



# UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO  
INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

## Sistema de control autónomo para robots en FPGAs de software libre

---

### Autor

Juan Ordóñez Cerezo

### Directores

Encarnación del Castillo Morales  
Jose María Cañas Plaza



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

—  
Granada, Noviembre de 2018







# **Sistema de control autónomo para robots en FPGAs de software libre**

---

## **Autor**

Juan Ordóñez Cerezo

## **Directores**

Encarnación del Castillo Morales

Jose María Cañas Plaza



# **Sistema de control autónomo para robots en FPGAs de software libre**

Juan Ordóñez Cerezo

**Palabras clave:** FPGA, IceStudio, IceZumAlhambra, microcontrolador.

## **Resumen**

Resumen...



# **Design and manufacturing of sinaptic circuits: Sinaptic Circuits and Memristors**

Alberto Medina Rull

**Keywords:**

## **Abstract**

Abstract.



---

Yo, **Juan Ordóñez Cerezo**, alumno de la titulación de Ingeniería de Tecnologías de Telecomunicación de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77143207-B, 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 5 de mes septiembre de 2018.



---

**D. Encarnación del Castillo Morales**, Profesora del Área de Electrónica del Departamento Electrónica y Tecnología de Computadores de la Universidad de Granada.

**D. Jose María Cañas Plaza**, Profesor del departamento de Teoría de la Señal y las Comunicaciones y Sistemas Telemáticos y Computación de la Universidad Rey Juan Carlos de Madrid.

**Informan:**

Que el presente trabajo, titulado *Sistema de control autónomo para robots en FPGAs de software libre*, 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 5 de mes septiembre de 2018.

**Los directores:**

**Encarnación del Castillo Morales      Jose María Cañas Plaza**



# Agradecimientos

En primer lugar, y si por ello más importante, quiero dar las gracias **a mi familia, Juan, Paqui y María**, los cuáles han sufrido y disfrutado tanto como yo este proyecto, por su credibilidad, paciencia, por los momentos de debilidad suplidos con alegría y sobretodo por creer en la capacidad de alguien que nunca dió motivos para ello.

Por supuesto, a mis tutores Jose María Cañas Plaza y Encarnación del Castillo Morales que creyeron y me hicieron creer desde el principio en este proyecto y que me han llevado en volandas hasta lo que es hoy.

A mis compañeros y amigos de Munich, de Eesy Innovation e Infineon Technologies, que convirtieron un verano cualquiera en el mejor verano del mundo y que me ayudaron sin preguntar porqué, porque si algo tiene que salir bien, saldrá bien.

A mis amigos de Granada a los que tantas veces he tenido que decir que no por dedicar tiempo, porque nunca me canso de estar con ellos y porque dan ánimos en forma de risas y chistes.

Por último, recuerdo parte del discurso de dos profesores en mi graduación, dos profesores, que no por ser profesores se olvidaron de tratarnos como personas y que nos vieron llorar, reír y sobretodo crecer. Y esque, si me tengo que llevar algo de aquí, no son sólo los conocimientos aprendidos, son los amigos y esa gente que te ayuda a escalar cuando no te quedan fuerzas para seguir subiendo, personas que creen en ti aún cuando has caído unas cuántas veces.

**Por todo ello, Gracias.**

**En Granada, a X de Noviembre de 2018.**



# Índice general

<b>1. Introducción</b>	<b>19</b>
1.0.1. Motivación y objetivos . . . . .	19
1.0.2. Planificación (Diagrama de Gantt) . . . . .	21
1.0.3. Metodología de trabajo . . . . .	22
1.0.4. Estructura de la memoria . . . . .	24
<b>2. Estado del arte</b>	<b>25</b>
2.1. Concepto FPGAs . . . . .	25
2.1.1. Evolución y escenario . . . . .	26
2.1.2. Arquitectura FPGA . . . . .	27
2.1.3. Lenguajes de descripción hardware . . . . .	29
2.1.4. Verilog . . . . .	29
2.2. FPGAs libres . . . . .	32
2.2.1. Evolución . . . . .	32
2.2.2. IceZum Alhambra . . . . .	34
2.2.3. IceStudio . . . . .	36
2.3. Coexistencia Microcontrolador-FPGA . . . . .	39
2.3.1. Diferencia microcontrolador-FPGA . . . . .	39
2.3.2. Necesidad . . . . .	41
2.4. Unidad de medida inercial . . . . .	42
2.5. Robótica educativa, motivaciones y necesidad. . . . .	47
2.6. Sensores, actuadores y sistema de control . . . . .	48
2.7. Controlador PID clásico . . . . .	51
<b>3. Robot Balancín</b>	<b>55</b>
3.1. Descripción del problema . . . . .	55
3.2. Diseño del sistema . . . . .	57
3.3. Implementación del sistema . . . . .	58
3.3.1. Fabricación estructura mecánica (Física de un Balancing Robot) . . . . .	58
3.3.2. Unidad de medida inercial MPU6050 en Arduino Nano . .	61
3.3.3. Implementación PCB . . . . .	63
3.3.4. Implementación IceZum Alhambra-Arduino Nano . . . .	73
3.3.5. Control PID en IceZum Alhambra . . . . .	84
3.3.6. Controlador de motores . . . . .	90
3.3.7. Sistema de alimentación . . . . .	93
3.3.8. Materiales y coste del prototipo . . . . .	95
3.4. Experimentos . . . . .	96

---

3.4.1. VGA Module . . . . .	96
3.4.2. Protocolo I2C . . . . .	96
3.4.3. Controlador motor brushless . . . . .	96
<b>4. Cuadricoptero con vision artificial</b>	<b>97</b>
4.1. Diseño . . . . .	97
4.2. Implementación de la percepción . . . . .	98
4.3. Implementación del control . . . . .	103
4.4. Experimentos . . . . .	103
<b>5. Conclusiones y trabajo futuro</b>	<b>105</b>

# Índice de figuras

1.1.	21
1.2. Logo GitHub.	22
1.3. Commits GitHub.	23
1.4. Appear.in.	24
2.1. PLD vs FPGA	25
2.2. Ley de Moore	27
2.3. PLD vs FPGA	28
2.4. Paralelización de Procesos	31
2.5. Lattice iCE40HX1K	33
2.6. Tiny FPGA BX	33
2.7. BlackIce II	33
2.8. ico Board	34
2.9. IceZum Alhambra Board	35
2.10. IceZum Alhambra II Board	36
2.11. Ventana principal IceStudio.	37
2.12. Escritura I2C IceStudio alto nivel.	38
2.13. Escritura I2C IceStudio bajo nivel.	38
2.14. Funcionalidad FPGA y Micro-controlador.	39
2.15. Arquitectura básica procesador	40
2.16. Puertas lógicas después de una implementación hardware.	41
2.17. Sistema bipedo coexistencia microcontrolador-FPGA.	42
2.18. Ángulos de Tait-Brain.	43
2.19. Unidad de Medida Inercial.	43
2.20. Unidad de Medida Inercial.	43
2.21. Unidad de Medida Inercial.	44
2.22. Trigonométrica en sensor acelerómetro 2D.	44
2.23. Trigonométrica en sensor acelerómetro 3D.	45
2.24. Sensor CMOS para adquisición de imágenes.	50
2.25. Unidad de Medida Inercial.	50
2.26. Potenciómetro.	50
2.27. Motor DC.	51
2.28. Diagrama de bloques controlador PID.	53
3.1. SegWay comercial.	55
3.2. Representación péndulo invertido.	56
3.3. Diagrama de bloques final.	57
3.4. Vista frontal Balancing Robot.	58

3.5.	Vista lateral derecha Balancing Robot. . . . .	59
3.6.	Perspectiva Balancing Robot. . . . .	60
3.7.	Centro de masas en sistema final. . . . .	61
3.8.	MPU6050 IMU. . . . .	61
3.9.	MPU6050 IMU. . . . .	62
3.10.	Ventaja de uso en la utilización de DMP. . . . .	63
3.11.	Pin headers para IceZum Alhambra II. . . . .	64
3.12.	Conecotor GND y VCC. . . . .	65
3.13.	Módulo MPU6050. . . . .	65
3.14.	Jumpers para configuración i2c MPU6050. . . . .	66
3.15.	Vista en 3D de Shield para IceZum Alhambra II. . . . .	67
3.16.	Vista en 3D de Shield para IceZum Alhambra II. . . . .	67
3.17.	Vista en 3D de Shield para IceZum Alhambra II. . . . .	68
3.18.	Vista en 3D de Shield para IceZum Alhambra II. . . . .	68
3.19.	Esquemático Shield IceZum Alhambra II . . . . .	70
3.20.	Vista de la composición de capas en Altium. . . . .	73
3.21.	Pines hardware de coexistencia micro-FPGA. . . . .	74
3.22.	Ejemplo de comunicación serie. . . . .	74
3.23.	Separación de procesos micro-FPGA. . . . .	75
3.24.	Diagrama interno Arduino Nano. . . . .	76
3.25.	Envío de ángulo usando el puerto serie. . . . .	76
3.26.	Corrección de la dirección en Balancing Robot. . . . .	77
3.27.	Apariencia del módulo interfaz de Arduino Nano en IceStudio. .	78
3.28.	Diagrama de flujo de la interfaz para Arduino. . . . .	80
3.29.	Módulo para ordenar datos provenientes de Arduino. . . . .	82
3.30.	Diagrama de flujo modulo ordenación de bytes. . . . .	83
3.31.	Comunicación entre Arduino y IceZum Alhambra en IceStudio. .	84
3.32.	Diagrama de flujo control P. . . . .	85
3.33.	Aspecto en IceStudio del módulo de control P. . . . .	86
3.34.	Diagrama de flujo control D. . . . .	87
3.35.	Aspecto en IceStudio del módulo de control D. . . . .	88
3.36.	Aspecto final en IceStudio del Balancing-Robot. . . . .	89
3.37.	Ejemplo de señal PWM con dutty diferente. . . . .	90
3.38.	MC33926 para el control de los motores DC. . . . .	91
3.39.	Esquemático de conexiones MC33926. . . . .	91
3.40.	Aspecto en IceStudio del generador de señal PWM. . . . .	92
3.41.	Envío de ángulo usando el puerto serie. . . . .	92
3.42.	Batería LIPO 11.1V y 2.2A. . . . .	95
3.43.	Batería LIPO 3.7V y 4mAh. . . . .	95
4.1.	Diseño a alto nivel del cuadricóptero con visión. . . . .	97
4.2.	Cámara OV7670. . . . .	99
4.3.	Formación de píxel en OV7670. . . . .	99
4.4.	Aspecto en IceStudio del módulo de escritura I2C. . . . .	99
4.5.	Diagrama de flujo de la escritura I2C. . . . .	101
4.6.	Esquemático para cámara OV7670. . . . .	102
4.7.	Buffer triestado para bus I2C. . . . .	103
4.8.	Ejemplo de escritura I2C en OV7670. . . . .	103

# Índice de cuadros

2.1. Modo de operación de sensores. . . . .	49
3.1. Composición de capas en PCB. . . . .	72
3.2. Tabla eléctrica de componentes. . . . .	94
3.3. Coste total del prototipo Balancing-Robot. . . . .	96



# Capítulo 1

## Introducción

### 1.0.1. Motivación y objetivos

La electrónica digital forma parte de la rama de electrónica más moderna y que evoluciona más rápidamente de la actualidad, debido principalmente a las ventajas que esta ofrece y las cuáles serán analizadas en el presente proyecto. Por ello, es importante llevar a todo el mundo esta tecnología, hacerla amigable y dotar a la sociedad de las herramientas necesarias para su correcto entendimiento, sin que ello suponga la realización de unos estudios superiores.

Involucrarse de lleno en un proyecto tan poco desarrollado y con tan poco información, puede parecer un poco abrumador en principio, sobretodo cuando los problemas empiezan a aparecer, pero para cualquier ingeniero es todo un reto poder empezar a abrir camino sobre un campo determinado, ofreciendo a la sociedad algunas herramientas útiles para futuras implementaciones y mejoras.

Trabajar sobre una plataforma nativa de tu ciudad como es la tarjeta Ice-Zum Alhambra teniendo en cuenta el poco desarrollo en este campo, es además un honor que no hay que perder de vista a la hora de definir las motivaciones y objetivos.

Electrónica digital, FPGAs, micro-controlador, lenguaje de descripción hardware, robótica educativa, son sólo algunos de los conceptos más importantes sobre los que se sustentan el presente trabajo, y forma parte también de una de las más importantes líneas de investigación sobre ingenieros de todo el mundo. Es un objetivo poder llevar este concepto a las aulas de los más pequeños de la sociedad, haciendo uso para ello de librerías de bloques de implementación hardware que ayudarán en los sucesivos trabajos y abriendo el concepto de “robótica educativa”.

A continuación se presentan los objetivos más importantes del presente trabajo y los cuáles deberían ser alcanzados al finalizar este:

- Capacidad para entender e implementar todo tipo de comportamientos robóticos mediante lenguajes de implementación hardware.
- Caracterización de sistemas robóticos actuales.

- Uso de herramientas libres y aptas para la robótica educativa como IceStudio.
- Capacidad para el desarrollo de placas de circuito impreso (Printed Circuit Board, PCB) con Altium Designer.
- Capacidad para el diseño y construcción de estructuras mecánicas mediante SolidWorks e impresión 3D.
- Entendimiento de la importancia de la coexistencia entre micro-controlador y FPGA.

### 1.0.2. Planificación (Diagrama de Gantt)

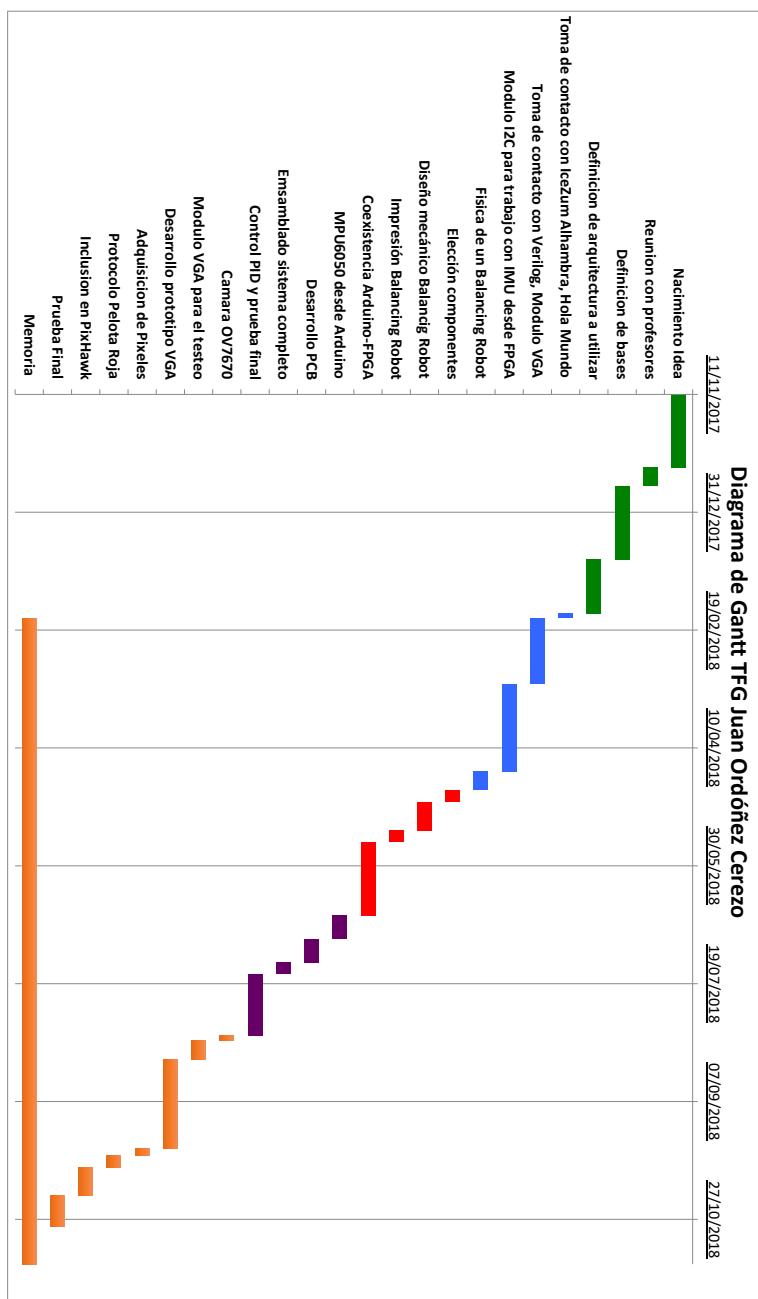


Figura 1.1

### 1.0.3. Metodología de trabajo

Para introducir la metodología de trabajo seguida, se presenta la herramienta GitHub [] (figura 1.2).



Figura 1.2: 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 1.3.

