

# Máster Universitario en Software de Sistemas Distribuidos y Empotrados

*Sistema de Advertencia para Aeronaves Comerciales sobre  
tecnología FPGA (SAAC-FPGA)*

## PROYECTO FIN DE MÁSTER

*Oskar Andrzej Stepien*

Enero 2021



# Máster Universitario en Software de Sistemas Distribuidos y Empotrados

*Sistema de Advertencia para Aeronaves Comerciales sobre  
tecnología FPGA (SAAC-FPGA)*

## PROYECTO FIN DE MÁSTER

*Autor: Oskar Andrzej Stepien*

*Director: Javier García Martín*

Enero 2021



# Índice

|   |      |
|---|------|
| Índice .....                                | I    |
| Índice de Figuras.....                      | IV   |
| Índice de Tablas .....                      | VII  |
| Resumen .....                               | VIII |
| Palabras Clave .....                        | VIII |
| Abstract .....                              | IX   |
| Keywords.....                               | IX   |
| Objetivo del proyecto .....                 | X    |
| 1. Marco del problema.....                  | 1    |
| 2. Alternativas Existentes.....             | 2    |
| 2.1. GPWS/EGPWS.....                        | 2    |
| 2.2. AMSADD.....                            | 4    |
| 2.3. Sistema NASA.....                      | 5    |
| 3. Fases del proyecto.....                  | 7    |
| 4. Requisitos Específicos .....             | 10   |
| 4.1. Requisitos de Negocio.....             | 10   |
| 4.2. Requisitos de Usuario .....            | 10   |
| 4.3. Requisitos Funcionales.....            | 11   |
| 4.4. Requisitos No Funcionales .....        | 15   |
| 4.5. Requisitos de Interfaces Externas..... | 18   |
| 4.6. Restricciones de Desarrollo .....      | 19   |
| 4.7. Requisitos de Entorno Físico .....     | 19   |
| 5. Casos de Uso .....                       | 20   |
| 5.1. Diagrama .....                         | 20   |
| 5.2. Extendidos.....                        | 22   |
| 6. Diseño del Sistema .....                 | 39   |
| 6.1. Diagrama de Componentes .....          | 39   |
| 6.2. Mapa de Tareas .....                   | 42   |
| 7. Implementación del proyecto .....        | 44   |
| 7.1. Investigación Previa .....             | 44   |
| 7.2. Dispositivos Utilizados .....          | 50   |

## Índice

|        |  |     |
|--------|--|-----|
| 7.3.   | Tareas Individuales del Sistema .....                  | 59  |
| 7.4.   | Diseño del conjunto de tareas .....                    | 89  |
| 8.     | Pruebas del sistema .....                              | 94  |
| 8.1.   | Pruebas Casos de Uso .....                             | 94  |
| 8.2.   | Pruebas en el entorno final.....                       | 100 |
| 9.     | Aspectos Sociales, Ambientales, Éticos y Legales ..... | 105 |
| 9.1.   | Identificación de Aspectos Relevantes .....            | 105 |
| 9.2.   | Selección .....  | 107 |
| 9.3.   | Integración en el proyecto.....                        | 107 |
| 9.4.   | Valoración Final.....                                  | 108 |
| 10.    | Planificación del proyecto y Tabla de Costes .....     | 110 |
| 10.1.  | Planificación Final.....                               | 110 |
| 10.2.  | Tabla de Costes .....                                  | 112 |
| 11.    | Conclusión .....                                       | 113 |
| 12.    | Líneas futuras .....                                   | 115 |
|        | Referencias .....                                      | 116 |
| A1.    | Anexo I: Investigación IDE .....                       | 120 |
| A1.1.  | Comparativa de IDEs .....                              | 120 |
| A1.2.  | Resolución.....  | 133 |
| A1.3.  | Carga en la placa FPGA .....                           | 135 |
| A2.    | Anexo II: Investigación Dispositivos .....             | 137 |
| A2.1.  | Dispositivos testados.....                             | 137 |
| A3.    | Anexo III : Códigos del sistema .....                  | 152 |
| A3.1.  | Tarea Lector del Teclado .....                         | 152 |
| A3.2.  | Tarea Utilizar Sistema.....                            | 153 |
| A3.3.  | Tarea Acelerómetro .....                               | 155 |
| A3.4.  | Tarea Ultrasonidos .....                               | 158 |
| A3.5.  | Tarea Velocímetro.....                                 | 160 |
| A3.6.  | Tarea Presencia .....                                  | 163 |
| A3.7.  | Tarea Control de Riesgo .....                          | 165 |
| A3.8.  | Tarea Display .....                                    | 171 |
| A3.9.  | Tarea Tarjeta microSD.....                             | 172 |
| A3.10. | Función “main”.....                                    | 175 |



# Índice de Figuras

|  |    |
|--|----|
| Figura 1: Caja GPWS [11] .....   | 2  |
| Figura 2: Esquema comparativo GPWS - AMSADD [16] .....   | 5  |
| Figura 3: Captura Sistema NASA [17] .....  | 5  |
| Figura 4: Modelo de ciclo de vida en "V" seguido .....   | 9  |
| Figura 5: Diagrama de Casos de Uso .....   | 20 |
| Figura 6: Diagrama de Componentes completo.....  | 40 |
| Figura 7: Diagrama de Componentes del sistema.....   | 40 |
| Figura 8: Mapa de Tareas del sistema .....   | 42 |
| Figura 9: Placa FPGA Zybo Z7-10 .....  | 44 |
| Figura 10: Esquema de una FPGA [20] .....  | 45 |
| Figura 11: Diseño IP de ejemplo GPIO.....  | 46 |
| Figura 12: Ejes Pitch, Roll y Yaw de un avión. [24] .....  | 50 |
| Figuras 13a y 13b: Acelerómetro GY-521; a) frontal b) parte trasera .....  | 51 |
| Figura 14: Bloque IP AXI I2C con salida iic_rtl. .....   | 52 |
| Figura 15: Módulo detector de distancia por ultrasonidos HC-SR04.....  | 53 |
| Figura 16: Bloque IP PmodGPIO y salida Pmod .....  | 53 |
| Figura 17: Módulo ESP8266.....   | 53 |
| Figura 18: Módulo Level Converter .....  | 54 |
| Figura 19: Potenciómetro deslizante .....  | 55 |
| Figura 20: Módulos detectores de presencia por Infrarrojos; Izquierda: Módulo de presencia por IR FC-51; Derecha: Módulo sensor de movimiento HC-SR501 ..... | 55 |
| Figura 21: Teclado matricial PmodKYPD .....  | 56 |
| Figura 22: Módulo IP PmodKYPD y puerto Pmod asociado .....   | 56 |
| Figuras 23a y 23b: a)Diodo LED RGB de 4 pines; b) Módulo PmodLED con 4 diodos LED .....  | 58 |
| Figura 24: Zumbador RS-457011 .....  | 58 |
| Figura 25: Diseño del sistema completo en Vivado 2019.1 .....  | 59 |
| Figura 26: Captura del fichero de restricciones "constraints.xdc" .....  | 60 |
| Figura 27: Prueba tarea KeyPad .....   | 62 |
| Figuras 28a, 28b, 28c, 28d: Pruebas UseSystem. a, b, c) Vista en TeraTerm con ejecución; d)Registro en microSD .....   | 64 |
| Figuras 29a y 29b: Configuración para establecer "exFAT" y "timer_stack", en "BSP Settings"; a)Configuración "timer_stack"; b)Formato “exFAT” .....          | 65 |
| Figuras 30a, 30b y 30c: Prueba múltiples ficheros. a) Ejecución en "TeraTerm"; b) Contenido de Registros; c) Vista de los archivos.....                      | 66 |
| Figuras 31a, 31b, 31c: Angulo en función de la inclinación; a) Figuras de su obtención; b) Formula para el eje X; c) Formula para el eje Y [31] .....        | 68 |
| Figuras 32a, 32b, 32c y 32d: Prueba "accelerometerTask". a, b, c) TeraTerm d)Registro en microSD.....  | 70 |

|  |     |
|--|-----|
| Figuras 33a, 33b, 33c, 33d: Pruebas de la tarea "ultrasonicTask"; a, b, c)Muestras del display "TeraTerm"; b) Registro en la microSD .....                       | 74  |
| Figuras 34a, 34b, 34c, 34d: Prueba de "velocimeterTask"; a, b, c)Salida por display "TeraTerm"; b) Registro en tarjeta microSD .....                             | 76  |
| Figuras 35a, 35b, 35c, 35d: Prueba de "presenceTask"; a, b, c) Pruebas en display "TeraTerm"; d) Registro en microSD .....                                       | 78  |
| Figuras 36a, 36b, 36c y 36d: Prueba tarea "riskControlTask"; a, b, c) Display en "TeraTerm"; d) Registro en microSD .....  | 83  |
| Figuras 37a, 37b, 37c, 37d: Prueba tarea "displayTask"; a, b, c) Muestra en display "TeraTerm"; d) Registro en microSD .....                                     | 85  |
| Figuras 38a, 38b, 38c: Prueba de tarea "writeRecordTask"; a) Vista en "TeraTerm"; b) Registros de tareas; c) Registro de la tarea .....                          | 88  |
| Figuras 39a y 39b: Imágenes del prototipo; a)Vista lateral; b)Vista superior .....   | 90  |
| Figura 40: Propiedad "m" en librerías del proyecto .....   | 92  |
| Figuras 41a, 41b, 41c y 41d: Medidas de los tiempos de cómputo en display y registro; a, b) En TeraTerm; c, d) Registro en SD .....                              | 103 |
| Figura 42: Diagrama de Gantt del proyecto .....  | 111 |
| Figura 43: Captura Inicio Vivado 2019.1.....   | 123 |
| Figuras 44a, 44b, 44c y 44d: Capturas de la creación de un proyecto en Vivado; a) Tipo proyecto; b) Actualización placas; c) Selección placa; d) Finalizar ..... | 123 |
| Figura 45: Creación Diseño en Vivado 2019.1.....   | 124 |
| Figuras 46a y 46b: Añadir bloque Zynq7 al diseño IP; a) Incluir b) Auto conectar .....   | 124 |
| Figuras 47a y 47b: Capturas creación interfaz conexión GPIO al diseño IP; a) Añadido y selección b) Auto conexión.....   | 125 |
| Figura 48: Añadido de librerías Pmod Digilent para Vivado 2019.1 .....   | 126 |
| Figuras 49a, 49b y 49c: Capturas de creación de "HDL wrapper"; a) Diseño completo; b) Creación de HDL wrapper; c) Ejecución “bitstream” .....                    | 126 |
| Figuras 50a, 50b y 50c: Capturas creación hardware y exportación; a) Exportar hardware; b) Añadir bitstream; c) Ejecutar Xilinx SDK.....                         | 127 |
| Figura 51: Creación nuevo proyecto Xilinx SDK y configuración .....  | 128 |
| Figuras 52a y 52b: Programación y ejecución en placa FPGA; a) Programación de FPGA con código; b) Ejecución del código remotamente.....                          | 129 |
| Figuras 53a y 53b: Prueba de la placa; a) Fotografía de la placa durante la prueba; b) Captura de la salida por UART por medio de TeraTerm .....                 | 130 |
| Figura 54: Captura entorno Vitis 2020.1 .....  | 130 |
| Figuras 55a y 55b: Captura de la creación de un proyecto nuevo en Vitis 2020.1; a) Creación; b) Asignación de diseño hardware .....                              | 131 |
| Figura 56: Dominio de la aplicación .....  | 131 |
| Figuras 57a y 57b: Compilación y ejecución del código; a) Compilación; b) Ejecución en dispositivo.....  | 132 |
| Figuras 58a y 58b: Prueba de la placa; a) Fotografia de la placa durante la prueba; b) Captura de la slaida por UART por medio de Teraterm .....                 | 132 |
| Figura 59: Opciones para grabar el sistema .....   | 135 |
| Figura 60: Imagen del módulo Pcam 5C de Digilent .....   | 137 |

## Índice de Figuras

|  |     |
|--|-----|
| Figura 61: Prueba de la Pcam.....  | 139 |
| Figura 62: Fotografía módulo PmodLED.....  | 140 |
| Figura 63: Captura del diseño IP para probar GPIO y PmodLED .....  | 140 |
| Figura 64: Pestaña I/O Ports en "Elaborated Design" .....  | 143 |
| Figura 65: Dispositivos GPIO adicionales. Botón y diodo LED verde.....   | 144 |
| Figura 66: Imagen de potenciómetros (3 pines) .....  | 146 |
| Figura 67: Captura diseño IP usando XADC Wizard.....   | 147 |
| Figuras 68a, 68b, 68c y 68d: Capturas configuración XADC Wizard; a) Conf. Básica [AXI4Lite, Continuo, Secuenciador Canales]; b) Conf. ADC [Media a 16, desactivar Calibración]; c) Conf. Alarmas [Desactivar]; d) Conf. Canales [Añadir los indicados].. | 148 |
| Figura 69: Interfaz de "Generate Block Design" .....   | 149 |
| Figura 70: Diseño del circuito divisor de voltaje a 0.942V .....   | 150 |
| Figura 71: Salida analógica del terminal TeraTerm (Data14) .....   | 151 |

# Índice de Tablas

|   |     |
|---|-----|
| Tabla 1: Resumen comparativa de IDEs .....                                    | 47  |
| Tabla 2: Medidas Tiempo de Cómputo .....                                      | 103 |
| Tabla 3: Tabla con fechas de realización de cada fase .....                   | 111 |
| Tabla 4: Tabla de Costes del proyecto.....                                    | 112 |
| Tabla 5: Código prueba diodos LED y botones en Zybo Z7-10 .....               | 129 |
| Tabla 6: Resumen comparativa de IDEs .....                                    | 134 |
| Tabla 7: Código ejemplo GPIO (input) .....                                    | 141 |
| Tabla 8: Código ejemplo GPIO (output) .....                                   | 142 |
| Tabla 9: Código utilizado en la prueba de GPIO (output).....                  | 144 |
| Tabla 10: Código utilizado para GPIO (input y Output) .....                   | 146 |
| Tabla 11: Fragmento respecto al PmodJA(XADC) en fichero Constraints.xdc ..... | 149 |
| Tabla 12: Código utilizado para probar el módulo XADC y potenciómetro .....   | 150 |

## Resumen

---

Este proyecto busca ofrecer un sistema complementario al Sistema de Aviso de Proximidad de Suelo (GPWS) instalado actualmente en la mayoría de las aeronaves comerciales. Estos sistemas pueden resultar poco fiables ante determinadas situaciones al permitir un vuelo a baja altitud, estableciendo un peligro para los viajeros del avión.

El proyecto sugiere la creación de un sistema que evalúe varios posibles riesgos para el vuelo, valorando en cada instante la necesidad de una respuesta a ellos. También, se realizará un seguimiento del ciclo de vida de este sistema estableciendo entre otros, requisitos y casos de uso. Para la creación del sistema complementario se hará uso de un sistema operativo de tiempo real (RTOS) sobre una placa con la tecnología FPGA y varios dispositivos para medir los valores a tener en cuenta.

Los dispositivos utilizados en el sistema permiten medir la inclinación, altitud y velocidad de la aeronave junto a la presencia de la tripulación en la cabina, haciendo uso de varios dispositivos para ello. En caso de superar los límites establecidos, se creará una respuesta visual y auditiva. Además el sistema mostrará información de la aeronave a los pilotos, manteniendo un registro de las acciones tomadas haciendo uso de una tarjeta microSD.

En el trabajo se valorarán varios entornos y dispositivos con una vista enfocada a la realización de proyectos similares para la cual es necesaria una investigación de estos. Por ello, se disponen varios anexos para comprender esta investigación previa adicional, junto con el código del sistema.

## Palabras Clave

Seguridad en Aviación; Ciclo de vida; EGPWS; FPGA; RTOS.

## Abstract

---

This project aims to find a complementary system to the Ground Proximity Warning System (GPWS) currently installed on most commercial aircraft. These systems could be unreliable in certain circumstances by allowing low altitude flight, creating a hazard for the aircraft passengers.

The project suggests the design of a system that evaluates several possible risks for the flight, assessing at each moment the need for a response to them. Also, the life cycle will be monitored using between others, requirements, and use cases. For the development of the complementary system, a real-time operating system (RTOS) will be used on a board with FPGA technology and several devices to measure the values to consider.

The devices used in the system allow to measure the inclination, altitude, and speed of the aircraft along with the presence of the crew at the cabin, making use of several devices for this. If the established limits are exceeded, a visual and auditory response will be created. Also, the system will display information from the aircraft to the pilots, keeping a record of the actions taken using a microSD card.

In this work, several environments and devices will be evaluated with a view focused on the realization of similar projects for which an investigation of these is necessary. Therefore, several annexes are available to understand this additional previous research, together with the system code.

## Keywords

Aviation Security; Life cycle; EGPWS; FPGA; RTOS.

## Objetivo del proyecto

---

El objetivo del proyecto será implementar un sistema para mejorar la seguridad de un avión comercial. Para ello se realizará un sistema complementario a los sistemas de aviso instalados en muchas aeronaves comerciales. Para ello el objetivo es detectar situaciones de riesgo y activar las alertas en caso necesario. Estas pueden ser una baja altitud por debajo de la establecida o una inclinación del avión excesiva en pleno vuelo. Se establece por normativas de vuelo que la condiciones en pleno vuelo de una aeronave son 11000 metros a velocidad de crucero (827 km/h).

El proyecto se realizará sobre una placa FPGA [1], estas posibilitan una programación de sus componentes con facilidad. Cabe indicar, que se considera cómo el punto medio entre un circuito impreso en PCB y una programación íntegra en software. Se ha decidido por utilizar la tecnología FPGA, al ser muy solicitada en los últimos años y a la cual se le augura mucho futuro [2].

En este proyecto no se va a incidir en la programación de sus componentes, si no, la programación de esa placa FPGA para sistemas de tiempo real, los cuales usan el acrónimo de STR. Estos sistemas interactúan con un entorno físico basado en entradas y salidas donde se establecen intervalos de tiempo definidos. Su funcionamiento está basado en un conjunto de ejecuciones de tareas concurrentes, las cuales disponen de plazos de ejecución y esquemas de activación que indican su periodicidad.

En el caso del proyecto a realizar, es un STR con requisitos críticos, caracterizados por disponer de plazos de tiempo que no admiten retrasos. El sistema que se creará será un controlador de un avión que activará la función de piloto automático, en condiciones excepcionales del vuelo. Estas condiciones se basarán en las actuales medidas o recomendaciones que deben respetar los pilotos durante el vuelo.

Como objetivo secundario, se incluirá una utilidad educativa donde se realizará un recorrido del ciclo de vida de un sistema completo de tiempo real como el propuesto.

## 1. Marco del problema

---

El avión, y en general el medio aéreo, es considerado uno de los medios de transporte más rápido y el más seguro de ellos [3], con una tasa de accidentes de 1 por cada 7.35 millones de vuelos. Esta tasa, es obviamente muy pequeña y genera confianza, aun así, se siguen produciendo accidentes.

Existen varios motivos por los cuales se pueden llegar a producir [4], varios de ellos ambientales o errores humanos no relacionados con los pilotos. Se estima que el 72% de los casos ocurridos, se deben a fallos del piloto o fallos mecánicos del avión. Uno de los accidentes que más impactaron y por el cual se tomó una medida inusual respecto a los pilotos es el ocurrido el 24 de marzo de 2015 con el vuelo GWI9525 de Germanwings [5] . En él, y gracias a la caja negra del avión se supo que el copiloto, forzó el piloto automático para un descenso fuera de lo normal y acelerado, acabando trágicamente, en un accidente donde fallecieron 150 personas.

Desde ese día, las compañías exigen más medidas de seguridad, las cuales ha recopilado El País en 2016 [6]. Entre las medidas tomadas relacionadas con la salud del piloto y su estado psicológico, las cuales no es sencillo de diagnosticar en pleno vuelo, se exige la presencia de dos miembros de la tripulación en todo momento del vuelo, uno de ellos debe ser, en los casos de emergencia, un piloto. Esta medida se tendrá en cuenta en el sistema final.

De esta manera, y para reducir aún más la tasa de accidentes, se propone un sistema que sea capaz de activar un piloto automático forzadamente en caso de extrema emergencia y éste no pueda ser modificado manualmente.

También, el actual sistema en funcionamiento de detección de advertencias, *Enhanced Ground Proximity Warning System* (EGPWS por sus siglas en inglés) se ha convertido en un sistema poco fiable en los últimos años. Como ejemplo de esto, el incidente ocurrido el 7 de Julio de 2017 con el vuelo AC759 [7], de Air Canada, que no consiguió advertir correctamente a los pilotos de un cruce de los aviones en la pista de aterrizaje. Esto supuso un peligro potencial y retrasos en varios vuelos, por ese motivo, se han activado varios proyectos de mejora del sistema actual.

## 2. Alternativas Existentes

---

La existencia de varias alternativas se ha visto reducida a lo largo de los años, debido a que el sistema real que utilizan los aviones ya es lo suficientemente avanzado, y la aparición de un sistema que compita contra él es poco usual. El primer sistema de este tipo fue creado prácticamente al mismo tiempo que la aviación moderna, a mediados de 1970, por el canadiense C. Donald Bateman [8]. A él se le atribuye la patente de uno de los sistemas de aviso y notificación en aviones más utilizados, *GPWS* o *EGPWS* [9].

### 2.1. GPWS/EGPWS

*GPWS* [9] [10] es el acrónimo de *Ground Proximity Warning System*, y es un sistema muy utilizado en los aviones en su versión mejorada (*EGPWS*), que añade varios mapas con la localización y topografía de posibles obstáculos, pero su acrónimo se mantiene como *GPWS*. Su funcionamiento, se basaba en la lectura y entendimiento de los diferentes sensores que tiene integrados, produciendo respuestas visuales y auditivas en la cabina de los pilotos. El diseño físico de éste se puede ver en la siguiente figura (Figura 1).



Figura 1: Caja GPWS [11]

Este sistema ofrece 7 modos de funcionamiento o alerta. [12] [13]

- **Modo 1** (Régimen de bajada excesivo)

Se activa Master Warning y suena el mensaje "*Sink rate*" en cabina. Indica una bajada muy pronunciada, en comparación con la separación respecto al terreno. La respuesta por parte de los pilotos sería elevar el morro del avión.

- **Modo 2** (Régimen de acercamiento al terreno excesivo)

Evaluá la separación que hay entre el avión y el terreno, podría ser causado por una bajada del avión o una subida del terreno abrupta. Suena el mensaje "*Terrain Terrain*" y en caso de omisión, se une el mensaje "*Pull Up*". Para elevar el avión, este aviso distingue entre subida, crucero, despegue o aterrizaje.

- **Modo 3** (Perdida de altitud tras el despegue)

Advertencia que indica una bajada poco usual al despegue, se desactiva al alcanzar los 1000 pies (~300 metros). El mensaje que suena en la cabina es "*Don't pull*" y un aviso luminoso en toda la cabina.

- **Modo 4** (Demasiado cerca del terreno)

Este modo es activado cuando el terreno aumenta de forma progresiva, avisando con el mensaje "*Too Low Terrain*". También avisa si se tiene el tren arriba o si se dispone de los flaps mal configurados, indicando sonoramente "*Too Low Gear*" y "*Too Low Flaps*", respectivamente.

- **Modo 5** (Desviación por debajo de la senda del planeo)

En la fase donde el avión se acerca a la pista de aterrizaje, se activa la senda de planeo. Donde en caso de una bajada por debajo de ésta, sonará el aviso de "*Glide Slope*", indicando que el avión tenga que reajustarse en altura, o elevar el vuelo, hasta continuar con la senda de planeo.

- **Modo 6** (Ángulo excesivo de alabeo y alertas de altitud)

En caso de realizar un ángulo excesivo en el alabeo, el sistema avisará mediante el mensaje "*Bank Angle*". Adicionalmente, también indicará la altura hasta el aterrizaje en pista, avisando en pies o metros.

- **Modo 7** (Aviso de cizalladura)

La cizalladura es un fenómeno, donde el viento realiza un cambio de velocidad o dirección bruscamente, es más común a menor altura. El sistema avisará con el mensaje "*Windshear*".

Tras la búsqueda de muchas incidencias relacionadas con el sistema *GPWS*, se puede ver que es un sistema muy fiable de vuelo, y las causas de esas incidencias en

los vuelos se deben a un error del piloto automático y/u omisión de las alertas del GPWS al pensar que son falsos positivos. Entre los artículos, hay un documento de *National Aeronautics and Space Administration (NASA)* [14] donde se especifican varias incidencias.

Entre estas, se quiere informar sobre un error en la activación del GPWS, que se entendió como una bajada excesiva por parte del piloto de marzo de 2017 (Caso 3 de 50, pág. 18). Otra, un fallo del GPWS en la detección de terreno elevado en un lugar sin incidencias previamente en febrero de 2017, que se entendió como un fallo en el diseño y mapas utilizados (Caso 8 de 50, pág. 28). Por último se destaca un incidente donde el sistema GPWS no indicó la existencia de 2 torres en la ruta del vuelo. El piloto modificó la ruta para evitar esas torres, el sistema GPWS no las advirtió (Caso 27 de 50, pág. 73). Estos casos fueron notificados por los propios pilotos a *Aviation Safety Reporting System (ASRS)*, sistema de notificación de informes que NASA habilitó.

Anteriormente se indicó el incidente de *Air Canada* [7] ocurrido el 7 de julio de 2017 en San Francisco. En él, el sistema GPWS no detectó cuatro aviones en la pista durante la rutina de aterrizaje designada, y el piloto reaccionó con la aprobación de la torre de control. La gran cantidad de aviones se debía a un retraso en el despegue de otros vuelos, lo que supuso ocupar una pista en el aeropuerto de destino. El incidente se resolvió, alegando que el sistema GPWS no detecta objetos en el eje de dirección del avión, y que se requería del piloto manual.

## 2.2. AMSADD

Adicionalmente al GPWS, existió un proyecto que no era tan eficaz como éste, el cual se basaba en una base de datos de la topografía del terreno, AMSADD [15], acrónimo de *Automatic Minimum Safe Altitude Digital Display*, en este caso, AMSADD es un proyecto moderno, en base a varios incidentes ocurrido con la altura y las lecturas de GPWS. Cómo indican los propios autores, este sistema es sólo una mejora o añadido a los sistemas actuales, que permite a los pilotos disponer de un dato preciso con la altura y posibles direcciones a tomar. Se aprovecha de tener una base de datos actualizada y avisar 2 minutos antes a cualquier otro sistema. Esta mejora, cómo ya se comentó sobre EGPWS que dispone también de mapas, tiene en cuenta muchos más factores y es posible que no identifique correctamente los obstáculos.

En la teoría, AMSADD es capaz de prevenir incidentes con terrenos altos repentinos o muy pronunciados. Ya sea por falta de visibilidad, y principalmente, tiene en cuenta la altitud mínima segura de vuelo (MSA) en base a esos mapas, de esta manera, es capaz de avisar previamente.

Se muestra una comparativa de esos sistemas en la siguiente figura (Figura 2), donde es posible identificar con una altitud superior cualquier obstáculo y cómo reaccionarían otros sistemas o GPWS con la altura del vuelo sin definir.

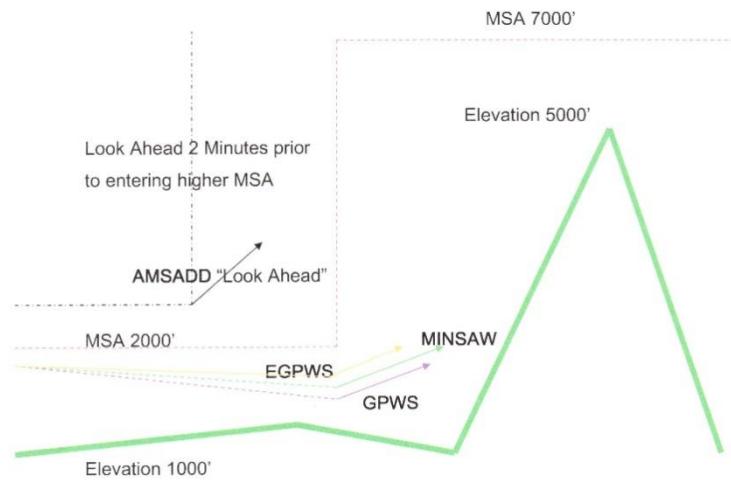


Figura 2: Esquema comparativo GPWS - AMSADD [16]

## 2.3. Sistema NASA

Otro sistema en desarrollo es el que está relacionado con la Administración Nacional de Aeronáutica y el Espacio (NASA), quienes aseguran que el GPWS es inadecuado y sus medidas insuficientes. Como muestra de su interés, crearon la ASRS, comentada previamente, para realizar informes de incidencias.

También muestran el funcionamiento de su sistema [17], cuya captura se ha mostrado como diseño de éste en la figura siguiente (Figura 3). Éste fue comentado negativamente por el proyecto de AMSADD [10] porque al igual que GPWS obvia la MSA, altitud mínima segura, la cual ha creado varios incidentes evitables al volar por debajo de ésta.

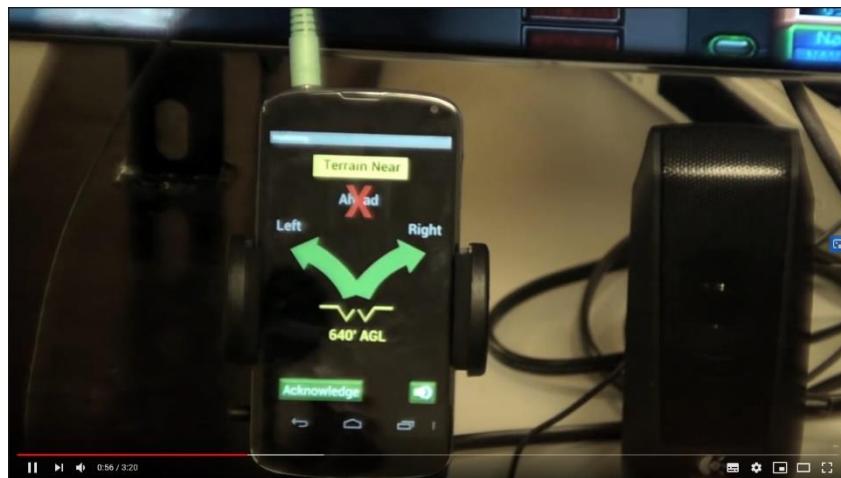


Figura 3: Captura Sistema NASA [17]

## Alternativas Existentes

Sin embargo, AMSADD admite que este sistema es mejor que GPWS, pero sigue sin ser adecuado.

Estos últimos proyectos, en especial el diseñado por NASA, es un display que se mantiene conectado al sistema y te muestra las acciones a realizar, sin mostrarte el radar que ofrece GPWS, mejorando el sistema. Sin embargo, AMSADD es una aplicación gratuita, que te indica la MSA adecuada en cada circunstancia y cuando elevarse o bajar, que tiene su utilidad al ser el elemento que necesita GPWS para prevenir situaciones de peligro respecto al terreno.

### 3. Fases del proyecto

---

Para conseguir abarcar el proyecto de manera correcta, éste será dividido en varias fases o etapas a seguir.

En primer lugar, el proyecto está dividido en dos grandes fases, la fase relacionada con Ingeniería de Requisitos y Sistemas, donde se detallan los aspectos formales del sistema a desarrollar, junto con sus diagramas y casos de uso para obtener una visión completa del sistema. La segunda gran fase, es la que se autocompletará con la primera, consistente en el desarrollo práctico del sistema, en esta fase se especifican detalles de cada componente y su composición en el proyecto final.

Cada una de estas fases se corresponden a varios hitos, subfases o etapas que deben ser medidas en el tiempo para finalizar el sistema en un plazo fijado. Por ese motivo, ambas pueden ser intercaladas en su realización sin interferir entre sí.

La fase de Ingeniería de Requisitos y Sistemas abarca los aspectos más formales del sistema. Puede ser dividida en las siguientes etapas.

- **I1. Descripción del proyecto y su sistema.**

En esta etapa, se crea una primera visión del sistema en su conjunto, gran parte de ésta se rectificará posteriormente con el objetivo de dar la visión adecuada. Esta etapa incluye a su vez, obtener un resumen del sistema donde con una breve lectura se pueda identificar al sistema.

- **I2. Investigación sobre alternativas existentes**

Se cree indicado realizar una búsqueda de los posibles sistemas con uso similar que existen. Debido a que es un tipo de sistema que no se suele modificar, no existe una gran cantidad de ellos. Servirá para situar el sistema en desarrollo basándose en el funcionamiento o ideas ya propuestas para sistemas similares.

- **I3. Identificar los requisitos del proyecto**

Etapa imprescindible, dado que el objetivo del proyecto es realizar el ciclo de vida de un sistema “básico”, esta etapa es la más costosa en el tiempo,

## Fases del proyecto

que consigue simplificar su realización, con una descripción correcta del sistema y conociendo su funcionamiento.

- **I4. Realización de casos de uso**

En esta etapa se indicará qué casos de uso se definen en el sistema, su funcionalidad de manera sencilla y comunicación con otros casos de uso del sistema. También vendrá indicado cada caso de uso de una manera más extensa con su funcionalidad detallada para especificar de la manera más precisa cada uno.

- **I5. Realización de un diagrama de componentes**

Con este diagrama, se puede ver de manera general el funcionamiento del sistema, además de conocer las tareas que lo componen.

- **I6. Realización de diagramas adicionales**

Valorando las etapas anteriores, se decidirá qué otros diagramas son necesarios para comprender al sistema. Los diagramas adicionales pueden ser un mapa de tarea o un diagrama de estados.

La siguiente fase general del proyecto es la fase de desarrollo, ésta es más práctica y hace uso de los resultados obtenidos de la fase anterior. Se divide en las siguientes etapas.

- **D1. Valoración del material e Investigación**

Partiendo de una idea ya definida, y especificadas sus posibilidad y objetivos, se hará un análisis y prueba de cada dispositivo utilizado. Principalmente, el uso y posibilidades de la placa FPGA. En esta fase se definirá con mayor detalle qué dispositivos utilizará el sistema y su propósito final, al igual que la elección de un entorno de desarrollo. Se asegurará que todos los dispositivos utilizados funcionen de manera correcta evitando cambios en fases cercanas a la fecha límite.

- **D2. Crear las tareas del sistema**

Conociendo las tareas y en base al diagrama de componentes realizado en la fase I5 y los dispositivos previsto a utilizar en el sistema, se creará y probará cada tarea en el entorno final utilizado. Inicialmente cada tarea hará su funcionamiento sin interrupción, para definir y probar su correcto funcionamiento individual.

- **D3. Realizar el conjunto de tareas**

Tras las pruebas de tareas individuales, se realizará gradualmente un conjunto de los dispositivos, verificando que no exista un solapamiento de las tareas en el conjunto. En esta fase, también se incluirán detalles del montaje del prototipo inicial para que no existan problemas en su montaje.

- **D4. Realización de pruebas con el prototipo**

En esta fase, se parte con la premisa de que el prototipo se encuentra montado y debe ser probado para ver que funciona como se indica en la especificación. En caso de algún problema, se valoran pequeños cambios en las tareas.

- **D5. Verificación y valoración final del prototipo**

A pesar de ser similar a la fase anterior, en esta fase sólo se realizarán pruebas físicas con el entorno y el prototipo del sistema. También se valorará el cambio de imagen del prototipo en caso de ser necesario.

El ciclo de vida del sistema sigue el modelo de desarrollo en V. Siendo en este caso el modelo más adecuado para el sistema. La división en tareas del proyecto es posible asociarlas a este esquema, visible en la siguiente figura (Figura 4). En ésta se puede ver la agrupación de fases comentadas previamente donde se puede asociar las fases de diseño con las de verificación, permitiendo una trazabilidad del proyecto.

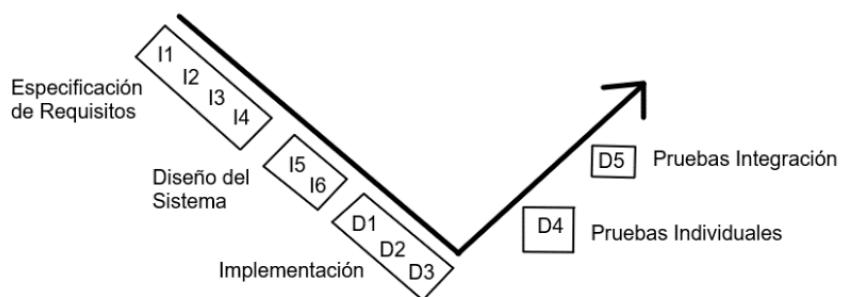


Figura 4: Modelo de ciclo de vida en "V" seguido

Adicionalmente, se realizará una valoración de aspectos del proyecto en general, la metodología seguida, finalizando con una conclusión, posibles mejoras y futuro del proyecto.

## 4. Requisitos Específicos

---

Estos requisitos han sido especificados siguiendo la división por el estándar IEEE 830 [18].

### 4.1. Requisitos de Negocio

- RN1.** El sistema deberá ser instalado en la cabina de un avión.
- RN2.** El sistema será capaz de detectar de manera automática situaciones peligrosas.
- RN3.** El sistema será funcional en pleno vuelo.
- RN4.** Un técnico podrá evaluar el vuelo tras el aterrizaje por medio del sistema.
- RN5.** El sistema utilizará un sistema operativo de tiempo real instalado en un dispositivo FPGA de Xilinx, Zybo Z7-10.
- RN6.** El sistema hará uso de un sistema de piloto automático estricto ya implementado.

### 4.2. Requisitos de Usuario

- RU1.** Los pilotos recibirán la información por medio de:
  - RU1.1.** Luz LED RGB para los 3 niveles de criticidad de situación.
  - RU1.2.** Módulo de luces LED para indicar la cantidad de situaciones peligrosas cometidas.
  - RU1.3.** Display que mostrará:
    - RU1.3.1.** Velocidad actual.
    - RU1.3.2.** Giro en X e Y actual.
    - RU1.3.3.** Altitud actual.
    - RU1.3.4.** Situaciones detectadas con el sensor asociado.
  - RU1.4.** Alarma sonora con 3 niveles de pitido para cada situación.
  - RU1.5.** Luz LED indicadora del estado activado del sistema.

- RU2.** El sistema alertará de manera leve pero eficaz a los pilotos por medio del uso de un pitido y luces LED dependiendo de la situación detectada.
- RU3.** Las respuestas del sistema no interferirán con la labor de los pilotos.
- RU4.** El sistema creará un entorno seguro tanto como para la tripulación como los pasajeros.
- RU5.** La información será mostrada a los pilotos por medio de un display instalado en la cabina.
- RU6.** El sistema requerirá la introducción manual de un código de 6 caracteres hexadecimales para la activación y desactivación de éste.
- RU7.** El sistema detectará la presencia en cabina de los pilotos, requiriendo la presencia de 2 personas en ésta en todo momento.
- RU8.** Se detectará cualquier movimiento generado en la puerta de la cabina.
- RU9.** Se almacenará un registro de actividad en una tarjeta microSD instalada en el sistema.
- RU10.** El uso del piloto automático estricto restringirá cualquier cambio manual por parte de los pilotos en caso de una situación crítica.
- RU11.** El sistema respetará la altitud mínima recomendable (MSA).
- RU12.** El sistema permitirá la utilización de otros sistemas de Identificación de Riesgos sin interferir en su funcionamiento.

### 4.3. Requisitos Funcionales

- RF1.** El sistema analizará la información del vuelo a través de varios sensores para detectar situaciones peligrosas durante el vuelo.
  - RF1.1.** Acelerómetro.
  - RF1.2.** Sensor de Ultrasonidos
  - RF1.3.** Velocímetro.
  - RF1.4.** Detector de proximidad por infrarrojos.
  - RF1.5.** Sensor de Movimiento.
- RF2.** El sistema responderá con notificaciones sonoras y visuales en la cabina del avión haciendo uso de los actuadores del sistema.
- RF3.** El sistema de alerta se divide en situaciones específicas y situaciones generales.
  - RF3.1.** Las situaciones específicas son las que detectan los sensores, además vienen con el motivo de la situación como alerta.
    - RF3.1.1.** Se clasifican en situaciones leves, graves y críticas (definidas más adelante).
      - RF3.1.1.1.** La detección de una situación específica leve activará la situación general leve con su respuesta asociada.

## Requisitos Específicos

**RF3.1.1.2.** La detección de 3 situaciones específicas leves o una grave, activará la situación general grave con su respuesta asociada.

**RF3.1.1.3.** La detección de 2 situaciones específicas graves o una de nivel crítico, activará la situación general crítica con la respuesta asociada.

**RF3.2.** Las situaciones generales son las que tienen asociada una respuesta por parte del sistema, dependiendo de las situaciones específicas.

**RF3.2.1.** Se clasifican en situaciones leves, graves y críticas.

**RF3.2.1.1.** Situación general leve. El sistema activará la señal sonora al nivel 1, y encenderá el LED al color azul.

**RF3.2.1.2.** Situación general grave. El sistema activará la señal sonora al nivel 2, y encenderá el LED amarillo.

**RF3.2.1.3.** Situación general crítica. El sistema activará la señal sonora al nivel 3, encenderá el LED rojo y activará el piloto automático estricto, restringiendo la modificación del uso a la tripulación.

**RF3.2.2.** La situación general leve puede ser desactivada con un correcto vuelo durante 5 segundos.

**RF3.2.3.** La situación general grave puede ser reducida a leve, si se mantiene durante 5 segundos las condiciones de la situación general leve.

**RF3.2.4.** La situación general crítica, no será posible de desactivar manualmente, se desactivará automáticamente tras 15 segundos.

**RF3.2.4.1.** La desactivación pasará a la situación general normal, sin respuesta del sistema.

**RF4.** El acelerómetro medirá la información de los ejes X e Y (roll y pitch).

**RF4.1.** Si en el eje X (roll o alabeo) se detecta durante 750 milisegundos (3 lecturas consecutivas) una lectura de la inclinación superior a 30 grados en cualquier sentido del eje X, será considerado como una situación específica leve del dispositivo. Activando además la alerta "High Roll Angle".

**RF4.1.1.** En caso de mantener el valor durante los 500 milisegundos (2 lecturas) siguientes, el nivel será aumentado a situación específica grave.

**RF4.2.** Si en el eje Y (pitch o cabeceo) se detecta durante 750 milisegundos (3 lecturas consecutivas) una inclinación superior a 20 grados en cualquier sentido del eje Y, será considerado como una situación específica leve, y se activará también la alerta "High Pitch Angle".

**RF4.2.1.** En caso de mantener el valor durante los 500 milisegundos (2 lecturas) siguientes, el nivel será aumentado a situación específica grave.

**RF4.3.** En caso de darse ambas condiciones, tanto X superior a 30 e Y superior a 20, durante 750 milisegundos (3 lecturas consecutivas) el sistema activará la situación específica grave en el sistema, activando la alerta “High Pitch and Roll Angle”.

**RF4.3.1.** En caso de mantenerse el valor durante los 500 milisegundos (2 lecturas) siguientes, el nivel será aumentado a situación específica crítica.

**RF4.4.** Mantener el valor ideal (0 grados) en X e Y durante 500 milisegundos (2 lecturas consecutivas), reducirá en un nivel la situación específica de peligrosidad actual, siendo posible desactivar la señal correspondiente.

**RF5.** El sensor de ultrasonidos devolverá la altitud a la cual se encuentra el avión.

**RF5.1.** La altitud máxima de 12000 metros (en el prototipo, 170 cm) corresponde al techo de vuelo común en muchas aeronaves comerciales. Superar ese límite durante 900 milisegundos (3 lecturas consecutivas) se considerará como una situación específica grave, activando además la alerta “High Altitude”.

**RF5.1.1.** Mantener la altitud durante otros 900 milisegundos (3 lecturas consecutivas), aumentará a una situación específica crítica.

**RF5.2.** La altitud mínima, 10000 metros (en el prototipo, 141 cm), respeta la distancia mínima recomendada (MSA) de 2000 metros por encima de cualquier terreno. Reducir esa altitud, activará una situación específica leve, además de la alerta “Low Altitude”.

**RF5.2.1.** Mantener la altitud por debajo durante otros 900 milisegundos (3 lecturas consecutivas), aumentará la situación del sensor actual.

**RF5.3.** Mantenerse entre los límites de 10000 y 12000 metros (en el prototipo, 141-170 cm) durante 900 milisegundos (3 lecturas consecutivas), reducirá un nivel la situación actual de este sensor.

**RF5.3.1.** Las alertas se mantendrán excepto si no hay ninguna situación detectada.

**RF6.** El velocímetro medirá la velocidad del avión.

**RF6.1.** En el prototipo se sustituye por un potenciómetro horizontal.

**RF6.2.** El rango se dividirá en 5 subrangos que indicarán situaciones peligrosas.

**RF6.2.1.** Si el avión mantiene durante 1500 milisegundos (3 lecturas consecutivas) una velocidad entre el subrango [754, 783.2] será considerado como una velocidad demasiado baja, activando la

## Requisitos Específicos

alerta “Too Low Speed” e indicando una situación específica grave.

**RF6.2.2.** Si el avión se encuentra durante 1500 milisegundos (3 lecturas consecutivas) en una velocidad del subrango [783.2, 812.4] se considerará como una velocidad baja, activando la alerta “Low Speed” e indicará una situación específica leve.

**RF6.2.3.** Si el avión se mantiene en el subrango [812.4, 841.6] se entenderá como un vuelo adecuado.

**RF6.2.4.** Si el avión se mantiene durante 1500 milisegundos (3 lecturas consecutivas) en el subrango [841.6, 870.8] será considerado como una velocidad alta, activando la alerta “High Speed” y la situación específica leve.

**RF6.2.5.** Si el avión se mantiene durante 1500 milisegundos (3 lecturas consecutivas) en el subrango [870.8, 900] será considerado como una velocidad demasiado alta, activando la alerta “Too High Speed” y la situación específica grave.

**RF6.2.6.** Será posible reducir el nivel de la situación específica, manteniéndose en los niveles ideales durante 1000 milisegundos (2 lecturas consecutivas), a velocidad de crucero.

**RF7.** El detector de proximidad por infrarrojos detectará si se encuentran presentes ambos pilotos.

**RF7.1.** Si uno de los detectores no detecta la presencia del piloto durante 1800 milisegundos (3 lecturas consecutivas), se activará la situación específica grave, con la alerta “Low Presence”.

**RF7.1.1.** Se reducirá el nivel de situación específica a leve o se desactivará si se detecta durante 1800 milisegundos (3 lecturas consecutivas) una de las siguientes condiciones.

**RF7.1.1.1.** Presencia de ambos pilotos por el detector de proximidad

**RF7.1.1.2.** Presencia adicional de un miembro de la tripulación por el sensor de movimiento.

**RF7.2.** Si la situación persiste durante los siguientes 1800 milisegundos (3 lecturas consecutivas), se activará la situación específica crítica.

**RF8.** El detector de movimientos detectará cualquier presencia o movimiento de la puerta a la cabina.

**RF8.1.** En caso de detectar la presencia adicional durante al menos 1800 milisegundos (3 lecturas consecutivas).

**RF8.1.1.** Si se encuentran ambos pilotos, indicará que hay más de dos personas en la cabina, activando la situación específica leve, con el mensaje “High Presence”.

**RF8.1.2.** Si uno de los pilotos no se detecta con el sensor de proximidad, indicará que hay 2 personas en la cabina, y no activará ninguna situación específica.

- RF8.1.2.1.** Es posible, que esa persona sea el piloto abandonando temporalmente la cabina.
- RF8.1.3.** Si la situación persiste durante otros 1800 milisegundos (3 lecturas consecutivas), se aumentará el nivel a grave.
- RF8.1.3.1.** Puede ser reducida, si se detecta únicamente 2 personas en la cabina durante 3 lecturas.
- RF9.** El sistema almacenará un registro en una tarjeta microSD donde se almacenará.
- RF9.1.** La velocidad del avión
- RF9.2.** El giro en los ejes X e Y
- RF9.3.** La altitud del avión.
- RF9.4.** El nivel de riesgo con el sensor asociado.
- RF9.5.** La situación del piloto automático
- RF9.6.** El instante donde se activa/desactiva el sistema.

## 4.4. Requisitos No Funcionales

### 4.4.1. Requisitos de Precisión

- RNFpre1.** La inclinación del acelerómetro,
- RNFpre1.1.** Vendrá indicada en grados angulares con valores entre -90 y 90 grados.
- RNFpre1.2.** El avión en su posición ideal mantendrá los ejes X (roll) e Y (pitch) en un valor cercano a 0 grados.
- RNFpre2.** La altitud tomada por el sensor de ultrasonidos
- RNFpre2.1.** Vendrá indicada en centímetros, al ser un prototipo con valores entre 2 y 250 cm.
- RNFpre2.2.** La altitud adecuada que respetará el avión es 2000 metros sobre cualquier terreno, por eso, la condición ideal del vuelo sería entre 10000 y 12000 metros. En la altitud utilizada, equivale a una altura entre 141 y 170 centímetros.
- RNFpre2.3.** La altitud ideal será 11000 metros, equivalente a 155 centímetros.
- RNFpre3.** La velocidad será tomada por medio de un potenciómetro.
- RNFpre3.1.** Vendrá indicada en un valor entero entre 0 y 65535, siendo estos valores correspondidos a una velocidad entre 754 y 900 km/h.
- RNFpre3.2.** La velocidad que será entendida como la indicada es la velocidad de crucero, a 827 km/h.
- RNFpre3.2.1.** Permitirá una variación entre los valores 812.4 y 841.6 km/h como velocidad adecuada.

**RNFpre4.** La presencia en la cabina será realizada a través de un sensor de proximidad

**RNFpre4.1.** Devolverá un valor binario, 0 no presente, 1 presente.

**RNFpre5.** La presencia de tripulación o presencia adicional en la cabina se realizará con un detector de movimiento.

**RNFpre5.1.** Devolverá un valor binario, 0 no hay movimiento, 1 hay movimiento.

#### 4.4.2. Requisitos de Disponibilidad

**RNFdis1.** El sistema estará disponible en pleno vuelo

**RNFdis1.1.** No se mantendrá activado durante el despegue y aterrizaje.

**RNFdis1.2.** Será activado cuando se alcance la altura mínima recomendada (10000 metros).

#### 4.4.3. Requisitos de Interoperabilidad

**RNFint1.** El sistema permitirá el uso y funcionamiento de otros sistemas similares de aviso y seguridad.

**RNFint2.** El sistema hará uso de una placa FPGA de Xilinx configurada para usar un sistema operativo de tiempo real (RTOS).

**RNFint3.** El acelerómetro se comunicará a través del protocolo I2C con el sistema.

#### 4.4.4. Requisitos de Modificabilidad

**RNFmod1.** El sistema podrá ser modificado funcionalmente usando C y un sistema operativo de tiempo real sobre una placa FPGA de Xilinx.

**RNFmod2.** El sistema podrá ser modificado para admitir dispositivos a través de herramientas compatibles con placas FPGA de Xilinx.

**RNFmod3.** El sistema podrá admitir más funcionalidad usando los puertos Pmod de la placa FPGA.

#### 4.4.5. Requisitos de Manejabilidad

**RNFmane1.** El sistema hará uso de una tarjeta microSD con los registros del vuelo.

**RNFmane1.1.** Podrá ser consultable tras el aterrizaje o desactivación del sistema.

#### 4.4.6. Requisitos de Portabilidad

**RNFport1.** El sistema podrá ser instalado en cualquier placa FPGA de Xilinx.

**RNFport1.1.** Con una arquitectura Zynq-7000.

**RNFport2.** El sistema podrá ser usado en dispositivos que permitan un sistema operativo de tiempo real.

#### 4.4.7. Requisitos de Rendimiento

**RNFrend1.** El acelerómetro tomará el giro cada 250 milisegundos.

**RNFrend2.** El altímetro o sensor de ultrasonidos medirá la altitud cada 300 milisegundos.

**RNFrend3.** El velocímetro o potenciómetro enviará la velocidad cada 500 milisegundos.

**RNFrend4.** El control de presencia se realizará cada 600 milisegundos.

**RNFrend5.** Se registrará la situación en la tarjeta microSD cada 5 segundos.

**RNFrend6.** El display será actualizado cada 2 segundos.

**RNFrend7.** Para indicar que el sistema está activo, parpadeará un led cada 2 segundos.

**RNFrend8.** Para detectar los posibles riesgos, se analizarán los síntomas cada 500 milisegundos.

**RNFrend9.** El pitido, en caso de ser activado, sonará durante 3 segundos.

**RNFrend10.** El estado de los diodos LED permanecerá activado de manera permanente hasta que el sistema se encuentre en la situación general normal.

**RNFrend11.** Se establecerá como plazo de “*deadline*” el valor del periodo de la tarea, al disponer estas de un breve tiempo de cómputo.

#### 4.4.8. Requisitos de Recuperación

**RNFrec1.** En caso de llenar el almacenamiento SD, se reescribirá el fichero con fecha más antigua.

**RNFrec2.** En caso de una posible catástrofe, los datos de la SD serán no volátiles y ésta estará asegurada correctamente.

**RNFrec3.** En caso posible y evitable, el sistema retomará la normalidad tras un indicio de catástrofe.

#### 4.4.9. Requisitos de Seguridad

**RNFseg1.** El sistema se activará o desactivará por medio del teclado numérico.

**RNFseg1.1.** El código introducido serán 6 caracteres hexadecimales (0-9, A-F).

**RNFseg2.** El sistema hará uso de un sistema de pilotaje automático estricto fiable implementado por terceros.

### 4.5. Requisitos de Interfaces Externas

#### 4.5.1. Requisitos de Interfaces de Usuario

**RIEusu1.** Se mostrará en el display la información de cada sensor y las alertas detectadas.

**RIEusu2.** Se hará uso de un teclado matricial para interactuar con el sistema, haciendo uso de los códigos introducidos por uno de los pilotos.

#### 4.5.2. Requisitos de Interfaces Hardware

**RIEhw1.** Se tomarán los ejes X e Y del acelerómetro que indican el roll (alabeo) del avión y el pitch (cabeceo) del avión, respectivamente.

**RIEhw2.** El sensor de ultrasonidos devuelve el tiempo en milisegundos para calcular la altitud o distancia.

**RIEhw3.** El sistema hará uso de una tarjeta microSD de 4 GB de almacenamiento para almacenar los registros.

**RIEhw4.** La potencia utilizada por el sistema son 5V.

**RIEhw4.1.** La salida de la placa utilizada por el sistema es de 3.3V.

**RIEhw4.2.** El sistema requerirá de una alimentación adicional de 5V para algunos sensores.

### 4.5.3. Requisitos de Interfaces de Comunicación

**RIEcom1.** La comunicación a la tarjeta microSD será por formato exFAT y escribirá un archivo “.txt”.

## 4.6. Restricciones de Desarrollo

**RD1.** El sistema será programado en lenguaje C.

**RD1.1.** Permitiendo resolver cualquier error de manera software.

**RD2.** El sistema hará uso de los módulos Pmod de Digilent.

**RD2.1.** Que se podrán sustituir o mejorar en caso necesario.

**RD3.** El sistema hará uso del sistema operativo de tiempo real.

**RD3.1.** Pudiendo ajustar los parámetros a través de él.

## 4.7. Requisitos de Entorno Físico

**REF1.** El sistema será funcional a una gran altura.

**REF2.** El sistema requerirá de una resistencia a cambios de presión.

**REF3.** El sistema resistirá cambios de temperatura.

**REF4.** El sistema será resistente a la humedad y al agua.

## 5. Casos de Uso

---

En este apartado, se van a detallar los casos de uso del sistema, con la finalidad de obtener una organización del sistema. Comenzando con un diagrama de clases o casos de uso, que se detallaran con sus casos de uso extendidos. Para poder designar tareas específicas a programar para el sistema.

### 5.1. Diagrama

En primer lugar, el sistema tendrá actuación por parte de diferentes actores, sobre todo del entorno y de los pilotos de ésta. También se incluyen actores menos relevantes, como la tripulación que accede a la cabina o el técnico que leerá la tarjeta microSD y hará un diagnóstico de la situación. En la siguiente figura (Figura 5) se puede ver el diagrama de casos de uso.

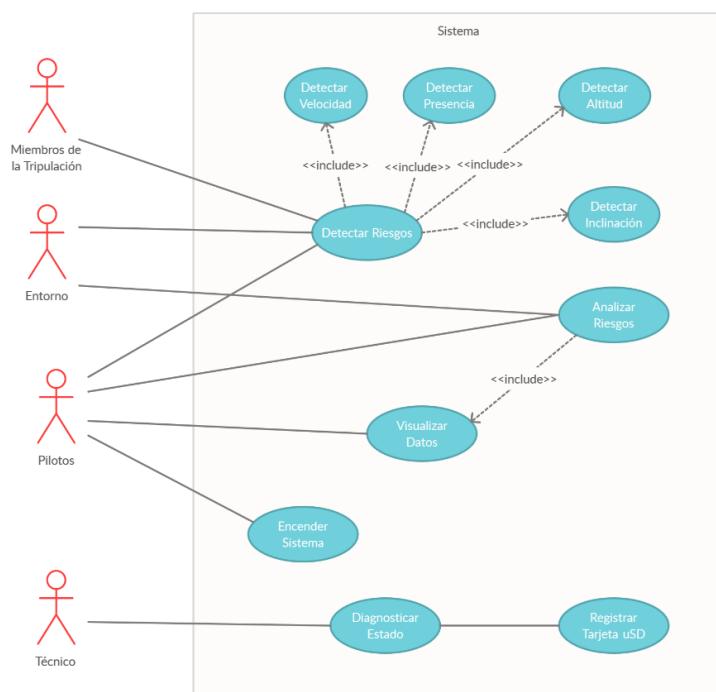


Figura 5: Diagrama de Casos de Uso

El entorno es utilizado por varios casos de uso, para obteniendo la inclinación, altitud y velocidad a la cual se encuentra el sistema. El caso de uso que más requiere de este actor es “Detectar Riesgos”. Su finalidad es analizar el entorno, la situación del avión y de los pilotos o tripulación para generar riesgos a evaluar. Estos pueden ser generados por el avión en los siguientes casos de uso: “Detectar Velocidad”, “Detectar Altitud” y “Detectar Inclinación”. Estas tareas medirán la velocidad, altitud e inclinación a la cual se encuentra el sistema, haciendo uso de sensores del sistema instalados en el avión.

De estos casos de uso, cada uno evaluará individualmente el riesgo generado por su sensor en tres niveles, leve, grave y crítico. El caso de uso “Detectar Riesgos”, recogerá esa información y la registrará en el sistema.

Otro de los casos de uso que requieren del entorno es “Analizar Riesgos”. Éste, en base a los niveles de riesgo generados por el caso de uso “Detectar Riesgos”, valorará la salida que debe ser realizada. Haciendo uso de los actuadores del sistema, también instalados en la cabina del avión.

Otros actores necesarios para el sistema son los pilotos y la tripulación, ambos interactúan con el sistema en el caso de uso “Detectar Riesgos” donde se almacena o recibe la salida de “Detectar Presencia”, este caso de uso no sólo obtiene la presencia del piloto en cabina, sino también de la tripulación que entre en la misma. Estos últimos serán identificados con el actor “Tripulación”.

Los pilotos también interactúan con el sistema en el caso de uso “Encender Sistema”, encargado de encender o apagar el sistema o el funcionamiento de éste. No se considerará que el sistema esté encendido o apagado hasta que no se reciba un código por parte de los pilotos.

Por último, los pilotos se relacionan con “Visualizar Datos” donde se muestra en un display la información recibida del avión o del sistema. Este actor también se encuentra asociado con el caso de uso “Analizar Riesgos” al recibir la respuesta generada por este último.

