Universitat Autònoma
de Barcelona

ESTUDIO DE UN KIT DE DISEÑO MIXTO CON SOFT-CORE MICROBLAZE E IP's, PARA EL CONTROL ESP, ASR Y ABS

Memoria del proyecto

de Ingeniería Técnica en

Informática de Sistemas

realizado per

Danilo Sanz Moreno

i dirigido per

Raúl Aragonés Ortiz

Escola d'Enginyeria

Sabadell, *Junio* de *2010*

Índice

1. Introducción

1.1. Tipología y palabras clave

Tipología : Codiseño hardware/software

Palabras clave: PSoC(1), FPGA(2), Sensor, Codiseño hardware/software, C#

1.2. Descripción

Éste proyecto consiste en el estudio de una placa de prototipado mixta analógico-digital formada principalmente por un PSoC(1), una FPGA(2) y memoria flash para determinar sus capacidades en sistemas de control ESP(3), ASR(4) y ABS(5).

El estudio se basa en concluir la lógica que se puede añadir al dispositivo para enfocarlo a unas aplicaciones que, a pesar de ser muy comunes en coches, está poco desarrollado en motocicletas y ciclomotores. Es por ello surge el interés de diseñar un sistema del más bajo coste posible para impulsar su desarrollo.

El valor final del estudio se encuentra en la comparación y el comentario de las opciones de configuración más adecuadas para, posteriormente, llevar a cabo la implementación de un sistema hardware que mida las principales variables necesarias para poder llevar su finalidad mediante la programación software.

(1) FPGA (Field Programmable Gate Array) es un chip reprogramable que permite la implementación de circuitos hardware digitales por medio de conexiones entre bloques lógicos mediante su programación via software.

(2) PSoC (Programmable System On Chip) es un microcontrolador reprogramable de la marca *Cypress* que como particularidad permite la implementación de determinadas funciones analógicas y digitales gracias a la incorporación de módulos prediseñados.

(3) ESP (*Electronic Stability Program*) es un medio de seguridad que actúa sobre una de las ruedas del vehículo frenándola en casos de giro para evitar casos de pérdida de adherencia.

(4) ASR (*Automatic Stability Control*) es un sistema de seguridad que corrige la situación de patinaje de una rueda desacelerándola o frenándola.

(5) ABS (Antilock Brake System) es un dispositivo que actúa sobre los frenos haciendo que estos no bloqueen la rueda, ya que en ese caso pierden adherencia.

1.3. Estructura de la memoria

Esta memoria se divide en ocho capítulos divididos cada uno de ellos en sub-apartados correspondientes a la explicación llevada a cabo.

En el primer capítulo se hace una pequeña introducción donde se introduce al lector a los objetivos del proyecto y el contexto que abarca.

El segundo capítulo se dedica a hacer un estudio de viabilidad. En este punto se estudia el caso al que estará destinado el hardware final y la función de los componentes. Se explican los componentes que se utilizarán y el estado del arte para determinar el sistema base.

El tercer capítulo corresponde al diseño, donde se explica el sistema base, su funcionamiento y los elementos que contiene. También aquí se muestran los resultados de las pruebas de rendimiento realizadas, y se compara con otras.

El quinto capítulo corresponde a la implementación del sistema, se explican todos los elementos que interactúan y sus aspectos de programación más importante.

El sexto capítulo muestra los resultados de las pruebas de todas las implementaciones llevadas a cabo.

En el séptimo punto corresponde a las conclusiones acerca del sistema y a las mejoras que se le pueden hacer.

El octavo capítulo muestra la bibliografía consultada por el alumno.

1.4. Definiciones, acrónimos y abreviaciones

AVNET xc3s400A Eval Board : marca y modelo de la placa de prototipado estudiada.

Big-endian : Determina el orden de los bits en memoria, que se escriben en el orden natural.

BlockRAM : memoria formada por bloques lógicos reconfigurables embebidos en la FPGA.

Caché : Memoria de rápido acceso situada muy cerca del procesador.

CLB: Son las siglas de *Configurable Logic Block* y es cada uno de los bloques que contiene un FPGA en su interior para formar funciones.

Cypress : Fabricante de microcontroladores PSoC, en concreto, el modelo *CY8C24894* es el que se encuentra en la placa.

g : Unidad estándar de medida utilizada por acelerómetros, equivale a la unidad de la gravitación terrestre de 9.81 m/s^2 aproximadamente.

Harvard : Arquitectura en que se divide las instrucciones de los datos para ubicarlos en memorias diferentes. Es el método alternativo a la arquitectura Von Neumann.

Interrupciones : Sistema de procesamiento por el cual se interrumpe el flujo de ejecución del programa para atender la petición de un periférico de entrada/salida.

IPCore (*Intellectual Property Core*) : Software prediseñado para su implementación en FPGA's u otros dispositivos programables, su punto fuerte es la portabilidad entre dispositivos y en muchas ocasiones la optimización de recursos de éstos.

LUT : Son las siglas de Logic-Up-Table, es un vector de memoria con valores predefinidos.

Pipeline : Sistema por el que al dividir una instrucción de procesamiento en varias etapas, varias instrucciones pueden estar ejecutándose al mismo tiempo cada una en su correspondiente etapa, de esta manera el número de etapas define el número de instrucciones que pueden estar ejecutándose simultáneamente.

PSoC (*Programmable System On Chip*) : Microcontrolador reprogramable de la marca *Cypress* que como particularidad permite la implementación de determinadas funciones analógicas y digitales.

RISC : Son las siglas de *Reduced Instruction Set Computer* y es una característica de la arquitectura de microprocesador. En ésta las instrucciones son pequeñas y del mismo tamaño con lo que suele tomar menos tiempo de ejecución.

SPARC : Son las siglas de Schalable Processor Architecture, es una arquitectura RISC de carácter libre.

Spartan-3A XC3S400A : Familia y modelo del FPGA contenido en la placa objeto del estudio.

System-On-Chip : Tipo de microcontroladores que se diferencian a los habituales por permitir la reconfiguración sin estar sometidos a la dependencia de hardware que lleven alrededor.

Xilinx : Famoso fabricante de FPGA's, es el que desarrolla Spartan-3A.

1.5. Objetivos

- O.1. Tener un estudio donde se analicen las opciones de implementación más provechosas para la FPGA y se elija la más adecuada.
- O.2. Implementar un sistema base embebido en la FPGA con las opciones escogidas.
- O.3. Realizar pruebas de rendimiento sobre el sistema de procesado implementado y hacer una comparativa con otros.
- O.4. Obtener un programa destinado a PC con el que se pueda interactuar con el dispositivo.
- O.5. Realizar la medida, el tratamiento y el envío a PC de datos referentes a la temperatura ambiente a través de un sensor de temperatura.
- O.6. Realizar un IPCore que nos permita obtener la aceleración a través de un acelerómetro. Estos datos deben ser tratados mediante cálculos para obtener velocidad, inclinación y enviarlos a PC.
- O.7. Estudiar e implementar el tratamiento de señales analógicas por el PSoC y enviarlas en formato digital a la FPGA, una vez aquí se podrán tratar junto al resto de datos.

Tabla 1.1: Prioridad de los objetivos

	Crítico	*Prioritario*	*Secundario*
O1	X		
O2	X		
O3	X		
O4		X	
O5		X	
O6		X	
O7			X

1.6. Material entregado

a) Un documento que contiene los detalles del estudio llevado a cabo.

b) Una aplicación diseñada para PC que permite visualizar los datos recibidos del hardware programado.

c) Un cd que contiene los archivos fuentes, los proyectos y los ejecutables con las instrucciones a seguir.

2. Estudio de viabilidad

2.1. Marco del proyecto

Este apartado introduce al lector a los sistemas a los que se pretende enfocar el hardware y explica su funcionamiento, justificando en todos los casos la forma en que se podría tratar con los componentes utilizados.

2.1.1. Contexto

Desde hace años nos encontramos en los automóviles sistemas de seguridad pasiva basados en el antibloqueo de frenos, el control de tracción y el control de estabilidad. En las motocicletas estos sistemas son novedosos y no están muy estandarizados debido a su precio y a que en la mayoría de ocasiones ni siquiera se dispone como equipamiento adicional.

Para llevar a cabo estos sistemas se requiere realizar medidas de velocidad en las ruedas. Aunque estas medidas se pueden limitar únicamente a la velocidad de la rueda para conocer la desaceleración, se pueden complementar con un acelerómetro de dos ejes y un sensor de temperatura. De esta manera el acelerómetro puede medir el ángulo del manillar con un eje y la inclinación lateral de la motocicleta con el otro, mientras, el sensor de temperatura se destina a detectar el sobrecalentamiento del disco de freno.

El complemento a este proyecto sería un tratamiento de las señales que permita actuar frenando ligeramente la rueda.

Lo que se pretende es montar un hardware que quizás se pueda implementar para llevar a cabo aplicaciones de este calibre. Para tal fin se dispone de una placa de prototipado que puede albergar sistemas basados en microprocesadores sintetizables, combinados con los IPCores necesarios, para el tratado de las señales externas. Además, ésta dispone de un SystemOnChip PSoC incorporado que permite el tratado de señales de Entrada/Salida tanto analógicas como digitales. De este modo surge un interés por conocer las capacidades que pueda tener el kit en este ámbito. Además de la placa, el acelerómetro y el sensor de temperatura se dispone de un sensor óptico que se encargaría de medir la velocidad de la rueda.

2.1.2. Frenado antibloqueo

El ABS o sistema de frenos antibloqueo se basa en la detección del bloqueo, lo cual produce que el neumático se deslice en el asfalto. Para tratar el problema se dispone el sensor óptico y el sensor de temperatura.

- *Bloqueo de ruedas*

El método antibloqueo que se puede tratar a partir de este proyecto es la detección de un sobrecalentamiento de los discos de frenos y una desaceleración muy brusca de las ruedas, con lo cual la rueda está a punto de quedar bloqueada.

Para obtener la desaceleración de la ruedas se utiliza el foto-sensor, con el que se puede calcular la velocidad angular de la rueda y la aceleración, teniendo en cuenta que la aceleración es el diferencial de velocidad respecto al tiempo. El sensor de temperatura situado cerca del disco de frenos nos debe aproximar la temperatura del disco de frenos. En caso de calentarse en exceso, también se soltaría el freno momentáneamente para que el disco pueda recuperar su temperatura.

El hecho de contemplar la temperatura como una variable supone un avance en estas aplicaciones, ya que hasta el momento ningún sistema ABS ha incorporado la medición térmica.

- *Análisis*

Tabla 2.1: Actuadores ABS

Elementos actuadores	Acelerómetro	Sensor óptico	Sensor de temperatura
Bloqueo de frenos.	No.	Sí, para obtener la velocidad de giro y sacar la desaceleración.	Sí, para conocer la temperatura del disco.

2.1.3. El control de tracción

El ASR o control de tracción es un sistema que protege el vehículo frente el patinaje de una rueda. Este caso es especial en aceleraciones o frenadas y, a diferencia del ESP descrito más abajo, el ASR está enfocado únicamente a los casos de pérdida de adherencia de las ruedas. Estos casos pueden darse debidos a una aceleración o bien en una frenada. Para ello utiliza los mismos mecanismos que el ABS y su actuación es la de desacelerar el vehículo o la de frenar una de las ruedas.

- *Actuación en aceleración*

El primer caso a tratar es la pérdida de tracción de una rueda debido a una aceleración. Hay que destacar que todas las motos a nivel comercial son de tracción trasera y que una aceleración, en determinadas condiciones del asfalto y atmosféricas, puede provocar que la rueda trasera pierda adherencia y patine.

El tratamiento de este problema pasaría por la medida de la diferencia de velocidad angular o tangencial de cada rueda, resuelta mediante el tiempo entre cada lectura del fotosensor, conociendo los dientes o agujeros del disco exactos que corresponden a una vuelta de la rueda.

Figura 1.1: Descripción velocidades angulares

En esta situación se produciría que wA sería mayor que wB, con lo que se frenaría el disco trasero gradualmente a la diferencia entre wA-wB.

- *Actuación en el frenado*

El segundo caso es la pérdida de tracción durante una frenada. En esta situación la motocicleta derrapa ligeramente en una de las dos ruedas debido a la diferencia de desaceleración entre ellas. Este sistema detectaría el patinaje de la misma forma que en el caso de la aceleración, por medio del diferencial de velocidad angular entre ambas ruedas, dado el caso se deberá reducir la frenada en la rueda que gira más rápido.

- *Análisis*

Tabla 2.2: Actuadores ASR

Elementos actuadores	Acelerómetro	Sensor óptico	Sensor de temperatura
Actuación en aceleración	No.	Sí, para velocidad de giro en rueda delantera y trasera.	Sí, para conocer la temperatura del disco.
Actuación en el frenado	No	Sí, para velocidad de giro en rueda delantera y trasera.	Sí, para conocer la temperatura del disco.

2.1.4. Control de estabilidad

El ESP (Electronic Stability Program), también llamado ESC (Electronic Stability Control), es un dispositivo de seguridad incorporado en vehículos de tracción que actúa durante los virajes, evitando que una de las ruedas derrape y se desestabilice.

La implementación del control del ESP en motocicletas pasa principalmente por controlar tres aspectos básicos que son los siguientes:

- La aceleración tangencial al giro no corresponde con la aceleración de la rueda delantera girada determinando el ángulo de giro del manillar.
- El ángulo de inclinación de la motocicleta respecto a la posición vertical no corresponde con el ángulo de giro del manillar ni la velocidad.
- La velocidad de giro de las ruedas no corresponde.

- *Diferencia de aceleración tangencial/tangencial*

En este caso el factor determinante es conocer el ángulo de giro del manillar. La actuación en esta situación sería frenar la rueda delantera hasta el punto en que corresponda con la aceleración tangencial.

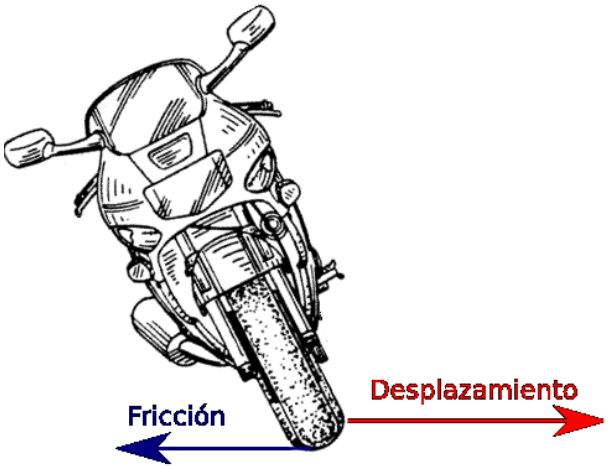
Para hacer la medición del ángulo del manillar se podría utilizar uno de los ejes del acelerómetro, para conocer la velocidad tangencial se utilizaría el sensor óptico que proporcionaría la aceleración de la rueda trasera, y para conocer la aceleración angular el sensor óptico en la rueda delantera.

- *Diferencia entre inclinación y giro del manillar*

Es esta situación es necesario contemplar la velocidad de la motocicleta ya que a mayor velocidad es menos decisiva esta diferencia. A velocidades altas es necesario tomar la curva inclinando la motocicleta mucho más que el manillar y al revés. Este caso es debido a que un giro de manillar a mucha velocidad desviará la base de la motocicleta al interior de la curva mientras las zonas más alejadas tienden a ir hacia delante por su propia inercia. Por lo tanto, a velocidades altas, la inclinación de la motocicleta al lado que gira contrarresta esa inercia. Además ayuda a girar, dado que la fuerza de fricción hacia el centro de la curva contrarresta la tendencia que tienen las ruedas a desplazarse al exterior.

