UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

INSTITUTO DE INGENIERÍA

PROCESAMIENTO DE SEÑALES DE SENSORES DE NAVEGACIÓN PARA SATÉLITES SOBRE UNA PLATAFORMA FPGA

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA

INGENIERÍA ELÉCTRICA– TELECOMUNCIACIONES
P R E S E N T A :

PAUL DOMÍNGUEZ MÉNDEZ

TUTOR:
DR. ESAÚ VICENTE VIVAS

INSTITUTO DE INGENIERÍA UNAM

2011

JURADO ASIGNADO:

Presidente:	DR. MARTYNYUK OLEKSANDR
Secretario:	DR. GUTIÉRREZ CASTREJÓN RAMÓN
Vocal:	DR. VICENTE VIVAS ESAU
1er. Suplente:	DR. MARTÍNEZ LÓPEZ JOSÉ ISMAEL
2do. Suplente:	DRA. MOUMTADI FATIMA

Lugar donde se realizó la tesis:

INSTITUTO DE INGENIERÍA, UNAM.

TUTOR DE TESIS:

_____

DR. ESAÚ VICENTE VIVAS

Agradecimientos

A la Universidad Nacional Autónoma de México y sus profesores por la formación, conocimientos y experiencias que me han brindado.

A la Coordinación de Estudios de Posgrado (CEP) por su apoyo que hizo posible la realización de mis estudios de maestría.

Al Instituto de Ingeniería UNAM por su soporte en el desarrollo de esta tesis.

Al Dr. Esaú Vicente Vivas por permitirme formar parte de este proyecto y por su valioso apoyo y dirección.

A mis compañeros del Instituto de Ingeniería UNAM, especialmente a Mario Alberto Mendoza, Emilio Jiménez y Rodrigo Córdova, por sus consejos, apoyo, aportaciones y participación en este proyecto.

A los sinodales Dr. Oleksandr Martynyuk, Dr. Ramón Gutiérrez Castrejón, Dr. José Ismael Martínez López y Dra. Fatima Moumtadi por su ayuda en la revisión de esta tesis.

Índice de contenido

Índice de tablas

Índice de figuras

Capítulo 1
Introducción

1.1. Objetivo

Planear, diseñar e integrar un sistema de sensores de orientación y de navegación, en el cual se incorporará un sistema de posicionamiento global GPS para la navegación, así como de sensores de referencia y sensores inerciales para la orientación. Con el fin de contrarrestar los efectos adversos típicos de los sensores, especialmente de los giróscopos, se emplearán principios de conjunción de sensores y estimación de señales, como es el método de TRIAD en unión con el filtro de Kalman, para mejorar la calidad informativa de los sensores.

Para ello, se implantarán los algoritmos, que se han desarrollado previamente dentro del Instituto de Ingeniería, en sistemas de desarrollo para FPGA. Parte muy importante del desarrollo de la tesis comprende el diseño de diversos modelos de hardware, dentro del FPGA, con el objetivo de generar una solución óptima a la implementación de los algoritmos que estiman la orientación satelital en tres ejes. La validación de la arquitectura de procesamiento que se realiza en esta propuesta de tesis, para mejorar el comportamiento de señales de sensores de navegación inercial, se realiza comparando los datos crudos de los sensores con las salidas procesadas que dependerán entonces de los algoritmos de estimación y filtrado.

El presente desarrollo se aplica al satélite educativo SATEDU, que el Instituto de Ingeniería ha desarrollado en los últimos años para que se constituya en una herramienta de entrenamiento de recursos humanos en tecnología satelital. Respecto a SATEDU, el trabajo de esta tesis permitirá mejorar notablemente la operación de su modo de seguimiento en tiempo real. Con ello la presente tesis realiza una gran aportación a este sistema educativo de alta tecnología.

Por otro lado, se destaca que esta instrumentación también se desarrollará para que pueda trasladarse directamente a la instrumentación de navegación inercial del satélite Mexicano para constelación internacional Humsat, por lo cual las aportaciones adicionales de esta tesis van más allá de una simple aplicación para equipo terrestre. Por tales razones, la tesis aborda temas como la especificación de sensores para satélites pequeños reales.

1.2. Definición del problema

Cuando un satélite real se libera en el espacio se ve sometido a perturbaciones externas que lo obligan a estabilizarse con el fin de ejecutar ciertas tareas y operaciones que demandan requisitos muy estrictos de apuntamiento del satélite hacia Tierra. Lo anterior se puede alcanzar si el satélite dispone de medios eficientes para controlar su orientación, como son los actuadores y sensores de estabilización.

Los satélites que operan en órbita espacial requieren de sensores de navegación inercial con los cuales sea posible obtener datos de estabilización, orientación y dinámica de vuelo. En el caso de satélites pequeños estabilizados en tres ejes resulta fundamental incluir sensores que permitan conocer la orientación del satélite y que a su vez alimenten modelos matemáticos de estabilización y algoritmos de control de apuntamiento.

Por estas razones y considerando que el Instituto de Ingeniería participa en un proyecto internacional de constelación de satélites, junto con España y Estados Unidos como co-líderes del proyecto y otros países tanto europeos, asiáticos, africanos y latinoamericanos; para el cual se desarrollará un satélite con grandes capacidades de estabilización que permitirán capturar imágenes de nuestro planeta, así como el empleo de sistemas de comunicaciones directivos de amplio ancho de banda, en esta tesis se desarrolla el hardware inteligente de tratamiento de señales de sensores del subsistema de navegación inercial.

Adicionalmente y en vista de que el Instituto de Ingeniería ha desarrollado y terminado exitosamente el proyecto SATEDU, el cual se encuentra actualmente en registro de patente, en esta tesis se le desarrolla una mejora significativa al subsistema de sensores de navegación inercial, el cual ofrecerá de esta forma excelentes interfaces para entrenar tanto a estudiantes de este posgrado como a diversos grupos de recursos humanos interesados en el tema de desarrollo de tecnología satelital.

Capítulo 2
Antecedentes: Subsistema de sensores de navegación en SATEDU

Este capítulo tiene el objetivo de presentar el punto de partida del presente proyecto de tesis, mediante la descripción del trabajo que se realizó en el Instituto de Ingeniería con respecto a la determinación de la orientación del prototipo satelital SATEDU.

2.1. Arquitectura global del Proyecto SATEDU

SATEDU es un proyecto satelital educativo que consta de un prototipo satelital totalmente autónomo de 10x10x20 cm y de software que se ejecuta en una laptop, de tal forma que ambos dispositivos constituyen un sistema completamente portátil y amigable muy útil para dar demostraciones de operatividad de sistemas satelitales.

2.1.1. Subsistemas de SATEDU

El sistema SATEDU está formado por los siguientes subsistemas:

a) Un subsistema de potencia (SP) que cuenta con electrónica, procesador, baterías de Li+, reguladores de voltaje, convertidores de voltaje y medios de activación de subsistemas electrónicos;
b) Una computadora de vuelo (CV), con capacidad de subirle nuevos programas;
c) Un subsistema de comunicaciones inalámbricas (SC) con procesador dedicado y operando en la banda de 2.4 GHz;
d) Un subsistema de sensores de navegación (SSE) que tiene un procesador dedicado, giróscopos en 3 ejes, acelerómetro en 3 ejes y una brújula digital.
e) Una tarjeta con medios activos de estabilización ó subsistema de estabilización (SE) que incluye una rueda inercial y bobinas de torque magnético en 3 ejes.
f) Una laptop que hace las veces de Estación Terrestre del sistema, a la cual se le conecta vía puerto USB una tarjeta de comunicaciones inalámbricas que también forma parte del SC y que también opera en la banda de 2.4 GHz.

Todos los subsistemas de SATEDU operan bajo una arquitectura de sistemas distribuidos, en el que la CV opera como servidor ante las peticiones que realiza el software de Estación Terrena (ET) por medio de comandos. El software de ET permite definir, de esta forma, las funciones o tareas que debe realizar SATEDU.

SATEDU emula de forma prácticamente completa a un satélite real, no obstante que en su fabricación se emplean exclusivamente partes comerciales muy económicas, a diferencia de un satélite real que se ensambla con partes especializadas de uso militar o aeroespacial y por tanto de costo elevado. Es por ello que la modificación o incorporación de nuevos componentes como un receptor GPS y otros sensores al subsistema de navegación, siguen la misma política de utilizar componentes comerciales de bajo costo.

2.2. Arquitectura de la tarjeta de sensores de navegación.

Los sensores de navegación de SATEDU persiguen el objetivo de ofrecer datos sobre la posición y orientación del satélite educativo en referencia a una posición conocida.

2.2.1. Componentes de la tarjeta

La tarjeta de sensores pretende emular a una unidad de medición inercial UMI (IMU-Inertial Measurement Unit), similar a la que emplean satélites reales, aunque más modesta y menos precisa que las comerciales, puesto que emplea componentes de bajo costo y de no muy buena calidad, pero suficientemente adecuados para una herramienta didáctica como SATEDU. Una UMI mide su propia velocidad y dirección de movimiento usando una combinación de acelerómetros y giróscopos, la cual le permite a una computadora guía seguir sus movimientos con un proceso llamado “cálculo muerto“ que es el proceso que estima una posición actual con base en una posición previa calculada mediante un velocidad y tiempo conocidos.

La tarjeta de sensores (Figura 2.1) que se desarrolló para SATEDU está compuesta principalmente por:

- Tres giróscopos ADIS16100 de Analog Devices, de un eje, montados en un arreglo ortogonal.
- Un acelerómetro triaxial MA7260QT de Freescale Motorola.
- Una brújula electrónica CMPS03 desarrollada por Devantech como sensor de referencia para realizar correcciones de deriva.
- Un microcontrolador PIC18F2520 de Microchip como unidad comunicaciones.

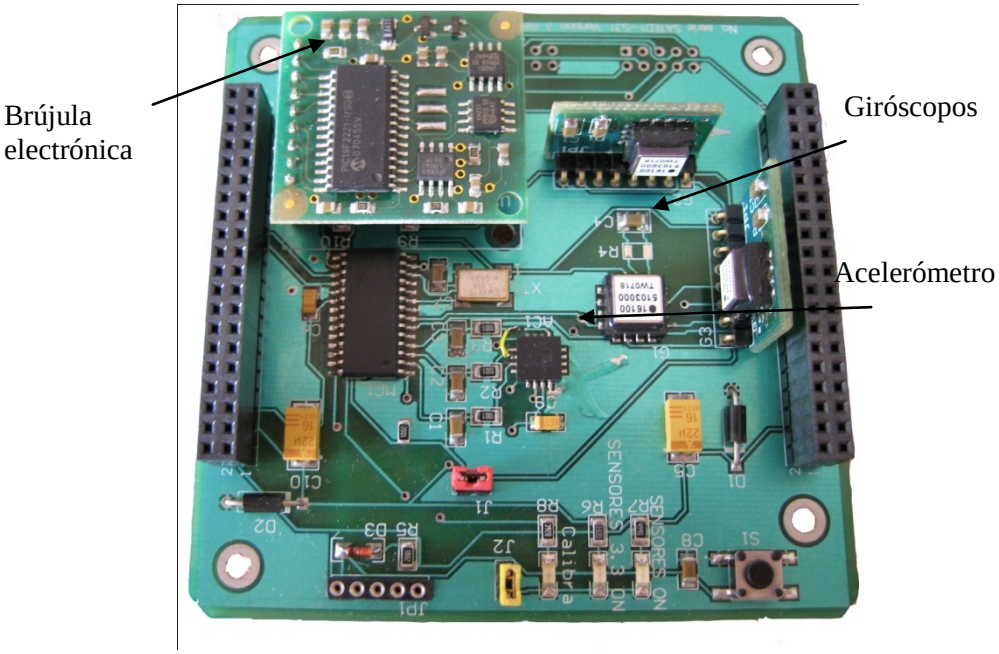


Figura 2.1 Tarjeta de sensores de navegación de SATEDU.

2.3. Funcionamiento del subsistema

El software de ET tiene medios para solicitar información al subsistema de sensores sobre las mediciones que éste realiza y de desplegar la telemetría recibida a través de su interfaz gráfica. Para el monitoreo de SATEDU en tiempo real, se envía un comando de ET al satélite que le indica al subsistema de sensores que debe enviar continuamente la lectura de sensores a la ET con una frecuencia de 10Hz.

Las lecturas de los sensores se colocan en una trama con un formato especial para su envío seguro. Esta tarea la realiza el microcontrolador de la tarjeta de sensores, en tanto que el software de ET se encarga de adaptar estos datos para su despliegue en las unidades correctas.

De forma independiente, las lecturas de los acelerómetros se utilizan para calcular el ángulo de cabeceo de la tarjeta. Las lecturas de los giróscopos se utilizan también de forma independiente a los otros sensores para el monitoreo en tiempo real a través de un sencillo proceso de integración numérica. Esto es, previo al desarrollo de la presente tesis, el subsistema de sensores de navegación en SATEDU no contaba con algún algoritmo o método que permitiera utilizar dicha información en conjunto para entregar mediciones más precisas y reducir los errores inherentes a la naturaleza de los sensores.

Capítulo 3
Sistemas de navegación inercial

Este capítulo describe el concepto de sistema de navegación inercial así como los elementos de los que está compuesto. También se describen las principales características, capacidades y limitaciones de dichos componentes. De igual forma, se presenta una propuesta de nuevos sensores comerciales y la selección de los mismos para sustituir a los sensores originales de SATEDU, los que más se adaptan a los requerimientos del proyecto, incluyendo aquellos que cuentan con calificación aeroespacial o experiencia de vuelo. Finalmente, se presenta la computadora de vuelo y se introducen los requerimientos de cómputo y procesamiento para la implementación de los algoritmos de estimación y filtrado, así como la solución propuesta para este problema.

3.1. Introducción

El funcionamiento de los sistemas de navegación inercial (INS por sus siglas en Inglés) se basa en la capacidad de medir la aceleración de un objeto, y posteriormente calcular su velocidad y su posición mediante una sucesión de integrales matemáticas con respecto al tiempo (1), (2) y (3). En el caso de los movimientos angulares, sucede algo muy parecido. La orientación o posición angular de un objeto puede ser conocida a partir de la medición y posterior integración con respecto al tiempo de la velocidad angular.

A diferencia de otros sistemas de navegación, los sistemas inerciales están completamente contenidos dentro del vehículo (1), en este sentido, son independientes de fuentes externas (3). Una vez que el sistema conoce la posición u orientación inicial mediante algún otro medio o referencia externa, las mediciones que se obtengan posteriormente podrán ser utilizadas para estimar la posición actual del vehículo, su recorrido y su orientación.

De lo anterior se puede inferir que un INS está compuesto principalmente por dos componentes elementales:

a) Una unidad de sensores de referencia.

b) Una unidad de medición inercial (IMU) que contiene al conjunto de sensores inerciales.

c) Una computadora de navegación para el procesamiento de las mediciones y para la estimación de la orientación y/o posición del vehículo.

3.2. Sensores de referencia

La referencia de orientación y posición de un vehículo puede obtenerse mediante la medición de la dirección o estimación de la posición relativa del vehículo con respecto a algún objeto, tal como alguna estrella, el Sol o la Tierra (4) a partir de la utilización de sensores de referencia tales como :

- Sensores de sol
- Sensores de horizonte de la Tierra
- Sensores de estrellas
- Magnetómetros
- Sistemas satelitales de navegación global (GNSS)

3.2.1. Sensores de sol

El sol puede ser una referencia de orientación para un satélite, debido a que su radio angular es relativamente pequeño (0.267° desde la Tierra), y se puede considerar una fuente puntual de luz para la mayoría de las aplicaciones (5) y (6). Estos sensores se basan en elementos fotosensibles que detectan la luz solar, como por ejemplo, elementos fotovoltaicos (7), foto-resistencias, fotodiodos y algún otro tipo de elemento fotoeléctrico. El procesamiento de estas señales bajo diferentes técnicas e implementaciones prácticas permite conocer la dirección relativa del sol al satélite.

Los sensores de sol pueden clasificarse como finos y gruesos, y también como analógicos y digitales, (6). Los sensores digitales son más precisos y más versátiles, pero también más costosos. Por otro lado, los sensores analógicos pueden ser suficientes para gran número de aplicaciones, (5).

La mayoría de los sensores analógicos utilizan celdas solares, las cuales producen una intensidad de corriente proporcional al coseno del ángulo de incidencia de los rayos solares, (5). También es común que los sensores de sol estén compuestos de arreglos de elementos foto-detectores o de sensores montados en varios ejes con el fin de determinar el vector de sol, el cual define la dirección hacia la cual se encuentra el astro, (8).

3.2.2. Magnetómetros

Pueden utilizarse para obtener una referencia inicial de baja precisión (3) de la orientación de un cuerpo a través de la medición del campo magnético de la Tierra. Combinando la magnitud y orientación de dicho campo, medido con respecto al cuerpo del satélite, con un modelo del

campo magnético Terrestre y con la ubicación del satélite, resulta posible determinar la orientación del satélite, (8).

La gran mayoría de los instrumentos de campo magnético para este tipo de aplicaciones miden tres componentes ortogonales del campo magnético local referenciado a un sistema coordenado inercial, debido a que proveen información de magnitud y dirección. Los magnetómetros de tipo vector más comúnmente utilizados en aplicaciones espaciales son sensores de núcleo saturado (fluxgate).

El magnetómetro debe tener la sensibilidad y resolución adecuadas para detectar la densidad de flujo magnético terrestre, el cual oscila alrededor de los 60µT (9). Sin embargo, para operaciones espaciales su uso está limitado a una altura de unos cuantos miles de kilómetros, debido a que la intensidad de campo magnético de la Tierra es inversamente proporcional al cubo de la distancia entre el centro de ésta y el satélite, (6).

Existe un compromiso entre la energía que consume el sensor y su estabilidad o susceptibilidad al ruido, principalmente en aplicaciones satelitales donde la energía es un recurso que debe cuidarse. Las posibles fuentes de ruido y de errores para estos sensores en un satélite pueden ser hasta cierto punto impredecibles, debido a que son causados por la operación de la electrónica dentro del satélite, (10). Otros factores son bastantes predecibles y pueden compensarse con una simple calibración, como los que producen elementos metálicos o magnéticos propios del satélite, los cuáles, en general se pueden evitar.

Otros parámetros a considerar son la estabilidad de la sensibilidad a los cambios de temperatura, el “offset” o desviación en la medición, y la susceptibilidad a la radiación. Los sensores más susceptibles a la radiación son los sensores de semiconductores, sin embargo, estos soportan mucho más altas dosis que otros dispositivos electrónicos de mayor escala de integración, como una memoria o un procesador, por ejemplo, (9).

El costo del sensor es usualmente un parámetro importante. Por ello, los sensores simples basados en efecto Hall pueden ser muy económicos, sin embargo su precio se incrementa rápidamente según su desempeño. Aquellos sensores de alto desempeño usualmente no se venden por separado, si no como parte de un magnetómetro. Este es el caso de sensores de resonancia y sensores de núcleo saturado (fluxgate) de alto desempeño, (9). Aunque este tipo de magnetómetro tiene el mejor desempeño con un bajo consumo de energía, los magnetómetros magneto-resistivos son más pequeños y por lo tanto preferidos, (11).

La frecuencia de muestreo típicamente requerida en sistemas espaciales está en el rango de una muestra por segundo hasta algunos cientos de muestras por segundo, dependiendo de los objetivos de la misión.

3.2.3. Sistemas satelitales de navegación global (GNSS)

El desempeño de los sistemas de navegación inercial puede ser mejorado, sin volverse muy costoso, incorporando un sistema de navegación global satelital, es decir, un GPS, a la unidad de navegación, (12), (3) y (13). La prueba de ello es que los errores de sensores inerciales que eran inaceptables en su operación independiente, han llegado a convertirse en aceptables cuando operan en conjunto con un GNSS.

El GPS provee continuamente información de posición y tiempo, (12), en cualquier lugar en donde sea posible recibir la señal proveniente de una constelación de satélites MEO (Órbita Terrestre media) que orbitan la tierra a 20,200 km de altura.

En la primera mitad de la década de los 80's fue probado el primer receptor experimental SGPS (Spaceborne GPS o GPS para aplicaciones espaciales) en las misiones Landsat 4 y Landsat 5, sin embargo, no fue sino hasta más de diez años después que los primeros satélites comerciales y militares comenzaron a utilizar receptores SGPS prototipo, incluyendo micro satélites como el PoSAT-1, (14). Inclusive, se realizaron pruebas para investigar su funcionamiento en la zona GEO, por encima de la constelación GPS, como el experimento Falcon Gold, (15).

Comparado con receptores GPS terrestres, la adaptación de los receptores GPS para sobrevivir en ambiente espacial caracterizado por altos extremos de temperaturas y alto vacío, vibraciones y radiación ionizante es un asunto clave, (16), por lo que dichos receptores requieren construirse con componentes electrónicos de calificación espacial, los cuales son comúnmente menos poderosos y requieren de mayores recursos (masa, energía) que sus contrapartes comerciales.

Además de las diferencias en hardware, un receptor GPS requiere de importantes modificaciones para operar a bordo de un satélite, debido principalmente a la dinámica orbital y las altas velocidades, (17). Entre las modificaciones están el cambio al rango de medición de la portadora, suavizar la portadora y compensación del efecto Doppler, entre otros. La complejidad técnica de estas modificaciones reside principalmente en el procesamiento digital de las señales GPS.

El éxito y bajo costo la tecnología GPS ha fomentado la percepción de que su aplicación es relativamente directa y económica. Sin embargo, no entender su complejidad y asumir que la aplicación "sencilla y económica" de tecnología GPS para aplicaciones terrestres resultará ser igualmente "sencilla y económica" en aplicaciones espaciales, conducirá a estimaciones irreales en cuanto a los recursos técnicos, económicos y de agenda, (18).

El reducido tamaño del mercado y la alta especialización de los receptores SGPS, así como los procesos de pruebas y calificación, resultan inevitablemente en unidades de alto costo que van desde los 100 k€ hasta 1 M€. Por ello, varias compañías e institutos de investigación han realizado esfuerzos para construir soluciones de bajo costo basadas en componentes COTS (comercial-off-the-shelf), (16).

Un componente COTS es un producto completamente desarrollado y disponible a la venta para el público en general. La utilización de estos componentes reduce el costo y tiempo de desarrollo, (19), pues los componentes no tienen que ser diseñados para aplicación satelital en específico, ni mucho menos desarrollados desde cero. El componente simplemente se modifica para la aplicación y para su integración con otros componentes.

Los receptores SGPS basados en componentes COTS han sido considerados y exitosamente probados en diversas misiones satelitales, (16).

3.3. IMU y sensores inerciales

Las mediciones del INS se efectúan mediante sensores inerciales como acelerómetros para la medición de la aceleración lineal y giróscopos, ya sea para medir la velocidad angular o bien, el ángulo de rotación mediante giróscopos de desplazamiento. Al ser estas mediciones cantidades vectoriales, los sensores de inercia tienen especificado un eje de entrada que determina la componente del vector que es capaz de medir, (3).

La cantidad de diseños de sensores de inercia es muy grande y pueden estar basados en una variedad de principios físicos o tecnologías diferentes. Algunos de ellos se presentan resumidos en la tabla 2.1.

Sensor	Giróscopos			Acelerómetros		
Efecto físico utilizado	Conservación del momento angular	Efecto Coriolis	Efecto Sagnac	Gyroscopic precession	Fuerza electromagnética	Tensión bajo una carga
Método de implementación	Desplazamiento angular	Vibración	Anillo de láser	Desplazamiento angular	Drag cup	Piezoeléctrico
	Reequilibrio del torque	Rotación	Fibra óptica	Reequilibrio del torque	Electromagnético	Piezoresistivo

Tabla 3.1 Distintos tipos de giróscopos y acelerómetros.

En aplicaciones que demandan bajo costo y tamaño reducido, los sensores basados en sistemas electromecánicos micro maquinados (MEMS) son la opción más viable para implementar un INS, dadas sus características y el desarrollo que estos dispositivos han tenido en los últimos años.

Sin embargo, la reducción en los elementos de sensado trae consigo problemas para alcanzar un buen desempeño de los propios sensores por lo que es necesario aplicar técnicas complementarias con el fin de obtener mediciones más confiables, (2).

3.3.1. Errores en los sensores de inercia

La integración de la aceleración medida causa errores en la estimación de la velocidad, los cuales crecen linealmente con el tiempo y en consecuencia, así lo harán los errores de la posición. Estos errores generalmente son desconocidos, excepto por sus propiedades estadísticas, (3).

Entre los tipos de error más comunes que se presentan en un sensor, (1), (3) y (20), se encuentran los siguientes:

(a) Sesgo por descentramiento (bias), es una salida del sensor diferente de cero cuando la entrada al sensor es cero, es decir, cuando la salida debería de ser cero

(b) Error de factor de escala, es generalmente el resultado del uso del sensor (envejecimiento) o de las tolerancias de manufactura

(c) No linealidades, las cuales están presentes en todos los sensores hasta cierto grado.

(d) Asimetría de signo del factor de escala

(e) Zona muerta, cuando el sensor no presenta una salida con respecto a una entrada.

(f) Error de cuantización, inherente a los sistemas digitales

3.3.2. MEMS para pico satélites

A pesar de su tamaño, masa reducida, bajo consumo de energía y bajo costo, los sensores MEMS comerciales que actualmente se usan en aplicaciones automotrices, aún requieren de importantes adaptaciones para emplearse en aplicaciones espaciales, (21).

La utilización de estos sistemas en el espacio ha sido limitada principalmente por asuntos de confiabilidad y calificación, (22) ante factores que deben resistir estos dispositivos, como son las vibraciones y choques mecánicos durante el despegue (10 Hz-2000 Hz y 3g-10,000g), los

cambios de presión debido al ruido acústico (20 Hz-10 KHz) que pueden causar fatiga mecánica y fracturas, y también son importantes la radiación electromagnética y los ciclos térmicos, (23) y (24).

Sin embargo, para los proyectos pico-satelitales y nano-satelitales éste no ha sido un obstáculo. Debido a su pequeño tamaño y bajo presupuesto. Estos satélites comúnmente evitan utilizar componentes de calificación espacial, y en cambio, recurren a MEMS comerciales y ampliamente disponibles, lo que se ha convertido en una forma efectiva de calificar la confiabilidad de estos sensores en el espacio, (24).

Además, la radiación generalmente no es un asunto de gran preocupación en los nano-satélites debido a su operación en órbitas bajas (LEO) y porque su tiempo de vida está limitado por la degradación de sus baterías recargables, más que por la radiación que recibirán los componentes, de 13 a 15 krad por año, (24).

Un buen ejemplo del uso de sensores MEMS comerciales es el proyecto Danés AAUSAT-II, lanzado en el año 2008, que lleva a bordo 6 giróscopos modelo ADXRS401 de la compañía Analog Devices y que operaron tal como se esperaba una vez en órbita, (24).

Es por estas razones que los sensores inerciales de tecnología MEMS serán utilizados también en nuestra aplicación.

3.3.3. Acelerómetros

La mayoría de los acelerómetros MEMS detectan la aceleración ya sea mediante el desplazamiento de una masa de prueba o mediante el cambio de frecuencia de un elemento vibratorio, ambos causados por un cambio en la tensión mecánica sobre dicho elemento que resulta de la aceleración, (1), (2) y (3).

La forma en la que se mide el desplazamiento de la masa de prueba involucra generalmente la evaluación de los cambios en la capacitancia que se generan al acercar o alejar estructuras de material semiconductor de los que está compuesto el acelerómetro. La implementación del segundo tipo de acelerómetros implica la utilización de algún tipo de material piezoeléctrico.

A comparación de los acelerómetros piezoeléctricos, los acelerómetros capacitivos presentan un amplio rango de salida, son estables y consumen poca energía, además, muestran una mejor sensibilidad y son menos dependientes a los cambios de temperatura. Su principal desventaja es su susceptibilidad a las fuentes de interferencia electromagnética, (25) y (26). Por su parte, los acelerómetros piezoeléctricos presentan una buena linealidad e inmunidad ante la interferencia electromagnética, (26).

3.3.4. Giróscopos

Los giróscopos MEMS están basados en la detección de la fuerza que actúa sobre una masa que está sujeta a un movimiento linear vibratorio en un sistema de referencia que está rotando. Existen una gran variedad configuraciones prácticas para lograr esta detección, cada una con sus propias características y grado de desempeño, como las que se resumen en la siguiente tabla.

Los requerimientos de estabilidad en vehículos espaciales se encuentran en el rango de 0.001 a 0.01 °/h (27) y una sensibilidad de 0.001 a 0.01°/s (28), es decir, requieren de un cierto nivel o grado de desempeño. Los dispositivos comerciales disponibles actualmente, los cuales son más que adecuados para aplicaciones automotrices, están algo lejos de este grado desempeño. Sin embargo, gracias al continuo mejoramiento en las técnicas de fabricación, el uso de los giróscopos MEMS en satélites es prometedor (21).

Parámetro de desempeño	Unidades	Grado de desempeño		
		Inercial	Intermedio	Moderado
Rango máximo de entrada	°/h °/s	10^{2}-10^{6} 10^{-2}-10^{2}	10^{2}-10^{6} 10^{-2}-10^{2}	10^{2}-10^{6} 10^{-2}-10^{2}
Factor de escala	Partes/partes	10^{-6}-10^{-4}	10^{-4}-10^{-3}	10^{-3}-10^{-2}
Estabilidad en el descentramiento (bias stability)	°/h °/s	10^{-4}-10-^{2} 10^{-8}-10-^{6}	10^{-2}-10 10^{-6}-10^{-3}	10-10^{2} 10^{-3}-10^{2}
Deriva	°/√h °/√s	10^{-4}-10^{-3} 10^{-6}-10^{-5}	10^{-2}-10^{-1} 10^{-5}-10^{-4}	1-10 10^{-4}-10^{3}

Tabla 3.2 Grados de desempeño de diferentes giróscopos.

3.4. Selección de los sensores de navegación inercial y GPS

Con el fin de seleccionar los sensores más adecuados para nuestra aplicación se realizó una investigación sobre los sensores, sus características y limitaciones, así como los sensores disponibles en el mercado que cumplan los requerimientos y con base principalmente en la búsqueda de sensores que hayan sido utilizados y probados en misiones satelitales.

Con base en los resultados más substanciales de esta investigación y en conjunto con la viabilidad económica y costo de los sensores, se llevó a cabo la selección de los sensores que finalmente podrán ser utilizados.

3.4.1. Magnetómetro

a) Honeywell Serie HMC

Honeywell produce una amplia variedad de magnetómetros basados en sensores magneto-resistivos que son adecuados para diversas aplicaciones. En este apartado, nos enfocaremos en el circuito integrado HMC2003 Magnetic Hybrid, del cual se desprenden otros dos dispositivos que tienen aplicaciones aeroespaciales y calificación militar, de ahí su importancia de mencionarlo. Y el magnetómetro digital HMC5843 diseñado para aplicaciones de navegación de bajo costo.

El circuito integrado HMC2003 Magnetic Hybrid es un magnetómetro analógico en tres ejes con un tamaño de 25.4mm x 19 mm y un consumo máximo de 0.2 W. Tiene un rango de medición de +/- 2 Gauss con una resolución de hasta 40uG. Está equipado con la posibilidad de establecer un “offset” en cada uno de sus ejes, (29). Su costo es de $220.00 USD, figura 3.1.

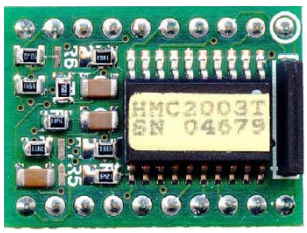
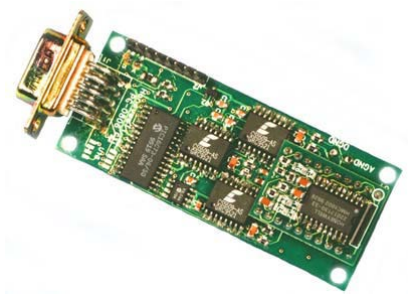
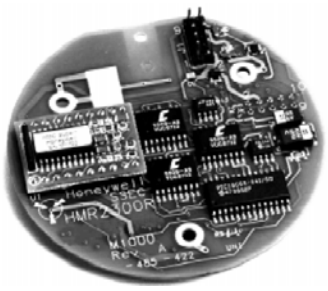


Figura 3.1 Magnetómetro HMC2003.



HMR2300 Smart Serial



HMR2300r Three-Axis Strapdown

Figura 3.2 Magnetómetros basados en el circuito HMC2003.

El magnetómetro HMR2300 Smart Digital es una tarjeta basada en el sensor anterior con un tamaño de 74.3mm x 30.5 mm y un peso de 28g, figura 3.2.a, que brinda una interfaz de comunicación serial RS2322 a través de un conector de 9 pines y elimina la necesidad de circuitería adicional necesaria para el correcto funcionamiento del sensor, (30). Su costo en catálogo asciende a los $725.00 USD.

El magnetómetro HMR2300r Three-Axis Strapdown es un dispositivo de calificación aeroespacial y militar (bajo el estándar MIL-STD-810E) diseñado especialmente para sistemas de vuelos comerciales y para aplicaciones en vehículos satelitales, figura 3.2.b. Tiene características mejoradas, especialmente en cuanto el rechazo al ruido. Un microcontrolador interno maneja el sensado, el filtrado digital, y las comunicaciones bajo el estándar RS-422 o RS-485 con datos en formato BCD ASCII, eliminando la necesidad de calibración externa y ajustes adicionales, (31). Su diámetro es de 71mm y su costo de lista es de $895.00 USD.

Finalmente, el circuito HMC5843, figura 3.3, es un magnetómetro de 3 ejes con interfaz digital I2C. Su tamaño reducido (4.0mm x 4.0mm x 1.3mm), bajo costo ($50 USD) y su amplio rango de medición y sensibilidad (10 µG -6 G) lo hacen ideal para un gran número de aplicaciones, (32).

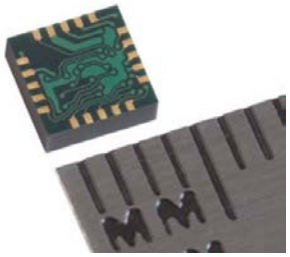


Figura 3.3 Magnetómetro digital HMC5843.

b) PNI Coorporation MicroMag3

Es un módulo digital de sensores magnéticos en 3 ejes diseñado para asistir en la evaluación y desarrollo de prototipos. El módulo está basado un sensor magneto-inductivo (uno por cada eje) formado por un circuito oscilador LR, el cual cambia su inductancia cuando se aplica un cambio en el campo magnético.

El esquema de medición diferencial con el que cuenta lo hace independiente de la temperatura y estable dentro de un amplio rango de trabajo. Tiene una alta resolución en el sensado de la densidad de flujo magnético (0.015uT) dentro de un rango de ±1100 T. Cuenta con una amplia inmunidad al ruido y está libre de offset por deriva. Su frecuencia de muestro alcanza las 2000 muestras/segundo. Es utilizado en aplicaciones aeronáuticas y en satélites pequeños (CUBESAT

UWE-2) por su bajo costo, bajo consumo de energía y tamaño reducido (25.4 x 25.4 x 19 mm). Su interfaz de comunicaciones es SPI (Serial Pheriphal Interface), pero también se le puede añadir un módulo adicional de comunicaciones RS-232 o RS-485 (33). Su precio es de $200 USD.

c) Elección del magnetómetro

Debido a la gran variedad de sensores disponibles a nivel comercial, nuestra búsqueda se concentró en magnetómetros de 3 ejes de bajo costo que hayan sido utilizados y probados en satélites pequeños, lo cual redujo las opciones a sólo unos cuantos.

Aunque con grandes ventajas, los modelos HMR2003 y HMR2003r fueron descartados por su costo y por su tamaño. Al igual que el módulo MicroMag3, el modelo HMC2003 ha sido probado a bordo de diversos satélites entre los que están la formación ION-F (Ionospheric Observation Nanosatellite Formation) y el CANX-1 (34). Aunque este último modelo podría brindar mayor flexibilidad en el diseño debido a que es analógico, se prefirió utilizar el módulo HMC5843, pues esta decisión implica la simplificación del diseño y la reducción de costos al no tener que desarrollar un módulo de comunicaciones digitales ni tener que añadir elementos adicionales al magnetómetro analógico.

3.4.2. Acelerómetros

Se tomaron en consideración 3 diferentes sensores de distintas compañías. Todos ellos son acelerómetros MEMS digitales de 3 ejes integrados en un solo chip con empaquetado de apenas unos cuantos milímetros. Las aplicaciones de estos sensores se encuentran principalmente en dispositivos móviles, automóviles, detectores de vibración e instrumentación industrial y médica.

