



ugr | Universidad
de **Granada**

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Título de mi proyecto

Subtitulo del Proyecto

Autor

Juan Ordóñez Cerezo

Directores

Encarnación Castillo Morales

Jose María Cañas Plaza



MI FACULTAD

—
Granada, Junio de 2017

Título de mi proyecto: Subtítulo del proyecto

Juan Ordóñez Cerezo

Palabras clave:

Resumen

Project Title: Project Subtitle

Juan Ordóñez Cerezo

Keywords:

Abstract

Yo, **Juan Ordóñez Cerezo**, alumno de la titulación Grado en Ingeniería de Tecnologías de Telecomunicación de la **Mi facultad**, con DNI 77143207B, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Juan Ordóñez Cerezo

Granada a 15 de Noviembre de 2018.

D. **Encarnación Castillo Morales**, Profesora del Departamento...

D. **Jose María Cañas Plaza** , Profesor del Departamento....

Informan:

Que el presente trabajo, titulado ***Título de mi proyecto, Subtítulo del proyecto***, ha sido realizado bajo su supervisión por **Juan Ordóñez Cerezo**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 26 de Junio de 2017.

Los directores:

Encarnación Castillo Morales Jose María Cañas Plaza

Agradecimientos

Poner aquí agradecimientos...

Índice general

1. Introducción	1
1.1. Introducción	1
1.1.1. Motivación y objetivos	1
1.1.2. Planificación (Diagrama de Gantt)	3
1.1.3. Metodología de trabajo	4
1.1.4. Estructura de la memoria	5
2. Estado del arte	7
2.1. Concepto FPGAs	7
2.1.1. Evolución y escenario	8
2.1.2. Arquitectura FPGA	9
2.1.3. Lenguajes de descripción hardware	10
2.1.4. Verilog	11
2.2. FPGAs libres	13
2.2.1. Evolución	13
2.2.2. IceZum Alhambra	15
2.2.3. IceStudio	17
2.3. Coexistencia Microcontrolador-FPGA	20
2.3.1. Diferencia microcontrolador-FPGA	20
2.3.2. Necesidad	21
2.4. Robótica educativa, motivaciones y necesidad.	23
2.5. Sensores, actuadores y sistema de control	24
3. Balancing Robot	29
3.1. Diseño del sistema	29
3.1.1. Descripción del problema	29
3.1.2. Descripción de la solución. (Diagrama de bloques alto nivel)	30
3.1.3. Física de un Balancing Robot	30
3.1.4. Integración IceZum Alhambra-Arduino Nano	30
3.1.5. Unidad de medida inercial	32
3.1.6. Motores	36
3.1.7. Control PID clásico.	37
3.1.8. Diseño estructura mecánica	37
3.1.9. Diseño PCB	37
3.1.10. Batería	39
3.2. Implementación del sistema	41
3.2.1. Integración IceZum Alhambra-Arduino Nano	41
3.2.2. Unidad de medida inercial MPU6050 en Arduino Nano	49
3.2.3. Control P	49
3.2.4. Control D	49
3.2.5. Controlador PWM	49

3.2.6. Controlador driver motor	49
3.2.7. Fabricación estructura mecánica	49
3.2.8. Fabricación PCB	49
3.2.9. Elección de materiales y coste del prototipo	56
3.3. Experimentos	56
3.3.1. VGA Module	56
3.3.2. Protocolo I2C	56
3.3.3. Controlador motor brushless	56
4. Cuadricoptero con vision artificial	57
5. Conclusiones y trabajo futuro	59
Bibliografía	61

Índice de figuras

1.1. Logo GitHub.	4
1.2. Commits GitHub.	5
1.3. Appear.in.	5
2.1. PLD vs FPGA	7
2.2. Ley de Moore	8
2.3. PLD vs FPGA	9
2.4. Paralelización de Procesos	12
2.5. Lattice iCE40HX1K	14
2.6. Tiny FPGA BX	14
2.7. BlackIce II	15
2.8. ico Board	15
2.9. IceZum Alhambra Board	16
2.10. IceZum Alhambra II Board	17
2.11. Ventana principal IceStudio.	18
2.12. Escritura I2C IceStudio alto nivel.	19
2.13. Escritura I2C IceStudio bajo nivel.	19
2.14. Funcionalidad FPGA y Micro-controlador.	20
2.15. Puertas lógicas después de una implementación hardware.	21
2.16. Sistema bipedo coexistencia Micro-FPGA.	23
2.17. Sensor CMOS para adquisición de imágenes.	26
2.18. Unidad de Medida Inercial.	26
2.19. Potenciómetro.	26
2.20. Motor DC.	27
3.1. SegWay comercial.	29
3.2. Representación péndulo invertido.	30
3.3. Separación de procesos micro-FPGA.	31
3.4. Pines hardware de coexistencia micro-FPGA.	32
3.5. Ángulos de Tait-Brain.	33
3.6. Unidad de Medida Inercial.	33
3.7. Unidad de Medida Inercial.	33
3.8. Unidad de Medida Inercial.	33
3.9. Trigonometría en sensor acelerómetro 2D.	34
3.10. Trigonometría en sensor acelerómetro 3D.	34
3.11. Pin headers para IceZum Alhambra II.	37
3.12. Conector GND y VCC.	38
3.13. Módulo MPU6050.	38
3.14. Jumpers para configuración i2c MPU6050.	39
3.15. Diagrama interno Arduino Nano.	41
3.16. Envío de ángulo usando el puerto serie.	42

3.17. Apariencia del módulo interfaz de Arduino Nano en IceStudio.	43
3.18. Diagrama de flujo de la interfaz para ArduinO.	45
3.19. Ordenación de la parte entera y decimal de un ángulo.	47
3.20. Diagrama de flujo modulo ordenación de bytes.	48
3.21.	50
3.22.	53
3.23.	54
3.24.	54
3.25.	55
3.26.	55

Índice de cuadros

2.1. Modo de operación de sensores.	25
3.1. Composición de capas en PCB.	52

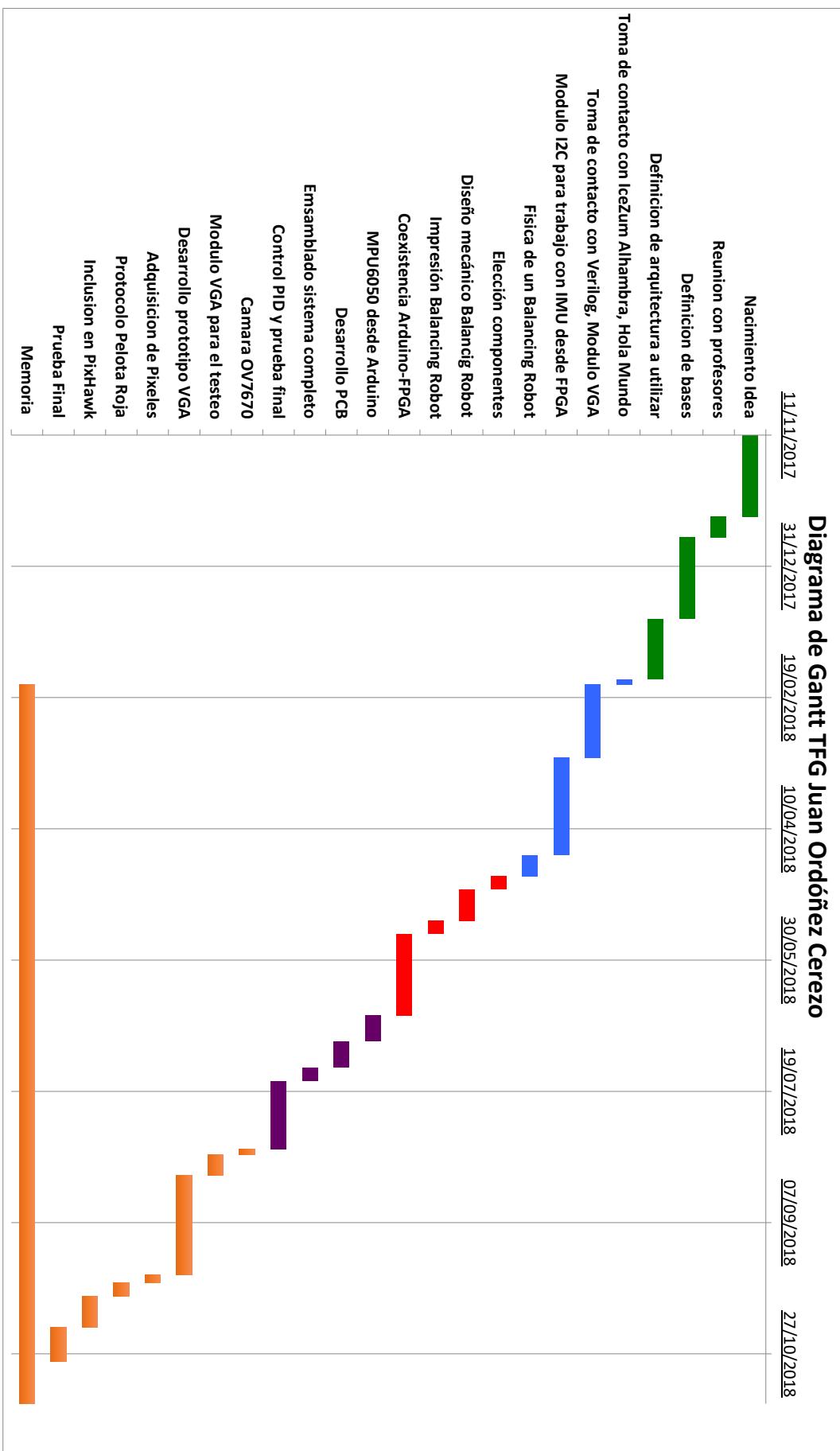
Capítulo 1

Introducción

1.1. Introducción

1.1.1. Motivación y objetivos

1.1.2. Planificación (Diagrama de Gantt)



1.1.3. Metodología de trabajo

Para explicar la metodología de trabajo seguida, se presenta la herramienta GitHub (figura 1.1).



Figura 1.1: Logo GitHub.

GitHub es una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones Git. GitHub aloja tu proyecto en un repositorio y brinda herramientas muy útiles para el trabajo en equipo.

Proporciona además la posibilidad de una Wiki para el mantenimiento de las versiones e información acerca de ellas.

En el presente proyecto GitHub ha sido usado como un contenedor, donde se ha ido subiendo todo de manera parcial, normalmente, cuando se obtenía una versión estable sobre algunas de las ramas.

De esta forma, y al ser abierto, cualquier persona ha podido seguir los avances de este, dudas, problemas, o incluso utilizar algunos de los módulos o material subidos.

El proyecto puede encontrarse en la siguiente URL:

<https://github.com/RoboticsURJC-students/2017-tfg-juan-ordonez>

Un ejemplo de la trayectoria de este proyecto se representa en la captura de pantalla de GitHub en la figura 1.2.

Commits on Oct 18, 2018
18102018033 jordonezcerezo committed 13 hours ago ✓
18102018002 jordonezcerezo committed 13 hours ago ✓
Commits on Oct 17, 2018
171020182258 jordonezcerezo committed 14 hours ago ✓
Commits on Oct 15, 2018
15102018 jordonezcerezo committed 3 days ago ✓
Commits on Oct 12, 2018
121020182035 jordonezcerezo committed 6 days ago ✓
Commits on Oct 11, 2018
111020182316 jordonezcerezo committed 7 days ago ✓
111020181731 jordonezcerezo committed 7 days ago ✓
111020181657 jordonezcerezo committed 7 days ago ✓
111020181420 jordonezcerezo committed 7 days ago ✓
111020181413 jordonezcerezo committed 7 days ago ✓

Figura 1.2: Commits GitHub.

Para el buen cumplimiento de los objetivos planteados en primera instancia, y teniendo en cuenta la diferente localización de los componentes del trabajo, se hizo necesario el planteamiento de reuniones semanales (normalmente los Viernes) donde se pusiese en común lo trabajado durante la semana y se fijasen los siguientes objetivos.

Para ello, se utilizo la herramienta se software libre llamada ”Appear.in”, la cuál ofrece videoconferencias entre varios usuarios al mismo instante (figura 1.3).



Figura 1.3: Appear.in.

1.1.4. Estructura de la memoria

Capítulo 2

Estado del arte

2.1. Concepto FPGAs

Una FPGA o (Field programmable Gate Arrays) es un dispositivo reconfigurable que puede ser eléctricamente programado para implementar una alta variedad de circuitos lógicos. Consiste en un "array" uniforme de estructuras lógicas programables que son interconectadas por una configurable red de enrutamiento. Un ejemplo de una red de enrutamiento puede verse en la figura 2.1

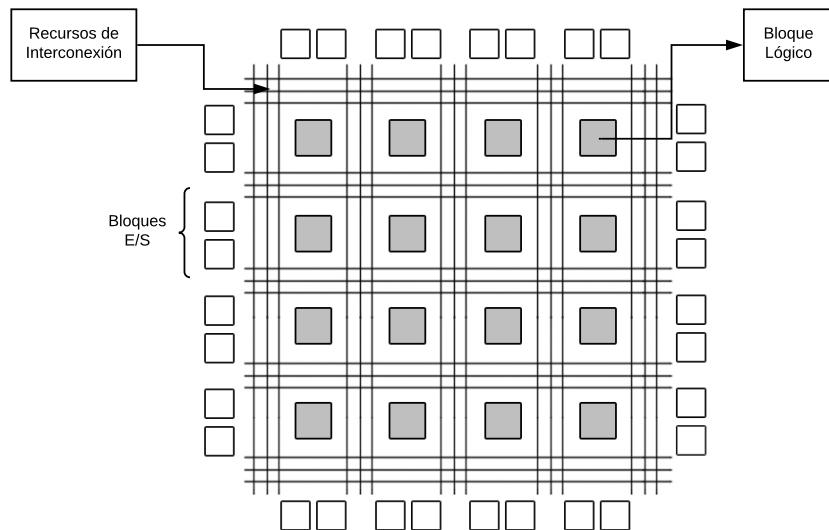


Figura 2.1: PLD vs FPGA

Lo lógica y las estructuras de interconexión pueden ser configuradas gracias a potentes herramientas CAD, las cuales permiten que el usuario final pueda definir esta interconexión física de bloques lógicos mediante lenguajes de descripción hardware (HDL). Algunos de los más conocidos son VHDL, Verilog o ABEL.

Si bien no es el más usado actualmente, a lo largo de esta memoria se realizará una breve introducción a Verilog, comentando y analizando sus principales características.

2.1.1. Evolución y escenario

Las FPGAs fueron inventadas en el año 1984 por Ross Freeman y Bernard Vonderschmitt, co-fundadores de Xilinx. Originalmente fueron diseñadas para servir como prototipos o demostraciones de la funcionalidad de circuitos digitales.

Las FPGAs conforman actualmente la máxima evolución de los PLDs (Programmable Logic Device), definidos estos como circuitos integrados en los que se pueden programar ecuaciones lógicas booleanas.

El revolucionario éxito de estos dispositivos programables puede ser atribuido a la flexibilidad en la implementación del diseño. Así, la capacidad de re-programar al instante la FPGA con varios circuitos sin costo adicional, promueve la re-utilización del dispositivo, permite una rápida verificación del diseño y por ello, reduce el gasto en la etapa de desarrollo.

Si bien actualmente no representan todo el sistema de un producto final, sus ventajas en las propiedades del diseño hacen que estén empezando a formar parte de una pieza importante en el desarrollo de un producto electrónico.

Para intentar entender el porqué de la importancia de dispositivos de implementación hardware sería adecuado introducir la Ley de Moore.

La ley de Moore expresa que aproximadamente cada dos años se duplica el número de transistores en un microprocesador. Como se puede ver en la figura 2.2, esta ley que formuló el cofundador de Intel E. Moore el 19 de Abril de 1965 no se aleja de la realidad:

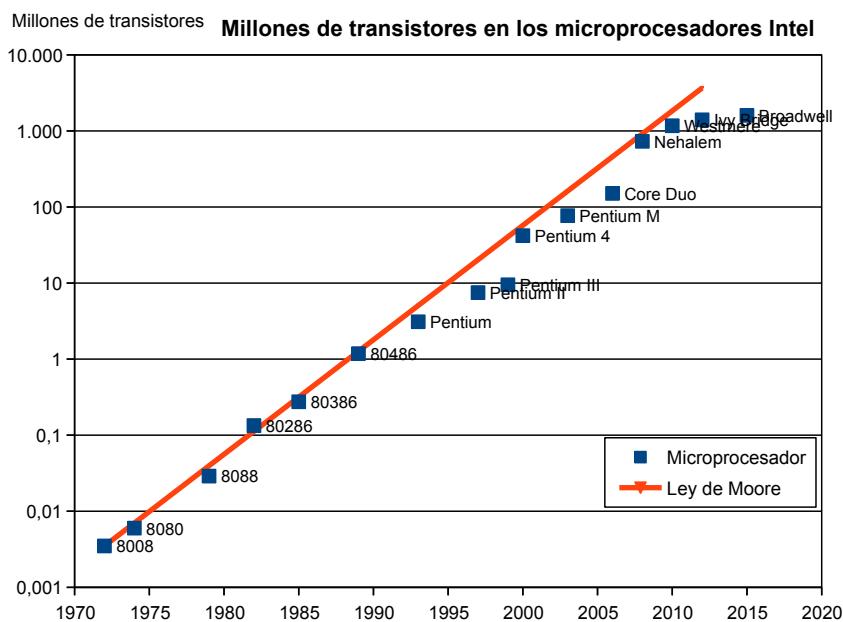


Figura 2.2: Ley de Moore

No obstante, tal y como ya han anunciado las principales empresas de microprocesadores actuales como son Intel y AMD en su hoja de ruta tecnológica para semiconductores, la ley de Moore llegará a su fin en 2021.