Finalmente, se encuentra el actor de “Técnico”, que no tiene que estar necesariamente en el sistema ni en su funcionamiento, pero es el encargado de leer la tarjeta microSD del sistema con sus registros tras el aterrizaje del avión, y realizar un diagnóstico de los datos. La tarjeta microSD es escrita durante el funcionamiento del avión por medio del caso de uso “Escribir en Tarjeta uSD”, este caso de uso preparará y escribirá en ficheros el registro del sistema mientras éste se encuentre activado. El caso de uso “Diagnosticar” establecerá qué pautas se deben revisar para valorar el correcto funcionamiento del sistema.

## 5.2. Extendidos

|                     |  |   |                    |  |  |
|---------------------|--|---|--------------------|--|--|
| ID:                 | CU-1   | Nombre:   | Detectar Velocidad |  |  |
| Creado por:         | Oskar Stepien  | Fecha de creación:  | 19/11/2020         |  |  |
| Actor primario:     | Entorno  | Actores secundarios:  |                    |  |  |
| <b>Descripción:</b> |  | Detectar la velocidad actual del avión usando un velocímetro. Valorar si la velocidad se encuentra entre los niveles establecidos cercanos a 827 km/h (velocidad de crucero).<br>Se permite una variación del 1,2%. Si es superior a este rango pero inferior al 5.3%, será un riesgo leve. Y si la velocidad supera ese umbral, será considerado como un riesgo grave. |                    |  |  |
| Evento activador:   | Encendido del sistema.   |   |                    |  |  |
| Precondiciones:     | El sistema debe estar encendido y el avión se encuentre en pleno vuelo.  |   |                    |  |  |
| Postcondiciones:    | Establecer un nivel de riesgo específico respecto a la velocidad.<br>Enviar una alerta con la información del nivel.   |   |                    |  |  |
| Flujo Normal:       | 1.1. 1.2. Leer el velocímetro del sistema.<br>1.3. 1.4. Comprobar que la velocidad leída no se encuentra entre 754 y 783.2 km/h.<br>1.5. 1.6. Comprobar que la velocidad leída no se encuentra entre 783.2 y 812.4 km/h.<br>1.7. 1.8. Comprobar que la velocidad leída se encuentra entre 812.4 y 841.6 km/h.<br>1.9. 1.10. Establecer el valor del contador de lecturas erróneas consecutivas a 0.<br>1.11. 1.12. Aumentar contador de lecturas correctas consecutivas.<br>1.13. 1.14. Comprobar que el contador de lecturas consecutivas es superior a 2.<br>1.15. 1.16. Volver a 1.1. |   |                    |  |  |
| Flujo Alternativo:  | 1.2.a. Velocidad en el rango [754; 783.2].<br>1.2.b. Asignar el contador de lecturas correctas consecutivas a 0.<br>1.2.c. Incrementar el contador de lecturas erróneas consecutivas.<br>1.2.d. Comprobar que el contador de lecturas erróneas es inferior a 3.<br>1.2.e. Volver a 1.1.  |   |                    |  |  |

|  |   |
|--|---|
|  | 1.2.d.a. Contador de lecturas erróneas consecutivas es 3 o superior.                                  |
|  | 1.2.d.b. Registra una situación específica de carácter “Grave”, indicando el valor “2” a la variable. |
|  | 1.2.d.c. Registrar o establecer la alerta “Too Low Speed”.  |
|  | 1.2.d.d. Volver a 1.1.  |
|  | 1.3.a. Velocidad en el rango [783; 812.4].  |
|  | 1.3.b. Asignar el contador de lecturas correctas consecutivas a 0.                                    |
|  | 1.3.c. Incrementar el contador de lecturas erróneas consecutivas.                                     |
|  | 1.3.d. Comprobar que el contador de lecturas erróneas es inferior a 3.                                |
|  | 1.3.e. Volver a 1.1.  |
|  | 1.3.d.a. Contador de lecturas erróneas consecutivas es 3 o superior.                                  |
|  | 1.3.d.b. Registra una situación específica de carácter “Leve”, indicando el valor “1” a la variable.  |
|  | 1.3.d.c. Registrar o establecer la alerta “Low Speed”.  |
|  | 1.3.d.d. Volver a 1.1.  |
|  | 1.4.a. Velocidad fuera del rango [812.4; 841.6].  |
|  | 1.4.b. Comprobar que la velocidad leída se encuentra entre 841.6 y 870.8 km/h.                        |
|  | 1.4.c. Asignar el contador de lecturas correctas consecutivas a 0.                                    |
|  | 1.4.d. Incrementar el contador de lecturas erróneas consecutivas.                                     |
|  | 1.4.e. Comprobar que el contador de lecturas erróneas es inferior a 3.                                |
|  | 1.4.f. Volver a 1.1.  |
|  | 1.4.e.a. Contador de lecturas erróneas consecutivas es 3 o superior.                                  |
|  | 1.4.e.b. Registra una situación específica de carácter “Leve”, indicando el valor “1” a la variable.  |
|  | 1.4.e.c. Registrar o establecer la alerta “High Speed”.   |
|  | 1.4.e.d. Volver a 1.1.  |
|  | 1.4.b.a. Velocidad fuera del rango [841.6; 870.8].  |
|  | 1.4.b.b. Comprobar que la velocidad leída se encuentra entre 870.8 y 900 km/h.                        |
|  | 1.4.b.c. Asignar el contador de lecturas correctas consecutivas a 0.                                  |
|  | 1.4.b.d. Incrementar el contador de lecturas erróneas consecutivas.                                   |
|  | 1.4.b.e. Comprobar que el contador de lecturas erróneas   |

## Casos de Uso - Extendidos

|                           |   |
|---------------------------|---|
|                           | <p>es inferior a 3.</p> <p>1.4.b.f. Volver a 1.1.</p> <p>1.4.b.e.a. Contador de lecturas erróneas consecutivas es 3 o superior.</p> <p>1.4.b.e.b. Registra una situación específica de carácter “Grave”, indicando el valor “2” a la variable.</p> <p>1.4.b.e.c. Registrar o establecer la alerta “Too High Speed”.</p> <p>1.4.b.e.d. Volver a 1.1.</p> <p>1.7.a. Comprobar si contador de lecturas correctas consecutivas es 1.</p> <p>1.7.b. Volver a 1.1.</p> <p>1.7.a.a. Comprobar si contador de lecturas consecutivas es 2.</p> <p>1.7.a.b. Comprobar si la situación específica actual es “Normal” o “0”.</p> <p>1.7.a.c. Volver a 1.1.</p> <p>1.7.a.b.a. La situación específica no es “Normal”.</p> <p>1.7.a.b.b. Comprobar que la situación actual es “Leve” o “1”.</p> <p>1.7.a.b.c. Reducir situación específica a “Normal” o “0”.</p> <p>1.7.a.b.d. Eliminar alerta o mensaje registrado.</p> <p>1.7.a.b.e. Volver a 1.1.</p> <p>1.7.a.b.b.a. La situación específica no es “Leve”.</p> <p>1.7.a.b.b.c. Comprobar que la situación actual es “Grave” o “2”.</p> <p>1.7.a.b.b.c. Reducir situación específica a “Leve” o “1”.</p> <p>1.7.a.b.b.d. Cambiar mensaje o alerta a “Rectified Speed”.</p> <p>1.7.a.b.b.e. Volver a 1.1.</p> |
| <b>Excepciones:</b>       | 1.1.a. Si el sensor no devuelve el valor o el dispositivo no funcione. Se toma en cuenta el valor más pequeño, en este caso 754 km/h.   |
| <b>Frecuencia de uso:</b> | La misma que el sistema.  |
| <b>Reglas de negocio:</b> | RN1, RN3  |
| <b>Otra información:</b>  | Si la situación general se encuentra en riesgo crítico, no será registrada ninguna situación específica, al estar activado el piloto automático.  |
| <b>Suposiciones:</b>      | Se toma la suposición que el sensor no dispone de fallos que devuelvan un valor erróneo, y sea detectado.   |

|                           |   |                           |                             |                    |
|---------------------------|---|---------------------------|-----------------------------|--------------------|
| <b>ID:</b>                | CU-2  |                           | <b>Nombre:</b>              | Detectar Presencia |
| <b>Creado por:</b>        | Oskar Stepien   | <b>Fecha de creación:</b> | 20/11/2020                  |                    |
| <b>Actor primario:</b>    | Piloto  |                           | <b>Actores secundarios:</b> | Tripulación        |
| <b>Descripción:</b>       | <p>Detectar la presencia actual en cabina con el uso de sensores de proximidad y un sensor de movimiento durante 3 lecturas consecutivas.</p> <p>Se detectará si hay menos de 2 personas, 2 personas o más en la cabina.</p>  |                           |                             |                    |
| <b>Evento activador:</b>  | Encendido del sistema.  |                           |                             |                    |
| <b>Precondiciones:</b>    | El sistema debe estar encendido y el avión se debe encontrar en pleno vuelo.  |                           |                             |                    |
| <b>Postcondiciones:</b>   | <p>Establecer un nivel de riesgo específico respecto a la presencia.</p> <p>Enviar una alerta con la información del nivel.</p>   |                           |                             |                    |
| <b>Flujo Normal:</b>      | <p>2.1. Leer sensores de proximidad de los pilotos.</p> <p>2.2. Leer sensor de movimiento.</p> <p>2.3. Comprobar qué sensores detectan presencia y acumularlo en una variable “presencia”.</p> <p>2.4. Comprobar que la presencia son 2 personas únicamente.</p> <p>2.5. Incrementar el valor del contador de lecturas correctas consecutivas.</p> <p>2.6. Establecer el valor del contador de lecturas erróneas consecutivas a 0.</p> <p>2.7. Comprobar que el contador de lecturas correctas es superior a 3.</p> <p>2.8. Volver a 2.1.</p>   |                           |                             |                    |
| <b>Flujo Alternativo:</b> | <p>2.4.a. Presencia detectada por debajo de “2”.</p> <p>2.4.b. Asignar el contador de lecturas correctas consecutivas a 0.</p> <p>2.4.c. Incrementar el contador de lecturas erróneas consecutivas.</p> <p>2.4.d. Comprobar que el contador de lecturas erróneas es inferior a 3.</p> <p>2.4.e. Volver a 2.1.</p> <p>2.4.d.a. Contador de lecturas erróneas consecutivas es 3 o superior.</p> <p>2.4.d.b. Comprobar que el contador de lecturas erróneas consecutivas es inferior a 6.</p> <p>2.4.d.c. Registra una situación específica de carácter “Grave”, indicando el valor “2” a la variable.</p> <p>2.4.d.d. Registrar o establecer la alerta “Low Presence”.</p> <p>2.4.d.e. Volver a 2.1.</p> <p>2.4.d.b.a Contador de lecturas erróneas consecutivas es 6.</p> <p>2.4.d.b.b. Registra una situación específica de carácter “Crítica”, indicando el valor “3” a la variable.</p> |                           |                             |                    |

## Casos de Uso - Extendidos

|                           |  |
|---------------------------|--|
|                           | 2.4.d.b.c. Volver a 2.1.   |
|                           | 2.4.a.a. Presencia detectada superior a “2”.   |
|                           | 2.4.a.b. Asignar el contador de lecturas correctas consecutivas a 0.   |
|                           | 2.4.a.c. Incrementar el contador de lecturas erróneas consecutivas.  |
|                           | 2.4.a.d. Comprobar que el contador de lecturas erróneas es inferior a 3.   |
|                           | 2.4.a.e. Volver a 2.1.   |
|                           | 2.4.a.d.a. Contador de lecturas erróneas consecutivas es 3 o superior.   |
|                           | 2.4.a.d.b. Comprobar que el contador de lecturas erróneas consecutivas es inferior a 6.                                      |
|                           | 2.4.a.d.c. Registra una situación específica de carácter “Leve”, indicando el valor “1” a la variable.                       |
|                           | 2.4.a.d.d. Registrar o establecer la alerta “High Presence”.   |
|                           | 2.4.a.d.e. Volver a 2.1.   |
|                           | 2.4.a.d.b.a Contador de lecturas erróneas consecutivas es 6.   |
|                           | 2.4.a.d.b.b. Registra una situación específica de carácter “Grave”, indicando el valor “2” a la variable.                    |
|                           | 2.4.a.d.b.c. Volver a 2.1.   |
|                           | 2.7.a. Comprobar si el contador de lecturas correctas consecutivas es inferior a 3 (no incluido).                            |
|                           | 2.7.b. Volver a 2.1.   |
|                           | 2.7.a.a. Lectura del contador de lecturas correctas es 3.  |
|                           | 2.7.a.b. Comprobar si la situación específica actual es “Normal” o “0”.  |
|                           | 2.7.a.c. Volver a 2.1.   |
|                           | 2.7.a.b.a. La situación específica no es “Normal”.   |
|                           | 2.7.a.b.b. Comprobar si la situación específica actual es “Leve” o “1”.  |
|                           | 2.7.a.b.c. Reducir situación específica a “Normal” o “0”.  |
|                           | 2.7.a.b.d. Eliminar alerta o mensaje registrado.   |
|                           | 2.7.a.b.e. Volver a 2.1.   |
|                           | 2.7.a.b.b.a. La situación específica no es “Leve”.   |
|                           | 2.7.a.b.b.b. Comprobar que la situación específica actual es “Grave” o “2”.  |
|                           | 2.7.a.b.b.c. Reducir situación específica a “Leve” o “1”.  |
|                           | 2.7.a.b.b.d. Cambiar mensaje o alerta a “Rectified Presence”.  |
|                           | 2.7.a.b.b.e. Volver a 2.1.   |
| <b>Excepciones:</b>       | 2.1.a. Si el sensor no devuelve el valor o no se detecta el dispositivo. Se tomará el valor predefinido de cada dispositivo. |
| <b>Frecuencia de uso:</b> | La misma que el sistema.   |

|                           |  |
|---------------------------|--|
| <b>Reglas de negocio:</b> | RN1, RN3   |
| <b>Otra información:</b>  | Si la situación general se encuentra en riesgo crítico, no será registrada ninguna situación específica, al estar activado el piloto automático. |
| <b>Suposiciones:</b>      | Se toma la suposición que el sensor no dispone de fallos que devuelvan un valor erróneo, y sea detectado.  |

|                           |  |                             |                  |
|---------------------------|--|-----------------------------|------------------|
| <b>ID:</b>                | CU-3   | <b>Nombre:</b>              | Detectar Altitud |
| <b>Creado por:</b>        | Oskar Stepien  | <b>Fecha de creación:</b>   | 20/11/2020       |
| <b>Actor primario:</b>    | Entorno  | <b>Actores secundarios:</b> |                  |
| <b>Descripción:</b>       | Detectar la altitud actual con el uso de un sensor de altitud tras 3 lecturas consecutivas.<br>Se detectará si se encuentra en el rango de 10000 y 12000 metros.   |                             |                  |
| <b>Evento activador:</b>  | Encendido del sistema.   |                             |                  |
| <b>Precondiciones:</b>    | El sistema debe estar encendido y el avión se encuentre en pleno vuelo.  |                             |                  |
| <b>Postcondiciones:</b>   | Establecer un nivel de riesgo específico respecto a la altitud.<br>Enviar una alerta con la información del nivel.   |                             |                  |
| <b>Flujo Normal:</b>      | 3.1. Leer el sensor de altitud.<br>3.2. Comprobar que la altitud se encuentra es superior a 10000 metros<br>3.3. Comprobar que la altitud es inferior a 12000 metros.<br>3.4. Incrementar el valor del contador de lecturas correctas consecutivas.<br>3.5. Establecer el valor del contador de lecturas erróneas consecutivas a 0.<br>3.6. Comprobar que el contador de lecturas correctas es superior a 3.<br>3.7. Volver a 3.1.   |                             |                  |
| <b>Flujo Alternativo:</b> | 3.2.a. Altitud por debajo de 10000 metros.<br>3.2.b. Asignar el contador de lecturas correctas consecutivas a 0.<br>3.2.c. Incrementar el contador de lecturas erróneas consecutivas.<br>3.2.d. Comprobar que el contador de lecturas erróneas es inferior a 3.<br>3.2.e. Volver a 3.1.<br><br>3.2.d.a. El contador de lecturas erróneas es 3.<br>3.2.d.b. Comprobar que el contador de lecturas erróneas consecutivas es inferior a 6.<br>3.2.d.c. Registra una situación específica de carácter “Leve”, indicando el valor “1” a la variable.<br>3.2.d.d. Registrar o establecer la alerta “Low Altitude”. |                             |                  |

## Casos de Uso - Extendidos

|                     |   |
|---------------------|---|
|                     | <p>3.2.d.e. Volver a 3.1.</p> <p>3.2.d.b.a Contador de lecturas erróneas consecutivas es 6.</p> <p>3.2.d.b.b. Registra una situación específica de carácter “Grave”, indicando el valor “2” a la variable.</p> <p>3.2.d.b.c. Volver a 3.1.</p> <p>3.3.a. Altitud por encima de 12000 metros.</p> <p>3.3.b. Asignar el contador de lecturas correctas consecutivas a 0.</p> <p>3.3.c. Incrementar el contador de lecturas erróneas consecutivas.</p> <p>3.3.d. Comprobar que el contador de lecturas erróneas es inferior a 3.</p> <p>3.3.e. Volver a 3.1.</p> <p>3.3.d.a. El contador de lecturas erróneas es 3.</p> <p>3.3.d.b. Comprobar que el contador de lecturas erróneas consecutivas es inferior a 6.</p> <p>3.3.d.c. Registra una situación específica de carácter “Grave”, indicando el valor “2” a la variable.</p> <p>3.3.d.d. Registrar o establecer la alerta “High Altitude”.</p> <p>3.3.d.e. Volver a 3.1.</p> <p>3.3.d.b.a Contador de lecturas erróneas consecutivas es 6.</p> <p>3.3.d.b.b. Registra una situación específica de carácter “Crítico”, indicando el valor “3” a la variable.</p> <p>3.3.d.b.c. Volver a 3.1.</p> <p>3.6.a. Comprobar si el contador de lecturas correctas consecutivas es inferior a 3 (no incluido).</p> <p>3.6.b. Volver a 3.1.</p> <p>3.6.a.a. Lectura del contador de lecturas correctas es 3.</p> <p>3.6.a.b. Comprobar si la situación específica actual es “Normal” o “0”.</p> <p>3.6.a.c. Volver a 3.1.</p> <p>3.6.a.b.a. La situación específica no es “Normal”.</p> <p>3.6.a.b.b. Comprobar si la situación específica actual es “Leve” o “1”.</p> <p>3.6.a.b.c. Reducir situación específica a “Normal” o “0”.</p> <p>3.6.a.b.d. Eliminar alerta o mensaje registrado.</p> <p>3.6.a.b.e. Volver a 3.1.</p> <p>3.6.a.b.b.a. La situación específica no es “Leve”.</p> <p>3.6.a.b.b.b. Comprobar que la situación específica actual es “Grave” o “2”.</p> <p>3.6.a.b.b.c. Reducir situación específica a “Leve” o “1”.</p> <p>3.6.a.b.b.d. Cambiar mensaje o alerta a “Rectified Altitude”.</p> <p>3.6.a.b.b.e. Volver a 3.1.</p> |
| <b>Excepciones:</b> | <p>3.1.a. Si el sensor no devuelve el valor. Mostrar “Error” por el display y registrar en la tarjeta USD, con su “timestamp”</p>   |

|                           |   |
|---------------------------|---|
|                           | respectivo.   |
| <b>Frecuencia de uso:</b> | La misma que el sistema.  |
| <b>Reglas de negocio:</b> | RN1, RN3  |
| <b>Otra información:</b>  | <p>El sistema dispondrá de valores modificables para emular la altitud del sistema, aplicando una escala al valor obtenido.</p> <p>Si la situación general se encuentra en riesgo crítico, no será registrada ninguna situación específica, al estar activado el piloto automático.</p> |
| <b>Suposiciones:</b>      | Se toma la suposición que el sensor no dispone de fallos que devuelvan un valor erróneo, y sea detectado.   |

|                           |  |                             |                      |
|---------------------------|--|-----------------------------|----------------------|
| <b>ID:</b>                | CU-4   | <b>Nombre:</b>              | Detectar Inclinación |
| <b>Creado por:</b>        | Oskar Stepien  | <b>Fecha de creación:</b>   | 20/11/2020           |
| <b>Actor primario:</b>    | Entorno  | <b>Actores secundarios:</b> |                      |
| <b>Descripción:</b>       | Detectar la inclinación actual con el uso de un acelerómetro tras 3 lecturas consecutivas.<br>Se indicará si se encuentra el eje X entre -30 y 30 grados. Y el eje Y entre -20 y 20 grados.  |                             |                      |
| <b>Evento activador:</b>  | Encendido del sistema.   |                             |                      |
| <b>Precondiciones:</b>    | El sistema debe estar encendido y el avión se encuentre en pleno vuelo.  |                             |                      |
| <b>Postcondiciones:</b>   | Establecer un nivel de riesgo específico respecto a la inclinación.<br>Enviar una alerta con la información del nivel.   |                             |                      |
| <b>Flujo Normal:</b>      | 4.1. Leer el acelerómetro.<br>4.2. Comprobar que el valor del eje X se encuentra entre -30 y 30 grados.<br>4.3. Asignar el valor "0" a la variable "AxisX".<br>4.4. Comprobar que el valor del eje Y se encuentra entre -20 y 20 grados.<br>4.5. Asignar el valor "0" a la variable "AxisY"<br>4.6. Comprobar que tanto "AxisX" como "AxisY" tienen el valor "0".<br>4.7. Incrementar el valor del contador de lecturas correctas consecutivas.<br>4.8. Comprobar que el contador de lecturas correctas es superior a 2.<br>4.9. Volver a 4.1. |                             |                      |
| <b>Flujo Alternativo:</b> | 4.2.a. Inclinación en el eje X no está en el rango [-30; 30].<br>4.2.b. Incrementar en 1 el valor de la variable "AxisX".<br>4.2.c. Volver a 4.4.<br>4.4.a. Inclinación en el eje Y no está en el rango [-20; 20].   |                             |                      |

## Casos de Uso - Extendidos

|   |
|---|
|   |
| 4.4.b. Incrementar en 1 el valor de la variable “AxisY”.  |
| 4.4.c. Volver a 4.6.  |
| 4.6.a. “AxisX” y/o “AxisY” no son “0”.  |
| 4.6.b. Establecer el valor del contador de lecturas correctas consecutivas a 0.                           |
| 4.6.c. Comprobar que el valor de “AxisX” y “AxisY” es inferior a 3.                                       |
| 4.6.d. Comprobar que el valor de “AxisX” es inferior a 3.   |
| 4.6.e. Comprobar que el valor de “AxisY” es inferior a 3.   |
| 4.6.f. Volver a 4.1.  |
| 4.6.c.a. “AxisX” y “AxisY” son 3 o más cada uno.  |
| 4.6.c.b. Comprobar que el valor de cada uno, “AxisX” y “AxisY” es inferior a 5.                           |
| 4.6.c.c. Registra una situación específica de carácter “Grave”, indicando el valor “2” a la variable.     |
| 4.6.c.d. Registrar la alerta “High Pitch and Roll Angle”.   |
| 4.6.c.e. Volver a 4.1.  |
| 4.6.c.b.a. “AxisX” y “AxisY” son 5 o más cada uno.  |
| 4.6.c.b.b. Registra una situación específica de carácter “Crítico”, indicando el valor “3” a la variable. |
| 4.6.c.b.c. Volver a 4.1.  |
| 4.6.d.a. “AxisX” es 3 o más   |
| 4.6.d.b. Comprobar que el valor de “AxisX” es inferior a 5.   |
| 4.6.d.c. Registra una situación específica de carácter “Leve”, indicando el valor “1” a la variable.      |
| 4.6.d.d. Registrar la alerta “High Roll Angle”.   |
| 4.6.d.e. Volver a 4.1.  |
| 4.6.d.b.a. “AxisX” es 5 o más.  |
| 4.6.d.b.b. Registra una situación específica de carácter “Grave”, indicando el valor “2” a la variable.   |
| 4.6.d.b.c. Volver a 4.1.  |
| 4.6.e.a. “AxisY” es 3 o más   |
| 4.6.e.b. Comprobar que el valor de “AxisY” es inferior a 5.   |
| 4.6.e.c. Registra una situación específica de carácter “Leve”, indicando el valor “1” a la variable.      |
| 4.6.e.d. Registrar la alerta “High Pitch Angle”.  |
| 4.6.e.e. Volver a 4.1.  |
| 4.6.e.b.a. “AxisY” es 5 o más.  |
| 4.6.e.b.b. Registra una situación específica de carácter “Grave”, indicando el valor “2” a la variable.   |
| 4.6.e.b.c. Volver a 4.1.  |
| 4.8.a. Comprobar si el contador de lecturas correctas   |

|                           |  |
|---------------------------|--|
|                           | consecutivas es 1.   |
|                           | 4.8.b. Volver a 4.1.   |
|                           | 4.8.a.a. Lectura del contador de lecturas correctas es 2.  |
|                           | 4.8.a.b. Comprobar si la situación específica actual es "Normal" o "0".  |
|                           | 4.8.a.c. Volver a 4.1.   |
|                           | 4.8.a.b.a. La situación específica no es "Normal".   |
|                           | 4.8.a.b.b. Comprobar si la situación específica actual es "Leve" o "1".  |
|                           | 4.8.a.b.c. Reducir situación específica a "Normal" o "0".  |
|                           | 4.8.a.b.d. Eliminar alerta o mensaje registrado.   |
|                           | 4.8.a.b.e. Volver a 4.1.   |
|                           | 4.8.a.b.b.a. La situación específica no es "Leve".   |
|                           | 4.8.a.b.b.b. Comprobar que la situación específica actual es "Grave" o "2".  |
|                           | 4.8.a.b.b.c. Reducir situación específica a "Leve" o "1".  |
|                           | 4.8.a.b.b.d. Cambiar mensaje o alerta a "Rectified Angle".   |
|                           | 4.8.a.b.b.e. Volver a 4.1.   |
| <b>Excepciones:</b>       | 4.1.a. Si el sensor no devuelve el valor. Mostrar "Error" por el display y registrar en la tarjeta uSD, con su "timestamp" respectivo.           |
| <b>Frecuencia de uso:</b> | La misma que el sistema.   |
| <b>Reglas de negocio:</b> | RN1, RN3   |
| <b>Otra información:</b>  | Si la situación general se encuentra en riesgo crítico, no será registrada ninguna situación específica. Al estar activado el piloto automático. |
| <b>Suposiciones:</b>      | Se toma la suposición que el sensor no dispone de fallos que devuelvan un valor erróneo, y sea detectado.  |

|                          |  |                             |                      |
|--------------------------|--|-----------------------------|----------------------|
| <b>ID:</b>               | CU-5   | <b>Nombre:</b>              | Detectar Riesgos     |
| <b>Creado por:</b>       | Oskar Stepien  | <b>Fecha de creación:</b>   | 20/11/2020           |
| <b>Actor primario:</b>   | Entorno  | <b>Actores secundarios:</b> | Piloto y Tripulación |
| <b>Descripción:</b>      | Detectar los posibles riesgos y su nivel con el uso de diversos sensores.  |                             |                      |
| <b>Evento activador:</b> | Encendido del sistema.   |                             |                      |
| <b>Precondiciones:</b>   | El sistema debe estar encendido y el avión se encuentre en pleno vuelo.    |                             |                      |
| <b>Postcondiciones:</b>  | Activar señales, alertas e indicadores para el funcionamiento del sistema  |                             |                      |
| <b>Flujo Normal:</b>     | 5.1. Activar rutina de casos de uso de detección (CU-1, CU-2, CU-3, CU-4). |                             |                      |

## Casos de Uso - Extendidos

|                           |   |
|---------------------------|---|
|                           | <p>5.2. Leer salidas de los sistemas de distracciones.</p> <p>5.3. Transmitir las señales de salida y estados a otros componentes como Analizar Riesgos, Mostrar Datos, Registrar en Tarjeta uSD.</p>   |
| <b>Frecuencia de uso:</b> | La misma que el sistema.  |
| <b>Reglas de negocio:</b> | RN1, RN2, RN3   |
| <b>Suposiciones:</b>      | <p>Se toma la suposición que los sensores no disponen de fallos que devuelvan un valor erróneo, y sean detectado.</p> <p>También se toma la suposición de registrar con marca de tiempo o “timestamp” todos los mensajes por display y el registro en la uSD.</p> |

|                          |  |  |                             |                  |
|--------------------------|--|--|-----------------------------|------------------|
| <b>ID:</b>               | CU-6   |  | <b>Nombre:</b>              | Analizar Riesgos |
| <b>Creado por:</b>       | Oskar Stepien  |  | <b>Fecha de creación:</b>   | 20/11/2020       |
| <b>Actor primario:</b>   | Entorno  |  | <b>Actores secundarios:</b> |                  |
| <b>Descripción:</b>      | Detectar las situaciones específicas de los diferentes sensores y obtener una situación general del sistema.   |  |                             |                  |
| <b>Evento activador:</b> | Encendido del sistema.   |  |                             |                  |
| <b>Precondiciones:</b>   | El sistema debe estar encendido y el avión se encuentre en pleno vuelo.  |  |                             |                  |
| <b>Postcondiciones:</b>  | <p>Establecer un nivel de riesgo general respecto a los riesgos específicos.</p> <p>Enviar una respuesta en función del nivel de riesgo general asignado al sistema.</p>   |  |                             |                  |
| <b>Flujo Normal:</b>     | <p>6.1. Leer riesgos específicos de los sensores, haciendo uso del caso de uso CU-5, asignándolos a un array de enteros.</p> <p>6.2. Inicializar los contadores por cada nivel de riesgo al valor “0” (leve, grave y crítico).</p> <p>6.3. Incrementar por cada riesgo el contador del mismo nivel de riesgo. (Por cada sensor, hasta 4)</p> <p>6.4. Establecer el nivel de situación general interna a “0”.</p> <p>6.5. Comprobar que el contador de riesgo leve tiene el valor 0.</p> <p>6.6. Comprobar que el contador de riesgo grave tiene el valor 0.</p> <p>6.7. Comprobar que el contador de riesgo crítico tiene el valor 0.</p> <p>6.8. Comprobar que el valor de la situación de riesgo general interno es igual a “0”.</p> <p>6.9. Incrementar el tiempo del contador de tiempo correcto consecutivo.</p> <p>6.10. Comprobar que el contador de tiempo correctas es superior a 5 segundos.</p> |  |                             |                  |

|                           |  |
|---------------------------|--|
|                           | <p>6.11. Realizar respuesta asociada si el nivel de situación general es “0” o “Normal”.</p> <p>6.12. Volver a 6.1.</p>  |
| <b>Flujo Alternativo:</b> | <p>6.5.a. Contador de riesgo leve no es “0”.</p> <p>6.5.b. Comprobar que el contador de riesgo leve es inferior a 3.</p> <p>6.5.c. Registra una situación general de carácter “Leve”, indicando el valor “1” a la variable.</p> <p>6.5.d. Iluminar en el módulo PmodLED los diodos en función del valor del contador (1 ó 2).</p> <p>6.5.e. Volver a 6.5.</p> <p>6.5.b.a. El contador de riesgo leve es 3 o superior.</p> <p>6.5.b.b. Incrementar el contador de riesgo grave en 1.</p> <p>6.5.b.c. Volver a 6.5.</p> <p>6.6.a. Contador de riesgo grave no es “0”.</p> <p>6.6.b. Comprobar que el contador de riesgo grave es inferior a 2.</p> <p>6.6.c. Registra una situación general de carácter “Grave”, indicando el valor “2” a la variable.</p> <p>6.6.d. Iluminar en el módulo PmodLED 3 diodos.</p> <p>6.6.e. Volver a 6.6.</p> <p>6.6.b.a. El contador de riesgo grave es 2 o superior.</p> <p>6.6.b.b. Incrementar el contador de riesgo crítico en 1.</p> <p>6.6.b.c. Volver a 6.6.</p> <p>6.7.a. Contador de riesgo crítico no es “0”.</p> <p>6.7.b. Registra una situación general de carácter “Crítico”, indicando el valor “3” a la variable.</p> <p>6.7.c. Iluminar en el módulo PmodLED 4 diodos.</p> <p>6.7.d. Volver a 6.8.</p> <p>6.8.a. Si el nivel de riesgo general interno no es “0”, reiniciar el contador de lecturas correctas consecutivas a 0.</p> <p>6.8.b. Comprobar si el nivel de riesgo interno es “1” o “Leve”.</p> <p>6.8.c. Establecer el nivel de riesgo general externo a “1” o “Leve”.</p> <p>6.8.d. Realizar respuesta asociada al nivel “1” o “Leve”.</p> <p>6.8.e. Volver a 6.1.</p> <p>6.8.b.a. Si el nivel de riesgo general no es “1”.</p> <p>6.8.b.b. Comprobar si el nivel de riesgo es “2” o “Grave”.</p> <p>6.8.b.c. Establecer el nivel de riesgo general externo a “2” o “Grave”.</p> <p>6.8.b.d. Realizar respuesta asociada al nivel “2” o “Grave”.</p> <p>6.8.b.e. Volver a 6.1.</p> <p>6.8.b.b.a. Si el nivel de riesgo general no es “2”.</p> <p>6.8.b.b.b. Comprobar si el nivel de riesgo es “3” o “Crítico”.</p> <p>6.8.b.b.c. Establecer el nivel de riesgo general externo a “3” o “Crítico”.</p> |

|                           |   |
|---------------------------|---|
|                           | <p>6.8.b.b.d. Realizar respuesta asociada al nivel “3” o “Crítico”.</p> <p>6.8.b.b.e. Volver a 6.1.</p> <p>6.10.a. Si el temporizador o contador de lecturas correctas consecutivas es menor a 5 segundos.</p> <p>6.10.b. Volver a 6.1.</p> <p>6.10.a.a. Si el temporizador o contador de lecturas correctas consecutivas es 5.</p> <p>6.10.a.b. Comprobar si la situación general actual es “Normal” o “0”.</p> <p>6.10.a.c. Volver a 6.1.</p> <p>6.10.a.b.a. La situación general no es “Normal”.</p> <p>6.10.a.b.b. Comprobar si la situación general actual es “Leve” o “1”.</p> <p>6.10.a.b.c. Reducir situación general a “Normal” o “0”.</p> <p>6.10.a.b.d. Eliminar alertas o mensajes registrados.</p> <p>6.10.a.b.e. Volver a 6.1.</p> <p>6.10.a.b.b.a. La situación general no es “Leve”.</p> <p>6.10.a.b.b.b. Comprobar que la situación general actual es “Grave” o “2”.</p> <p>6.10.a.b.b.c. Reducir situación general a “Leve” o “1”.</p> <p>6.10.a.b.b.d. Realizar respuesta asociada al nivel “1” o “Leve”.</p> <p>6.10.a.b.b.e. Volver a 6.1.</p> |
| <b>Frecuencia de uso:</b> | La misma que el sistema.  |
| <b>Reglas de negocio:</b> | RN1, RN2, RN3, RN6.   |
| <b>Otra información:</b>  | <p>El caso de uso tiene acciones adicionales que se han resumido como “respuestas asociadas” que utilizan los actuadores del sistema, incluyendo un mensaje por display y registro en la tarjeta uSD.</p> <p>Respuestas Asociadas:</p> <p>Nivel 0:</p> <ul style="list-style-type: none"> <li>- Diodo LED RGB apagado.</li> <li>- Zumbador apagado.</li> </ul> <p>Nivel 1:</p> <ul style="list-style-type: none"> <li>- Diodo LED RGB azul hasta su apagado.</li> <li>- Zumbador en nivel 1 durante 3 segundos.</li> </ul> <p>Nivel 2:</p> <ul style="list-style-type: none"> <li>- Diodo LED RGB amarillo hasta su apagado.</li> <li>- Zumbador en nivel 2 durante 3 segundos.</li> </ul> <p>Nivel 3:</p> <ul style="list-style-type: none"> <li>- Diodo LED RGB rojo hasta su apagado.</li> <li>- Zumbador en nivel 3 durante 3 segundos.</li> <li>- Activado del piloto automático exclusivo, desactivado a los 15 segundos.</li> </ul>  |
| <b>Suposiciones:</b>      | Se toma la suposición de registrar cada activación y cambio con   |

|  |  |
|--|--|
|  | marca de tiempo o “timestamp” por display y el registro en la uSD. |
|--|--|

|                           |   |                             |                |                  |  |  |
|---------------------------|---|-----------------------------|----------------|------------------|--|--|
| <b>ID:</b>                | CU-7  |                             | <b>Nombre:</b> | Visualizar Datos |  |  |
| <b>Creado por:</b>        | Oskar Stepien   | <b>Fecha de creación:</b>   | 23/11/2020     |                  |  |  |
| <b>Actor primario:</b>    | Pilotos   | <b>Actores secundarios:</b> |                |                  |  |  |
| <b>Descripción:</b>       | Mostrar la información del sistema por medio de un display.   |                             |                |                  |  |  |
| <b>Evento activador:</b>  | Encendido del sistema   |                             |                |                  |  |  |
| <b>Precondiciones:</b>    | El sistema debe estar encendido.  |                             |                |                  |  |  |
| <b>Postcondiciones:</b>   | Enviar al display la información más relevante del sistema para la lectura de los pilotos.  |                             |                |                  |  |  |
| <b>Flujo Normal:</b>      | 7.1. Leer los datos almacenados de los sensores (altitud, inclinación, velocidad, presencia).<br>7.2. Leer alertas almacenadas en el sistema.<br>7.3. Leer el nivel de situación general almacenado.<br>7.4. Mostrar por display los datos de los sensores.<br>7.5. Mostrar por display el nivel de la situación general con su equivalente en caracteres (“Normal”, “Leve”, “Grave” o “Crítico”).<br>7.6. Mostrar las alertas activas o la situación de estas (en caso de ser una alerta rectificada).<br>7.7. Volver a 7.1. |                             |                |                  |  |  |
| <b>Flujo Alternativo:</b> | 7.4.a. Si se detecta que se encuentra activado el modo “Debug”, mostrar el nivel de las situaciones específicas de cada sensor.   |                             |                |                  |  |  |
| <b>Excepciones:</b>       | 7.2.a. Pueden no existir alertas, estas se omitirán en el display en esa situación.<br>7.3.a. Puede no existir un nivel de situación general, en ese caso, se omitirá en el display.  |                             |                |                  |  |  |
| <b>Frecuencia de uso:</b> | Cada 2 segundos.  |                             |                |                  |  |  |
| <b>Reglas de negocio:</b> | RN1   |                             |                |                  |  |  |
| <b>Suposiciones:</b>      | Se indicará con marca de tiempo o “timestamp” la activación del display, y se registrará en la tarjeta microSD.   |                             |                |                  |  |  |

|                        |   |                             |                |                  |
|------------------------|---|-----------------------------|----------------|------------------|
| <b>ID:</b>             | CU-8  |                             | <b>Nombre:</b> | Utilizar Sistema |
| <b>Creado por:</b>     | Oskar Stepien   | <b>Fecha de creación:</b>   | 23/11/2020     |                  |
| <b>Actor primario:</b> | Pilotos   | <b>Actores secundarios:</b> |                |                  |
| <b>Descripción:</b>    | Permite activar o desactivar funciones del sistema mediante un código numérico introducido manualmente por uno de los pilotos. También, se encarga de encender el sistema como tal. |                             |                |                  |

|                           |  |
|---------------------------|--|
| <b>Evento activador:</b>  |  |
| <b>Precondiciones:</b>    | El avión se encuentra en pleno vuelo.  |
| <b>Postcondiciones:</b>   | Encender o apagar el sistema. O activar la función deseada.  |
| <b>Flujo Normal:</b>      | <p>8.1. Esperar a obtener un código de 6 dígitos de la función del teclado matricial.</p> <p>8.2. Comprobar el código recogido con el código de activación o desactivación de alguna función del sistema.</p> <p>8.3. Realizar la acción indicada con el código obtenido.</p> <p>8.4. Volver a 8.1.</p>  |
| <b>Flujo Alternativo:</b> | <p>8.1.a. Si no se recibe y el sistema está apagado.</p> <p>8.1.b. Volver a 8.1.</p> <p>8.1.a.a. Si no se recibe entrada y el sistema está encendido.</p> <p>8.1.a.b. Encender el diodo LED de la placa FPGA.</p> <p>8.1.a.c. Apagar el diodo LED de la placa FPGA.</p> <p>8.1.a.d. Volver a 8.1.</p> <p>8.3.a. Si el código no es correcto.</p> <p>8.3.b. Enviar mensaje por display y registrar en tarjeta microSD “Wrong Code!”</p> <p>8.3.c. Volver a 8.1.</p> |
| <b>Frecuencia de uso:</b> | Cada 2 segundos.   |
| <b>Reglas de negocio:</b> | RN1  |
| <b>Suposiciones:</b>      | <p>Se realiza la suposición que el teclado matricial no tiene ningún error de ejecución.</p> <p>Se toma la suposición de registrar cada activación y desactivación con marca de tiempo o “timestamp” por display y en el registro en la microSD.</p>   |

|                          |   |                             |                          |
|--------------------------|---|-----------------------------|--------------------------|
| <b>ID:</b>               | CU-9  | <b>Nombre:</b>              | Registrar en tarjeta uSD |
| <b>Creado por:</b>       | Oskar Stepien   | <b>Fecha de creación:</b>   | 23/11/2020               |
| <b>Actor primario:</b>   |   | <b>Actores secundarios:</b> |                          |
| <b>Descripción:</b>      | Registrar la información del sistema en un fichero .txt de la tarjeta microSD de éste, con su marca de tiempo o “timestamp”.  |                             |                          |
| <b>Evento activador:</b> | Encendido del sistema.  |                             |                          |
| <b>Precondiciones:</b>   | El sistema debe estar encendido.  |                             |                          |
| <b>Postcondiciones:</b>  | Escribir en la tarjeta microSD las acciones detectadas o activadas con su “timestamp”, y mostrar la información del sistema completa periódicamente   |                             |                          |
| <b>Flujo Normal:</b>     | <p>9.1. Leer los datos almacenados de los sensores (altitud, inclinación, velocidad, presencia).</p> <p>9.2. Leer alertas almacenadas en el sistema.</p> <p>9.3. Leer el nivel de situación general almacenado.</p> |                             |                          |

|                           |   |
|---------------------------|---|
|                           | <p>9.4. Registrar en el fichero los datos de los sensores.</p> <p>9.5. Registrar en el fichero el nivel de la situación general con su equivalente en caracteres (“Normal”, “Leve”, “Grave” o “Crítico”).</p> <p>9.6. Registrar las alertas activas o la situación de estas (en caso de ser una alerta rectificada).</p> <p>9.7. Volver a 8.1.</p>  |
| <b>Flujo Alternativo:</b> |   |
| <b>Excepciones:</b>       | <p>9.1.a. Pueden no existir datos almacenados, en esa situación, se devolverá el array “NaN” (<i>Not a Number</i>), para indicar la no existencia de datos numéricos.</p> <p>9.2.a. Pueden no existir alertas, estas se registrarán con “No Alerts Registered”.</p> <p>9.3.a. Puede no existir un nivel de situación general, en ese caso, se registrará con “No Risk Situation or Not Registered”.</p> |
| <b>Frecuencia de uso:</b> | Cada 5 segundos.  |
| <b>Reglas de negocio:</b> | RN1, RN4  |
| <b>Otra información:</b>  | El registro en la tarjeta microSD se realizará cuando aparezca la situación para identificar el momento exacto, también se indicará con los datos relevantes para cada alerta.  |

|                          |   |                             |              |
|--------------------------|---|-----------------------------|--------------|
| <b>ID:</b>               | CU-10   | <b>Nombre:</b>              | Diagnosticar |
| <b>Creado por:</b>       | Oskar Stepien   | <b>Fecha de creación:</b>   | 23/11/2020   |
| <b>Actor primario:</b>   | Técnico   | <b>Actores secundarios:</b> |              |
| <b>Descripción:</b>      | Realizar o mostrar un diagnóstico del sistema por medio de las pautas indicadas.<br>Es el único caso de uso que el sistema no puede realizar.   |                             |              |
| <b>Evento activador:</b> | Lectura de la tarjeta microSD.  |                             |              |
| <b>Precondiciones:</b>   | El sistema debe estar apagado o desactivado.  |                             |              |
| <b>Postcondiciones:</b>  | Confirmar el funcionamiento del sistema y garantizar su veracidad por medio de un análisis del técnico.   |                             |              |
| <b>Flujo Normal:</b>     | <p>10.1. Extraer la tarjeta microSD del sistema e introducirla a un equipo informático.</p> <p>10.2. Comprobar que el sistema se ha activado y desactivado correctamente con el código indicado (al principio y final del fichero). En caso de existir varios, el fichero más reciente.</p> <p>10.3. Verificar que la existencia de alertas es correcta y justificada. Comprobando su nivel de situación corresponde al indicado con la información del sistema.</p> <p>10.4. Comprobar que el tiempo de activación entre una situación específica y una general es menor a 500 milisegundos.</p> |                             |              |

## Casos de Uso - Extendidos

|                           |   |
|---------------------------|---|
|                           | 10.5. Comprobar que los actuadores han sido activados y desactivados correctamente con la información del sistema en cada momento.  |
|                           | 10.6. Verificar que el plazo de activación y desactivación de las mismas tareas activadas no se solapan.  |
|                           | 10.7. Comprobar que se ha reducido la situación específica de cada sensor tras las lecturas indicadas.  |
|                           | 10.8. Comprobar que la reducción de la situación general se ha realizado en un plazo de 5 segundos respecto al cambio de la situación específica.                               |
|                           | 10.9. Verificar la activación y desactivación de una situación general de riesgo crítico, 15 segundos con piloto automático sin obstaculizar las lecturas y escrituras debidas. |
|                           | 10.10. Comprobar que la desactivación del sistema se ha realizado sin situaciones generales activadas.  |
| <b>Reglas de negocio:</b> | RN4.  |
| <b>Suposiciones:</b>      | Se toma la suposición que la tarjeta sólo será leída por un técnico cualificado que comprende el funcionamiento del sistema.  |

## 6. Diseño del Sistema

---

En este apartado se indicarán los diagramas utilizados que son relevantes para el diseño del sistema. Cada uno contiene una explicación para poder entender de manera adecuada el sistema completo. Se ha elegido realizar un diagrama de componentes y un mapa de tareas, descartando la realización de un diagrama de estados debido a que el conjunto de diagramas utilizados y explicación de la tarea, cumplen el mismo cometido.

### 6.1. Diagrama de Componentes

Se ha añadido un diagrama de componentes para crear una relación entre los casos de uso, tanto el diagrama como los extendidos con las tareas que forman el sistema, las cuales serán comentadas en el Mapa de Tareas posterior y durante el desarrollo del proyecto, también comentado en un apartado posterior.

En el diagrama reflejado en la Figura 6, se da una visión obviando a los actores reflejados en el caso de uso anterior (Figura 5). Debido a que estos no forman parte del diagrama de componentes como tal, los dispositivos, en cambio, son la relación entre estos actores y el sistema. Cabe indicar que se ha incluido el caso de uso **CU-10** dentro del sistema, siendo el único elemento o caso de uso que no se ha considerado cómo dispositivo pero sí como un elemento interno que no tiene función asociada. Tal y como se comentó en su caso de uso extendido y en el diagrama, este caso de uso está reflejado de manera abstracta como una revisión por parte de un técnico del registro en la tarjeta microSD. No es posible incluir en este proyecto una tarea capaz de entender y captar si el registro es correcto, tampoco se podría omitir al ser un sistema en pruebas que debe ser verificado de alguna manera.

## Diseño del Sistema - Diagrama de Componentes

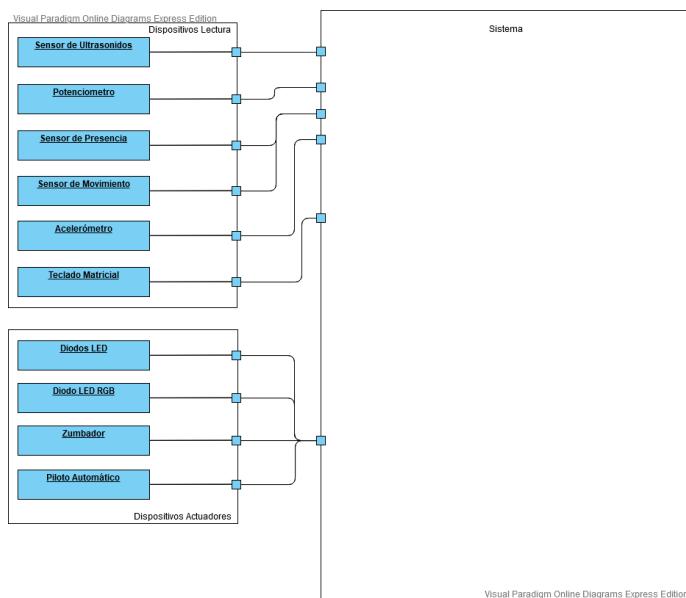


Figura 6: Diagrama de Componentes completo

Tal y como se ha visto en la figura anterior del diagrama de componentes completo (Figura 6), se ha dividido los dispositivos en dispositivos de lectura y los dispositivos actuadores, ambos tipos de componentes pueden interactuar con el entorno, al igual que con los pilotos. No se cree necesario añadir más respecto a esta figura, dado que sólo ofrece la conexión entre los actores de los casos de uso y el sistema. En la siguiente figura, (Figura 7), se ve el diagrama de componentes del sistema, se ofrece la posibilidad de ver este diagrama con mayor detalle en el enlace de OneDrive incluido en el título de la figura. En este caso, se ha seguido la división de funcionalidades del sistema, tal y como se realizó en el diagrama de casos de uso anterior (Figura 5).

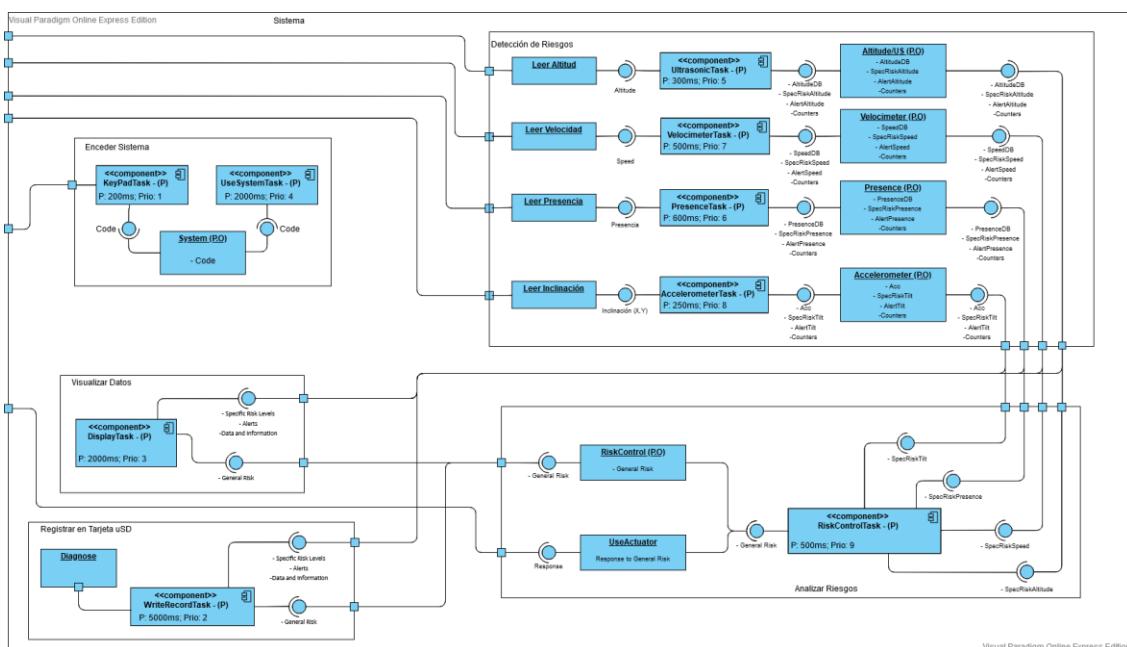


Figura 7: Diagrama de Componentes del sistema. Disponible en [https://upm365-my.sharepoint.com/:i/g/personal/oa\\_stepien\\_alumnos\\_upm\\_es/EeolkmSO4ttKroGq1N9rZqoBSWJNwQRp\\_KELC9h-R5olgQ?e=XtQlvA](https://upm365-my.sharepoint.com/:i/g/personal/oa_stepien_alumnos_upm_es/EeolkmSO4ttKroGq1N9rZqoBSWJNwQRp_KELC9h-R5olgQ?e=XtQlvA)

En la figura anterior, (Figura 7), se llama como componentes a las tareas que forman el sistema, cualquier elemento que no tenga la identificación de “<<component>>” será reflejado como una función en el sistema o activará algún dispositivo externo. En cada componente se ve el periodo de éste y si es Cíclico o Periódico “(P)” o Esporádico “(S)”, en este caso todas las tareas son cíclicas, también incluyen la prioridad que tiene asociada en un texto subtitulado del componente.

En el diagrama viene indicado qué elementos son los recogidos o suministrados cómo interfaces del diagrama, siguiendo el flujo del sistema, al leer los dispositivos, se emite el dato recogido por estos, los “datos del dispositivo”. Dentro del componente asociado se analiza el dato, y se emite en un “objeto protegido” los datos que el sistema utilizará (“Nivel de Riesgo Específico” (“SpecRisk”), “Alerta Asociada” (“Alert”) y el “Valor del dato recogido”). Estos son los datos útiles del sistema, aunque se incluyan también los contadores de cada tarea en el registro en la tarjeta microSD. Los datos comentados se distribuyen tanto al display, como al registro microSD y al “RiskControlTask” para obtener un veredicto del nivel de riesgo general.

Este último, requiere únicamente del “Nivel de Riesgo Específico” de cada interfaz de los componentes de “Detección de Riesgos”. Obteniendo en su desarrollo, el nivel de riesgo general “General Risk”, transmitida a las tareas de registro y “DisplayTask”. También se hace uso la función “UseActuator” que en función del nivel de riesgo general, activa las respuestas asociadas. Ambos componentes, “DisplayTask” y “WriteRecordTask” recogen la información del sistema, con el añadido de los contadores asociados por el registro en la microSD, y los muestran por pantalla o son registrados. En el registro además se incluye el momento de la ejecución de cada cambio aplicado en el sistema o sus respuestas.

También se incluye el elemento “Diagnose” que como se ha comentado previamente, es una función no programada que requiere del análisis del registro por parte de un técnico. Éste no es considerado como un dispositivo más dado que es un caso de uso que realiza un técnico por medio de una revisión.

En la figura anterior se muestran los nombres de los objetos protegidos y los componentes con los nombres tal y como serán nombrados en el desarrollo, en inglés. Se muestra también los objetos protegidos de cada dispositivo o componente, donde son almacenados durante la ejecución las variables de estos últimos. De esta manera es posible establecer un enlace con el siguiente elemento, el mapa de tareas.

## 6.2. Mapa de Tareas

Se ha realizado un mapa de tareas para comprender con mayor facilidad la funcionalidad y coordinación de las tareas. En la siguiente figura (Figura 8), se muestra este mapa de tareas. En él se indican con un romboide las tareas, especificando con colores la finalidad de cada uno. En naranja se encuentran los que utiliza el sistema , y en azul los que no interactúan directamente con él. También se incluyen los dispositivos de entrada/salida con un rectángulo con el borde discontinuo. Entre estos destaca uno de color azul, éste es un dispositivo de salida que requiere de un semáforo para acceder.

También se incluyen los objetos protegidos junto con la información contenida en ellos, se identifican con un rectángulo en naranja claro. Para no crear confusión, los colores de cada asociación al objeto protegido son diferentes para no complicar la visión. En un apartado posterior se detallarán las tareas y sus objetos protegidos. En esta sección se especificarán las uniones y motivos de este mapa de tareas.

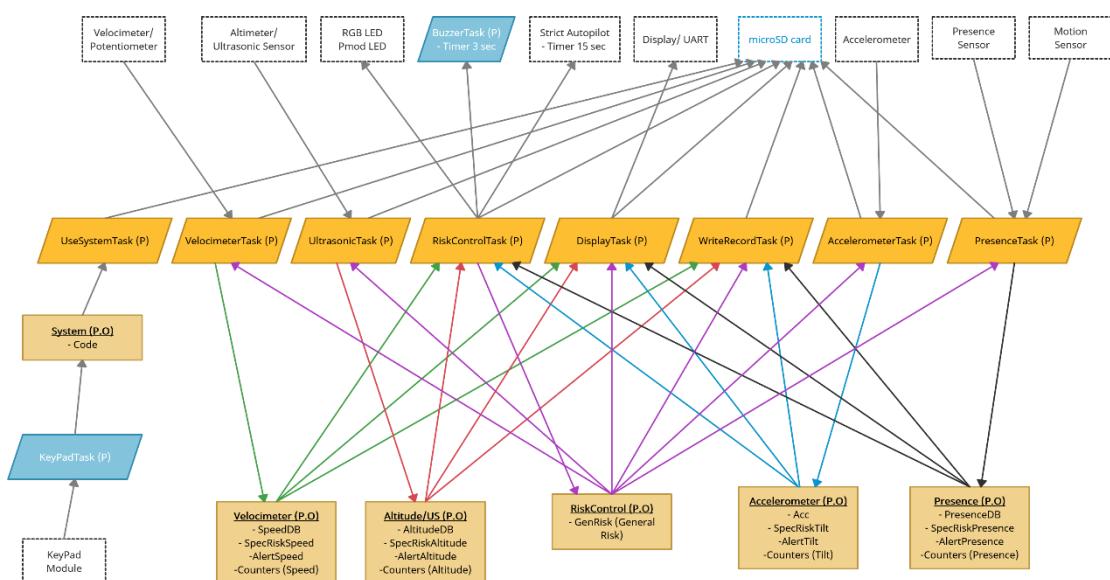


Figura 8: Mapa de Tareas del sistema

Tal y como se ha comentado previamente, el sistema se compone de 10 tareas, indicados con un romboide, de las cuales 8 son las que utiliza el sistema periódicamente, indicadas en color naranja. Existen otras dos tareas, marcadas en un tono azul, que a pesar de estar indicadas como periódicas "(P)", estas dos tareas no se consideran del sistema. Una de ellas "KeyPadTask" se trata de la tarea que lee el teclado, y puede estar durante toda la ejecución del sistema en un bucle vacío o sin ninguna operación. La otra tarea que no se considera tampoco del sistema, es la encargada de hacer funcionar el zumbador. Ésta requiere de la activación de un semáforo para ser activada, pero mantiene un periodo para conseguir que el zumbador se active correctamente, por tanto mantiene una función "esporádica" con una ejecución periódica. Ignorando estas tareas, el resto son tareas periódicas que

aunque puedan no devolver datos, su periodo y tiempo de cómputo no varía en su ejecución.

También se ha indicado que existe un dispositivo “*microSD card*” que a diferencia del resto de dispositivos requiere de un semáforo para funcionar. No se cree necesario indicar su objeto protegido dado que no existen valores o variables que se modifiquen en él, salvo el nombre del fichero. La manera de indicar esto, evitando añadir un objeto protegido adicional, es indicar este dispositivo de un color distinto. Internamente, el dispositivo requiere exclusión mutua y de un semáforo para ésta. Su funcionalidad se comenta en el apartado de las tareas, en el bloque de la tarea “*WriteRiskControl*”, siendo la tarea encargada de escribir un registro en la tarjeta. También se ve que es un dispositivo utilizado por todas las tareas, debido a los mensajes que son registrados ante un cambio de las variables. Es necesario añadir que este dispositivo registra la ejecución del display en cada ciclo de este último. Además de la activación o desactivación de los riesgos de las tareas de los dispositivos de entrada del sistema, sin escribir de forma constante en su ejecución.