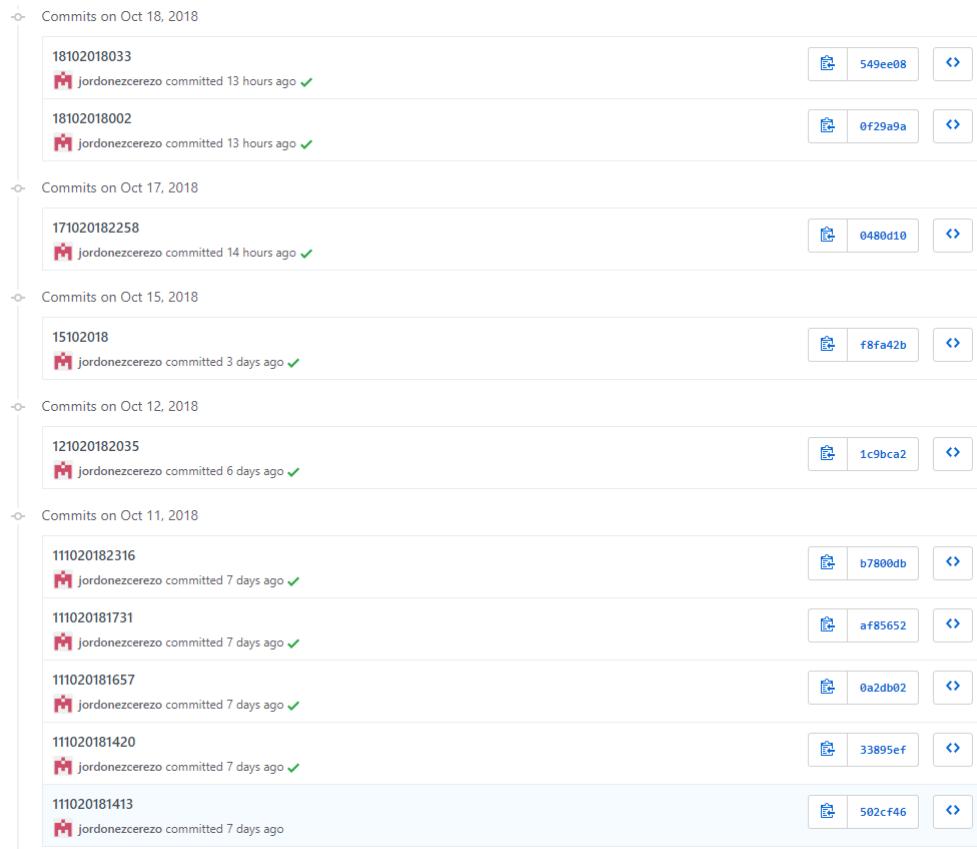


Figura 1.3: Commits GitHub.

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 diferentes objetivos.

Para ello, se utilizó la herramienta software libre llamada “[appear.in](#)”, la cual ofrece videoconferencias entre varios usuarios al mismo instante (figura 1.4).



Figura 1.4: Appear.in.

#### 1.0.4. Estructura de la memoria

La memoria está dividida en tres capítulos o partes diferenciadas. En la sección 2 se explicará brevemente toda la parte teórica y conocimientos necesarios para el entendimiento del presente trabajo. Además se comentará la evolución hasta la actualidad de algunos sistemas propuestos.

En la sección 3 se abordará el problema del Balancing Robot, y se diferenciará entre una parte de diseño y una parte de implementación. El Balancing Robot será capaz de mantenerse estable sobre dos ruedas horizontales.

En la sección 4 se abordará el problema de visión a bordo de un cuadricóptero. Una vez conseguidos los objetivos necesarios en la sección anterior, un cuadricoptero será capaz de seguir una pelota roja, por lo que se implementará el protocolo necesario para ello.

Para terminar, en la sección 5 se expondrán las conclusiones derivadas del trabajo completo, así como un posible trabajo futuro, errores a corregir, etc.

# Capítulo 2

## Estado del arte

En la presente sección se desarrollarán los aspectos teóricos a tener en cuenta para un correcto entendimiento del trabajo realizado. Además se presentará la evolución de algunos sistemas, así como sus ventajas e inconvenientes.

### 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.

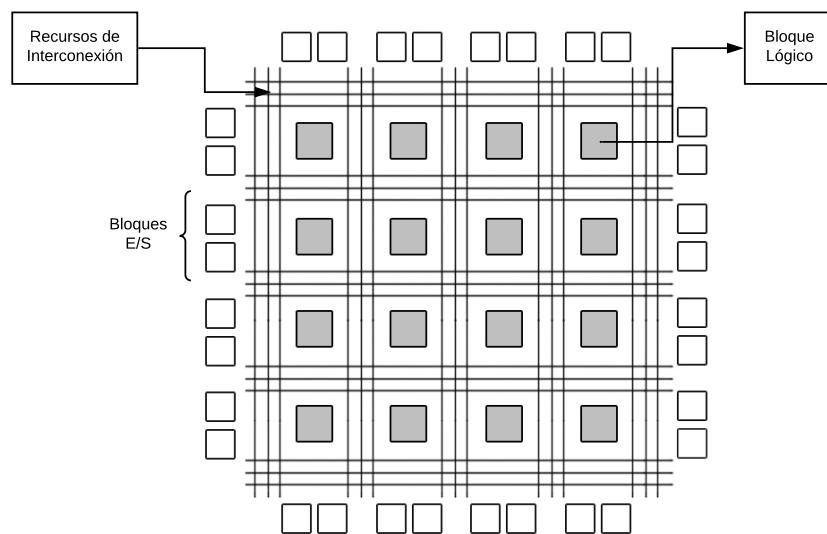


Figura 2.1: PLD vs FPGA

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

Si bien no es el más usado actualmente, a lo largo de la sección 2.1.4 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 la máxima evolución de los PLDs (Programmable Logic Device), definidos estos como circuitos integrados en los que se pueden programar ecuaciones lógicas booleanas. Algunos de sus usos en la actualidad son:

- Sistemas de visión artificial.
- Sistemas de imágenes médicas.
- Codificación y encriptación.
- Reconocimiento de voz.
- Aeronáutica y defensa.

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.

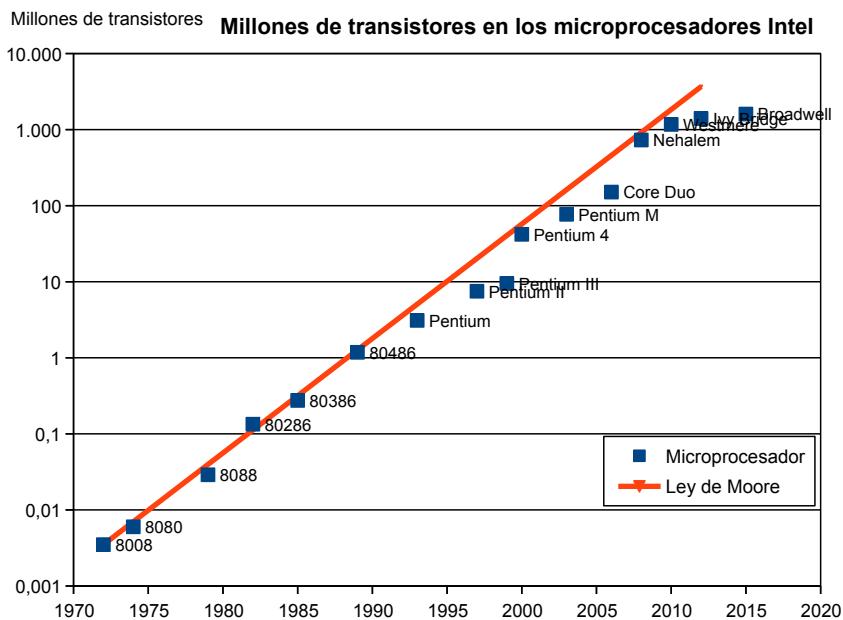


Figura 2.2: Ley de Moore

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 las secciones 2.1.2 y 2.1.3.

### 2.1.2. Arquitectura FPGA

En un PLD, las interconexiones entre elementos ya están prefijadas, 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).

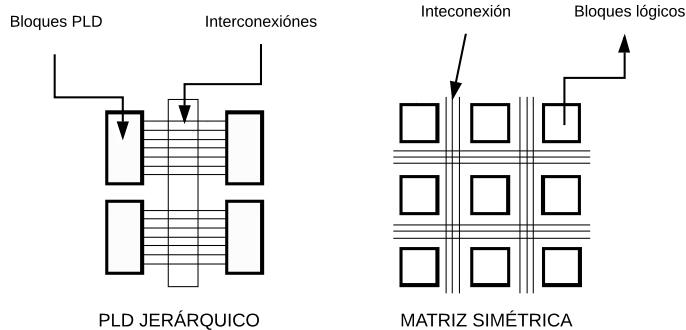


Figura 2.3: PLD vs FPGAs

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 las FPGAs 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 consulta y dos o más flip-flops. La tabla de consulta, 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. Dependiendo el tamaño de la LookUp Table, se pueden implementar funciones de más o menos variables.

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 diseño. Esto se hace por medio de código programable, que puede ser un lenguaje de descripción hardware, el cuál es introducido en la sección 2.1.3.

### 2.1.3. Lenguajes de descripción hardware

Un lenguaje de descripción hardware (Hardware Description Language, HDL) 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<sup>1</sup> 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.

---

<sup>1</sup>lenguajes procedurales: ejecución de la aplicación se inicia con la primera línea de código, y sigue una ruta predefinida a través de la aplicación, llamando procedimientos según sea necesario.

### PROCESOS ORGANIZADOS EN PARALELO

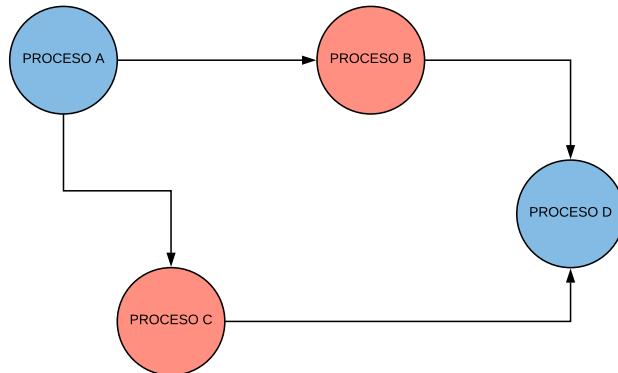


Figura 2.4: Paralelización de Procesos

Toda implementación de Verilog se declara dentro de un proceso que puede ser de 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. El término bytestream se utiliza para describir la configuración con la que se implementará un determinado diseño 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 desarrolló 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 <sup>2</sup>, muchas personas ya están aumentando sus posibilidades.

---

<sup>2</sup>Proyecto libre: El término libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar ese hardware o software.

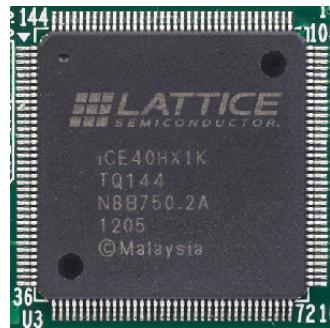


Figura 2.5: Lattice iCE40HX1K

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

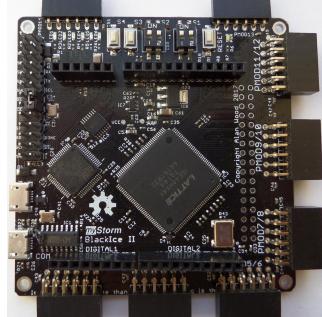


Figura 2.7: BlackIce II



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 analizará en a sección 2.2.3). Alguna de sus características más importantes son:

- 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 tarjeta para el desarrollo del presente proyecto.



Figura 2.9: IceZum Alhambra Board

Un punto a tener en cuenta para el desarrollo hardware con esta FPGA es su memoria de 1K lo que ha supuesto una limitación importante en el desarrollo. Para el presente proyecto se ha hecho uso también de la nueva versión de la IceZum Alhambra, IceZum Alhambra II, la cuál 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 tarjeta IceZum Alhambra II se representa en la figura 2.10

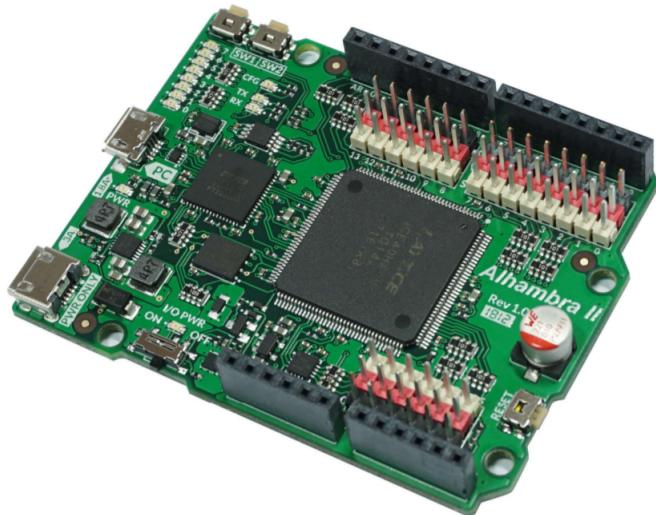


Figura 2.10: IceZum Alhambra II Board

### 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 propias 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 en el que nos basaremos a lo largo del presente documento.

IceStudio es IDE gráfico para FPGAs libres y está 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 apariencia que se muestra en la 2.11:

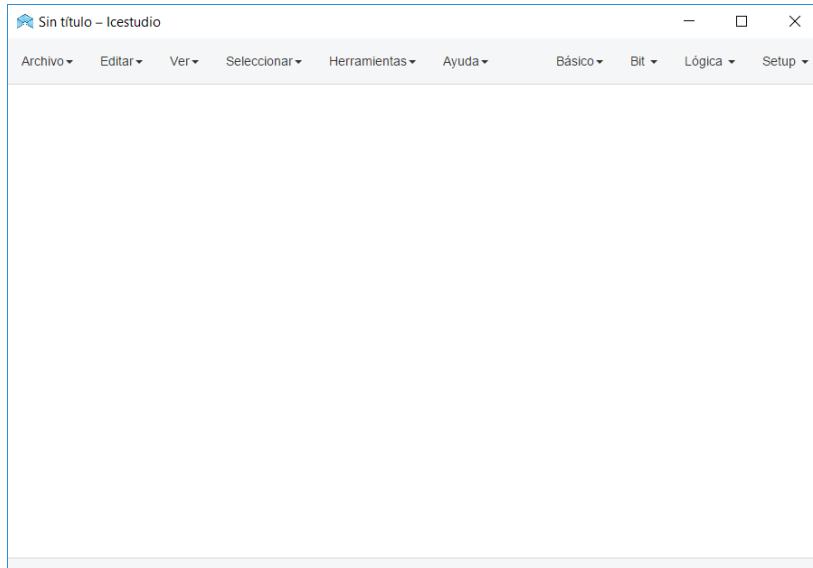


Figura 2.11: Ventana principal IceStudio.

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 caso práctico;

El módulo que se presenta en la figura 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).

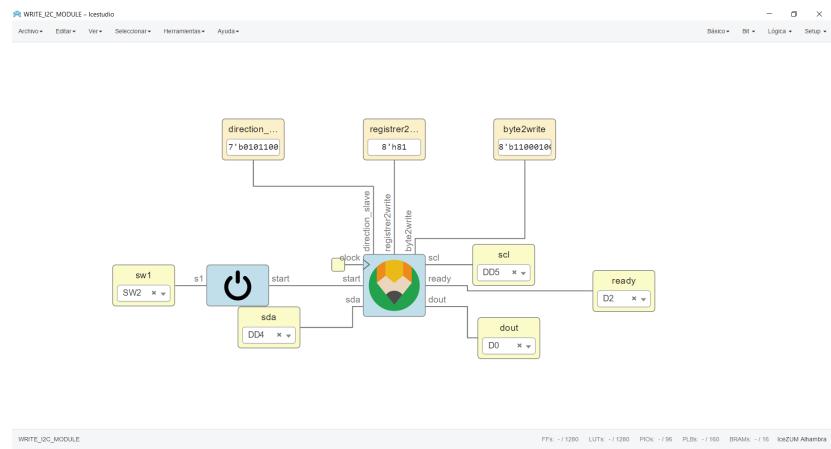


Figura 2.12: Escritura I2C IceStudio alto nivel.

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.

