de ultrasonidos, apuntando a izquierda, delante, derecha, detrás y abajo respectivamente. Los sensores empleados son HC SR04 y MaxSonar Ez1 respectivamente.

Contar con este módulo implementado en hardware permite realizar lecturas continuas en todos los sensores sin bloquear el microprocesador, de forma que desde el software sólo sea necesaria una lectura del registro del bus donde queda almacenado el ancho en número de ciclos del último pulso enviado por el sensor de ultrasonidos correspondiente.

El sensor HC SR04 funciona empleando dos señales, una de entrada y otra de salida, sin contar con la tierra y la alimentación de 5V. La primera denominada "Trigger" sirve para solicitar al sensor una medida cuando se mantiene en estado alto durante 10 µs. El valor de la medida se obtiene a través del ancho de pulso de la señal "Echo", la salida del sensor. Este ancho de pulso indica el tiempo que ha tardado la ráfaga de ultrasonidos enviada por el sensor en alcanzar un obstáculo y ser recibida por el sensor. La ***Figura 32*** muestra el funcionamiento del sensor a nivel temporal.

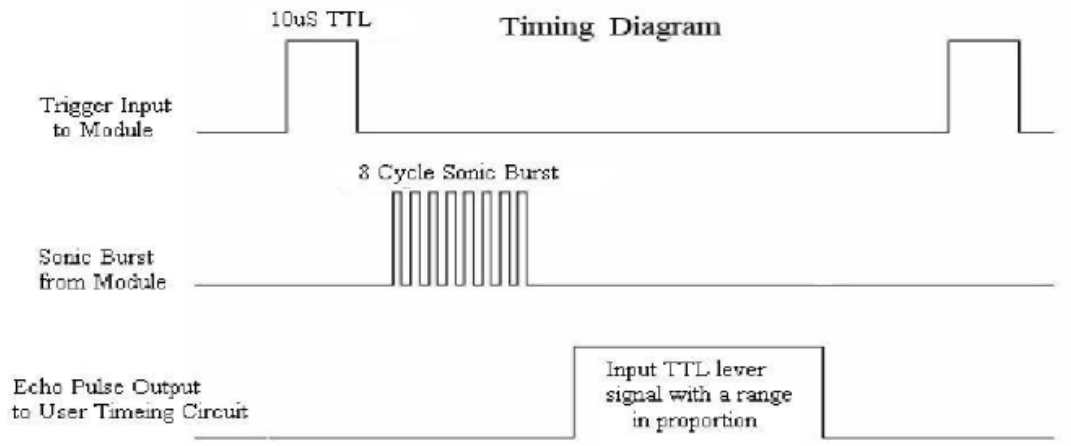


Figura 32 - Diagrama temporal del HC-SR4[viii]

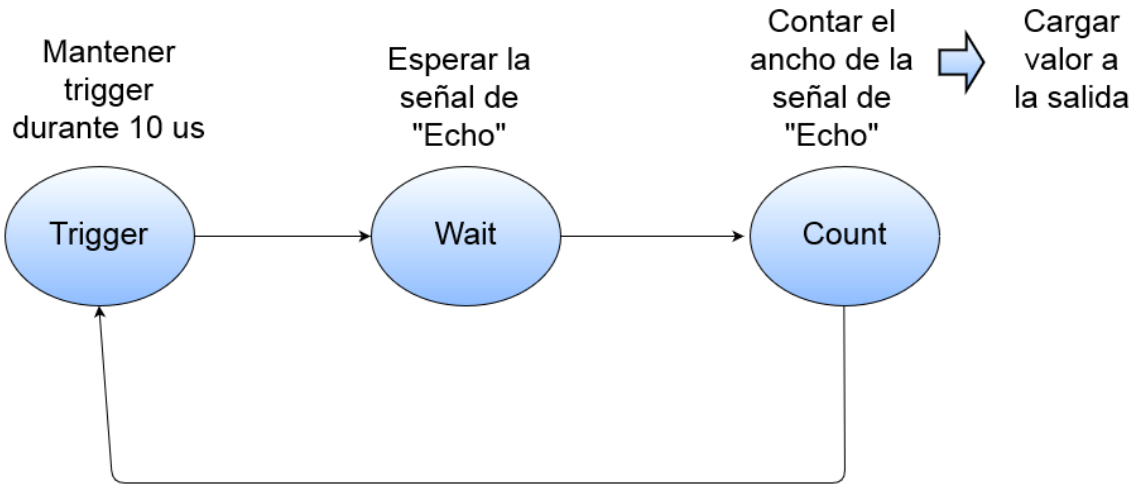


Figura 33 - Diagrama de funcionamiento del periférico lector de ultrasonidos

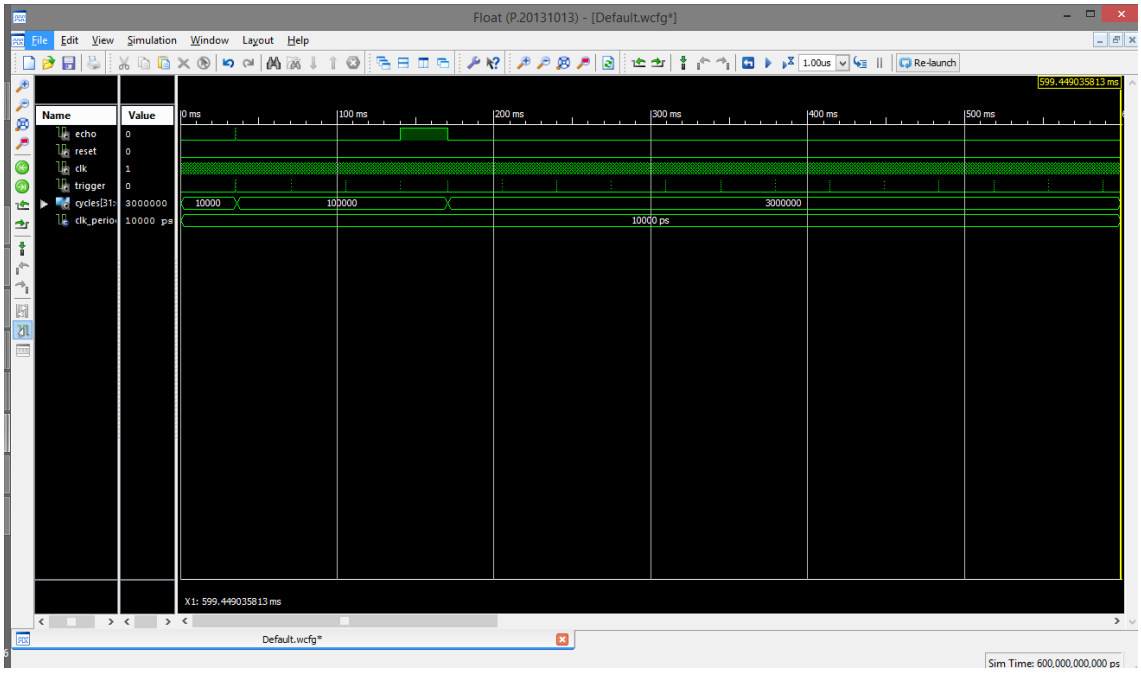


Figura 34 - Simulación del módulo lector de sensores de ultrasonidos con ISim

En la ***Figura 34*** se comprueba mediante un *testbench* en la herramienta ISim que el hardware es capaz de medir el ancho de los pulsos de la señal "Echo" enviada por el sensor y que en caso de no recibir respuesta es capaz de detectar el *timeout* y enviar otra señal de trigger.