Por último, tal y como ya se indicó, los objetos protegidos se han indicado en el mapa anterior, en estos se indica además del nombre, su contenido. Para que no exista confusión entre qué tareas usan cada uno, se han indicado de varios colores las asociaciones a cada tarea. Además el sentido de la flecha indica si lee el contenido o escribe el contenido dentro. Los objetos protegidos cuyas tareas tienen asociado un sensor están formados por una variable que contiene el valor del dispositivo para ser leído posteriormente, el nivel de riesgo específico, la alerta y los contadores de las lecturas correctas e incorrectas de cada una de las tareas. En el caso del objeto protegido de “*RiskControlTask*” únicamente contiene el riesgo general. Y en el asociado a la tarea de “*UseSystemTask*” contiene el código introducido. Todos los objetos protegidos requieren el uso de un semáforo mutex, que no permita la interrupción por tareas de la misma prioridad o inferior.

El resto de los componentes del mapa anterior se opina que no es necesario añadir más información de la que se dará en la fase de desarrollo del código y de las tareas.

## 7. Implementación del proyecto

---

### 7.1. Investigación Previa

Antes de comenzar el desarrollo y pruebas necesarias para el proyecto, se especificará la placa FPGA que se utilizará. Al existir varias versiones para el entorno de desarrollo, se realizará también una investigación sobre estas, con el objetivo de indicar cual es la más adecuada para los Sistemas de Tiempo Real. Adicionalmente, se indicarán los dispositivos que se han investigado.

#### 7.1.1. Placa FPGA

La placa FPGA utilizada es Zybo Z7-10 de Digilent [19] (Figura 9). No se requiere de una comparativa de placas FPGA, dado que el objetivo del proyecto es mostrar la versatilidad de esta placa en concreto y su desarrollo para ser utilizada con Sistemas de Tiempo Real.



Figura 9: Placa FPGA Zybo Z7-10

Para alguien que no conozca las ventajas de una placa FPGA, estas incorporan un procesador FPGA (*Field Programming Gate Array*), se caracterizan por permitir una reprogramación de sus circuitos internos. Estos circuitos están formados por varios subcomponentes llamados CLB o Bloques Lógicos Configurables, que permiten adaptar su funcionamiento a códigos escritos en Lenguajes de Descripción Hardware (HDL) como son Verilog o VHDL. Internamente están formados por transistores que se

pueden conectar entre sí, dando el funcionamiento deseado. En la imagen inferior (Figura 10), se puede ver un diseño muy ampliado de una FPGA.

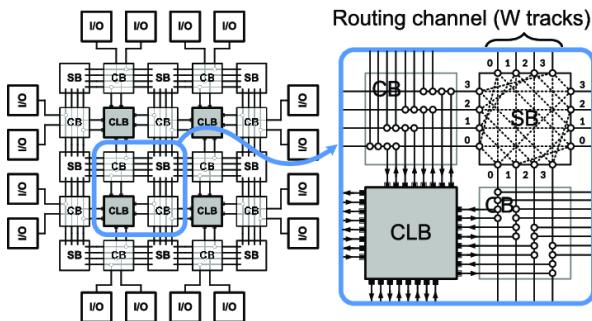


Figura 10: Esquema de una FPGA [20]

El uso que se les da a las FPGAs abarca un amplio abanico, desde aplicaciones recreativas para emular sistemas informáticos míticos hasta sistemas de defensa y aeroespaciales. Pasando por procesamiento de imágenes y soluciones automovilísticas [1]. En este caso, el objetivo es crear un sistema para prevenir o alertar sobre peligros durante un vuelo en avión.

La placa que se va a utilizar dispone de una FPGA que pertenece a la familia de procesadores Zynq-7000 (Z7) de Xilinx [21]. Esta placa dispone de 17600 LUTs (Look-up table) que son unidades que permiten replicar el funcionamiento de puertas NOR, XOR, OR, NOT, etc... Junto con las unidades flip-flop, de las cuales dispone 35200, forman varios miles de bloques CLB, que se reprograman según la necesidad de cada sistema. Además, dispone de 1GB de memoria RAM y 16 MB de memoria Flash. Incorpora 2 HDMI (entrada y salida), conector de cámara *Pcam* y 5 conectores *Pmod* para conexión de dispositivos, esta última es una de las principales ventajas que ofrece respecto a otras placas FPGA de Xilinx.

Respecto a los dispositivos integrados, dispone de 4 interruptores *switch*, 6 botones, 5 *leds* monocromos y 1 *led* RGB. La conexión para la programación de la placa se realiza por medio de un cable microUSB. Es necesario indicar que una parte de estos dispositivos se encuentran conectados directamente al procesador, para utilizarlos desde el propio código.

Como se ha comentado anteriormente, el uso común de la reprogramación de estas placas es mediante HDLs, como no es el objetivo del proyecto, y existe una alternativa, bloques IP, se utilizarán estos últimos. Estos bloques, son códigos prefabricados que configuran el funcionamiento de componentes de la placa sin necesidad de conocer Verilog o VHDL. Internamente, crean esos códigos y configuran sus conexiones, pero se reflejan como componentes hardware (Figura 11). Estas unidades facilitan el avance de algunos proyectos, aunque existan algunos bloques IP que son de pago para utilizarlos, dado que pertenecen a empresas privadas o no se permite su distribución. Normalmente, en los entornos de desarrollo vienen incluidos varios de estos bloques de uso común. Sin embargo, dado que muchos de ellos no son de uso común, se ofrecen mejoras de los entornos para que incluyan los bloques como pueden ser los puertos *Pmod* o el uso de otros dispositivos de la placa.

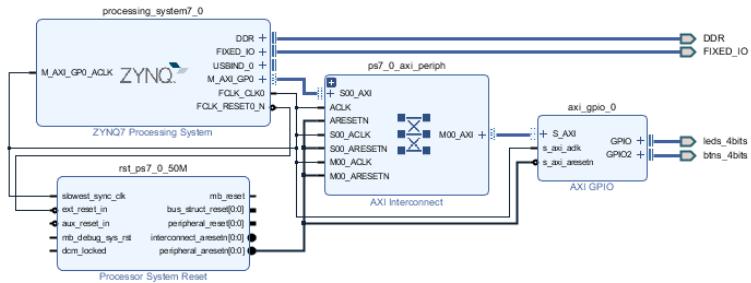


Figura 11: Diseño IP de ejemplo GPIO

## 7.1.2. Investigación respecto a los IDE de Xilinx

Otro de los apartados a indicar es el software utilizado, principalmente, la elección del entorno de desarrollo (IDE) que se va a utilizar para crear el proyecto completo, junto con las pruebas previas. Al ser una placa de Xilinx, tiene cierta limitación el intento de encontrar alguna IDE de libre distribución, a pesar de que las alternativas que ofrece Xilinx son pocas, son suficientes para valorar cual es el indicado para este proyecto.

Los IDEs que se van a comparar son SDSoc [22], un conjunto de herramientas que recomienda la propia Digilent, y Vitis [23], otra de las alternativas similares que ofrece Xilinx. Esta comparativa es más compleja debido a que funcionalmente ambos realizan la misma acción, de hecho, Vitis es el sustituto, junto con otras herramientas de Xilinx. Aun así, es necesario realizar una comparativa, para asegurar cual es más completa, aunque no esté actualizada.

La comparativa valorará el aspecto técnico de cada programa, con sus ventajas y características. También será valorada la instalación del entorno, creando una guía de ésta. Al tratarse de un dato informativo, donde se valoran diferentes aspectos se cree innecesario incluirse en la memoria, sin embargo para interés del lector, se incluyen en un anexo posterior [Anexo I].

### 7.1.2.1. Resumen de la comparativa de la parte técnica y el uso de IDEs

Se ofrecerá un resumen de los entornos valorados en esta primera fase, en el sentido técnico, ambos entornos son similares entre sí, debido a que Vitis agrupa varios entornos de la familia de Xilinx bajo un único nombre. Por tanto, se sigue manteniendo el funcionamiento de SDSoc aunque sea con otro nombre.

Respecto a la parte práctica de la comparativa, tanto SDSoc como los IDE de otras gamas de procesadores de Xilinx, utilizan Vivado. Por este motivo, no es posible

especificar o seleccionar un entorno concreto de estos dos, comprobándose que su funcionamiento o “puesta en marcha” es prácticamente igual. Sin embargo, una de las características claves, es la falta de actualización de SDSoc, finalizada a mediados de 2019. Y el coste de su licencia, la cual se puede obtener por la compra de determinadas placas de Xilinx SoC, mantiene un coste más elevado por defecto.

En la siguiente tabla (Tabla 1) se resumen algunos aspectos más relevantes, pero la resolución primera de esta comparación se resume en decantarse debido a sus actualizaciones, por Vitis y su versión más actualizada a fecha de septiembre de 2020 (2020.1).

|                     | SDSoc DE  | Vitis Platform  |
|---------------------|---|---|
| Versiones           | 2016.4 – 2019.1   | 2019.2 – 2020.1(2020.2)   |
| Guías               | No, adaptaciones intuitivas de versión 2016.4.<br>Existe documentación más detallada. | No, adaptaciones intuitivas de SDSoc.<br>Existe documentación.                  |
| SDK para Zybo Z7-10 | No, usando Vivado.  | No, usando Vivado.  |
| Precio (Licencia)   | Licencia SDSoc: 1000\$<br>Posible licencia Vivado: 3000\$                             | Licencia Vitis: Gratuita<br>Posible licencia Vivado: 3000\$                     |
| Añadidos            | No.   | Vitis AI y librerías Open-Source.   |
| Uso de OpenCV       | Uso de OpenCV o herramientas, por medio de software adicional (p.e. Petalinux).       | Uso de OpenCV o herramientas, por medio de software adicional (p.e. Petalinux). |

Tabla 1: Resumen comparativa de IDEs

### 7.1.3. Dispositivos testados

Dada la falta de especificación de la anterior fase o resolución, se probará el funcionamiento de varios dispositivos para obtener una resolución del entorno de desarrollo final. Los dispositivos testados son un diodo LED, un botón, un potenciómetro, el módulo PmodLED y la cámara. A continuación se indicará un breve resumen de cada uno de ellos y la manera en la cual afectó a la resolución final. En caso de un interés adicional a la resolución en este apartado, se provee en un anexo información adicional.

#### 7.1.3.1. Cámara Pcam 5C

El uso de este dispositivo de la marca Digilent para placas FPGA, es el más significativo respecto a la elección del entorno final. Una de las carencias de Vitis es la falta de proyectos en él, pero gracias a la retrocompatibilidad de Vivado, es posible actualizar desde una versión anterior de un proyecto. En este caso, y debido a la falta de experiencia en el diseño en FPGAs se ha decidido usar un proyecto de Vivado 2019.1, es decir, de SDSoc 2019.1.

Tras varios intentos fallidos de compilación, se encontró en varios foros un problema que puede ocurrir con la importación de proyectos a la versión de Vitis 2020.1 que no permite la compilación de códigos. Debido a esto, se optó por mantener en el entorno de Vitis pero volver a una versión anterior, 2019.2. En ésta, sí que se consiguió hacer funcionar la cámara correctamente. Por tanto se optó por continuar con las pruebas con esta versión de Vitis. Una explicación más detallada, se puede encontrar en el Anexo I ya mencionado.

#### 7.1.3.2. Dispositivos de Entrada/Salida

Los dispositivos de entrada salida (I/O) no fueron tan significativos como el uso de la cámara respecto al entorno, pero sí que se llegó a ver algunos fallos que dispone la placa, todos ellos solucionables o que no afectan al funcionamiento de manera crítica. Los dispositivos probados son un módulo PmodLED con varios diodos led conectados, algunos botones y led para probar los puertos Pmod y por último un potenciómetro, para conectar entradas analógicas.

El principal problema es debido a que uno de los puertos Pmod de la placa tiene el orden de sus parejas alterado, es decir, en lugar de seguir un orden 1, 2, 3, ...8. El orden de las parejas se ve invertido, el pin 1 se encuentra en el pin 2 y viceversa. Esto ocurre con todos los pines de este puerto PmodJD, al tratarse de un caso

exclusivo y poco relevante, dado que cambiar de orden las conexiones es más sencillo que buscar una solución en el diseño, la cual no se consiguió hacer funcionar al intentar alternar bloques del mismo puerto.

Respecto a la entrada/salida analógica, el uso de un bloque IP (Intellectual Property) de Xilinx, XADC Wizard, permitió su conexión y funcionamiento, llegando a un problema general debido a la potencia que suministra el puerto PmodJA, que es de 1.1V para evitar recibir demasiada potencia, requiriendo un divisor de potencia. En el anexo posterior (Anexo II), estos dispositivos incluyen un código de prueba y/o pautas en su diseño IP para hacerlos funcionar, además de los imprevistos ya comentados.

#### 7.1.4. Resolución final del entorno

Tras la realización de las pruebas de los dispositivos en el apartado anterior, que junto a las características de éste, se concluye que el entorno utilizado será Vitis 2019.2. Se llega a esta conclusión, inicialmente, por la posibilidad y una mayor facilidad de adaptar proyectos de sistemas de Tiempo Real en el futuro. En este caso, la versión utilizada es 2019.2, pero es probable que otras versiones de este más actualizadas sean capaces de incluir mejoras y mayor compatibilidad entre dispositivos. La versión 2020.1 de Vitis ofrecía varias mejoras a la versión finalmente elegida, pero debido a una incompatibilidad con la importación de proyectos de SDSoc a ésta, se decidió por elegir una versión de Vitis menos actualizada.

Otro de los motivos de esta elección, es el precio de la licencia, Vitis, como ya se comentó es un software de Xilinx gratuito, se indicó que el precio de Vivado está limitado a algunos dispositivos, pero estos son muy específicos, con poca probabilidad para un posible desarrollo en ellos. En cambio, SDSoc la licencia siempre es de pago. También se indicó que fue desarrollado para ser compatible con librerías “*open-source*”. Estas, aunque no se vayan a utilizar en el proyecto, o al menos, esa no es la intención, establecen una posibilidad a la creación de proyectos de mayor relevancia.

Respecto al modo de funcionamiento, apenas es perceptible una diferencia, principalmente porque Vitis contiene SDSoc entre otras plataformas, y permite mantener un único entorno, independientemente del modelo de procesador utilizado. Por tanto, aunque se elija cualquier otra placa FPGA de Xilinx, se podrá seguir usando Vitis sin necesidad de la instalación de cualquier otro entorno.

La decisión de elegir este programa ha sido provocada también por la compatibilidad de los dispositivos, que ha mejorado el entendimiento de algunos puertos de la placa o al menos, garantizaba su funcionamiento.

## 7.2. Dispositivos Utilizados

En esta sección se van a detallar los dispositivos utilizados en el proyecto, además de los ya testados, se indicará el uso de cada uno para precisar posteriormente los posibles objetivos que requerirán. Se establece que el sistema funcionará en pleno vuelo, ignorando maniobras de despegue o aterrizaje que pueden incidir en algunos aspectos de los dispositivos utilizados.

### 7.2.1. Acelerómetro

El dispositivo más importante del sistema es el acelerómetro, en él se utilizarán los 2 ejes del avión, X e Y, dado que el eje Z depende de la dirección del norte para dar una medida precisa. Estos movimientos en los ejes son provocados por los alerones que tienen instalados los aviones en las alas y la cola.

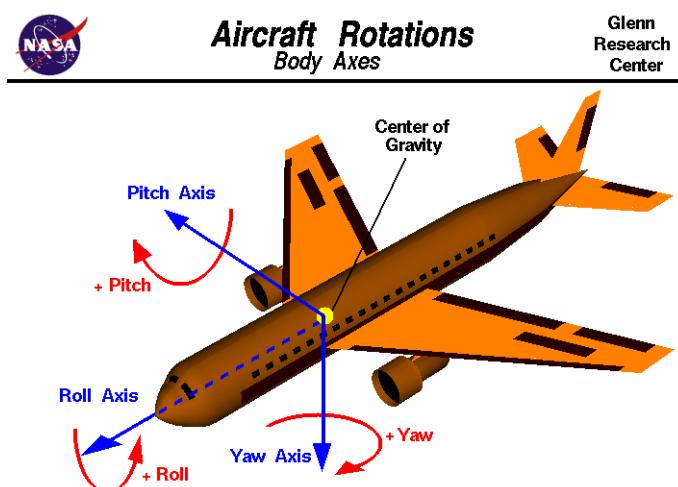


Figura 12: Ejes Pitch, Roll y Yaw de un avión. [24]

Los ejes que dispone un avión son los que se pueden ver en la figura superior (Figura 12). En el proyecto se utilizará el eje transversal, en el eje Y del avión, que produce el “movimiento del cabeceo” del avión. Este movimiento es el que realiza para la subida o bajada del morro del avión. Se ha establecido un rango de (-20, 20) grados para valorar si se considera que el ángulo es adecuado en el vuelo. Superar este ángulo, es excesivo incluso para una maniobra de bajada o aterrizaje. Al considerarse que el avión está en pleno vuelo, debería mantenerse el eje Y en los 0 grados, hacia el destino para considerarse que no hay ningún peligro.

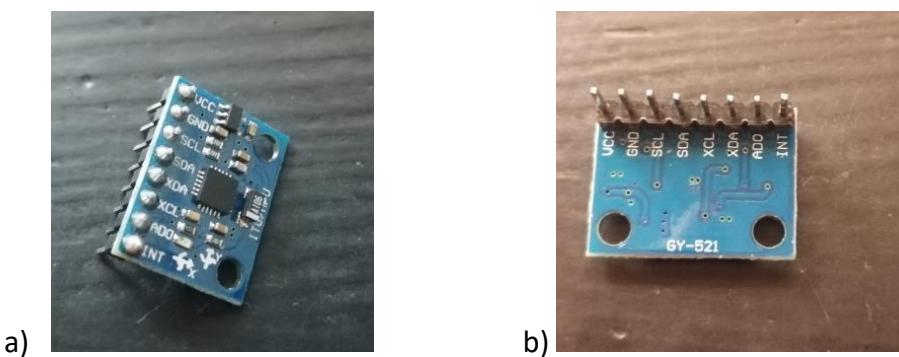
Otro de los ejes es el eje longitudinal, el situado en el eje X, que se identifica con el nombre de “movimiento de alabeo”, también es indispensable en el sistema, éste indica que el avión se inclina hacia un lado mediante una combinación de los alerones para probablemente cambiar el rumbo o inclinarse por cualquier otro motivo.

El rango que se ha considerado para su funcionamiento es de (-30, 30) grados, es más amplio que el asignado en el eje Y, pero también es más probable una inclinación en este eje por cualquier motivo, principalmente para girar. Del mismo modo, se cree que superar este ángulo es demasiado peligroso, en cualquier circunstancia. Considerando en pleno vuelo que debería mantenerse recto en eje X, es decir, a 0 grados hasta el destino.

Por último, el tercer eje, el eje Z es el llamado “movimiento de guiñada”, es menos común utilizarlo, pero es un movimiento que se suele utilizar mientras el avión se encuentra en pista. Como el sistema se establece un funcionamiento en pleno vuelo, no se considerará esa situación. También, puede ocurrir en el vuelo, debido al cizallamiento o “windshearing”, donde una racha de viento realiza movimientos bruscos que puede desestabilizar el avión. En pleno vuelo ideal, no es un movimiento necesario, porque para girar el avión, se puede hacer una combinación de los otros dos ejes con el mismo resultado más leve. Por ese motivo, el eje Z no se valorará para obtener medidas, además de la complicación que supone obtener este eje sin un magnetómetro, para conocer el campo gravitacional que aplica en cada momento.

El dispositivo utilizado es un acelerómetro GY-521 (Figuras 13ab), funciona con I2C con la placa para poder obtener la inclinación o ángulo en el cual se encuentra la aeronave. En una primera fase de pruebas del dispositivo se ha detectado que mantiene un error de unos 6 grados en el eje X y sobre los 3 grados en el eje Y. El eje Z, es posible calcularlo pero aumenta gradualmente con el tiempo, para obtener una medida precisa se requeriría de un magnetómetro, que no se ha considerado incluir. De esta manera, es posible obtener los ejes X e Y del propio acelerómetro del dispositivo. Estos ejes son llamados *Pitch* y *Roll*. Se obtienen con la aceleración para obtener el giro o ángulo actual de la aeronave.

Para resolver el problema de los 6 grados de inclinación que surgen al utilizar el dispositivo sobre la placa, es posible resolverlo aplicando un soporte en la parte externa del mismo. Con una correcta colocación, es posible obtener en un terrero recto valores estáticos, cercanos al 0, que es el funcionamiento ideal.



Figuras 13a y 13b: Acelerómetro GY-521; a) frontal b) parte trasera

Respecto a su testado sobre la placa, sin interferir en el sistema operativo FreeRTOS, se hace uso de la librería de Xilinx “xiic.h”, ésta permite utilizar funciones para configurar, enviar y recibir datos por el módulo digital I2C “AXI IIC” que se añade

al diseño (Figura 14). Este módulo permite usar cualquier par de pines no asociados al uso por I2C.

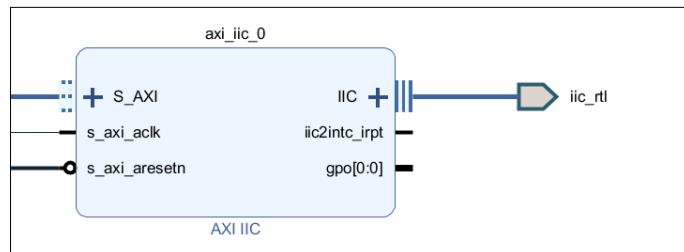


Figura 14: Bloque IP AXI I2C con salida iic\_rtl.

### 7.2.2. Altímetro (Sensor ultrasonidos)

Debido a que no se puede probar el sistema en las condiciones de altura adecuadas, el uso de este dispositivo se sustituye por un sensor de ultrasonidos. De esta manera, es capaz de medir una altura proporcional a la real. Con el uso de este dispositivo se pretende estimar una altura proporcional a 10000-12000 metros en las capacidades del sensor.

Con este sensor se indicará la altura del avión y que éste se encuentre en las alturas mínima y máxima recomendadas. Estas alturas se suelen establecer en 10000 metros (MSA, “*Minimum Safety Altitude*”) y los 12000 que suele ser el techo de vuelo de muchos aviones comerciales. Reducir la MSA por debajo de los 10000 metros, puede suponer un peligro de colisión con otras aeronaves o terreno escarpado, y la recomendación es de volar 2000 metros por encima del terreno, en este caso 10000 metros es adecuado en cualquier terreno. La altitud mínima es en muchos casos uno de los valores que menos respetan los pilotos porque no supone un elevado peligro, pero es uno de los factores que el sistema tendrá en cuenta y avisará de su infracción.

En otro caso, al superar el techo de vuelo, éste puede suponer una despresurización del avión, pudiendo causar fallos o daños irreversibles, que el sistema avisará. En la situación en la cual trabajará el sistema, las condiciones ideales, será adecuado que vuele a una altura entre el MSA y el techo de vuelo, por tanto, un rango entre 10000 y 12000 metros, idealmente a 11000 metros.

El sensor ultrasonidos utilizado es HC-SR04 (Figura 15), este sensor funciona por medio de pines digitales y se puede obtener una altura máxima de 4 metros, pero desde los 250 centímetros (2.5 metros) las medidas son menos fiables. La distancia mínima capaz de detectar son 4 centímetros que en el caso del sistema nunca se alcanzarán. En el rango de funcionamiento fiable, se tomarán medidas para ser comparadas con la altitud necesaria para el sistema. Al funcionar de manera correcta con los puertos digitales, es suficiente con utilizar el módulo digital “PmodGPIO” de Digilent (Figura 16) y la librería “PmodGPIO.h” que permite su utilización.



Figura 15: Módulo detector de distancia por ultrasonidos HC-SR04

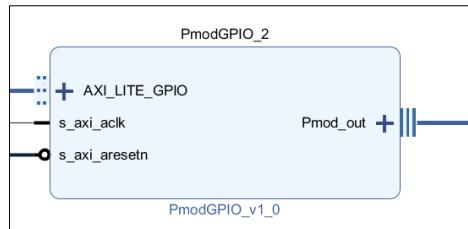


Figura 16: Bloque IP PmodGPIO y salida Pmod

Debido a la necesidad de este dispositivo de requerir un voltaje de entrada de 5V, voltaje que la placa no es capaz de suministrar, se requiere para su correcto funcionamiento dos dispositivos asociados adicionales.

### 7.2.2.1. Módulo ESP8266

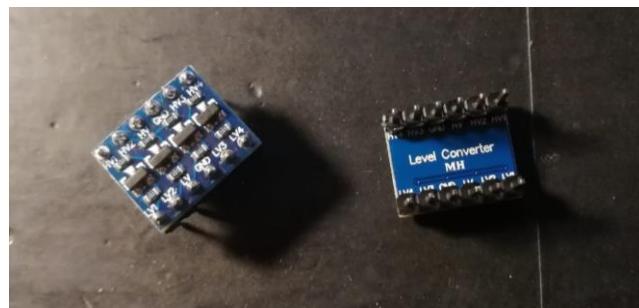
El uso de este dispositivo sólo es necesario para suministrar el voltaje de 5V requerido, para ello se utilizará el módulo sin programar o con un programa de prueba que haga parpadear el led integrado para conocer si se encuentra encendido. El módulo ESP8266 [25] permite una programación por el entorno de Arduino [26], que no se utilizará aparte de en su necesidad para un programa de prueba simple, el módulo permite también el uso de WiFi, éste se puede ver en la siguiente figura. (Figura 17).



Figura 17: Módulo ESP8266

### *7.2.2.2. Level Converter o Level Shifter*

El otro dispositivo es un circuito que adapta un voltaje a otro mayor o inferior, se puede ver en la siguiente figura (Figura 18). El uso de este dispositivo es únicamente para asegurar que el voltaje que se introduce en la placa es de un máximo de 3.3V. No requiere de programación alguna, dado que requiere conectar el voltaje Bajo y Alto para que sea capaz de adaptar el voltaje.



*Figura 18: Módulo Level Converter*

Se cree que utilizando este dispositivo, aunque el voltaje suministrado sea insignificante en el tiempo, garantiza el voltaje máximo deseado en ambos extremos, tanto placa FPGA como dispositivo. Además de comprobar que el funcionamiento del sensor es adecuado, sin grandes variaciones que podrían ser causa de una activación anticipada por parte de la placa.

### *7.2.3. Velocímetro (Potenciómetro)*

El sistema también tendrá en cuenta la velocidad a la cual se encuentra el avión, para definir con mayor precisión, si el avión se encuentra en una situación peligrosa o no. El sistema actual dispondrá de un velocímetro, que en el proyecto realizado se reflejará usando un potenciómetro para indicar la velocidad que mantiene el avión. Como el sistema está pensado que se encuentre activado en pleno vuelo, se tomará como la velocidad ideal, la velocidad de crucero 827 km/h. Con un margen de error del 3% por la posible imprecisión del dispositivo.

La alerta se activará si la velocidad es inferior o superior a la velocidad de crucero, y también tendrá en cuenta que el avión dispone de una velocidad máxima que no se deberá superar, en este caso 871 km/h. El potenciómetro utilizado es deslizante (Figura 19), permite un ajuste más preciso o es capaz de indicar de manera más precisa, el voltaje suministrado, por tanto, permite conocer el estado en el cual se encuentra en cada instante.

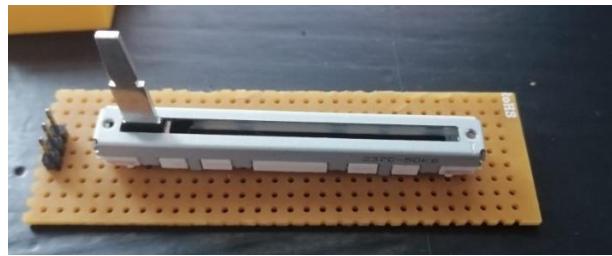


Figura 19: Potenciómetro deslizante

En este caso, el potenciómetro utiliza salidas analógicas, por tanto se utiliza el módulo de Xilinx “XADC Wizard” el cual viene detallado en el siguiente anexo (A2.1.2) junto con otro potenciómetro

#### 7.2.4. Detector de presencia

La Agencia Europea de Seguridad Aérea (EASA), tras el incidente de *Germanwings* en 2015, establece un mínimo de 2 personas en todo momento [27]. Para detectar esta presencia en la cabina, se van a utilizar dos dispositivos. En primer lugar, un detector de presencia por infrarrojos, en este caso, uno por piloto. En el caso de que no se detecte presencia en uno o ambos de estos detectores configurados para una distancia cercana, se activaría la alerta. Esto supondrá que hay menos de dos personas en la cabina.

Adicionalmente, se valora que pueden acceder a la cabina una o más personas, para ello, se instalará un detector de movimiento cercano a la puerta, que aunque no sea preciso, será capaz de identificar que en la cabina no sólo se encuentren los pilotos.

El módulo infrarrojo de detección de obstáculos, se utilizará el modelo FC-51 (Figura 20), que permite ajustar la distancia a la cual se activará. El módulo de detección de presencia o movimiento es el modelo HC-SR501 (Figura 20), es posible ajustar la sensibilidad y retardo de desactivación de éste.



Figura 20: Módulos detectores de presencia por Infrarrojos; Izquierda: Módulo de presencia por IR FC-51; Derecha: Módulo sensor de movimiento HC-SR501

Del mismo modo que el dispositivo sensor de ultrasonidos usado como altímetro, utilizan entradas digitales, por tanto, es suficiente con usar el módulo “PmodGPIO” (Figura 16) y su librería “PmodGPIO.h”.

### 7.2.5. Teclado matricial 16 teclas

El sistema incluye un teclado matricial de 16 teclas Pmod de Digilent, este teclado al ser un módulo Pmod dispone de una librería para su uso que simplifica el funcionamiento al utilizarlo. El uso que se le dará es la introducción de un código para la activación y desactivación del sistema, estos códigos serán diferentes para evitar una desactivación accidental del sistema. Los códigos tendrán el formato “C/D”+ código de 4 dígitos hexadecimales + “A”. El primer carácter, definirá si se pretende encender o conectar “Connect” el sistema, o se desea apagar o desconectar “Disconnect” el sistema. Posteriormente, se almacenará el código de 4 dígitos o caracteres, y se pulsará “A” para aceptar el input. En caso de introducir erróneamente el código el sistema no se encenderá o no se apagará. Otros códigos que también comprenderá el sistema se detallan en un próximo apartado con una vista detallada de la tarea.

El módulo utilizado es el modelo PmodKYPD, mostrado en la siguiente figura (Figura 21). La instalación del módulo al ocupar un puerto Pmod completo, se le asigna la utilización al puerto PmodJC, que no dificulta el acceso a otros puertos. En el diseño utilizado, existe un módulo de Digilent con el mismo nombre “PmodKYPD” (Figura 22) lo que llega a indicar que es el indicado para su funcionamiento. El módulo tiene asociada la librería “PmodKYPD.h” para su uso, facilitando el código que se requiere usar.



Figura 21: Teclado matricial PmodKYPD

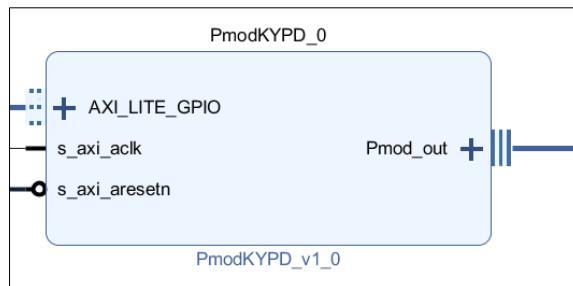


Figura 22: Módulo IP PmodKYPD y puerto Pmod asociado

## 7.2.6. Puerto microSD

Se ha incluido en el proyecto un lector de tarjetas SD, la utilidad que se le dará es como almacenamiento para un posterior análisis. En la tarjeta microSD se almacenarán uno o varios registros de la actividad durante el vuelo. Estos contendrán los datos recogidos durante la actividad además de las medidas tomadas si fuese necesario alertar o activar algún otro dispositivo. Este lector viene integrado en la propia placa Zybo Z7-10, por tanto no requiere de una conexión adicional a la placa.

Para su uso con la placa, requiere de dos librerías adicionales “xsdps.h” y “ff.h”, ambas necesarias para su prueba y escritura en la SD. Adicionalmente, se requiere convertir el texto en un valor numérico para el buffer de escritura de la memoria, esto se puede hacer aplicando pequeños añadidos al código.

## 7.2.7. Puerto UART

El puerto UART que dispone el sistema mostrará por medio de un terminal la información más relevante o útil que ofrezca el sistema, mostrando a los pilotos por medio de éste o por un display la información.

Al pertenecer a la configuración del procesador Zynq-7000, no requiere de bloque IP o diseño IP para su funcionamiento.

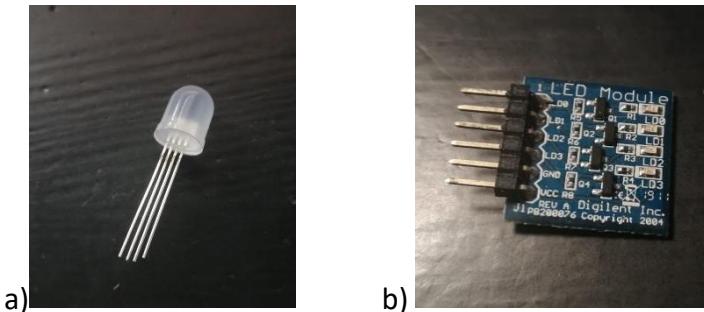
## 7.2.8. Diodos LED

Para indicar a los pilotos de cualquier situación tanto crítica como no, se hará uso de dos sets de diodos LED. Un primer set, será un led RGB (Figuras 23a), para indicar los colores indicados: azul, naranja/amarillo y rojo. Adicionalmente, se utilizará un módulo PmodLED para indicar la cantidad de fallos leves o la situación en la cual se encuentra el vuelo, por medio de 4 diodos LED rojos (Figuras 23b). El conjunto de varios LED de colores se traducirá en una luz capaz de iluminar la cabina, detrás del piloto, para no imposibilitar el vuelo pero alertar de la situación a los pilotos. Por último, para ver que el sistema está activado y funcionando, uno de los diodos LED de la placa parpadeará para indicar la actividad.

Al requerir únicamente de unos pocos colores (azul, amarillo y rojo), es suficiente con usar los puertos digitales, por tanto, el uso del módulo digital ya comentado “PmodGPIO” (Figura 16) es suficiente. Otro de los dispositivos usados, será el módulo físico “PmodLED”, éste no dispone de su análogo digital, pero utiliza el mismo módulo PmodGPIO.

## *Implementación del Proyecto - Dispositivos Utilizados*

Por último, como se indica el uso de un diodo LED en la propia placa, éste debe ser especificado en el diseño, con el bloque IP “AXI GPIO”, visible en el diseño de prueba de testeo de entornos en el anexo siguiente (A2.1: Dispositivos testados). Requiere el uso de la librería “xgpio.h” que difiere en funcionamiento con la de Digilent, por tanto, no es posible acceder al resto de dispositivos GPIO con esta última librería.



Figuras 23a y 23b: a) Diodo LED RGB de 4 pines; b) Módulo PmodLED con 4 diodos LED

### 7.2.9. Zumbador

Otro de los dispositivos de salida utilizados, su utilidad es avisar a los pilotos en caso de alguna situación de cierta peligrosidad o para que rectifiquen el movimiento del avión. Se dividen en varios niveles de intensidad o en un número equivalente de repeticiones del pitido realizado.

El zumbador utilizado es RS-457011 (Figura 24), permite un funcionamiento por medio de pulsos, que dependiendo de la frecuencia especificada en el bucle, puede sonar con un tono diferente, más agudo o grave. Como la entrada utilizada es digital, al igual que con el resto de los dispositivos, funciona adecuadamente con el módulo “PmodGPIO” (Figura 16) y su librería.



Figura 24: Zumbador RS-457011

### 7.3. Tareas Individuales del Sistema

En esta fase, se creará un sistema que puede ser utilizado para el sistema final mediante el uso de Vivado, por el cual, se tendrá que especificar de manera adecuada todos los dispositivos que se pretende utilizar, de la manera que se realizó en el anterior apartado de Investigación Previa.

Para crear un diseño del sistema de tiempo real usando las placas FPGA de Xilinx, se ha decidido utilizar los diseños IP ya comentados previamente. Como se requiere del uso de FreeRTOS y la tarjeta microSD, en el bloque de procesamiento Zynq7 se debe añadir una conexión retroalimentada. Esto permitirá el uso de FreeRTOS incluso sin módulos adicionales. El uso de la tarjeta microSD requiere además una configuración por Vitis para activar esta funcionalidad.

Los módulos que se añaden deben ser identificados y situados en los puertos Pmod que se pretenden utilizar. Por ello, el diseño del sistema es el que se puede ver en la siguiente figura (Figura 25).

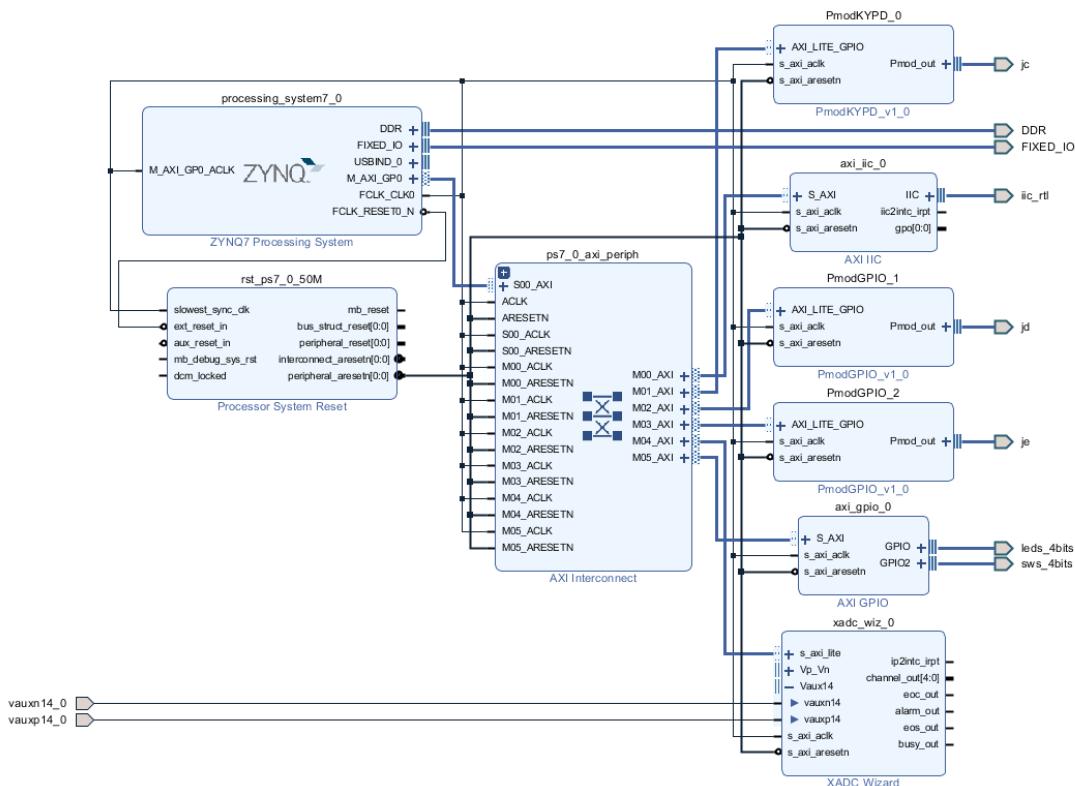


Figura 25: Diseño del sistema completo en Vivado 2019.1

El diseño ha sido configurado de manera automática, pero se pueden identificar diferentes secciones del sistema. Las partes relevantes son las siguientes:

- **Módulo “PmodKYPD\_0”:**

Es el módulo que conecta el teclado matricial a la placa, por la situación de los demás componentes del sistema le ha sido asignado el puerto “PmodJC” completo.

- **Módulo “axi\_iic\_0”:**

Este módulo es el encargado de utilizar la función I2C de la placa, se decide utilizar este módulo que simplifica el uso de este protocolo de comunicación en la placa elegida. Se le conecta al par “iic\_rtl”, aplicando las restricciones de los pines 4 y 10 del puerto “PmodJA”. Estas restricciones vienen indicadas en el fichero de constraints, mostrado en la siguiente figura (Figura 26). En este caso, los pines K14 y J14 corresponden a las posiciones 4 y 10 del puerto, según el manual de referencia de la placa FPGA Zybo Z7-10 [28].

```

Diagram × Address Editor × constraints.xdc ×
E:/PFM/RTOS_v1/RTOS_v1.srcc/constrs_1/new/constraints.xdc
Q | H | ← | → | X | // | ■ | ? |
1: set_property -dict { PACKAGE_PIN N16  IOSTANDARD LVCMS33 } [get_ports { vauxn14_0 }];
2: set_property -dict { PACKAGE_PIN N15  IOSTANDARD LVCMS33 } [get_ports { vauxp14_0 }];
3: set_property -dict { PACKAGE_PIN K14  IOSTANDARD LVCMS33 } [get_ports { iic_rtl_scl_io }];
4: set_property -dict { PACKAGE_PIN J14  IOSTANDARD LVCMS33 } [get_ports { iic_rtl_sda_io }];

```

Figura 26: Captura del fichero de restricciones “constraints.xdc”

- **Los módulos “PmodGPIO\_1” y “PmodGPIO\_2”:**

Estos módulos conectarán los dispositivos de entrada y salida digital de la placa. Sería posible configurarlos para que un puerto permita la entrada y salida por diferentes posiciones, pero dado que se requiere de 8 pines de salida *u output* y pocas entradas al sistema, uno de estos puertos Pmod será dedicado únicamente a las salidas de los actuadores. Mientras que el otro, será el dedicado a los sensores. Estos harán uso de los puertos “PmodJD” y “PmodJE”, respectivamente.

- **Módulo “axi\_gpio\_0”:**

Este módulo no pertenece a Digilent, aunque puede ser utilizado por dispositivos de éste. El uso de este módulo en el sistema es tener acceso a los dispositivos GPIO de la placa, en este caso, al set de diodos LED y *switch* que ofrece ésta.

- **Módulo “XADC\_Wizard”:**

Este módulo ya ha sido indicado previamente, y permite el uso de los pines analógicos en el puerto “PmodJA”, en este caso, al ser utilizado únicamente para el potenciómetro horizontal, sólo se requiere de un par, los pines N16 y N15 que corresponde al par “AD14”

### 7.3.1. Tareas individuales

El sistema hará uso de varias tareas, para cada sensor o funcionalidad del sistema. Para ello, en este apartado se va a indicar el funcionamiento de cada tarea individualmente. Cada tarea se probará individualmente en caso de ser posible, algunas de ellas requieren el uso de otras tareas para funcionar. Cabe indicar que todos los dispositivos se han testado previamente y las tareas individuales corresponden a la prueba con el periodo final y con las respuestas a los riesgos específicos y/o generales.

#### 7.3.1.1. Tarea Lector del Teclado (*keyPadTask*)

Esta primera tarea es la encargada de leer el teclado matricial y almacenar el código en un objeto protegido asociado a él. La tarea tiene un periodo de 200 milisegundos y una prioridad de “*tskIDLE\_PRIORITY+1*”, equivalente a 1, siendo la menor prioridad del sistema. El motivo de un periodo tan reducido es para detectar las pulsaciones de la tecla, y con un periodo de 200 ms se ha probado que es capaz de detectar las pulsaciones del teclado. El periodo de activación se realiza a través de la siguiente función: “*vTaskDelayUntil(&xLastTimeAwake, 200/portTICK\_RATE\_MS)*”.

Los dígitos introducidos (0-F) son almacenados en una variable interna de la tarea. Al registrarse los 6 dígitos de un código se actualiza el código almacenado en el objeto protegido compartido, garantizando su exclusión mutua con el semáforo “*xSemaphoreKB*”.

El código es una adaptación del disponible para las pruebas del dispositivo. Comprueba si el estado de las teclas varía entre ejecuciones y lo asigna al array de caracteres. Tras cada pulsación se imprime por el display el código recogido hasta el momento, de esta manera, aunque sea expulsado por otra tarea, será posible introducir un código de 6 caracteres. El código se puede ver en el siguiente anexo (A3.1.2).

También dispone de una función “*setup*” que inicializa el “*keypad*”, en esta función, se establece el código vacío, inicializa el puerto Pmod asociado y se carga la “*keytable*” como referencia. Esta función se puede ver en el siguiente anexo (A3.1.3). También se incluyen algunas funciones para inicializar el GPIO de la placa y los actuadores del sistema.

Dado que requiere funciones y parámetros, en la definición e inicialización de sus variables, se han añadido las variables “*started*”, “*debug*” y “*options*”. Estas variables no pertenecen al objeto protegido al cual se refiere porque son variables

globales del sistema. A pesar de ello, se han agrupado en el mismo objeto protegido para mantener una estructura, siendo variables modificadas debido al código. También incluye la declaración de los dispositivos XGpio y PmodGPIO utilizados, estos últimos incluyen un sufijo con la acción que realizaran, “\_o” como “output” o “\_io” como “input/output”. También existen dispositivos que no requieren de esa declaración como son el teclado o el puerto analógico. Estas declaraciones se encuentran disponibles en el siguiente anexo (A3.1.1).

#### 7.3.1.1.1. Pruebas

Esta tarea se puede probar individualmente, para ello, al final de la tarea además de almacenar el código en la variable indicada, también lo mostrará por display. El resultado es el mostrado en la siguiente figura (Figura 27).

```
File Edit Setup Control Window Help
prueba tarea Keypad:
Key Pressed: 14
Key Pressed: 147
Key Pressed: 1470
Key Pressed: 14702
Key Pressed: 147025
14703
Key Pressed: 8
Key Pressed: 8F
Key Pressed: 8F3
Key Pressed: 8F26
Key Pressed: 8F29
Key Pressed: 8F309E
8F309E
Key Pressed: A
Key Pressed: AB
Key Pressed: ABC
Key Pressed: ABCD
Key Pressed: ABCD1
Key Pressed: ABCD14
ABC14
```

Figura 27: Prueba tarea Keypad

#### 7.3.1.2. Tarea Usar Sistema (useSystemTask)

La siguiente tarea es la encargada de activar las funcionalidades del sistema en función del código introducido. Tiene una periodicidad de 2 segundos, y una prioridad de “*tskIDLE\_PRIORITY+4*”, equivalente a una prioridad 4. Esta tarea, también se utiliza para hacer parpadear un LED integrado en la placa con el mismo periodo si el sistema se encuentra activado. Aun así, su principal uso es activar funcionalidades por medio de códigos, los cuales son introducidos por el teclado matricial.

La tarea obtiene el código almacenado en el objeto protegido de KeyPad, haciendo uso de la exclusión mutua. En caso de no ser obtenido en un plazo de 30 milisegundos se almacena un código vacío. De esta manera, se evita retrasar esta tarea mientras se esté escribiendo un código en el objeto protegido.

El sistema tiene varias secuencias de códigos con una utilidad. El primer par de códigos es “C1234A” y “D1234A”. Estos códigos se usan para activar o encender el

sistema, y desactivar o apagar el sistema, respectivamente. Ambos interaccionan con la variable “*started*” del sistema. Adicionalmente en caso correcto un mensaje es registrado en la tarjeta microSD por medio de la función “*SDFFileWrite*”. Esta última se detallará en el apartado sobre la tarea de la microSD. Para usar la tarea es necesario usar el semáforo “*xSemaphoreSD*” y pasar a la función el mensaje a escribir en la microSD. La función de desconectar adicionalmente apaga los actuadores Led, independientemente del bloqueo.

El siguiente par de código es el formado por “F0001A” y “F0000A”, usados para activar y desactivar el modo debug del sistema, respectivamente. El modo debug del sistema, muestra por display los riesgos específicos de cada sensor.

Cualquier código no vacío, puede seguir la estructura “BxxxxA”, que permite modificar el techo de vuelo del sistema. Modificando éste, también se genera un límite inferior para respetar y un valor para obtener la distancia real. Se calcula proporcionalmente suponiendo que el techo de vuelo es de 12000 metros y la altitud mínima segura (MSA) es 10000 metros, que son los valores estándar del Airbus A320. Del mismo modo, se almacena este cambio en la microSD. Por último, en caso de no cumplirse las condiciones, y no ser un código vacío, se indica que el código es incorrecto y se limpia la variable.

Finalmente, también se incluye una manera de personalizar el nombre del registro en la tarjeta microSD. Por defecto el nombre del fichero es “Record\_.txt”. Con el uso de un código con la siguiente estructura “ExxxxE”, similar a la estructura anterior, se permite modificar el nombre del fichero siempre que no esté activado el sistema, únicamente la placa. Este código se registrará como el primer mensaje del nuevo fichero se entiende que el valor de “xxxx” es un código hexadecimal para identificar el vuelo, permitiendo distinguir varios vuelos con el mismo sistema, obteniendo un fichero con el nombre “Record\_xxxx.txt”.

La tarea también utiliza un LED integrado de la placa para indicar el funcionamiento del sistema, este led únicamente parpadeará cada 2 segundos si el sistema se encuentra activado. Para ello se utiliza el elemento “*Xgpio options*”, que tiene asociados todos los LEDS e interruptores de la placa, previamente incluidos en el diseño. El código de la tarea se puede ver en el siguiente anexo (A3.2.1).

### 7.3.1.2.1. Pruebas

En este caso, y en la gran mayoría de próximas tareas, se requiere el uso de la tarea “keyPadTask”. En las próximas tareas, también se requerirá esta misma tarea para activar el sistema. En las pruebas, se introducirán los códigos que modifican el sistema y un código erróneo, mostrando también el registro en la microSD. En las siguientes figuras se puede ver la prueba (Figuras 28abcd). En ellas se puede ver qué cambios se han realizado en el sistema con cada código.

The figure consists of four panels labeled a, b, c, and d.

- a)** Shows a terminal window titled "Tera Term - [disconnected] VT". The log output includes: "Prueba tarea KeyPad y UseSystem!", followed by multiple entries of "Key Pressed: F", "Key Pressed: F0", "Key Pressed: F00", "Key Pressed: F000", "Key Pressed: F0001", "Key Pressed: F0001A", "Code: F0001A", "- Debug Mode: ON", and then a series of "Key Pressed" and "Code" entries for letters C through D.
- b)** Shows a terminal window titled "Tera Term - [disconnected] VT". The log output continues from panel a) with "Key Pressed: C", "Key Pressed: C1", "Key Pressed: C12", "Key Pressed: C123", "Key Pressed: C1234", "Key Pressed: C1234A", "C1234A", "Code: C1234A", "- System Working", and then "Key Pressed: D", "Key Pressed: D1", "Key Pressed: D12", "Key Pressed: D123", "Key Pressed: D1234", "Key Pressed: D1234A", "D1234A", "Code: D1234A", "- System Deactivated".
- c)** Shows a terminal window titled "Tera Term - [disconnected] VT". The log output shows key presses for letter B, followed by altitude conversion logic: "Code: B0001A", "- Debug Mode: ON", "Code: B0000A", "- Debug Mode: OFF", "Code: C1234A", "- System Working", "Code: D1234A", "- System Deactivated", "Code: B0080A", "- New Min Altitude: 66", "- New Max Altitude: 80", "- New Ratio Conversion to Real: 150", "Key Pressed: B", "Key Pressed: B0", "Key Pressed: B00", "Key Pressed: B008", "Key Pressed: B0080", "Key Pressed: B0080A", "B0080A", "Code: B0080A", "- New Min Altitude: 66", "- New Max Altitude: 80", "- New Ratio Conversion to Real: 150", "Key Pressed: B", "Key Pressed: B0", "Key Pressed: B00", "Key Pressed: B004", "Key Pressed: B0040", "Key Pressed: B0040A", "B0040A", "Code: B0040A", "- New Min Altitude: 33", "- New Max Altitude: 40", "- New Ratio Conversion to Real: 300".
- d)** Shows a file named "RECORDII: Bloc de notas" in a text editor. The content is a copy of the log from panel c), showing the recorded data on microSD.

Figuras 28a, 28b, 28c, 28d: Pruebas UseSystem. a, b, c) Vista en TeraTerm con ejecución; d) Registro en microSD

### 7.3.1.2.2. Cambios y modificaciones

Una de las modificaciones más tardías o el añadido de una de ellas, es la inclusión del código para cambiar el nombre del fichero, ese es el motivo por el cual no se incluye en la prueba anterior. También, requiere de un ajuste para el formato exFAT, el cual un nombre de ficheros de gran longitud, que anteriormente sólo permitía 8 caracteres en mayúsculas sin incluir el tipo de archivo. Con el añadido del formato de archivo exFAT, se permiten también las minúsculas en el código, además de la característica del nombre del fichero. Para realizar este cambio, sólo es posible aplicarlo en el sistema “FreeRTOS” para evitar reiniciar el proyecto en Vitis. Debido a incompatibilidades con los “timers” usados, es necesario añadir también más memoria de “stack” para estos últimos. Ambos es posible modificarlos en la vista “Board Support Package” en el apartado de “freertos10...”, e incrementando el tamaño de

profundidad de stack para los temporizadores. En la figura inferior (Figuras 29a) se ve este valor que ha sido incrementado “*time\_task\_stack\_depth*” de 200 a 512. También es necesario activar el formato de “exFAT”, en la pestaña “xilffs” de la misma ventana (Figuras 29b). En este caso se indica “*enable\_exfat*” al valor “*true*”. Con esto, sería suficiente volver a compilar el proyecto entero para aplicar los cambios.

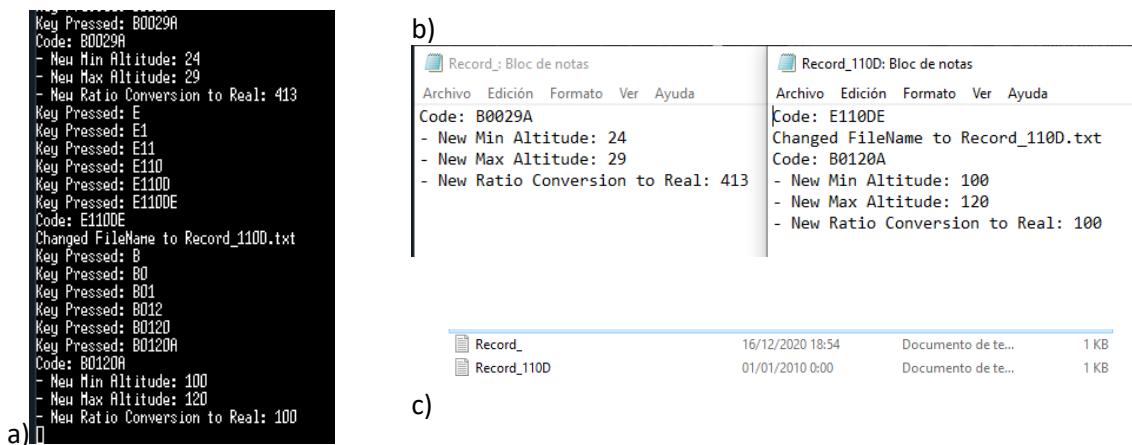
| Configuration for OS: freertos10_xilinx |                            |                            |
|---|----------------------------|----------------------------|
| Name                                    | Value                      | Default                    |
| SYSINTC_SPEC                            | *                          |                            |
| SYSTMR_DEV                              | *                          |                            |
| SYSTMR_SPEC                             | true                       |                            |
| stdin                                   | ps7_uart_1                 | none                       |
| stdout                                  | ps7_uart_1                 | none                       |
| > enable_stm_event_trace                | false                      | false                      |
| > hook_functions                        | true                       | true                       |
| > kernel_behavior                       | true                       | true                       |
| > kernel_features                       | true                       | true                       |
| > software_timers                       | true                       | true                       |
| timer_command_queue_length              | 10                         | 10                         |
| timer_task_priority                     | (configMAX_PRIORITIES ...) | (configMAX_PRIORITIES ...) |
| timer_task_stack_depth                  | 512                        | (configMINIMAL_STACK ...)  |
| use_timers                              | true                       | true                       |
| > tick_setup                            | true                       | true                       |

| Configuration for library: xilffs |         |         |
|-----------------------------------|---------|---------|
| Name                              | Value   | Default |
| enable_exfat                      | true    | false   |
| enable_multi_partition            | false   | false   |
| fs_interface                      | 1       | 1       |
| num_logical_vol                   | 2       | 2       |
| read_only                         | false   | false   |
| set_fs_rpath                      | 0       | 0       |
| use_chmod                         | false   | false   |
| use_lfn                           | 0       | 0       |
| use_mkfs                          | true    | true    |
| use_strfunc                       | 0       | 0       |
| word_access                       | true    | true    |
| > ramfs_size                      | 3145728 | 3145728 |

Figuras 29a y 29b: Configuración para establecer “exFAT” y “timer\_stack”, en “BSP Settings”; a)Configuración “timer\_stack”; b)Formato “exFAT”

Para realizar la prueba de esta característica, se adjuntan las siguientes figuras (Figuras 30abc). La prueba realizada es cambiar la escala del sensor de ultrasonidos, crear un nuevo archivo con otro nombre y establecer otra escala. En la figura de “TeraTerm”, (Figuras 30a), se ve esta ejecución, donde se establece una primera escala de 29 centímetros de techo de vuelo, se cambia el nombre a “Record\_110D” y se establece una escala a 120 centímetros. En la siguiente figura, (Figuras 30b), se ve el contenido de ambos ficheros junto con el mensaje de cambio de nombre de fichero. En la última figura, (Figuras 30c), se muestran ambos ficheros en el explorador del ordenador.



Figuras 30a, 30b y 30c: Prueba múltiples ficheros. a) Ejecución en "TeraTerm"; b) Contenido de Registros; c) Vista de los archivos

### 7.3.1.3. Tarea Acelerómetro (accelerometerTask)

Esta tarea está compuesta por 2 elementos, uno de ellos es la tarea periódica con una frecuencia de 250 milisegundos y prioridad 8, siendo una de las tareas más prioritarias. También dentro de esta tarea se llama a una función, “***readAccelerometer()***”, que obtiene los datos del sensor y activa los riesgos y alertas asociados a este dispositivo.

Para que no haya datos incompletos, la ejecución de la función asociada se realiza dentro del objeto protegido “simulado” para el dato, siendo accesible únicamente por el semáforo tipo Mutex “*xSemaphoreTilt*”, siendo el encargado de la inclinación. Al hacer uso de un acelerómetro por I2C (o IIC) se requiere definiciones adicionales para hacer funcionar esta comunicación, asociada al par AD6 del puerto PmodJA.

En el objeto protegido de acelerómetro se guarda el valor de la inclinación “*Acc[2]*” siendo los ejes X e Y respectivamente en cada posición. El riesgo específico “*SpecRiskTilt*” y la alerta “*alertTilt[32]*”, un array para almacenar la alerta a mostrar. También incluye contadores usados para evaluar o detectar los riesgos obtenidos al realizarse en la función indicada previamente. Entre las funciones declaradas, se encuentra “***setup()***” para inicializar el I2C y el dispositivo, y la función “***readAccelerometer()***” cuyo funcionamiento se detalla posteriormente. En el anexo A3.3.1 se ve estas definiciones y declaraciones.

En la tarea “***accelerometerTask***” se establece una activación periódica cada 250 milisegundos desde la anterior activación. También se comprueba que la variable global “*started*” es equivalente a “1”, indicando que el sistema se encuentra activo. Si es así, se recoge el semáforo respectivo de la tarea y su objeto protegido y se activa la función encargada de leer y detectar la inclinación, al finalizar devuelve el semáforo indicado. En el anexo A3.3.2 se ve el código de esta tarea periódica.

El siguiente elemento para analizar es la función encargada de leer y analizar o detectar la inclinación del sistema. La tarea hace uso de un array de “*float*” para almacenar la inclinación. El dispositivo utilizado (Acelerómetro) es un acelerómetro GY-521 que permite obtener la aceleración y giro en cada instante. Dado que se activará esta función cuando lo solicite la tarea, se tomará el ángulo obtenido en ese preciso instante.