MMA8451Q Freescale Semiconductor: es un acelerómetro capacitivo con resolución de 14 bits. Tiene funciones programables que permiten establecer esquemas de ahorro de energía. Consume entre 6uA y 165uA. Su rango de medición es seleccionable en escalas de ±2g, ±4g y ±8g. Su nivel de offset o descentramiento en cero-g es de ±20mg. Su sensibilidad es de 1g/1024 hasta 1g/4096 por dígito (según la escala seleccionada). El protocolo de comunicaciones que utiliza es I^2C.

ADXL345 Analog Devices: se trata de un sensor digital de 13 bits con escala de medición de hasta ±16g. Su sensibilidad alcanza 1g/256. Incluye funciones especiales como detección de movilidad cero. Su consumo de energía puede ir desde los 0.1uA hasta los 140uA. Su nivel de offset o descentramiento en cero-g es de ±35mg Es capaz de utilizar dos protocolos de comunicaciones: SPI o I^2C. Tiene una resistencia de choque de hasta 10,000g (35).

LIS331HH ST Microelectronics: cuenta con una resolución de 16 bits y alcanza un rango de medición de hasta ±24g. Su sensibilidad va desde los 1g/83 hasta 1g/330. Su nivel de offset o descentramiento en cero-g es de ±70mg. Al igual que el sensor anterior, es capaz de utilizar dos protocolos de comunicaciones: SPI o I^2C y de igual forma tiene una resistencia de choque de hasta 10,000g. Su consumo de energía puede ir desde los 1ua hasta los 250uA, según el modo de operación en que se encuentre.

3.4.3. Giróscopos

En el mercado existe una gran variedad de giróscopos MEMS para un rango muy amplio de aplicaciones. La especialización, calificación y características de estos dispositivos es igualmente variada y su costo depende en gran medida de ello, aunque la mayor parte de estos dispositivos están dirigidos hacia el mercado automotriz y de dispositivos móviles. A continuación, se presentan algunos de los sensores que, de acuerdo a sus características, podrían ser los más adecuados para la presente aplicación. Todos ellos son giróscopos en 3 ejes encapsulados en un solo chip. La tabla 3.2 resume las principales características de los dispositivos que fueron considerados.

Aunque de la comparativa se concluyó que el giróscopo ADXRS450 de la compañía Analog Devices podría ser la mejor opción de acuerdo a sus características, hubo un factor adicional que condujo a la decisión por seleccionar al dispositivo ITG_3200 de la compañía InvenSense, la integración de este último sensor, con el acelerómetro y el magnetómetro seleccionados en una sólo unidad, comúnmente conocida como MARG, por las siglas en inglés de los parámetros físicos que puede medir (Magnetic, Angular Rate and Gravity), figura 3.4.

El proveedor de componentes electrónicos Sparkfun desarrollo una tarjeta que integra a los tres sensores sobre un bus de comunicaciones I^2C. Esta solución beneficia al proyecto en muchos sentidos: el costo de la tarjeta es menor que el que se tendría si se adquirieran los sensores por separado, se resuelve la problemática del soldado de componentes de montaje superficial de tamaños muy reducidos, se reducen los tiempos de diseño y fabricación así como los costos y tamaño de la tarjeta que albergará al subsistema de navegación.

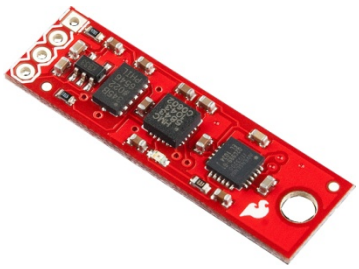


Figura 3.4 Unidad MARG.

Fabricante	Dispositivo	Costo (USD)	Aplicaciones	Salida	Características especiales	Vibración	Bias	Sensibilidad	Rango a escala completa
ST	L3G4200D	$13.00	Control de movimiento (interfaz hombre-máquina), sistemas de navegación, robótica.	Digital (SPI o I2C)	Mide temperatura (8 bits)	Alta resistencia a choques (no se proporciona una medida)	±10 °/s ±15 °/s ±75 °/s	8.75 mdps/dígito	250 °/s 500 °/s 2000 °/s
Analog Devices	ADXRS450	$50.00	Equipo médico, instrumentación industrial, estabilización de plataformas con alto desempeño	SPI	Tiene un sistema automático que revisa la estructura interna del MEM	2000g	±3°/s	12.5 mdps/dígito	±300°/s
Sensor Dynamics	SD 740	$60.00	Sistemas de navegación, estabilización, control de movimiento.	Analógica Digital (SPI o I2C)	Unidad DSP dedicada Memoria no volátil	2000 g	±10°/s	%5 (precisión)	1024 °/s
InvenSense	ITG-3200	$50.00	Equipo médico, interfaces hombre máquina.	I2C	Filtro digital programable		±40°/s	14.375 mdps/dígito	2000 °/s

Tabla 3.3 Resumen de las características de los giróscopos, (36), (37), (38) y (39).

Como complemento, es muy importante para nuestros objetivos destacar el modelo SiRSS01 diseñado por la compañía BAE Systems, al cual la ESA (Agencia espacial europea) ha seleccionado para incluirlo en sus programas TRP (Programa de Investigación Tecnológica) y GSTP (Programa de Soporte General de Tecnología), (40). Gracias a sus excelentes características de estabilidad (3°/hr), descentramiento (0.3°/s), deriva (0.2°/√hr) y el hecho de que ha sido analizado como un factible candidato a ser utilizado en órbita, (41), podrá ser considerado para futuras versiones del presente proyecto y otros desarrollos, figura 3.5.

Figura 3.5 Giróscopo SiRSS01 diseñado por BAE Systems.

3.4.4. Receptor GPS

La selección del GPS difiere a la de los sensores anteriores pues el receptor para el simulador (satélite educativo) y el receptor para el satélite real requieren cumplir con distintas características y su diferencia de costo es enorme.

La elección del receptor GPS para el satélite educativo se basó principalmente en la disponibilidad y el costo de un de un receptor completo integrado en un solo módulo o chip, que no requiriera de mayores adaptaciones para ser funcional.

El dispositivo GPS9548SLP de la compañía Leadtek, figura 3.6, es un módulo GPS compacto (24 x 20 x 2.9 mm) diseñado para aplicaciones de navegación terrestre y marítima, así como para dispositivos móviles. Es capaz de entregar toda la información típica de un GPS bajo el protocolo NMEA-0183 ó SiRF Binario a través de una interfaz serial con niveles de voltaje TTL a diferentes velocidades (4600bps por defecto). Su precisión en posición es de 10m, en velocidad 0.1m/s, y de tiempo de 1 us. Está basado en el chipset SiRF Star III, el mismo que utilizan los receptores comerciales GPS más avanzados. Su costo es de $73 USD.

Debido al ambiente espacial y la dinámica orbital, no es posible utilizar este mismo dispositivo, como tal, en aplicaciones satelitales. La adaptación o desarrollo de un GPS con las características necesarias para este propósito está fuera del alcance de esta tesis. Por ello, se realizó una búsqueda de receptores que en algún momento podrían ser utilizados en el proyecto satelital.

Como se indicó en secciones anteriores, un receptor especializado para aplicaciones espaciales puede ser muy costoso. Sin embargo, existen versiones relativamente más económicas, basadas en elementos comerciales, que han pasado por un proceso de modificaciones, pruebas y calificaciones, y que han sido utilizados a bordo de gran número de misiones satélites con éxito.

Figura 3.6 Módulo GPS GPS9548SLP de la compañía Leadtek.

a) Phoenix Reciever

Es un receptor GPS desarrollado por DLR (Agencia Espacial Alemana) basado en el chipset GP4020 de la compañía Zarlink y el microprocesador de 32 bits ARM7TDMI. El correlacionador y otros aspectos del firmware han sido modificados por DLR para brindar medidas precisas de posición, dirección y tiempo bajo condiciones dinámicas extremas, (42), (43), (44), (45) y (46). El receptor Phoenix es la evolución del receptor Orión, el primer receptor GPS alemán en órbita, usado en la misión PCSat, (47).

El receptor Phoenix constituye el sistema de navegación primario para la misión de cohetes sonda del programa Europeo de micro gravedad dentro del programa espacial nacional alemán, (48), figura 3.7.

Figura 3.7 Receptor GPS Phoenix con calificación espacial.

b) **SGR-05**

Es un receptor GPS espacial construido por la compañía Surrey Satellite Technology especialmente para satélites LEO, figura 3.8. Con un tamaño muy compacto (70x45x10mm), puede proveer una precisión de 10m en la posición y de 15cm/s en la velocidad. Es tolerante a la radiación y ambiente espacial. Ha sido probado en dos misiones satelitales con éxito. Al igual que el receptor Phoenix, el SGR-05 está basado en el chipset GP4020 de Zarlink y un microprocesador ARM. Utiliza una interfaz de comunicaciones UART para enviar datos en formato SBPP. Su costo es de £10,000 libras, su disponibilidad no es inmediata, requiere de varios meses y de autorización del gobierno británico.

Figura 3.8 Receptor GPS SGR-05 con calificación espacial.

3.5. Computadora de navegación

El objetivo de la computadora de navegación es obtener las mediciones realizadas por los sensores y procesar dicha información con el fin de determinar la orientación y posición del satélite.

La posición del satélite puede obtenerse directamente de la información generada por el receptor GPS, sin embargo, la determinación de la orientación sigue un proceso conceptualmente simple, pero matemáticamente muy complejo, (49).

Este proceso consiste en la estimación de la orientación del satélite con respecto a un marco de referencia, generalmente inercial o geocéntrico, expresado como un conjunto de ángulos de Euler, una matriz de cosenos directores (DCM) o como un cuaternión, en un esquema más o menos continuo, (49) y (50), figura 3.9. El proceso se puede resumir en los siguientes pasos:

i) Adquisición: se reúne la información de los sensores.
ii) Estimación: los datos obtenidos de sensores son corregidos (pues contienen errores) y son analizados mediante algoritmos matemáticos para hacer una estimación de la orientación (o de los estados).

iii) Expresión de la orientación: a partir de de la información anterior, se expresa la localización de los ejes del sistema de referencia del cuerpo (satélite) con respecto al marco de referencia.

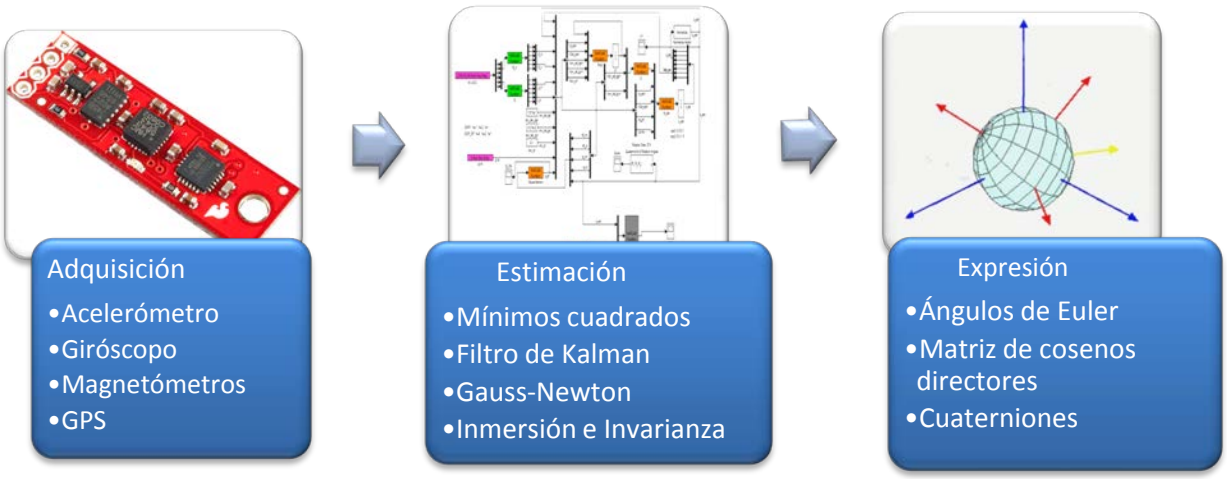


Figura 3.9 Diagrama de bloques de las funciones de la computadora de navegación.

Los sensores inerciales, por sí mismos presentan una serie de errores inherentes ya mencionados anteriormente. Este problema se vuelve mayor cuando se utilizan de sensores de bajo costo y de tecnología MEMS, por lo que es necesario aplicar técnicas complementarias con el fin de obtener mediciones más confiables (2).

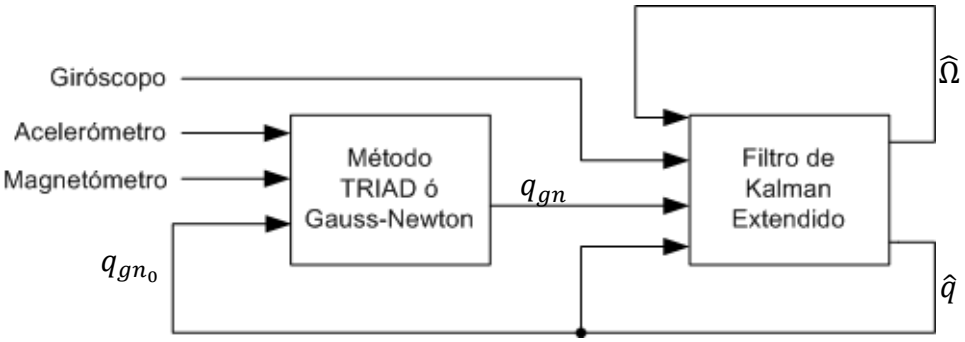
Algunas de estas técnicas, como la que se describe en la siguiente sección, están orientadas a utilizar información redundante de varios sensores, y de cierta forma, fusionar esta información en una sola medida más confiable (51). En el caso de los sensores inerciales por ejemplo, algunos de los errores inherentes a los giróscopos pueden ser compensados por los acelerómetros y viceversa a través de estas técnicas.

3.5.1. Estimación de la orientación

Existen distintos métodos de estimación, algunos pueden ser métodos determinísticos, los cuales utilizan la menor cantidad de datos necesaria para determinar la orientación, pero no necesariamente de manera precisa y otros pueden ser métodos óptimos que utilizan el menor número de mediciones para encontrar la orientación. Estos últimos pueden ser secuenciales o recursivos, lo cual implica obtener una nueva estimación justo después de cada observación, (8).

El método utilizado para resolver el problema de estimación de la orientación fue el propuesto en (52), un trabajo realizado dentro del Instituto de Ingeniería donde se plantea una conjunción del método (recursivo) Gauss-Newton con el Filtro de Kalman Extendido (EDK) Discreto (método óptimo) para el control y estimación del nanosatélite HUMSAT-México, figura 3.10.

En adición al método Gauss-Newton, en el mismo trabajo se plantea el método TRIAD como una alternativa a éste último, pero con requerimientos de cómputo menores, además de la versión continua del EKF. Como resultado de aquella investigación, se generó también un algoritmo simplificado del Filtro de Kalman Extendido Continuo con el propósito específico de poder validar el esquema de estimación y control propuesto en el trabajo mencionado, físicamente mediante un experimento en una mesa suspendida en aire y que es justamente el algoritmo a implementar en el presente trabajo.



Donde:

q_{gn_0} es la condición inicial
$\hat{q}$ es el cuaternión estimado
$\widehat{\Omega}$ es la velocidad angular estimada
q_{gn} es el cuaternión de medición

Figura 3.10 Esquema del algoritmo de estimación y filtrado.

3.5.2. Descripción de los algoritmos de estimación y filtrado

El algoritmo TRIAD requiere básicamente de funciones que calculen vectores unitarios y operaciones producto cruz de vectores columnas de tres elementos, a comparación del método Gauss-Newton, un procedimiento iterativo más complejo que primeramente requiere calcular la matriz de medición y un Jacobiano formado por las derivadas parciales de dicha matriz con respecto a los elementos del cuaternión, para después utilizar dichas matrices en una operación que requiere el cálculo de una matriz pseudo-inversa, todo ello en cada iteración. Por otro lado,

el Filtro de Kalman Discreto requiere de un gran número de operaciones matriciales, principalmente multiplicaciones y cálculo de matrices inversas.

a) Método TRIAD

El método TRIAD consiste en encontrar una matriz de transformación que relaciona dos sistemas coordenados usando dos vectores de medición y dos vectores de referencia, expresados en el sistema coordenado en el que se efectuó la medición y en el de referencia respectivamente, (53) (54).

En primer lugar se calcula una serie de vectores ortogonales entre sí r_1, r_2 y r_3, en el sistema coordenado donde se efectuó la medición tales que:

$$r_1 = \frac{u_M}{\|u_M\|} \qquad \text{Ecuación 3-1}$$

$$r_2 = \frac{r_1 \times v_M}{\|r_1 \times v_M\|} \qquad \text{Ecuación 3-2}$$

$$r_3 = r_1 \times r_2 \qquad \text{Ecuación 3-3}$$

Donde u_M y v_M son vectores de medición adquiridos por medio de dos sensores distintos.

Se calcula también una serie de vectores ortogonales entre sí s_1, s_2 y s_3, en el sistema coordenado de referencia tales que:

$$s_1 = \frac{u_R}{\|u_R\|} \qquad \text{Ecuación 3-4}$$

$$s_2 = \frac{s_1 \times v_R}{\|s_1 \times v_R\|} \qquad \text{Ecuación 3-5}$$

$$s_3 = s_1 \times s_2 \qquad \text{Ecuación 3-6}$$

Donde u_R y v_R son vectores de referencia adquiridos por medio de dos sensores distintos.

Posteriormente la matriz de transformación que relaciona ambos sistemas coordenados se calcula con :

$$A_m^r = r_1 \cdot {s_1}^T + r_2 \cdot {s_2}^T + r_3 \cdot {s_3}^T \qquad \text{Ecuación 3-7}$$

Finalmente, es posible convertir la matriz de transformación en una entidad matemática más práctica para fines de cálculo, como lo es el cuaternión. Para ello se calcula la suma de los elementos de la diagonal principal, a la cual llamaremos $trace$ y el vector v definido como sigue:

$$v(1) = A(2,3) - A(3,2)$$

$$v(2) = A(3,1) - A(1,3)$$

$$v(3) = A(1,2) - A(2,1)$$

Ecuación 3-8

A partir de la suma de la diagonal principal de la matriz de transformación se calculan los siguientes elementos:

$$cb = \frac{trace-1}{2} \qquad \text{Ecuación 3-9}$$

$$\theta = \text{acos}(cb) \qquad \text{Ecuación 3-10}$$

$$\eta = \cos\left(\frac{\theta}{2}\right) \qquad \text{Ecuación 3-11}$$

$$\zeta = \sin\left(\frac{\theta}{2}\right) \qquad \text{Ecuación 3-12}$$

El elemento η constituye la componente escalar del cuaternión. El vector v se normaliza para formar un vector unitario λ, el cual se utiliza para calcular los elementos restantes del cuaternión de medición $q_{gn} = (\eta\ \epsilon_1\ \epsilon_2\ \ \epsilon_3)$, mediante las siguientes expresiones:

$$\epsilon_1 = \zeta\lambda_1 \qquad \text{Ecuación 3-13}$$

$$\epsilon_2 = \zeta\lambda_2 \qquad \text{Ecuación 3-14}$$

$$\epsilon_3 = \zeta\lambda_3 \qquad \text{Ecuación 3-15}$$

b) Método Gauss-Newton

El método Gauss-Newton se basa en un criterio de mínimos cuadrados del error. A diferencia del método TRIAD, se pueden integrar más de dos vectores de medición con sus respectivos vectores de referencia, (55).

c) Filtro de Kalman

El Filtro de Kalman es un estimador que permite determinar el estado instantáneo de un sistema dinámico perturbado por ruido blanco, mediante mediciones linealmente relacionadas con el estado pero afectadas por ruido. En el control de sistemas dinámicos, no siempre es posible obtener o medir cada variable que se quiere controlar, el Filtro de Kalman proporciona los medios para inferir la información faltante a partir de mediciones indirectas y ruidosas, (56).

Estas propiedades hacen del Filtro de Kalman una solución idónea para el problema de estimación de orientación.

Considere el siguiente sistema no lineal:

$$\dot{x} = f(x,u,t) + \omega(t) \qquad \text{Ecuación 3-16}$$

$$y = h(x,t) + v(t) \qquad \text{Ecuación 3-17}$$

Donde $v(t)$ se asume que es una función de ruido blanco de las mediciones, $\omega(t)$ es una función de ruido blanco del proceso del sistema, x es el vector que contiene las variables del sistema de espacio de estados, u es el vector de comandos de control, y es el proceso de medición, $f(x,u,t)$ y $h(x,t)$ son funciones no lineales.

En el Filtro de Kalman Extendido Continuo, el proceso de estimación está dado por:

$$\dot{\hat{x}} = F(\hat{x},u,t) + Kv \qquad \text{Ecuación 3-18}$$

Donde v es el proceso de innovación definido por:

$$v = y - H(\hat{x}) \qquad \text{Ecuación 3-19}$$

Y K es la ganancia de Kalman dada por:

$$K = PH^T R^{-1} \qquad \text{Ecuación 3-20}$$

Siendo R la matriz de covarianza de las mediciones, Q la matriz de covarianzas del proceso P la ecuación de error de covarianza:

$$\dot{P} = FP + PF^T - PH^T R^{-1} HP + Q \qquad \text{Ecuación 3-21}$$

Para la primera iteración, se debe de establecer una condición inicial para el vector de estados $\hat{x} = \hat{x}_0$ y su matriz de error de covarianza $P = P_0$.

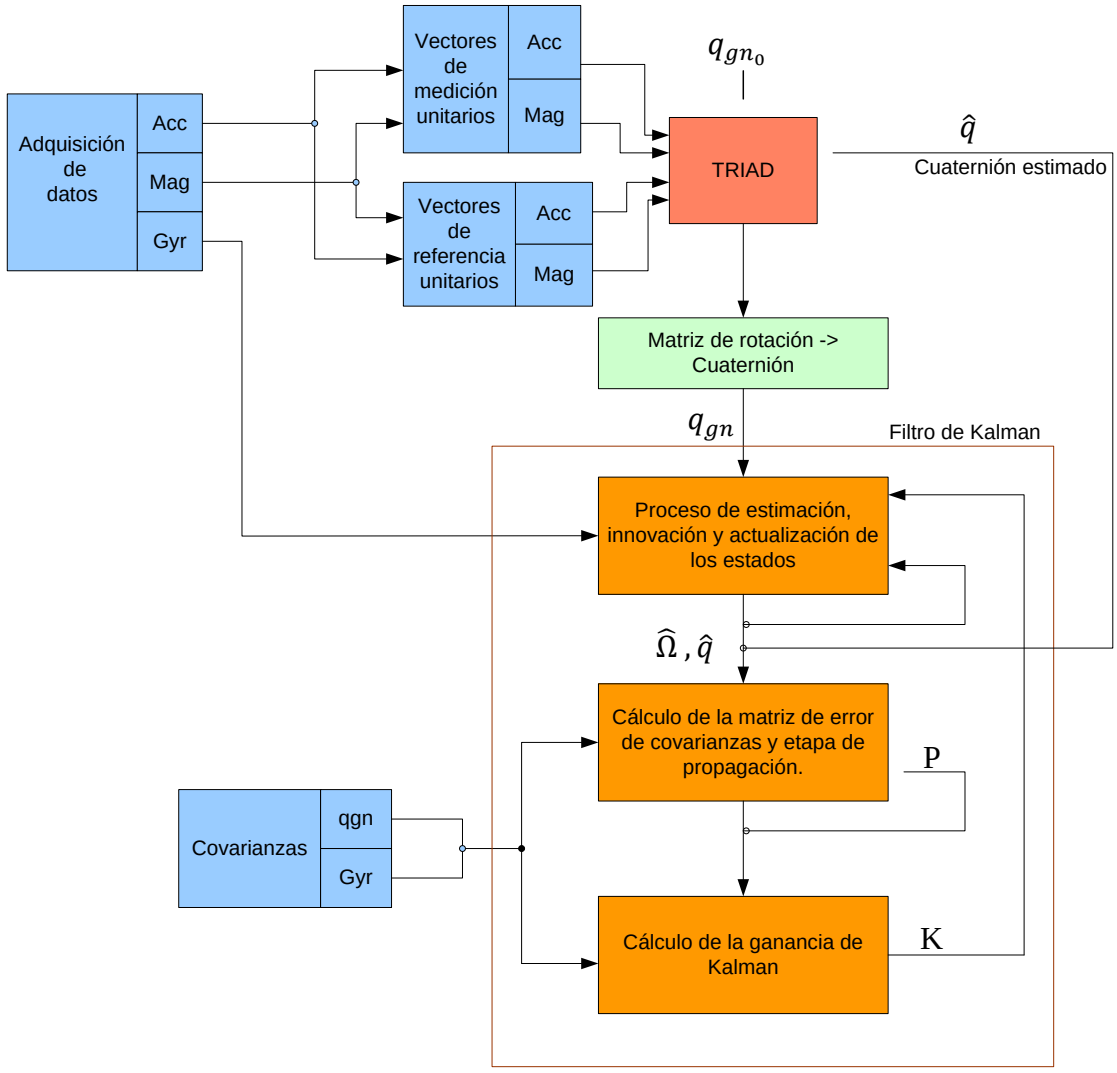
Finalmente, $F(\hat{x},u,t)$ y $H(\hat{x},t)$vienen del proceso de linealizar a lo largo de las trayectorias de las variables estimadas $\hat{x}$:

$$F = \left.\frac{\partial f(x,u,t)}{\partial x}\right|_{x=\hat{x}} \qquad H = \left.\frac{\partial h(x,t)}{\partial x}\right|_{x=\hat{x}} \qquad \text{Ecuación 3-22}$$

En (52) se puede encontrar una descripción matemática más detallada y completa de este procedimiento. Como resultado de aquella investigación, se generó un modelo en Simulink con una versión simplificada del algoritmo que aquí se presenta y que tiene como propósito evaluar

el desempeño de los algoritmos de estimación, filtrado y estabilización en un experimento sobre una mesa suspendida en aire (MSA).

Partiendo de dicho modelo se evaluó, analizó y planificó la implementación de los algoritmos TRIAD y Filtro de Kalman. En la figura 3.11 se muestra un esquema del algoritmo a implementar.



Donde:

K es la ganancia de Kalman

P es la matriz de error de covarianza

Figura 3.11 Diagrama de bloques del algoritmo de estimación y filtrado a implementar.

Con el fin de tener una idea de los requerimientos de cómputo de este algoritmo, se calculó el número aproximado de operaciones aritméticas en punto flotante que se necesita para la implantación. En la tabla 3.3 se presenta un resumen de este cálculo, considerando el algoritmo de eliminación de Gauss para resolver la inversión de matrices.

Proceso	Sumas	Multiplicaciones	Divisiones	Raíz Cuadrada	Total
TRIAD	70	84	33	10	197
Innovación y actualización	165	152	13	1	331
Propagación	196	1899	6	0	2101
Ganancia de Kalman	0	546	6	0	552
Total	431	2681	58	10	3181

Tabla 3.4 Resumen de las características de los giróscopos (36) (37) (38) (39).

Además de estas operaciones, el proceso de conversión de matriz de rotación a cuaternión requiere las funciones trascendentales de seno, coseno y ángulo cuyo coseno, las cuales requieren recursos importantes de cómputo.

3.5.3. Sistema embebido

Considerando las funciones y objetivos que la computadora de vuelo debe cumplir, ésta se puede clasificar como un sistema embebido. Un sistema embebido es un sistema que integra diversas funcionalidades de hardware y software con el fin de realizar una tarea específica y limitada, (57), (58) y (59). Por ello, el problema del desarrollo de la computadora de vuelo y en general del subsistema de navegación se puede atacar bajo la perspectiva del desarrollo de un sistema embebido.

En el Instituto de Ingeniería, UNAM (IIUNAM), se han desarrollado y validado diversas computadoras de vuelo para aplicaciones satelitales, entre ellas las que han usado componentes de calificación militar (para el proyecto microsatelital Satex) hasta otras para aplicaciones terrestres (para el satélite SATEDU). Actualmente en el IIUNAM se desarrollan diversos sistemas embebidos para aplicaciones específicas que demandan arquitecturas muy eficientes elaboradas para ejecutar tareas de cómputo específicas. Estos sistemas se están desarrollando con tecnología FPGA empleando diversos núcleos de procesamiento interno que permiten resolver tareas de cómputo con esquemas de procesamiento serial, paralelo y multiprocesamiento. Estas arquitecturas se están desarrollando además para incluir la posibilidad de reprogramar a distancia las arquitecturas de hardware embebido de acuerdo con necesidades que pueda requerir un satélite durante su vida útil. Uno de tales casos es precisamente el sistema embebido para estimación y filtrado que se ha descrito en este capítulo.

Capítulo 4
Plataforma FPGA

En este capítulo se describirán las herramientas que se utilizaron para la implementación de la computadora de navegación. Se hace una breve descripción de los FPGA y sus principales características. Se introduce el concepto de Plataforma FPGA así como la metodología de diseño y desarrollo que comúnmente se sigue al crear sistemas embebidos con estos dispositivos, haciendo énfasis en la capacidad natural de éstos para albergar arquitecturas de cómputo paralelo.

4.1. Introducción

El éxito o falla en el desarrollo de un proyecto, medido en términos de tiempo de desarrollo y de desempeño final del sistema, depende en gran medida de que tan bien se implementa la aplicación y sus algoritmos respectivos (adaptados a los recursos de la plataforma en que se ejecutarán).

Por ello, en el desarrollo de un proyecto es indispensable identificar los módulos del diseño que deben implantarse en hardware, cuáles deberán implantarse en software en un procesador tradicional y cuándo se utilizará un circuito integrado de propósito específico (ASIC, por sus siglas en inglés), (60).

Actualmente existen dispositivos que cuentan cada vez con mayores capacidades; que permiten combinar y optimizar ambas porciones del diseño en un solo componente físico. En este sentido nos referimos particularmente a los FPGA (Field Programmable Gate Array).

Los FPGA tienen la habilidad de combinar núcleos de procesadores embebidos con periféricos estándar y módulos de hardware con funciones personalizadas. Esto hace posible construir soluciones adecuadas y a la medida para resolver tareas específicas dentro de una aplicación, lo que ha convertido a los FPGA en una excelente opción en el desarrollo de sistemas embebidos, (61).

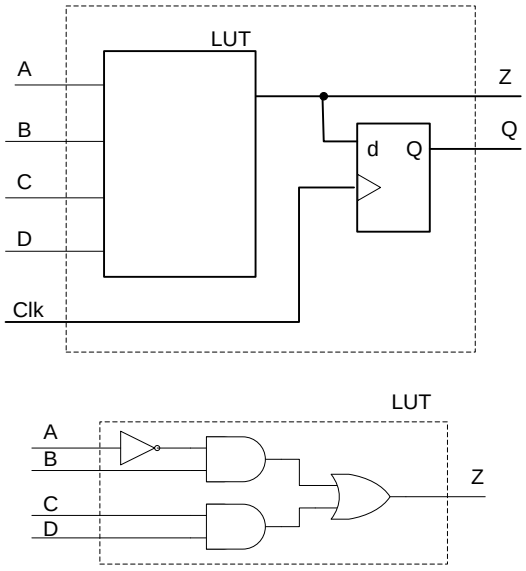
4.2. Dispositivos FPGA

Un FPGA es un dispositivo lógico que contiene un arreglo bidimensional de celdas lógicas genéricas o elementos lógico, además de tener interruptores programables, (62) y (63). El uso de estos dispositivos permite implementar funciones y circuitos digitales sumamente complejos en un solo circuito integrado (IC), (64).

Las celdas lógicas o LC (por sus siglas en inglés) pueden configurarse para desempeñar una función, en tanto que los interruptores pueden programarse para realizar interconexiones entre las celdas lógicas, (62). A partir de ambas configuraciones, es posible crear en el dispositivo una funcionalidad específica.

Tanto la arquitectura interna de un FPGA como la forma de implementar las celdas lógicas varía entre fabricantes, sin embargo, el concepto de su estructura es el mismo, (65). Las celdas lógicas usualmente están formadas por un circuito combinacional programable y por un "flip-flop" tipo D (DFF). El método más común para implementar un circuito combinacional programable emplea una tabla de búsqueda ó LUT (look-up-table). A su vez, una LUT de n entradas puede considerarse como una memoria de tamaño 2^n X 1.

En la figura 4.1 se presenta el diagrama conceptual de una LUT y el ejemplo de un circuito combinacional con su correspondiente LUT.



A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
	.	.	.	
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Figura 4.1 Diagrama de una tabla de búsqueda (LUT) y su correspondiente circuito.

Algunos FPGA también están compuestos por Macro Celdas o Macro Bloques, los cuales están diseñados y fabricados a nivel de transistores, de tal forma que sus funcionalidades complementan las características de las celdas lógicas.

En el caso de la familia de FPGAs Spartan-3 de XILINX, por ejemplo, la unidad básica que los forma es la celda lógica, la cual contiene una LUT con cuatro entradas, un DFF, un circuito de acarreo, que se utiliza para implementar operaciones aritméticas y un circuito de multiplexaje que se utiliza para implementar multiplexores más grandes. Con el fin de incrementar su

flexibilidad y desempeño, se agrupan dos LC para formar un "*slice*", en tanto que cuatro "*slices*" forman un bloque lógico configurable (CLB).

Los dispositivos Spartan-3 tienen cuatro tipos de macro bloques: multiplicador lógico, bloque de RAM, manejador de reloj digital (DCM) y un bloque de entrada y salida (IOB), (66).

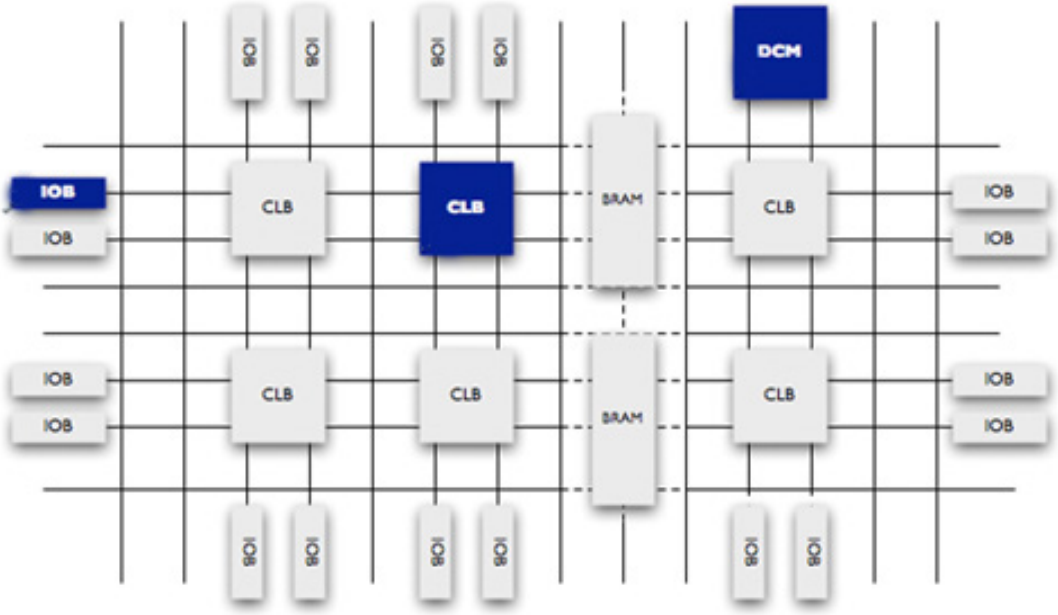


Figura 4.2 Principales bloques que componen a un FPGA.

Los multiplicadores aceptan dos números de 18 bits como entradas y con ellos calcula su producto. Los bloques de RAM tienen una memoria SRAM síncrona de 18Kbits que puede arreglarse en varios tipos de configuraciones. Un DCM utiliza un ciclo con retardo para reducir las desviaciones del reloj y controlar lo corrimientos de frecuencia y desfasamientos. Un IOB controla el flujo de datos entre los pines de entrada y salida del dispositivo (I/O) y la lógica interna. Este último puede configurarse para soportar una amplia variedad de estándares, (61).

Las celdas lógicas y macro celdas que contiene un FPGA varían en cantidad y densidad entre las diversas subfamilias que existen. Así mismo, la estructura de las celdas lógicas puede variar de acuerdo a la familia y al fabricante al que pertenezca el dispositivo.