La interconexión de los bloques IP para la lectura de ultrasonidos se muestra en la ***Figura 35***.

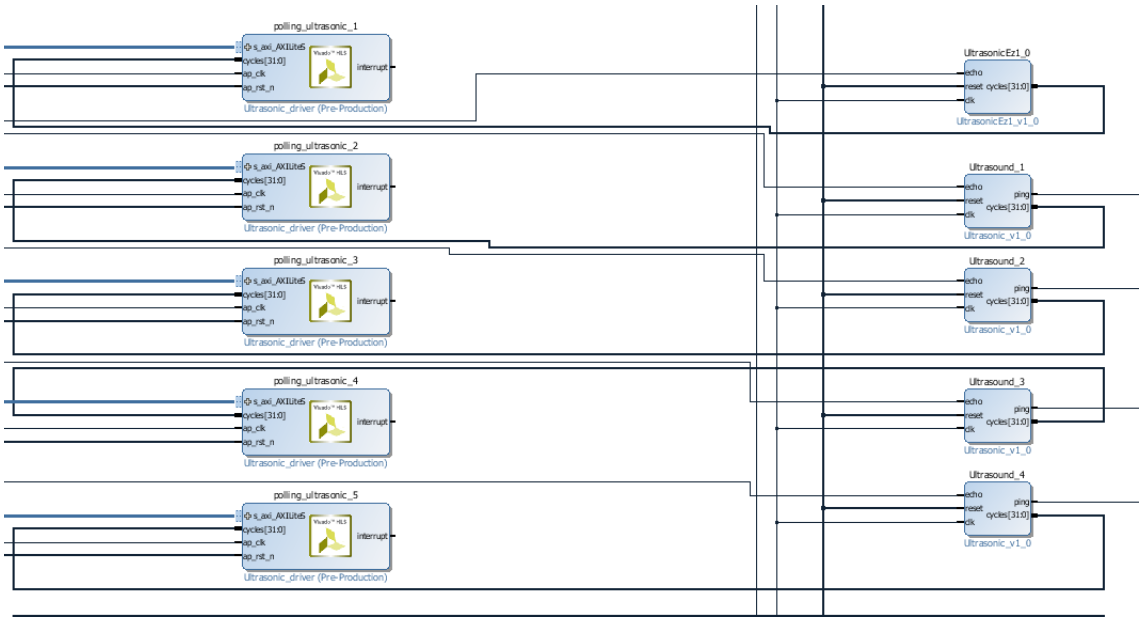


Figura 35 - Vista en Vivado de los bloques de ultrasonidos

Para la lectura del sensor MaxSonar Ez1 se parte del periférico diseñado para los HC-SR04 y se elimina el estado de "Trigger", ya que este sensor no precisa de esa señal para su funcionamiento, puede funcionar disparándose automáticamente.

Para obtener la distancia del obstáculo detectado debe aplicarse la siguiente fórmula: ancho de pulso en microsegundos dividido entre 58 para obtener centímetros o entre 148 para obtener pulgadas. Este cálculo por comodidad se realiza en la API software, ya que implica el uso de flotantes y no presenta ventajas de rendimiento en hardware frente a la manejabilidad del software. También debe comprobarse mediante software que la medida es válida, y solo deberían tenerse en cuenta los datos obtenidos que estén dentro del intervalo 10-300 cm.

En las ***Figuras 36 y 37*** se muestra las señales de disparo generadas por el hardware diseñado junto a las de los ecos de respuesta de los sensores vistas en el osciloscopio.

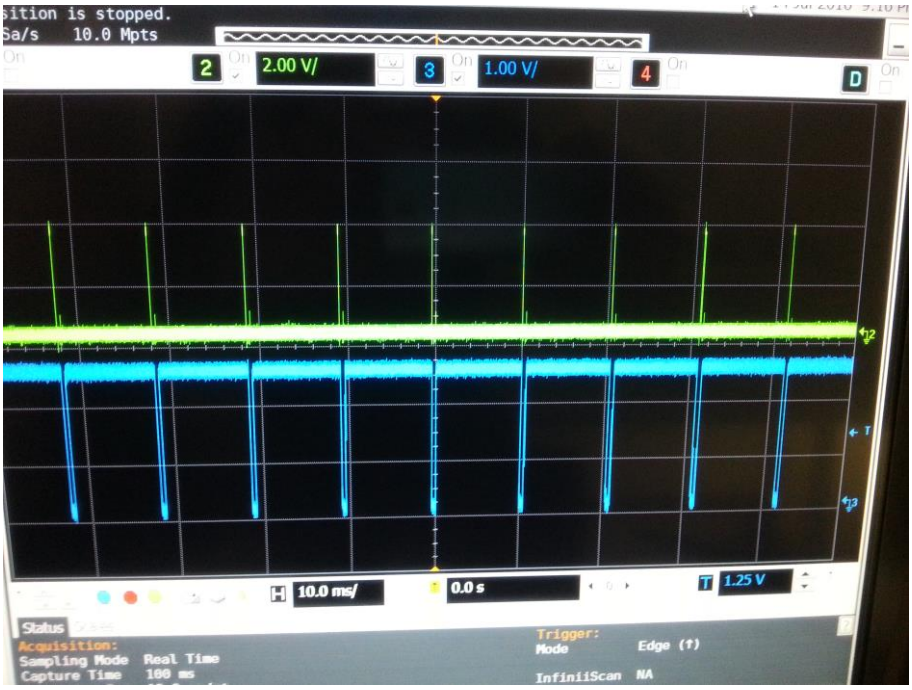


Figura 36 - Prueba sobre el osciloscopio del sensor de ultrasonidos midiendo una distancia de alrededor de 200 cm. Señal de "Trigger" en verde y "Echo" en azul.

Figura 37 - Prueba sobre el osciloscopio del sensor de ultrasonidos midiendo una distancia de alrededor de 20 cm. Señal de "Trigger" en verde y "Echo" en azul.

Convertidor DC-DC para la alimentación del sistema electrónico:

El sistema de alimentación de la electrónica añadida está basado en un convertidor DC-DC de tipo *Buck*, que permite obtener la tensión requerida por la ZedBoard, de 12V, a la salida a partir de la tensión que proporciona la batería del cuadricóptero, de 14.8 V.