Debido a un posible cambio en los decimales del acelerómetro que puede generar un cambio no deseado, se realizará la media de 3 lecturas consecutivas del sensor. Permitiendo ajustarse a un valor cercano al real, esto tampoco es capaz de analizar a la perfección el ángulo pero sí obtendrá un valor cercano al real. También la precisión a la cual trabaja el acelerómetro supone que un ligero cambio es capaz de cambiar el signo, por ese motivo el dispositivo se tiene que ajustar de manera manual para que dé un valor cercano a 0. Esto no es posible solucionarlo de manera software, debido a la sensibilidad y medidas obtenidas de manera matemática incluso el dispositivo inmóvil, es capaz de variar medio grado de inclinación.

El código utilizado es el que se puede ver en el anexo A3.3.3, éste es una adaptación para su uso con el I2C que ofrece la placa, dado que el sensor utilizado, al igual que la mayoría de los módulos y dispositivos tienen más contenido utilizando placas de la arquitectura de Arduino, en este caso por SinapTec [29]. La modificación del código adapta la funcionalidad I2C o IIC que ofrece la librería “xiic.h” de Xilinx al código. También se cree innecesario tomar el ángulo en función del tiempo porque esto puede generar un cambio gradual con la aceleración obtenida, porque al ser llamada sólo por la tarea de “Detectar Inclinación” no es relevante su medición en función del tiempo. De esta manera, se pierde precisión, pero es posible obtener valores en cada instante.

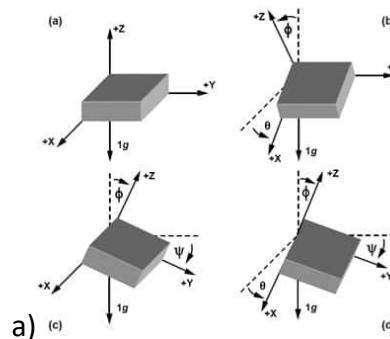
Como se comentó anteriormente se decide realizar 3 mediciones del sensor para obtener la media de ellos. Para ello se utiliza una variable auxiliar que tomará las mediciones incrementando el valor anterior. En cada ejecución la variable es restaurada a 0.0 para tomar una nueva medición.

El fragmento de código exclusivo de “xiic.h” es el envío y recepción de datos por medio de I2C. “Xlic\_Send”, enviar a la dirección del módulo y del dispositivo, indicada en una función “setup()”, un buffer, que en este caso permite hasta 128 bytes, en este caso, sólo requiere de 1 byte, que es 0x3B, la dirección donde debe comenzar la lectura para obtener la aceleración del dispositivo. Posteriormente, “Xlic\_Recv”, recibe de esa dirección, 6 bytes, que son almacenados en otro buffer de 128 bytes. De esta manera, se dispone de 6 bytes que dan información sobre la inclinación, esta dirección viene indicada en el datasheet del MPU6050 [30].

Debido a que la aceleración y giro constan de 2 bytes, 16 bits en total, se debe agruparlos en las variables indicadas para ello. En este caso, es suficiente con la lectura de los bytes recibidos.

El cálculo de la inclinación o del ángulo, es la fórmula para obtener la inclinación desde una aceleración (Figuras 31a). Las fórmulas se pueden deducir de razones trigonométricas funcionando con una fuerza G de 1, obtenido de las siguientes formulas (Figuras 31b y c). También es posible aplicar esta razón trigonométrica al eje Z, pero el resultado obtenido no se ajusta a la inclinación real de este eje.

El motivo de utilizar una ratio de conversión “A\_R” (*Acceleration\_ratio*), se debe al valor que se obtiene por el sensor, da un valor entre -2G y 2G de fuerza gravitacional para la aceleración, para dar valores en fuerza gravitacional, se debe hacer una conversión por 16384, dado que el rango de los valores es desde -32767 hasta 32767. De esta manera se obtiene un valor de fuerza cercano a 1G para obtener la inclinación del sistema.



$$b) \theta = \tan^{-1} \left( \frac{A_{X,OUT}}{\sqrt{A_{Y,OUT}^2 + A_{Z,OUT}^2}} \right) \quad (11)$$

$$c) \psi = \tan^{-1} \left( \frac{A_{Y,OUT}}{\sqrt{A_{X,OUT}^2 + A_{Z,OUT}^2}} \right) \quad (12)$$

Figuras 31a, 31b, 31c: Ángulo en función de la inclinación; a) Figuras de su obtención; b) Formula para el eje X; c) Formula para el eje Y [31]

Finalmente, se divide el valor acumulado para hacer la media de cada ángulo, el orden de los ejes es, primero el eje Y (cabecero o *Tilt*) y el segundo es el equivalente al eje X (alabeo o *Roll*).

La medida obtenida es la almacenada en el array del “objeto protegido” del acelerómetro. En la comparación, se comprueba en primer lugar si esa inclinación se encuentra sobre pasando los límites establecidos de [-30, 30] en el eje X, incrementando el contador “AxisX” o estableciéndolo a 0 en caso de encontrarse dentro. Del mismo modo se evalúa en el eje Y con un rango admitido de [-20, 20] en este eje, con su contador “AxisY”. Esto se establece de esta manera para contabilizar las veces que se sobre pasan los límites, estableciendo un máximo de 20, aunque el máximo evaluable es 6.

El siguiente paso es identificar si se debe activar o no el riesgo específico de inclinación, con su respectiva alerta. Para ello se evalúa si el riesgo específico es distinto a 3, debido a que sí se encuentra en riesgo de nivel 3 o “crítico”, el piloto automático desactiva la alerta y riesgo pasado un tiempo establecido en 15 segundos. No debería ser posible modificar los campos ya que no tendrían efecto alguno.

La primera comprobación se realiza si la inclinación es la adecuada, dentro de los límites indicados. En ese caso, se incrementa el valor del contador de lecturas correctas consecutivas hasta un máximo de 20. Si este contador equivale a 2 (2 lecturas consecutivas correctas), se realiza una rutina para comprobar si debe ser reducido un nivel de riesgo específico, si el nivel específico es 2 “Riesgo Grave”, se ve reducido al nivel 1 “Riesgo Leve”, modificando su alerta a “*Rectified Angle*” (“Angulo Rectificado”), reiniciando el contador de lecturas correctas consecutivas. Si en su defecto existía un nivel de riesgo 1 “Leve”, se establece el nivel de riesgo a 0 “Sin Riesgo” y se elimina la alerta, en este caso, en contador no tiene por qué ser reiniciado, en caso de ser un riesgo nulo o crítico, no hay cambios asociados.

En caso de no ser ambos contadores “AxisX” y “AxisY” nulos, se reinicia el contador de lecturas correctas consecutivas, y se comienza con el caso donde “AxisX” o “AxisY” es menor que 3, sólo 1 es mayor o igual a 3, en este caso, se comprueba si la inclinación es mayor en el eje X o en el eje Y. En ambos casos la comparación es igual, si son mayores o iguales a 3, se debe comprobar si son mayores o iguales a 5. Si son mayores de 5, se activa el riesgo específico de inclinación al nivel 2 “Grave” sin modificar la alerta ya registrada. En caso contrario (entre 3 y 5) el riesgo específico almacenado es 1 “Leve”, y se establece la alerta “*High Roll Angle*” o “*High Pitch Angle*”, dependiendo del eje.

En el caso de ser ambos contadores de “lecturas erróneas” en los ejes superiores a 3, indica que el sistema se encuentra fuera de cualquier límite de inclinación. Si ambos contadores son a su vez superiores a 5, se activa el riesgo específico de nivel 3 “Crítico”, si es menor de 5, se activa el riesgo específico a nivel 2 “Grave” y la alerta “*High Roll and Pitch Angle*”. Todas las activations de riesgo son registradas en la tarjeta microSD con la alerta en el formato “[tiempo en ms]] [alerta] - Level: [nivel]\r\n”.

Se indica además en este apartado, el *setup* del dispositivo y las definiciones relevantes. En primer lugar se define el valor de “*IIC\_DEVICE*” que equivale a la dirección que tiene la placa FPGA para el módulo “*XPAR\_AXI\_IIC\_0\_DEVIDE\_ID*” o “*Axi\_IIC*” del diseño. También es necesario indicar la dirección del dispositivo *MPU6050* del sensor, en este caso es 0x68, por el *datasheet* del MPU [30]. El ratio de conversión utilizado es “*A\_R*” con un valor de **16384.0** (equivalente a 1G por el sensor). Por último, hay que definir la conversión de radianes a grados, para ello se pone el valor de “ $180/\pi$ ”, 57.295779. La función *setup*, es la que se indica en el siguiente anexo (A3.3.4).

En él se puede ver la configuración de un fichero para indicar el funcionamiento de éste, además requiere de la activación de mismo, haciendo uso de la dirección 0x6B y 0x00 para activar el dispositivo, esto se hace, al igual que la lectura de IIC anterior, por el *datasheet* del *MPU6050*. Con esta función activada, se tiene el dispositivo activado, esperando a recibir y enviar datos por medio de I2C.

### 7.3.1.3.1. Pruebas

Para realizar las pruebas, se parte de tener activas las tareas del lector de teclado y la tarea de utilizar sistema. En la prueba, se mostrará pasada la ejecución el contenido de las variables del objeto protegido del acelerómetro, junto con sus alertas y riesgos específicos.

Al ser una tarea con un periodo reducido, se mostrará únicamente un fragmento de la ejecución en las siguientes figuras (Figuras 32abcd). En la prueba se inclinará el prototipo para llegar a superar los límites establecidos, que harán tras varias lecturas erróneas, la activación del riesgo específico, junto con su alerta. El mensaje mostrado, en el prototipo final será registrado en la tarjeta microSD en un fichero para una posterior verificación.

```

a) VT Tera Term - [disconnected] VT
File Edit Setup Control Window Help
Alert: ""
Actual Tilt Angle (X, Y): (-5.08, -4.43)
Counters (Correct, Wrong AxisX, Wrong AxisY): (0, 0, 0)

Specific Risk (Accelerometer/Tilt):
Risk Level: 0 (No Risk)
Alert: ""
Actual Tilt Angle (X, Y): (5.11, -6.91)
Counters (Correct, Wrong AxisX, Wrong AxisY): (10, 0, 0)

Specific Risk (Accelerometer/Tilt):
Risk Level: 0 (No Risk)
Alert: ""
Actual Tilt Angle (X, Y): (0.97, -19.26)
Counters (Correct, Wrong AxisX, Wrong AxisY): (11, 0, 0)

Specific Risk (Accelerometer/Tilt):
Risk Level: 0 (No Risk)
Alert: ""
Actual Tilt Angle (X, Y): (-1.01, -27.10)
Counters (Correct, Wrong AxisX, Wrong AxisY): (0, 0, 1)

b) VT Tera Term - [disconnected] VT
File Edit Setup Control Window Help
Alert: "High Roll Angle"
Actual Tilt Angle (X, Y): (-29.21, 15.09)
Counters (Correct, Wrong AxisX, Wrong AxisY): (1, 0, 0)

Specific Risk (Accelerometer/Tilt):
Risk Level: 2 (Severe Risk)
Alert: "High Roll Angle"
Actual Tilt Angle (X, Y): (-25.84, 22.46)
Counters (Correct, Wrong AxisX, Wrong AxisY): (0, 0, 1)

Specific Risk (Accelerometer/Tilt):
Risk Level: 2 (Severe Risk)
Alert: "High Roll Angle"
Actual Tilt Angle (X, Y): (-21.99, 32.73)
Counters (Correct, Wrong AxisX, Wrong AxisY): (0, 0, 2)

Specific Risk (Accelerometer/Tilt):
Risk Level: 1 (Light Risk)
Alert: "High Pitch Angle"
Actual Tilt Angle (X, Y): (-18.45, 38.68)
Counters (Correct, Wrong AxisX, Wrong AxisY): (0, 0, 3)

c) VT Tera Term - [disconnected] VT
File Edit Setup Control Window Help
Alert: "High Roll and Pitch Angle"
Actual Tilt Angle (X, Y): (-31.93, 32.78)
Counters (Correct, Wrong AxisX, Wrong AxisY): (0, 20, 20)

Specific Risk (Accelerometer/Tilt):
Risk Level: 3 (Critical Risk)
Alert: "High Roll and Pitch Angle"
Actual Tilt Angle (X, Y): (-31.04, 33.85)
Counters (Correct, Wrong AxisX, Wrong AxisY): (0, 20, 20)

Specific Risk (Accelerometer/Tilt):
Risk Level: 3 (Critical Risk)
Alert: "High Roll and Pitch Angle"
Actual Tilt Angle (X, Y): (-28.14, 30.32)
Counters (Correct, Wrong AxisX, Wrong AxisY): (0, 0, 20)

Specific Risk (Accelerometer/Tilt):
Risk Level: 3 (Critical Risk)
Alert: "High Roll and Pitch Angle"
Actual Tilt Angle (X, Y): (-29.09, 10.33)
Counters (Correct, Wrong AxisX, Wrong AxisY): (0, 0, 0)

d) RECORD3A: Bloc de notas
Archivo Edición Formato Ver Ayuda
Code: C1234A
- System Working
[9500] High Pitch Angle - Level: 1
[9750] High Pitch Angle - Level: 1
[10000] High Pitch Angle - Level: 2
[10250] High Pitch Angle - Level: 2
[10500] High Pitch Angle - Level: 2
[10750] High Pitch Angle - Level: 2
[11000] High Pitch Angle - Level: 2
[11250] High Pitch Angle - Level: 2
[11500] High Pitch Angle - Level: 2
[11750] High Pitch Angle - Level: 2
[12000] High Pitch Angle - Level: 2
[12250] High Pitch Angle - Level: 2
[12500] Rectified Angle - Level: 1
[13750] No Risk Angle
[14500] High Pitch Angle - Level: 1
[14750] High Pitch Angle - Level: 1
[15000] High Pitch Angle - Level: 2
[15250] High Pitch Angle - Level: 2
[15500] High Pitch Angle - Level: 2
[15750] High Pitch Angle - Level: 2
[16000] High Pitch Angle - Level: 2
[16250] High Pitch Angle - Level: 2
[16500] High Pitch Angle - Level: 2
[16750] High Pitch Angle - Level: 2
[17250] Rectified Angle - Level: 1
[17750] No Risk Angle
[23750] High Roll Angle - Level: 1
[24000] High Roll Angle - Level: 1
[24250] High Roll Angle - Level: 2

```

Figuras 32a, 32b, 32c y 32d: Prueba "accelerometerTask". a, b, c) TeraTerm d)Registro en microSD

En la Figuras 32a se quiere indicar que una inclinación superior a 20 grados en cualquier sentido del eje Y, resetea los contadores, también no llega a activarse una situación específica al no alcanzar las lecturas erróneas consecutivas. En la Figuras 32b,

muestra el cambio de alerta y situación que ocurre al inclinarse en el eje Y por encima de los límites establecidos y alcanzar 3 lecturas consecutivas. En la Figuras 32c se indica que una vez alcanzado el nivel crítico de alerta, el sistema mantiene este riesgo específico hasta su desactivación automática.

Por último, en la Figuras 32d se muestra parte del registro de la tarjeta microSD, en éste se puede ver un timespan de cuando se activa cada alerta junto con su nivel específico asociado. También se puede ver la rectificación de la inclinación y que el periodo indicado se respeta de manera adecuada.

#### *7.3.1.4. Tarea Sensor de Ultrasonidos (UltrasonicTask)*

Esta tarea al igual que la anterior también está formada por dos códigos, en primer lugar, la tarea periódica con un periodo de 300 milisegundos y una prioridad de 5, siendo la menos prioritaria entre las tareas de los sensores. Del mismo modo que otras tareas que utilizan los sensores, dispone de un objeto protegido accesible por medio de un semáforo mutex, en este caso es “*xSemaphoreAltitude*”. Este objeto protegido es detallado en las declaraciones respectivas, mostradas en el siguiente anexo (A3.4.1).

Su funcionamiento es similar al utilizado por la tarea del acelerómetro, sólo que en este caso se hace uso del sensor de ultrasonidos, obteniendo la altitud a la cual se encuentra el sistema. Para ello se utiliza el sensor de ultrasonidos HC-SR04, comentado previamente en los dispositivos utilizados, el cual activa la función “*readUltrasonicSensor()*”. El código de la tarea se puede ver en el siguiente anexo (A3.4.2).

Para poder utilizar este sensor la función debe activar una señal de “Trigger” de 10 microsegundos ( $\mu s$ ). Debido a que la medición de tiempo de FreeRTOS actúa sobre los milisegundos, se debe hacer uso de las funcionalidades que ofrece la propia placa FPGA, usando la función “*usleep()*”.

En Arduino, es posible obtener la medición de tiempo por medio de la función “*PulseIn()*”. Debido a que Xilinx no dispone de una función similar, se ha buscado una manera de replicarlo en la placa FPGA. En primer lugar, se esperará, hasta que la señal del pin “Echo” del sensor se encuentre apagada usando un bucle *while* mientras ésta se encuentre a 0. El motivo de esto es para contar sólo el tiempo en el cual se encuentra activa la señal posteriormente. Tras salir del bucle, se toma el tiempo inicial usando “*XTime\_GetTime(&tStart)*”, esta función pertenece a la librería “*xtime\_l.h*” de Xilinx. El uso de esta función, a diferencia de usar otros métodos de C o de FreeRTOS dispone de la ventaja de medir los ciclos del propio procesador, permitiendo mediciones más precisas de tiempo. Se valoró utilizar la librería “*time.h*” pero el sistema no es capaz de indicar una fecha correcta, en caso de conseguir que funcione

la obtención del tiempo. Por tanto, se ha optado por utilizar la herramienta de Xilinx para medir el tiempo en cada instante.

El siguiente paso, sería mantener un bucle mientras la señal se encuentre activa en el pin “Echo” del dispositivo, también se ha optado por utilizar un bucle *while*. Al salir de este segundo bucle, se vuelve a tomar el tiempo, en este caso, el tiempo final de ejecución. Dado que el valor obtenido son los ciclos, que puede ser contradictorio al utilizar la función “*XTime\_GetTime()*”, se debe operar para obtener el tiempo en microsegundos que se ha utilizado finalmente. Para obtenerlo, primero se obtiene el número de ciclos que ha durado la señal, este valor se divide entre los ciclos por microsegundo, obtenidos al dividir los ciclos por segundo “*COUNTS\_PER\_SECOND*” entre 1000000 (un millón). Con esta operación se dispone del tiempo en microsegundos que ha tardado en desactivarse el pin “Echo” del sensor.

Finalmente, para obtener la distancia detectada en función del tiempo, se divide entre 59. Este valor es obtenido de convertir la velocidad del sonido (343-344 m/s) a cm/μs. Contando también que se debe dividir entre 2 porque el sonido realizará la ida y la vuelta. Para dar un valor entero, se decide hacer una división en lugar de la multiplicación, también el valor obtenido es 58.30, se decide redondearlo por arriba, obteniendo 59. La ecuación que muestra esto, es la siguiente (Ecuación 1).

$$\text{sonido en cm} \rightarrow \frac{\left(343 \text{ (m/s)} * \left(\frac{1\text{s}}{1000000\mu\text{s}}\right) * \left(\frac{100\text{cm}}{1\text{m}}\right)\right)}{2} = 0.01715 \rightarrow \frac{1}{\text{sonido en cm}} = \frac{1}{0.01715} = [58.30] = 59$$

Ecuación 1: valor entero de la velocidad del sonido.

Dado que los valores obtenidos, son más precisos, pero pueden tener errores, se realizará una media de la altura en 3 obtenciones, esto será realizado con un bucle *for* de 3, para obtener la suma de los valores y una división posterior que devuelva la media de la distancia tomada. Este valor será redondeado automáticamente para trabajar con tipo *long int*. Entre cada lectura del sensor, se realizará un breve descanso de 20 μs, para evitar una lectura constante. En la primera parte del código del anexo 0 se puede ver la secuencia explicada.

Del mismo modo que hace la función análoga del ultrasonidos, también se encarga de hacer una comparación sobre la altitud obtenida. Únicamente funcionará si el riesgo específico es distinto a 3, dado que si fuese un riesgo general crítico (nivel 3), el autopiloto se situaría en la altura adecuada y se desactivaría automáticamente. Si no se cumple esta condición, la función compara el valor medido con la altura mínima del sistema, que simula ser en el prototipo 10000 metros aplicado a una escala personalizable por código. Si esta condición se cumple, el contador incrementado hasta un valor de 20 es el de “Altitud por debajo”, en la posición 0 del array de contadores erróneos. El resto de los contadores se reinicia a 0, dado que no puede estar en sistema en dos alturas simultáneamente. Al llegar a un valor igual o superior a 3, comprobará si el contador es igual o superior a 6, para activar un nivel u otro, y establecer la alerta “Low Altitude”. En este caso puede activarse el nivel 1 “Leve” o el nivel 2 “Grave” si se mantiene durante más tiempo la altitud por debajo.

De manera análoga funciona la condición de altitud entre los límites de 10000 y 12000 metros, que en el prototipo son escalados, y la condición de mantener una altura por encima de los 12000, siendo el techo de vuelo. En la condición de mantenerse en los límites indicados, el contador aumentado mide las lecturas con valores correctos para reducir o rectificar los niveles específicos, al igual que en el acelerómetro, no hay respuesta asociada con el nivel 3 “Crítico” ni el nivel 0 “Sin Riesgo/Normal”. La última condición sobre la altitud es que el sistema se sitúe por encima del techo de vuelo, en este caso, el funcionamiento es igual al realizado si el sistema se encontrase por debajo, en este caso, se activa el riesgo 2 “Grave” al llegar a 3 lecturas consecutivas, y a nivel 3 “Crítico” si persiste otras 3 lecturas consecutivas. En este último caso la alerta es “High Altitude”. El código se puede consultar en el siguiente anexo (A3.4.3).

En este caso, el dispositivo HC-SR04 de ultrasonidos funciona por salida digital y su inicialización se ha realizado en la función “setupGPIO()” ya mostrada en el anexo (A3.1.3).

#### 7.3.1.4.1. Pruebas

Para probar esta tarea, se necesitará como en la anterior, la activación por parte de un código introducido de manera manual, además de ajustar la altura que será detectada para verificar que los cambios realizados y las alertas se activan en los momentos adecuados. En la prueba, se mostrará el contenido de los elementos del objeto protegido para verificar su funcionamiento. Siendo posible porque todas las tareas que leen los sensores utilizan su objeto protegido durante toda su ejecución.

En las siguientes figuras (Figuras 33abcd), se pueden ver varios fragmentos de la ejecución por medio del display (TeraTerm) y el registro posteriormente almacenado en la microSD. En la Figura 33a, se indica el cambio de nivel realizado al mantener por un tiempo prolongado la altitud por debajo del límite inferior, ajustado en este caso al prototipo a 41 cm reales. En la segunda figura, Figura 33b, el objetivo es mostrar el cambio de alerta y nivel de riesgo específico producido por varias lecturas consecutivas. En la última figura del display, Figura 33c, se muestra la activación de la alerta específica si se mantiene una altitud superior al techo de vuelo, en este caso del prototipo, 50cm reales.

Finalmente, en la Figura 33d, se muestra parte del registro en la tarjeta microSD, en él, es posible apreciar el periodo de la tarea para verificar que no existen lecturas consecutivas, también que las alertas activas son las que deberían ser, sin ningún valor fuera de lo normal.

```

a) Specific Risk (Altimeter/Altitude):
Risk Level: 1 (Light Risk)
Alert: "Low Altitude"
Actual Altitude "Simulated" (Real): 8880 m (37 cm)
Set Altitude (Min - Max): (41 - 50)
Counters (Wrong Low, Correct, Wrong High): (4, 0, 0)

Specific Risk (Altimeter/Altitude):
Risk Level: 1 (Light Risk)
Alert: "Low Altitude"
Actual Altitude "Simulated" (Real): 8880 m (37 cm)
Set Altitude (Min - Max): (41 - 50)
Counters (Wrong Low, Correct, Wrong High): (5, 0, 0)

Specific Risk (Altimeter/Altitude):
Risk Level: 2 (Severe Risk)
Alert: "Low Altitude"
Actual Altitude "Simulated" (Real): 8880 m (37 cm)
Set Altitude (Min - Max): (41 - 50)
Counters (Wrong Low, Correct, Wrong High): (6, 0, 0)

b) Specific Risk (Altimeter/Altitude):
Risk Level: 2 (Severe Risk)
Alert: "Low Altitude"
Actual Altitude "Simulated" (Real): 10800 m (43 cm)
Set Altitude (Min - Max): (41 - 50)
Counters (Wrong Low, Correct, Wrong High): (0, 1, 0)

Specific Risk (Altimeter/Altitude):
Risk Level: 2 (Severe Risk)
Alert: "Low Altitude"
Actual Altitude "Simulated" (Real): 10320 m (43 cm)
Set Altitude (Min - Max): (41 - 50)
Counters (Wrong Low, Correct, Wrong High): (0, 2, 0)

Specific Risk (Altimeter/Altitude):
Risk Level: 1 (Light Risk)
Alert: "Rectified Altitude"
Actual Altitude "Simulated" (Real): 10320 m (43 cm)
Set Altitude (Min - Max): (41 - 50)
Counters (Wrong Low, Correct, Wrong High): (0, 0, 0)

c) Set Altitude (Min - Max): (41 - 50)
Counters (Wrong Low, Correct, Wrong High): (0, 0, 1)

Specific Risk (Altimeter/Altitude):
Risk Level: 0 (No Risk)
Alert: ""
Actual Altitude "Simulated" (Real): 12480 m (52 cm)
Set Altitude (Min - Max): (41 - 50)
Counters (Wrong Low, Correct, Wrong High): (0, 0, 2)

Specific Risk (Altimeter/Altitude):
Risk Level: 2 (Severe Risk)
Alert: "High Altitude"
Actual Altitude "Simulated" (Real): 12480 m (52 cm)
Set Altitude (Min - Max): (41 - 50)
Counters (Wrong Low, Correct, Wrong High): (0, 0, 3)

Specific Risk (Altimeter/Altitude):
Risk Level: 2 (Severe Risk)
Alert: "High Altitude"
Actual Altitude "Simulated" (Real): 12480 m (52 cm)
Set Altitude (Min - Max): (41 - 50)
Counters (Wrong Low, Correct, Wrong High): (0, 0, 4)

d) [21900] Low Altitude - Level: 1
[22000] Low Altitude - Level: 2
[22500] Low Altitude - Level: 2
[22800] Low Altitude - Level: 2
[23100] Low Altitude - Level: 2
[24000] Rectified Altitude - Level: 1]
[24900] No Risk Altitude
[27000] Low Altitude - Level: 1
[27300] Low Altitude - Level: 1
[27600] Low Altitude - Level: 1
[27900] Low Altitude - Level: 2
[28200] Low Altitude - Level: 2
[28500] Low Altitude - Level: 2
[28800] Low Altitude - Level: 2
[31200] Rectified Altitude - Level: 1
[32100] No Risk Altitude
[37800] High Altitude - Level: 2
[38110] High Altitude - Level: 2
[38410] High Altitude - Level: 2
[38700] High Altitude - Level: 3

```

Figuras 33a, 33b, 33c, 33d: Pruebas de la tarea "ultrasonicTask"; a, b, c)Muestras del display "TeraTerm"; b) Registro en la microSD

### 7.3.1.5. Tarea Velocímetro

La siguiente tarea encargada de obtener un valor de los sensores, obtendrá el valor del potenciómetro horizontal que incluye el prototipo. Dado que el potenciómetro dará un valor fiable, no se requerirán varias lecturas de éste para conocer la velocidad real. En el caso práctico, es posible que el velocímetro incorpore algún error que requiera varias lecturas. La tarea tiene un periodo de 500 milisegundos y una prioridad de 7, la segunda más prioritaria de las tareas que leen los sensores. Su estructura es igual a las anteriores tareas, dispone de un objeto protegido con las variables utilizadas, al cual se puede acceder por el uso del semáforo mutex "*xSemaphoreSpeed*". También del mismo modo, dispone de un array para los contadores erróneos, además de la alerta y la situación específica del sensor "Velocímetro". Los códigos de la declaración de variables del objeto protegido y de la tarea se pueden ver en los siguientes anexos A3.5.1 y A3.5.2, respectivamente.

Esta tarea hace uso de la función "*readVelocimeter*", esta función obtiene el valor analógico del potenciómetro conectado al par analógico AD14, y posteriormente analiza el dato recogido para obtener una situación de riesgo y alerta asociada. El potenciómetro horizontal devuelve un valor entero entre 0 y 65535, debido a la limitación de potencia del puerto Pmod analógico (1.1V), se le aplica un circuito divisor de potencia. En este caso el utilizado, reduce la potencia de 3.3V a 1 V, para ello usa tres resistencias, dos agrupadas para obtener 230Ω y otra resistencia de 100Ω. De

esta manera, la potencia de 1V es capaz de ser dividida proporcionalmente entre 65535. Al utilizar el puerto analógico, se ha elegido usar la entrada AD14, situada en los pines 1 y 5 del puerto. Al tratarse de un par analógico, uno de los pines se conecta a Tierra para establecer un punto de comparación en la placa.

Para obtener el valor de la velocidad, hay que tener en cuenta que el sistema acepta por limitaciones de precisión del potenciómetro únicamente velocidades entre 754 y 900 km/h para que se pueda manejar y modificar la velocidad con mayor precisión. De esta manera se puede ampliar el rango de la velocidad adecuadas, centrándola en el punto de crucero 827 km/h en mitad del potenciómetro. Para obtener la velocidad adecuada, se operará el valor entre 0 a 65535 asignándole valores entre 0 y 146 (diferencia entre 754 y 900). Finalmente, se le sumará 754 que es la velocidad mínima aceptada.

Con la velocidad obtenida, ésta es almacenada en el objeto protegido para ser leído por el display y el registro. Al igual que en los anteriores casos, si la situación específica es 3 o crítica no se actualiza los valores, dado que estos serán reiniciados por el piloto automático restringido. El sistema calificará la velocidad obtenida en 5 rangos desde “Velocidad muy baja (*Too Low Speed*)” hasta “Velocidad muy alta (*Too High Speed*)”. Estos rangos han sido divididos equitativamente a lo largo de todo el potenciómetro, estos son  $[[754.0, 783.2), [783.2, 812.4), [812.4, 841.6), [841.6, 870.8), [870.8, 900.0]]$ .

Debido a que la manera de calificar cada situación es prácticamente igual en los casos errores, con la diferencia de los niveles de riesgo específicos activados. En todos los casos erróneos, se reinician el resto de los valores, y se incrementará hasta un máximo de 20 su propio contador. Al registrar tres lecturas consecutivas erróneas, se activará la situación de riesgo específico, en los extremos la situación activada es de nivel 2 “Grave”, los siguientes rangos son de nivel 1 “Leve”, y el rango central, es el rango nulo, donde se puede rectificar el valor establecido. Cada uno de los valores erróneos, excepción del rango de velocidad adecuada, sólo tiene un nivel de criticidad, con una alerta asociada en cada caso, registrando todos los mensajes y activaciones en el registro de la microSD.

En el caso de la lectura correcta consecutiva en el rango de velocidades adecuadas, al llegar el contador a 3, se puede reducir 1 nivel de la situación específica actual, si ésta no es 3 o 0. Estos cambios también se registran en la tarjeta microSD. Este código se puede consultar en el anexo A3.5.3. Al utilizar entrada analógica, ésta no requiere ser declarada dispositivo internamente en el código, únicamente conocer que se utiliza el módulo “XADC Wizard” en el diseño creado.

### 7.3.1.5.1. Pruebas

Las pruebas de esta tarea se realizan de la misma manera que las dos anteriores, haciendo uso del dispositivo se mostrará el contenido del objeto protegido para verificar el correcto funcionamiento. En este caso, al medir la velocidad por medio de un potenciómetro, se realizará un barrido por las distintas situaciones, se pueden ver capturas de la prueba realizada en las siguientes figuras (Figuras 34abcd).

```

a) Specific Risk (Velocimeter/Speed):
Risk Level: 2 (Severe Risk)
Alert: "Too High Speed"
Actual Speed: 834.95 km/h
Counters (Wrong Too Low, Wrong Low, Correct, Wrong High, Wrong Too High): (0, 0, 1, 0, 0)

Specific Risk (Velocimeter/Speed):
Risk Level: 1 (Light Risk)
Alert: "Rectified Speed"
Actual Speed: 824.60 km/h
Counters (Wrong Too Low, Wrong Low, Correct, Wrong High, Wrong Too High): (0, 0, 0, 0, 0)

Specific Risk (Velocimeter/Speed):
Risk Level: 1 (Light Risk)
Alert: "Rectified Speed"
Actual Speed: 824.51 km/h
Counters (Wrong Too Low, Wrong Low, Correct, Wrong High, Wrong Too High): (0, 0, 1, 0, 0)

b) Specific Risk (Velocimeter/Speed):
Risk Level: 1 (Light Risk)
Alert: "High Speed"
Actual Speed: 866.86 km/h
Counters (Wrong Too Low, Wrong Low, Correct, Wrong High, Wrong Too High): (0, 0, 0, 5, 0)

Specific Risk (Velocimeter/Speed):
Risk Level: 1 (Light Risk)
Alert: "High Speed"
Actual Speed: 880.56 km/h
Counters (Wrong Too Low, Wrong Low, Correct, Wrong High, Wrong Too High): (0, 0, 0, 0, 1)

Specific Risk (Velocimeter/Speed):
Risk Level: 1 (Light Risk)
Alert: "High Speed"
Actual Speed: 888.36 km/h
Counters (Wrong Too Low, Wrong Low, Correct, Wrong High, Wrong Too High): (0, 0, 0, 0, 2)

c) Specific Risk (Velocimeter/Speed):
Risk Level: 2 (Severe Risk)
Alert: "Too Low Speed"
Actual Speed: 781.13 km/h
Counters (Wrong Too Low, Wrong Low, Correct, Wrong High, Wrong Too High): (7, 0, 0, 0, 0)

Specific Risk (Velocimeter/Speed):
Risk Level: 2 (Severe Risk)
Alert: "Too Low Speed"
Actual Speed: 785.09 km/h
Counters (Wrong Too Low, Wrong Low, Correct, Wrong High, Wrong Too High): (0, 1, 0, 0, 0)

Specific Risk (Velocimeter/Speed):
Risk Level: 2 (Severe Risk)
Alert: "Too Low Speed"
Actual Speed: 792.75 km/h
Counters (Wrong Too Low, Wrong Low, Correct, Wrong High, Wrong Too High): (0, 2, 0, 0, 0)

d) [13000] Too Low Speed - Level: 2
[13500] Too Low Speed - Level: 2
[15000] Low Speed - Level: 1
[15500] Low Speed - Level: 1
[16000] Low Speed - Level: 1
[16500] Low Speed - Level: 1
[17000] Low Speed - Level: 1
[18000] No Risk Speed
[21500] Low Speed - Level: 1
[25000] High Speed - Level: 1
[25500] High Speed - Level: 1
[26000] High Speed - Level: 1
[27500] Too High Speed - Level: 2
[28000] Too High Speed - Level: 2
[28500] Too High Speed - Level: 2
[30500] Low Speed - Level: 1
[31000] Low Speed - Level: 1
[31500] Low Speed - Level: 1
[32000] Low Speed - Level: 1
[32500] Low Speed - Level: 1
[33000] Low Speed - Level: 1
[33500] Low Speed - Level: 1
[34000] Low Speed - Level: 1
[36000] Too High Speed - Level: 2
[36500] Too High Speed - Level: 2
[37000] Too High Speed - Level: 2
[38000] Rectified Speed - Level: 1
[39000] No Risk Speed

```

Figuras 34a, 34b, 34c, 34d: Prueba de "velocimeterTask"; a, b, c)Salida por display "TeraTerm"; b) Registro en tarjeta microSD

En la figura anterior, Figuras 34a, se quiere mostrar la rectificación de un caso poco usual, pasar de un riesgo 2 a leer lecturas correctas, en este caso, se rectificará el mensaje y la situación específica. Al ser poco usual, pero posible de realizar, se valora esa situación. En la Figuras 34b, el objetivo es mostrar cómo el contador de lecturas correctas del rango de “Alta Velocidad” se reinicia al leer una velocidad del rango “Muy Alta Velocidad”. En la siguiente ejecución, la alerta será actualizada y el riesgo específico aumentará. La última figura del display, Figuras 34c, muestra la aumento de una velocidad “Muy Baja” a “Baja”, es decir, la operación inversa a la Figuras 34b.

Finalmente, en el registro de la Figuras 34d se quiere mostrar los diferentes niveles alcanzables por el dispositivo, el cumplimiento del periodo, y la mecánica de “rectificar” la velocidad, que en este caso se puede ver de manera más clara. Hay que añadir que en el “timespan” de 18000 la situación se desactiva al cambiar de una situación leve, siendo lo normal en este caso.

### 7.3.1.6. Tarea Presencia

La última tarea que guarda relación con los sensores es el control de presencia en cabina, de forma reducida se llamará “Presencia”. Esta tarea obtendrá la posible presencia en la cabina del avión. Está formado por dos dispositivos detectores de presencia y un detector de movimiento. El par de detectores de presencia son los encargados de identificar que los pilotos se encuentran en sus asientos. El detector de movimiento sólo indicará un posible movimiento cerca de la puerta a la cabina. Principalmente, y debido a normativas de vuelo actuales, debe existir un mínimo de 2 personas en cabina. Esta aclaración se ha extendido para el proyecto, indicando posibles distracciones en caso de existir más de 2 personas en cabina, con el objetivo de no distraer a los pilotos durante el vuelo.

La tarea dispone de un periodo de 600 milisegundos, y una prioridad de nivel 6. También, al igual que las anteriores tareas hace uso de un objeto protegido “simulado”, el cual es accesible por un semáforo mutex, en este caso “*xSemaphorePresence*”. Tanto las declaraciones y definiciones como el cuerpo de la tarea se pueden ver en los anexos A3.6.1 y A3.6.2, respectivamente. Dentro de la tarea se hace uso de la función “*readPresence*”, que obtendrá valores que deben ser analizados, es la encargada de indicar la presencia en cabina.

Para ello en primer lugar se contabiliza los pilotos que se encuentran en cabina, además de indicar si existe algún movimiento cercano a la puerta de la cabina. Esto último puede indicar que existe más gente en la cabina pero no es capaz de especificar cuanta, por tanto se contabilizará cómo 1 persona en la cabina. Dado que los sensores son digitales, y no trabajan con tiempo o decimales, se puede especificar que no requiere de lecturas adicionales.

Posteriormente, se hará una comprobación de cuantas personas o individuos se encuentran en la cabina, en el caso ideal, y es la primera comprobación, sólo se encuentran 2 personas. Estas pueden ser los dos pilotos, o un piloto y alguien cerca de la puerta. En este caso, se reiniciará el contador de las lecturas erróneas, y se incrementará en 1 el contador de lecturas correctas. También se evaluará, en caso de acumular más de 3 lecturas correctas consecutivas, una reducción del nivel de riesgo, éste sólo será efectivo con lecturas correctas, es decir, cuando hay 2 personas en la cabina, una de ellas, al menos debe estar sentada en el asiento de uno de los pilotos.

En caso de existir un número de personas diferente a 2, se comprobará si hay menos de 2 personas, incluyendo que no haya nadie en la cabina. En este caso, la función reiniciará el contador de lecturas correctas y será valorado como un riesgo grave, pero deberá ser confirmado por medio de 3 lecturas consecutivas, si estas se cumplen, el nivel de riesgo específico respecto a la presencia “*specRiskPresence*” será activado al nivel “Grave” o valor numérico “2”, activando además la alerta “Low Presence”. Si esta situación persiste durante otras 3 lecturas incorrectas, el nivel será

aumentado al nivel “Crítico” o “3”, el cual activará tras obtenerse por Análisis de Riesgos, el piloto automático, y la alerta asociada “Risk: Low Presence”.

En el caso último, de detectar una presencia superior a 2, en este caso el máximo son 3 personas, pero se deja en un valor general superior a 2. La situación específica será valorada como un riesgo leve tras 3 lecturas de un valor por encima. Estableciendo el valor “1” o “Leve” a la variable “specRiskPresence” y activando la alerta de presencia como “High Presence”. Si tras esta activación, el riesgo persiste durante otras 3 lecturas, el nivel de riesgo específico será aumentado a un riesgo grave o el valor “2”, en este caso, no se cambia la alerta.

El código de esta función se puede ver en el siguiente anexo (A3.6.3), y cómo se puede apreciar, utiliza varias variables globales como son los contadores, las alertas y el riesgo específico, todos ellos, incluidos en el objeto protegido. Todas las situaciones, cambios y alertas son registradas en la tarjeta microSD.

### 7.3.1.6.1. Pruebas

En esta prueba, al igual que en las anteriores realizadas, la tarea mostrará el contenido del objeto protegido, este contenido también será mostrado en el registro microSD en el prototipo final. En las figuras siguientes (Figuras 35abcd) se puede ver la rutina de esta prueba, en ésta se va probando una presencia que aumenta desde 2 a 3, para después ser rectificada, y reducir la presencia por debajo de 2.

```

a) Specific Risk (Presence):
    Risk Level: 0 (No Risk)
    Alert: " "
    Actual Presence: 2
    Counters (Wrong Low, Correct, Wrong High): (0, 7, 0)

    Specific Risk (Presence):
    Risk Level: 0 (No Risk)
    Alert: " "
    Actual Presence: 3
    Counters (Wrong Low, Correct, Wrong High): (0, 0, 1)

    Specific Risk (Presence):
    Risk Level: 0 (No Risk)
    Alert: " "
    Actual Presence: 3
    Counters (Wrong Low, Correct, Wrong High): (0, 0, 2)

    Specific Risk (Presence):
    Risk Level: 1 (Light Risk)
    Alert: "High Presence"
    Actual Presence: 3
    Counters (Wrong Low, Correct, Wrong High): (0, 0, 3)

b) Specific Risk (Presence):
    Risk Level: 2 (Severe Risk)
    Alert: "High Presence"
    Actual Presence: 3
    Counters (Wrong Low, Correct, Wrong High): (0, 0, 25)

    Specific Risk (Presence):
    Risk Level: 2 (Severe Risk)
    Alert: "High Presence"
    Actual Presence: 2
    Counters (Wrong Low, Correct, Wrong High): (0, 1, 0)

    Specific Risk (Presence):
    Risk Level: 2 (Severe Risk)
    Alert: "High Presence"
    Actual Presence: 2
    Counters (Wrong Low, Correct, Wrong High): (0, 2, 0)

    Specific Risk (Presence):
    Risk Level: 1 (Light Risk)
    Alert: "Rectified Presence"
    Actual Presence: 2
    Counters (Wrong Low, Correct, Wrong High): (0, 0, 0)

c) Specific Risk (Presence):
    Risk Level: 2 (Severe Risk)
    Alert: "Low Presence"
    Actual Presence: 1
    Counters (Wrong Low, Correct, Wrong High): (3, 0, 0)

    Specific Risk (Presence):
    Risk Level: 2 (Severe Risk)
    Alert: "Low Presence"
    Actual Presence: 1
    Counters (Wrong Low, Correct, Wrong High): (4, 0, 0)

    Specific Risk (Presence):
    Risk Level: 2 (Severe Risk)
    Alert: "Low Presence"
    Actual Presence: 1
    Counters (Wrong Low, Correct, Wrong High): (5, 0, 0)

    Specific Risk (Presence):
    Risk Level: 3 (Critical Risk)
    Alert: "Risk: Low Presence"
    Actual Presence: 1
    Counters (Wrong Low, Correct, Wrong High): (6, 0, 0)

d) [21600] High Presence - Level: 2
[22200] High Presence - Level: 2
[22800] High Presence - Level: 2
[23400] High Presence - Level: 2
[24000] High Presence - Level: 2
[24600] High Presence - Level: 2
[25200] High Presence - Level: 2
[25800] High Presence - Level: 2
[26400] High Presence - Level: 2
[27000] High Presence - Level: 2
[27600] High Presence - Level: 2
[28200] High Presence - Level: 2
[28800] High Presence - Level: 2
[29400] High Presence - Level: 2
[33000] Rectified Presence - Level: 1
[34800] No Risk Presence
[37800] Low Presence - Level: 2
[38400] Low Presence - Level: 2
[39000] Low Presence - Level: 2
[39600] Risk: Low Presence - Level: 3

```

Figuras 35a, 35b, 35c, 35d: Prueba de “presenceTask”; a, b, c) Pruebas en display “TeraTerm”; d) Registro en microSD

Se prueban diferentes presencias posibles en la cabina con los dispositivos instalados, comenzando con una presencia de 2 que no tiene asociado un mensaje, que aumenta a 3 personas en cabina usando el sensor de movimiento, de esta manera se obtiene el nivel de riesgo específico 1 “Leve”, que al persistir aumenta al nivel de riesgo 2 “Grave”, junto con la alerta “High Presence”, esto se puede ver en la Figuras 35a. En la siguiente figura, Figuras 35b, el objetivo es rectificar un riesgo específico grave, realizando lecturas correctas consecutivas.

En la Figuras 35c, se quiere mostrar el aumento del nivel de riesgo a crítico por una baja presencia, este caso, es el más importante porque respetaría la normativa propuesta en 2015 por la EASA [27]. Finalmente, en la Figuras 35d, se muestra una parte del registro almacenado, en éste se puede ver una rectificación del nivel de riesgo y un aumento al nivel crítico 3, el cual bloquea cualquier cambio adicional.

### *7.3.1.7. Tarea Control de Riesgos (riskControlTask)*

Esta tarea será la encargada de analizar las diferentes situaciones o riesgos específicos del sistema para obtener una evaluación general y actuar respecto a ellos. Esta tarea tiene un periodo de 500 milisegundos y con una prioridad de valor 9, siendo la más prioritaria del sistema.

Esta tarea también tiene asociado un objeto protegido al cual se accede por medio de un semáforo de tipo “mutex”, “*xSemaphoreRisk*”. En él, la única variable es “*genRisk*” que es utilizada para indicar el nivel de riesgo general del sistema. También tiene varias funciones asociadas, las cuales hacen uso de los dispositivos actuadores como son los LEDs y el zumbador. Las declaraciones se pueden ver en el siguiente anexo (A3.7.1).

La tarea de control de riesgos, al igual que las anteriores, sólo será activada por el sistema si se ha introducido el código correcto, si es así, y el sistema se encuentra en funcionamiento, recogerá los diferentes niveles de riesgos específicos que han sido registrados en las tareas previas, asignándolas a un array. También se recoge el valor previo de la variable “*genRisk*” asignándola a “*oldRiskValue*”. Si este último valor es 3, es decir, es un riesgo crítico, se omitirá la contabilización de riesgos específicos. En esta fase, se recorrerá el array anterior, incrementando en 1 el valor del contador respectivo. Al finalizar, se comprobará si los contadores deben ser considerados de un riesgo superior, esto es, si hay más de 2 riesgos leves, se incrementa en 1 el contador de riesgo grave, y si hay 2 o más riesgos graves, se incrementa el riesgo crítico. Los casos donde es menor, o en el caso de existir un valor superior a 0 en el contador de riesgo crítico, se asocia a una variable interna el riesgo general.

Tras eso, se evalúa el riesgo general interno o local asociado, si éste es 0, indicará que el sistema se encuentra en una situación sin riesgos, y deberá comenzar un contador de 5 segundos para evaluar si reducir el riesgo actual. En este caso, el

contador se incrementa en tiempo cada vez que la tarea se activa debido a que debe ser reiniciado en caso de variar el valor, y con este método, se tiene un mayor control sobre el tiempo. La reducción es similar a la realizada en anteriores tareas, en este caso, si se reduce al riesgo general de nivel 0, en caso de algún imprevisto, se reinician las alertas asociadas, actuando con los actuadores del respectivo nivel, dado que se compara el nivel general del objeto protegido.

En caso contrario, el riesgo general local no es 0, se reinicia el contador de lecturas correctas y se evalúa el riesgo local de la tarea. Si éste es 1, se actualiza el riesgo general del sistema al valor 1, se activan los actuadores si el valor ha variado de la última ejecución, activándose únicamente la primera vez que cambia de nivel de riesgo, haciendo uso de “useActuator()” y pasando el riesgo y el contador asociado a éste. También se registra en la tarjeta microSD esta activación. En el resto de los niveles de riesgo general, el funcionamiento es análogo, cada uno con los actuadores necesarios, ya identificados en la función “useActuator()”. El código de esta tarea se puede ver en el siguiente anexo (A3.7.2).

#### [7.3.1.7.1. Función UseActuator\(\)](#)

Esta función es la encargada de activar los temporizadores del sistema y las luces LED del sistema, también activará el zumbador y en el caso del riesgo crítico, el sistema de autopiloto a través de un mensaje, dado que el prototipo no incluye la programación de este actuador. Es necesario indicar el contador para un correcto uso de las luces LED del módulo Pmod en el riesgo general de nivel 1 o “Leve”. Esta función se puede consultar en el siguiente anexo (A3.7.3).

Los actuadores asociados a cada nivel son los siguientes:

- Nivel 0 “Sin riesgo”: PmodLED a nivel 0 y LED RGB apagado.
- Nivel 1 “Leve”: PmodLED a nivel 1 o 2 (dependiendo del contador), LED RGB en color azul y zumbador a nivel 1 durante 3 segundos.
- Nivel 2 “Grave”: PmodLED a nivel 3, LED RGB en color amarillo y zumbador a nivel 2 durante 3 segundos.
- Nivel 3 “Crítico”: PmodLED a nivel 4, LED RGB en color rojo, zumbador a nivel 3 durante 3 segundos y mensaje de activación de autopiloto.

En el caso de los actuadores LED, tienen asociado una función con los colores configurados o la cantidad de LEDS que debe encender, en el caso del Pmod. Estas funciones no requieren de un análisis adicional, y se pueden consultar en el siguiente anexo (A3.7.4). En cambio, el zumbador, debido a su funcionamiento que requiere un pulso digital periódico para sonar, utiliza una tarea. Esta tarea, recoge un semáforo binario, no excluyente, que es cedido por UseActuator(), con este valor, y con un periodo de 15 milisegundos, ejecuta el sonido del zumbador. En el siguiente apartado se indica su funcionamiento con mayor detalle.

En el caso del zumbador y del “piloto automático” hacen uso de dos temporizadores, uno de 3 segundos y otro de 15, ambos son temporizadores de FreeRTOS que deben ser declarados previamente y creados en la función “*main*” del fichero. Un parámetro importante es indicar que no se repita el contador, dado que son “*timers*” asíncronos y se activan en determinados momentos. Los temporizadores requieren además de una función de retorno, en el caso del temporizador de 3 segundos, y debido al funcionamiento del zumbador, se recoge el semáforo de éste, para que deje de ejecutarse el dispositivo.

En cambio en la función de retorno del temporizador de 15 segundos, se anulan los riesgos del sistema, tanto los específicos como el general, y se apagan los diodos LED. También, recoge el semáforo binario respectivo al piloto automático.

#### 7.3.1.7.2. Tarea Zumbador(buzzerTask)

Debido a que el zumbador es un actuador dependiente del control de riesgos, utiliza una tarea periódica que es activada por “*useActuator*”. En esta tarea se espera para recoger el semáforo binario “*xSemaphoreBuzzer*”, debido al funcionamiento del zumbador, ya comentado previamente, requiere de pulsos digitales para sonar, aprovechando que admite varias frecuencias, se ha dividido en diferentes frecuencias dependiendo del nivel de riesgo. El nivel de riesgo se lee en cada ejecución accediendo al objeto protegido del riesgo general.

En esta tarea fue requerida una mayor investigación hasta llegar al funcionamiento y frecuencia adecuada, dado que FreeRTOS no permite establecer pausas menores de 1 milisegundo, en esta tarea se hace uso de los microsegundos que ofrece la placa de Xilinx. Los tiempos entre cada activación y desactivación del pin digital son los siguientes: 500  $\mu$ s para el nivel 1, 2000  $\mu$ s para el nivel 2 y 6000  $\mu$ s para el nivel 3. Para detectar el momento adecuado para parar, el semáforo se libera en cada ciclo del zumbador para permitir obtener el semáforo por la función de retorno del temporizador de 3 segundos. El código se puede consultar en el siguiente anexo (A3.7.5).

#### 7.3.1.7.3. Temporizadores

Se cree indicado añadir un apartado adicional con los temporizadores utilizados, estos son temporizadores software que permitirían una medición de hasta 1 milisegundo, como no se requiere esperar un tiempo menor, los ofrecidos por FreeRTOS son más que suficientes. Las funciones de “*callback*” o retorno, difieren en su respuesta, dado que en los riesgos leve y grave los diodos LED persisten hasta un

cambio “manual”, en el nivel crítico, será desactivado automáticamente al finalizar el temporizador.

Ambos temporizadores son asíncronos y no se repiten al finalizar, requiriendo un reinicio cada vez que se quiera encender, en ambos se obtiene el semáforo binario respectivo, pero en la función encargada del autopiloto, además requiere reiniciar los niveles específicos y generales del sistema.

Un problema surgido con esto es la imposibilidad de usar la opción del riesgo nulo de “useActuator”, la cual llega a bloquear al sistema y el stack de la función IDLE colapsa. Por tanto, todos los cambios se realizan en esta función de retorno. Estas funciones se pueden ver en el siguiente anexo (A3.7.6), junto con la declaración y creación de estos semáforos.

#### 7.3.1.7.4. Pruebas

Para probar este dispositivo, se requiere tener el resto de las tareas activadas, llegando a disponer de casi todo el sistema completo, también se hace uso del display, no comentado aún, para evaluar el valor de cada sensor.

La prueba en este caso es más compleja dado que no es posible controlar fácilmente la presencia en el prototipo, y el tiempo de respuesta desde que se activa el riesgo específico hasta que se enciende el riesgo general es muy corto. Aun así, se puede llegar a controlar de manera adecuada.

Por complicaciones al usar el acelerómetro y la presencia en cabina, se ha decidido no controlar el acelerómetro aunque se encuentre activado. Para ver el funcionamiento del sistema de manera adecuada, se reduce el techo de vuelo a 50 centímetros para que pueda ser evaluable. Para controlar la presencia, y por la disposición en el prototipo, se puede controlar con una mano adecuadamente.

Un cambio en la prueba importante ha sido reducir el segundo temporizador, de 15 segundos a 5, para evaluar el encendido y apagado de los diodos LED de manera rápida. También, en esta prueba, se omite el mensaje de activación del “piloto automático” al ser únicamente un mensaje adicional al encenderse el nivel de riesgo crítico.

```

[66030] Display Task
Values
Altitude Values: 9120 meters (Sim: 38 cm)
Tilt Values: X = 0.04 ||, Y = 1.96 ||
Presence Values: 2
Speed Values: 826.38 km/h
General Risk: 1 (Light Risk)

Alerts
Altitude: Rectified Altitude
[68020] Display Task
Values
Altitude Values: 9360 meters (Sim: 39 cm)
Tilt Values: X = 0.24 ||, Y = 2.05 ||
Presence Values: 1
Speed Values: 830.46 km/h
General Risk: 1 (Light Risk)

Alerts
Altitude: Low Altitude
a) [70020] Display Task
Values
Altitude Values: 9360 meters (Sim: 39 cm)
Tilt Values: X = 0.22 ||, Y = 1.95 ||
Presence Values: 1
Speed Values: 830.98 km/h
General Risk: 3 (Critical Risk)

Alerts
Altitude: Low Altitude
Presence: Low Presence
Altitude: Too High Speed
b) [80020] Display Task
Values
Altitude Values: 12000 meters (Sim: 50 cm)
Tilt Values: X = 0.31 ||, Y = 1.74 ||
Presence Values: 2
Speed Values: 822.30 km/h
General Risk: 3 (Critical Risk)

Alerts
Altitude: High Altitude
Presence: High Presence
[90030] Display Task
Values
Altitude Values: 12240 meters (Sim: 51 cm)
Tilt Values: X = -0.03 ||, Y = 1.69 ||
Presence Values: 2
Speed Values: 822.79 km/h
General Risk: 0 (No Risk)
c) [83500] Level 3 activated!
[83510] Activated PmodLED - LEDs ON: 4
[83510] Activated RGB LED - Critical Risk
[83710] High Altitude - Level: 2
[84000] Too High Speed - Level: 2
[84000] Level 3 activated!
[84020] High Altitude - Level: 3
[84030] Display Task Executed
[84500] Too High Speed - Level: 2
[84500] Level 3 activated!
[85000] Level 3 activated!
[85200] Low Presence - Level: 2
[85500] Rectified Speed - Level: 1
[85500] Level 3 activated!
[86000] Level 3 activated!
[86010] Display Task Executed
[86500] No Risk Speed
[86500] Level 3 activated!
[87000] Level 3 activated!
[87000] High Presence - Level: 1
[87500] Level 3 activated!
[88000] Level 3 activated!
[88010] Display Task Executed
[88500] Level 3 activated!
[88520] Activated PmodLED - LEDs ON: 0
[88520] Activated RGB LED - No Risk
[88530] No risk due to autopilot
[90020] Display Task Executed
[92010] Display Task Executed
d)

```

Figuras 36a, 36b, 36c y 36d: Prueba tarea "riskControlTask"; a, b, c) Display en "TeraTerm"; d) Registro en microSD

En las figuras anteriores (Figuras 36abcd) se han indicado fragmentos de la ejecución que se consideran de interés. En la Figuras 36a se indica que la evaluación es la adecuada con los datos que se muestran, en ese caso, una altura por debajo de los 41 cm (establecido como MSA), inclinación correcta, presencia correcta y velocidad adecuada, además de indicar la alerta asociada y el nivel general. En la siguiente figura, Figuras 36b, el objetivo es indicar el rápido cambio, aunque el display se ejecute cada 2 segundos el cambio podría llegar a ser de medio segundo, en él se cambia de un riesgo general leve a uno crítico, debido en este caso, a la baja altura prolongada, que activa el riesgo grave, una baja presencia (grave) y una alta velocidad (grave), el conjunto de estos riesgos específicos graves se convierte en un riesgo general crítico.

La última imagen de TeraTerm, Figuras 36c, indica una rectificación de la situación de crítica a nula, debido al autopiloto que no viene representado en el display, pero sí en el registro en la microSD, Figuras 36d. En esta última figura, se ha querido mostrar la ejecución que se realiza desde que se activa el riesgo grave hasta que se apaga por el autopiloto, en este caso 5 segundos. Se ve cuando se ejecuta el display e incluso cuando acaba la ejecución del "piloto automático".

### *7.3.1.8. Tarea Display/Monitor (displayTask)*

Tras finalizar las tareas de los sensores, acelerómetro, velocímetro, sensor de ultrasonidos y sensor de presencia, y la tarea de control de riesgo se obtienen todos los datos necesarios para mostrar el display del sistema. Esta tarea tiene un periodo de 2 segundos, considerado adecuado para visualizar los datos por display sin excesivos mensajes, la prioridad de esta tarea es de 3, siendo una de las tareas menos prioritarias del sistema. Debido a esto, se ha tomado la decisión de enviar cada línea individualmente, a diferencia de cómo se haría con el registro.

Esta tarea utiliza los semáforos de los objetos protegidos de los sensores para almacenar una copia de estos al activarse la tarea. Sin embargo, puede ser expulsada durante la escritura en el display al disponer de una baja prioridad.

Respecto al funcionamiento, en primer lugar, se registra en la microSD la ejecución de la tarea, éste es un paso necesario para mantener que la tarea se haya ejecutado en el plazo indicado. Posteriormente se obtienen los valores de cada objeto protegido, junto con la situación específica y alerta. En el caso del riesgo general, sólo se obtiene este valor. Dado que los riesgos específicos no son excesivamente relevantes si se tiene el riesgo general, esta tarea dispone de un método “*debug*” para ver qué nivel de riesgo específico tiene cada objeto protegido de los sensores (acelerómetro, velocímetro, altímetro, control de presencia).