4.3. Programación de un FPGA

La programación de un FPGA consiste básicamente en configurar las celdas lógicas y especificar los puntos de conexión entre los diferentes tipos de bloques lógicos. En los FPGA reprogramables, la información de configuración e interconexión o ruteo se almacena en RAM estática dentro del mismo dispositivo, (65) y (67). La tarea de programar se resume entonces en

crear una cadena de bits (bitstream) que contenga la información de configuración para descargarla al FPGA.

El proceso de programación comienza típicamente con la descripción del sistema mediante un lenguaje descriptor de hardware (HDL) como Verilog o VHDL (Very High Speed Integrated Circuit Hardware Description Lenguage). Estos lenguajes permiten el modelado de sistemas digitales en diferentes niveles, grados de abstracción y jerarquía, como pueden ser los de estructura y comportamiento, (68). En el nivel de comportamiento, el más abstracto, se indican las funciones que realiza el sistema, aunque no se indica cómo es que se implementan dichas funciones. El nivel de estructura tiene que ver con la forma en que el sistema está compuesto y la interconexión de los subsistemas, (69). Cada uno de estos niveles puede dividirse a su vez en distintos grados de abstracción, de ahí la clasificación de jerárquico.

El paso siguiente es la síntesis, en el cual la lógica de alto nivel especificada en el modelo de estructura y el modelo de comportamiento, son convertidos a lógica de bajo nivel, como compuertas lógicas, por ejemplo. Después, el proceso de mapeo separa las compuertas en grupos que mejor se adapten a los recursos lógicos del FPGA. Finalmente, se crea un archivo binario que contiene la información para configurar los bloques lógicos y para realizar el ruteo apropiadamente, (67).

4.4. Familias de FPGAs

En la actualidad existe un importante número de fabricantes de FPGA, así como una amplia variedad de modelos con distintas características y capacidades, los cuales están orientados a diferentes propósitos y aplicaciones.

Un FPGA se puede distinguir de otros dispositivos de su mismo tipo por características como la capacidad (número de celdas lógicas o compuertas lógicas), número de pines de entrada y salida, estructura interna, dispositivos embebidos, memoria, entre otras. Dichas características pueden servir como criterios de selección al momento decidir el dispositivo FPGA en particular que se utilizará en un proyecto. Además, los recursos tanto del dispositivo como de la plataforma FPGA en general deben ser suficientes para cubrir con los requerimientos del proyecto establecidos previamente como podrían ser: velocidad, capacidad de cómputo, tiempo de procesamiento, consumo de energía, tamaño del empaquetado y costos.

En la tabla 4.1 se presenta una comparativa de las características básicas de algunas de las familias de dispositivos de dos fabricantes de FPGA.

Fabricante	**Xilinx**		
Familia	Spartan	Kintex	Virtex
Descripción	Se utilizan principalmente en aplicaciones automotrices. Establecen un balance óptimo entre costo, consumo de energía	Son dispositivos de alto desempeño con un costo mejorado, principalmente dirigido a comunicaciones inalámbricas.	Orientado a sistemas de alto desempeño y gran ancho de banda. Algunos modelos presentan calificación militar y aeroespacial.
Recursos	Cuentan hasta con 150 mil celdas lógicas, 4.8 Mb de memoria, 540 entradas/salidas y hasta 104 bloques multiplicadores de 18x18 bits (70) (71).	Cuentan hasta con 480 mil celdas lógicas, 34 Mb de memoria, 500 entradas/salidas y bloques multiplicadores de 18x18 bits (72).	Cuentan hasta con 2 millones de celdas lógicas, 85 Mb de memoria, 1200 entradas/salidas y bloques multiplicadores de 18x18 bits (72).
Fabricante	**Altera**		
Familia	Cyclone	Arria	Stratix
Descripción	Son FPGA de bajo costo, bajo consumo de energía e ideales para producciones de altos volúmenes.	Proveen un óptimo balance entre desempeño, consumo de energía y precio para aplicaciones basadas en transceptores.	Son los FPGA de mayor densidad o escala de integración y ancho de banda. Actualmente existen cinco generaciones de esta familia.
Recursos	Cuentan con hasta 300 mil celdas lógicas, bloques de memoria interna de hasta 12Mb, hasta 688 GPIO[1] y hasta 812 multiplicadores de 18x18 bits. Su frecuencia de reloj es de 625MHz (73).	Cuentan con amplia variedad en capacidad de memoria, lógica y funciones de DSP. Cuentan con hasta 495 mil celdas lógicas, memoria de hasta 23.8Mb, 688 GPIO y más de 2000 multiplicadores. Su frecuencia de reloj es de 625MHz (74).	Incluyen transceptores con una tasa de transferencia de hasta 28 Gb. Cuenta con hasta 397 mil módulos lógicos adaptables (equivalente a 597K celdas lógicas), hasta 704 bloques DSP para operaciones aritméticas de precisión variables que incluyen 798 multiplicadores y hasta 2640 bloques de memoria de 20k equivalentes a 52Mb (75).

Tabla 4.1 Familias de FPGA y sus principales características, (63) y (76).

[1] GPIO (General Purpose Input Output) son los bloques que permiten la comunicación al resto de los bloques del FPGA con el exterior.

4.5. Plataforma FPGA

Una plataforma FPGA es una serie de recursos y funcionalidades que rodean a un dispositivo FPGA y que permiten el desarrollo de un sistema embebido por completo, (61). Los fabricantes de FPGA, además de proveer los dispositivos físicos, también brindan a sus clientes plataformas muy completas para crear sistemas embebidos. Dos de los más grandes y populares fabricantes de estos dispositivos son XILINX y ALTERA. A través de sus entornos integrados de desarrollo (IDE), Quartus II en el caso de Altera e ISE para el caso de XILINX, es posible manejar una plataforma FPGA con todos sus recursos y un gran número de herramientas que facilitan el desarrollo de un proyecto.

Un ejemplo de estos recursos son los llamados núcleos o "*cores*". Un "*core*" es un dispositivo prediseñado y reutilizable con una funcionalidad específica que se incorpora dentro de algunos FPGAs, (77). Su complejidad puede ir desde decodificadores y multiplexores sencillos, hasta ALUs (Unidad Lógica Aritmética), DSPs (Procesador Digital de Señales), microprocesadores, periféricos estándar como puertos de comunicaciones serie y unidades para la solución de algoritmos específicos como FFT (Transformada Rápida de Fourier).

Un núcleo de este tipo está descrito más por su función lógica que por su implementación física. Cuando un bloque es diseñado por una compañía o un desarrollador independiente al proyecto en donde se utiliza, se le llama *IP core* (Intelectual Property), (78). Los usuarios pueden comprar estos bloques a las compañías que los desarrollan y les permite acortar sus costos y tiempos de desarrollo, (65) y (78). Estos dispositivos embebidos son optimizados y permiten a los FPGA que los contienen, convertirse en un una solución muy adecuada al problema, a lo que se conoce como sistema en un chip o SOC (System On a Chip), lo cual permite disminuir el área en las tarjetas electrónicas y su consumo de energía, (65).

En este espacio conviene hacer una distinción entre las dos formas básicas en la que estos recursos de hardware embebidos se implementan. Un "*hard core*" es un bloque de hardware que está físicamente construido en el dispositivo como parte de la pastilla de silicio cuya funcionalidad se construye mediante transistores al manufacturarlo, (61). Los dispositivos resultantes son rápidos y sus resultados son precisos, pero los parámetros de su construcción no pueden modificarse, lo que los hace poco flexibles. En contraste, un "*soft core*" se implementa con los recursos reconfigurables del FPGA, generalmente están descritos en HDL por lo que requieren ser compilados por una herramienta de síntesis. Estos núcleos son altamente flexibles, por que se pueden modificar sus características, e incluso cambiar su código, pero al no estar diseñados para un dispositivo en específico son menos eficientes, (77) y (78).

Los núcleos permiten crear sistemas embebidos eficientes por que le brindan al FPGA la capacidad de configurarlos de manera óptima de acuerdo con las necesidades de la aplicación, pero además con cierto grado de flexibilidad.

4.5.1. Procesadores embebidos: MicroBlaze y PowerPC

En la sección anterior se indicó que es posible tener un microprocesador embebido dentro de un FPGA mediante el concepto de *"IP core"*. Un ejemplo de este tipo de dispositivos son los procesadores PowerPC y MicroBlaze, ambos desarrollados por la compañía Xilinx como parte de su plataforma para la creación de sistemas embebidos.

PowerPC es un procesador RISC (Computadora Con Conjunto Reducido de Instrucciones) de 32 bits integrado como un núcleo "*hard core"* a la familia de FPGA Virtex de XILINX. MicroBlaze es también un procesador RISC de 32 bits, pero en este caso se trata de un núcleo "soft core" que forma parte de la plataforma de desarrollo EDK (Embedded Development Kit) de XILINX, (79).

Este tipo de procesadores son capaces interactuar con otros núcleos o "*IP cores"*, generalmente mediante interfaces y buses que provee la misma plataforma de desarrollo. De esta forma, el procesador gana funcionalidades, pues es posible por ejemplo, conectarle un puerto de comunicaciones, una memoria adicional y algún otro "*core"* de propósito específico. Inclusive, en el caso del MicroBlaze, es posible conectarle otro procesador idéntico.

El MicroBlaze implementa una arquitectura Harvard, lo que significa que utiliza buses separados para acceso a los datos e instrucciones almacenados en los bloques de memoria RAM. Para conectarse a cada bus LMB (Local Memory Bus), el MicroBlaze requiere de un bloque controlador, el cual también forma parte de la plataforma.

El uso de uno u otro procesador (*hard core o soft core*) dependerá de los requerimientos de la aplicación.

4.5.2. Enlaces FSL y PLB

Con el fin de establecer conexiones con otros núcleos y periféricos, los procesadores embebidos de Xilinx utilizan típicamente dos clases de enlaces: FSL (Fast Simple Link) y PLB (Processor Local Bus).

Los enlaces FSL son canales dedicados unidireccionales punto a punto para la transmisión de datos. Tienen un tamaño de 32 bits y se pueden alcanzar tasas de transmisión de hasta 300 Mbps, aunque este tamaño puede ser configurado. Se puede utilizar un bit adicional para indicar si la palabra transmitida (o recibida) es de control o de datos. Este enlace es ideal para comunicaciones entre procesadores MicroBlaze o *streaming*[2]. Un MicroBlaze tiene puertos suficientes para ocho conexiones FSL de entrada y ocho de salida, (80).

Las comunicaciones a través de un enlace FSL están basadas en un esquema FIFO (First Input First Output). Los dispositivos conectados a través del bus FSL deben ser configurados ya sea

[2] Comunicación continua de datos sin interrupciones.

como maestros o como esclavos. Un dispositivo maestro sólo puede escribir datos a la FIFO del bus, mientras que un dispositivo esclavo sólo podrá leer.

En la figura 4.3 se muestra el principio de funcionamiento del bus FSL y las señales con las que cuenta. El "core" que se encuentra al centro de la figura tiene un enlace FSL maestro de lado izquierdo y un enlace FSL esclavo del lado derecho, implementando de esta forma una comunicación bidireccional para el núcleo.

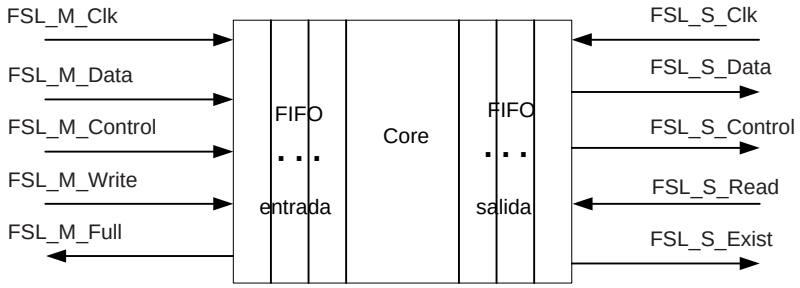


Figura 4.3 Señales de entrada y salida de un bloque con enlace FSL (Fast Simple Link).

PLB es un enlace primario compartido de alto desempeño que provee la infraestructura necesaria para conectar un número determinado de dispositivos maestros y esclavos. El bus incluye una unidad de control, un temporizador *watchdog,* direcciones de memoria separadas para la escritura y la lectura y un registro de control opcional DCR (Device Control Register), (81).

En la figura 4.4 se ejemplifica la forma en que el bus PLB es compartido por diversos dispositivos para comunicarse con el procesador embebido.

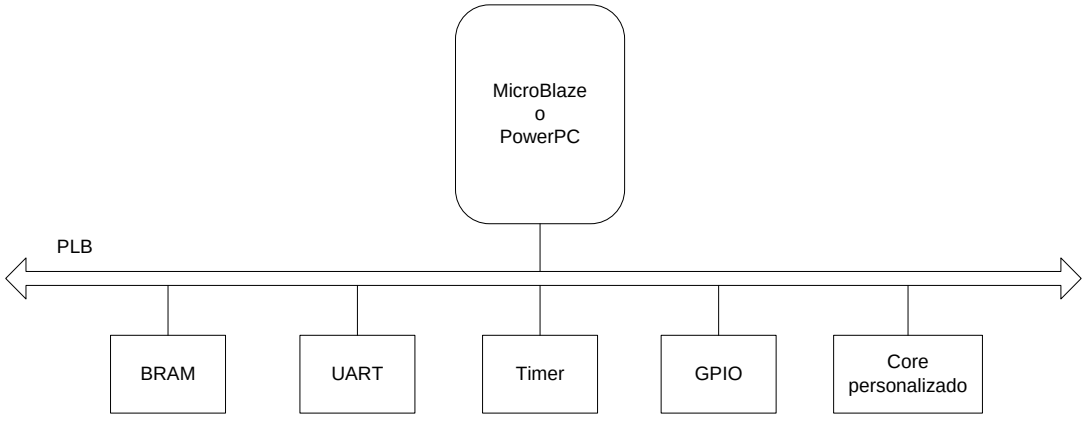


Figura 4.4 Conexión de un procesador embebido a otros "*cores*" mediante un enlace PLB.

4.5.3. Entorno de desarrollo

Los fabricantes suelen proporcionar al cliente su propia plataforma de desarrollo de sistemas embebidos en software, optimizada para los dispositivos que fabrican. En el caso de Xilinx, se cuenta con los entornos de desarrollo EDK e ISE, los cuales a su vez, están compuestos de distintas herramientas que se mencionan a continuación.

a) EDK

El Kit de Desarrollo Embebido EDK (*Embedded Development Kit*) es un conjunto de herramientas de diseño que permite crear de principio a fin, sistemas basados en un procesador embebido y que además permite implementarlos en un dispositivo FPGA de Xilinx, (82). Estas herramientas son:

- La interfaz gráfica de usuario XPS (*Xilinx Platform Studio*).
- La suite de herramientas del sistema embebido.
- Núcleos de procesadores embebidos y otros "*IP cores*" como periférico.
- El Kit de Desarrollo de Software SDK (*Software Development Kit*).

b) ISE

Junto con el EDK, es necesario instalar también el Entorno de Software Integrado ISE (Integrated Software Environment). EDK depende de ISE para realizar la síntesis del diseño en hardware del microprocesador embebido, mapearlo al FPGA destino, así como generar y descargar el archivo binario al dispositivo, (83). Algunas de las herramientas con las que cuenta ISE son las siguientes:

- El navegador de proyectos.
- La herramienta de síntesis XST (Xilinx Synthesis Tool).
- Las herramientas de ruteo y posicionamiento de lógica.
- Herramientas para la programación del dispositivo (iMPACT).
- Librerías de componentes lógicos de propósito general.

En nuestro caso, ISE resultó ser una herramienta muy útil para crear y sintetizar núcleos personalizados (*Customized Cores*) y componentes propios, a través de su editor de VHDL y las herramientas XST y Xilinx Core Generator.

c) XPS

XPS brinda una interfaz gráfica de usuario (GUI) para crear las especificaciones de hardware y software de sistemas basados en los procesadores embebidos MicroBlaze y PowerPC. Provee de un editor y un administrador de proyectos para crear y editar código fuente. También posee un

editor gráfico para personalizar las conexiones del procesador, periféricos y buses, (82). Mediante XPS es posible realizar las siguientes tareas:

- Añadir núcleos, editar sus parámetros y realizar sus conexiones.
- Crear y modificar los archivos MHS (Microprocessor Hardware Specification) y MSS (Microprocessor Software Specification).
- Generar y visualizar un diagrama de bloques del sistema, así como el reporte del diseño.

El archivo MHS define la "plataforma en hardware", la cual consiste básicamente en los componentes que comprenden el sistema embebido (procesador y periféricos), sus características y la forma en que están interconectados (buses y puertos).

El archivo MSS captura la "plataforma en software", que se refiere a la colección de controladores (software) para los distintos elementos y componentes en hardware (*IP cores*, buses, procesadores embebidos), y opcionalmente, al sistema operativo sobre el cual se construirá una aplicación.

Mediante esta herramienta es posible también integrar los núcleos diseñados previamente en ISE al proyecto principal, mediante la interconexión de éstos con el MicroBlaze a través de enlaces tipo FSL. En ocasiones, es necesario modificar los archivos MHS y MSS con el fin de especificar ciertos requerimientos de la aplicación.

d) SDK

Es una interfaz gráfica de usuario complementaria a XPS, que provee un entorno de desarrollo para aplicaciones de software incluidas en el proyecto, como son los programas que el procesador embebido deberá ejecutar. Algunas de sus características son las siguientes, (84):

- Un editor de código C/C++, junto con un entorno para corrección de errores, depuración de código y compilación.
- Un administrador de proyectos.

4.6. Diseño y desarrollo de sistemas embebidos sobre plataformas FPGA

Una gran ventaja de los FPGA sobre otros dispositivos, ya mencionada en varias ocasiones, es su capacidad para integrar al diseño núcleos personalizados (*Costumized Cores o IP Cores*) lo que resulta en una dramática aceleración de los procesos y tiempo de ejecución en software, debido a que los algoritmos se ejecutan paralelamente en bloques de hardware y no secuencialmente en código de software, (80).

En la figura 4.5, por ejemplo, se puede observar una comparativa de la ejecución de un algoritmo mediante una solución en software (izquierda) y la ejecución del mismo algoritmo en hardware (derecha), mediante núcleos. La solución en software requiere ejecutar un mayor número de operaciones para llegar a la misma solución que se obtendría mediante hardware.

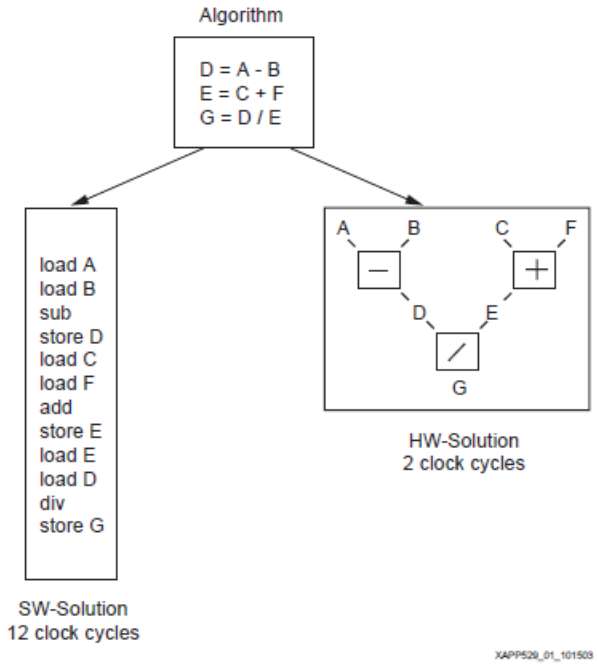


Figura 4.5 Comparativa de algoritmo ejecutado en software y en hardware (80).

Por otro lado, los procesadores *"soft core"* (SCP) también presentan ventajas sobre otros procesadores como es la de sólo utilizar las características mínimas requeridas del procesador para una aplicación en específico.

Gracias a esta dualidad de recursos, es decir, a la combinación de la flexibilidad del software con el alto desempeño del hardware, un diseñador que trabaja con FPGA debe pensar diferente de aquellos diseñadores que utilizan otros dispositivos. Los desarrolladores de software típicamente escriben programas secuenciales que explotan la habilidad de los microprocesadores para ejecutar con rapidez una serie de instrucciones. En contraste, un diseño de alta calidad en un FPGA requiere pensar en el "paralelismo espacial", esto es, utilizar simultáneamente los recursos distribuidos a lo largo del chip para realizar una gran cantidad de operaciones de cómputo, (67).

En la figura 4.6 se muestra un esquema de cómputo en paralelo en el que dos SCP del tipo MicroBlaze (MB_0 y MB_1) interactúan entre sí y con otros núcleos que pueden realizar diversas tareas. Los SCP trabajan bajo un esquema de co-procesamiento, en el que es posible incluir un número mayor de procesadores de este tipo, si las necesidades del sistema así lo requieren. Al mismo tiempo los "*cores*", además de ser empleados como periféricos, pueden

también aliviar el trabajo de software de los procesadores realizando tareas en paralelo bajo un esquema de aceleración por hardware.

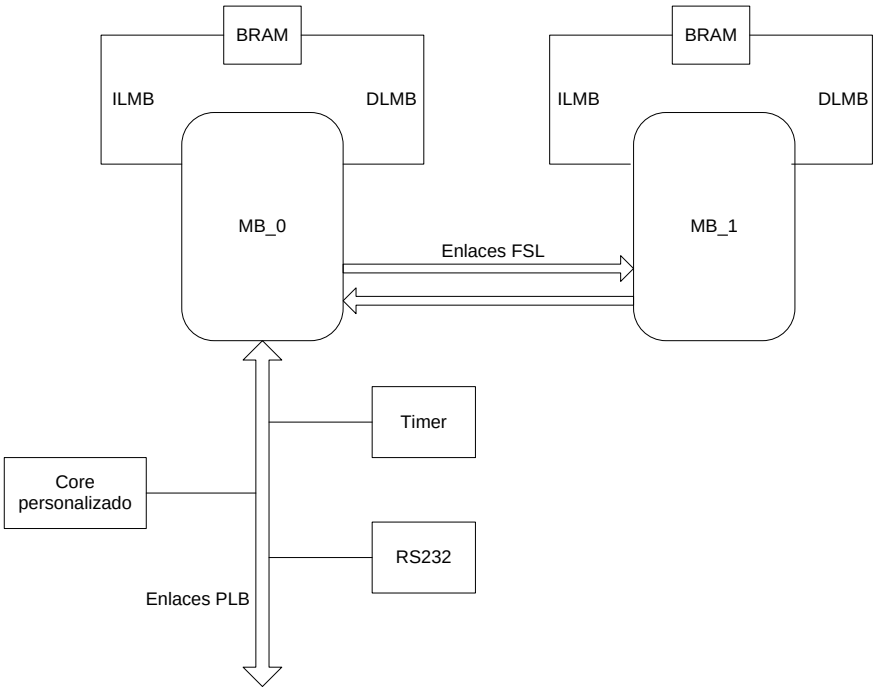


Figura 4.6 Esquema de cómputo en paralelo y co-procesamiento.

La importancia de este modelo de desarrollo radica en que permite considerar las aplicaciones y algoritmos en nuevas maneras, como la de utilizar técnicas de cómputo en paralelo que tal vez sean poco comunes, con el fin de obtener el máximo desempeño posible.

Una gran mayoría de aplicaciones que requieren de tareas de procesamiento en tiempo real pueden ser reducidas a un conjunto de operaciones matriciales básicas o primitivas, tales como la multiplicación de una matriz por un vector o por otra matriz, la suma de matrices y la inversión de matrices.

Muchos algoritmos han sido desarrollados para efectuar este tipo de operaciones básicas. Éstos varían en sus características y requerimientos de cómputo, como memoria, cantidad y tipo de operadores, así como la cantidad y tipo de paralelismo. Generalmente los algoritmos utilizados se basan en subrutinas del álgebra lineal básica y requieren altas velocidades de cómputo para cumplir con los tiempos que requieren las aplicaciones.

Algunos algoritmos trabajan de forma adecuada para su ejecución en forma secuencial (con un solo procesador) por que requieren de un número reducido de operaciones y memoria. Por otro lado, existen algoritmos que aprovechan las características de la ejecución en paralelo, los

cuales han llegado a convertirse en los algoritmos preferidos para desarrollar operaciones con matrices debido a su eficiencia.

Sin embargo, esta eficiencia es reducida cuando los algoritmos se ejecutan en arquitecturas o sistemas de propósito general, y se obtienen mejores resultados cuando se construye una arquitectura de propósito específico para ejecutar el algoritmo.

Los arreglos sistólicos, por otro lado, constituyen una clase de arquitectura de propósito específico que se adecúa de manera natural a las operaciones con matrices.

Generalmente las operaciones con matrices constituyen sólo una parte de una aplicación más grande y compleja, sin embargo, en otras ocasiones corresponden a una importante parte de la carga de trabajo de cómputo. Un arreglo o arquitectura de propósito específico como ésta puede conectarse a un procesador que realice las funciones de entrada y salida (I/0) así como funciones de control, tal como se muestra en la figura 4.6. En consecuencia, es posible construir un sistema heterogéneo en donde un módulo de procesamiento de matrices se conecte con otros módulos para realizar la tarea completa.

4.7. Arreglos sistólicos y algoritmos para cómputo en paralelo

Un arreglo sistólico es una red local de elementos de procesamiento interconectados en una estructura regular finita (multidimensional). Los elementos de procesamiento (PE) son idénticos o de una variedad muy limitada, (85). Los datos fluyen rítmicamente a través de un arreglo sistólico gracias al control y sincronización con una señal de reloj global. Los arreglos son por naturaleza modulares y capaces de repetirse, (86).

El poder de procesamiento de estas estructuras radica en el uso concurrente de un gran número de celdas más que en el uso secuencial o repetido de un número pequeño de ellas. Son particularmente adecuadas para la implementación de algoritmos paralelos y algoritmos con flujo regular de datos, como las operaciones basadas en matrices, (87).

El máximo paralelismo se consigue cuando el mayor número de celdas están activas al mismo tiempo. Los requerimientos en cuanto a recursos de entrada-salida, los cuales generalmente son un factor limitante, se reducen cuando los datos son “bombeados” de una celda a otra dentro del arreglo aumentando el número de operaciones internas realizadas entre cada operación de entrada-salida, (87).

En la figura 4.7 se presentan algunos ejemplos de arreglos sistólicos. El primero es un arreglo lineal en donde cada procesador es conectado con sus vecinos a la izquierda y derecha, el segundo es un arreglo de anillo, el cual se obtiene a partir de realimentar el resultado del último procesador al primero. Estos dos primeros arreglos son de una sola dimensión, pero también existen de dos o más dimensiones, los más comunes son los arreglos ortogonales y hexagonales. También son comunes los arreglos en forma de árbol binario.

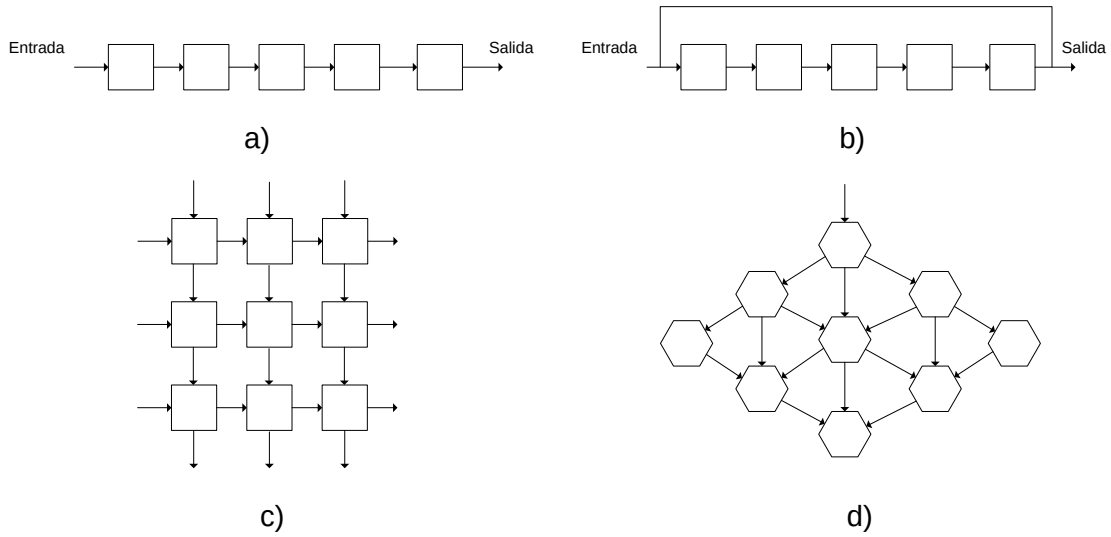


Figura 4.7 Arreglos sistólicos: a) pipelined, b) pipelined con realimentación, c) ortogonal, d) hexagonal.

Capítulo 5
Implementación

El presente capítulo se concentra en la descripción de la implementación del algoritmo de estimación y filtrado, a través del desarrollo de los tres principales bloques en los que si dividió la computadora de navegación: Adquisición de datos de sensores, Método TRIAD y Filtro de Kalman. En la implementación de dicho sistema embebido, se pretende aprovechar de forma óptima los recursos de la plataforma FGPA en donde se ejecutarán los algoritmos, es por ello que estos bloques fueron desarrollados mediante una conjunción de recursos en hardware (cores) y elementos de software (rutinas ejecutadas por el procesador embebido MicroBlaze), consiguiendo así, un compromiso entre factores como el tiempo de ejecución y los recursos físicos del dispositivo FPGA.

5.1. Adquisición de datos de sensores

Como se mencionó en la sección 3.3.3, los sensores acelerómetro, giróscopo y magnetómetro están integrados en una sola tarjeta y conectados a un bus I^2C. La lectura de las mediciones se llevó a cabo añadiendo al desarrollo un periférico (*IP core*) que maneja este protocolo de comunicación serial en adición a un componente en software (driver) que gestiona el control de las comunicaciones, así como el propio programa encargado de la adquisición de datos que se ejecuta dentro del procesador embebido *MicroBlaze.*

Para entender mejor este desarrollo, se explicarán brevemente las bases del estándar I2C. Posteriormente se hablará del funcionamiento y utilización de la interfaz del periférico, y finalmente de la estructura del programa de adquisición.

5.1.1. Transferencia de datos mediante el estándar I^2C

IIC (Inter Integrated Circuit) ó I^2C es un estándar de comunicaciones serial bi-direccional entre circuitos integrados que utiliza únicamente dos señales: SDA para la transferencia de datos y SCL para la señal de reloj[3]. Cada dispositivo conectado al bus tiene una dirección única de 7 o 10 bits, que opera como receptor y transmisor sobre la misma línea y también como maestro o esclavo, (88) y (89).

[3] En la práctica, generalmente se utilizan cuatro alambres: uno para alimentación o voltaje de polarización (VCC), uno para la señal SDA, otro para la señal SCL y uno más como referencia eléctrica (GND).

La transferencia de datos se lleva a cabo como sigue: el dispositivo configurado como maestro (en este caso el MicroBlaze) genera la señal de reloj e inicia la transferencia de datos enviando la dirección del dispositivo con quien desea comunicarse más un bit de lectura o escritura (R/W), todo esto a partir de una condición de inicio o START definida por una combinación específica de los niveles de las señales SDA Y SCL.

Escritura de un byte

Maestro	S	AD+W		DR		Dato		P
Esclavo			A		A		A	

Lectura de un byte

Maestro	S	AD+W		DR		P	S	AD+R			$A/\bar{A}$	P
Esclavo			A		A				A	Dato		

Escritura de múltiples bytes (escritura en *burst*)

Maestro	S	AD+W		DR		Dato		Dato		P
Esclavo			A		A		A		A	

Lectura de múltiples bytes (lectura en *burst*)

Maestro	S	AD+W		DR		P	S	AD+R			A		$A/\bar{A}$	P
Esclavo			A		A				A	Dato		Dato		

Términos utilizados

Señal	Descripción
S	Condición de inicio: SDA en flanco de baja mientras SCL está en alto
AD	Dirección I^2C del dispositivo esclavo
W	Bit de escritura (0)
R	Bit de lectura (1)
A	Acuse de recibido: SDA está en bajo y SCL está en alto durante el noveno ciclo de reloj
$\bar{A}$	No recibido: SDA permanece en alto durante el novena ciclo de reloj
DR	Registro interno del dispositivo
Dato	Dato transmitido o recibido
P	Condición de paro: SDA en flanco de subida mientras SCL está en alto

Figura 5.1 Esquema de transferencia de datos bajo el estándar I^2C.

Si en el bus se encuentra montado un dispositivo con esa dirección, éste responderá con una señal de recibido ACK (aknowledge) con la señal SDA en bajo durante un ciclo de reloj, figura 5.1. El dispositivo configurado como maestro (en nuestro caso los sensores) sólo responderá a la trasferencia con otro dato de 8 bits (escribir) o aceptará los datos (leer) si éstos están destinados para él. Es posible que el esclavo siga enviando datos de 8 bits si el receptor responde con señales de recibido ACK.

La transferencia de datos termina cuando el maestro establece la condición de paro o STOP, por ejemplo, cuando se obtiene una señal ACK negativa o se puede continuar con ese mismo u otro dispositivo si se establece la condición de inicio repetido o *repeated* START.

En la figura 5.1 se presenta el esquema de transferencia de datos utilizado en la implementación. Cabe mencionar que éste tiene ligeras diferencias con el esquema especificado por los fabricantes de cada uno de los sensores, sin embargo, como todos ellos cumplen con el estándar es posible utilizar el que se muestra enseguida, (35), (32) y (38).

5.1.2. IP core y driver para la transferencia de datos

La adquisición de datos es una función más de la computadora de navegación. Al tener ya un procesador embebido *MicroBlaze* como pieza central de ésta, sólo es necesario agregar dicha funcionalidad mediante elementos de hardware y software extra.

El elemento de hardware agregado es el periférico XPS IIC Interface v.2.00.a que se encuentra dentro del catálogo de *IP cores* del EDK, figura 5.2. El periférico se conecta al MicroBlaze mediante el bus PLB y se especifican sus características como la frecuencia de la señal SCL y la utilización de una dirección I^2C de 7 bits. Además, en al archivo UCF (User Constraints File), se configura la interfaz eléctrica, que involucra las resistencias de pull-up en los pines de salida (activadas) y los niveles de voltaje (3.3 [V]).

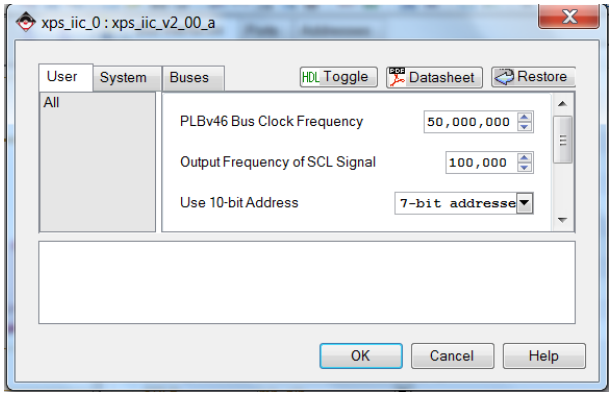


Figura 5.2 Configuración del bloque para la comunicación I^2C.

Al agregar el bloque de hardware, el software controlador se añade automáticamente al proyecto, aunque es posible modificar la versión manualmente, alterando el archivo MSS. En nuestro caso se utilizó la versión 1.13.b del controlador llamado iic.

El controlador está formado por diversos archivos y librerías escritas en código C que contienen funciones básicas que permiten al usuario enviar datos a través del bus I^2C con cierto grado de transparencia. Éstas permiten primordialmente enviar los bits de dirección del dispositivo, los bits de lectura y escritura (R/W) y los datos a transferir. Queda por parte del usuario programar

la forma en que estás funciones serán utilizadas, es decir, programar la secuencia o estructura de transferencia de datos especificada en la sección anterior.

Por ello, a partir las funciones predefinidas del controlador, se creó una librería propia (iiclib.h) que permite escribir o leer datos y obtener mediciones de los sensores de manera trasparente para la aplicación final. De esta forma se consigue simplificar el desarrollo del programa de adquisición de datos.