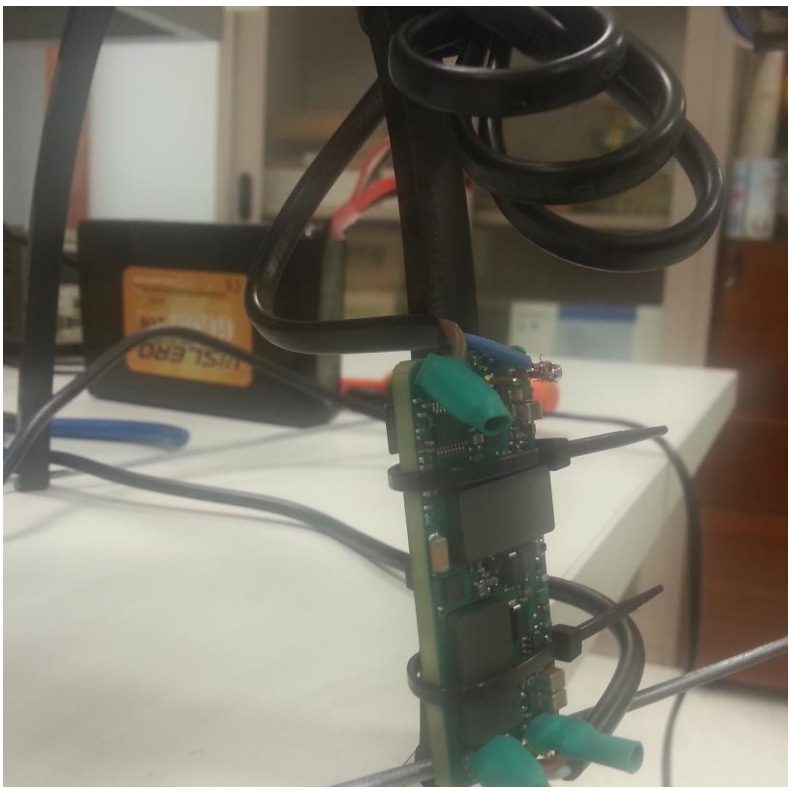


Figura 38 - Fotografía del montaje del convertidor DC-DC

Para el montaje del sistema, presentado en la ***Figura 38***, se ha empleado un conector de tipo barril de 2.5 mm de diámetro interior y 5.5 mm de diámetro exterior con la toma positiva en el interior conectado a la toma de alimentación de la ZedBoard. Este conector está soldado a través de un cable a la salida del convertidor DC-DC, a los pines 4, que está cortocircuitado con el pin 5 en el caso del terminal negativo y al 7, que está cortocircuitado al pin 8 en el caso del positivo, siguiendo el esquema presentado en la ***Figura 39***.

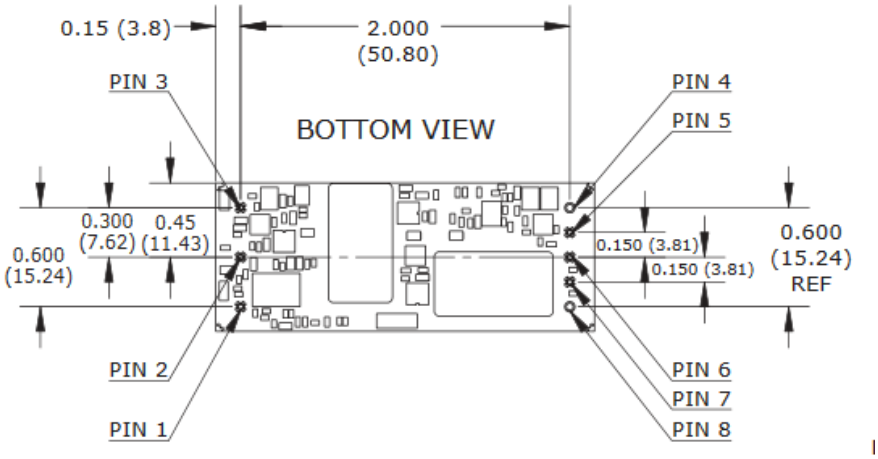


Figura 39 - Plano de la parte inferior del convertidor DC-DC[ix]

En el otro lado, a la entrada del convertidor DC-DC se encuentra la tensión de 14.8 V proporcionada por la batería LiPo del vehículo, que se extrae de dos tomas disponibles en la placa Bl-Ctrl. En este caso la toma positiva está soldada al pin 3, mientras que la toma negativa está conectada al pin 1, como se muestra en la fotografía de la ***Figura 40***.

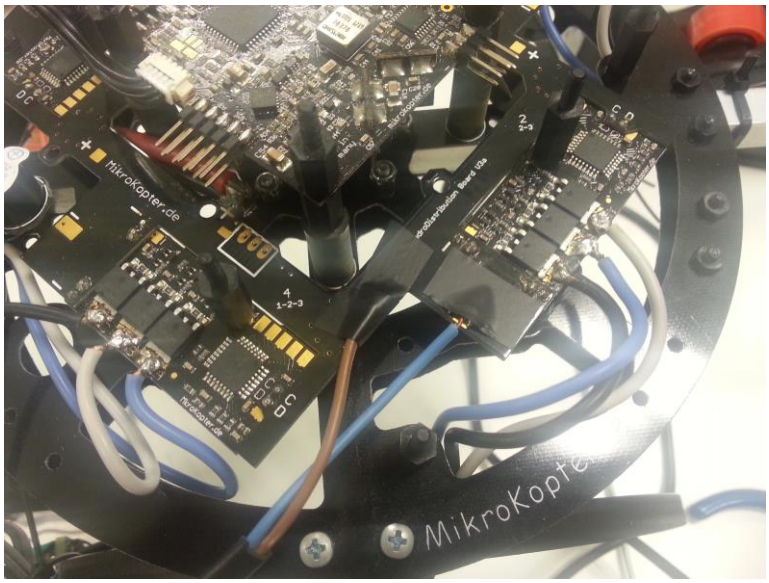


Figura 40 - Fotografía de la toma de alimentación a 14.8 V

Parte Software

La parte software del sistema está soportada sobre un sistema Linaro Linux, funcionando sobre los dos cores ARM Cortex A9 que incorpora el chip Zynq.

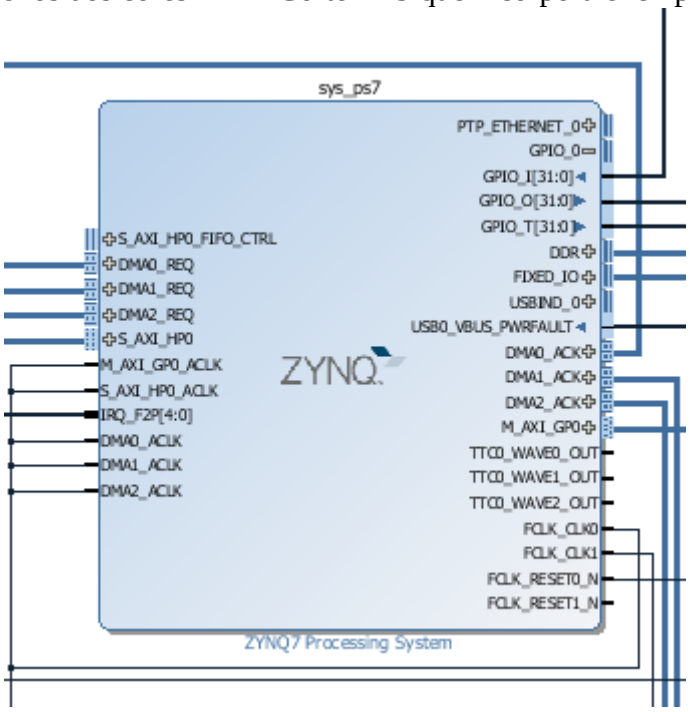


Figura 41 - Módulo IP del microprocesador del sistema

Este módulo, cuyo bloque IP está representado en la ***Figura 41***, es un microprocesador con dos núcleos funcionando a una frecuencia de 667 MHz sobre una tecnología de transistor de 28 nm que cuenta 32 KB de cache de nivel 1, 512 KB de cache de nivel 2, 512 KB de RAM on-chip y que tiene acceso a un módulo de memoria RAM de 512 MB para su funcionamiento[x].

El sistema funciona con una tarjeta SD como soporte de almacenamiento y dispositivo de arranque.