Para resolver esta situación se deberá concluir la función que nos determina un giro correcto en función de las tres variables comentadas. En el caso que la función determine un giro excesivo se frenarían ambas ruedas para intentar recuperar el equilibrio.

Figura 2.2: Fuerzas presentes en la inclinación



- *Diferencia de giro entre ruedas*

Para controlar la estabilidad de la moto en una curva también se hace uso del sistema de control de tracción por el que ambas ruedas tienen que llevar una velocidad angular equivalente. Para este caso se frenará la rueda que gire más rápido para evitar el patinaje.

- *Análisis*

Tabla 2.3: Actuadores ESP

Elementos actuadores	Acelerómetro	Sensor óptico	Sensor de temperatura
Diferencia aceleración angular y tangencial	Sí, para conocer giro del manillar.	Sí, para velocidad de giro en rueda delantera y velocidad tangencial con velocidad de giro rueda trasera.	Sí, para conocer la temperatura del disco.
Diferencia inclinación y giro del manillar	Sí, para obtener inclinación lateral de motocicleta y giro del manillar.	Sí, para conocer la velocidad de la motocicleta.	Sí, para conocer la temperatura del disco.
Diferencia de giro entre ruedas	No	Sí, determinar la velocidad angular de cada rueda.	Sí, para conocer temperatura del disco.

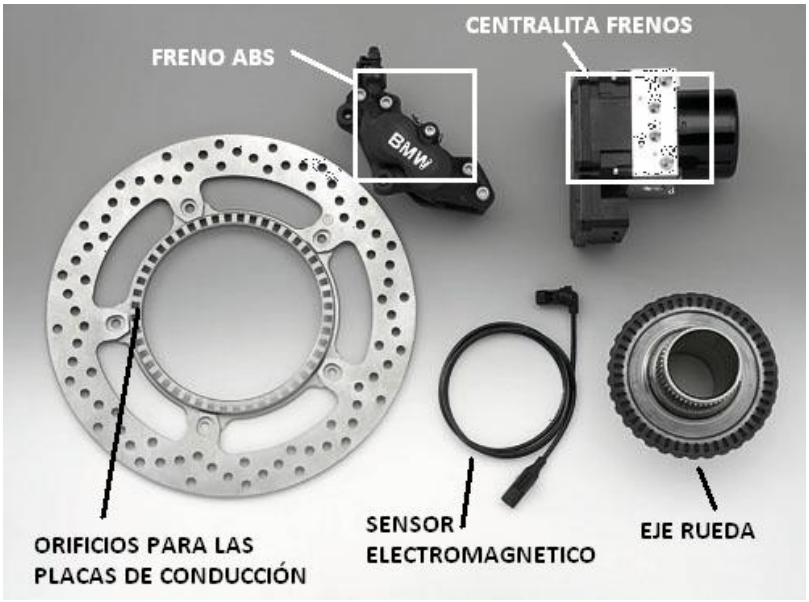
2.1.5. Disponibilidad en el mercado

El sistema ABS hace más de diez años que se empezó a integrar en motocicletas. Aunque a simple vista cuesta reconocer el sistema ABS, este destaca por ser un poco más grande que los sistemas de frenos habituales (ver figura 2.3).

Figura 2.3: Frenos ABS

Sobre los sistemas ESC- ABS disponibles actualmente cabe decir que funcionan mediante un sensor electromagnético y unas placas que al cruzar entre los sensores producen que exista o no corriente. Los resultados obtenidos son mucho mejores de los que se podría obtener con un sensor óptico en el que el polvo o la suciedad son suficientes para eliminar el paso de fotones entre el emisor y el receptor.

Figura 2.4: ESP-ABS de BMW



2.2. Partes interesadas

2.2.1. Intereses (Stakeholders)

Tabla 2.4: Stakeholders

Nombre	Descripción	Responsabilidad
Raúl Aragonés Ortiz	Director de proyecto	Aprobación del proyecto. Participa en su definición y hace el seguimiento. Supervisa el trabajo hecho por el alumno y evalúa el proyecto.
Danilo Sanz Moreno	Alumno responsable del proyecto	Realización del proyecto. Definición de requisitos y funcionalidades.

2.2.2. Perfiles de usuario

El proyecto que se realiza no dispone de usuarios específicos dado que es un sistema autónomo. En cualquier caso existen grandes posibilidades de ampliación dado que el sistema hardware final no está suficientemente explotado y existe la posibilidad de la implementación de los algoritmos de control mediante el tratamiento de las señales captadas.

Tabla 2.5: Perfiles de usuario

Perfil	Responsabilidad
Ingeniero o diseñador hardware/software	Explotación del estudio para la implementación en dispositivos semejantes y prever la viabilidad de otro proyectos. Partiendo de la aplicación obtenida, ampliar el sistema con nuevas funcionalidades y algoritmos para su implementación en la vida real.
Empresa dedicada al sector de seguridad en automóviles	Uso del dispositivo medidor para la implementación de sistemas de frenado.

2.2.3. Project Team

Tabla 2.6: Project Team

Nombre	Descripción	Responsabilidad
Raúl Aragonés Ortiz	Director de proyecto y tutor.	Define, planifica y sigue el proyecto. Supervisa el trabajo hecho por el alumno.
Danilo Sanz Moreno	Alumno responsable del proyecto	Colabora con el director de proyecto en el estudio de viabilidad y la planificación. Desarrolla la investigación que requiere y en la que se basa el proyecto. Implementa las funcionalidades requeridas.

2.3. Fases del proyecto y planificación

2.3.1. Fases del proyecto

El proyecto se ha estructurado en 7 etapas diferentes que se detallan a continuación:

1ª etapa: Familiarización con la placa y documentación sobre los componentes.

2ª etapa: Decisión y estudio de la implementación de microprocesador y bus en la FPGA.

3ª etapa: Estudio sobre la implementación de sistemas operativos con soporte para el microprocesador escogido.

3ª etapa: Diseño del sistema hardware base.

4ª etapa: Estudio e implementación de la lectura de la temperatura.

5ª etapa: Desarrollo del software con interfaz gráfica para PC de interacción con el dispositivo.

6ª etapa: Diseño e implementación del IPCore para el tratamiento de la aceleración.

7ª etapa: Diseño e implementación del sistema de tratado de señal analógica en el PSoC y un IPCore en FPGA para recibir datos del PSoC.

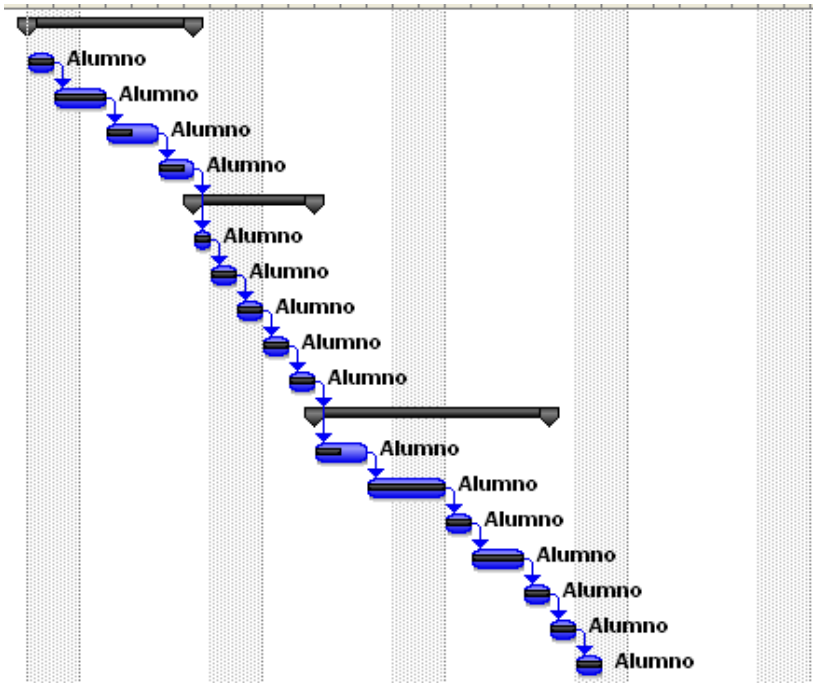
2.3.2. Planificación

Las etapas del proyecto se han dividido según la relación de la memoria, que a su vez determina los seguimientos que se han ido haciendo.

Figura 2.6: Tabla de tareas

Nombre de tarea	Duración prevista	redecesora:	Duración real
Estudio de viabilidad	**120 horas**		**117 horas**
Familiarización con la placa	20 horas		18 horas
Estudio Leon-3-AMBA	40 horas	2	45 horas
Estudio e implementación MicroBlaze-PL	40 horas	3	31 horas
Estudio e implementación kernel	20 horas	4	23 horas
Diseño	**65 horas**	**5**	**63 horas**
Estudio componentes de MicroBlaze	15 horas	5	14 horas
Estudio e Implementación ExecuteInPlace	15 horas	7	16 horas
Estudio y adaptación pruebas de rendim	20 horas	8	18 horas
Implementación pruebas de rendimiento	10 horas	9	8 horas
Estudio IPCore's	5 horas	10	7 horas
Implementación	**175 horas**	**11**	**192 horas**
Lectura de temperatura	30 horas	11	33 horas
Implementación del IPCore acel2axis	50 horas	13	50 horas
Implementación sensor optico	30 horas	14	30 horas
Implementación Programa MicroBlaze	35 horas	15	39 horas
Diseño e implementación aplicación PC	30 horas	16	40 horas
Pruebas	30 horas	17	30 horas
Documentación del proyecto	30 horas	18	30 horas
Total horas	420 horas	19	420 horas

Figura 2.7: Diagrama de Gantt

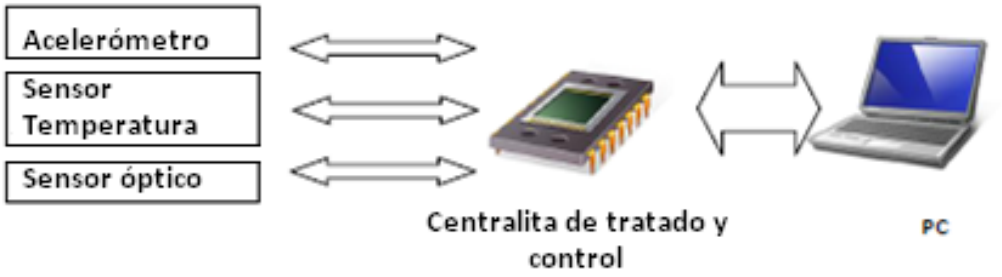


2.4. Descripción del sistema

El sistema final obtendrá los datos de los sensores y hará cálculos para poder visualizarlos en PC. La placa deberá encender unos led's que lleva integrados para la medición dinámica de una variable, también actuará en función de unos botones que lleva.

En la aplicación de PC debe existir la posibilidad de pedir o no los datos.

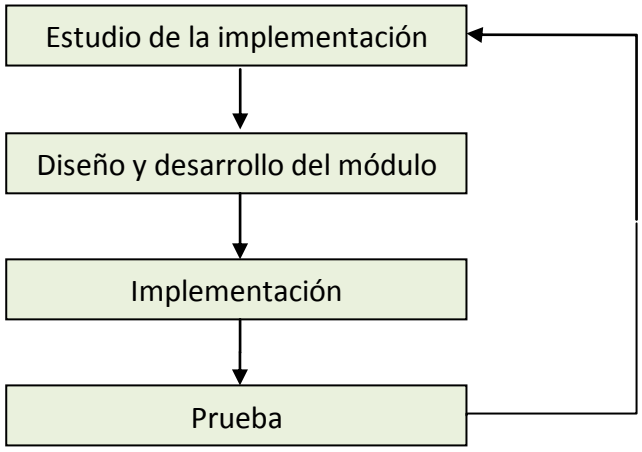
Figura 2.7. Esquema del sistema



2.5. Metodología de desarrollo

La metodología de desarrollo usada en todo el proyecto ha sido el modelo clásico ya que se trata de un proyecto con personal reducido que no permite un desarrollo más elaborado por los posibles riesgos de fracaso que puedan haber.

Figura 2.8: Metodología de desarrollo



2.6. Entorno de prototipado

2.6.1. Descripción

Para la realización de este proyecto se trabaja sobre el kit *AVNET Spartan-3A Evaluation Board* que trabaja a un voltaje de 3.3 Volts.

La placa se compone principalmente por un FPGA, un PSoC, puertos de entrada/salida y memorias flash.

Acerca de la memoria se ha de decir que el dispositivo lleva incorporados dos chips: uno de ellos es memoria de acceso serie síncrona de 16 MegaBytes compartida por el FPGA y PSoC, y la otra es de 16 líneas de datos en paralelo de 4 MegaBytes, accesible únicamente por la FPGA.

Además la placa cuenta con todos los puertos necesarios dedicados a reprogramación y reseteado.

El puerto USB se reconoce en el PC como si de un puerto serie se tratara dado que viene controlado por un módulo USB-UART en el PSoC. La velocidad a la que trabaja este UART es de 115.800 bits/segundo, y se ha de mantener la misma configuración para programar/comunicar el dispositivo.

Figura 2.9. Placa de prototipado (vista frontal)

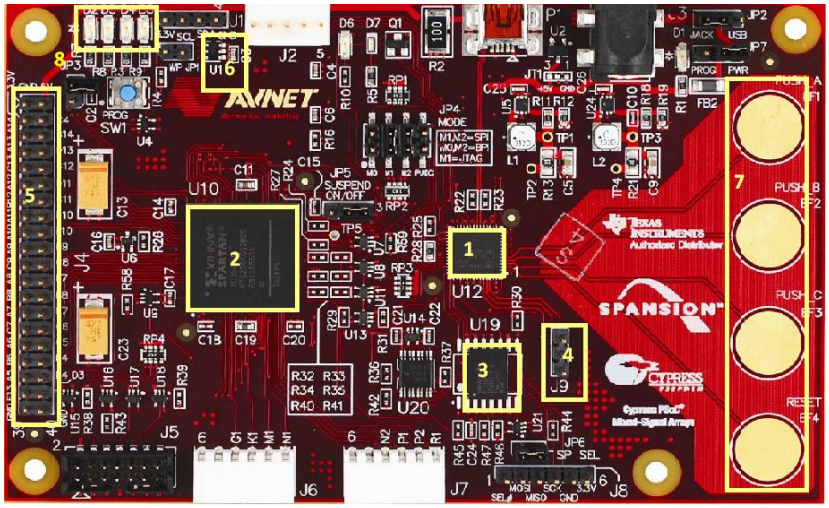
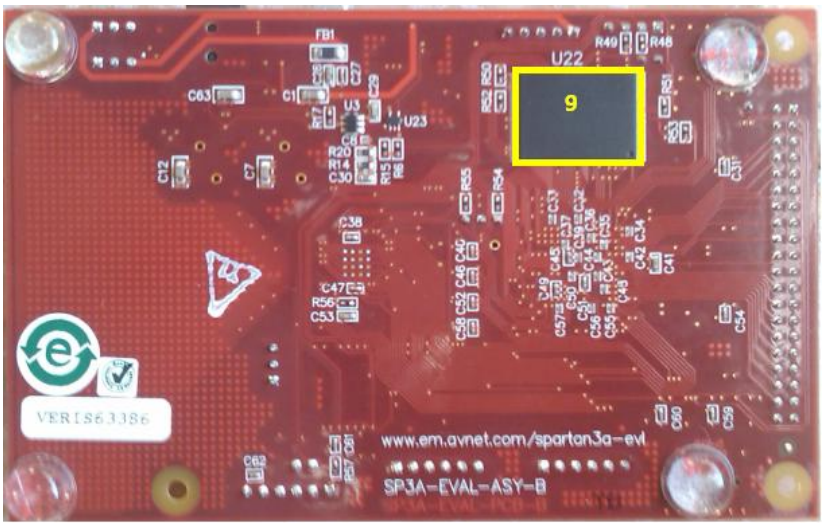


Figura 2.10: Placa de prototipado (vista trasera)



Partiendo de la figura 2.6 podemos distinguir los principales componentes de la placa de prototipado:

1. Cypress PSoC CY8C24894-24LFXI
2. FPGA Xilinx Spartan-3A 400 ft256
3. La memoria Flash serie Spansion *S25FL128P.*
4. Pines de entrada/salida mapeados en el PSoC en los puertos P0_1, P0_3 y P0_5 de arriba hacia abajo
5. Pines de entrada/salida mapeados en la FPGA en los que se indican sus puertos en el misma placa y en los esquemáticos.
6. Sensor de temperatura Texas Instruments TMP100 conectado a PSoC y FPGA.
7. Botones sensitivos al tacto *CapSense* de Cypress.
8. 4 LED's de salida conectados directamente al FPGA.
9. Memoria flash paralela Spansion *S29GL032N.*

En los siguientes apartados se describen los componentes más importantes.