A continuación se presenta una descripción de las funciones creadas:

Nombre	writeI2C	
Función	Se utiliza para escribir un dato en un registro de algún dispositivo que esté conectado al bus I2C.	
Parámetro	Tipo de dato	Descripción
DISP_ADD	No signado 8 bits	Dirección I2C del dispositivo
reg_DISP	No signado 8 bits	Registro interno del dispositivo
data	No signado 8 bits	Dato que se quiere escribir en el registro

Nombre	readI2C	
Función	Se utiliza para leer un registro o varios registros de algún dispositivo que esté conectado al bus I2C.	
Parámetro	Tipo de dato	Descripción
DISP_ADD	No signado 8 bits	Dirección I2C del dispositivo
reg_DISP	No signado 8 bits	Registro interno del dispositivo
*Data_Rec	No signado 8 bits	Apuntador al arreglo donde se almacenarán los datos recibidos
length	No signado 8 bits	Cantidad de datos que se requieren leer

Nombre	Measure	
Función	Realiza la lectura de los registros que almacenan el resultado de la medición de un sensor.	
Parámetro	Tipo de dato	Descripción
DISP_ADD	No signado 8 bits	Dirección I2C del dispositivo
*Measures	Entero 16 bits	Apuntador a un arreglo de 3 elementos que almacena las mediciones de los 3 ejes del sensor.

5.1.3. Validación de la lectura de las mediciones de los sensores

Con el fin de validar el funcionamiento de las funciones programadas y de la correcta utilización de los periféricos, se implementó un diseño que permite obtener los datos de los sensores y enviarlos a una PC de forma serial bajo el estándar RS-232, figura 5.3.

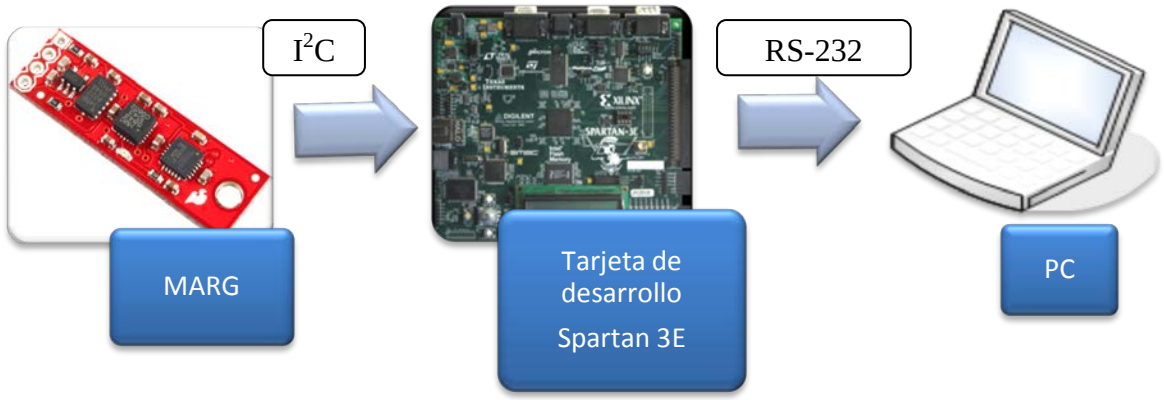


Figura 5.3 Diagrama de bloques para la validación del bloque de adquisición de datos de sensores.

El sistema implementado en el FPGA consta de un procesador embebido MicroBlaze que controla las comunicaciones de la interface I^2C con los sensores y las comunicaciones del bloque UART (Transmisor-Receptor Asíncrono Universal) con la PC. Ambos periféricos se conectan con el procesador a través del bus PLB.

Con el fin de establecer un periodo de muestreo de sensores, se agregó también al desarrollo un temporizador y un controlador de interrupciones. El temporizador de periodo fijo FIT (Fixed Interval Timer) genera una señal interrupción cada vez que se alcanza el periodo previamente configurado. Dado que el *MicroBlaze* sólo es capaz de manejar las interrupciones generadas por una sola fuente, y pensando en que futuros desarrollos podrían requerir de más fuentes de interrupciones, se agregó al presente desarrollo un *core* que permite gestionar diversas fuentes de interrupciones. El esquema del diseño implementado se presenta en la figura 5.4.

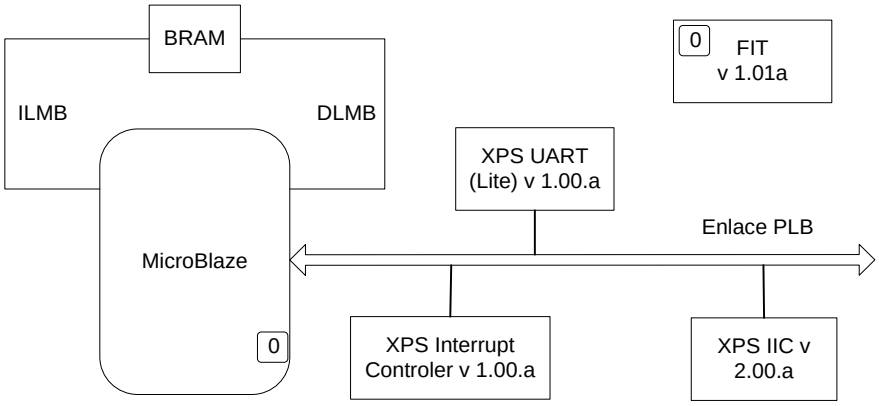
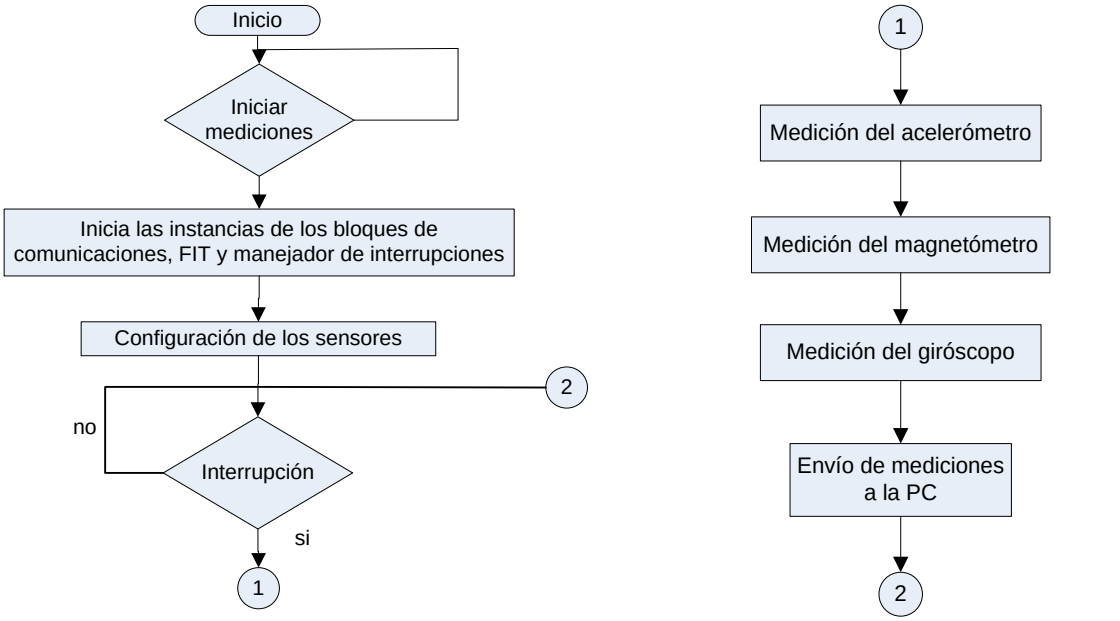


Figura 5.4 Arquitectura de hardware para la adquisición de datos de sensores.

En primera instancia el programa realiza la configuración inicial de los periféricos (*cores*) utilizados y posteriormente de los sensores con el fin de obtener lecturas adecuadas, mediante la escritura a registros particulares. Se habilita la medición del acelerómetro escribiendo un '1' en el cuarto bit del registro POWER_CTL (0x2D). En el magnetómetro, se configura el rango de medición a ±1.0 [Ga] en el registro de configuración B (0x01) y se coloca al dispositivo en Modo de Conversión Simple configurando el registro de Modo (0x02). Finalmente, en el giróscopo se selecciona una medición de escala completa, una frecuencia de muestreo de 1[kHz] y una frecuencia de corte del filtro paso bajas en 42 [Hz] escribiendo el dato 0x1F en el registro DLPF (0x16).

Después, se toman las mediciones de cada uno de los sensores para ser finalmente enviadas a la PC. El diagrama de la figura 5.5 resume el flujo del programa de adquisición de datos.



a) Función principal b) Rutina de manejo de interrupción

Figura 5.5 Diagrama de flujo del programa de adquisición de datos.

Por su parte, la PC ejecuta un script de MATLAB que lee los datos que llegan al puerto de comunicaciones seriales de ésta, y los graba en memoria. Una vez en la memoria, los datos se transforman a las unidades correctas mediante los factores de escala que se presentan en la tabla 5.1 y se despliegan en gráficas para su posterior análisis.

Dispositivo	Factor de escala [4]
Acelerómetro	3.9/1000*g[m/s^2]
Magnetómetro	1/1300 [mGa]
Giróscopo	1/14.375 [rad/s]

Tabla 5.1 Factores de conversión para los datos adquiridos desde los sensores.

5.2. Implementación del método TRIAD

El primer paso para llevar a cabo la implementación del método TRIAD fue la identificación los procesos repetitivos y los procesos complejos que pueden exigir altos recursos de cómputo y que, por lo tanto, son candidatos a ser desarrollados en bloques de hardware independientes. En seguida se analizó la viabilidad de convertir dichos procesos, que originalmente se encuentran descritos como modelos en software, a bloques de hardware y por lo tanto a describirlos en HDL.

En base a lo anterior, se determinó que estos procesos deberían ser el cálculo de vectores unitarios y el cálculo del producto cruz de vectores, los cuales se repiten en diversas ocasiones dentro del algoritmo. Por su complejidad, parte del proceso de transformación de una matriz de rotación a un cuaternión también fue construida en un bloque de hardware, específicamente, el cálculo de las funciones trigonométricas: *ángulo cuyo coseno*, *seno* y *coseno*.

5.2.1. Implementación del núcleo Norma para el cálculo de vectores unitarios

Este núcleo tiene la intención de normalizar un vector, es decir efectuar la siguiente operación:

$$\widehat{u} = \frac{u}{\|u\|} \quad \text{Ecuación 5-1}$$

Donde $\|u\|$ es la norma del vector u de dimensión n y que se calcula de la siguiente forma:

$$\|u\| = \sqrt{u_1^2 + u_2^2 + u_3^2 + \cdots + u_n^2} \quad \text{Ecuación 5-2}$$

El primer paso de la implementación fue describir el método en pseudo-código o mediante algún lenguaje de programación software, de tal forma que se puedan visualizar las operaciones básicas o fundamentales a realizar. A continuación se muestra el algoritmo para la normalización de un vector de tres elementos.

[4] g = 9.80665

```
u    = [u(1), u(2), u(3)];
norma = u(1)*u(1)+u(2)*u(2)+u(3)*u(3);
norma = sqrt(norma);
n(1) = u(1)/norma;
n(2) = u(2)/norma;
n(3) = u(3)/norma;
n    = [n(1), n(2), n(3)];
```

En seguida, el código anterior es transformado a un diagrama de bloques, donde los bloques son precisamente los encargados de realizar dichas operaciones básicas, figura 5.6. En este caso dichas operaciones básicas son: multiplicación, suma, raíz cuadrada y división.

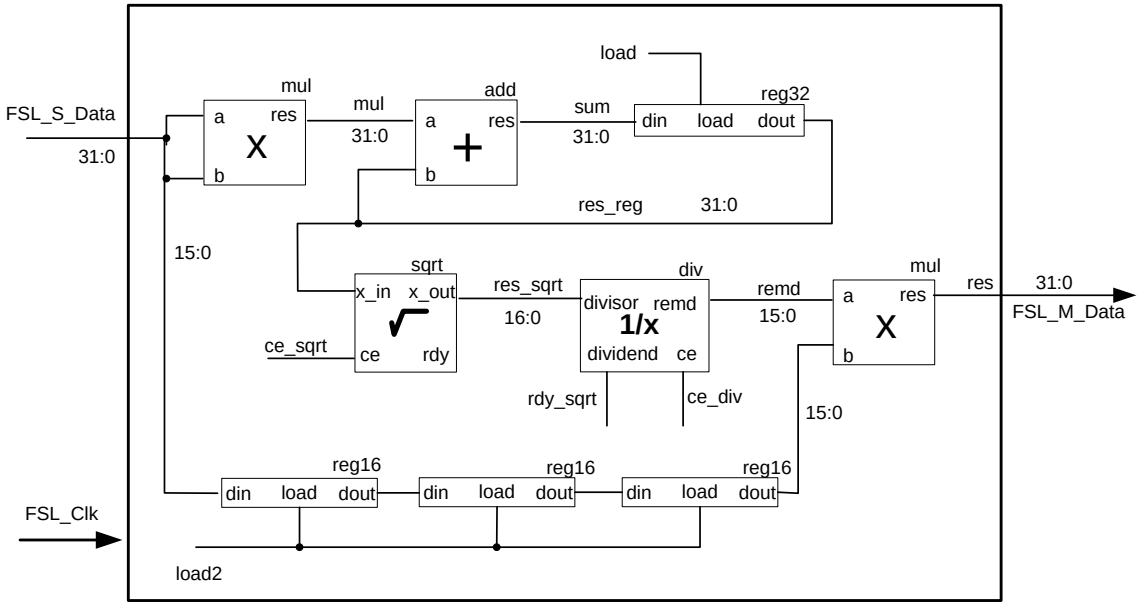


Figura 5.6 Diagrama de bloques para el cálculo de vectores unitarios.

Esta técnica facilita el análisis para la implementación en hardware. Por ejemplo, en el diagrama de bloques se observa que el algoritmo consiste en encontrar la norma del vector, calcular el recíproco de ésta y multiplicar el resultado obtenido por cada uno de los elementos del vector, a diferencia del pseudo-código donde cada elemento del vector es dividido entre la norma. Esto último es ineficiente, pues de hacerlo así se requerirían realizar tres divisiones, una operación de gran complejidad en hardware, por lo que resulta mejor calcular un recíproco y realizar tres multiplicaciones.

Por último, el diagrama de bloques es llevado a un modelo de hardware mediante VHDL utilizando el entorno de desarrollo ISE de Xilinx. Para ello, es necesario realizar un análisis de la naturaleza y características de los datos que serán operados por el bloque de hardware.

a) Descripción del modelo de hardware

En primer lugar se consideró que los datos de los sensores tienen una resolución de 16 bits por lo que la misma resolución para las operaciones dentro del bloque sería adecuada. Una resolución mayor implicaría la utilización de mayores recursos del FPGA y mayores tiempos de ejecución.

El análisis de la resolución es de la mayor importancia, pues los datos y las operaciones aritméticas se realizan en punto fijo, lo que igualmente se traduce en eficiencia de ejecución, sin embargo, también aumenta considerablemente la complejidad y tiempo de diseño.

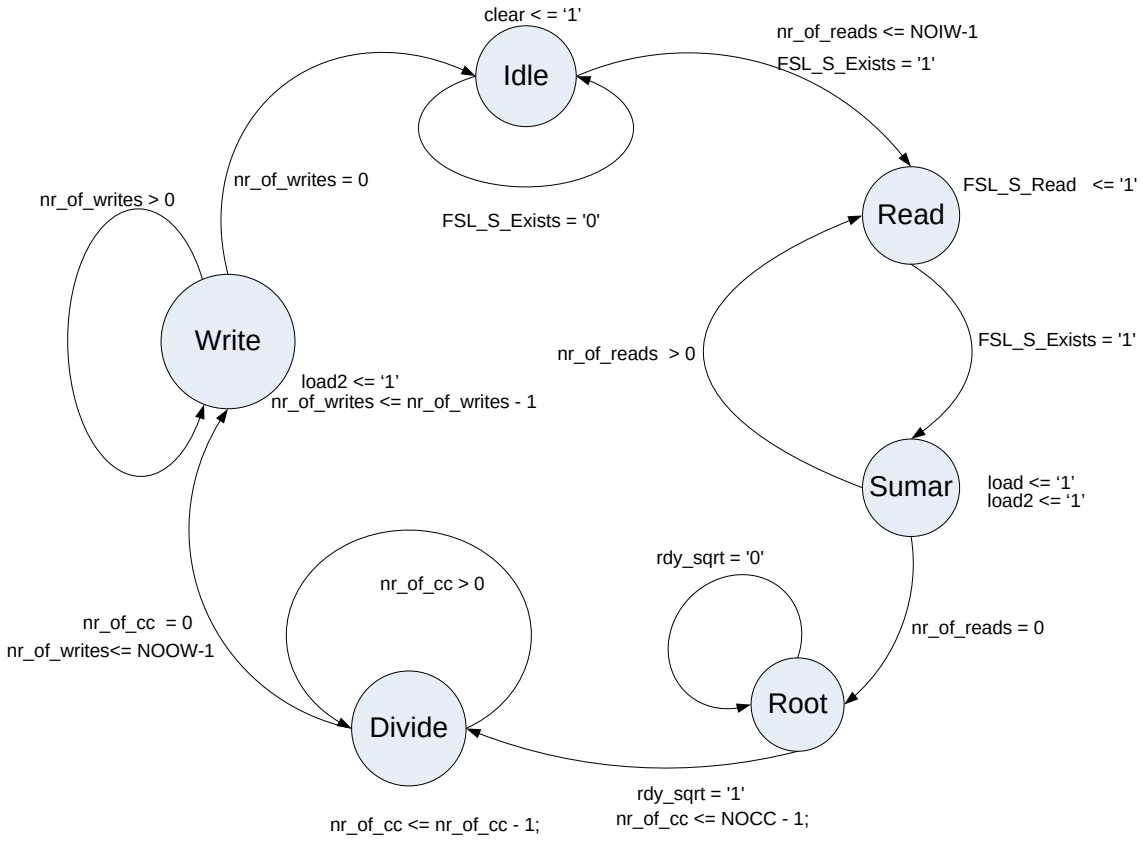
Otro aspecto importante al trabajar con aritmética en punto fijo, consiste en no perder de vista la posición del punto después de realizar cada operación y también en considerar en que operaciones se podría tener un desbordamiento de datos (overflow) con el fin de tomar las medidas pertinentes, que normalmente consisten en realizar corrimientos a la derecha, los cuales irremediablemente implican pérdidas en la precisión.

La descripción de este bloque en VHDL utiliza modelos previamente diseñados como son registros para almacenamiento temporal, multiplicador y sumador de números signados e IP cores de Xilinx generados mediante la herramienta *CORE Generator* del entorno de desarrollo ISE. Para calcular el recíproco de un número se utilizó el núcleo *Pipelined Divider* v3.0 y para el cálculo de la raíz cuadrada se utilizó el núcleo CORDIC v3.0 que implementa dicho algoritmo.

b) Funcionamiento

Una máquina de estados controla el flujo de los datos dentro del bloque de hardware. Dado que éste se comunica mediante un enlace FSL con el procesador embebido *MicroBlaze*, se requiere además de un proceso que controle dicha comunicación. Con el fin de simplificar el diseño y ahorrar recursos, ambos procesos puede unirse en uno sólo bajo la misma máquina de estados en un mismo proceso, tal como se refleja en la figura 5.7.

El proceso consta de seis estados. Durante el estado *Idle*, se espera la llegada del primer dato. Si llega un dato al bloque (FSL_S_Exists='1'), el sistema cambia al estado de *Read* donde se activa la lectura del dato (FSL_S_Read='1') y se transita inmediatamente al estado Sumar, donde se lleva a cabo la multiplicación del dato por sí mismo y la suma de este producto con el resultado de la suma anterior (al inicio del proceso este resultado es igual a cero). Este proceso se repite hasta que se hayan ingresado los tres datos que conforman el vector (nr_of_reads = 0), para entonces poder transitar al estado de *Root*, donde se calcula el módulo del vector. Una vez que la raíz cuadrada fue calculada (rdy_sqrt = '1') entonces se calcula el recíproco en el estado Divide. En este estado es necesario esperar NOCC ciclos hasta que el cálculo se haya completado.



Constantes:

NOIW = 3	Number of input words
NOCC = 20	Number of clock cycles (Latencia de la operación división)
NOOW = 3	Number of output words

Figura 5.7 Máquina de estados del bloque para el cálculo de vectores unitarios.

Una vez completado el cálculo, se transita hacia el estado *Write* donde se realizan las multiplicaciones de los elementos originales del vector con el recíproco del módulo del vector y además se envía el resultado de vuelta al procesador embebido *MicroBlaze*.

5.2.2. Implementación del núcleo Cross_P para el cálculo de producto cruz

El propósito de este núcleo es el de realizar el producto cruz de dos vectores de tres elementos cada uno, es decir, efectuar la operación:

$$u \times v = r \qquad \text{Ecuación 5-3}$$

Los elementos de $r = [r_1\ r_2\ r_3]$ son calculados mediante las siguientes operaciones aritméticas:

$$r_1 = u_2 v_3 - u_3 v_2$$
$$r_2 = u_3 v_1 - u_1 v_3$$
$$r_3 = u_1 v_2 - u_2 v_1$$

Ecuación 5-4

El siguiente paso es analizar tales expresiones de tal forma que se pueda construir una arquitectura capaz de implementarlas con un mínimo de operaciones que se ejecuten en un mínimo de tiempo. Lo anterior implica evaluar un compromiso entre tiempo de ejecución y recursos del FPGA. Para ello fue necesario identificar propiedades comunes entre todas las expresiones, tales como características repetitivas, patrones u operaciones en paralelo.

Este análisis permitió identificar que expresiones pueden evaluarse eficientemente en hardware realizando dos multiplicaciones simultáneas, seguidas de una resta del resultado de dichas multiplicaciones en tres tiempos.

El diagrama de bloques que se presenta en la figura 5.8 muestra la lógica que se utilizó para evaluar las expresiones.

Además de los bloques para realizar las operaciones aritméticas (multiplicación y resta) se requieren de otros elementos para darle funcionalidad al bloque de hardware, como son los registros para almacenar datos de entrada y la máquina de estados que controla el proceso de comunicación con el procesador embebido y el flujo de datos a través del módulo.

a) Funcionamiento

Al igual que en el caso del bloque para calcular vectores unitarios, los procesos de comunicación y control del flujo de datos del bloque para el cálculo de producto cruz están fusionados en uno sólo a través de una máquina de estados que básicamente controla la forma en que son cargados los datos en los registros temporales de 16 bits.

La máquina se conforma de tres estados (Figura 5.9.). En el estado *Idle*, el *core* está en espera de recibir un dato proveniente del procesador embebido *MicroBlaze*. Cuando se recibe el primer dato (FSL_S_Exists = ’1’) el proceso transita al estado de *Read,* en el cual, los elementos de los vectores a operar son ingresados y almacenados temporalmente dentro del *core* a través del multiplexor de entrada con la señal de selección en alto.

Una vez que los datos se encuentran almacenados en el registro temporal correcto, el proceso transita hacia el estado de *Write*. En este estado la señal de selección del multiplexor de entrada cambia a bajo, de tal forma que el arreglo de registros de entrada se convierte en una especie de arreglo de registros de corrimiento circular. Los datos en este arreglo fluyen en sentido de las manecillas del reloj, lo que permite que los datos entren a los bloques de multiplicación en el

orden correcto, aprovechando así, un patrón que se identificó en el algoritmo para el cálculo del producto cruz. Al mismo tiempo, que esto sucede, los bloques de multiplicación y resta realizan sus operaciones y entregan un resultado que es devuelto al procesador embebido, todo en un solo ciclo de reloj.

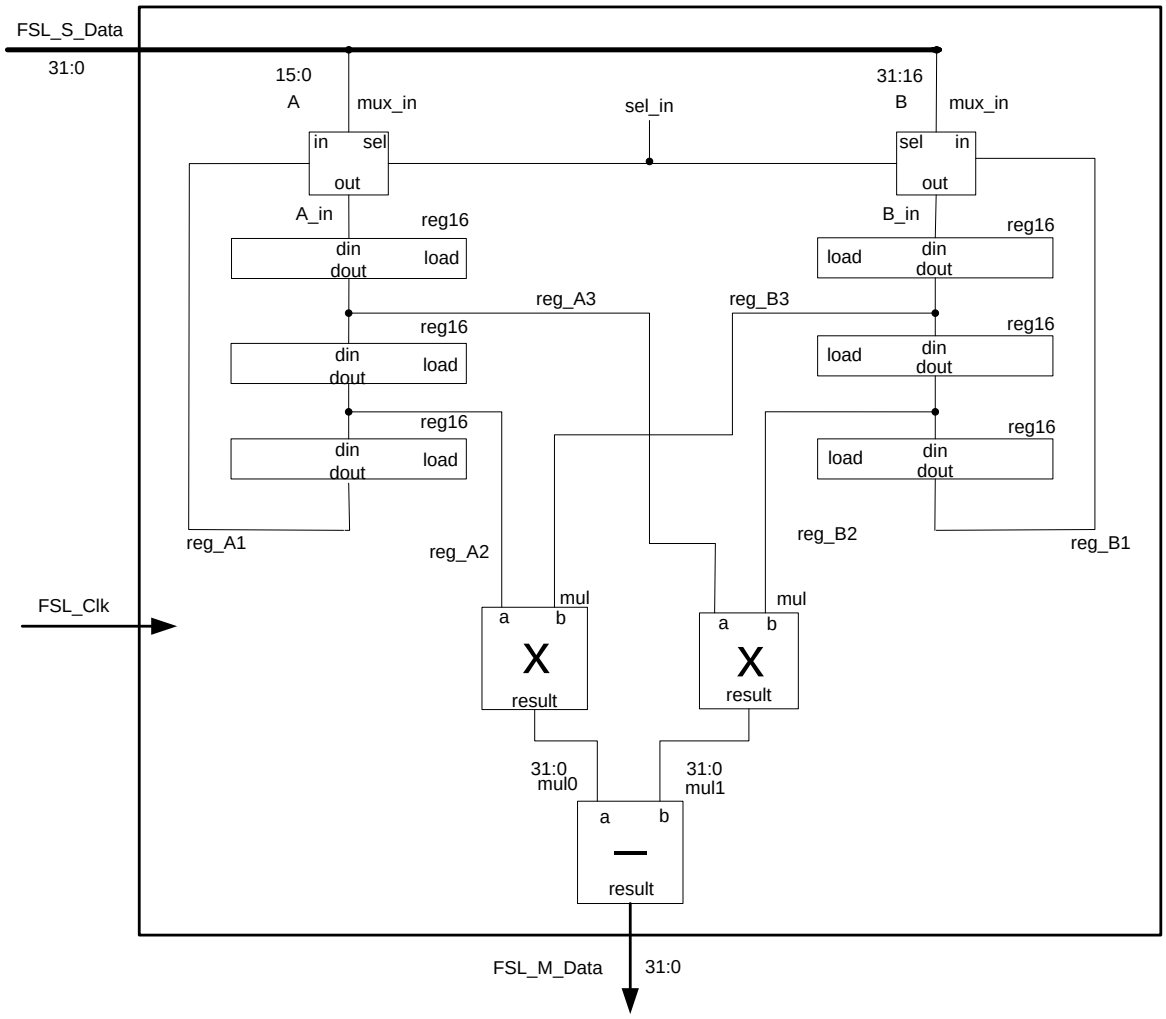


Figura 5.8 Diagrama de bloques del módulo para el cálculo del producto cruz de dos vectores.

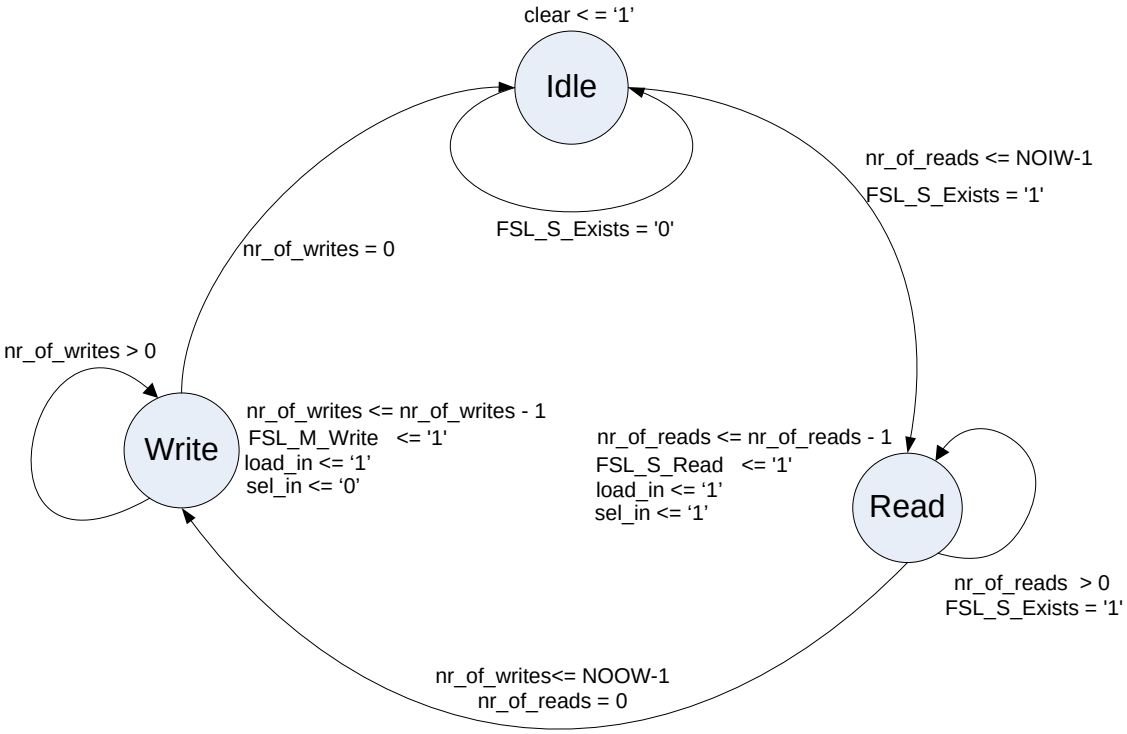


Figura 5.9 Máquina de estados del bloque para calcular el producto cruz de dos vectores.

5.2.3. Implementación del núcleo SinCos para la conversión de Matriz de rotación a Cuaternión

La implementación de este núcleo es utilizada para evaluar las ecuaciones

$$cb = \frac{trace-1}{2} \qquad \text{Ecuación 3-9}$$

$$\theta = \text{acos}(cb) \qquad \text{Ecuación 3-10}$$

$$\eta = \cos\left(\frac{\theta}{2}\right) \qquad \text{Ecuación 3-11}$$

$$\zeta = \sin\left(\frac{\theta}{2}\right) \qquad \text{Ecuación 3-12}$$

Descritas en la sección 3.5.2, donde $-1 \leq trace < 3$ y $-1 \leq cb < 1$.

Dada la cantidad de recursos de cómputo que se requieren para efectuar operaciones trigonométricas en software, se exploró la posibilidad de implementar dichas operaciones en bloques de hardware.

La primera propuesta fue la de construir un bloque de hardware que realizara la función ángulo cuyo coseno *acos()* y uno más que realizara las funciones seno *sin()* y coseno *cos(),* todo lo anterior utilizando el módulo propietario de Xilinx (IP core) CORDIC v.3.00 que es capaz de realizar las operaciones *atan(), sin() y cos()* bajo ciertas restricciones. Dado que este bloque no realiza la función *acos(),* ésta se debe calcular indirectamente a través de la siguiente expresión:

$$\text{acos}(cb) = 2atan\left(\frac{\sqrt{1-cb^2}}{1+cb}\right) \qquad \text{Ecuación 5-5}$$

Para ello se construyó un *core* como se muestra en la figura 5.10, que devuelve un resultado en punto fijo con formato 2QN signado y con una entrada en punto fijo con formato 1QN signado para tamaño de palabra de 16bits.

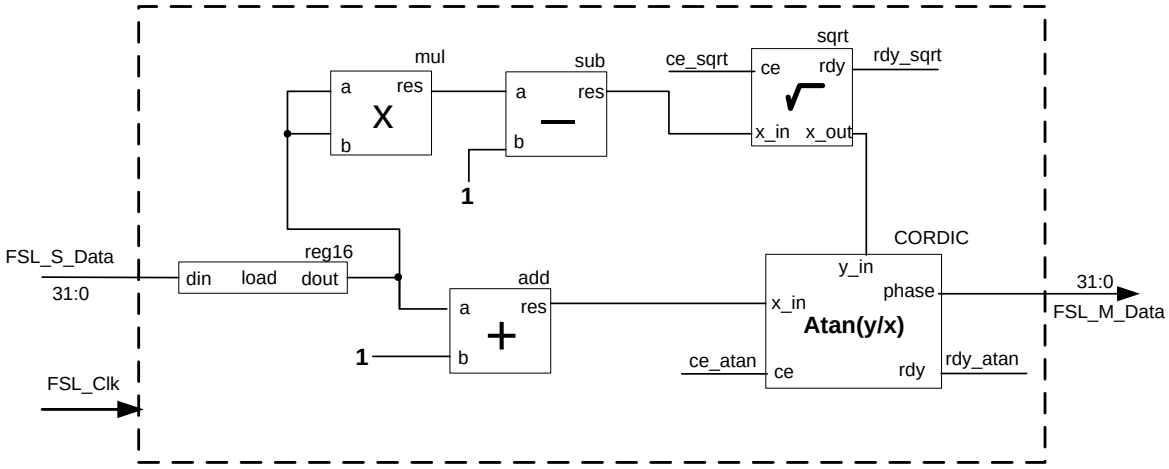


Figura 5.10 Bloque para la el cálculo de la función ángulo cuyo coseno.

La cantidad de recursos del FPGA que utiliza esta arquitectura es muy alta (20% en Slices), a lo que se deberá sumar también los recursos que utilizará el bloque para calcular las funciones *sin()* y *cos()* (12% en Slices).

Por estas razones se decidió utilizar una técnica alternativa de implementación basada en tablas de búsqueda que consiste en evaluar las expresiones y almacenar los resultados fuera del tiempo de ejecución. Para ello, primeramente se redujo el número de ecuaciones mediante substitución a lo siguiente:

$$\eta = \cos\left(\frac{\text{acos}\,(cb)}{2}\right) \qquad \text{Ecuación 5-6}$$

$$\zeta = \sin\left(\frac{\text{acos}\,(cb)}{2}\right) \qquad \text{Ecuación 5-7}$$

El resultado de evaluar las nuevas expresiones (rango de la función) para los valores posibles de cb (dominio de la función) es almacenado en un arreglo en forma de lista. En este caso dos arreglos son declarados en una entidad de VHDL, uno para la función *sin()* y otro para la función *cos().* La funcionalidad de dicha entidad, que posteriormente se convertirá en un bloque de hardware (core), está dada por un proceso que relaciona la entrada a la entidad (argumento de la función) con un índice del arreglo y por lo tanto con el dato almacenado en esa posición, que es finalmente el resultado de evaluar la función y la salida de la entidad.

Dado que se trata de un sistema digital, tanto el rango como el dominio de la función debe discretizarse y digitalizarse. Con el fin de simplificar el proceso que relaciona la entrada con el índice del arreglo, durante la discretización se eligió recorrer el dominio de la función en uno y asignarla a una variable $0 \leq x < 2$ que sería verdaderamente la entrada al bloque de hardware. Por lo tanto:

$$x = cb + 1 \qquad \text{Ecuación 5-8}$$

Con esto, es posible que el argumento de la función se convierta en el propio índice del arreglo, reduciendo así el problema de implementación a un análisis de tamaño del arreglo de acuerdo a las necesidades de precisión y recursos disponibles.

Con un tamaño para la entrada x de 12 bits, se tendrían 4096 posibles valores de entrada y por lo tanto, el mismo número de elementos en el arreglo y salidas del bloque. Considerando el intervalo $0 \leq x < 2$, y un formato de entrada 2QN no signado, se lograría conseguir una resolución para la entrada de 2/4096 (4.8828 x 10-4).

Con un tamaño para la salida x de 12 bits, y por lo tanto para los elementos del arreglo y considerando que el rango de las funciones es $0 \leq \eta < 1$ y $0 \leq \zeta < 1$, y un formato de salida 1QN no signado, se lograría conseguir una resolución de 1/4096, lo que es adecuado para la aplicación.

Con todos estos datos, es ahora posible evaluar las funciones y encontrar los valores de x , η y ζ en los formatos especificados (punto fijo). Para ello, se programó un script de MATLAB que permite evaluar dichas funciones, construir la tabla de búsqueda y entregar un archivo de texto que contiene el arreglo en el formato requerido para insertarlo en un script VHDL.

Dadas las características complementarias de las funciones seno y coseno, no es necesario declarar dos arreglos para cada una de las funciones. A partir de la información del arreglo para la función coseno, es posible obtener el resultado de evaluar la función seno. Esto se logra accediendo al arreglo en orden inverso, es decir, comenzando por el último dato. Esta estrategia ahorra valiosos recursos del FPGA.