Esta tarjeta SD debe contener dos particiones, una de tipo FAT para que el sistema de arranque la reconozca y arranque el FSBL[6], el ejecutable que se encarga de volcar los archivos necesarios para el arranque del sistema Linux.

El FSBL es empaquetado junto con el fichero de configuración de la lógica programable (.bit) y el archivo de boot (*u-boot.elf*) generado a partir de las herramientas proporcionadas por Xilinx en un fichero llamado *BOOT.BIN* que se encarga de copiar en memoria el kernel de Linux (*uImage* y *Devicetree*) y programar la FPGA.

Por otro lado se encuentra el sistema de archivos de Linux, en una partición de tipo EXT4, donde se encuentran los ficheros del sistema y donde el usuario puede realizar modificaciones. La base del sistema se ha obtenido del repositorio de Linaro y se ha extraído en esta partición.

Sobre esta base se han instalado las herramientas necesarias para el trabajo de la plataforma, tales como drivers y herramientas de captura de webcam, herramientas para el trabajo con buses I2C y la modificación del fichero de tareas programadas, llamado c*rontab* para ejecutar el programa de vuelo cada vez que se inicie el sistema.

Desde este sistema Linux se deberá controlar el conjunto de periféricos hardware descritos en el apartado anterior y a su vez tomar información de los mismos a través del bus AXI.

Mapa y descripción de interfaces de los periféricos del bus

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
axi_hdmi_dma					
Data_MM2S (32 address bits : 4G)					
sys_ps7	S_AXI_HP0	HP0_DDR_L...	0x0000_0000	512M	0x1FFF_FFFF
sys_ps7					
Data (32 address bits : 0x40000000 [1G])					
axi_hdmi_clkgen	s_axi	axi_lite	0x7900_0000	64K	0x7900_FFFF
axi_hdmi_core	s_axi	axi_lite	0x70E0_0000	64K	0x70E0_FFFF
axi_spdif_tx_core	S_AXI	reg0	0x75C0_0000	64K	0x75C0_FFFF
axi_i2s_adi	S_AXI	reg0	0x7760_0000	64K	0x7760_FFFF
axi_hdmi_dma	S_AXI_LITE	Reg	0x4300_0000	64K	0x4300_FFFF
axi_iic_fmc	S_AXI	Reg	0x4162_0000	64K	0x4162_FFFF
axi_iic_main	S_AXI	Reg	0x4160_0000	64K	0x4160_FFFF
transfer_0	s_axi_AXILiteS	Reg	0x43C0_0000	64K	0x43C0_FFFF
transfer_mux_0	s_axi_AXILiteS	Reg	0x43C1_0000	64K	0x43C1_FFFF
polling_ultrasonic_0	s_axi_AXILiteS	Reg	0x43C2_0000	64K	0x43C2_FFFF
polling_ultrasonic_1	s_axi_AXILiteS	Reg	0x43C3_0000	64K	0x43C3_FFFF
polling_ultrasonic_2	s_axi_AXILiteS	Reg	0x43C4_0000	64K	0x43C4_FFFF
polling_ultrasonic_3	s_axi_AXILiteS	Reg	0x43C5_0000	64K	0x43C5_FFFF
polling_ultrasonic_4	s_axi_AXILiteS	Reg	0x43C6_0000	64K	0x43C6_FFFF
polling_ultrasonic_5	s_axi_AXILiteS	Reg	0x43C7_0000	64K	0x43C7_FFFF
channel_monitor_interface_0	s_axi_AXILiteS	Reg	0x43C8_0000	64K	0x43C8_FFFF

Figura 42 - Listado de periféricos conectados al bus y sus direcciones físicas

La ***Figura 42*** muestra las direcciones físicas de memoria donde se encuentran los periféricos son proporcionadas por la herramienta Vivado cuando se realiza la interconexión de los bloques IP correspondientes.

[6] First Stage Boot Loader

A continuación se describe el funcionamiento de la interfaz de los bloques IPs diseñados con HLS para este proyecto, vistos desde la parte software:

Generador PPM

Offset	Nombre
0x00	Señales de Control

31	8	7	6	5	4	3	2	1	0

bit 0 - ap_start (Read/Write/SC)
bit 1 - ap_done (Read/COR)
bit 2 - ap_idle (Read)
bit 3 - ap_ready (Read)
bit 7 - auto_restart (Read/Write)
otros - reservado

Offset	Nombre
0x04	Global Interrupt Enable Register

31	1	0

bit 0 - Global Interrupt Enable (Read/Write)
otros - reservado

Offset	Nombre
0x08	IP Interrupt Enable Register (Read/Write)

31	1	0

bit 0 - Channel 0 (ap_done)
otros - reservado

Offset	Nombre
0x0c	IP Interrupt Status Register (Read/TOW)

31	1	0

bit 0 - Channel 0 (ap_done)
otros – reservado

Offset	Nombre
0x10	Data signal of data_in_0
0x18	Data signal of data_in_1
0x20	Data signal of data_in_2
0x28	Data signal of data_in_3
0x30	Data signal of data_in_4
0x38	Data signal of data_in_5
0x40	Data signal of data_in_6
0x48	Data signal of data_in_7
0x50	Data signal of data_in_8
0x58	Data signal of data_in_9
0x60	Data signal of data_in_10
0x68	Data signal of data_in_11
0x70	Data signal of data_in_12
0x78	Data signal of count_zero_in

31 21	20 0

bit 20~0 - data[20:0] (Read/Write)
otros - reservado

Offset	Nombre
0x80	Data signal of valid_in

31 1	0

bit 0 - valid[0] (Read/Write)
otros – reservado

Multiplexor de señales PPM:

Offset	Nombre
0x00	Señales de Control

31 8	7	6	5	4	3	2	1	0

bit 0 - ap_start (Read/Write/SC)

bit 1 - ap_done (Read/COR)
bit 2 - ap_idle (Read)
bit 3 - ap_ready (Read)
bit 7 - auto_restart (Read/Write)
otros - reservado

Offset	Nombre
0x04	Global Interrupt Enable Register

31	1	0

bit 0 - Global Interrupt Enable (Read/Write)
otros - reservado

Offset	Nombre
0x08	IP Interrupt Enable Register (Read/Write)

31	1	0

bit 0 - Channel 0 (ap_done)
otros - reservado

Offset	Nombre
0x0c	IP Interrupt Status Register (Read/TOW)

31	1	0

bit 0 - Channel 0 (ap_done)
otros – reservado

Offset	Nombre
0x10	Data signal of switch_channel

31	4	3	2	1	0

bit 3~0 - switch_channel[3:0] (Read/Write)
otros - reservado

Offset	Nombre
0x18	Data signal of switching_time

31	0

bit 31~0 - switching_time[31:0] (Read/Write)

Offset	Nombre
0x20	Data signal of switching_time

31	0

bit 31~0 - switching_value[31:0] (Read/Write)

Offset	Nombre
0x28	Data signal of switching_time

31	0

bit 31~0 - off_time[31:0] (Read/Write)

Offset	Nombre
0x30	Data signal of switching_time

31	0

bit 31~0 - sync_time[31:0] (Read/Write)

Offset	Nombre
0x38	Data signal of switching_time

31	1	0

bit 0 - valid[0] (Read/Write)
otros - reservado

Monitor de la señal recibida:

Offset	Nombre
0x00	Señales de Control

31	8	7	6	5	4	3	2	1	0

bit 0 - ap_start (Read/Write/SC)
bit 1 - ap_done (Read/COR)
bit 2 - ap_idle (Read)
bit 3 - ap_ready (Read)
bit 7 - auto_restart (Read/Write)
otros - reservado

Offset	Nombre
0x04	Global Interrupt Enable Register

31	1	0

bit 0 - Global Interrupt Enable (Read/Write)
otros - reservado

Offset	Nombre
0x08	IP Interrupt Enable Register (Read/Write)

31	1	0

bit 0 - Channel 0 (ap_done)
otros - reservado

Offset	Nombre
0x0c	IP Interrupt Status Register (Read/TOW)

31	1	0

bit 0 - Channel 0 (ap_done)
otros – reservado

Offset	Nombre
0x80	Data signal of switch_channel

31	4	3	2	1	0

bit 3~0 - channel[3:0] (Read/Write)
otros – reservado

Offset	Nombre
0x40 ~ 0x7f	Memory value[n]

31	0

Palabra n : bit [31:0] - value[n]

Interfaz de programación de aplicaciones

Sobre las direcciones del bus con los registros descritos anteriormente se ha programado una interfaz para permitir al programador acceder al hardware mediante código con un nivel de abstracción mayor.