Como se ha comentado previamente, esta tarea sólo utiliza los semáforos durante la lectura de los objetos protegidos, siendo posible una expulsión de la tarea mientras se escribe en el display. El primer mensaje contabiliza el momento en el cual se ha comenzado a escribir, poniendo un timespan al comienzo, esto se ve en el propio display. En su ejecución, se muestran los valores de cada sensor, si el modo “*debug*” se encuentra activado se muestran sus niveles de riesgo específico. Se finaliza mostrando por el display el nivel de riesgo general, que en caso de ser diferente a 0, mostrará las alertas de los sensores con un riesgo activado, siempre que estas no sean nulas. El código de esta tarea se puede consultar en el siguiente anexo (A3.8.1).

#### *7.3.1.8.1. Pruebas*

Al tratarse de una tarea sencilla, sin mucho trabajo de cómputo, se tomó la decisión de poner los mensajes individualmente para añadir una mayor carga de trabajo y posibilitar interrupciones, aunque estas no sean visibles. En las pruebas, se modificarán los valores de cada tarea, es necesario añadir, debido a la similitud con la prueba de la tarea de Control de Riesgos “*riskControlTask*”, la cual ya usaba el display para ver los datos.

```
[78030] Display Task
Values
Altitude Values: 10080 meters (Sim: 42 cm)
Tilt Values: X = -0.01 ||, Y = 1.93 ||
Presence Values: 2
Speed Values: 824.72 km/h
General Risk: 1 (Light Risk)

Alerts
Presence: Rectified Presence
[80010] Display Task
Values
Altitude Values: 10080 meters (Sim: 42 cm)
Tilt Values: X = -0.11 ||, Y = 1.90 ||
Presence Values: 2
Speed Values: 824.70 km/h
General Risk: 1 (Light Risk)

Alerts
Presence: Rectified Presence
a) [80010] Display Task
Values
Altitude Values: 11760 meters (Sim: 49 cm)
Tilt Values: X = -1.61 ||, Y = -30.84 ||
Presence Values: 2
Speed Values: 866.73 km/h
General Risk: 3 (Critical Risk)

Alerts
Tilt: High Pitch Angle
Presence: Rectified Presence
Speed: High Speed
[284030] Display Task
Values
Altitude Values: 11760 meters (Sim: 49 cm)
Tilt Values: X = -2.72 ||, Y = -32.66 ||
Presence Values: 2
Speed Values: 866.85 km/h
General Risk: 3 (Critical Risk)

Alerts
Tilt: High Pitch Angle
Speed: High Speed
b) [284030] Display Task
Values
Altitude Values: 11760 meters (Sim: 49 cm)
Tilt Values: X = -2.72 ||, Y = -32.66 ||
Presence Values: 2
Speed Values: 866.85 km/h
General Risk: 3 (Critical Risk)

Alerts
Tilt: High Pitch Angle
Speed: High Speed
c) [18050] Display Task
Values
Altitude Values: 8160 meters (Sim: 34 cm)
Tilt Values: X = 0.11 ||, Y = 1.75 ||
Presence Values: 2
Speed Values: 861.13 km/h

Specific Risk (Debug)
Altitude Values: 2 (Severe Risk)
Tilt Values: 0 (No Risk)
Presence Values: 0 (No Risk)
Speed Values: 1 (Light Risk)
General Risk: 2 (Severe Risk)

Alerts
Altitude: Low Altitude
Speed: High Speed
d) [66040] Display Task Executed
[66250] High Pitch Angle - Level: 2
[66500] Level 3 activated!
[66500] High Pitch Angle - Level: 2
[66750] High Pitch Angle - Level: 2
[67000] Level 3 activated!
[67000] High Pitch Angle - Level: 2
[67010] Activated PmodLED - LEDs ON: 0
[67010] Activated RGB LED - No Risk
[67020] No risk due to autopilot
[67210] High Altitude - Level: 3
[67250] High Pitch Angle - Level: 2
[67500] Level 3 activated!
[67500] Activated PmodLED - LEDs ON: 4
[67510] Activated RGB LED - Critical Risk
[67510] High Pitch Angle - Level: 2
[68000] Level 3 activated!
[68010] Rectified Angle - Level: 1
[68020] Display Task Executed
[68500] Level 3 activated!
[68510] No Risk Angle
[69000] Level 3 activated!
[69500] Level 3 activated!
[70000] Level 3 activated!
[70010] Display Task Executed
```

Figuras 37a, 37b, 37c, 37d: Prueba tarea "displayTask"; a, b, c) Muestra en display "TeraTerm"; d) Registro en microSD

En las figuras referentes a esta prueba de la tarea (Figuras 37abcd) se centra en el contenido mostrado, los datos y la periodicidad. En la Figuras 37a, se quiere mostrar el correcto periodo que sigue la tarea, aun disponiendo de tareas más prioritarias, en ese instante, se ve por la alerta asociada que se está corrigiendo de un riesgo por presencia desde un riesgo grave. En la siguiente figura, Figuras 37b, el riesgo general es de nivel 3, un riesgo crítico, se quiere indicar que aunque en este caso, el autopiloto se encuentre activado, el display sigue ejecutándose, dado que no hay una tarea que restrinja su ejecución.

La Figuras 37c muestra la ejecución del display con el modo "debug" activado, viendo los riesgos específicos asociados a cada tarea. Finalmente, en el registro de la Figuras 37d, se quiere indicar la periodicidad de la tarea encargada de mostrar el display, siendo el único valor registrado de esta tarea.

### *7.3.1.9. Tarea Tarjeta microSD (writeRecordTask)*

Ésta es la última tarea que compone el sistema, es el equivalente del modo “debug” del display que escribe los posibles valores en la tarjeta microSD introducida en la placa. Esta tarea tiene prioridad 2, siendo la tarea con la prioridad más baja, sólo por encima de la tarea lector del teclado numérico. Se ha decidido poner una prioridad muy baja debido a su largo periodo, de 5 segundos. Esta tarea se espera que sea la tarea que más tiempo ocupe en la CPU dado que tiene que abrir, escribir y cerrar el fichero en la tarjeta microSD. Esta tarea requiere algunos parámetros globales de la tarjeta microSD, pero cualquier escritura en la tarjeta requiere el uso de un semáforo exclusivo “mutex” para evitar pérdidas, las declaraciones y definiciones se pueden ver en el siguiente anexo (A3.9.1).

La implementación de esta tarea tiene similitud con el código de prueba del dispositivo de la tarjeta microSD de Xilinx Zybo. En este caso, una de sus características es la escritura por números enteros. Se considera una característica, porque la función encargada de escribir en la tarjeta microSD no admite caracteres e introducir otra secuencia de caracteres no-ASCII hace ilegible el texto, debido a esto, es más sencillo escribir en el fichero, porque cada carácter a introducir ocupa un byte, y el tamaño de escribir un fichero durante 5 minutos por parte del sistema, no ocupa un tamaño excesivo, permitiendo posiblemente, varias horas de escritura de un fichero.

En la tarea, cuyo código se puede ver en el siguiente anexo (A3.9.2), se recogen todos los valores de cada objeto protegido, incluyendo los contadores para conocer en cada instante el valor cada variable. Para evitar crear variables por cada una del objeto protegido, cada vez que se accede a un objeto protegido con su semáforo, se añade una línea al buffer del mensaje a enviar. Este mensaje tiene una longitud de 2048 bytes (2KB) por línea. Aunque nunca se llegue a superar los 1400 bytes, para evitar imprevistos se decide ampliar el tamaño del buffer a un valor más grande.

Una vez se registra cada valor en el buffer, incluyendo el timespan de inicio y final. Se llama a la función protegida “**SDFileWrite**”, esta función se encarga de montar la tarjeta microSD, abrir el registro correspondiente, convertir el array de caracteres a enteros, que deben ser registrados individualmente en un segundo array o buffer. Este último puede ser escrito en el fichero. Para evitar problemas de desconexión, tras escribir, el fichero se cierra para no perder datos. Este fichero se abre para escribir y adjuntar información, restringiendo la lectura del fichero y no permitiendo reiniciar la escritura de éste, incluso en varias ejecuciones se seguirá escribiendo en el mismo fichero.

Esta función, es la utilizada por todo el sistema para registrar en la tarjeta microSD todos los valores o riesgos activados, únicamente requiriendo un array de caracteres como parámetro. Se ha indicado previamente un buffer de 2048 bytes, o 2048 posiciones, este tamaño es la cantidad máxima que puede ser escrita cada vez, el fichero no tiene establecido un límite de escritura, pero el tamaño de cada fichero de

prueba es muy reducido y apenas afecta al tamaño general. El código de esta función se puede ver en el siguiente anexo (A3.9.3).

Otra tarea que tiene que ver con el mostrado de datos es la tarea “*getLevelRisk*” que devuelve a partir de un valor de riesgo, su equivalente en texto. Se puede consultar en el anexo (A3.9.4).

### 7.3.1.9.1. Pruebas

Aunque la prueba se pueda realizar sin datos, dado que registra el contenido de todas las variables, se decide por dejar el resto de las tareas, llegando a disponer de casi el sistema completo, a falta de algunos valores puntuales y de calibración.

En esta prueba se realizará una pequeña prueba, similar a la anterior, pero centrándose en el contenido registrado en la microSD. En esta prueba, se ha analizado una ejecución de la tarea, y los registros previos. En las figuras inferiores (Figuras 38abc) se muestra la prueba en cuestión. En la captura de “*TeraTerm*”, Figuras 38a, se quiere resalta el momento que se va a analizar, en este caso es la ejecución en los 110 segundos del sistema, en ésta se puede ver que llega a ejecutarse el display, con su periodo de 2 segundos, con un riesgo general Leve, donde la presencia se ha rectificado hasta quitar la alerta, y la altura persiste al estar a una altura ligeramente baja de la recomendada.

```

[105035] Ended SD Recording.

[105300] Low Altitude - Level: 2
[105500] Level 3 activated!
[105510] Activated PmodLED - LEDs ON: 0
[105510] Activated RGB LED - No Risk
[105520] No risk due to autopilot
[105600] High Presence - Level: 2
[106000] Activated PmodLED - LEDs ON: 3
[106000] Activated RGB LED - Severe Risk
[106010] Level 2 activated!
[106020] Display Task Executed
[106500] Level 2 activated!
[107000] Level 2 activated!
[107400] Rectified Presence - Level: 1
[107500] Activated PmodLED - LEDs ON: 1
[107500] Activated RGB LED - Light Risk
[107510] Level 1 activated!
[108000] Level 1 activated!
[108020] Display Task Executed
[108500] Level 1 activated!
[109000] Level 1 activated!
[109200] No Risk Presence
[109210] Low Altitude - Level: 1
[109500] Level 1 activated!
[110000] Level 1 activated!
[110010] Display Task Executed
[110038] Started SD Recording.

a) [108030] Display Task
Values
Altitude Values: 9600 meters (Sim: 40 cm)
Tilt Values: X = 0.51 ||, Y = 0.61 ||
Presence Values: 2
Speed Values: 829.03 km/h
General Risk: 1 (Light Risk)

Alerts
Altitude: Low Altitude
Presence: Rectified Presence
[110020] Display Task
Values
Altitude Values: 9840 meters (Sim: 41 cm)
Tilt Values: X = 0.46 ||, Y = 0.85 ||
Presence Values: 2
Speed Values: 828.99 km/h
General Risk: 1 (Light Risk)

Alerts
a) Altitude: Low Altitude
b)
```

## Implementación del Proyecto - Tareas Individuales del Sistema

```
[110038] Started SD Recording.  
Modes enabled:  
    Active/Started: True  
    Debug: False  
  
General Risk:  
    Level: 1 (Light Risk)  
  
Specific Risk (Accelerometer/Tilt):  
    Risk Level: 0 (No Risk)  
    Alert: ""  
    Actual Tilt Angle (X, Y): (0.46, 0.85)  
    Counters (Correct, Wrong AxisX, Wrong AxisY): (20, 0, 0)  
  
Specific Risk (Altimeter/Altitude):  
    Risk Level: 1 (Light Risk)  
    Alert: "Low Altitude"  
    Actual Altitude "Simulated" (Real): 9840 m (41 cm)  
    Set Altitude (Min - Max): (41 - 50)  
    Counters (Wrong Low, Correct, Wrong High): (0, 2, 0)  
  
Specific Risk (Velocimeter/Speed):  
    Risk Level: 0 (No Risk)  
    Alert: ""  
    Actual Speed: 828.99 km/h  
    Counters (Wrong Too Low, Wrong Low, Correct, Wrong High, Wrong Too High): (0, 0, 20, 0, 0)  
  
Specific Risk (Presence):  
    Risk Level: 0 (No Risk)  
    Alert: ""  
    Actual Presence: 2  
    Counters (Wrong Low, Correct, Wrong High): (0, 4, 0)  
  
c) [110038] Ended SD Recording.
```

Figuras 38a, 38b, 38c: Prueba de tarea "writeRecordTask"; a) Vista en "TeraTerm"; b) Registros de tareas; c) Registro de la tarea

En la Figuras 38b, se indica todos los mensajes que han ocurrido en este plazo, desde que se acaba de finalizar la última ejecución del registro hasta la próxima vez. En él, se ve que la última ejecución de "riskControlTask" ha sido indicar el nivel de riesgo 3, debido al "cooldown" de 5 segundos del sistema tras activarse. Con ello, unos milisegundos más tarde se apagan los diodos LED a la situación sin riesgo. Al existir una alta presencia en la cabina persistente, se activa el riesgo grave de Presencia, activando los diodos a este nivel. En el instante donde se cumplen 107,400 segundos, la presencia se rectifica, pero la baja altura, que viendo la ejecución, oscila entre lecturas correctas e incorrectas, no llega a reducirse en esa situación. Activándose el riesgo general de nivel 1, donde se mantiene sin reducirse hasta la ejecución del registro automático.

En la última figura, Figuras 38c, se muestra el registro automático del sistema, en él se ven todas las variables de cada tarea de los sensores (acelerómetro, velocímetro, ...) junto con los contadores de cada uno. Se ve que el velocímetro ha alcanzado el máximo valor del contador de lecturas correctas, del mismo modo el acelerómetro. También se puede ver que la única alerta activa es del altímetro pero su contador de lecturas correctas está aumentando, por tanto en la siguiente ejecución de su tarea, se anulará esa alerta. También se ve que el control de presencia lleva 4 lecturas consecutivas con una presencia de 2, lo que ha hecho reducir su nivel de riesgo específico. Al mantener el timespan del mensaje se puede conocer si en la ejecución de la tarea se ha expulsado la tarea del registro.

Este conjunto de tareas es el que más cerca se encuentra de disponer del sistema completo, viendo las pruebas realizadas y comentadas, se pueden valorar algunos cambios, la mayoría respecto a la forma de visualizar los datos en el display y en el registro.

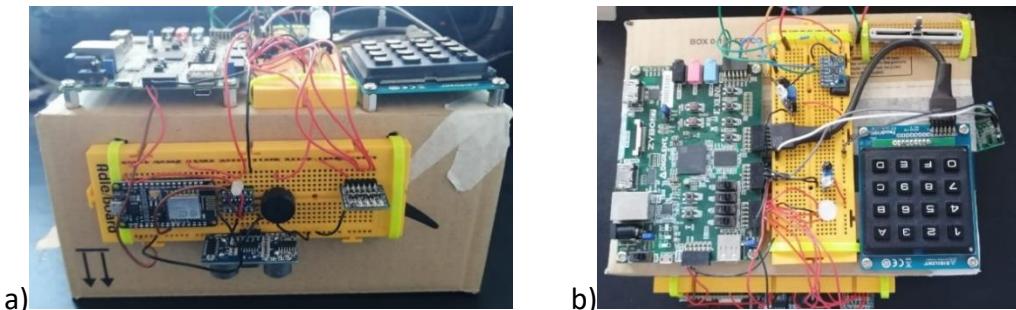
## 7.4. Diseño del conjunto de tareas

Tras las pruebas realizadas, y debido a las exigencias de la tarea de control de riesgo, display y la tarea encargada del registro. La planificación de un conjunto de tareas gradual se ha omitido, dado que es un proceso que se fue realizando a medida que se completaba el proyecto o se avanza en otras tareas. En las pruebas se puede ver que el conjunto de tareas se complementa y no requiere de un conjunto diferente.

### 7.4.1. Diseño del prototipo

En esta fase del proyecto se tratará el diseño físico del prototipo y la resolución de algunos inconvenientes surgidos. El prototipo hace uso de los dispositivos ya comentados en el apartado “

“Dispositivos Utilizados” anterior. Debido a la realización de un primer prototipo no definitivo, se han usado placas “breadboard” o “Protoboard”, que ha derivado en dividir los dispositivos en dos grupos. Los dispositivos que pueden funcionar o deben funcionar colocados de manera vertical y los que funcionan de manera horizontal, esto es visible en la siguiente imagen (Figuras 39ab).



*Figuras 39a y 39b: Imágenes del prototipo; a)Vista lateral; b)Vista superior*

En el primer conjunto, mostrado en la Figuras 39a, los dispositivos que deben o pueden funcionar en posición vertical, destaca el sensor de ultrasonidos, este dispositivo requiere apuntar hacia el suelo o hacia abajo para obtener la altitud del sistema. Otros de los dispositivos conectados en la placa en posición vertical son el módulo PmodLED, que ofrece una mejor visibilidad de los diodos y el zumbador, que a pesar de no requerirlo, no causa problemas en posición vertical. También el aspecto para decidir dónde colocar cada dispositivo viene seguido de la posición de los puertos Pmod, este tema vendrá tratado en el siguiente apartado.

En el segundo conjunto, mostrado en la Figuras 39b, los dispositivos en posición horizontal, viene destacado por el acelerómetro. Este dispositivo, para medir correctamente los ángulos requiere estar situado en posición horizontal. También se han incluido el potenciómetro, el teclado matricial, la placa FPGA, el diodo LED RGB y los detectores de presencia y movimiento. De manera análoga, vienen restringidos varios por los puertos Pmod.

#### 7.4.2. Puertos Pmod

El prototipo hace uso de todos los puertos Pmod, excepto el exclusivo de la placa, PmodJF o PmodMIO. Este último es utilizado únicamente para proveer de potencia la placa en posición vertical.

Uno de los inconvenientes que surgieron es la problemática ya comentada del puerto PmodJD, que al ser difícil de solucionar, se prefiere adaptar los dispositivos a este último. En este puerto, el orden de los pines viene alterado en pares, debido a esto, no es posible conectar el teclado, el cual utiliza todos los pines para su conexión. También, el prototipo limita la posición de la entrada analógica donde se conecta el potenciómetro junto con un circuito divisor de 3.3V a 1V.

El puerto PmodJA, el puerto analógico, es usado como tal sólo por el potenciómetro, para incluir un uso adicional a este puerto, uno de los pares de éste se utiliza para el acelerómetro por I2C, dado que el puerto sólo limita las entradas analógicas a 1.1V, la comunicación I2C no se ve afectada.

En el siguiente puerto, PmodJC, se ha conectado el teclado matricial, al requerir de una posición más cercana de otros puertos a los dispositivos vertical, evitando un cruce excesivo de cables. Al usar todos los pines del puerto, no es posible conectar ningún dispositivo adicional.

En el puerto PmodJD, es donde se han conectado los dispositivos de entrada a la placa y algún dispositivo de salida. De los dispositivos de entrada, se tienen los sensores encargados de la presencia, los dos detectores de presencia y el sensor de movimiento, además de incluir los dos pines necesarios para utilizar el sensor de ultrasonidos, uno de entrada y otro de salida. El uso de estos dos últimos pines viene limitado por la ocupación completa del siguiente puerto, PmodJE. Y siendo los 2 únicos cables que cruzan otros cables en la primera versión del prototipo. Los pines 1, 2 y 4 (debido a la distribución de los pines) son asociados a los dos sensores de presencia y el sensor de movimiento respectivamente. Los pines 3 y 7 son los pines de “Echo” y “Trigger”, respectivamente. Los tres pines restantes sin conectar se han configurado como pines de salida.

Finalmente, dado que el puerto PmodJF o PmodMIO es limitado a sólo el procesador y no se requiere de su uso, en el puerto PmodJE se han conectado sólo dispositivos de salida, los pines 1, 2 y 3, son usados por el diodo LED RGB para obtener los colores respectivos. El pin 4, se le ha asignado al zumbador. Y la secuencia de pines 5-8 se han conectado al módulo PmodLED con su pin para cada diodo LED. Este puerto, por proximidad y para evitar cruces innecesarios de cables, tienen situados los dispositivos cerca del puerto PmodJE.

#### 7.4.3. Función “main()”

Para mantener y disponer de todas las tareas en funcionamiento, se debe disponer de la función “*main*” que inicie todas las tareas y semáforos. También falta la declaración de las tareas como tal, que por colocación en el código se ha preferido agrupar las tareas en un único sitio del código.

Comenzando por las librerías utilizadas, que ya han sido comentadas a lo largo de la investigación, se dispone de dos grupos de librerías, las librerías de “FreeRTOS” y las librerías de Xilinx. Las librerías que ofrece “FreeRTOS” son las que ofrecen el temporizador, las funciones para crear tareas, los semáforos o las funciones matemáticas utilizadas. Esta última requiere de añadir el parámetro “-m”, en las propiedades del proyecto, a las librerías de “ARM gcc”, tal y como se puede ver en la siguiente figura (Figura 40).

## Implementación del Proyecto - Diseño del conjunto de tareas

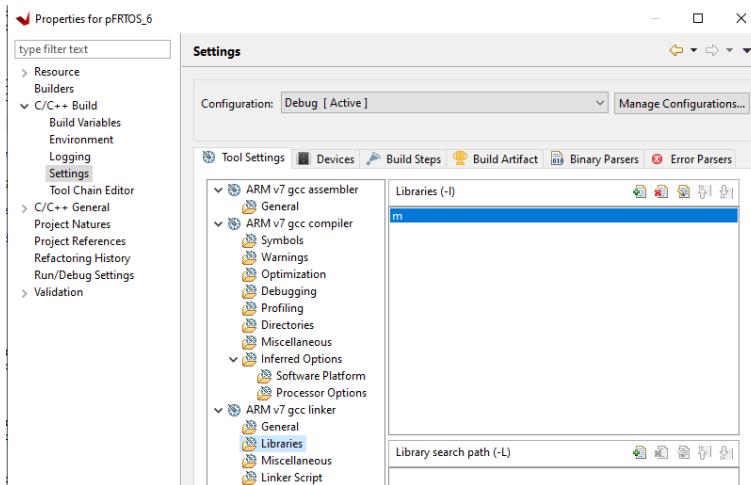


Figura 40: Propiedad "m" en librerías del proyecto

Entre las librerías de Xilinx, son muchas de ellas, librerías para hacer funcionar el código en la placa como lo es “xplatform\_info.h” o “xparameters.h”, aunque la gran mayoría de ellas son librerías para los dispositivos utilizados. En el siguiente anexo (A3.10.1), se pueden ver las librerías que utiliza el sistema completo.

El código necesita declarar varios componentes como son las tareas, sus controladores y los semáforos, en una primera fase, estos son únicamente declarados. En el siguiente anexo (A3.10.2) se puede ver esta declaración, para identificar cada uno, las tareas y controladores tienen el sufijo “Task”, y los controladores incluyen además un prefijo “x” delante del nombre. En el caso de los semáforos, tienen cada uno el prefijo “xSemaphore” seguido del nombre de la tarea o el dato que controlan.

La función “main” se encarga de inicializar las tareas y semáforos, además de algunos dispositivos del sistema. En primer lugar, se inicializa con el uso de las funciones “setupBoard()” y “setupGPIO()”, que inician algunos dispositivos que no se pueden generar y crear en una tarea. Se inicializan también la especificación de los temporizadores, que ya se ha comentado previamente.

Tras esa inicialización, se declaran todas las tareas con la función “xTaskCreate()” donde se declara qué función utiliza la tarea, un nombre descriptivo, el tamaño de pila, que por defecto se ajusta a 512 y en algunas tareas más exigentes ha sido necesario aumentar este valor. También se declara si requiere de parámetros, que se ha decidido dejarlo en blanco, la prioridad siendo un mínimo de prioridad 0 por defecto, donde se ha omitido asignar la prioridad 0. Finalmente se indica el controlador de la tarea, aunque tenga un uso irrelevante en el proyecto, se puede usar para eliminar tareas cuando no sean necesarias.

Tras la declaración de las tareas, sólo es necesario inicializar los semáforos, como ya se ha comentado, “xSemaphoreBuzzer” y “xSemaphoreAutopilot” son semáforos binarios, el resto de los semáforos utilizados, son semáforos “mutex”, con exclusión mutua, donde no puede ejecutarse otra tarea al mismo tiempo que se esté usando un semáforo, son útiles para objetos protegidos. En cambio, los semáforos

binarios, no tienen esa característica y se utilizan para activar o desactivar algunos dispositivos. Para verificar que se hayan creado se comprueba si no tienen el valor “*null*”. En el caso de los semáforos binarios, no es necesario recoger el semáforo, porque la primera función que admiten tras crearse es “*xSemaphoreGive()*”, siendo la función que se utiliza para activar los dispositivos de piloto automático y zumbador. El código de la tarea se puede consultar en el siguiente anexo (A3.10.3).

Finalmente, se ejecuta la función “*vTaskStartScheduler()*” para crear una secuencia de ejecución de las tareas e iniciar todas las tareas. Para no parar la ejecución al acabar esta función, se inicia un bucle *for* infinito.

Con esta última función, se ha recorrido el código entero y tras probar anteriormente cada tarea individualmente o en conjunto, y con el prototipo funcional, es posible avanzar a una verificación de las tareas en base a la especificación y casos de uso indicados anteriormente.

## 8. Pruebas del sistema

---

En esta fase de desarrollo, indicada como **D4** en la planificación, se probará la funcionalidad del sistema y las tareas respecto a los casos de uso extendidos y los requisitos y especificaciones ya indicados. Para valorar todas las alternativas, se dividirá por los casos de uso que se han indicado en el apartado (5.2) y el conjunto del sistema en un entorno final.

En primer lugar, se revisarán los Requisitos que el sistema cumple en general, para pasar posteriormente a los requisitos cumplidos por cada caso de uso. Los requisitos de negocio (RN-1...RN6) se cumplen al ser la idea del sistema de la cual se ha partido.

Otros de los requisitos que el sistema cumple aún sin probar el funcionamiento de cada dispositivo, son los requisitos: **RNFint2**, **RNFmod1**, **RNFmod2**, **RNFmod3**, **RNFport1**, **RNFport2**, **RIEhw4(.1/.2)**, **RD1**, **RD2**, **RD3**. Todos estos requisitos se cumplen tanto en portabilidad, desarrollo y modificabilidad dado que el sistema está desarrollado en una placa Xilinx con arquitectura Zynq-7000, bajo un sistema operativo FreeRTOS, en lenguaje C y admite sin ningún problema los dispositivos de Digilent además de otros, dado que son dispositivos no propietarios de ninguna plataforma específica. Se ha decidido agruparlos al tratarse de requisitos que afectan al sistema en general y no a los casos de uso específicamente.

Respecto a la propiedad de plataforma, los dispositivos de Digilent, pueden ser utilizados bajo cualquier sistema, pero Xilinx ofrece librerías por defecto para ellos, al ser uno de los fabricantes de las placas de Xilinx.

### 8.1. Pruebas Casos de Uso

Se realizarán pruebas donde se relacionarán los casos de uso con las tareas y los requisitos que cumplan. También, se comprobarán que los casos de uso extendidos coincidan con la ejecución del sistema.

### 8.1.1. CU-1

Este caso de uso es el asociado al velocímetro, éste cumple el requisito **RF1.3** al ser un dispositivo incluido en el sistema. Viendo la tarea del velocímetro (disponible su código en el siguiente anexo, (A3.5.3) y comentado en el apartado, “7.3.1.5. Tarea Velocímetro” se ve que los requisitos bajo **RF6** son respetados al basarse en ellos durante la creación del sistema, de manera similar los requisitos acogidos bajo **RNFpre3**, siendo resuelto por las especificaciones del dispositivo, respecto a rendimiento, se cumplen los 500 ms establecidos por el requisito **RNFrend3**.

Para verificar que la tarea cumple su flujo indicado y alternativas, se seguirá el especificado en los casos de uso. El flujo normal, al no indicar por pantalla ningún dato ni registro, no crea líneas de texto, sí que se ha incidido en el cambio de nivel de riesgo al mantener 2 lecturas consecutivas correctas. En las lecturas incorrectas, al no tener un segundo nivel de criticidad, al llegar el contador a 3, se puede ver que ha sido activado el nivel correspondiente. También se ha probado la “inactividad” en caso de un riesgo crítico, dejando funcionar al piloto automático.

### 8.1.2. CU-2

Este caso de uso se encarga de mostrar el flujo de la tarea de detección de presencia. Según los requisitos de usuario **RU7** y **RU8**, se requiere la presencia de dos personas en cabina y detectando movimiento cerca de la puerta, para mayor seguridad, estos requisitos son cumplidos por esta tarea. El uso de dos detectores de proximidad y un detector de movimiento, hacen que los requisitos **RF1.4** y **RF1.5** sean efectuados.

Los sets de requisitos bajo **RF7** y **RF8** se han utilizado como indicaciones para realizar el funcionamiento de la tarea, indicada en el anterior apartado (Tarea Presencia). De esta manera, tanto la activación como desactivación del riesgo específico, viene medido a través de 3 (tres) lecturas consecutivas. Posteriormente los requisitos **RNFpre4**, **RNFpre5** y **RNFrend4** son cumplidos, al ser el funcionamiento de los dispositivos y el periodo de la tarea, respectivamente.

En la prueba para seguir su flujo, se han probado todos los flujos posibles, tanto normales como alternativos, corresponden al funcionamiento del sistema en el prototipo, se ha seguido cada flujo con el correspondiente resultado. También con ayuda del registro se puede conocer el periodo de lecturas erróneas de esta tarea. Mantiene un posible error en el prototipo con el uso del sensor de movimiento, a excepción de eso, cumple la ruta indicada.

### 8.1.3. CU-3

El caso de CU-3 es “Detectar Altitud”, este caso de uso utiliza un sensor de ultrasonidos, debido a la potencia necesaria de 5V para su funcionamiento, se requiere usar dispositivos adicionales indicados en un apartado anterior (Altímetro (Sensor ultrasonidos)). El requisito principal es respetar la altitud mínima recomendable de seguridad (“MSA”, por sus siglas en inglés) establecida a 10000 metros, e indicada en el requisito **RU11**, llevando a cumplir **RF1.2** al usar este sensor.

Al igual que cómo ocurrió en anteriores casos de uso, tiene un requisito funcional asociado que ha servido para formalizar el código, en este caso, **RF5**. Se indica que se respetará la “MSA” y se mantendrá por debajo del techo de vuelo. Se ha implementado además una manera de modificar el techo de vuelo, establecido inicialmente a 170, con su equivalente MSA a 141. Por medio del teclado, el techo de vuelo del sistema puede ser modificado, manteniendo también las 3 lecturas consecutivas que utilizaran otras tareas también para modificar el nivel de riesgo. Los requisitos bajo **RNFpre2**, **RNFrend2** y **RIEhw2** también ha sido de ayuda para precisar los niveles del dispositivo.

En el seguimiento de su flujo de actividad o funcionamiento, se ha vuelto a seguir el ya indicado en su caso de uso extendido CU-3. El seguimiento de estos ha obtenido un resultado correcto, sin crear problemas con otras tareas. Excluyendo un posible efecto retardado debido al “*cooldown*” de lecturas correctas de “*RiskControlTask*” de 5 segundos de lecturas correctas. El flujo de la tarea se cumple con lo esperado en el prototipo.

### 8.1.4. CU-4

Este caso de uso es el asociado a “Detectar Inclinación”, para ello utiliza un sensor de aceleración GY-521, que transmite la información por medio de I2C, cumpliendo los requisitos **RF1.1** y **RNFint3**. Cumpliendo el primero de ellos, se completarán todos los subrequisitos de **RF1**. El código del acelerómetro cumple con lo indicado bajo los requisitos **RF4**, con la precisión indicada en los requisitos **RNFpre1**, cumpliendo con el tiempo indicado en **RNFrend1**. Tomando los ejes indicados en el requisito **RIEhw1**.

Respecto al flujo, resulta la tarea más complicada de conseguir el flujo seguido, dado que al inclinar el prototipo se activa automáticamente el sensor de movimiento o el sensor de ultrasonidos, no permitiendo valorar únicamente esta tarea. A pesar de ello se ha probado la gran mayoría de los flujos alternativos, salvo alguno de rectificación de datos que al seguir el mismo esquema que en tareas anteriores, se podría dar una gran probabilidad de su correcto funcionamiento.

### 8.1.5. CU-5

Este caso de uso “Detectar Riesgos”, no tiene ningún fragmento de código asociado salvo ser el encargado de activar las tareas y preparar el funcionamiento de todos los dispositivos. De este modo, al ser un caso de uso para agrupar los anteriores casos de uso y crear un enlace con el resto del sistema, no dispone de requisitos ni un modo para probar su funcionamiento.

Ante la suposición anterior de ser el equivalente para inicializar las variables y las tareas, actuando de enlace, su funcionamiento sería un fragmento de la función “main” del código. Al funcionar el sistema correctamente, lleva a entender que cumple con el flujo indicado.

### 8.1.6. CU-6

El siguiente caso de uso a analizar y probar es “Analizar Riesgos”, en el código viene indicado como “*RiskControlTask*” y es la tarea encargada de crear las respuestas del sistema. También es la que más requisitos abarcará. La realización de este caso de uso supone completar los siguientes requisitos en relación al sistema, **RU1.1**, **RU1.2**, **RU1.4**, **RU2**, **RU10**, **RF2** y los requisitos bajo **RF3**, siendo éste el funcionamiento de la tarea y las funciones asociadas, junto con los semáforos del zumbador y del piloto automático. Se cumplen todos estos requisitos debido al uso de los actuadores por parte de funciones asociadas, y la división de las tareas que ofrece el requisito **RF3**, con su desactivación y respuestas a cada nivel. En el caso de los requisitos bajo **RF3.2.4** se ha modificado finalmente en el código para que se desactive tras 15 segundos, independientemente de la situación actual, en lugar de “en condiciones ideales”, suponiendo en el prototipo que el piloto automático situará el avión en las condiciones ideales.

La tarea asociada, cumple con el rendimiento indicado en el requisito **RNFrend8** de 500 milisegundos, también se cumplen los plazos indicados en los requisitos **RNFrend9** y **RNFrend10** que hacen referencia a plazos de tiempo de los actuadores de la tarea, también cumple el requisito **RNFseg2** en relación con el piloto automático estricto que no ha sido implementado en este proyecto.

Al igual que en los anteriores casos de uso, se comprobará el flujo deseado e indicado en su caso de uso extendido. Durante el seguimiento del flujo del caso de uso, no se ha percibido ningún error o fallo, respecto a lo esperado e indicado, por tanto, se califica esta prueba de adecuada. Incluyendo las respuestas asociadas, que son las esperadas en cada momento y con los plazos indicados en los requisitos y el caso de uso extendido.

### 8.1.7. CU-7

El siguiente caso de uso es CU-7, es el asignado a “Visualizar Datos” o a la tarea de “*DisplayTask*”. En este caso, tiene el objetivo de cumplir los requisitos asignados al display como **RU1.3**, obteniendo los datos de los sensores o **RU5**, donde se ofrece esta información a los pilotos por medio de un display instalado. Dado que muestra el contenido de los sensores, se ha aplicado al código los requisitos indicados como **RNFrend6** y **RIEusu1**. Como no dispone de más requisitos asociados para su desarrollo, no se cumplen otros que no tengan en cuenta sólo el display.

En la comprobación del flujo seguido, del mismo modo, se sigue el indicado en su caso de uso extendido. Como no existen muchos flujos alternativos, únicamente el asociado al modo “*Debug*” que incluye más información al display, concluyendo que cumple con los establecido en su caso de uso, y las excepciones asociadas.

### 8.1.8. CU-8

Este caso de uso es el asociado a la tarea “*UseSystemTask*” o “Utilizar Sistema”, en él se permite interactuar con el sistema a través de códigos de 4 dígitos hexadecimales para varias funciones del sistema, como entrar en el modo debug, encender, apagar, cambiar el techo de vuelo o cambiar el nombre del fichero en la microSD. Se espera que cumpla los requisitos **RU6**, **RNFseg1** y **RNFseg1.1**. Todos ellos sólo indican el uso de un código para el uso del sistema especificando su longitud. En relación a rendimiento, se cumple el indicado bajo **RNFrend7** pudiendo comprobar que hay un diodo LED en la placa que parpadea cada 2 segundos indicando que el sistema está activado.

También se comprobará el flujo indicado en su caso de uso extendido, donde no se especifican todas las funcionalidades del sistema, pero bajo las tareas a seguir, no deberían crear ningún inconveniente. En este caso, hace uso también de la tarea “*KeyPadTask*” para recoger el código indicado y almacenarlo, cumpliendo el requisito **RIEusu2**. Tras su comprobación realiza la acción indicada con el código introducido, al igual, cumple el resto de los flujos alternativos, llegando a concluir que también realiza su flujo correctamente.

### 8.1.9. CU-9

La tarea asociada a este caso de uso es “*WriteRecordTask*” y es el encargado de escribir en una tarjeta microSD un fichero con el registro acumulado desde el encendido de la placa, para registrar los cambios de fichero y techo de vuelo. En este caso, se cumple el requisito de usuario indicado como **RU9**, almacenando un registro de la actividad.

Con la funcionalidad de escribir el código, se ha basado en lo especificado bajo el requisito **RF9**, teniendo un gran parecido con el anterior caso de uso, “Visualizar Datos”. Respecto a este requisito funcional, incluye tanto los mensajes periódicos de las tareas del sistema como los esporádicos que surgen activan los dispositivos actuadores o los temporizadores. Junto con el siguiente caso de uso, cumplen el requisito de manejabilidad **RNFmane1**, siendo disponible su lectura tras la desactivación del sistema por parte de un técnico. Debido a que la escritura en los ficheros abre y cierra el fichero en cada intento de escritura, se podría asegurar que la información se registrará y mantiene correctamente. Cumpliendo posiblemente los requisitos **RNFrec2** y **RNFrec3** para recuperar los datos en caso de alguna catástrofe. No se puede asegurar del cumplimiento del requisito **RNFrec1** debido al reducido tamaño de los registros almacenados, donde a pesar de disponer de varios registros, no se ha conseguido que ocupen un gran tamaño, y no se ha llegado a valorar tal situación con el código. También por la especificación de la tarjeta microSD, se cumplen los requisitos indicados como **RIEhw3** y **RIEcom1**, donde se especifica el tamaño y formato de la tarjeta microSD.

Respecto al código, se ha comentado que es muy similar al funcionamiento del display, salvo que en este caso, registra el estado del sistema y las variables necesarias y estructuradas por funcionalidad. Dado que no dispone de un flujo alternativo, el flujo normal incorpora el añadido de los contadores de cada tarea para controlar si la activación ha sido realizada correctamente en función de estos. Obteniendo una prueba satisfactoria del flujo de esta tarea.

### 8.1.10. CU-10

Este caso de uso, asociado a “Diagnosticar” no dispone de una tarea en el sistema, ni tampoco es un concepto que agrupe otras tareas, como lo era “Detectar Riesgos”. La funcionalidad de éste es indicar unas pautas a realizar por parte del técnico que revise el registro almacenado en la tarjeta microSD. Dando la suposición que el técnico está cualificado para comprender el registro y detectar cualquier anomalía en él, en caso de existir alguna. También si tras varias lecturas del registro de varios vuelos, éste no causa ningún error, se puede entender que el sistema es completamente funcional y este caso de uso será requerido sólo ante emergencias.

Debido a que es un caso de uso externo del sistema, aunque se incorpora su realización, no depende de ningún requisito, únicamente es una operación rutinaria para verificar la integridad del sistema y comprobar cualquier defecto que pueda tener el sistema en caso de ocurrir. El flujo seguido es una verificación del orden de los valores mostrado y en caso de detectar algún periodo anómalo, trabajar en su resolución en las fases tempranas de su desarrollo e implementación en el entorno real.

Se han dejado varios requisitos sin indicar, debido a que tratan al sistema como un total, sin ningún caso de uso de por medio. En este caso, tras la prueba del prototipo creado, se llegan a cumplir los requisitos **RU3** y **RU4**, dado que no supone ninguna distracción adicional de la requerida. El tiempo de desactivación del zumbador ha sido modificado de 1 segundo a 3 segundos, para mantener el aviso un tiempo adicional. En este aspecto, también se ha cumplido el requisito **RNFdis1** donde se especifica el entorno donde el sistema debería de funcionar, sin poder llegar a probar los requisitos del entorno físico (**REF1-4**) dado que el prototipo no cumple estos, pero al ser instalado en una aeronave, deberían cumplirse.

Señalando las posibles interferencias que pueden surgir con otros sistemas similares, tal y como se indica en los requisitos **RU12** y **RNFint1**, se ha intentado evitar las respuestas o posibles interferencias con otros sistemas similares, como puede ser el EGPWS (GPWS/EGPWS) o AMSADD (AMSADD).

Con esto, se han repasado los requisitos indicados, algunos que se ha modificado al sistema final y los casos de uso propuestos, con un resultado favorable en relación al cumplimiento de estos, el cual no ha supuesto un gran desafío, al tener que realizar algunas de estas fases en coordinación con el desarrollo del código.

## 8.2. Pruebas en el entorno final

Esta fase de pruebas es la planificada como fase de desarrollo **D5: Verificación y valoración final del prototipo**, en esta fase, se explicará la instalación del sistema programa en la placa FPGA, además de realizar una prueba completa con el prototipo. Al tratarse del prototipo, su entorno final se reduce, a un espacio sin obstáculos, a una altura máxima de 80 centímetros de techo de vuelo. Para garantizar el funcionamiento, se probará con 50 centímetros de techo de vuelo, para evitar la detección de algún obstáculo como una estantería u otros muebles. En las anteriores figuras (Figuras 39ab) se puede ver el diseño que tiene el prototipo, tal y como se muestra, se ha buscado una solución sencilla para disponer de todos los dispositivos.

En primera instancia, es necesario indicar, que el prototipo a utilizar en esta fase es un sistema compacto montado sobre una caja de paquetería. Tal y como se

explicó en un apartado previo, hay componentes que requieren o pueden estar instalados en posición vertical, se cree que este prototipo es funcional y compacto, aunque no ofrezca una presentación “profesional”. Cabe destacar que en este prototipo, todas las conexiones están identificadas y cómo se ha comentado anteriormente, se ha intentado no tener conexiones que atraviesen otras, siendo este prototipo la solución adecuada. Esta presentación podría ser mejorada en un futuro con un prototipo en metacrilato o incluso realizado en impresora 3D.

Para la prueba, se realizará una ejecución de aproximadamente 10 minutos donde se intentará simular el vuelo del avión a escala reducida, con algunas distracciones y problemas que pueden ocurrir, para verificar que el sistema es plenamente funcional en su totalidad. Para realizar la acción por el piloto automático, se volverá a la posición inicial de la prueba. Una segunda ejecución de ésta se realizará para medir los tiempos de cómputo de cada tarea y al utilizar el recurso de la tarjeta microSD, comprobar si no existe un cuello de botella para este dispositivo. Esto se realizará por medio de mediciones en microsegundos tal y como se hizo para medir el tiempo del sensor de ultrasonidos.

El resultado de la primera ejecución, a vista de cómo lo podría ver un piloto o el propio usuario ha sido satisfactorio, el único dispositivo que ha resultado molesto para la prueba realizada ha sido el sensor de movimiento que ya se encuentra en su configuración menos sensible y retardada, pero sigue detectando movimientos leves que se realizan con el prototipo. En el entorno final, esto se traduciría como un miembro de la tripulación que comprueba el estado de la cabina cada cierto tiempo, activando esta señal.

Uno de los posibles cambios que se podría hacer en el código, es dudar de la fiabilidad de los sensores de presencia, para el cual se podrían tomar varias mediciones antes de cambiar el nivel de riesgo. Otro cambio sería aplicar este mismo retardo a “Control de Riesgos” o “Análisis de Riesgos”, para poder mantener este nivel durante un plazo más prolongado antes de la respuesta. El primero de estos cambios se consideraría un añadido al sistema, dado que no se detecta una extrema necesidad de indicar la existencia de más personas en cabina. Pero debido a la posibilidad de la presencia de sólo un piloto, donde esta situación es grave de por sí, la tarea de ofrecer respuestas no supone un problema a una rápida reacción. Se entiende que las medidas tomadas y las mediciones que se realizan en estas tareas como en todo el sistema son las más adecuadas, y no requieren el uso de contadores adicionales.

Sin embargo, aunque sea de una manera muy leve, y apenas apreciable, es posible cambiar el periodo de la tarea que comprueba la presencia a 1 segundo, así, se reduce un porcentaje de las activaciones por esta tarea, pero llega a ser apenas perceptible al realizar el movimiento de alabeo y cabeceo del prototipo.

Finalmente, antes de pasar a una comprobación de cómo se registran las tareas y sus tiempos de cómputo, se quiere destacar un posible efecto que puede parecer que no se activa el teclado. Esto se debe a que no aparece en el display al pulsarlo en

algunas ocasiones, esto es provocado a la baja prioridad que tiene esta tarea, donde cualquier otra es capaz de expulsarla de su ejecución. También, debido al periodo de ésta, se ha estimado que una velocidad de 200 milisegundos es suficiente para detectar la pulsación de una tecla, pero en los códigos de prueba, este periodo es prácticamente 1 milisegundo, donde éste no podría coexistir con ninguna otra tarea. De esta manera, se quiere indicar la solución tomada para saber si se pulsa una tecla que es la aparición del código registrado hasta el momento al pulsar una nueva tecla, hasta un máximo de 6 caracteres.

Se ha realizado una segunda prueba, más corta, que ha acabado sobrepasando la pila de memoria de algunas tareas, siendo suficiente la ejecución tomada, además de ser únicamente una medida para conocer los tiempos de cómputo del sistema.

El primer aspecto a tener en cuenta es que el tiempo que tarda cada tarea en leer y realizar los cálculos necesarios, se tienen que medir en microsegundos para conocer el valor de estos. También, las acciones que realizada cada tarea, en muchos casos es apenas perceptible, de esta manera se quiere demostrar que no es necesario establecer en este sistema plazos de “*deadline*” dado que su tiempo de cómputo es muy bajo. Durante la breve ejecución, abortada por el sistema y la tarea del ultrasonidos, ha sido suficiente para obtener la gran mayoría de los valores y tiempos relevantes. En el código creado, se ha añadido funciones de medición de tiempo que no estarán disponibles en la versión final del código.

Con los resultados obtenidos visibles en las figuras inferiores (Figuras 41abcd), se puede ver una secuencia de tiempos que aparecen o se toman entre dos ejecuciones del display, se ha elegido en este caso la ejecución desde 54000 ms a 56000 ms donde se incluye la ejecución de la escritura del registro del sistema.

```

Alerts
Altitude: Low Altitude
[54070] - displayTask: 23776 us
[54250] - accelerometerTask: 2604 us
[54310] - write in SD: 5082 us
[54310] - ultrasonicTask: 12331 us
[54500] - velocimeterTask: 1 us
[54500] - write in SD: 5670 us
[54500] - riskControlTask: 5700 us
[54500] - accelerometerTask: 2604 us
[54600] - presenceTask: 2 us
[54610] - write in SD: 5287 us
[54610] - ultrasonicTask: 12529 us
[54750] - accelerometerTask: 2604 us
[54910] - write in SD: 5084 us
[54910] - ultrasonicTask: 12279 us
[55000] - velocimeterTask: 1 us
[55000] - write in SD: 5085 us
[55000] - riskControlTask: 5115 us
[55000] - accelerometerTask: 2605 us
[55010] - write in SD: 10760 us
[55010] - writeRecordTask: 10887 us
[55200] - presenceTask: 2 us

[55210] - write in SD: 5292 us
[55210] - ultrasonicTask: 12540 us
[55250] - accelerometerTask: 2604 us
[55500] - velocimeterTask: 1 us
[55500] - write in SD: 9828 us
[55500] - riskControlTask: 9859 us
[55510] - accelerometerTask: 2604 us
[55520] - write in SD: 5680 us
[55520] - ultrasonicTask: 12928 us
[55750] - accelerometerTask: 2604 us
[55800] - presenceTask: 2 us
[55810] - write in SD: 5297 us
[55810] - ultrasonicTask: 12503 us
[56000] - velocimeterTask: 1 us
[56000] - write in SD: 5102 us
[56000] - riskControlTask: 5131 us
[56000] - accelerometerTask: 2604 us
[56010] - useSystemTask: 10002 us
[56020] - write in SD: 5103 us

a) [55200] Display Task
b) [56020] Display Task

```

[54040] Display Task Executed  
 [54300] Low Altitude - Level: 2  
 [54500] Level 2 activated!  
 [54600] Low Altitude - Level: 2  
 [54900] Low Altitude - Level: 2  
 [55000] Level 2 activated!  
 [55009] Started SD Recording.  
 Modes enabled:  
 Active/Started: True  
 Debug: False  
 General Risk:  
 Level: 2 (Severe Risk)  
 Specific Risk (Accelerometer/Tilt):  
 Risk Level: 0 (No Risk)  
 Alert: ""  
 Actual Tilt Angle (X, Y): (0.27, 1.29)  
 Counters (Correct, Wrong AxisX, Wrong AxisY): (12, 0, 0)  
 Specific Risk (Altimeter/Altitude):  
 Risk Level: 2 (Severe Risk)  
 Alert: "Low Altitude"  
 Actual Altitude "Simulated" (Real): 6960 m (29 cm)  
 Set Altitude (Min - Max): (41 - 50)  
 Counters (Wrong Low, Correct, Wrong High): (10, 0, 0)

Specific Risk (Velocimeter/Speed):  
 Risk Level: 0 (No Risk)  
 Alert: ""  
 Actual Speed: 824.54 km/h  
 Counters (Wrong Too Low, Wrong Low, Correct, Wrong High, Wrong  
 Specific Risk (Presence):  
 Risk Level: 0 (No Risk)  
 Alert: ""  
 Actual Presence: 2  
 Counters (Wrong Low, Correct, Wrong High): (0, 5, 0)

[55009] Ended SD Recording.  
 [55200] Low Altitude - Level: 2  
 [55500] Level 2 activated!  
 [55520] Low Altitude - Level: 2  
 [55800] Low Altitude - Level: 2  
 [56000] Level 2 activated!  
 d) [56010] Display Task Executed

Figuras 41a, 41b, 41c y 41d: Medidas de los tiempos de cómputo en display y registro; a, b) En TeraTerm; c, d) Registro en SD

Por resumir la ejecución y el tiempo de cómputo de las tareas, se puede deducir la siguiente tabla con los tiempos de cómputo de cada tarea, obviamente, la ejecución de la tarea del teclado y zumbador no están medidas, al no ser relevantes en esta prueba.

| Tarea                                  | Periodo       | Tiempo de Computo         |
|--|---------------|---------------------------|
| "Utilizar Sistema" / "useSystemTask"   | 2 s / 2000 ms | ~*1-10002 us / ~*1-11ms   |
| "Velocímetro" / "velocimeterTask"      | 500 ms        | ~1 us / *1ms              |
| "Ultrasonidos" / "ultrasonicTask"      | 300 ms        | ~12518 us / ~12-13 ms     |
| "Analizar Riesgos" / "riskControlTask" | 500 ms        | ~5300-9858 us / ~5-10 ms  |
| "Display" / "displayTask"              | 2 s / 2000 ms | ~23776 us / ~24 ms        |
| "Registro SD" / "writeRecordTask"      | 5 s / 5000 ms | ~10887 / ~11 ms           |
| "Acelerómetro" / "accelerometerTask"   | 250 ms        | ~2604 us / ~3 ms          |
| "Presencia" / "presenceTask"           | 600 ms        | ~2 us / *2ms              |
| "Escribir SD" / "SDFileWrite"          |               | ~5082-10760 us / ~5-11 ms |

Tabla 2: Medidas Tiempo de Cómputo

En la tabla superior (Tabla 2), se puede ver que todos los tiempos de cómputo son apenas detectables en comparación con el periodo de cada tarea. Una de las tareas que más cerca está de este plazo o "deadline" que supone el periodo es la tarea del ultrasonidos, que ocupa un 5% del tiempo hasta que vuelva a ejecutarse.

Se han indicado, de color rojo, las estimaciones "exageradas" en algunos plazos, todos ellos, suponen una lectura y una comparación, siendo los dispositivos "fiables" o que no requieren de una lectura adicional, por ello se ha redondeado al valor de 1 milisegundo.

También cabe indicar el tiempo de cómputo es por cada tarea, esto se puede saber dado que la tarea de "Utilizar Sistema" dispone de una función que la deja "dormida" 100 milisegundos. Con esto se puede saber además que los tiempos que se han medido son exactos en su gran mayoría, por el "timespan" que se incluye a su

lado, conociendo el instante de activación de cada tarea, éste se acerca al indicado por los tiempos de cómputo.

Para concluir este apartado, como ya se ha dicho en el requisito **RNFrend11**, se establecerá cómo tiempo de “*deadline*” el valor del periodo, dado que muchas de estas tareas ocupan normalmente el tiempo durante 5 o 10 milisegundos y un “plazo de finalización” menor del 10% del periodo resultaría muy ineficiente. La mejor solución es no establecer un “*deadline*” como tal, sino que éste se rija por el propio periodo de la tarea, dado el reducido tiempo de cómputo de cada tarea.

Se quiere indicar un “solapamiento” de escrituras por display, visible en el instante **55500** de la Figuras 41b, esto se debe a un cambio de prioridades, que al dejar el semáforo mutex del objeto protegido, es expulsado en la escritura por display. En un caso normal de ejecución, esto no ocurriría al existir sólo unas pocas tareas que usen el display.

## 9. Aspectos Sociales, Ambientales, Éticos y Legales

---

Este apartado está enfocado a definir y comentar los Aspectos Sociales, Ambientales, Éticos y Legales que pueden aplicar al proyecto realizado. Se comenzará con la identificación de algunos aspectos significativos en él, posteriormente, se elegirá una selección de ellos a resolver e integrar. Finalizando con la valoración final de estos aspectos seleccionados.

### 9.1. Identificación de Aspectos Relevantes

Para analizar e identificar los aspectos relevantes que afectan al proyecto realizado se responderán a las siguientes cuestiones.

- *¿Cómo se resuelve actualmente el problema que pretendo resolver?*

El problema que se pretende resolver es una reestructuración de las distracciones o problemas más comunes en los accidentes de avión. Actualmente el sistema utilizado por varias compañías de aeronaves comerciales es EGPWS (*“Enhanced Ground Proximity Warning System”*), este sistema es muy eficaz y funcional pero tiene el defecto de la falta de identificación de la altitud mínima de seguridad para evitar varias colisiones por terreno. También, debido al accidente causado en marzo de 2015 en los Alpes Franceses, se requiere de al menos 2 personas en todo momento en la cabina.

El proyecto realizado no pretende sustituir el sistema EGPWS ni desestimar su funcionalidad, si no ofrecer un sistema adicional para resolver la falta de algunas características. Este sistema sería capaz de retomar el vuelo con la ayuda de un piloto automático al encontrarse a una baja altitud o con una baja presencia entre otros síntomas en la cabina del avión.

- *¿Existe una necesidad real de proyecto?*

No es necesario, pero sin embargo, sería de gran ayuda para prevenir muchos informes de pilotos al detectar fallos provocados por el sistema de prevención. Todos ellos han sido rectificados por los propios pilotos o evitados con la ayuda de otros sistemas de prevención. Además de ser recomendable respetar una altitud mínima segura en el vuelo, y ser obligatorio mantener la presencia por la normativa emitida de la EASA (“European Aviation Safety Agency”).

Del mismo modo que se explica en algunos requisitos, el sistema pretende ser un añadido de los sistemas de prevención actuales.

- *¿El proyecto permitirá mejorar, directa o indirectamente, la calidad de vida de las personas?*

El simple hecho de ser un sistema que sería capaz de evitar varios incidentes debido a la escasa presencia en cabina o la altitud que se mantiene en el vuelo, podría mejorar potencialmente la calidad de vida de las personas y sobre todo, la seguridad mantenida durante los vuelos comerciales.

- *¿Hay algún colectivo que podría verse perjudicado? ¿En qué medida?*

El único colectivo que podría verse afectado negativamente sería el encargado de crear o diseñar sistemas de prevención en aviones, motivo por el cual, se ha evitado en la mayoría de los casos advertir de esos mismos síntomas o fallos.

- *¿El proyecto permitirá reducir el uso de recursos, la generación de residuos, contaminación o algún otro aspecto de la huella ecológica?*

El uso de utilizar una placa FPGA, la cual permitiría una reprogramación en caso de un posible fallo del sistema, reduciendo el gasto que supondría rediseñar un sistema de prevención completo. Esta reprogramación e incluso añadidos de nuevos dispositivos para medir parámetros, se resolvería con un cambio a un modelo más actualizado, basado en la anterior versión del mismo sistema sobre los mismos dispositivos y componentes, aprovechando la reutilización de los dispositivos.

- *¿Hay normativas, leyes o regulaciones que he de tener en cuenta al desarrollar el proyecto?*

Al tratarse de un sistema de prevención de riesgos en pleno vuelo, el sistema debe respetar las normativas de vuelo de cada país, adicionalmente, el sistema deberá ser aplicado a varios entandares de seguridad en sistema aeronáuticos como podría ser el estándar RTCA DO-178C, en este estándar se especifica el proceso a seguir durante el desarrollo eficiente del software, además de los factores que debe tener en cuenta el sistema. Del mismo modo, para verificar que los Requisitos de Entorno Físico sean cumplidos, será necesario cumplir y verificar estos con el estándar RTCA DO-160G con procedimientos para probar la integridad de todos los sistemas instalados en la aeronave.

Otro cumplimiento es la ordenanza europea “*Commission Regulation (EU) No 748/2012*” cuyo propósito es crear reglas a cumplir para el desarrollo del sistema y su implementación en el entorno final.

También se ha tenido en cuenta el efecto que puede suponer el sistema en los pilotos, para el cual se ha evitado que sea molesto o pueda distraer al piloto en lugar de sólo advertirle.

Al trabajar con dispositivos de Xilinx, Digilent y NodeMCU, junto con librerías y un entorno de libre distribución para hardware específico, se tiene en cuenta que estos dispositivos ya cumplen cualquier estándar de seguridad en componentes o normativas vigentes.

## 9.2. Selección

Los principales aspectos que deben tenerse en cuenta para el proyecto, dado que su objetivo, es crear un sistema seguro que no interfiera con sistemas de prevención actuales y respete los márgenes legales, estos tres aspectos son los que más debería afianzar el sistema.

En un segundo plano, el impacto medioambiental, es un factor para tener en cuenta en cualquier proyecto de diseño de sistemas, pero el enfoque de este proyecto sería obtener un sistema con una alta probabilidad de ser instalado en las aeronaves y respetar los estándares de seguridad vigentes internacionalmente. Del mismo modo, ofrecer un sistema seguro para la comunidad es un aspecto más relevante que un posible impacto medioambiental, aunque este último no se abandonaría, únicamente tendría un menor impacto en el sistema final.

## 9.3. Integración en el proyecto

En el proyecto se ha desarrollado un prototipo para estudiar la viabilidad, alternativas y adecuación del sistema. Dado el carácter de éste, no se ha creído conveniente hacer uso de los estándares de seguridad para el desarrollo. Sin embargo, estos estándares sí deben aplicarse en el desarrollo de un sistema definitivo.