La síntesis o transformación de un arreglo de éste tipo descrito en código VHDL a su implementación física en componentes de hardware, se puede realizar básicamente en dos formas: como un bloque de RAM (4096 x 12bits) a través de elementos primitivos del FPGA conocidos como BRAMS, o mediante una agrupación de *slices*. Ambos son recursos muy valiosos, pues son limitados. En el dispositivo FPGA de la tarjeta de desarrollo Spartan 3E, sólo hay 20 bloques de RAM y el procesador embebido ya ocupa el 80% de ellos cuando se le asigna una memoria de programa y de datos de 32Kbytes (8K x 32bits). Un sólo arreglo, ya sea para la función seno o coseno, ocupa 3 bloques de RAM ó 17% de recursos del FPGA en *slices,* por lo que colocar dos arreglos de este tipo es prohibitivo.

En la figura 5.11 se muestran los primeros siete valores y los últimos siete valores de la tabla de búsqueda para la función coseno. También se muestra la simulación, en donde la señal adrr[11:0] de 12 bits representa a la variable x, que además de ser una señal de entrada a la entidad hace las veces de índice de la tabla. La señal de salida cos_out[11:0] se obtiene simplemente buscando el valor en la tabla que corresponde al índice de entrada. La salida sin_out[11:0] se obtiene buscando la posición que corresponde a la resta de la cantidad total de elementos de la tabla menos el índice de entrada.

```
COS_ROM = ( x"0", x"40", x"5b", x"6f", x"80", x"8f", x"9d",
.
.
.
x"ffd", x"ffd", x"ffe", x"ffe", x"fff", x"fff", x"fff");
```

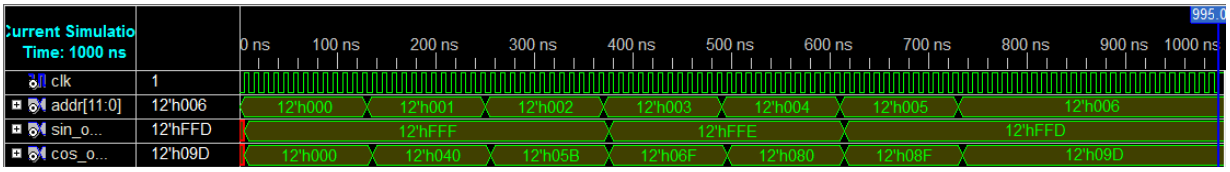


Figura 5.11 Arreglo que resulta de evaluar la función coseno y simulación de la entidad SinCos_lut.vhdl

El resto de las operaciones necesarias para la transformación de la matriz de rotación a un cuaternión se realizan en el *MicroBlaze* debido a que, por los recursos de cálculo que éstas requieren, no ameritan trasladarse a un bloque de hardware.

5.2.4. Integración de los componentes y del programa controlador (driver) para la implementación del método TRIAD

La integración de los bloques que se describieron anteriormente se lleva a cabo a través de la configuración de una arquitectura de hardware como la que se muestra en la figura 5.12 construida mediante la herramienta EDK de Xilinx, en donde todos los bloques están conectados al procesador central mediante enlaces FSL. Además, un programa que se ejecuta dentro del procesador embebido *MicroBlaze* controla el flujo del algoritmo TRIAD, enviando los datos al

bloque adecuado y con el formato adecuado, según la operación que corresponda realizar. Con ello se busca aprovechar los recursos y herramientas de la plataforma FPGA de manera óptima.

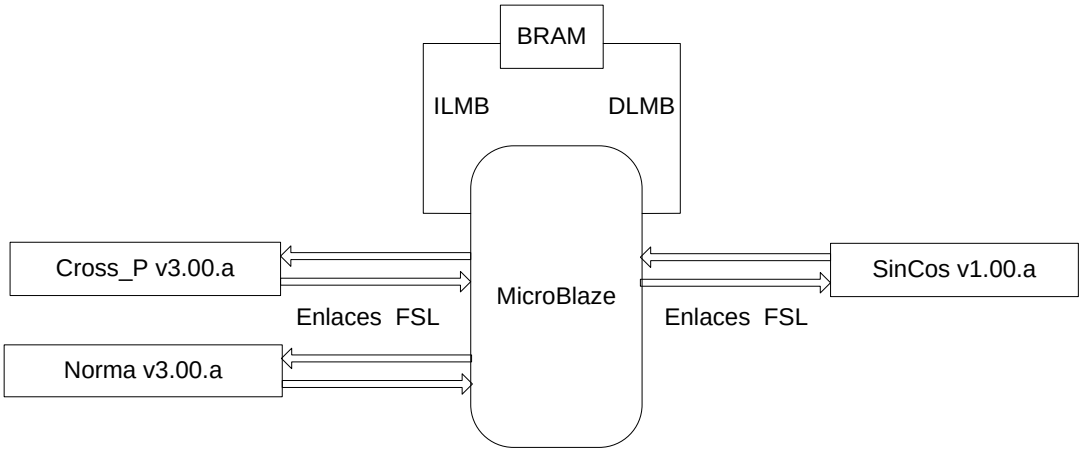


Figura 5.12 Arquitectura de hardware que integra los diversos bloques necesarios para la implementación del método TRIAD.

El flujo del programa es como sigue. Primeramente se establecen las condiciones iniciales que tienen que ver con los vectores de referencia relacionados con el campo magnético (u_R) y campo gravitacional (v_R). Posteriormente se normalizan dichos vectores de referencia mediante el bloque Norma, y además se calculan una serie de vectores ortogonales de referencia (s_1,s_2 y s_3) como lo dicta el método TRIAD utilizando los bloques Cross_P y Norma con las ecuaciones 3-4, 3-5 y 3-6 de la sección 3.5.2.

Dentro de la rutina que maneja la interrupción para la lectura de los sensores, se normalizan los vectores de medición que provienen del magnetómetro y del acelerómetro. A partir de estos vectores se calcula una serie de vectores ortogonales de medición, de forma similar al paso anterior, como lo indica la sección 3.5.2, mediante las ecuaciones 3-1, 3-2 y 3-3.

Una vez que se tienen ambas series de vectores se calculan los elementos de la matriz de rotación mediante la ecuación 3-7, la cual es evaluada en software utilizando aritmética de punto fijo.

Sin embargo, el cálculo no se lleva a cabo tal cual aparece en la expresión. En primer lugar se calculan los elementos de la diagonal principal y se suman, después se calcula el resto de los elementos y se agrupan en un vector v mediante los algoritmos que se presentan a continuación. Posteriormente, el vector v se normaliza para formar un vector unitario λ mediante el *core* Norma. Esta forma de evaluar los elementos de la matriz de rotación simplifica la conversión de dicha matriz a un cuaternión.

```
trace = 0;
for i = 1:1:3
  trace = trace + r1(i)*s1(i)+r2(i)*s2(i)+r3(i)*s3(i);
end

v = [0 0 0];
v(1) = r1(2)*s1(3)+r2(2)*s2(3)+r3(2)*s3(3)-r1(3)*s1(2)-r2(3)*s2(2)-r3(3)*s3(2);
v(2) = r1(3)*s1(1)+r2(3)*s2(1)+r3(3)*s3(1)-r1(1)*s1(3)-r2(1)*s2(3)-r3(1)*s3(3);
v(3) = r1(1)*s1(2)+r2(1)*s2(2)+r3(1)*s3(2)-r1(2)*s1(1)-r2(2)*s2(1)-r3(2)*s3(1);
```

A partir de la suma de los elementos de la diagonal principal ($trace$) se calculan los ángulos η y ζ mediante el bloque Matriz de rotación a Cuaternión. Dichos ángulos, junto con el vector unitario λ se utilizan para formar finalmente los elementos del cuaternión de medición $q_{gn} = (\eta\ \epsilon_1\ \epsilon_2\ \ \epsilon_3)$, mediante las expresiones 3-13, 3-14 y 3-15.

El flujo del programa para el cálculo del método TRIAD puede resumirse en el diagrama de flujo de la figura 5.13. Con el fin de probar y validar el funcionamiento de dicho programa, el bloque de envío del cuaternión al filtro de Kalman es substituido por un bloque que envía el cuaternión a la PC, en un esquema similar al de la validación del programa de adquisición de datos de sensores.

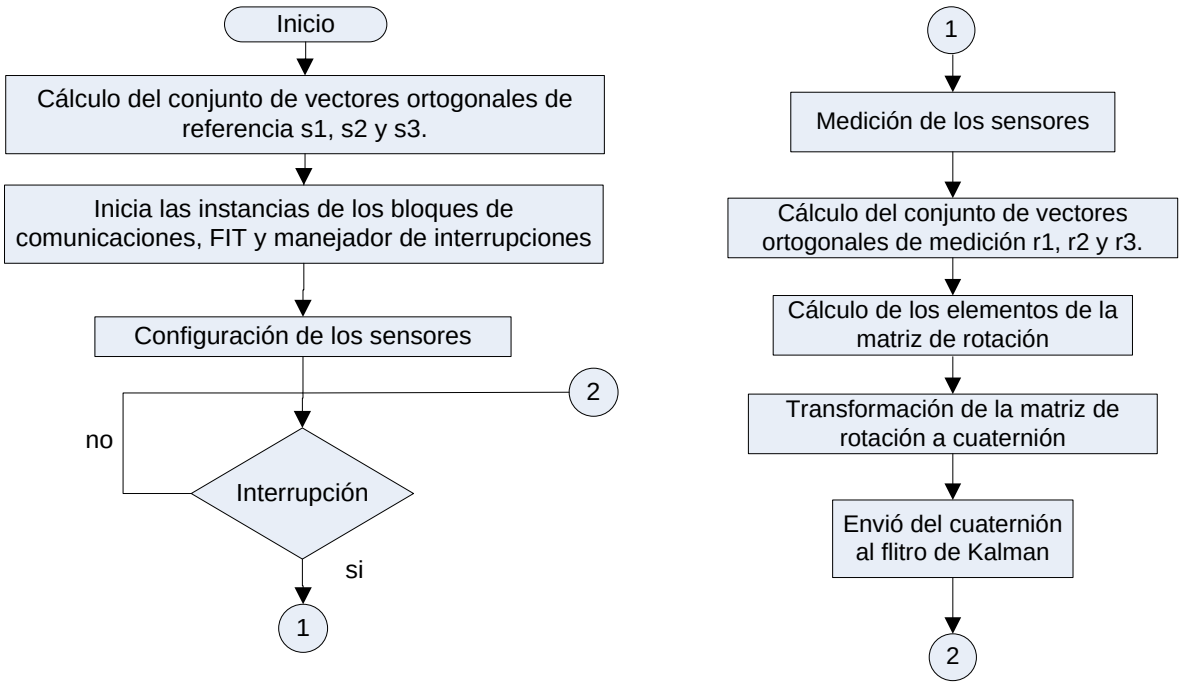


Figura 5.13 Diagrama de flujo del programa para el cálculo del cuaternión de medición a través del método TRAD.

5.3. Implementación del Filtro de Kalman

La implementación del Filtro de Kalman se llevó a cabo en un programa que es ejecutado por el procesador embebido *MicroBlaze.*

Como se describió en la sección 3.4.2 y en la figura 3.11, el Filtro de Kalman se puede descomponer en tres bloques, en los que esencialmente se resuelven una serie de ecuaciones diferenciales (52).

5.3.1. Cálculo de la matriz de error de covarianzas y etapa de propagación

Por conveniencia de la implementación, el primer bloque que es procesado es el correspondiente al cálculo de la matriz de error de covarianzas mediante la expresión:

$$\dot{P} = AP + PA^T - PH^T R^{-1} HP \qquad \text{Ecuación 5-9}$$

$$P(0) = P_0$$

La ecuación implementada es una versión reducida de la ecuación 3-21, que se obtiene al aplicar las siguientes consideraciones y simplificaciones. En primer lugar, la matriz de covarianza del proceso Q, es un elemento nulo. La matriz de covarianzas de las mediciones R, es una matriz diagonal que contiene la covarianza del cuaternión de medición y la covarianza de las mediciones de los giróscopos, por lo que la inversa de esta matriz puede calcularse con facilidad. La matriz A, que substituye a la función F de la ecuación 3-21, se forma con elementos del cuaternión estimado y la velocidad angular estimada en la iteración anterior del algoritmo. La matriz P_0 (condición inicial de la matriz de error de covarianzas) es una matriz diagonal cuyos elementos son iguales a 0.001. La matriz H, que sustituye a la función H, se define como sigue:

$$\mathrm{H} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Dadas estas condiciones, el producto $PH^T R^{-1} H$ y la ecuación en general pueden simplificarse, por lo que el número de operaciones aritméticas también es reducido considerablemente.

$$B = PH^T R^{-1} H = \begin{bmatrix} 0 & P_{12}R_{11}^{-1} & P_{13}R_{22}^{-1} & P_{14}R_{33}^{-1} & P_{15}R_{44}^{-1} & P_{16}R_{55}^{-1} & P_{17}R_{66}^{-1} \\ 0 & P_{22}R_{11}^{-1} & P_{23}R_{22}^{-1} & P_{24}R_{33}^{-1} & P_{15}R_{44}^{-1} & P_{16}R_{55}^{-1} & P_{17}R_{66}^{-1} \\ 0 & P_{32}R_{11}^{-1} & P_{33}R_{22}^{-1} & P_{34}R_{33}^{-1} & P_{15}R_{44}^{-1} & P_{16}R_{55}^{-1} & P_{17}R_{66}^{-1} \\ 0 & P_{42}R_{11}^{-1} & P_{43}R_{22}^{-1} & P_{44}R_{33}^{-1} & P_{15}R_{44}^{-1} & P_{16}R_{55}^{-1} & P_{17}R_{66}^{-1} \\ 0 & P_{52}R_{11}^{-1} & P_{53}R_{22}^{-1} & P_{54}R_{33}^{-1} & P_{15}R_{44}^{-1} & P_{16}R_{55}^{-1} & P_{17}R_{66}^{-1} \\ 0 & P_{62}R_{11}^{-1} & P_{63}R_{22}^{-1} & P_{64}R_{33}^{-1} & P_{15}R_{44}^{-1} & P_{16}R_{55}^{-1} & P_{17}R_{66}^{-1} \\ 0 & P_{72}R_{11}^{-1} & P_{73}R_{22}^{-1} & P_{74}R_{33}^{-1} & P_{15}R_{44}^{-1} & P_{16}R_{55}^{-1} & P_{17}R_{66}^{-1} \end{bmatrix}$$

$$\dot{P} = AP + PA^T - BP \qquad \text{Ecuación 5-10}$$

$$P(0) = P_0$$

De esta forma ya sólo es necesario realizar 3 multiplicaciones de matrices de 7 X 7 y dos sumas de matrices que se llevan a cabo mediante una función dedicada a este propósito. Además de la integración que se realiza de la siguiente forma:

$$P_{[n]} = \dot{P}_{[n]} * t_s + P_{[n-1]} \qquad \text{Ecuación 5-11}$$

Donde el subíndice n indica el número de iteración del algoritmo y t_s es el tiempo de muestreo.

La matriz B puede generarse con facilidad a través del siguiente algoritmo, en donde la matriz diagonal R se reduce a un arreglo de seis elementos:

```
for i=1:7
   for j=2:7
      B(i,j) = P(i,j)*1/R(j-1);
   end
end
```

5.3.2. Cálculo de la ganancia de Kalman

El segundo bloque procesado es el encargado de calcular la ganancia de Kalman a través de la ecuación

$$K = PH^T R^{-1} \qquad \text{Ecuación 3-20}$$

Las matrices H y R son idénticas a las del primer bloque, sin embargo la matriz P involucrada en esta ecuación, es el resultado del primer bloque (resultado de la ecuación de covarianza).No es necesario realizar la multiplicación de estas tres matrices en tiempo de ejecución. Esta misma operación, dadas las características de las matrices H y R, pueden reducirse a lo siguiente:

$$K = \begin{bmatrix} P_{12}R_{11}^{-1} & P_{13}R_{22}^{-1} & P_{14}R_{33}^{-1} & P_{15}R_{44}^{-1} & P_{16}R_{55}^{-1} & P_{17}R_{66}^{-1} \\ P_{22}R_{11}^{-1} & P_{23}R_{22}^{-1} & P_{24}R_{33}^{-1} & P_{15}R_{44}^{-1} & P_{16}R_{55}^{-1} & P_{17}R_{66}^{-1} \\ P_{32}R_{11}^{-1} & P_{33}R_{22}^{-1} & P_{34}R_{33}^{-1} & P_{15}R_{44}^{-1} & P_{16}R_{55}^{-1} & P_{17}R_{66}^{-1} \\ P_{42}R_{11}^{-1} & P_{43}R_{22}^{-1} & P_{44}R_{33}^{-1} & P_{15}R_{44}^{-1} & P_{16}R_{55}^{-1} & P_{17}R_{66}^{-1} \\ P_{52}R_{11}^{-1} & P_{53}R_{22}^{-1} & P_{54}R_{33}^{-1} & P_{15}R_{44}^{-1} & P_{16}R_{55}^{-1} & P_{17}R_{66}^{-1} \\ P_{62}R_{11}^{-1} & P_{63}R_{22}^{-1} & P_{64}R_{33}^{-1} & P_{15}R_{44}^{-1} & P_{16}R_{55}^{-1} & P_{17}R_{66}^{-1} \\ P_{72}R_{11}^{-1} & P_{73}R_{22}^{-1} & P_{74}R_{33}^{-1} & P_{15}R_{44}^{-1} & P_{16}R_{55}^{-1} & P_{17}R_{66}^{-1} \end{bmatrix}$$

Y puede generarse fácilmente mediante el siguiente algoritmo, en donde la matriz diagonal R se reduce a un arreglo de seis elementos:

```
for i=1:7
   for j=1:6
      K(i,j) = P(i,j+1)*1/R(j);
   end
end
```

5.3.3. Proceso de estimación e innovación

El proceso de innovación, descrito en la sección 3.5.2 mediante la ecuación 3-19, se reduce a encontrar la diferencia entre los componentes del cuaternión de medición y las mediciones de los giróscopos con los componentes del cuaternión estimado y la velocidad angular estimada:

$$\nu = \begin{bmatrix} \epsilon_{1\_gn} \\ \epsilon_{2\_gn} \\ \epsilon_{3\_gn} \\ \Omega_1 \\ \Omega_2 \\ \Omega_3 \end{bmatrix} - \begin{bmatrix} \widehat{\epsilon_1} \\ \widehat{\epsilon_2} \\ \widehat{\epsilon_3} \\ \widehat{\Omega_1} \\ \widehat{\Omega_2} \\ \widehat{\Omega_3} \end{bmatrix} \qquad \text{Ecuación 5-12}$$

Finalmente el proceso de estimación representado por las funciones 5-13 y 5-14, se redujo a un conjunto de operaciones aritméticas como multiplicaciones y sumas que relacionan el vector ν, los elementos del cuaternión estimado y la velocidad angular estimada en la iteración anterior, y la ganancia de Kalman K, con el fin de realizar la nueva estimación de los estados del sistema expresados como la derivada del cuaternión estimado y la derivada de la velocidad angular estimada.

$$\hat{\dot{\Omega}} = f(\mathrm{K}, \nu) \qquad \text{Ecuación 5-13}$$

$$\hat{\dot{\mathrm{q}}} = f\left(\hat{\mathrm{q}}, \mathrm{K}, \nu, \widehat{\Omega}\right) \qquad \text{Ecuación 5-14}$$

Dadas las características numéricas de los datos en cuanto a su rango y las necesidades de resolución, el filtro de Kalman fue implementado mediante aritmética de punto flotante. Aunque esto aumenta la memoria de programa requerida considerablemente, también aumenta la precisión de los resultados y la flexibilidad para realizar modificaciones al algoritmo.

Al elegir la representación en punto flotante, ya no es posible reutilizar el módulo de normalización de cuaterniones que se utilizó en el método TRIAD, por lo que fue necesario encontrar la forma de realizar esta operación de manera eficiente mediante elementos de software únicamente.

En la sección 5.2.1 se describe el algoritmo para normalizar un vector de n elementos. Las operaciones más complicadas en este algoritmo son el cálculo de la norma del vector, específicamente la raíz cuadrada de la suma de los cuadrados de los elementos y las divisiones de los elementos entre la norma. Dadas las capacidades limitadas del procesador embebido no es posible utilizar funciones predefinidas en librerías de lenguaje ANSI C como la función sqrt() de la librería math.h.

Para resolver este problema, se utilizó un algoritmo novedoso conocido como Raíz cuadrada inversa rápida (Fast Inverse Square Root) desarrollado para solucionar problemas de normalización de vectores en videojuegos sobre procesadores de 32 bits con el formato de punto flotante estándar IEEE 754 (90) (91) el cual se presenta a continuación:

```
float InvSqrt (float x){
     float xhalf = 0.5f*x;
     int i = *(int*)&x;
     i = 0x5f3759df - (i>>1);
     x = *(float*)&i;
     x = x*(1.5f - xhalf*x*x);
     return x;
}
```

5.14 Algoritmo Raíz cuadrada inversa rápida.

5.4. Integración de los bloques para la implementación de la computadora de navegación

Con la integración de los subsistemas de adquisición de datos de sensores, método TRIAD y Filtro de Kalman, se logra conformar todas las funciones de la computadora de navegación.

La arquitectura que se muestra en forma de diagrama de bloques en la figura 5.14 incluye todos los núcleos de hardware (cores) utilizados en cada uno de los subsistemas mencionados y descritos en secciones anteriores.

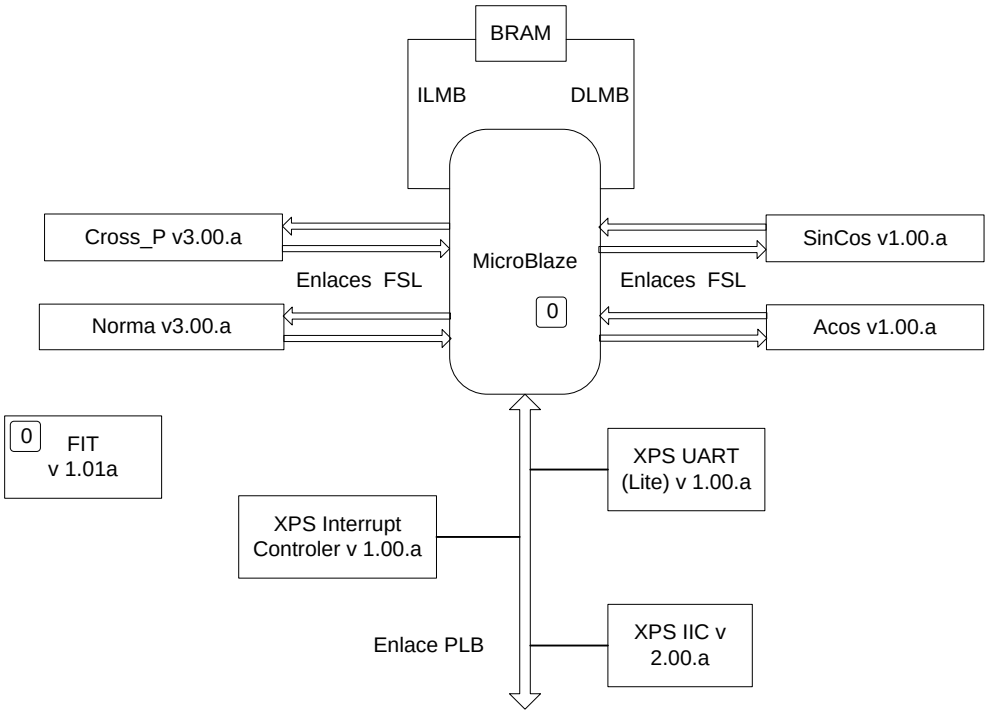


Figura 5.15 Núcleos que componen la computadora de navegación.

5.5. Propuesta de un núcleo para el cálculo de multiplicación de matrices mediante arreglos sistólicos

Aunque se logró reducir las expresiones en el Filtro de Kalman, aún restan algunas operaciones complejas que ocupan importantes recursos de memoria del procesador embebido, así como tiempo de ejecución, como es el caso de la multiplicación de matrices. Es por ello que como parte de la presente tesis se propone una arquitectura capaz de reducir dichas operaciones mediante la construcción de un núcleo dedicado exclusivamente a este propósito mediante componentes de hardware.

5.5.1. Descripción

Se creó un núcleo de hardware (core) capaz de realizar una multiplicación de matrices de un orden o tamaño particular. El IP recibe los elementos de las matrices desde el procesador embebido (MicroBlaze) a través de un enlace FSL y envía el resultado de vuelta a través de otro enlace FSL distinto. La secuencia con la que los datos son enviados o recibidos por el programa residente en el *MicroBlaze* debe ser consistente con el funcionamiento del núcleo.

Dicho funcionamiento está regido por una máquina de estados descrita en un proceso dentro del código VHDL. La estructura del núcleo, es decir, los componentes de los que está

conformado y la manera en que éstos se interconectan está descrita en el mismo código VHDL, es decir, ambas descripciones conviven en el mismo módulo. Esta forma de describir el hardware aprovecha dos niveles de abstracción, el de la estructura y el de comportamiento, lo cual permite que el sistema entero sea más fácil de diseñar, manejar y modificar.

5.5.2. Estructura

Desde el punto de vista estructural, el núcleo se puede entender dividiéndolo en los siguientes bloques:

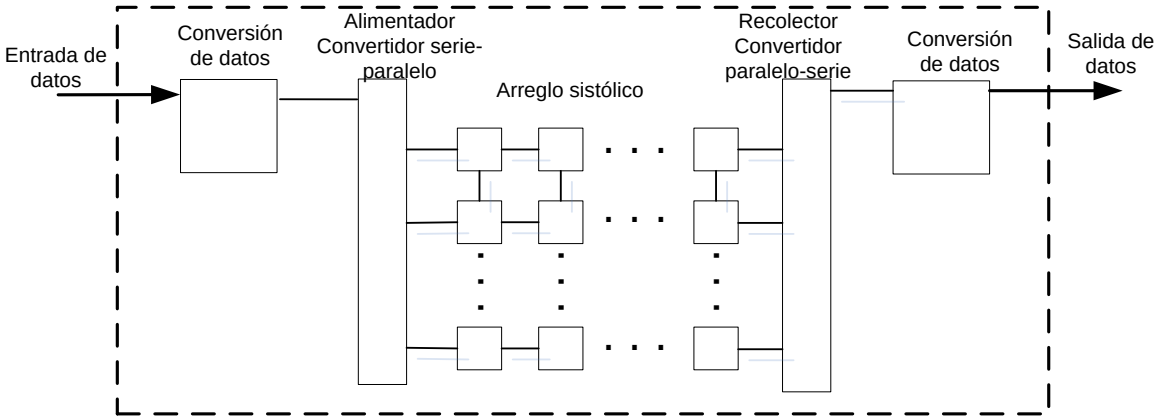


Figura 5.16 Estructura del núcleo para la multiplicación de matrices.

El bloque de conversión de datos a la entrada del núcleo transforma los datos numéricos de punto flotante a punto fijo, pues en el interior del núcleo todas las operaciones se llevan a cabo en punto fijo. El bloque de conversión de datos a la salida transforma los datos numéricos de punto fijo a punto flotante, pues el procesador embebido *MicroBlaze* facilita el manejo aritmético en este formato gracias a su unidad de punto flotante.

El Alimentador es un convertidor serie-paralelo que recibe la cadena de datos del bloque de conversión de datos en serie y los envía al arreglo sistólico en paralelo a través de sus distintos canales de salida. El Recolector realiza la operación inversa, mediante una conversión paralelo-serie.

El arreglo sistólico es el encargado de realizar las operaciones aritméticas con los datos recibidos, tal como se ha descrito en secciones anteriores.

5.5.3. Implementación

Para implementar los bloques anteriormente mencionados es necesario recurrir a diferentes elementos básicos que, como el propio IP, pueden estar descritos por su comportamiento o por su estructura. A través de la interconexión de dichos elementos se forman elementos más

complejos que forman los bloques estructurales con los cuales podemos describir la arquitectura general del núcleo.

a) Convertidor de datos: Punto flotante a Punto fijo y Punto fijo a Punto flotante

Para implementar este bloque se utilizó el núcleo propietario de XILINX (IP Core) *Floating-point* versión 3.0, el cual permite realizar operaciones aritméticas básicas con números en punto flotante, así como la transformación de este formato a punto fijo y viceversa, figura 5.17.

Dicho módulo contiene gran número de características y parámetros que pueden ser establecidas a conveniencia del usuario mediante la herramienta *XILINX CORE Generator.* También posee una variedad de señales de entrada y salida, de las cuales es posible indicar si serán o no utilizadas, y de esta forma simplificar el módulo y ahorrar recursos.

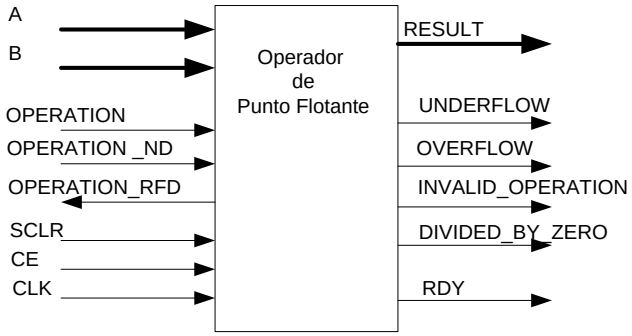


Figura 5.17 IP core para la conversión de números de punto fijo a punto flotante y viceversa.

Para poder utilizar este módulo, es necesario añadir una nueva fuente al proyecto, seleccionar la opción IP (Core Generator & Architecture Wizard) y escoger el módulo *Floating-point* v3.0 de la librería de Funciones Matemáticas.

Una vez realizado lo anterior, tendremos acceso al asistente para la personalización del núcleo. El primer paso de la configuración consiste en indicar la operación que realizará el núcleo, en este caso será *Float-to-fixed* para el componente fl2fx y *Fixed-to-float* para el componente fx2fl (figura 5.18).

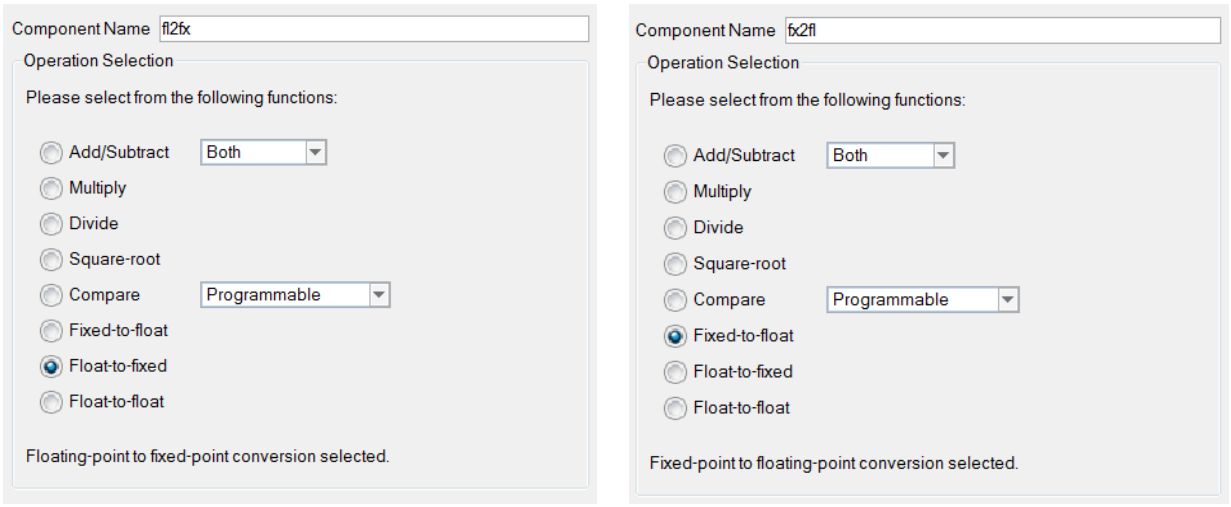


Figura 5.18 Configuración del convertidor numérico. Paso 1.

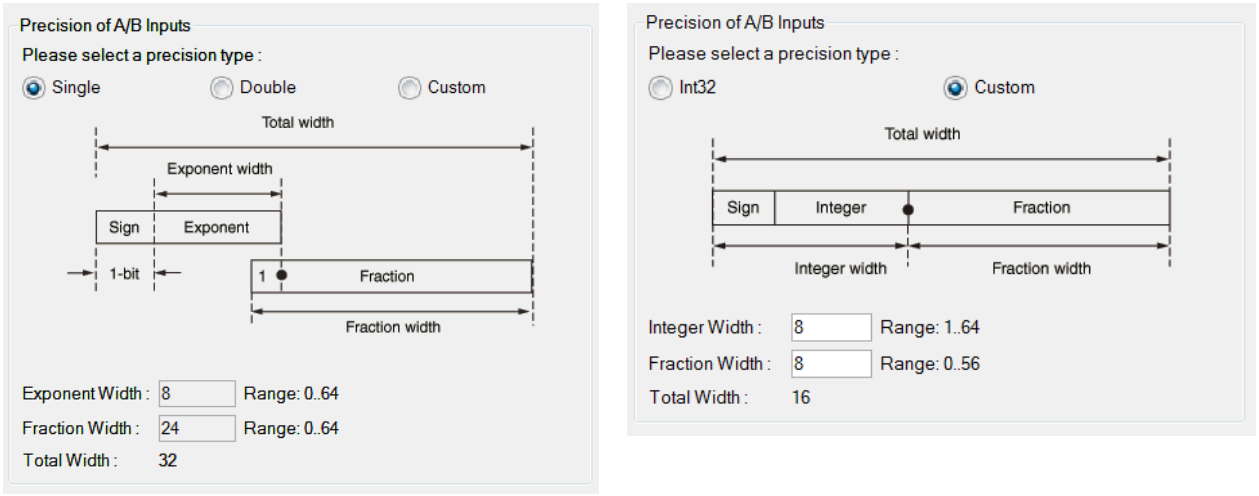


Figura 5.19 Configuración del convertidor numérico. Paso 2.

En el siguiente paso (paso 2), figura 5.19, se establece el formato del dato que ingresa al bloque. El formato de números en punto flotante utilizado está basado en el estándar IEEE-754, pues tanto el *MicroBlaze* como el IP utilizan este mismo estándar, especificando un tamaño de 8 bits para la parte del exponente y un tamaño de 24 bits para la parte de la mantisa o fracción. El formato de números en punto fijo se ha elegido de 8 bits para la parte entera y 8 bits para la parte fraccional, sin embargo, estas cantidades puede ser modificadas según convenga para nuestra aplicación, procurando que el tamaño total no rebase los 16 bits, pues los bloques aritméticos están diseñados para operar números con esta precisión.

En seguida (paso 3), figura 5.20, se especifica el formato de salida, de manera similar a como se hizo en el paso anterior.

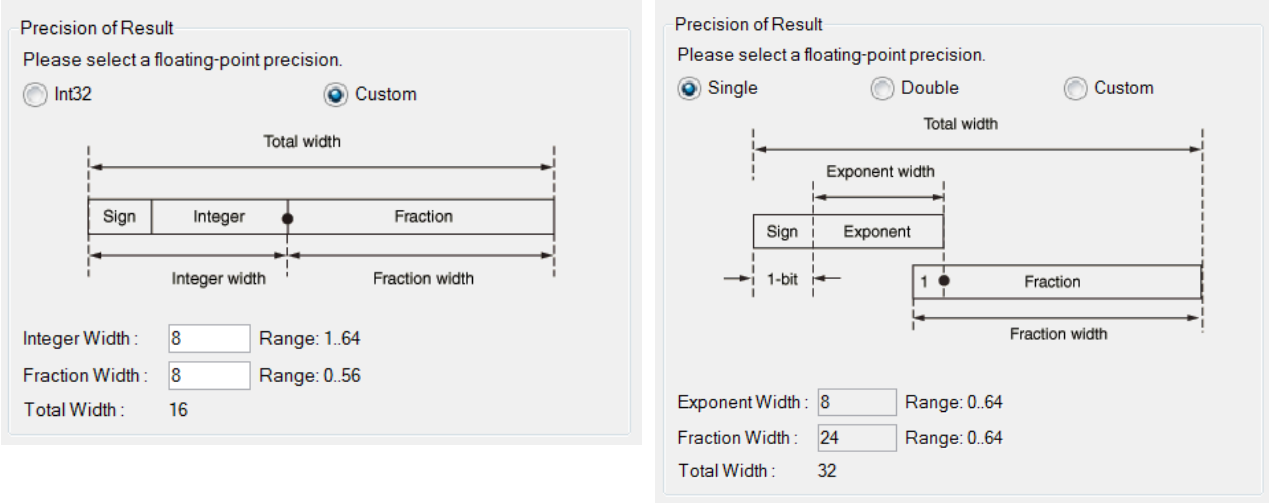


Figura 5.20 Configuración del convertidor numérico. Paso 3.