Las funciones que permiten al programador acceder al hardware sin tener que trabajar en bajo nivel se incluyen en el fichero *functions.cc* y la cabecera *functions.h*.

Para comunicarse con el hardware no es posible emplear las direcciones físicas del bus, ya que el sistema corre un sistema operativo que sólo permite trabajar con direcciones virtuales, las direcciones físicas son protegidas por la MMU y no son accesibles por el programador directamente. Por tanto es necesario obtener una forma de acceder a la dirección física deseada a través de una dirección virtual que apunte a esta dirección física.

El método más cómodo para realizar estar función es el uso de la función de sistema *mmap* que permite obtener un puntero a una dirección virtual que está mapeada en la dirección física especificada. En la función de inicialización del hardware del cuadricóptero se realiza una llamada a *mmap* para obtener direcciones con las que controlar cada periférico, a través de variables globales incluidas en la cabecera .h o bien en el retorno de función en el caso del periférico principal, el generador de señales PPM.

El conjunto de funciones de las que dispone el programador es el siguiente:

Nombre	**Argumentos**	**Retorna**	**Función**
quadcopter_init	-	Puntero del generador de señal	Inicialización de todos los periféricos necesarios para el funcionamiento del sistema, obtención de los punteros a los periféricos, calibración automática del cuadricóptero y arranque de los motores.
read_distance	Puntero al periférico del sensor	Distancia detectada en cm	Retorna la última distancia medida por el sensor escogido.
shutdown	Puntero al generador de señales	-	Genera una señal que es interpretada por la placa de control de vuelo como la recibida por el receptor cuando el mando está apagado.
stop	Puntero al generador de señales	-	Genera una señal que produce el apagado de los motores.
start_engines	Puntero al generador de señales	-	Genera una señal que produce el encendido de los motores.
check_values	Puntero al primer dato del generador de señales (offset 0x10)	-	Función de depuración que imprime los valores que está empleando el generador de señales para generar la señal.
check_values_base	Puntero al generador de señales	-	Función de depuración que imprime los valores que está empleando el generador de señales para generar la señal.

check_value	Puntero al primer dato del generador de señales (offset 0x10)	-	Función de depuración que imprime el valor que está generando el generador de señales en el canal index+1.
set_gas	Puntero a la dirección base del generador de señal Valor entero de 0 a 100	-	Permite controlar el valor del parámetro "gas" escribiendo en el canal correspondiente del generador de señal. También actualiza el ancho del pulso de sincronismo para mantener la duración de la trama. El programador debe proporcionar un valor entero del parámetro con un rango de 0 a 100.
set_pitch	Puntero a la dirección base del generador de señal Valor entero de -100 a 100	-	Permite controlar el valor del parámetro "gas" escribiendo en el canal correspondiente del generador de señal. También actualiza el ancho del pulso de sincronismo para mantener la duración de la trama. El programador debe proporcionar un valor entero del parámetro con un rango de -100 a 100.
set_roll	Puntero a la dirección base del generador de señal Valor entero de -100 a 100	-	Permite controlar el valor del parámetro "roll" escribiendo en el canal correspondiente del generador de señal. También actualiza el ancho del pulso de sincronismo para mantener la duración de la trama. El programador debe proporcionar un valor entero del parámetro con un rango de -100 a 100.
set_yaw	Puntero a la dirección base del generador de señal Valor entero de -100 a 100	-	Permite controlar el valor del parámetro "yaw" escribiendo en el canal correspondiente del generador de señal. También actualiza el ancho del pulso de sincronismo para mantener la duración de la trama. El programador debe proporcionar un valor entero del parámetro con un rango de -100 a 100.

read_gas	Puntero a la dirección base del monitor de canales	Valor del parámetro "gas" recibido desde el mando	Permite obtener una medida del valor del parámetro "gas" recibido desde el mando. El dato se recibe como un entero con rango de 0 a 100.
read_pitch	Puntero a la dirección base del monitor de canales	Valor del parámetro "pitch" recibido desde el mando	Permite obtener una medida del valor del parámetro "pitch" recibido desde el mando. El dato se recibe como un entero con rango de -100 a 100.
read_roll	Puntero a la dirección base del monitor de canales	Valor del parámetro "roll" recibido desde el mando	Permite obtener una medida del valor del parámetro "roll" recibido desde el mando. El dato se recibe como un entero con rango de -100 a 100.
read_yaw	Puntero a la dirección base del monitor de canales	Valor del parámetro "yaw" recibido desde el mando	Permite obtener una medida del valor del parámetro "yaw" recibido desde el mando. El dato se recibe como un entero con rango de -100 a 100.

Utilización del sistema:

Este capítulo se dedicará a describir cómo trabajar con la plataforma desarrollada, fundamentalmente modificar y trabajar con la plataforma hardware y el diseño de aplicaciones software y ajustes del sistema operativo.

En general la forma más sencilla de generar rutinas de vuelo es mediante el desarrollo de aplicaciones software que hagan uso de la API desarrollada para la comunicación con el hardware, si bien es posible trabajar directamente sobre la interfaz de registros presente en el bus o desarrollar nuevo hardware en caso de que fuese necesario para la aplicación concreta, por ejemplo en aplicaciones de vídeo donde el rendimiento que proporcionen los cores Cortex A9 no sea suficiente.

Modificación de la plataforma hardware:

Es posible la incorporación de nuevos bloques a la plataforma hardware, así como la modificación de los presentes, a través de la herramienta Vivado. Para ello deben obtenerse bloques IP compatibles con Vivado a partir del código HDL que describe los componentes.

Este proceso se puede realizar de forma sencilla desde la propia herramienta, que permite empaquetar los proyectos realizados en forma de IP de forma rápida y sencilla. Una de las limitaciones del empaquetador de IPs de Vivado es que no permite generar IPs con entradas definidas como arrays de buses o vectores, es decir arrays de líneas con varios bits de datos. La forma de evitar esta limitación es definir para cada componente del array un puerto separado.

Otra opción es obtener módulos IPs desde diseños generados con la herramienta de Xilinx HLS, que permite generar descripciones hardware a partir de código C y C++. Esta herramienta proporciona facilidades y automatizaciones en el diseño que son de utilidad, por ejemplo la gestión de un puerto del módulo a través del protocolo AXI simplemente con añadir una directiva que lo especifique.