Ante la resolución de este mismo aspecto, se ha planteado crear un sistema que no imposibilite de manera drástica otros sistemas similares, los cuales sí cumplen estos estándares de seguridad. También, al tratar de evitar respuestas molestas o que puedan distraer a los pilotos, se ha intentado aumentar esta seguridad ante el entorno. También, el añadido de un piloto automático “crítico” capaz de seguir la ruta indicada por el avión o el rumbo predefinido, ignorando posibles cambios manuales

que pueden suponer errores humanos, puede ser una de las medidas más drásticas, pero ésta se limita a una situación muy grave teniendo en cuenta varios factores del entorno o situaciones que puedan detectar los dispositivos.

Con esto, se busca una solución segura, que no suponga un gran impacto social, pero que siga siendo efectiva y pueda mejorar la calidad de vida de los pasajeros, por medio de un vuelo seguro que podría ser afectado por errores humanos, aunque se trate de un sistema de prevención de situaciones peligrosas y accidentes de aeronaves, y no de un piloto automático restrictivo ante los pilotos. Esto es posible realizarlo planificando correctamente qué acciones o qué “síntomas” pueden surgir y cómo estos podrían afectar a un vuelo comercial, también buscando un equilibrio entre distraer al piloto y no ser una respuesta efectiva, que se traduce en una respuesta capaz de alertar o advertir al piloto de una manera breve y eficaz.

En cuanto al aspecto medioambiental, aunque se ha comentado como un aspecto secundario a tomar en cuenta, se ha realizado una construcción de un prototipo compacto que hace uso de algunas soluciones ya implementadas. Debido a esto, se ha decidido utilizar la placa FPGA que ofrece de un bajo consumo. También dispone de una gran versatilidad junto con el aspecto “reutilizable” del código, al hacer uso de la arquitectura SoC (“System-On-Chip”). Ésta permite la instalación de un sistema operativo como es FreeRTOS, posibilitando implantar un sistema de tiempo real con sus herramientas. Además, estas placas permiten crear sistemas “hardware” o desde cero. Para ello, hacen uso de las herramientas asociadas, en este caso, Xilinx Vivado.

Estos objetivos o forma de resolver el problema junto con sus aspectos sociales y legales son los que se han seguido durante la búsqueda de requisitos y el desarrollo del sistema, al igual que en el diseño de un prototipo para probar el sistema en un entorno real, aunque no sea el definitivo.

## 9.4. Valoración Final

Como valoración final del sistema realizado, en los aspectos comentados, se ha podido ver la complejidad de la programación de un sistema de tiempo real “desde cero”, y el gran plazo de tiempo que puede suponer el desarrollo y diseño de éste. Tal y como se comentó anteriormente, la búsqueda de una solución que respete y cumpla los estándares emitidos por diferentes organizaciones y entornos es la más compleja de conseguir, dado que muchos de estos estándares son de carácter “no público” o no son de libre distribución, dado que estos sistemas entran en aspectos técnicos que pueden suponer grandes fallos en caso de ser publicados.

También, la disponibilidad del tiempo suficiente para valorar y resolver los posibles “fallos” del sistema, suponen una gran complicación y más al tratarse de un prototipo sencillo, que no tiene pensado ser un prototipo plenamente funcional en el

entorno final por las limitaciones de algunos dispositivos como el sensor de ultrasonidos.

En cambio se ha intentado que el sistema diseñado y desarrollado evite algunas situaciones que un pasajero de un vuelo comercial no desearía que ocurriesen, como es una inclinación muy pronunciada en cualquier eje en un plazo de tiempo corto, o la idea de estar a una altitud demasiado baja o alta en pleno vuelo, pasando por la sensación de una velocidad inadecuada. Se han tomado estos aspectos, porque muchos de ellos son la causa de varios accidentes causados por errores humanos, y son situaciones en las que un pasajero no se sentiría completamente seguro.

Por tanto, el sistema ofrece la sensación de seguridad a los pasajeros respetando las normativas de vuelo vigentes.

## 10. Planificación del proyecto y Tabla de Costes

---

Se ha añadido un apartado donde se explicará la planificación del proyecto, seguida con la ya comentada en el apartado donde se especifican las fases (Fases del proyecto) el cual no difiere en muchos aspectos de la planificación real. También se incluye la tabla de costes final del proyecto para valorar su coste.

### 10.1. Planificación Final

En la tabla inferior (Tabla 3) se indican los días tomados en función de la planificación comentada en un apartado previo. En este caso, la planificación seguida ha sido la misma que especificada. La única fase que se ha visto reducida es la **I6: Realización de diagramas adicionales**, que se ha visto reducida a realizar sólo el mapa de tareas. El motivo de esto es no extender más de lo necesario el contenido de esta memoria, el diagrama que se realizaría sería un diagrama de estados de cada tarea, pero se cree que con la explicación del código, el mismo y los casos de uso extendidos, es posible entender correctamente los estados por los que puede pasar el sistema. También, en caso de realizar este diagrama de estados, no sería muy complejo al usar en muchos casos valores específicos que no suponen mucha complejidad en el código más que incluir un semáforo en él.

| Fase | Titulo                                      | Fecha Inicio | Fecha Final | Tiempo total |
|------|---|--------------|-------------|--------------|
| I1   | Descripción del proyecto y su sistema       | 01/09/2020   | 23/10/2020  | 52           |
| I2   | Investigación sobre alternativas existentes | 18/09/2020   | 30/09/2020  | 12           |
| I3   | Identificar los requisitos del proyecto     | 13/10/2020   | 15/11/2020  | 33           |
| I4   | Realización de casos de uso                 | 15/11/2020   | 26/11/2020  | 11           |
| I5   | Realización de un diagrama de componentes   | 10/11/2020   | 05/12/2020  | 25           |
| I6_1 | Realización de un diagrama de Tareas        | 02/12/2020   | 21/12/2020  | 19           |
|      |   |              |             |              |

|         |   |            |            |    |
|---------|---|------------|------------|----|
| D1      | Valoración del material e Investigación       | 18/09/2020 | 12/10/2020 | 24 |
| D2/D3   | Creación del código                           | 18/09/2020 | 18/12/2020 | 91 |
| D2/D3_1 | * Pruebas de Dispositivos                     | 18/09/2020 | 30/09/2020 | 12 |
| D2/D3_2 | * Creación de Tareas                          | 03/10/2020 | 10/11/2020 | 38 |
| D2/D3_3 | * Pruebas de Tareas                           | 11/11/2020 | 10/12/2020 | 29 |
| D2/D3_4 | * Conjunto del sistema                        | 01/12/2020 | 18/12/2020 | 17 |
| D4      | Realización de pruebas con el prototipo       | 22/12/2020 | 31/12/2020 | 9  |
| D5      | Verificación y valoración final del prototipo | 28/12/2020 | 05/01/2021 | 8  |

Tabla 3: Tabla con fechas de realización de cada fase

En la última columna de esta tabla se indican los días que han sido utilizados para cada fase, estos son orientativos, dado que algunas fases como la **D1** o **I1**, pueden ser modificadas hasta el último momento sin necesidad de volver a rehacer el sistema. También, muchas fases, debido a la división entre las fases de especificación y las de desarrollo, se han podido realizar y coordinar simultáneamente durante el tiempo de realización. En la siguiente figura (Figura 42) se puede ver el diagrama de Gantt generado con estos datos.

Tal y como se puede ver, una de las tareas más costosas del proyecto es la compuesta de **D2** y **D3** que se ha subdividido en varias subfases del proyecto, y se han combinado bajo un mismo conjunto. Estas fases son las encargadas de la creación y pruebas del sistema, se han decidido combinar dado que al probar los dispositivos. Se requería realizar un código que ya se planificó como un componente del sistema final, al cual que se requería introducir en una tarea para que se realice la función indicada.

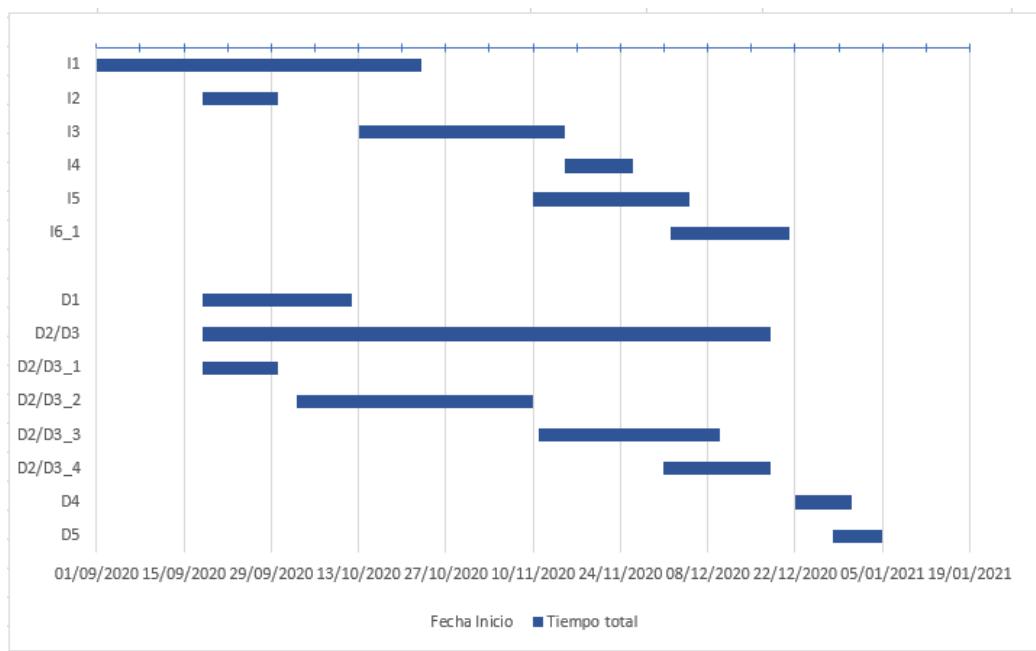


Figura 42: Diagrama de Gantt del proyecto

Junto con la tabla anterior, se puede ver el progreso en relación a la planificación seguida en función del tiempo, también cómo se comentó previamente, se ve una coordinación de los distintos ámbitos que completan al sistema. Del mismo

modo, es visible que la creación del mapa de tareas **I6\_1** coincide su finalización en una fecha similar a las fases de desarrollo **D2/D3**, en concreto a la subfase donde se crea y especifica el conjunto del sistema, para el cual se ha ido tomando ideas entre sí para obtener una solución coherente.

Aunque se había propuesto una planificación “a grandes rasgos”, en el desarrollo del proyecto completo, establecen un buen conjunto de objetivos a cumplir, sin llegar a definir detalles por cada dispositivo o tarea.

## 10.2. Tabla de Costes

En este apartado se mostrará el coste del proyecto, basándose en los precios de cada componente o dispositivos, junto con el enlace del dispositivo. En la siguiente tabla (Tabla 4) se puede ver el precio (sin IVA) de cada dispositivo y finalmente el valor final del proyecto.

| Componente/<br>Dispositivo        | Enlace (en caso existente)  | Precio          |
|-----------------------------------|---|-----------------|
| Placa FPGA<br>Zybo Z7-10          | <a href="https://es.rs-online.com/web/p/herramientas-de-desarrollo-de-fpga/1643486/">https://es.rs-online.com/web/p/herramientas-de-desarrollo-de-fpga/1643486/</a>   | 186.68 €        |
| Módulo<br>PmodLED                 | <a href="https://www.digikey.es/product-detail/es/digilent-inc/410-076/1286-1144-ND/4090177">https://www.digikey.es/product-detail/es/digilent-inc/410-076/1286-1144-ND/4090177</a>   | 5.70 €          |
| Módulo<br>PmodKYPD                | <a href="https://www.digikey.es/products/es?keywords=pmodkyp">https://www.digikey.es/products/es?keywords=pmodkyp</a>   | 9.16 €          |
| Sensor<br>Ultrasonidos<br>HC-SR04 | <a href="https://www.digikey.es/product-detail/es/adafruit-industries-llc/3942/1528-2711-ND/9658069">https://www.digikey.es/product-detail/es/adafruit-industries-llc/3942/1528-2711-ND/9658069</a>   | 3.22 €          |
| Potenciómetro<br>deslizante       | <a href="https://es.rs-online.com/web/p/potencios/2499418/">https://es.rs-online.com/web/p/potencios/2499418/</a>   | 2.16 €          |
| Acelerómetro 3<br>ejes            | <a href="https://es.rs-online.com/web/p/sensores-de-acelerometro/9054665/">https://es.rs-online.com/web/p/sensores-de-acelerometro/9054665/</a>   | 6.95 €          |
| Sensor de<br>movimiento           | <a href="https://www.digikey.es/product-detail/es/parallax-inc/555-28027/555-28027-ND/1774435">https://www.digikey.es/product-detail/es/parallax-inc/555-28027/555-28027-ND/1774435</a>   | 12.22 €         |
| Sensor FC-51                      | <a href="https://www.tiendatec.es/arduino/sensores/575-sensor-fc-51-detector-obstaculos-por-infrarrojos-para-arduino-8405751480002.html">https://www.tiendatec.es/arduino/sensores/575-sensor-fc-51-detector-obstaculos-por-infrarrojos-para-arduino-8405751480002.html</a> | 1.95 €<br>(x2)  |
| Led RGB 4 pines                   | <a href="https://es.rs-online.com/web/p/leds/8614307/">https://es.rs-online.com/web/p/leds/8614307/</a>   | 1.24 €          |
| Zumbador                          | <a href="https://es.rs-online.com/web/p/componentes-de-piezo-buzzer/0457011">https://es.rs-online.com/web/p/componentes-de-piezo-buzzer/0457011</a>   | 0.60 €          |
| ESP 8266                          | <a href="https://es.rs-online.com/web/p/kits-de-desarrollo-de-microcontroladores/1245505/">https://es.rs-online.com/web/p/kits-de-desarrollo-de-microcontroladores/1245505/</a>   | 15.00 €         |
| Cable Pmod                        | <a href="https://www.digikey.es/product-detail/es/digilent-inc/240-021-2/1286-1102-ND/5456480">https://www.digikey.es/product-detail/es/digilent-inc/240-021-2/1286-1102-ND/5456480</a>   | 4.88 €          |
| <b>Total</b>                      |   | <b>251.81 €</b> |

Tabla 4: Tabla de Costes del proyecto

## 11. Conclusión

---

Por medio de este proyecto se ha conseguido desarrollar un sistema complementario a los sistemas GPWS. El sistema hace uso de una placa FPGA de Xilinx junto con varios dispositivos de Digilent. Por medio de estos y otros dispositivos como el acelerómetro o sensor de ultrasonidos, es capaz de detectar riesgos en el avión. El añadido del teclado matricial ha permitido incluir funcionalidades al sistema, una de ellas es permitir modificar el techo de vuelo simulado por medio de un código.

Otro aspecto, es el diseño del prototipo, en éste se ha buscado una solución compacta y simple para el sistema, pero perdiendo el aspecto “profesional” que podría tener éste. Por eso se ha decidido no mejorar el prototipo por una posible falta de tiempo y fallos que puedan ocurrir y sean irreversibles.

También, se opina que se ha cumplido el objetivo de mostrar el ciclo de vida de un sistema de tiempo real, siguiendo el modelo en V. Se ha buscado para ello incluir varios anexos con información adicional para conocer las pruebas o el motivo para seleccionar un IDE respecto a otro. En el desarrollo del proyecto se ha intentado dar una perspectiva para mostrar el avance y progreso de este tipo de proyectos, para agilizar futuros proyectos similares. La valoración personal del proyecto y no solo del sistema, se considera satisfactoria, dado que la idea inicial se ha visto mejorada con varios dispositivos adicionales.

El conjunto final del sistema, y por ende del proyecto, ha sido satisfactorio, se ha mejorado la perspectiva que se tenía sobre los sistemas de tiempo real y la complejidad de su desarrollo “desde cero”. También el uso de una placa que admita un sistema operativo para ello ha supuesto un desafío y la gran cantidad de diferentes dispositivos ha mejorado la percepción del sistema. Con él ha sido posible planificar futuras mejoras de este mismo proyecto, o los cambios que puedan tener en el entorno final. Se opina que las soluciones tomadas, como la medición de la altitud, escalado y la posibilidad de cambiar el techo de vuelo, han permitido planificar qué soluciones alternativas se pueden tomar con otros dispositivos o con otra funcionalidad. De este modo, las tecnologías y dispositivos utilizados pueden ser útiles también en proyectos similares.

Se quiere indicar también que el proyecto original se había diseñado de una manera distinta. Siendo posible aplicarse al entorno real en el cual podría ocurrir,

## *Conclusión*

usando una cámara para detectar la presencia en cabina. Esta última opción fue pospuesta y finalmente, rechazada debido a una posible carga excesiva de la placa FPGA. Incluyendo que la utilización de algunos de sus módulos que interfieren con los dispositivos ya instalados, por ello se buscó una solución equivalente a la detección de presencia.

Para finalizar, se quiere valorar el aprendizaje obtenido por medio de este proyecto. Su realización ha permitido aumentar el conocimiento de la tecnología FPGA usando placas SoC como la utilizada. También, ha mejorado la visión sobre algunos de los dispositivos utilizados en el proyecto, cómo es el sensor de ultrasonidos o la comunicación I2C. En resumen, tanto en el aspecto de Ingeniería de Requisitos como en el desarrollo del sistema, se valora de una forma positiva.

## 12. Líneas futuras

---

En este apartado final se van a comentar algunas mejoras o líneas futuras del proyecto realizado. Muchas de estas mejoras no son complejas, pero sí requieren de una planificación adicional a la inicial por su complejidad.

La primera mejora que se puede implementar es un rediseño del montaje físico del prototipo para darle un diseño más adecuado y estructurado. La realización de este prototipo podría ser en una estructura de metacrilato, con la posibilidad de crear un entorno para simular la cabina de un avión o disponer de los dispositivos con una dispersión mayor. Esto requerirá posiblemente, rediseñar el diseño de los puertos Pmod y la realización de varias placas PCB para los dispositivos a utilizar.

También, como otra línea futura no descartable es el añadido del módulo de la cámara, dado que éste se ha rechazado por una posible falta de espacio y una falta de tiempo para crear la detección e identificación de presencia. Con un prototipo mejorado y más tiempo de desarrollo, se podría valorar la inclusión de un sistema que detecte la presencia en cabina por medio del procesamiento de imágenes.

Una tercera mejora más simple, sería añadir otras funcionalidades haciendo uso de los dispositivos de este primer prototipo. Añadiendo su activación por medio de códigos introducidos por el teclado matricial. A modo de ejemplo, se podría añadir un código para realizar pruebas de los dispositivos sin encender el sistema, o una calibración de ellos.

Dado que la placa FPGA dispone de puertos HDMI, Ethernet, USB y audio, se podría dar una funcionalidad adicional usando alguno de estos puertos. Esto requeriría una investigación suplementaria para conocer el uso de estos por medio de un sistema operativo como puede ser FreeRTOS o similar. En este aspecto también se incluiría el uso del módulo Pcam de Digilent para una visualización de la cabina, sin aplicar una detección de presencia.

## Referencias

- [1] Xilinx, «What is an FPGA?,» [En línea]. Available: <https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html>. [Último acceso: 4 Enero 2021].
- [2] D. Streams, «La Tecnología FPGA: futuro y beneficios principales,» 20 Noviembre 2016. [En línea]. Available: <https://www.davidstreams.com/2016/11/20/la-tecnologia-fpga-futuro-y-beneficios-principales/>. [Último acceso: 6 Enero 2021].
- [3] A. I. López, «3 razones por las que ocurren los accidentes aéreos,» 31 Julio 2018. [En línea]. Available: <https://culturacolectiva.com/tecnologia/accidentes-aereos-razones-por-las-que-ocurren>. [Último acceso: 6 Enero 2021].
- [4] M. A. Pérez, «¿Por qué chocan los aviones?,» 13 Agosto 2014. [En línea]. Available: <https://blogthinkbig.com/causas-de-accidentes-aereos>. [Último acceso: 6 Enero 2021].
- [5] N. Catalán, «Detalles del siniestro del Airbus A320,» 31 Julio 2017. [En línea]. Available: [https://elpais.com/elpais/2015/03/24/media/1427195967\\_070152.html](https://elpais.com/elpais/2015/03/24/media/1427195967_070152.html). [Último acceso: 5 Enero 2021].
- [6] F. Sanchez, «Controles sorpresa y nuevas normas en cabina, lecciones de Germanwings,» 24 Marzo 2016. [En línea]. Available: [https://elpais.com/internacional/2016/03/23/actualidad/1458753834\\_097198.html](https://elpais.com/internacional/2016/03/23/actualidad/1458753834_097198.html). [Último acceso: 6 Enero 2021].
- [7] NBC News, «Air Canada Jet Just 59 Feet Above Ground in Near Miss at San Francisco Airport,» 3 Agosto 2017. [En línea]. Available: <https://www.nbcnews.com/news/us-news/air-canada-jet-just-59-feet-above-ground-near-miss-n789081>. [Último acceso: 8 Enero 2021].
- [8] NIHF, «C. Donald Bateman,» [En línea]. Available: <https://www.invent.org/inductees/c-donald-bateman>. [Último acceso: 4 Enero 2021].
- [9] Honeywell, «EGPWS,» [En línea]. Available: <https://aerospace.honeywell.com/en/pages/enhanced-ground-proximity-warning-system>. [Último acceso: 5 Enero 2021].
- [10] AMSADD, «History of Ground Proximity Warning,» [En línea]. Available: <https://www.amsadd.com/egpws.htm>. [Último acceso: 5 Enero 2021].
- [11] Honeywell, «Mark V EGPWS,» [En línea]. Available: <https://aerospace.honeywell.com/en/learn/products/terrain-and-traffic-awareness/mark-v-egpws>. [Último acceso: 5 Enero 2021].
- [12] AviaciónD, «Sistemas de emergencia: el EGPWS,» 9 Enero 2017. [En línea]. Available:

- <http://aviaciond.com/sistemas-de-emergencia-gpws/>. [Último acceso: 6 Enero 2021].
- [13] Honeywell, «Performance Assessment Of Pilot Response To EGPWS,» [En línea]. Available: [https://aerospace.honeywell.com/content/dam/aero/en-us/documents/learn/products/terrain-and-traffic-awareness/research/Performance\\_Assessment\\_Of\\_Pilot\\_Response\\_To\\_EGPWS.pdf](https://aerospace.honeywell.com/content/dam/aero/en-us/documents/learn/products/terrain-and-traffic-awareness/research/Performance_Assessment_Of_Pilot_Response_To_EGPWS.pdf). [Último acceso: 5 Enero 2021].
- [14] NASA, «ASRS Database Report Set,» 31 Mayo 2017. [En línea]. Available: [https://www.amsadd.com/cftt\(1\).pdf](https://www.amsadd.com/cftt(1).pdf). [Último acceso: 5 Enero 2021].
- [15] AMSADD, «Design Evolution of AMSADD,» [En línea]. Available: <https://www.amsadd.com/evolution.htm>. [Último acceso: 5 Enero 2021].
- [16] AMSADD, «AMSADD Profile Vs EGPWS and GPWS,» [En línea]. Available: <https://www.amsadd.com/comparison.htm>. [Último acceso: 5 Enero 2021].
- [17] AVweb, «NASA's CFIT App,» 27 Julio 2015. [En línea]. Available: <https://www.youtube.com/watch?v=Erz4cdRNWZI>. [Último acceso: 6 Enero 2021].
- [18] IEEE, «Especificación Requisitos IEEE 830,» 22 Octubre 2008. [En línea]. Available: <https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf>. [Último acceso: 7 Enero 2021].
- [19] Xilinx, «Zybo Z7-10,» [En línea]. Available: <https://www.xilinx.com/products/boards-and-kits/1-pukimv.html>. [Último acceso: 4 Enero 2021].
- [20] S. J. Bleiker, «Island-style global FPGA architecture. A unit tile consists of Configurable Logic Block (CLB), Connect Box (CB) and Switch Box(SB).,» Marzo 2018. [En línea]. Available: [https://www.researchgate.net/figure/Island-style-global-FPGA-architecture-A-unit-tile-consists-of-Configurable-Logic-Block\\_fig6\\_323820898](https://www.researchgate.net/figure/Island-style-global-FPGA-architecture-A-unit-tile-consists-of-Configurable-Logic-Block_fig6_323820898). [Último acceso: 5 Enero 2021].
- [21] Xilinx, «Zynq-7000 SoC,» [En línea]. Available: <https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html>. [Último acceso: 4 Enero 2021].
- [22] Xilinx, «SDSoC,» [En línea]. Available: <https://www.xilinx.com/products/design-tools/software-zone/sdsoc.html>. [Último acceso: 4 Enero 2021].
- [23] Xilinx, «Vitis,» [En línea]. Available: <https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html>. [Último acceso: 4 Enero 2021].
- [24] NASA, «Aircraft Rotations,» 5 Mayo 2015. [En línea]. Available: <https://www.grc.nasa.gov/www/k-12/airplane/rotations.html>. [Último acceso: 7 Enero 2021].
- [25] Isaac, «ESP8266: el módulo WIFI para Arduino,» [En línea]. Available: <https://www.hwlible.com/esp8266/>. [Último acceso: 8 Enero 2021].

## *Referencias*

- [26] Arduino, [En línea]. Available: <https://www.arduino.cc/>. [Último acceso: 4 Enero 2021].
- [27] ElComercio.es, «La UE recomienda ahora que haya siempre dos personas en cabina,» 27 Marzo 2015. [En línea]. Available: <https://www.elcomercio.es/politica/201503/27/varias-aerolineas-obligaran-haya-20150327104328-rc.html>. [Último acceso: 6 Enero 2021].
- [28] Digilent, «Zybo Z7 Reference Manual,» [En línea]. Available: <https://reference.digilentinc.com/reference/programmable-logic/zybo-z7/reference-manual>. [Último acceso: 4 Enero 2021].
- [29] SinapTec, «Tutorial #23 ESP8266 – Obtener Inclinación con MPU6050 (GY-521),» 15 Octubre 2017. [En línea]. Available: <http://www.sinaptec.alomar.com.ar/2017/10/tutorial-23-esp8266-obtener-inclinacion.html>. [Último acceso: 6 Enero 2021].
- [30] InvenSense Inc., «MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2,» 19 Agosto 2013. [En línea]. Available: <https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf>. [Último acceso: 6 Enero 2021].
- [31] C. J. Fisher, «Using An Accelerometer for Inclination Sensing,» 6 Mayo 2011. [En línea]. Available: <https://www.digikey.com/en/articles/using-an-accelerometer-for-inclination-sensing>. [Último acceso: 7 Enero 2021].
- [32] Eclipse, [En línea]. Available: <https://www.eclipse.org/>. [Último acceso: 7 Enero 2021].
- [33] Xilinx, «SDSoC - Boards, Kits & Modules,» [En línea]. Available: <https://www.xilinx.com/products/design-tools/software-zone/sdsoc.html#boardskits>. [Último acceso: 6 Enero 2021].
- [34] Xilinx, «SDSoC - Download/Buy,» [En línea]. Available: <https://www.xilinx.com/products/design-tools/software-zone/sdsoc.html#buy>. [Último acceso: 6 Enero 2021].
- [35] Xilinx, «XSDK,» [En línea]. Available: <https://www.xilinx.com/products/design-tools/embedded-software/sdk.html>. [Último acceso: 4 Enero 2021].
- [36] Xilinx, «SDAccel Development Environment,» [En línea]. Available: <https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html>. [Último acceso: 4 Enero 2021].
- [37] Xilinx, «Vitis AI,» [En línea]. Available: <https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html>. [Último acceso: 5 Enero 2021].
- [38] Xilinx, «Vitis Libraries,» [En línea]. Available: <https://www.xilinx.com/products/design-tools/vitis/vitis-libraries.html>. [Último acceso: 6 Enero 2021].
- [39] Xilinx, «XRT,» [En línea]. Available: <https://www.xilinx.com/products/design-tools/vitis/xrt.html>. [Último acceso: 5 Enero 2021].

- [40] Alveo, «Alveo Product Brief,» [En línea]. Available:  
<https://www.xilinx.com/publications/product-briefs/alveo-product-brief.pdf>. [Último acceso: 4 Enero 2021].
- [41] Xilinx, «Vivado,» [En línea]. Available: <https://www.xilinx.com/products/design-tools/vivado.html>. [Último acceso: 7 Enero 2021].
- [42] «Zybo Z7-20 Pcam 5C Demo,» 24 Junio 2020. [En línea]. Available:  
<https://github.com/Digilent/Zybo-Z7-20-pcam-5c/tree/v2019.1-1>. [Último acceso: 4 Enero 2021].
- [43] OSDN, «TeraTerm Home Page,» 1 Julio 2020. [En línea]. Available:  
<https://ttssh2.osdn.jp/index.html.en>. [Último acceso: 8 Enero 2021].
- [44] «Zybo-Z7-master.xdc,» 12 Septiembre 2017. [En línea]. Available:  
<https://github.com/Digilent/digilent-xdc/blob/master/Zybo-Z7-Master.xdc>. [Último acceso: 4 Enero 2021].
- [45] Xilinx, «XADC Wizard v3.0,» 1 Abril 2015. [En línea]. Available:  
[https://www.xilinx.com/support/documentation/ip\\_documentation/xadc\\_wiz/v3\\_0/pg091-xadc-wiz.pdf](https://www.xilinx.com/support/documentation/ip_documentation/xadc_wiz/v3_0/pg091-xadc-wiz.pdf). [Último acceso: 6 Enero 2021].

## A1. Anexo I: Investigación IDE

En este anexo se aporta información adicional al apartado 7.1.2: Investigación respecto a los IDE de Xilinx. Dado que el apartado ya comentado pretende ser autoconclusivo, al tratar aspectos comparativos de los entornos SDSoc y Vitis. Se cree innecesaria gran parte de esta información, la cual se deja para un carácter meramente informativo en caso de requerir de detalles al apartado indicado.

### A1.1. Comparativa de IDEs

Para comenzar la comparativa, tal y como se comentó en el apartado de la memoria 0, Vitis sustituye a SDSoc junto con otras herramientas. No hay motivo para rechazar ninguno, porque ambos siguen disponibles en la página de Xilinx. Y ambos tienen tutoriales completos para varias placas, tema que se comentará en otro subapartado.

La comparativa se realizará a través de 2 fases, una primera fase “teórica” o técnica, donde se detallará por medio de las páginas oficiales y otros recursos, el aspecto teórico de cada uno de los IDEs. Posteriormente, se realizará una fase “práctica”, donde se valorará el uso de cada herramienta por medio de un programa de prueba su uso.

#### A1.1.1. Parte Teórica

Se comenzará con los datos que ofrece el software SDSoc Development Environment [22], según la propia página, es un entorno de desarrollo embebido que utiliza C/C++/OpenCL como lenguajes de programación. Para ello, se apoya en Eclipse [32], con el cual se complementa de manera eficaz.

Esta herramienta fue creada especialmente para programar de manera secuencial los procesadores Zynq-7000 SoC [21], como el que dispone la placa. Entre los dispositivos que Xilinx muestra como compatibles e integrados con los *SDK* (*Software Development Kit*) [33], únicamente aparecen los kit de evaluación de la gama de procesadores Zynq-7000. Esto supone un problema, porque la placa FPGA Zybo Z7 no pertenece a ese grupo, por tanto, requiere un programa adicional, Vivado. Con este programa, se podría especificar con más detalle cómo funcionará la placa y permitirá la ejecución en SDSoc o Xilinx SDx IDE, como se llama el programa en cuestión.

Otro de los aspectos para tener en cuenta es la licencia [34], en este caso, Xilinx ofrece una prueba de 60 días para SDSoc y 30 días para Vivado que para una comparación es suficiente. En caso de no disponer de ese periodo de prueba, la licencia es muy costosa, llegando casi al precio de 1000 euros, siendo prácticamente inasequible económicamente para un proyecto de este tipo en solitario. Por suerte, la compra de la placa nueva ofrece un cupón para canjear por una licencia del programa durante un año, tras ese periodo, no sería posible actualizar el conjunto de programas. De esta manera, la opción de utilizar SDSoc no es automáticamente descartada.

La última característica que ofrece la página del entorno es que su última versión es la 2019.1, dejando de actualizarse y confirmando que no existirán posteriores versiones. Esto se debe al problema que generaba tener diferentes programas para distintas arquitecturas de procesadores. Entre ellos estaba Xilinx SDK [35] que permitía utilizar los procesadores Zynq-7000 y la arquitectura MicroBlaze de algunas placas FPGA y SDAccel [36] para procesadores Alveo de Xilinx. En marzo de 2019, se decidió por crear el entorno Vitis [23], éste unifica todas estas arquitecturas bajo un mismo programa que, por las políticas de Xilinx, permite además de un periodo de prueba, una descarga gratuita para estudiantes y profesionales.

Sin embargo, aunque esté poco actualizado, esto equivale a una existencia de gran cantidad de soporte ajeno a Xilinx para usar este programa, y varios recursos adicionales.

La otra herramienta, es Vitis Unified Platform [23], como se comentó previamente, unifica todos los programas anteriores a marzo de 2019 bajo una única aplicación. Esta aplicación no sólo tiene las mismas utilidades, también se encuentra en constante desarrollo, incluyendo nuevas librerías y aplicaciones preparadas.

Una de las novedades, es el entorno de Vitis AI [37], el cual permite la inclusión de Inteligencia Artificial en el uso de las placas FPGA, en concreto las placas con tecnología Alveo. También, ofrece una gran cantidad de librerías para diferentes materias [38], entre ellas, Análisis, Compresión o Seguridad de Datos, además de librerías para el uso de imágenes y su gestión. En relación con los lenguajes de programación, Vitis permite entre otros C/C++, además de Python u OpenCL. Esto se puede realizar a través de un conjunto de sistemas operativos preparados para ello, *Xilinx Runtime (XRT)* [39].

Todo ello, con la posibilidad de usar el despliegue en la nube, característica no disponible en SDSoc. Una de las importantes mejoras de este programa, es la gran versatilidad respecto a hardware y su programación por parte de Vitis, tanto de placas FPGA como tarjetas de aceleración basadas en Alveo para centros de datos, como las U250 [40]. Pero, al utilizar una placa FPGA considerada de desarrollo, Zybo Z7-10, no hay ningún SDK ya integrado para ésta en Vitis.

La solución, al igual que con SDSoc, es utilizar Vivado [41], en el caso de SDSoc, la última versión soportada es la 2019.1, en Vitis, permite la última disponible, 2020.2.

Por tanto, es adecuado pensar que las versiones previas funcionaran en versiones futuras.

Una de las ventajas que mantiene Vitis sobre SDSoC, es el precio de la licencia, dado que Vitis es gratuito al ser una herramienta que se apoya en el *Open Source* y está disponible para todas las placas de Xilinx por ese motivo. Del mismo modo, Vivado sí es de pago, pero ofrece una versión limitada gratuita para algunos dispositivos, con la cual se puede trabajar junto con Vitis sin coste adicional.

Se puede ver que Vivado [41] es necesario en ambos entornos, debido a que no existe ningún SDK ya integrado en ninguno de ambos. Esta herramienta te permite especificar el funcionamiento de la placa FPGA por medio de VHDL o Verilog. Como en el proyecto no es objetivo programar por medio de lenguajes de descripción hardware (HDL), se utilizarán diseños de circuitos integrados o diseños IP. Con esto, es posible encargar a Vivado que realice las conexiones entre bloques IP, creando los ficheros correspondientes.

Un bloque IP (*Intellectual Property*) es un fichero o conjunto de ellos que facilita la construcción de la lógica en la placa FPGA. Aunque son códigos escritos en VHDL o Verilog que se han desarrollado como un conjunto para simplificar el diseño.

Con esta parte teórica de la comparación, se puede ver, que el funcionamiento entre ambos es prácticamente igual, por tanto, para realizar una comparativa más acertada, se debe valorar la parte práctica de cada entorno.

### A1.1.2. Puesta en Marcha

La parte práctica de la comparativa será una puesta en marcha de ambos entornos, siendo la manera más adecuada de tratar este segmento. Teniendo en cuenta que ambos entornos requieren de un programa adicional, Vivado, la instalación de ambos ya tiene en cuenta este programa, por lo que no es posible decidir qué programa es más adecuado, en términos de descarga. Si es cierto, que la versión con Vitis ocupa varios gigabytes más en el disco, pero es un proceso bastante intuitivo el cual no se pretende detallar.

En cambio, el uso de Vivado de manera rápida para proyectos así es general en ambos, y sí requiere de una breve guía para evitar la necesidad de conocer VHDL, aunque con éste se obtenga un resultado más preciso al buscado.

La versión utilizada de Vivado es 2019.1, pero se ha comprobado que en versiones posteriores, salvo un detalle, se realiza de la misma manera. Una vez se tenga abierto el entorno de Vivado (Figura 43), se procede a crear un nuevo proyecto usando “**Create Project**”, con el nombre deseado para el proyecto, en este caso “test”.

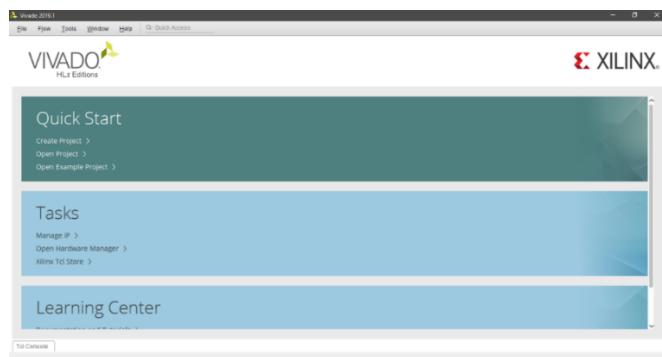
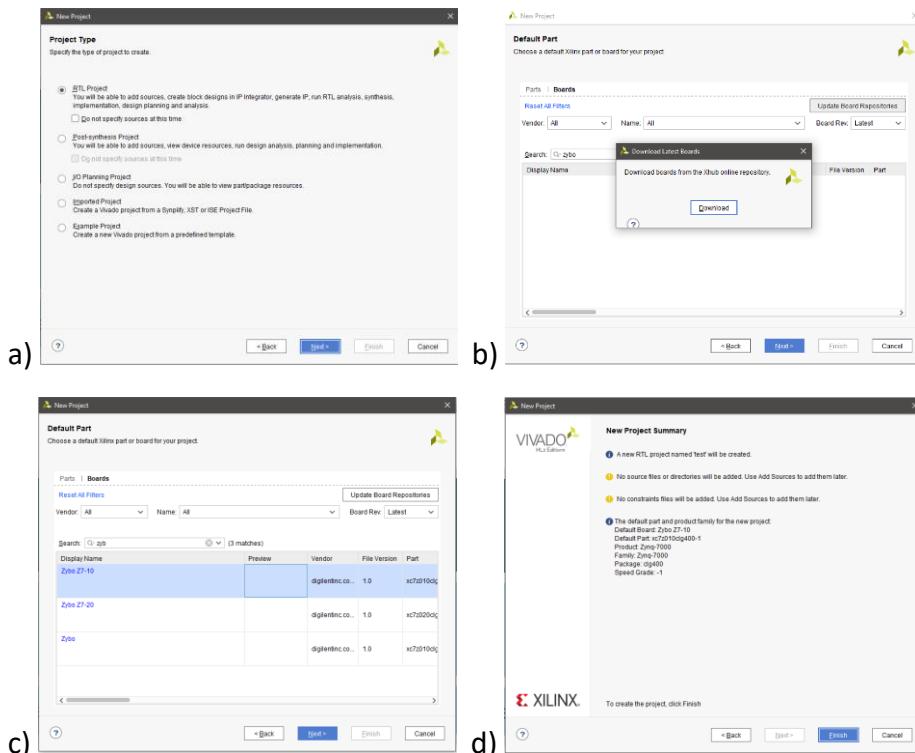


Figura 43: Captura Inicio Vivado 2019.1

En las siguientes capturas, se debe indicar que es un proyecto RTL (Figuras 44a), para que se pueda realizar diseños usando componentes IP, que acelera el proceso para enfocarse en el uso de C/C++. No es necesario indicar ningún fichero fuente, y en el caso de querer establecer el funcionamiento de puertos, se puede añadir un fichero de limitaciones (constraints) en formato “.xdc”. El siguiente paso que es clave en el proyecto, es indicar la placa que se va a utilizar, seleccionando la pestaña “Boards”. Al no tener el SDK por defecto instalado, es necesario incluir nuevos modelos de placas, usando la opción “Update Board Repositories” (Figuras 44b).

En la versión Vivado 2020.1, es posible encontrar el modelo específico y sólo instalar el necesario. En versiones previas, se instalan todos los modelos existentes. Tras tener el modelo de la placa instalado, se procede a seleccionarlo (Figuras 44c). El resultado del proyecto para una prueba serán los que se ven en la última imagen inferior (Figuras 44d).



Figuras 44a, 44b, 44c y 44d: Capturas de la creación de un proyecto en Vivado; a) Tipo proyecto; b) Actualización placas; c) Selección placa; d) Finalizar

## Anexo I - Comparativa de IDEs

Una vez creado el proyecto, es necesario crear un diseño IP para disponer de todos los dispositivos necesarios. Para ello, en las opciones de la parte izquierda se selecciona “Create Block Design” y se le pone el nombre deseado al diseño. (Figura 45)

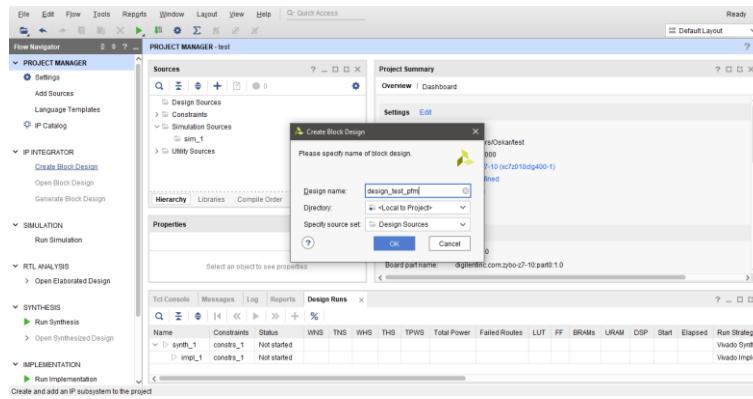
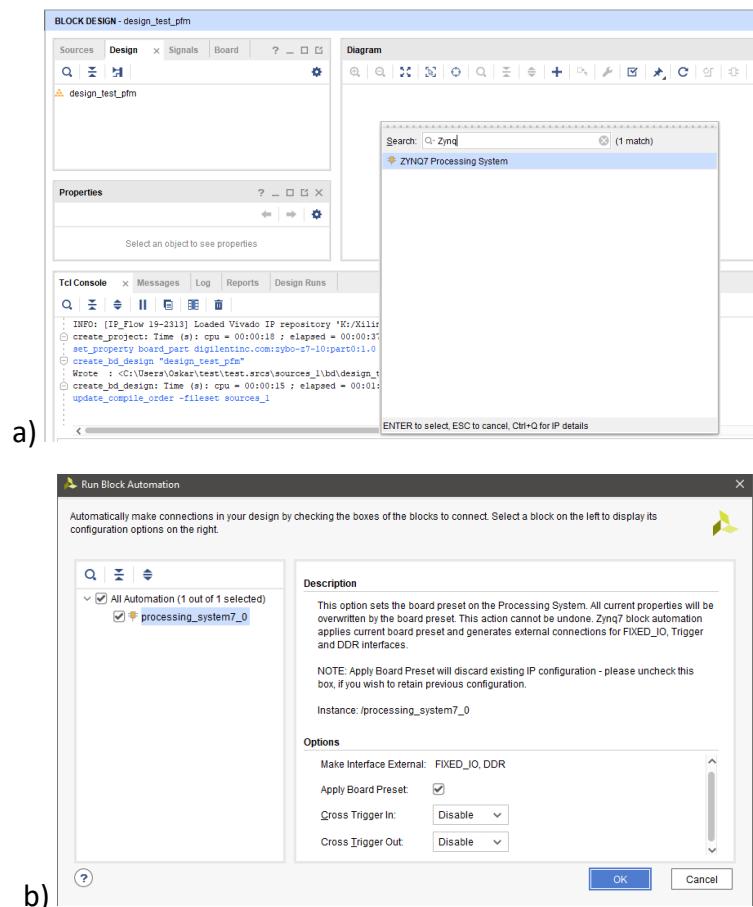


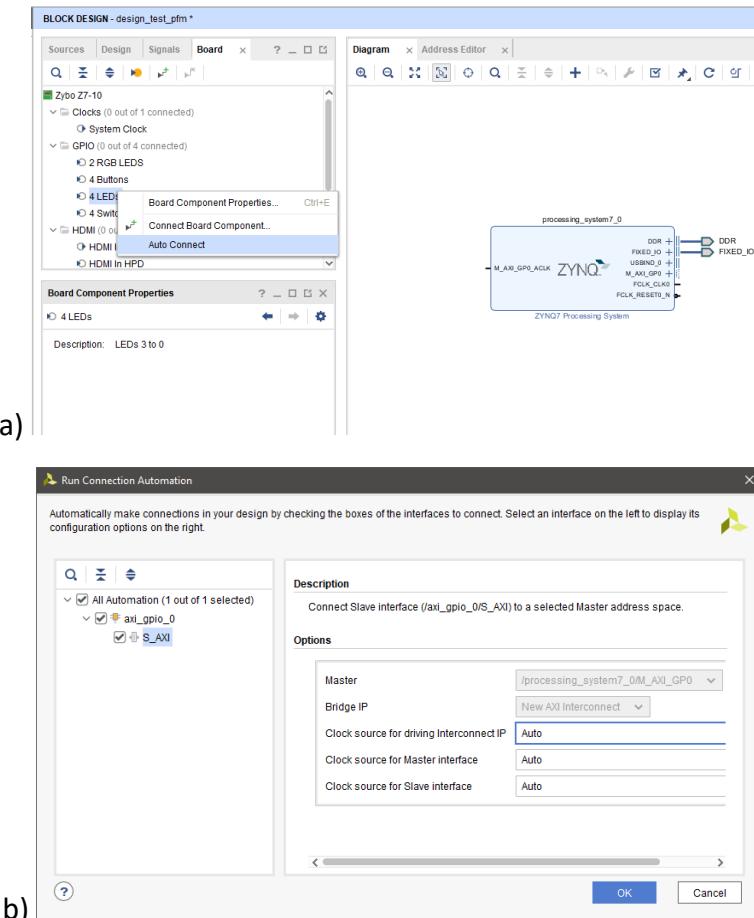
Figura 45: Creación Diseño en Vivado 2019.1

En el diseño, se requiere de un bloque principal que sea el procesador “**ZYNQ7 Processing System**” esta parte es obligatoria, y no se puede omitir (Figuras 46a). Al incluirlo, aparecerá un aviso con la opción “**Run Block Automation**”, éste conectará de manera automática el procesador con los elementos necesarios para su uso (Figuras 46b), sin modificar ninguna opción. Posteriormente, se le puede dar uso a los puertos MIO, I2C, SPI que dispone la placa, pero en la demo no se requiere.



Figuras 46a y 46b: Añadir bloque Zynq7 al diseño IP; a) Incluir b) Auto conectar

Tras esto, es posible incluir los dispositivos I/O que se desean, en la opción “**Board**”, es posible seleccionarlos. Se recomienda usar “**Auto Connect**”, para evitar un paso intermedio entre estos, a medida que se añadan, se irán incluyendo bloques “AXI\_GPIO”, en pares de dispositivos, por lo que es relevante el orden que se desea. En este caso, se añaden “**Leds**” y “**Button**”, respectivamente (Figuras 47a). Al finalizar de añadirlos, se ejecuta la opción de conectarlos automáticamente usando “**Run Connection Automation**”, que creará un sistema con los GPIO deseados y adaptados a la placa. (Figuras 47b)



Figuras 47a y 47b: Capturas creación interfaz conexión GPIO al diseño IP; a) Añadido y selección b) Auto conexión

En el caso de querer usar un dispositivo que no se encuentra ni en la configuración de la placa, o en la lista en “**Board**”, como son los puertos Pmod, que requieren añadir estos bloques mediante GitHub (Figura 48). Un punto que cambia con respecto a las versiones actualizadas de Vivado es que estos bloques IP no se encuentran visibles y son “*branches*” sin ser oficiales y en fase de pruebas.

## Anexo I - Comparativa de IDEs

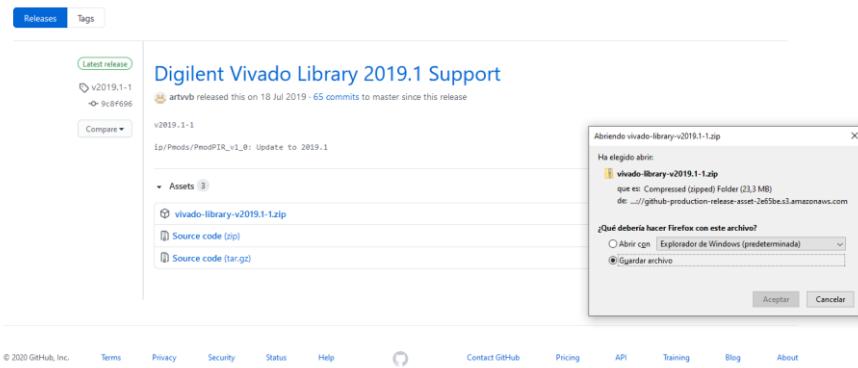
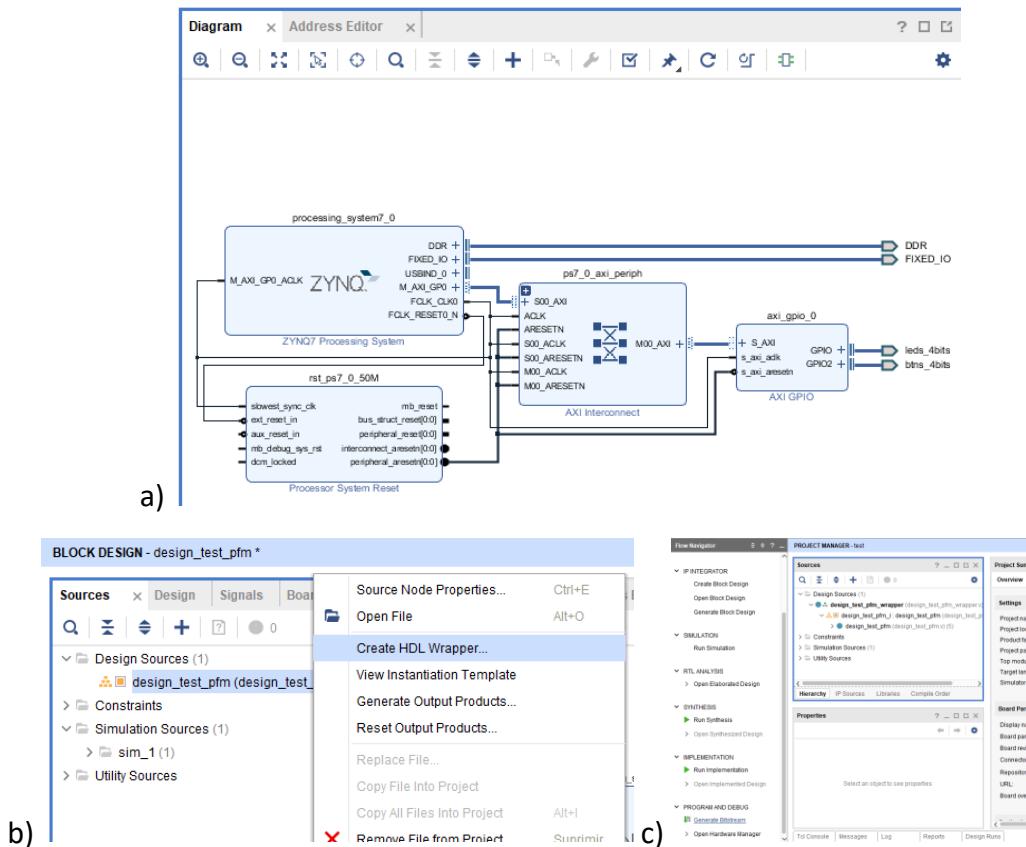


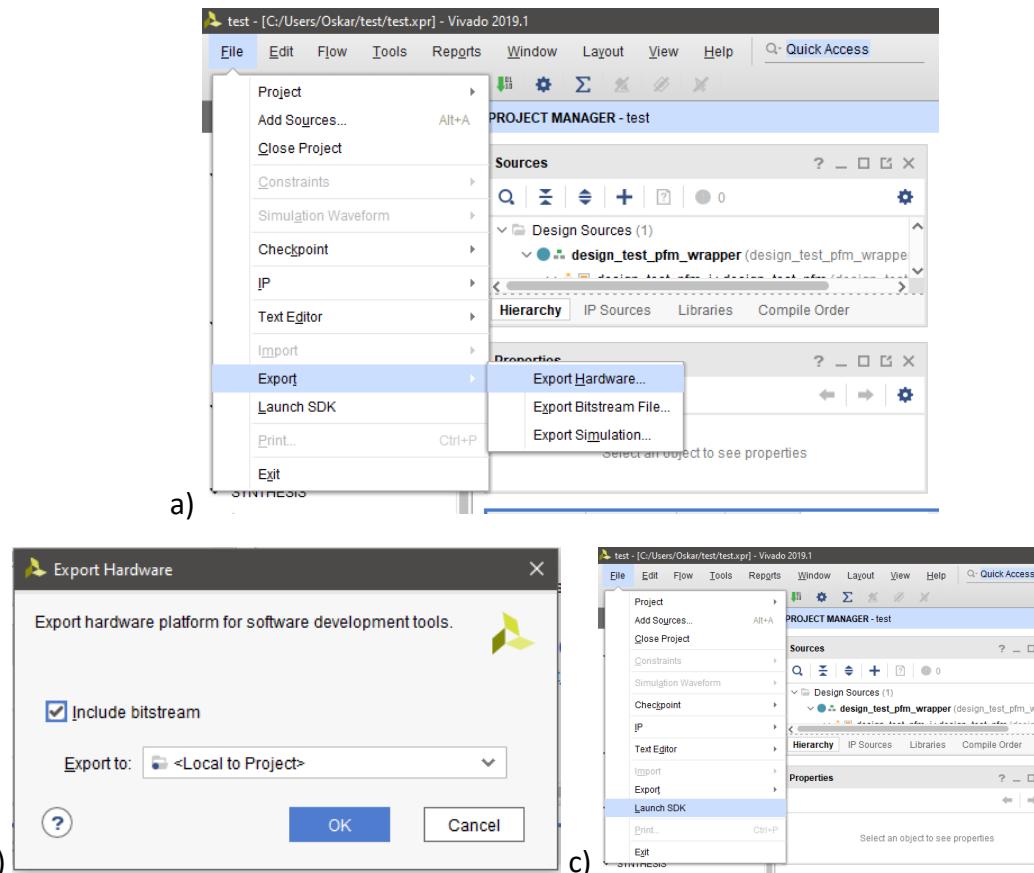
Figura 48: Añadido de librerías Pmod Diligent para Vivado 2019.1

El sistema resultante se puede ver en la imagen inferior (Figuras 49a), donde todos los dispositivos se encuentran conectados a la unidad de procesamiento Zynq7. Para comenzar a usar Xilinx SDK o Vitis, es necesario, crear un “wrapper”, por ello, se crea con la opción “Create HDL wrapper” en el propio diseño (Figuras 49b). Con esto se crea un conjunto de códigos en VHDL, que conectan el procesador con el diseño creado. De esta manera se evita la programación en lenguajes de desarrollo hardware (HDL). También es necesario crear el fichero con el bitstream del diseño, para ello, se selecciona la opción “Create Bitstream” del panel izquierdo, con las opciones por defecto (Figuras 49c).



Figuras 49a, 49b y 49c: Capturas de creación de "HDL wrapper"; a) Diseño completo; b) Creación de HDL wrapper; c) Ejecución "bitstream"

De esta manera sólo es necesario preparar el fichero para usar el hardware diseñado, por ello, se debe elegir la opción “**Export Hardware**” en ““File” > “**Export**” > “**Export Hardware**”” (Figuras 50a). Se debe incluir el bitstream creado previamente (Figuras 50b), y en caso de la versión actualizada del programa, dirigido a sistemas con software empotrado. Una vez preparado, se puede abrir Xilinx SDK o Vitis usando “**Launch SDK/Vitis**” en la pestaña de “**File**” en Vivado 2019.1 (Figuras 50c), y en “**Tools**” en Vivado 2020.1. Con esto se ejecuta el entorno correspondiente, y se finalizará el uso de Vivado en el proyecto.



Figuras 50a, 50b y 50c: Capturas creación hardware y exportación; a) Exportar hardware; b) Añadir bitstream; c) Ejecutar Xilinx SDK

### A1.1.2.1. Xilinx SDK

En esta primera instancia, es necesario crear un proyecto de aplicación, también es posible crear una plataforma, pero el proyecto previo se autocontiene en este sentido. Se especifica el lenguaje que se quiere utilizar, plataforma y nombre a elección. Dejando las opciones como salen en la figura siguiente (Figura 51). En este caso, se ha elegido usar el código de “**Hello World!**” y modificarlo para usar los botones y leds.