Figura 5.21 Configuración del convertidor numérico. Paso 4.

Finalmente (paso 4), figura 5.21, se indica que señales se utilizarán, además del dato numérico de entrada y salida. Para el caso de ambos convertidores sólo se requiere conocer el momento en que el bloque ha terminado de realizar una transformación mediante la señal de salida RDY (ready) y también se necesita indicarle al bloque que hemos colocado un nuevo dato en su entrada y que se desea realizar otra operación, mediante la señal de entrada OPERATION_ND (new data). Es importante hacer notar que la señal RDY, se puedo ocupar también para conocer el momento en que el bloque esté disponible para una siguiente operación.

b) Alimentador

Se consideraron dos opciones para implementar este bloque: un arreglo de registros o un de-multiplexor. Se eligió la primera opción pues resultó ser la más eficiente en términos de recursos del FPGA y la más sencilla de manejar.

La idea es acumular los datos de entrada en una pila que se llena conforme los datos arriban, también que los almacene temporalmente y que, finalmente, una vez que la pila esté llena, sea posible descargar los datos en paralelo cuando sean requeridos, tal como intenta describirlo la figura 5.22.

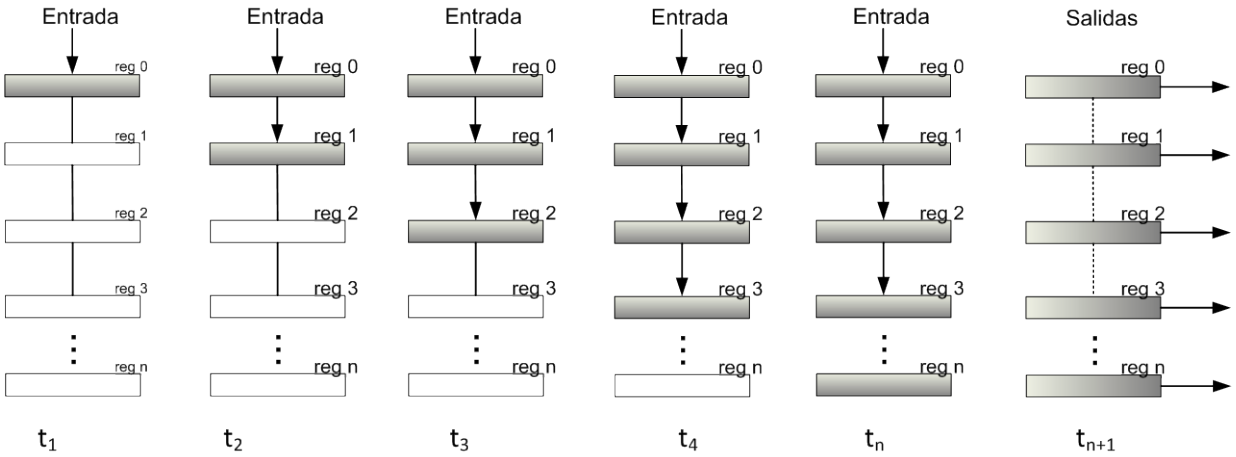


Figura 5.22 Esquema de funcionamiento del alimentador (convertidor serie-paralelo).

Para la implementación de esta pila se crearon los componentes reg16 y reg32, los cuales se utilizan para almacenar temporalmente un dato de 16 o 32 bits, según sea el caso.

En realidad, cada componente es un arreglo de 16 o 32 Flip-Flops tipo D, cuya salida de datos (din) adquiere el valor de la entrada de datos (dout) cuando la señal de reloj transita de un valor lógico bajo ‘0’ a un valor lógico alto ‘1’, pero únicamente cuando la entrada de habilitación (load) está en un valor lógico alto ‘1’.

Figura 5.23 Registro para el almacenamiento temporal de datos.

c) Arreglo Sistólico

Está conformado por un arreglo de elementos de procesamiento (PE), los cuales se encargan de realizar las operaciones aritméticas.

Aunque se diseñaron diversos PE con distintas características y funcionalidades, en su forma más simple, cada elemento de procesamiento está conformado por un multiplicador binario de 16 bits, un sumador binario de 32 bits y un registro de 32 bits (reg32), figura 5.24.

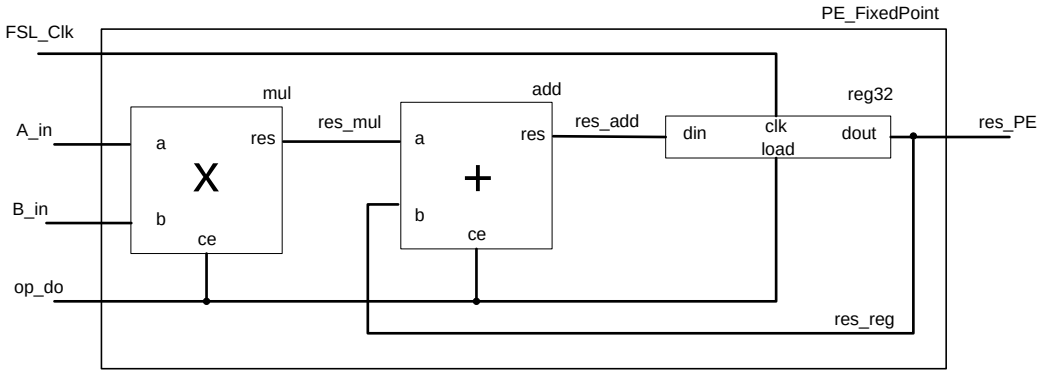
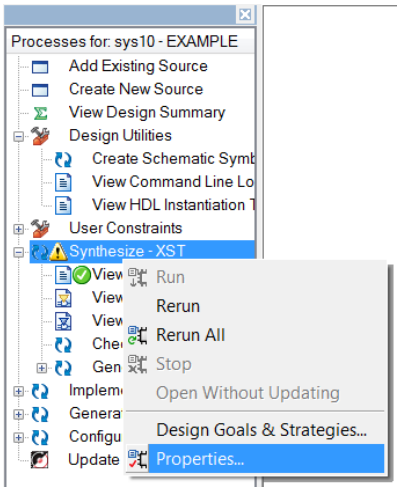


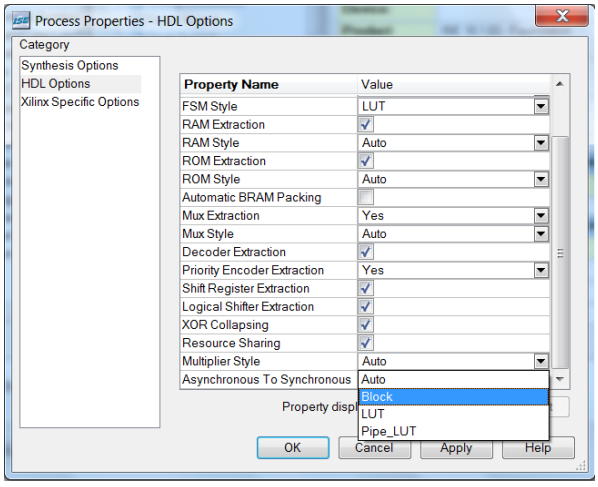
Figura 5.24 Elemento de procesamiento.

El resultado de la multiplicación, que se trata de un número de 32 bits, es sumado con el resultado de la operación anterior, el cual había quedado almacenado en el registro de 32 bits. En este mismo registro es necesario guardar un dato con valor de 0 (cero) antes de realizar la primera operación. Para este propósito, el componente reg32 tiene una señal *clear* que reinicia los *Flip-Flops* de los que está formado.

Para asegurarse de que el multiplicador que se describe mediante VHDL sea sintetizado en un bloque multiplicador dedicado, con los cuales cuenta el FPGA utilizado, es necesario indicárselo a la herramienta de síntesis. En ISE esto se logra seleccionando la opción de Propiedades del proceso de Síntesis que se encuentra en la ventana de Procesos, ver figura 5.25 (a) del Navegador de Proyectos. Una vez en el menú de Propiedades del Proceso, elegimos la categoría de Opciones de HDL. Se localiza la propiedad llamada Estilo de Multiplicador y se cambia su valor a Block si se quiere utilizar el bloque de multiplicación embebido o a LUT si se requiere que el multiplicador binario sea sintetizado en arreglo de compuertas lógicas, ver figura 5.25 (b).



Paso a. Abrir ventana de Propiedades del Proceso de síntesis.



Paso b. Seleccionar el método de implementación.

Figura 5.25 Configuración de la herramienta de síntesis XST del entorno de desarrollo ISE.

d) Recolector

La función recolección de datos se podría realizar mediante un multiplexor que seleccione cada uno de los elementos del arreglo sistólico y de esta forma leer todos los resultados, sin embargo esta opción resultaría muy poco eficiente cuando se tengan matrices de gran tamaño. Una forma más eficiente es utilizar dos multiplexores, uno para seleccionar las columnas del arreglo sistólico y otro para seleccionar cada elemento de la columna seleccionada.

Partiendo de esta última idea la implementación de este bloque se realizó a través de dos componentes muy similares basados en multiplexores los cuales realizan una conversión paralelo-serie.

En el ejemplo de la figura 5.26, se observa una columna de un arreglo sistólico de tres renglones formada por componentes PE_FixedPoint conectados a componentes del tipo reg32 y mux_reg_sal. Este arreglo es un recurso que se utilizó para simplificar la lectura de los resultados de los PE's.

Los componentes PE_FixedPoint son los elementos de procesamiento de los cuales se quiere obtener su resultado. El componente mux_reg_sal es un multiplexor formado por dos compuertas AND y una compuerta OR, con el que es posible seleccionar dos canales de entrada mediante la señal binaria de entrada reg_sal_sel. Los componentes reg32 forman una pila muy similar al bloque Alimentador.

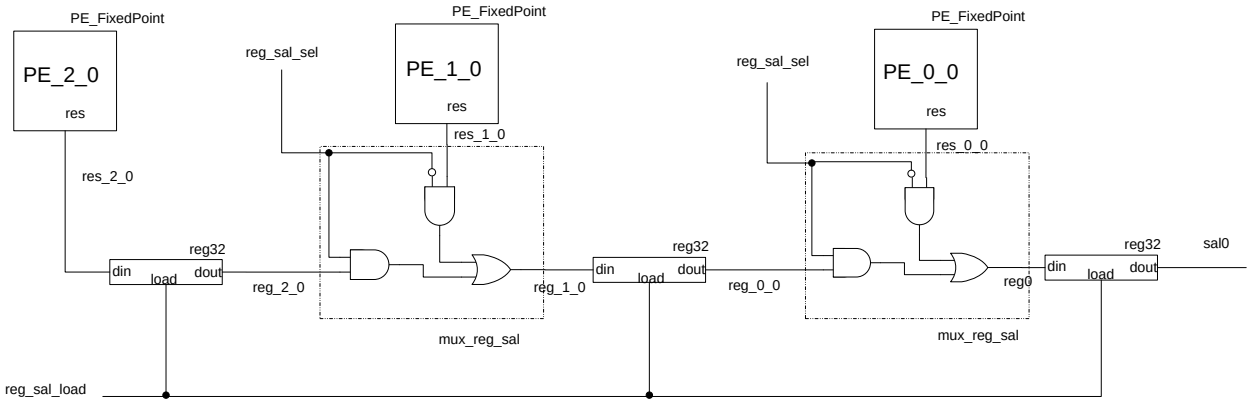


Figura 5.26 Esquema de los elementos que conforman el recolector.

En una primera instancia, con la señal reg_sal_load en un nivel lógico alto '1' y reg_sal_sel en bajo '0' se llena la pila con los resultados de los PE's arribando en paralelo. En instancias posteriores, con la señal reg_sal_load en un nivel lógico bajo '0' y reg_sal_sel en alto '1', la pila es vaciada a través de la salida sal0 entregando los datos en serie.

La figura 5.27 generaliza el funcionamiento de la pila de salida para una sola columna.

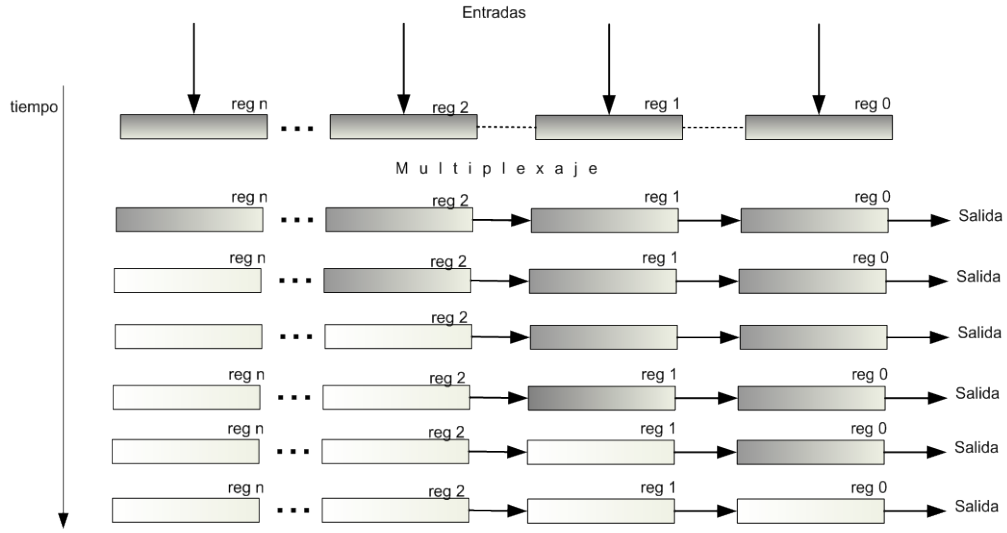


Figura 5.27 Esquema del funcionamiento del recolector (convertidor paralelo-serie).

Hasta aquí se han recolectado los datos de una columna, ahora es necesario recolectar los datos del resto de las columnas. El multiplexor mux_sal permite seleccionar las columnas del arreglo de PE's, por lo que tendrá tantas entradas como columnas haya en el arreglo sistólico, pero sólo una salida. La señal de entrada mux-sel, se utiliza para seleccionar el canal de entrada. El ejemplo de la figura 5.28 se refiere a un multiplexor para un arreglo sistólico con tres columnas.

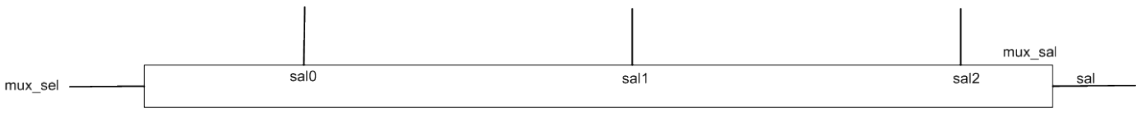


Figura 5.28 Multiplexor de salida.

Uniendo todos estos elementos finalmente se obtiene una arquitectura capaz de multiplicar dos matrices, sin embargo, aún es necesario definir la forma en que los datos deberán fluir a través de todos estos componentes en conjunto, que se muestran figura 5.29.

Funcionamiento

El aspecto funcional de la estructura descrita está definido por una máquina de estados. Dicha máquina se encarga de enviar e interpretar las señales de control de los diversos componentes, así como de manejar la forma y los tiempos en que los datos fluyen a través de ellos.

La máquina de estados controla tanto la comunicación con el procesador embebido a través de los enlaces FSL, como la operación propia del núcleo que realiza la multiplicación. Ambas funciones dependen una de la otra y están fusionadas en el mismo proceso dentro del código VHDL, lo que simplifica su implementación en términos de recursos del FPGA.

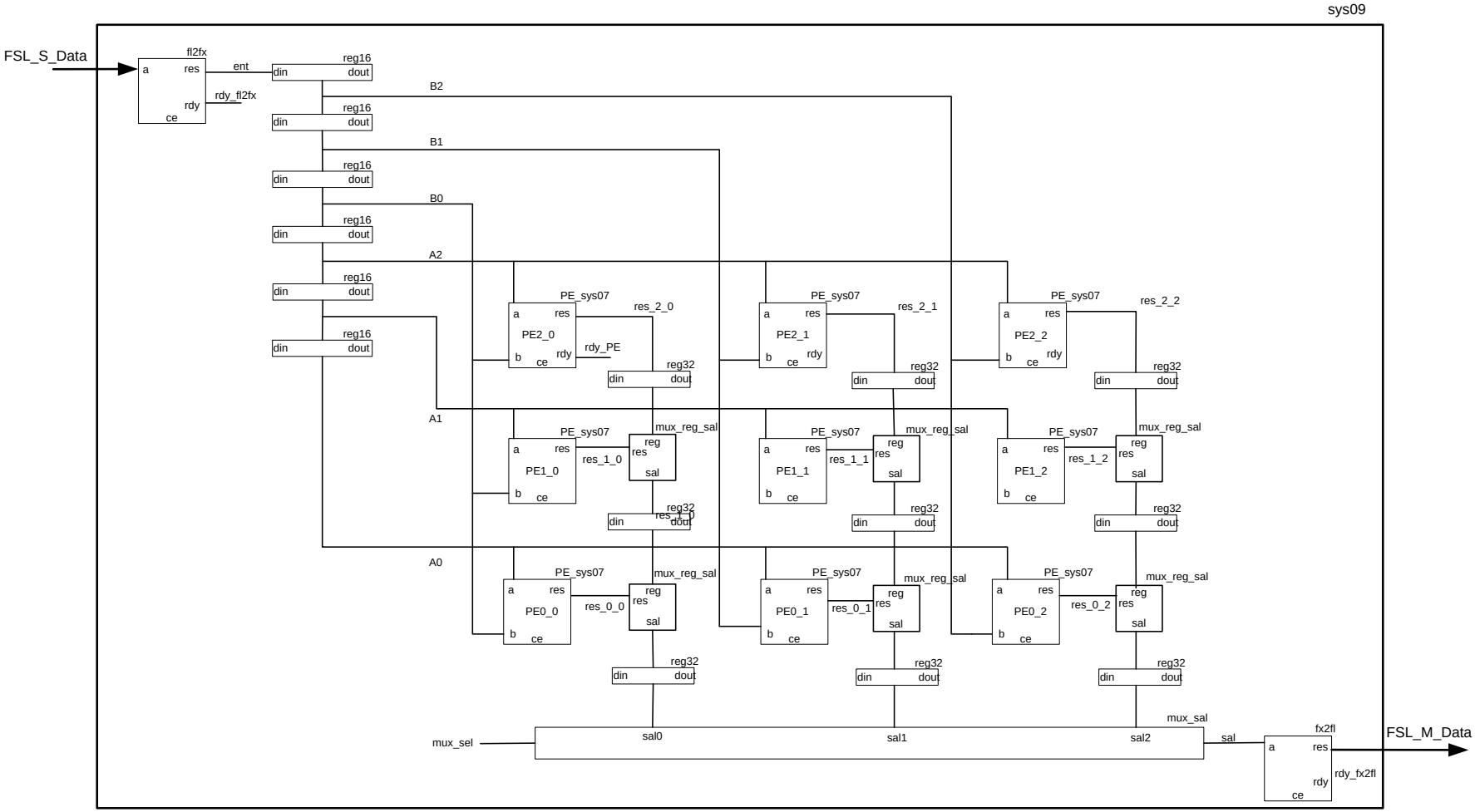


Figura 5.29 Núcleo para la multiplicación de dos matrices cuadradas de 3x3.

El entorno de desarrollo XILINX XPS provee de una herramienta que facilita la creación de núcleos ó módulos periféricos desarrollados por el usuario, que se comuniquen a través de enlaces propietarios de XILINX como FSL y PLB. El Asistente para la Creación e Importación de Periféricos permitió crear una plantilla para el desarrollo de nuestro núcleo. En principio, la plantilla contiene la descripción de una máquina de estados para la comunicación a través de enlaces FSL. Dicha máquina de estados se modificó y adaptó a las necesidades de comunicación de nuestra aplicación y posteriormente se amplió para darle funcionalidad a nuestra estructura. Finalmente se montó el resto de nuestra estructura sobre este mismo código.

Máquina de estados

La máquina de estados está compuesta por seis estados: *Idle, Read_Inputs*, Operar, Write_*Idle, Write_Outputs* y *Load_Outputs*. En los tres primeros estados se realiza la recepción de datos y realización de las operaciones aritméticas, los segundos tres estados están involucrados en la recolección de resultados y el envío de estos hacia el procesador embebido *MicroBlaze*. Se cuenta además con una condición de reinicio, con la cual el módulo regresa a sus condiciones iniciales.

En el estado de *Idle* el núcleo espera la llegada de un dato o palabra de 32 bits por parte del *MicroBlaze* a través del enlace FSL. En caso de que esto suceda (FSL_S_Exists = '1'), se avanza al estado *Read_Inputs,* figura 5.30.

En el estado *Read_Inputs*, el dato leído desde la pila FSL se envía al bloque fl2fx, el cual convierte el número de punto flotante a punto fijo. Cuando el bloque indica que ha terminado dicha conversión (rdy_fl2fx = '1'), se realiza el cambio de estado a *Idle* con el fin de leer el siguiente dato. En caso de que se hayan leído todos los elementos requeridos, entonces se avanza hacia el estado Operar.

En el estado Operar, los PE's realizan las operaciones pertinentes y la máquina de estados permite que el registro de 32 bits dentro de cada PE cargue y almacene el resultado de las operaciones. Una vez que el PE indique que se ha realizado la operación (rdy_PE = '1'), la máquina regresa al estado *Idle,* con el fin de recabar datos para la siguiente operación. Si todas las operaciones se han realizado (nr_of_ops = 0), entonces se avanza al estado *Write_Idle* y se carga la pila de salida del bloque Recolector.

Write_Idle es un estado auxiliar que permite realizar la carga y el vaciado de la pila de salida del bloque Recolector de forma adecuada. Después de estar en este estado durante un ciclo de reloj, la máquina de estados avanza hacia el estado *Write_Outputs*.

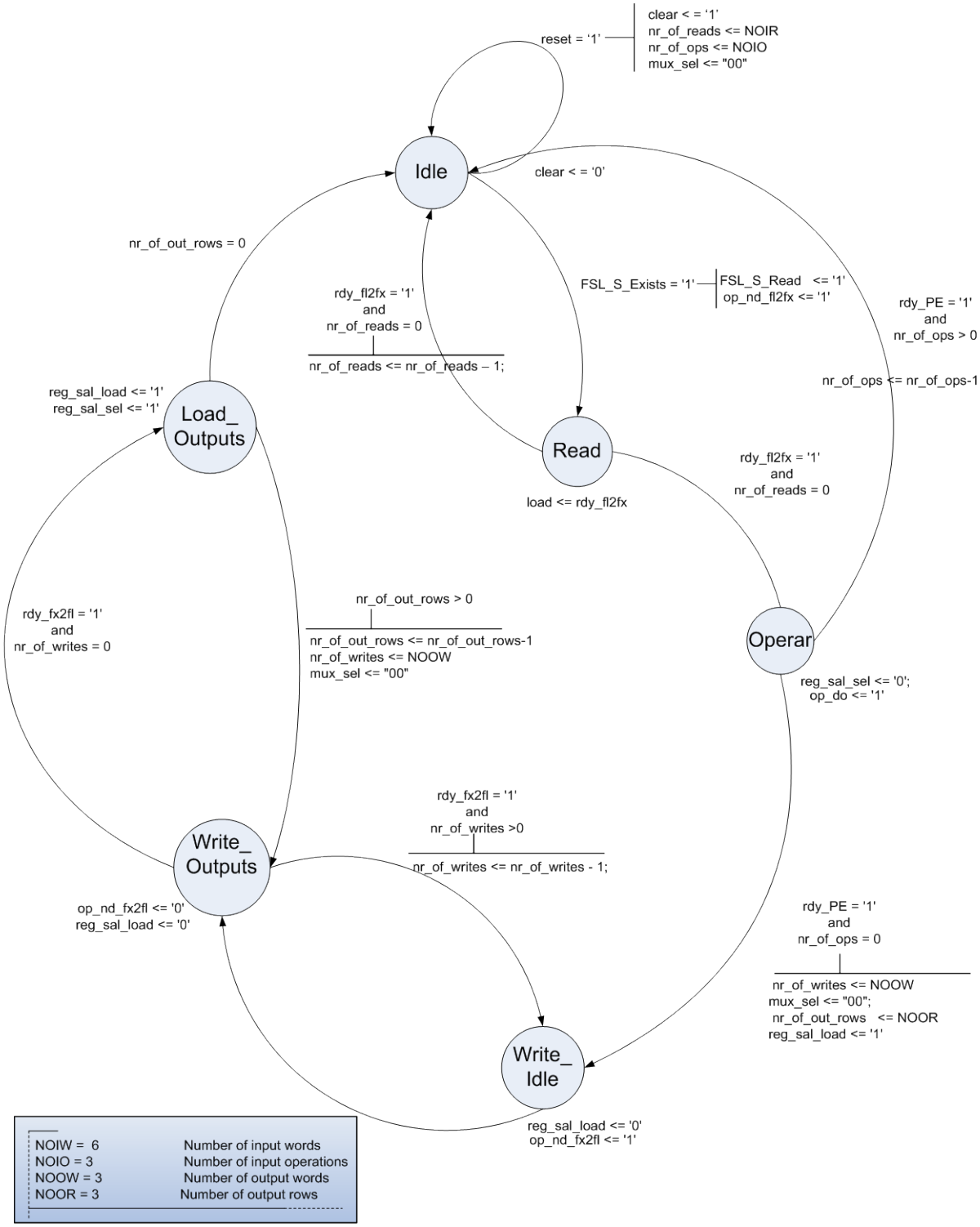


Figura 5.30 Diagrama de la máquina de estados que controla el flujo de datos del núcleo.

En el estado *Write_Outputs* se controla el multiplexor de salida mux_sal, se realiza la transformación de los datos de punto fijo a punto flotante y se envía el dato transformado de vuelta al *MicroBlaze*. Si se ha terminado de enviar todos los resultados de un renglón del arreglo (nr_of_writes = 0), se avanza hacia el estado *Load_Outputs* con el fin de realizar la misma operación con el siguiente renglón, en caso contrario se regresa al estado *Write_Idle*.

Una vez en el estado *Load_Outputs* se revisa si se ha terminado de enviar los resultados de todos los renglones (nr_of_out_rows = 0) , de ser así, el núcleo regresa a estado *Idle*, listo para otra operación de matrices, en caso contrario regresa al estado *Write_Idle* hasta terminar con todos los renglones.

Capítulo 6
Resultados

El presente capítulo tiene por objetivo mostrar y resumir los resultados obtenidos en este proyecto, los cuales abarcan desde los resultados de la búsqueda de sensores adecuados para el subsistema de navegación hasta los resultados de la implementación de algoritmos en recursos de la computadora de navegación como sistema embebido, es decir, el resultado de implementar los algoritmos de estimación en el dispositivo FPGA.

6.1. Búsqueda de sensores de navegación

Se consiguió reunir información sobre los requerimientos de desempeño de sensores de navegación para satélites pequeños y sobre sensores comerciales que pudieran satisfacer estos requerimientos, basados fundamentalmente en artículos y en experiencia que han tenido otros grupos dedicados a esta línea de investigación.

Se logró concentrar información sobre características específicas de diversos sensores, así como sus costos. También se establecieron contactos con los proveedores de algunos de estos productos, específicamente con la compañía Surrey Satellite Technology (SSTL) sobre su receptor GPS con calificación espacial utilizado en diversas misiones de satélites pequeños.

Se eligió un conjunto de sensores que satisfacen ampliamente los objetivos y requerimientos del presente trabajo y de otras investigaciones que se desarrollan actualmente en el Instituto de Ingeniería.

6.2. Implementación de la computadora de navegación mediante una plataforma FPGA

La implementación de la computadora de navegación mediante un dispositivo FPGA y las herramientas asociadas a este desarrollo han permitido explorar nuevas opciones para otras investigaciones dentro del mismo grupo donde se elaboró el presente proyecto.

La plataforma FPGA probó ser una herramienta útil y versátil para este tipo de proyectos. Sin embargo, el proceso de aprendizaje en la utilización de estas herramientas es largo y bajo ciertas condiciones resultaría costoso.

Es por ello que, la experiencia y conocimientos adquiridos durante el proceso de desarrollo de la presente tesis serán utilizados y aplicados a otros proyectos directamente y mediante la

transferencia de estos conocimientos a otros estudiantes e investigadores dentro de nuestro grupo de trabajo, disminuyendo de esta forma el tiempo de aprendizaje y por tanto el tiempo de desarrollo de nuevos proyectos.

6.3. Adquisición de datos de sensores

Se diseñó, implementó y probó un sistema capaz de adquirir las mediciones de los sensores de navegación mediante un FPGA a través del estándar de comunicaciones I^2C.

A continuación, se presenta en las figuras las mediciones de los sensores acelerómetro, magnetómetro y giróscopo adquiridas mediante la arquitectura descrita en la sección 5.1.3.

Para esta serie de mediciones, los ejes de la unidad de sensores fueron desplazados a distintas posiciones, con el fin de validar con éxito tanto el funcionamiento de estos como del sistema de adquisición. Tal como se esperaba, las mediciones del giróscopo resultaron ser las más inestables, figuras 6.1, 6.2 y 6.3.

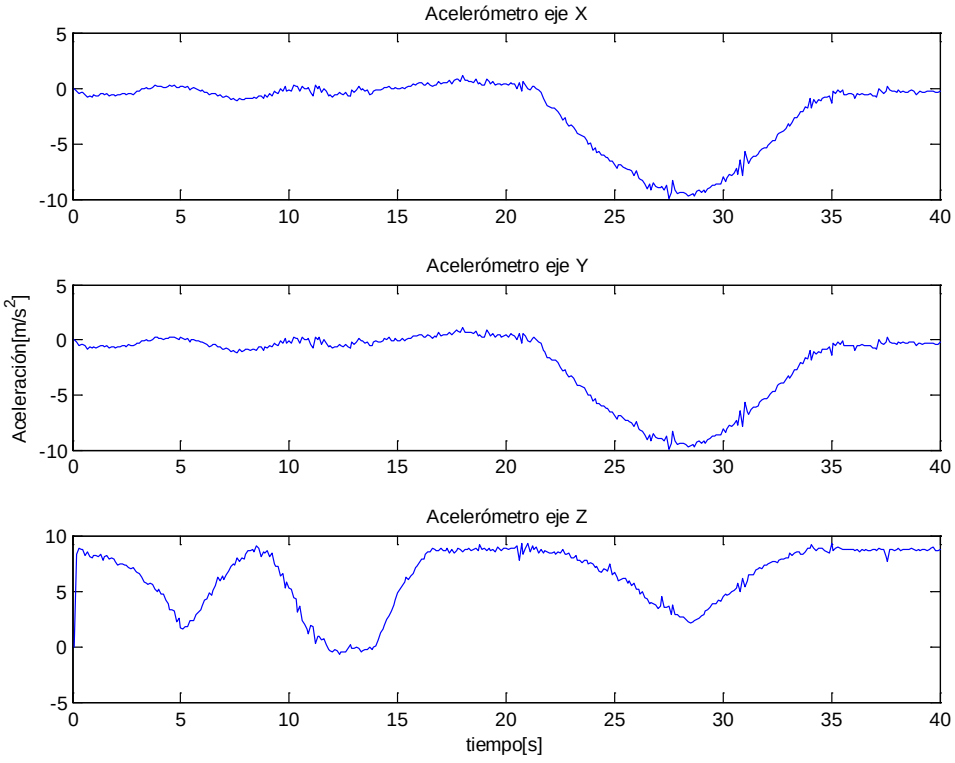


Figura 6.1 Mediciones del acelerómetro en sus tres ejes.

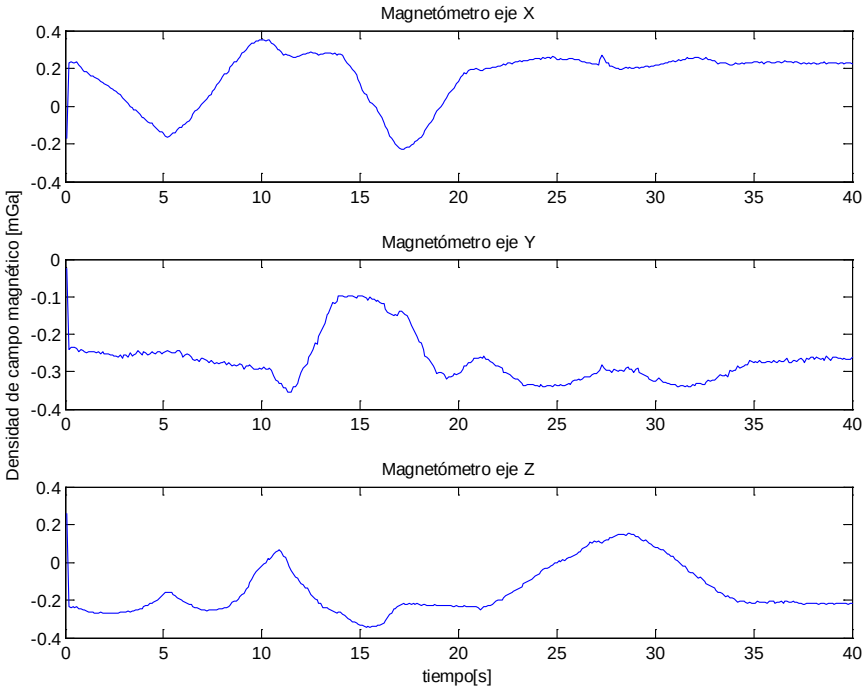


Figura 6.2 Mediciones del magnetómetro en sus tres ejes.

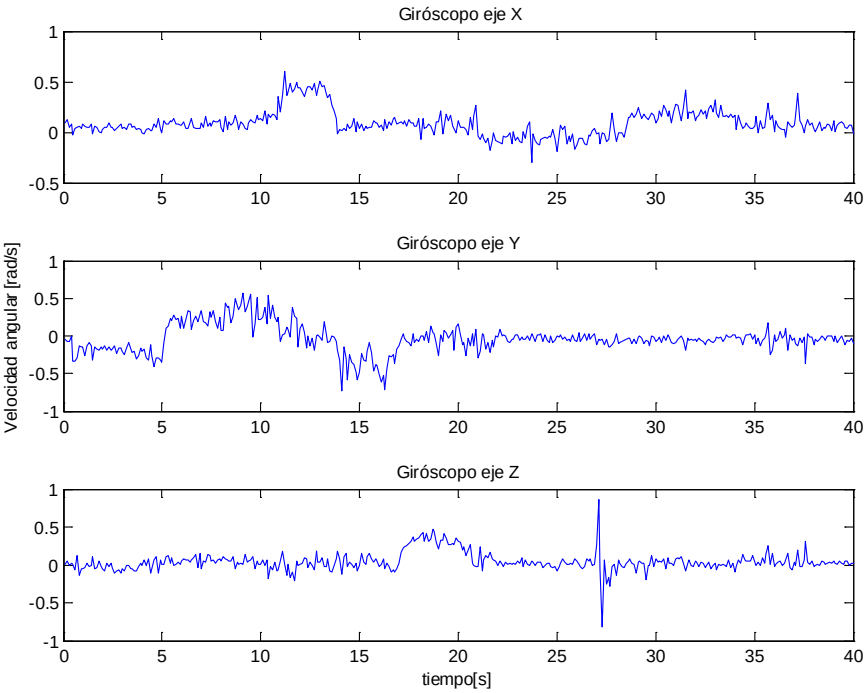


Figura 6.3 Mediciones del giróscopo en sus tres ejes.

6.4. Construcción de bloques de hardware para la implementación

En total, se diseñaron, construyeron y validaron 4 bloques de hardware (cores) completamente funcionales, de los cuales, 3 fueron utilizados en la implementación del método TRIAD, y el restante se propuso como método de aceleración por hardware para resolver la multiplicación de matrices en el filtro de Kalman.

Los bloques descritos en la sección 5.2 permiten realizar operaciones en punto fijo como el cálculo de un vector unitario a partir de cualquier vector de cuatro elementos reales (Norma) con una resolución de 16 bits, el cálculo del producto cruz de dos vectores (CrossP) de tres elementos reales con una resolución de 16 bits y el cálculo de una expresión que involucra tres funciones trigonométricas a través de una tabla de búsqueda con resolución de 12 bits.