Este diseño en concreto cuenta con dos niveles de bloques o wrappers HDL, como se puede observar en la ***Figura 43***, de forma que aunque la herramienta actualice de forma automática el fichero donde se describen las interconexiones del sistema en caso de querer añadir señales que estén mapeadas en pines de la FPGA, será necesario realizar el mapeo en la entidades superior, descrita en Verilog, llamada *system_top.v*.

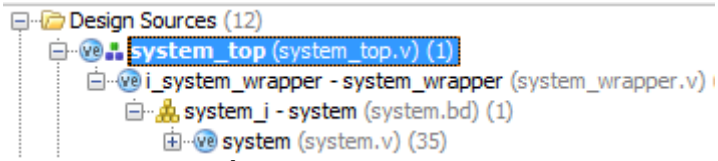


Figura 43 - Árbol de ficheros HDL del proyecto

Por último antes de obtener el fichero de configuración para la FPGA, si se pretende modificar las conexiones externas de la FPGA se deberá modificar el fichero de mapeo de pines, llamado *zed_system_costr.xdc*, presentado en la ***Figura 44***.

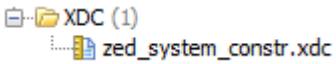


Figura 44 - Fichero de configuración de pines

Se debe tener en cuenta que el conector JB se encuentra en su totalidad ocupado por los pines para la lectura de sensores de ultrasonidos y que en el conector JA se encuentran el pin donde se recibe la señal del receptor de radio, el pin donde el sistema envía la señal PPM generada y tres pines ocupados por lectura de sensores de ultrasonidos, quedando sólo libres los pines JA8, JA9 y JA10. Además de estos 3 pines los conectores JC, JD y JE se encuentran libres. En la ***Figura 45*** se muestra el esquema de las conexiones de los conectores PMOD presentes en la placa.

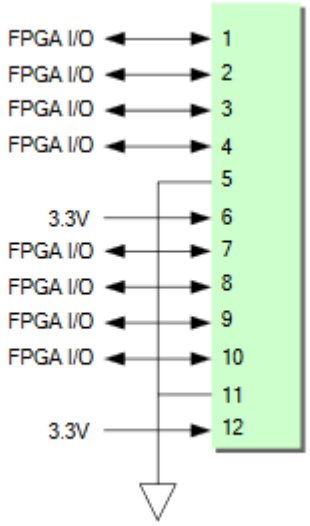


Figura 45 - Esquema de los conectores PMOD de la ZedBoard[xi]

En caso de requerirse alimentación de 5V para algún componente existe un pin de la placa desde donde se ha distribuido un árbol de alimentación para los sensores HC-SR04 que requieren esta tensión de alimentación.

Una vez completados estos pasos se puede proceder a la obtención del fichero de configuración de la FPGA, el *Bitstream*. Este fichero será empleado más tarde, debiendo ser empaquetado junto con el fichero *u-boot.elf* y *fsbl.elf* para generar el fichero *BOOT.BIN*, que programa la FPGA y lanza el sistema operativo. Este proceso se realiza desde el SDK de Xilinx.

Generación de aplicaciones software y modificación de la plataforma software:

Guía de uso de las funciones de la API:

La forma más fácil de controlar la acción de los motores a través de la señal enviada a la placa de control de vuelo es empleando la API desarrollada para actuar sobre el hardware de la plataforma.

Una nota importante para el desarrollo de software para la plataforma es que se recomienda compilar el código dentro del propio sistema embebido empleando gcc/g++, ya que el compilador cruzado del SDK de Xilinx no genera ejecutables reconocibles por el sistema con la configuración por defecto. El sistema cuenta con el editor de texto *nano*, de forma que es posible editar el código desde PC comunicándose con el sistema embebido mediante el puerto UART y empleando una aplicación de terminal serie

En general el primer paso para realizar el programa será llamar a la función ***quadcopter_init***, que realiza las siguientes funciones:

-Mapeo en memoria virtual de los periféricos empleados, a través de la función del sistema *mmap*.
-Inicialización de los registros generados por HLS para cada periférico en modo auto_restart, de modo que todos los módulos estén funcionando constantemente.
-Configuración del módulo multiplexor con uno valores definidos por defecto y que han sido probados de forma satisfactoria.
-Calibración del cuadricóptero.
-Arranque de motores.

Una vez realizada la llamada a esta función se obtiene un puntero al módulo generador de señal, así como al offset donde se encuentran los datos del primer canal, de forma que con este último puntero y aplicando las funciones ***set_gas***, ***set_roll***, ***set_yaw*** *y* ***set_pitch*** se puede controlar completamente el movimiento del cuadricóptero. Estas funciones requieren como argumento un valor entero entre -100 y 100 (de 0 a 100 en el caso de ***set_gas***, ya que no tiene sentido aplicar una potencia negativa), donde 0 equivaldría al joystick del mando correspondiente en el punto neutro, -100 al joystick completamente abajo en el caso de ángulos en el eje vertical (“pitch”) o completamente a la izquierda en el caso de los ángulos en el eje horizontal (“roll” y “yaw”) y 100 al joystick completamente arriba en el caso de los ángulos del eje vertical o completamente a la derecha en el caso de los ángulos del eje horizontal.

La otra función que permite controlar la señal enviada, esta vez a bajo nivel y con los anchos de pulso en ms como dato es ***set_values***. Esta función requiere como parámetros el puntero al primer dato del generador de señales y un vector de catorce valores, los trece primeros los anchos de pulso de los doce canales con el de sincronismo al final y el último correspondiente al tiempo que permanece la señal a ‘0’ entre canales.

Además de estas funciones para escribir sobre el generador de señales existen varias funciones pensadas para comprobar el funcionamiento del dispositivo a bajo nivel. ***check_values_base*** imprime en pantalla los valores y direcciones de los registros de

datos del generador de funciones recibiendo como parámetro la dirección base del periférico, mientras que ***check_values*** realiza la misma función pero con el puntero al primer dato como parámetro.

La función ***shutdown***, que recibe como parámetro el puntero a la dirección base del periférico genera una trama que el cuadricóptero interpreta como mando apagado, procediendo al apagado de los motores.

La función ***stop***, que recibe como parámetro el puntero a la dirección base del periférico genera el patrón de apagado de los motores, correspondiente a ambos joysticks apuntando completamente hacia abajo y a la izquierda.

La diferencia entre la recepción de la señal de mando apagado y la de apagado de motores radica en que al apagar el mando hay que volver a realizar la calibración del dispositivo, mientras que con la señal de apagado de motores es suficiente con enviar la señal de encendido de motores, ambos joysticks hacia abajo y la derecha.

Por último la función ***start_engines*** realiza la función contraria, recibe un puntero a la dirección base del generador de señal y produce una señal equivalente a ambos joysticks apuntando hacia abajo y a la derecha, provocando el arranque de los motores.

El siguiente grupo de funciones básicas son las de lectura de información, integradas por las que leen los valores obtenidos por los sensores de distancia y las que leen los valores obtenidos por el receptor de radiofrecuencia a través del módulo monitor de canales.

En primer lugar se encuentran las funciones ***read_gas, read_pitch, read_roll*** **y** ***read_yaw*** que permiten realizar lecturas de los valores de los parámetros enviados por el mando. Requieren la dirección del primer dato del monitor de canales y retornan un valor entero entre 0 y 100 en el caso de ***read_gas*** y un entero entre -100 y 100 en el caso de ***read_pitch***, ***read_roll*** y ***read_yaw***. Las funciones han sido diseñadas de estar forma para hacer su uso conjunto con las funciones ***set_gas***, ***set_roll***, ***set_yaw*** *y* ***set_pitch*** más intuitivo y sencillo.