## Anexo I - Comparativa de IDEs

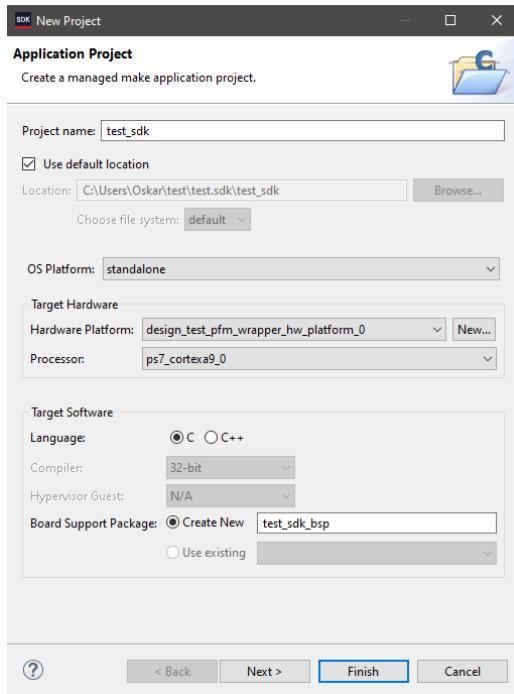


Figura 51: Creación nuevo proyecto Xilinx SDK y configuración

Se ha creado el siguiente código de prueba (Tabla 5), que dependiendo del botón pulsado, muestra un patrón en los leds monocromos integrados. Adicionalmente, se ha incluido un mensaje que indica que botón se ha pulsado, el cual es mostrado por UART. Y para facilitar la lectura, el ciclo se “duerme” durante 200000 microsegundos, o 0.2 segundos.

```
#include "xgpio.h"
#include "xparameters.h"
#include "xil_printf.h"
#include "sleep.h"

int main ()
{
    XGpio gpio;
    u32 btn, led;

    XGpio_Initialize(&gpio, 0);

    XGpio_SetDataDirection(&gpio, 2, 0xF);
    XGpio_SetDataDirection(&gpio, 1, 0x0);

    while (1)
    {
        btn = XGpio_DiscreteRead(&gpio, 2);

        if(btn == 0b0000)
        {
            led = 0x0;
        }

        else if(btn == 0b0001)
        {
            xil_printf("boton 0 pulsado\n\r");
            led = 0x1;
        }
        else if(btn == 0b0010)
        {
            xil_printf("boton 1 pulsado\n\r");
            led = 0xA;
        }

        sleep(200000);
    }
}
```

```

}
else if(btn == 0b0100)
{
    xil_printf("boton 2 pulsado\n\r");
    led = 0x2;
}
else if(btn == 0b1000)
{
    xil_printf("boton 3 pulsado\n\r");
    led = 0xF;
}
else
{
    xil_printf("multiples botones\n\r");
    led = 0x8;
}

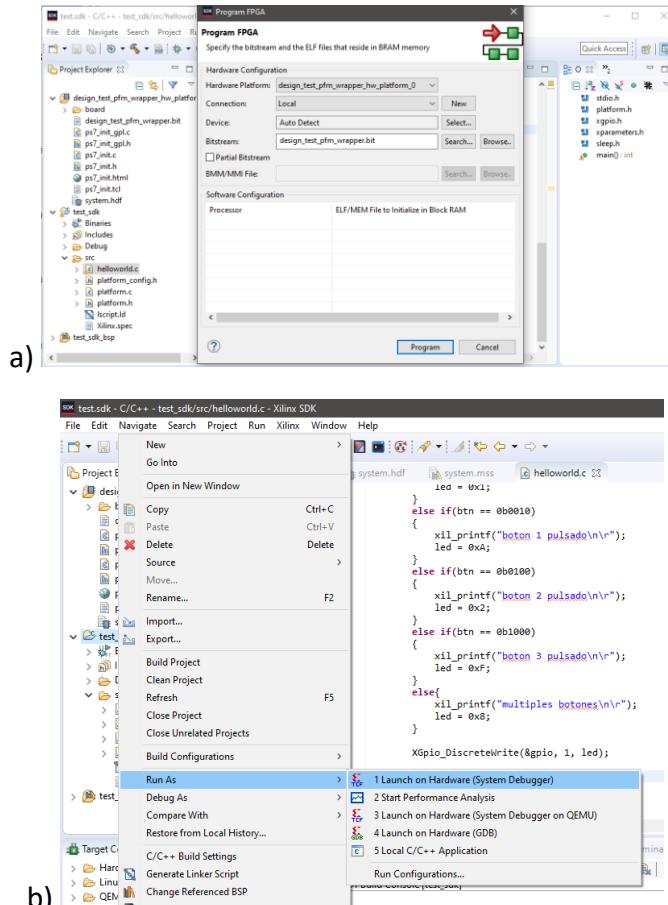
XGpio_DiscreteWrite(&gpio, 1, led);

usleep(200000);
}
}

```

Tabla 5: Código prueba diodos LED y botones en Zybo Z7-10

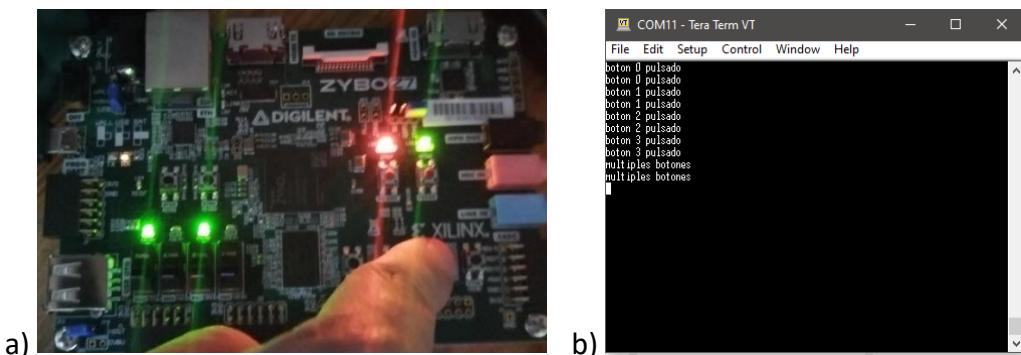
Para programar la placa, se utiliza la opción “**Program FPGA**”, que tiene varios iconos en el entorno (Figuras 52a). Para ejecutar el código sin grabarlo en la memoria *flash*, se debe seleccionar “**Run As... Launch On Hardware (System Debugger)**” pulsando el botón derecho en la aplicación. Tras varios segundos, el sistema estará en ejecución (Figuras 52b).



Figuras 52a y 52b: Programación y ejecución en placa FPGA; a) Programación de FPGA con código; b) Ejecución del código remotamente

## Anexo I - Comparativa de IDEs

En las siguientes imágenes, se puede ver el resultado al pulsar un botón de la placa y el resultado en los LEDs. También se registra en el terminal TeraTerm, aplicación usada para este propósito (Figuras 53ab).



Figuras 53a y 53b: Prueba de la placa; a) Fotografía de la placa durante la prueba; b) Captura de la salida por UART por medio de TeraTerm

### A1.1.2.2. Vitis Unified Platform

En el caso de utilizar Vitis, el procedimiento para instalar es prácticamente igual al de cualquier herramienta de Xilinx. Para iniciar un proyecto, usando Vivado 2020.1, se siguen los mismos pasos que para las versiones anteriores, ya detalladas, con los cambios indicados.

Al ejecutar Vitis, se abrirá una pantalla como la que se puede ver a continuación (Figura 54). En este caso se usará directamente la opción de crear un proyecto de aplicación.

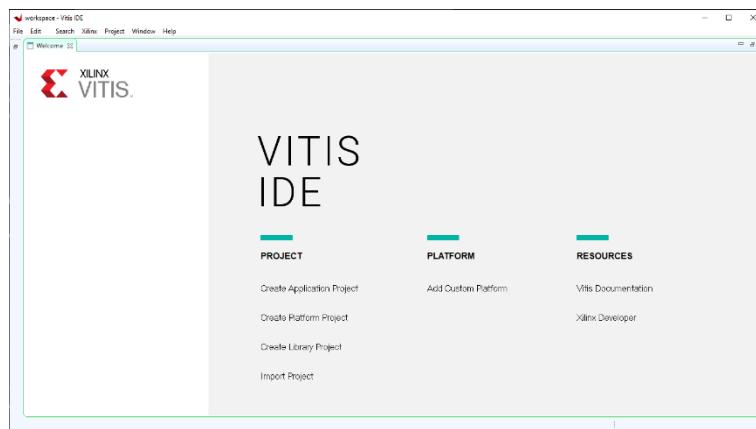
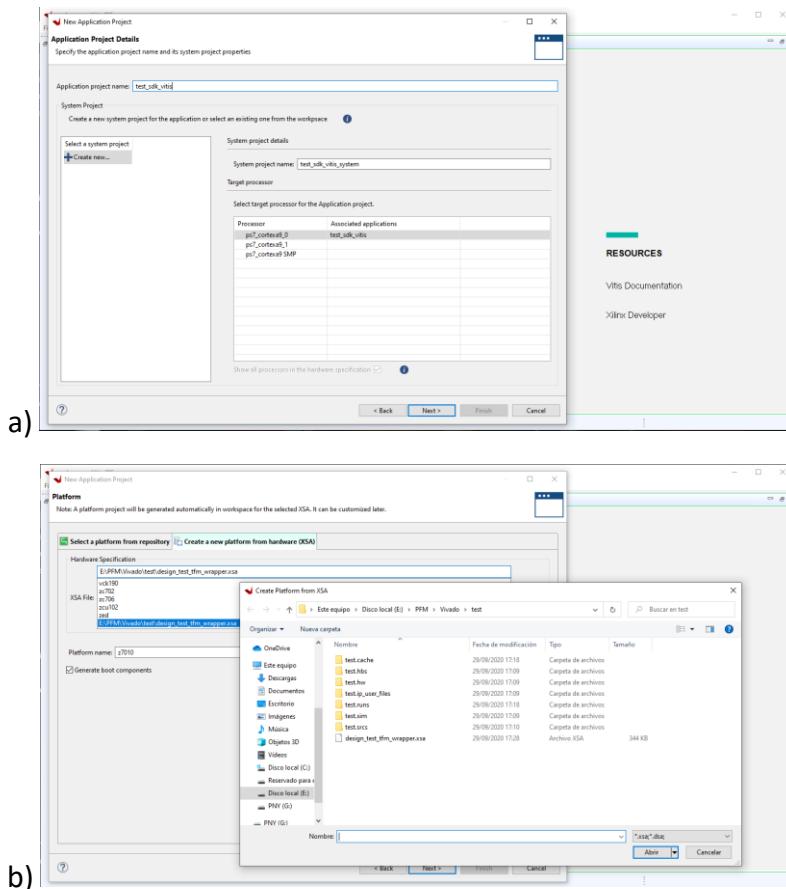


Figura 54: Captura entorno Vitis 2020.1

Al no disponer de la plataforma de Zybo Z7-10, se debe usar la que se ha creado previamente en Vivado 2020.1 en el fichero guardado con formato ".xsa". Es recomendable usar la misma versión de Vivado que de Vitis. Se le puede poner un nombre para esta plataforma, en este caso, z7010 que es un nombre alternativo a

Zybo Z7-10 (Figuras 55a). A continuación, se le establecerá un nombre a la aplicación, en este caso se dejará la configuración por defecto (Figuras 55b).



Figuras 55a y 55b: Captura de la creación de un proyecto nuevo en Vitis 2020.1; a) Creación; b) Asignación de diseño hardware

Respecto al dominio de la aplicación (Figura 56), se deja en la ya establecida automáticamente, pero se puede modificar el sistema operativo para que utilice “freeRTOS10” para Zybo y Xilinx. De igual manera que en Xilinx SDK, se usará el código en C anterior (Tabla 5).

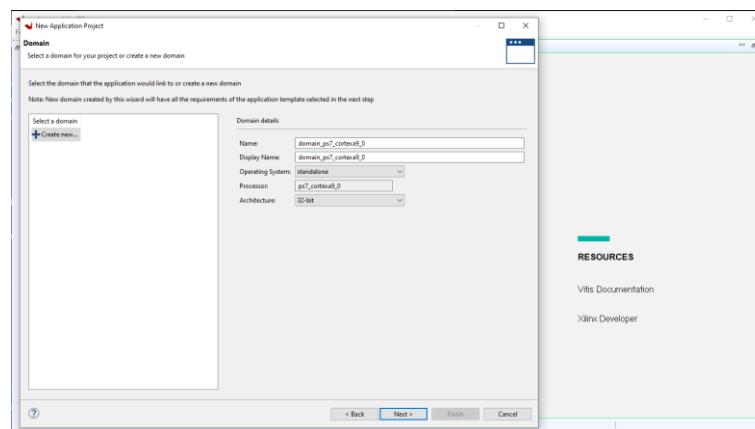
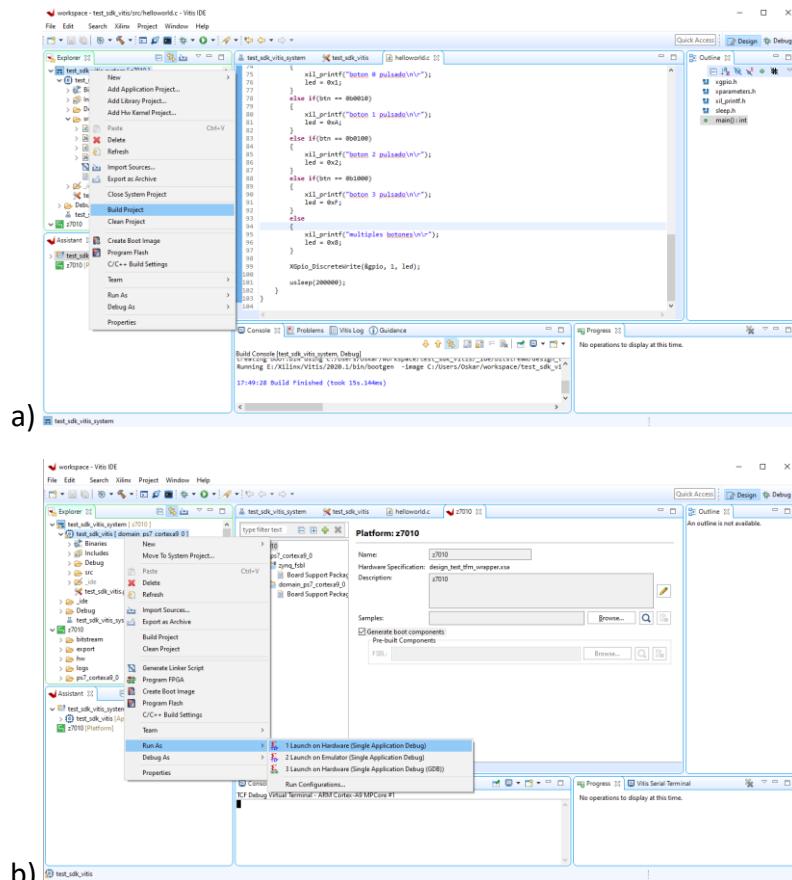


Figura 56: Dominio de la aplicación

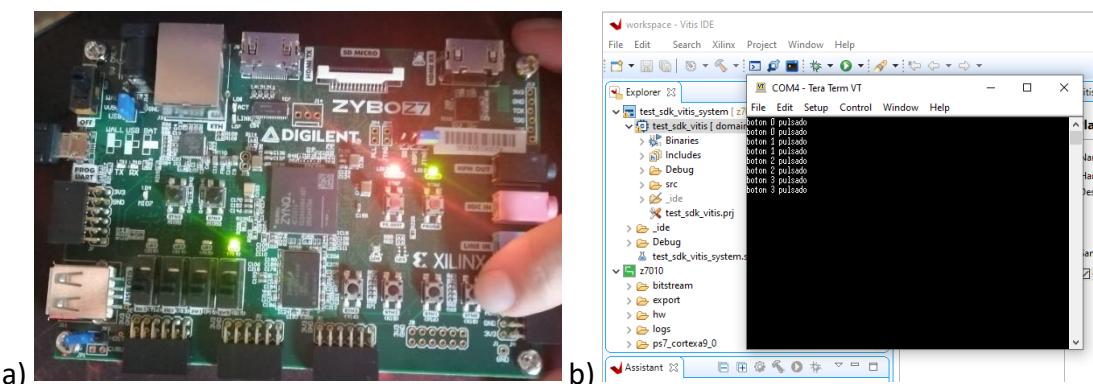
## Anexo I - Comparativa de IDEs

Para cargar el código en la placa, es un proceso intuitivo, donde se construye la aplicación, usando “**Build Project**” (Figuras 57a). Después, se puede aplicar directamente el código en la placa usando “**Run As... Launch On Hardware (System Debugger)**”, cómo en el anterior entorno (Figuras 57b). Automáticamente, se programará la FPGA con el código, y se podrá poner en ejecución usando la acción “**Run As...**” previa.



Figuras 57a y 57b: Compilación y ejecución del código; a) Compilación; b) Ejecución en dispositivo

La ejecución del programa en la placa es igual a la ya probada anteriormente, sin ningún cambio, pero sí que se ha realizado en un IDE diferente (Figuras 58ab).



Figuras 58a y 58b: Prueba de la placa; a) Fotografía de la placa durante la prueba; b) Captura de la salida por UART por medio de Teraterm

## A1.2. Resolución

Como se puede apreciar, tanto las características de SDSoc como de Vitis, son las mismas en relación con la placa Zybo Z7-10. Ambos entornos requieren el uso de Vivado, en una versión distinta, y ambos admiten esta placa para su programación. Esto se debe, como se comentó anteriormente, a que Vitis es la plataforma unificada de Xilinx para programar SDKs y placas Xilinx. Por tanto, es complicado sacar una conclusión de qué entorno es mejor para este proyecto, porque ambos permiten el uso adecuado, teóricamente, del sistema diseñado. Es posible calificar a Vitis, como la versión actualizada de SDSoc.

Se ha comentado también, que no existen guías o manuales aprobados por Xilinx para la versión Vivado o Vitis 2020.1. Esto es cierto dado que son manuales de Digilent, sin aprobar por Xilinx de cómo usar el entorno, pero tienen una gran comunidad para su ayuda, y el uso apenas cambia entre Xilinx SDx y Vitis según la parte práctica de la comparativa. Respecto al uso de Vivado, también existe este “retraso” en la documentación, pero se debe a una falta de cambio en el entorno y su funcionamiento, incluso desde su primera versión 2015.4, los manuales indican que funciona con determinados cambios, pero no requiere una guía nueva. En este sentido, y siendo fieles a la documentación, no existe ninguna para la versión de SDSoc 2019.1 ni para Vitis 2020.1, dado que el cambio es apenas perceptible entre ambos e intuitivamente se puede manejar con la misma guía. De esta manera, se deduce, que tampoco hay cambios en la documentación, porque son guías sin actualizar o no aceptadas por Xilinx en ambas versiones.

Respecto a la placa que se utilizará, Zybo Z7-10, tampoco hay ninguno de estos entornos que lo integre nativamente, se requiere un programa adicional, Vivado para crear la plataforma y utilizar la placa. Así pues, tampoco es un punto donde destaque algún entorno. También existe una gran cantidad de artículos y documentación por internet para resolver incidencias, que se encuentran fuera del alcance de Xilinx, por si no hay suficiente información en ésta.

Un punto donde se puede apreciar un cambio es respecto a la licencia y precio. En caso de no disponer de una licencia, SDSoc te ofrece una prueba de 60 días, que para este proyecto puede ser limitado, y el coste de una licencia por medio de Xilinx es elevado (1000 dólares). Por el otro lado, Vitis se anunció como un entorno gratuito para placas Xilinx, similar a SDSoc, donde guarda un parecido. Sin embargo, esto no es del todo cierto, Vitis es gratuito pero Vivado es limitado en dispositivos. En este caso, es un punto que afecta a ambos, porque Vivado tiene un coste más elevado que el propio SDSoc, y su prueba es de sólo 30 días. Existe una versión de Vivado gratuita, pero está limitada a ciertos dispositivos comerciales, por suerte, la mayoría de los procesadores Zynq como MicroBlaze están incluidos, pero en caso de usar un dispositivo específico industrial, el coste de la licencia de Vivado asciende hasta los

## Anexo I - Resolución

3000 dólares, en caso de no disponer de una. Por tanto, el conjunto de SDSoc saldría más costoso en ese aspecto contando que su licencia ya es de pago.

Finalmente, un punto donde SDSoc no puede competir contra Vitis, dado que ambos admiten el tiempo real y existen varias guías de cómo implementarlo usando FreeRTOS, es la compatibilidad con librerías de libre distribución y Python. Una de las características de Vitis, es Vitis AI [37], la cual permite incluir la inteligencia artificial en los proyectos usando OpenCV o similares, que en sistemas de detección y procesamiento de imagen es prácticamente necesario. También, incluye varias aplicaciones, ejemplos y librerías para otros muchos usos que tienen los sistemas embebidos o la Inteligencia Artificial.

|                     | SDSoc DE  | Vitis Platform  |
|---------------------|---|---|
| Versiones           | 2016.4 – 2019.1   | 2019.2 – 2020.1   |
| Guías               | No, adaptaciones intuitivas de versión 2016.4.<br>Existe documentación más detallada. | No, adaptaciones intuitivas de SDSoc.<br>Existe documentación.                  |
| SDK para Zybo Z7-10 | No, usando Vivado.  | No, usando Vivado.  |
| Precio (Licencia)   | Licencia SDSoc: 1000\$<br>Posible licencia Vivado: 3000\$                             | Licencia Vitis: Gratuita<br>Posible licencia Vivado: 3000\$                     |
| Añadidos            | No.   | Vitis AI y librerías Open-Source.   |
| Uso de OpenCV       | Uso de OpenCV o herramientas, por medio de software adicional (p.e. Petalinux).       | Uso de OpenCV o herramientas, por medio de software adicional (p.e. Petalinux). |

Tabla 6: Resumen comparativa de IDEs

En la tabla previa (Tabla 6), se ha resumido los datos obtenidos de cada entorno. Se puede concluir, que aunque Vitis sea una versión actualizada de SDSoc, la incorporación de Vitis AI y librerías *open-source*, ofrece una visión más adecuada ante futuros proyectos. Estos proyectos, pueden utilizar directamente la inteligencia artificial, y al disponer de una gran cantidad de librerías *open-source* tiene más adaptación a futuros proyectos.

Por tanto, el entorno elegido inicialmente sería Vitis 2020.1, por ese motivo. Aunque existan muy pocas diferencias entre ellos, se opina que elegir el entorno más actualizado ofrecerá mejoras en futuros proyectos, en general. En un siguiente anexo, se realizará una investigación en más detalle con la placa, entorno y dispositivos para comprobar que no existe ningún problema entre entornos y dispositivos.

## A1.3. Carga en la placa FPGA

En este apartado, se indicará el modo de cargar el sistema en la placa FPGA. Este método es general para todos los proyectos y placas FPGA similares a la utilizada en este proyecto, Zybo Z7-10. Es necesario indicar que este procedimiento es totalmente opcional, no es necesario para que el sistema funcione. A diferencia de la carga por JTAG o por cable, este método borra la memoria previa de la placa y carga un fichero de arranque internamente “*BOOT.BIN*”.

Para grabar el sistema en la placa, es necesario crear este fichero “*.BIN*” que permite activar el sistema al encender la placa. Este procedimiento borrará cualquier otro sistema o programa ya grabado. Las ventajas de grabar en la memoria *flash* del sistema, dado que es necesario disponer de FreeRTOS en la placa, te permite usar cualquier equipo como display, sin necesidad de disponer de Vivado o Vitis, o usar únicamente una fuente de alimentación para alimentar a la placa FPGA.

Para crear el archivo “*BOOT.BIN*” y “*.elf*”, se utiliza la opción “**Create Boot Image**” del proyecto indicado, visible en la siguiente figura (Figura 59). Automáticamente te especificará los nombres necesarios, por lo que ya será suficiente con aceptar la configuración propuesta. Tras este paso, es necesario cargar el programa en la memoria flash de la placa FPGA. En este sentido, la placa tras permitir una carga del proyecto por JTAG, debería ser capaz de contener el mismo sistema en su memoria interna.

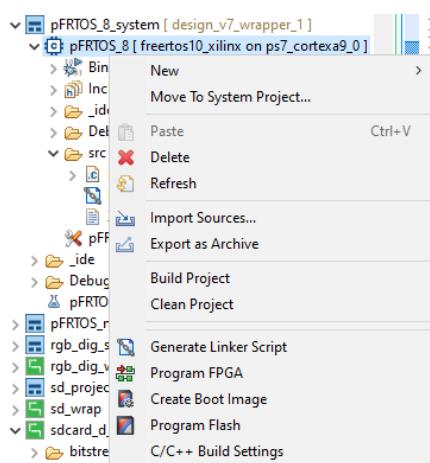


Figura 59: Opciones para grabar el sistema

Para grabar en la memoria de la placa, es necesario elegir la opción “**Program Flash**”, también visible en la anterior imagen (Figura 59), siendo aconsejable elegir la opción de “**System**” para grabar tanto la aplicación realizada como FreeRTOS. Automáticamente al elegir el proyecto a instalar se encuentran los archivos y ficheros necesarios, este procedimiento debe ser realizado con el *jumper* o selector de JP5 en el modo “JTAG”.

## *Anexo I - Carga en la placa FPGA*

Tras grabar, se deberá apagar la placa, cambiar el *jumper* J5 a “QSPI” para utilizar el programa grabado en la memoria *flash* de la placa. Con este sencillo procedimiento ya debería de estar grabado en la placa FPGA el proyecto completo, y es posible prescindir de las herramientas de programación de Xilinx.

## A2. Anexo II: Investigación Dispositivos

### A2.1. Dispositivos testados

Otro de los aspectos que se deben realizar para conocer qué requisitos se pueden incluir y qué dispositivos. Dada la falta de especificación de la comparativa de entornos, se realizará una prueba por medio de varios dispositivos comprobando la eficacia del entorno ya seleccionado. Los dispositivos testados han sido un diodo LED, un botón, un potenciómetro, el módulo PmodLED y la cámara.

#### A2.1.1. Cámara Pcam 5C

Esta cámara de Digilent está diseñada para las FPGA de la misma marca, dispone de una lente de 5 Megapíxeles, además de permitir la transferencia de alta calidad al dispositivo. La conexión con la placa es a través de un cable FFC, que ya viene incluido con la cámara. En la figura inferior (Figura 60), se puede ver el aspecto del dispositivo y el puerto al cual se conectará con la placa.



Figura 60: Imagen del módulo Pcam 5C de Digilent

Al tratarse de un dispositivo complejo con varias capas de abstracción, en el sentido del diseño en la placa FPGA, se ha utilizado el proyecto de prueba de la propia Digilent en GitHub para hacer funcionar la propia cámara [42]. Como se ha indicado previamente, el entorno usado es Vitis 2020.1, aunque este dispositivo se probó en varias versiones de programa e incluso Xilinx SDK. Este proyecto fue diseñado en su última versión publicada para Vivado 2019.1, es decir, para Xilinx SDK.

Dado que Vivado permite una retrocompatibilidad, y ésta es una característica que debido a que se actualiza aproximadamente cada medio año, se da por entendida que no requerirá crear un proyecto nuevo desde 0. La última versión, a principios de octubre de 2020, es 2020.1. Por tanto, el primer intento por hacer funcionar la cámara es en esta última versión, el inconveniente, que tras buscar información, se ha

## Anexo II - Dispositivos testados

detectado un error en la importación de algunos proyectos y su actualización, que no permite la compilación en Vitis 2020.1. Esto únicamente ocurrirá, si el proyecto es de una versión anterior, es necesario actualizar los bloques IP y se utiliza Vitis 2020.1 para ello. El motivo de este error que ocurre durante la compilación en Vitis, se debe a una inconsistencia en la creación del *hardware* en Vivado, con esas características. Lamentablemente, se cumplieron todas ellas durante la prueba de la cámara y no se permitía su compilación. Una de las soluciones sería hacer el proyecto desde 0, lo cual implica una codificación de algún bloque IP específico y tampoco se garantizaría su funcionamiento.

La otra solución, es la tomada finalmente, utilizar una versión anterior, en este caso Vitis 2019.2, con su versión de Vivado correspondiente. Antes de probar esta versión, se pensó que lo apropiado sería asegurarse que la cámara funciona en la versión 2019.1, es decir, SDSoc. Es motivo de esto se debe a la necesidad de cambiar la placa en la que se prueba, siendo un posible fallo para resolver en la versión actual de Vitis. Al ser ambos para la misma versión, no se requerirá de la actualización de los bloques IP, pero sí de la placa a usar.

Como era de esperar, la cámara funciona según el ejemplo de Digilent, aunque el primer uso de la cámara tenía algún error en los colores y el contraste, que se resolvió automáticamente volviendo a programar la FPGA con el código. Por tanto, se dispone de la garantía de funcionamiento en SDSoc 2019.1. Dado que, esta versión se encuentra en estado “descatalogado”, o al menos no recibe soporte oficial, se insinuó el uso de Vitis 2019.2 que parece una versión más estable de Vitis, es cierto que la versión actualizada dispondría de varias mejoras, o en cierto modo, resolvería los errores de la versión anterior. Pese a esos detalles, el error con respecto a programas actualizados a la última versión y su problema de compilación, la versión de Vitis 2019.2 es una versión que debería ser probada.

El procedimiento es el mismo que se realizaría con cualquier actualización de proyectos y bloques IP en Vivado, que se traduce en varios *clicks* para actualizar todo. Tras el cambio y la necesidad de exportar el *hardware* por medio de Vivado, en Vitis, no se requirió de ningún cambio, reforzando que los idea de que códigos de Xilinx SDK y Vitis pueden ser los mismos. En esta ocasión, el resultado fue favorable, y se puede ver un correcto uso de la cámara por medio de la FPGA, HDMI y un control por terminal. En este caso, y en otros muchos, el programa usado es TeraTerm [43], con el cual el autor de esta memoria tiene familiaridad por otros proyectos ya realizados, además de ser un software indicado por muchos miembros de la comunidad de Xilinx.

A modo de demostración de la cámara se puede ver en la imagen inferior su uso (Figura 61), que requiere de una pantalla HDMI, además del ordenador para el terminal.



Figura 61: Prueba de la Pcam.

El resultado de esta prueba indica que Vitis 2020.1 no es una versión tan estable de momento, y dispone de ciertos errores que pueden retrasar otros proyectos inesperadamente. Aunque en un proyecto donde no se utilice la cámara o se tenga conocimiento para su diseño en VHDL, pueda ser adecuado, la situación de encontrar un error de este tipo puede replantear el proyecto por completo. Se ha preferido realizar el proyecto con la alternativa de usar Vitis 2019.2 para evitar futuros errores, pero la interfaz de Vivado 2019.2 en comparación con 2019.1 es la misma, y no supone ningún cambio a simple vista. Únicamente, la idea de un software unificado por parte de Xilinx.

### A2.1.2. Dispositivos de Entrada/Salida

Otro de la idea inicial del proyecto, y la principal, es el uso de dispositivos de entrada/salida: botones, diodos LED, potenciómetros, etc... Para ello, la placa dispone de puertos Pmod, en concreto 5. De estos, hay 3 que permiten una entrada salida digital, dentro de estos hay 2 que permiten alta velocidad según las necesidades. También se dispone de un puerto Pmod con funciones de entrada o salida analógica. Por último también se incluye un Pmod, no accesible de manera natural, que se conecta directamente al procesador y tiene funciones SPI e I2C, entre otras.

En primer lugar se prueba el módulo PmodLED, se trata de un dispositivo de Digilent que ofrece 4 leds monocromos rojos en un módulo como el que se puede ver en la figura inferior (Figura 62). Tiene 6 pines, los mismos que una fila de puertos Pmod, de ellos, hay 2 que son la potencia de entrada (3.3V) y otro que es “toma a tierra” o “Ground”. Restando 4 pines, uno para cada led.

## Anexo II - Dispositivos testados



Figura 62: Fotografía módulo PmodLED

Entre los muchos bloques IP ofrecidos por Digilent, muchos de ellos toman nombre del módulo en sí. En el caso de los dispositivos de entrada salida, como de este módulo, utilizan un bloque separado llamado PmodGPIO. Como indica su nombre, es para dispositivos de entrada/salida. El código o diseño que requiere para un uso básico es el que se puede ver en la siguiente figura (Figura 63). Usando el procedimiento estándar para configurar bloques IP automáticamente, se cumple con el objetivo deseado.

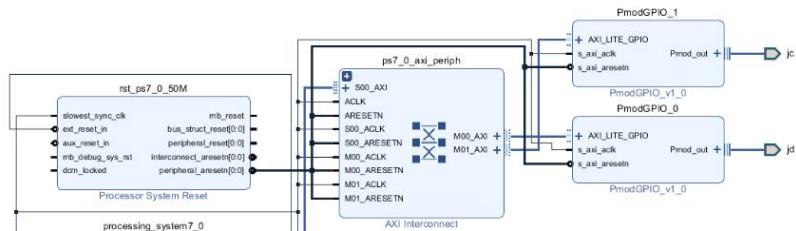


Figura 63: Captura del diseño IP para probar GPIO y PmodLED

Ya en Vitis, la propia plataforma o *hardware* te ofrece en algunas ocasiones códigos de prueba, que se han utilizado en este caso. Cómo es un bloque que admite tanto entrada como salida, tienen un parecido con los códigos de prueba para otros componentes que se aplicaran en el proyecto.

```
#include "PmodGPIO.h"
#include "xil_cache.h"
#include "xil_printf.h"

PmodGPIO myDevice;

void DemoInitialize();
void DemoRun();
void DemoCleanup();
void EnableCaches();
void DisableCaches();

int main() {
    DemoInitialize();
    DemoRun();
    DemoCleanup();
    return 0;
}

void DemoInitialize() {
    EnableCaches();
    GPIO_begin(&myDevice, XPAR_PMODGPIO_0_AXI_LITE_GPIO_BASEADDR, 0xFF);
}
```

```

void DemoRun() {
    int pin1 = 0;
    int pin2 = 0;
    int pin3 = 0;
    int pin4 = 0;
    int all_pins = 0;

    xil_printf("GPIO Input Demo\n\r");

    while (1) {
        // Read individually
        pin1 = GPIO_getPin(&myDevice, 1);
        pin2 = GPIO_getPin(&myDevice, 2);
        pin3 = GPIO_getPin(&myDevice, 3);
        pin4 = GPIO_getPin(&myDevice, 4);

        // Read all together
        all_pins = GPIO_getPins(&myDevice);

        // Print individual reads and the total read masked to the first four bits
        xil_printf(
            "pin 1: %d    pin 2: %d    pin 3: %d    pin 4: %d    all pins: %x\n\r",
            pin1,
            pin2,
            pin3,
            pin4,
            all_pins & 0x0f
        );
    }
}

void DemoCleanup() {
    DisableCaches();
}

void EnableCaches() {
#define __MICROBLAZE__
#define XPAR_MICROBLAZE_USE_ICACHE
    Xil_ICacheEnable();
#endif
#define XPAR_MICROBLAZE_USE_DCACHE
    Xil_DCacheEnable();
#endif
#endif
}

void DisableCaches() {
#define __MICROBLAZE__
#define XPAR_MICROBLAZE_USE_DCACHE
    Xil_DCacheDisable();
#endif
#define XPAR_MICROBLAZE_USE_ICACHE
    Xil_ICacheDisable();
#endif
#endif
}

```

Tabla 7: Código ejemplo GPIO (input)

```

#include "PmodGPIO.h"
#include "sleep.h"
#include "xil_cache.h"
#include "xil_printf.h"

PmodGPIO myDevice;

void DemoInitialize();
void DemoRun();
void DemoCleanup();
void EnableCaches();

```

## Anexo II - Dispositivos testados

```

void DisableCaches();

int main() {
    DemoInitialize();
    DemoRun();
    DemoCleanup();
    return 0;
}

void DemoInitialize() {
    EnableCaches();
    GPIO_begin(&myDevice, XPAR_PMODGPIO_0_AXI_LITE_GPIO_BASEADDR, 0x00);
}

void DemoRun() {
    u8 count = 0;
    int i = 0;

    xil_printf("GPIO Output Demo\n\r");
    while (1) {
        GPIO_setPin(&myDevice, count, 1);
        if (count >= 8) {
            count = 0;
            for (i = 0; i < 4; i++) {
                GPIO_setPins(&myDevice, 0xFF);
                usleep(300000);
                GPIO_setPins(&myDevice, 0x00);
                usleep(300000);
            }
        } else {
            count++;
        }
        usleep(200000);
    }
}

void DemoCleanup() {
    DisableCaches();
}

void EnableCaches() {
#ifdef __MICROBLAZE__
#ifdef XPAR_MICROBLAZE_USE_ICACHE
    Xil_ICacheEnable();
#endif
#ifdef XPAR_MICROBLAZE_USE_DCACHE
    Xil_DCacheEnable();
#endif
#endif
}

void DisableCaches() {
#ifdef __MICROBLAZE__
#ifdef XPAR_MICROBLAZE_USE_DCACHE
    Xil_DCacheDisable();
#endif
#ifdef XPAR_MICROBLAZE_USE_ICACHE
    Xil_ICacheDisable();
#endif
#endif
}

```

Tabla 8: Código ejemplo GPIO (output)

Como se ve en las tablas superiores (Tabla 7 y Tabla 8), la librería que se utiliza es “PmodGPIO.h”, entre sus instrucciones están “**GPIO\_getPin(..., ...)**” y “**GPIO\_setPin(..., ...)**”, es necesario indicar, que en algunos Pmod, por un posible error de los bloques IP que utilizan los conectores Pmod, el orden de los pines se ve

alterado. Este error no se ha considerado de carácter crítico, porque cambia el pin al cual se conecta cada pin del Pmod con su adyacente, el cual sabiendo el orden que tiene, no genera ningún problema. Este error también ocurre en otras versiones de Vivado, y se entiende que es un error de los diseños IP del sistema que se han importado.

Para estas situaciones, es aconsejable tener un fichero con las restricciones en los puertos antes de generar el bitstream y no retrasarse en el proyecto. Una manera de hacerlo es usando la función de “***Open Elaborated Design***” y buscar “***I/O Ports***”, con esto saldrá una pestaña con la configuración de cada pin. Conociendo cada uno de ellos y con la guía de referencia de la placa Zybo Z7 [28] que indica el nombre interno de cada pin se puede ver que ha sido el único cuyo orden se ha visto alterado. En la figura inferior (Figura 64) se puede ver el diseño que tiene la pestaña de I/O Ports.

| Name                  | Direction        | Board Part Pin | Board Part Interface | Neg Diff Pair | Package Pin | Fixed | Bank       | I/O Std     | Vcco       | VI |
|-----------------------|------------------|----------------|----------------------|---------------|-------------|-------|------------|-------------|------------|----|
| All ports (146)       |                  |                |                      |               |             |       |            |             |            |    |
| > DDR_54576 (71)      | INOUT            |                |                      |               |             | ✓     | 502        | (Multiple)* | 1.350      | {  |
| > FIXED_IO_54576 (59) | INOUT            |                |                      |               |             | ✓     | (Multiple) | (Multiple)* | (Multiple) | {  |
| > jd_54576 (0)        | INOUT            |                |                      |               |             | ✓     | 34         | LVCMOS33*   | 3.300      |    |
| > jd_54576 (0)        | Scalar ports (0) |                |                      |               |             |       |            |             |            |    |
| > jd_54576 (0)        | INOUT            |                |                      |               |             | ✓     | 34         | LVCMOS33*   | 3.300      |    |
| > jd_54576 (0)        | Scalar ports (0) |                |                      |               |             |       |            |             |            |    |
| > jd_pin1_io          | INOUT            | JD1            |                      |               | T15         | ✓     | 34         | LVCMOS33*   | 3.300      |    |
| > jd_pin2_io          | INOUT            | JD2            |                      |               | T14         | ✓     | 34         | LVCMOS33*   | 3.300      |    |
| > jd_pin3_io          | INOUT            | JD3            |                      |               | R14         | ✓     | 34         | LVCMOS33*   | 3.300      |    |
| > jd_pin4_io          | INOUT            | JD4            |                      |               | P14         | ✓     | 34         | LVCMOS33*   | 3.300      |    |
| > jd_pin7_io          | INOUT            | JD7            |                      |               | U15         | ✓     | 34         | LVCMOS33*   | 3.300      |    |
| > jd_pin8_io          | INOUT            | JD8            |                      |               | U14         | ✓     | 34         | LVCMOS33*   | 3.300      |    |
| > jd_pin9_io          | INOUT            | JD9            |                      |               | V18         | ✓     | 34         | LVCMOS33*   | 3.300      |    |
| > jd_pin10_io         | INOUT            | JD10           |                      |               | V17         | ✓     | 34         | LVCMOS33*   | 3.300      |    |
| Scalar ports (0)      |                  |                |                      |               |             |       |            |             |            |    |

Figura 64: Pestaña I/O Ports en “Elaborated Design”

Retomando la prueba del módulo PmodGPIO, el código por defecto de prueba activa gradualmente cada pin para finalizar con un parpadeo de todos los pines. De esta manera, se disponen de todas las funciones necesarias para leer los valores y encender todos pines correctamente. Tras probar el módulo comentado, no sé detectó ningún error, al tratarse de 4 diodos LED, fue necesario reducir el bucle realizado en el código de prueba. Posteriormente, se realizó un código adaptado para usar sólo un pin, usando las instrucciones ya indicadas, disponible en la tabla inferior (Tabla 9).

```
#include "PmodGPIO.h"
#include "sleep.h"
#include "xil_cache.h"
#include "xil_printf.h"

PmodGPIO myDevice;

void DemoInitialize();
void DemoRun();
void DemoCleanup();
void EnableCaches();
void DisableCaches();

int main() {
    DemoInitialize();
    DemoRun();
    DemoCleanup();
    return 0;
}

void DemoInitialize() {
```

## Anexo II - Dispositivos testados

```
EnableCaches();
GPIO_begin(&myDevice, XPAR_PMODGPIO_0_AXI_LITE_GPIO_BASEADDR, 0x00);
}

void DemoRun() {
    xil_printf("GPIO Output Demo\n\r");
    while (1) {
        GPIO_setPin(&myDevice, 2, 1);
        usleep(300000);
        GPIO_setPin(&myDevice, 2, 0);
        usleep(300000);
    }
}

void DemoCleanup() {
    DisableCaches();
}

void EnableCaches() {
#ifndef __MICROBLAZE__
#ifndef XPAR_MICROBLAZE_USE_ICACHE
    Xil_ICacheEnable();
#endif
#endif
#ifndef XPAR_MICROBLAZE_USE_DCACHE
    Xil_DCacheEnable();
#endif
#endif
}

void DisableCaches() {
#ifndef __MICROBLAZE__
#ifndef XPAR_MICROBLAZE_USE_DCACHE
    Xil_DCacheDisable();
#endif
#endif
#ifndef XPAR_MICROBLAZE_USE_ICACHE
    Xil_ICacheDisable();
#endif
#endif
}
```

Tabla 9: Código utilizado en la prueba de GPIO (output)

Otro de los dispositivos E/S que se quería probar es un botón y diodo LED, comunes en cualquier kit de electrónica. Para ello, se utilizó una *breadboard*, para conectar los cables y componentes a ésta y a la placa. El objetivo de estos dispositivos es probar el funcionamiento de otros Pmod, junto con su posibilidad de permitir la entrada y salida en el mismo puerto Pmod de la placa y la conexión a la *breadboard* directamente. Para ello, los dispositivos utilizados son los que se pueden ver en la siguiente figura (Figura 65).

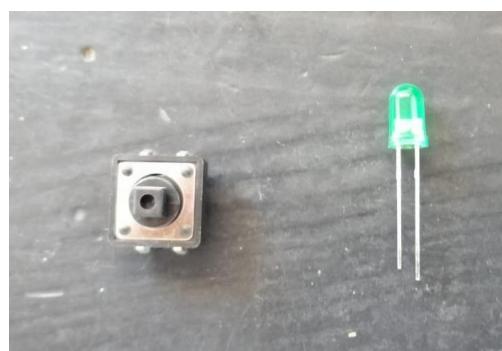


Figura 65: Dispositivos GPIO adicionales. Botón y diodo LED verde

Para probar el funcionamiento de todos los Pmod, se ha creado un diseño con varios de los módulos PmodGPIO conectados a cada puerto Pmod. De esta manera, tras una prueba sencilla, se detectó el error de los pines en el puerto PmodJD ya comentado. El diseño del GPIO utilizado se puede ver en una Figura anterior (Figura 63).

Por medio de esta conexión, se ha probado la compatibilidad entre varios puertos Pmod y también entre el funcionamiento tanto de entrada como salida de un mismo puerto. El código utilizado para el segundo objetivo es el que se puede ver en la siguiente tabla (Tabla 10).

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "PmodGPIO.h"
#include "sleep.h"
#include "xil_cache.h"
#include "xil_io.h"
#include "xil_types.h"
#include "xparameters.h"

PmodGPIO myDeviceI;
PmodGPIO myDeviceO;

void DemoInitialize();
void DemoRun();
void DemoCleanup();
void EnableCaches();
void DisableCaches();

int main() {
    DemoInitialize();
    DemoRun();
    DemoCleanup();
    return 0;
}

void DemoInitialize() {
    GPIO_begin(&myDeviceI, XPAR_PMODGPIO_0_AXI_LITE_GPIO_BASEADDR, 0x0F);
    GPIO_begin(&myDeviceO, XPAR_PMODGPIO_1_AXI_LITE_GPIO_BASEADDR, 0x0F);
}

void DemoRun() {
    int pin1 = 0;

    while (1) {
        pin1 = GPIO_getPin(&myDeviceI, 1);

        xil_printf("pin 1: %d\r\n", pin1);

        if(pin1 == 1){
            GPIO_setPin(&myDeviceI, 7, 1);
            usleep(300000);
            GPIO_setPin(&myDeviceI, 7, 0);
            usleep(300000);
        }

        usleep(300000);
    }
}

void DemoCleanup() {
    DisableCaches();
}

void EnableCaches() {
#ifndef __MICROBLAZE__

```

```

#define XPAR_MICROBLAZE_USE_ICACHE
    Xil_ICacheEnable();
#endif
#define XPAR_MICROBLAZE_USE_DCACHE
    Xil_DCacheEnable();
#endif
#endif
}

void DisableCaches() {
#define __MICROBLAZE__
#define XPAR_MICROBLAZE_USE_DCACHE
    Xil_DCacheDisable();
#endif
#define XPAR_MICROBLAZE_USE_ICACHE
    Xil_ICacheDisable();
#endif
#endif
}

```

Tabla 10: Código utilizado para GPIO (input y Output)

En la función **DemoInitialize()**, se especifica el sentido de cada uno de los pines o puertos del Pmod elegido. En este caso, y debido a la primera comprobación entre varios Pmod, se han creado instancias de PmodGPIO para cada puerto Pmod. Se debe indicar en cada puerto, el sentido, en este caso **0x0F**, indica que los pines superiores (pin1...pin4) de Gpio\_0, el puerto PmodJD son de entrada, y los pines inferiores (pin5...pin8) son de salida. La ejecución del código realizará una lectura del pin 1 del puerto PmodJD, y si éste está activo, realizará un parpadeo con el led conectado al pin 7 del mismo puerto.

Por último, también se ha probado el puerto PmodJA, que es utilizado para entrada analógica y conectar un potenciómetro, el potenciómetro utilizado para este propósito es un dispositivo sencillo con 3 pines, mostrado en la siguiente figura (Figura 66).

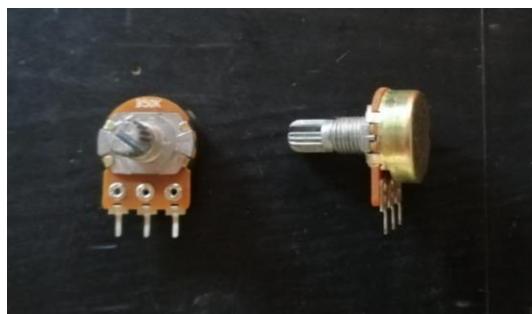


Figura 66: Imagen de potenciómetros (3 pines)

Para conseguir el objetivo de usar un potenciómetro, el módulo que se utilizará es XADC, como este módulo viene por defecto en la configuración de Vivado, no utiliza las librerías de Digilent importadas. En el ejemplo seguido, se probarán los diferentes canales que tiene el módulo para garantizar su funcionamiento. Para ello, se ha creado el siguiente diseño en Vivado 2019.2 (Figura 67), en éste se utilizan un único módulo “XADC Wizard”, pero se prefiere importar cada input de manera individual para evitar cualquier asignación automática errónea.

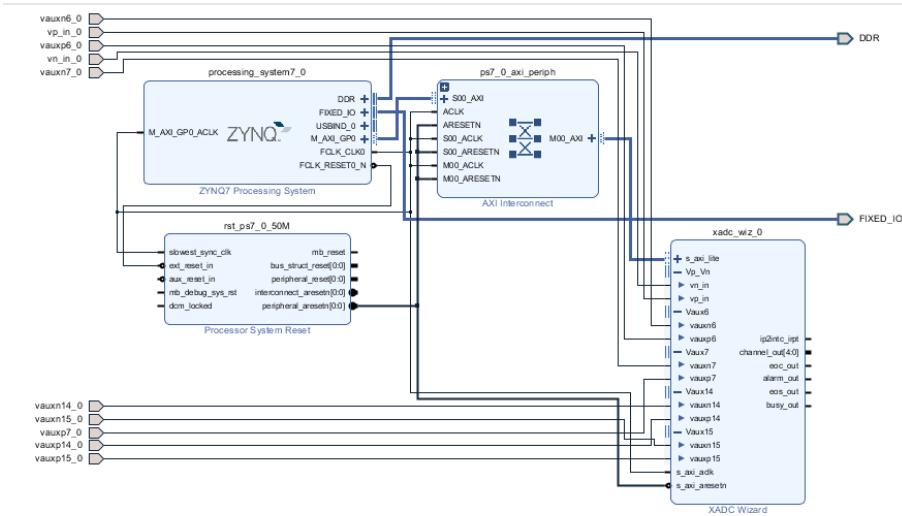


Figura 67: Captura diseño IP usando XADC Wizard

El módulo “XADC Wizard”, permite la conexión analógica de dispositivos, como en este caso no es necesario un valor de salida. Se desactivarán varios valores para que el sistema tenga el propósito deseado, se debe modificar el módulo haciendo doble click en él.

Los cambios requeridos son utilizar la interfaz de AXI4, que ya se utiliza con otros dispositivos, también se le aplicará un funcionamiento continuo, para realizar lecturas continuamente. Se aplicará la configuración de Secuenciador de Canales (“Channel Sequencer”) porque se quiere probar todas las entradas, esto es visible en la figura A (Figuras 68a).

El promedio utilizado será 16, y se desactivará la calibración automática de ADC, indicando que tampoco se quiere la ganancia. Esto se pondrá de manera manual en cada canal, se indica en la figura B (Figuras 68b).

Todas las alarmas se desactivarán, para evitar interrupciones durante la lectura, esto es visible en la figura C (Figuras 68c). Por último, se añadirán los canales usados, en este caso, estos vienen indicados en la propia placa junto al puerto y tienen la configuración AD14, AD7, AD15 y AD6 respectivamente. Cada uno está formado por una pareja de pines P14 y N14, en el primer par, y de manera análoga en el resto. También se marcará la calibración de cada uno de los pares, esto se puede ver en la figura D (Figuras 68d).

## Anexo II - Dispositivos testados

a)

b)

c)

d)

Figuras 68a, 68b, 68c y 68d: Capturas configuración XADC Wizard; a) Conf. Básica [AXI4Lite, Continuo, Secuenciador Canales]; b) Conf. ADC [Media a 16, desactivar Calibración]; c) Conf. Alarmas [Desactivar]; d) Conf. Canales [Añadir los indicados]

Tras esto se dispondrá de un bloque adecuado para realizar esta prueba, y se añadirán todos los puertos necesarios al propio módulo. Se marcan todas las entradas de los canales creados y con el botón derecho del ratón, la opción “**Make External**” esto generará un puerto nuevo por cada pin del módulo, y un quinto valor con la potencia interna.

Para asignar los pines se crearán las restricciones manualmente, incorporando la funcionalidad de estas para activar/desactivar los componentes de la placa disponibles. Para ello, se comenzará incluyendo el fichero con *constraints* en formato “.xdc” al proyecto. Este fichero viene disponible en el propio GitHub de Digilent para la placa [44]. Nos interesa en este caso usar el puerto XADC, para ello se busca el puerto y se quita el comentario, modificando la sección con los nombres asignados por Vivado. Este fragmento modificado se cree conveniente añadirlo en la siguiente Tabla (Tabla 11).

```
[...]
##Pmod Header JA (XADC)
set_property -dict { PACKAGE_PIN N16      IOSTANDARD LVCMOS33 } [get_ports { vauxn14_0 }];
#IO_L21N_T3_DQS_AD14N_35 Sch=JA1_R_N    IOSTANDARD LVCMOS33 } [get_ports { vauxp14_0 }];
set_property -dict { PACKAGE_PIN N15      IOSTANDARD LVCMOS33 } [get_ports { vauxn14_0 }];
#IO_L21P_T3_DQS_AD14P_35 Sch=JA1_R_p    IOSTANDARD LVCMOS33 } [get_ports { vauxp14_0 }];
set_property -dict { PACKAGE_PIN L15      IOSTANDARD LVCMOS33 } [get_ports { vauxn7_0 }];
```

```
#IO_L22N_T3_AD7N_35 Sch=JA2_R_N
set_property -dict { PACKAGE_PIN L14 } IO_STANDARD LVCMOS33 } [get_ports { vauxp7_0 }];
#IO_L22P_T3_AD7P_35 Sch=JA2_R_P

set_property -dict { PACKAGE_PIN J16 } IO_STANDARD LVCMOS33 } [get_ports { vauxn15_0 }];
#IO_L24N_T3_AD15N_35 Sch=JA3_R_N
set_property -dict { PACKAGE_PIN K16 } IO_STANDARD LVCMOS33 } [get_ports { vauxp15_0 }];

set_property -dict { PACKAGE_PIN J14 } IO_STANDARD LVCMOS33 } [get_ports { vauxn6_0 }];
#IO_L20N_T3_AD6N_35 Sch=JA4_R_N
set_property -dict { PACKAGE_PIN K14 } IO_STANDARD LVCMOS33 } [get_ports { vauxp6_0 }];
#IO_L20P_T3_AD6P_35 Sch=JA4_R_P

[...]
```

Tabla 11: Fragmento respecto al PmodJA(XADC) en fichero Constraints.xdc

A continuación, se creará el “*HDL Wrapper*” del diseño, y para evitar problemas con el diseño posteriormente, se selecciona la opción “**Generate Block Design**” del menú izquierdo, bajo “**IP Integrator**”. Usando la opción “**Out of context per-IP**” para crear un diseño que use las restricciones aportadas, esta opción se puede ver en la siguiente figura (Figura 69). En adelante, la configuración de síntesis e implementación es la misma que en cualquier proyecto similar, ya explicada anteriormente.

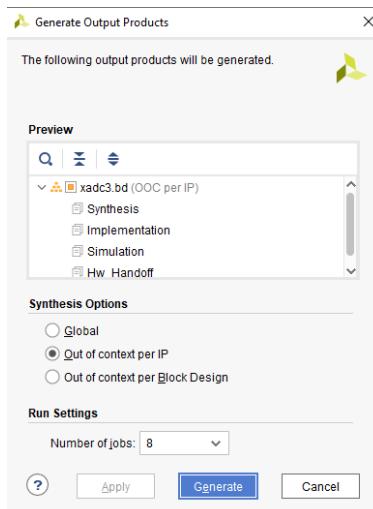


Figura 69: Interfaz de “Generate Block Design”

El siguiente paso, sería crear el fichero bitstream y volver a Vitis para programar la placa FPGA con el código deseado para probar el potenciómetro. El código utilizado usa el módulo “*xsysmon.h*”, que permite la lectura de los valores analógicos en la placa. El código utilizado es el disponible en la siguiente tabla (Tabla 12).