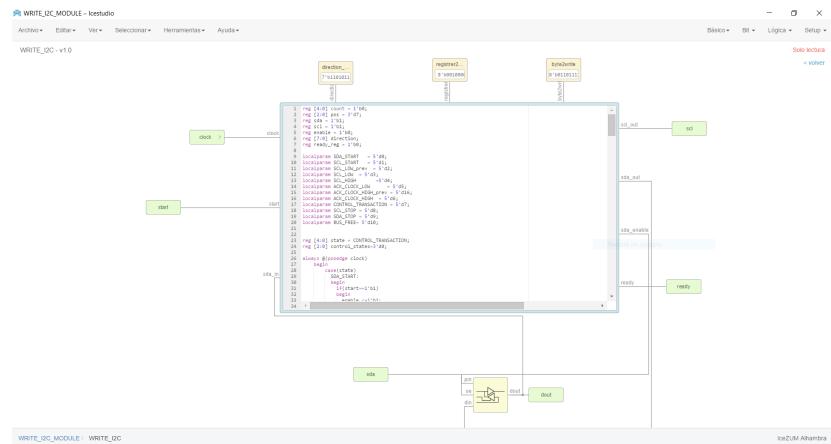


Figura 2.13: Escritura I2C IceStudio bajo nivel.

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 caso práctico 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 desarrollados, y pueden encontrarse en el siguiente enlace:

<https://groups.google.com/forum/forum/fpga-wars-explorando-el-lado-libre>

## 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 tal y como se muestra en la figura 2.14.

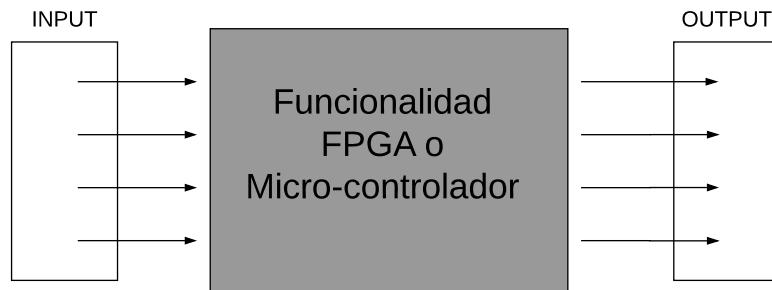


Figura 2.14: Funcionalidad FPGA y Micro-controlador.

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 bits (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.

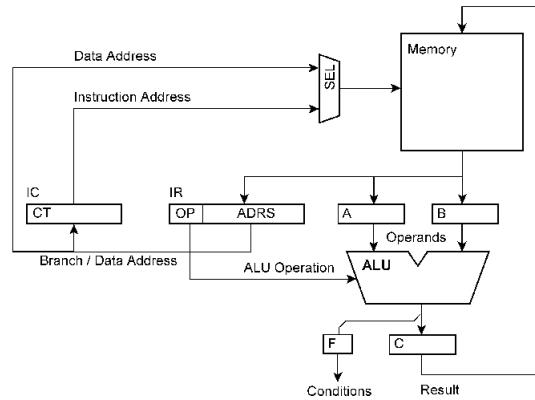


Figura 2.15: Arquitectura básica procesador

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 la sección 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 representar como varias funciones de transferencia en un mismo sistema hardware.

En la figura 2.16 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, notese el problema de la memoria en una FPGA, siendo ésta el número total de puertas lógicas físicas que pueden ser utilizadas.

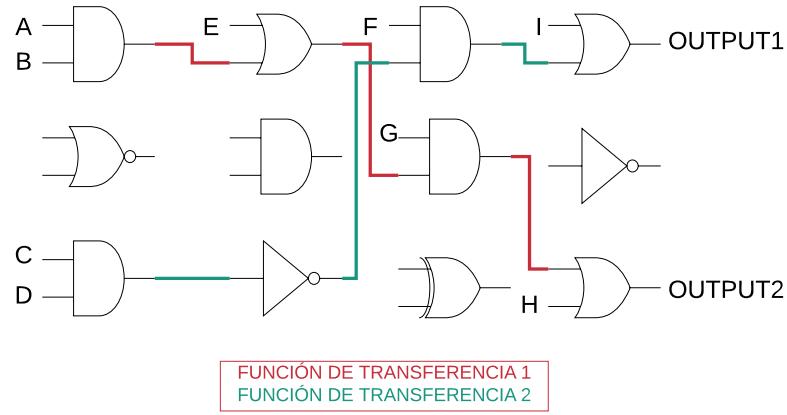


Figura 2.16: Puertas lógicas después de una implementación hardware.

### 2.3.2. Necesidad

Se propone el siguiente caso práctico:

Se requiere monitorizar con exactitud 4 diferentes sensores provenientes del exterior, de una manera exacta, los cuatro al mismo tiempo y a una velocidad especificada por un reloj externo, siendo necesario además una posterior actuación por parte del sistema (apertura de válvula, cierra de puertas, etc).

El diagrama de bloques del flujo de trabajo en un procesador que implemente lo anterior podría parecerse al siguiente:

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, ¿por qué 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 figura 2.17 se puede ver como ejemplo real de un sistema bípedo, el cuál se explicará con mas detalle en los siguientes capítulos.

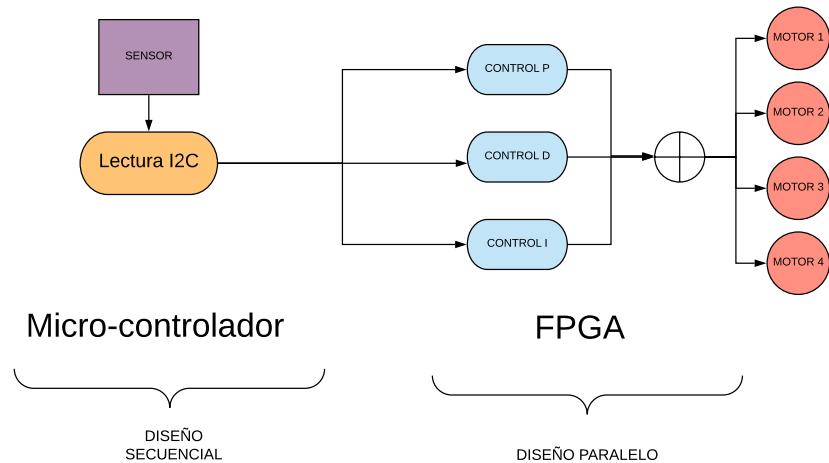


Figura 2.17: Sistema bipedo coexistencia microcontrolador-FPGA.

## 2.4. 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-Brain, 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 2.18.

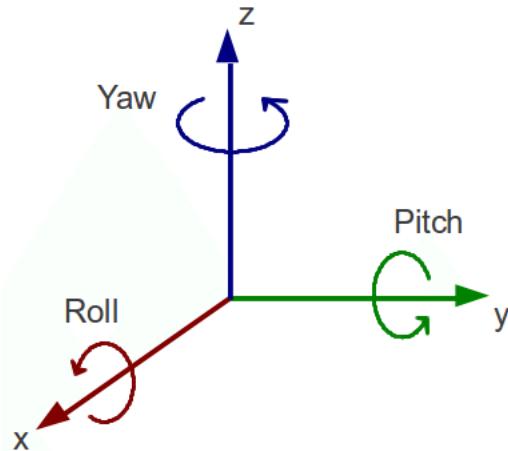


Figura 2.18: Ángulos de Tait-Brain.

Una IMU queda definida por su número de grados de libertad (DOF) los cuales 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 2.19, 2.20, 2.20 se presentan algunos ejemplos de IMUs.

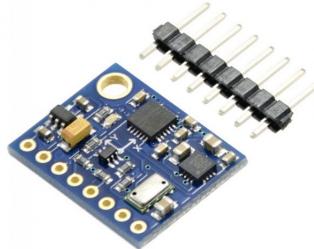


Figura 2.19: Unidad de Medida Inercial.



Figura 2.20: Unidad de Medida Inercial.

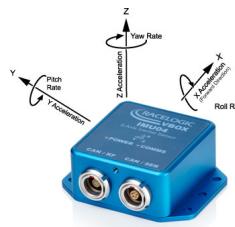


Figura 2.21: Unidad de Medida Inercial.

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 2.22.

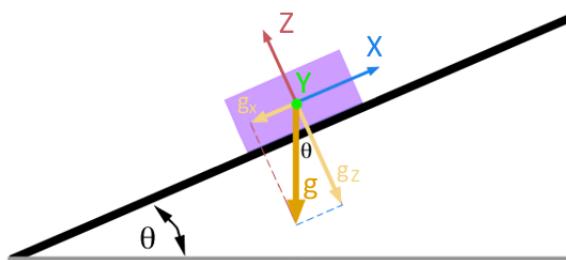


Figura 2.22: Trigonométrica en sensor acelerómetro 2D.

$$\theta = \tan^{-1} \frac{A_x}{A_z} \quad (2.1)$$

Si se aplica en una representación 3D como la representada en la 2.23

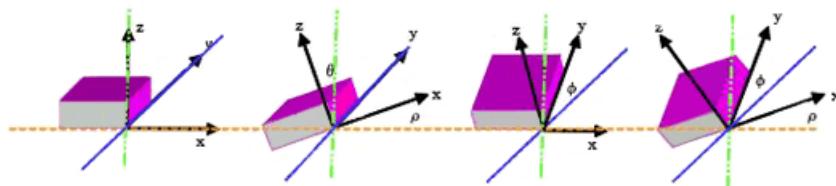


Figura 2.23: Trigonométrica en sensor acelerómetro 3D.

$$\theta_x = \tan^{-1} \frac{A_x}{\sqrt{A_y^2 + A_z^2}} \quad (2.2)$$

$$\theta_y = \tan^{-1} \frac{A_y}{\sqrt{A_x^2 + A_z^2}} \quad (2.3)$$

$$\theta_z = \tan^{-1} \frac{A_z}{\sqrt{A_x^2 + A_y^2}} \quad (2.4)$$

## 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 2.5):

$$\omega = \frac{\delta \boldsymbol{\omega}}{\delta t} \quad (2.5)$$

Así, para obtener el ángulo es necesario realizar la integración respecto del tiempo (ecuación 2.6).

$$\theta_{gyro} = \omega_{gyro} * \Delta t \quad (2.6)$$

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 cuáles 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 con 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 2.7.

$$\theta = A * (\theta_{prev} + \theta_{gyro}) + B * \theta_{accel} \quad (2.7)$$

- 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 cuáles 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 3.3.2 se desarrollará la IMU elegida para este sistema y sus características propias.

## 2.5. 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 mas 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.

## **2.6. 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 inmórbica, 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 entradas a 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 2.1** *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 lo 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 de este proyecto.

Magnitud	Transductor	Característica
Posición lineal y angular	Potenciómetro	Analógica
	Encoder	Digital
	Sensor Hall	Digital
Velocidad lineal y angular	Dinamo tacometrífica	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. Las figuras 2.24, 2.25 y 2.26 muestran ejemplos de algunos sensores.

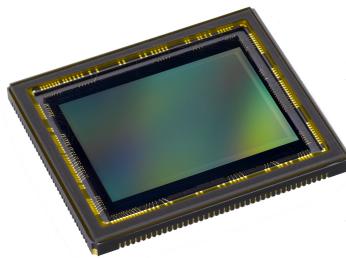


Figura 2.24: Sensor CMOS para adquisición de imágenes.

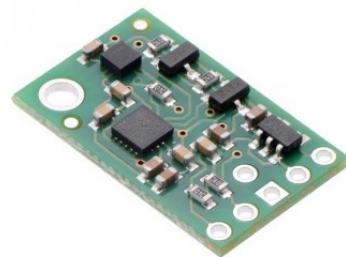


Figura 2.25: Unidad de Medida Inercial.



Figura 2.26: 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 más conocidos son los motores, el cuál será muy utilizado a lo largo de esta

aplicación.

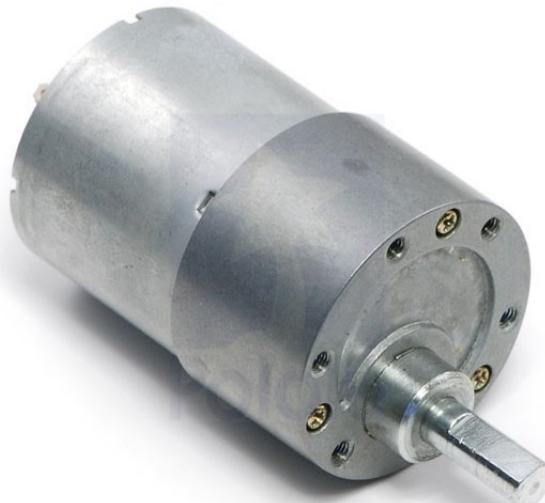


Figura 2.27: Motor DC.

Hay de muchos diferentes tipos de sistemas de control en relación de la aplicación la cual se quiere desarrollar, normalmente se elije 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 o la IceZum Alhambra para las tareas que puedan ser paralelizadas.

## 2.7. Controlador PID clásico

Proporcional integral y derivativo (Proportional-Integral-Derivative, PID) [] es el más común de los controles utilizados en la industria y ha sido universalmente aceptado en la industria de control. La popularidad de este control viene dada a su robustez y a su facilidad de uso, lo que permite a ingenieros trabajar de manera simple.

Como su nombre indica, el algoritmo consiste en tres operadores básicos, proporcional, integral y derivativo los cuales son la clave para obtener una respuesta óptima.

La idea básica de un controlador PID es leer un sensor y calcular una respuesta proporcional, integral y derivativa con el objetivo de calcular la mejor salida de un actuador.

Para conocer bien cómo actúa un controlador PID y conocer el significado de cada uno de sus componentes, se hace necesario conocer que significa un sistema en bucle cerrado.

### Sistema en bucle cerrado

Un sistema de control de lazo cerrado [1] son los sistemas en los que la acción del control está en función de la señal de salida. Los sistemas de circuito cerrado usan la retroalimentación desde un resultado final para ajustar la acción de control en consecuencia. El punto de consigna se define como el valor al cual el sistema quiere alcanzar.

Ahora sí se profundiza un poco más es cuál es el significado de cada una de las componentes por las que está formado el control PID.

### Respuesta Proporcional

La componente proporcional depende sólo de la diferencia entre el valor actual y la consigna. Esta diferencia es referida como el término Error. Así, la ganancia proporcional  $K_p$  determina la relación entre la respuesta de salida y la señal de error. Si el error tiene magnitud 10, una ganancia proporcional de 5 produciría una respuesta proporcional de 50. La ganancia indica la velocidad de corrección de ese error, pero si es demasiado rápido, el sistema oscilará.

### Respuesta Integral

La componente integral suma la componente error en el tiempo, un incremento pequeño del error hasta que la componente integral aumente ligeramente con el tiempo. La respuesta integral aumentará continuamente con el tiempo a menos que el error sea cero, por lo que el efecto es llevar el error de estado estacionario a cero.

### Respuesta Derivativa

La respuesta derivativa hace que la salida disminuya si la variable del proceso aumenta rápidamente. La respuesta derivada es proporcional a la tasa de cambio de la variable de proceso. El aumento de la constante derivativa  $K_d$  hará que el sistema de control reaccione con mayor fuerza a los cambios en el término del error y aumentará la velocidad de la respuesta general del sistema de control.

Un diagrama de bloques del sistema completo de un controlador PID puede verse en la figura 2.28.

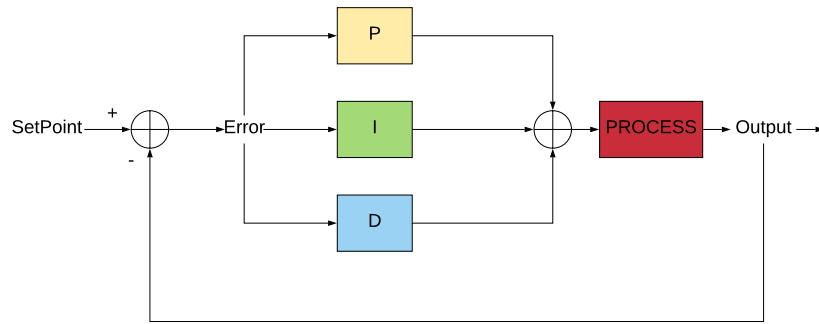


Figura 2.28: Diagrama de bloques controlador PID.

**Ejemplo 2.2** Se hace uso de un PID para el control de la temperatura de una habitación. La consigna es la temperatura a la que se quiere llegar, la planta es la habitación en sí, la cuál puede modelarse como una función de transferencia. Un sensor de temperatura cierra el lazo, indicando la temperatura en cada momento, mientras que una válvula abre y cierra el conducto de aire dependiendo de los valores del controlador PID, que se calcularán en relación al valor de consigna y al valor de la temperatura actual de la habitación.



## Capítulo 3

# Robot Balancín

En el presente capítulo se pretende abordar el problema del péndulo invertido mediante la utilización de una FPGA en coexistencia con un micro-controlador. 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. Se comienza con una descripción del problema a resolver (sección 3.1), a continuación se proporciona una solución básica con un alto nivel de abstracción (sección 3.2), para terminar con una explicación mas detallada de los bloques anteriores (sección 3.3).