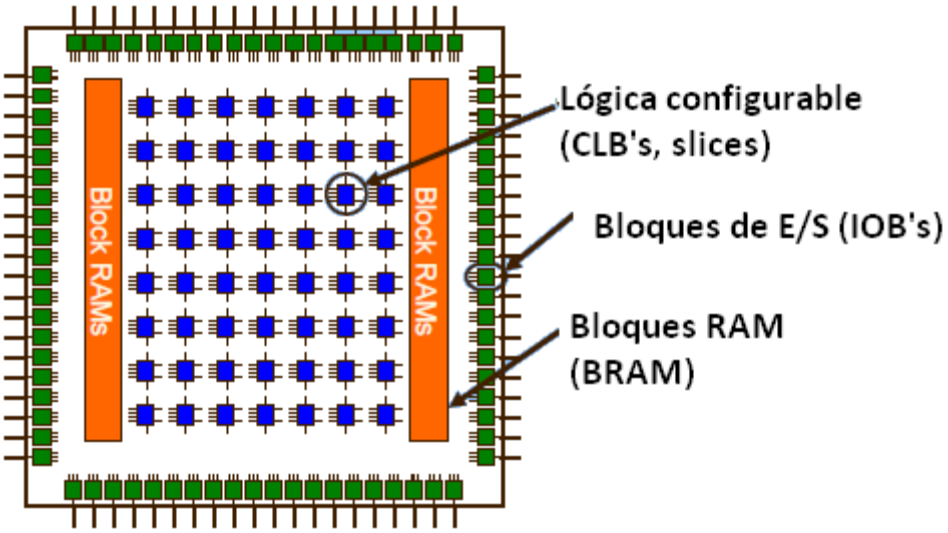
2.6.2. El FPGA

El FPGA es un dispositivo de los llamados SoC (*SystemOnChip*) . SystemOnChip quiere decir que a diferencia de las placas de evaluación de microcontroladores habituales, en los que toda la programación ha de estar adaptada al diseño inicial de la placa de evaluación, este se puede programar incorporando elementos hardware para obtener diseños más completos.

- *Introducción*

La estructura interna de una FPGA es una matriz de conexiones separadas por bloques, encontrándose los más cercanos al exterior los de entrada/salida y memoria RAM y en el núcleo los slices o CLB's (Cell logic blocks), compuestos de Flip-Flops y LUT's con un determinado número de entradas (4 en nuestro modelo).

Figura 2.11: Esquema FPGA



Una vez que realizamos el proceso de co-diseño hardware/software se deben generar las netlist del diseño, que forma el bloque con los elementos, y a continuación el sintetizado que adapta el diseño al modelo de FPGA.

- *La familia Spartan-3A*

La familia de FPGA Spartan-3A se corresponde con la serie de FPGA's de bajo coste de la marca Xilinx. El modelo que contiene la placa de prototipado contiene 400 mil puertas lógicas distribuidas en 7168 slices o CLB's. También dispone de 360 mil BlockRAMbits, es decir, hasta 45 KBytes de memoria que se puede utilizar como BlockRAM.

Esta familia de FPGA está optimizada para lógica de entrada/salida disponiendo de más bloques para interactuar con el exterior que otros modelos de la serie.

Las señales de reloj disponibles para utilizar con el chip FPGA son tres, uno de ellos es de un generador de onda cuadrada MAXIM que funciona a 16 MHz y se encuentra mapeado en el pin C10, y los otro dos provienen del PSoC, uno es de 12 MHz que viene desde el puerto P2[6] de PSoC al pin N9 de la FPGA y el otro es de 32 MHz que llega desde el P3[7] al T7.

2.6.3. SystemOnChip PSoC

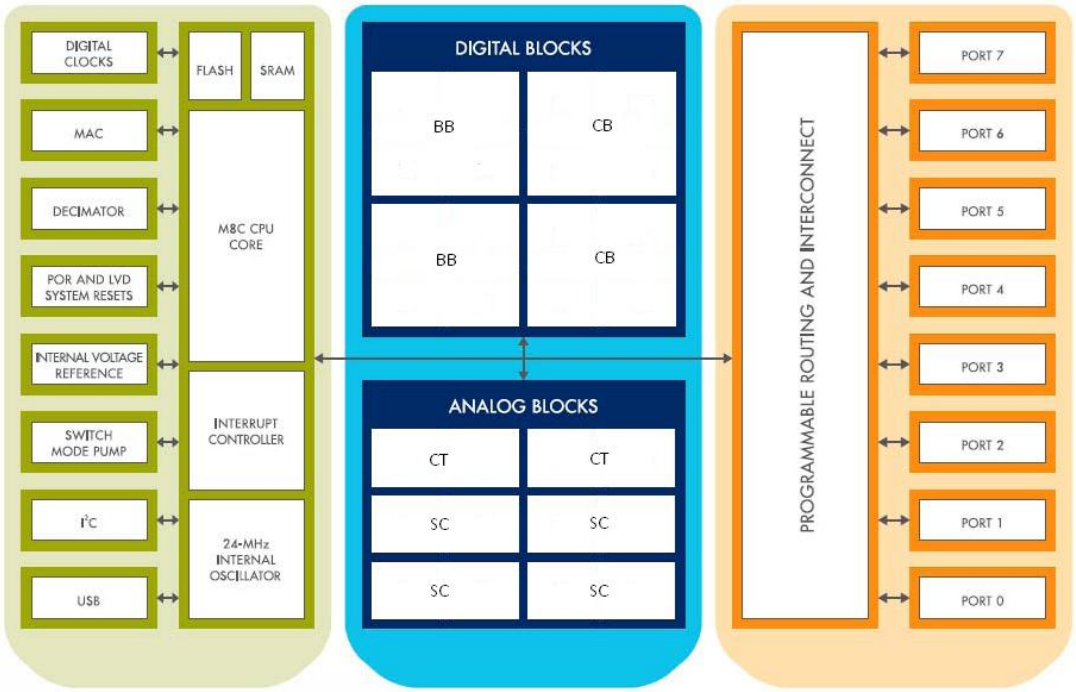
El PSoC es un microcontrolador *SystemOnChip* reprogramable de la marca *Cypress*. A diferencia de la FPGA, éste se reprograma mediante módulos analógicos y digitales que se interconectan a unos bloques para tal propósito. Esto hace que sea fácil adaptar a múltiples diseños y propósitos.

La cantidad de bloques lógicos disponibles para diseñar el sistema periférico dependen del modelo de PsoC que vayamos a programar, en nuestro caso existen 4 bloques digitales y 4 analógicos.

Hay que comentar que existen dos tipos de bloques digitales, tenemos los BB que son *Building Blocks* y los CB *Comunication Blocks*. La diferencia entre ellos es que los Comunication Blocks son los únicos que pueden disponer bloques de comunicaciones para I2C, SPI, UART.

De bloques analógicos también distinguimos dos tipos, los CT (*Continuous Time*) se componen de circuitos amplificadores y los SC (*Switch Capacitor*) que están formados por circuitos con capacidades conmutadas.

Figura 2.12: Esquema PSoC

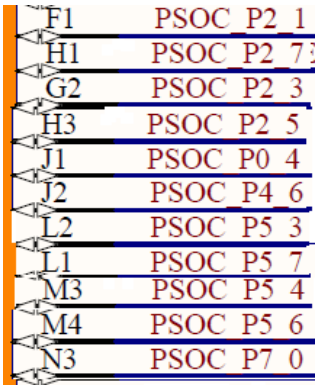


Además de lo dicho, el PsoC es reprogramable dinámicamente, es decir que podemos montar más de un diseño en bloques y que respondiendo a determinados eventos pasemos de tener un diseño cargado a otro sin que el consumo de bloques lógicos se acumule.

En el PSoC de la placa incluye un programa inicial compuesto por dos diseños diferentes. Uno de ellos responde a la configuración *USBUart* el cual hace que la conexión USB sea tratada como una conexión UART y es la configuración que se usa en la comunicación PC-placa. Las otra configuración es la *SPIconfig* que se carga al configurar la memoria SPI.

Existen once señales que comunican el PSoC con la FPGA (ver figura 2.13).

Figura 2.13: Señales PSoC-FPGA



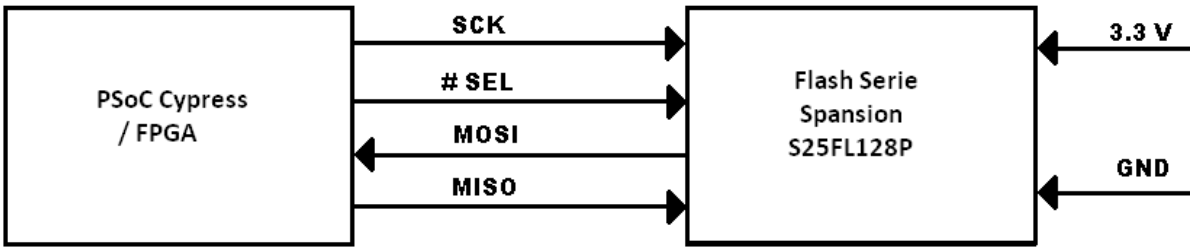
2.6.4. Memoria flash serie

La memoria flash serie incluida es de 16 MB (128 Mbits), esta memoria utiliza el protocolo SPI y puede ser accesible o bien por el PsoC o por la FPGA. Funciona a 4 MHz de frecuencia de reloj que viene determinada por la configuración del módulo SPIconfig configurado en el PSoC, por lo que entrega 1 bit cada 250 nanosegundos. En este tipo de memoria la limitación viene dada por la capacidad de transferencia.

El protocolo SPI se trata de un protocolo serie síncrono entre periféricos. Es full-dúplex lo que permite comunicación bidireccional.

El funcionamiento de este protocolo depende de una señal maestra *SCK* que lleva la señal de reloj (proporcionada por el PsoC), una de activación de transferencia *#SEL*, y dos de datos: una *MOSI* (master-output slave-input) que entrega el esclavo al master y otra *MISO*(master-input slave-output) para salida de datos.

Figura 2.14: Memoria flash serie



2.6.5. Memoria flash paralelo

La otra memoria flash es de comunicación paralela con 16 líneas de datos. Es una memoria asíncrona con una latencia de 90 ns en el tiempo de acceso y 15 ns en el de escritura. Dispone de 22 líneas de direcciones para direccionar los 4 MegaBytes y tiene una mayor velocidad y menor latencia que la memoria serie, es por ello que se le busca la utilidad como memoria ROM de programa.

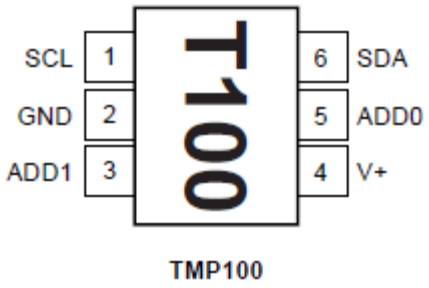
Tabla 2.7: Señales Memoria flash paralela

Señal	Nº de líneas	Utilidad
Data	16	Envío de datos a leer/escribir
Address	22	Dirección de memoria a leer/escribir
CE (Chip Enable)	1	Habilitación del dispositivo para operar
WE (Write Enable)	1	Junto a CE, operación de escritura
RE (Read Enable)	1	Junto a CE, operación de lectura
Reset	1	Borrar memoria
OE (Output Enable)	1	Habilitar salida
RY/BY (Ready/Busy)	1	Indica memoria preparada a 0 o ocupada si es 1
Byte	1	Transferencia es en modo 1 byte a 0 o 2 bytes a 1

2.6.6. Sensor de temperatura

La placa incluye un sensor de temperatura Texas Instruments TMP100 que conecta con la FPGA y el PSoC por medio del protocolo I2C. Este sensor mide temperaturas en un rango de -55 a 125 ºC con un máximo de error absoluto de ±3º.

Figura 2.15: Sensor de temperatura

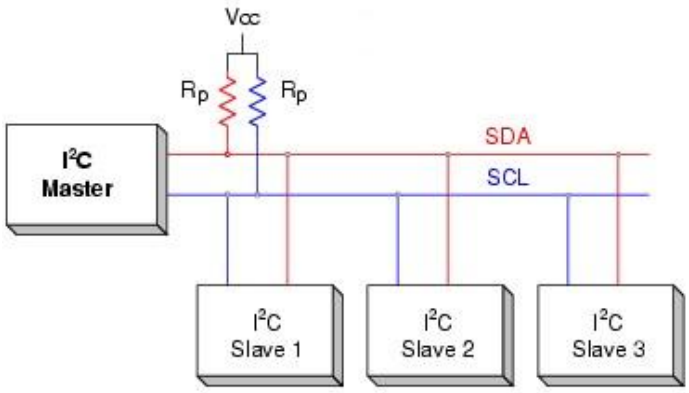


TMP100

El I2C es un bus que consta de dos señales básicas, la *SCL* (Serial Clock) para el señal de reloj y la *SDA* (Serial Data) para transportar los datos. Dos resistencias *pull-up* dejan el bus a '1' cuando no existe ninguna transferencia.

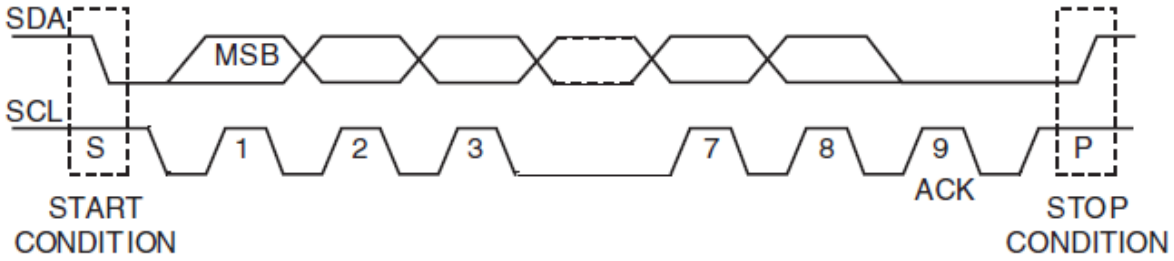
El protocolo de funcionamiento de este bus es muy parecido al protocolo serie. Durante cada transferencia existe un dispositivo maestro que genera la señal de reloj, en nuestro caso es el microprocesador y otro esclavo, que en nuestro caso es el sensor de temperatura, con una dirección de de siete bits que proporciona el fabricante (*0x48* para *TMP100* con un solo dispositivo conectado).

Figura 2.16. Esquema protocolo I^2C



Para iniciar la transferencia, el maestro crea una situación de *START* antes de la comunicación y una situación de *STOP* al acabar de enviar datos.