Al efectuar lecturas con estas funciones se comprueba que los valores leídos no corresponden exactamente con los ideales, presentan leves variaciones sobre éstos. Esto se observa de forma evidente al efectuar medidas con todos los controles del mando en posición neutra, que debería retornar el valor 0 al llamar a cada una de las funciones, sin embargo en la ***Figura 46*** se observa que los valores están ligeramente desviados:

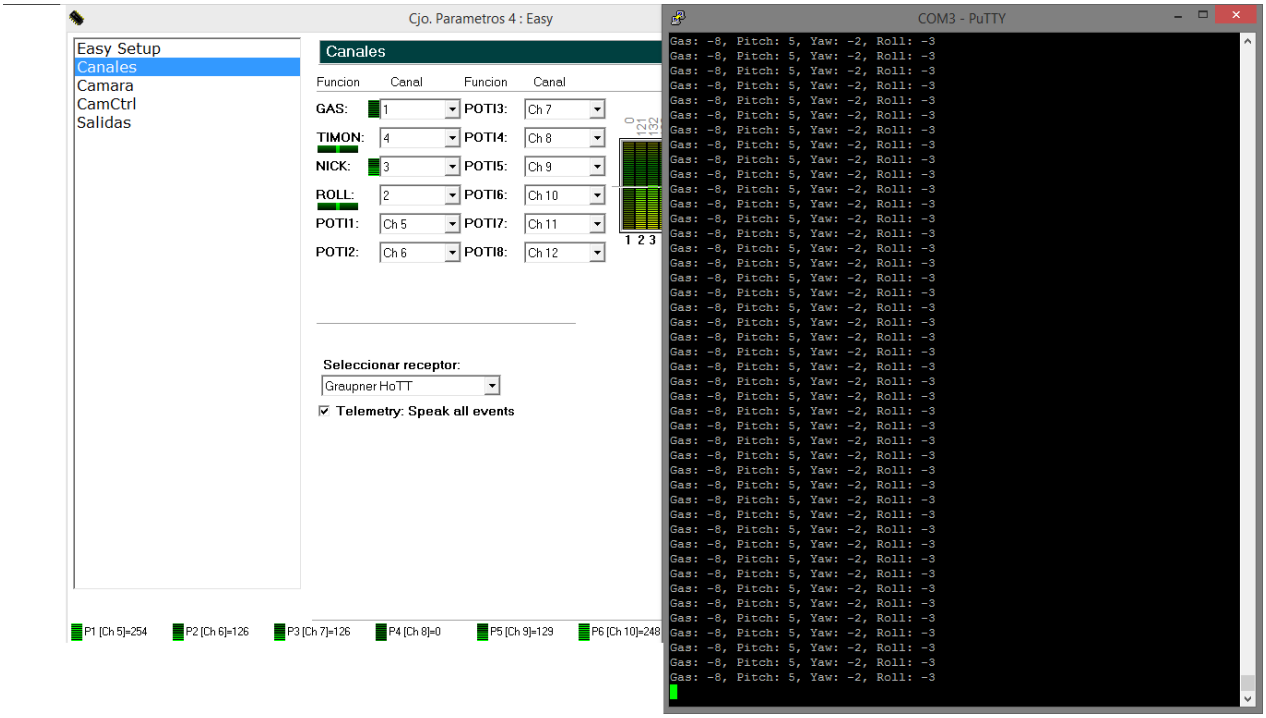


Figura 46 - Lectura realizada con todos los controles del mando en posición neutra

De igual forma, como se ve en la ***Figura 47***, este error es observable al colocar los mandos en sus posiciones extremas, ya que el valor leído no es exactamente 100:

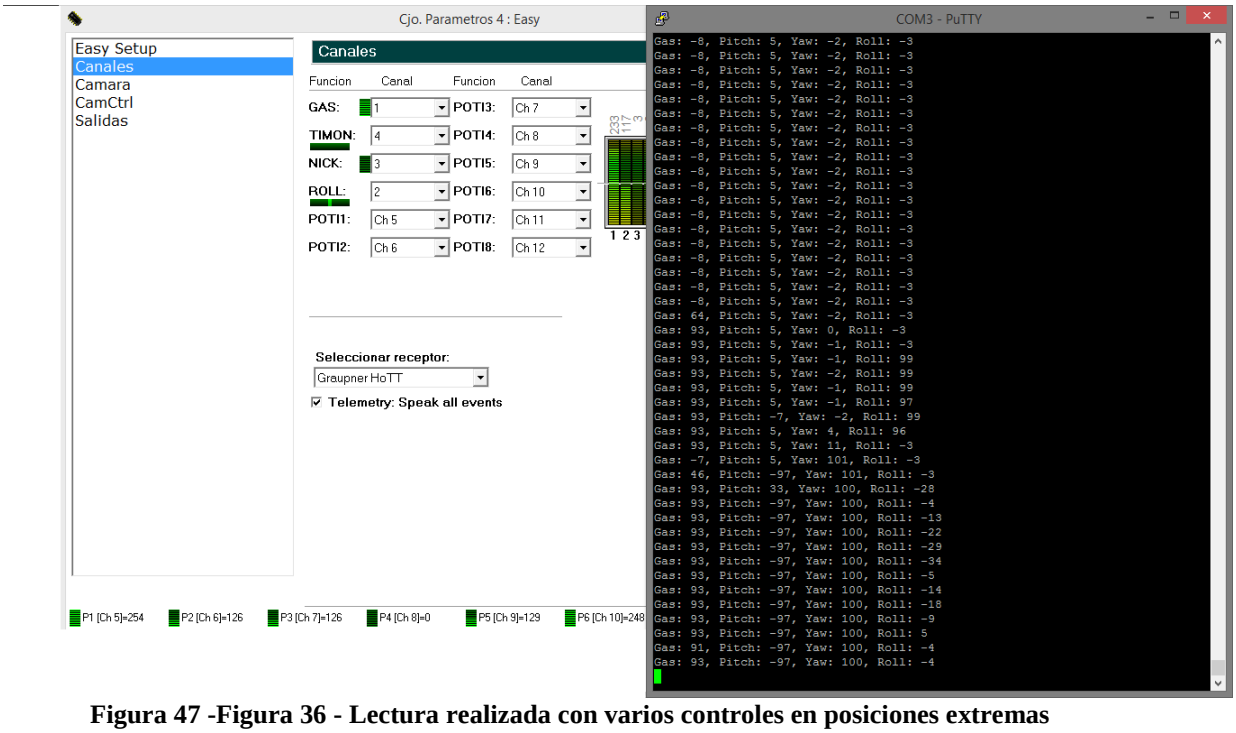


Figura 47 -Figura 36 - Lectura realizada con varios controles en posiciones extremas

Una consecuencia inmediata de este comportamiento es que no podría considerarse en una aplicación que emplee conjuntamente una parte programada y otra dirigida mediante el mando por el usuario que este está actuando sobre el mando sólo cuando todas las lecturas sean 0, sino que sería necesario dejar un umbral de margen en el que se considere que no hay acción sobre el mando.