### 3.1. Descripción del problema

Para comprender el trabajo a realizar, se enunciará brevemente el problema del péndulo invertido, cuya solución ha dado lugar a muchas herramientas muy famosas en la actualidad, una de ellas, el llamado *SegWay* (figura 3.1).



Figura 3.1: SegWay comercial.

**Definición 3.1** *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 me-*

*dianante 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.

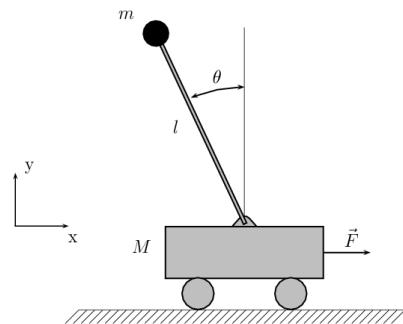


Figura 3.2: Representación péndulo invertido.

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 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 cuánto a teoría de control y dinámica de sistemas.

### 3.2. Diseño del sistema

Mediante una comunicación i2c con un sensor IMU, el micro-controlador obtiene el ángulo actual de sistema. Obtenido el ángulo por el micro-controlador una comunicación de tipo serie lo envía a la FPGA, en un formato binario de 1 byte para la parte entera y 1 byte para la parte decimal. Una shield con un driver de motores DC conectada a la FPGA dota a esta de la posibilidad de variar la velocidad y el sentido de dos motores DC que permiten la estabilización del sistema. La velocidad de los motores para una correcta corrección del ángulo se calcula mediante un controlador PD básico implementado en la FPGA.

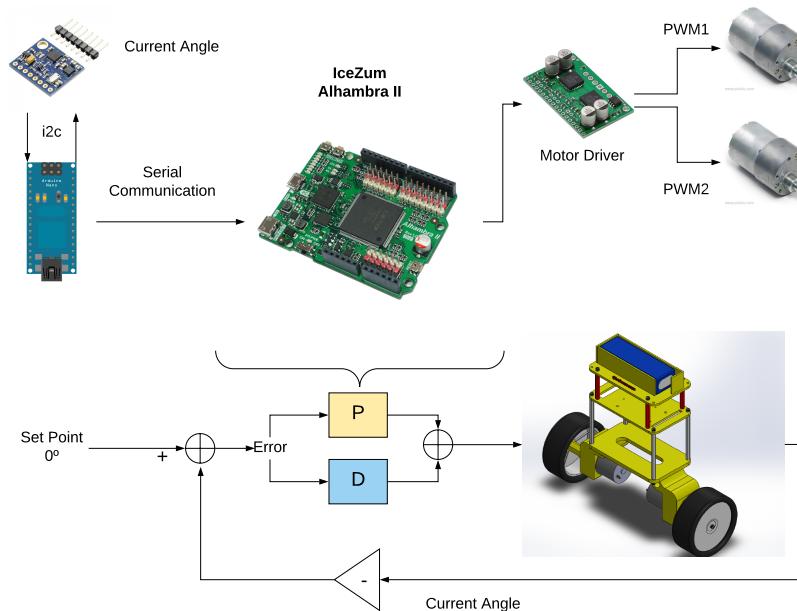


Figura 3.3: Diagrama de bloques final.

En los sucesivos capítulos se irá profundizando en cada uno de los bloques anteriores que forman la solución final (figura 3.3).

### 3.3. Implementación del sistema

#### 3.3.1. Fabricación estructura mecánica (Física de un Balancing Robot)

Teniendo en cuenta la física de un balancing robot y con el objetivo por tanto de solucionar el problema clásico del péndulo invertido, se propone la estructura mecánica de las figuras 3.4, 3.5 y 3.6, diseñadas con SolidWorks[] y a partir de la cuál se ensamblarán el resto de los componentes.

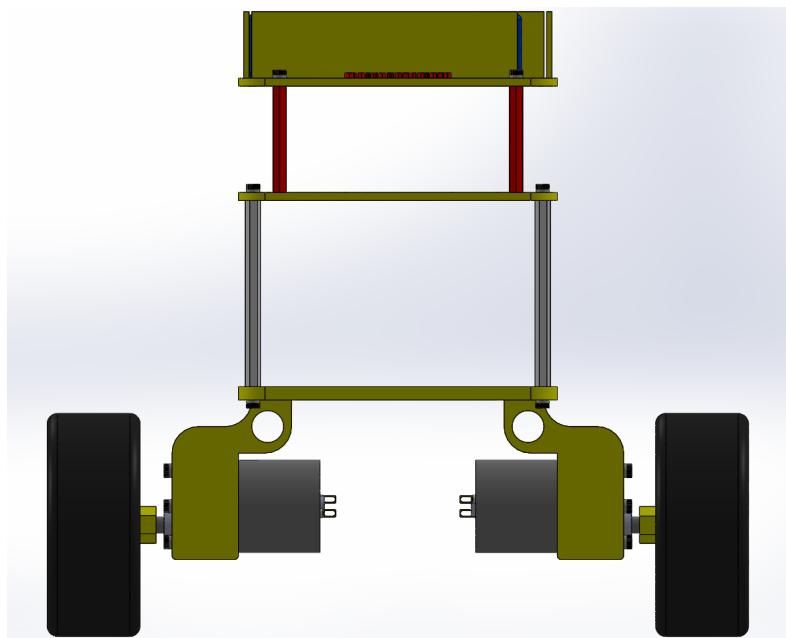


Figura 3.4: Vista frontal Balancing Robot.

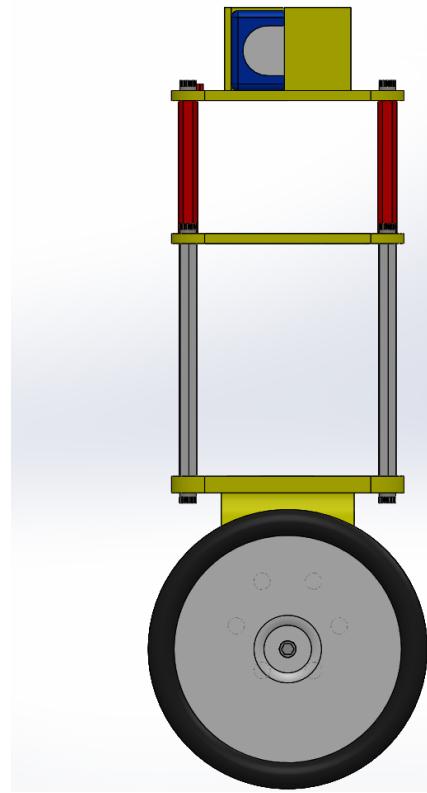


Figura 3.5: Vista lateral derecha Balancing Robot.

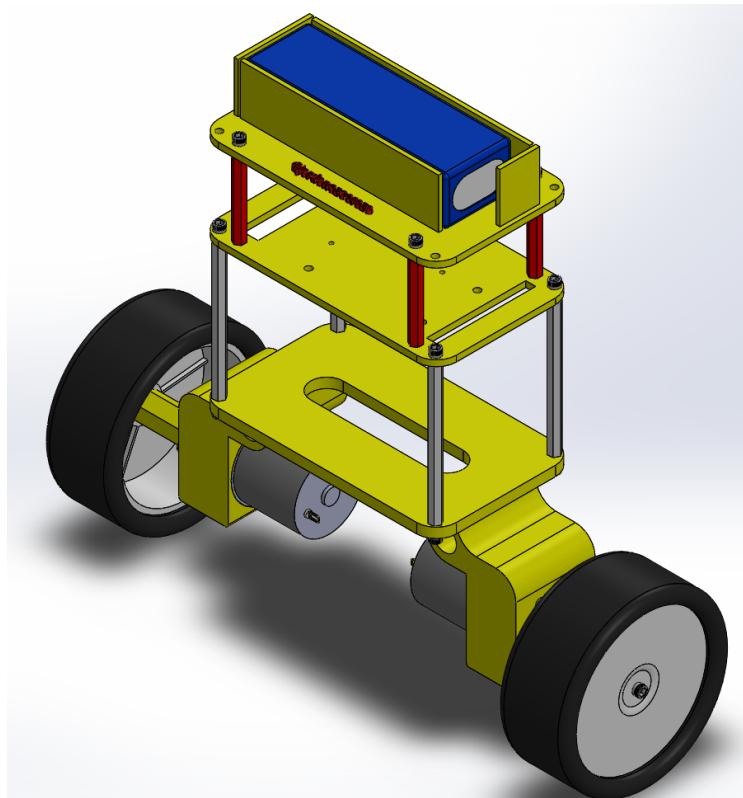


Figura 3.6: Perspectiva Balancing Robot.

Se tienen en cuenta diferentes aspectos en el diseño de esta estructura que se relacionan directamente con la física de un Robot Balancín y con ello, del péndulo invertido.

Como se argumenta en la sección 3.1, un sistema en reposo es estable cuando mas cercano está del plano horizontal su centro de masas. Si se tiene en cuenta que la naturaleza del sistema planteado es inherentemente inestable, se hace necesario conocer el mejor punto donde debe estar el centro de masas para permitir una mejor estabilización.

Asumiendo la caracterización del modelado matemático en [10], se asume por tanto que para conseguir una mayor facilidad en la estabilización, el centro de masas debe estar colocado por encima del punto medio del eje vertical de nuestro sistema. Por lo tanto, se ha de tener en cuenta el peso de todos los componentes para una colocación tal que permita lo anterior.

En la figura 3.7 se representa el cálculo con SolidWorks de este centro de masas donde sólo se han tenido en cuenta los elementos más pesados del sistema final, esto es, motores DC, batería, estructura mecánica y ruedas.

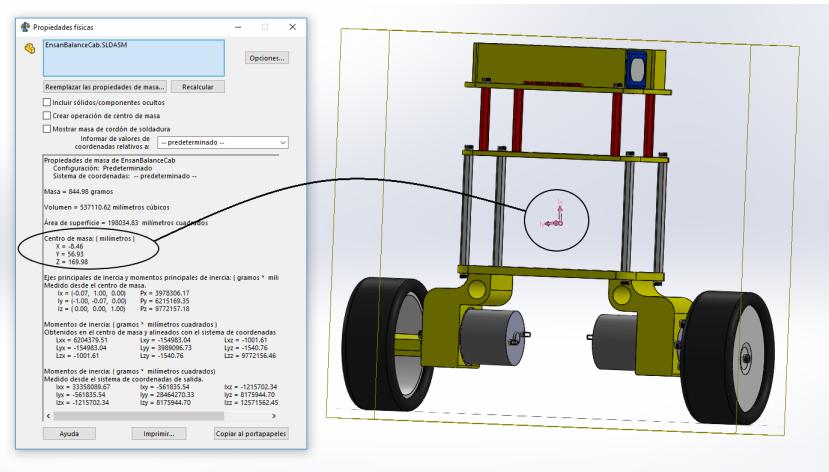


Figura 3.7: Centro de masas en sistema final.

### 3.3.2. Unidad de medida inercial MPU6050 en Arduino Nano

Es necesario un conocimiento constante del ángulo del sistema para su posterior análisis y corrección, para ello se hace uso del MPU6050 conectado mediante una comunicación I2C a un Arduino Nano.

El MPU6050<sup>[1]</sup> es una unidad de medida inercial (Inertial Measure Unit, IMU) con 6 grados de libertad (6DOF) fabricado por Invensense<sup>[2]</sup>. Cuenta con un acelerómetro y giroscopio y permite una comunicación tanto por SPI como por bus I2C. Para corregir algunos de los problemas de captación de datos planteados en la sección 2.4 incorpora un procesador interno (Digital Motion Processor, DMP) que ejecuta algoritmos de fusión de datos (Motion Fusion) para combinar las mediciones de los sensores internos evitando tener que realizar los filtros de forma exterior.

Debido a su bajo precio y gran calidad es uno de los IMUs más utilizados en la actualidad. En la figura 3.8 se muestra una imagen del MPU6050.

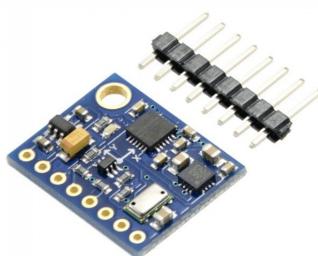


Figura 3.8: MPU6050 IMU.

### Pin out

En la figura 3.9 se muestra el diagrama esquemático de conexiones del MPU6050.

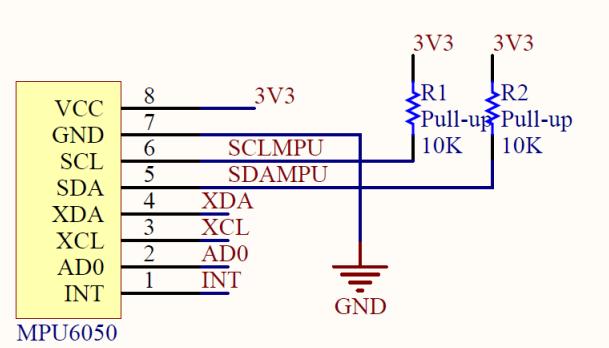


Figura 3.9: MPU6050 IMU.

Tiene una tensión de alimentación de 3.3V. El pin de reloj para la conexión I2C (Serial Clock Line, SCL) y el pin de datos (Serial Data Line, SDA) representan la conexión para el bus 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 cambia a 0x69h. El pin INT produce una señal en alta cuando el dato en cuestión está disponible por parte del MPU para ser capturado, y avisará por medio de una interrupción al Arduino Nano con el fin de poder ser obtenido.

### Programa para Arduino nano

Para la implementación en Arduino Nano se utiliza la librería desarrollada por Jeff Rowberg [\[\]](#). La razón del uso de esta librería es debido a que incorpora la utilización del DMP cuyas ventajas se representan en la figura 3.10

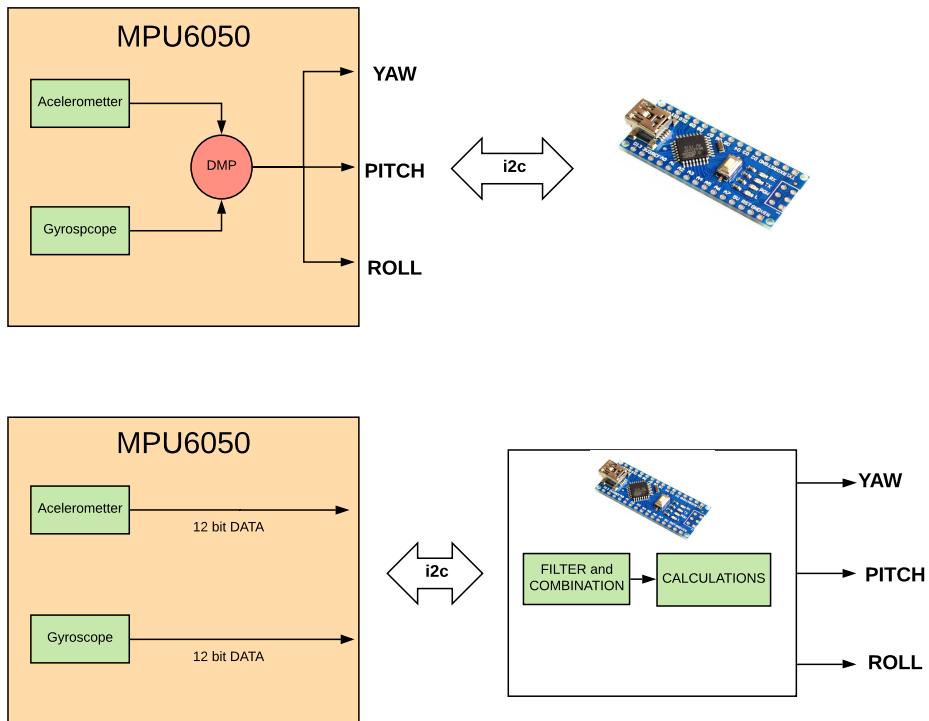


Figura 3.10: Ventaja de uso en la utilización de DMP.

Se exime al micro-procesador (Arduino Nano en este caso) de un complejo filtrado y cálculo con el fin de obtener los valores pitch, yaw y roll.

#### Valores del sensor.

Un ejemplo de los valores obtenidos por parte del sensor se muestran en la figura tal.

**foto de un ejemplo de los ángulos**

#### 3.3.3. Implementación 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 tam-

bién para el módulo de cámara OV7670 y el driver del motor, es conveniente y apropiado un circuito impreso que resuelva algunos problemas de ruido, cables excesivos, etc.

El circuito impreso alberga 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.