```
#include "xsysmon.h"
#include "xparameters.h"
#include "xstatus.h"
#include "xil_exception.h"
#include "xil_printf.h"
#include "sleep.h"

#define C_BASEADDR 0x43C00000

int main()
{
    u16 data = 1;
    u16 data6 = 0;
```

## Anexo II - Dispositivos testados

```

u16 data7 = 0;
u16 data14 = 0;
u16 data15 = 0;

while(1)
{
    data6 = Xil_In32(C_BASEADDR + 0x258);
    data7 = Xil_In32(C_BASEADDR + 0x25C);
    data14 = Xil_In32(C_BASEADDR + 0x278);
    data15 = Xil_In32(C_BASEADDR + 0x27C);
    data = Xil_In32(C_BASEADDR + 0x20C);

    data = (data*0.0152588);
    data6 = (data6*0.0152588);
    data7 = (data7*0.0152588);
    data14 = (data14*0.0152588);
    data15 = (data15*0.0152588);

    xil_printf("Input data = %03dmv\n\r",data);
    xil_printf("data6 = %03dmv\n\r",data6);
    xil_printf("data7 = %03dmv\n\r",data7);
    xil_printf("data14 = %03dmv\n\r",data14);
    xil_printf("data15 = %03dmv\n\r\n\r",data15);

    usleep(300000);
}
return 0;
}

```

Tabla 12: Código utilizado para probar el módulo XADC y potenciómetro

La función **Xil\_In32(...)**, permite realizar una lectura del valor que se lee del puerto analógico, para ello usa directamente la dirección del puerto para realizar la lectura de cada par de puertos. La dirección base “**C\_BASEADDR**” viene indicada en las direcciones del propio Vivado, en este caso es **0x43C00000**. Para conocer a qué dirección se debe apuntar, se ha tomado en cuenta el manual del módulo XADC Wizard, disponible en la **Tabla 2-4** del siguiente documento PDF [45].

Los datos requieren una reducción a decimal, debido a que aunque los puertos sólo admiten entrada de 1.1 V, debido a que se almacena un número demasiado alto para conocer el voltaje exacto, éste debe ser reducido. Por ello se multiplica por 0.0152588 ( $1/2^{16}$ ) para tenerlo en milivoltios (mV). Finalmente, se realiza una pausa de 300000 us, para permitir una lectura más pausada de los datos.

Debido a que el potenciómetro y la placa funciona con un voltaje de 3.3V, aunque la placa disponga de un sistema que no permite un voltaje mayor a 0.942 V. Se ha decidido aplicar un divisor de potencia para alcanzar ese valor, el circuito usado es el mostrado en la siguiente figura (Figura 70).

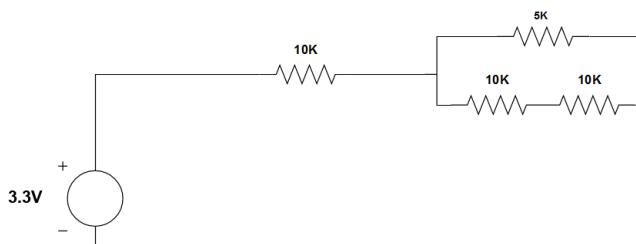
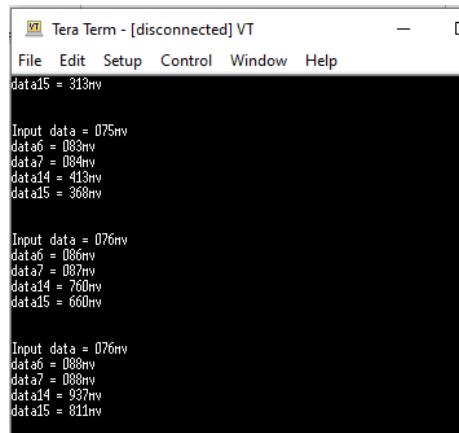


Figura 70: Diseño del circuito divisor de voltaje a 0.942V

Tras una prueba del potenciómetro se observa el siguiente resultado, visible en la siguiente figura (Figura 71). En él se puede ver que el valor de éste es 0.937 V, cercano al valor total de 0.942 V. Este error se puede deber a que se utiliza una media entre 16 tomas que puede crear este valor reducido. Cómo se ha comentado anteriormente, se trata de pares de pines, por lo que requiere que el pin inferior se mantenga a 0 V para tener un valor adecuado.



The screenshot shows a terminal window titled "Tera Term - [disconnected] VT". The menu bar includes File, Edit, Setup, Control, Window, and Help. The main window displays three sets of analog input data:

| Input  | data15 | data14 | data13 | data12 | data11 | data10 | data9 | data8 | data7 | data6 | data5 | data4 | data3 | data2 | data1 | data0 |
|--------|--------|--------|--------|--------|--------|--------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| data15 | 313mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |
| Input  | 075mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |
| data16 | 083mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |
| data17 | 084mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |
| data14 | 413mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |
| data15 | 368mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |
| Input  | 076mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |
| data16 | 086mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |
| data17 | 087mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |
| data14 | 760mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |
| data15 | 660mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |
| Input  | 076mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |
| data16 | 088mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |
| data17 | 088mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |
| data14 | 937mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |
| data15 | 811mV  |        |        |        |        |        |       |       |       |       |       |       |       |       |       |       |

Figura 71: Salida analógica del terminal TeraTerm (Data14)

## A3. Anexo III : Códigos del sistema

### A3.1. Tarea Lector del Teclado

#### A3.1.1. Definiciones

```
//KeyPad & System Defines (P.0)
#define DEFAULT_KEYTABLE "0FED789C456B123A"
char code[6] = " ";

XGpio options;
u8 debug, started;

PmodGPIO myDeviceJD_io, myDeviceJE_o;
PmodKYPD myDeviceKP;

void setupKP();
void setupBoard();
void setupGPIO();
```

#### A3.1.2. Código “keyPadTask”

```
static void keyPadTask( void *pvParameters ){
    TickType_t xLastTimeAwake;
    u16 keystate;
    XStatus status, last_status = KYPD_NO_KEY;
    u8 key, last_key = 'x';
    char codeKP[6];
    strncpy(codeKP, " ",6);
    xLastTimeAwake = 0;

    int kp = 0;
    setupKP();

    for(;;){
        keystate = KYPD_getKeyStates(&myDeviceKP);
        status = KYPD_getKeyPressed(&myDeviceKP, keystate, &key);

        // Print keys detect if a new key is pressed or if status has changed
        if (status == KYPD_SINGLE_KEY && (status != last_status || key != last_key)) {
            sprintf(codeKP + kp, "%c", (char) key);
            xil_printf("Key Pressed: %s\r\n", codeKP);
            kp++;
            last_key = key;
        }
        else if (status == KYPD_MULTI_KEY && status != last_status)
            xil_printf("Error: Multiple keys pressed\r\n");

        if(kp == 6){
            xSemaphoreTake(xSemaphoreKB, portMAX_DELAY);
            strcpy(code, codeKP);
            xSemaphoreGive(xSemaphoreKB);
            kp = 0;
            strncpy(codeKP, " ",6);
        }
        last_status = status;
        vTaskDelayUntil(&xLastTimeAwake, 200/portTICK_RATE_MS);
    }
}
```

### A3.1.3. Código “setupKP()”, “setupBoard()” y “setupGPIO()”

```

void setupKP(){
    strcpy(code, " ");
    KYPD_begin(&myDeviceKP, XPAR_PMODKYPD_0_AXI_LITE_GPIO_BASEADDR);
    KYPD_loadKeyTable(&myDeviceKP, (u8* ) DEFAULT_KEYTABLE);
    Xil_Out32(myDeviceKP.GPIO_addr, 0xF);
}

void setupBoard(){
    XGpio_Initialize(&options, 0);
    XGpio_SetDataDirection(&options, 2, 0xF);
    XGpio_SetDataDirection(&options, 1, 0x0);
}

void setupGPIO(){
    GPIO_begin(&myDeviceJD_io, XPAR_PMODGPIO_1_AXI_LITE_GPIO_BASEADDR, 0x0F);
    GPIO_begin(&myDeviceJE_o, XPAR_PMODGPIO_2_AXI_LITE_GPIO_BASEADDR, 0x00);
}

```

## A3.2. Tarea Utilizar Sistema

### A3.2.1. Código “useSystemTask”

```

static void useSystemTask( void *pvParameters ){
    TickType_t xLastTimeAwake;
    xLastTimeAwake = 0;
    char message[128];
    message[0] = '\0';
    char codeSys[6];

    for( ; ; ){
        if(xSemaphoreTake( xSemaphoreKB, ( TickType_t ) 30) == pdTRUE){
            strcpy(codeSys, code);
            strcpy(code, " ");
            xSemaphoreGive( xSemaphoreKB );
        }else{
            strcpy(codeSys, " ");
        }

        if(strcmp(codeSys,"C1234A") == 0){
            started = 1;
            sprintf(message, "\r\nCode: %s\r\n- System Working\r\n",codeSys);
            printf("%" ,message);
            xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
            if(SDFileWrite(message) != XST_SUCCESS){
                printf("Can't access SD to start!\r\n");
            }
            xSemaphoreGive(xSemaphoreSD);
            strcpy(codeSys," ");
        }else if(strcmp(codeSys,"D1234A") == 0){
            started = 0;
            rgbLedWork(0);
            pmodLedWork(0);
            sprintf(message, "\r\nCode: %s\r\n- System Deactivated\r\n",codeSys);
            printf("%" ,message);
            xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
            if(SDFileWrite(message) != XST_SUCCESS){
                printf("Can't access SD!\r\n");
            }
            xSemaphoreGive(xSemaphoreSD);
        }
    }
}

```

### Anexo III - Tarea Utilizar Sistema

```

strcpy(codeSys, " ");
}else if(strcmp(codeSys, "F0001A") == 0){
    debug = 1;
    sprintf(message, "\r\nCode: %s\r\n- Debug Mode: ON\r\n", codeSys);
    printf("%s", message);
    xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
    if(SDFileWrite(message) != XST_SUCCESS){
        printf("Can't access SD!\r\n");
    }
    xSemaphoreGive(xSemaphoreSD);
    strcpy(codeSys, " ");
}else if(strcmp(codeSys, "F0000A") == 0){
    debug = 0;
    sprintf(message, "\r\nCode: %s\r\n- Debug Mode: OFF\r\n", codeSys);
    printf("%s", message);
    xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
    if(SDFileWrite(message) != XST_SUCCESS){
        printf("Can't access SD to start!\r\n");
    }
    xSemaphoreGive(xSemaphoreSD);
    strcpy(codeSys, " ");
}else if(strcmp(codeSys, "")){
    if((codeSys[0] == 'B') && (codeSys[5] == 'A')){
        maxAltitude = (((int) (codeSys[1]-48))*1000 +
            ((int) (codeSys[2]-48))*100 +
            ((int) (codeSys[3]-48))*10 +
            ((int) (codeSys[4]-48)));
        minAltitude = (maxAltitude*10000)/12000;
        realValueRatio = 12000/maxAltitude;
        sprintf(message, "\r\nCode: %s\r\n- New Min Altitude: %d\r\n- New Max Altitude:
%d\r\n- New Ratio Conversion to Real: %d\r\n", codeSys, minAltitude, maxAltitude,
realValueRatio);
        printf("%s", message);
        xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
        if(SDFileWrite(message) != XST_SUCCESS){
            printf("Can't access SD to start!\r\n");
        }
        xSemaphoreGive(xSemaphoreSD);
        strcpy(codeSys, " ");
    }else if((codeSys[0] == 'E') && (codeSys[5] == 'E')){
        if(started == 0){
            xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
            sprintf(fileName,"Record_%c%c%c%c.txt", codeSys[1], codeSys[2], codeSys[3],
codeSys[4]);
            sprintf(message, "\r\nCode: %s\r\nChanged FileName to %s\r\n", codeSys,
fileName);
            printf("%s", message);
            if(SDFileWrite(message) != XST_SUCCESS){
                printf("Can't access SD to start!\r\n");
            }
            xSemaphoreGive(xSemaphoreSD);
        }else{
            printf("System is started. Can't change FileName right now!\r\n");
        }
        strcpy(codeSys, " ");
    }else{
        sprintf(message, "\r\nCode: %s\r\n- Wrong Code!\r\n", codeSys);
        printf("%s", message);
        xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
        if(SDFileWrite(message) != XST_SUCCESS){
            printf("Can't access SD to start!\r\n");
        }
        xSemaphoreGive(xSemaphoreSD);
        strcpy(codeSys, " ");
    }
}
if(started == 1){
    XGpio_DiscreteWrite(&options, 1, 1);
    usleep(10000);
    XGpio_DiscreteWrite(&options, 1, 0);
}
vTaskDelayUntil(&xLastTimeAwake, 2000/portTICK_RATE_MS);
}
}

```

## A3.3. Tarea Acelerómetro

### A3.3.1. Definiciones

```
//IIC Defines & Declarations //Accelerometer
#define IIC_DEVICE XPAR_AXI_IIC_0_DEVICE_ID
#define MPU 0x68
//Buffers
u8 bufferSend[128];
u8 bufferRecv[128];
int ByteCount;
int StatusIIC;
//Config
XIic_Config *ConfigPtr;
XIic XiicInstance;
//Conversion Rates
#define A_R 16384.0 // 32768/2
//Conversion Rad to Degrees 180/PI
#define RAD_A_DEG 57.295779
//RAW Data
int16_t AcX, AcY, AcZ;
//Accelerometer Protected Object (P.O)
//Angles
float Acc[2];
int specRiskTilt = 0;
char alertTilt[32] = " ";
int countCorrectReadingsTilt = 0;
int AxisX = 0;
int AxisY = 0;

void setup();
void readAccelerometer();
```

### A3.3.2. Tarea “accelerometerTask”

```
static void accelerometerTask( void *pvParameters ){
    TickType_t xLastTimeAwake;
    xLastTimeAwake = 0;

    setup();

    for( ;; ){
        if(started == 1){
            xSemaphoreTake( xSemaphoreTilt, portMAX_DELAY );
            readAccelerometer();
            xSemaphoreGive( xSemaphoreTilt );
        }
        vTaskDelayUntil(&xLastTimeAwake, 250/portTICK_RATE_MS);
    }
}
```

### A3.3.3. Función “readAccelerometer”

```
void readAccelerometer(){
    float AuxAcc[2] = {0.0, 0.0};

    for(int i=0; i<3; i++){
        bufferSend[0]= 0x3B;
        ByteCount = XIic_Send(XPAR_AXI_IIC_0_BASEADDR, MPU, bufferSend, 1, XIIC_STOP);
        if(ByteCount == 0){
            printf("Error: GY-521!\r\n");
        }
    }
}
```

### Anexo III - Tarea Acelerómetro

```

xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
    if(SDFileWrite("Error: GY-521!\r\n") != XST_SUCCESS){
        printf("Can't access SD - Accelerometer!\r\n");
    }
xSemaphoreGive(xSemaphoreSD);
break;
}
ByteCount = XIic_Recv(XPAR_AXI_IIC_0_BASEADDR, MPU, bufferRecv, 6, XIIC_STOP);

AcX = bufferRecv[0]<<8|bufferRecv[1];
AcY = bufferRecv[2]<<8|bufferRecv[3];
AcZ = bufferRecv[4]<<8|bufferRecv[5];

AuxAcc[0] += atan((-1)*(AcX/A_R)/sqrt(pow((AcY/A_R),2) + pow((AcZ/A_R),2)))*RAD_A_DEG;
AuxAcc[1] += atan((-1)*(AcY/A_R)/sqrt(pow((AcX/A_R),2) + pow((AcZ/A_R),2)))*RAD_A_DEG;
}

Acc[1] = AuxAcc[1]/3;
Acc[0] = AuxAcc[0]/3;

//Check AxisX
if((Acc[0] >= (-30.0)) && (Acc[0] <= (30.0))){
    AxisX = 0;
}else{
    if(AxisX<20){
        AxisX++;
    }
}
//Check AxisY
if((Acc[1] >= (-20.0)) && (Acc[1] <= (20.0))){
    AxisY = 0;
}else{
    if(AxisY<20){
        AxisY++;
    }
}
//Check counters
if(specRiskTilt!=3){
    if((AxisX == 0) &&(AxisY == 0)){ //Within tilt range
        if(countCorrectReadingsTilt<20){
            countCorrectReadingsTilt++;
        }
        if(countCorrectReadingsTilt == 2){
            //specRiskTilt->3 or 0, cannot rectify
            if(specRiskTilt == 2){
                specRiskTilt = 1;
                countCorrectReadingsTilt = 0;
                strcpy(alertTilt, "Rectified Angle");

                xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
                    sprintf(send2uSD, "\r\n[%ld] %s - Level: %d\r\n", (xTaskGetTickCount()*10),
alertTilt, specRiskTilt);
                    if(SDFileWrite(send2uSD) != XST_SUCCESS){
                        printf("Can't access SD - Accelerometer!\r\n");
                    }
                xSemaphoreGive(xSemaphoreSD);

            }
            else if(specRiskTilt == 1){
                specRiskTilt = 0;
                strcpy(alertTilt, " ");
                xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
                    sprintf(send2uSD, "\r\n[%ld] No Risk Angle\r\n", (xTaskGetTickCount()*10));
                    if(SDFileWrite(send2uSD) != XST_SUCCESS){
                        printf("Can't access SD - Accelerometer!\r\n");
                    }
                xSemaphoreGive(xSemaphoreSD);
            }
        }
    }
}
else{
    if(specRiskTilt != 3){
        countCorrectReadingsTilt = 0;
        if((AxisX < 3) || (AxisY < 3)){
            if(AxisX >= 3){
                if(AxisX >= 5){

```



### A3.3.4. Función setup()

```
void setup(){
    ConfigPtr = XIic_LookupConfig(IIC_DEVICE);
    Status = XIic_CfgInitialize(&XiicInstance, ConfigPtr, ConfigPtr->BaseAddress);
    Status = XIic_SetAddress(&XiicInstance, XII_ADDR_TO_SEND_TYPE, MPU);
    Status = XIic_Start(&XiicInstance);

    bufferSend[0]=0x6B;
    bufferSend[1]=0x00;
    ByteCount = XIic_Send(XPAR_AXI_IIC_0_BASEADDR, MPU, bufferSend, 2, XIIC_STOP);
}
```

## A3.4. Tarea Ultrasonidos

### A3.4.1. Definiciones

```
//Ultrasonic (P.O)
int minAltitude = 141;
int maxAltitude = 170;
#define AltitudeRatio (10000/12000) //10/12=0.833333
int realValueRatio = 12000/170; //Variable
int specRiskSpeed = 0;
char alertSpeed[32] = " ";
int countCorrectReadingsAltitude = 0;
int countWrongReadingsAltitude[2] = {0, 0};
int altitudeDB = 0;
void readUltrasonicSensor();
```

### A3.4.2. Tarea “ultrasonicTask”

```
static void ultrasonicTask( void *pvParameters ){
    TickType_t xLastTimeAwake;
    xLastTimeAwake = 0;

    for( ;; ){
        if(started == 1){
            xSemaphoreTake( xSemaphoreAltitude, portMAX_DELAY );
            readUltrasonicSensor();
            xSemaphoreGive( xSemaphoreAltitude );
        }
        vTaskDelayUntil(&xLastTimeAwake, 300/portTICK_RATE_MS);
    }
}
```

### A3.4.3. Función “readUltrasonicSensor”

```
void readUltrasonicSensor(){
    long distance = 0;
    long altitude = 0;
    XTime tStart, tEnd;
    u32 tUsed;

    //Retrieving data
    for(int iUs = 0; iUs < 3; iUs++){
        GPIO_setPin(&myDeviceJD_io, 7, 1);
```

```

usleep(10);
GPIO_setPin(&myDeviceJD_io, 7, 0);

while (GPIO_getPin(&myDeviceJD_io, 3) == 0){
}

XTime_GetTime(&tStart);

while (GPIO_getPin(&myDeviceJD_io, 3) == 1){
}

XTime_GetTime(&tEnd);

tUsed = ((tEnd - tStart)/(COUNTS_PER_SECOND/1000000));
distance = tUsed/59;

if(distance > altitude){
    altitude = distance;
}
usleep(200);
}

altitudeDB = altitude;

//Comparing and detect phase

if(specRiskAltitude!=3){
    if(altitude < minAltitude){ //Altitude under 10000 m (Low Altitude)
        countCorrectReadingsAltitude = 0;
        if(countWrongReadingsAltitude[0] < 20){
            countWrongReadingsAltitude[0]++;
        }
        countWrongReadingsAltitude[1] = 0;
        if(countWrongReadingsAltitude[0] >= 3){
            if(countWrongReadingsAltitude[0] >=6){
                specRiskAltitude = 2;
                xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
                sprintf(send2uSD, "\r\n[%d] %s - Level: %d\r\n", (xTaskGetTickCount()*10),
alertAltitude, specRiskAltitude);
                if(SDFileWrite(send2uSD) != XST_SUCCESS){
                    printf("Can't access SD - Ultrasonic!\r\n");
                }
                xSemaphoreGive(xSemaphoreSD);
            }else{
                specRiskAltitude = 1;
                strcpy(alertAltitude, "Low Altitude");
                xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
                sprintf(send2uSD, "\r\n[%d] %s - Level: %d\r\n", (xTaskGetTickCount()*10),
alertAltitude, specRiskAltitude);
                writeRecordTask(send2uSD);
                xSemaphoreGive(xSemaphoreSD);
            }
        }
    } else if((altitude >= minAltitude) && (altitude <= maxAltitude)){ //Altitude between
10000 & 12000 m (OK)
        if(countCorrectReadingsAltitude < 20){
            countCorrectReadingsAltitude++;
        }
        countWrongReadingsAltitude[0] = 0;
        countWrongReadingsAltitude[1] = 0;
        if(countCorrectReadingsAltitude == 3){ //Rectify specRisk
            //specRiskAltitude->3 or 0, cannot rectify
            if(specRiskAltitude == 2){
                specRiskAltitude = 1;
                countCorrectReadingsAltitude = 0;
                strcpy(alertAltitude, "Rectified Altitude");

                xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
                sprintf(send2uSD, "\r\n[%d] %s - Level: %d\r\n", (xTaskGetTickCount()*10),
alertAltitude, specRiskAltitude);
                if(SDFileWrite(send2uSD) != XST_SUCCESS){
                    printf("Can't access SD - Ultrasonic!\r\n");
                }
                xSemaphoreGive(xSemaphoreSD);
            }
        }
    }
}

```

### A3.5. Tarea Velocímetro

### A3.5.1. Definiciones

```
//Velocimeter (P.0)
int specRiskAltitude = 0;
char alertAltitude[32] = " ";
int countCorrectReadingsSpeed = 0;
int countWrongReadingsSpeed[4] = {0, 0, 0, 0};
float speedDB = 0.0;

void readVelocimeter();
```

### A3.5.2. Tarea “velocimeterTask”

```
static void velocimeterTask( void *pvParameters ){
    TickType_t xLastTimeAwake;
    xLastTimeAwake = 0;

    for( ;; ){
        if(started == 1){
            xSemaphoreTake( xSemaphoreSpeed, portMAX_DELAY );
            readVelocimeter();
            xSemaphoreGive( xSemaphoreSpeed );
        }
        vTaskDelayUntil(&xLastTimeAwake, 500/portTICK_RATE_MS);
    }
}
```

### A3.5.3. Función “readVelocimeter”

```
void readVelocimeter(){
    float speed;
    u16 data14 = 0;

    data14 = Xil_In32(C_BASEADDR + 0x278);

    //
    // 0->65535 equivalent 754->900 =>
    // data14/65535*146 is 0->146(difference 900-754)
    //
    speed = ((data14/65535.0)*146.0)+754.0;

    speedDB = speed;

    if(specRiskSpeed!=3){
        if((speed >= 754.0) && (speed < 783.2)){ //1st sector (Too Slow)
            countCorrectReadingsSpeed = 0;
            if(countWrongReadingsSpeed[0]<20)
                countWrongReadingsSpeed[0]++;
            countWrongReadingsSpeed[1] = 0;
            countWrongReadingsSpeed[2] = 0;
            countWrongReadingsSpeed[3] = 0;
            if(countWrongReadingsSpeed[0] >= 3){
                specRiskSpeed = 2;
                strcpy(alertSpeed, "Too Low Speed");

                xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
                sprintf(send2uSD,"%r\n[%ld] %s - Level: %d\r\n", (xTaskGetTickCount()*10),
alertSpeed, specRiskSpeed);
                if(SDFFileWrite(send2uSD) != XST_SUCCESS){
                    printf("Can't access SD - Velocimeter!\r\n");
                }
                xSemaphoreGive(xSemaphoreSD);
            }
        }else if((speed >= 783.2) && (speed < 812.4)){ //2nd sector (Slow)
            countCorrectReadingsSpeed = 0;
            countWrongReadingsSpeed[0] = 0;
            if(countWrongReadingsSpeed[1]<20)
                countWrongReadingsSpeed[1]++;
            countWrongReadingsSpeed[2] = 0;
            countWrongReadingsSpeed[3] = 0;
            if(countWrongReadingsSpeed[1] >= 3){
                specRiskSpeed = 1;
                strcpy(alertSpeed, "Low Speed");

                xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
                sprintf(send2uSD,"%r\n[%ld] %s - Level: %d\r\n", (xTaskGetTickCount()*10),
alertSpeed, specRiskSpeed);
                if(SDFFileWrite(send2uSD) != XST_SUCCESS){
```

### Anexo III - Tarea Velocímetro

```

        printf("Can't access SD - Velocimeter!\r\n");
    }
    xSemaphoreGive(xSemaphoreSD);
}
else if((speed >= 812.4) && (speed < 841.6)){ //3rd sector (OK)
    if(countCorrectReadingsSpeed<20)
        countCorrectReadingsSpeed++;
    countWrongReadingsSpeed[0] = 0;
    countWrongReadingsSpeed[1] = 0;
    countWrongReadingsSpeed[2] = 0;
    countWrongReadingsSpeed[3] = 0;
    if(countCorrectReadingsSpeed == 2){ //Rectify specRisk
        //specRiskSpeed->3 or 0, cannot rectify
        if(specRiskSpeed == 2){
            specRiskSpeed = 1;
            countCorrectReadingsSpeed = 0;
            strcpy(alertSpeed, "Rectified Speed");

            xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
            sprintf(send2uSD, "\r\n[%ld] %s - Level: %d\r\n", (xTaskGetTickCount()*10),
alertSpeed, specRiskSpeed);
            if(SDFileWrite(send2uSD) != XST_SUCCESS){
                printf("Can't access SD - Velocimeter!\r\n");
            }
            xSemaphoreGive(xSemaphoreSD);
        }
        else if(specRiskSpeed == 1){
            specRiskSpeed = 0;
            strcpy(alertSpeed, " ");
            xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
            sprintf(send2uSD, "\r\n[%ld] No Risk Speed\r\n", (xTaskGetTickCount()*10));
            if(SDFileWrite(send2uSD) != XST_SUCCESS){
                printf("Can't access SD - Velocimeter!\r\n");
            }
            xSemaphoreGive(xSemaphoreSD);
        }
    }
}else if((speed >= 841.6) && (speed < 870.8)){ //4th sector (Fast)
    countCorrectReadingsSpeed = 0;
    countWrongReadingsSpeed[0] = 0;
    countWrongReadingsSpeed[1] = 0;
    if(countWrongReadingsSpeed[2]<20)
        countWrongReadingsSpeed[2]++;
    countWrongReadingsSpeed[3] = 0;
    if(countWrongReadingsSpeed[2] >= 3){
        specRiskSpeed = 1;
        strcpy(alertSpeed, "High Speed");

        xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
        sprintf(send2uSD, "\r\n[%ld] %s - Level: %d\r\n", (xTaskGetTickCount()*10),
alertSpeed, specRiskSpeed);
        if(SDFileWrite(send2uSD) != XST_SUCCESS){
            printf("Can't access SD - Velocimeter!\r\n");
        }
        xSemaphoreGive(xSemaphoreSD);
    }
}else if((speed >= 870.8) && (speed <= 900)){ //5th sector (Too Fast)
    countCorrectReadingsSpeed = 0;
    countWrongReadingsSpeed[0] = 0;
    countWrongReadingsSpeed[1] = 0;
    countWrongReadingsSpeed[2] = 0;
    if(countWrongReadingsSpeed[3]<20)
        countWrongReadingsSpeed[3]++;
    if(countWrongReadingsSpeed[3] >= 3){
        specRiskSpeed = 2;
        strcpy(alertSpeed, "Too High Speed");

        xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
        sprintf(send2uSD, "\r\n[%ld] %s - Level: %d\r\n", (xTaskGetTickCount()*10),
alertSpeed, specRiskSpeed);
        if(SDFileWrite(send2uSD) != XST_SUCCESS){
            printf("Can't access SD - Velocimeter!\r\n");
        }
    }
}

```

```
    xSemaphoreGive(xSemaphoreSD);
}
}else{ //Other or error
    printf("Velocimeter Out of Value");
}
}
```

### A3.6. Tarea Presencia

### A3.6.1. Definiciones

```
//Presence (P.O)
int specRiskPresence = 0;
char alertPresence[32] = " ";
int countCorrectReadingsPresence = 0;
int countWrongReadingsPresence[2] = {0, 0};
int presenceDB = 0;

void readPresence();
```

### A3.6.2. Tarea “presenceTask”

```
static void presenceTask( void *pvParameters ){
    TickType_t xLastTimeAwake;
    xLastTimeAwake = 0;

    for( ;; ){

        if(started == 1){
            xSemaphoreTake( xSemaphorePresence, portMAX_DELAY );
            readPresence();
            xSemaphoreGive( xSemaphorePresence );
        }
        vTaskDelayUntil(&xLastTimeAwake, 600/portTICK_RATE_MS);
    }
}
```

### A3.6.3. Función “readPresence”

```
void readPresence(){
    int presence = 0;

    if(GPIO_getPin(&myDeviceJD_io, 1) == 0){ //Detected first pilot?
        presence++;
    }
    if( GPIO_getPin(&myDeviceJD_io, 2) == 0){ //There is a second pilot?
        presence++;
    }
    if(GPIO_getPin(&myDeviceJD_io, 4) == 1){ //There is movement near the door?
        presence++;
    }

    presenceDB = presence;

    if(specRiskPresence!=3){
        if(presence == 2){ //2 people in airplane cockpit
            if(countCorrectReadingsPresence<20)
```

### Anexo III - Tarea Presencia

```

        countCorrectReadingsPresence++;
        countWrongReadingsPresence[0] = 0;
        countWrongReadingsPresence[1] = 0;
        if(countCorrectReadingsPresence == 3){ //Rectify specRisk
            //specRiskPresence->3 or 0, cannot rectify
            if(specRiskPresence == 2){
                specRiskPresence = 1;
                countCorrectReadingsPresence = 0;
                strcpy(alertPresence, "Rectified Presence");

                xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
                sprintf(send2uSD, "\r\n[%ld] %s - Level: %d\r\n", (xTaskGetTickCount()*10),
                        alertPresence, specRiskPresence);
                if(SDFileWrite(send2uSD) != XST_SUCCESS){
                    printf("Can't access SD - Presence!\r\n");
                }
                xSemaphoreGive(xSemaphoreSD);
            }
            else if(specRiskPresence == 1){
                specRiskPresence = 0;
                strcpy(alertPresence, " ");

                xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
                sprintf(send2uSD, "\r\n[%ld] No Risk Presence\r\n", (xTaskGetTickCount()*10));
                if(SDFileWrite(send2uSD) != XST_SUCCESS){
                    printf("Can't access SD - Presence!\r\n");
                }
                xSemaphoreGive(xSemaphoreSD);
            }
        }
    } else if(presence <= 1){ //only 1 person detected (or no pilots)
        countCorrectReadingsPresence = 0;
        if(countCorrectReadingsPresence<20)
            countWrongReadingsPresence[0]++;
        countWrongReadingsPresence[1] = 0;
        if(countWrongReadingsPresence[0] >= 3){
            if(countWrongReadingsPresence[0] >=6){
                specRiskPresence = 3;
                strcpy(alertPresence, "Risk: Low Presence");

                xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
                sprintf(send2uSD, "\r\n[%ld] %s - Level: %d\r\n", (xTaskGetTickCount()*10),
                        alertPresence, specRiskPresence);
                if(SDFileWrite(send2uSD) != XST_SUCCESS){
                    printf("Can't access SD - Presence!\r\n");
                }
                xSemaphoreGive(xSemaphoreSD);
            }
        } else{
            specRiskPresence = 2;
            strcpy(alertPresence, "Low Presence");

            xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
            sprintf(send2uSD, "\r\n[%ld] %s - Level: %d\r\n", (xTaskGetTickCount()*10),
                    alertPresence, specRiskPresence);
            if(SDFileWrite(send2uSD) != XST_SUCCESS){
                printf("Can't access SD - Presence!\r\n");
            }
            xSemaphoreGive(xSemaphoreSD);
        }
    }
} else if(presence > 2){ // 2+ people in airplane cockpit
    countCorrectReadingsPresence = 0;
    countWrongReadingsPresence[0] = 0;
    if(countCorrectReadingsPresence<20)
        countWrongReadingsPresence[1]++;
    if(countWrongReadingsPresence[1] >= 3){
        if(countWrongReadingsPresence[1] >=6){
            specRiskPresence = 2;
        }
    } else{
        specRiskPresence = 1;
        strcpy(alertPresence, "High Presence");
    }
}

```

### A3.7. Tarea Control de Riesgo

### A3.7.1. Definiciones

```
//RiskControl (P.0)
int genRisk = 0;

void pmodLedWork(u8 led);
void rgbLedWork(u8 risk);
void useActuator(u8 risk,u8 counter);
```

### A3.7.2. Tarea “riskControlTask”

```

static void riskControlTask( void *pvParameters){
    TickType_t timed, old_timed, xLastTimeAwake;
    TickType_t totalCorrect = 0;
    char messageSD[64] = " ";
    u8 specRisk[4];
    u8 counterL;
    u8 counterG;
    u8 counterC;
    int intGenRisk;
    int oldRiskValue = 0;

    xLastTimeAwake = 0;
    old_timed = 0;
    timed = 0;

    for( ; ; ){
        if(started == 1){
            xSemaphoreTake( xSemaphoreAltitude, portMAX_DELAY );
            specRisk[0] = specRiskAltitude;
            xSemaphoreGive( xSemaphoreAltitude );

            xSemaphoreTake( xSemaphoreTilt, portMAX_DELAY );
            specRisk[1] = specRiskTilt;
            xSemaphoreGive( xSemaphoreTilt );

            xSemaphoreTake( xSemaphorePresence, portMAX_DELAY );
            specRisk[2] = specRiskPresence;
            xSemaphoreGive( xSemaphorePresence );

            xSemaphoreTake( xSemaphoreSpeed, portMAX_DELAY );
            specRisk[3] = specRiskSpeed;
            xSemaphoreGive( xSemaphoreSpeed );

            xSemaphoreTake(xSemaphoreRisk, portMAX_DELAY );
            oldRiskValue = genRisk;
            xSemaphoreGive(xSemaphoreRisk);
        }
    }
}

```

```

if(oldRiskValue != 3){
    counterL = 0;
    counterG = 0;
    counterC = 0;

    for(int i=0;i<sizeof(specRisk);i++){
        if(specRisk[i]==1){
            counterL++;
        }else if(specRisk[i]==2){
            counterG++;
        }else if(specRisk[i]==3){
            counterC++;
        }
    }

    intGenRisk = 0;

    if(counterL > 0){
        if(counterL < 3){
            intGenRisk = 1;
        }else{
            counterG++;
        }
    }
    if(counterG > 0){
        if(counterG < 2){
            intGenRisk = 2;
        }else{
            counterC++;
        }
    }
    if(counterC > 0){
        intGenRisk = 3;
    }
}

if(intGenRisk == 0){
    if(totalCorrect < pdMS_TO_TICKS(5000UL)){
        timed = xTaskGetTickCount();
        totalCorrect += timed-old_timed;
    }if((totalCorrect >= pdMS_TO_TICKS(5000UL)) && (oldRiskValue != 0)){
        if(oldRiskValue == 2){
            oldRiskValue = 1;
            xSemaphoreTake(xSemaphoreRisk, portMAX_DELAY );
            genRisk = oldRiskValue;
            xSemaphoreGive(xSemaphoreRisk);
            xSemaphoreTake(xSemaphoreSD, portMAX_DELAY );
            sprintf(messageSD, "\r\n[%ld] Risk level rectified to %d (%s)\r\n",
(xTaskGetTickCount()*10), oldRiskValue, getLevelRisk(oldRiskValue));
            if(SDfileWrite(messageSD) != XST_SUCCESS){
                printf("Can't access SD - Risk!\r\n");
            }
            xSemaphoreGive(xSemaphoreSD);
            totalCorrect = 0;
            useActuator(1, 1);
        }else if(oldRiskValue == 1){
            oldRiskValue = 0;
            xSemaphoreTake(xSemaphoreRisk, portMAX_DELAY );
            genRisk = oldRiskValue;
            xSemaphoreGive(xSemaphoreRisk);

            xSemaphoreTake( xSemaphoreAltitude, portMAX_DELAY );
            strcpy(alertAltitude, " ");
            xSemaphoreGive( xSemaphoreAltitude );

            xSemaphoreTake( xSemaphoreTilt, portMAX_DELAY );
            strcpy(alertTilt, " ");
            xSemaphoreGive( xSemaphoreTilt );

            xSemaphoreTake( xSemaphorePresence, portMAX_DELAY );
            strcpy(alertPresence, " ");
            xSemaphoreGive( xSemaphorePresence );
        }
    }
}

```

```

        xSemaphoreTake( xSemaphoreSpeed, portMAX_DELAY );
        strcpy(alertSpeed, " ");
        xSemaphoreGive( xSemaphoreSpeed );

        xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
        sprintf(messageSD,"\\r\\n[%ld] Risk level rectified to %d (%s)\\r\\n",
(xTaskGetTickCount()*10), oldRiskValue, getLevelRisk(oldRiskValue));
        if(SDFileWrite(messageSD) != XST_SUCCESS){
            printf("Can't access SD - Risk!\\r\\n");
        }
        xSemaphoreGive(xSemaphoreSD);
        useActuator(0, 0);
    }else if(oldRiskValue == 0){
        useActuator(0, 0);
    }
}
}else{
    totalCorrect = 0;
    timed = xTaskGetTickCount();

    if(intGenRisk == 1){
        xSemaphoreTake(xSemaphoreRisk, portMAX_DELAY );
        genRisk = 1;
        xSemaphoreGive(xSemaphoreRisk);

        //Activate Level 1 response if 1st time
        if(oldRiskValue != intGenRisk){
            useActuator(intGenRisk, counterL);
        }
        xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
        sprintf(messageSD,"\\r\\n[%ld] Level %d activated!\\r\\n",
(xTaskGetTickCount()*10), intGenRisk);
        if(SDFileWrite(messageSD) != XST_SUCCESS){
            printf("Can't access SD - Risk!\\r\\n");
        }
        xSemaphoreGive(xSemaphoreSD);
    } else if(intGenRisk == 2){
        xSemaphoreTake(xSemaphoreRisk, portMAX_DELAY );
        genRisk = 2;
        xSemaphoreGive(xSemaphoreRisk);

        //Activate Level 2 response if first time
        if(oldRiskValue != intGenRisk){
            useActuator(intGenRisk, counterG);
        }
        xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
        sprintf(messageSD,"\\r\\n[%ld] Level %d activated!\\r\\n",
(xTaskGetTickCount()*10), intGenRisk);
        if(SDFileWrite(messageSD) != XST_SUCCESS){
            printf("Can't access SD - Risk!\\r\\n");
        }
        xSemaphoreGive(xSemaphoreSD);
    } else if(intGenRisk == 3){
        xSemaphoreTake(xSemaphoreRisk, portMAX_DELAY );
        genRisk = 3;
        xSemaphoreGive(xSemaphoreRisk);
        xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
        sprintf(messageSD,"\\r\\n[%ld] Level %d activated!\\r\\n",
(xTaskGetTickCount()*10), intGenRisk);
        if(SDFileWrite(messageSD) != XST_SUCCESS){
            printf("Can't access SD - Risk!\\r\\n");
        }
        xSemaphoreGive(xSemaphoreSD);

        if(oldRiskValue != intGenRisk){
            useActuator(intGenRisk, counterC);
        }
    }
}
old_timed = timed;
}
vTaskDelayUntil(&xLastTimeAwake, 500/portTICK_RATE_MS);
}
}

```

### A3.7.3. Función “useActuator”

```

void useActuator(u8 risk, u8 counter){
    char message[64];
    if(risk == 3){
        pmodLedWork(4);
        rgblLedWork(risk);
        xSemaphoreGive(xSemaphoreBuzzer);
        if (xTimerReset(timerHndl3SecTimeout, 0)!=pdPASS) { // start timer 3 seconds
            for(;;);
        }
        xSemaphoreGive(xSemaphoreAutopilot);
        sprintf(message,"\\r\\n[%ld] * Autopilot started! *\r\n", (xTaskGetTickCount()*10));
        xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
        if (SDFileWrite(message) != XST_SUCCESS){
            printf("Can't access SD - Autopilot!\r\n");
        }
        xSemaphoreGive(xSemaphoreSD);
        printf("%s",message);
        if (xTimerReset(timerHndl15SecTimeout, 0)!=pdPASS) { // start timer 15 seconds
            for(;;);
        }
    }else if(risk == 2){
        pmodLedWork(3);
        rgblLedWork(risk);
        xSemaphoreGive(xSemaphoreBuzzer);
        if (xTimerReset(timerHndl3SecTimeout, 0)!=pdPASS) { // start timer 3 seconds
            for(;;);
        }
    }else if(risk == 1){
        pmodLedWork(counter);
        rgblLedWork(risk);
        xSemaphoreGive(xSemaphoreBuzzer);
        if (xTimerReset(timerHndl3SecTimeout, 0)!=pdPASS) { // start timer 3 seconds
            for(;;);
        }
    }else if(risk == 0){
        pmodLedWork(0);
        rgblLedWork(0);
    }
}

```

### A3.7.4. Funciones diodos LED

```

void pmodLedWork(u8 led){
    GPIO_setPin(&myDeviceJE_o, 5, 0);
    GPIO_setPin(&myDeviceJE_o, 6, 0);
    GPIO_setPin(&myDeviceJE_o, 7, 0);
    GPIO_setPin(&myDeviceJE_o, 8, 0);

    if(led >= 1){
        GPIO_setPin(&myDeviceJE_o, 5, 1);
    }
    if(led >= 2){
        GPIO_setPin(&myDeviceJE_o, 6, 1);
    }
    if(led >= 3){
        GPIO_setPin(&myDeviceJE_o, 7, 1);
    }
    if(led >= 4){
        GPIO_setPin(&myDeviceJE_o, 8, 1);
    }

    xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
    sprintf(send2uSD,"\\r\\n[%d] Activated PmodLED - LEDs ON: %d\\r\\n",
    (xTaskGetTickCount()*10), led);
    if (SDFileWrite(send2uSD) != XST_SUCCESS){

```

```

        printf("Can't access SD - PmodLED!\r\n");
    }
    xSemaphoreGive(xSemaphoreSD);
}

void rgbLedWork(u8 risk){
    if(risk == 1){ //Light Risk = Blue
        GPIO_setPin(&myDeviceJE_o, 1, 0);
        GPIO_setPin(&myDeviceJE_o, 2, 0);
        GPIO_setPin(&myDeviceJE_o, 3, 1);
    }else if(risk == 2){ //Grave/Serious Risk = Yellow
        GPIO_setPin(&myDeviceJE_o, 1, 1);
        GPIO_setPin(&myDeviceJE_o, 2, 1);
        GPIO_setPin(&myDeviceJE_o, 3, 0);
    }else if(risk == 3){ //Critical Risk = Red
        GPIO_setPin(&myDeviceJE_o, 1, 1);
        GPIO_setPin(&myDeviceJE_o, 2, 0);
        GPIO_setPin(&myDeviceJE_o, 3, 0);
    }else{ //Risk 0 or not defined = OFF
        GPIO_setPin(&myDeviceJE_o, 1, 0);
        GPIO_setPin(&myDeviceJE_o, 2, 0);
        GPIO_setPin(&myDeviceJE_o, 3, 0);
    }

    xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
    sprintf(send2uSD, "\r\n[%ld] Activated RGB LED - %s\r\n", (xTaskGetTickCount()*10),
    getLevelRisk(risk));
    if(SDfileWrite(send2uSD) != XST_SUCCESS){
        printf("Can't access SD - RGB LED!\r\n");
    }
    xSemaphoreGive(xSemaphoreSD);
}
}

```

### A3.7.5. Tarea “buzzerTask”

```

static void buzzerTask(void *pvParameters){ //Not using
    TickType_t xLastTimeAwake;
    xLastTimeAwake = 0;
    int level = 0;

    for(;;){
        xSemaphoreTake(xSemaphoreBuzzer, portMAX_DELAY);
        xSemaphoreTake(xSemaphoreRisk, portMAX_DELAY );
        level = genRisk;
        xSemaphoreGive(xSemaphoreRisk);

        if(level == 1){
            GPIO_setPin(&myDeviceJE_o, 4, 1);
            usleep(600);
            GPIO_setPin(&myDeviceJE_o, 4, 0);
            usleep(600);
        }else if(level == 2){
            GPIO_setPin(&myDeviceJE_o, 4, 1);
            usleep(2000);
            GPIO_setPin(&myDeviceJE_o, 4, 0);
            usleep(2000);
        }else if(level == 3){
            GPIO_setPin(&myDeviceJE_o, 4, 1);
            usleep(5000);
            GPIO_setPin(&myDeviceJE_o, 4, 0);
            usleep(5000);
        }else{
            GPIO_setPin(&myDeviceJE_o, 4, 0);
        }

        xSemaphoreGive(xSemaphoreBuzzer);

        vTaskDelayUntil(&xLastTimeAwake, 15/portTICK_RATE_MS);
    }
}

```

### A3.7.6. Temporizadores

```

static void vTimerCallback3SecExpired(xTimerHandle pxTimer);
static void vTimerCallback15SecExpired(xTimerHandle pxTimer);

xTimerHandle timerHndl3SecTimeout;
xTimerHandle timerHndl15SecTimeout;

timerHndl3SecTimeout = xTimerCreate(
    "timer3Sec", /* name */
    pdMS_TO_TICKS(3000), /* period/time */
    pdFALSE, /* auto reload */
    (void*)1, /* timer ID */
    vTimerCallback3SecExpired); /* callback */
if (timerHndl3SecTimeout==NULL) {
    for(;;); /* failure! */
}

timerHndl15SecTimeout = xTimerCreate(
    "timer15sec",
    pdMS_TO_TICKS(15000),
    pdFALSE,
    (void*)2,
    vTimerCallback15SecExpired);
if (timerHndl15SecTimeout==NULL) {
    for(;;);
}

static void vTimerCallback3SecExpired(xTimerHandle pxTimer) {
    xSemaphoreTake(xSemaphoreBuzzer, portMAX_DELAY);
}

static void vTimerCallback15SecExpired(xTimerHandle pxTimer) {
    xSemaphoreTake(xSemaphoreAutopilot, portMAX_DELAY);
    pmodLedWork(0);
    rgbLedWork(0);
    xSemaphoreTake(xSemaphoreRisk, portMAX_DELAY );
    genRisk = 0;
    xSemaphoreGive(xSemaphoreRisk);
    xSemaphoreTake(xSemaphoreAltitude, portMAX_DELAY );
    specRiskAltitude = 0;
    xSemaphoreGive(xSemaphoreAltitude);
    xSemaphoreTake(xSemaphorePresence, portMAX_DELAY );
    specRiskPresence = 0;
    xSemaphoreGive(xSemaphorePresence);
    xSemaphoreTake(xSemaphoreSpeed, portMAX_DELAY );
    specRiskSpeed = 0;
    xSemaphoreGive(xSemaphoreSpeed);
    xSemaphoreTake(xSemaphoreTilt, portMAX_DELAY );
    specRiskTilt = 0;
    xSemaphoreGive(xSemaphoreTilt);
    xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
    sprintf(send2uSD, "\r\n[%ld] No risk due to autopilot\r\n", (xTaskGetTickCount()*10));
    if(SDFileWrite(send2uSD) != XST_SUCCESS){
        printf("Can't access SD - Risk!\r\n");
    }
    xSemaphoreGive(xSemaphoreSD);
}

```

## A3.8. Tarea Display

### A3.8.1. Tarea “displayTask”

```

static void displayTask( void *pvParameters ){
    TickType_t xLastTimeAwake;
    char alertDALatitude[32] = " ";
    char alertDPresence[32] = " ";
    char alertDSpeed[32] = " ";
    char alertDTilt[32] = " ";
    u8 specRisk[4];
    float AccDis[2];
    int PresenceDis;
    float SpeedDis;
    int AltitudeDis;
    int genRiskD;

    xLastTimeAwake = 0;
    for( ;; ){
        if(started == 1){
            xSemaphoreTake(xSemaphoreSD, portMAX_DELAY);
            sprintf(send2uSD, "\r\n[%ld] Display Task Executed\r\n", (xTaskGetTickCount()*10));
            if(SDFFileWrite(send2uSD) != XST_SUCCESS){
                printf("Can't access SD - Display!\r\n");
            }
        }
        xSemaphoreGive(xSemaphoreSD);

        xSemaphoreTake( xSemaphoreAltitude, portMAX_DELAY );
        specRisk[0] = specRiskAltitude;
        strcpy(alertDALatitude, alertAltitude);
        AltitudeDis = altitudeDB;
        xSemaphoreGive( xSemaphoreAltitude );

        xSemaphoreTake( xSemaphoreTilt, portMAX_DELAY );
        specRisk[1] = specRiskTilt;
        strcpy(alertDTilt, alertTilt);
        AccDis[0] = Acc[0];
        AccDis[1] = Acc[1];
        xSemaphoreGive( xSemaphoreTilt);

        xSemaphoreTake( xSemaphorePresence, portMAX_DELAY );
        specRisk[2] = specRiskPresence;
        strcpy(alertDPresence, alertPresence);
        PresenceDis = presenceDB;
        xSemaphoreGive( xSemaphorePresence );

        xSemaphoreTake( xSemaphoreSpeed, portMAX_DELAY );
        specRisk[3] = specRiskSpeed;
        strcpy(alertDSpeed, alertSpeed);
        SpeedDis = speedDB;
        xSemaphoreGive( xSemaphoreSpeed );

        xSemaphoreTake( xSemaphoreRisk, portMAX_DELAY );
        genRiskD = genRisk;
        xSemaphoreGive(xSemaphoreRisk);

        printf("\r\n[%ld] Display Task\r\n", (xTaskGetTickCount()*10));

        printf("\r\nValues\r\n");
        printf("\tAltitude Values: %d meters (Sim: %d cm)\r\n",
AltitudeDis*realValueRatio, AltitudeDis);
        printf("\tTilt Values: X = %0.2f °, Y = %0.2f °\r\n", AccDis[0], AccDis[1]);
        printf("\tPresence Values: %d\r\n", PresenceDis);
        printf("\tSpeed Values: %0.2f km/h\r\n", SpeedDis);

        if(debug == 1){
            printf("\r\nSpecific Risk (Debug)\r\n");
            printf("\tAltitude Values: %d (%s)\r\n", specRisk[0], getLevelRisk(specRisk[0]));
        }
    }
}

```

### Anexo III - Tarea Tarjeta microSD

```
printf("\tTilt Values: %d (%s)\r\n", specRisk[1], getLevelRisk(specRisk[1]));
printf("\tPresence Values: %d (%s)\r\n", specRisk[2], getLevelRisk(specRisk[2]));
printf("\tSpeed Values: %d (%s)\r\n", specRisk[3], getLevelRisk(specRisk[3]));
}

printf("\r\nGeneral Risk: %d (%s)\r\n", genRiskD, getLevelRisk(genRiskD));
if(genRiskD != 0){
    printf("\r\nAlerts\r\n");
    if(strcmp(alertDAltitude, " ") ){
        printf("\tAltitude: %s \r\n", alertDAltitude);
    }
    if(strcmp(alertDTilt, " ") ){
        printf("\tTilt: %s \r\n", alertDTilt);
    }
    if(strcmp(alertDPresence, " ") ){
        printf("\tPresence: %s \r\n", alertDPresence);
    }
    if(strcmp(alertDSpeed, " ") ){
        printf("\tSpeed: %s \r\n", alertDSpeed);
    }
}
vTaskDelayUntil(&xLastTimeAwake, 2000/portTICK_RATE_MS);
}
```

## A3.9. Tarea Tarjeta microSD

### A3.9.1. Definiciones

```
//SD Card
FIL fil;
FATFS fatfs;
char FileName[32] = "Record_.txt";
char *SD_File;

u8 SourceAddress[1024*2] __attribute__ ((aligned(32)));
char send2uSD[64];

int SDFileWrite(char *record);
char * getLevelRisk(int risk);
```

### A3.9.2. Tarea “writeRecordTask”

```
static void writeRecordTask(void *pvParameters){
    int Status;
    char dataRecord[1024*2];
    TickType_t xLastTimeAwake;
    XTime timeX;

    dataRecord[0] = '\0';
    xLastTimeAwake = 0;

    for(;;){
        if(started){
            //Create the record to write at once
            XTime_GetTime(&timeX);
            sprintf(dataRecord, "\r\n[%lld] Started SD Recording.\r\n\r\n",
(timeX/(COUNTS_PER_SECOND/1000)));

            xSemaphoreTake( xSemaphoreKB, portMAX_DELAY );
            sprintf(dataRecord + strlen(dataRecord), "\tModes enabled:\r\n\t\tActive/Started:
%s\r\n\t\tDebug: %s\r\n\r\n", ((started)? "True": "False"), ((debug)? "True": "False")));
        }
    }
}
```

```

        xSemaphoreGive( xSemaphoreKB);

        xSemaphoreTake( xSemaphoreRisk, portMAX_DELAY);
        sprintf(dataRecord + strlen(dataRecord), "\tGeneral Risk:\r\n\t\tLevel: %d
(%s)\r\n\r\n", genRisk, getLevelRisk(genRisk));
        xSemaphoreGive( xSemaphoreRisk);

        xSemaphoreTake( xSemaphoreTilt, portMAX_DELAY);
        sprintf(dataRecord + strlen(dataRecord), "\tSpecific Risk
(Accelerometer/Tilt):\r\n\t\tRisk Level: %d (%s)\r\n\t\tAlert: \'%s\'\r\n\t\tActual Tilt Angle
(X, Y): (%0.2f, %0.2f)\r\n\t\tCounters (Correct, Wrong AxisX, Wrong AxisY): (%d, %d,
%d)\r\n\r\n", specRiskTilt, getLevelRisk(specRiskTilt), alertTilt, Acc[0], Acc[1],
countCorrectReadingsTilt, AxisX, AxisY);
        xSemaphoreGive( xSemaphoreTilt);

        xSemaphoreTake( xSemaphoreAltitude, portMAX_DELAY);
        sprintf(dataRecord + strlen(dataRecord), "\tSpecific Risk
(Altimeter/Altitude):\r\n\t\tRisk Level: %d (%s)\r\n\t\tAlert: \'%s\'\r\n\t\tActual Altitude
\"Simulated\" (Real): %d m (%d cm)\r\n\t\tSet Altitude (Min - Max): (%d - %d)\r\n\t\tCounters
(Wrong Low, Correct, Wrong High): (%d, %d, %d)\r\n\r\n", specRiskAltitude,
getLevelRisk(specRiskAltitude), alertAltitude, (altitudeDB*realValueRatio), altitudeDB,
minAltitude, maxAltitude, countWrongReadingsAltitude[0], countCorrectReadingsAltitude,
countWrongReadingsAltitude[1]);
        xSemaphoreGive( xSemaphoreAltitude);

        xSemaphoreTake( xSemaphoreSpeed, portMAX_DELAY);
        sprintf(dataRecord + strlen(dataRecord), "\tSpecific Risk
(Velocimeter/Speed):\r\n\t\tRisk Level: %d (%s)\r\n\t\tAlert: \'%s\'\r\n\t\tActual Speed:
%0.2f km/h\r\n\t\tCounters (Wrong Too Low, Wrong Low, Correct, Wrong High, Wrong Too High):
(%d, %d, %d, %d)\r\n\r\n", specRiskSpeed, getLevelRisk(specRiskSpeed), alertSpeed,
speedDB, countWrongReadingsSpeed[0], countWrongReadingsSpeed[1], countCorrectReadingsSpeed,
countWrongReadingsSpeed[2], countWrongReadingsSpeed[3]);
        xSemaphoreGive( xSemaphoreSpeed);

        xSemaphoreTake( xSemaphorePresence, portMAX_DELAY);
        sprintf(dataRecord + strlen(dataRecord), "\tSpecific Risk (Presence):\r\n\t\tRisk
Level: %d (%s)\r\n\t\tAlert: \'%s\'\r\n\t\tActual Presence: %d\r\n\t\tCounters (Wrong Low,
Correct, Wrong High): (%d, %d, %d)\r\n\r\n", specRiskPresence, getLevelRisk(specRiskPresence),
alertPresence, presenceDB, countWrongReadingsPresence[0], countCorrectReadingsPresence,
countWrongReadingsPresence[1]);
        xSemaphoreGive( xSemaphorePresence);

        XTime_GetTime(&timeX);
        sprintf(dataRecord + strlen(dataRecord), "\r\n[%lld] Ended SD Recording.\r\n",
(timeX/(COUNTS_PER_SECOND/1000)));

        //Write into SD
        xSemaphoreTake( xSemaphoreSD, portMAX_DELAY);
        Status = SDFileWrite(dataRecord);
        if (Status != XST_SUCCESS) {
            xil_printf("Writing record into SD failed!\r\n");
        }
        xSemaphoreGive( xSemaphoreSD);

    }
    vTaskDelayUntil(&xLastTimeAwake, 5000/portTICK_RATE_MS);
}
}

```

### A3.9.3. Función “SDFileWrite()”

```

int SDFileWrite(char *record){
    HRESULT Res;
    UINT NumBytesWritten;
    u32 BuffCnt;
    u32 MaxSize = (1024*2);
    TCHAR *Path = "0:/"; //Main partition
    char text[MaxSize];

    //Copy message to Buffer

```

```

strcpy(text, record);

for(BuffCnt = 0; BuffCnt < strlen(text); BuffCnt++){
    SourceAddress[BuffCnt] = (int) text[BuffCnt];
}

//Mount SD_card
Res = f_mount(&fatfs, Path, 0);
if (Res != FR_OK) {
    xil_printf("SD f_mount failed \r\n");
    return XST_FAILURE;
}

SD_File = (char *)FileName;

//Open the file
Res = f_open(&fil, SD_File, FA_OPEN_APPEND | FA_WRITE );
if (Res) {
    xil_printf("SD f_open failed \r\n");
    return XST_FAILURE;
}

//Write into file the buffer
Res = f_write(&fil, (const void*)SourceAddress, strlen(text), &NumBytesWritten);
if (Res) {
    xil_printf("SD f_write failed \r\n");
    return XST_FAILURE;
}

//Close the file
Res = f_close(&fil);
if (Res) {
    return XST_FAILURE;
}

return XST_SUCCESS;
}

```

### A3.9.4. Función “getLevelRisk()”

```

char * getLevelRisk(int risk){
    if(risk == 0){
        return "No Risk";
    }else if(risk == 1){
        return "Light Risk";
    }else if(risk == 2){
        return "Severe Risk";
    }else if(risk == 3){
        return "Critical Risk";
    }else{
        return '\0';
    }
}

```

## A3.10. Función “main”

### A3.10.1. Librerías

```
/* FreeRTOS includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "semphr.h"
#include "stdio.h"
#include "math.h"
/* Xilinx includes. */
#include "xil_printf.h"
#include "xparameters.h"
#include "PmodGPIO.h"
#include "xgpio.h"
#include "sleep.h"
#include "xsysmon.h"
#include <xtime_l.h>
#include "xiic.h"
#include "PmodKYPD.h"
#include "xsdps.h"
#include "ff.h"
#include "xil_cache.h"
#include "xplatform_info.h"

#define C_BASEADDR 0x43C00000
```

### A3.10.2. Declaraciones

```
/*-----TASKS-----*/
static void velocimeterTask( void *pvParameters );
static void keyPadTask( void *pvParameters );
static void accelerometerTask( void *pvParameters );
static void displayTask( void *pvParameters );
static void presenceTask( void *pvParameters );
static void ultrasonicTask( void *pvParameters );
static void useSystemTask( void *pvParameters );
static void riskControlTask( void *pvParameters );
static void writeRecordTask( void *pvParameters );
static void buzzerTask( void *pvParameters );
/*-----*/

/*-----Task Handle-----*/
static TaskHandle_t xVelocimeterTask;
static TaskHandle_t xKeyPadTask;
static TaskHandle_t xAccelerometerTask;
static TaskHandle_t xDisplayTask;
static TaskHandle_t xPresenceTask;
static TaskHandle_t xUsonicTask;
static TaskHandle_t xUseSystemTask;
static TaskHandle_t xRiskControlTask;
static TaskHandle_t xRecordTask;
static TaskHandle_t xBuzzerTask;
/*-----*/
```

```
/*-----Semaphores-----*/
SemaphoreHandle_t xSemaphoreBuzzer = NULL;
SemaphoreHandle_t xSemaphoreAutopilot = NULL;
SemaphoreHandle_t xSemaphoreKB = NULL;
SemaphoreHandle_t xSemaphoreSpeed = NULL;
SemaphoreHandle_t xSemaphoreAltitude = NULL;
SemaphoreHandle_t xSemaphoreTilt = NULL;
SemaphoreHandle_t xSemaphorePresence = NULL;
SemaphoreHandle_t xSemaphoreRisk = NULL;
SemaphoreHandle_t xSemaphoreSD = NULL;
/*-----*/
```

### A3.10.3. Código “main()”

```
int main( void ){
    setupBoard();
    setupGPIO();
    xil_printf("Main task system!\r\n" );

    timerHndl3SecTimeout = xTimerCreate(
        "timer3Sec", /* name */
        pdMS_TO_TICKS(3000), /* period/time */
        pdFALSE, /* auto reload */
        (void*)1, /* timer ID */
        vTimerCallback3SecExpired); /* callback */
    if (timerHndl3SecTimeout==NULL) {
        for(;;); /* failure! */
    }

    timerHndl15SecTimeout = xTimerCreate(
        "timer15sec",
        pdMS_TO_TICKS(15000),
        pdFALSE,
        (void*)2,
        vTimerCallback15SecExpired);
    if (timerHndl15SecTimeout==NULL) {
        for(;;);
    }

    //TASKS
    xTaskCreate( keyPadTask,
        ( const char * ) "KeyPad",
        configMINIMAL_STACK_SIZE,
        NULL,
        tskIDLE_PRIORITY+1,
        &xKeyPadTask );

    xTaskCreate( writeRecordTask,
        ( const char * ) "Record",
        2048,
        NULL,
        tskIDLE_PRIORITY+2,
        &xRecordTask );

    xTaskCreate(displayTask,
        ( const char * ) "Display",
        1024,
        NULL,
        tskIDLE_PRIORITY+3,
        &xDisplayTask );

    xTaskCreate( useSystemTask,
        ( const char * ) "System",
        1024,
        NULL,
        tskIDLE_PRIORITY+4,
        &xUseSystemTask );

    xTaskCreate( ultrasonicTask,
        ( const char * ) "USonic",
```

```

    configMINIMAL_STACK_SIZE,
    NULL,
    tskIDLE_PRIORITY+5,
    &xUsonicTask );

xTaskCreate( presenceTask,
    ( const char * ) "Presence",
    configMINIMAL_STACK_SIZE,
    NULL,
    tskIDLE_PRIORITY+6,
    &xPresenceTask );

xTaskCreate( velocimeterTask,
    ( const char * ) "Speed",
    configMINIMAL_STACK_SIZE,
    NULL,
    tskIDLE_PRIORITY+7,
    &xVelocimeterTask );

xTaskCreate( accelerometerTask,
    ( const char * ) "Tilt",
    configMINIMAL_STACK_SIZE,
    NULL,
    tskIDLE_PRIORITY+8,
    &xAccelerometerTask );

xTaskCreate( riskControlTask,
    ( const char * ) "RiskControl",
    1024,
    NULL,
    tskIDLE_PRIORITY+9,
    &xRiskControlTask );

xTaskCreate( buzzerTask,
    ( const char * ) "Buzzer",
    configMINIMAL_STACK_SIZE,
    NULL,
    tskIDLE_PRIORITY+2,
    &xBuzzerTask);

//Binary Semaphores
xSemaphoreBuzzer = xSemaphoreCreateBinary();
xSemaphoreAutopilot = xSemaphoreCreateBinary();

if(xSemaphoreBuzzer != NULL){
    /* The semaphore was created successfully and
    can be used. */
}
if(xSemaphoreAutopilot != NULL){
    /* The semaphore was created successfully and
    can be used. */
}

//Mutex Semaphores
xSemaphoreKB = xSemaphoreCreateMutex();
xSemaphoreAltitude = xSemaphoreCreateMutex();
xSemaphorePresence = xSemaphoreCreateMutex();
xSemaphoreSpeed = xSemaphoreCreateMutex();
xSemaphoreTilt = xSemaphoreCreateMutex();
xSemaphoreRisk = xSemaphoreCreateMutex();
xSemaphoreSD = xSemaphoreCreateMutex();

if(xSemaphoreKB != NULL){
    /* The semaphore was created successfully and
    can be used. */
}
if(xSemaphoreAltitude != NULL){
    /* The semaphore was created successfully and
    can be used. */
}
if(xSemaphorePresence != NULL){
    /* The semaphore was created successfully and
    can be used. */
}

```

### Anexo III - Función “main”

```
if(xSemaphoreSpeed != NULL){
    /* The semaphore was created successfully and
    can be used. */
}
if(xSemaphoreTilt != NULL){
    /* The semaphore was created successfully and
    can be used. */
}
if(xSemaphoreRisk != NULL){
    /* The semaphore was created successfully and
    can be used. */
}
if(xSemaphoreSD != NULL){
    /* The semaphore was created successfully and
    can be used. */
}

vTaskStartScheduler();

for( ; );
}
```