También se construyó y validó un bloque que permite realizar la multiplicación de dos matrices de 3 x 3 con elementos reales en punto flotante (entre cierto intervalo de valores, pues el funcionamiento interno se realiza en punto fijo) en tan solo 256 ciclos de reloj y que eventualmente se puede utilizar para optimizar la implementación de ciertas operaciones involucradas en dentro del filtro de Kalman.

La validación de los bloques fue realizada de forma independiente, previamente a la integración de los bloques en una arquitectura, mediante datos de prueba.

6.5. Implementación del método TRIAD

Una vez validados los bloques de hardware de forma independiente, estos se integraron tal y como se mencionó en la sección 5.2.4 (ver también figura 5.15). La validación del funcionamiento de estos bloques de forma integral se llevó a cabo probando distintas secciones del algoritmo que se consideraron de importancia, pero con datos reales de los sensores y en tiempo real.

Para ello, primeramente se estableció una referencia arbitraria para definir la orientación de la unidad de sensores. Se programó mediante MATLAB un script que reproduce el método TRIAD descrito en (52) y que además captura datos enviados a través del puerto serie de la PC. El programa que se ejecuta en el procesador embebido dentro del FGPA se modificó para enviar a la PC resultados parciales de la ejecución del algoritmo TRIAD, además de las mediciones de los sensores.

Mediante las mediciones reales y en tiempo real de los sensores, el algoritmo ejecutado en la PC calcula los mismos resultados parciales y los compara con los obtenidos por el *MicroBlaze*. De esta forma, se logra establecer la precisión y la eficiencia en general de este segmento del sistema embebido.

En la figura 6.4, se presenta el resultado de la comparación de dichos resultados parciales, comenzado con los vectores de medición, los cuales dan una idea muy clara del comportamiento y precisión de los bloques de hardware Norma y CrossP.

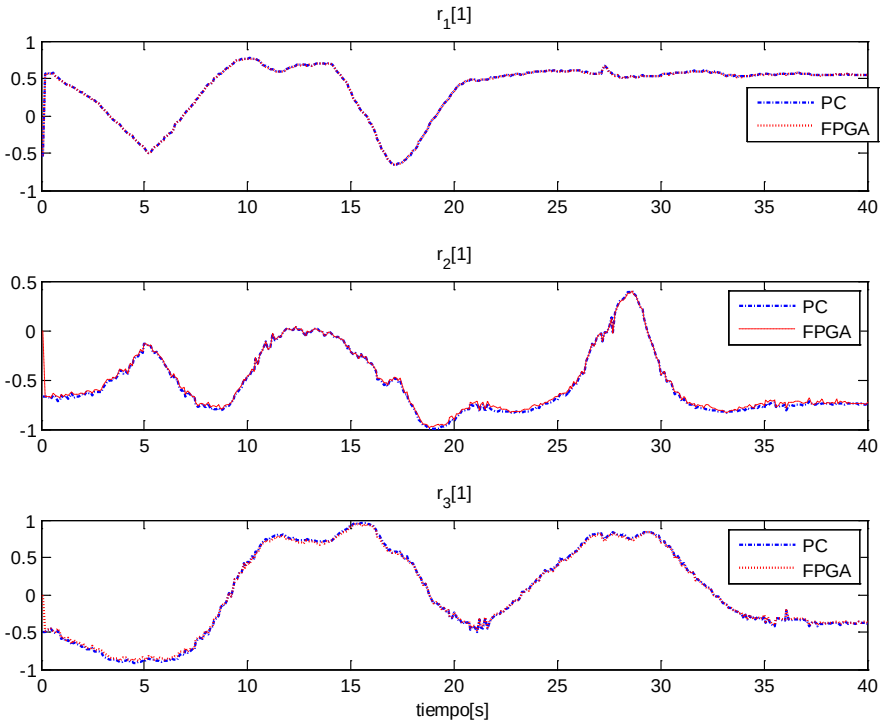


Figura 6.4 Comparativa del cálculo del primer elemento del conjunto de vectores ortogonales de medición r_1, r_2 y r_3 mediante la PC (MATLAB) y el FPGA.

Con el caso del cálculo del vector r1, que es simplemente la normalización del vector de medición del magnetómetro, se puede evaluar de forma más directa la precisión bloque Norma. En la tabla 6.1 se resume el error cuadrático medio en el cálculo de cada uno de estos vectores y sus elementos.

	[1]	[2]	[3]
r_1	0.0025	0.0032	0.0024
r_2	0.0376	0.0404	0.0146
r_3	0.0305	0.0208	0.0459

Tabla 6.1 Error cuadrático medio en el cálculo de los elementos de los vectores ortogonales de medición.

En la figura 6.5 se muestra la comparativa de las variables $trace$ y cb, las cuales son calculadas por el *MicroBlaze* a partir de los vectores de medición y de referencia. Finalmente, se realiza la comparación del cálculo del cuaternión de medición en la figura 6.6.

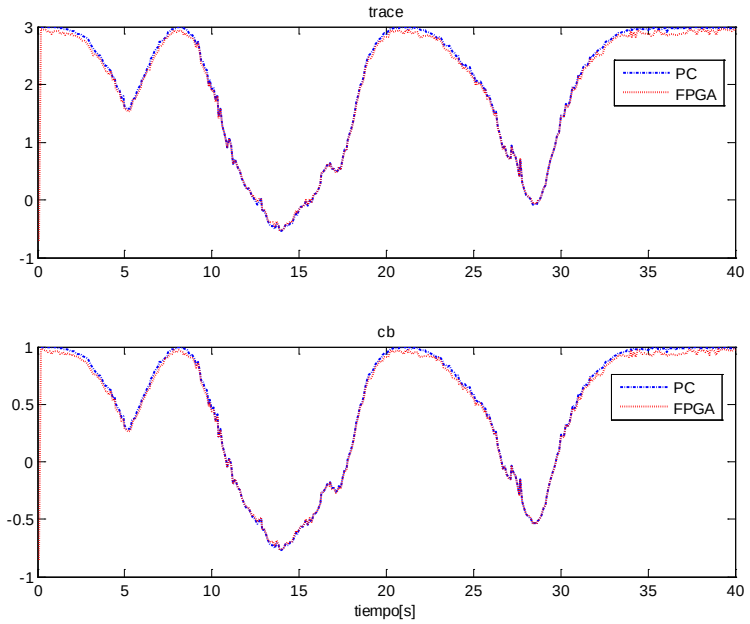


Figura 6.5 Comparativa del cálculo de las variables $trace$ y cb mediante la PC y el FPGA.

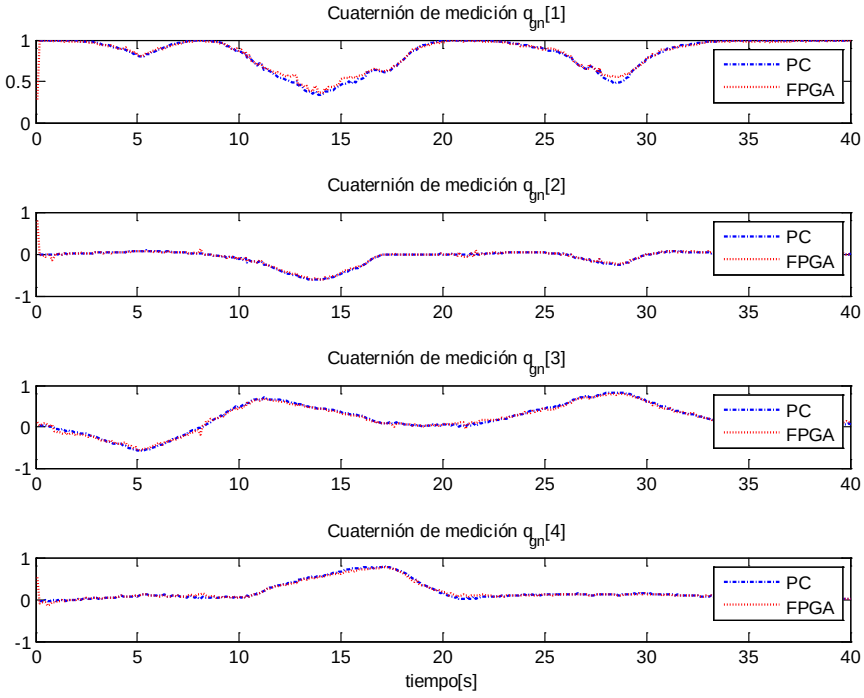


Figura 6.6 Comparativa del cálculo de los elementos del cuaternión de medición mediante la PC (MATLAB) y el FPGA.

El error cuadrático medio en el cálculo de los elementos del cuaternión de medición es el siguiente:

Elemento del cuaternión	Error cuadrático medio
q_{gn1}	0.0218
q_{gn2}	0.0148
q_{gn3}	0.0417
q_{gn4}	0.0245

Tabla 6.2 Error cuadrático medio para cada uno de los elementos del cuaternión calculado mediante el FPGA y comparado con calculado en la PC.

6.6. Implementación del filtro de Kalman

En MATLAB se programó un script que reproduce el filtro de Kalman descrito en (52) y en las secciones 3.5.2y 5.3 , que además captura datos enviados por el FPGA a través del puerto serie de la PC. El programa que se ejecuta en el procesador embebido ,dentro del FGPA, envía a la PC resultados parciales de la ejecución del algoritmo TRIAD, el filtro de Kalman extendido y las mediciones de los sensores, incluyendo esta vez al giróscopo.

Al igual que el caso del método TRIAD, mediante las mediciones reales y en tiempo real de los sensores, el algoritmo ejecutado en la PC calcula los mismos resultados parciales y los compara con los obtenidos por el MicroBlaze. De esta forma, se logra establecer la precisión y la eficiencia en general de este segmento del sistema embebido.

En las figuras 6.7, 6.8 y 6.9, se puede apreciar con claridad el efecto de filtrado del algoritmo EKF. La velocidad angular medida por los giróscopos presenta una amplia variabilidad en sus datos. La velocidad angular estimada, en contraste, presenta un comportamiento más estable en sus tres componentes. Por otro lado, el desempeño del sistema embebido en el FPGA es muy cercano al de la PC.

Algo similar sucede en el caso de los componentes del cuaternión (figuras 6.10, 6.11, 6.12 y 6.13). El comportamiento de las curvas generadas por el sistema embebido, por el FPGA, es muy cercano al de la PC. Se observa también que el EKF tiene un efecto de retardo sobre el cuaternión de medición, aunque se espera que la información que da el cuaternión estimado sobre la orientación, sea de mayor confiabilidad y utilidad para el proceso de control de estabilización.

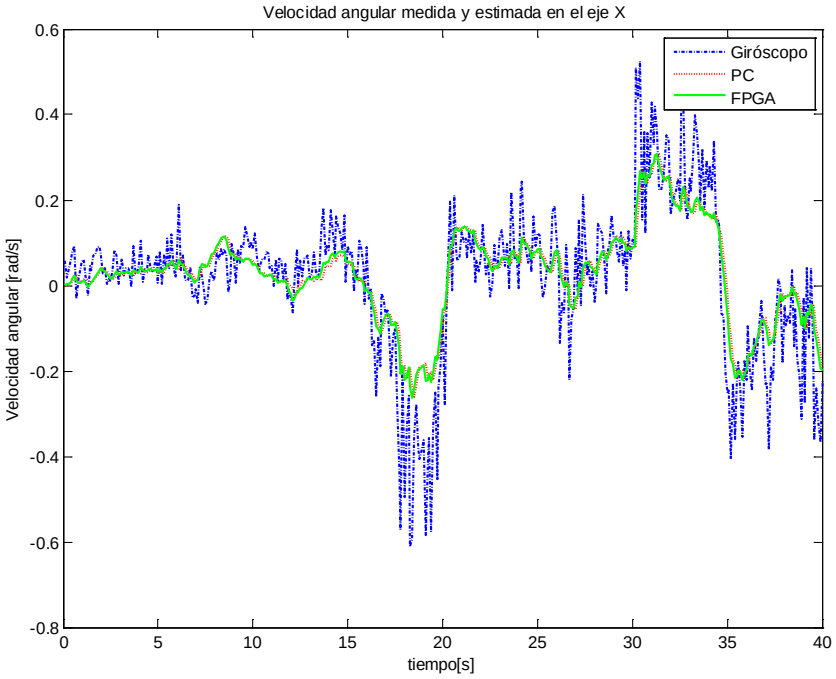


Figura 6.7 Comparativa de la velocidad angular medida por los giróscopos con la velocidad angular estimada por el filtro de Kalman en la PC y el FPGA en el eje X.

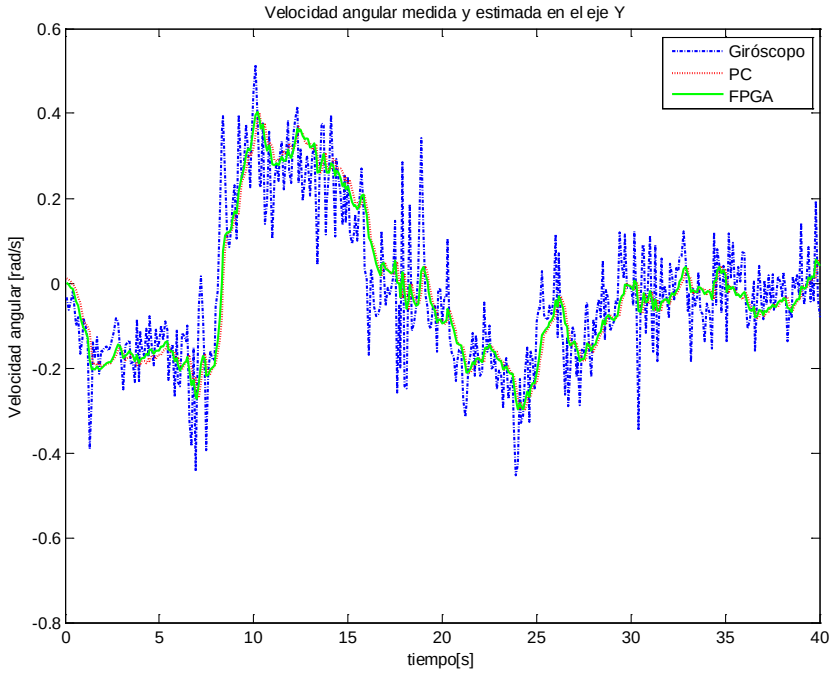


Figura 6.8 Comparativa de la velocidad angular medida por los giróscopos con la velocidad angular estimada por el filtro de Kalman en la PC y el FPGA en el eje Y.

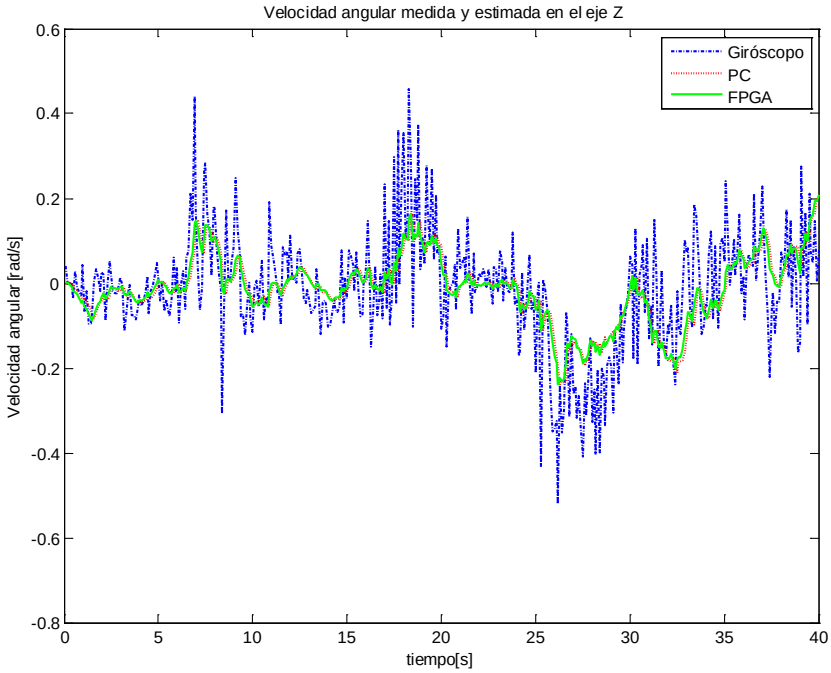


Figura 6.9 Comparativa de la velocidad angular medida por los giróscopos con la velocidad angular estimada por el filtro de Kalman en la PC y el FPGA en el eje Z.

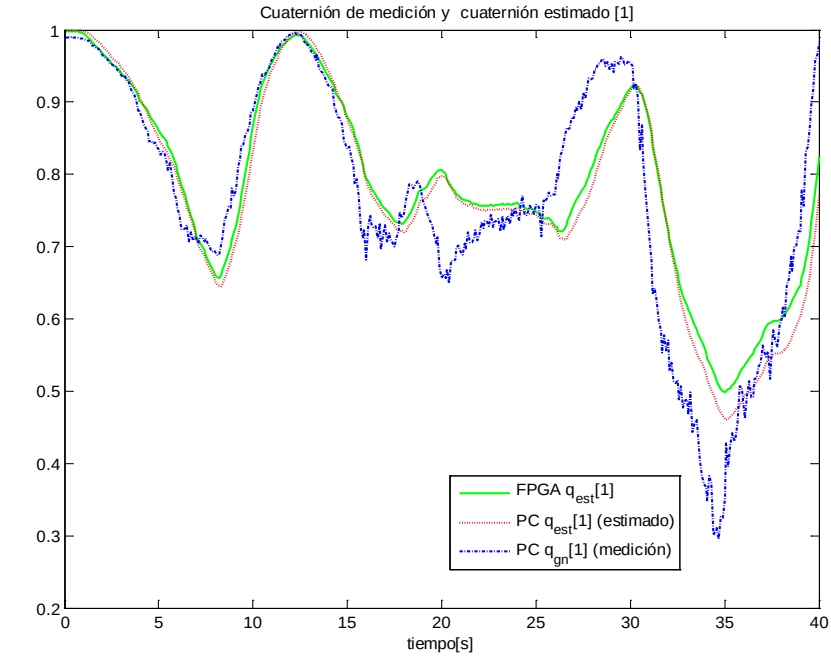


Figura 6.10 Comparativa del cuaternión de medición con el cuaternión estimado por el filtro de Kalman en la PC y el FPGA en su primer componente.

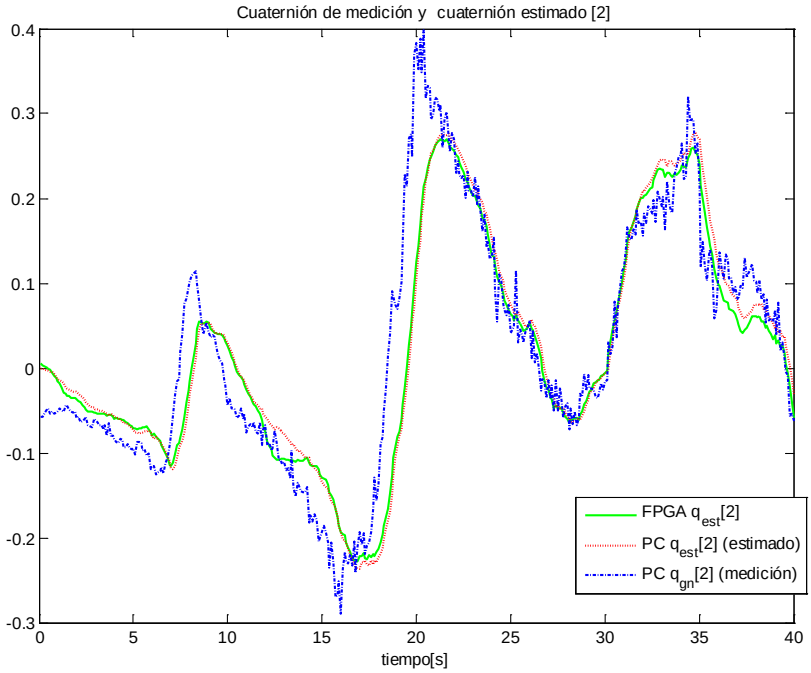


Figura 6.11 Comparativa del cuaternión de medición con el cuaternión estimado por el filtro de Kalman en la PC y el FPGA en su segundo componente.

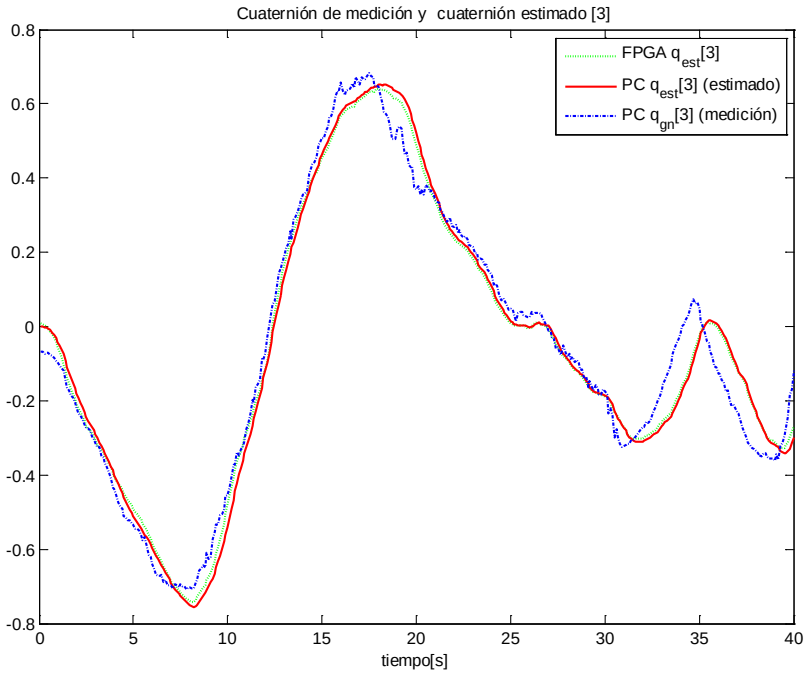


Figura 6.12 Comparativa del cuaternión de medición con el cuaternión estimado por el filtro de Kalman en la PC y el FPGA en su tercer componente.

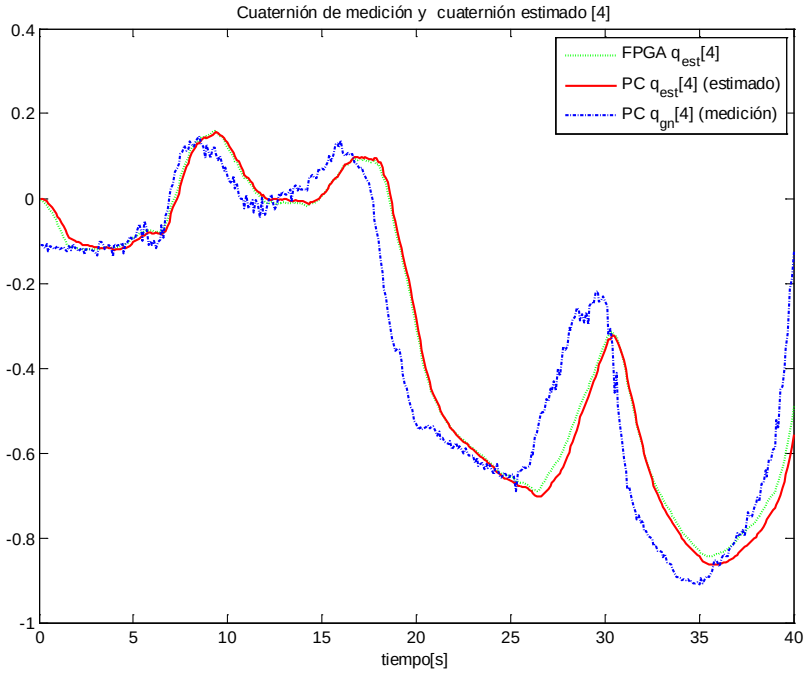


Figura 6.13 Comparativa del cuaternión de medición con el cuaternión estimado por el filtro de Kalman en la PC y el FPGA en su cuarto componente.

6.7. Utilización de recursos del FPGA y tiempo de ejecución

El tiempo de ejecución de los diferentes procesos que lleva a cabo la computadora de navegación constituye un parámetro importante para evaluar la eficiencia de la implementación del sistema embebido.

Para medir dicho tiempo fue necesario conectar un elemento adicional a la arquitectura de la computadora de navegación: el núcleo de hardware propietario (*IP core*) XPS Timer/Counter. Con ayuda de este temporizador, manejado a través de instrucciones de *software* programadas en el *MicroBlaze*, es posible contar el número de ciclos de reloj que transcurren entre dos eventos o procesos.

De esta forma fue posible obtener los tiempos de ejecución promedio para cada uno de los tres principales bloques de la computadora de navegación: programa de adquisición de datos, método TRIAD y algoritmo EKF, los cuales se resumen en la tabla 6.3:

Proceso	Tiempo
Adquisición de datos	3.2694[ms]
Método TRIAD	0.5024 [ms]
EKF	28.4052[ms]
Total	32.1770[ms]

Tabla 6.3 Tiempo de ejecución promedio de cada uno de los componentes de la computadora de navegación.

Gracias a estas mediciones se puede comprobar que una iteración completa de todos los algoritmos, se lleva a cabo en un tiempo menor (32.1770[ms]) al tiempo requerido por la aplicación (100[ms]).

El tiempo de ejecución tan alto en el caso del algoritmo EKF a comparación de los otros, se debe primordialmente, a que este proceso se lleva a cabo completamente en software y mediante aritmética de punto flotante, en contraste al proceso del método TRIAD, que además de requerir un menor número de operaciones, se lleva a cabo en punto fijo y aplicando técnicas de aceleración por hardware.

Así mismo, se midieron los tiempos promedio de lectura y escritura para los bloques de normalización de vectores y cálculo de producto cruz. Los resultados se muestran en la tabla 6.4. El tiempo de ejecución del bloque está incluido en los tiempos de lectura y escritura.

Bloque	Tiempo de escritura	Tiempo de lectura
Norma	1.54[µs]	1.52[µs]
Cross_P	1.70[µs]	1.44[µs]

Tabla 6.4 Tiempo de lectura y escritura de los bloques personalizados de hardware para el método TRIAD.

Por otro lado, la cantidad de recursos del dispositivo FPGA es igualmente importante. En las tablas 6.5, 6.6, 6.7 y 6.8, se resumen respectivamente los recursos consumidos por los bloques Norma, Cross_P, SinCos y el sistema embebido en general, indicando el porcentaje que representa del total de recursos del dispositivo.

Bloque Norma		
Recurso	**Cantidad**	**Porcentaje**
Slices	60 de 4656	1%
Flip Flops	92 de 9312	0%
LUTs	78 de 9312	0%
MULT18X18	2 de 20	10%

Tabla 6.5 Recursos utilizados por el bloque Norma.

Bloque Cross_P		
Recurso	**Cantidad**	**Porcentaje**
Slices	77 de 4656	1%
Flip Flops	103 de 9312	1%
LUTs	80 de 9312	0%
MULT18X18	2 de 20	10%

Tabla 6.6 Recursos utilizados por el bloque Cross_P.

Bloque SinCos		
Recurso	**Cantidad**	**Porcentaje**
Slices	11 de 4656	0%
Flip Flops	4 de 9312	0%
LUTs	19 de 9312	0%
BRAMs	3 de 20	15%

Tabla 6.7 Recursos utilizados por el bloque SinCos.

Sistema Embebido		
Recurso	**Cantidad**	**Porcentaje**
Slices	3042 de 4656	65%
Flip Flops	3125 de 9312	33%
LUTs	4881 de 9312	52%
BRAMs	19 de 20	95%
MULT18X18	11 de 20	55%

Tabla 6.8 Recursos utilizados por el sistema embebido de la computadora de navegación.

De las tablas anteriores, se puede observar que los recursos utilizados por los bloques Norma y Cross_P son relativamente bajos, además de que cuentan con tiempos de ejecución muy pequeños, por lo que resultan una solución ideal al problema de cálculo de vectores unitarios y productos cruz involucrados en el algoritmo del método TRIAD. Por su parte, aunque el bloque SinCos ocupa importantes recursos en bloques de RAM, el resto de los recursos utilizados son mínimos y considerando que libera al microprocesador de ejecutar rutinas de software que podrían llegar a ser extensas, complejas y que requieran mucho mayor tiempo en ejecutarse, se considera también como una solución muy adecuada para esta aplicación.

En la tabla 6.8, se puede observar un aumento significativo en los recursos utilizados. Esto es porque en ella se incluyen los recursos necesarios para la síntesis del procesador embebido *MicroBlaz*e, el cual, fue configurado con una memoria para almacenamiento del programa de 32 Kb y con la activación de la unidad de punto flotante, lo cual se ve reflejado en un incremento de los recursos, principalmente de bloques de RAM (BRAM) y multiplicadores embebidos (MULT18X18), respectivamente. Finalmente, las piezas de software programadas para los procesos de adquisición de datos, método TRIAD y EKF, superaron los 28Kb, lo que nos permite concluir que los recursos del dispositivo fueron aprovechados óptimamente.

Capítulo 7
Conclusiones

Del trabajo realizado en esta tesis se desprenden las siguientes conclusiones:

Se logró desarrollar e integrar un sistema de navegación capaz de reunir la información de diferentes sensores y utilizarla para obtener datos más confiables sobre la orientación de un cuerpo, con aplicación directa a prototipos satélites y satélites reales.

Esto permitirá implantar y evaluar por primera vez en México el desempeño de algoritmos de estimación y filtrado, y posteriormente algoritmos de control de orientación desarrollados en el Instituto de Ingeniería (IdeI), fuera de un simulador de Matlab y realizarlo físicamente en una mesa suspendida en aire, la cual se está terminando de integrar en el IdeI, UNAM.

La investigación que se realizó durante el desarrollo de esta tesis para especificar las características y requerimientos de los sensores de navegación servirá como referencia y punto de partida para proyectos satelitales inmediatos que sigan esta línea de investigación. También se logró actualizar las referencias sobre el estado del arte en materia de tecnología de sensores a nivel comercial, y la utilización de estos dispositivos en misiones satelitales reales.

Las plataformas FPGA son una herramienta muy adecuada para el desarrollo de sistemas embebidos. En nuestro caso, resultó serlo para implementar algoritmos de estabilización satelital en 3 ejes que se validarán en Tierra y de la forma más aproximada a la ausencia de fricción que existe en órbita espacial, es decir, para implementar la computadora de navegación de un satélite.

Los resultados exitosos que se obtuvieron en esta tesis han reforzado la idea original de emplear tecnología FPGA para automatizar funciones de navegación en los proyectos HUMSAT y SATEDU dentro del Instituto de Ingeniería.

Respecto al satélite SATEDU, la plataforma de navegación basada en FPGAs hará posible como producto adicional a este trabajo de tesis, formar recursos humanos especializados en este campo, gracias a la experiencia y conocimientos adquiridos durante su desarrollo.

Este aspecto es muy importante de subrayar, pues este desarrollo y esta tecnología es pionera en la UNAM y en muchas instituciones nacionales, tanto en la parte de complejidad de

modelado matemático, como en la posibilidad de automatizar funciones complejas de estabilización en satélites con técnicas de capacidad de respuesta en tiempo real.

Desde que al inicio de la tesis se bosquejó la idea general de la implementación de la computadora de navegación como sistema embebido se planteó como objetivo el uso eficiente de todos los recursos del dispositivo FPGA. Esto es, la combinación de elementos de hardware y software conviviendo dentro de un mismo sistema a través de esquemas de aceleración por hardware, así como el análisis para determinar los procesos en donde estos esquemas debían ser utilizados, permitió optimizar el uso de los recursos del dispositivo de tal forma que se superaron con éxito las expectativas planteadas al inicio.

Más allá de la optimización en el uso de los recursos del dispositivo FPGA a partir principalmente del desarrollo de bloques de hardware personalizados y especializados, se logró una implementación completamente funcional de la computadora de navegación, que permitió a los algoritmos ejecutados dentro del FPGA generar resultados lo suficientemente cercanos a los resultados obtenidos mediante simulaciones en una PC. Lo anterior es un importante avance en el Grupo de Desarrollo de Sistemas Aeroespaciales del IdeI, UNAM, pues permitirá evaluar no sólo el desempeño de los algoritmos de estimación y filtrado implementados en un experimento real, sino también nuevos algoritmos de control de orientación, además de que podrá ser utilizado para mejorar la operación en modo de seguimiento en tiempo real de SATEDU y finalmente, genera la posibilidad inmediata de trasladarse a aplicaciones reales.

A pesar de lo satisfactorio de los resultados obtenidos, sabemos que la implementación es susceptible de mejorarse aún más. El tamaño del programa en términos de memoria, podría reducirse significativamente (en casi 50%), al implementar el algoritmo EFK utilizando únicamente aritmética en punto fijo, lo cual, reduciría también el tiempo de ejecución y recursos del FPGA utilizados para sintetizar la unidad de punto flotante del procesador, liberando así recursos para otras aplicaciones. Sin embargo, es importante destacar que esta reducción limitaría la flexibilidad del software ante modificaciones, por lo que la recomendación es que la migración a aritmética en punto fijo sea realizada en la implementación final, una vez que los algoritmos hayan sido evaluados y definidos por completo.

Debe subrayarse que en el IdeI, UNAM se continúa trabajando para realizar en las próximas semanas la primer prueba de validación de estabilización satelital con una mesa suspendida en aire instrumentada con sensores de navegación inercial y con ruedas inerciales. La primera prueba se realizará empleando como computadora de navegación a la tarjeta Spartan III-E, con la que se desarrolló el trabajo en esta tesis. Esta prueba será contundente para que todo el grupo de trabajo obtenga una realimentación sobre el comportamiento de diversos sistemas involucrados, entre ellos: el modelado matemático realizado, el comportamiento de sensores, la

respuesta de ruedas inerciales, y principalmente la respuesta del FPGA como medio de integración y ejecución de algoritmos. Una vez realizada la primera prueba se harán las modificaciones pertinentes en cada subsistema y nuevamente se realizarán pruebas similares hasta llegar a una prueba con resultados adecuados y repetibles. Una vez alcanzado este punto de avance se realizarán pruebas similares pero ahora sustituyendo a la tarjeta Spartan III-E por SATEDU junto con su nueva tarjeta de estabilización basada en FPGA, la cual se termina de construir en este instituto.

Finalmente se acentúan los resultados y experiencias de trabajo multidisciplinario que ha desarrollado el autor durante este trabajo de maestría, en la que se ha interactuado con personas jóvenes, que ya son expertos en sus capos de trabajo, entre ellos Ingenieros de campos como la Mecatrónica, Mecánica, Instrumentación, Procesamiento de Señales y de Computación.

Apéndice A

Esquemas RTL

El objetivo de este apéndice es mostrar, en forma de esquemas RTL, el resultado de la síntesis del código VHDL que describe cada uno de los bloques utilizados en la implementación del método TRIAD, descrito en la sección 5.2.