Figura 2.17. Señales protocolo I^2C



La señal de START se define con un flanco de bajada en SDA mientras SCL está a '1' y la de stop es el paso contrario, un cambio a '0' con el señal SCL a '1'. Las transmisiones son de 8 bits consecutivos con uno final de confirmación.

2.7. Periféricos

2.7.1. Acelerómetro

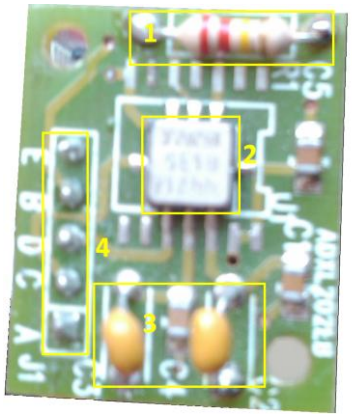
Para la medida de inclinación se dispone de un acelerómetro analógico/digital Analog Devices ADXL202JE de dos ejes y con un rango de ±2 *g*'s. Éste se encuentra encastado a una placa de evaluación ADXL202EB. La aceleración se traduce en función del ancho de pulso activo de una señal cuadrada, de la cual, la frecuencia viene determinada por una resistencia conectada a dicha placa.

La señal se muestra activa un 50% aproximadamente del periodo si la aceleración es de 0 *g*'s, aumentando o disminuyendo en función de un 12.5% por *g* aproximadamente, dependiendo del voltaje de entrada. De esta manera tenemos que el ancho de pulso será entre un 25% (-2*g*) y un 75% (+2g) aproximadamente del periodo.

Tabla 2.8: Relación Vin-DutyCycle Acelerómetro

Vin	% Duty Cycle mínimo por g	% Duty Cycle típico por g	% Duty cycle máximo per g
5 V	10.5	12.5	14.5
3 V	9	11	13

Figura 2.18. Vista acelerómetro



La resistencia (1 en la figura) es de 220 kΩ, esto quiere decir que la señal cuadrada tendrá un periodo de 2ms aproximadamente siguiendo la tabla 2.9, esto se traduce en una frecuencia de aproximadamente 500 Hz.

Tabla 2.9: Relación R_{SET}– Periódo Acelerómetro

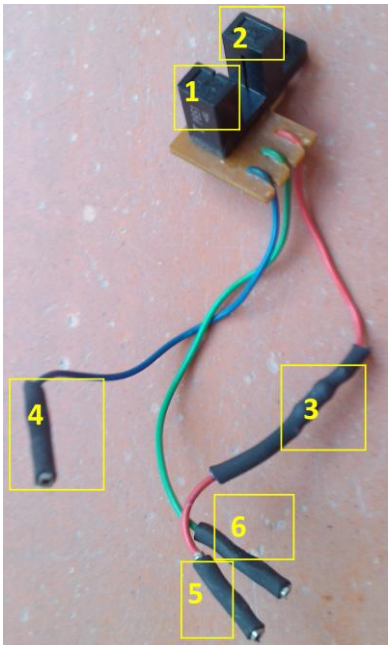
Periodo T2	Resistencia R_{SET}
1 ms	124 kΩ
2 ms	248 kΩ
5 ms	620 kΩ
10 ms	1.24 MΩ

El acelerómetro dispone de 3 entradas, una de autotest (E en la figura 2.13) conectada a 0 Volts, otra de V+ (A en la figura 2.13) para conectar un voltaje entre 3V y 5V y la de tierra(E en la figura 2.13) a 0 V. También incorpora dos salidas X (C) e Y (D) conectados con los pines de la FPGA A13 y C13 para obtener los datos. El cálculo de la aceleración viene determinado por la siguiente ecuación:

Aceleración (en g's) = (Duty Cycle – Duty Cycle at 0g)/Duty Cycle por g

2.7.2. Sensor óptico

Figura 2.19: Vista Sensor Óptico

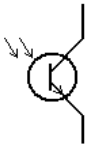


El sensor óptico está formado un diodo fotoemisor y un fotoreceptor.

- El fotoemisor (1 en la figura 2.14) no es más que es un led que emite luz infraroja que viene determinado en el periférico por el símbolo indicado en la figura 2.15. Es operativo a 1.6 Volts por lo que se ha añadido una resistencia de 100 kΩ (3 en la figura 2.14) con tal que produzca una diferencia de potencial de 1.7 Volts.

Figura 2.20: Símbolo fotoemisor

Figura 2.21: Símbolo fotoreceptor



- El fotoreceptor (2 en figura 2.14) consiste en un fototransistor con el colector conectado a V+ (3 en la figura 2.14) y en el que la luz infrarroja actúa como corriente de base. En función de la luz infrarroja aplicada a la base pasará más o menos corriente ésta se entrega por el emisor y da la salida del circuito (4 en figura 2.14).

2.8. Estado del arte

En este apartado se describe el sistema que se ha incorporado al proyecto y el estudio llevado a cabo.

2.8.1. El microprocesador

Para el desarrollo de proyectos sobre FPGA's resulta mucho más cómodo incorporar un microprocesador, de este modo obtenemos más flexibilidad de uso y podemos realizar los cálculos necesarios por medio de programación en ensamblador o C. Estos microprocesadores se dividen en dos conjuntos: los hard-core y los soft-core.

- *Microprocesadores hard-core*

En las placas de prototipado de más alta gama se incorporan los denominados microprocesadores hard-core, éstos son microprocesadores físicos de silicio desarrollados por empresas dedicadas como comúnmente conocemos AMD o Intel.

Dado el avance tecnológico es posible incorporar hard-core's en FPGA's. Algunos modelos de las series Virtex incorporan microprocesadores PowerPC de IBM a una frecuencia que ronda los 550 MHz y configuraciones multi-core. Estos microprocesadores se encuentran mapeados dentro del chip de la misma forma que lo haría en una placa base con la ventaja

de poder incorporar IPCore's a su alrededor. Obtienen mejores resultados que los soft-cores ya que la lógica de construcción en FPGA's requiere latencias mayores y limita la frecuencia.

Nuestra placa de prototipado no dispone de ningún microprocesador hard-core por lo tanto se tendrá que acudir a buscar una solución soft-core.

- *Microprocesadores Soft-core*

Los soft-core son microprocesadores IP-core sintetizables escritos en lenguajes de descripción hardware. El proceso de sintetización es el que se lleva a cabo para generar el archivo que define las conexiones internas del sistema y determina que se cumplan las limitaciones con la FPGA sobre el que se implementa. De este modo es parecido a la compilación de un programa software con la diferencia de que lo que se diseña es hardware.

Leon

Leon es un microprocesador RISC de 32-bits con arquitectura SPARC V8, es ampliamente parametrizable y está escrito en VHDL. Fue desarrollado por la Agencia Europea Espacial (ESA) y actualmente se mantiene por el grupo Gaisler Research. La versión probada ha sido la Leon-3.

El microprocesador se distribuye como parte del conjunto GRLIB IP Library que se compone además de varios IP's de licencia libre para conectar sobre buses AMBA.

Existen dos versiones disponibles, una es la versión FT (*Fault-tolerant*) con tolerancia a fallos y comercial que se utiliza principalmente para aplicaciones espaciales, la otra es la versión estándar con la que se ha tratado en el proyecto. A continuación podemos ver sus principales características:

- Conjunto de instrucciones SPARC V8 con ordenación big-endian. Dispone de tres modos de direccionamiento: immediato, directo e indirecto. Incluye instrucciones MAC, multiplicación y división.
- Disponible la incorporación de unidades MAC hardware hasta de 32bits y divisores hardware.
- Soporte multi-core con una unidad despachadora de procesos.

- Tratamiento de interrupciones.
- Varios tipos de unidades de cálculo para punto flotante FPU.
- Pipeline configurable de hasta 7 etapas.
- Caché de arquitectura Harvard de hasta 4 niveles configurable que pueden ir desde 1KB hasta 256 KB por vía, y hasta 4 vías.
- Soporte de unidad de manejo de memoria para protección de zonas críticas de sistemas operativos.
- Utiza el bus AMBA 2.0 (Advanced Microcontroller Bus Architecture) con soporte del AHB.

Todo el conjunto, con todas las opciones activadas puede llegar a obtener un rendimiento de hasta 1.4 DMIPS/MHz (ver 3.5) y puede llegar a funcionar a 125 MHz en FPGA's.

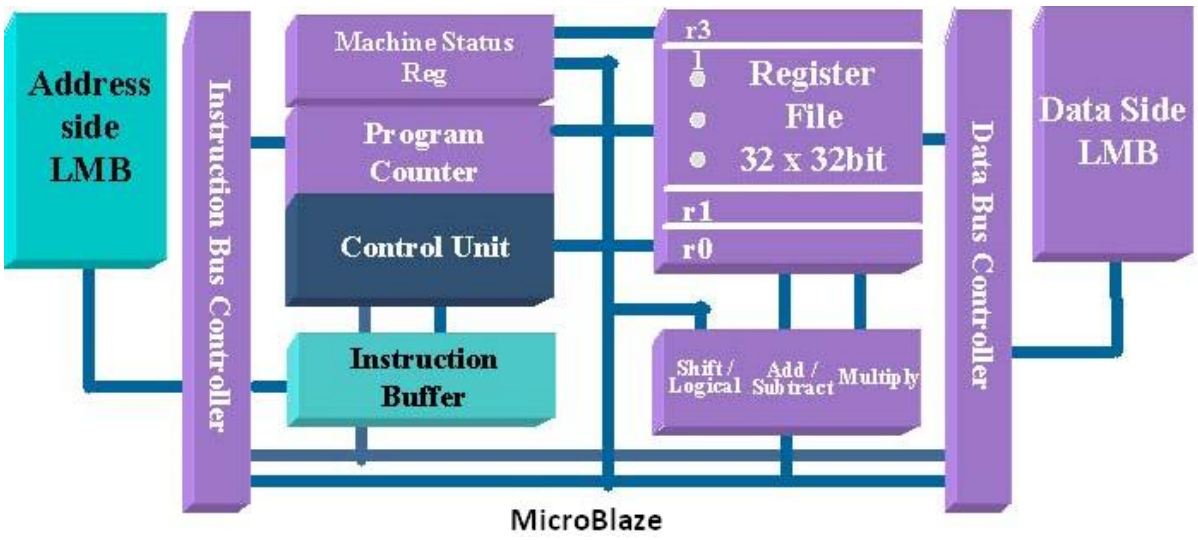
MicroBlaze

MicroBlaze es el microprocesador RISC sintetizable que Xilinx dispone para sus FPGA Spartan y Virtex. Es un microprocesador de 32-bit que usa el orden *big-endian*. La versión tratada de MicroBlaze en este proyecto es la 7.2 y a continuación mostramos datos de su arquitectura.

- Conjunto de instrucciones propio que se basa en instrucciones de 32 bits con 2 o 3 operandos.
- 32 registros de propósito general de 32-bits cada uno
- Bus de procesador de 32-bit CoreConnect PLB.
- Bus de memoria BRAM LMB configurable en velocidad
- Configuraciones de shifter hardware, divisores y multiplicadores
- Cachés Harvard configurables de hasta 256 KB para datos y 256 KB instrucciones
- Unidad de manejo de memoria.
- Pipeline de 3 o 5 etapas.
- Tratamiento de interrupciones
- Soporte multi-core

- Soporte de múltiples sistemas operativos

Figura 2.22: Estructura interna MicroBlaze



Evaluación

Después de hacer varias pruebas y búsquedas de configuraciones de Leon-3 se determinó la dificultad de incorporar este microprocesador en la placa objeto.

Para probar el microprocesador Leon-3 se necesita un sistema operativo Linux en el PC y los archivos de *constraints* de la placa. Para parametrizar el sistema se utilizó un programa de interfaz gráfica llamado *xconfig*, para simularlo se usó el simulador vhdl Modelsim en línea de comandos.

Para compilar un programa *Holamundo* en C, desarrollado para Leon, se requirió instalar el paquete correspondiente al compilador GCC para Sparc.

Después de simular se decidió la implementación sobre la placa de prototipado y surgieron problemas relacionados de capacidad al sintetizar el Leon-3. Todo el proceso se realizó primeramente en una placa soportada y posteriormente con la nuestra. Dado que el proceso de sintetizado requería tiempos de espera de hasta 30 minutos y siempre daba saltaban errores por falta de espacio se decidió probar el procesador MicroBlaze que es más común y existe más documentación.

El sistema final dispondrá MicroBlaze con una configuración limitada a 32 KB de BRAM y 50 MHz utilizando el método *Execute in Place* (véase 3.2), para ello se requiere del software EDK de Xilinx.

2.8.2. El Bus

Otro de los elementos a determinar en el sistema es el bus de procesador que queremos utilizar. En este apartado se explican los buses utilizados por ambos y se desarrollan las opciones de implementación y funcionamiento de éstos.

AMBA

Es la arquitectura de bus con la que trabaja el soft-core Leon. Se desarrolló por la marca ARM y se trata de una arquitectura orientada a microcontroladores de altas prestaciones, la versión incluida en la *GRLIB* para el diseño de Leon-3 es el AMBA 2.0 de 32-bits.

En la versión 2.0 de AMBA se disponen de tres tipos de buses. Éstos son el ASB, APB y el AHB.

El APB es un bus orientado a periféricos de baja velocidad de reloj. La forma como este bus se incorpora al sistema es mediante el APBbride, que no es más que un IPCore que conecta con uno de los buses de alto rendimiento realizando la conversión de datos necesaria para la compatibilidad con los periféricos. La incorporación de buses dedicados a este propósito baja el consumo final del sistema.

El ASB es el bus de alta velocidad en la primera versión de AMBA. Su uso va destinado a transacciones entre el microprocesador y la memoria u otras que requieran de estas prestaciones. Este bus permite el envío consecutivo de instrucciones por medio del uso de un pipeline en el controlador.

El AHB es un nuevo bus de alto rendimiento incorporado en la versión AMBA 2.0. Soporta mayores frecuencias que ASB y configuraciones de 64 y hasta 128 bits.

CoreConnect

CoreConnect es la arquitectura de buses para incorporación sobre MicroBlaze. Se compone de tres buses en que los más destacados son el PLB, un bus de alto rendimiento para elementos de memoria estandarizado entre los desarroladores de IPCore's, y el OPB, equivalente al APB de AMBA. En nuestro diseño final solo figura el bus PLB.

El PLB se trata de un bus síncrono configurado a 50 MHz que soporta lecturas y escrituras concurrentes con un direccionamiento hasta 32-bits. Tiene líneas independientes de control de lectura y escritura, y para proporcionar mayor rendimiento soporta el direccionamiento en multi-etapa (*pipeline*) con lo que el master puede pedir el bus mientras se ejecuta una

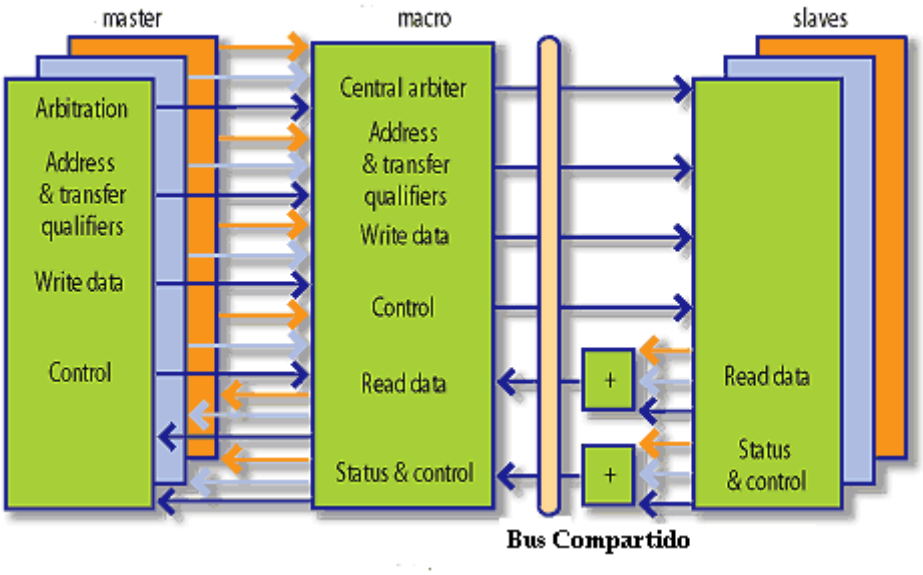
transacción. Existen 4 niveles de prioridad de acceso a este bus que se determinan con el señal de arbitraje que proporciona el master a la transacción

El controlador master incorpora las señales de control de lectura/escritura, las direcciones a leer o escribir, los datos a escribir y la prioridad de la transacción.