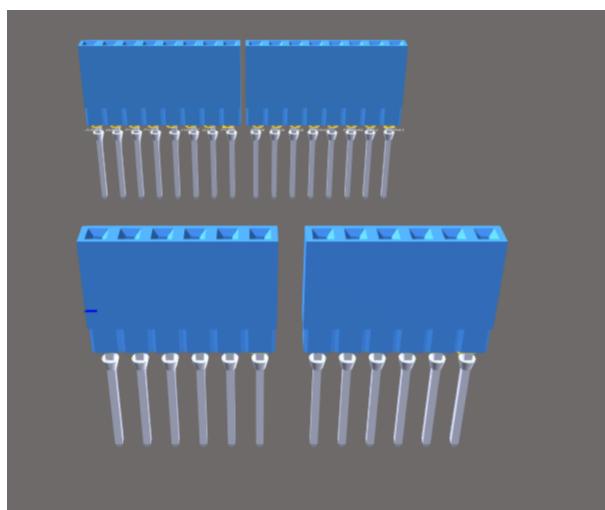


Figura 3.11: Pin headers para IceZum Alhambra II.

- 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:

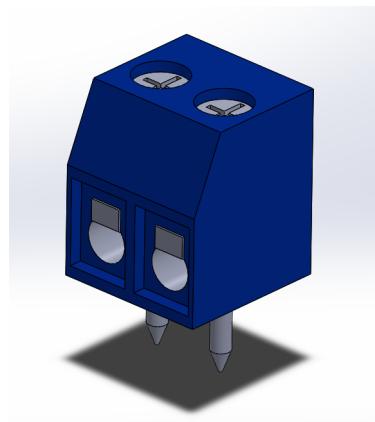


Figura 3.12: Conector GND y VCC.

- 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 ??
- Módulo de pines "header" para la conexión del MPU6050 anteriormente descrito (figura

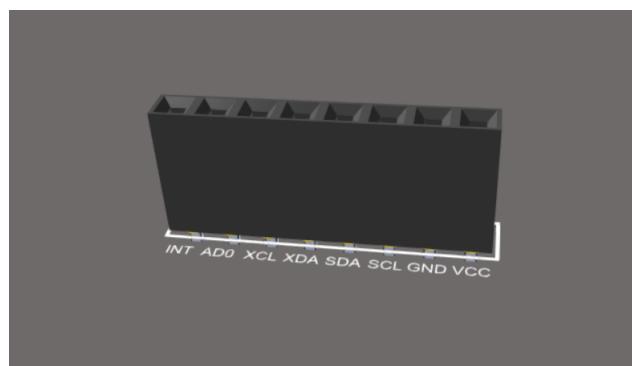


Figura 3.13: Módulo MPU6050.

- 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.<sup>a</sup>" 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.

En las figuras 3.15, 3.16, 3.17, 3.18 se incluye una representación en 3D del sistema final con todos los componentes añadidos.

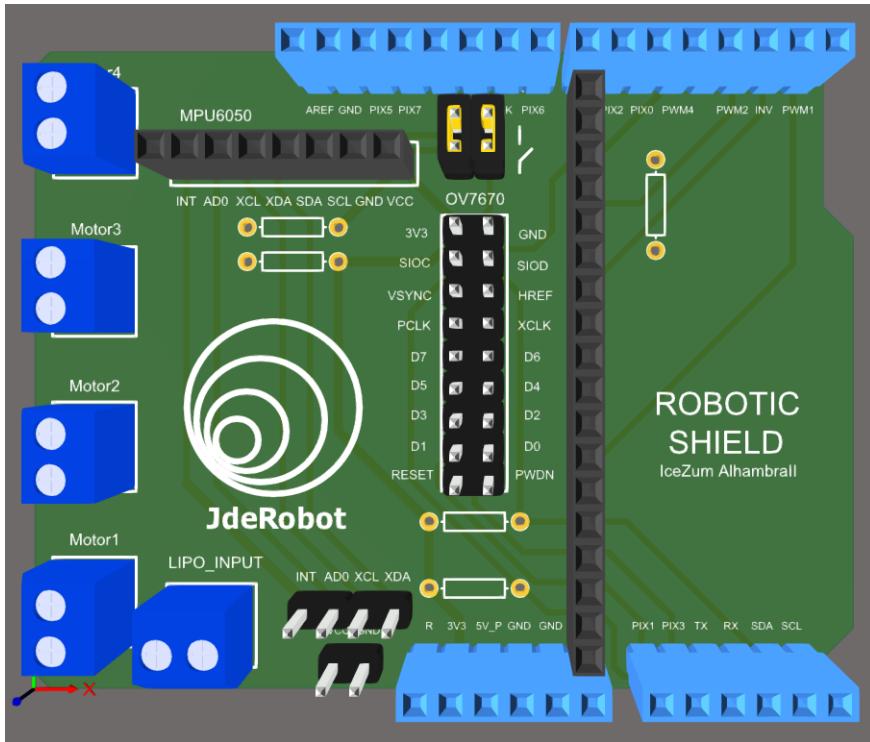


Figura 3.15: Vista en 3D de Shield para IceZum Alhambra II.

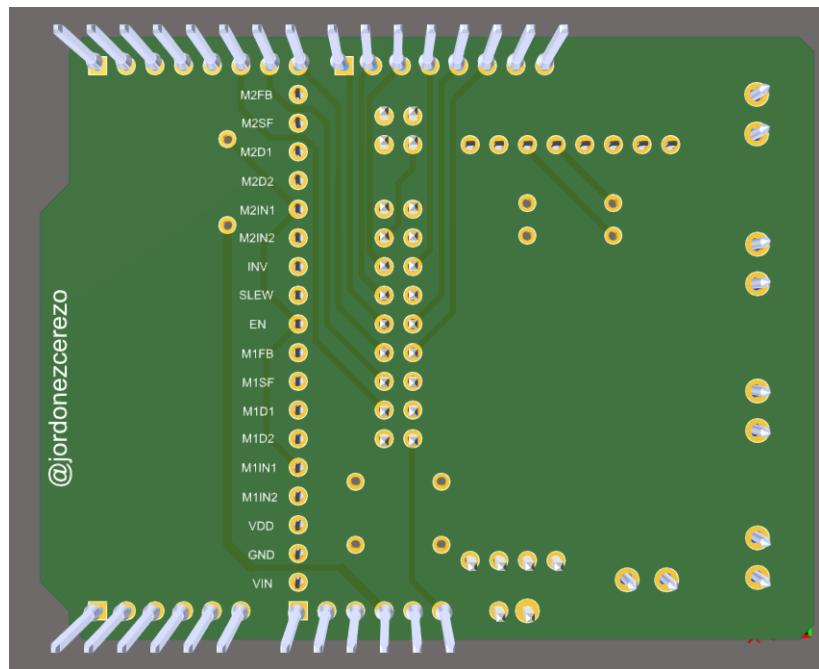


Figura 3.16: Vista en 3D de Shield para IceZum Alhambra II.

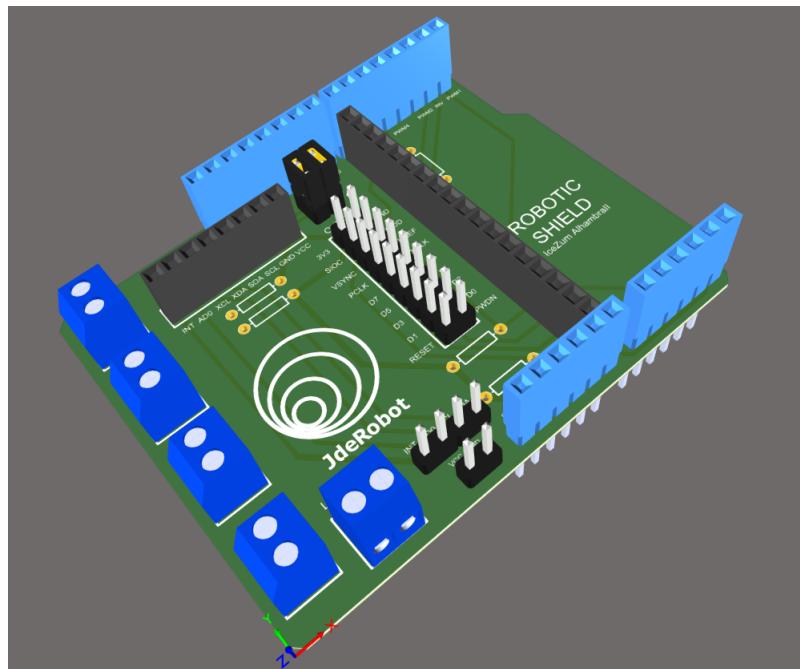


Figura 3.17: Vista en 3D de Shield para IceZum Alhambra II.

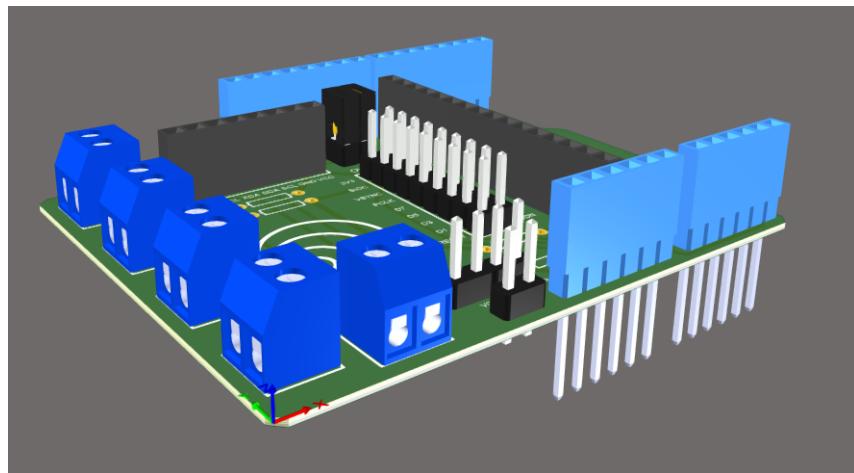


Figura 3.18: Vista en 3D de Shield para IceZum Alhambra II.

Para el desarrollo de una PCB con los requerimientos descritos 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 3.19:

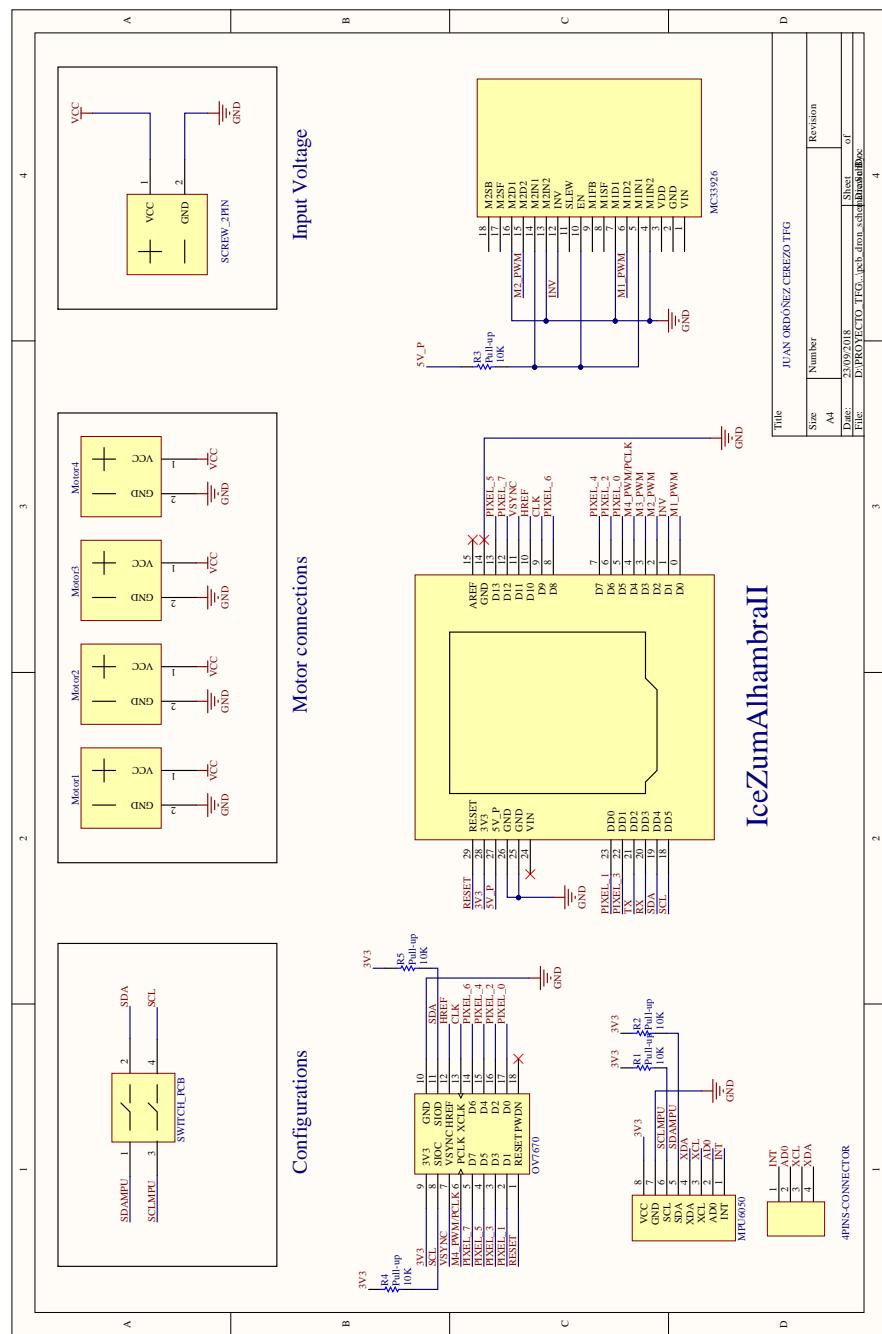


Figura 3.19: Esquemático Shield IceZum Alhambra II

**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 3.1:

$$I = K * dT^{0.44} * (W * H)^{0.725} \quad (3.1)$$

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 .°zz 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 buena referencia 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
<b>Top Layer</b>	Signal
Dielectric 1	Dielectric
<b>VCC</b>	Signal
Dielectric 2	Dielectric
<b>GND</b>	Signal
Dielectric 4	Dielectric
<b>Bottom Layer</b>	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 3.20.

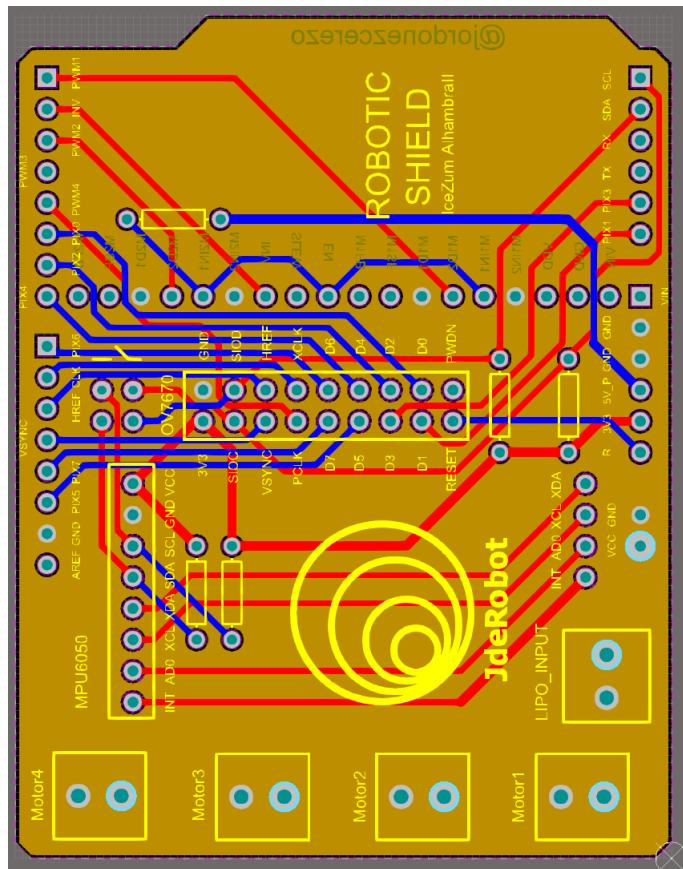


Figura 3.20: Vista de la composición de capas en Altium.

### 3.3.4. Implementación 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.

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 3.21.

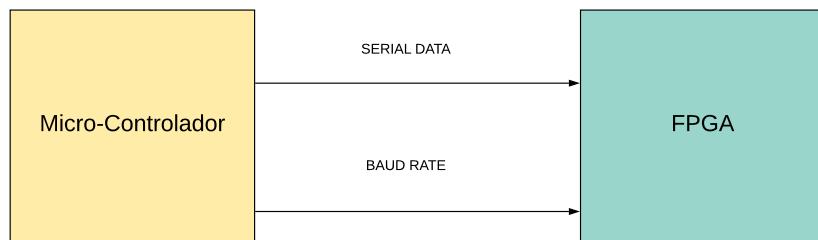


Figura 3.21: Pines hardware de coexistencia micro-FPGA.

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 mostrado en la 3.22.