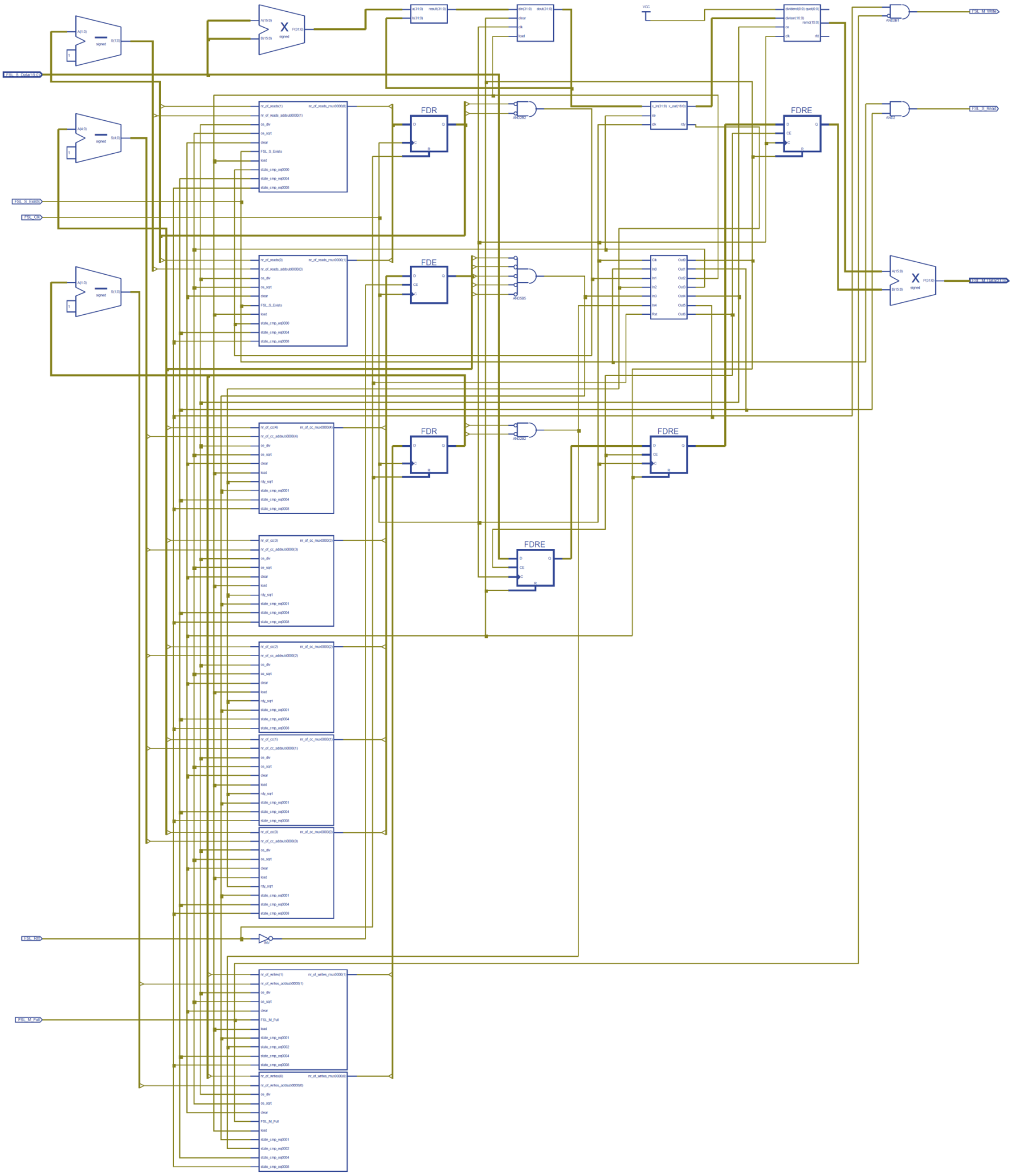


Figura A.1 Esquema RTL (Lógica Resistor Transistor) del núcleo Norma para el cálculo de vectores unitarios.

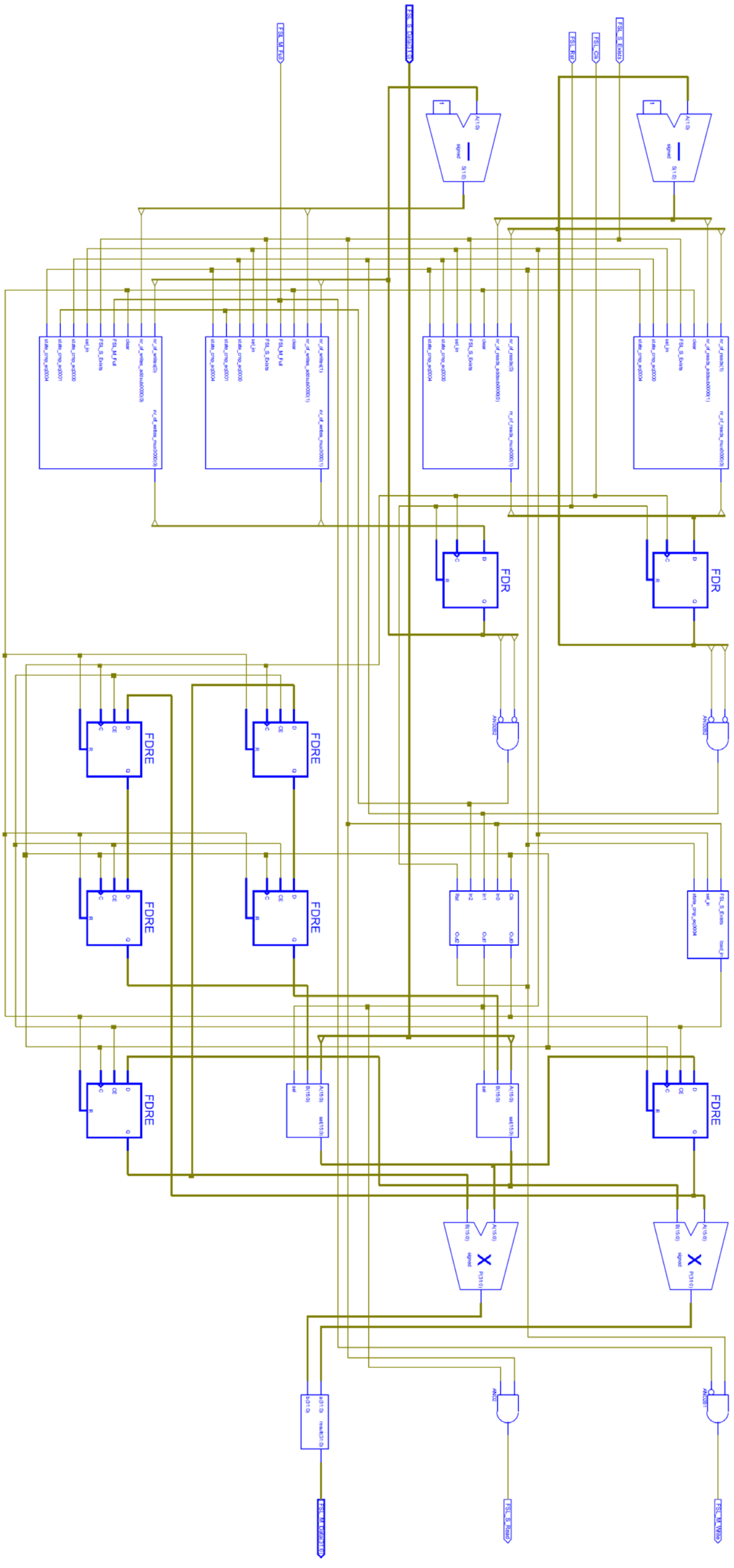


Figura A.2 Esquema RTL (Lógica Resistor Transistor) del núcleo Cross_P para el cálculo de producto cruz.

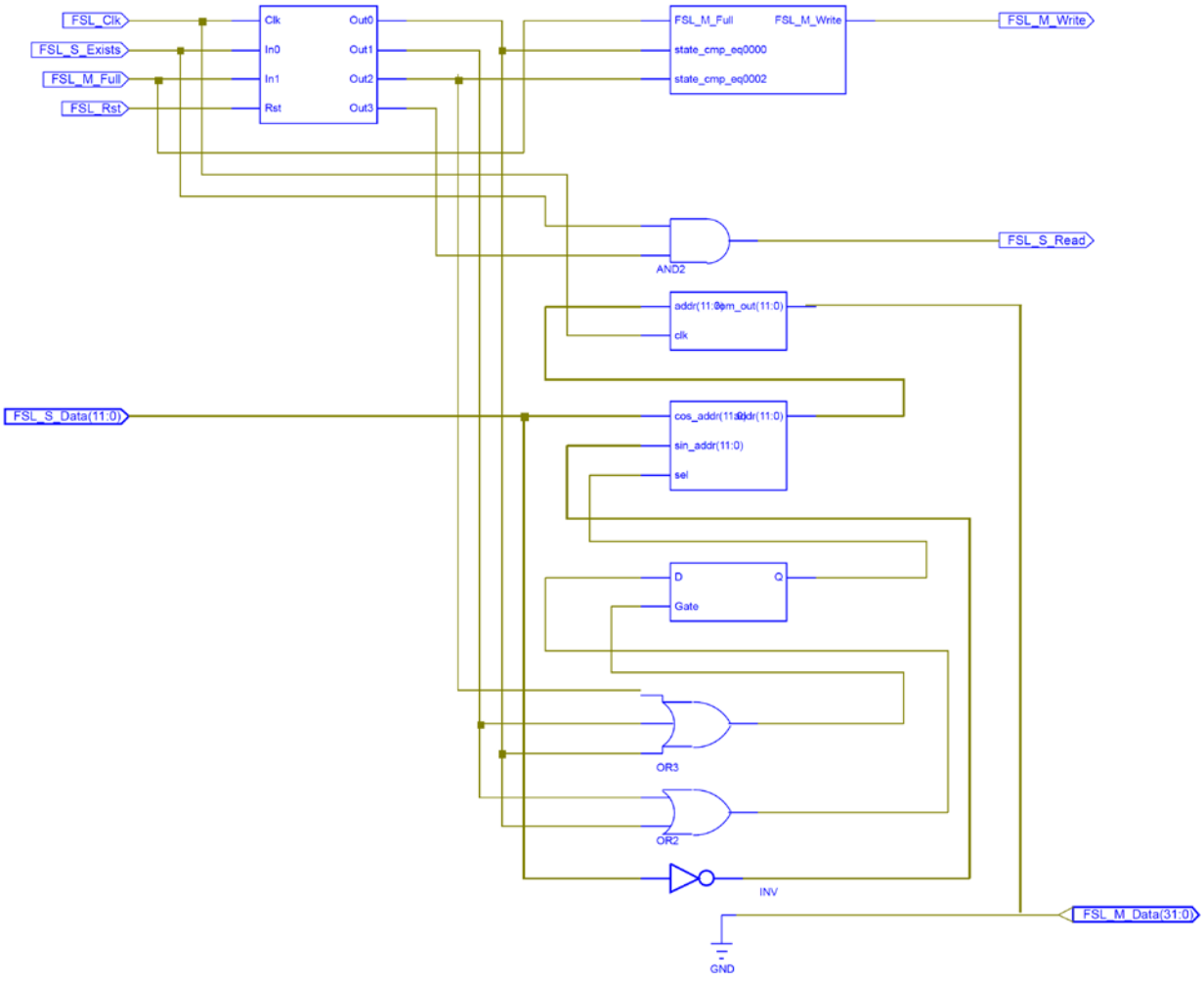


Figura A.3 Esquema RTL (Lógica Resistor Transistor) del SinCos para la conversión de Matriz de rotación a Cuaternión.

Apéndice B

Punto Fijo

En este apéndice se describirán de forma práctica, las reglas y técnicas básicas para la implementación de operaciones aritméticas en punto fijo y la manipulación datos numéricos en este formato, las cuáles fueron utilizadas durante el desarrollo del este trabajo de investigación.

B.1. Introducción

Aunque hoy en día, algunos de los procesadores disponibles comercialmente para su uso en sistemas embebidos ya cuentan con una o varias unidades de punto flotante para la realización de operaciones aritméticas en este formato, los recursos necesarios de espacio en silicio y tiempo de ejecución sigue siendo costosos. Por otro lado, la emulación de aritmética de punto flotante mediante software es también costosa en cuanto a recursos de memoria y tiempo de ejecución. En la mayoría de las aplicaciones, el compromiso entre la costosa implementación de algoritmos mediante punto flotante, y la alternativa más rápida, aunque menos precisa del punto fijo, comúnmente favorece a esta última, (92).

La utilización de aritmética en punto flotante en bloques de hardware es especialmente costosa, llegándose a utilizar más del 20% de los recursos de lógica programable de un dispositivo FPGA de la familia Spartan3E en la implementación de un bloque de multiplicación o suma en este formato.

Por lo anterior, el desarrollo de bloques de hardware y parte importante de los algoritmos de estimación fueron implementados utilizando punto fijo y aritmética binaria.

Finalmente, es muy importante resaltar el hecho de que la representación de número en punto fijo, así como la implementación de algoritmos en este formato, requiere de un análisis de precisión, resolución y rango de los datos, con el fin de representar a los datos en el formato más adecuado. Es posible que el resultado de dicho análisis conduzca a la conclusión de que no es factible la implementación en punto fijo para determinada aplicación, debido a que el rango de los datos es demasiado grande para ser representado con la resolución requerida y el tamaño de palabra disponible.

B2. Representación de números en punto fijo

Lo primero que debemos tomar en cuenta es que al trabajar con dispositivos electrónicos, es necesario utilizar aritmética binaria, pero más importante aún es que el valor de un número representado por N bits depende enteramente de su interpretación, (93).

En esta sección se expondrá la representación de un subconjunto de los números racionales que es posible representar con un tamaño de palabra finito, es decir, un número determinado de bits. Recordemos antes que los números racionales son el conjunto de números que pueden ser expresados como a/b donde a,b ϵZ, b ≠ 0 (Z representa al conjunto de los enteros). El subconjunto al que aquí nos referimos son los racionales para los cuales b = 2^n (n ϵZ).

B.2.1. Representación de números racionales no signados

La representación de números racionales en punto fijo requiere que el programador coloque un punto virtual entre dos bits, de tal forma que la palabra quede dividida en una parte entera y una parte fraccionaria (94) :

Q[I].[F]

Donde I es el número de bits que representan la parte entera y F es el número de bits que representan la parte fraccionaria. La suma de I y F es igual al tamaño de la palabra (N).

I + F = N Ecuación B-1

El valor de un número racional D en sistema decimal representado por un número binario x_2 de N bits en formato QI.F está dado por la siguiente expresión:

$$D = \frac{1}{2^F}\sum_{n=0}^{N-1} 2^n b_n \quad \text{Ecuación B-2}$$

Donde b_n es el n bit del número x_2. Es importante notar que la suma dentro de la ecuación B-2 resume el método utilizado para encontrar la representación en base diez de un número en base dos de N bits, por lo que dicha ecuación podría reescribirse de la siguiente forma:

$$D = \frac{1}{2^F} x_{10} \quad \text{Ecuación B-3}$$

Donde x_{10} es la representación en sistema decimal del número binario x.

El rango de datos que pueden ser representados bajo el formato QI.F (ó formato IQF) está dado por la siguiente expresión:

$$0 \leq D \leq 2^I - 2^{-F} \quad \text{Ecuación B-4}$$

O de otra forma también:

$$0 \leq D \leq (2^N - 1)/2^F \quad \text{Ecuación B-5}$$

De las expresiones anteriores, se puede deducir una forma práctica para determinar el valor de una representación bajo formato QI.F, o el caso inverso, encontrar la representación de cierto valor dado.

Por ejemplo, en una representación con formato 2Q6, se tienen dos bits para representar la parte entera y seis para representar la parte decimal:

- Representación de un número binario x de 8 bits (utilizada por el procesador):

$$b_7b_6b_5b_4b_3b_2b_1b_0$$

- Representación de un número racional en formato 2Q6 (utilizada por el programador):

$$b_1b_0.b_{-1}b_{-2}b_{-3}b_{-4}b_{-5}b_{-6}$$

El rango de datos que es posible representar con este formato es de:

$$0 \leq D \leq 2^2 - 2^{-6} \text{ es decir } \quad 0 \leq D \leq 3.9844$$

Si se tiene el número binario x = 1100 1001_2 (201_{10}) y consideramos un formato 2Q6, entonces la interpretación dicho número binario estará dada por la ecuación B-3, es decir:

$$D = \frac{201}{2^6} = 3.1406$$

Por lo tanto, para encontrar la representación en formato 2Q6 de un número racional, dentro del rango especificado anteriormente, basta con multiplicar dicho número racional por $2^F = 2^6$, y convertir la parte entera del producto a su representación en sistema binario, y colocar el punto virtual en el lugar correspondiente:

- Producto: $3.14 * 2^6 = 200.9600$
- Representación en sistema binario del producto: $x = 201_{10} = 1100\ 1001_2$
- Representación del número racional en formato 2Q6: $11.00\ 1001_2$

B.2.2. Representación de números racionales no signados (complemento a dos)

Se utiliza para representar tanto números positivos como negativos. Bajo un formato QI.F, el rango de los datos está dado por:

$$-2^I \leq D \leq -2^I - 2^{-F} \qquad \text{Ecuación B-6}$$

Pero en este caso, la suma de I y F no es igual al tamaño de la palabra N. Para representar números signados, se requiere un bit más que en el caso de números no signados, por lo tanto:

$$N = I + F + 1 \qquad \text{Ecuación B-7}$$

Por lo tanto, valor de un número racional Den base diez representado por un número binario x_2 de N bits en formato QI.F signado está dado por la siguiente expresión:

$$D = \frac{1}{2^F}\left[-2^{N-1} + \sum_{n=0}^{N-2} 2^n b_n\right] \qquad \text{Ecuación B-8}$$

Donde b_n es el n bit de x_2. Es importante destacar que la expresión entre corchetes formula el método para convertir un número binario con representación en complemento a dos, a su equivalente representación en sistema decimal.

Por ejemplo, en una representación con formato 2Q6 signado, se tienen tres bits para representar la parte entera y seis para representar la parte decimal.

- Representación de un número binario x de 9 bits: $b_8b_7b_6b_5b_4b_3b_2b_1b_0$
- Representación del número racional en formato 2Q6: $b_2b_1b_0.b_{-1}b_{-2}b_{-3}b_{-4}b_{-5}b_{-6}$

El rango de datos que es posible representar con este formato es de:

$$-2^2 \leq N \leq 2^2 - 2^{-6} \text{ es decir } -4 \leq D \leq 3.9844$$

Si se tiene el número binario x = 1 0011 0111_2 y consideramos un formato 2Q6 signado, se puede determinar que la representación de dicho número en base 10 corresponde a x= -201_{10}, mediante la expresión entre corchetes de la ecuación B-8, o también de una forma más práctica, mediante el siguiente procedimiento que utiliza el complemento a dos del número x.

- Si el número x es negativo (Bit más significativo = 1): x = 1 0011 0111_2

- Calcular el complemento a dos de x y encontrar la representación del resultado en base 10:

$$0\ 1100\ 1000_2 + 1_2 = 0\ 1100\ 1001_2 = 201_{10}$$

- Colocar el signo negativo a dicha representación en base 10: x= -201_{10}

- En caso contrario, simplemente encontrar la representación de x en base 10, como el caso de números racionales no signados.

Entonces la interpretación dicho número binario estará dada por la ecuación B-8, es decir:

$$D = \frac{-201}{2^6} = -3.1406$$

B.3. Reglas básicas de aritmética en punto fijo

A continuación se presentan un conjunto de reglas prácticas que se aplican al realizar operaciones con números que se encuentran bajo una representación en punto fijo. Estas reglas básicas fueron aplicadas para el desarrollo de los bloques de hardware en la implementación del método TRIAD.

B.3.1. Suma

El formato del resultado de la suma de dos números en formato QI.F es Q I+1.F. Esto significa que el resultado de la suma de dos números de M bits es de tamaño M+1 bits.

$$Q\ I.F + Q\ I.F = Q\ I+1.F$$

B.3.2. Multiplicación

El formato del producto de dos números en punto fijo es diferente para los casos signado y no signado.

- Números no signados: $Q\ I_1.F_1 + Q\ I_2.F_2 = Q\ I_1 + I_2\ .\ F_1 + F_2$

- Números signados: $Q\ I_1.F_1 + Q\ I_2.F_2 = Q\ I_1 + I_2+1\ .\ F_1 + F_2$

B.3.3. Corrimientos

A continuación, se describirán dos tipos de corrimientos: el literal y el virtual, y los resultados de cada uno de ellos. Nótese que los corrimientos están expresados en términos de corrimiento a la derecha por un número n de bits. El corrimiento a la izquierda se consigue con un número n negativo.

a) Literal

- Multiplicación/División por una potencia de dos

Es posible dividir o multiplicar un número binario por 2^n haciendo un corrimiento de n bits, pero conservando el formato de punto fijo, es decir, la posición del punto.

$$Q\ I.F >> n \quad = \quad Q\ I.F$$

$$x >> n \quad => \quad \frac{D}{2^n}$$

En la figura B.1 se muestra un ejemplo con un formato 2Q6 no signado.

Número original:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									b_7	$b_{6\bullet}$	b_5	b_4	b_3	b_2	b_1	b_0

Multiplicación por 8 (n=3):

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						b_7	b_6	b_5	b_4	$b_{3\bullet}$	b_2	b_1	b_0	0	0	0

División por 8 (n=3):

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
										$0_\bullet$	0	b_7	b_6	b_5	b_4	b_3

Figura B.1. Ejemplo de corrimiento para efectuar multiplicaciones/divisiones por una potencia de dos.

- Escalamiento

El corrimiento literal se puede utilizar para escalar un número binario, es decir hacerlo más grande o pequeño, pero sin alterar el valor que representa.

$$Q\ I.F >> n \quad = \quad Q\ I + n\ .\ F - n$$

En la figura B.2 se muestra un ejemplo con un formato 2Q6 no signado:

Número original:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									b_7	$b_{6\bullet}$	b_5	b_4	b_3	b_2	b_1	b_0

Escalamiento(n=-3):

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						b_7	$b_{6\bullet}$	b_5	b_4	b_3	b_2	b_1	b_0	0	0	0

Escalamiento(n=3):

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
												b_7	$b_{6\bullet}$	b_5	b_4	b_3

Figura B.2. Ejemplo de corrimiento para efectuar un escalamiento.

b) Virtual

Se refiere al corrimiento del punto virtual sin modificar el número binario, es decir, los bits no se mueven de su posición, sin embargo, es necesario reinterpretar el valor que representan dicho número binario de acuerdo a la nueva posición del punto. Se puede utilizar como un método alternativo para multiplicar o dividir un número por una potencia de dos que evita pérdida de precisión.

$$Q\ I.F >> n \quad = \quad Q\ I - n\ .\ F + n$$

En la figura B.3 se muestra un ejemplo con un formato 2Q6 no signado.

Finalmente, es muy importante resaltar el hecho de que la representación de número en punto fijo, así como la implementación de algoritmos en este formato, requiere de un análisis de precisión, resolución y rango de los datos, con el fin de representar a los datos en el formato más adecuado. Es posible que el resultado de dicho análisis conduzca a la conclusión de que no es factible la implementación en punto fijo para determinada aplicación, debido a que el rango de los datos es demasiado grande para ser representado con la resolución requerida y el tamaño de palabra disponible.

Número original:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									b_7	$b_{6\bullet}$	b_5	b_4	b_3	b_2	b_1	b_0

Multiplicación por 8:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									b_7	b_6	b_5	b_4	$b_{3\bullet}$	b_2	b_1	b_0

Se reinterpreta el número con un formato 4Q2 no signado.

División por 8:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
							$0_\bullet$	0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

Se reinterpreta el número con un formato Q9 no signado.

Figura B.3. Ejemplo de corrimiento para efectuar multiplicaciones/divisiones por una potencia de dos sin pérdida de precisión.

Referencias

1. **Lawrence, Anthony.** *Modern Inertial Tecnology. Navegation, Guidance, and Control.* EUA : Springer, 1998.
2. **Titterton, David y Weston, John.** *Strapdown Inertal Navegation.* GB : The Institute of Electrical Engineering, 2004.
3. **Mohinder, S. G., Lawrence, R.W. y Angus, P.A.** *Global Positioning Systems, Inertial Navegation and Integration.* EUA : John Wiley & Sons, 2001.
4. **Jiménez, Emilio.** *Subsistema de estabilización activa y sensores para un simulador satelital.* México : UNAM, 2009.
5. **Sidi, Marcel J.** *Spacecraft Dynamics and Control: A Practical Engineering Approach.* EUA : Cambridge University Press, 2002.
6. **A.M. Cruise, J.A. Bowles, T.J. Patrick, C.V. Goodvall.** *Principles of space instrument design.* EUA : Cambridge Universtiy Press, 1998.
7. **Maral, Gérard y Bousquet, Michel.** *Satellite Communications Systems: Systems, Techniques and Technology.* Singapore : John Wiley and Sons, 2009.
8. **Pisacane, Vincent L.** *Fundamentals of space systems* . EUA : Oxford University Press, 2005.
9. **Ripka, Pavel.** *Magnetic Sensors and Magnetometers.* EUA : Artech House, 2001.
10. **Kurz, Oliver.** *Design and Implemnetation of an Attitude Determination System for the Cubesat UWE-2 (Hardware based).* s.l. : Lulea University of Technology, 2008.
11. **Svartveit, Kristian.** *Attitude determination of the NCUBE satellite.* Noruega : Universidad Noruega de Ciencia y Tecnología (NTNU), 2003.
12. **El-Rabbany, Ahamed.** *Introduction to GPS. The Golbal Positioning System.* EUA : Artech House, Inc., 2002.
13. **Samana, Nel.** *Global Positioning. Technologies and Performance.* EUA : John Wiley and Sons, 2008.
14. **Kramer, H.J.** *Observation of the Earth and its environment: survey of missions and sensors.* Berlin : Springer, 2002.
15. **Powell, Thomas D., y otros.** *GPS Signals in a Geosynchronous Transfer Orbit: "Falcon Gold" Data Processing.* s.l. : AIR FORCE ACADEMY COLORADO SPRINGS CO, 1999.
16. **Oliver Montenbruck, Markus Markgraf, Miquel Garcia-Fernandez, Achim Helm.** *GPS for microsatellites. Status and perspectives.* Berlin : 6th IAA Symposium on Small Satellites for Earth Observation, 2007.
17. **Kaplan, Elliott D.** *Understanding GPS. Principles and applications.* EUA : Artech House, Inc., 1996.
18. **Goodman, John L.** *The Space Shuttle and GPS - A Safety-Critical Navigation Upgrade.* EUA : s.n.
19. **Smith, Cristin Anne.** *Leveraging COTS Hardware for Rapid Design and Development of Small Satellites at the USAF Academy.* Los Angeles,CA : 2nd Responsive Space Conference, 2004.
20. **Banerjee, Paritosh, Somnath,Puri, Amitava,Bose.** *Modern Inertial Sensors And Systems.* s.l. : Prentice Hall, 2008.

21. **Stéphane Dussy, Dick Durrant, Tony Moy.** *MEMS gyro for space applications. Overview of European activities .* s.l. : American Institute of Aeronautics and Astronautics, 2005.
22. **Man, Kin F.** *MEMS reliability for space applications by elimination of potential failure modes through testing and analysis.* EUA : Jet Propulsion Laboratory.
23. **Shea, Herbert R.** *Reliability of MEMS for space applications.* s.l. : SPIE, 2006.
24. **Shea, H.R.** *MEMS for pico- to micro-satellites.* s.l. : Ecole Polytechnique Fédérale de Lausanne.
25. **Beeby, Stephen.** *MEMS mechanical sensors .*
26. **Bao, M.-H.** *Handbook of sensors and actuators.* Paises Bajos : Elsevier Science B.V., 2000.
27. **George, Thomas.** *Overview of MEMS/NEMS Technology. Development for Space Applications at NASA/JPL.* s.l. : Proceedings of SPIE, 2003.
28. **M. N. Armenise, C. Ciminelli, F. De Leonardis, R. Diana, V. Passaro, F. Peluso.** *Gyroscope technologies for space applications.* Italia : Politecnico di Bari.
29. **Honeywell.** *Hoja de especificaciones. Three-Axis Magnetic Sensor Hybrid.*
30. —. *Hoja de especificaciones. Smart digital magnetometer.*
31. —. *Hoja de especificaciones. Three-Axis strapdown magnetometer.*
32. **Honeywell International Inc.** *Hoja de especificaciones. 3-Axis Digital Compass IC HMC5843.* Frebrero 2009.
33. **PNI Coorporation.** *Hoja de especificaciones. MicroMag3. 3-Axis Magnetic Sensor Module.* 2005.
34. *Small Magnetic Sensors for Space Applications.* **Díaz-Michelena, Marina.** Suiza : Molecular Diversity Preservation International, 2009, Revista sensors. ISSN 1424-8220.
35. **Analog Devices, Inc.** *Hoja de especificaciones. Digital Accelerometer: ADXL345.* 2009.
36. **STMicroelectronics.** *MEMS motion sensor: three-axis digital output gyroscope L3G4200D.* Septiembre 2010. Doc ID 17116 Rev 2.
37. **Analog Devices, Inc.** *High Performance, Digital Output Gyroscope ADXRS450.* 2010.
38. **InvenSense Inc.** *ITG-3200 Product Specification Revision 1.4.* Marzo 2010. PS-ITG-3200A-00-01.4.
39. **SensorDynamics AG.** *FAIL-SAFE INERTIAL MODULE SINGLE AXIS GYROSCOPE & SINGLE AXIS ACCELEROMETER.* Austria : s.n., 2010.
40. ESA Technology. [En línea] 09 de Septimebre de 2009. http://www.esa.int/SPECIALS/Technology/SEMVXYUHYXF_0.html.
41. **Prezzavento, Antonio M.** *3-AXIS MICRO GYROSCOPE FEASIBILITY STUDY.* Inglaterra : Astrium Ltd, 2002.
42. Phoenix Miniature GPS Receiver. [En línea] DLR. http://www.weblab.dlr.de/rbrt/GpsNav/Phoenix/Phoenix.html.
43. **Oliver Montenbruck, Ben Nortie, Sias Mostert.** *A Miniature GPS Receiver for Precise Orbit Determination of the Sunsat 2004 Micro-Satellite.* EUA : ION National Technical Meeting, 2004.
44. **Oliver Montenbruck, Eberhard Gill, Markus Markgraf.** *PHOENIX-XNS – A MINIATURE REAL-TIME NAVIGATION SYSTEM FOR LEO SATELLITES.* Noordwijk : Conferencia NAVITEC, 2006.
45. **Markus Markgraf, Oliver Montebruck.** *PHOENIX-HD – A MINIATURE GPS TRACKING SYSTEM FOR SCIENTIFIC AND COMMERCIAL ROCKET LAUNCHES.* Alemania : DLR/GSOC, 2005.

46. **M. Markgraf, O. Montenbruck.** *Total Ionizing Dose Testing of the Orion and Phoenix GPS Receivers.* Alemania : DLR/GSOC, 2004. TN 04-01.
47. **Oliver Montenbruck, Sunny Leung, Robert Bruninga.** *GPS Operations on the PCsat Microsatellite .* Portland, OR : ION GPS, 2002.
48. **DLR/GSOC.** *Phoenix GPS Data Sheet.* Alemania : DLR/GSOC, 2007.
49. **Brown, Charles D.** *Elements of spacecraft design.*
50. **Rogers, Robert M.** *Applied mathematics in integrated navigation systems.*
51. **Hartana, J.Z. Sasiadek and P.** *Sensor Data Fusion Using Kalman Filter.* s.l. : Carleton University.
52. **Córdova Alarcón, José Rodrigo.** *Estimación y control de orientación para el nanosatélite HUMSAT-México.* México : UNAM, 2011.
53. **I. Bar-Itzhack, R. Harman.** *Optimized triad algortihm.* s.l. : Journal of Guidance, Control and Dynamics, 1997.
54. *The Optimization of TRIAD.* **Shuster, Malcolm D.** 2, s.l. : The Journal of the Astronautical Sciences, Abril-Junio 2007, Vol. 55.
55. **J.L. Marins, Xiaoping Yun, E. R. Bachmann, R. McGhee.** *An extended Kalman Filter for quaternion-based orientation estimation using marg sensors.* s.l. : Proceedings of the 2001 IEEE/RSJ, International Conference on Intelligent Robots and Systems, 2001.
56. **Mohinder S. Grewal, Angus P. Andrews.** *Kalman Filtering: Theory and Practice Using MATLAB.* s.l. : John Wiley & Sons, Inc., 2001.
57. **Noergaard, Tammy.** *Embedded systems architecture: a comprehensive guide for engineers and programmers.* EUA : Newnes, 2005.
58. **A.P.Godse, A.O.Mulani.** *Embedded Systems.* India : Technical Publications Pune, 2009.
59. **Kamal, Raj.** *Embedded systems: architecture, programming and design.* Delhi : Tata McGraw-Hill, 2008.
60. **Moreno, Jaime H. y Lang, Tomás.** *Matrix computations on systolic-type arrays.* EUA : Kluwer Academic Publishers, 1992.
61. **Sass, Ron y Schmidt, Andrew G.** *Embedded Systems Design with Platform FPGAs Principles and Practices.* EUA : Morgan Kaufmann Publishers, 2010.
62. **Chu, Pong P.** *FPGA Prototyping by VHDL Examples.* s.l. : John Wiley & Sons, Inc., 2008.
63. **Altera Corporation.** FPGAs. [En línea] 2011. http://www.altera.com/products/fpga.html.
64. **Charles H. Roth, Jr.** *Digital Systems Design Using VHDL.* EUA : PWS Plushing Company, 1998.
65. **Zeidman, Bob.** *Designing with FPGAs and CPLDs.* EUA : CMP Books, 2002.
66. **Xilinx, Inc.** [En línea] 2011. http://www.xilinx.com/company/gettingstarted/index.htm.
67. **Hauck, Scott y DeHon, André.** *Reconfigurable computing: the theory and practice of FPGA-based computation.* EUA : Morgan Kaufmann Publishers, 2008.
68. **Ashenden, Peter J.** *The designer's guide to VHDL: Systems on sylicon.* EUA : Morgan Kaufmann, 2001.
69. **Daniel D. Gajski, Samar Abdi,Andreas Gerstlauer,Gunar Schirner.** *Embedded System Design: Modeling, Synthesis and Verification.* EUA : Springer, 2009.
70. **Xilinx, Inc.** *Spartan-6 Family Overview.* March 21, 2011. DS160 (v1.7).
71. —. *Using Embedded Multipliers in Spartan-3 FPGAs.* May 13, 2003. XAPP467 (v1.1).

72. —. FPGA Families. [En línea] 2011. http://www.xilinx.com/products/silicon-devices/fpga/index.htm.
73. **Altera Corporation.** *Cyclone V Device Family Advance Information Brief.* March 2011. AIB-01016-1.1.
74. —. *Arria V Device Family Advance Information Brief.* April 2011. AIB-01015-1.4.
75. —. Stratix V FPGA Family Overview. [En línea] 2011. http://www.altera.com/products/devices/stratix-fpgas/stratix-v/overview/stxv-overview.html.
76. —. Altera Devices. [En línea] 2011. http://www.altera.com/products/devices/dev-index.jsp.
77. **Meyer-Baese, Uwe.** *Digital signal processing with field programmable gate arrays.*
78. **Cofer, R. C. y Harding, Benjamin F.** *Rapid system prototyping with FPGAs.*
79. **Xilinx, Inc.** Embedded Processing. [En línea] 2011. http://www.xilinx.com/products/technology/embedded-processing/index.htm.
80. **Rosinger, Hans-Peter.** *Connecting Customized IP to the MicroBlaze Soft Processor Using the Fast Simplex Link (FSL) Channel.* s.l. : Xilinx, Inc., May 12, 2004. XAPP529 (v1.3).
81. **Xilinx, Inc.** *LogiCORE IP Processor Local Bus (PLB) v4.6 (v1.05a).* September 21, 2010. DS531.
82. —. *Embedded System Tools Reference Manual.* June 23, 2006. UG111 (v6.0).
83. —. Xilinx Platform Studio (XPS) Overview. [En línea] 2005. http://www.xilinx.com/itp/xilinx8/help/platform_studio/html/ps_b_gst_overview.htm.
84. —. Software Development Kit (SDK). [En línea] 2011. http://www.xilinx.com/tools/sdk.htm.
85. **Tchuente, Maurice.** *Parallel computation on regular arrays.* Gran Bretaña : Mancheste University Press, 1991.
86. **Nam Ling, Magdy A. Bayoumi.** *Specification and verification of systolic arrays.* Singapore : World Scientific Publishing Co., 1999.
87. **Roger Woods, John McAllister.** *FPGA-based implementation of signal processing systems.* Reino Unido : John Wiley & Sons Ltd., 2008.
88. **Philips Semiconductors.** *The I2C-BUS Specification Version 2.1.* Enero 2000.
89. **Xilinx, Inc.** *XPS IIC Bus Interface (v2.00a).* July 22, 2008. DS606.
90. **Lomont, Chris.** *Fast inverse square root.* Indiana, EUA : Purdue Univesity, 2003. www.math.purdue.edu/~clomont.
91. **Eberly, David.** *Fast inverse square root.* s.l. : Geometric Tools, LLC, 2010.
92. **Corporaal, Andrea G. M. Cilio and Henk.** *Floating Point to Fixed Point Conversion of C Code.* Holanda : Delft University of Technology, 2002.
93. **Yates, Randy.** Fixed-Point Arithmetic: An Introduction. *Digital Signal Labs.* [En línea] 07 de Julio de 2009. http://www.digitalsignallabs.com/papers.htm.
94. **Oberstar, Erick L.** *Fixed-Point Representation and Fractional Math.* s.l. : Oberstar Consulting, 2007.
95. **Chu, Pong P.** *FPGA prototyping by VHDL examples.* EUA : Jhon Wiley & Sons, 2008.