La zona macro del PLB incluye el arbitraje central para determinar la prioridad de la operación y los registros para las señales que recibe por los controladores maestro y esclavo.

Los controladores esclavos incluyen en la macro datos de lectura a través de las líneas de conexión (el bus) y los registros de estado y control.

Figura 2.23: Esquema PLB



Evaluación

Dado que el microprocesador escogido ha sido MicroBlaze y no disponemos de una gran cantidad de periféricos, la configuración escogida ha sido la de montar un bus PLB único para controlar todas las señales.

2.8.3. Sistema operativo

La última de las necesidades que presenta el proyecto es la de instalar un sistema operativo de tiempo real (RTOS) para poder añadir escalabilidad y comodidad al desarrollar el sistema. Las ventajas de la implementación de estos sistemas son múltiples, pero las más importantes van relacionadas con la posibilidad de creación de procesos y el control de zonas críticas.

El uso por el que se requiere un kernel es el de obtener tiempos inferiores al segundo, dado que éste proporciona una API con funciones para tal propósito.

A pesar de que existe una gran cantidad de sistemas operativos disponibles con soporte para MicroBlaze, propietarios y libres, se contempla la posibilidad de instalar unicamente UCLinux o Xilkernel, por ser dos de los más conocidos.

UCLinux

uCLinux es un sistema operativo basado en el kernel de Linux 2.6 en su última versión. Este sistema ha sido portado para microcontroladores que no disponen de unidad de gestión de memoria. El inconveniente de este sistema operativo es que necesita demasiada memoria RAM para datos, el sistema mínimo requiere más de 1 MB de datos cuando estamos limitados a 32 KB.

Xilkernel

Xilkernel es la otra opción como sistema operativo, está disponible únicamente para soft-cores MicroBlaze y hard-cores PowerPC. Se incluye en el EDK de Xilinx y está dotado de la POSIX API de C con lo que permite el multi-proceso, la planificación por prioridades, la intercomunicación de procesos, las opciones de sincronización y el manejo de interrupciones.

Este kernel ocupa apenas 9 KB de memoria y es altamente configurable mediante *Software Platform Settings* de EDK.

Elección

Aunque se han contemplado más sistemas operativos de los aquí descritos se optó directamente por implementar el Xilkernel. El motivo fue la gran cantidad de documentación que existe en torno a éste, sus bajos requisitos en el sistema y la no necesidad de compilar el código, encontrándose su implementación en forma de librería.

2.9. Entorno de desarrollo software

A continuación se detallan las herramientas de desarrollo software para la programación del sistema obtenido.

2.9.1. Xilinx EDK

El Xilinx EDK (Embedded development kit) es el entorno de desarrollo de Xilinx para sistemas embebidos que contiene herramientas, procesadores e IPCores prediseñados. La versión utilizada es la 11.5 para Windows XP 32-bits que se incluye en el Design Suite 11.5 de Xilinx.

Se compone de dos entornos, el XPS (Xilinx Platform Studio) para el desarrollo de hardware y el Xilinx SDK (Software development kit) para el desarrollo de software.

Aunque el entorno incorpora los archivos que dan soporte a las principales placas de desarrollo, en nuestro caso se debe instalar previamente los archivos de conexionado al exterior.

El EDK permite trabajar con los procesadores MicroBlaze y PowerPC e incorpora una parte de los códigos fuente del soft-core MicroBlaze. Incluye también el código de los principales buses y asistentes para su implementación, de la misma forma que una gran cantidad de controladores IPCore para incorporar a nuestro diseño y un kernel parametrizable para sistemas de tiempo real.

2.9.2. Cypress PsoC Designer

Es la herramienta para diseñar y crear el código que programaremos en el PsoC. La versión utilizada ha sido el PsoC Designer 5.0. Esta herramienta nos permite escoger los User Modules, ubicarlos en los bloques lógicos y programar el procesador en C o en Assembler. Proporciona además los datasheets de los componentes donde indica su funcionamiento y la API

del componente, así como opciones de generación de datasheets del proyecto diseñado y la programación del dispositivo sin necesidad de usar el PsoC programmer.

2.9.3. CypressPsoc Programmer

Es la herramienta usada habitualmente para la programación del dispositivo. Para utilizarlo debemos instalar además los drivers de los PsoC que se proporcionan junto a la herramienta.

2.9.4. Avnet Programming Utility

Esta herramienta sirve para descargar nuestros diseños de FPGA a la placa de evaluación. Permite programar directamente sobre la FPGA, sobre la memoria flash paralela y sobre la memoria flash serie escogiendo el puerto COM correspondiente a nuestro dispositivo y la velocidad a la que tengamos configurado el modulo USBUART del PsoC (115200 por defecto).

2.9.5. Microsoft Visual C#

Es un entorno de desarrollo incorporado en la suite Visual Studio 2008. Se ha usado para hacer la aplicación con interfaz gráfica que se ejecutará en PC. Se trata de un lenguaje orientado a objetos para aplicaciones sobre la plataforma .NET 3 o superior de Microsoft.

2.10. Coste del sistema

Costes de personal

• **Director de proyecto:**	Horas dedicadas:	30
	Coste/hora:	35 €
	Coste total:	1050 €

• **Alumno:**	Horas dedicadas:	420
	Coste/hora:	15 €
	Coste total:	6300 €
• **Coste total del personal:**		**7350 €**

Costes hardware

• Placa de evaluación *Avnet Spartan-3A Evaluation Board*:	49€
• Circuito integrado ADXL202EB con acelerómetro ADXL202JE:	24€
• Emisor infrarojos:	0.16€
• Fototransistor:	2.61€
Coste total hardware:	***75.77 €***

Costes software

• Microsoft Office Word 2007	189 €
• Microsoft Visual Studio 2008	329 €
• Xilinx Design Suite EDK	595 €
• Avnet Evaluation Board	0 €
Coste total software:	***1113 €***
Coste total :	**8538.77 €**

2.11. Evaluación de riesgos

2.11.1. Lista de riesgos

R1.Planificación temporal optimista: estudio de viabilidad. No se acaba en la fecha prevista, aumentan los recursos.
R2.Falta alguna tarea necesaria: estudio de viabilidad. No se cumplen los objetivos del proyecto.
R3.Cambio de requisitos: estudio de viabilidad, análisis. Retraso en el resto de tareas.
R4.Equipo del proyecto demasiado reducido: estudio de viabilidad. Retraso en la finalización del proyecto, no se cumplen los objetivos del proyecto.
R5.Herramientas de desarrollo inadecuadas: implementación. Retraso en la finalización del proyecto, menos calidad.
R6.Dificultad de acceder a los *stakeholders:* estudio de viabilidad, estudio del dispositivo, diseño, pruebas... Faltan requisitos o son inadecuados, retrasos, insatisfacción usuarios.
R7.No se hace correctamente la etapa de pruebas: desarrollo, implantación. Falta de calidad, deficiencias en la puesta en marcha, insatisfacción usuarios, pérdidas económicas.
R9.Abandono del proyecto antes de la finalización: en cualquier fase. Pérdidas económicas, frustración.
R10.Avería de recursos materiales: en cualquier fase. Retraso del proyecto.

2.11.2. Catalogación de riesgos

Tabla 2.10: Catalogación de riesgos

Riesgo	***Probabilidad***	***Impacto***
R1	Alta	Crítico
R2	Alta	Crítico
R3	Alta	Marginal
R4	Alta	Crítico
R5	Baja	Crítico
R6	Baja	Crítico
R7	Alta	Crítico
R8	Media	Crítico
R9	Media	Catastrófica
R10	Baja	Crítico

2.11.3. Plan de contingencia

Tabla 2.11: Plan de contingencia

Riesgo	***Solución a adoptar***
R1	Aplazar alguna funcionalidad, afrontar posibles pérdidas.
R2	Revisar el estudio de viabilidad, modificar la planificación.
R3	Aplazar alguna funcionalidad, modificar planificación i presupuesto.
R4	Pedir un aplazamiento.
R5	Mejorar la formación del equipo, prevenir herramientas alternativas, mejorar la calidad.
R6	Fijar un calendario de reuniones, mejorar el contacto con el jefe de proyecto.
R7	Diseñar las pruebas con antelación, negociar contrato de mantenimiento, dar garantías.
R8	Revisar las normas, legislación y patentes. Consultar a un experto.
R9	No tiene solución.
R10	Prevenir la reposición de material de recambio.

2.12. Conclusiones

Dado que se ha implementado el sistema hardware mínimo y a pesar de los riesgos encontrados deducimos que este proyecto es viable.

3. Diseño del sistema

3.1. Introducción

El sistema MicroBlaze con PLB montado en la placa de prototipado consiste en un microprocesador de 32-bits con multiplicador de enteros de 32 bits por hardware y una unidad de punto flotante. La frecuencia de reloj es de 50 MHz, el pipeline es de 3 etapas y la memoria BRAM es de 32 KB dado que MicroBlaze solo nos permite escoger BRAM en potencias de 2.

3.2. Componentes principales de Microblaze

Para poder trabajar cómodamente con el microprocesador MicroBlaze resulta esencial conocer algunos archivos de configuración y el significado que tienen. Estos archivos se pueden encontrar distribuidos en las carpetas que se crean con la realización del proyecto.

System.mhs: es el archivo que contiene la descripción del soft-core MicroBlaze y de las IP's. Se describen los IPCore's utilizados y la dirección base de estos. Es uno de los archivos a verificar en los errores hardware que nos puede reportar el sintetizador.

System.mss: Es el archivo que determina el sistema operativo utilizado, el compilador del microprocesador y el driver y versión utilizado para cada IPCore.

Bitgen.ut: Contiene parámetros de configuración utilizados a la hora de crear el archivo .bit. Estos parámetros determinan entre otras cosas la velocidad de reconfiguración del FPGA, el señal de reloj que determinará el inicio del proceso de configuración y el estado de los pines en el momento de configuración.

Xparameters.h: Contiene directivas de las direcciones de memoria donde se encuentran mapeados los IPCore. Se utiliza como cabecera de programa para no tener que escribir las direcciones de acceso a los IP's a mano.

Systen.ucf: Este archivo es el que usa el sintetizador para interconexionar los pines de la FPGA con los puertos de entrada y/o salida de cada IPCore.

Proyecto.ld: Aquí se encuentra la configuración del proyecto software para ejecutar en la FPGA. Se describen los segmentos en los que se divide el programa y su ubicación, así como las direcciones de inicio de la memoria donde ubicaremos el software.

Archivo.bit: es el archivo final que se obtiene para descargar el diseño en placa. Este archivo lo podemos ubicar en la memoria serie o paralela en otro formato, estableciendo los *jumpers* para indicar que el archivo.bit se ha de buscar en una memoria o en otra cada vez que se resetee el sistema.

Archivo.bin: Es el formato de archivo para programar la memoria. En el caso de la conversión de archivo.bit a .bin los dos archivos dispondrán el mismo contenido pero distinto en formato.

Executable.elf: Es el archivo que contiene el ejecutable del programa codificado para MicroBlaze.

3.3. El método Execute in Place

Dado que nuestro sistema MicroBlaze solo dispone de 32KB para código y datos. Este espacio es insuficiente para almacenar un programa medianamente grande. Es por ese motivo que se decide buscar la forma para que la memoria flash paralela se utilice como memoria para instrucciones.

Para realizar el proceso se requiere indicar en el archivo *.ld* que segmentos ubicaremos en memoria BRAM y que segmentos ubicaremos en memoria flash y a partir de que dirección, para tal motivo se determinan los segmentos de solo lectura *.text*, *.init*, *.fini*, *.rodata*, *.sdata2* y *.sbss2* que se pueden ubicar en la memoria flash, a posteriori de la compilación se han de generar estas partes en .bin con el comando:

```
mb-objcopy –O binary –j .text –j –init …. ./Proyecto/executable.elf
./Proyecto/Codigo.bin
```

El proceso posterior es actualizar el bitstream con el archivo *.ld* para poder indicar al hardware a partir de que posición de memoria se encuentra el código. Se generará un archivo .bit que habremos de portar a .bin con un script llamado *make_bpi_image.bat* que se encuentra en la carpeta *FLASH_BURN*.

3.4. Los IPCore's

Para añadir funcionalidades al procesador es necesario utilizar periféricos y memoria. Pues bien, los IPCore no son más que controladores que describen un comportamiento para periféricos y memoria.

Para el montaje de nuestro sistema encontramos que existen algunos IPCore's ya creados que se pueden implementar en nuestro sistema. Cada IPCore dispone de una API que podemos utilizar además de estar mapeados en una dirección de memoria con lo que podemos hacer accesos de lectura y escritura de datos sobre esa dirección en función del funcionamiento que tenga el IPCore. A continuación describiremos su cantidad y su función:

Tabla 3.1: IPCores del sistema

IPCore	Número	Dispositivo que controla	Función
Microblaze	1		Microprocesador
lmb_v10	1		Bus que comunica MicroBlaze y la memoria BRAM o periféricos.
plb_v46	1	bus de datos	IPCore que define el bus PLB.
lmb_bram_if_cntrl	2	lmb_v10	Controlador de memoria BRAM que conecta el bus con ésta.
xps_mch_emc	1	Memoria flash paralela	Controlador de memoria flash paralela
bram_block	1	controlador de memoria BRAM de datos y de instrucciones	Bloque de memoria BRAM.
xps_gpio	1	Botones Cap Sense	Controla los elementos de entrada/salida

IPCore	Número	Dispositivo que controla	Función
xps_iic	1	Sensor de temperatura	Incorpora el protocolo I2C para control del TMP100
xps_intc	1	Señales de interrupción	Controlador de interrupciones
xps_timer	1	Contador de ciclos	Contador de ciclos de 32-bits
xps_uartlite	1	Conexión serie con PSoC	Controlador protocolo serie
clock_generator	1	Divisor de señal	Genera una señal de reloj de 50 MHz a partir del reloj de entrada de 16 MHz
proc_sys_reset	1		Restaura el sistema

3.5. Pruebas de rendimiento

Las pruebas de rendimiento o *Benchmarks* se utilizan para medir la capacidad de cálculo de un microprocesador. De forma ideal se deben hacer *benchmarks* orientados al propósito ideal al que se vaya a destinar la aplicación, pero de forma general se utilizan benchmarks sintéticos de cálculo para medir el potencial de un componente, en este caso el microprocesador. Los benchmarks utilizados son Dhrystone para cálculo con enteros y Whetstone para cálculo en punto flotante, ambos sintéticos.

3.5.1. Dhrystone

Dhrystone es un benchmark que mide el potencial de cálculo con enteros. Está hecho en C lo que le permite su fácil portabilidad entre distintos sistemas, aunque siempre requiere de la intervención del usuario para modificar algunos parámetros y añadir o eliminar algunas directivas de C en función del compilador utilizado. La versión utilizada es la 2.0. Los Dhrystones son cantidad de iteraciones y los VAX MIPs son millones de instrucciones por segundo. Las pruebas de compilación optimizada se indican con el símbolo ***(o)***.

Tabla 3.2: Benchmark Dhrystone

CPU	Reloj(MHz)	Compilador y configuración	Dhrystones por segundo	VAX MIPs	MIPs por MHz
ZiLOG Z8F6422	18.432	ZDS II 4.8	2105	1.20	0.065
Atmel ATmega64	14.7456	AVR-GCC 3.3.2 (o)	8487	4.83	0.328
TI MSP430F149	8.000	CrossWorks	4047	2.30	0.288
MicroBlaze	50.000	GCC 4.1.1 (o)	6667	3.79	0.076
AM386/387	40.000	Watcom C 10.5	7959	4.53	0.113
AM386/387	40.000	Watcom C 10.5 (o)	24070	13.7	0.342
MAXQ2000	8.000	CrossWorks	6762	3.85	0.481
MAXQ2000	20.000	CrossWorks	16906	9.62	0.481
IBM 486 D2	50	Watcom C 10.5 (o)	39356	22.4	0.448
IBM 486 D2	50	Watcom C 10.5	13867	7.89	0.158

3.5.2. Whetstone

El Whetstone es un benchmark sintético para cálculos en coma flotante. A continuación se muestra una tabla comparativa. Los casos optimizados se indican con "o". Existen versiones para el cálculo en doble precisión y en simple. Dado que la FPU que incluye el sistema es de simple precisión hemos utilizado la versión de simple precisión (*float*).

Tabla 3.3: Benchmark Whetstone single precision

CPU	Reloj(MHz)	Compilador y configuración	MWIPS
MicroBlaze Sp3A	50,000	11.1 EDK-GCC 4.1.1 (o)	3.846
AM386/387	40,000	Watcom C/C++ 10.5 (o)	5.68
AM386/387	40,000	Watcom C/C++ 10.5	3.07
80486DX2	66,000	Watcom C/C++ 10.5 (o)	15.3
80486DX2	66,000	Watcom C/C++ 10.5	9.01

3.6. Conclusiones

El sistema base montado obtiene unos resultados medios comparados con algunos otros microcontroladores de silicio. A pesar de esto obtiene resultados de varios millones de instrucciones por segundo tanto en coma flotante como en enteros y hay que tener en cuenta que el desarrollo en FPGA's es más lento que en silicio. Aún así es un sistema que se compara con procesadores de la serie 386 que también tienen una arquitectura de 32-bits.

El motivo por el que obtenemos tan pocos Dhrystones por MHz se debe a que el código de Dhrystone no cabe en memoria BRAM (más rápida que la flash) y ello perjudica al rendimiento real del sistema. En cambio, con Whetstone, los millones de instrucciones por segundo es casi comparable al rendimiento de un AM386 y supera a los de enteros.

Los resultados obtenidos son suficientes para el sistema al que se enfoca dado que podríamos obtener repuesta en tiempos inferiores al milisegundo.

4. Implementación

4.1. Medida de la temperatura

4.1.1. Obtención de los datos

Para la medida de la temperatura es esencial conocer el funcionamiento del dispositivo que estamos tratando.

El sensor de temperatura dispone de 4 registros internos y un registro apuntador de 8 bits, éstos pueden ser de solo lectura o de lectura/escritura, la correcta configuración de estos registros define la acción que queremos llevar a cabo. Los dos bits menos significativos del apuntador indican el registro sobre el cual queremos interactuar.

Tabla 4.1: Registros sensor de temperatura

P1	P0	Registro
0	**0**	Temperature Register (READ ONLY)
0	**1**	Configuration Register (READ/WRITE)
1	**0**	T_{LOW} Register (READ/WRITE)
1	**1**	T_{HIGH} Register (READ/WRITE)

Realizar una lectura en uno de estos registros es tan simple como pasar por el bus I^2C el número de registro del que queremos obtener los datos.

Para hacer una escritura se enviarán a través del bus dos bytes, uno que indique el registro al que queremos escribir y otro para indicar el valor a escribir.

En la aplicación se utilizan únicamente los registros *Configuration Register* y el *Temperature Register*. *Configuration Register* es un registro de 8 bits de lectura/escritura en el que dos bits indican la precisión con la que obtendremos la temperatura. La precisión escogida por nosotros son 12 bits de precisión por lo tanto según las figuras 4.1 y 4.2 tendremos que ponerlos a '1'.

Tabla 4.2: Disposición bits de Configuration Register

Byte	D7	D6	D5	D4	D3	D2	D1	D0
1	OS/Alert	**R1**	**R0**	F1	F0	POL	TM	SD

Tabla 4.3: Configuración resolución de temperatura

R1	R0	Resolución
0	**0**	9 bits (0.5 grados)
0	**1**	10 bits (0.25 grados)
1	**0**	11 bits (0.125 grados)
1	**1**	12 bits (0.0625 grados)

De esta forma la primera actuación es enviar a través del I^2C la cadena 0x0160. Donde el "01" indica el registro a acceder (*Configuration Register*) y el "60" el número a escribir en el registro en hexadecimal.

A continuación, para realizar una lectura debemos acceder al registro *Temperature Register*. Este registro es de 12 bits y contiene la última lectura de temperatura. Para obtener su resultado debemos poner el registro de puntero apuntando él. Para ello hacemos una escritura sobre el I^2C de la cadena "0x00". Con esto ya podremos realizar una lectura y obtener dos bytes siguiendo la siguiente disposición:

Tabla 4.4: Byte # 1 de Temperature Register

D7	D6	D5	D4	D3	D2	D1	D0
T11	T10	T9	T8	T7	T6	T5	T4

Tabla 4.5: Byte #2 de Temperature Registeer

D7	D6	D5	D4	D3	D2	D1	D0
T3	T2	T1	T0	-	-	-	-

El primer byte contiene la parte entera de la temperatura mientras que la parte decimal se encuentra en los cuatro MSB del segundo byte.

Para captar estos bytes debemos tratar con variables tipo *char*. Para la lectura correcta de la parte decimal de temperatura debemos desplazar los cuatro MSB del segundo byte hacia la derecha y posteriormente tratarlos, multiplicando por 10000 y dividiendo por 16, para obtener el número entero correspondiente en la representación entera.

4.1.2. Programación

El primer paso para poder tratar con el sensor de temperatura es configurar una interrupción para este dispositivo en el kernel y habilitarla. Esto nos permite saber si el bus está libre o no (*ver 2.6.6*). Para ello debemos obtener el id de la interrupción a tratar.

```
int id = XPAR_XPS_INTC_0_TEMP_SENSOR_IIC2INTC_IRPT_INTR;
```

A continuación debemos registrar esta interrupción en el kernel y habilitarla.

```
register_int_handler(id, (XInterruptHandler) XIic_InterruptHandler,
IicInstPtr);

enable_interrupt (id);
```

El siguiente paso es crear una instancia de protocolo I^2C, inicializar la instancia con el dispositivo I2C que queramos tratar y configurar los manejadores de interrupción con las funciones *DatoEnviado* y *DatoRecibido*:

```
#include "xiic.h"

#include "xintc.h"

XIic IicInstance;

XIic_Initialize(&IicInstance, XPAR_IIC_0_DEVICE_ID);

      XIic_SetSendHandler(&IicInstance, &IicInstance,(XIic_Handler)
DatoEnviado);

      XIic_SetRecvHandler(&IicInstance, &IicInstance,(XIic_Handler)
DatoRecibido);
```

Se ha de configurar la dirección esclavo a la que debemos acceder donde *XII_ADDR_TO_SEND_TYPE* se configura a la dirección SLAVE_ADDRESS que indica el fabricante del dispositivo, para nuestro caso es 0x48 utilizando un único dispositivo esclavo en el bus. A continuación enviamos los datos, esperando a continuación que se haya ejecutado la rutina de interrupción *DatoEnviado* mediante la comprobación de una variable global.

```
XIic_SetAddress(&IicInstance, XII_ADDR_TO_SEND_TYPE,SLAVE_ADDRESS );

if (XIic_MasterSend(&IicInstance, WriteBuffer, ByteCount) != XST_SUCCESS);
```

Una vez que hayamos escrito los bytes indicados en el apartado 4.1 debemos hacer una lectura sobre un vector de dos bytes. En nuestro caso es *BufferPtr*:

```
XIic_MasterRecv(&IicInstance, BufferPtr, ByteCount);
```

Entonces podemos parar el dispositivo i2c con la siguiente función para que quede disponible para que otro periférico lo pueda coger.

```
XIic_Stop(&IicInstance);
```

4.2. IPCore acel2axis

Para el tratado de la aceleración es necesario hacer un IPCore que tenga como puertos de entrada las salidas X e Y del acelerómetro, además el asistente de EDK para la generación de IPCores pone a nuestra disposición las señales del bus que queramos utilizar. Para su realización hemos optado por el lenguaje VHDL dado que es el lenguaje de descripción hardware más común en Europa.

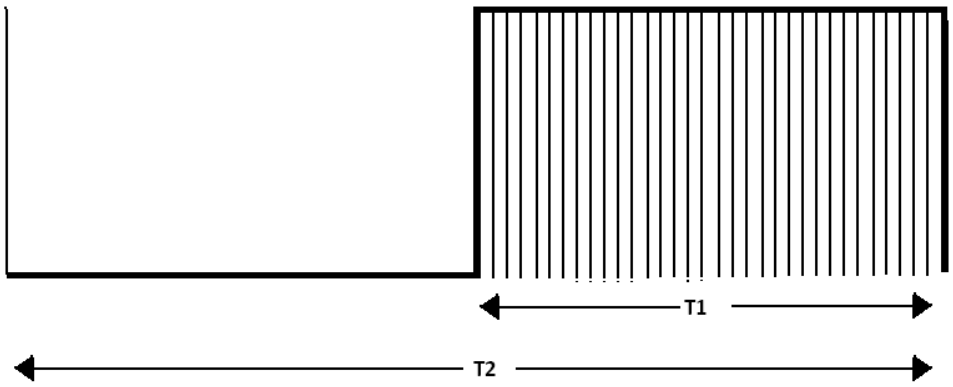
Esta IP debe demodular el ancho de pulso, obteniendo que porcentaje del periodo la señal del eje X está a '1' y tratarlo con los cálculos de la fórmula del punto 2.7.1.

4.2.1. Lectura pulso activo

El IPCore necesita contar mediante un señal de reloj los pulsos que se encuentran activos en un periodo del acelerómetro. Para realizar este proceso haremos servir el reloj del BUS que es de 50 MHz, y dado que la frecuencia del acelerómetro es de 500 Hz se nos proporcionan 100 mil ciclos de bus por cada uno del acelerómetro. Dado que lo que estamos diseñando es hardware cuantos más procesos hagamos más rápido funcionará, de este modo hemos optimizado el IPCore para que pueda trabajar a la máxima velocidad posible.

Por cada flanco de subida incrementamos una señal T2 que determinará el ancho del periodo. A su vez, por cada flanco de subida que la señal del eje X esté activa incrementamos otra señal T1 que indicará el número de ciclos del pulso activo.

Figura 4.1: Cuenta del ancho de pulso activo



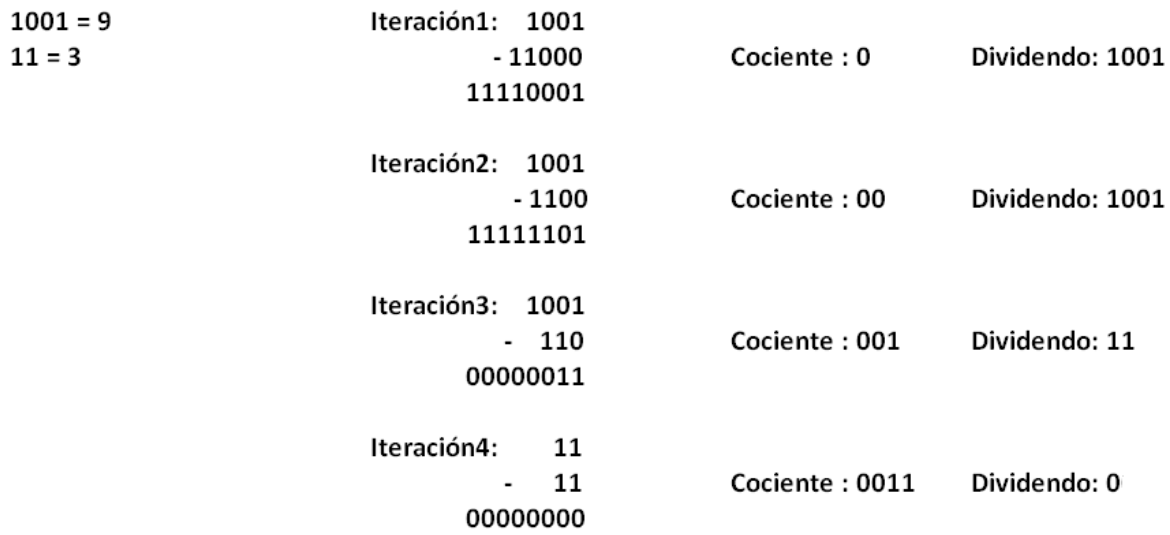
4.2.2. Demodulación de ancho de pulso

Una vez tenemos el tiempo que está activo la señal cuadrada y el periodo total debemos realizar la división T1/T2 y, dado que el sintetizador de Xilinx no da soporte a la división en VHDL, debemos diseñar uno.

El divisor a implementar es un divisor en serie sin resto. El funcionamiento de este divisor proporciona un bit de resultado por ciclo de reloj, para evitar trabajar con decimales se ha multiplicado T1 por un valor suficientemente grande para determinar con precisión la aceleración. El valor utilizado ha sido 1024, de esta manera debemos tenerlo en cuenta en el cálculo final.

La división serie empieza por situar el LSB del divisor en la posición del MSB del dividendo, realizar la resta *dividendo-divisor* y el MSB del resultado entra negado en el *cociente*, que es un registro de desplazamiento hacia la izquierda. Además, en el caso de que la resta no produzca *overflow* se toma como nuevo valor de dividendo el resultado de la resta, este proceso se repite *n* veces donde *n* es el número de bits del dividendo.

Figura 4.2: División serie con desplazamiento



4.2.3. Cálculo de la aceleración

El proceso posterior a la división se ejecuta mediante la activación de un bit que se pone a '1' cuando se ha hecho la división. Este proceso descarta las señales y trabaja con variables, dado que el valor de una variable se modifica de manera atómica, no nos perjudicará en el retardo final.

Para no perder precisión debido a la existencia de decimales se ha dejado para el final la única división por el valor 1024 que antes multiplicamos.

El código del último proceso es:

```
if done_s_x='1' then
      calculo_x := conv_integer(percent_x(Tam_T2-1 downto 0));
      calculo_x := calculo_x - K2X; --Extrapolamos al centro
      calculo_x := calculo_x * PWMperGinvX; --Multiplicamos por la inversa
de Ancho de pulso por g
      calculo_x := calculo_x/K; --Dividimos por el factor K que nos ayudo a
                              --definir la precision de la multiplicacion
      acel_x <= conv_std_logic_vector(calculo_x,16);
end if;
```

El primer cálculo en este caso es el de extrapolar el valor obtenido en el centro, esto es restar al cociente de la división el valor 512 que es el 50% del factor antes multiplicado (hay que tener en cuenta que hemos obtenido un valor sobre la escala 1024). El resultado irá de +256 a -256.

El segundo cálculo debería realizar una división más para obtener la aceleración final, pero dado que conocemos la constante podemos hacer el proceso inverso, esto consiste en multiplicar por 1/128 que es aproximadamente 0.08. Estos valores se han modificado en el código multiplicándolos por 1000 para no tener que tratar decimales.

El último paso es el de dividir por el factor que antes multiplicamos, es decir, 1024.

El valor final lo convertimos a una señal vector de bits y lo conectamos a la salida del IPCore.

Adicionalmente, en esta IPCore se trata el encendido de los LED's en función de la aceleración, este proceso se realiza con un contador y un valor de PWM que varía en función de los bits más altos de la aceleración en centésimas de g.

4.2.4. Mapeado de puertos

Una vez disponemos del IPCore podemos incorporarla a nuestro microprocesador, pero antes de sintetizar debemos modificar el archivo .ucf de nuestro proyecto y configurar los puertos con la variable LOC, y generar las direcciones automáticamente con el botón *Generate Adress* de la *System Assembly View* de EDAK. Tal como se dijo en la conexión del acelerómetro corresponden a los puertos C13 y A13.

```
Net acel2axis_0_entrada_x_pin LOC=C13 | IOSTANDARD = LVCMOS33;
Net acel2axis_0_entrada_y_pin LOC=A13 | IOSTANDARD = LVCMOS33;
Net acel2axis_0_led_x_positive_pin LOC=B15 | IOSTANDARD = LVCMOS33;
Net acel2axis_0_led_x_negative_pin LOC=C15 | IOSTANDARD = LVCMOS33;
Net acel2axis_0_led_y_positive_pin LOC=C16 | IOSTANDARD = LVCMOS33;
Net acel2axis_0_led_y_negative_pin LOC=D14 | IOSTANDARD = LVCMOS33;
```

4.2.5. Calibrado de acelerómetro

Al realizar las pruebas iniciales del acelerómetro éste daba resultados erróneos. Estos dispositivos tienen un cierto porcentaje de imprecisión, de manera que al adquirir dos acelerómetros idénticos y realizando la misma medida con el mismo programa obtengamos resultados diferentes en ambos. Existen dos métodos de calibrar el acelerómetro de los que solo se ha podido implementar uno.

El calibrado de alta precisión requiere dar un giro de más de 360° al acelerómetro con el eje Z encarado al horizonte con lo que se obtienen todos los valores entre -1 y +1 *g*'s y obtener el máximo y mínimo valor de T1 y T2. Éste método se intentó llevar a cabo pero su sensibilidad siempre provocaba valores distintos en cálculos sucesivos.

El segundo calibrado es el implementado. Consiste en tomar los valores de T1 y T2 para +1 y -1 g. A partir de estos datos se determina el ancho de pulso en 0 *g*'s por el valor medio y el porcentaje de ancho de pulso por la diferencia entre 0 g y 1 g dividido por el valor T2 medio.

4.3. Sensor óptico

El sensor óptico del que se dispone envía una señal analógica de entre 0 y 1.6 volts. Para la medida de ésta es necesario utilizar el PSoC. Como ya hemos visto en el estudio de viabilidad existen tres puertos analógicos de entrada/salida en el PSoC y once líneas libres que conectan el PSoC y la FPGA.

4.3.1. Conversión de la entrada

Para obtener el valor de la señal analógica del PSoC necesitamos colocar un módulo SAR6 en los bloques analógicos, éste es un conversor analógico a digital de 6 bits de resolución, esta opción es debida a que no hay espacio suficiente en la configuración de serie para incorporar un ADC de más capacidad.

Dado que el SAR6 es del tipo *switch capacitor* el sistema requiere montar un bloque analógico del tipo *CT* que no nos modifique la señal de entrada. El único amplificador para obtener ganancia 1 es el de ganancia programable (PGA).

De esta manera conectamos la entrada del PGA al Port_0_1, y la salida de éste a la entrada del SAR6 y lo configuramos para ganancia 1. Tal como lo muestran las figuras 4.3 y 4.4.

Figura 4.3: Configuración bloques PSoC

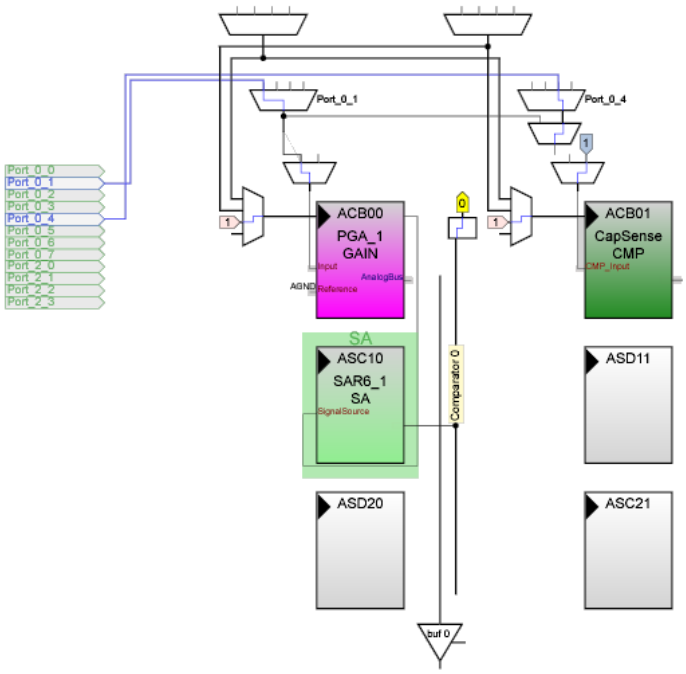


Figura 4.4: Configuración puerto de entrada

P0[1]	A_GPI_1, AnalogInput, High Z Analog, DisableInt
Name	A_GPI_1
Port	P0[1]
Select	AnalogInput
Drive	High Z Analog
Interrupt	DisableInt

4.3.2. Protocolo comunicación PSoC-FPGA

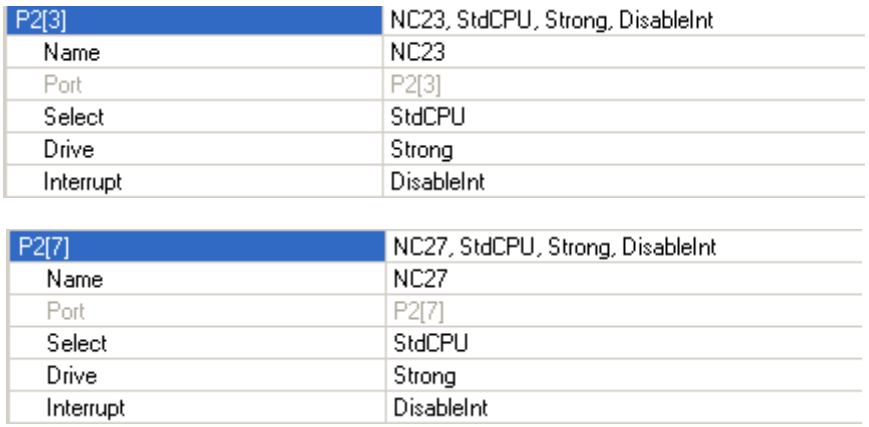
PSoC

Dado que no existe ningún puerto del PSoC que comunique enteramente con la FPGA para montar un protocolo paralelo. Se ha montado un protocolo serie síncrono con 2 líneas que lleguen del PSoC a la FPGA.

El puerto escogido ha sido el Port_2 y las líneas son la [3] para datos y [7] para el reloj. De esta manera la línea [7] se comporta como señal de reloj pasando de '0' a '1' en cada bit de dato enviado y de '1' al terminar la transferencia del bit. Este proceso se repite 6 veces para enviar los 6 bits.

Estos puertos se deben configurar a tipo *Strong* que significa que su valor solo será '0' o '1'.

Figura 4.5: Configuración puertos PSoC protocolo



A continuación se muestra el código base del protocolo para el envio del bit menos significativo de la muestra recibida, este proceso se repite 6 veces excepto la primera línea.

```
car = SAR6_1_GetSample();
PRT2DR &= 0x77;
if ((car & 0x01)==0x01) PRT2DR |= 0x88;
else PRT2DR |= 0x80;
```

FPGA

Para captar los datos en la FPGA debemos hacer un nuevo IPCore al que llamamos *ip2* para recoger los datos con la señal Port2[7] del PSoC activa. Para ello necesitamos un registro de 6 bits donde siempre tenderemos la salida, un contador incremental y un registro en el que el bit de la posición del contador se iguala al valor de la señal dato.

Dado que no se trata de una señal de reloj, el sintetizador VHDL no permite la misma implementación que la señal de reloj, por lo que se ha tenido que tomar una opción equivalente que espere el señal de reloj '1' para asignar el bit recibido al bit correspondiente del contador. El código se muestra a continuación:

```
if(PSoC_SAR_Ck = '1') then
      Data(conv_integer(count_bits))<=PSoC_SAR_Data;
      if(count_bits = "101") then
            count_bits <= "000";
            sortida(5 downto 0) <= Data(5 downto 0);
```

```
        else
                count_bits <= count_bits + 1;
        end if;
end if;
wait until PSoC_SAR_Ck = '0';
```

4.3.3. Mapeado de puertos

De la misma forma que se hizo con el IPCore acel2axis debemos mapear los puertos del IPCore en el archivo .ucf del proyecto. En este caso se han usado las conexiones *H1* y *G2*.

```
Net ip2_0_PSoC_SAR_Ck_pin LOC=H1 | IOSTANDARD = LVCMOS33;
Net ip2_0_PSoC_SAR_Data_pin LOC=G2 | IOSTANDARD = LVCMOS33;
```

4.4. Programa MicroBlaze

El programa para MicroBlaze se encarga de obtener los datos de los diferentes IPCores, hacer los cálculos necesarios y mandarlos a través del puerto UART al PSoC que se comunica con el PC. A continuación se explican los elementos más relevantes.

4.4.1. Envío a PC

Para realizar el envío de datos a PC el sistema más fácil es configurar la *stdout* como el IPCore USBUart. De esta forma cada vez que escribamos un dato en la salida estándar será enviado a PC.

Las funciones de escritura en la salida estándar de MicroBlaze no permite la escritura de variables en punto flotante. Es por ello que se requiere de tratar los datos para poder enviarlos como tal mediante la colocación de un punto entre el tipo *int* que representa el decimal y el que representa al fraccionario tal como se indica a continuación.

```
xil_printf("Y%d.%d\n",velocidadenty,velocidaddecy);
```

4.4.2. Cálculos tratados

Los cálculos realizados por el programa van relacionados con la aceleración. Dado que el IPCore sólo envía aceleraciones en centésimas de g, el procesador principal ha de calcular la aceleración en m/s^2.

Dado que MicroBlaze no puede tratar con variables de punto flotante para enviarlas al *stdout* , pero sí las de entero nos limitamos a realizar cálculos para obtener los decimales en variables separadas.

Dado que 1 g = 9.81 m/s^2 entonces 1 cg = 0.0981 m/s^2, que equivale a 9.81 cm/s^2. Si lo que queremos es obtener mm/s^2 debemos entonces multiplicar por 98.1 el valor de las centésimas.

$$Aceleración\ en\ mm/s^2 = centésimas\ de\ g \times 98.1$$

Ahora queremos separar los m/s^2 y los mm/s^2. Con especial atención a los últimos porque no queremos que sean negativos, utilizamos la función *abs* para obtener el valor absoluto:

$$m/s^2 = Aceleración/1000$$

$$(mm/s^2\ \%\ 1000\ ) = abs(\ Aceleración\%1000)$$

A continuación debemos calcular la velocidad, que gracias al *timer* y a la API de *xilkernel* podemos sacar mediante los milisegundos transcurridos desde la última lectura. La velocidad la queremos en m/s con un decimal por lo que realizamos:

$$Velocidad\ en\ m/s = (m/s^2 \times miliseg)/1000 + (mm/s^2 \times miliseg)/10^6$$

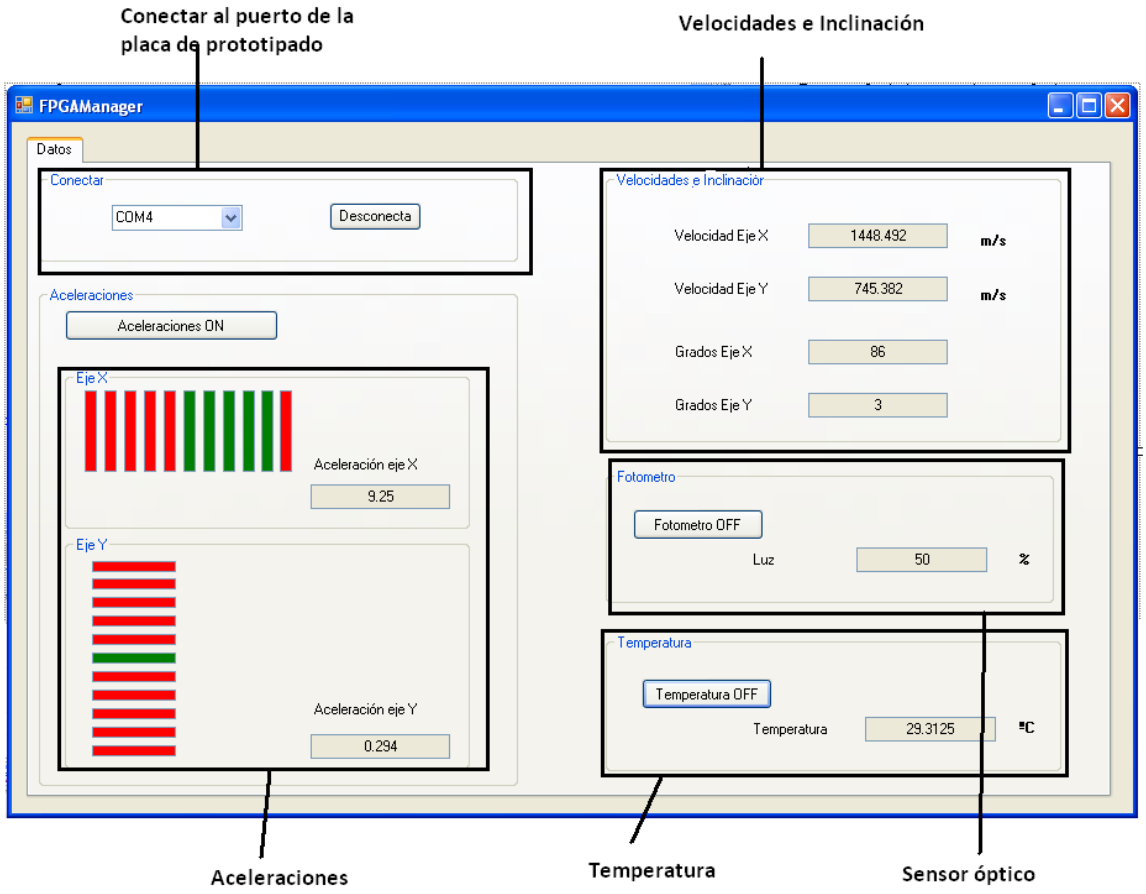
$$Velocidad\ en\ (mm/s\ \%\ 1000) = [\ (mm/s^2 \times miliseg)/1000\ ]\ \%\ 10^3$$

Para obtener los grados de inclinación, finalmente, solo hemos de multiplicar la aceleración en centésimas de g por el valor 0.9 ya que 1 g indica una inclinación de 90°.

$$Inclinación\ en\ grados = centésimas\ de\ g \times 0.9$$

4.5. Aplicación para PC

La aplicación diseñada para PC es un programa en Visual C# que monitoriza los datos recibidos y se muestran en varias cajas de texto donde se puedan visualizar los datos captados por el FPGA. Este software permite además interactuar con la placa de prototipado mediante los botones dedicados a ello y actúa un vúmetro en función de la aceleración recibida en caso del encendido.


Conectar al puerto de la placa de prototipado
Velocidades e Inclinación
FPGAManager
Datos
Conectar
COM4
Desconecta
Aceleraciones
Aceleraciones ON
Eje X
Aceleración eje X
9.25
Eje Y
Aceleración eje Y
0.294
Velocidades e Inclinación
Velocidad Eje X
1448.492
m/s
Velocidad Eje Y
745.382
m/s
Grados Eje X
86
Grados Eje Y
3
Fotometro
Fotometro OFF
Luz
50
%
Temperatura
Temperatura OFF
Temperatura
29.3125
ºC
Aceleraciones
Temperatura
Sensor óptico

5. Pruebas

5.1. Introducción

En este apartado se muestran los diferentes tipos de pruebas que se han realizado durante el proyecto. Estas pruebas deben proporcionar los resultados correctamente medidos. En caso de encontrar un fallo en la medida se debe recurrir a explorar el motivo que puede provocar ése fallo. La mayoría de las pruebas se llevan a cabo en HyperTerminal de Windows verificando además el correcto envío de los datos.

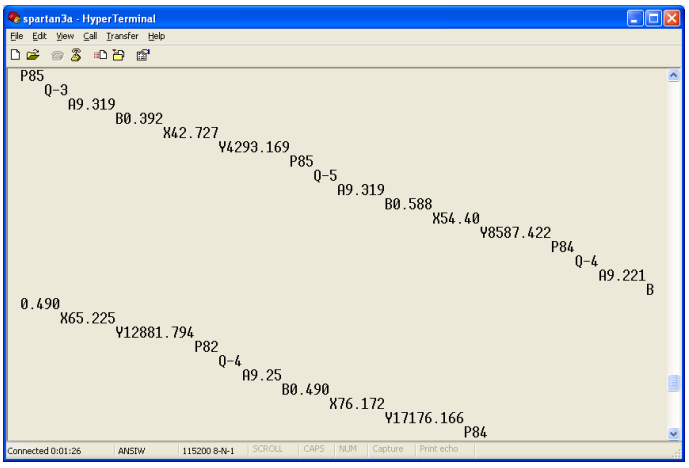
5.2. Pruebas del acelerómetro

Las pruebas sobre el acelerómetro verifican la medida correcta de la aceleración, la velocidad y los grados de inclinación. Para esta prueba hay que identificar que el acelerómetro se encuentra con el eje X mirando hacia arriba (1 g) y el eje Y mirando al horizonte (0 g). La nomenclatura de medida es:

Para el eje X: P = grados, A = aceleración en m/s^2, X = velocidad en m/s

Para el eje Y: Q = grados, B = aceleración en m/s^2, Y = velocidad en m/s

Figura 5.1: Pruebas acelerómetro

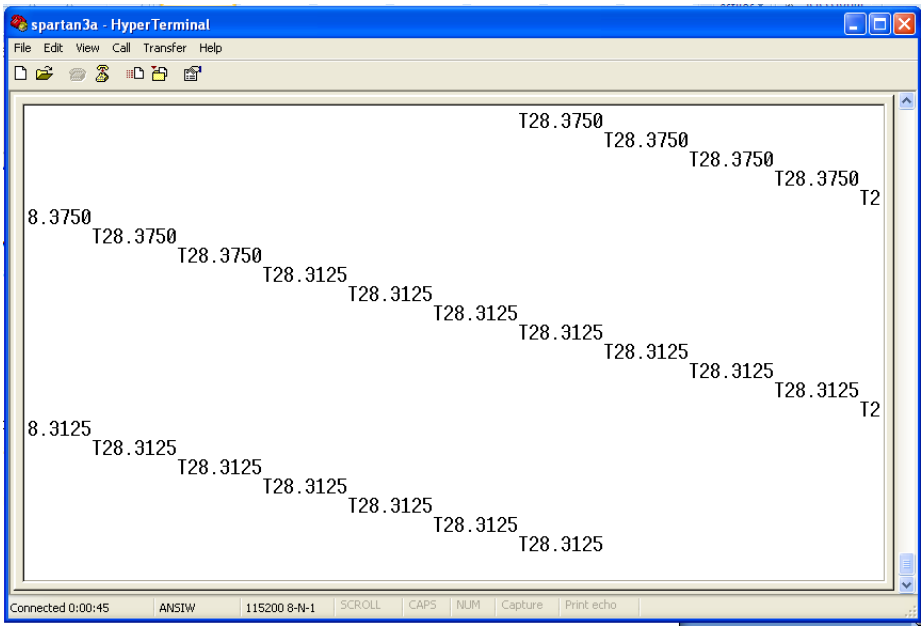


En la figura podemos ver los valores que corresponden, no se mantiene estable dado que es un elemento muy sensible y pequeño.

5.3. Pruebas del sensor de temperatura

Los resultados satisfactorios del sensor de temperatura requieren principalmente el tratado correcto de los bytes recibidos y la precisión del sensor. Tal y como se indica en el punto 2.6.6 existe un rango de error de ±3° y el hecho de encontrarse el sensor tan cercano a la placa varia el resultado. Para asegurar el funcionamiento correcto se ha estimado la temperatura ambiente.

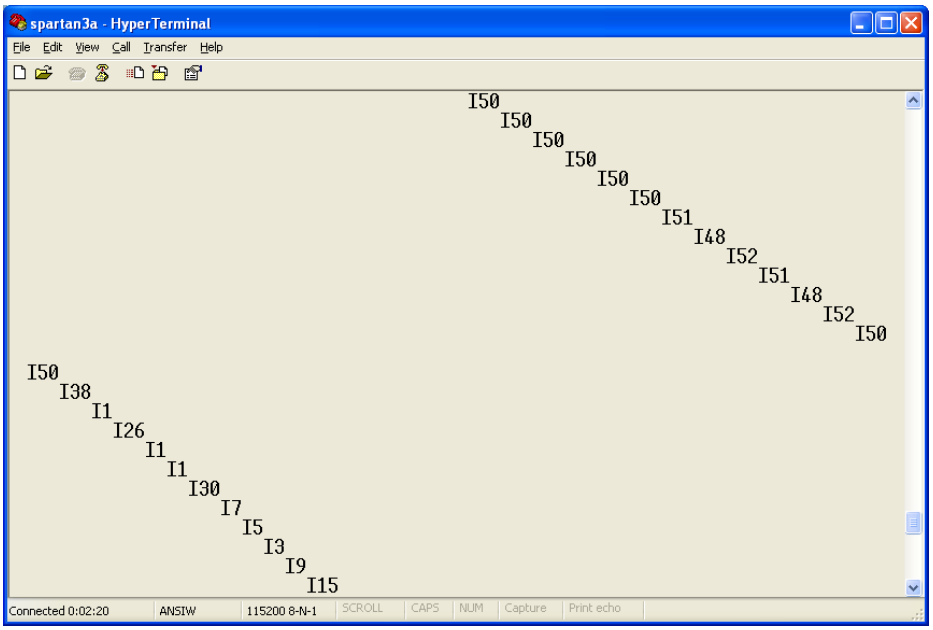
Figura 5.2 : Pruebas sensor de temperatura



5.4. Pruebas del sensor óptico

Sobre el sensor óptico las pruebas se han de hacer a oscuras para asegurar que la única fuente de fotones es la que llega del fotoemisor. Para asegurar su correcto funcionamiento el valor obtenido se debe ver reducido al interponer obstáculos. El elemento escogido ha sido una hoja de papel con varios pliegues para visualizar la variación.

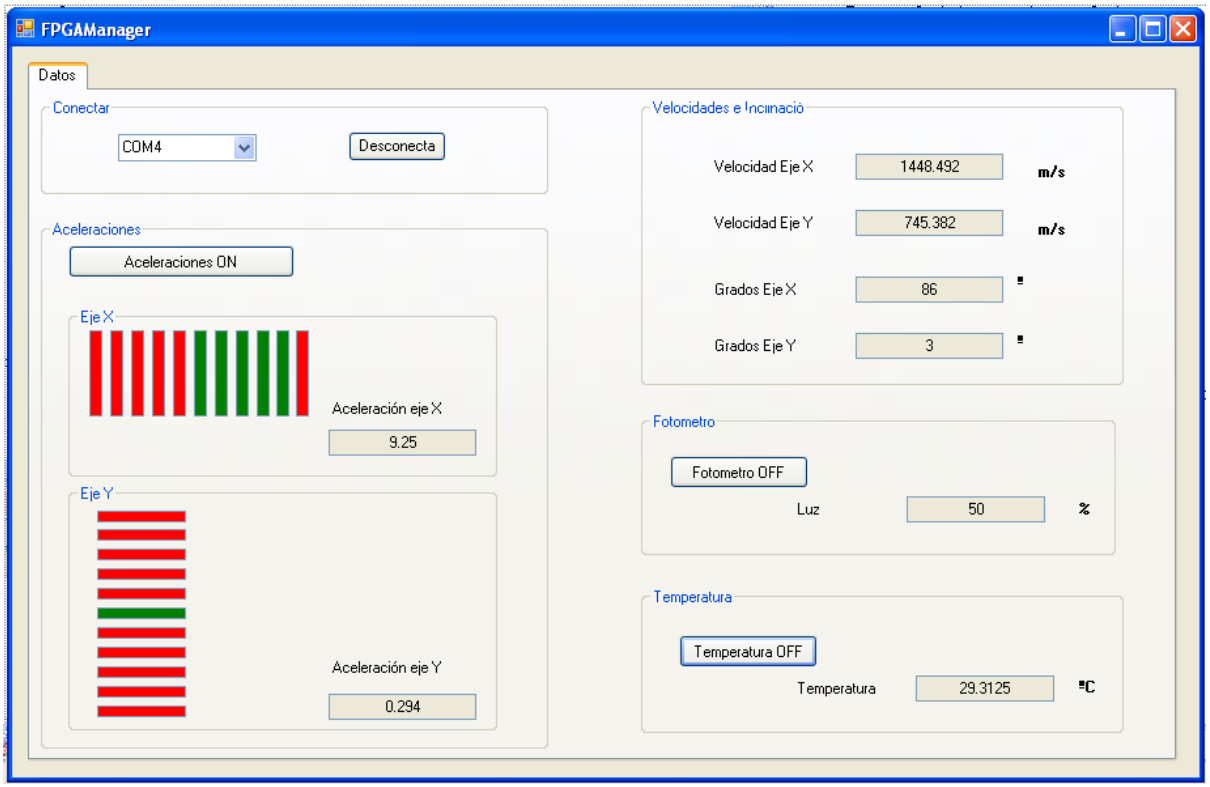
Figura 5.2: Pruebas sensor óptico



Se puede visualizar claramente como al entrar la hoja plegada en el sensor se distorsiona el valor a menos de 50 unidades. La nomenclatura precede a los datos del sensor óptico.

5.5. Pruebas del sistema global

Las pruebas del sistema global implican la correcta visualización en la aplicación PC de los datos recibidos. Como se ha comprobado se modifican todas las varibles.



6. Conclusiones y futuras ampliaciones

6.1. Conclusiones

El objetivo de este proyecto era ver la viabilidad de proyectos para el control de ABS, ASR y ESP sobre el hardware objeto. Dado que hemos conseguido la medida de las principales variables que pueden influir en este tipo de aplicaciones y disponemos de un hardware estable para la medida de estos datos y su tratado en tiempo real.

El hardware, además trata la temperatura del disco de freno para la actuación del ABS. Un variable que no se ha tenido en cuenta en ningún dispositivo de ABS.

Aunque el hecho de medir la velocidad angular de la rueda con un sensor óptico no es una buena solución, este se puede cambiar por un sensor electromagnético sin necesidad de grandes modificaciones en la motocicleta.

El bajo coste del sistema también influye en proponer una implementación para sistemas de seguridad muy poco estandarizados y que deberían tenerse muy presentes. Más aún tratándose de motocicletas, donde el conductor está desprotegido y resulta muy fácil tener una caída.

Aún así, la implementación del proyecto en la vida real debería ser comprobada por los ingenieros de los fabricantes de estos sistemas de seguridad dado el poco conocimiento en la materia técnica en los sistemas hidráulicos de frenos incorporados en los sistemas comerciales.

La motivación por este proyecto, que empezó siendo un mero interés por el desarrollo hardware, ha hecho que me introdujera en los desarrollos de sistemas embebidos digitales y analógicos, así como ha aportado un gran conocimiento al alumno acerca del desarrollo con microprocesadores soft-core y el desarrollo de IP's destinados a MicroBlaze.

6.2. Futuras Ampliaciones

Este sistema hardware ha sido desarrollado para permitir ampliaciones mediante software. La ampliación de este sistema requiere de multitud de cálculos que contemplen las situaciones descritas. Es por tal motivo que esas ampliaciones quedarían en otro marco.

Sobre las ampliaciones hardware deberían ir dedicadas a proporcionar dos señales, una que se activa en el caso de que se deba frenar y la otra que se activa en el caso de soltar el freno, es por ese motivo por el que se han reservado los pines C8 y E7 para tal fin.

7. Bibliografía

ABS, ESP y ASR:

- http://www.csharpcorner.com/uploadfile/mgold/xyplotcontrol09222005015109am/xyplotcontrol.aspx
- http://www.circulaseguro.com/vehiculos-y-tecnologia/conduccion-de-motocicletas-la-fisica-2
- http://www.supermotor.com/revista/actualidad/231793/motocicletas:-trucos-para-conducir-invierno.html

Acelerómetro:

Datasheets

- http://www.xilinx.com/ipcenter/catalog/logicore/docs/microblaze_risc_32bit_proc_final.pdf
- http://www.alldatasheet.net/datasheet-pdf/pdf/48923/AD/ADXL202JE.html

Kit Spartan-3 eval board:

Sección Spartan-3A Evaluation Board – Página del fabricante

- http://www.em.avnet.com/

AMBA:

Página del fabricante

- http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.amba/index.html

Benchmarks:

- http://www.benchmarkhq.ru/english.html?/be_cpu.html
- http://www.iuma.ulpgc.es/~nunez/procesadoresILP/DhrystoneMIPS-CriticismbyARM.pdf
- http://www.xanthos.se/~joachim/vaxmips.html
- http://www.futuretech.blinkenlights.nl/perf.html

- http://www.ecrostech.com/Other/Resources/Dhrystone.htm
- http://homepage.virgin.net/roy.longbottom/whetstone.pdf

Precios componentes:

- http://es.farnell.com/
- http://emea.microsoftstore.com/es/es-ES/Microsoft/Office/Programas-2010

Protocolos:

- http://www.comunidadelectronicos.com/articulos/i2c.htm
- http://www.i-micro.com/pdf/articulos/spi.pdf

PSoC:

- http://www.cypress.com/?rID=3371
- http://www.cannic.uab.es/docencia/cannicframedocencia.htm
- http://www.psoc-chile.es.tl/

Leon3:

- http://www.sparc.org/standards/V8.pdf
- http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=53
- http://www.gaisler.com/cms/index.php?option=com_content&task=section&id=5&Itemid=51
- http://www.gaisler.com/doc/leon3_product_sheet.pdf
- http://permalink.gmane.org/gmane.comp.hardware.opencores.leon-sparc/12165

MicroBlaze :

- http://www.xilinx.com/support/documentation/index.htm
- http://www.xilinx.com/ipcenter/catalog/logicore/docs/microblaze_risc_32bit_proc_final.pdf

IPCores:

- http://opencores.org/

- http://www3.euitt.upm.es/taee/Congresosv2/2008/papers/2008SP108.pdf
- http://en.wikipedia.org/wiki/Division_%28digital%29

PLB:

- http://electronicdesign.com/article/boards-modules-systems/page/page/1/coreconnect-the-on-chip-bus-system4089.aspx

Xilkernel:

- http://ececmpsysweb.groups.et.byu.net/cmpsys.2005.winter/teams/ibox/groups/ibox/public/aubrey/index.html
- http://warp.rice.edu/trac/wiki/xilkernel_ref

Otras fuentes:

- http://www.xilinx.com/support/