Después de esa fecha, no resultará económicamente eficiente reducir más el tamaño de los transistores de silicio. La predicción de la industria no es que simplemente su tasa de reducción será cada vez más lenta, sino que se va a detener definitivamente.

Es aquí donde la electrónica digital comenzará a tener un papel importante en los microprocesadores. Tanto es así que multinacionales dedicadas a la fabricación de procesadores como pueden ser Intel o AMD ya están trabajando en implementar esos comportamientos en FPGAs. Sus ventajas e inconvenientes serán presentadas en los capítulos 2.1.2 y 2.1.3.

2.1.2. Arquitectura FPGA

En un PLD, las interconexiones entre elementos ya está prefijada, y solo es posible habilitar o deshabilitar esta conexión.

Por el contrario, sobre FPGA estas interconexiones no están prefijadas, siendo el usuario final el que decide las interconexiones entre bloques lógicos (figura 2.3)

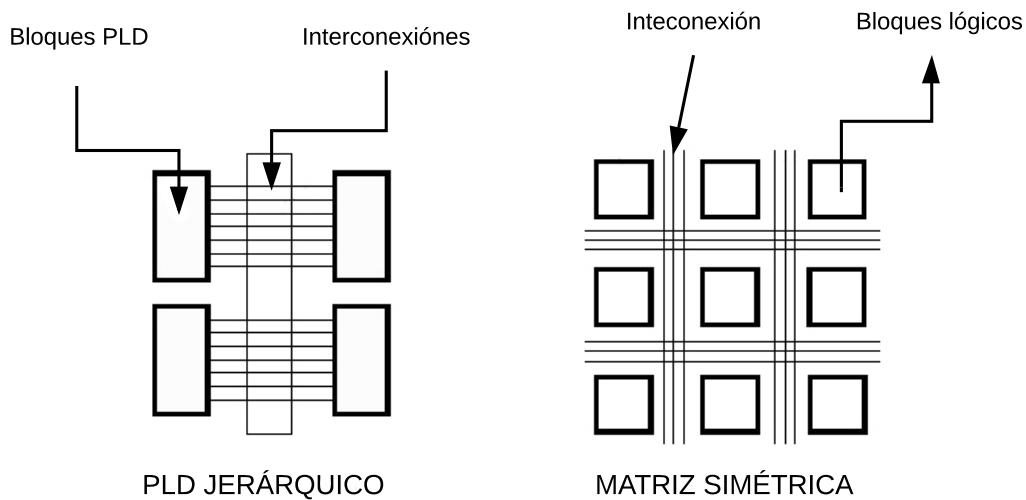


Figura 2.3: PLD vs FPGA

A la hora de elegir una FPGA para un proyecto determinado es importante tener en cuenta una de sus propiedades más importantes, el número de bloques lógicos.

El número de bloques lógicos determinan la capacidad del dispositivo y forma parte de una de las características mas limitantes de las FPGAS en la actualidad. Los bloques lógicos son independientes entre sí y pueden interconectarse para formar un módulo más complejos.

A continuación se introducirán los conceptos de granularidad fina y granularidad gruesa, que dará lugar al correcto entendimiento de la arquitectura en una FPGA.

Estos módulos más complejos realizan las operaciones básicas que en conjunto representan la función que operará en una FPGA. Se dice que la FPGA son dispositivos con una arquitectura avanzada por la densidad de sus componentes y por los diferentes caminos de interconexión

entre módulos. Así, de acuerdo al tipo de módulos lógicos que la conforman, encontramos dos derivaciones estructurales:

- Granularidad Gruesa.
- Granularidad Fina.

Los módulos lógicos en una arquitectura de granularidad gruesa son módulos grandes generalmente consistentes de una o más tablas de búsqueda y dos o más flip-flops. La tabla de búsqueda, también conocida como LookUp Table (LUT), actúa como una memoria donde se encuentra almacenada la tabla de verdad que representa la función lógica del circuito, así, en una LUT se puede implementar cualquier función deseable.

Por otra parte, una arquitectura de granularidad fina, está estructurada por una gran cantidad de módulos lógicos pequeños que realizan funciones relativamente simples. Cada módulo está compuesto de un circuito de dos entradas que realiza una función lógica determinada, o en algunos otros casos por un multiplexor. También suele contener algún flip-flop.

La FPGAs mas utilizadas actualmente suelen contar con tecnología de granularidad gruesa, que permite elevar el nivel de abstracción con respecto a las FPGA de granularidad fina.

El primer caso permite implementaciones menos detalladas ya que desde un nivel muy básico se tienen módulos más complejos y el hecho de utilizar LUT deja entrever que se pueden realizar diseños más grandes. A lo largo de este proyecto se trabajará con el primer caso, ejemplo de ello es la FPGA utilizada, IceZum Alhambra II.

El diseñador propone la función lógica a realizar a través de métodos de descripción hardware y define los parámetros de su problema. Esto se hace por medio de código programable, que puede ser un lenguaje de descripción hardware, los cuales son introducidos en el capítulo 2.1.3.

2.1.3. Lenguajes de descripción hardware

Un HDL(Hardware Description Language) es un lenguaje usado para modelar el funcionamiento de un bloque hardware en una forma textual.

Al igual que podemos encontrar diferencias entre los diferentes tipos de lenguajes de programación en cuanto a su sintaxis de codificación y sus métodos de simulación y síntesis, se observan ciertas diferencias entre HDL's.

Se tienen dos tipos diferentes de lenguajes de descripción hardware:

- Bajo modelado. No permiten la jerarquización de módulos y son capaces de realizar solo descripciones simples.
- Alto modelado. Son los mas utilizados actualmente, permiten diseñar sistemas más complejos. Ejemplos de alto modelado son Verilog (Verify Logic) y VHDL (VHSIC Hardware Description Language).

Este trabajo se basará en Verilog por permitir un nivel más alto de implementación que VHDL, como será analizado en la sección 2.1.4 .

Ambos lenguajes comparten algunas características comunes, como el soporte para cualquier nivel de modelado y abstracción, y que cada elemento de diseño tiene una interfaz bien definida, para permitir la conexión rápida con otros elementos lógicos.

2.1.4. Verilog

Debido a su utilización a lo largo de este proyecto, se analizarán algunas características importantes cuyo conocimiento será básico para entender el código.

Tipos de datos

Existen dos tipos de datos en Verilog los cuáles tienen que ser entendidos para poder alcanzar la funcionalidad buscada:

- reg: Representan variables con capacidad para almacenar información.
- wire: Representan conexiones entre componentes, no tienen capacidad de almacenamiento.

Implementación en módulos

En la mayoría de los casos y para no perder el nivel de abstracción, un proyecto en Verilog suele estar compuesto por un conjunto de módulos los cuáles forman una funcionalidad completa, y cada uno de ellos, una específica.

Las características principales de la implementación digital por módulos son:

- Cada módulo dispone de una serie de entradas y salidas cuya función principal es interconectar otros módulos, aunque puede no tener entradas y salidas.
- No existen variables globales.
- Cada módulo puede describirse de forma arquitectural o de comportamiento.

Esta implementación por módulos Verilog permite la configuración del nivel de abstracción deseado por el usuario final.

El desarrollo de módulos de funcionalidad específica puede tener sus ventajas e inconvenientes dependiendo de su uso final.

Si se quieren reutilizar para otros flujos de trabajo (ejemplo: sumador 8 bits), es bueno tener módulos muy definidos según su funcionalidad (ejemplo: sumador de 2 bits). Sin embargo, la polarización no tiene porque ser imprescindible, si bien aconsejada.

Paralelización de Procesos

Una de las características mas importantes que diferencia a Verilog del resto de lenguajes procedurales es la posibilidad de ejecutar varios procesos en paralelo, aspecto fundamental en el lenguaje, y el cuál le brinda gran parte de las ventajas de usar implementación hardware.

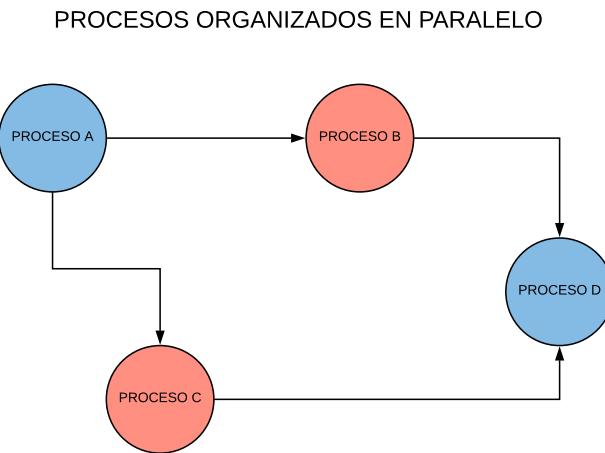


Figura 2.4: Paralelización de Procesos

Toda la descripción del comportamiento en Verilog debe declararse dentro de un proceso, estos pueden ser dos tipos:

- Initial: Este tipo de procesos se ejecutan sólo una vez comenzando su ejecución al inicio, y por tanto no existen retardos. Este proceso no es sintetizable, es decir, no puede ser utilizado en una descripción RTL.
- Always: Este tipo de procesos se ejecuta continuamente a modo de bucle, y como su propio nombre indica, está continuamente ejecutándose. Este proceso si que es sintetizable y es controlado por la temporización o por eventos. Si dicho bloque se ejecuta por más de un evento, dicho conjunto se denomina lista sensible.

Estructuras de control

Al igual que los lenguajes de tipo procedural, Verilog dispone de una serie de estructuras de control:

- if - else
- Case. Es una de las estructuras de control mas utilizadas a lo largo de este proyecto, permite la generación de máquinas de estados
- For
- While
- Forever
- Wait

Asignación continua

Mediante la asignación continua se puede modelar lógica combinacional, es decir, no se necesita una lista de sensibilidad para realizar la asignación. Sólo puede ser declarada fuera de cualquier proceso.

Asignación procedural

A las variables se le asigna un valor dentro de un proceso always o initial, el tipo de variable a la que se le asigna el valor puede ser de cualquier tipo.

2.2. FPGAs libres

2.2.1. Evolución

Muchos lenguajes de implementación hardware así como su arquitectura de FPGA utilizada están ligados a importantes empresas tales como Xilinx, Intel (anteriormente Altera), etc, y poder trabajar con ellos requiere un elevado presupuesto.

Lo anterior por lo tanto, lleva a que no muchas empresas ni particulares puedan beneficiarse de las ventajas de la utilización de FPGAs y a su vez, que el avance tecnológico sea aún más lento. Una de las claves del éxito de empresas como Arduino no es más que la comunidad de gente que existe detrás creando nuevas librerías, componentes, etc. Todo ello a su vez gracias al bajo precio de sus productos, y a la posibilidad de encontrar todo el hardware y software en la web.

Para entender el nacimiento de las FPGAs libres es importante conocer qué es el bytestream. Un bytestream es una secuencia de bytes que se utiliza en telecomunicaciones y computación. Frecuentemente, el término bytestream es utilizado para describir la configuración a ser cargada en una FPGA. Este formato detallado de flujo de bits para una FPGA particular es típicamente propietario del proveedor de FPGA.

Es por ello que Clifford Wolf decidió interpretar el bytestream del modelo Lattice iCE40 y desarrollo la herramienta IceStorm.

IceStorm se desarrolló como software de traducción de Verilog (lenguaje de descripción en FPGAs, sección 2.1.4) al bytestream. Esta traducción fue posible gracias a la ingeniería inversa, esto es, no se le da el uso habitual, sino el inverso.

Ya no se depende de ningún fabricante y todo el conocimiento además está disponible. A partir de estas herramientas se puede crear cualquier interfaz o cualquier aplicación que no haya sido prevista por el fabricante.

Sólo las FPGAs de Lattice iCE40 (modelos HX1K-TQ144 y HX8K-CT256) son hasta el momento con las que se pueden trabajar (figura 2.5), pero al ser un proyecto libre, muchas personas ya están trabajando en aumentar las posibilidades.

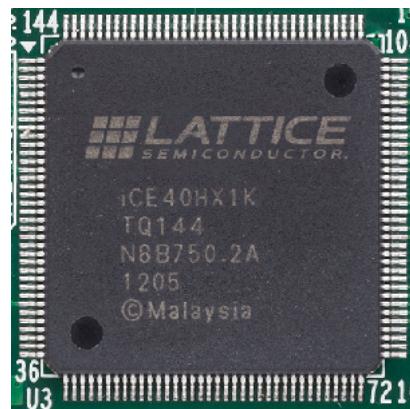


Figura 2.5: Lattice iCE40HX1K

Algunos ejemplos de FPGAs ya disponibles para ser usadas se exponen en las figuras 2.6, 2.7, 2.8



Figura 2.6: Tiny FPGA BX

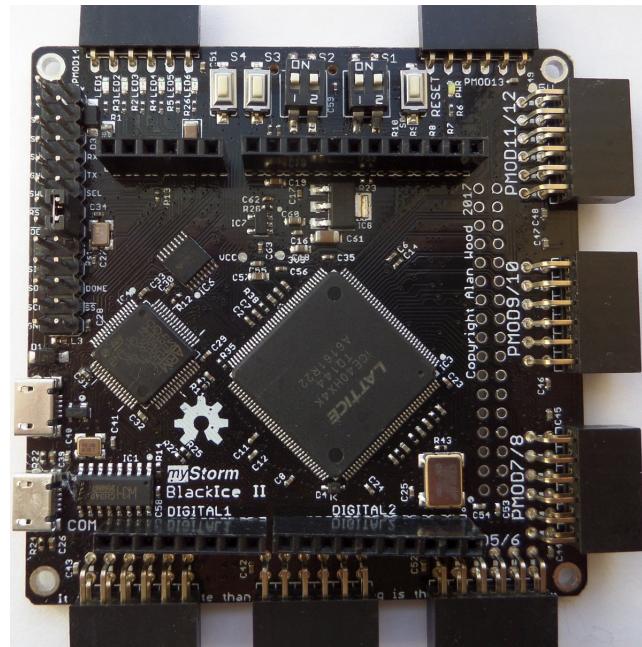


Figura 2.7: BlackIce II



Figura 2.8: ico Board

2.2.2. IceZum Alhambra

Para este trabajo se ha optado por trabajar con la IceZum Alhambra, la cuál ha sido íntegramente diseñada y ensamblada en España.

Es una FPGA libre y compatible con IceStudio (el cuál se verá a continuación). Alguna de sus características más importantes pueden ser:

- Placa FPGA de desarrollo iCE40HX1K-TQ144 de la empresa Lattice.
- Open hardware.
- Compatible con IceStorm toolchain.
- Compatible con shields de Arduino Uno.

- 12MHz Oscilador.
- Interruptor ON/OFF para activar o desactivar los pines digitales.
- 20 Input/output 5v pines.
- 8 Input/Output 3.3V pines
- USB micro-B para programar la FPGA desde el pc.
- Botón de reset.
- 8 leds de propósito general.
- TX/RX Leds
- 4 entradas analógicas disponibles a través de i2c.

Es conocido que existen placas con mejores características, pero el hecho de que sea Open Hardware y que se pueda implementar con IceStudio, ha llevado a la elección final de esta placa para el desarrollo del presente proyecto.



Figura 2.9: IceZum Alhambra Board

Un punto a tener en cuenta para el desarrollo de hardware con esta FPGA es su limitada memoria de 1K lo que ha supuesto una limitación importante en el desarrollo. Es por ello que también se utilizó para el proyecto la nueva versión de la IceZum Alhambra, IceZum Alhambra II, la cual aún no estaba en el mercado al inicio del proyecto y que conlleva algunas mejoras, como la ampliación de 8K en su memoria, la mejora del bus de datos i2c, la posibilidad de alimentación mediante batería LIPO, etc.

La IceZum Alhambra II es la representada en la figura 2.10

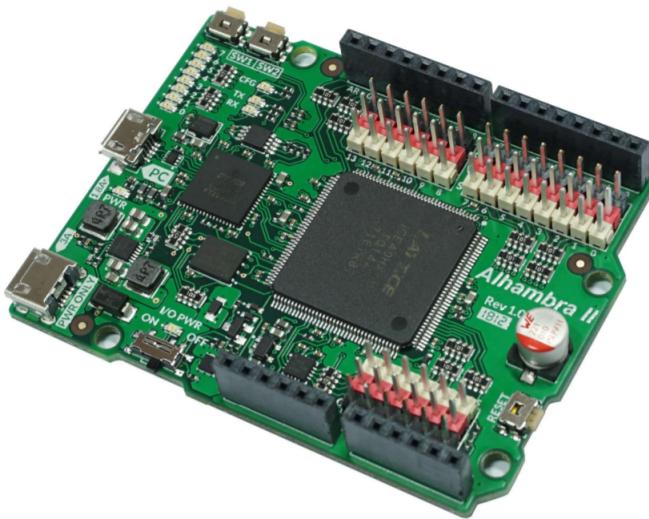


Figura 2.10: IceZum Alhambra II Board

2.2.3. IceStudio

Los lenguajes HDL suelen tener una curva de aprendizaje difícil, debido esto en gran medida al nivel de abstracción tan bajo necesario para diseñar un sistema en concreto. Es necesario conocer las características hardware de nuestro sistema para poder trabajar con este tipo de lógica.

Como se ha desarrollado anteriormente, algunos fabricantes proporcionan herramientas comerciales para programar sus propios FPGA. Si bien en la actualidad son entornos complejos, cuentan con una gran cantidad de herramientas y funcionalidades. Lamentablemente la mayoría de ellos no son gratuitos y están unidos a la arquitectura de un único fabricante.

Con la evolución de las FPGAs han empezado a aparecer lenguajes que permiten un mayor nivel de abstracción. Además, también han aparecido herramientas centradas en la implementación gráfica. Un ejemplo de este tipo de implementación es LabVIEW FPGA o IceStudio.

IceStudio es un proyecto Open Source desarrollado por Jesús Arroyo Torrens y sobre el que nos basaremos a lo largo del presente documento.

IceStudio es IDE gráfico para FPGAs libres y esta construido sobre el proyecto IceStorm. El proyecto IceStorm tiene como objetivo la ingeniería inversa y la documentación del formato bitstream de la FPGA Lattice iCE40 (aunque más adelante fueron surgiendo algunas más). Proporciona herramientas simples para analizar y crear archivos de flujo de bits, esto es, el más bajo de nivel de implementación para una FPGA.

Para acercar al lector al conocimiento y funcionamiento de IceStudio se irán incorporando a lo largo del documento una serie de capturas de pantalla representativas para que no se pierda la visión de lo que se está haciendo. Por ejemplo, la ventana principal de IceStudio y sobre la que se desarrollará todo lo demás tendrá la siguiente apariencia:

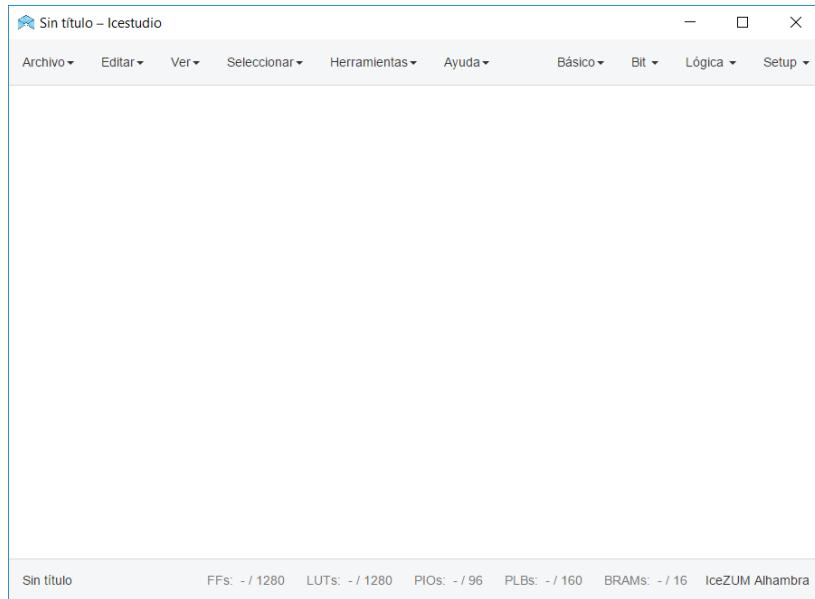


Figura 2.11: Ventana principal IceStudio.

El hecho de que IceStudio sea un editor gráfico puede hacer pensar que el nivel de abstracción podría ser más alto de lo deseado, pero lo cierto es que este nivel es configurable. Es el usuario final el que decide con qué nivel de abstracción se trabaja, siendo necesario para eso una amplia biblioteca de módulos como veremos a continuación. Para poder explicar la potencia de IceStudio, se procederá con un ejemplo;

El módulo que se presenta en la imagen 2.12 es una escritura normal de i2c, en la cuál se parametriza la dirección del esclavo y la dirección que se quiere leer (más adelante se explicará con más detalle).

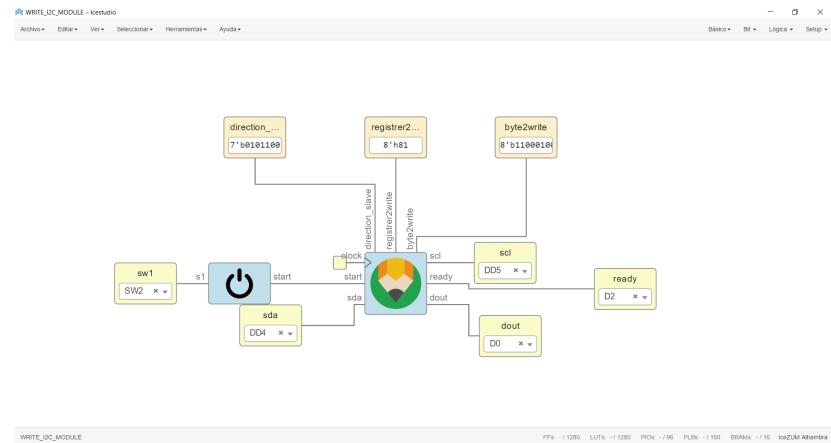


Figura 2.12: Escritura I2C IceStudio alto nivel.

Así, si una persona no experimentada con este tipo de código y cuyo fin no es entenderlo quiere hacer uso de eso no deberá de bajar mucho de nivel.

No obstante, existe la posibilidad de que se requieran cambiar valores como la frecuencia de reloj, el modo de operación i2c...etc. Para ello podemos bajar de nivel e introducirnos en el módulo en cuestión, en este caso, haciendo doble clic, como aparece en la figura 2.13:

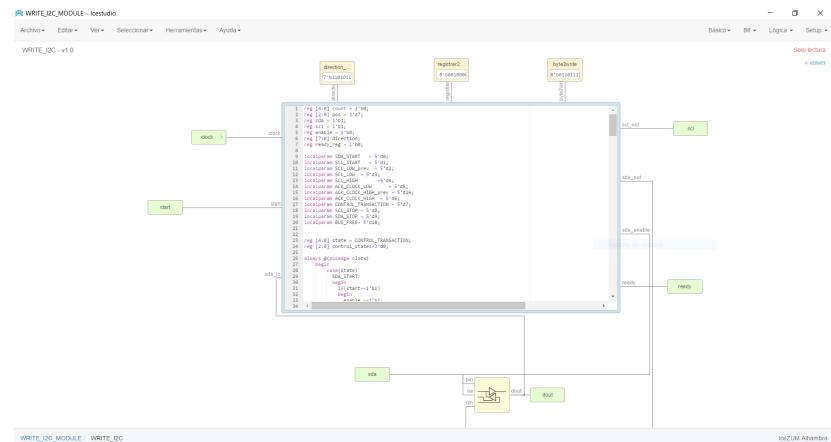


Figura 2.13: Escritura I2C IceStudio bajo nivel.

Se podría decir entonces que se ha bajado un nivel más de abstracción, pudiendo entrar ahora en detalles hardware más específicos si fuese necesario.

En la anterior demostración se ha podido ver una de las ventajas de IceStudio. La modularidad permite configurar el nivel de abstracción. Para ello hace falta una biblioteca de módulos, algunos de los cuales serán desarrollados a lo largo de este trabajo, otros de ellos, están siendo trabajados, y pueden encontrarse en el siguiente enlace:

Foro de Google con discusión sobre temas y módulos para IceStudio.

2.3. Coexistencia Microcontrolador-FPGA

2.3.1. Diferencia microcontrolador-FPGA

En un principio puede parecer que un procesador y FPGA son dispositivos similares porque ambos pueden realizar ciertas tareas pre-configuradas. Lo cierto es que al profundizar se pueden encontrar mas diferencias que similitudes. Ambos son capaces de implementar una función de transferencia, pero la forma en la que lo hacen es diferente para cada uno de ellos.

Así, podríamos ver las FPGA y los microcontroladores como una caja negra en la que tenemos unas entradas y ciertas salidas.

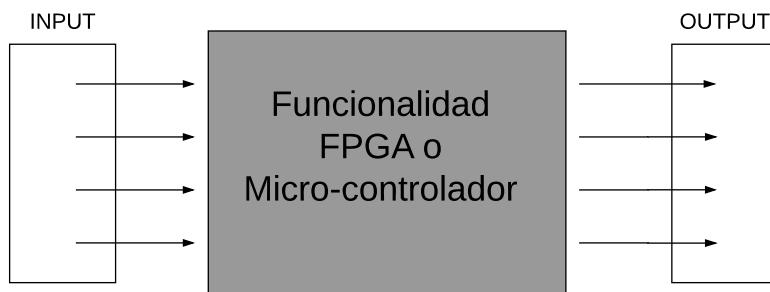


Figura 2.14: Funcionalidad FPGA y Micro-controlador.

Para comprobar de forma resumida como implementan de manera diferente esa función de transferencia, se explicará brevemente la forma de trabajar con un procesador.

Un procesador contiene una serie de instrucciones que realizan operaciones sobre un conjunto de operaciones binarias (sumar, incrementar, leer y escribir de la memoria). Dependiendo del tipo de procesador y de su arquitectura tenemos más o menos instrucciones asociadas, siendo este aspecto uno de los mas importantes que determinan su rendimiento.

Arquitectura procesador

Se dispone de una serie de registros, una memoria para almacenar la información y una pila de instrucciones, que contiene el programa que va a ejecutarse en código máquina, además de un reloj.

Su modo de funcionamiento a alto nivel; en cada ciclo de reloj el procesador lee de su pila de instrucciones los valores necesarios, llama a la instrucción oportuna y ejecuta un determinando cálculo.

Como se argumentó en el capítulo 2.1.2, al implementar un diseño lógico en una FPGA, se está modificando una matriz de conexiones físicas. Modificando esa matriz de conexiones se pueden implementar diferentes bloques de funcionalidad, es decir, se podría ver como varias funciones de transferencia en un mismo sistema.

En la imagen 2.15 se representa un ejemplo real de como están implementadas las conexiones físicas de puertas lógicas en una FPGA, y de como eso permite tener módulos independientes unos de otros. Además, note el problema de la memoria en una FPGA, siendo ésta el número total de puertas lógicas físicas que pueden ser utilizadas;

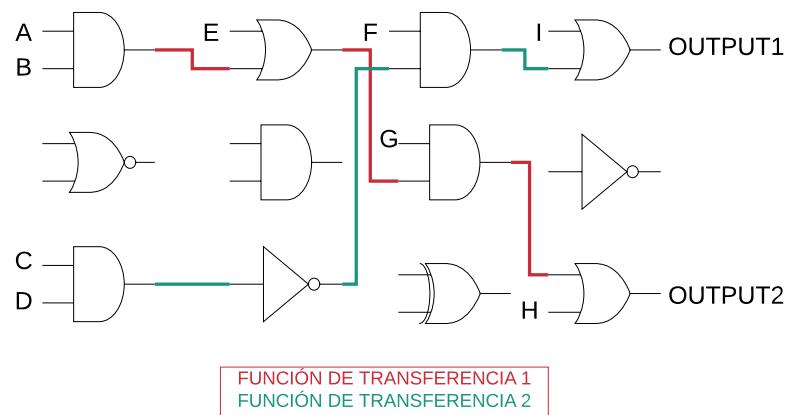
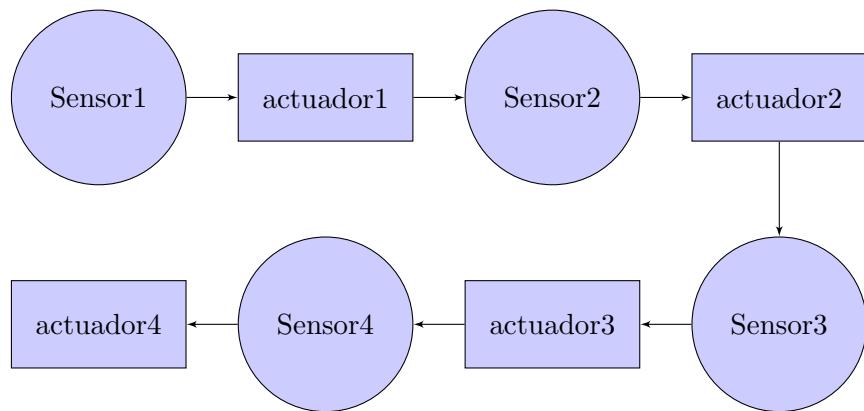


Figura 2.15: Puertas lógicas después de una implementación hardware.

2.3.2. Necesidad

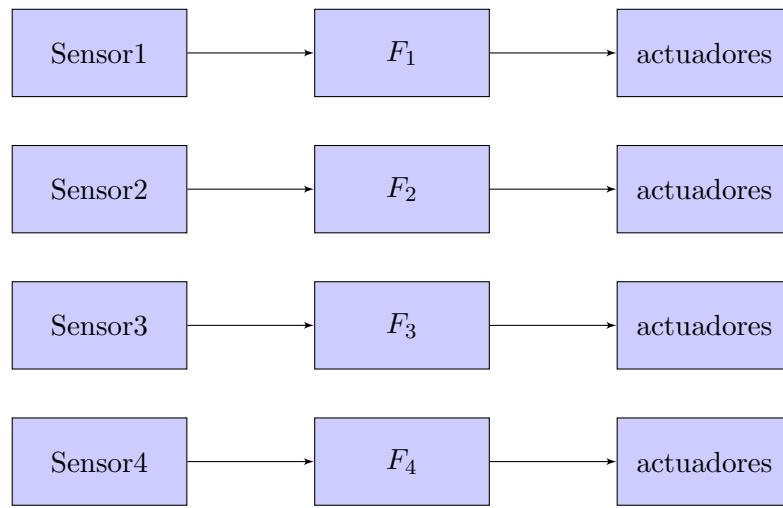
Se imagina un ejemplo real el que se quieren monitorizar con exactitud 4 diferentes sensores provenientes del exterior, de una manera exacta, al mismo tiempo y a la velocidad del reloj, siendo necesaria además una posterior actuación por parte del sistema. El diagrama de bloques del flujo de trabajo en un procesador que implemente lo anterior podría parecerse al siguiente:



Así, el usuario final de este sistema, deberá de manera cíclica comprobar cada sensor y su posterior actuación, dejando de cumplir entonces las especificaciones de tiempo.

Si se trabajase con interrupciones se configurarían las diferentes interrupciones externas o se podrían usar ejecutivos cíclicos para acercarse a esos requerimientos finales. No obstante, cualquiera de estas soluciones, no dejan de ser una aproximación.

En cambio, con el uso de una FPGA, el diagrama de bloques del flujo de trabajo tendría el siguiente aspecto:



Se implementarían diferentes funciones de transferencia para cada uno de los bloques a desarrollar, pudiendo ejecutarse estos en paralelo.

No obstante, no siempre es necesario un ejecutivo en paralelo, y no solo puede no ser necesario sino que podría ser perjudicial. Cuando un sistema debe ser secuencial, ¿porqué utilizar una implementación de naturaleza paralela?.

Es muy común tener sistemas donde conviene poder implementar ambos tipos de funcionamiento, por eso una coexistencia FPGA/Micro-controlador podría ser suficiente para adaptarse a los requerimientos.

En la imagen 2.16 se puede ver como ejemplo real de un sistema bípedo, el cuál se explicará con mas detalle en los siguientes capítulos:

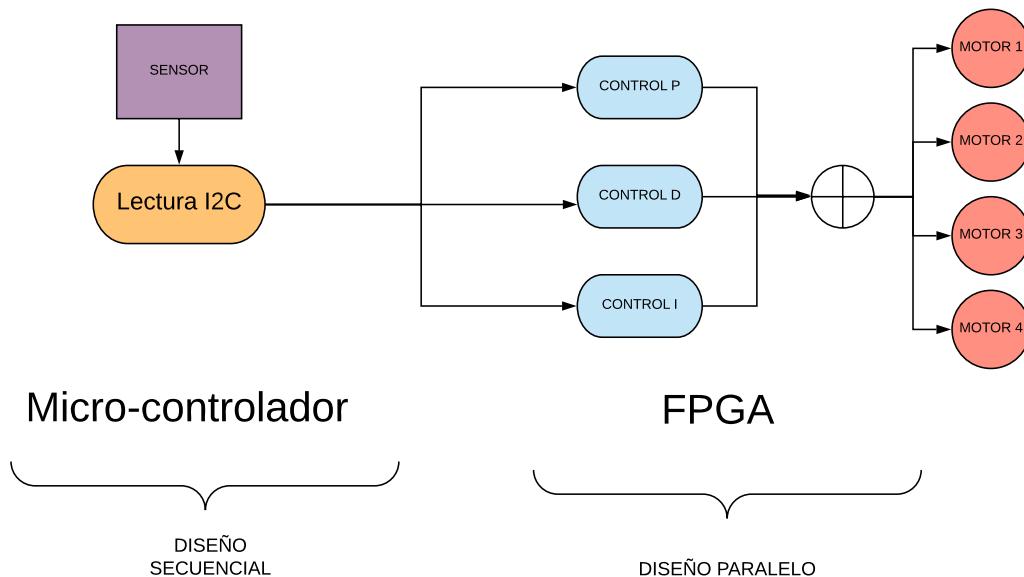


Figura 2.16: Sistema bipedo coexistencia Micro-FPGA.

2.4. Robótica educativa, motivaciones y necesidad.

La robótica se puede considerar sin duda como una de las áreas tecnológicas con mas auge de la actualidad y basada en el estudio de los robots, que son sistemas compuestos por mecanismos que le permiten hacer movimientos y realizar tareas específicas, programables e inteligentes.

Dependiendo de la aplicación por tanto, la robótica puede extenderse y generar beneficios no sólo en la industria sino también en las aulas de clase, posibilitando la aparición de nuevos sistemas de aprendizaje.

Además en un mundo cuyo futuro va encaminado a la utilización de robots para cualquier actividad, el acercamiento desde las aulas con estos sistemas posibilita su desarrollo tecnológico a una edad temprana, siendo más fácil su integración a una edad adulta.

Algunos beneficios de la robótica educativa son expuestos a continuación:

- Impulsa la iniciativa y la creatividad
- Mayor sociabilización
- Incentiva el pensamiento algorítmico y matemático
- Trabajo en equipo
- Resolución de problemas

- Aprendizaje activo
- Aumento de la autoestima

No obstante, para que la integración en las aulas de la robótica educativa sea aún más fácil, los sistemas deben cumplir algunas características:

- No es recomendable la integración a alto nivel tecnológico.
- Los robots deben ser de carácter amigable y divertido.
- Los entornos de programación no deben ser complejos, y aunque su funcionalidad esté algo limitada, tiene que llamar la atención del alumno y hacerle sentir cómodo.
- Es importante que el robot cuente con una serie de sensores y actuadores, unas entradas y salidas para que los resultados sean visuales.

Después de haber analizado cuáles son las ventajas de la electrónica digital, resulta conveniente poder acercar estos dos campos de conocimiento; electrónica digital-robótica educativa.

Si la electrónica digital y el mundo de los robots están llamados a formar parte de nuestras vidas en un futuro cercano, la necesidad de un acercamiento a estos dos conceptos a edades tempranas es básico para un correcto avance de la tecnología.

Con esta idea nace IceStudio, hacer amigable la electrónica digital para que los más pequeños puedan hacer uso de ella, y además, cumple todas las características antes expuestas.

2.5. Sensores, actuadores y sistema de control

Antes de comenzar con el desarrollo del proyecto, es importante tener claro los conceptos de sensores, actuadores y elementos del sistema de control, los cuales forman parte de cualquier plataforma robótica móvil.

Cualquier instalación de control, ya sea robótica o inmóvil, está compuesta por tres componentes fundamentales:

- Sensores
- Actuadores
- Sistema de control

Los sensores son dispositivos que recogen información del mundo que nos rodea y lo transforma en señales eléctricas que puedan ser asimiladas por un sistema de control.

Así, el sistema de control recibe información del entorno sobre el que queremos realizar algún tipo de acción por medio de los sensores, es la función de transferencia del sistema, a partir de unas entradas de tipo conocido, son generadas unas salidas, normalmente, dependientes de las entradas.

Estas salidas son denominadas actuadores, que son dispositivos que, siguiendo los parámetros dados por el sistema de control realizan acciones que repercuten en el entorno.

Ejemplo: Un sensor indica al sistema de control la intensidad lumínica de nuestra habitación, el sistema de control reconoce que el nivel no es el adecuado para la lectura, y activa un actuador, en este caso, una luz, para contrarrestar ese nivel.

A la hora de elegir un determinado sensor, es importante conocer su modo de operación, para poder configurar o mantener sistemas que los incorporen. Existen diferentes tipos de sensores según:

- Tipo de salida:
 - Analogicos
 - Binarios
 - Digitales
- Estructura interna:
 - Pasivos
 - Activos
- Tipo de parámetros capaces de detectar

En tabla 2.1 se muestran algunos de los más interesantes para el desarrollo del proyecto:

Magnitud	Transductor	Característica
Posición lineal y angular	Potenciómetro	Analógica
	Encoder	Digital
	Sensor Hall	Digital
Velocidad lineal y angular	Dinamo tacométrica	Analógica
	Encoder	Digital
	Detector inductivo	Digital
	Servo-inclinómetros	A/D
	RVDT	Analógica
	Giróscopo	Digital
Aceleración	Acelerómetro	Analógico
	Servo-acelerómetros	
Visión artificial	Cámaras de video	Procesamiento digital
	Cámaras CCD o CMOS	Procesamiento digital

Tabla 2.1: Modo de operación de sensores.

Otra clasificación posible es la del ámbito de aplicación, es decir, donde y para qué son usados estos sensores.

De entre las características técnicas más importantes de un sensor, y haciendo una introducción al posible vocabulario que se utilizará, encontramos:

- Rango de medida: dominio en la magnitud medida en el que puede aplicarse el sensor.
- Precisión: es el error de medida máximo esperado.

- Offset o desviación de cero: valor de la variable de salida cuando la variable de entrada es nula.
- Sensibilidad de un sensor: suponiendo que es de entrada a salida y la variación de la magnitud de entrada.
- Resolución: mínima variación de la magnitud de entrada que puede detectarse a la salida.
- Derivas: son otras magnitudes, aparte de la medida como magnitud de entrada, que influyen en la variable de salida.

Por lo general, la señal de salida de estos sensores no es apta para su lectura directa y a veces tampoco para su procesado, por lo que se usan circuitos de acondicionamiento. Un ejemplo de algunos sensores se representan en las figuras 2.17, 2.18 y 2.19.

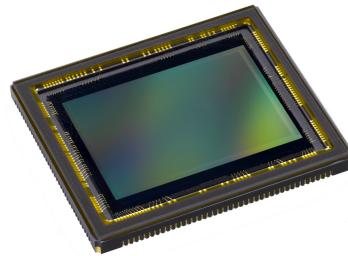


Figura 2.17: Sensor CMOS para adquisición de imágenes.

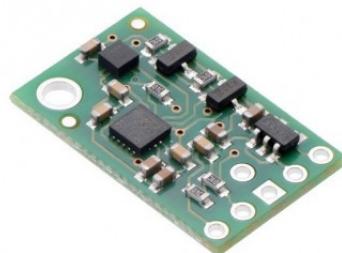


Figura 2.18: Unidad de Medida Inercial.



Figura 2.19: Potenciómetro.

Los actuadores son dispositivos que permiten al sistema de control ‘actuar’ sobre el ‘mundo real’ para realizar las acciones deseadas. Uno de los actuadores mas conocidos son los motores, el cuál será muy utilizado a lo largo de esta aplicación.



www.pololu.com

Figura 2.20: Motor DC.

Sistemas de control hay de muchos tipos en relación de la aplicación la cual se quiere desarrollar, normalmente se elige un micro-controlador como sistema de control y se programa la función de transferencia a ser realizada.

En el caso de este proyecto, y para poder adquirir por un lado las ventajas de una FPGA y por otro las ventajas de un micro-controlador (sección 2.3), se utilizará tanto arduino para las tareas secuenciales como la IceZum Alhambra para las tareas que puedan ser paralelizadas.

Capítulo 3

Balancing Robot

3.1. Diseño del sistema

3.1.1. Descripción del problema

En el presente capítulo se pretende abordar el problema del péndulo invertido mediante la utilización de una FPGA. Para ello, en los respectivos capítulos se tratará la física de un Balancing robot, el cálculo de su estructura, los sensores y actuadores utilizados, el sistema de control y el diseño y fabricación de una PCB que resuelva de manera más adecuada algunos problemas de los anteriormente planteados. Se utilizará una comunicación entre FPGA-Micro-controlador y se dará una versión más global del sistema propuesto, con un diagrama de bloques general.

Para entrar el contacto con el problema a resolver, se enunciará brevemente el problema del péndulo invertido, cuya solución ha dado lugar a muchas herramientas muy famosas en la actualidad, una de ellas, por ejemplo, el llamado *SegWay* (3.1).



Figura 3.1: SegWay comercial.

Péndulo: Es un sistema físico que puede oscilar bajo la acción gravitatoria u otra característica física (elasticidad, por ejemplo) y que está configurado por una masa suspendida de un punto o de un eje horizontal fijos mediante un hilo, una varilla, u otro dispositivo que sirve para medir el tiempo.

Así, y como el lector podrá imaginar, un péndulo invertido tendrá el aspecto de la figura 3.2.

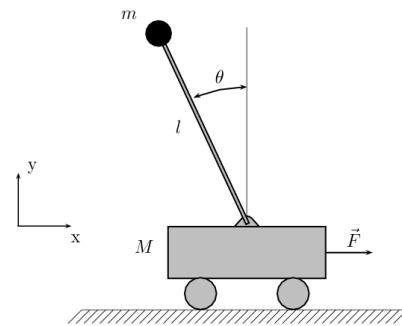


Figura 3.2: Representación péndulo invertido.

Consiste en un péndulo donde el centro de masas se encuentra situado por encima del punto o eje de balanceo. Como cabe esperar, esta disposición dota al sistema de una inestabilidad estática. Recordamos que un sistema es estable cuánto más cercano está del plano horizontal de apoyo su centro de gravedad.

El fundamento de este proyecto consistirá por tanto en intentar corregir esta inestabilidad, y forma parte de uno de los problemas más famosos en cuanto a teoría de control y dinámica de sistemas.

3.1.2. Descripción de la solución. (Diagrama de bloques alto nivel)

En los sucesivos capítulos se irá profundizando en cada uno de los bloques anteriores.

3.1.3. Física de un Balancing Robot

3.1.4. Integración IceZum Alhambra-Arduino Nano

Una integración entre un micro-controlador y FPGA permite diferenciar tareas secuenciales y paralelas, asignando cada proceso o bien al micro-controlador si necesariamente tiene que ser secuencial o bien a la FPGA si el proceso puede ser paralelizado y obtener con ello algunas ventajas.

Hay varias opciones para hacer una integración Micro-Controlador-FPGA:

- Emular el comportamiento de un micro-controlador en una FPGA.
- Coexistencia física de una FPGA y micro-controlador creando una comunicación entre cada una de ellas.

En este proyecto se ha elegido la segunda como opción por no contar con los recursos suficientes para llevar a cabo la primera, a pesar de que esta sería la más adecuada en cuanto ahorro de recursos y facilidad de uso.

Así, el sistema elegido deberá contar con algún modelo de comunicación que permita una integración como si de un solo sistema se tratase, aspecto sobre el que se basa este capítulo. Para claridad por parte del lector, un esquema básico del objetivo final se representa en la figura ??.



Figura 3.3: Separación de procesos micro-FPGA.

En este caso la comunicación será solo uni-direccional, el micro-controlador enviará información a la FPGA sobre el ángulo actual del objeto en cuestión con el objetivo de que la FPGA analice y actué a partir de ese ángulo.

Si no se conociese el ángulo actual, el sistema final no sabría reconocer hacia qué dirección ni con qué velocidad debe corregir este error (recordamos aquí que el objetivo final es conseguir una estabilización).

Existen dos tipos de comunicación posibles para este propósito:

- Comunicación serie: Es una comunicación secuencial, se envían los bits uno a uno, secuencialmente y haciendo uso solo de un bus de datos.
- Comunicación en paralelo: Todos los bits de cada símbolo se envían a la vez.

Para hacer uso de la comunicación en paralelo se necesitan tantos canales como bits tenga la información a transmitir (si se quiere enviar un byte se debería hacer uso de un total de 8 canales, correspondientes a cada bit). En este caso habría que utilizar 8 pines de la FPGA para poder llevar a cabo este tipo de transmisión. Por ello, y a pesar de que la comunicación paralela es mas rápida que en serie, se elige la primera como opción más conveniente.

El tipo de comunicación serie desarrollada podría asemejarse a un protocolo SPI, aunque sólo tiene capacidad para datos en una dirección. El esquemático de esta comunicación desarrollada es el expuesto en la figura 3.4 y su desarrollo será explicado en la sección 3.2.1.

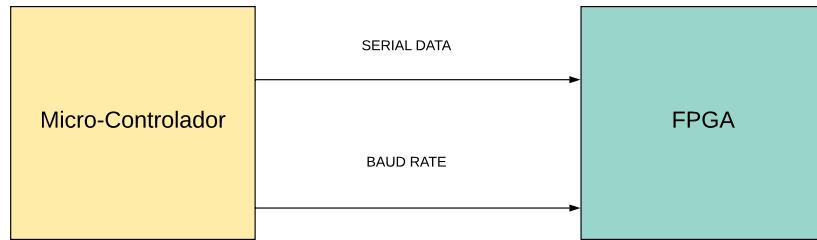


Figura 3.4: Pines hardware de coexistencia micro-FPGA.

El sistema general cuenta con dos conexiones:

- Una línea de datos para el envío de la información.
- Una linea de reloj para que la FPGA pueda obtener en todo momento la velocidad de la información.

Un posible ejemplo de una comunicación en serie de un byte de datos podría ser el siguiente:

Ejemplo com SPI

3.1.5. Unidad de medida inercial

El componente central del sistema propuesto esta definido como IMU o unidad de medida inercial. Está compuesto por una serie de sensores los cuales serán utilizados para conocer de manera exacta aspectos de localización del sistema a bordo tales como velocidad, orientación, fuerzas gravitacionales etc.

Esta información puede ser utilizada para el control o el simple conocimiento del sistema en un momento concreto.

En el caso del presente proyecto, será utilizado para obtener los ángulos de navegación o ángulos de Tait-Braun, en los que la orientación se presenta con tres rotaciones ortogonales en torno al eje X, Y y Z. Un ejemplo de este tipo de representación se encuentra en la figura 3.5

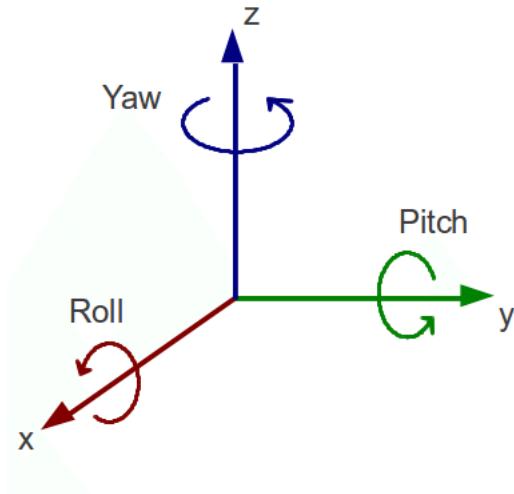


Figura 3.5: Ángulos de Tait-Brain.

Una IMU queda definida por su número de grados de libertad (DOF) los cuáles dependerán del número de sensores a bordo (acelerómetro, giroscopio) y el número de ejes en los que se aplican. Así una IMU con seis grados de libertad (por ejemplo un acelerómetro de 3 ejes y un giroscopio de 3 ejes) se diría que es 6DOF.

En las figuras 3.6, 3.7, 3.7 se presentan algunos ejemplos de IMUs.

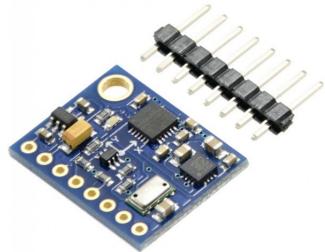


Figura 3.6: Unidad de Medida Inercial.



Figura 3.7: Unidad de Medida Inercial.

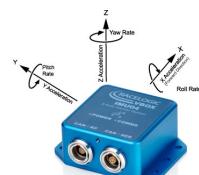


Figura 3.8: Unidad de Medida Inercial.

Otra de las características importantes y de la cuál dependerá en gran medida el precio del producto es el rango de los sensores a bordo y de la disponibilidad en el propio sistema de conversores analógico digitales, encargados estos de convertir el valor del sensor a un valor binario que pueda ser utilizado para posterior análisis.

A continuación se definen algunos de los aspectos más importantes que pueden ser encontrados en una unidad de medida inercial y los cuáles serán muy útiles a lo largo del presente proyecto.

Acelerómetro

Un acelerómetro, como su propio nombre indica es un dispositivo que permite medir la aceleración a la que un cuerpo está sometido. Es un componente fundamental en las unidades de medida inercial pues se puede detectar por ejemplo, condiciones de caída libre, aunque el principal uso es para determinar la orientación del sensor.

Los acelerómetros normalmente son de tres ejes, es decir, son capaces de medir independientemente la aceleración en el eje X, Y y Z, lo que permite conocer la magnitud y dirección del vector aceleración en cada uno de los ejes.

El caso que interesa en este proyecto es poder determinar la orientación del sensor. Para ello, se debe aplicar trigonometría. Suponiendo un sistema 2D con el representado en la figura 3.9.

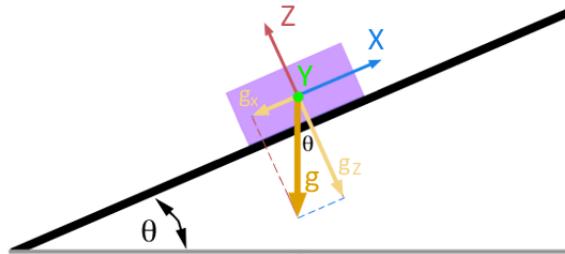


Figura 3.9: Trigonometría en sensor acelerómetro 2D.

$$\theta = \tan^{-1} \frac{A_x}{A_z} \quad (3.1)$$

Si se aplica en una representación 3D como la representada en la 3.10

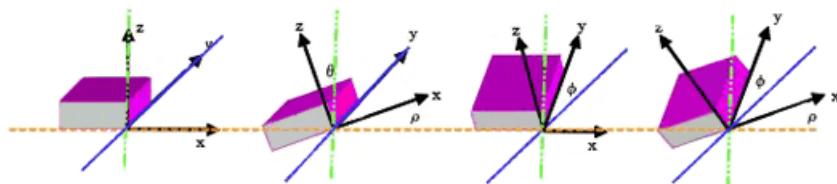


Figura 3.10: Trigonometría en sensor acelerómetro 3D.

$$\theta_x = \tan^{-1} \frac{A_x}{\sqrt{A_y^2 + A_z^2}} \quad (3.2)$$

$$\theta_y = \text{atan} \frac{A_y}{\sqrt{A_{x^2} + A_{z^2}}} \quad (3.3)$$

$$\theta_z = \text{atan} \frac{A_z}{\sqrt{A_{x^2} + A_{y^2}}} \quad (3.4)$$

Giroscopio

Un giroscopio es un dispositivo que permite medir el ángulo de rotación de un determinado dispositivo.

En un giroscopio, siempre se miden ángulos relativos a una referencia arbitraria (a diferencia de los acelerómetros). El giroscopio que se emplea en este proyecto es denominado giroscopio vibratorio de efecto Coriolis.

Al igual que en el acelerómetro del apartado anterior, los giroscopios suelen emplear tres ejes, es decir, son capaces de registrar de forma independiente la rotación en los ejes X, Y y Z, se tiene por tanto el modulo y dirección del vector de rotación.

Es importante tener en cuenta que este tipo de giroscopios basados en el efecto Coriolis no detectan el ángulo girado sino la velocidad angular, aspecto importante para su depuración.

Si se recuerda el concepto de velocidad angular (ecuación 3.5);

$$\omega = \frac{\delta \omega}{\delta t} \quad (3.5)$$

Así, para obtener el ángulo es necesario realizar la integración respecto del tiempo (ecuación 3.6).

$$\theta_{gyro} = \omega_{gyro} * \Delta t \quad (3.6)$$

Los acelerómetros son dispositivos que frecuentemente son muy sensibles a las vibraciones, por lo que para su correcto procesamiento hay que tener en cuenta que presentará mucho ruido de alta frecuencia. Un filtrado a una frecuencia determinada resolverá parte de este problema.

Tener que hacer una integración con respecto al tiempo lleva consigo algunos problemas, los cuales serán vistos en la siguiente sección.

Problema de deriva

Gran parte de las unidades de medida inercial del mercado están compuestas como mínimo de acelerómetros y giroscopios, el motivo de esta combinación es que uno complementa las limitaciones del otro y viceversa.

La deriva en un sensor electrónico o "drift" es una variación con el tiempo de la salida del medidor (con respecto a la medida real) a pesar de que la variable pueda ser constante, es provocado en cierta medida por cambios en la temperatura o por la acumulación de errores.

Tanto en un acelerómetro como en un giroscopio encontramos errores asociados a la deriva:

- Los acelerómetros no tienen deriva a medio o largo plazo, sin embargo, se ven influenciados por los movimientos del sensor y el ruido por lo que no son fiables a medio o corto plazo.
- En cuanto a los giroscopios, funcionan muy bien en movimientos bruscos y cortos pero al realizar una integración con respecto al tiempo aparece el problema de deriva a medio o largo plazo.

Después de analizar los problemas de ambos, parece razonable poder combinar ambas mediciones para obtener orientaciones más precisas.

Posibles soluciones

Para solucionar algunos de los problemas anteriores, puede ser muy útil aplicar algunas de las soluciones propuestas a continuación:

- Combinar y filtrar las señales mediante un filtro complementario. Es el más utilizado en la actualidad debido a su no muy elevada complejidad. Su expresión más sencilla se representa en la ecuación 3.7.

$$\theta = A * (\theta_{prev} + \theta_{gyro}) + B * \theta_{accel} \quad (3.7)$$

- Combinar y filtrar las señales mediante un filtro de Kalman, el cuál realiza una estimación del valor futuro de la medición. Sin embargo, utiliza unos cálculos complejos.
- Algunas unidades de medida inercial incorporan de manera interna procesadores DMPs (Digital Motion Processor) los cuales ejecutan complejos algoritmos evitando tener que realizar filtros y liberando al sistema de procesamiento.

Una vez explicados las características básicas de las IMUs comerciales, en la sección 3.2.2 se desarrollará la IMU elegida para este sistema y sus características propias.

3.1.6. Motores

Controlador PWM

Tener un módulo con la capacidad para generar PWM soluciona muchos problemas posteriores a la vez que mejora la visibilidad del código en el sistema final. Como se observa en las características de los motores, gran parte de ellos están comandados por una señal PWM que si bien es cierto, depende del motor en cuestión.

La modulación por ancho de pulsos o PWM (Pulse Width Modulation) es una técnica en la que se modifica el ciclo de trabajo de una señal periódica (en nuestro caso cuadrada) que se usa para transmitir información a través de un canal de comunicación o para controlar la cantidad de energía que se envía a una carga. Un ejemplo de una señal PWM cuadrada es la mostrada en la figura tal

figura PWM

Aplicando esta señal, por ejemplo, a un motor DC clásico se varía la cantidad de energía que se aplica a la carga, el motor en este caso. Sencillamente funciona como un interruptor en el cual un nivel lógico alto es abierto y un nivel bajo cerrado. Si se consigue variar el tiempo en el cual al motor le está llegando carga y el tiempo en el cual no le llega corriente, se puede controlar su

velocidad.

Así, las características de una señal PWM son:

- D = ciclo de trabajo.
- θ
- T = periodo de la función

3.1.7. Control PID clásico.

3.1.8. Diseño estructura mecánica

3.1.9. Diseño PCB

Después de caracterizar todo el sistema, y teniendo en cuenta el diagrama de conexiones necesario no solo entre el micro-controlador y FPGA sino también para el módulo de cámara OV7670 y el driver del motor, podía ser conveniente y apropiado un circuito impreso que resolviese algunos problemas de ruido, cables excesivos, etc.

Para ello, y antes del diseño, es necesario un análisis funcional de los requerimientos, no sólo actuales, sino también futuros, así como para la implementación de un vehículo aéreo no tripulado. El circuito impreso debería poder albergar los siguientes componentes y comportamientos:

- Un total de 28 pines en la parte exterior de la PCB y dispuestos en la posición correcta para un encaje en la placa IceZum Alhambra II (figura 3.11), lo cuál permite poder usar como entradas o salidas los pines de la FPGA. Para conocer la posición exacta de los pines en la placa se hizo uso del proyecto en Altium el cuál está disponible en GitHub.

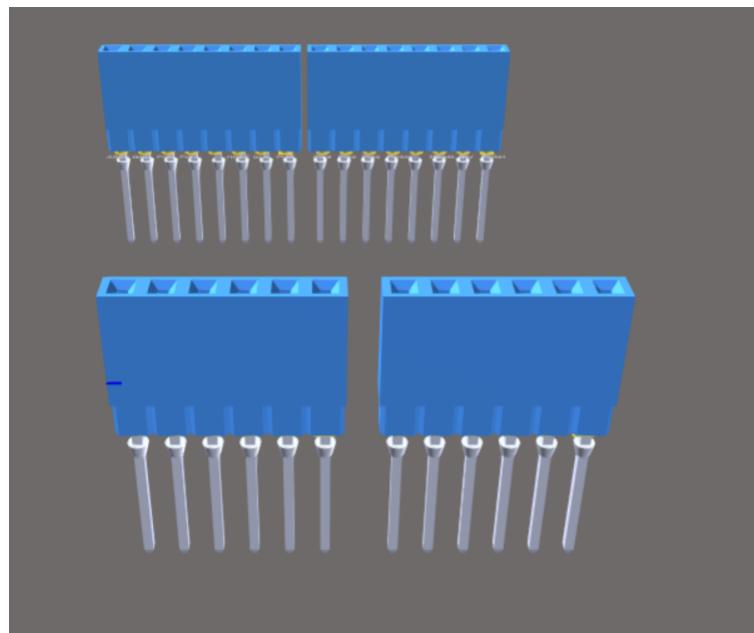


Figura 3.11: Pin headers para IceZum Alhambra II.

- 4 conexiones VCC y GND de 12 Voltios para alimentar los ESC de los motores brushless del vehículo aéreo. Para ello se ha elegido el componente de la figura 3.12, por cumplir con

las características adecuadas de temperatura máxima a la que puede estar sometido y las cuales serán analizadas más adelante:

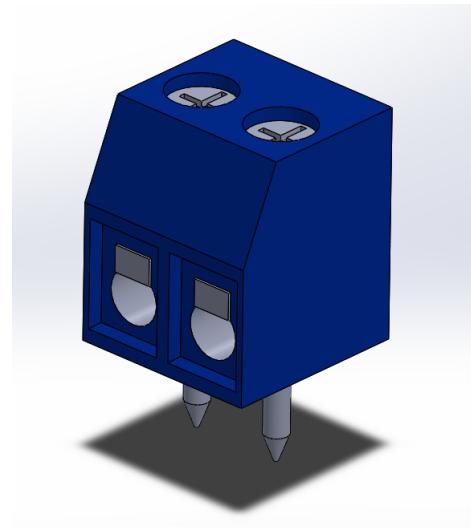


Figura 3.12: Conector GND y VCC.

- Una conexión VCC y GND para alimentar las anteriores conexiones. Este conector irá directo a una batería LIPO de 11.1V (3 celdas) y 2200mAh. El hecho de elegir esta batería viene dado al voltaje mínimo por el que se alimentan los ESCs de los motores brushless así como los motores y el driver del motor utilizado para el Balancing-robot. Un análisis mas detallado de la batería puede encontrarse en subsección 3.1.10
- Módulo de pines "header" para la conexión del MPU6050 anteriormente descrito (figura

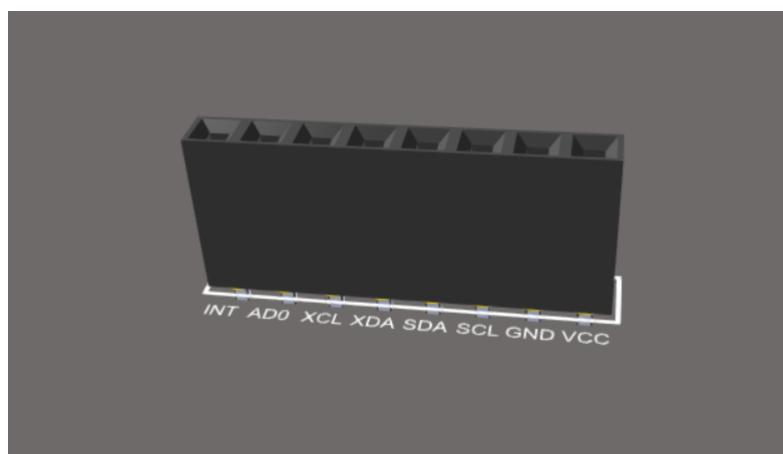


Figura 3.13: Módulo MPU6050.

- Se hace una extensión de los pines mas importantes del MPU6050 para que puedan ser utilizados por el micro-controlador en caso de que el análisis del ángulo, como en este proyecto, forme parte de un proceso gobernado por dicho micro-controlador.
- Se dispone para el usuario final la posibilidad de unos "jumpers" (figura 3.14) para dar la opción al usuario sobre quién gobernará la comunicación I2C, la FPGA o el micro-controlador, para el cuál se podrán usar los conectores del apartado anterior.

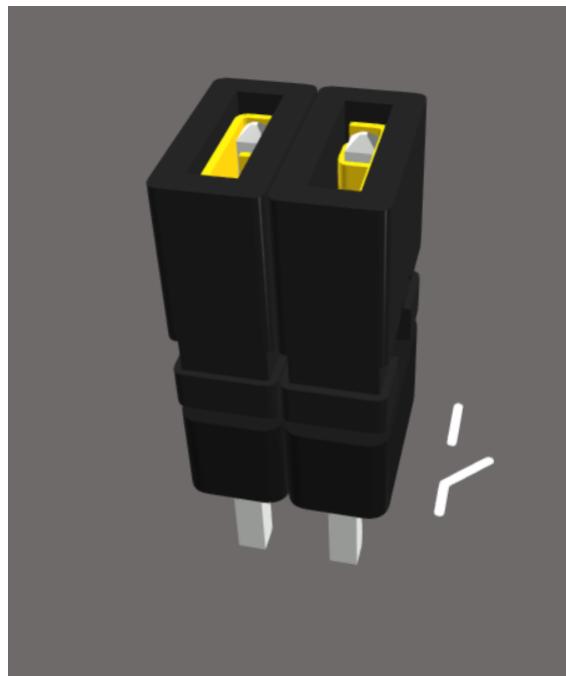


Figura 3.14: Jumpers para configuración I2C MPU6050.

Al albergar una línea de datos no se hace necesario tener en cuenta las características térmicas del conector en cuestión, y al trabajar a una frecuencia relativamente pequeña, puede aceptarse el ruido que los "jumpers" pudiesen introducir en la comunicación I2C.

- Gran parte de los micro-controladores actuales trabajan a 3.3-5v pero la tensión de entrada que soportan puede llegar hasta 12V, por lo que se aprovecha la alimentación de la batería LIPO y se implementa un nuevo conector con dos pin "header" a VCC y GND que alimentarán el micro-controlador.
- Un módulo que pueda albergar la cámara OV7670 formado por pin "headers" de tipo macho y cada uno de los cuales estará unido a uno de los pines in-out de la FPGA, como se verá mas adelante en el esquemático general.
- Un módulo que pueda albergar el driver del motor DC utilizado en este caso y que permita la conexión directa con los pines de la IceZum Alhambra II, pues por ejemplo la señal pwm que definirá la velocidad de los motores será una salida por uno de los pines de la FPGA, y deberá ser una entrada al driver del motor.
- Para que no haya errores en la transmisión I2C se dispone en cada línea una resistencia de pull-up de 4,7KΩ.

3.1.10. Batería

Como todo sistema electrónico, es necesario una fuente de alimentación que permita el correcto funcionamiento de todos los componentes, por lo tanto, y como tarea fundamental, un análisis de requerimientos de dichos componentes que forman el sistema completo es prioritario para poder elegir una alimentación adecuada. Además se tiene en cuenta que la finalidad es tener un sistema móvil y que en la medida de lo posible se evita una conexión directa a la red eléctrica o una conexión usb a un ordenador. Así, solo queda elegir que tipo de batería es adecuada.

A continuación se nombran los diferentes tipos de batería con los que se cuenta actualmente en el mercado analizando sus ventajas e inconvenientes más importantes:

- Baterías de plomo ácido. Son económicas y fáciles de fabricar pero no admiten sobrecargas ni descargas profundas, además, son de un peso y volumen muy elevados para la poca energía que son capaces de almacenar.
- Baterías de níquel-cadmio (Ni-Cd). Funcionan bien en un rango amplio de temperaturas, y se pueden sobrecargar sin sufrir daños. Admiten descargas profundas y proporcionan un buen número de ciclos. Como en la anterior, tienen un peso y volumen muy elevado.
- Baterías de níquel-hidruro metálico (Ni-MH). Mejora las características de las anteriores, sin embargo, proporciona un número menos de ciclos.
- Baterías de iones de litio (Li-ion). En comparación con las anteriores estas son de un desarrollo más reciente y han facilitado la existencia de tecnologías portátiles. Tienen una capacidad elevada en relación a su peso y volumen, tienen un factor de auto-descarga muy elevado. Casi no se ven afectadas por el efecto memoria y pueden cargarse sin haber sido descargadas previamente. En contrapartida, no soportan bien los cambios de temperatura.
- Baterías de polímero de litio (Li-Po). Son una variación de las baterías de Li-on que mejoran sus características de peso y volumen así como su tasa de descarga. Quedan prácticamente inutilizadas si se descargan en exceso.

Teniendo en cuenta cuál tiene que el sistema final y más restrictivo es un vehículo aéreo remotamente pilotado y que el balancing-robot necesita de un peso no muy elevado, es importante que las características de peso, volumen, y descarga sean las adecuadas. Por ello se elige para la alimentación de los sistemas en este proyecto las baterías de tipo Li-po, las cuales pueden almacenar gran cantidad de energía y ofrecen una tasa de descarga muy alta.

Las baterías de tipo Li-Po tienen una nomenclatura diferente del resto, la cual es necesario analizar:

- Clasificación por número de celdas "S". El número S corresponde con el número de celdas, las cuales son de 3,7 voltios pero pueden llegar a 4,2 si están totalmente cargadas. Una batería de 3 celdas (3S) está compuesta por tanto de 3 sub-baterías puestas en serie, es decir, un total de 11,1 voltios.
- Capacidad indicada en "mAh". A mayor de número de miliamperios hora más capacidad de carga. Un error común es pensar que a mayor capacidad mas posibilidad de alargar el tiempo del sistema en cuestión. A mayor capacidad, el peso y volumen de la batería aumenta, por lo que hay que encontrar la mejor configuración para nuestro sistema.
- Tasa de descarga "C". El número C corresponde con la tasa de descarga de la batería. Si una batería es de 1C significa que la máxima tasa de descarga a la que puede llegar es a la que se corresponde a su capacidad. Si el número C es distinto de 1 significa que multiplicamos la tasa de descarga por ese valor, reduciendo el tiempo de descarga proporcionalmente, esto es, una batería de 1000mAh 2C se descagaría a 2A en media hora.

El siguiente planteamiento sería por tanto elegir qué valores anteriores necesita nuestro sistema, para ello se desarrolla la siguiente tabla con los requerimientos de cada componente:

Componente	Voltaje de operación	Corriente de salida por canal	Canales de motor	PWM n.
MC33926 Motor Driver	5V-28V	2.5 A	2	20kHz

3.2. Implementación del sistema

3.2.1. Integración IceZum Alhambra-Arduino Nano

Para el desarrollo de este sub-capítulo se parte de la base de que el ángulo actual ya ha sido obtenido el micro-controlador debido a que este concepto se explica en 3.2.2. Así, y para la claridad del lector, un esquemático interno del arduino nano es mostrado en la figura 3.15. En este apartado y con respecto a la parte del micro-controlador, solo se analiza la parte sombreada.

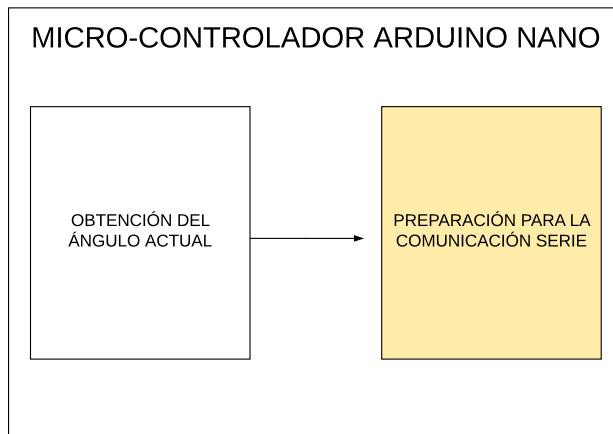


Figura 3.15: Diagrama interno Arduino Nano.

El diagrama de flujo sobre el cuál se basa el código en C del micro-controlador se representa en la figura 3.16

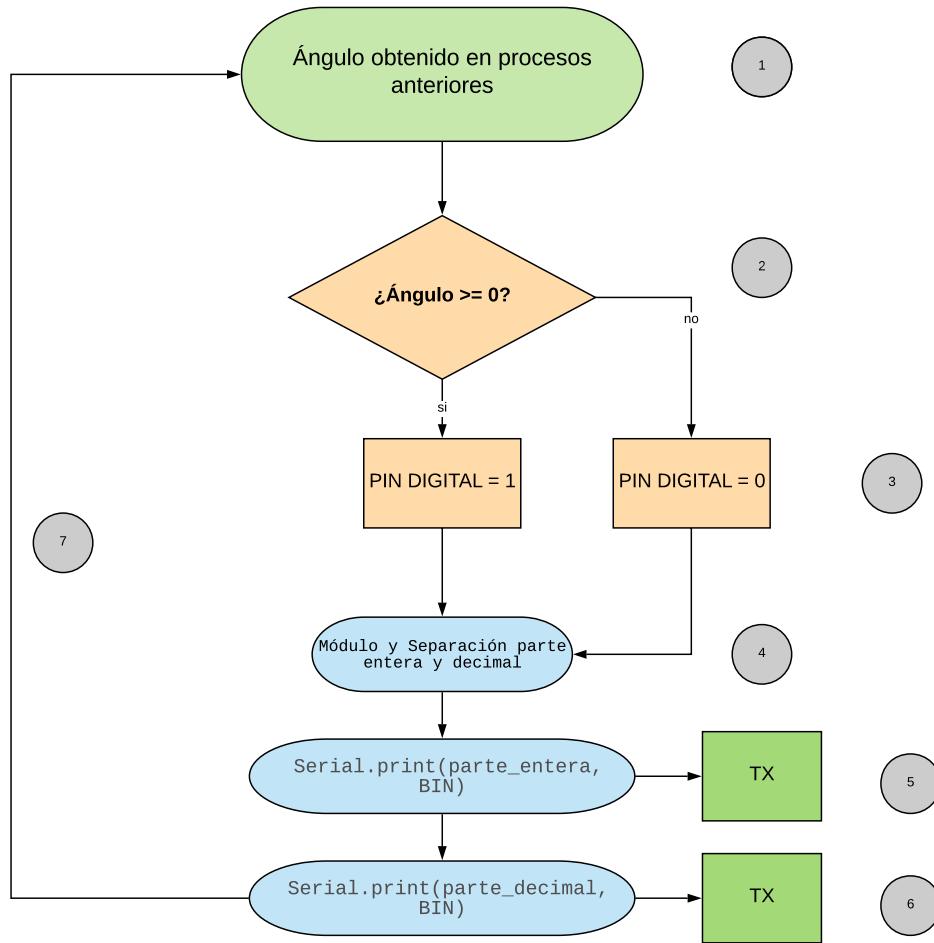


Figura 3.16: Envío de ángulo usando el puerto serie.

A continuación se pasa a explicar los aspectos mas relevantes del desarrollo anterior:

- **1.-**Se parte de la base de que el ángulo ya ha sido obtenido y es correcto. En la sección 3.2.2 se puede encontrar más información sobre ello.
 - **2.-**Una de las partes más importantes para corregir el control del balancín es conocer la inclinación en cada momento para poder corregir esta desviación (figura en la que se ve como se hace esa corrección). Para ello, la FPGA debe conocer si el ángulo es positivo o negativo. Este aspecto no formará parte del propio protocolo de comunicación como se verá a continuación.
 - **3.-**Se utiliza un solo pin del micro-controlador el cuál cambiará su valor a 1 o 0 dependiendo del signo del ángulo en cada momento. Así, la FPGA sólo tendrá que leer esta información cuando le sea necesario.
 - **4.-**Para un correcto entendimiento por parte de la FPGA es necesario enviar el ángulo representado en bytes y no en código ASCII. Es por ello que para ese envío de utilizará el comando "Serial.print(ángulo)". Esta función de arduino envía por el puerto serie la representación en bytes del ángulo en cuestión.
- Si se intenta enviar todo el ángulo también se enviarán tanto el símbolo como el código ASCII de la coma y esto es un aspecto que no interesa tener en cuenta en lo que al procesamiento sobre la FPGA se refiere. Para corregirlo primero se hace el módulo del

ángulo (no se necesita el signo porque ya hay un pin destinado para ello) y posteriormente de hace una separación entre la parte entera y la parte decimal para poder eliminar el carácter "comma".

- 5.-Se envía por el puerto serie el byte correspondiente a la parte entera.
- 6.-Se envía por el puerto serie el byte correspondiente a la parte decimal.
- 7.-Es un proceso cíclico que se irá reproduciendo cada X segundos, esto es, el ángulo podrá corregirse cada X segundos.

Un ejemplo real del envío del un ángulo con su correspondiente signo es mostrado en la figura ??

Ya se ha analizado la parte de la comunicación serie por parte del microcontrolador. A continuación se desarrolla la parte de la lectura de este ángulo sobre la FPGA IceZum Alhambra.

Por un PIN de entrada a la FPGA le estarán continuamente entrando datos provenientes del pin de transmisión y para que se puede hacer una lectura correcta del byte es necesario conocer:

- Cuando comienza una transmisión de un byte.
- Cuando termina una transmisión de un byte.
- Cuando un bit puede ser capturado.
- Se vayan almacenando en un buffer los bits necesarios hasta que el byte este completo.

Para poder implementar en la FPGA un módulo intermedio con las anteriores características es necesario conocer de antemano la velocidad de la transmisión por parte del micro-controlador, la cuál se ha fijado en 16200 baudios.

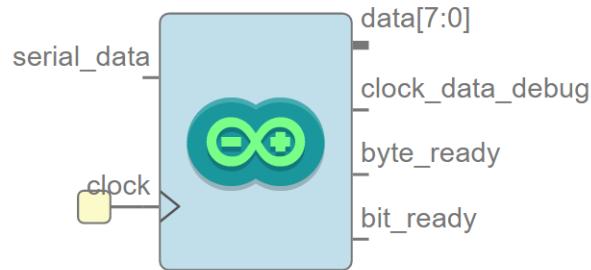


Figura 3.17: Apariencia del módulo interfaz de Arduino Nano en IceStudio.

El aspecto en IceStudio del módulo intermedio a la obtención tanto de la parte entera como de la parte decimal tendrá el aspecto que se muestra en la imagen 3.17. Si se aprecia, sus entradas y salidas tienen mucho que ver con las características requeridas para la lectura correcta de un byte. Como entrada tendrá el bus de transmisión del micro-controlador (TX), y como salidas:

- `data[7:0]`: Consiste en un buffer en el cuál se van almacenando los bits cuando es necesario hasta tener el byte. Es importante que el módulo siguiente conozca cuando el byte está preparado para su captura.
- `clock_data_debug`: La utilización de esta salida es sólo para depuración.
- `byte_ready`: Un flag de reloj cambiará su valor cuando un byte este listo para ser capturado. Como se ha visto en anteriores desarrollos, este byte será o bien la parte entera o bien la parte decimal.
- `bit_ready`: La utilización de esta salida de solo para depuracion.

La implementación en IceStudio del comportamiento anterior será mediante dos máquinas estados con sus correspondientes listas de sensibilidad y el diagrama de flujo se representa en la figura 3.18

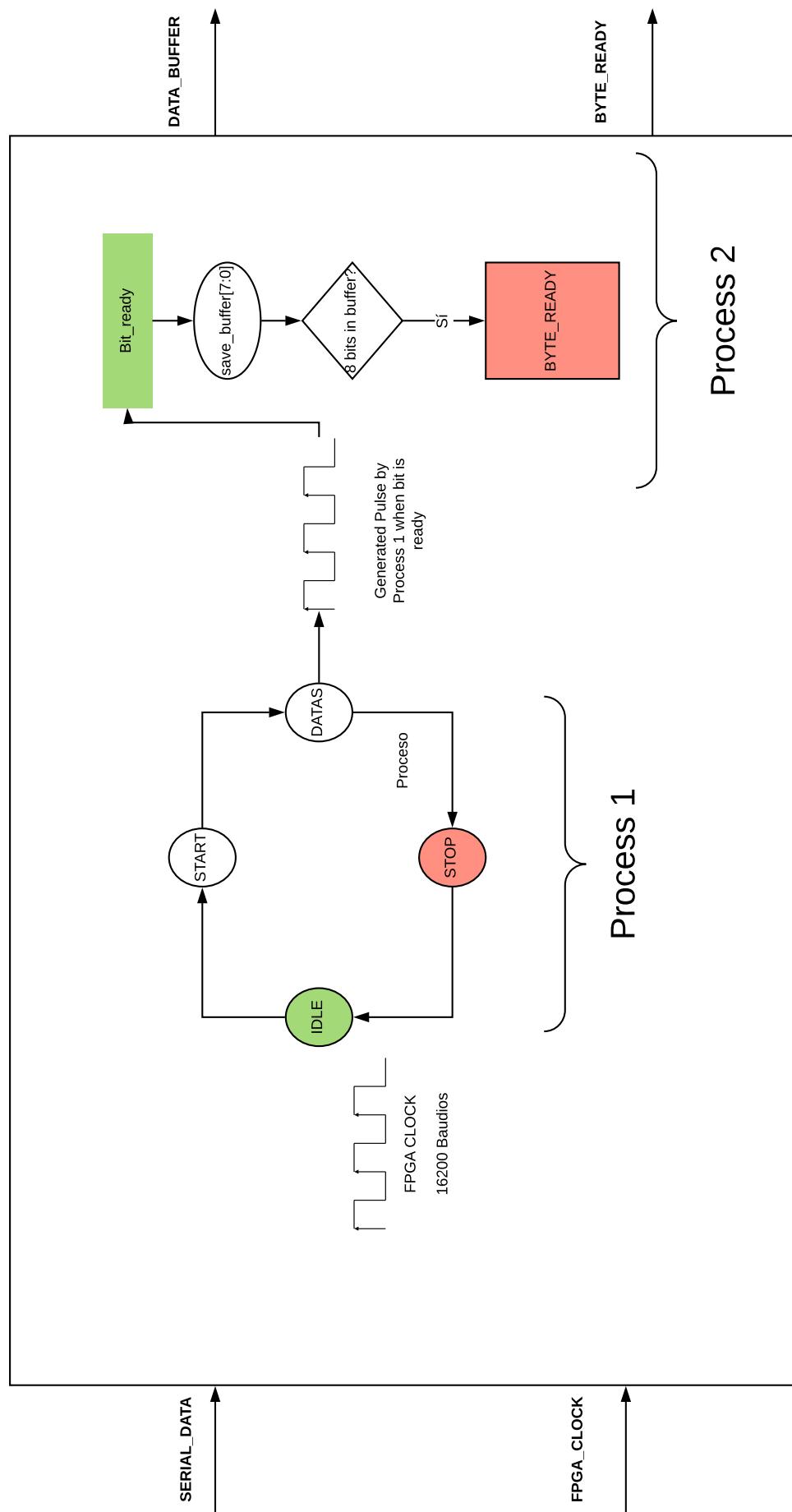


Figura 3.18: Diagrama de flujo de la interfaz para Arduino.

Proceso 1: Este proceso solo dota al sistema siguiente del momento exacto en el cual puede capturar un bit y guardararlo en el buffer, para ello, debe conocer la velocidad de la transmisión comentada anteriormente. Los estados serán los siguientes:

- IDLE: El proceso se mantiene en este estado hasta que se inicie la transmisión, que pasará al siguiente estado (START).
- START: Como se ha desarrollado en la sección ?? el protocolo de transmisión serie comienza con una condición de start, este estado permitirá reconocer cuando acaba esta condición para poder empezar a guardar bits en el buffer.
- DATAS: Como ya se conoce la velocidad de transmisión y se ha reconocido la condición de START en el estado anterior, en este estado un flag cambiará su valor cuando el bit esté preparado para ser guardado en el buffer, de lo cuál se encargará el proceso 2.
- STOP: Además de una condición de START, el protocolo de transmisión en serie usado en Arduino tiene una condición de STOP. Este estado permite reconocer el tiempo que tarda Arduino en llevar a cabo esta última condición, después, volverá al primer estado hasta que empiece una nueva transacción.

Proceso 2: El proceso 2 está activado por el proceso 1. Cuando el proceso 1 determine que un bit está disponible en el bus para ser capturado, pondrá en alta un flag de reloj, iniciando el proceso 2 mediante una lista de sensibilidad. El diagrama de flujo podría ser:

- Esperar hasta que se active la lista de sensibilidad, eso indicará que un bit podrá ser capturado.
- Los bits se irán guardando en un buffer que formará un byte, el cuál representa la parte entera o decimal del ángulo en ese momento.
- Cuando el byte esté preparado para ser capturado por los consecutivos módulos un canal se pondrá en alta, estando disponibles como salidas tanto el buffer con 8 los bits y este canal de "byte_ready".

En este punto la FPGA es capaz de diferenciar cuando puede capturar un byte (BYTE_READY) y de dónde tiene que capturar el bus de datos (DATA_BUFFER). No obstante un aspecto que no forma parte de la comunicación en sí es importante de analizar si se quiere conseguir un correcto funcionamiento. Esto es, si se ha dicho anteriormente que el micro-controlador envía continuamente la parte entera y decimal del ángulo, si no se hace una buena interpretación de estos datos, es posible que un ángulo sobre la FPGA esté formado por una parte decimal de un ángulo n y la parte entera del ángulo $n+1$.

Para ello se crea un módulo en IceStudio que sea capaz de ordenar estos valores que le llegan. El aspecto de este módulo en IceStudio será el representado en la figura 3.19

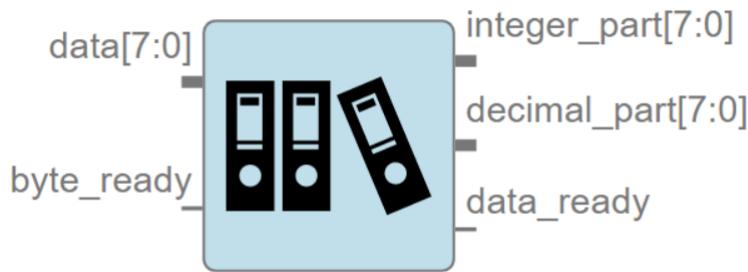


Figura 3.19: Ordenación de la parte entera y decimal de un ángulo.

Como entradas se podrán encontrar:

- `data[7:0]` : Es el buffer de salida del modulo anterior donde se van acumulando los bits capturados hasta conseguir el byte.
- `byte_ready` : Flag de reloj que se activa cuando el byte este disponible para ser capturado.

Es importante tener en cuenta que el dato estará disponible siempre y cuando este disponible tanto la parte entera como la parte decimal del ángulo en cuestión. Así, se pueden encontrar las siguientes salidas:

- `integer_part[7:0]` : byte que representa la parte entera del ángulo.
- `decimal_part[7:0]` : byte que representa la parte decimal del ángulo.
- `data_ready` : flag de reloj que se activa cuando el dato (parte decimal y parte entera) este preparado para ser capturado.

Así, y al igual que se ha profundizado anteriormente, el diagrama de flujo que explica el código en Verilog que implementa el anterior comportamiento se puede encontrar en la figura 3.20 y será comentado mas adelante.

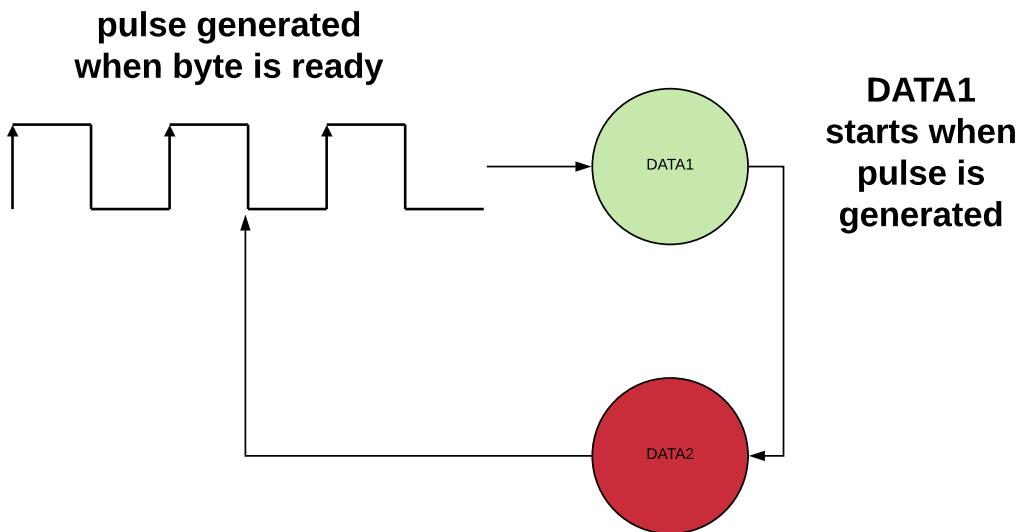


Figura 3.20: Diagrama de flujo modulo ordenación de bytes.

En este caso es un proceso con una lista de sensibilidad que cuenta como señal sensible con el bus de "byte_ready", y que es salida del modulo anterior.

Cada vez que se active el flag que indica que un byte esta listo para ser capturado comienza una máquina de estados cíclica que cuenta con los dos siguientes estados:

- DATA1: Es el primer dato a ser capturado y corresponde con la parte entera del primer ángulo. La segunda vez que se active el flag de "byte_ready" corresponderá a la parte decimal del primer ángulo, y por lo tanto pasará al siguiente estado DATA2.
- DATA2: En ese estado no sólo se capture la parte decimal del ángulo en cuestión sino que además se activa un nuevo flag el cuál indica que el dato completo (ángulo) esta listo para ser capturado.

Así, el módulo tendrá como salidas el buffer de la parte entera, el buffer de la parte decimal, y un bus que avisará a los sucesivos módulos de cuándo el dato completo está preparado para ser capturado. De esta forma se ha evitado el problema explicado anteriormente de la no ordenación en los datos llegados del micro-controlador.

Se da por finalizado el protocolo de comunicación Arduino-FPGA y se brindan las herramientas necesarias para que los sucesivos procesos y módulos puedan conocer el ángulo en cada momento representado por su parte entera (8 bits), parte decimal (8 bits) y un pin que indicará el valor del signo (positivo o negativo).

3.2.2. Unidad de medida inercial MPU6050 en Arduino Nano

El MPU6050 es un sensor y acelerometro para Arduino con 6 grados de libertad.

El MPU6050 necesita un voltaje de 3.3V.

Pin out

EL MPU tiene una tension de alimentacion de 3.3V. El pin SCL (Serial Clock Line) y el pin CDA (Serial Data Line) representan la conexión para el bus de i2c con Arduino Nano. El pin AD0 permite al usuario final cambiar la dirección del MPU (slave) la cuál por defecto es 0x68h conectado a GND. Si se conecta a Vcc la dirección cambiaría a 0x69h. El pin INT produce una señal en alta cuando el dato en cuestion esta disponible por parte del MPU, y avisará por medio de una interrupcion al Arduino Nano con el fin de poder ser obtenido.

Una imagen de un MPU6050 puede encontrarse en la figura ??

MPU6050image

Valores del sensor. DMP.

Los valores

Programa para Arduino nano

3.2.3. Control P

3.2.4. Control D

3.2.5. Controlador PWM

3.2.6. Controlador driver motor

3.2.7. Fabricación estructura mecánica

3.2.8. Fabricación PCB

Para el desarrollo de una PCB con los requerimientos descritos en la subssección 3.1.9 se ha utilizado Altium Designer. El proceso de la elaboración de una PCB en Altium puede ser diferente dependiendo del usuario final, pero en este proyecto se ha seguido la siguiente hoja de ruta:

- 1) En primera instancia se crea el proyecto en cuestión.
- 2) Para cada componente utilizado se crea una nueva librería formada por el esquemático (Schematic) y por el layout en la pcb (footprint).
- 3) Una vez todas las librerías creadas se diseña el esquemático de la placa final, teniendo especial cuidado en que las conexiones sean las adecuadas.
- 4) Con el esquemático ya creado, se puede implementar la pcb, definiendo sus bordes, las pistas, los pads, etc.

El esquemático es representado en la figura 3.21:

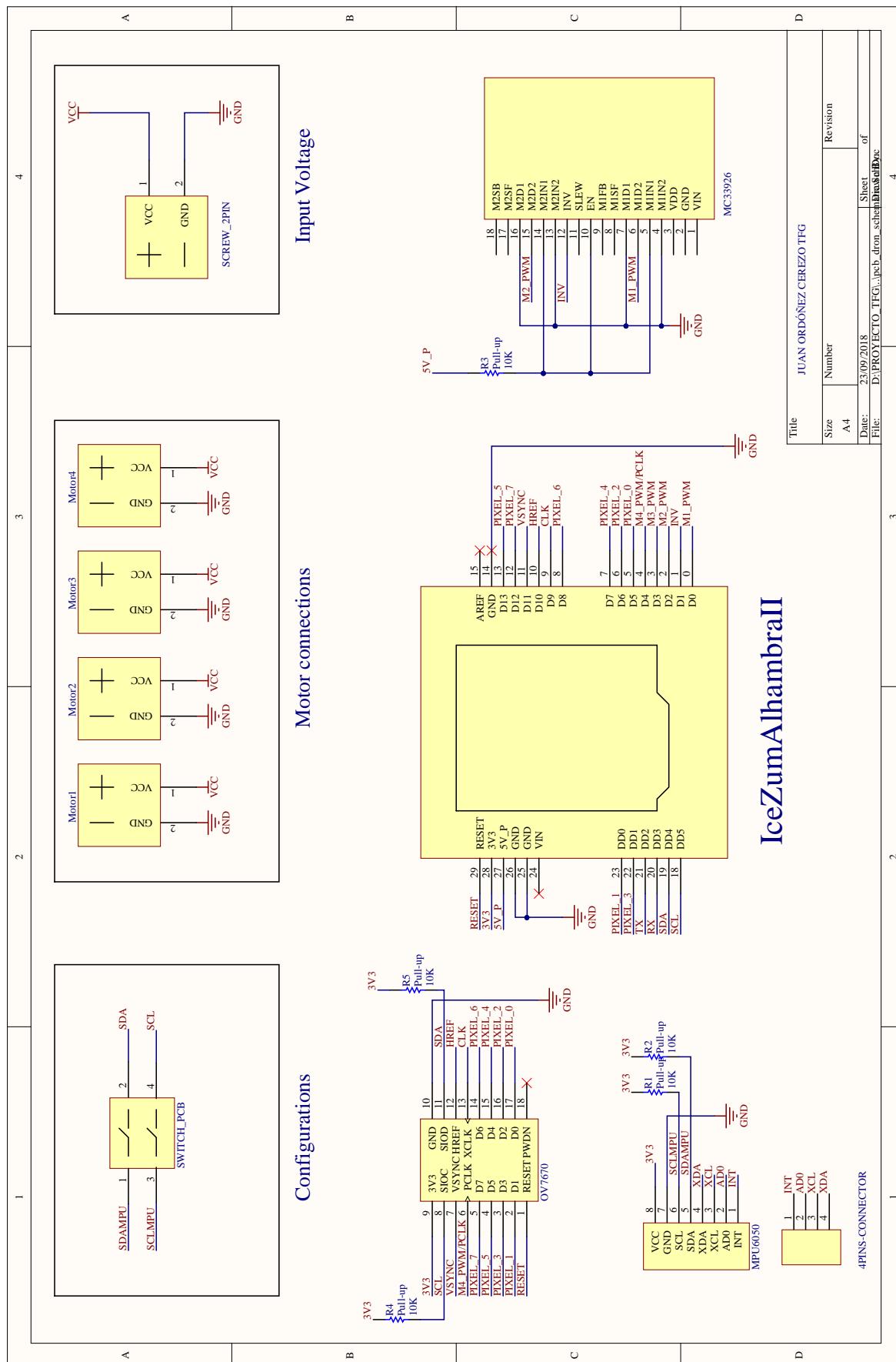


Figura 3.21

Tamaño de pista:

A la hora del diseño de la PCB hay que tener en cuenta el tamaño de las pistas de cobre, sobre todo si por estas la corriente puede ser muy elevada. De no ser así, la PCB podría sufrir daños o llegar incluso a quemarse.

Para calcular un ancho de pista es necesario primero conocer la máxima intensidad que podría circular por ella. Se parte del conocimiento de que uno de los motores del vehículo aéreo no tripulado (este es el peor caso y para el cual es necesario el cálculo de pistas) puede consumir un máximo de 20 amperios. Esta cantidad ha sido extraída del datasheet y suele ser común para este tipo de motores.

La fórmula utilizada para la obtención del ancho de pista ha sido extraída del IPC-2221B, el cual establece los requisitos genéricos para el diseño de tarjetas de circuitos impresos.

Así, la formula se define como en la ecuación 3.8:

$$I = K * dT^{0.44} * (W * H)^{0.725} \quad (3.8)$$

Donde:

- I = intensidad máxima en amperios
- dT = aumento de temperatura sobre ambiente en °C
- W,H = ancho y grosor en mils
- K = 0.024 para pistas internas y 0.048 para externas

Se obtienen los siguientes resultados tanto para pistas internas como externas:

$$W_{externas} = 18.716mm$$

$$H_{externas} = 0.035mm$$

$$W_{internas} = 48.6876mm$$

$$H_{internas} = 0.035mm$$

El grosor de la capa es un dato proporcionado por el fabricante. En el presente proyecto se ha utilizado la siguiente web para el pedido de las PCBs: <https://www.jlcpcb.com//>.

El grosor de pista es medido comúnmente en "oz" y en este caso, la empresa fabricante permite una PCB con 1oz de grosor de pista, que equivale a un grosor de 0.035mm.

Si se analizan los resultados obtenidos de la ecuación 3.1 se aprecia claramente la diferencia entre una pista en una capa interna y una pista en una externa.

Una placa de circuito impreso esta dividida en capas, cada una de las capas tiene su función y organizarlas de manera adecuada es una buena práctica para evitar malos comportamientos. Así, una vez analizados los requerimientos, eran necesarias dos capas para la conexión de pines de datos, además siempre es recomendable un plano de tierra en el que todos los pines conectados a GND tengan una capa común. Esto evita ruidos e interferencias además de asegurar una

buenas referencias de tierra.

En el proveedor de PCBs utilizado, no hay diferencia de precio entre una PCB de tres capas y una PCB de cuatro capas, así, y considerando que en el mejor de los casos el ancho de pista para los motores "brushless" debe ser de 1.8716cm, se llegó a la determinación de la utilización de una de las capas como plano común para las pistas de alimentación de dichos motores. La distribución de las capas será por tanto lo representado en la tabla 3.1:

Para el resto de pistas de datos, se asume un ancho de 20mil, que permite una corriente de 1,46 Amperios, más que necesario para esta aplicación

Top Overlay	Overlay
Top Solder	Solder Mask/Coverlay
Top Layer	Signal
Dielectric 1	Dielectric
VCC	Signal
Dielectric 2	Dielectric
GND	Signal
Dielectric 4	Dielectric
Bottom Layer	Signal
Bottom Solder	Solder Mask/Coverlay
Bottom Overlay	Overlay

Tabla 3.1: Composición de capas en PCB.

Una imagen en 2D de las capas Top Layer, VCC, GND, Bottom Layer, Top Overlay y Bottom Overlay se representan en la figura 3.22 mientras que en las figuras 3.23, 3.24, 3.25, 3.26 se incluye una representación en 3D del sistema final con todos los componentes añadidos.

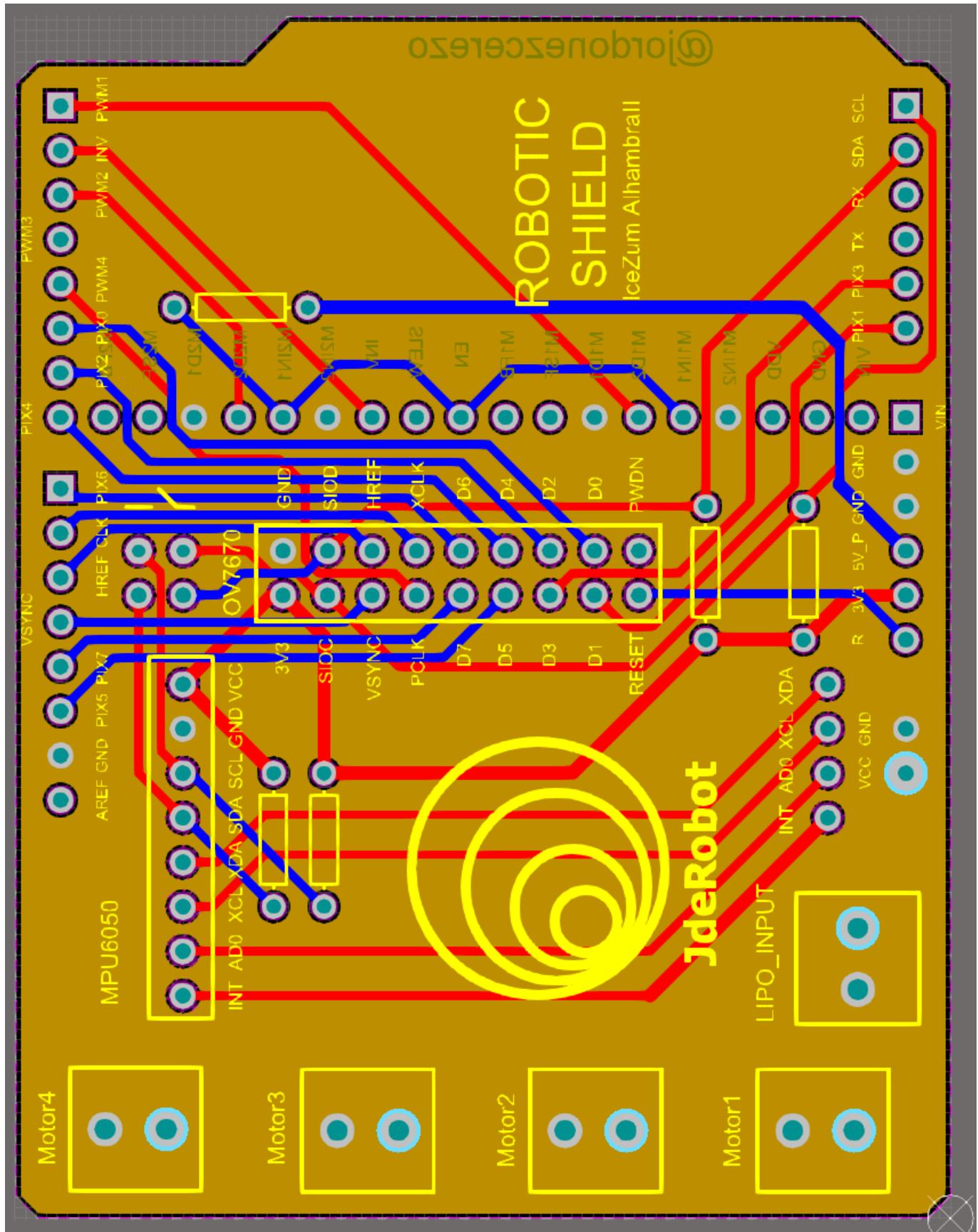


Figura 3.22

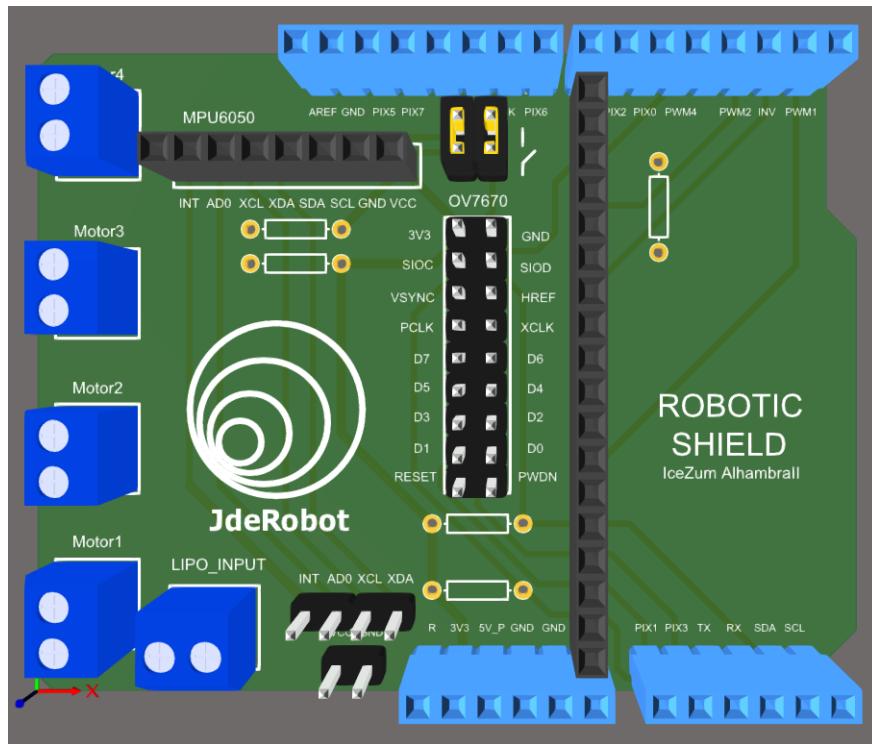


Figura 3.23

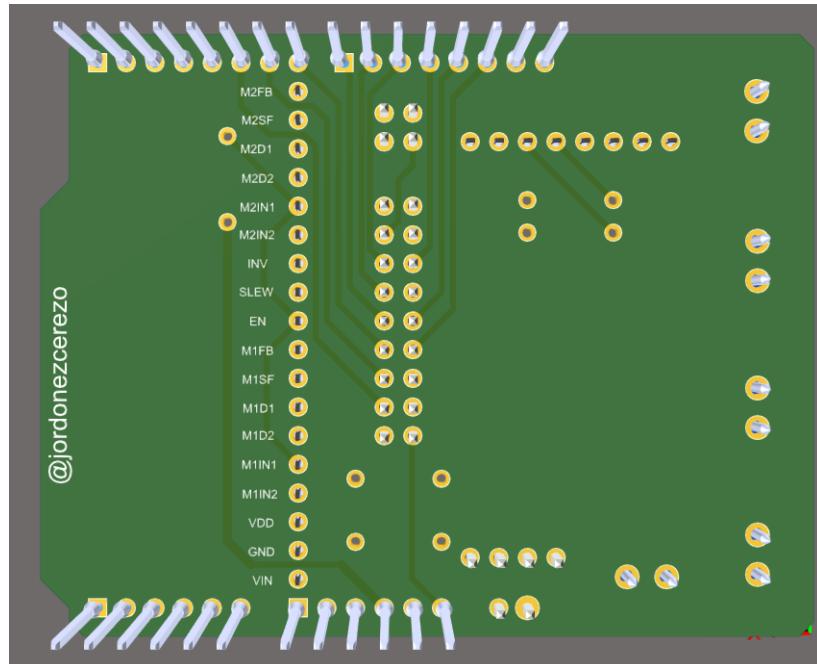


Figura 3.24

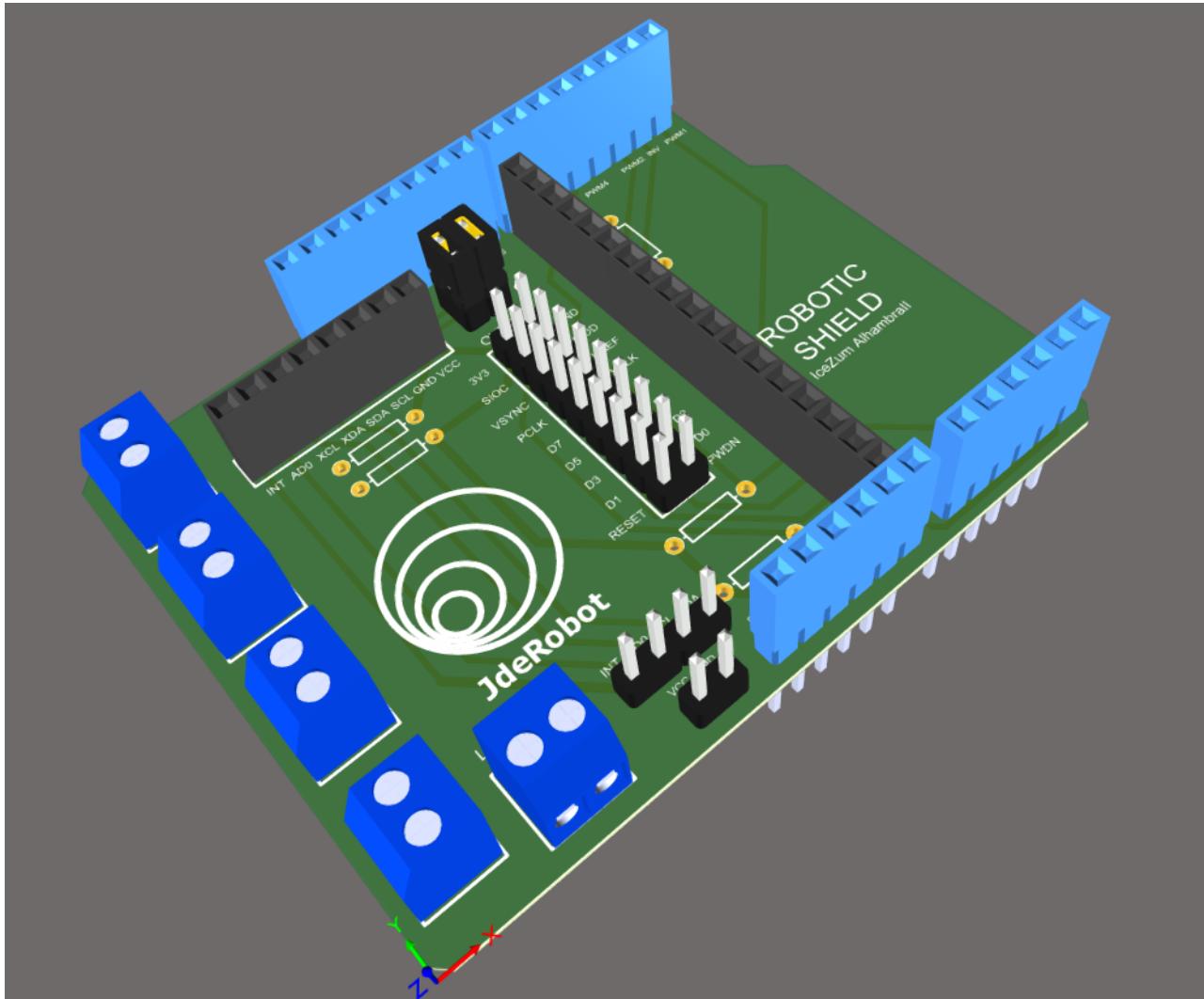


Figura 3.25

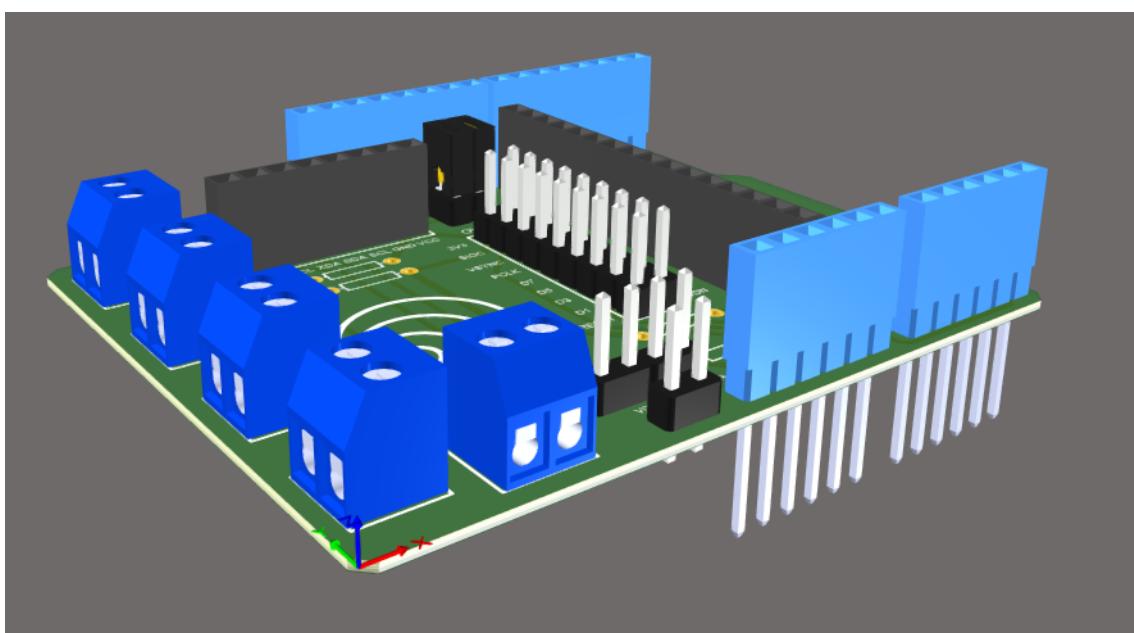


Figura 3.26

3.2.9. Elección de materiales y coste del prototipo**3.3. Experimentos****3.3.1. VGA Module****3.3.2. Protocolo I2C****3.3.3. Controlador motor brushless**

Capítulo 4

Cuadricoptero con vision artificial

Capítulo 5

Conclusiones y trabajo futuro

Con este capítulo se cierra la documentación del proyecto. A lo largo de esta memoria, se ha expuesto el desarrollo de un sistema de electrocardiografía portable abordando cada una de las etapas y partes del sistema necesarias para la adquisición, procesado y visualización de las señales. Los puntos o cuestiones clave desarrolladas en este proyecto se podrían resumir en las siguientes:

- **Diseño del sistema hardware en PSoC:** Se ha desarrollado todo el diseño para la adquisición y transmisión de la señales ECG basado en electrónica reconfigurable. Se ha empleado la plataforma PSoC por su integración de sistemas tanto analógicos como digitales en la misma arquitectura.
- **Diseño del filtrado digital en fase lineal:** Conseguir que la respuesta de los filtros sea lo suficientemente selectiva y además cumpla con el criterio de fase lineal ha sido una de las cuestiones clave para conseguir que la señal tenga la calidad adecuada.
- **Desarrollo de la app para Android:** Es obvio que el objetivo final es que la señal adquirida sea útil para el diagnóstico, para ello es esencial disponer de un sistema donde mostrar la señal. En el presente TFG, además, se ha añadido el procesado de la señal en la app haciéndola una de las partes clave del proyecto. A la importancia de este punto hay que sumar el reto que ha supuesto debido a la necesidad de un rápido aprendizaje en el desarrollo de aplicaciones partiendo de conocimientos prácticamente nulos en este campo.
- **Diseño del prototipo en PCB:** Este ha sido otro de los grandes retos del proyecto puesto que se debía realizar la integración lo más reducida posible. Empleando uno de los encapsulados más pequeños de Cypress se ha conseguido este objetivo con suficiencia, resultando en una PCB de 2x5 cm aproximadamente. En este punto el principal reto fue la soldadura de la placa por las reducidas dimensiones de los chips y la necesidad de aprendizaje de la soldadura por *reflow*.

En general se han cumplido los objetivos planteados al inicio de este trabajo, sin embargo, aun restan retos a los que no se les ha podido dar cobertura en este proyecto o que han surgido durante el desarrollo del mismo. Los principales retos que se plantean se pueden resumir en los siguientes:

- **Desarrollo de aplicación con BLE:** La aplicación desarrollada emplea Bluetooth 2.0 para las comunicaciones. El desarrollo de una aplicación capaz de comunicarse mediante BLE permitiría completar la funcionalidad desarrollada en el prototipo.

- **Desarrollo prototipo con saturación de oxígeno:** Hasta ahora se han llevado a cabo prototipos independientes para la medida de saturación de oxígeno en sangre y ECG, en el presente trabajo. Para completar el trabajo de investigación se deben fusionar ambos prototipos en uno que permita realizar la adquisición de las dos señales a un tiempo.
- **Diseño de gafas e integración del dispositivo:** Una vez se haya conseguido un dispositivo totalmente funcional para las medidas descritas en el punto anterior, se debe encapsular de manera que sea vestible y cumpla con las necesidades del soldado en zonas de conflicto en cuanto a movilidad y versatilidad del sistema.
- **Implementación de eliminación de artefactos mediante MEMS:** Puesto que el dispositivo estará en movimiento en gran parte de su empleo, existirán gran cantidad de artefactos que provocan una pésima calidad de la señal. Existen trabajos como [?, ?, ?] que muestran este tipo de procesado para eliminación de artefactos.

Como se puede ver aún quedan retos por superar en este campo, por esta razón se pretende continuar avanzando en el mismo a través de becas de colaboración o a través de una beca FPU.