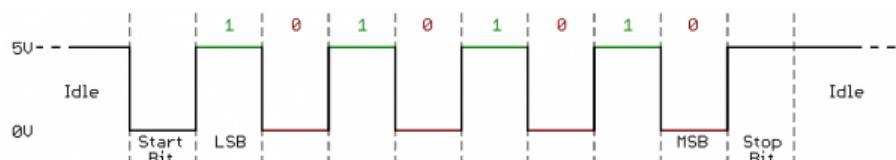


Figura 3.22: Ejemplo de comunicación serie.

El ángulo tiene que ser conocido en la IceZum-Alhambra para la toma de decisiones tanto en dirección como velocidad de los motores. Es necesario por tanto una comunicación micro-controlador/FPGA (figura 3.23).



Figura 3.23: Separación de procesos micro-FPGA.

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.

Se hace una separación por tanto de dos partes de la comunicación, desde el punto de vista del micro-controlador, y desde el punto de vista de la FPGA. Ambos serán explicados más específicamente en la subsección 3.3.4 y 3.3.4

#### Desde el punto de vista del micro-controlador:

Para el desarrollo de este sub-capítulo se parte de la base de que el ángulo actual ya ha sido obtenido en el micro-controlador. Así, y para la claridad del lector, un esquemático interno del arduino nano es mostrado en la figura 3.24. En este apartado y con respecto a la parte del micro-controlador, solo se analiza la parte sombreada.

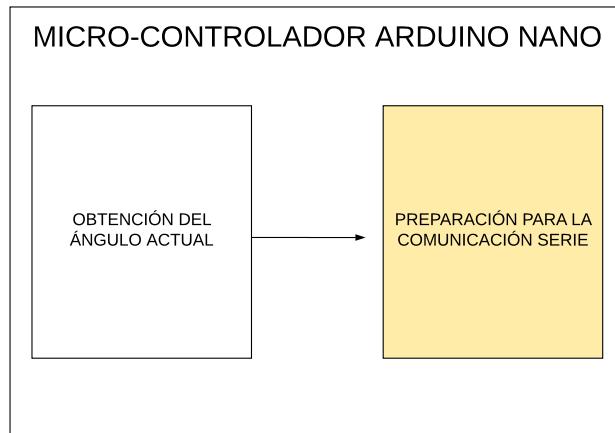


Figura 3.24: Diagrama interno Arduino Nano.

El diagrama de flujo sobre el cuál se basa el código en C del micro-controlador se representa en la figura 3.25.

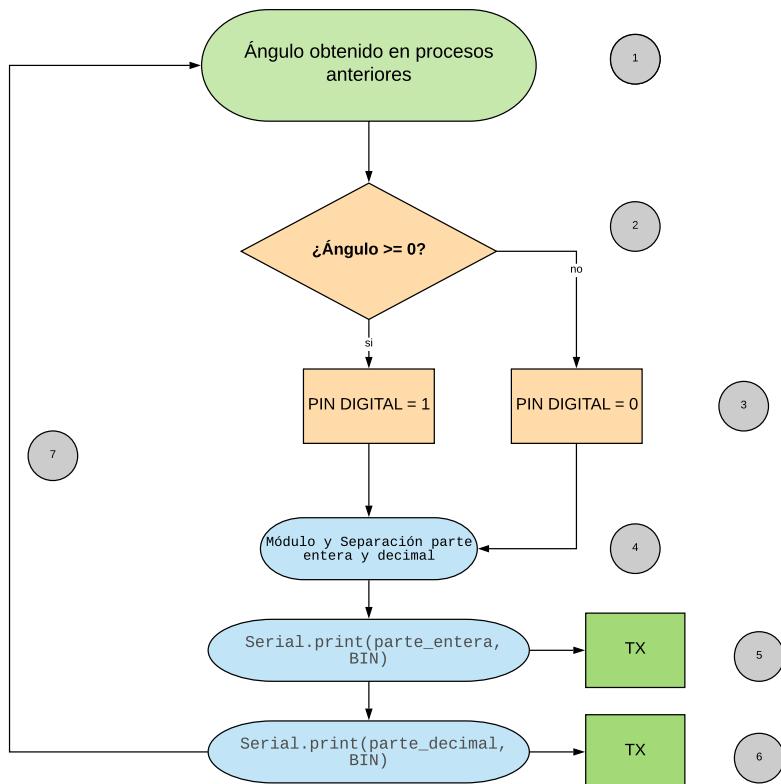


Figura 3.25: Envío de ángulo usando el puerto serie.

A continuación se explican los aspectos mas relevantes:

- **1.-**Se parte de la base de que el ángulo ya ha sido obtenido y es correcto. En la sección 3.3.2 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 (3.26). 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.

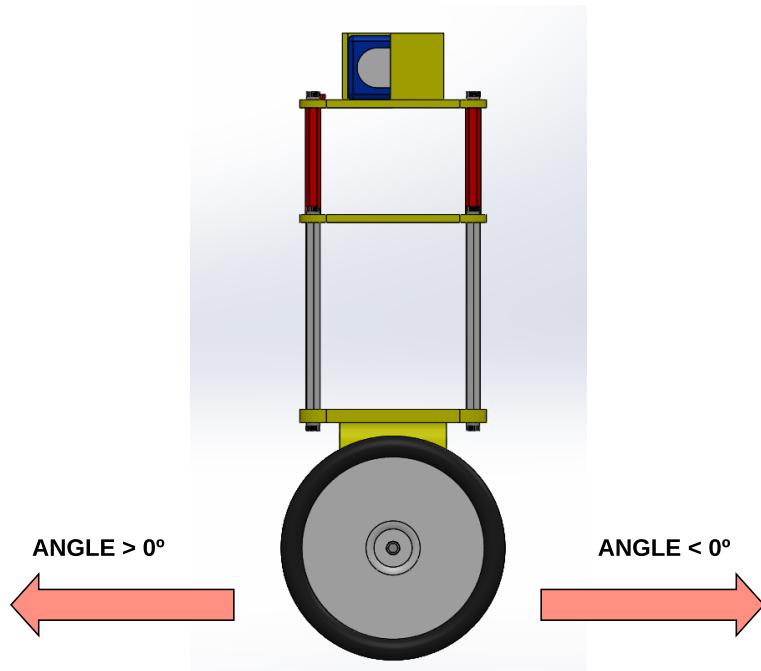


Figura 3.26: Corrección de la dirección en Balancing Robot.

- **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 (por contabilizar), 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 ??

#### **captura del envio del angulo**

#### **Desde el punto de vista de la FPGA:**

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 (aspecto en IceStudio 3.27)es necesario conocer de antemano la velocidad de la transmisión por parte del micro-controlador, la cuál se ha fijado en 16200 baudios.

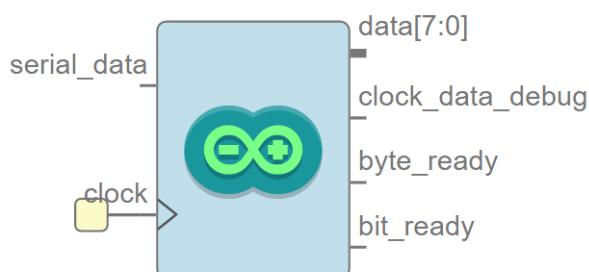


Figura 3.27: Apariencia del módulo interfaz de Arduino Nano en IceStudio.

Se detallan las diferentes entradas y salidas del módulo anterior:

- `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 esta 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 está implementada mediante dos máquinas estados con sus correspondientes listas de sensibilidad, y el diagrama de flujo se representa en la figura 3.28.

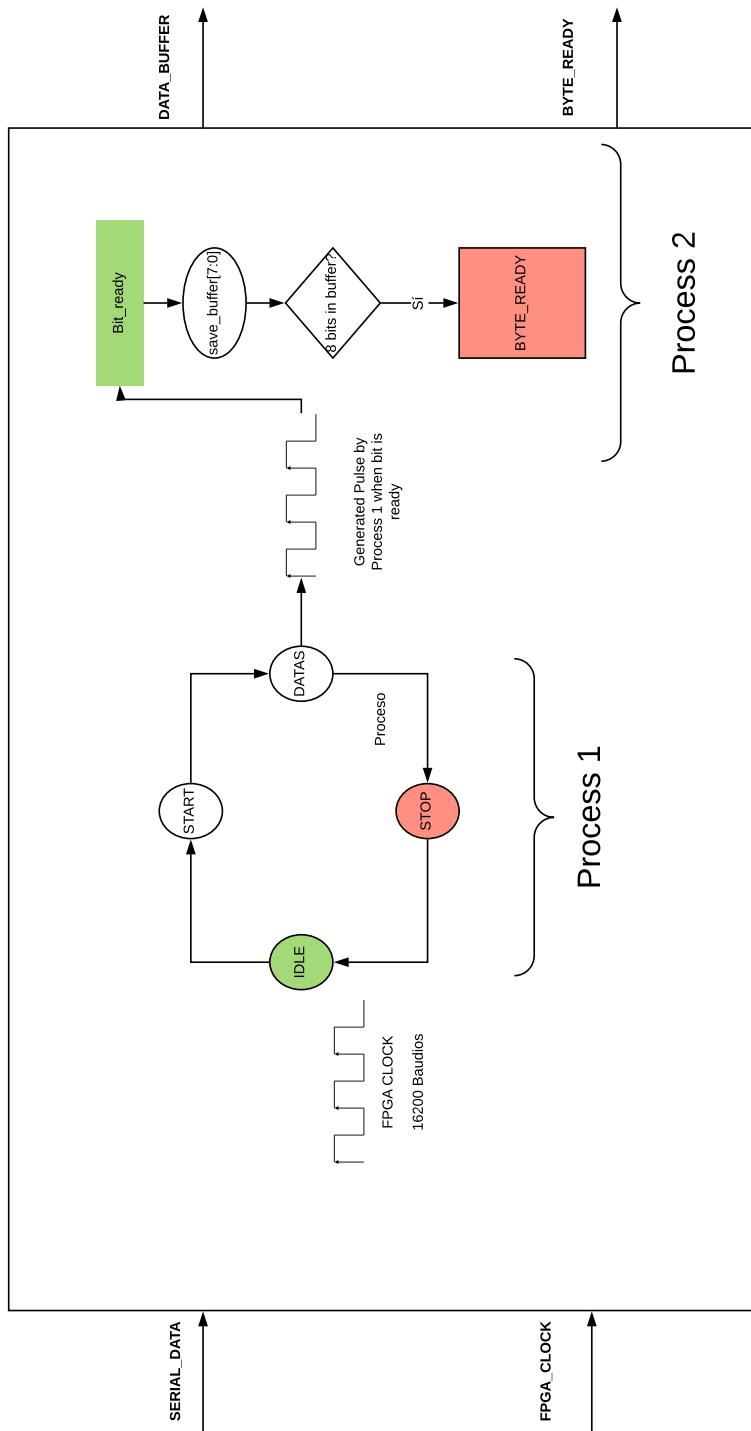


Figura 3.28: Diagrama de flujo de la interfaz para Arduino.

Se hace uso de dos procesos bien diferenciados.

**Proceso 1:** Este proceso solo dota al sistema siguiente del momento exacto en el cual puede capturar un bit y guardarla 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 1 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. El aspecto de este módulo en IceStudio se muestra en la figura 3.29.

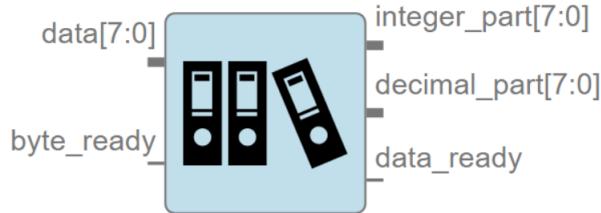


Figura 3.29: Módulo para ordenar datos provenientes de Arduino.

Como entradas:

- `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, como salidas se encuentran:

- `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.

Como se ha profundizado anteriormente, el diagrama de flujo del código en Verilog que implementa el anterior comportamiento se representa en la figura 3.30.

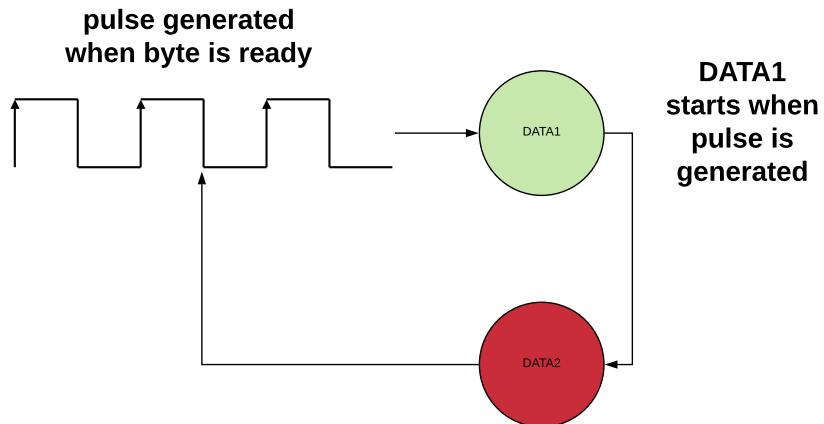


Figura 3.30: Diagrama de flujo modulo ordenación de bytes.

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

Cada vez que se active el flag que indica que un byte está 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 captura 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).

El sistema final de comunicación entre Arduino y IceZum Alhambra desde el punto de vista de la FPGA se representa en la figura 3.31.

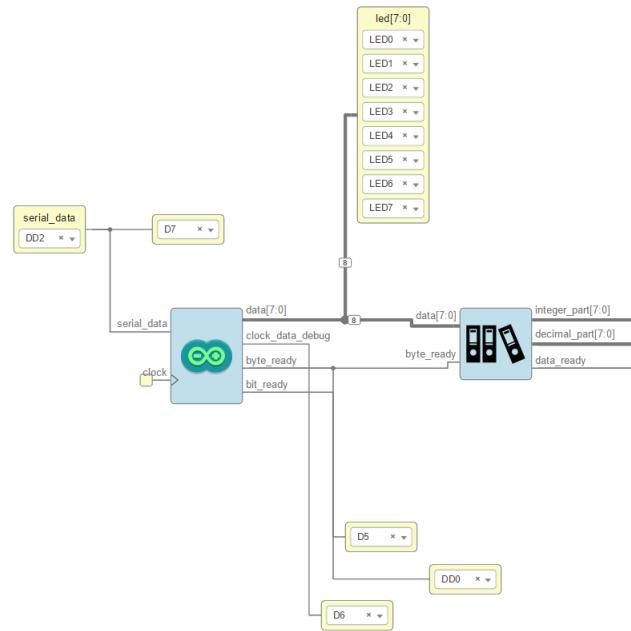


Figura 3.31: Comunicación entre Arduino y IceZum Alhambra en IceStudio.

### 3.3.5. Control PID en IceZum Alhambra

Como ha sido explicado en la sección 2.7, un controlador PID puede ser utilizado de manera sencilla para controlar la estabilidad del sistema. Una de las ventajas en el uso de este tipo de controladores recae en su fácil implementación.

#### Controlador P

El diagrama de flujo que caracteriza el comportamiento del controlador P se muestra en la 3.32.

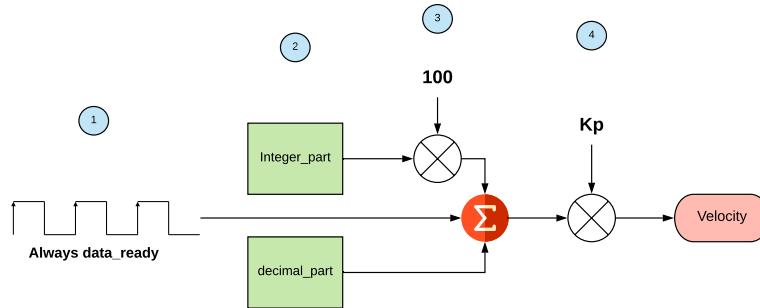


Figura 3.32: Diagrama de flujo control P.

Debido a su importancia, se explica su funcionamiento brevemente:

- 1.-El proceso completo será actualizado cada vez que se tenga un nuevo dato, esto es, cada vez que cambie el ángulo del sistema en este caso.
- 2.-Como entradas se tienen tanto la parte decimal del ángulo como la parte entera.
- 3.-Tanto la parte decimal como la entera están representados como datos de 8 bits sin signo. Así, y para darle más importancia a la parte entera, existe la opción de dividir la parte decimal por 100 o multiplicar la parte entera por 100. Con la primera de las opciones no se obtiene un buen comportamiento debido al trato digital de la coma flotante, por lo que se opta por la segunda opción.
- 4.- Para finalizar se suman las dos componentes entera y decimal y se multiplica por la constante Kp, definida como parámetro para poder ser cambiada dinámicamente.

Su representación en IceStudio tendrá el aspecto de la figura 3.33.

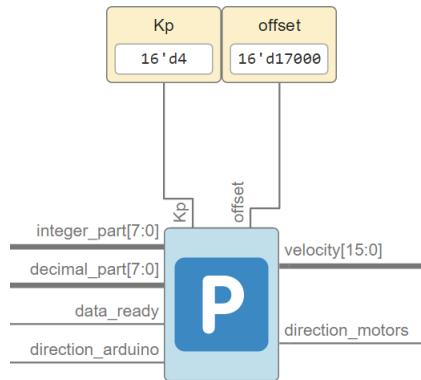


Figura 3.33: Aspecto en IceStudio del módulo de control P.

### Controlador D

En cuanto al controlador D, el diagrama de flujo de su implementación en Verilog se muestra en la 3.34.

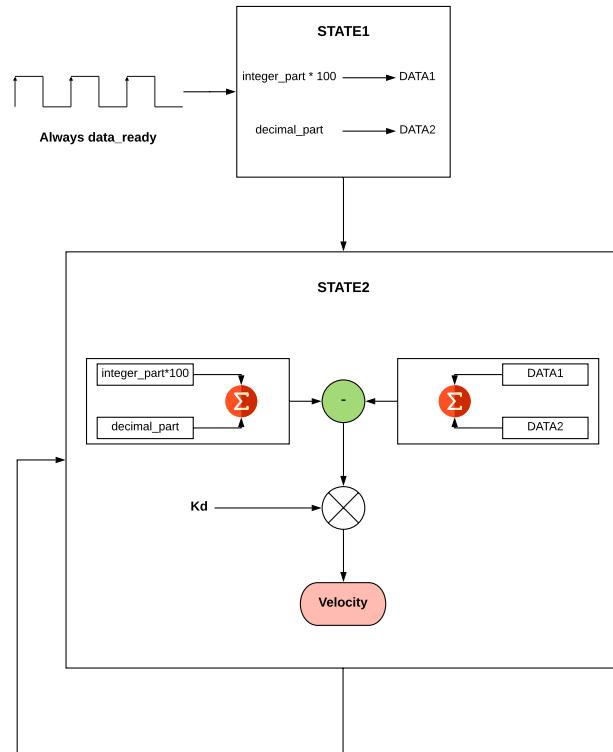


Figura 3.34: Diagrama de flujo control D.

Su implementación está compuesta por una máquina de estados con dos estados, y los cuales cambiarán cada pulso en alta de la señal *data\_ready*, esto es, cambiará siempre que un nuevo ángulo esté disponible.

Si se recuerda, el controlador D basa su funcionamiento en la predicción de los errores futuros. La acción de control derivativa genera una señal de control proporcional a la derivada de la señal del error.

Se va llevando a cabo por tanto una resta (derivada del error en el tiempo) entre el error actual y el error pasado. Su resultado se multiplica por la constante *Kd*.

Su representación en IceStudio se representa en la 3.35.

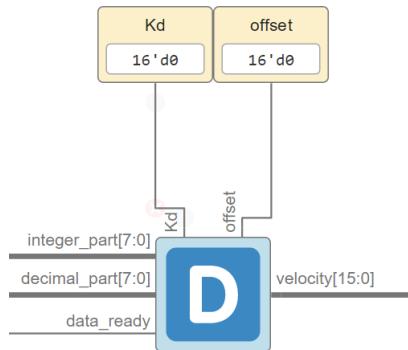


Figura 3.35: Aspecto en IceStudio del módulo de control D.

#### Controlador PD

Referido al sistema de realimentación por lazo cerrado analizado en la sección 2.7, es importante mostrar el resultado final del aspecto del presente proyecto en IceStudio, haciendo una comparativa directa entre éste (figura 3.36 ) y la figura 2.28.

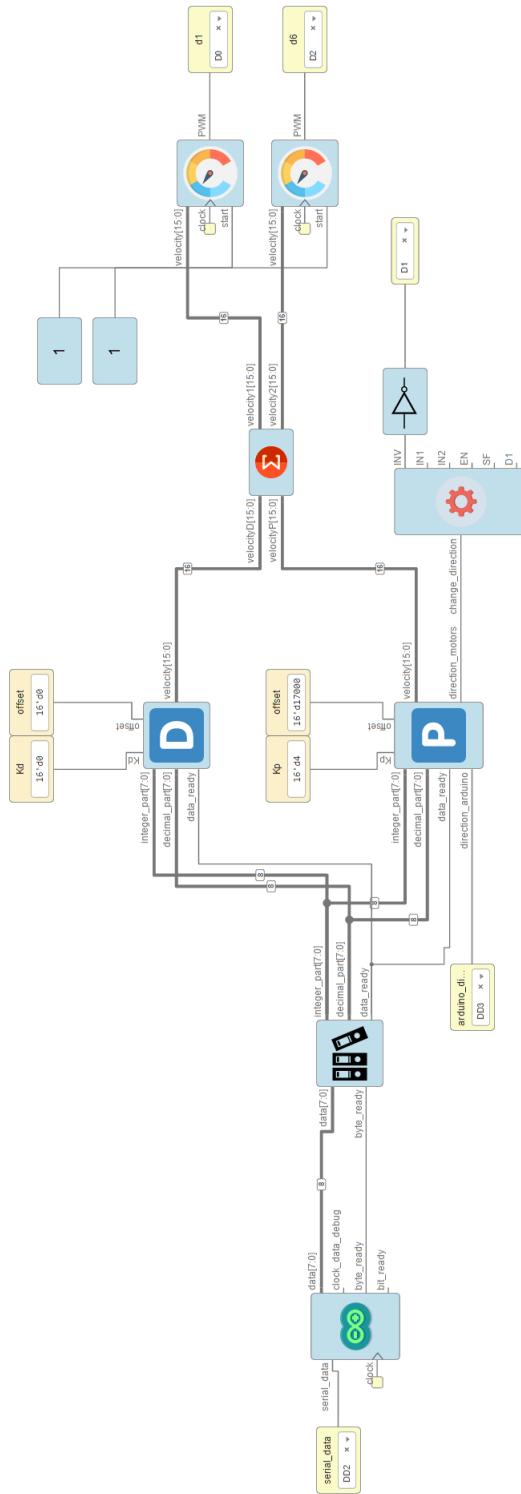


Figura 3.36: Aspecto final en IceStudio del Balancing-Robot.

### 3.3.6. Controlador de motores

Tener un modulo 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 3.37.

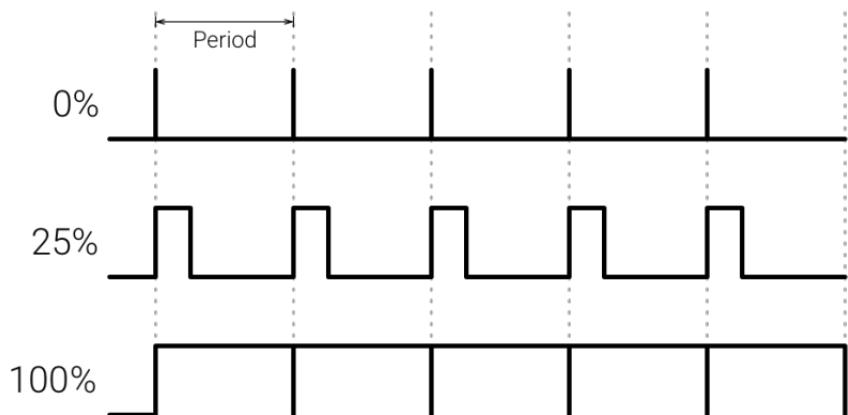


Figura 3.37: Ejemplo de señal PWM con duty diferente.

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 cuál un nivel lógico alto es abierto y un nivel bajo cerrado. Si se consigue variar el tiempo en el cuál al motor le está llegando carga y el tiempo en el cuál no le llega corriente, se puede controlar su velocidad.

Las características de una señal PWM son:

- D = ciclo de trabajo.
- $\tau$  = tiempo en que la función es positiva (ancho de pulso).
- T = periodo de la función.

El driver del motor utilizado para el control de los motores DC (MC33926 [figura 3.38]), necesita una serie de configuraciones para poder trabajar acorde a nuestras necesidades y las cuales pueden ser obtenidas de su datasheet [1]. El diagrama final de entradas y salidas así como las conexiones necesarias se detallan en el esquemático de la figura 3.39.

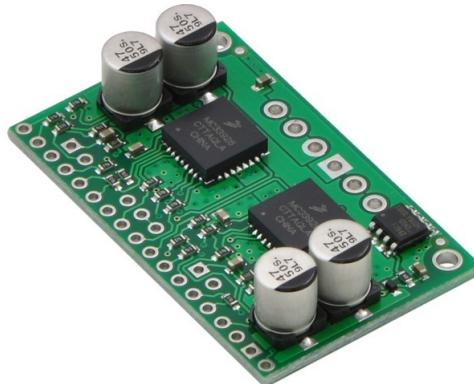


Figura 3.38: MC33926 para el control de los motores DC.

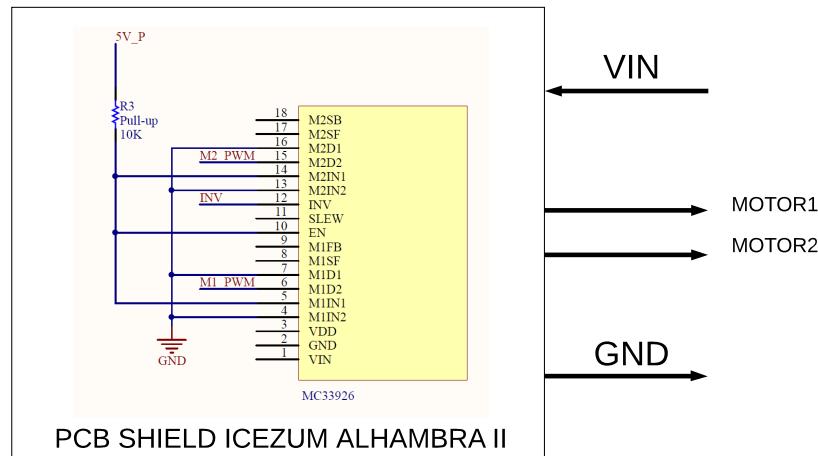


Figura 3.39: Esquemático de conexiones MC33926.

### Control PWM

Los velocidad de los motores DC se controlan mediante un generador de PWM conectado a su patilla 15 o M2D2 de la figura 3.39 para uno de los motores y otro generador de PWM conectado a la patilla 6 o M1D1 del driver para motores DC. Este módulo generador tiene el aspecto de la figura 3.40 en IceStudio.



Figura 3.40: Aspecto en IceStudio del generador de señal PWM.

El diagrama de bloques de su funcionamiento se expone en la figura 3.41.

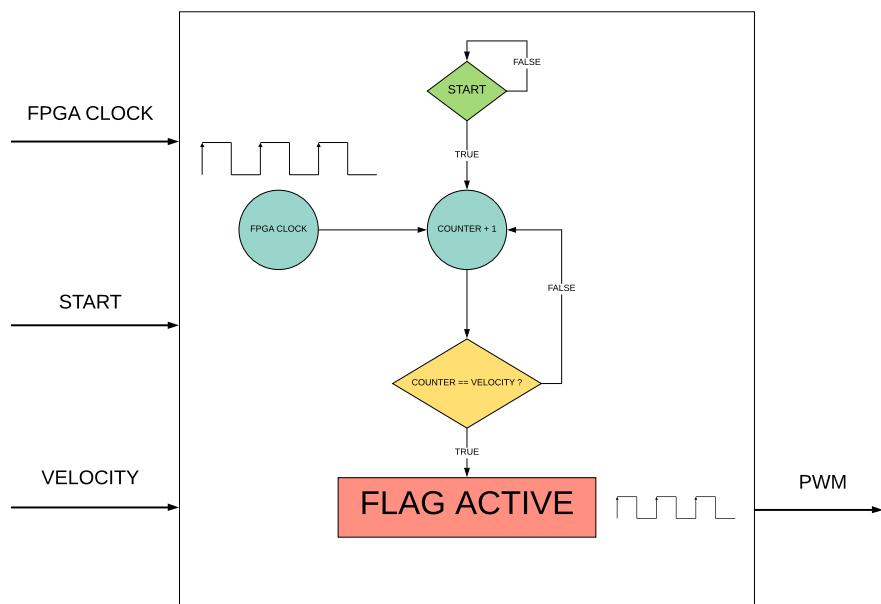


Figura 3.41: Envío de ángulo usando el puerto serie.

Basa su funcionamiento en un contador de pulsos de entrada del reloj de la FPGA. Un registro de velocidad impondrá cuantos pulsos tienen que ser contados antes de poner en alta una señal de salida, el PWM buscado. Como entradas se encuentran:

- **FPGA CLOCK:** Es el reloj de la FPGA de 12Mhz y sobre el cuál se realizará el conteo de pulsos.
- **START:** Simplemente es una señal común a todo el sistema, funcionando como switch para dar un inicio general.
- **VELOCITY:** Registro que marca cuántos pulsos de reloj tienen que ser contados antes de activar la señal de salida PWM.

Como salidas:

- PWM: Señal de tipo PWM con un duty proporcional a la velocidad impuesta.

### 3.3.7. Sistema de alimentación

Como todo sistema electrónico, es necesario una fuente de alimentación que permita el correcto funcionamiento de todos los componentes.

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 qué tipo de batería es adecuada.

A continuación se nombran los diferentes tipos de batería con los 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). Mejoran 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 descargaría 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	Corriente
MC33926		
Motor DC		
MPU6050		
IceZum Alhambra II		
Arduino Nano		

Tabla 3.2: Tabla eléctrica de componentes.

Se diferencian dos subsistemas independientes en cuanto a la alimentación se refiere:

- Alimentación para los motores DC y Arduino Nano.
- Alimentación de la propia tarjeta IceZum Alhambra II y demás componentes.

#### Alimentación de los motores DC y Arduino Nano:

Para la alimentación de los motores DC y el Arduino Nano se hace uso de una batería LIPO de 11.1V y 2200maH como en la figura 3.42. Como se representa en el esquemático final de la shield desarrollada (figura 3.19), la batería se conectará a dicha shield, encargada esta de alimentar tanto a los motores DC como al Arduino Nano.



Figura 3.42: Batería LIPO 11.1V y 2.2A.

#### Alimentación IceZum Alhambra II y demás componentes:

Para la alimentación de la IceZum Alhambra II y el resto de componentes que forman el sistema final, se hace uso de una batería LIPO de 3.7V y 4mAh como la de la figura 3.43.



Figura 3.43: Batería LIPO 3.7V y 4mAh.

#### 3.3.8. Materiales y coste del prototipo

En la tabla 3.3 se desglosa el coste total del prototipo del sistema planteado a lo largo del capítulo 3.

MATERIAL	CANTIDAD	COSTE UNITARIO (€)	COSTE TOTAL (€)
IceZum Alhambra II	1	60	60
Arduino Nano	1	8	8
Separador Hexagonal Nylon 10 mm	4	0.20	0.80
Separador Metálico Hexagonal M3 25 mm	8	0.25	2
Separador Metálico Hexagonal M3 50 mm	4	0.41	1.64
Tornillo Nylon M3	16	0.09	1.44
Tuerca Nylon M3	8	0.05	0.40
Tornillo Nylon M1	4	0.09	0.39
Tuerca Nylon M1	8	0.05	0.40
Placa de Circuito Impreso 4 capas	1	5	5
Rueda 7 cm	2	7.90	15.80
Motor DC	2	24.95	49.9
Driver Motor DC Dual MC33926	1	30	30
IMU MPU6050	1	2.50	2.50
Estructura mecánica 3D	1	20	20
Screw 3.5 mm	5	0.80	4
Cable 5-pin	1	0.82	0.82
Regleta PIN 2,54 mm 4 contactos Macho	1	0.08	0.08
Regleta PIN 2,54 mm 8 contactos Shield	2	0.54	1.08
Regleta PIN 2,54 mm 6 contactos Shield	2	0.58	1.16
Juniper 2,54 mm Puente de conexión	2	0.08	0.16
Regleta PIN 2,54 mm 10 contactos Hembra	2	0.10	0.20
Regleta PIN 2,54 mm 10 contactos Macho	5	0.10	0.50
Resistencia 4K7 5%	3	0.21	0.63
Estañío	1	5.50	5.50
Metro de cinta Termoretractil	2	1.50	3
Metro de cable 12 AWG	1	0.90	0.90
Batería Lipo 11.1 V 2200 mA	1	19.95	19.95
Batería Lipo 3.7 V 5 mA	1	4.20	4.20
<b>COSTE TOTAL PROTOTIPO:</b>			<b>240.45 €</b>

Tabla 3.3: Coste total del prototipo Balancing-Robot.

## 3.4. Experimentos

### 3.4.1. VGA Module

### 3.4.2. Protocolo I2C

### 3.4.3. Controlador motor brushless

## Capítulo 4

# Cuadricoptero con vision artificial

En el presente capítulo se pretende abordar el diseño y la implementación de un cuadricóptero con la capacidad para el seguimiento de un objeto, haciendo uso para ello de una FPGA como parte fundamental, y algunas otras herramientas que se expondrán a lo largo de esta sección.

### 4.1. Diseño

En la figura se presenta un esquema con un alto nivel de abstracción en el que se da una primera aproximación del funcionamiento del sistema general.

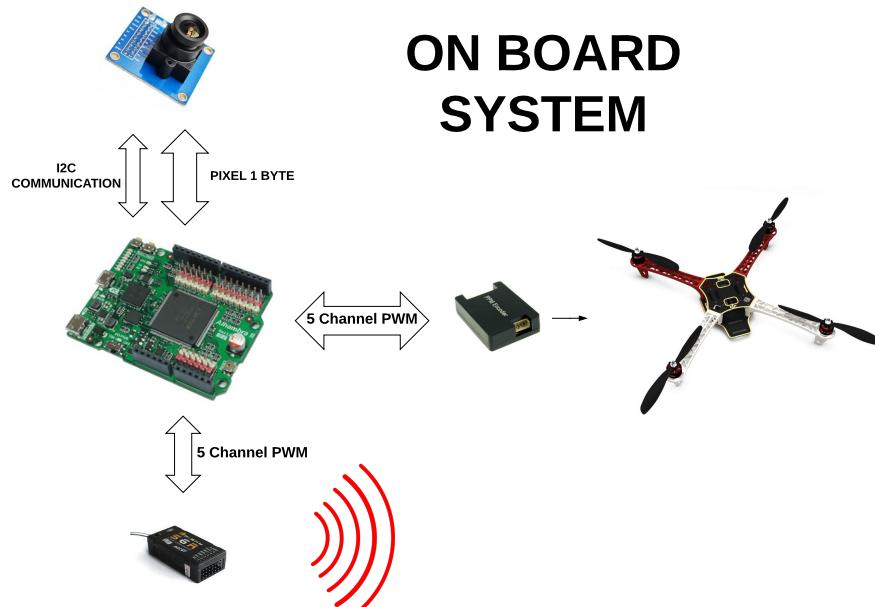


Figura 4.1: Diseño a alto nivel del cuadricóptero con visión.

Se diferencia en este diseño la parte del control de la estabilización y la parte de percepción. En cuanto a la parte de la percepción, una cámara de bajo coste se conectará a la FPGA la cuál deberá implementar todo el algoritmo de reconocimiento del objeto. Previo a ello, y mediante protocolo i2c la cámara sera configurada de modo que facilite ese reconocimiento. Posteriormente y como salida de este algoritmo de reconocimiento la FPGA generará 4 canales PWM correspondientes al yaw, pitch, roll y altitud. Se hace uso de un PPM encoder, el cuál transforma estas 8 señales PWM en un canal PPM, entrada del sistema de estabilización del cuadricóptero "Pixhawk".

## **4.2. Implementación de la percepción**

El módulo OV7670 [1] cuenta con un sensor de imagen CMOS VGA OV7670, que permite trabajar a un máximo de 30 cuadros por segundos y una resolución de 640x480 píxeles. Es un sistema en chip (System on Chip, SoC) que es capaz de realizar procesamiento de imágenes, como: control de exposición, gamma, balance de blancos, saturación de color, control de tono. Estos parámetros se pueden configurar mediante la interface SCCB (Bus de Control de Cámara Serial).

Algunas de las características más importantes se presentan a continuación, y las cuales pueden ser obtenidas de su datasheet:

- Voltaje de operación 3.3 VDC
- Corriente Sleep 20  $\mu$ A
- Transmisión de datos 8bits en paralelo
- Interface de control estándar SCCB, compatible con I2C
- Lente óptico de 1/6"
- Ángulo de visión (FOV): 25°C
- Resolución 640x480 VGA
- Sensibilidad 1.3V / (Lux-sec)
- Ratio Señal-Ruido (SNR): 46 dB
- Alta sensibilidad en ambientes de poca iluminación
- Bajo voltaje, adecuado para aplicaciones portátiles

Una imagen del módulo OV7670 se representa en la 4.2.



Figura 4.2: Cámara OV7670.

La configuración de la cámara se realiza a través de una interfaz SCCB [], muy parecida a una comunicación I2C y cuyos registros se pueden encontrar en el datasheet. Por tanto, se ha de elegir una configuración adecuada para este propósito:

- Se elige un tamaño de ventana de 640x480 debido a los pocos recursos de la tarjeta FPGA utilizada.
- El tipo de datos de salida será RGB, por la simplicidad en el uso. Obtenremos por tanto dos bytes por cada píxel capturado en el siguiente orden:

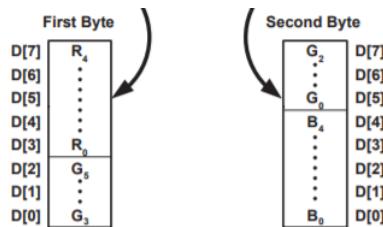


Figura 4.3: Formación de píxel en OV7670.

Para conseguir esa configuración es necesario una comunicación I2C con el módulo OV7670 que permite escribir en determinados registros.

El aspecto en IceStudio de esta escritura en I2C tiene el aspecto de la figura 4.4.

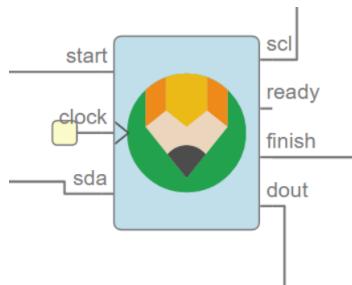


Figura 4.4: Aspecto en IceStudio del módulo de escritura I2C.

Debido a que ha sido un módulo desarrollado íntegramente para esta aplicación, se representa el diagrama de flujo en la figura 4.5

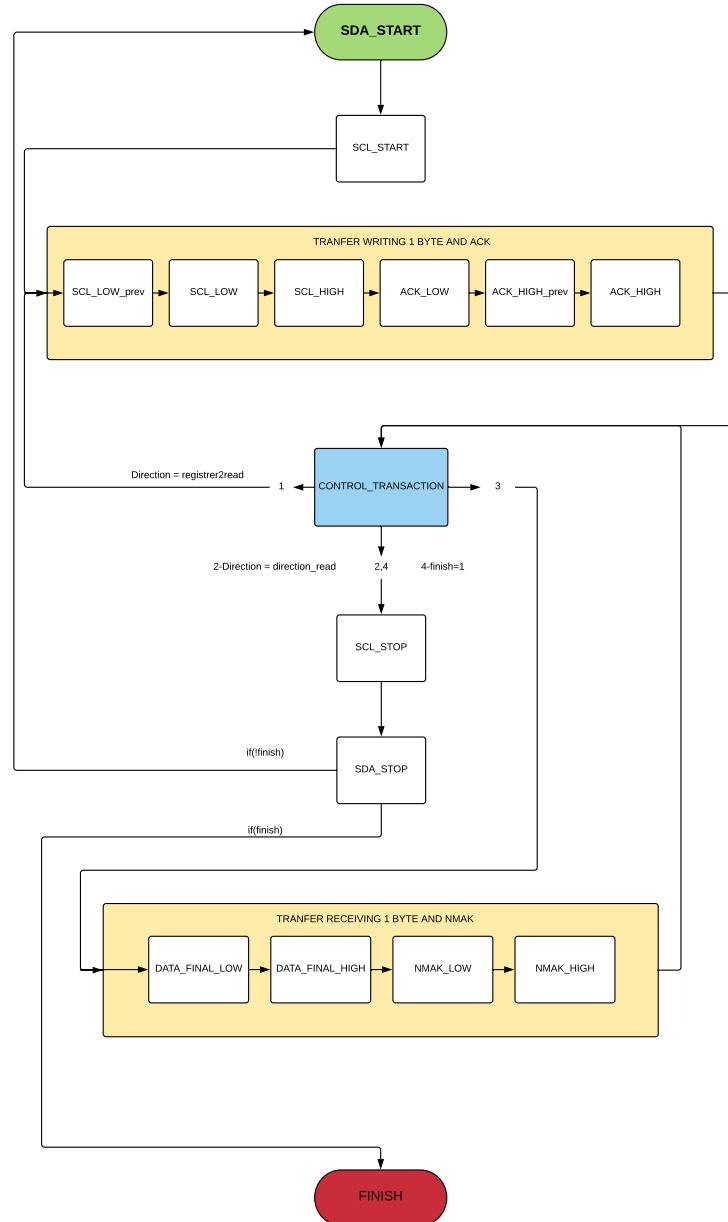


Figura 4.5: Diagrama de flujo de la escritura I2C.

Está compuesto por una máquina de 13 estados en los que cada uno tiene una función fundamental para conseguir el resultado deseado. Hay que trabajar a muy bajo nivel para evitar errores en la transmisión y dotar al sistema de las herramientas necesarias para la detección de posibles anomalías.

En una comunicación I2C es importante que tanto maestro como esclavo compartan los buses scl y sda, los cuáles estarán en alta siempre excepto cuando alguno de los dos anteriores impone su paso a bajo nivel de voltaje, tierra en este caso. I2C basa su funcionamiento en reconocer en todo momento estos niveles de voltaje por lo que hay que asegurarse de que siempre exista la condición de que si el bus está libre, el nivel de voltaje es alto. Para ello, ambos buses suelen tener una conexión a una resistencia de pull-up que facilite esta características y evite en todo momento posibles glitches de reloj, mala referencia de tierra, ruido, etc. El esquemático de conexiones de la cámara será por tanto el representado en la 4.6.

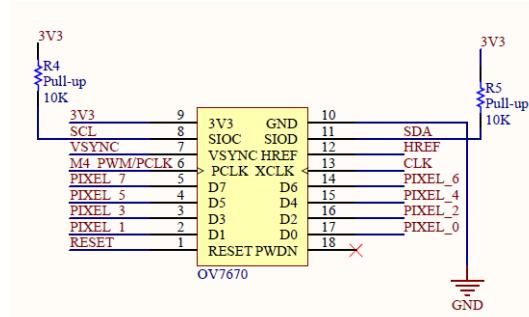


Figura 4.6: Esquemático para cámara OV7670.

Uno de los problemas más importantes en el desarrollo de este módulo recae el gobierno temporal de estos buses por parte del esclavo y el maestro. Para entenderlo se describe el ejemplo 4.1.

**Ejemplo 4.1** Se requiere en primera instancia que el maestro imponga el bus SDA a alto nivel de voltaje. A continuación, el maestro debe quedarse a la espera de una contestación por parte del esclavo, el cuál es un dispositivo no controlado y externo a nuestro sistema. Cuando el esclavo intente escribir en el bus, se encontrará un nivel de voltaje impuesto por el maestro, y el cuál no se podrá cambiar.

Para corregir el problema expuesto en el ejemplo 4.1 el dispositivo a la escucha deberá imponer un estado de alta impedancia eléctrica con respecto al bus. Se hace uso por tanto de un buffer triestado como el que se representa en la figura 4.7 y que permite ese comportamiento de manera controlada.

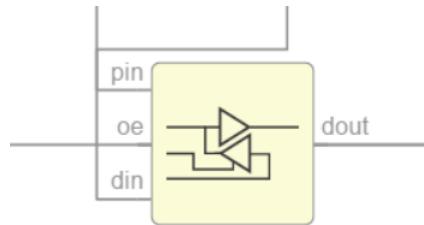


Figura 4.7: Buffer triestado para bus I2C.

Un ejemplo de la escritura en I2C en el módulo OV7670 se representa en la figura 4.8.



Figura 4.8: Ejemplo de escritura I2C en OV7670.

#### 4.3. Implementación del control

#### 4.4. Experimentos



## Capítulo 5

# Conclusiones y trabajo futuro

Con esté capitulo se da por finalizada la documentación del proyecto. A lo largo de esta memoria se ha expuesto el diseño e implementación de dos sistemas robotizados utilizando para ello FPGAs libres. Los puntos o cuestiones clave desarrolladas en este proyecto se pueden resumir en las siguientes:

- Diseño e implementación de la estructura mecánica del robot Balancín mediante el uso de SolidWorks e impresión 3D.
- Diseño e implementación de una PCB shield para IceZum Alhambra II mediante Altium Designer.
- Implementación hardware mediante Verilog del controlador PID y de los sistemas que componen el robot balancin.
- Diseño e implementación del protocolo de reconocimiento de pelota en cuadricóptero.
- Lectura de cámara de bajo coste mediante FPGA y Verilog.
- Diseño e implementación del control de un cuadricóptero.

Se han cumplido con los objetivos planteados al inicio del trabajo, sin embargo, algunos son los retos a los que no se les ha podido dar cobertura o que han surgido durante el desarrollo del mismo. Los principales retos que se plantean para futuros trabajos se exponen a continuación:

- Caracterizar de manera más exacta el modelo matemático del robot balancín para una mejora en la estructura mecánica.
- Integrar un controlador PID (hasta ahora el controlador es sólo PD) o hacer uso de otros controladores que mejoren la estabilidad.
- Mejora del protocolo I2C desarrollado para permitir errores en la transmisión.
- Mejora en el protocolo de comunicación FPGA-Micro para aumentar la velocidad y permitir comunicación bidireccional.

- Mejora en el reconocimiento de la pelota roja para permitir un entorno de actuación menos ideal.

# Bibliografía

- [1] Varios Autores. Archivo situacionista hispano. url`http://sindominio.net/ash`, 1999. Accedido 01-10-2013.
- [2] Austin Richard Wyer. The synthesis of memristive neuromorphic circuits. mathesis, The University of Tennessee, Knoxville, December 2017.
- [3] Leon O. Chua. Memristor-the missing circuit element. *Transactions on Circuit Theory*, CT-18(5):507–519, September 1971.
- [4] Yuriy V. Pershin y Leon O. Chua Massimiliano Di Ventra. Circuit elements with memory: memristors, memcapacitors and meminductors. *IEEE*, January 2009.
- [5] Sung Mo Kang Lion O. Chua. Memristive devices and systems. *IEEE*, 64(2):209–223, February 1976.
- [6] Steven La Fontaine y Massimiliano Di Ventra Yuriy V. Pershin. Memristive model of amoeba’s learning. *arXiv:0810.4179*, October 2008.
- [7] Duncan R. Stewart y R. Stanley Williams Dmitri B. Strukov, Gregory S. Snider. The missing memristor found. *Nature*, 453:80–83, May 2008.
- [8] Alexander A Bessonov, Marina N Kirikova, Dmitrii I Petukhov, Mark Allen, Tapani Ryhänen, and Marc JA Bailey. Layered memristive and memcapacitive switches for printable electronics. *Nature materials*, 14(2):199, 2015.
- [9] Kuk-Hwan Kim, Siddharth Gaba, Dana Wheeler, Jose M Cruz-Albrecht, Tahir Hussain, Narayan Srinivasa, and Wei Lu. A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications. *Nano letters*, 12(1):389–395, 2011.
- [10] Eric A. Vittoz. Origins of weak inversion (or sub-threshold) circuit design. *Springer*, pages 7–9, 2006.
- [11] John Hertz, Anders Krogh, and Richard G Palmer. *Introduction to the theory of neural computation*. Addison-Wesley/Addison Wesley Longman, 1991.
- [12] TH Borgstrom, M Ismail, and SB Bibyk. Programmable current-mode neural network for implementation in analogue mos vlsi. *IEE Proceedings G (Circuits, Devices and Systems)*, 137(2):175–184, 1990.

- [13] Christof Koch and Idan Segev. *Methods in neuronal modeling: from ions to networks*. MIT press, 1998.
- [14] Carver Mead and Mohammed Ismail. *Analog VLSI implementation of neural systems*, volume 80. Springer Science & Business Media, 2012.
- [15] Mona E Zaghloul, Jack L Meador, and Robert W Newcomb. *Silicon implementation of pulse coded neural networks*, volume 266. Springer Science & Business Media, 2012.
- [16] Kwabena A Boahen. Communicating neuronal ensembles between neuromorphic chips. In *Neuromorphic systems engineering*, pages 229–259. Springer, 1998.
- [17] Paul Merolla and Kwabena A Boahen. A recurrent model of orientation maps with simple and complex cells. In *Advances in neural information processing systems*, pages 995–1002, 2004.
- [18] Barrie Gilbert. Translinear circuits: An historical overview. *Analog Integrated Circuits and Signal Processing*, 9(2):95–118, 1996.
- [19] Christoph Rasche and Richard HR Hahnloser. Silicon synaptic depression. *Biological Cybernetics*, 84(1):57–62, 2001.
- [20] Chiara Bartolozzi, Srinjoy Mitra, and Giacomo Indiveri. An ultra low power current-mode filter for neuromorphic systems and biomedical signal processing. In *Biomedical Circuits and Systems Conference, 2006. BioCAS 2006. IEEE*, pages 130–133. IEEE, 2006.