Este efecto puede ser consecuencia tanto de una ligera falta de calibración del mando, tanto de la presencia de ruido o degradaciones en la señal recibida como la de la imposibilidad del hardware de leer con exactitud los valores de la señal recibida por alguna deficiencia en el diseño. Las causas más probables son las dos primeras, ya que la señal recibida por la placa de vuelo, monitorizada en la aplicación, que procede directamente de la obtenida del receptor MR16 a través del módulo multiplexor de señales, refleja también desviaciones de los valores ideales.

Por último la función ***read_distance*** que recibe como parámetro el puntero al sensor que se desea leer proporciona por el retorno de función un dato de tipo flotante con la distancia medida en centímetros. Es necesario verificar que el dato recibido es válido, ya que en ocasiones cuando el sensor no percibe ningún obstáculo o el obstáculo está demasiado cercano las medidas carecen de sentido, por tanto sería recomendable acotar los valores válidos en el intervalo de 10 a 300 centímetros.

Modificaciones y ajustes sobre el SO:

El primer apunte sobre el funcionamiento del SO corresponde al método empleado para que el sistema ejecute automáticamente el programa de vuelo al encender. Para realizar esta función se ha optado por un método que consiste en programar una tarea para el SO que se active cada vez que el sistema se inicia o reinicia, a través del fichero *crontab*, localizado en el directorio */etc*. En la ***Figura 48*** se muestra la estructura de este fichero.

```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
@reboot  /home/src/init config.sh
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
```

Figura 48 - Modificación sobre el fichero *crontab*

En él se ha añadido una tarea disparada por el evento de reinicio que ejecuta un script llamado *init_config.sh* localizado en la carpeta */home/src* que a su vez llama al programa de vuelo, llamado *quadcopter_init* situado en la misma carpeta. Si se quisiera ejecutar otro programa, bastaría con añadir otra tarea en el fichero *crontab* o bien modificar el scrip *init_config.sh* para que lanzase el programa deseado.

Otra opción posible sería el uso del comando *update-rc.d*, sin embargo no se ha comprobado su funcionamiento.

El resto de características corresponden a la posibilidad de emplear interfaces i2c a través de */dev/i2c0* y de la toma de imágenes de la cámara a través del programa *kamoso* instalado en el sistema.

Conclusiones y líneas futuras de trabajo:

Para finalizar el documento se presentarán las conclusiones obtenidas al término del proyecto y las posibles nuevas líneas de trabajo relacionadas con el mismo.

Conclusiones:

Tras el desarrollo de este trabajo se ha conseguido diseñar una plataforma que permite programar el control de los motores del cuadricóptero, así como especificar el funcionamiento del mismo en modo manual o automático a través de un interruptor del mando y la lectura de sensores de distancia por ultrasonidos.

El sistema electrónico puede ser alimentado con la batería del cuadricóptero y el programa de vuelo es capaz de ejecutarse automáticamente al encender la plataforma.

El objetivo final propuesto era realizar una prueba de demostración de un vuelo programado, sin embargo la peligrosidad de la realización de este tipo de pruebas sin contar con un entorno de pruebas que garantice una probabilidad de daños en el vehículo mínimas, así como la ausencia actualmente de una herramienta operativa que permita la simulación del comportamiento del mismo y la falta de un montaje físico fiable, sumados a la imposibilidad de la implementación de varios de los sensores que se te pretendía integrar han hecho que esto no sea posible.

En su lugar las pruebas se han realizado con las hélices del dispositivo aflojadas, de forma que no se transmite la potencia de los motores a las mismas, impidiendo que el vehículo despegue. De esta forma se puede comprobar que al menos superficialmente el comportamiento es el esperado anulando la posibilidad de dañar el vehículo.

Por tanto se podría decir que este proyecto representa la finalización de un primer paso para contar con una plataforma docente y de investigación basada en un UAV y con una alta flexibilidad a la hora de incorporar aplicaciones, debido a que cuenta con un sistema electrónico capaz de incluir diseños hardware y software bajo el mismo chip de forma rápida y cómoda.

Esta plataforma a falta de un montaje físico robusto y la incorporación de la sensórica necesaria restante puede ser empleada para programar rutinas de vuelo, incluyendo por tanto el desafío del mundo de los sistemas ciberfísicos y en concreto de los UAV en la docencia e investigación universitaria.

Líneas futuras de trabajo

Una vez finalizado el trabajo se aprecian varios puntos clave de mejora y caminos de evolución del proyecto:

- Finalización de la implementación de los sensores, principalmente altímetro, giróscopo y acelerómetro, todos ellos conectados mediante el bus I2C, así como la introducción de GPS, controlado mediante el puerto serie y la incorporación de un módulo WiFi por USB, de forma que el sistema tenga un gran conocimiento del entorno a través de los datos recibidos de estos módulos.

- Instalación de una librería de procesado de imágenes como OpenCV en el SO para facilitar el trabajo con la cámara.

- Montaje de una plataforma física adecuada para el montaje de toda la sensórica y componentes que se han incorporado al cuadricóptero, que actualmente se encuentra en un estado de prototipo donde el montaje no es del todo fiable, cómodo o estético. La incorporación de elementos de seguridad que protejan el vehículo y todos sus componentes sería un buen avance para la plataforma.

- Búsqueda o desarrollo de una plataforma de simulación donde poder comprobar el funcionamiento de las rutinas planteadas, de forma que a la hora de realizar pruebas físicas el riesgo de que el vehículo sufra daños sea mínimo.

- Adaptación, modificación de la API desarrollada, de forma que con la experiencia de uso mejore en cuanto a comodidad, flexibilidad y prestaciones.

- Mejora y depuración de los módulos instanciados en el sistema operativo, de forma que cuente con sólo lo necesario para esta plataforma en concreto y no contenga elementos que puedan reducir su rendimiento.

- Incorporación de un sistema de monitorización y depuración mediante WiFi u otro tipo de sistema inalámbrico que permita que el UAV pueda enviar información de estado al PC del usuario.

- Incorporación de un módulo LIDAR-Lite 2, que no supone un gasto excesivamente elevado y que cuenta con un rango de 40 m para poder realizar medidas a distancias considerables y aumentar las posibilidades de la plataforma.

Referencias:

[i] THE DRONES REPORT: Market forecasts, regulatory barriers, top vendors, and leading commercial applications – *Bussiness Insider* (2015, Jul 20) http://www.businessinsider.com/drones-report-market-forecast-2015-3

[ii] The Global UAV Payload Market 2012-2022 – *Strategic Defence Intelligence* (2012, Dec 31)

[iii] Global small UAV market growth of 6.52% CAGR by 2020 - analysis, technologies & forecast report 2016-2020 - key vendors: AeroVironment, elbit systems & SAAB - research and markets. (2016, Feb 12). *Business Wire* Retrieved from http://search.proquest.com/docview/1764735218?accountid=14497

[v]Cyber-physical Systems (Radhakisan Baheti and Helen Gill – IEEE Control System Society) http://ieeecss.org/sites/ieeecss.org/files/documents/IoCT-Part3-02CyberphysicalSystems.pdf

[vi] Wikipedia Aircraft principal axes
https://en.wikipedia.org/wiki/Aircraft_principal_axes

[vii] AXI reference guide (Xilinx Support) - UG761 (v13.1) (2011, Mar 7)
http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf

[viii] Datasheet del dispositivo HC-SR04: http://www.micropik.com/PDF/HCSR04.pdf

[ix] Datasheet del convertidor DC-DC http://power.murata.com/data/power/uwe.pdf

[x] Xilinx Zynq 7000 Overview
http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[xi] Guía HW de la ZedBoard
http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf