POSGRADO INTERINSTITUCIONAL DE CIENCIA Y TECNOLOGÍA

ARQUITECTURA VLSI OPTIMIZADA DEL FILTRO KALMAN PARA APLICACIONES DE NAVEGACIÓN INERCIAL

TESIS

QUE PARA OBTENER EL GRADO

ACADÉMICO DE

DOCTOR EN CIENCIA Y TECNOLOGÍA

EN LA ESPECIALIDAD DE

MECATRÓNICA

PRESENTA

M.I.E. RAMÓN CHÁVEZ BRACAMONTES

QUERÉTARO, MÉXICO, DICIEMBRE 2015.

POSGRADO INTERINSTITUCIONAL
CIENCIA Y TECNOLOGÍA

Director de Posgrado
PICYT – CIDESI
Querétaro

Los abajo firmantes, miembros del Comité Tutorial del alumno **Ramón Chávez Bracamontes**, una vez leída y revisada la Tesis titulada **"ARQUITECTURA VLSI OPTIMIZADA DEL FILTRO KALMAN PARA APLICACIONES DE NAVEGACIÓN INERCIAL"**, aceptamos que la referida tesis revisada y corregida sea presentada por el alumno para aspirar al grado de **Doctorado en Ciencia y Tecnología** en la opción terminal de **Mecatrónica** durante el Examen de Grado correspondiente.

Y para que así conste firmo la presente a los 14 días del mes de Diciembre del año dos mil quince.

Dr. Manuel Bandala Sánchez
Director de tesis

Dr. Marco Antonio Gurrola Navarro
Codirector de tesis

CIENCIA Y TECNOLOGÍA

Director de Posgrado
PICYT – CIDESI
Querétaro

Los abajo firmantes, miembros del Jurado del Examen de Grado del alumno **Ramón Chávez Bracamontes**, una vez leída y revisada la Tesis titulada **"ARQUITECTURA VLSI OPTIMIZADA DEL FILTRO KALMAN PARA APLICACIONES DE NAVEGACIÓN INERCIAL"**, aceptamos que la referida tesis revisada y corregida sea presentada por el alumno para aspirar al grado de **Doctorado en Ciencia y Tecnología** en la opción terminal de **Mecatrónica** durante el Examen de Grado correspondiente.

Y para que así conste firmamos la presente a los 14 días del mes de Diciembre del año dos mil quince.

Dr. Jorge Alberto Soto Cajiga
Presidente

Dr. Guillermo Ronquillo Lomelí
Secretario

Dr. Gengis Kang Toledo Ramírez
Vocal 1

Dr. Marco Antonio Gurrola Navarro
Vocal 2

Dr. Manuel Bandala Sánchez
Vocal 3

Resumen

En esta tesis doctoral se presenta el diseño de una arquitectura VLSI del algoritmo filtro de Kalman convencional con la formulación secuencial y la estabilizada de Joseph para aplicaciones en navegación inercial, optimizada en área de silicio y consumo de energía. La arquitectura se definió bajo un esquema parametrizable para *n* estados y abierta en la configuración de los parámetros que definen al sistema dinámico, así como el ajuste de valores de sintonía del filtro. En la arquitectura se emplea aritmética de punto fijo de 24 bits con representación numérica de magnitud y signo. Se fabricó un circuito integrado para un filtro de dos estados con tecnología de proceso CMOS de 0.5 μm de ON Semiconductor que requirió de 70,192 transistores en una área de silicio de 5.6 mm^2. Las pruebas sobre silicio mostraron un consumo de potencia de 1.1 mW a una frecuencia de reloj de $200KHz$ en un tiempo de ejecución por iteración de 750 μs con un rendimiento de 1,333 actualizaciones por segundo pudiendo alcanzar hasta $6.6E-3$ veces la frecuencia de reloj y tiempos de ejecución en el orden de los μs. La arquitectura presentada permite la síntesis del filtro de Kalman de un mayor número de estados cumpliendo con la eficiencia y consumo de energía requerida en las aplicaciones móviles de navegación inercial.

Abstract

A VLSI architecture design of the conventional Kalman filter with the sequential and Joseph stabilized formulation for inertial navigation applications is presented in this doctoral thesis. The design is optimized in silicon area and power consumption. The architecture was design based on the possibility to scale the filter for computing n states. Also, it enables the configuration of the parameters that define the system's dynamics and the filter's tuning. The architecture uses a 24-bit fixed-point arithmetic with magnitude and sign numeric representation. The fabricated integrated circuit estimates two states and was done using the 0.5 um CMOS ON Semiconductor technology that required 70,192 transistors, spread in a silicon area of 5.6 square millimeters. Tests showed a power consumption of 1.1 mW with a clock frequency of 200 kHz, computing one iteration in 750 us or 1,333 updates per second. The performance of the integrated circuit can be improved up to $6.6E-3$ times the clock frequency and computing times in the μs order. Based on the tests' results, the VLSI synthesis for a greater number of states can be done while fulfilling the efficiency requirements of a wide variety of mobile inertial navigation applications.

Agradecimientos

A mi esposa Leticia y mis hijos Ramón, María José y Gabriel, gracias por todos esos momentos en los que sin duda alguna me han apoyado durante este proyecto de vida.

A mis padres, quienes me han enseñado a ver la vida con entusiasmo y deseos de superación.

A mis hermanos y sus familias que nos acompañaron durante esta estancia en Querétaro.

A mi director de tesis el Dr. Bandala, así como mi asesor el Dr. Gurrola, sus orientaciones, consejos y apoyos en todos los sentidos me ayudaron a alcanzar esta gran meta.

A mis maestros y compañeros con quien compartimos grandes momentos y siempre me ofrecieron su ayuda incondicional: David, Alberto, Erwirn, Itzel, Fernando, Salatiel, Pablo, Javier, Claudia, Alan y Humberto.

Al Instituto Tecnológico de Cd. Guzmán por los permisos y prestaciones otorgadas y el CONACYT quien me asignó una beca durante el programa doctoral.

Gracias a Dios.

Índice general

Lista de figuras

Lista de tablas

Capítulo 1

Introducción

1.1. Antecedentes

Una de las áreas tecnológicas que ha generado avances innovadores es el área de microsistemas y sistemas micro-electromecánicos (MEMS) con aplicaciones que abarcan campos tan diversos como la microcirugía, los vehículos autónomos (aéreos, terrestres y sub-acuáticos), las misiones espaciales y los servicios basados en la localización del usuario. Es posible afirmar que las funciones de casi cualquier dispositivo portátil o diseñado para desplazarse en el espacio (autónomamente o no) son susceptibles de ser mejoradas, ampliadas o automatizadas gracias a la inclusión de un sistema de navegación.

El control de sistemas dinámicos como es el caso de los sistemas de navegación requiere del empleo acertado de sensores como los giroscopios y acelerómetros, quienes proveen señales que en la mayoría de los casos presentan características de error como la deriva, inestabilidad en el factor de escala y ruido aleatorio, haciendo difícil la tarea de control del sistema.

Las aplicaciones en el campo de la navegación han tenido un gran avance gracias a las tecnologías digitales de alta escala de integración (VLSI), así como el desarrollo de actuadores y sensores miniaturizados con tecnología MEMS. Estos avances en la electrónica digital han permitido la implementación de sistemas embebidos con una gran capacidad de cómputo basados con microprocesador, microcontrolador, DSP (*Digital Signal Processor*), FPGA (*Field Programmable Gate Array*) o ASIC (*Application Specific Integrated Circuit*).

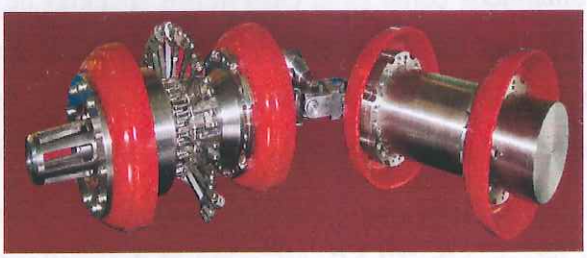
Proyectos para el control de vehículos no tripulados como los mostrados en la Fig. 1.1 son desarrollados en el Centro de Ingeniería y Desarrollo Industrial (CIDESI), con la motivación de

generar productos innovadores con tecnología mexicana y que resuelvan una necesidad del mercado nacional. Una consecuencia importante de esta realidad es la posibilidad de implementar poderosos algoritmos numéricos que estimen en tiempo real los parámetros de navegación y los errores instrumentales. Estos algoritmos, llamados de navegación integrada, emplean métodos de fusión de datos con origen en la teoría del filtrado lineal y no lineal de procesos estocásticos.

Entre los algoritmos más utilizados en problemas de estimación se encuentran el Filtro de Kalman convencional (FK) y diferentes formulaciones como el caso del FK Extendido (EKF), el FK Linealizado (LKF), el FK Sigma Point (SPKF) y el FK Unscented (UKF) entre otros. La principal ventaja de estas técnicas es que, se benefician de la diversidad de fuentes de información instrumental para reducir la incertidumbre de los estados estimados. Una característica importante del FK es la complejidad computacional, que crece polinómicamente en función de los estados de un sistema, es por ello que su implementación por software no es factible para la mayoría de aplicaciones de tiempo real.

(a) Helicóptero no tripulado [1]

(b) ROV [2]



(c) Inspección de ductos [3]

Figura 1.1: Proyectos desarrollados en CIDESI

El proyecto que se presenta se particulariza para aplicaciones de navegación inercial donde existen restricciones de consumo de energía, espacio, peso y es necesario procesar información en tiempo real. En estas aplicaciones predominan los efectos aleatorios en las señales eléctricas.

1.2. Definición del problema

La complejidad computacional del algoritmo del filtro de Kalman crece polinómicamente en función de los estados de un sistema, por lo que su implementación por software no es factible para la mayoría de aplicaciones en tiempo real por cuestiones de tiempo, volumen, peso o consumo de energía en aplicaciones de navegación inercial.

1.3. Objetivos

Objetivo general

Diseñar e implementar en circuito integrado una arquitectura VLSI parametrizable que haga eficiente la solución del algoritmo del FK aplicado a sistemas de navegación inercial.

Objetivos específicos

- Experimentar con IMU-MEMS comerciales para caracterizar los parámetros de navegación.
- Experimentar con esquemas específicos del FK para evaluar los tiempos de ejecución por software y en sistemas embebidos.
- Definir las especificaciones del sistema VLSI a diseñar.
- Diseñar la arquitectura VLSI del FK en forma modular.
- Sintetizar la arquitectura propuesta.
- Verificar la arquitectura propuesta a través de patrones de prueba (Test-bench).
- Generar los layouts requeridos para la fabricación del chip.
- Validar y publicar resultados.

1.4. Justificación

Esta investigación servirá para:

- Obtener una *arquitectura* parametrizable que pueda ser sintetizada y que solucione el FK en tiempo real para aplicaciones de navegación inercial, aprovechando las ventajas de los dispositivos VLSI.
- Lograr una *síntesis VLSI* que permita posteriormente fabricar chips que puedan ser integrados con sensores inerciales.
- Generar desarrollos tecnológicos propios en el área de sensores inteligentes aplicados a sistemas de navegación inercial.
- Lograr una especialización en el área de diseño de circuitos integrados que permita la generación de conocimiento y tecnología en México.

1.5. Hipótesis

Si se logra implementar por hardware VLSI el algoritmo del filtro de Kalman, usando la formulación convencional en combinación con la estabilizada de Joseph y la formulación secuencial, empleando técnicas de diseño lógico que permitan optimizar la micro-arquitectura, entonces se puede mejorar el rendimiento del algoritmo del FK para aplicaciones de navegación inercial en tiempo real en comparación con las implementaciones por software.

Esto permitirá:

- Disminuir 3 ordenes de magnitud en los tiempos de cómputo en las iteraciones del algoritmo, pasando del orden de milisegundos a microsegundos.
- La potencia requerida para resolver el FK estará en el orden de miliwatts en lugar de watts, que típicamente consume una computadora de navegación inercial.
- Optimizar el área de silicio, de tal forma que la implementación en VLSI permita su integración con sensores inerciales para aplicaciones de navegación donde existan considerables restricciones de volumen, peso y consumo de energía.

1.6. Alcances

Los alcances a los cuales se limita este trabajo:

- Diseño de la arquitectura VLSI del FK parametrizable de n estados.
- Síntesis VLSI del FK en un número finito de estados.
- Publicación de dos artículos en revistas indizadas.

1.7. Metodología

Para lograr los objetivos establecidos se proponen los siguientes pasos metodológicos:

1. Revisión bibliográfica del estado del arte sobre la implementación en circuito integrado de algoritmos estimadores para aplicaciones de navegación inercial.
2. Es necesario establecer cuales son las variables y los parámetros a ser estimados, por lo cual se debe hacer una experimentación inicial con sensores inerciales para poder identificar las características y el tipo de ruido involucrado en las aplicaciones de navegación inercial.
3. Una vez que se disponga de la información generada por los sensores es importante hacer un estudio del algoritmo FK enfocado en aplicaciones de navegación inercial, llevando a cabo inicialmente implementaciones por software y posteriormente en sistemas embebidos, con la finalidad de determinar los efectos de la discretización de los datos.
4. Se deben definir los requerimientos computacionales del algoritmo FK para el esquema o formulación seleccionada de forma que se permita definir claramente las especificaciones del CI a diseñar.
5. Una vez definidas las especificaciones del CI se debe continuar con el ciclo de diseño VLSI el cual involucra el diseño de la arquitectura, su diseño lógico, la síntesis VLSI, el diseño físico o layout y las etapas de flujo de verificación correspondiente hasta la obtención de los archivos de diseño que se envían al fabricante.

6. La obtención de la síntesis de la arquitectura propuesta para el CI VLSI que resuelva el algoritmo FK debe permitir reportar resultados parciales en artículos científicos. Los layouts generados en el ciclo de diseño deben permitir la posterior fabricación del CI a través de un centro de servicios de manufactura de CI comercial.

7. Es necesario implementar un banco de pruebas físicas, bajo un entorno controlado, que permita caracterizar y llevar a cabo la validación de la funcionalidad de la arquitectura propuesta y finalmente reportar los resultados finales a través de artículos científicos en revistas arbitradas e indizadas.

El esquema de la Fig. 1.2 muestra la secuencia del flujo entre cada paso metodológico.

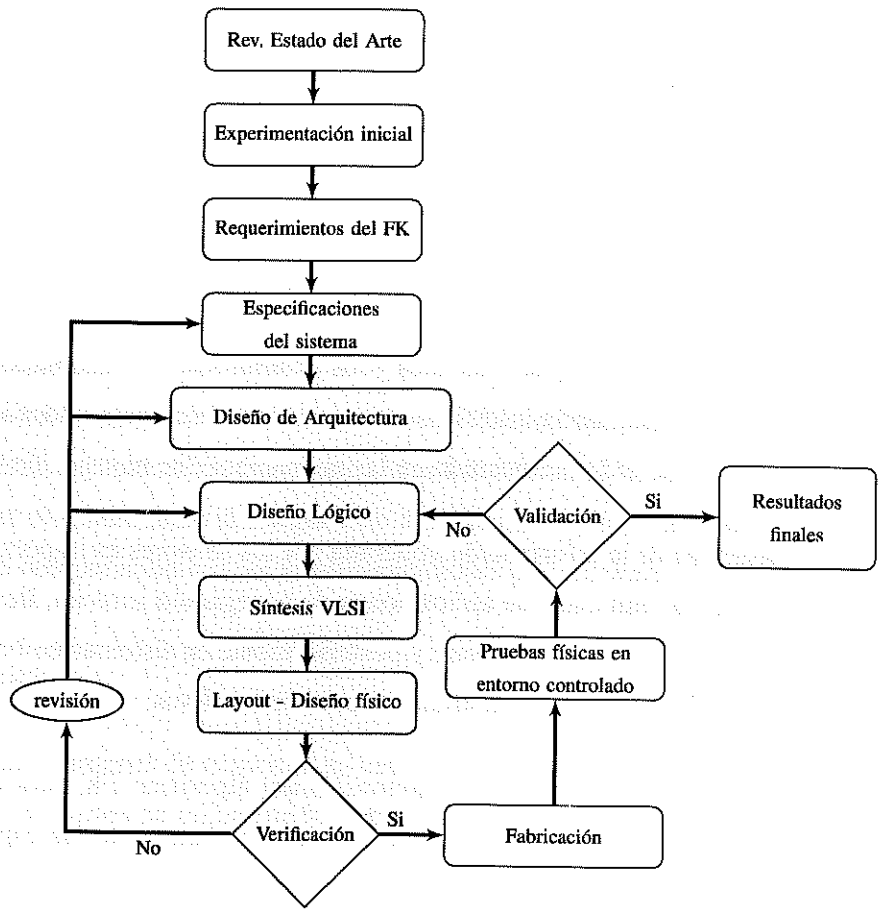


Figura 1.2: Metodología

1.8. Estructura del documento

En el Capítulo 1 se establece el planteamiento del proyecto, partiendo de los antecedentes, la definición del problema, los objetivos, hipótesis, justificación, alcances y metodología.

Los fundamentos del algoritmo FK están definidos en el Capítulo 2, así como una revisión del estado del arte y las formulaciones convencional y secuencial que fueron seleccionadas durante la definición de las especificaciones para el diseño de la arquitectura aquí propuesta.

El Capítulo 3 presenta las tareas comunes que se requieren en la síntesis e implementación de algoritmos en circuitos integrados digitales con tecnología CMOS. Se definen las etapas más importantes en el ciclo de diseño VLSI digital que parten de las especificaciones del sistema, el diseño de la arquitectura, diseño funcional, diseño lógico, la síntesis VLSI, diseño físico, verificación así como la metodología de diseño VLSI con herramientas CAD.

El desarrollo del proyecto se presenta en el Capítulo 4. La primera etapa consistió en evaluar el desempeño de los sensores inerciales tipo MEMS, integrados en Unidades de Medición Inercial (IMU), aquí se describe la experimentación en un banco de pruebas para la adquisición de las señales. Como producto de esta experimentación se obtuvieron las bases de datos con información de los sensores expuestos a diferentes dinámicas de movimiento; esta información posteriormente fue utilizada en las pruebas de simulación y validación del algoritmo.

Una segunda etapa en el desarrollo del proyecto de investigación fue la implementación por software del FK. El análisis del algoritmo haciendo uso de modelos con representación numérica tanto en punto flotante como punto fijo fue fundamental, ya que permitió la definición de aspectos clave en el diseño de la arquitectura VLSI como son el número de bits para el ancho de palabra y el tipo de representación numérica a emplear. Estos parámetros influyen considerablemente en la optimización de recursos computacionales del sistema. Además del análisis del FK a través de herramientas de simulación (como Matlab©), se realizó también su implementación en un sistema embebido empleando un procesador digital de 16 bits, la finalidad de este trabajo se encaminó en la medición de los tiempos de ejecución del algoritmo en tiempo real, los resultados permitieron establecer los parámetros cuantitativos en la definición de la hipótesis. Otro aspecto revisado en este capítulo es el análisis de la complejidad computacional del FK para la formulación convencional y su impacto en la eficiencia del algoritmo.

La tercera etapa del desarrollo del proyecto documenta el diseño del circuito integrado (CI). La implementación en CI comienza con la definición de los requerimientos de memoria y un análisis parametrizado del banco de registros en función del número de estados, entradas de control y

mediciones en el sistema donde fuera aplicado el filtro. Los resultados de la primera y segunda etapa fundamentan las especificaciones del sistema que sirven de base en la definición de la arquitectura VLSI. En esta etapa se sigue una metodología de diseño VLSI convencional, partiendo de la codificación de bloques del sistema empleando el lenguaje de descripción de hardware VHDL. En forma paralela al flujo de diseño se lleva a cabo un flujo de verificación realizando las simulaciones correspondientes en cada bloque del diseño y la síntesis VLSI que nos conduce al diseño físico donde se tiene que verificar el cumplimiento de las reglas de diseño para el proceso de fabricación elegido.

El Capítulo 5 contiene los resultados logrados en las diferentes etapas del desarrollo del proyecto comenzando con los resultados de las implementaciones por software, continuando con los resultados de la síntesis VLSI que sirvieron de base para el desarrollo de la arquitectura, y concluyendo con los resultados de la etapa de validación bajo un entorno controlado. En estos resultados se evalúa el desempeño en términos de las actualizaciones por segundo alcanzadas, consumos de corriente promedio, frecuencias del reloj del sistema, tiempos de ejecución del algoritmo entre otros parámetros. La validación se realiza bajo un entorno controlado para condiciones de aplicaciones en tiempo real mediante el procesamiento de los resultados fuera de linea con fines de comparación contra los esquemas de implementación por software.

En el Capítulo 6 se presentan las conclusiones extraídas de los resultados de experimentación y es seguido por una sección donde se discuten las contribuciones logradas durante la investigación. Posteriormente se mencionan las recomendaciones para futuras investigaciones y el capitulo finaliza con la relación de las publicaciones producto de la investigación.

El apéndice A presenta parte del código empleado durante la síntesis de la arquitectura. El apéndice B describe el banco de pruebas, mientras que en los apéndices del C al G se describe parte de la terminología empleada en los sistemas inerciales, fundamentos de la aritmética computacional, análisis numérico, las reglas de diseño VLSI y finalmente la descripción de proceso de fabricación CMOS y tecnología de proceso C5 de *On Semiconductor* empleado en el diseño del CI que soluciona el algoritmo del FK.

Capítulo 2

Filtro de Kalman

Las señales que provienen de las mediciones en un proceso físico se obtienen a través de sensores que convierten un tipo de energía a otro, por lo general de tipo eléctrico. Estas señales representan la magnitud de una variable medida en el proceso físico relacionada directa o indirectamente con los estados del sistema. Los dispositivos de medición, como por ejemplo los sensores inerciales, generan señales que presentan un comportamiento aleatorio, por lo que deben ser tratadas como estocásticas, ya que es imposible predecir un valor exacto futuro aún si se conocen todos sus valores pasados. Las señales aleatorias son abundantes en los procesos físicos y la mayoría de las mediciones con sensores contienen ruido aleatorio causado por el movimiento de cargas eléctricas en un conductor y por naturaleza estocástica [4]. Existen muchos avances para estimar un estado desconocido partiendo de un conjunto de medidas en un proceso, muchos de estos métodos no toman en consideración la naturaleza ruidosa de las medidas. Este ruido es estadístico por naturaleza (o puede ser modelado como tal), lo cual nos conduce a tratar este problema por métodos estocásticos. El algoritmo del Filtro de Kalman (FK) permite estimar el estado de un sistema usando mediciones que presentan un comportamiento aleatorio y es el más usado en la estimación de parámetros en sistemas dinámicos [5]. La función del FK no es la de un filtro común, cuya idea es la separación de componentes indeseadas, sino la estimación de parámetros a través de variables medibles que son función de las variables de interés principal, estimando la variable independiente como función invertida de la variable dependiente (medible), permitiendo inferir información perdida de mediciones indirectas y ruidosas. El FK es un estimador predictor-corrector que minimiza la covarianza del error en la estimación, propagando el estado actual partiendo del conocimiento de la dinámica del sistema e incluyendo la influencia estadística de perturbaciones dinámicas aleatorias así como los efectos de todas las mediciones pasadas.

2.1. Revisión del estado del arte

El FK es uno de los grandes descubrimientos en la historia de la teoría de la estimación. Rudolf Emil Kalman publicó en 1960 [6] una solución recursiva al problema del filtrado lineal de datos discretos. Kalman introdujo un modelo dinámico que incluía el ruido del proceso. Sus fórmulas para el cálculo de la covarianza del error contribuyeron a la amplia aceptación de sus ideas. La característica secuencial de la solución de Kalman al problema lineal de estimación de parámetros lo hizo ideal para su implementación en computadoras digitales.

En la literatura se han encontrado diferentes formulaciones que han sido propuestas por varios autores para implementar variantes al algoritmo original del FK, la mayoría se ha enfocado a la reducción de la complejidad computacional del algoritmo. A continuación se mencionan las principales aportaciones de acuerdo a un orden cronológico.

Al introducir la linealización de trayectorias, Schmidt [7] demostró que el FK podría ser aplicado a problemas no lineales. En lugar de la linealización de una trayectoria nominal, Schmidt propuso la linealización sobre una trayectoria estimada. Este algoritmo relinealizado fue llamado FK extendido y su primera aplicación fue en la navegación y control de la capsula espacial Apollo, en el programa espacial de la NASA [8]. Otras aplicaciones han sido en el control de sistemas dinámicos complejos tales como procesos de manufactura continua, aviación, barcos o el espacio aéreo. Para estas aplicaciones, no siempre es posible medir las variables que se quieren controlar.

Conforme los investigadores tuvieron más experiencia con el FK, se reportaron fenómenos que provocaban divergencia debido al uso de parámetros sin modelar, a la presencia de no linealidades y a los efectos de redondeo en los cálculos. Las divergencias por el uso de parámetros sin modelado fueron analizadas progresivamente por Soong [9], Nishimura [10], Griffin y Sage [11], entre otros. Estos autores aunque eran conscientes que esto podía ser la causa principal de la divergencia del filtro no fue hasta que se publicó el trabajo de Shlee et al. [12] quien hizo una demostración significativa de la divergencia debido a la incertidumbre a priori. La divergencia debido al redondeo en el cálculo [13] ha jugado un rol importante en el diseño de varios algoritmos de estimación que han buscado mantener la no negatividad y simetría de la covarianza calculada. Una de las primeras propuestas fue la formulación de Joseph [14], donde reemplaza la ecuación de actualización de la medición $(I - K_k H_k)P_k$ por una expresión más general $(I - K_k H_k)P_k(I - K_k H_k)^T + K_k R_k K_k^T$.

Los algoritmos que involucran raíces cuadradas en las matrices presenta una mejor estabilidad numérica en aritmética de precisión finita a expensas de una mayor complejidad computacional. Potter [15] introdujo la factorización con la raíz cuadrada y mostró que para mediciones escalares

era posible factorizar la formula de actualización del FK y reformular la recursión, logrando que la estimación sea menos sensible a errores por redondeo en el cálculo de la ganancia óptima del filtro. Andrews [16] extendió el trabajo de Potter para incluir el ruido del proceso y observó que el algoritmo de actualización de Potter podría ser aplicado a problemas con mediciones correlacionadas. Otro enfoque del algoritmo con raíz cuadrada es el de Golub [17], un analista numérico que estaba interesado en métodos con mayor exactitud y más estables numéricamente, en lugar de modificar la solución del FK como Potter lo hizo, Golub propuso el uso de transformaciones ortogonales. Hanson y Lawson [18] trabajaron sobre el algoritmo detallado del procedimiento de triangularización de Golub. Lo importante de su trabajo es que demostraron la eficacia y confiabilidad de la ortogonalización de Householder [19] en problemas de estimación condicionados pobremente. Cox [20], Dyer y McReynolds [21] revisaron formulaciones con la raíz cuadrada para incluir los efectos del ruido de proceso. Uno de los beneficios inesperados provenientes del enfoque Dyer-McReynolds a la estimación fue que las covarianzas y estimaciones asociadas se pueden obtener con un muy modesto incremento del cálculo. Kaminski [22], Bryson [23], Morf y Kailath [24] establecieron una confiabilidad y utilidad del FK de información de la raíz cuadrada secuencial. Otra formulación importante fue introducida por Bierman [25] como otra forma de incrementar la precisión numérica del FK usando la factorización U-D. Esta incrementa la complejidad computacional pero en menor proporción que la formulación de la raíz cuadrada.

A principios de los 80's con el advenimiento de los CI VLSI, varios autores comenzaron a proponer arquitecturas que aprovecharan las características de estos dispositivos. Kung [26] describió una arquitectura multiprocesador denominándola *arreglo sistólico* que recibió mucha atención por su eficiencia en la evaluación paralela de ecuaciones matriciales. Graham y Kadela [27] evaluaron arquitecturas de arreglos sistólicos rectangulares, lineales, y estructuras en arbol. Sung y Hu [28] propusieron un arreglo de procesadores VLSI tipo sistólico como un bloque de construcción básico para acelerar las operaciones matriciales requeridas en cada iteración sobre una implementación paralela de la formulación de la raíz cuadrada del FK. Scharf y Sigurdsson [29] realizaron una implementación de punto fijo del algoritmo *Fast Kalman Gain* haciendo uso de microprocesadores de 16 bits. Jover y Kailath [30] presentaron una arquitectura paralela para hacer más eficientes las ecuaciones de medición y actualización usando la técnica de la raíz cuadrada. El-Hawary y Ravindranath [31] emplearon un esquema paralelo enfocado al problema de exploración acústica bajo el agua. Kung et al. [32] desarrollaron una metodología de mapeo de algoritmos en arreglos sistólicos con la técnica de descomposición QR, la cual presentó una desventaja al ser difícil su implementación en hardware. Behera [33] trabajó en una arquitectura segmentada (*pipelined*).

Durante la decada de los 90's, siguió la tendencia hacia la implementacion del FK con arreglos sistólicos, Gaston et al. [34] compararon diferentes arquitecturas sistólicas, Rao y Bayoumi [35] emplearon el algoritmo Faddev explotando el paralelismo a nivel de algoritmo y arquitectura. Kung y Hwang [36] propusieron diseños basados en arreglos triangulares, Bayoumi et al. [37] implementaron una arreglo bitrapezoidal, Youmin et al. [38] usaron la técnica de factorización U-D y mejoraron la eficiencia empleando una computadora paralela. Moonen [39] empleó un arreglo sistólico tipo Jacobi, Fayomi et al. [40] aplicaron un arreglo semi-sistólico, Massicotte [41] trabajó sobre una arquitectura aplicada a la reconstrucción de señales y la comparó contra una implementación en DSP logrando reducir los tiempos de ejecución considerablemente, Sakkay et al. [42] y Mozipo [43] continuaron con los trabajos de Massicotte (la mayor parte de estos resultados solo fueron validados con herramientas de simulación en VHDL). Mozipo reportó una implementación con un ancho de palabra de 20 bits sobre una tecnología CMOS de 0.5 μm, que utiliza aproximadamente 750,000 transistores para un FK de 3 estados.

Kailath et al. [44] describió un algoritmo rápido del FK por medio de un conjunto diferente de recursiones conocidas como Chandrasekhar–Kailath–Morf–Sidhu (CKMS) para remplazar las recursiones de Riccati en el FK convencional y así reducir la complejidad computacional del algoritmo. Santha y Vaidehi [45] propusieron una arquitectura paralela segmentada y la validaron por simulación VHDL aplicándola a ecualización adaptiva. Yao [46] hizo una comparación de diferentes arquitecturas sistolicas respecto al número de procesadores utilizados y el número de pasos en cada iteración. Grewal y Kain [47] propusieron una formulación del FK que denominaron SigmaRho que tiene los beneficios de los algoritmos de raíz cuadrada tanto en estabilidad como en la reducción del rango dinámico, siendo factible su implementación en aplicaciones embebidas.

En resumen, en la literatura revisada los autores se enfocan a diferentes formulaciones del FK basadas principalmente en la propagación de la covarianza del error, la propagación de la matriz de información, la raíz cuadrada de la matriz de covarianza y la raíz cuadrada de la matriz de información. La mayoría tiene el enfoque de disminuir la complejidad computacional del algoritmo ($O(n^3) \Rightarrow O(n^2) \Rightarrow O(n)$) y en mejorar la estabilidad numérica y convergencia del algoritmo a través de la implementación con el uso de arquitecturas paralelas con arreglos sistólicos que utilizan técnicas como la descomposición QR, el algoritmo Faddev, los arreglos Bi-Tri Trapezoidales, y la descomposición de Cholesky entre otras. Se encuentra que la mayoría son trabajos que reportan resultados de simulación de los esquemas propuestos para ser implementados por software en computadoras de alto rendimiento, y existen muy pocos trabajos reportados sobre implementaciones en hardware que se preocupen por la optimización de los recursos computacionales.

2.2. Fundamentos del FK convencional

Las mediciones en un proceso deben tener en cuenta el ruido de los sensores. Cada tipo de sensor tiene limitaciones fundamentales relacionadas con el medio físico asociado, y al sobrepasar estas limitaciones las señales se degradan. Además, el sensor y los circuitos eléctricos agregan alguna cantidad de ruido eléctrico aleatorio. La variación del ruido eléctrico en las señales afecta la cantidad y la calidad de la información.

La teoría del FK se basa en asumir que las características espectrales del proceso que se filtra son conocidas. Se asume que todas las fuentes de ruido son ruido blanco y Gausiano [48] con media cero y desviación estándar conocida. Admitir que el ruido sea Gausiano implica presuponer que su función de densidad tiene la forma de una campana de Gauss y su función de distribución es una Normal. La varianza es una propiedad estadística muy usada en las señales aleatorias, la magnitud de la varianza daría una ponderación de cuánto ruido hay en la señal.

El comportamiento del sistema que se pretende filtrar tiene que corresponder a un modelo lineal. Este modelo describe la evolución en el tiempo de la cantidad que se desea estimar mediante un vector de estado $x_k \epsilon \Re^n$ véase Fig. 2.1, donde n corresponde al número de estados del sistema. La transición entre estados ($x_k \leftarrow x_{k-1}$), se caracteriza por la matriz de transición $\Phi_{k-1} \epsilon \Re^n$, que contiene los coeficientes que definen al sistema , y la adición de un ruido Gaussiano $w_k \epsilon \Re^n$ representado por un vector que tiene media cero y una matriz de covarianza $Q_k \epsilon \Re^n$. La matriz Φ_{k-1} relaciona el estado del paso previo $(k-1)$ con el estado en el paso actual (k) en ausencia de una función controlante o una perturbación de proceso. El sistema puede o no incluir entradas de control $u_k \epsilon \Re^m$ a través de la definición de la matriz de control $G_{k-1} \epsilon \Re^{nm}$ siendo m el número de entradas de control. El modelo de medición está definido por el vector $z \epsilon \Re^r$ y la matriz $H \epsilon \Re^{rn}$, donde r representa al número de mediciones.

2.3. Formulación convencional del FK

El modelo lineal discreto de la Fig. 2.1 puede ser modelado mediante un sistema de ecuaciones diferenciales lineales:

$$x_k = \Phi_{k-1} x_{k-1} + G_{k-1} u_{k-1} + w_{k-1}, \tag{2.1}$$

$$z_k = H_k x_k + v_k. \tag{2.2}$$

Es posible que el modelo del sistema representado por Φ y G no sea correcto, bien por error

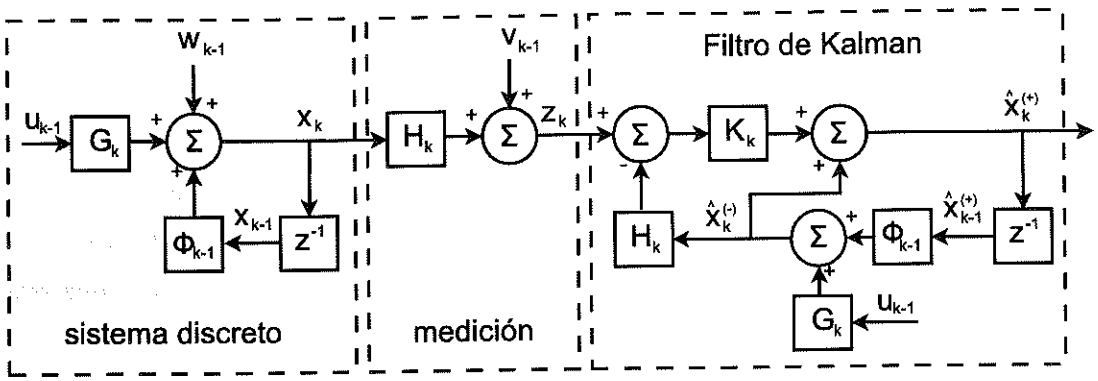


Figura 2.1: Modelo del sistema lineal discreto.

o bien por desconocimiento. También el hecho de discretizar los valores del vector de estado y las matrices del sistema, supone admitir que se cometerán errores numéricos en la resolución de la Ecuación (2.1). Todos estos errores se modelan mediante w_k, en forma de ruido blanco y Gausiano. Del mismo modo, se entiende que los sensores utilizados para obtener los valores de z_k en cada instante de tiempo no son perfectos; esto es que las lecturas que se obtienen de ellos serán inexactas. Asimismo, se cometerán errores de discretización , y aparecerán errores numéricos en los algoritmos necesarios para resolver la Ecuación (2.2). Estos errores se engloban nuevamente en el ruido blanco y Gausiano v_k.

La formulación convencional está basada en la propagación de la covarianza del error P y para su solución se requieren operaciones básicas matriciales de suma, resta, multiplicación, matriz inversa y transpuesta.

El algoritmo se evalúa en dos etapas, la primera conocida como *predicción* ó *time update* donde se calcula una estimación a priori del estado del sistema en el instante k, basada en el conocimiento del sistema en los instantes anteriores a la medición. Esta estimación se denota por $\hat{x}_k^{(-)}$ y no tiene en cuenta la información procedente de los sensores, se basa únicamente en el conocimiento parcial e inexacto del proceso a través de Φ y G.

La segunda etapa de la iteración del algoritmo conocida como *corrección* ó *measurement update*, consiste en calcular la estimación a posteriori del estado $\hat{x}_k^{(+)}$ utilizando la estimación a priori de la lectura de los sensores en el instante k y una matriz K conocida como ganancia de Kalman, que decide en cada instante de tiempo, cuanto peso tiene el conocimiento del comportamiento dinámico del sistema contenido en $\hat{x}_k^{(-)}$, y cuanto tienen las medidas de los sensores contenida en z_k.

Predicción (time update)

$$\hat{x}_k^{(-)} = \Phi_{k-1}\hat{x}_{k-1}^{(+)} + G_{k-1}u_{k-1}, \tag{2.3}$$
$$P_k^{(-)} = \Phi_{k-1}P_{k-1}^{(+)}\Phi_{k-1}^T + Q_{k-1}, \tag{2.4}$$

Corrección (measurement update)

$$K_k = P_k^{(-)}H_k^T(H_kP_k^{(-)}H^T + R_k)^{-1}, \tag{2.5}$$
$$\hat{x}_k^{(+)} = \hat{x}_k^{(-)} + K_k(z_k - H_k\hat{x}_k^{(-)}), \tag{2.6}$$
$$P_k^{(+)} = (I - K_kH_k)P_k^{(-)}. \tag{2.7}$$

Ecuaciones de covarianzas

$$Q_k = E(w_kw_k^T), \tag{2.8}$$
$$R_k = E(v_kv_k^T), \tag{2.9}$$
$$E\{w_kw_j^T\} = 0, \bigtriangledown k \neq j \text{ y } E\{v_kv_j^T\} = 0, \bigtriangledown k \neq j \tag{2.10}$$
$$P_k^{(-)} = E\{(x_k - \hat{x}_k^-)(x_k - \hat{x}_k^-)^T\} \tag{2.11}$$
$$P_k^{(+)} = E\{(x_k - \hat{x}_k^+)(x_k - \hat{x}_k^+)^T\} \tag{2.12}$$

Donde:

Q_k representa la covarianza del ruido del proceso y R_k la covarianza del ruido de medición.

x_o denota la estimación inicial antes de que estén disponibles las mediciones, por lo que $x_o^+ = E(x_o)$.

P_o es la covarianza inicial y representa la incertidumbre de la estimación inicial.

$p(w)$, $p(v)$ se asume que tienen una distribución tipo Gausiana $\mathcal{N}(0, Q_k)$ y $\mathcal{N}(0, R_k)$ respectivamente.

Un aspecto importante del FK es que el cálculo de $P_k^{(-)}$, K_k, y $P_k^{(+)}$ no depende de las mediciones z_k, sino dependen únicamente de los parámetros del sistema Φ, H, Q, R. Esto significa que la ganancia del filtro de Kalman (K_k) se pueda calcular y almacenar en memoria, fuera de linea, antes de que el sistema opere en linea. Luego, cuando el sistema opere en tiempo real, solamente se implemente la ecuación para $\hat{x}_k$, ahorrándose el esfuerzo computacional para calcular K_k durante la operación en tiempo real. La Fig. 2.2 muestra la dependencia entre operaciones para el algoritmo del FK evaluado en dos etapas, *predicción* y *corrección*.

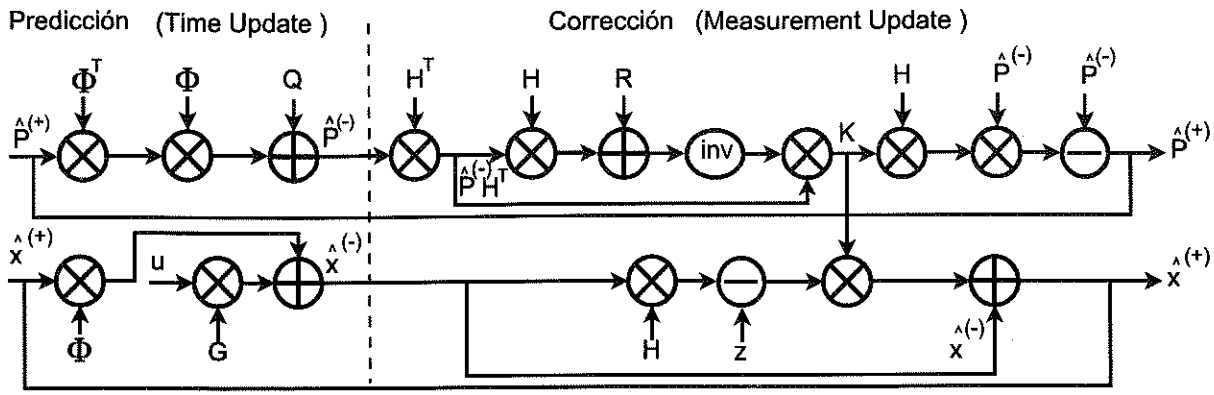


Figura 2.2: Dependencia de operaciones en el algoritmo FK.

En el lazo inferior se calcula la estimación a priori $\hat{x}_k^{(-)}$, posteriormente para corregir la estimación $\hat{x}_k^{(+)}$ se usa la diferencia entre el valor medido y el esperado $[z_k - H\hat{x}_k^{(-)}]$. En el lazo superior, se calcula la ganancia (K_k) y se utiliza en el lazo inferior para corregir los errores introducidos en la estimación tanto por efectos de redondeo como por efectos de ruido.

El lazo superior resuelve la ecuación de Riccati [5], como resultado se obtiene la matriz de covarianza del error de estimación $P_k^{(+)}$, y como resultado intermedio se calcula la ganancia de Kalman K_k. En este lazo no hay forma de detectar y corregir los errores por efectos de redondeo ya que no existe referencia externa para corregir la estimación de P_k, por lo que este error se va propagando y acumulando libremente. Este lazo incluye más operaciones que el lazo de estimación por lo que el cálculo de la ganancia K_k es más susceptible a fuentes de error por redondeo.

La principal causa de degradación de la eficiencia del FK es cuando se presenta asimetría de la matriz de covarianza del error P_k generando un síntoma de degradación numérica y una causa de inestabilidad numérica. Joseph [49] por razones de estabilidad numérica propuso reescribir la ecuación de covarianza (2.7) de la siguiente manera:

$$P_k^{(+)} = [I - K_k H_k] P_k^{(-)} [I - K_k H_k]^T + K_k R_k K_k^T. \tag{2.13}$$

El cálculo de P_k propuesto por Joseph ayuda a simetrizar la matriz de covarianza del error mejorando el resultado de la estimación.

2.4. Formulación secuencial del FK

La formulación secuencial tiene una gran similitud al FK convencional, con la ventaja de que en su solución se evita el cálculo de la matriz inversa. Esta es una formulación atractiva para sistemas embebidos en los cuales el costo computacional es primordial. Sin embargo, el filtrado secuencial solamente se puede usar si la matriz de covarianza del ruido es diagonal o constante. Esta es una manera de implementación del FK sin la matriz inversa, el cálculo de K_k requiere la inversión de una matriz de $r \times r$, donde r es el número de mediciones.

Supongamos que en lugar de medir z_k en el tiempo k, obtenemos r mediciones separadas en tiempo k. Esto es, primero medimos $z_k(1)$, luego $z_k(2)$, ..., y finalmente $z_k(r)$. Se usará la notación z_{ik} para el $i - \acute{e}simo$ elemento del vector de medición z_k.

Supongamos por ahora que R_k es diagonal; esto es, R_k está dada como

$$R_k = \begin{bmatrix} R_{1k} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & R_{rk} \end{bmatrix};$$

Donde H_{ik} es el $i - \acute{e}simo$ renglón de H_k, y v_{ik} es el $i - \acute{e}simo$ elemento de v_k y se obtiene

$$\begin{aligned} z_{ik} &= H_{ik}x_k + v_{ik} \\ z_{ik} &\sim \mathcal{N}(0, R_{ik}) \end{aligned}$$

Así que en lugar de procesar las mediciones en el momento k como un vector, se implementan las ecuaciones de predicción del FK una medición a la vez. Se usará la notación de K_{ik} para la ganancia de Kalman que se usa para procesar el $i - \acute{e}simo$ elemento en el tiempo k, $\hat{x}_{ik}^{(+)}$ es la estimación óptima después que la medición ha sido procesada en el tiempo k, y $P_{ik}^{(+)}$ es la covarianza del error de estimación después que se ha procesado la $i - \acute{e}simo$ medición en el tiempo k.

De estas definiciones se puede ver que

$$\begin{aligned} \hat{x}_{0k}^{(+)} &= \hat{x}_k^{(-)} \\ P_{0k}^{(+)} &= P_k^{(-)} \end{aligned}$$

Esto es, $\hat{x}_{0k}^{(+)}$ es la estimación después de que cero mediciones han sido procesadas, por lo que es igual a la estimación a priori. De manera similar, $P_{0k}^{(+)}$ es la covarianza de la estimación del error después que han sido procesadas cero mediciones, por lo que es igual a la estimación a priori de la

covarianza del error. La ganancia K_{ik} y la covarianza $P_{ik}^{(+)}$ son obtenidas de las Ecuaciones (2.3) y (2.6) del FK con el entendido que ellas aplican a las mediciones escalares z_{ik}.

Para $i = 1, ..., r$ tenemos:

$$K_{ik} = P_{i-1,k}^{(+)} H_{ik}^T [H_{ik} P_{i-1,k}^{(+)} H_{ik}^T + R_{ik}]^{-1} \quad (2.14)$$

$$\hat{x}_{ik}^{(+)} = \hat{x}_{ik}^{(-)} + K_{ik}(z_{ik} - H_{ik}\hat{x}_{i-1,k}^{(+)}) \quad (2.15)$$

$$P_{ik}^{(+)} = (I - K_{ik}H_{ik})P_{i-1,k}^{(-)} \quad (2.16)$$

Después que todas las mediciones son procesadas, podemos hacer $\hat{x}_k^{(+)} = \hat{x}_{rk}^{(+)}$ y $P_k^{(+)} = P_{rk}^{(+)}$, y tenemos nuestra estimación a posteriori y covarianza del error en el tiempo k. El filtro de Kalman secuencial no requiere ninguna matriz inversa porque todas las expresiones en la Ecuaciones (2.14)-(2.16) son operaciones escalares.

El FK convencional requiere la inversión de una matriz de $r \times r$, donde r es el número de mediciones. El filtro de información requiere al menos un par de matrices inversas de $n \times n$, donde n es el número de estados. Por lo tanto si r es mucho mayor que n (teniendo más mediciones que estados) podría ser computacionalmente más eficiente usar el filtro de información.

Ya sea si usamos el FK convencional o el filtro de información la ganancia de Kalman está dada por $K_k = P_k^{(+)} H_k^T R_k^{-1}$, por lo que tenemos que resolver la inversa de R_k. Pero si R_k es constante, podríamos invertirla en el proceso de inicialización, y así la ecuación de la ganancia podría no requerir la solución de la matriz inversa. Lo mismo aplica para la inversión de Q_{k-1}.

Si la incertidumbre inicial es infinita, no se puede ajustar numéricamente $P_0^{(+)} = \infty$, pero se puede ajustar numéricamente $I_0^{(+)} = 0$. Esto hace que el filtro de información sea matemáticamente preciso para ciertos casos de condiciones iniciales. Sin embargo, si la incertidumbre inicial es cero (conocimiento perfecto de x_0), se puede ajustar numéricamente $P_0^{(+)} = 0$, pero no se puede ajustar numéricamente $I_0^{(+)} = \infty$. Esto hace el FK convencional más preciso matemáticamente para casos donde la incertidumbre inicial es cero.

2.5. Análisis de la complejidad computacional para el FK

Para evaluar la complejidad computacional $O(f(n))$, es necesario determinar el polinomio $f(n)$ que represente las operaciones a realizar durante la ejecución del algoritmo del FK. En el Apéndice D se puede encontrar una definición detallada.

Analizando la formulación del algoritmo del FK convencional definido en (2.5), (2.6), y (2.7), podemos seccionar cada una de las operaciones matriciales necesarias para su solución. En la Tabla 2.1 se deducen las operaciones matriciales de suma, resta, multiplicación e inverso multiplicativo. Para llevar a cabo la transpuesta no se consideran operaciones ya que su implementación por hardware se realiza en base al acceso directo a las matrices.

Las expresión $O(n^3)$, representa las operaciones requeridas para llevar a cabo el cálculo de una iteración del FK. La complejidad computacional crece en forma asintótica, donde el orden del polinomio depende específicamente del tipo de formulación empleada para resolver el algoritmo, para el FK convencional es de orden cúbico, mientras que Gaston et al. [34] reportan formulaciones con complejidad computacional de orden cuadrático y lineal.

En la Tabla 2.1 se deducen las operaciones matriciales de suma, resta, multiplicación y recíproco.

La etapa de predicción se lleva a cabo con los pasos 1-3 que resuelven la covarianza del error a priori $P_k^{(-)}$ (Ecuación 2.4) y los pasos 4-5 calculan la estimación a priori $\hat{x}_k^{(-)}$ (Ecuación 2.3). La etapa de corrección se lleva a cabo con los pasos 6-17, de los cuales del paso 6-10 evalúan la ganancia de Kalman (Ecuación 2.5), los pasos 11-14 calculan el vector de estimación a posteriori $\hat{x}_k^{(+)}$ (Ecuación 2.6) y los pasos 15-17 evalúan la covarianza del error a posteriori $P_k^{(+)}$ (Ecuación 2.7). Para el caso del FK secuencial los pasos 6-17 se deben realizar secuencialmente r veces. Los pasos 18-22 recalculan P con la formulación de Joseph definida en la Ecuación 2.13.

En las Tablas 2.2, 2.3 y 2.4 se resumen las expresiones de los polinomios representativos del número de ciclos de reloj de acuerdo a las etapas de predicción y corrección. Evaluando los polinomios que representan la complejidad computacional se pueden deducir la cantidad de ciclos de reloj requeridos para llevar a cabo una iteración del algoritmo FK sin la reformulación de Joseph, quedando como:

$$O(n^3) = (2n^3 + 5n^2 + nm + m - n) + r(n^3 + 5n^2 + 4nr + 3nr^2 + 2n^2r + 26r)$$

Considerando la reformulación de Joseph la complejidad computacional queda como:

$$O(n^3) = (2n^3 + 5n^2 + nm + m - n) + r(2n^3 + 11n^2 + 4nr + 3nr^2 + 4n^2r + 26r)$$

Las Tablas 2.5 y 2.6 presentan el número de ciclos de reloj requeridos para ejecutar una iteración del FK en función de número de estados (n) y la cantidad de mediciones (r) en una iteración.

Tabla 2.1: Cálculo de operaciones aritméticas para el FK.

Paso.	Operación	$(\times)$	$(+)$	$(-)$	(x^{-1})	Destino
1	$P_k\,\Phi^T$	n^3	n^2			T_{nn}
2	$\Phi\ P_k\Phi^T$	n^3	n^2			P_{nn}
3	$\Phi P_k\Phi^T + Q_k$		n^2			P_{nn}
4	$\Phi\,\hat{x}_k$	n^2	$n(n-1)$			T_n
5	$G\,u_k$	nm	m			X_n
6	$P_k\,H_k^T$	n^2	$n(n-1)r$			S_{nr}
7	$H_k\,P_kH_k^T$	nr^2	$(n-1)r^2$			Q_{rr}
8	$H_kP_kH_k^T\ +\ R_k$		r^2			D_{rr}
9	$(H_kP_kH_k^T + R_k)^{-1}$				$25r$	Q_{rr}
10	$P_kH_k^T\,(H_kP_kH_k^T + R_k)^{-1}$	nr^2	$n(r-1)$			K_{nr}
11	$H_k\hat{x}_k$	nr	$r(n-1)$			Q_{rr}
12	$z_k - H_k\hat{x}_k$			r		Q_r
13	$K_k(z_k - H_k\hat{x}_k)$	nr	$n(r-1)$			T_r
14	$\hat{x}_k^- + K_k(z_k - H_k\hat{x}_k)$		n			$\hat{x}_k$
15	$K_k\,H_k$	n^2r	n^2			T_{nn}
16	$K_kH_k\,P_k$	n^3	n^2			S_{nn}
17	$P_k\ -\ K_kH_kP_k$		n^2	n^2		P_{nn}
18	$I\ -\ K_kH_k$		n^2	n^2		S_{nr}
19	$[I\ -\ K_kH_k]P_k[I\ -\ K_kH_k]^T$	n^3	n^2			T_{nn}
20	$K_k\,R_k$	n^2r	n^2			S_{nn}
21	$K_kR_k\,K^T$	n^2r	n^2			P_{nn}
22	$[I - K_kH_k]P_k^{(-)}[I - K_kH_k]^T + K_kR_kK_k^T$		n^2			P_{nn}

Tabla 2.2: $O(n^3)$ para la etapa de predicción del FK.

Operación	Polinomio representativo
Multiplicación	$2n^3 + n^2 + nm$
Suma	$4n^2 - n + m$
$O(n^3)$	$2n^3 + 5n^2 + nm + m - n$

Tabla 2.3: $O(n^3)$ para la etapa de corrección del FK.

Operación	Polinomio representativo
Multiplicación	$n^3 + n^2 + 2nr + 2nr^2 + n^2r$
Suma	$3n^2 - n + 2rn + nr^2 + n^2r$
Resta	$n^2 + r$
reciproco	$25r$
$O(n^3)$	$n^3 + 5n^2 + 4nr + 3nr^2 + 2n^2r + 26r$

Tabla 2.4: $O(n^3)$ para etapa de corrección incluyendo la reformulación de Joseph.

Operación	Polinomio representativo
Multiplicación	$2n^3 + n^2 + 2nr + 2nr^2 + 3n^2r$
Suma	$8n^2 - n + 2rn + nr^2 + n^2r$
Resta	$2n^2 + r$
reciproco	$25r$
$O(n^3)$	$2n^3 + 11n^2 + 4nr + 3nr^2 + 4n^2r + 26r$

Los resultados muestran un crecimiento polinómico de tercer orden. Para estimar el tiempo de ejecución por iteración del FK bajo una frecuencia de reloj de 20 MHz se tabularon (Tabla 2.7) un conjunto de valores para un rango de 2 a 10 estados requiriendo entre 5.7 μs y hasta 305 μs respectivamente. Estos resultados muestran como se eleva el tiempo de ejecución del FK para una iteración en función de los estados y mediciones.

Tabla 2.5: Número de ciclos de reloj por iteración del FK.

	Salidas (r)									
n	1	2	3	4	5	6	7	8	9	10
2	113	309	661	1205	1977	3013	4349	6021	8065	10517
3	237	540	1063	1860	2985	4492	6435	8868	11845	15420
4	439	889	1631	2737	4279	6329	8959	12241	16247	21049
5	737	1380	2395	3872	5901	8572	11975	16200	21337	27476
6	1149	2037	3385	5301	7893	11269	15537	20805	27181	34773
7	1693	2884	4631	7060	10297	14468	19699	26116	33845	43012
8	2387	3945	6163	9185	13155	18217	24515	32193	41395	52265
9	3249	5244	8011	11712	16509	22564	30039	39096	49897	62604
10	4297	6805	10205	14677	20401	27557	36325	46885	59417	74101

Tabla 2.6: Ciclos de reloj por iteración del FK con formulación de Joseph.

	Salidas (r)									
n	1	2	3	4	5	6	7	8	9	10
2	153	405	829	1461	2337	3493	4965	6789	9001	11637
3	336	774	1468	2472	3840	5626	7884	10668	14032	18030
4	631	1337	2399	3889	5879	8441	11647	15569	20279	25849
5	1062	2130	3670	5772	8526	12022	16350	21600	27862	35226
6	1653	3189	5329	8181	11853	16453	22089	28869	36901	46293
7	2428	4550	7424	11176	15932	21818	28960	37484	47516	59182
8	3411	6249	10003	14817	20835	28201	37059	47553	59827	74025
9	4626	8322	13114	19164	26634	35686	46482	59184	73954	90954
10	6097	10805	16805	24277	33401	44357	57325	72485	90017	110101

Tabla 2.7: Tiempo de ejecución por iteración del FK con una frecuecia de 20 MHz.

	Normal		Joseph	
n	ciclos	tiempo (μs)	ciclos	tiempo (μs)
2	113	5.7	153	7.7
3	237	11.9	336	16.8
4	439	22.0	631	31.6
5	737	36.9	1062	53.1
6	1149	57.5	1653	82.7
7	1693	84.7	2428	121.4
8	2387	119.4	3411	170.6
9	3249	162.5	4626	231.3
10	4297	214.9	6097	304.9

Capítulo 3

Diseño VLSI digital

El diseño VLSI involucra el área digital y el área analógica y en algunas circunstancias la combinación de ambas. El presente trabajo está orientado a el área digital, con la finalidad de implementar el algoritmo del FK en un circuito integrado VLSI.

3.1. Implementación de algoritmos en circuitos integrados

Para la implementación en hardware de un algoritmo es necesario conocer los requerimientos de procesamiento de datos y/o señales. Algunas tareas comunes para la implementación de un algoritmo en hardware involucran:

1. Conseguir una variedad de algoritmos que sea posible implementar en hardware.
2. Delimitar los requerimientos computacionales y de memoria.
3. Encontrar compromisos aceptables entre la complejidad computacional y la exactitud.
4. Analizar los efectos que conlleva el cálculo con palabras de longitud finita.
5. Decidir sobre un número de esquemas de representación numérica.
6. Cuantificar los requerimientos mínimos de recursos computacionales en términos de memoria, ancho de palabra, operaciones lógicas y aritméticas y sus frecuencias de ocurrencia.
7. Otras consideraciones a tomar en cuenta son: relación señal-ruido (SNR), ganancia, factor de compresión de datos, razón de error, etc.

3.1.1. Rendimiento para el algoritmo recursivo del FK

El rendimiento de una implementación del FK está relacionado con las actualizaciones (updates) que se ejecuten por unidad de tiempo. Esto depende de la velocidad del reloj del sistema digital a implementar, en operaciones por segundo, y de la complejidad computacional de la aplicación, en operaciones por actualización.

$$Rendimiento(actualizaciones/segundo) \approx \frac{Velocidad}{complejidad\ computacional} \tag{3.1}$$

Para aumentar el rendimiento se puede reducir la complejidad computacional del algoritmo o eficientar a nivel VLSI la microarquitectura de la implementación a través de:

- Reducir el tiempo de cálculo para lazos críticos.
- En el diseño lógico [50] usar técnicas como look-ahead, ripple-carry, pipeline, *Look-up Table* [51] entre otras.

3.1.2. Posturas que definen la relación costo-beneficio

En el diseño de CI's uno de los factores que más influyen, en la relación de costo-beneficio, es el área de silicio requerida por el diseño en particular. Los costos de la fabricación de un CI están íntimamente relacionados por el área del dado de silicio que ocupa el diseño, es por ello que, este parámetro es uno de los más importantes a considerar durante la etapa de diseño independientemente de la tecnología de proceso empleada en la fabricación.

Otro aspecto importante a tener en cuenta sobre el diseño VLSI es el tiempo de propagación en cada bloque del sistema. Este tiempo de propagación se relacionan con la frecuencia máxima de operación a la que el CI puede operar, trayendo como consecuencia efectos sobre el rendimiento total de operación. Es deseable minimizar los tiempos de propagación para lograr circuitos que funcionen a mayor velocidad.

3.1.3. Estrategias de diseño VLSI

- Una aplicación eficiente involucra diseñar con el hardware mínimo interno y externo además de optimizar el código.
- Para establecer la viabilidad en la implementación por hardware de algoritmos complejos, estos deben ser verificados usando lenguajes de alto nivel tales como C o MatLab.

- El lenguaje de descripción de hardware (HDL) debe ajustarse a las guías de codificación a nivel de transferencia entre registros (RTL).

- Se deben usar las herramientas CAD correctas para minimizar el periodo del ciclo de diseño.

Una solución óptima en el diseño de un circuito integrado es cuando el chip cumple todos sus requerimientos funcionales y usa la menor cantidad de recursos. Además, la tecnología de proceso elegida es la correcta para el proyecto. Aún cuando en la actualidad existe un gran avance en los procesos tecnológicos para la fabricación de CI's, el precio y la complejidad en la integración de cada tecnología se incrementa considerablemente conforme se disminuye el ancho de los transistores más pequeños, concepto conocido con el término *feature size*, definidos actualmente en el orden de micrómetros (μm) y nanómetros (nm). Los nodos tecnológicos más recientes están sobre 40, 22 y 14 nm; sin embargo el costo de estas tecnologías es tan elevado comparado con la tecnologías maduras como la de 0.5 μm (\$1,300 USD/mm^2 en 0.5μm vs. \$22,000 USD/mm^2 en 40 nm), para una referencia en costos consultar *www.europractice-ic.com* . Las tecnologías maduras aunque presentan la ventaja de tener menores costos de fabricación son más lentas y requieren de una mayor área de silicio comparadas con una tecnología nueva. Proyectos estudiantiles como de investigación que no tengan financiamiento para la fabricación pueden acceder sin costo a tecnologías maduras (ON Semiconductor 0.5 μm e IBM 180 nm) a través de programas educativos especiales [52].

3.2. Ciclo de diseño VLSI

El ciclo de diseño VLSI comprende diversas etapas, éstas se pueden agrupar en especificaciones del sistema, diseño de la arquitectura, diseño comportamental o funcional, diseño lógico, síntesis, diseño físico, verificación, fabricación y pruebas del diseño.

3.2.1. Especificaciones del sistema

La especificación del sistema es la representación a alto nivel donde los factores a considerar son la funcionalidad, eficiencia, dimensiones físicas del dado, tecnología de fabricación y tecnología de diseño. Los resultados finales comprenden las especificaciones para el tamaño, velocidad, potencia y funcionalidad del sistema VLSI.

3.2.2. Diseño de la arquitectura

La arquitectura VLSI decide esencialmente sobre los recursos de hardware necesarios y su organización de tal forma que implemente un algoritmo computacional conocido bajo restricciones de eficiencia, costo, potencia y otras impuestas por la aplicación objetivo. El diseño de la arquitectura implica seleccionar una tecnología objetivo y considerar sus posibilidades y limitaciones. Comienza con una abstracción simplificada de la funcionalidad de un circuito, para gradualmente detallar su representación.

El diseño de la arquitectura básica del sistema requiere decisiones sobre el sistema numérico a utilizar (ejemplo: punto fijo o punto flotante), tipos y tamaño de registros, estructuras para la implementación de operaciones aritmética y/o lógicas entre otras. Para ello es necesario:

- Dividir la tarea computacional en vista de su realización en hardware.
- Organizar la interrelación de varias sub-tareas.
- Decidir sobre los recursos de hardware que desarrollen cada sub-tarea.
- Definir la ruta de datos y el controlador.
- Decidir sobre el uso de memoria RAM dentro o fuera del chip y el uso de registros.
- Decidir sobre la topología de comunicación y protocolos (paralelo-serie)
- Definir cuánto paralelismo proveer en hardware.
- Definir si es necesario el uso de técnicas pipeline para proveer paralelismo.
- Definir tipo de circuito y tecnología de fabricación.
- Estimar una primera aproximación del tamaño del circuito y su costo.

Cambios en la microarquitectura son muy significativos para reducir el periodo de reloj de un circuito e incrementar la eficiencia. Una arquitectura *pipelined* puede incrementar la eficiencia de un circuito. Duplicar la lógica en técnicas de paralelismo incrementa el rendimiento del circuito, a costa de un incremento en el área. Este paso concluye con un plano en papel.

3.2.3. Diseño comportamental o funcional

En este paso se deben identificar las principales unidades funcionales del sistema, los requerimientos de interconexión entre unidades. Se debe estimar el área, potencia y otros parámetros de cada unidad. Por ejemplo se podría especificar que se require una multiplicación, pero exactamente no se especifica cómo puede ser ejecutada tal multiplicación. Podríamos usar una gran variedad de topologías de multiplicadores VLSI dependiendo de los requerimientos de velocidad y tamaño de la palabra. Lo importante es especificar el comportamiento en términos de entradas y salidas y tiempos de retardo para cada unidad. Esta información conduce a mejoras del proceso de diseño general y a la reducción de la complejidad de fases subsecuentes.

3.2.4. Diseño lógico

Este paso comienza con la definición de los esquemáticos del sistema a nivel microarquitectura donde se describe la topología o interconectividad de las compuertas. En este punto es cuando se determinan las topologías a usar en la implementación por ejemplo de un sumador carry-look-ahead o un ripple-carry. Posteriormente se hace la descripción a nivel de trasferencia de registros (RTL) y estructural para relizar la arquitectura. La codificación de realiza mediante un lenguaje de descripción de hardware (HDL) tal como VHDL o Verilog.

3.2.5. Síntesis VLSI

El propósito de la síntesis es desarrollar una representación de un circuito basada sobre el diseño lógico. Las expresiones booleanas son convertidas en una representación que tome en cuenta requerimientos de velocidad y área de silicio. Para ello se usan simuladores de circuitos para verificar que no haya errores. Usualmente se expresa en un diagrama detallado del circuito. Este diagrama (*netlist*) muestra los elementos del circuito, celdas, macros, compuertas, transistores e interconexiones entre otros elementos.

La síntesis VLSI es el proceso de conversión desde un alto nivel de abstracción en el dominio comportamental (*behavioral*) a un bajo nivel en el dominio físico a través del dominio estructural. La Tabla 3.1 muestra algunas herramientas de síntesis de alto nivel. Para más detalle sobre herramientas de síntesis y verificación consulte el Apéndice 3.3.1.

Tabla 3.1: Compañías que proveen herramientas de síntesis.

Compañía	Ambiente
Alliance CAD System	Unix
Bels Labs Design Automation	SunOS
Cadence Design System	Unix
Compass Design Automation	Unix
Mentor Graphics	Unix
Synopsys	Unix
Synplicity	Windows
Tanner	Unix, Windows
ViewLogic System	Unix, Windows

3.2.6. Diseño físico

En este paso el *netlist* es convertido a su representación geométrica que es llamado **Layout**. Los detalles exactos del layout dependen de las reglas de diseño (Apéndice F), las cuales son guías basadas en las limitaciones del proceso de fabricación y las propiedades eléctricas de los materiales de fabricación. El diseño físico es un proceso muy complejo y por lo tanto es divido en varios pasos. Se ejecutan varias verificaciones y validaciones durante el diseño físico y puede ser parcial o totalmente automatizado a través de herramientas de síntesis-layout (Apéndice 3.3.1). El diseño físico comienza con la definición del *Floorplan*, explicado a continuación.

Floorplan

El Floorplan es esencial para determinar si un diseño propuesto cabrá en el área del chip asignada. Debe prepararse tan pronto como la arquitectura esté completamente definida.

3.2.7. Verificación del diseño

Para la verificación de diseños digitales se lleva a cabo un proceso de prueba de que el chip que diseñamos ejecuta fielmente las funciones definidas en las especificaciones del sistema. Desde otro punto de vista es un proceso para encontrar problemas de diseño o errores funcionales, los cuales son introducidos sin intención durante el proceso de diseño.

El diseño de una arquitectura se prueba usualmente simulando con VHDL un "*test bench*" para verificar que la lógica es correcta. La verificación funcional sólo se ocupa de las propiedades lógicas y secuenciales del diseño; no tiene en cuenta el tiempo, la disposición (*layout*), la potencia ni consideraciones de fabricación. La verificación del diseño es esencial para identificar los errores antes de mandar a fabricar y comúnmente representa más de la mitad del esfuerzo dedicado al chip.

3.2.8. Fabricación

The MOSIS Service

Después de pasar todas las reglas de diseño, el layout está listo para su fabricación. La fabricación de circuitos integrados es un proceso complejo y costoso, sin embargo, existe un programa auspiciado por la Universidad del Sur de California (USC) [53] que provee a estudiantes e investigadores de universidades la fabricación sin costo de circuitos integrados a través de un programa educacional denominado MEP (MOSIS Educational Program) [54]. MOSIS es una organización que ofrece el servicio a empresas e instituciones gubernamentales. Comenzó en 1981 como un grupo patrocinado por el gobierno a través de DARPA (Defense Advanced Research Projects Agency), actualmente es una organización administrada por la USC.

MOSIS ofrece una amplia variedad de procesos de fabricación de CI's, tecnologías desde 0.5 μm hasta 14 nm para el ancho del transistor más pequeño que se puede fabricar, la mayoría en tecnología CMOS. MOSIS admite 3 conjuntos de reglas de diseño escalables (no específicas al proceso de fabricación): SCMOS (Scalable CMOS), SUBM (Sub-Micron), y DEEP. Para afiliarse a este programa es necesario que la institución que aplique tenga una cuenta con MOSIS. Existen tres tipos de cuentas denominadas "*Instructional, Research* y *Commercial Account*". Las diferencias entre las cuentas son principalmente sobre las dimensiones de los diseños y los tipos de procesos a los que se puede acceder.

TSMC, Globalfoundries, IBM, Austriamicrosystems, ON Semi, son algunos fabricantes disponibles a través de MOSIS para el método MPW (Multi Project Wafer) [54].

TSMC

Los procesos de fabricación de TSMC disponibles a través MOSIS incluyen diseños de: 28 nm, 40/45 nm, 65 nm, 90 nm, y "TinyChip" con las características de baja potencia, bajo y alto voltaje. El acceso está limitado a los titulares de cuentas comerciales MOSIS que son aprobados por TSMC.

GlobalFoundries

Los procesos de fabricación de Globalfoundries, incluyen diseños de 14 nm, 28 nm, 40 nm, 65 nm, 0.13 μm, 0.18 μm y 0.35 μm en la tecnología CMOS. El acceso está limitado a los titulares de cuentas comerciales MOSIS que son aprobados por GlobalFoundries. El tamaño máximo permitido del dado (*die*) es de 12.5 mm x 12.5 mm.

IBM

Los procesos de fabricación de IBM, incluyen diseños de 0.13 μm en tecnología CMOS, en SiGe de 8HP (0.13 μm), 8HP (0.13 μm), y 7WL (0.18 μm), y en SOI de 7RF SOI (0.18 μm). El acceso está limitado a los titulares de cuentas comerciales MOSIS que son aprobados por IBM.

Austriamicrosystems

Los procesos de fabricación de AMS, incluyen diseños de 0.18 μm y 0.35 μm en tecnología CMOS y CMOS de alto voltaje.

ON Semiconductor

ON Semi CMOS incluye procesos de diseños como el I3T80 (0.35 μm), C5 (0.5 μm), y I2T100 (0.7 μm) en tecnología CMOS. Cuenta con varios procesos de fabricación [55], incluyendo los procesos de señal mixta, en la Fig. G.1 se muestra la representación gráfica de los procesos que maneja ON Semi en relación del voltaje y la tecnología.

Las reglas de diseño para este proceso son SUBM utilizando una $\lambda = 0{,}3\ \mu m$. El polisilicio de compuerta es trazado a 2 λ = 0.6 μm, sin embargo después de difundir las zonas de drenaje y de fuente durante la fabricación el canal se ve reducido por $0.1\mu m$. De manera que la dimensión característica final del largo mínimo del transistor es de $0.5\mu m$.

Costos de fabricación

El costo de fabricación a través de MOSIS depende del proceso y la tecnología elegida. Una referencia es de 6,500 dls. por un área de $5mm^2$ para la tecnología de On semiconductor de $0{,}5\mu m$.

Europractice

Europractice es otra empresa similar a MOSIS pero en Europa, que ofrece también servicios de fabricación de circuitos integrados a nivel prototipado de pequeños ASIC para propósitos educacionales y de investigación. El costo de fabricación de una área de $1mm^2$ por medio de los servicios de Europractice puede ser de aproximadamente $\$1,000$ dls. para tecnologías de $0{,}7 \sim 0{,}35\mu m$ y de $\$20,000$ para $40nm$. Más detalles pueden encontrarse en la dirección: www.europractice-ic.com

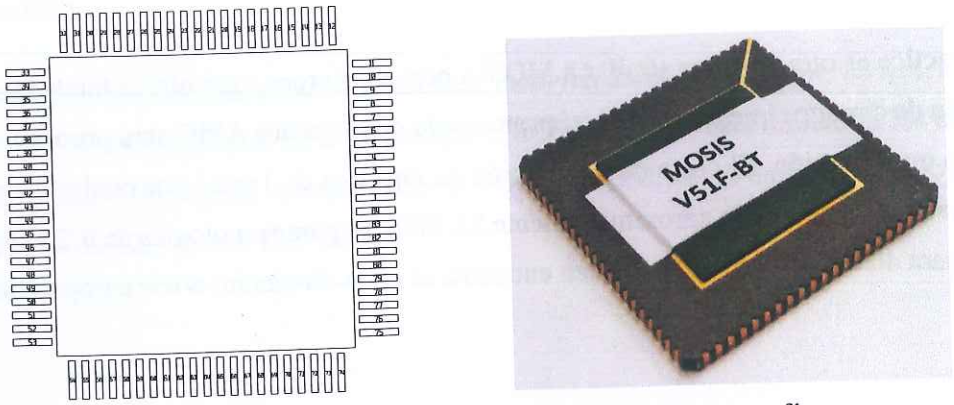
CMP

CMP al igual que MOSIS y Europractice ofrece servicios de prototipado y producción de bajo volumen de circuitos integrados y MEMS. CMP ofrece sus servicios desde inicio de los 90's, enlazando a los fabricantes de CI con empresas pequeñas y medianas que requieren de bajos volúmenes de producción con acceso a avanzadas tecnologías, aún las más costosas. Más detalles pueden encontrarse en la dirección: http://cmp.imag.fr/

Encapsulado y pruebas

Existen una variedad de encapsulados en formatos comerciales como los PLCC, DIP, PGA, QFP, TSOP, BGA (ver Fig. G.3). Para la cuenta instructional, MOSIS provee de máximo 5 chips que pueden ser o no encapsulados. Mosis entrega los encapsulados sin sellar, esto es, se tiene acceso a la cavidad donde se encuentra el dado con la finalidad de inspección y microfotografía del mismo. Los chips no son probados por el fabricante, por lo que el usuario debe hacer las pruebas necesarias para asegurar que el chip fabricado cumple todas las especificaciones de diseño y que funciona apropiadamente. Durante las pruebas pueden presentarse problemas por errores de diseño o errores de manufactura que deben ser reportados a MOSIS. En la cuenta de investigación y comercial el usuario debe pagar los gastos de empaquetado o en su caso solicitar el envío sin encapsular.

Como encapsulado de la implementación descrita en el presente proyecto de investigación, se eligió el encapsulado de tipo cerámico de montaje superficial denominado LCC84M, que dispone de 84 terminales distribuidas por los cuatro lados del CI. La Fig. 3.1 muestra el esquema y una fotografía del encapsulado seleccionado.



(a) Esquema de terminales (b) Fotografía

Figura 3.1: Encapsulado LCC84M.

3.3. Flujo de diseño VLSI con *Alliance CAD System*

El software Alliance CAD System es el nombre de un conjunto completo de herramientas de diseño CAD y bibliotecas de diseño VLSI que fueron desarrolladas en el laboratorio The Pierre et Marie Curie en Paris, Francia [56]. Alliance es un sistema CAD libre y gratuito para el entorno Linux bajo la distribucion GNU-Linux Fedora Electronic Lab. El proceso de diseño VLSI con las herramientas Alliance CAD system requiere del uso de computadoras basadas en microprocesadores i386 o estaciones de trabajo corriendo sobre una plataforma Unix/Linux. El software está bajo una Licencia Pública por lo que los archivos binarios, código fuente y librerías de celdas estándar son distribuciones de uso libre. Los paquetes están disponibles en línea para distribuciones RedHat Entreprise Linux 6 (RHEL6) y clones (Scientific Linux 6, CentOS 6), Fedora y Ubuntu LTS [56]. El software de Alliance provee de independencia tecnológica para que los diseñadores porten sus diseños de un proceso de fabricación a otro.

El flujo de diseño circuitos integrados VLSI comienza con las especificaciones del sistema digital y su descripción a través del lenguaje VHDL (*Very high speed integrated circuit harward description language*). Una vez que se tiene codificado en VHDL la descripción comportamental del sistema digital a diseñar se continúa con las herramientas de Alliance que permiten hacer la síntesis y verificación hasta lograr obtener el plano del CI, a este proceso se le conoce como *Flujo de Diseño* (Fig. 3.2). Las herramientas *vasy*, *asimut*, *boom*, *boog*, *loon*, *ocp*, *nero*, *cougar*, *lvx*, *s2r*, *genlib* y *genpat* se utilizan durante el flujo de síntesis y verificación de circuitos digitales VLSI, para una descripción más detallada refiérase a [57], [58], [59] y [60].

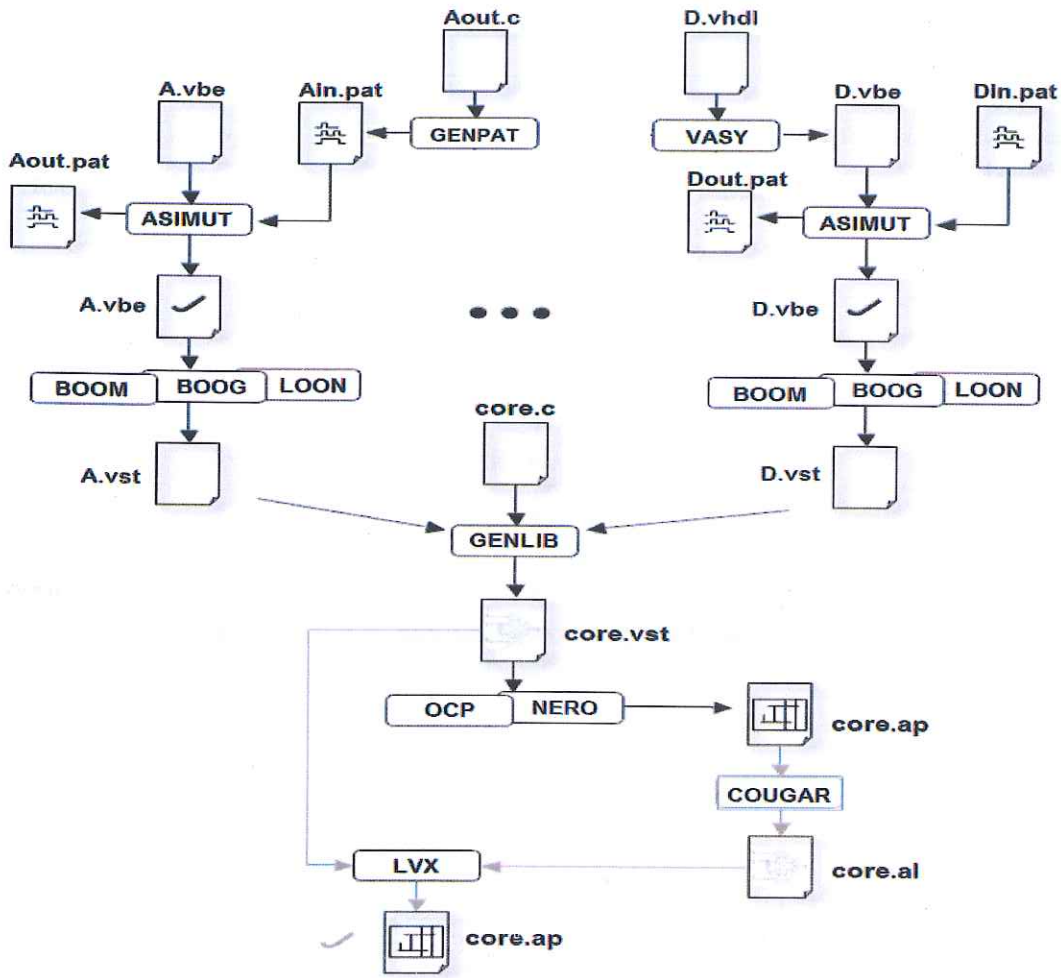


Figura 3.2: Flujo de diseño

3.3.1. Codificación del sistema en VHDL

Las herramientas de Alliance utilizan dos tipos de archivos para la codificación en VHDL. Ellos son archivos con formato *vbe* (VHDL *BEhavioral*) y *vst* (VHDL *STructural*).

Los archivos *vbe* contienen la descripción de un circuito digital de forma comportamental a través de código VHDL. Este archivo se codifica directamente en un editor de texto sin formato con extension *vhdl* o *vhd* y sirve de entrada a la herramienta *vasy* [58].

Código VHDL de un sumador de 1 bit definido en forma comportamental

```
-- Archivo: adder1.vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
entity Adder1 is
    port ( A    : in  Std_Logic ;
           B    : in  Std_Logic ;
           CIN  : in  Std_Logic ;
           COUT : out Std_Logic ;
           S    : out Std_Logic );
end Adder1;
architecture DataFlow OF Adder1 is
begin
    S <= (A AND (NOT B) AND (NOT CIN))+ ((NOT A) AND B AND (NOT CIN)) + ((NOT A) AND (NOT B) ...
        AND CIN) + (A AND B AND CIN);
    COUT <= (A AND B) + (A AND CIN) + (B AND CIN);
end DataFlow;
```

Código VHDL de un sumador de 1 bit para compilar con la herramienta GENLIB

```
#include <genlib.h>
/* Sumador de 1 bit utilizando la celda basica fulladder_x2 */
extern int main()
{
    GENLIB_DEF_LOFIG("fad1");
    GENLIB_LOCON ("A",      IN,     "A");
    GENLIB_LOCON ("B",      IN,     "B");
    GENLIB_LOCON ("CIN",    IN,     "CIN");
    GENLIB_LOCON ("S",      OUT,    "S");
    GENLIB_LOCON ("COUT",   OUT,    "COUT");
    GENLIB_LOCON ("vdd",    IN,     "vdd");
    GENLIB_LOCON ("vss",    IN,     "vss");
    GENLIB_LOINS ("fulladder_x2","fulladder_x2_1", "A", "A", "A", "A", "B", "B", "B", "B", ...
        "CIN", "CIN", "CIN", "COUT",  "S", "vdd", "vss", 0);
    GENLIB_SAVE_LOFIG();
    exit(0);
}
```

Este archivo se codifica directamente en un editor con extension *.c* y se compila con la herramienta GENLIB [61], la cual genera como resultado un archivo con extensión *vst*. Los archivos *vst* contienen la descripción de un circuito digital de forma estructural a través de código VHDL.

3.3.2. Flujo de síntesis

Se llama flujo de síntesis cuando se emplea un lenguaje de descripción de hardware (por ejemplo VHDL) para obtener el diagrama estructural mediante herramientas de síntesis.

VASY (*VHDL Analyzer for Synthesis*)
Convierte un archivo escrito con el subconjunto VHDL de *vasy* al subconjunto VHDL de Alliance. La Fig. 3.3 muestra el uso de los parámetros de entrada y salida para la herramienta *vasy*.

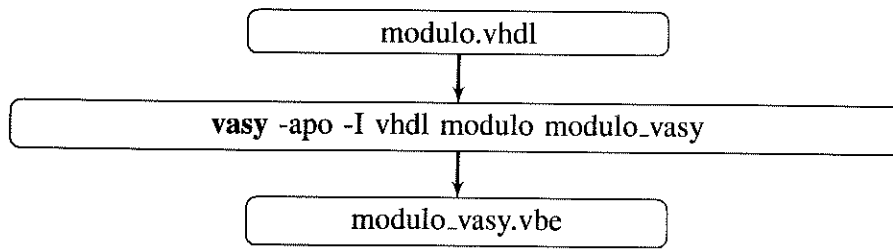


Figura 3.3: Parámetros de entrada y salida para la herramienta *vasy*.

Opciones:

-a Usa una descripción equivalente en formato Alliance *vbe* y/o *vst*. Todos los operadores aritméticos son expandidos a un conjunto de expresiones booleanas equivalentes.

-p Agrega conexiones de alimentación (vdd y vss).

-o Autoriza escribir-reescribir archivos existentes.

-I format Especifica el formato del archivo de entrada (*vbe*, *vst*, *vhd* ó *vhdl*).

Una vez que se genera el archivo *vbe* utilizando la herramienta VASY el siguiente paso es el uso de la herramienta BOOM para realizar una optimización y minimización booleana y enseguida se utiliza la herramienta BOOG que realiza el mapeo hacia las celdas estándar generando como archivo de salida el archivo estructural *vst*.

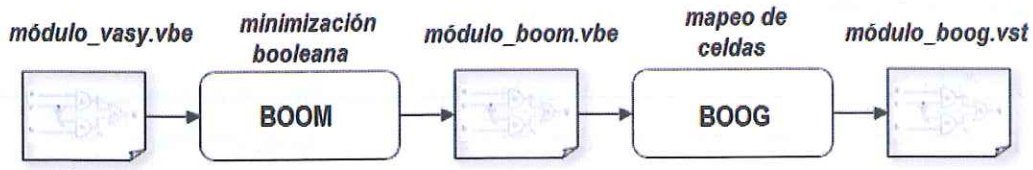


Figura 3.4: Flujo de síntesis (*boom-boog*)

BOOM (*Boolean Minimization*)
Esta herramienta realiza la minimización booleana sobre el archivo entrante escrito en el subconjunto VHDL de Alliance (extension *.vbe*). El archivo de salida también tiene extension *vbe*.

La Fig. 3.5 muestra el uso de los parámetros de entrada y salida para la herramienta *boom*.

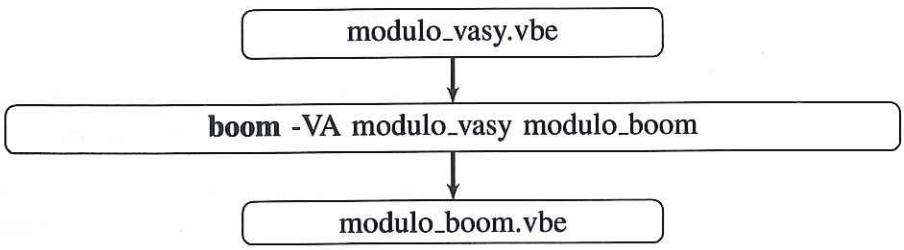


Figura 3.5: Parámetros de entrada y salida para la herramienta *boom*.

BOOM es el primer paso del proceso de síntesis. Optimiza una descripción comportamental usando "*Reduced Ordered Binary Decision Diagram*"para la representación de funciones lógicas.

Opciones:

-V Cada paso de la optimización se despliega en la salida estándar (consola).

-A Boom ejecuta una optimización local y mantiene la arquitectura de la descripción inicial almacenando la mayoría de las señales intermedias.

BOOG (*Binding and Optimizing On Gates*)
Esta herramienta mapea una descripción comportamental descrita en archivo de entrada en VHDL del subconjunto de Alliance (extensión vbe) dentro de una librería de celdas estándar como SXLIB.

Es el segundo paso de la síntesis lógica. El archivo de salida está en formato VHDL estructural de Alliance (extensión *vst*). La Fig. 3.6 muestra el uso de los parámetros de entrada y salida para la herramienta *boog*.

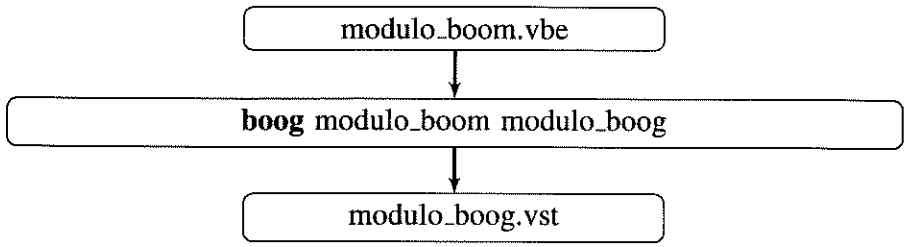


Figura 3.6: Parámetros de entrada y salida para la herramienta *boog*.

LOON (*Local Optimizations Of Nets*)

Esta herramienta cambia algunas celdas para corregir el factor de manejo de corriente, pero conserva la misma función lógica. Tanto el archivo de entrada como el de salida están escritos en VHDL estructural de Alliance (extension vst). La Fig. 3.7 muestra el uso de los parámetros de entrada y salida para la herramienta *loon*.

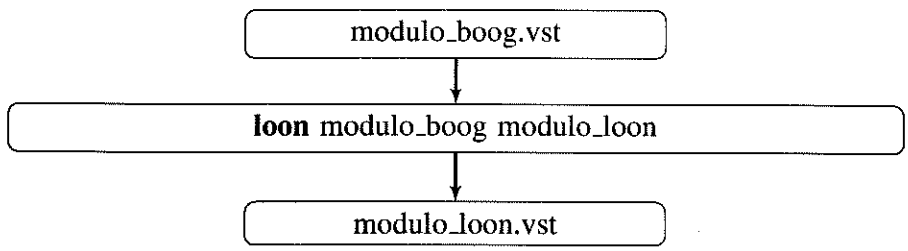


Figura 3.7: Parámetros de entrada y salida para la herramienta *loon*.

Loon es una herramienta CAD que permite remover problemas de FANOUT y optimizar tiempos de retardo, generando salidas con tiempos mejorados.

3.3.3. Flujo de verificación

Antes de aplicar la herramienta de simulación lógica ASIMUT, se genera con la herramienta GENPAT un archivo de patrones de prueba con extension .pat para cada módulo. Este archivo llamado *Test Bench*, contiene la respuesta esperada del módulo ante una secuencia dada de estímulos. La Fig. 3.8 muestra el uso de los parámetros de entrada y salida para la herramienta *genpat*.

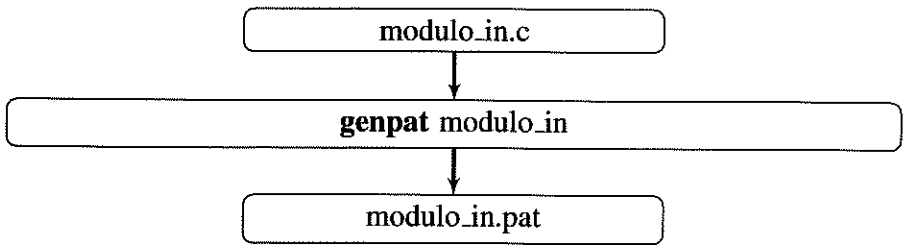


Figura 3.8: Parámetros de entrada y salida para la herramienta *genpat*.

ASIMUT (*A simulation tool for hardware description*) Se aplica sobre archivos de formato vbe y vst para verificar que el archivo de entrada tiene una sintaxis correcta, apegada al subconjunto VHDL de Alliance. Pero su uso mas importante es para comparar el archivo de entrada contra un banco de pruebas preparado de antemano.

La Fig. 3.9 muestra el uso de los parámetros de entrada y salida para la simulación con un archivo comportamental (.vbe).

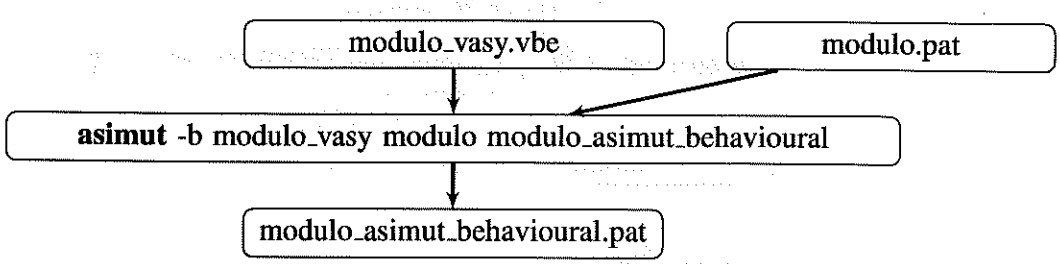


Figura 3.9: Parámetros de entrada y salida para la herramienta *asimut con archivos vbe.*

- modulo_ vasy.vbe - Archivo vhdl comportamental
- modulo.pat - Archivos de patrones (banco de pruebas)
- modulo_ asimut_ behavioural - Archivo de salida (formato pat)

Opciones: -b considera al archivo de entrada de tipo comportamental (vbe).

La Fig. 3.10 muestra el uso de los parámetros de entrada y salida para la simulación con un archivo estructural (.vst).

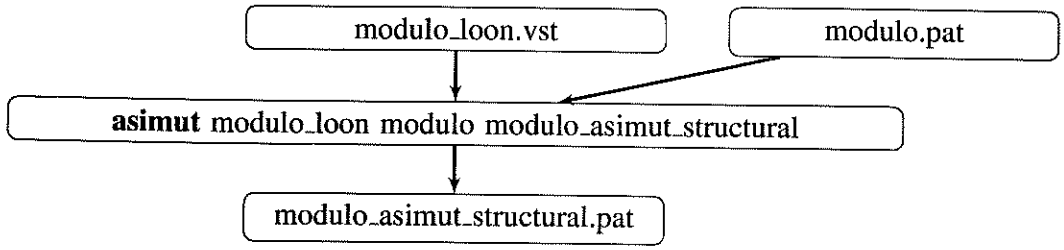


Figura 3.10: Parámetros de entrada y salida para la herramienta *asimut con archivos vst.*

- modulo_ loon.vst - Archivo estructural
- modulo.pat - Archivos de patrones (banco de pruebas)
- modulo_ asimut_ structural - Archivo de salida (formato pat)

Es importante no avanzar a la siguiente herramienta de síntesis si no se realiza la verificación correctamente.

Place & Route

Es el término asociado a las herramientas de diseño físico que permiten colocar y rutear (interconectar) de una manera automatizada las celdas estándar.

OCP (Herramienta de colocación automática de celdas estándar)

Herramienta que recibe un archivo VHDL estructural donde todas las entidades instanciadas corresponden a celdas de la biblioteca previamente indicada. OCP coloca el plano layout de cada celda instanciada en un plano coordenado que representa la superficie del chip. La extension del archivo de entrada es vst y la del archivo de salida es **ap** (*alliance physical*). El archivo de salida no contiene información de las interconexiones pero intenta colocar las celdas que posteriormente se interconectarán lo mas cerca posible. La Fig. 3.11 muestra el uso de los parámetros de entrada y

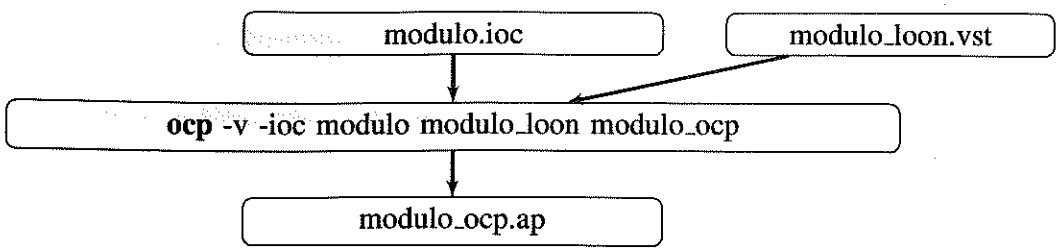


Figura 3.11: Parámetros de entrada y salida para la herramienta *ocp*.

salida para la herramienta *ocp*.

Opciones:

-V Emite mensajes.

-ioc Especifica donde se ponen los conectores donde van soldadas las terminales.

NERO (*Negotiating Router*)

Esta herramienta recibe dos archivos: el de extensión ap (generado por OCP) que contiene las celdas posicionadas en el plano del chip, y el archivo vst (generado por LOON) que contiene las interconexiones entre las celdas instanciadas. Genera un archivo ap que contiene las celdas posicionadas además de los metales de interconexión. El archivo de salida es el layout completo en unidades simbólicas. La Fig. 3.12 muestra el uso de los parámetros de entrada y salida para la herramienta *nero*.

Opciones:

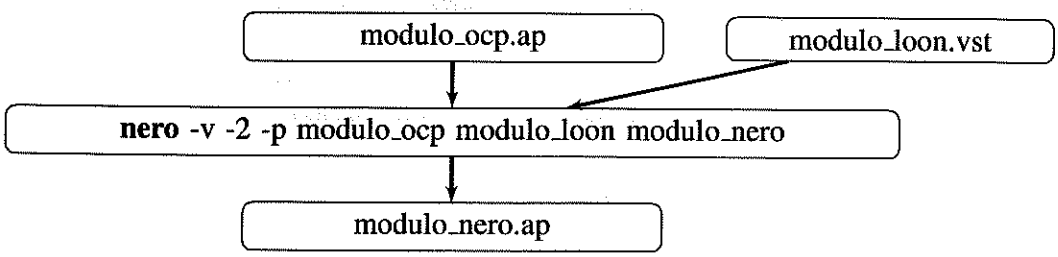


Figura 3.12: Parámetros de entrada y salida para la herramienta *nero*.

-V Informe muy detallado.

-2 Número de metales (depende de la tecnología de fabricación).

-p Especifíca un nombre para el archivo de colocación.

COUGAR (*Hierarchical Netlist Extractor*)

Esta herramienta se aplica sobre el archivo de Layout simbólico (extension ap) producido por Nero. Cougar realiza la extracción de un archivo estructural de extensión al (*alliance logical*) que posteriormente se comparara con el estructural vst producido por LOON. La Fig. 3.13 muestra el uso de los parámetros de entrada y salida para la herramienta *cougar*.

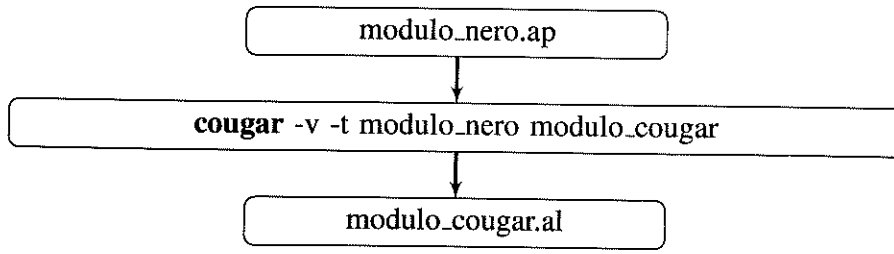


Figura 3.13: Parámetros de entrada y salida para la herramienta *cougar.*

Opciones:

-V Da información detallado.

-t Reporta el numero de transistores.

LVX (Logical Versus Extracted net-list Comparator)

Herramienta para la comparación de archivos estructurales de formato vst y al. La Fig. 3.14 muestra el uso de los parámetros de entrada y salida para la herramienta *lvx*.

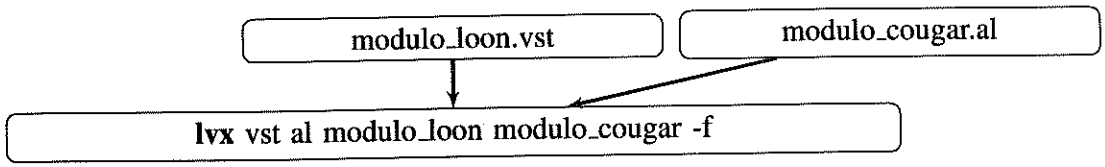


Figura 3.14: Parámetros de entrada y salida para la herramienta *lvx*.

S2R (*Symbolic to Real*)

Herramienta para convertir layout en unidades simbólicas (extensión ap) a un layout en unidades reales (archivo de extensión cif que es el que se enviará al fabricante para su construcción). La Fig. 3.15 muestra el uso de los parámetros de entrada y salida para la herramienta *s2r*.

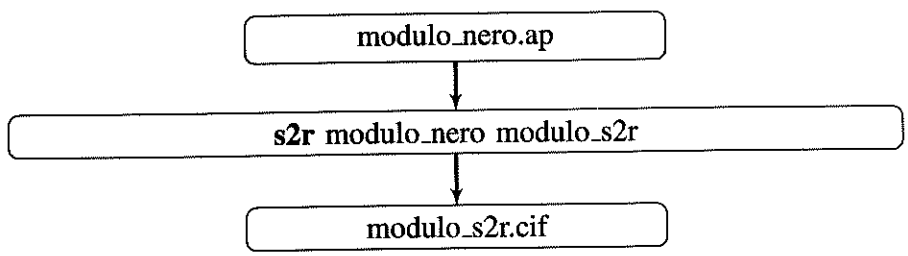


Figura 3.15: Parámetros de entrada y salida para la herramienta *s2r*.

XPAT (*Graphic Pattern Viewer*)

Esta es una herramienta de interfaz gráfica que nos permite visualizar los archivos de patrones en forma de diagrama de tiempos. La Fig. 3.16 muestra el uso de los parámetros de entrada y salida para la herramienta *xpat*.

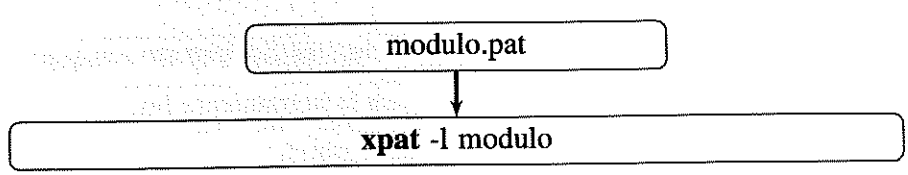


Figura 3.16: Parámetros de entrada y salida para la herramienta *xpat*.

3.3.4. Automatización del las herramientas de síntesis a través de *scripts*

Los *scripts* son archivos de texto que agrupan un conjunto de comandos que se ejecutan de una forma secuenciada. Estos permiten simplificar y eficientar los comandos requeridos para llevar a cabo la síntesis VLSI con las herramientas de Alliance. El siguiente listado corresponde al *script* que sintetiza el módulo flag codificado con VHDL.

```
#!/bin/bash
# Script para sintetizar flag.vst
nombre=flag
echo "PROYECTO ACTUAL: ${nombre}"
vasy -apo -I vhdl ${nombre}  &&
boom -VA ${nombre} ${nombre}_boom &&
boog ${nombre}_boom ${nombre}_boog &&
loon ${nombre}_boog ${nombre} &&
ocp -v -rows 1 ${nombre} ${nombre} &&
genpat -v ${nombre}_datos
asimut ${nombre} ${nombre}_datos ${nombre}_simula  &&
xpat -l ${nombre}_simula
exit 0
```

En el presente proyecto se implementa el diseño de CI´s siguiendo el flujo de diseño de la Fig. 3.2. Se crearon scripts para sintetizar de forma modular o jerárquica cada una de las etapas de una arquitectura VLSI, y se finaliza la síntesis con la unión de todos los sub-módulos sintetizados previamente utilizando un script como se muestra en el siguiente listado:

```
#!/bin/bash
nombre=core
genlib -v ${nombre}
ocp -v -ioc ${nombre} -partial ${nombre}_p ${nombre} ${nombre} &&
nero -V -L -2 -p ${nombre} ${nombre}  pem24 &&
genpat -v core_datos
asimut   ${nombre} core_datos ${nombre}_simula   &&
xpat -l ${nombre}_simula
export MBK_OUT_LO=al &&
cougar -v -t pem24 ${nombre}_trans &&
cougar -v pem24 ${nombre}_cougar &&
export MBK_OUT_LO=vst &&
lvx vst al ${nombre}  ${nombre}_cougar -f &&
export MBK_OUT_LO=spi &&
cougar -t pem24 pem24 &&
export MBK_OUT_LO=vst &&
graal -l pem24
exit 0
```

La síntesis VLSI es una tarea extensa; sin embargo, utilizando *scripts* para automatizar las tareas, se facilita el flujo de diseño descrito en la Secc. 3.3.

Capítulo 4

Desarrollo del proyecto

En este capítulo se presenta el desarrollo del proyecto siguiendo la metodología definida en el capítulo 1. En la primer fase se realizó una experimentación inicial con la finalidad de identificar las características de las señales producidas por los sensores inerciales que sirven de fuente de datos para el FK. En una segunda fase se revisó a detalle el algoritmo del FK aplicado en sistemas de navegación inercial, realizando primeramente simulaciones de la formulación convencional utilizando Matlab, y posteriormente se llevaron a cabo implementaciones en sistemas embebidos con el empleo de procesadores digitales de señal. Como resultado de esta fase se determinaron los requerimientos del algoritmo para su implementación en un CI. La tercera fase se desarrolló en varias etapas, la primera fue la definición de las especificaciones, para lo cual fue necesario determinar la representación numérica a emplear a través de un análisis y evaluación de los efectos de la implementación con aritmética de punto fijo y de punto flotante. Las etapas subsecuentes involucraron el flujo de diseño del CI, partiendo del diseño de la arquitectura como parte central de este proyecto de investigación, la síntesis, el diseño físico, la verificación, el envío a fabricación y finalizando esta fase con las pruebas de medición en laboratorio del CI.

4.1. Experimentación

4.1.1. Sensores inerciales

El uso de los sensores inerciales en la estimación de la posición y orientación tiene aplicaciones en navegación inercial de vehículos autónomos, realidad virtual, realidad aumentada, robótica, y

captura del movimiento humano. Las posiciones y orientaciones son calculadas integrando la aceleración y velocidad angular obtenidas por acelerómetros y giroscópios. El uso de los giroscopios introduce un error de deriva que es acumulativo a través del tiempo y se hace considerable en unos cuantos segundos, es por eso que es necesario el uso de estimadores como el filtro de Kalman para poder compensar los errores y obtener la mejor aproximación en los parámetros de navegación.

4.1.2. Unidad de medición inercial

La unidad de medición inercial o IMU (*Inertial Measurement Unit*) contiene sensores inerciales integrados en un mismo dispositivo, por ejemplo, en una IMU es común encontrar tres acelerómetros y tres giroscopios, además de la electrónica necesaria para el acondicionamiento de las señales, la Fig. 4.1 muestra algunos ejemplos de IMUs comerciales. En la Tabla 4.1 se encuentran las características principales que permiten identificar las diferentes tecnologías y parámetros que las definen. La tecnología más avanzada es la que utiliza como principio el láser. El giroscopio de láser en anillo (RLG) presenta el mayor grado de exactitud, sin embargo, su costo es el más elevado de todas las tecnologías. El giroscopio de fibra óptica (FOG) tiene algunas ventajas notables frente al RLG como menor insensibilidad para bajas velocidades angulares y un costo más reducido. En contraste, la tecnología MEMS que es la de más bajo costo, presenta la menor exactitud y mayores factores de ruido. Otra clasificación es el grado que puede ser de uso militar, navegación, táctico, sistemas AHRS (Attitude and Heading Reference Systems), control y de consumo.

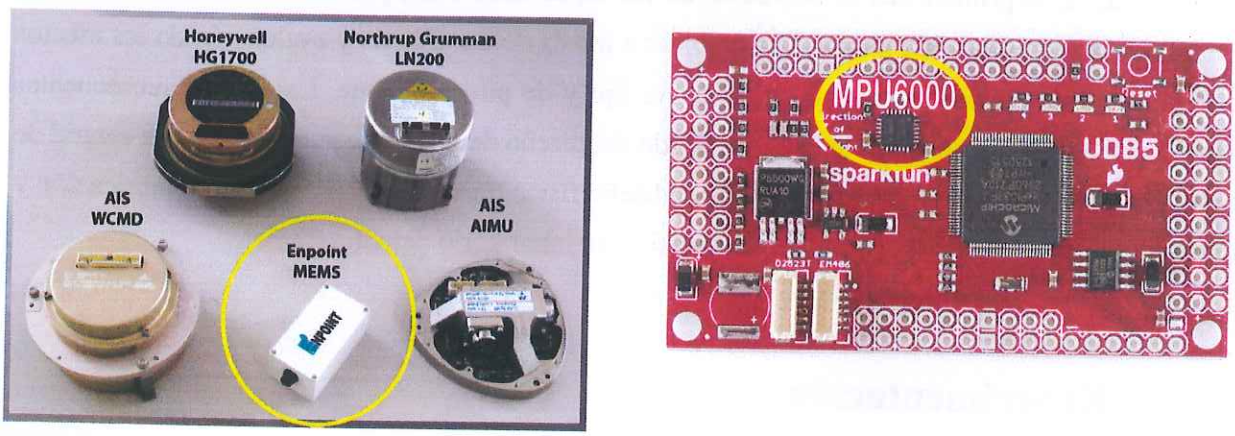


Figura 4.1: Tipos de IMUs.

Tabla 4.1: IMUS comerciales.

Modelo	CIMU	LN200	HG1700	AIMU	3DM-GX325	ADIS16365	MPU6000
Grado	Militar	Navegación	Táctico	AHRS	Control	Control	Consumo
Fabricante	Honeywell	Northup Grumman	Honeywell	Atlantic Inertial	MicroStrain	Analog Device	ST Micro
Tecnología	RLG	FOG	RLG	MEMS	MEMS	MEMS	MEMS
Costo (Dólares)	80,000	27,000	18,500	11,000	2,500	300	20
Giroscopio Bias (°/s)	.0035	1	1	2	0.2	25	20
Factor Escala (ppm)	5	-	150	-	2000	1000	2000
Accel Bias (mg)	.05	1	1-2	2	10	25	50
Factor Escala (ppm)	100	-	300	-	2000	2000	5000

Los sensores inerciales elegidos durante la fase de experimentación fueron la IMU ADIS16350 de Analog Device^{TM} que se muestra en la Fig. 4.2a, así como el MPU6000 de la compañía Invensense^{TM} (Fig. 4.2b). El objetivo en el uso de estos componentes es la generación de datos que sean utilizados en la fase de verificación y validación del algoritmo del FK, tanto para esquemas de tiempo real (en línea) como post-procesamiento (fuera de linea). Las señales generadas por el giroscopio y el acelerómetro en condiciones de reposo se muestran en la Fig. 4.2c, en la cual se observa una señal contaminada con ruido aleatorio en las salidas de los giroscopios y acelerómetros.

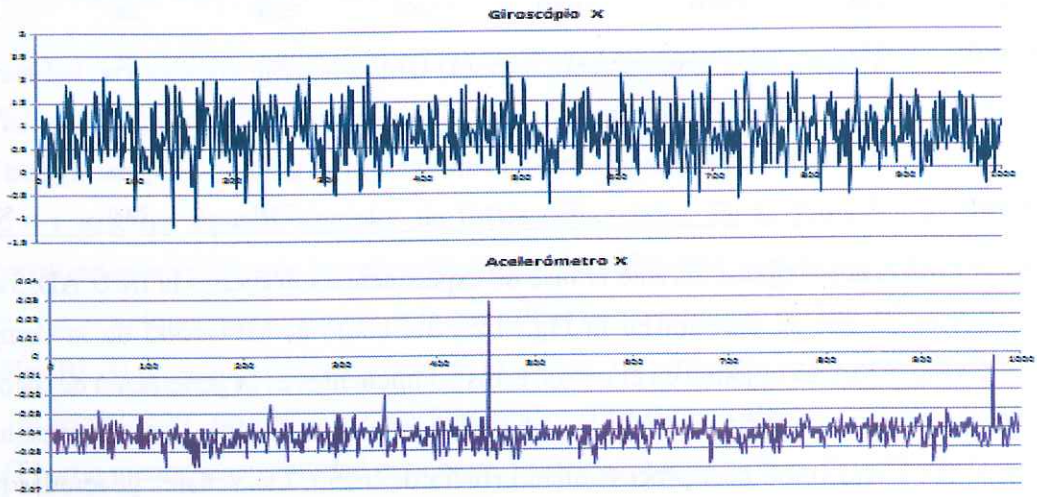
Para determinar el comportamiento de las señales en condiciones de movimiento, se implementó un banco de pruebas controlando el movimiento, para caracterizar las señales de la IMU bajo diferentes perfiles de movimiento llevando a cabo un análisis estadístico de las mismas. La Fig. 4.2d contiene la información generada por uno de los giroscopios del MPU6000.

4.1.3. Banco de pruebas para la adquisición de señales de sensores inerciales

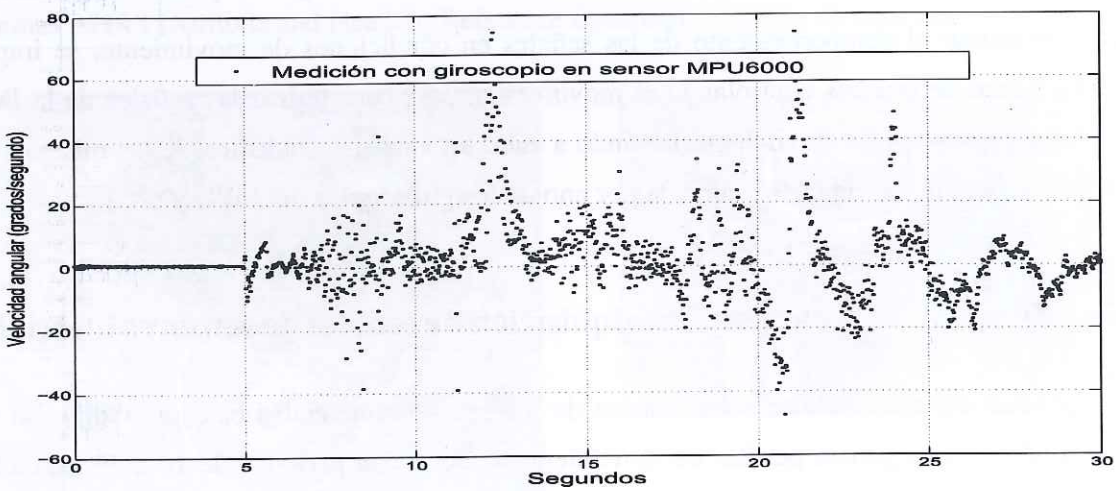
Para generar experimentalmente las señales de los sensores inerciales bajo el entorno de un sistema dinámico que genera perfiles de movimiento circular con periodos de 10 a 40 segundos a una tasa de muestreo de 20 ms, fue necesario crear una plataforma de pruebas a través de un sistema de evaluación llamado ADIS1635x/EVAL de Analog Device (Fig. 4.3a), que almacena los datos en archivos de texto plano (ASCII - Fig. 4.3b) con las señales generadas por los acelerómetros y giroscopios. Los resultados adquiridos por esta plataforma tienen como objetivo servir de datos de prueba durante la evaluación, simulación y validación del algoritmo del FK.

(a) Sensor ADIS16350.

(b) Sensor MPU6000.



(c) Señales en reposo de ADIS16350.



(d) Señales de giroscopio en MPU6000.

Figura 4.2: Sensores inerciales.

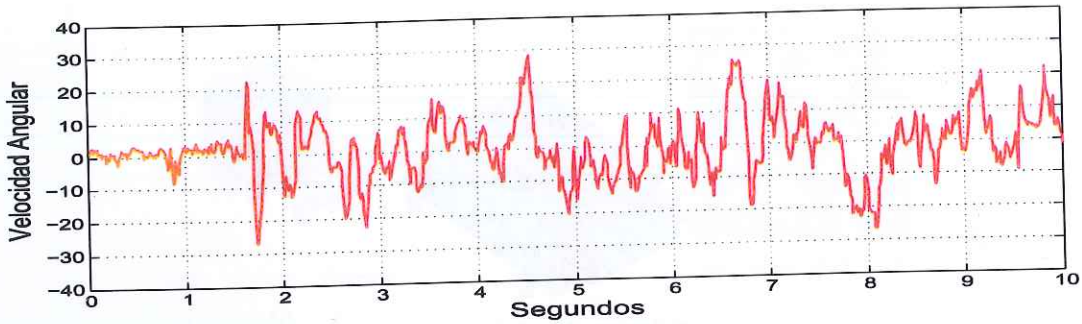
(a) Tarjeta electrónica para la adquisición de señales.

File Name	muestra_1.csv							
Date	03/05/2012							
Time	05:09:38 p.m.							
Begin Seconds	61774.16							
End Seconds	61778.24							
Sample #	Time (sec)	Power V	X Gryo	Y Gyro	Z Gyro	X acc	Y acc	Z acc
1	0	5.025636	0.14652	0.29304	-0.80586	-0.03783	0.035308	-1.02393
2	0	5.025636	0.43956	-0.14652	-0.58608	-0.04287	0.040352	-1.0315
3	0	5.025636	0.51282	0.65934	0.7326	-0.04287	0.040352	-1.0315
4	0.015625	5.023805	0	0.43956	0.87912	-0.0454	0.042874	-1.02393
5	0.015625	5.025636	0.14652	1.17216	-0.07326	-0.04035	0.040352	-1.02645
6	0.015625	5.025636	0.87912	1.61172	-1.02564	-0.03783	0.032786	-1.02645
7	0.015625	5.021973	1.24542	0.43956	-1.61172	-0.05044	0.03783	-1.02141
8	0.03125	5.023805	1.0989	0.14652	-1.0989	-0.04287	0.042874	-1.02645
9	0.03125	5.023805	0.7326	0.80586	-0.29304	-0.04035	0.040352	-1.02898
10	0.03125	5.023805	0.95238	0.58608	-0.14652	-0.03783	0.042874	-1.02645
11	0.03125	5.023805	0.95238	0.87912	-0.21978	-0.04287	0.047918	-1.02393
12	0.046875	5.023805	1.17216	1.39194	-0.65934	-0.04287	0.047918	-1.02141
13	0.046875	5.023805	0.7326	1.8315	-0.21978	-0.04035	0.042874	-1.02141
14	0.046875	5.023805	-0.29304	1.75824	-1.61172	-0.0454	0.027742	-1.02898
15	0.046875	5.021973	0.80586	1.4652	-1.0989	-0.05044	0.045396	-1.02645

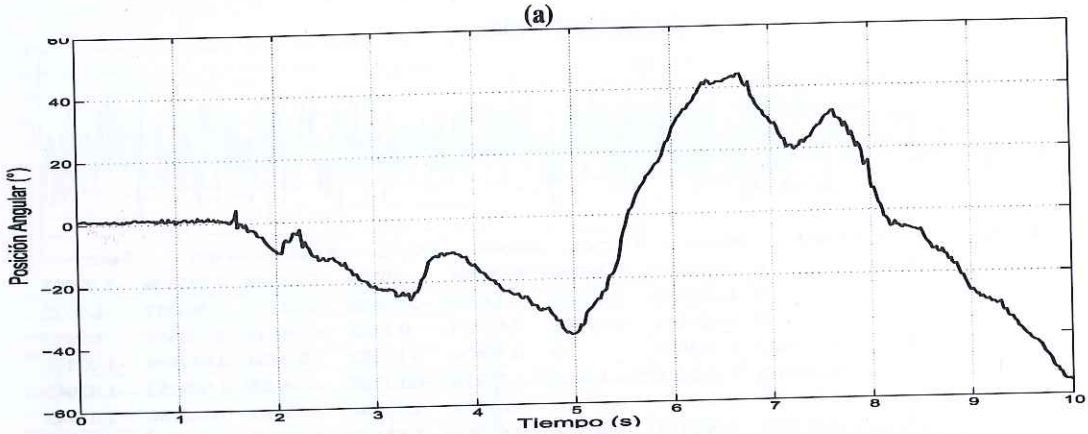
(b) Archivo de datos.

Figura 4.3: Plataforma de pruebas de sensores inerciales.

Las gráficas de la Fig. 4.4 son resultado de muestras de datos tomadas en un periodo de 10 segundos. La Fig. 4.4a representa la velocidad angular del movimiento del sensor, y la Fig. 4.4b la posición angular estática del sensor, que corresponde a la arco-tangente de la componente de aceleración en el eje y respecto de la aceleración del eje x.



(a)



(b)

Figura 4.4: Señales de sensores.

4.2. Requerimientos del algoritmo FK

4.2.1. Diseño del FK aplicado en sistemas de navegación inercial

Para llevar a cabo el diseño del filtro de Kalman es necesario establecer la dinámica del sistema al cual se pretende aplicar. Un sistema inercial, cuyo objetivo sea la estimación de la posición angular, requiere de sensores inerciales como acelerómetros y giroscopios para cada uno de los ejes coordenados x, y, z como se indica en la Fig. 4.5.

Con los acelerómetros se puede calcular la posición angular exacta (θ_a) cuando el sistema está en reposo, no siendo así cuando el sistema está en movimiento. Los giroscopios miden la velocidad angular en forma dinámica y con una integración de esta medición se puede calcular el desplazamiento angular o ángulo de inclinación (θ_g), sin embargo este cálculo será inexacto debido a que con el paso del tiempo el giroscopio genera una deriva que llamamos bias.

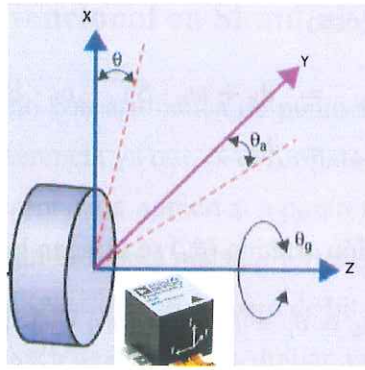


Figura 4.5: Ubicación de las coordenadas en el sistema inercial.

Una solución a este problema es fusionar la respuesta del acelerómetro y del giroscopio para obtener una estimación más exacta del ángulo de inclinación (θ). Esto se puede lograr con un filtro de Kalman de dos estados, donde los estados a estimar son la posición angular final y el bias del giroscopio (b_k). Con la estimación del bias se lleva a cabo la compensación dinámica del error producido por la deriva aleatoria del giroscopio.

4.2.2. Modelos del sistema y medición

El primer paso en el diseño del FK es definir matemáticamente el modelo del sistema lineal discreto y el modelo de medición, de acuerdo a las expresiones de las Ecuaciones (4.1 y 4.2).

$$Sistema \quad x_k = \Phi_{k-1}x_{k-1} + G_{k-1}u_{k-1} + w_{k-1}, \tag{4.1}$$

$$Medición \quad z_k = H_k x_k + v_k. \tag{4.2}$$

Partimos de la Ecuación (4.3) que representa el desplazamiento angular utilizando la velocidad angular medida con el giroscopio.

$$\theta_g = \theta_g + (w_z - w_{z_o}) \cdot S \cdot T \tag{4.3}$$

donde:

- θ_g desplazamiento angular en grados
- w_z velocidad angular giroscopio en el eje z
- w_{z_o} nivel en unidades a velocidad angular cero
- S sensibilidad en unidades de $\circ/s/LSB$
- T tiempo de muestreo

Si $b_k = w_{z_o}$ se puede reescribir como:

$$\theta_{k+1} = \theta_k + w_z \cdot ST - b_k \cdot ST, \tag{4.4}$$

$$b_{k+1} = b_k. \tag{4.5}$$

Para medir el ángulo de inclinación estático (θ_a) se utilizan las aceleraciones a_x,a_y.

$$z_k = \theta_k = \theta_a = arctan\left(\frac{a_y}{a_x}\right) \tag{4.6}$$

Sea $u_k = w_z$, la entrada de control de nuestro sistema dinámico. Entonces la representación matricial del modelo dinámico del sistema lineal estudiado queda definido como:

$$x_{k+1} = \begin{bmatrix} \theta_{k+1} \\ b_{k+1} \end{bmatrix} \tag{4.7}$$

$$x_{k+1} = \begin{bmatrix} 1 & -ST \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \theta_k \\ b_k \end{bmatrix} + \begin{bmatrix} ST \\ 0 \end{bmatrix} [u_k] + w_k \tag{4.8}$$

$$z_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \theta_k \\ b_k \end{bmatrix} + v_k \tag{4.9}$$

El segundo paso es obtener en forma experimental [62] la matriz de covarianza Q_k con las características del giroscopio, y la matriz de covarianza R_k con las características de los acelerómetros. El FK es calibrado a través de la matriz de covarianza de ruido de proceso Q_k y la matriz de covarianza de ruido de medición R_k. Los valores de R_k y Q_k se ajustan dependiendo que tanta confianza se tiene sobre un sensor en particular. Un valor bajo indica que se tiene más confianza en comparación a otros. El valor de R_k indica la cantidad de ruido esperado de la medición.

$$R_k = E(z_k\, z_k^T) \approx \sigma(\theta_a) \tag{4.10}$$

$$Q_k = E(x_k\, x_k^T) = \begin{bmatrix} E(\theta_k\, \theta_k^T) & 0 \\ 0 & E(b_k\, b_k^T) \end{bmatrix} \approx \begin{bmatrix} \sigma(w_z) \cdot T^2 & 0 \\ 0 & \sigma(b) \end{bmatrix} \tag{4.11}$$

El tercer paso es simular el algoritmo del FK usando algún lenguaje de alto nivel como "C" ó MatLab© para establecer la viabilidad del mismo.

4.2.3. Modelo del FK convencional en Simulink-Matlab©

Inicialmente se diseña un modelo con aritmética de punto flotante a 64 bits (variables de tipo double), el cual se utiliza como referencia ya que es el formato con la mayor precisión disponible. Posteriormente se cambia la representación numérica a punto flotante de 32 bits (variable de tipo simple) y finalmente se hacen los cambios para manejar formatos de punto fijo con diferentes dígitos de precisión con el propósito de identificar el número de bits requeridos en la parte fraccionaria del formato de punto fijo que produzca una respuesta similar a su contraparte en punto flotante.

Registros

En la Fig. 4.6 se muestran los registros utilizados para almacenar las matrices y vectores del sistema, sus parámetros de inicialización y resultados parciales.

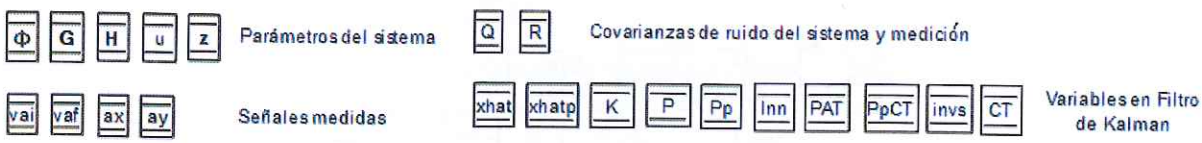


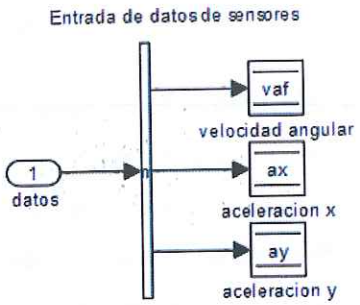
Figura 4.6: Registros para almacenar los parámetros del sistema.

Lectura de los datos provenientes de los sensores inerciales (Fig. 4.7a)

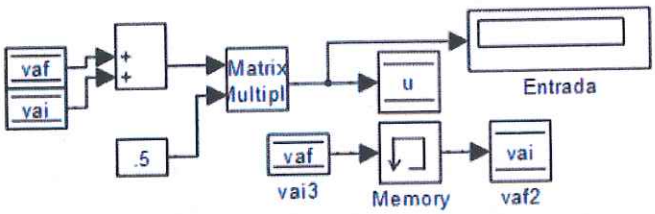
Antes de procesar las señales de los sensores en el FK se realizó un acondicionamiento, la velocidad angular proveniente del giroscopio se le aplicó un filtro promedio de dos lecturas como se aprecia en la Fig. 4.7b. La Fig. 4.7c muestra las lecturas de aceleración y su acondicionamiento con arcotangente para el cálculo del ángulo de inclinación estático y su conversión a grados.

$$u_k = \frac{v_{af}+v_{ai}}{2},\ z_k = arctan\left(\frac{a_y}{a_x}\right).$$

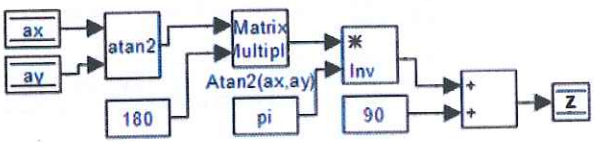
La solución del algoritmo del FK convencional se lleva a cabo en dos etapas, en el primera se hace una *predicción ó time update* utilizando el modelo conocido del sistema. La estimación a priori se calcula con $\hat{x}_k^{(-)} = \Phi\hat{x}_{k-1}^{(+)} + G_{k-1}u_{k-1}$. La Fig. 4.8a muestra el modelo para la estimación a priori $\hat{x}_k^{(-)}$. Y el cálculo de la covarianza a priori con $P_k^{(-)} = \Phi P_{k-1}^{+}\Phi^T + Q_k$ (Fig. 4.8b).



(a) Señales de medición.

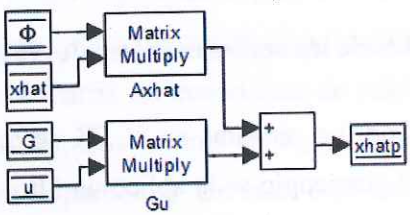


(b) Acondicionamiento de la señal de control u_k.

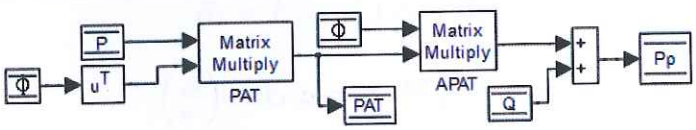


(c) Acondicionamiento de la señal medida y_k.

Figura 4.7: Acondicionamiento de datos de sensores inerciales.

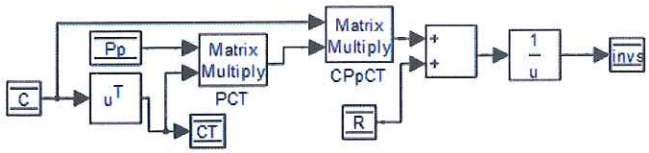


(a) Estimación a priori.

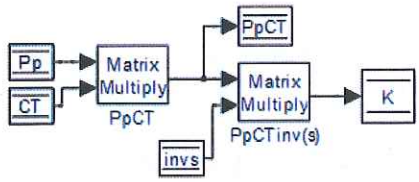


(b) Covarianza a priori.

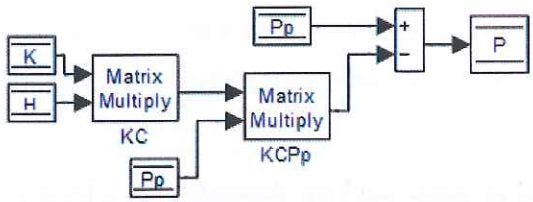
Figura 4.8: Etapa de predicción.



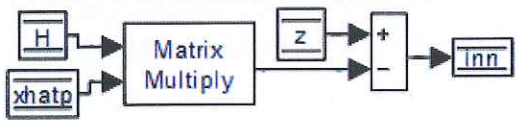
(a) Inversa.



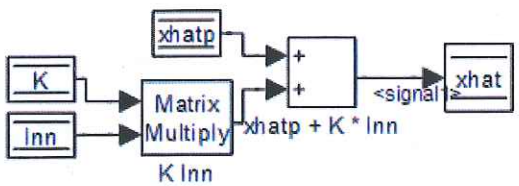
(b) Ganancia K.



(c) Covarianza a posteriori P_p.



(d) Cálculo de la inovación P_p.



(e) Estimación a posteriori P_p.

Figura 4.9: Etapa de corrección.

En la segunda etapa conocida como *corrección ó measurement update*, se calcula la ganancia de Kalman $K = P^- H_k^T [H_{k-1} P_k^- H^T + R_k]^{-1}$, la actualización de la covarianza a posteriori $P(+)_k = [I - KH]P_k^{(-)}$ (Fig. 4.9c), y la estimación a posterior $\hat{x}_k^+ = \hat{x}_k^- + K(y_k - H\hat{x}_k^{(-)})$ (Fig. 4.9e).

Una mejora en la estabilidad numérica del algoritmo se puede implementar con la reformulación por Joseph [63] del cálculo de la covarianza del error P, donde:
$P_k^{(+)} = [I - KH]P_k^{(-)}[I - KH]^T + KRK^T$.

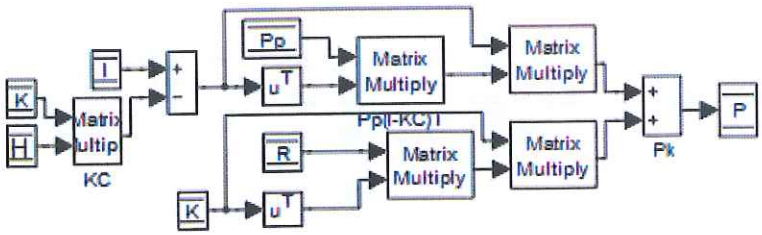


Figura 4.10: Formulación Joseph de $P_k^{(+)}$.

4.2.4. Requerimientos de memoria para el FK

La principal razón de poner atención en los requerimientos de memoria es determinar el límite del tamaño del problema con una asignación fija de memoria. Para el FK, el tamaño del problema está especificado por la dimensión del vector de estado (n), del vector de medición (r) y del vector de entradas de control (m).

Los requisitos de memoria de datos también dependen de la longitud de la palabra de datos (en bits) tanto como del tamaño de los vectores y matrices para almacenar los parámetros del sistema. De acuerdo al análisis realizado, se requiere que el ancho de la palabra para el grupo de registros sea de al menos 20 bits. Sin embargo, para evitar problemas numéricos posteriores y con la finalidad de dar un margen de tolerancia mayor, se eligió un ancho de palabra de 24 bits.

La Tabla 4.2 muestra el grupo de registro que se requieren para la llevar a cabo la evaluación del algoritmo del FK.

$$NR = 3(n \times n) + 2(n \times r) + (n \times m) + (r \times r) + (n + r + m) \tag{4.12}$$

El número de registros (NR) requeridos para la implementación del algoritmo del FK queda definido por la Ecuación 4.12. La Fig. 4.11 establece los requerimientos de memoria para diferentes configuraciones en función del número de estados del sistema (n), el número de variables medidas (r) y el número de variables de control (m). Desde el punto de vista de síntesis VLSI es más simple organizar bloques de registros en potencias de dos, por ejemplo: 32, 64, 128, 256, 512 o 1024. Los colores identifican la posibilidades para cada bloque de registros, ejemplo amarillo banco de 32 registros, naranja banco de 64 registros y así sucesivamente.

Tabla 4.2: Registros para la operación del FK.

Parámetro	Descripción	Dimensión
$\hat{x}$	Vector de estimación de estados	$n \times 1$
z	Vector de mediciones	$r \times 1$
u	Vector de entradas de control	$m \times 1$
Φ	Matriz de la dinámica del sistema	$n \times n$
G	Matriz de las entradas de control	$n \times m$
H	Matriz de medición	$r \times n$
Q	Matriz de covarianza del ruido del sistema	$n \times n$
R	Matriz de covarianza del ruido de sensores	$r \times r$
P	Matriz de covarianza del error	$n \times n$
K	Matriz de ganancia de Kalman	$n \times r$

4.3. Especificaciones del sistema

En la Fig. 4.12 se definen las etapas de un sistema mecatrónico con aplicación en navegación inercial. El bloque principal marcado como Filtro de Kalman, da solución al algoritmo que típicamente se implementa por software en una computadora de navegación inercial. Resolver el FK en tiempo real requiere un alto desempeño computacional, es por esto que el presente proyecto radica en dar solución al algoritmo del FK por medio de hardware VLSI, que pueda implementarse en aplicaciones de navegación inercial con limitaciones de espacio y consumo de energía.

Las etapas del sistema son: sensores inerciales, adquisición y acondicionamiento de datos, inicialización de parámetros, FK y registro de datos.

4.3.1. Etapa de sensores inerciales

Como se mencionó en la Secc. 4.1.1, los sistemas de navegación inercial hacen uso de sensores como acelerómetros y giroscopios para cumplir su propósito. La integración de estos sensores en dispositivos comerciales denominados unidades de medición inercial (IMU) han reducido los costos para la integración de dichos sistemas. Variables como posición, velocidad y aceleración pueden ser estimadas a partir de las mediciones de estos sensores. Los datos pueden ser entregados en forma analógica o digital. El protocolo de comunicación digital SPI (Serial Peripheral Interface) es el formato de bus digital más usual para la transferencia de datos de las IMU's comerciales.

ANALISIS DEL BANCO DE REGISTROS PARA EL FILTRO DE KALMAN

PARAMETROS n= estados estimados m= entradas de control r= mediciones

n = 1

r \ m	1	2	3	4	5
1	13	15	17	19	21
2	19	21	23	25	27
3	27	29	31	33	35
4	37	39	41	43	45
5	49	51	53	55	57

n = 2

r \ m	1	2	3	4	5
1	32	35	38	41	44
2	40	43	46	49	52
3	50	53	56	59	62
4	62	65	68	71	74
5	76	79	82	85	88

n = 3

r \ m	1	2	3	4	5
1	61	65	69	73	77
2	71	75	79	83	87
3	83	87	91	95	99
4	97	101	105	109	113
5	113	117	121	125	129

n = 4

r \ m	1	2	3	4	5
1	100	105	110	115	120
2	112	117	122	127	132
3	126	131	136	141	146
4	142	147	152	157	162
5	160	165	170	175	180

n = 5

r \ m	1	2	3	4	5
1	149	155	161	167	173
2	163	169	175	181	187
3	179	185	191	197	203
4	197	203	209	215	221
5	217	223	229	235	241

n = 6

r \ m	1	2	3	4	5
1	208	215	222	229	236
2	224	231	238	245	252
3	242	249	256	263	270
4	262	269	276	283	290
5	284	291	298	305	312

n = 7

r \ m	1	2	3	4	5
1	277	285	293	301	309
2	295	303	311	319	327
3	315	323	331	339	347
4	337	345	353	361	369
5	361	369	377	385	393

n = 8

r \ m	1	2	3	4	5
1	356	365	374	383	392
2	376	385	394	403	412
3	398	407	416	425	434
4	422	431	440	449	458
5	448	457	466	475	484

n = 9

r \ m	1	2	3	4	5
1	445	455	465	475	485
2	467	477	487	497	507
3	491	501	511	521	531
4	517	527	537	547	557
5	545	555	565	575	585

n = 10

r \ m	1	2	3	4	5
1	544	555	566	577	588
2	568	579	590	601	612
3	594	605	616	627	638
4	622	633	644	655	666
5	652	663	674	685	696
6	684	695	706	717	728

n = 11

r \ m	1	2	3	4	5
1	653	665	677	689	701
2	679	691	703	715	727
3	707	719	731	743	755
4	737	749	761	773	785
5	769	781	793	805	817
6	803	815	827	839	851

n = 12

r \ m	1	2	3	4	5
1	772	785	798	811	824
2	800	813	826	839	852
3	830	843	856	869	882
4	862	875	888	901	914
5	896	909	922	935	948
6	932	945	958	971	984

Figura 4.11: Estimación del tamaño para el banco de registros.

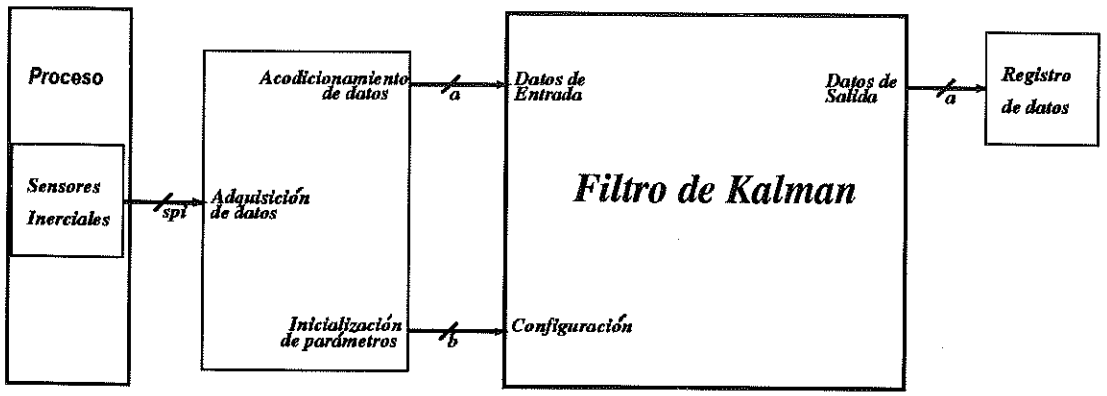


Figura 4.12: Etapas del sistema general de navegación inercial.

4.3.2. Adquisición, acondicionamiento e inicialización de parámetros

Los datos provenientes de los sensores inerciales deben ser adquiridos a una velocidad de muestreo tal que permita su adecuado procesamiento digital y es dependiente de las características del ancho de banda del sensor mismo. Los datos son transferidos por el bus de comunicación SPI a esta etapa y acondicionados a un formato digital de a bits con representación numérica de magnitud y signo para ser transmitidos a la etapa del FK. Esta tarea puede ser realizada a través de un sistema embebido que además permita la inicialización de los parámetros del sistema, con los que el FK debe operar. Estos corresponden a los valores reales, en formato digital de a bits, que definen al sistema discreto a través de un conjunto de matrices y vectores. De igual manera se definen parámetros para la sintonización del filtro.

4.3.3. Registro de datos

A través del registro de datos se obtendrá la información procesada por el FK, obteniendo los datos en un bus de a bits. Esta información podrá ser almacenada u observada a través de un sistema digital externo que lleve a cabo acciones de medición y control. También será el mecanismo por el cual se valide la respuesta del FK.

4.3.4. Bloque de filtro de Kalman

Esta etapa es la parte fundamental del presente proyecto de investigación. El propósito de ésta etapa fue dar solución en tiempo real del algoritmo FK implementado en un CI VLSI. La primera acción a realizar para poner a funcionar el sistema es la inicialización a través del bus de datos de

entrada y configuración. Una vez configurado se comienza a transferir los datos muestreados de los sensores inerciales. Finalmente el CI lleva a cabo la solución del algoritmo FK en forma recursiva. Los datos no se almacenan, sino que se transfieren al exterior por medio del bus de salida. La configuración del FK se lleva a cabo a través del sistema embebido externo, que es el encargado de almacenar los parámetros que definen la dinámica del sistema (Φ, G, H), las covarianzas de ruido de proceso y de medición (Q, R), y la inicialización de la covarianza del error y los estados (P_o, $\hat{x}_o$). Una vez almacenados los parámetros en el chip VLSI que soluciona el algoritmo del FK, entra en un ciclo recursivo, donde se calcula la ganancia óptima de Kalman con la que se estiman los estados del sistema. El algoritmo debe operar en tiempo real, procesando recursivamente las muestras de los sensores, para lo cual la etapa de adquisición y acondicionamiento estará suministrando al filtro las mediciones de los sensores inerciales de manera recursiva, y el FK estará estimando los parámetros de navegación ($\hat{x}_k$) en tiempo real y entregándolos a la etapa de registro de datos con la cual se validará la respuesta del filtro.

4.4. Arquitectura y diseño lógico

El diagrama a bloques y simplificado del FK se muestra en la Fig. 4.13 y 4.14 respectivamente, el cual contiene tres sub-bloques: banco de datos, unidad aritmética y secuenciador. En esta arquitectura se define al chip del FK como un esclavo, dependiente de un controlador maestro (sistema embebido externo) el cual lleva a cabo las tareas de configuración inicial de parámetros del sistema, acondicionamiento y transferencia de los datos en forma recursiva provenientes de los sensores inerciales, así como la lectura de los resultados entregados por el FK. De acuerdo a los requerimientos del algoritmo FK definidos en la Secc. 4.2.4, se definió una arquitectura basada en un formato de 24 bits utilizando una representación numérica de punto fijo, esto reduce significativamente tanto el área de silicio utilizada en el diseño VLSI, así como el consumo de energía del chip, que son los principales factores en la optimización de la arquitectura propuesta.

4.4.1. Banco de datos

El algoritmo del FK utiliza matrices y vectores con los que lleva a cabo las operaciones. El banco de datos contiene registros de propósito general para el almacenamiento temporal de las matrices y vectores, así como de otras variables temporales utilizadas durante el procesamiento de la información.

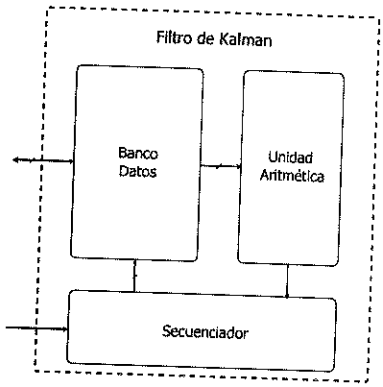


Figura 4.13: Diagrama a bloques de la arquitectura del FK.

La arquitectura del banco de datos se muestra en el diagrama de la Fig. 4.15. Los buses de entrada y salida de datos *DIN* y *DOUT* tienen un ancho de 12 bits, y transfieren por separado la parte baja y la parte alta del contenido del registro a leer ó grabar. Para ello es necesario activar la entrada *HL* con '1' para seleccionar la parte alta del registro, o en su caso '0' para seleccionar la parte baja y aplicar un pulso en la terminal *WRITE_HL*, con esto los datos del bus de entrada *DIN*, quedarían almacenados en los registros *RL* y *RH*. Otra fuente de datos para los registros del banco puede provenir del bus *SRMI* que contiene los resultados de las operaciones realizadas en la unidad aritmética, la terminal *READY* (salida del secuenciador) permite seleccionar si los datos provienen del bus *SRMI* ó de los registros *RL* y *RH*. Si lo que deseamos es almacenar los datos en uno de los registros del banco, es necesario aplicar la dirección de 5 bits correspondiente a través del bus *EDIR* y aplicar un pulso momentáneo en la terminal *WRITE_B*.

En el presente trabajo de tesis, con la finalidad de validar la arquitectura propuesta, se realizó la síntesis para un FK de 2 estados. El banco de registros contiene 32 registros de 24 bits de tipo dinámico (Fig. 4.16a). La configuración del banco es de doble puerto de salida, permitiendo al sistema redireccionar el contenido de dos registros a los buses *RA* y *RB* al mismo tiempo, dándole la capacidad al sistema de llevar a cabo la lectura de dos operandos en un solo ciclo de reloj. También existen dos registros adicionales e independientes de 24 bits, denominados *RQ* y *RD*. Estos registros se utilizan como registros auxiliares durante la ejecución de las operaciones. La dirección de 2 bits en el bus *DEST* es la que determina cual de los registros es en el que se almacenarán los datos (ver Tabla 4.3). Para almacenar información en uno de los registros es necesario establecer el dato a guardar primeramente en los registros *RL* y *RH* o en su defecto usar el dato proveniente del bus

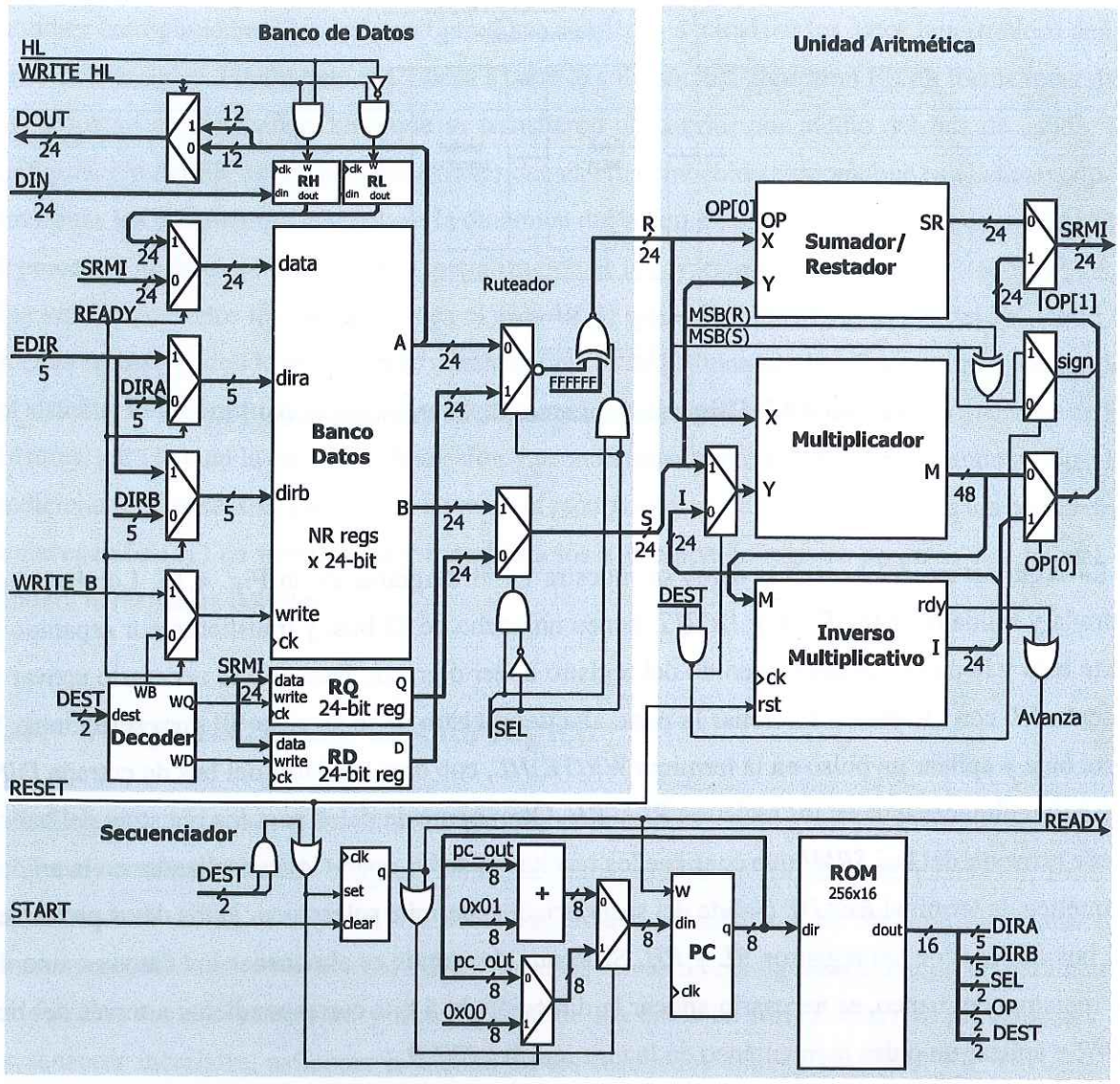


Figura 4.14: Arquitectura simplificada del FK.

SRMI. La dirección correspondiente debe estar en el bus *DIRB*, un valor alto en la entrada *wb* y el flanco de subida en la señal de *CLK*. Si se requiere leer uno o dos registros, entonces solo se debe aplicar la dirección correspondiente en ambos buses de dirección *DIRA* y *DIRB*. Los datos correspondientes siempre estarán disponibles en las salidas A y B.

La arquitectura del bloque de registros se muestra en la Fig. 4.16b . El decodificador recibe las direcciones *DIRA* y *DIRB*, en función de estas direcciones, se activarán solo las señales correspondientes para activar dos de los buffers de tres estados ubicados en cada uno de los canales de salida

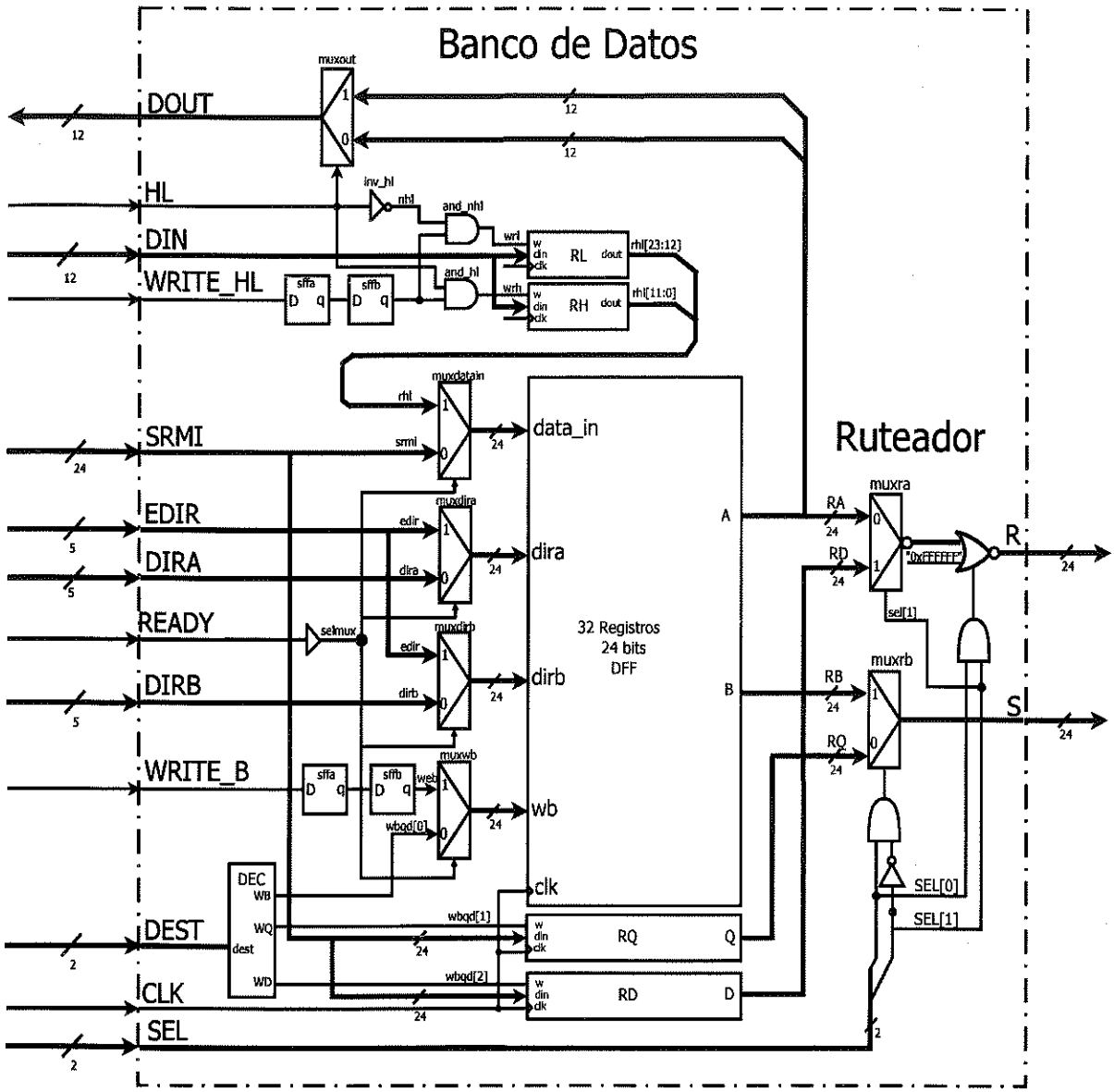


Figura 4.15: Arquitectura del Banco de Datos.

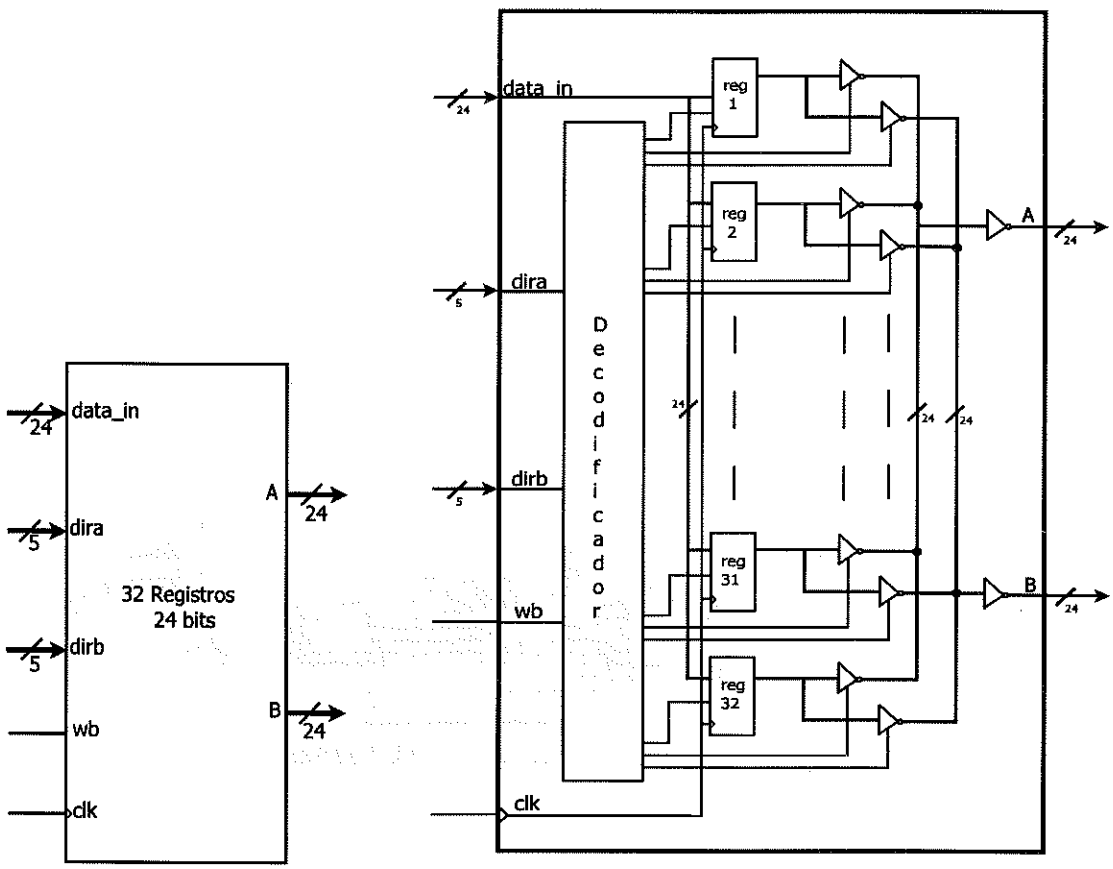
RA y *RB*. Con el objetivo de simplificar el esquema, cada símbolo de los buffers representa a 24 buffers de 1 bit y los registros marcados como reg 1 hasta reg 32 contienen 24 celdas de flip-flop cada uno. Otra Secc. importante del banco de datos es el ruteador, el cual selecciona la fuente de datos a través del bus *SEL* (ver Tabla 4.4), posibilitando el acceso ya sea a los registros de propósito específico (*RA* y *RB*) ó a los registros auxiliares (*RQ* y *RD*).

Tabla 4.3: Registro destino.

DEST[1]	DEST[0]	Registro
0	0	*RB*
0	1	*RQ*
1	0	*RD*
1	1	*STOP*

Tabla 4.4: Fuente de datos.

SEL[1]	SEL[0]	R	S
0	0	*RA*	*RQ*
0	1	*RA*	*RB*
1	0	*RD*	*RQ*
1	1	*RD*	*RB*



(a) Terminales. **(b)** Arquitectura.

Figura 4.16: Banco de Registros.

El banco de registros tiene la función de almacenar los parámetros de configuración del FK así como los estados estimados del sistema. La Tabla 4.5 describe cada uno de los registros y su parámetro asociado. La codificación definida es un tanto arbitraria y puede cambiarse de acuerdo

a la forma de codificar las operaciones matriciales del algoritmo. La definición propuesta establece los primeros dos registros para las mediciones del sistema y la entrada de control. Los siguientes registros están dedicados al vector de estimación $\hat{x}_k$, y los registros subsecuentes a los demás parámetros del sistema. Cabe señalar que ésta definición se puede ajustar de acuerdo a las condiciones de las dimensiones de los vectores y matrices. La matriz T y S se utilizan como registros temporales durante el cálculo de las operaciones matriciales. El último registro se define con la constante uno en el formato de punto fijo 24.14, esto significa que los 14 bits menos significativos equivalen a la parte fraccionaria siendo cero y los 9 bits siguientes corresponden al valor entero uno, y finalmente el bit más significativo que corresponde al signo, siendo 1 para números negativos y cero para los positivos.

La arquitectura propuesta de forma paramétrica permite crecer el número de estados. Los bloques del banco de datos y ROM del la Fig. 4.14 son los bloques que crecen en función del número de estados a estimar por el FK. De acuerdo al análisis descrito en la Fig. 4.11 algunos otros elementos como el tamaño del PC, el sumador de PC, y de los multiplexores junto al PC se deben modificar para manejar configuraciones de mayor número de estados. El algoritmo FK se codifica mediante microinstrucciones definidas en la ROM. Para el caso de la síntesis del FK de dos estados requiere 123 registros de 16 bits y las microinstrucciones se encuentran definidas en las Tablas 4.12 a 4.15.

4.4.2. Unidad aritmética

La unidad aritmética contiene la ruta de datos para llevar a cabo las operaciones de suma, resta, multiplicación e inverso multiplicativo que son suficientes para resolver las ecuaciones definidas en la Tabla 2.1. La Fig. 4.17 muestra la terminales de la unidad aritmética. Los datos deben ser alimentados por los buses de 24 bits *R* y *S* en representación numérica 24.14 de magnitud y signo. El bus de dos bits *OP*, es el encargado de seleccionar el tipo de operación a realizar: suma, resta, multiplicación e inverso multiplicativo según se puede apreciar en la Tabla 4.6. La terminal de *RESET* permite establecer a una condición inicial el estado del sistema, activando por defecto la salida de *Avanza*. La ejecución de las operaciones se llevan a cabo en un ciclo de reloj (*CLK*), a excepción del inverso multiplicativo, que requiere 25 ciclos de reloj para resolver la operación. La Ecuación (2.14) requiere de la solución de la matriz inversa, esta se puede evitar utilizando el FK secuencial como se definió en la Secc. 2.4, donde se evalúan de manera escalar las Ecuaciones (2.14 ,2.15 y 2.16) con cada medición (*r*), es por esto que se requiere del inverso multiplicativo.

La Fig. 4.18 muestra a bloques las conexiones entre cada una de las funciones aritméticas. Debido

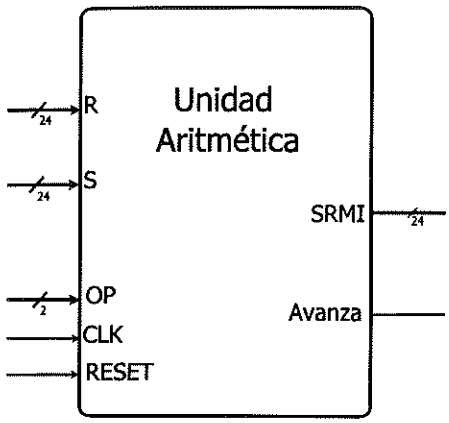


Figura 4.17: Bloque Unidad Aritmética.

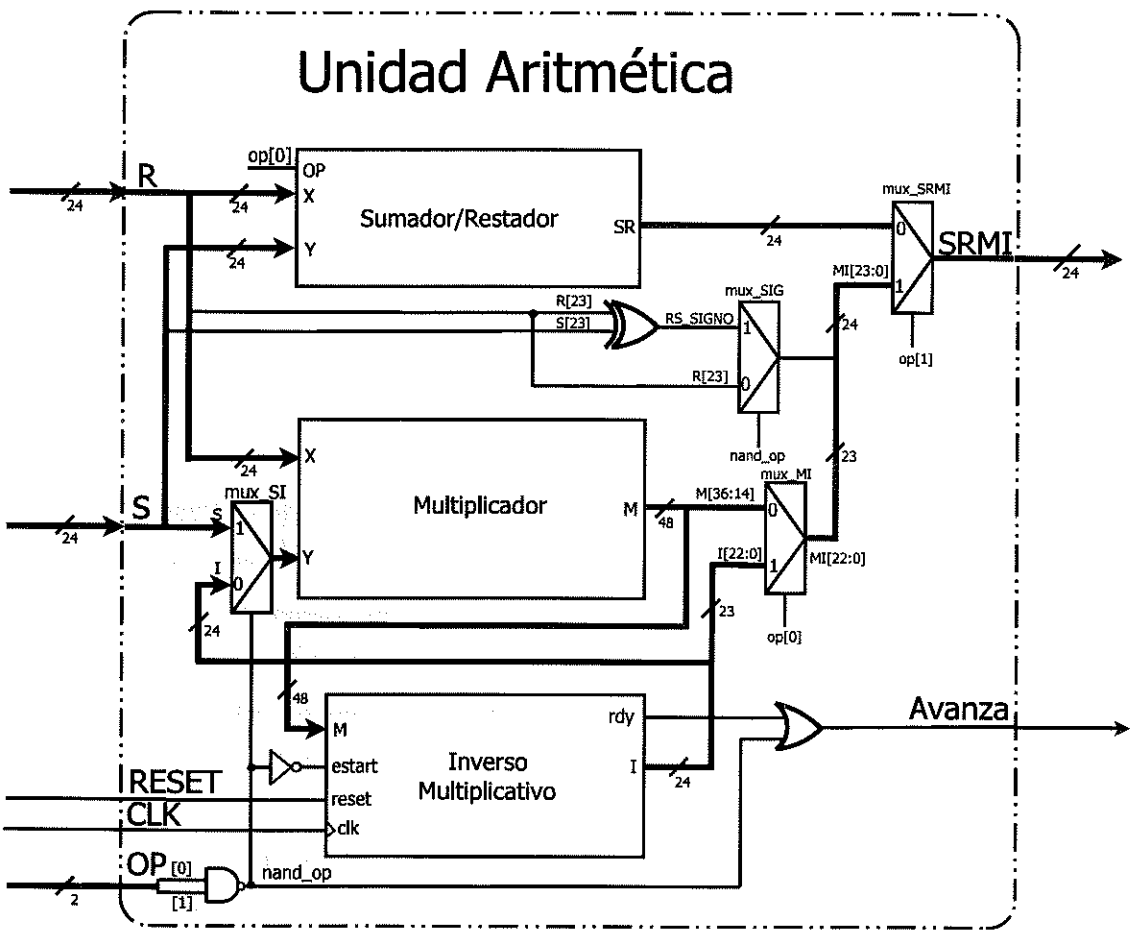


Figura 4.18: Diagrama a bloques de la Unidad Aritmética.

a que los bloques *Sumador/Restador* y *Multiplicador* son circuitos combinacionales, estas operaciones solo consumen el tiempo de retardo inherente a las celdas y a la topología implementada. Este retardo es básicamente el que limita el periodo mínimo del ciclo de reloj y en consecuencia la frecuencia de operación máxima.

Sumador-Restador

Sea $X, Y \ \epsilon \ \Re$, $A = |X|$, $B = |Y|$, S_X y S_Y representan el signo de la magnitud A y B respectivamente. Un valor de '0' indica una magnitud positiva y '1' una magnitud negativa.

Operación del módulo sumador-restador

El circuito sumador-restador de la Fig. 4.19 tiene como entradas dos buses X, Y de 24 bits, de los cuales el bit más significativo está representado por S_X o S_Y, y el resto de bits la magnitud en valor absoluto. El resultado de la operación se obtiene en el bus de salida *SR* de 24 bits con formato de magnitud y signo. Una señal de control denominada *OP* determina el tipo de operación a ejecutar. Si $OP[0] =' 0'$ se realiza la suma, y si $OP[0] =' 1'$ la resta. La definición de las terminales, su tipo y descripción se encuentra en la Tabla 4.7.

La arquitectura del Sumador-Restador opera con 24 bits, siendo el bit más significativo el que representa el signo. La interpretación de la magnitud se hace considerando 14 bits para la parte fraccionaria, quedando solo 9 bits para la parte entera. Esta convención se justifica con el análisis del algoritmo FK utilizando aritmética de punto fijo descrito en la Secc. de resultados. Los bloques SR_AB y R_BA calculan en forma paralela la suma y resta de A $\pm$ B y la resta de B-A respectivamente. Mediante el multiplexor mux_out se elije la salida de acuerdo al tipo de operación elegida mediante la terminal *OP*. El bit de signo de cada uno de los buses *X*, *Y* es extraído mediante buffers y es tratado por separado de la magnitud (bits 0 a 22). La característica principal de este modelo es la ejecución de la operación de suma y resta de una representación en magnitud y signo de forma combinacional, por lo cual solo tiene que respetarse un tiempo de propagación para que el resultado en la salida *SR* sea válido.

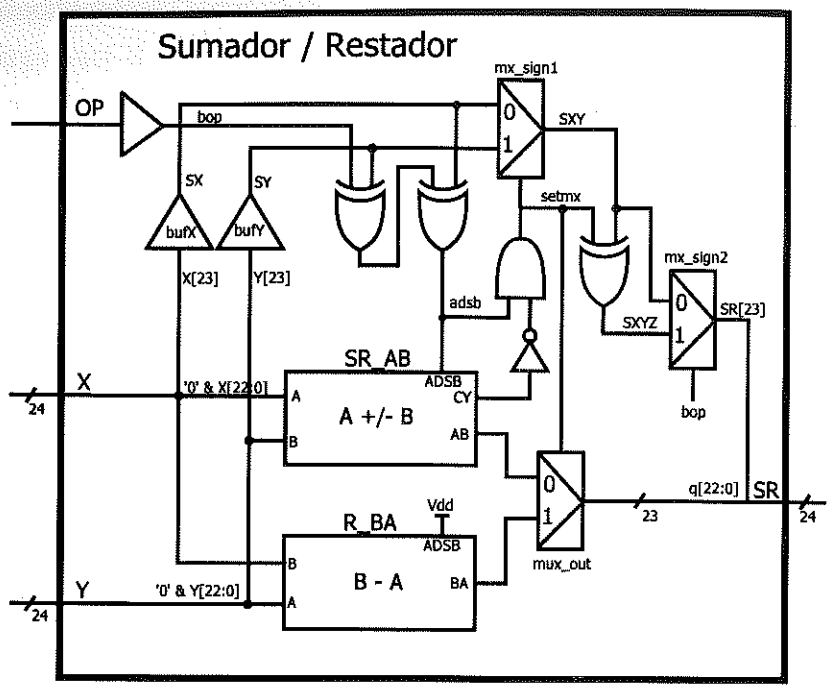


Figura 4.19: Arquitectura del Sumador-Restador.

Multiplicador

Para diseñar la arquitectura del multiplicador se evaluaron diferentes técnicas debido a que la multiplicación requiere el uso de medios sumadores y sumadores completos. La metodología empleada fue a través del uso de celdas estándar de librerías VLSI con la finalidad de optimizar área de silicio y minimizar retardos de propagación. Posteriormente se desarrolló un multiplicador paralelo optimizado con una arquitectura jerárquica de 24 bits. El bloque de la Fig. 4.20 muestra las terminales de entrada/salida del multiplicador,

La metodología de diseño en el módulo multiplicador de 24×24 bits se diseñó en base al algoritmo definido en las ecuaciones (4.13) - (4.18), el cual divide una multiplicación de n bits en 4 multiplicaciones de $\frac{n}{2} \times \frac{n}{2}$ bits, y cinco sumas de $\frac{n}{2}$ bits. Para esto es necesario partir en bloques de multiplicación básicos para 3, 6 y 12 bits. El multiplicador de 24 bits se construye con 4 multiplicadores de 12 bits y 5 sumadores de 12 bits; un multiplicador de 12 bits se implementa con 4 multiplicadores de 6 bits y 5 sumadores de 6 bits; un multiplicador de 6 bits se implementa con 4 multiplicadores de 3 bits y 5 sumadores de 3 bits. La Fig. 4.21 muestra la arquitectura del multiplicador de 24 bits.

$$A = a_{n-1} \ldots a_1 \, a_0 \tag{4.13}$$

$$B = b_{n-1} \ldots b_1 \, b_0 \tag{4.14}$$

$$M = A \times B = m_{2n-1} \ldots m_1 \, m_0 \tag{4.15}$$

$$A_0 = a_{(n/2)-1} \ldots a_0, \; A_1 = a_{n-1} \ldots a_{(n/2)} \tag{4.16}$$

$$B_0 = b_{(n/2)-1} \ldots b_0, \; B_1 = b_{n-1} \ldots b_{(n/2)} \tag{4.17}$$

$$M = A_1 \times B_1 2^n + A_1 \times B_0 2^{(n/2)} + A_0 \times B_1 2^{(n/2)} + A_0 \times B_0 \tag{4.18}$$

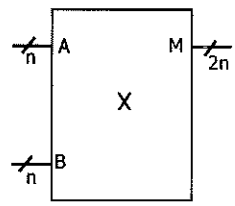
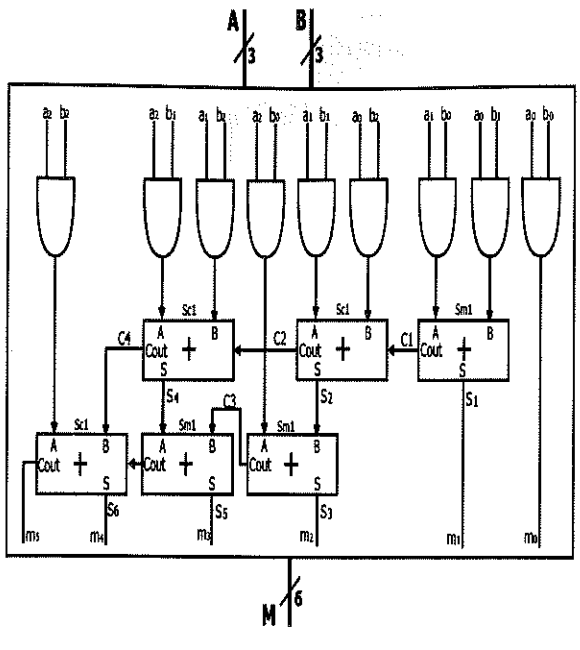


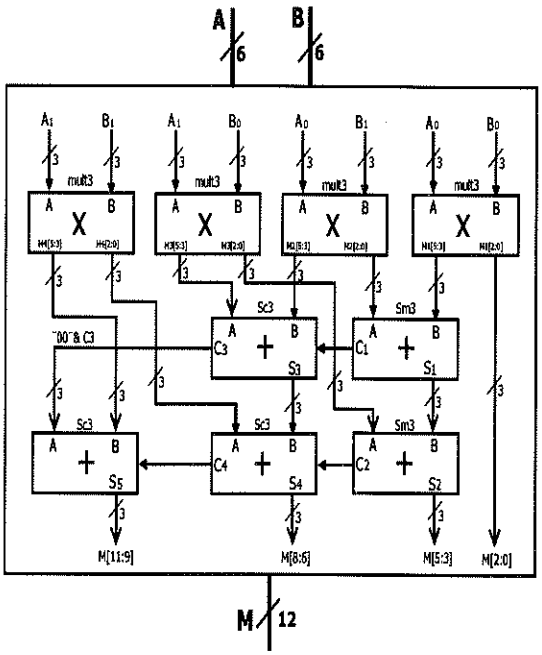
Figura 4.20: Bloque multiplicador.

La arquitectura de este multiplicador sigue una metodología jerárquica, donde el bloque inicial es el multiplicador de 3 bits de la Fig. 4.21a, este esquema utiliza tres sumadores completos (Sc1) y tres medios sumadores (Sm1) además de 9 compuertas AND de dos entradas. El diseño lógico se define en forma estructural y con la finalidad de optimizar área de silicio se utilizan celdas primitivas de la librería *sxlib* para la implementación de los sumadores en las herramientas de síntesis de Alliance. Esta metodología permite ahorrar una gran cantidad de área de silicio ya que de otra forma las herramientas de síntesis implementan los sumadores mediante compuertas XOR y NAND que ocupan demasiado espacio de silicio. Los datos a multiplicar se introducen en formato binario de magnitud positiva en los buses de tres bits A, B y el resultado se obtiene en el bus de seis bits M. Este esquema solo puede multiplicar números positivos, por lo que el signo se maneja por separado. Para crecer el ancho del bus del multiplicador se diseño la estructura del multiplicador de 6, 12 y 24 bits como lo muestran las Figs. 4.21b, 4.21c, y 4.21d.

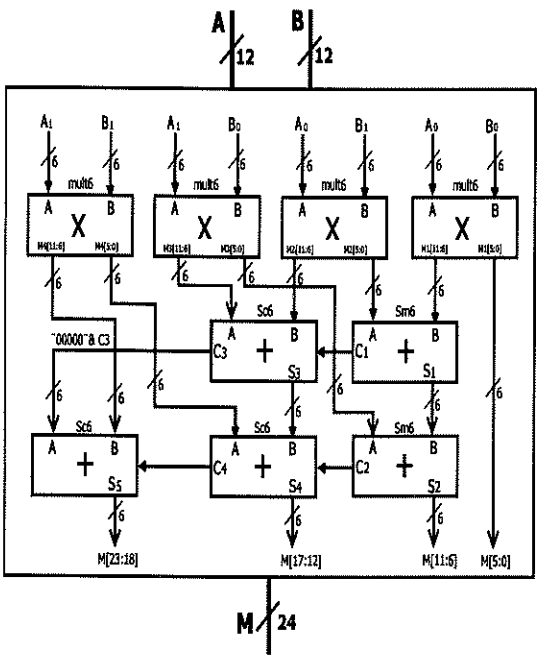
Cada estructura es uniforme, esto es, para cada multiplicador de n bits se requieren cuatro multiplicadores de n/2 bits, tres sumadores completos de n/2 bits y dos medios sumadores de n/2 bits. El resultado se obtiene en el bus M de 2n bits.



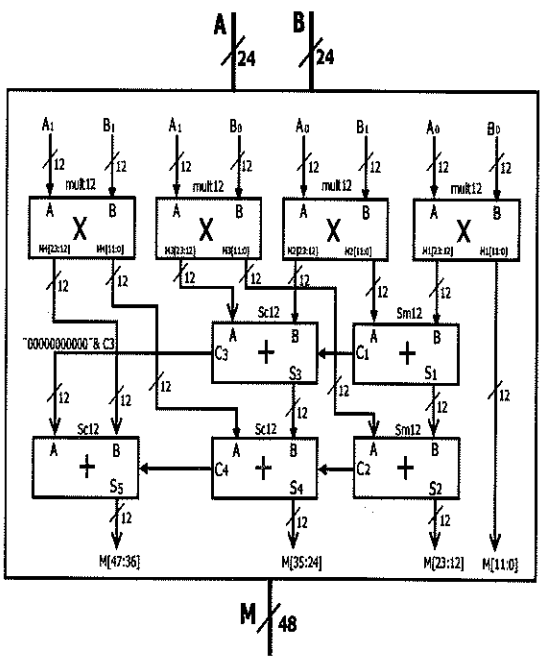
(a) 3 bits.



(b) 6 bits.



(c) 12 bits.



(d) 24 bits.

Figura 4.21: Multiplicador estructural.

Inverso multiplicativo

Considerando la Ecuación 4.19, donde X un número binario de N bits, y su inverso multiplicativo Y también de N bits, la multiplicación de X por Y da como resultado la unidad. En base a esta premisa, la solución del inverso multiplicativo de magnitudes binarias en formato de punto fijo se basa en la técnica de aproximaciones sucesivas. Este método iterativo requiere N+1 ciclos, donde N es el número de bits en el ancho de palabra del dato de entrada.

Se llama función inversa o recíproca de f a otra función f^{-1} que cumple que:

Si $f(a) = b$, entonces $f^{-1}(b) = a$.

Algoritmo para la obtención del inverso multiplicativo por aproximaciones sucesivas:

Sea X un número binario de N bits, y su inverso multiplicativo Y, donde:

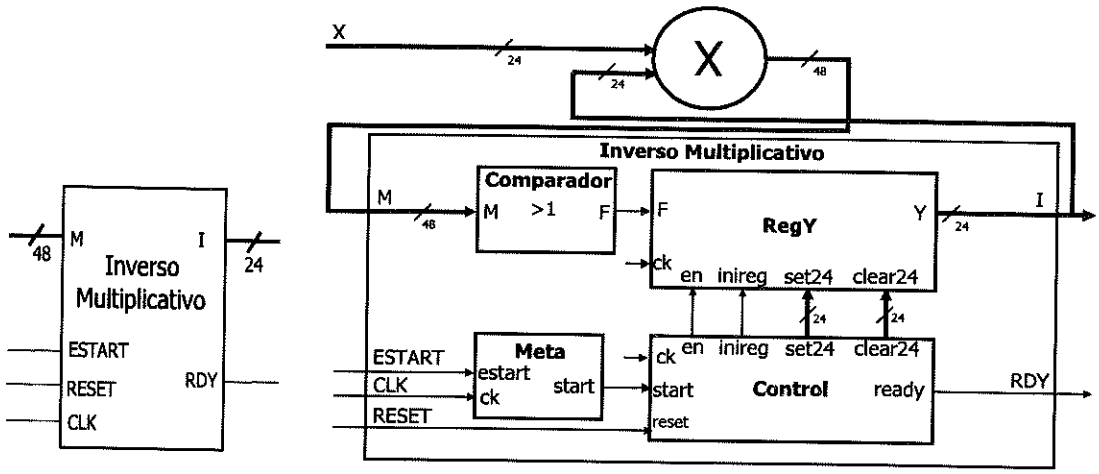
$$X * Y = 1 \tag{4.19}$$

1. Se inicializa el valor de Y con ceros excepto el bit más significativo.
2. Se multiplica el valor de entrada X por Y.
3. Si el resultado > 1 se cambia el bit evaluado a cero en caso contrario se mantiene en uno.
4. Se ajusta el bit menos significativo siguiente de Y a evaluar en uno.
5. Si el bit evaluado es el menos significativo continua al paso 6 en caso contrario al paso 2.
6. Fin.

El algoritmo se basa en evaluar cada bit de Y de izquierda a derecha con la finalidad de ir construyendo el valor de Y que satisface la Ecuación (4.19), una vez que se evalúa el bit menos significativo el valor de Y obtenido representa el inverso multiplicativo de X.

Como se puede observar en el algoritmo anterior, para la solución del inverso multiplicativo se tiene que realizar una multiplicación y una comparación. El bloque de la Fig. 4.22a contiene la arquitectura definida en la Fig. 4.22b a excepción del multiplicador, puesto que se reutiliza el multiplicador definido en la Secc. anterior, esto nos permite un ahorro substancial en área de silicio ya que el multiplicador es uno de los bloques que consumen más área. Debido a que el dato X es de 24 bits, la salida del multiplicador es de 48 bits y es la entrada al bloque inverso multiplicativo. El resultado del inverso multiplicativo es obtenido en el bus de salida I a 24 bits.

La ejecución de esté módulo es iterativo y requiere señales de control como *ESTART* para iniciar el cálculo del recíproco y después de 25 ciclos de reloj la señales de salida rdy que se activa, indicando que ya se puede disponer del resultado en el bus I. La terminal de *RESET* permite asegurarnos de que la operación del módulo pueda iniciarse correctamente. La Tabla 4.8 muestra las terminales y señales de control para el cálculo del inverso multiplicativo.



(a) Entidad.

(b) Arquitectura.

Figura 4.22: Módulo inverso multiplicativo.

Operación del módulo inverso multiplicativo

1. N es el número de bits del bus *X*.
2. F es la salida del comparador. '1' indica que el resultado de la multiplicación de *X*Y* es mayor que 1, en caso contrario es '0'.
3. El dato tiene que mantenerse en el bus *X* por 25 ciclos de reloj para obtener el recíproco.
4. El multiplicador lleva acabo la operación en forma combinacional.
5. Si *ESTART=1*, el modulo meta genera un pulso *START* de duración de un periodo de reloj. Esto da comienzo a la operación de la máquina de estado en el bloque de *Control*.
6. El módulo *RegY* recibe la bandera *F* y la señal de enable (*en*), generando el valor para *Y* (*1/X*), una vez determinado éste por aproximaciones sucesivas, se activa la salida *RDY* dando aviso de que el módulo terminó de calcular el inverso multiplicativo y el resultado está disponible en el bus *I*.

4.4.3. Secuenciador

El secuenciador tiene el propósito de generar las señales de control para cada uno de los multiplexores que controlan el flujo de información en cada uno de los bloques del sistema. La Fig. 4.23 muestra las terminales de entrada-salida del secuenciador y en la Tabla 4.9 se describe su funcionalidad. El esquema completo del secuenciador se muestra el la Fig. 4.24. En este esquema se utiliza la técnica de control mediante microcódigo a través de una memoria ROM, direccionada secuencialmente mediante el registro *PC*. En el momento en que se genera una nueva dirección, el PC apunta a la dirección de memoria ROM que contiene el microcódigo binario que determina el flujo de información. Un pulso en la terminal *ST* provoca que se la señal *READY* se establezca en un nivel bajo lo cual indicaría que el CI está realizando el cálculo de una iteración del algoritmo. La terminal *RST* permite reiniciar el funcionamiento del ciclo de iteración. El bus de dos bits *DEST* en la entrada del módulo secuenciador es el que determina el momento de finalización del ciclo de iteración cuando ambos *DEST*[0] y *DEST*[1] están en '1', volviendo a nivel alto la terminal de salida *READY*. A partir de este momento se pueden leer los resultados de FK y volver a actualizar la medición para comenzar otro ciclo de iteración.

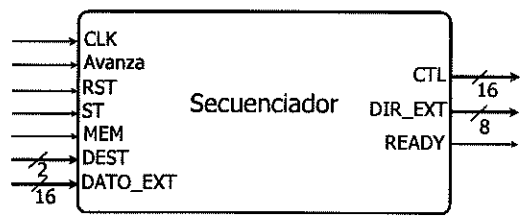


Figura 4.23: Diagrama a bloques del secuenciador.

El bus de control de 16 bits *CTL* se codifica de la siguiente manera:

DEST		OP		SEL		DIRB					DIRA				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Los buses *DIR_EXT* y *DAT_EXT* permiten controlar externamente el flujo de datos del chip, haciendo uso de una memoria RAM externa. La codificación del algoritmo de manera externa sustituye al código definido en la ROM interna definida en el bloque secuenciador. Esta memoria externa debe tener máximo 256 registros de 16 bits, direccionados por el bus externo DIR_EXT y recuperando la información a través del bus DAT_EXT. Esta arquitectura deja abierto el sistema para poder depurar por hardware la ejecución a través de esta memoria externa y modificar la

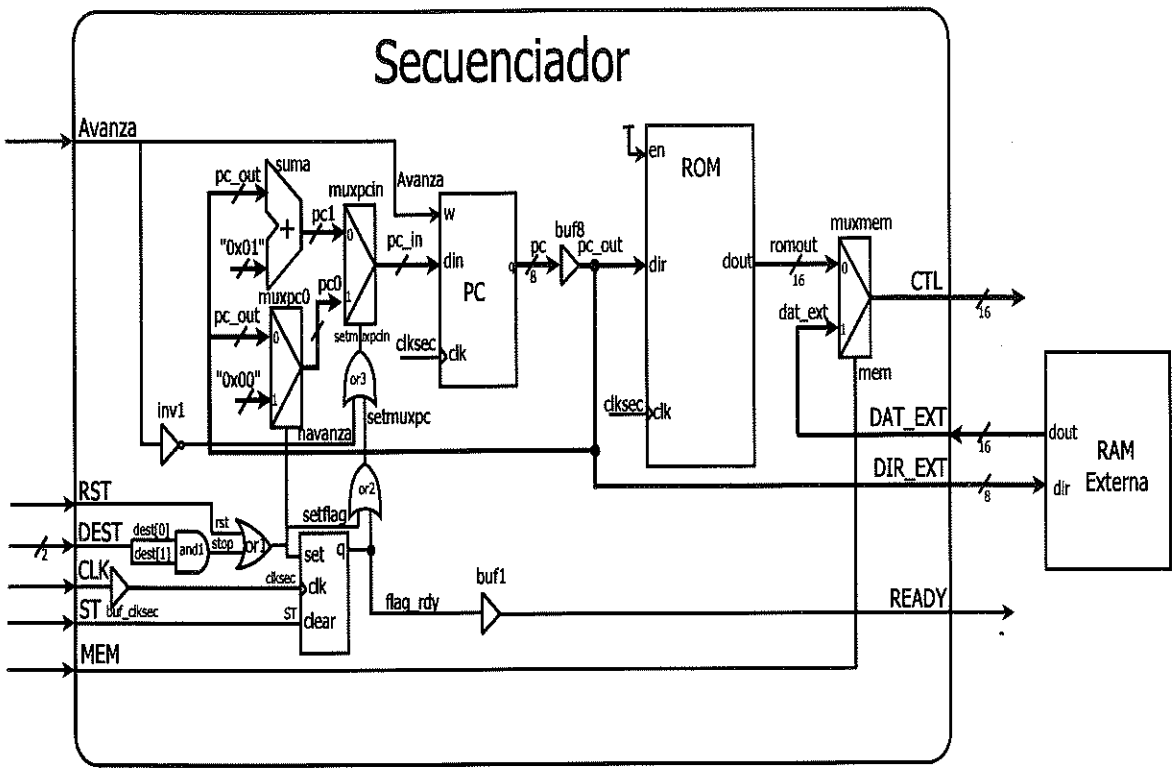


Figura 4.24: Diagrama a detalle del secuenciador.

funcionalidad del algoritmo implementado. La ejecución del algoritmo en chip comienza con la ejecución de la primera microinstrucción definida en la dirección 0 de la ROM interna o RAM externa. Las Tablas 4.12 a 4.15 describen las microistrucciones almacenadas en ROM que resuelven las operaciones matriciales de la Tabla 2.1, que dan solución al algoritmo convencional del FK. La nomenclatura usada para algunas matrices fue cambiada de la siguiente manera: $A = \phi$, $B = G$, $C = H$.

Una iteración completa requiere 148 ciclos de reloj. En la dirección 123_{10} está definido el código que detiene la ejecución del algoritmo, los bits 14 y 15 del bus de control se definen en alto haciendo que el contador de programa (PC) se establezca a cero, comenzando una nueva iteración del algoritmo.

En la codificación del algoritmo se consideró la formulación convencional con la modificación propuesta por Joseph para el cálculo de la covarianza. Una posibilidad de optimizar ciclos de ejecución implica codificar solo la formulación convencional, reduciendo el número requeridos de ciclos de reloj a solo 112, logrando con esto una reducción del 25 % en el tiempo de ejecución del algoritmo.

4.5. Síntesis VLSI

Para esta etapa de diseño, se requirieron herramientas de síntesis VLSI como las descritas en la Secc. 3.2.5. En el presente proyecto se utilizó Alliance CAD System [56] bajo el entorno operativo de Linux, empleando la distribución de Fedora^{TM}.

La codificación lógica tiene una estructura jerárquica de acuerdo a la Fig.4.13, siendo tres los bloques principales: secuenciador, banco de datos y unidad aritmética. Estos tres bloques son sintetizados independientemente de tal forma que se codifica un núcleo general (core) con el script core_script definido en el Apéndice A.

Cada uno de los bloques secuenciador, banco de datos y unidad aritmética tiene a su vez un conjunto de sub-bloques los cuales deben ser sintetizados por separado previo a la síntesis general del core. Los script utilizados para cada sub-bloque son sec_script, banco_script y sec_script. El orden para la síntesis de cada sub-bloque es independiente, sin embargo, para poder realizar la síntesis del core general es importante haber llevado a cabo la síntesis de todos los sub-bloques.

La Tabla 4.11 muestra la lista de los archivos fuente requeridos y el orden para realizar la síntesis de cada bloque; por ejemplo, para el bloque secuenciador primero debe ejecutarse el script flag_script, posteriormente rom_script y finalmente sec_script.

Cada uno de los script requiere de archivos fuentes descritos ya sea en código VHDL para Alliance o en forma estructural codificado en archivos en lenguaje C que se compilan con la herramienta *genlib* de Alliance. Por motivos de tramites de patente, se omitió la descripción del código fuente del proyecto.

En la Secc. 4.4 se describió la arquitectura y diseño lógico del chip PEMFK, el cual resuelve el algoritmo FK directamente en hardware. En esta Secc. se describen los resultados de los bloques principales de la arquitectura identificados como banco de datos, unidad aritmética y secuenciador.

El diseño a nivel comportamental permite que las herramientas de síntesis realicen la implementación de circuitos a nivel compuerta haciendo uso de celdas primitivas básicas para resolver operaciones aritméticas como la suma, produciendo resultados que aunque realizan la función especificada no son óptimos respecto al área y tiempos de retardo.

La unidad aritmética realiza las operaciones de suma, resta, multiplicación e inverso multiplicativo, durante la fase del diseño lógico, la unidad aritmética requirió especial atención debido a la importancia de hacer más eficientes las operaciones requeridas por el algoritmo FK, especialmente la operación de multiplicación que es la que se realiza con mayor frecuencia.

Para optimizar la unidad aritmética se evaluó la síntesis de sumadores y multiplicadores para diferentes esquemas lógicos y ancho de palabra (4, 8, 16, 24 y 32 bits) a través de una arquitectura jerárquica parametrizada de *n* bits descrita en la Secc. 4.4.

4.5.1. Síntesis de sumadores para *n* bits

Para optimizar la implementación de la arquitectura, se codificó y sintetizó un grupo de sumadores desde 2 hasta 16 bits. El estudio se realizó con diferentes esquemas lógicos en la implementación del sumador. La arquitectura comportamental (definida como tipo C en las Figs. 4.25 y 4.26) es codificada en base a compuertas digitales básicas como *xor*, *and* y *nand*, utilizando un lenguaje de descripción de hardware VHDL. Otra metodología en la síntesis es mediante una descripción en VHDL estructural definida como tipo E y E_half en las Figs. 4.25 y 4.26 haciendo uso de la celda primitiva para el sumador completo (*fulladder x2*) y el medio sumador (*halfadder x2*). La metodología tipo E y E_half es mucho más eficiente porque permite minimizar área y el número de transistores con respecto de la síntesis comportamental.

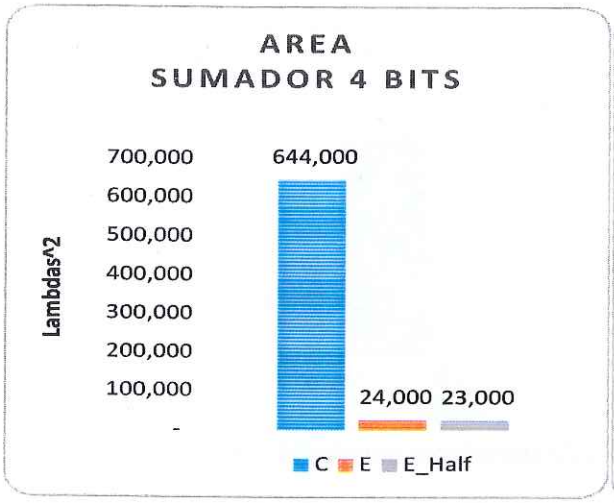
Las herramientas de síntesis de *Alliance* también permiten codificar estructuras estándar a través del uso de macros que facilitan la síntesis VLSI [61]. Los resultados de las Figs. 4.25 y 4.26 muestran la síntesis de sumadores de 4, 8 y 16 bits. Se observó que la síntesis utilizando la topología estructural, economiza área pero es la que tienen un mayor retardo. La síntesis utilizando macros en contraste son más rápidas pero consumen una mayor área. En consecuencia, las diferentes formas de expresar la síntesis de sumadores es valida, y depende en gran parte de las necesidades y recursos del diseñador. Para el presente proyecto estos resultados dieron la pauta para la utilización en algunos casos de sumadores sintetizados con macros de la herramienta *genlib* de Alliance y en casos como el diseño del multiplicador que se describe en la Secc. 4.4.2 fue de vital importancia la codificación estructural para optimizar principalmente el área en silicio.

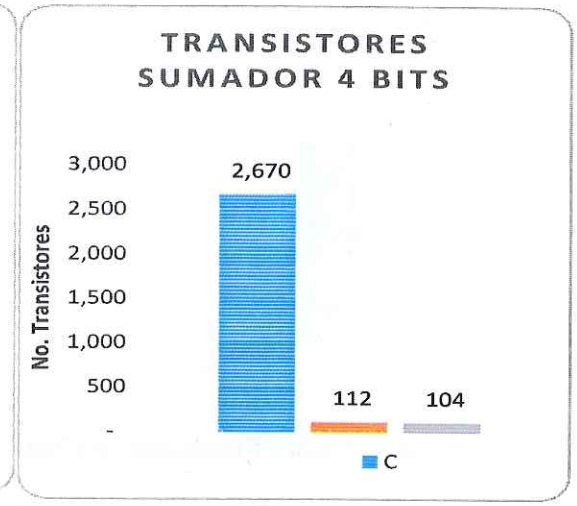
Los parámetros de retardos fueron obtenidos con la herramienta de *asimut* utilizando como tecnología una micra para el ancho del canal del transistor CMOS.

4.5.2. Síntesis de multiplicadores para n bits

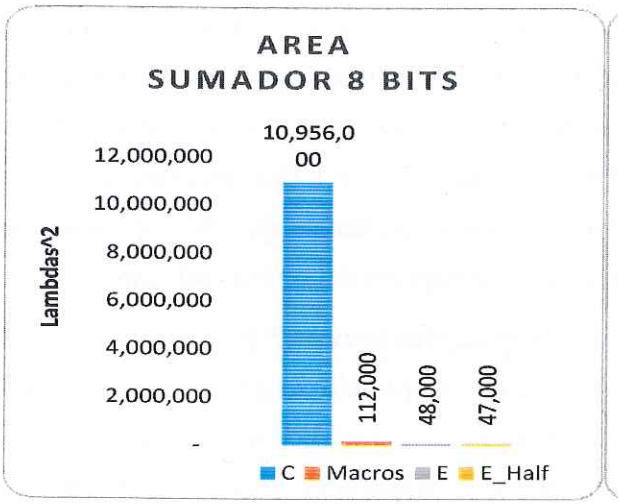
La multiplicación y el inverso multiplicativo son las operaciones que más atención requirieron durante la fase de diseño, debido a que su implementación VLSI requiere de una área de silicio y tiempos de retardo considerables. El diseño VLSI se implementó con modelos comportamentales (C) y estructurales (E) y (E_Half) utilizando el lenguaje de descripción de hardware VHDL.

TIPO	BITS	AREA		TRANSISTORES	RUTA CRITICA (ps)	PEOR RC
		lambdas	L^2			
C	4	805 x 800	644,000	2,670	4,224	432
E	4	240 x 100	24,000	112	4,878	32
E_Half	4	230 x 100	23,000	104	4,200	32





TIPO	BITS	AREA		TRANSISTORES	RUTA CRITICA (ps)	PEOR RC
		lambdas	L^2			
C	8	3320x3300	10,956,000	45,600	9,067	1692
Macros	8	320 x 350	112,000	482	3,560	92
E	8	240 x 200	48,000	224	9,758	32
E_Half	8	235 x 200	47,000	216	9,080	32



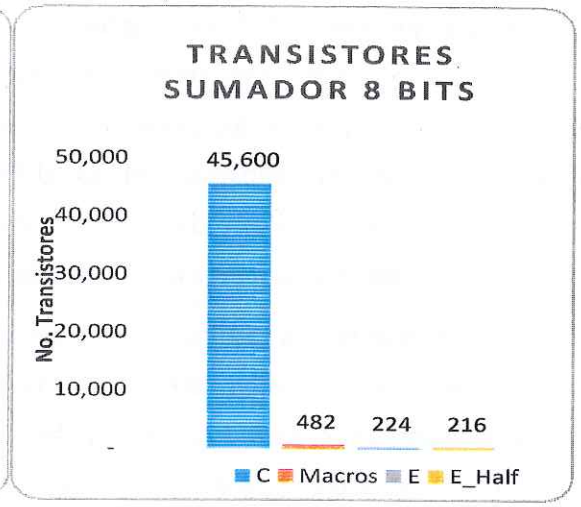


Figura 4.25: Resultados de síntesis de sumadores binarios de 4 y 8 bits.

TIPO	BITS	AREA		TRANSISTORES	RUTA CRITICA	PEOR RC
		lambdas	L^2		(ps)	
Macros	16	495 x 500	247,500	1,070	4,111	146
E	16	320 x 300	96,000	448	19,517	32
E_Half	16	475 x 200	95,000	432	18,839	32

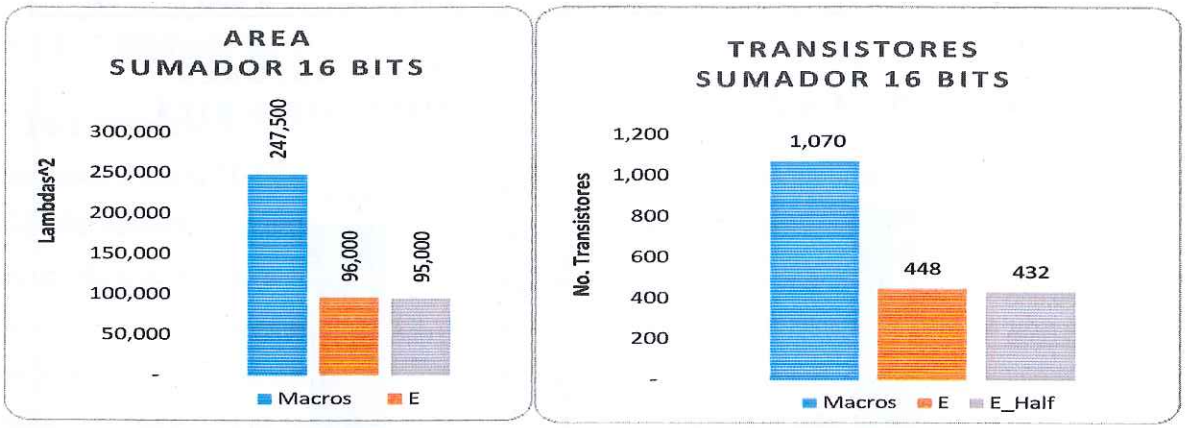
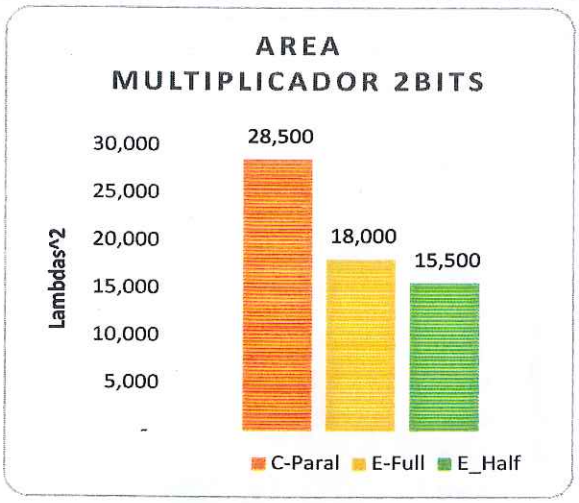


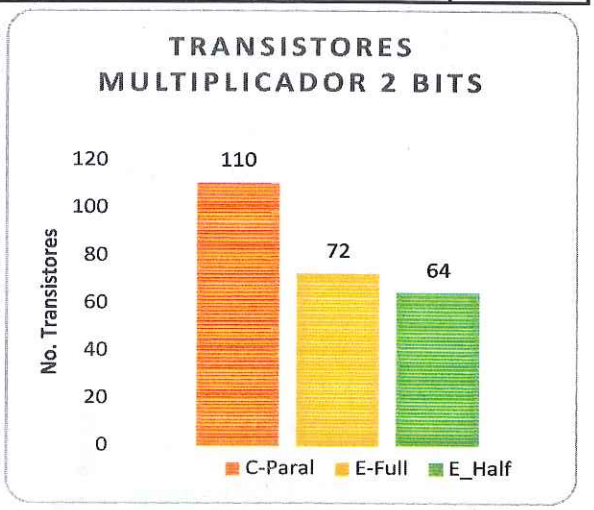
Figura 4.26: Resultados de síntesis de sumadores binarios de 16 bits.

Las Figs. 4.27 a 4.29 muestran los resultados de la síntesis para diferentes configuraciones tanto en el esquema lógico como el número de bits de los operandos. En general podemos apreciar que la síntesis de los modelos estructurales es considerablemente más óptima en área que la síntesis de modelos comportamentales . También se puede ver como el número de transistores puede incrementarse enormemente para multiplicadores de más de 32 bits, por lo que es importante optimizar lo mejor posible dichas configuraciones. El caso de los multiplicadores iterativos pueden ser más económicos en área pero tienen la enorme desventaja del tiempo de ejecución que requiere una operación, en el peor de los casos un número de ciclos equivalente al número de bits de los operándos. Por lo que si comparamos el caso del multiplicador serie de 8 bits con el estructural es muy poca la diferencia en área y el multiplicador definido estructuralmente ejecuta la operación de forma combinacional por lo que el tiempo de ejecución solo depende del retardo del circuito.

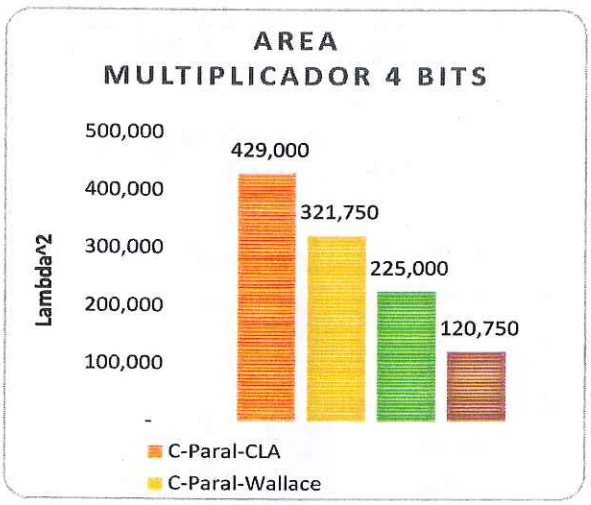
La Fig. 4.28 muestra los resultados de la síntesis de multiplicadores de 8 y 16 bits. Se aprecia como la multiplicación implementada en forma estructural ocupa solo 2,629 transistores contra la topología Carry Look Ahead (CLA) con 14,216 transistores, la cual ocupa 7 veces el área del estructural. Este ahorro en área de silicio implica un retardo de propagación mayor. La síntesis para este caso refleja un aumento de aproximadamente el doble de tiempo en la ruta crítica, haciendo más lento el circuito. Como resultado de esta experimentación se eligió el multiplicador de 24 bits tipo estructural por su gran ahorro de área comparado con el tipo Ripple y (CLA.

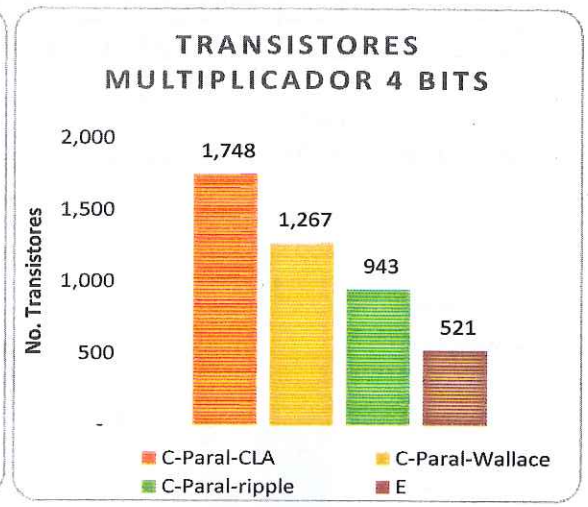
TIPO	BITS	AREA		TRANSISTORES	RUTA CRITICA	PEOR RC
		lambdas	L^2		(ps)	
C-Paral	2	190 x 150	28,500	110	1,285	100
E-Full	2	120 x 150	18,000	72	2,149	47
E_Half	2	155 x 100	15,500	64	1,484	46





TIPO	BITS	AREA		TRANSISTORES	RUTA CRITICA	PEOR RC
		lambdas	L^2		(ps)	
C-Paral-CLA	4	660 x 650	429,000	1,748	5,047	857
C-Paral-Wallace	4	585 x 550	321,750	1,267	3,364	509
C-Paral-ripple	4	500 x 450	225,000	943	5,098	127
E	4	345 x 350	120,750	521	9,394	47

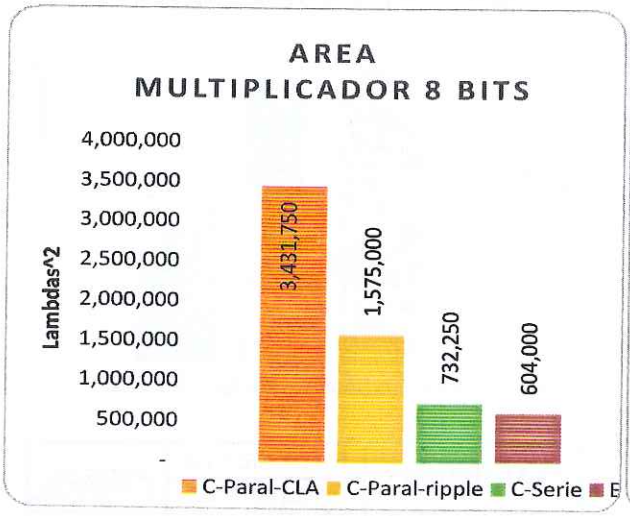


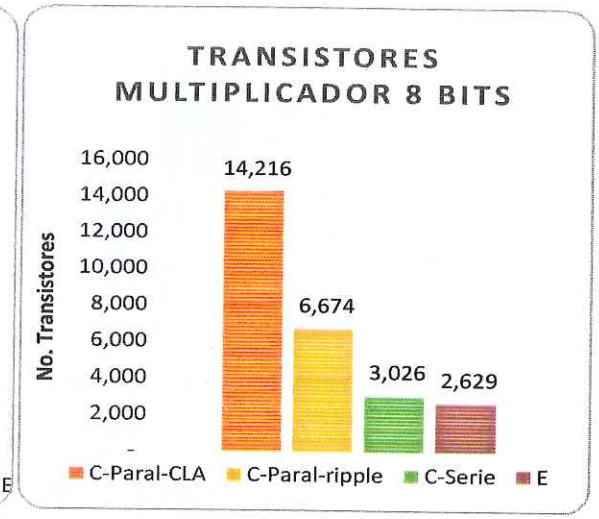


Nota: El tipo estructural utiliza celdas de suma de la librería sxlib
El tipo comportamental utiliza celdas de compuertas básicas para generar sumadores

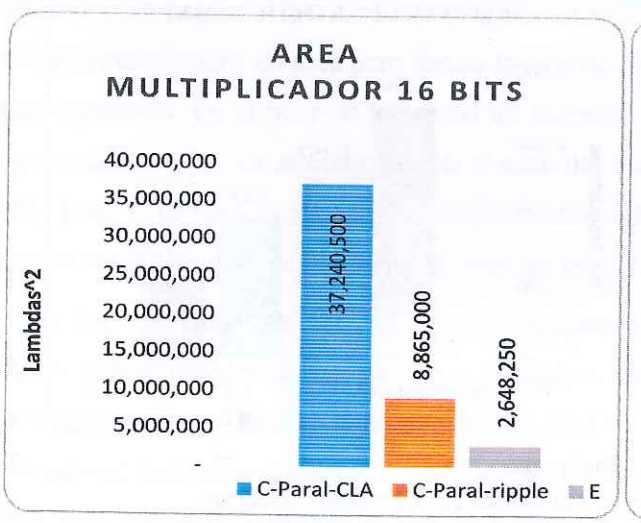
Figura 4.27: Resultados de síntesis de multiplicadores binarios de 2 y 4 bits.

TIPO	BITS	AREA		TRANSISTORES	RUTA CRITICA	PEOR
		lambdas	L^2		(ps)	RC
C-Paral-CLA	8	1855x1850	3,431,750	14,216	11,525	1263
C-Paral-ripple	8	1260x1250	1,575,000	6,674	12,076	230
C-Serie	8	505x1450	732,250	3,026	11,774	931
E	8	755x800	604,000	2,629	21,628	32





TIPO	BITS	AREA		TRANSISTORES	RUTA CRITICA	PEOR
		lambdas	L^2		(ps)	RC
C-Paral-CLA	16	6105X6100	37,240,500	122,188	21,924	32
C-Paral-ripple	16	2955x3000	8,865,000	37,950	21,897	4277
E	16	1605X1650	2,648,250	11,621	43,620	47



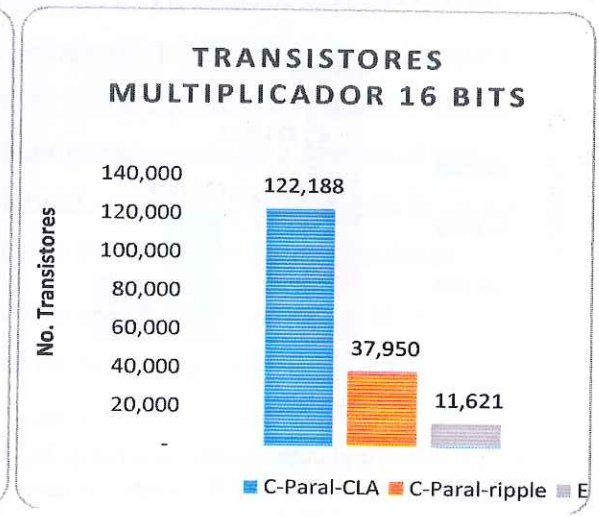


Figura 4.28: Resultados de síntesis de multiplicadores binarios de 8 y 16 bits.

TIPO	BITS	AREA		TRANSISTORES	RUTA CRITICA	PEOR
		lambdas	L^2		(ps)	RC
C-Paral-CLA	20	8580x8600	73,788,000	296,310	40,737	6759
E	32	3305x3350	11,071,750	48,709	85,132	47

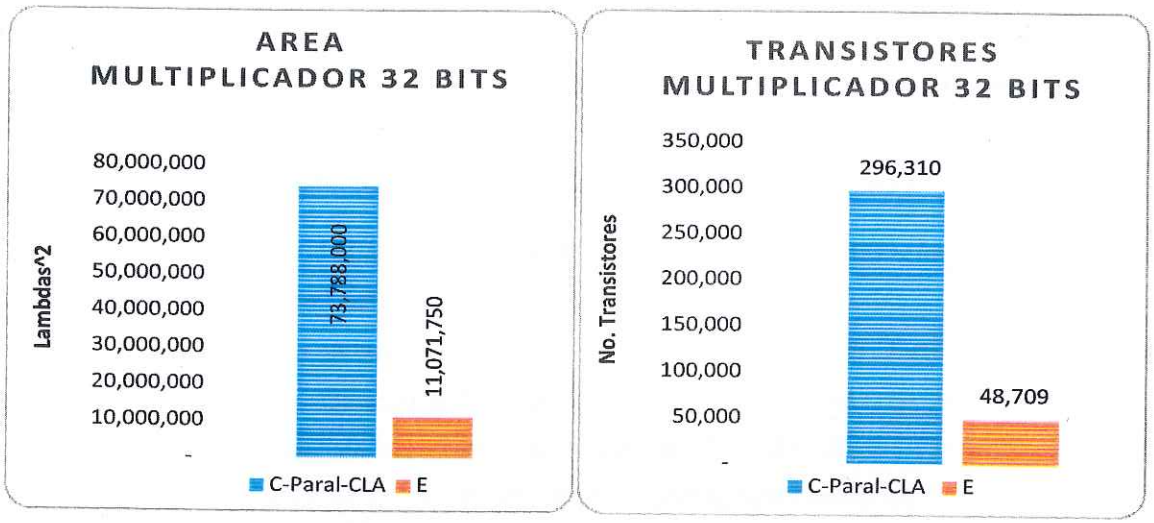


Figura 4.29: Resultados de síntesis de multiplicadores binarios de 32 bits.

4.6. Verificación

Durante la fase de diseño lógico es común provocar errores ya sea de sintaxis o de lógica, generando que la síntesis obtenida no cumpla con los requisitos establecidos para el diseño. Es por ello necesario que durante la etapa de síntesis de cada módulo se realice la verificación lógica a través de herramientas de simulación. La herramienta de *Alliance* que permite hacer verificación es *asimut*, la cual requiere del modelo comportamental o estructural en código VHDL, y de un archivo de patrones (.pat) que contenga los estímulos durante la fase de simulación. Para la generación de patrones existe una herramienta muy importante que es *genpat*, la cual permite codificar los patrones en código C, facilitando la creación de archivos con miles o millones de vectores de acuerdo a las necesidades.

Con los scripts definidos en el Apéndice A al mismo tiempo que se generan los modelos para cada bloque, se generan los archivos de patrones (.pat) que son utilizados en la simulación individual de cada módulo. Al mismo tiempo es importante verificar que los resultados de la simulación sean los correctos, esto se puede realizar revisando visualmente los archivos de salida de la simulación con la herramienta *xpat* de Alliance.

4.7. Diseño físico

Para realizar el diseño físico primero se define la tecnología de proceso en la que se fabricará el CI y posteriormente preparar el *Layout* con la distribución física de cada una de las capas requeridas en el proceso de fabricación. El chip PEMFK se implementó con la tecnología de proceso CMOS 0.5 μm con tres capas de metales y dos de polisilicio (proceso C5 de On Semiconductor [55]). El proceso de fabricación C5 está enfocado a aplicaciones de 5V de señal mixta. Por razones del costo elevado de prototipado y las restricciones establecidas en el programa MEP [54], la implementación fue restringida a una área de $3\,mm \times 3\,mm$ con un empaquetado de montaje superficial tipo LCC84.

El diseño del *layout* comienza con la herramienta *ocp* que realiza la colocación de celdas primitivas y *nero* que lleva a cabo la interconexión entre las mismas, ambas herramientas son parte de la suite de diseño Alliance CAD System [59] (Secc. 3.3.1). Posteriormente con la herramienta *cougar* se hace una extracción del circuito estructural en formato .al de parámetros a partir del layout obtenido para posteriormente realizar la verificación con la herramienta *lvx*, que compara los resultados obtenidos en la etapa de diseño lógico contra el diseño físico. Si en esta fase de la síntesis no coinciden ambos diseños, no podemos continuar a la fase siguiente. Debido a que las herramientas *ocp*, *nero* y *cougar* trabajan con layouts que utilizan como unidad de medida simbólica lambda (λ), es necesario realizar el escalamiento a unidades reales en micras. La herramienta *s2r* permite hacer la conversión de unidades simbólicas a unidades reales de 1 micra. El siguiente paso es realizar un escalamiento a las dimensiones del proceso tecnológico en el que se fabricará. En este paso se utilizó el software *sx10_c60* desarrollado por el Dr. Marco Antonio Gurrola, investigador de la Universidad de Guadalajara y coasesor del presente trabajo. El diseño físico del dado (Fig. 4.30a) se definió con una área de 5.6 mm^2 mediante el uso de la herramienta *Tanner EDA*^{TM}. Este paquete incluye herramientas que permiten dibujar las máscaras del layout, y verificar que el esquemático y las máscaras del layout tengan la misma topología. Con la herramienta *L-Edit*, se dibujan las máscaras que servirán para realizar el proceso de fotolitografía. Para utilizar esta herramienta se debe usar un archivo de "Tecnología" de acuerdo al proceso en el que será fabricado el circuito integrado. Este archivo indica que máscaras están disponibles en la tecnología y cuales son sus nombres y las reglas de diseño. Para asegurarse que un layout está correctamente dibujado, debe cumplir con las reglas de diseño y debe ser igual topológicamente al esquemático diseñado. Para realizar la comparación, entre el esquemático y el layout, se debe extraer un archivo en formato de Spice del esquemático, y también uno del layout con la herramienta *cougar*. Luego con la herramienta LVS de Tanner, se comparan los archivos de nodos extraídos. Si en esta fase del diseño, la comparación coincide, esto asegura cierto grado de certidumbre en la etapa de validación.

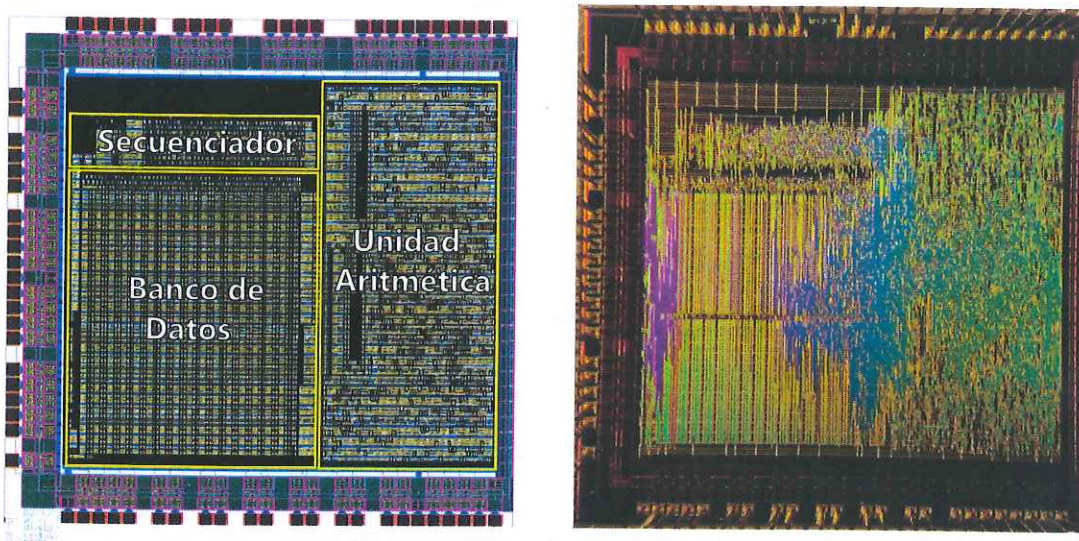
4.7.1. Estimación del consumo energético

La síntesis obtenida del PEMFK requirió de tan solo 70,197 transistores. Tomando como referencia los parámetros eléctricos de la tecnología de On Semiconductor C5 (ver Apéndice G), empleada para la fabricación, donde la potencia máxima queda definida por la Ecuación (4.20) y considerando que una compuerta en promedio contiene 4 transistores, podemos estimar una potencia máxima de 27 mW/MHz. Si la frecuencia para el circuito integrado PEMFK fuera de 100 KHz y la tensión de alimentación fuera de 3.3V, la potencia máxima consumida sería de 2.7 mW/MHz con una corriente de 840 μA.

$$P_{max} = 1{,}58\mu W/MHz/compuerta \tag{4.20}$$

4.7.2. Fabricación

La fabricación es un proceso complejo y ajeno al presente trabajo, en el Apéndice G se encuentra una referencia básica sobre el proceso C5 de On Semiconductor [55]. El diseño del chip fue enviado a fabricación a MOSIS mediante el programa educativo para universidades y centros de investigación. El tiempo promedio para la etapa de fabricación es de tres meses. Cinco chips son los enviados por MOSIS, la Fig. 4.30b muestra la microfoto tomada al dado PEMFK fabricado con tecnología CMOS de On Semiconductor.



(a) Layout. **(b)** Microfoto.

Figura 4.30: CI PEMFK (dimensiones del dado: 3mm x 3mm).

Tabla 4.5: Mapa de memoria.

Dirección	Parámetro
0x00	Z
0x01	U
0x02	X1
0x03	X2
0x04	K11
0x05	K21
0x06	R11
0x07	P11
0x08	P12
0x09	P21
0x0A	P22
0x0B	T11
0x0C	T12
0x0D	T21
0x0E	T22
0x0F	S11
0x10	S12
0x11	S21
0x12	S22
0x13	Q11
0x14	Q12
0x15	Q21
0x16	Q22
0x17	PHI11
0x18	PHI12
0x19	PHI21
0x1A	PHI22
0x1B	G11
0x1C	G21
0x1D	H11
0x1E	H12
0x1F	UNO

Tabla 4.6: Selección de operación.

OP[1]	OP[0]	Registro	Ciclos de reloj requeridos
0	0	*Suma*	1
0	1	*Resta*	1
1	0	*Multiplicación*	1
1	1	*Inverso multiplicativo*	25

Tabla 4.7: Terminales de entrada/salida del módulo sumador-restador.

Terminal	Tipo	Descripción
OP	entrada	Selección de suma ('0') o resta ('1')
X	entrada	dato de entrada X
Y	entrada	dato de entrada Y
SR	salida	suma o resta de X, Y

Tabla 4.8: Terminales de entrada/salida del módulo inverso multiplicativo.

Terminal	Tipo	Descripción	Cantidad de Bits
M	entrada	dato de entrada	48
I	salida	recíproco de X	24
ESTART	entrada	inicia proceso de iteraciones	1
CLK	entrada	señal de reloj	1
RESET	entrada	señal de reset	1
RDY	salida	activa cuando el módulo está listo para operar	1

Tabla 4.9: Terminales del secuenciador.

Terminal	Tipo	Descripción	Bits
CLK	entrada	Señal de reloj	1
Avanza	entrada	Un estado alto permite que el *PC* se incremente	1
RST	entrada	Señal de reset	1
ST	entrada	Inicia proceso del secuenciador	1
MEM	entrada	Un estado bajo ejecuta el algoritmo desde la ROM interna	1
DEST	entrada	Determina cuando deja de ejecutar el algoritmo	2
DAT_EXT	entrada	bus de la memoria externa con las señales de control	16
CTL	salida	bus de la memoria ROM con las señales de control	16
DIR_EXT	salida	bus que direcciona la memoria externa	8
READY	salida	Señal activa cuando una iteración se ha completado	1

Tabla 4.10: Bus de control.

Funcion	Bits	Descripción
DIRA	0-4	Define la dirección de lectura en el banco de registros
DIRB	5-9	Define la dirección de escritura en el banco de registros
SEL	10-11	Selecciona la fuente de datos (RA, RB, RQ, RD)
OP	12-13	Elige la operación a realizar (suma, resta, multiplicación, inversa)
DEST	14-15	Selecciona el destino del resultado (RB, RQ, RD)

Tabla 4.11: Archivos requeridos para la síntesis de la arquitectura PEMFK.

	fuente	destino	fuente	destino
SECUENCIADOR			flag_script	
	flag.vhdl	flag.vst flag.ap	flag datos.c	flag datos.pat flag simula.pat
			rom_script	
	rom.vbe	rom.vst rom.ap	rom datos.c	rom datos.pat rom simula.pat
			sec_script	
	sec.c	sec.vst sec.ap	sec datos.c	sec datos.pat sec simula.pat
BANCO DE DATOS			banco_regcore_script	
	banco reg core.c banco reg ctl.c banco reg dpt.c	banco reg core.vst banco reg core.ap	banco reg core datos.c	banco reg core datos.pat banco reg core simula.pat
			banco_decod_script	
	banco decod.vhdl	banco decod.vst banco decod.ap	banco decod datos.c	banco decod datos.pat banco decod simula.pat
			banco_sinc_write_script	
	banco sinc write.vhdl	banco sinc write.vst banco sinc write.ap	banco sinc write datos.c	banco sinc write datos.pat banco sinc write simula.pat
			banco_script	
	banco.c	banco.vst banco.ap	banco datos.c	banco datos.pat banco simula.pat
UNIDAD ARITMETICA			inv_comp_script	
	inv comp.vhdl	inv comp.vst	inv comp datos.c	inv comp datos.p inv comp simula.pat
			inv_controla_script	
	inv control.fsm	inv controla.vst	inv controla datos.c	inv controla datc inv controla simula.pat
			inv_meta_script	
	inv meta.vhdl	inv meta.vst	inv meta datos.c	inv meta datos.c inv meta simula.pat
			inv_regy_script	
	inv regv.vhdl	inv regv.vst	inv regv datos.c	inv regv datos.p inv regv simula.pat
			ua_inv_script	
	ua inv.c	ua inv.vst		
			ua_multiplica_script	
	ua multiplica.c sm1,sm3,sm6,sm12.c sc1,sc3,sc6,sc12.c m3, m6, m12.c	ua multiplica.vst	ua multiplica datos.c	ua multiplica dat ua multiplica simula.pat
			ua_sr_script	
	ua sr.c	ua sr.vst	ua sr datos.c	ua sr datos.pat ua sr simula.pat
			ua_script	
	ua.c	ua.vst	ua datos.c	ua datos.pat ua simula.pat
			sinc_sr_script	
	sinc sr.vhdl	sinc sr.vs sinc sr.ap	sinc sr datos.c	sinc sr datos.pat sinc sr simula.pat
PEMFK			core_script	
	core.c core.ioc	core.vst pemfk.ap core tran pemfk.spi	core datos.c	core datos.pat core simula.pat

Tabla 4.12: Microinstrucciones (primera parte).

						Código de Operación																		
						dest		op		sel		dirb					dira							
Función	ciclo	destino	R	OP	S	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CTL[15:0]	DIR
PA^T	1	D	P11	x	A11	1	0	1	0	0	1	1	0	1	1	1	0	0	1	1	1	0xA6E7	0
	2	Q	P12	x	A12	0	1	1	0	0	1	1	1	0	0	0	0	1	0	0	0	0x6708	1
	3	T11	D	+	Q	0	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	0x097F	2
	4	D	P11	x	A21	1	0	1	0	0	1	1	1	0	0	1	0	0	1	1	1	0xA727	3
	5	Q	P12	x	A22	0	1	1	0	0	1	1	1	0	1	0	0	1	0	0	0	0x6748	4
	6	T12	D	+	Q	0	0	0	0	1	0	0	1	1	0	0	1	1	1	1	1	0x099F	5
	7	D	P21	x	A11	1	0	1	0	0	1	1	0	1	1	1	0	1	0	0	1	0xA6E9	6
	8	Q	P22	x	A12	0	1	1	0	0	1	1	1	0	0	0	0	1	0	1	0	0x670A	7
	9	T21	D	+	Q	0	0	0	0	1	0	0	1	1	0	1	1	1	1	1	1	0x09BF	8
	10	D	P21	x	A21	1	0	1	0	0	1	1	1	0	0	1	0	1	0	0	1	0xA729	9
	11	Q	P22	x	A22	0	1	1	0	0	1	1	1	0	1	0	0	1	0	1	0	0x674A	10
	12	T22	D	+	Q	0	0	0	0	1	0	0	1	1	1	0	1	1	1	1	1	0x09DF	11

						Código de Operación																		
						dest		op		sel		dirb					dira							
EJECUTA	CICLO	DESTINO	R	OP	S	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CTL[15:0]	DIR
APA^T	13	D	A11	x	T11	1	0	1	0	0	1	0	1	0	1	1	1	0	1	1	1	0xA577	12
	14	Q	A12	x	T21	0	1	1	0	0	1	0	1	1	0	1	1	1	0	0	0	0x65B8	13
	15	P11	D	+	Q	0	0	0	0	1	0	0	0	1	1	1	1	1	1	1	1	0x08FF	14
	16	D	A11	x	T12	1	0	1	0	0	1	0	1	1	0	0	1	0	1	1	1	0xA597	15
	17	Q	A12	x	T22	0	1	1	0	0	1	0	1	1	1	0	1	1	0	0	0	0x65D8	16
	18	P12	D	+	Q	0	0	0	0	1	0	0	1	0	0	0	1	1	1	1	1	0x091F	17
	19	D	A21	x	T11	1	0	1	0	0	1	0	1	0	1	1	1	1	0	0	1	0xA579	18
	20	Q	A22	x	T21	0	1	1	0	0	1	0	1	1	0	1	1	1	0	1	0	0x65BA	19
	21	P21	D	+	Q	0	0	0	0	1	0	0	1	0	0	1	1	1	1	1	1	0x093F	20
	22	D	A21	x	T12	1	0	1	0	0	1	0	1	1	0	0	1	1	0	0	1	0xA599	21
	23	Q	A22	x	T22	0	1	1	0	0	1	0	1	1	1	0	1	1	0	1	0	0x65DA	22
	24	P22	D	+	Q	0	0	0	0	1	0	0	1	0	1	0	1	1	1	1	1	0x095F	23

						Código de Operación																		
						dest		op		sel		dirb					dira							
EJECUTA	CICLO	DESTINO	R	OP	S	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CTL[15:0]	DIR
$Q+APA^T$	25	P11	Q11	+	P11	0	0	0	0	0	1	0	0	1	1	1	1	0	0	1	1	0x04F3	24
	26	P12	Q12	+	P12	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0x0514	25
	27	P21	Q21	+	P21	0	0	0	0	0	1	0	1	0	0	1	1	0	1	0	1	0x0535	26
	28	P22	Q22	+	P22	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	0	0x0556	27
PC^T	29	D	P11	x	C11	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1	1	0xA7A7	28
	30	Q	P12	x	C12	0	1	1	0	0	1	1	1	1	1	0	0	1	0	0	0	0x67C8	29
	31	S11	D	+	Q	0	0	0	0	1	0	0	1	1	1	1	1	1	1	1	1	0x09FF	30
	32	D	P21	x	C11	1	0	1	0	0	1	1	1	1	0	1	0	1	0	0	1	0xA7A9	31
	33	Q	P22	x	C12	0	1	1	0	0	1	1	1	1	1	0	0	1	0	1	0	0x67CA	32
	34	S21	D	+	Q	0	0	0	0	1	0	1	0	0	0	1	1	1	1	1	1	0x0A3F	33

Tabla 4.13: Microinstrucciones (segunda parte).

						Código de Operación																	
						dest		op		sel		dirb					dira						
EJECUTA	CICLO	DESTINO	R	OP	S	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CTL[15:0]	DIR
CPC^T	35	D	C11	x	S11	1	0	1	0	0	1	0	1	1	1	1	1	1	1	0	1	0xA5FD	34
	36	Q	C12	x	S21	0	1	1	0	0	1	1	0	0	0	1	1	1	1	1	0	0x663E	35
	37	Q	D	+	Q	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	0x4BFF	36
$R+CPC^T$	38	D	R11	+	Q	1	0	0	0	0	0	1	1	1	1	1	0	0	1	1	0	0x83E6	37
INV	39-63	Q	D	INV	Q	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0x7BFF	38
K	64	K11	S11	x	Q	0	0	1	0	0	0	0	0	1	0	0	0	1	1	1	1	0x208F	39
	65	K21	S21	x	Q	0	0	1	0	0	0	0	0	1	0	1	1	0	0	0	1	0x20B1	40

						Código de Operación																	
						dest		op		sel		dirb					dira						
EJECUTA	CICLO	DESTINO	R	OP	S	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CTL[15:0]	DIR
KC	66	Q	K11	x	C11	0	1	1	0	0	1	1	1	1	0	1	0	0	1	0	0	0x67A4	41
	67	T11	Z	+	Q	0	0	0	0	1	1	0	1	0	1	1	1	1	1	1	1	0x0D7F	42
	68	Q	K11	x	C12	0	1	1	0	0	1	1	1	1	1	0	0	0	1	0	0	0x67C4	43
	69	T12	Z	+	Q	0	0	0	0	1	1	0	1	1	0	0	1	1	1	1	1	0x0D9F	44
	70	Q	K21	x	C11	0	1	1	0	0	1	1	1	1	0	1	0	0	1	0	1	0x67A5	45
	71	T21	Z	+	Q	0	0	0	0	1	1	0	1	1	0	1	1	1	1	1	1	0x0DBF	46
	72	Q	K21	x	C12	0	1	1	0	0	1	1	1	1	1	0	0	0	1	0	1	0x67C5	47
	73	T22	Z	+	Q	0	0	0	0	1	1	0	1	1	1	0	1	1	1	1	1	0x0DDF	48
KC P	74	D	T11	x	P11	1	0	1	0	0	1	0	0	1	1	1	0	1	0	1	1	0xA4EB	49
	75	Q	T12	x	P21	0	1	1	0	0	1	0	1	0	0	1	0	1	1	0	0	0x652C	50
	76	S11	D	+	Q	0	0	0	0	1	0	0	1	1	1	1	1	1	1	1	1	0x09FF	51
	77	D	T11	x	P12	1	0	1	0	0	1	0	1	0	0	0	0	1	0	1	1	0xA50B	52
	78	Q	T12	x	P22	0	1	1	0	0	1	0	1	0	1	0	0	1	1	0	0	0x654C	53
	79	S12	D	+	Q	0	0	0	0	1	0	1	0	0	0	0	1	1	1	1	1	0x0A1F	54
	80	D	T21	x	P11	1	0	1	0	0	1	0	0	1	1	1	0	1	1	0	1	0xA4ED	55
	81	Q	T22	x	P21	0	1	1	0	0	1	0	1	0	0	1	0	1	1	1	0	0x652E	56
	82	S21	D	+	Q	0	0	0	0	1	0	1	0	0	0	1	1	1	1	1	1	0x0A3F	57
	83	D	T21	x	P12	1	0	1	0	0	1	0	1	0	0	0	0	1	1	0	1	0xA50D	58
	84	Q	T22	x	P22	0	1	1	0	0	1	0	1	0	1	0	0	1	1	1	0	0x654E	59
	85	S22	D	+	Q	0	0	0	0	1	0	1	0	0	1	0	1	1	1	1	1	0x0A5F	60

						Código de Operación																	
						dest		op		sel		dirb					dira						
EJECUTA	CICLO	DESTINO	R	OP	S	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CTL[15:0]	DIR
P^+	86	Q	P11	-	S11	0	1	0	1	0	1	0	1	1	1	1	0	0	1	1	1	0x55E7	61
	87	P11	Z	+	Q	0	0	0	0	1	1	0	0	1	1	1	1	1	1	1	1	0x0CFF	62
	88	Q	P12	-	S12	0	1	0	1	0	1	1	0	0	0	0	0	1	0	0	0	0x5608	63
	89	P12	Z	+	Q	0	0	0	0	1	1	0	1	0	0	0	1	1	1	1	1	0x0D1F	64
	90	Q	P21	-	S21	0	1	0	1	0	1	1	0	0	0	1	0	1	0	0	1	0x5629	65
	91	P21	Z	+	Q	0	0	0	0	1	1	0	1	0	0	1	1	1	1	1	1	0x0D3F	66
	92	Q	P22	-	S22	0	1	0	1	0	1	1	0	0	1	0	0	1	0	1	0	0x564A	67
	93	P22	Z	+	Q	0	0	0	0	1	1	0	1	0	1	0	1	1	1	1	1	0x0D5F	68

Tabla 4.14: Microinstrucciones (tercera parte).

						Código de Operación																	
						dest		op		sel		dirb					dira						
EJECUTA	CICLO	DESTINO	R	OP	S	I5	I4	I3	I2	I1	I0	9	8	7	6	5	4	3	2	1	0	CTL[15:0]	DIR
I-KC	94	Q	UNO	-	T11	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	0x557F	69
	95	S11	Z	+	Q	0	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	0x0DFF	70
	96	Q	UNO	+	T21	0	1	0	0	0	1	0	1	1	0	1	1	1	1	1	1	0x45BF	71
	97	S12	UNO	-	Q	0	0	0	1	0	0	1	0	0	0	0	1	1	1	1	1	0x121F	72
	98	Q	UNO	+	T12	0	1	0	0	0	1	0	1	1	0	0	1	1	1	1	1	0x459F	73
	99	S21	UNO	-	Q	0	0	0	1	0	0	1	0	0	0	1	1	1	1	1	1	0x123F	74
	100	Q	UNO	-	T22	0	1	0	1	0	1	0	1	1	1	0	1	1	1	1	1	0x55DF	75
	101	S22	Z	+	Q	0	0	0	0	1	1	1	0	0	1	0	1	1	1	1	1	0x0E5F	76
P-KCP(I-T)	102	D	S11	x	P11	1	0	1	0	0	1	0	0	1	1	1	0	1	1	1	1	0xA4EF	77
	103	Q	S21	x	P12	0	1	1	0	0	1	0	1	0	0	0	1	0	0	0	1	0x6511	78
	104	T11	D	+	Q	0	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	0x097F	79
	105	D	S12	x	P11	1	0	1	0	0	1	0	0	1	1	1	1	0	0	0	0	0xA4F0	80
	106	Q	S22	x	P12	0	1	1	0	0	1	0	1	0	0	0	1	0	0	1	0	0x6512	81
	107	T12	D	+	Q	0	0	0	0	1	0	0	1	1	0	0	1	1	1	1	1	0x099F	82
	108	D	S11	x	P21	1	0	1	0	0	1	0	1	0	0	1	0	1	1	1	1	0xA52F	83
	109	Q	S21	x	P22	0	1	1	0	0	1	0	1	0	1	0	1	0	0	0	1	0x6551	84
	110	T21	D	+	Q	0	0	0	0	1	0	0	1	1	0	1	1	1	1	1	1	0x09BF	85
	111	D	S12	x	P21	1	0	1	0	0	1	0	1	0	0	1	1	0	0	0	0	0xA530	86
	112	Q	S22	x	P22	0	1	1	0	0	1	0	1	0	1	0	1	0	0	1	0	0x6552	87
	113	T22	D	+	Q	0	0	0	0	1	0	0	1	1	1	0	1	1	1	1	1	0x09DF	88
KR	114	Q	K11	x	R11	0	1	1	0	0	1	0	0	1	1	0	0	0	1	0	0	0x64C4	89
	115	S11	Z	+	Q	0	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	0x0DFF	90
	116	Q	K21	x	R11	0	1	1	0	0	1	0	0	1	1	0	0	0	1	0	1	0x64C5	91
	117	S21	Z	+	Q	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1	0x0E3F	92
KRK^T	118	Q	K11	x	S11	0	1	1	0	0	1	0	1	1	1	1	0	0	1	0	0	0x65E4	93
	119	P11	Z	+	Q	0	0	0	0	1	1	0	0	1	1	1	1	1	1	1	1	0x0CFF	94
	120	Q	K21	x	S11	0	1	1	0	0	1	0	1	1	1	1	0	0	1	0	1	0x65E5	95
	121	P12	Z	+	Q	0	0	0	0	1	1	0	1	0	0	0	1	1	1	1	1	0x0D1F	96
	122	Q	K11	x	S21	0	1	1	0	0	1	1	0	0	0	1	0	0	1	0	0	0x6624	97
	123	P21	Z	+	Q	0	0	0	0	1	1	0	1	0	0	1	1	1	1	1	1	0x0D3F	98
	124	Q	K21	x	S21	0	1	1	0	0	1	1	0	0	0	1	0	0	1	0	1	0x6625	99
	125	P22	Z	+	Q	0	0	0	0	1	1	0	1	0	1	0	1	1	1	1	1	0x0D5F	100
P^+	126	P11	T11	+	P11	0	0	0	0	0	1	0	0	1	1	1	0	1	0	1	1	0x04EB	101
	127	P12	T12	+	P12	0	0	0	0	0	1	0	1	0	0	0	0	1	1	0	0	0x050C	102
	128	P21	T21	+	P21	0	0	0	0	0	1	0	1	0	0	1	0	1	1	0	1	0x052D	103
	129	P22	T22	+	P22	0	0	0	0	0	1	0	1	0	1	0	0	1	1	1	0	0x054E	104

Tabla 4.15: Microinstrucciones (cuarta parte).

						Código de Operación																	
						dest		op		sel		dirb					dira						
EJECUTA	CICLO	DESTINO	R	OP	S	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CTL[15:0]	DIR
A X	130	D	X1	x	A11	1	0	1	0	0	1	1	0	1	1	1	0	0	0	1	0	0xA6E2	105
	131	Q	X2	x	A12	0	1	1	0	0	1	1	1	0	0	0	0	0	0	1	1	0x6703	106
	132	T11	D	+	Q	0	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	0x097F	107
	133	D	X1	x	A21	1	0	1	0	0	1	1	1	0	0	1	0	0	0	1	0	0xA722	108
	134	Q	X2	x	A22	0	1	1	0	0	1	1	1	0	1	0	0	0	0	1	1	0x6743	109
	135	T21	D	+	Q	0	0	0	0	1	0	0	1	1	0	1	1	1	1	1	1	0x09BF	110
B u	136	Q	U1	x	B11	0	1	1	0	0	1	1	1	0	1	1	0	0	0	0	1	0x6761	111
	137	X1	T11	+	Q	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0x004B	112
	138	Q	U1	x	B21	0	1	1	0	0	1	1	1	1	0	0	0	0	0	0	1	0x6781	113
	139	X2	T21	+	Q	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1	0x006D	114
C X	140	D	X1	x	C11	1	0	1	0	0	1	1	1	1	0	1	0	0	0	1	0	0xA7A2	115
	141	Q	X2	x	C12	0	1	1	0	0	1	1	1	1	1	0	0	0	0	1	1	0x67C3	116
	142	Q	D	+	Q	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	0x4BFF	117
Y-CX	143	Q	Y1	-	Q	0	1	0	1	0	0	1	1	1	1	1	0	0	0	0	0	0x53E0	118
K(Y-C X)	144	T11	K11	x	Q	0	0	1	0	0	0	0	1	0	1	1	0	0	1	0	0	0x2164	119
	145	T21	K21	x	Q	0	0	1	0	0	0	0	1	1	0	1	0	0	1	0	1	0x21A5	120
X^+	146	X1	T11	+	X1	0	0	0	0	0	1	0	0	0	1	0	0	1	0	1	1	0x044B	121
	147	X2	T21	+	X2	0	0	0	0	0	1	0	0	0	1	1	0	1	1	0	1	0x046D	122
STOP	148	-	-	-	-	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0xCFFF	123

Capítulo 5

Resultados

En este capítulo se describen los resultados más importantes obtenidos durante el proceso de investigación. Primeramente se muestran los resultados de la implementación por software del algoritmo FK convencional de 2 estados empleando un modelo en Matlab. Para esto se utilizó una computadora portátil con procesador Intel Quad Core i7 de 2.4 GHz, esta actividad se realizó con el propósito de análizar el algoritmo, verificar los efectos de convergencia y divergencia con el empleo de aritmética de punto flotante y punto fijo y en base a los resultados llevar a cabo la determinación de los aspectos cuantitativos en la definición de la hipótesis definida en la Secc. 1.5. Posteriormente se describen los resultados de la implementación por software sobre un sistema embebido que emplea un procesador digital de 16 bits de la compañía Microchip cuya finalidad fue medir los tiempos de ejecución del algoritmo en tiempo real. Por último se muestran los resultados de la implementación en VLSI del FK que es el objetivo principal de la investigación y se concluye con la fase de validación del proyecto.

5.1. Implementación a través de un modelo en Matlab

Para llevar a cabo el análisis del algoritmo del FK se utilizaron las herramientas de Matlab-Simulink. La finalidad de este análisis fue comprobar la respuesta del algoritmo con la implementación de diferentes representaciones numéricas, verificando la estabilidad y convergencia numérica del mismo. Los resultados del análisis nos permitieron establecer la definición del formato numérico y el planteamiento de la arquitectura empleada.

Para la implementación por software del algoritmo FK convencional de 2 estados se empleó el modelo descrito en la Secc. 4.2.3. Los parámetros de simulación que se muestran en la Tabla 5.1

definen las dimensiones de vectores y matrices del sistema, así como sus valores iniciales para la simulación. El tipo de dato empleado en el modelo fue inicialmente un formato de punto flotante a 64 bits, de tal forma que se obtenga la mayor precisión, evitando problemas por efectos de redondeo. Los datos empleados durante esta fase de simulación fueron obtenidos con la plataforma descrita en la Secc. 4.1.3 bajo condiciones controladas de movimiento.

Los resultados de la simulación son mostrados en las gráficas de la Fig. 5.1 donde se puede observar la eficiencia del algoritmo. La curva de estimación representa la posición angular estimada con el mínimo error y compensando errores por efectos del bias causado en el giroscopio, y suavizando la respuesta que entregan los acelerómetros.

Tabla 5.1: Valores iniciales para simulación del FK.

Matriz/ Vector	Dimensión	valor inicial	Descripción
$\hat{x}$	$n \times 1$	[0;0]	vector de estimación de estados
Φ	$n \times n$	[1 - T; 0 1]	dinámica del sistema
G	$n \times m$	[T;0]	matriz de entradas
H	$r \times n$	[1 0]	matriz de medición
u	$m \times 1$	[0]	vector de control
z	$r \times 1$	[0]	vector de medición
Q	$n \times n$	[0.02247 0; 0 0.03]	covarianza de ruido del sistema
R	$r \times r$	[0.36]	covarianza de ruido del medición
P	$n \times n$	[1 0; 0 1]	matriz de covarianza del error
K	$n \times r$	[0;0]	matriz de ganancia de Kalman
x_o	$n \times 1$	[0;0]	estimación inicial de estados
P_o	$n \times n$	[1 0;0 1]	covarizanza inicial del error
T		20 ms	Periodo de muestreo

Para definir la representación numérica en la implementación del FK, se realizó un análisis sobre los dos tipos de aritmética y se verificó la convergencia y estabilidad del algoritmo convencional utilizando formatos con diferente precisión. La decisión de considerar el uso de la aritmética de punto fijo en lugar de punto flotante radica en que los diseños de las operaciones matriciales del algoritmo requieren menos área de silicio y en consecuencia un menor consumo de energía.

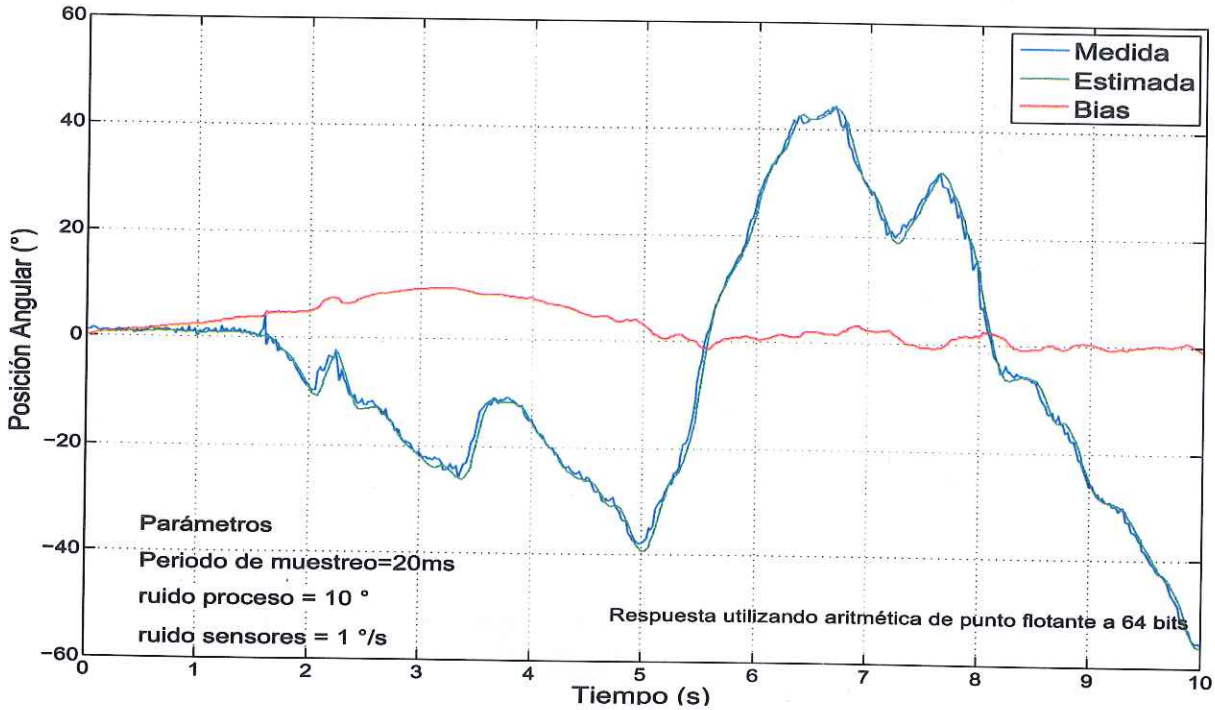
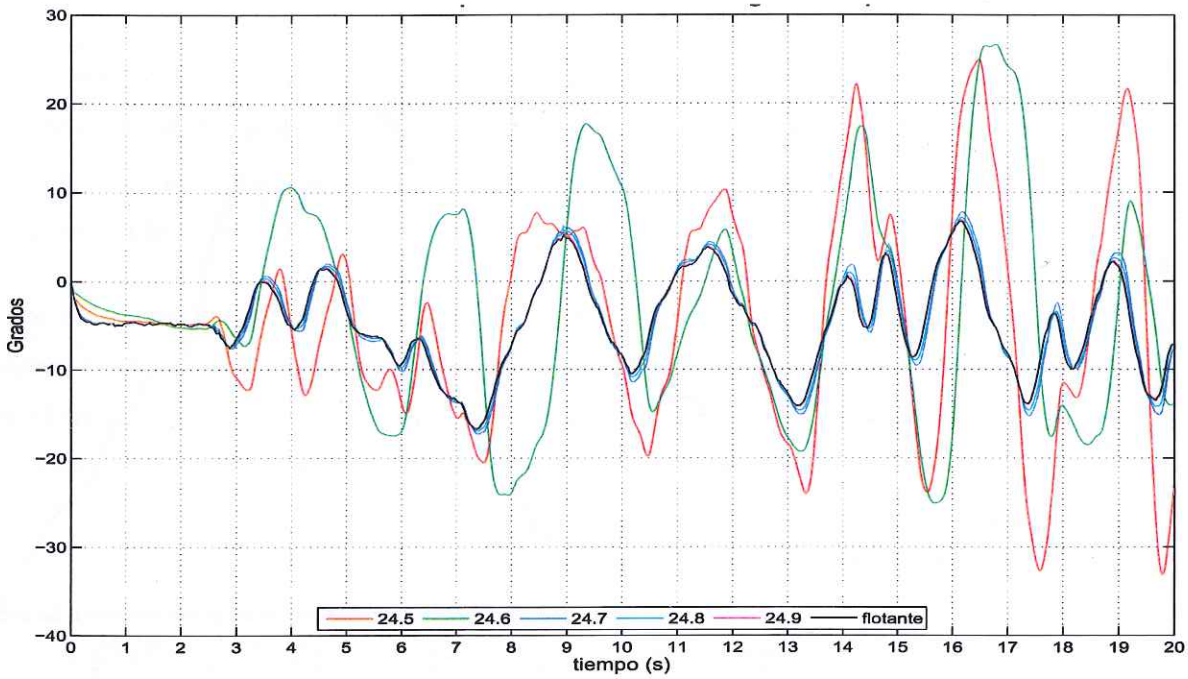


Figura 5.1: Estimación de posición angular con el FK utilizando punto flotante.

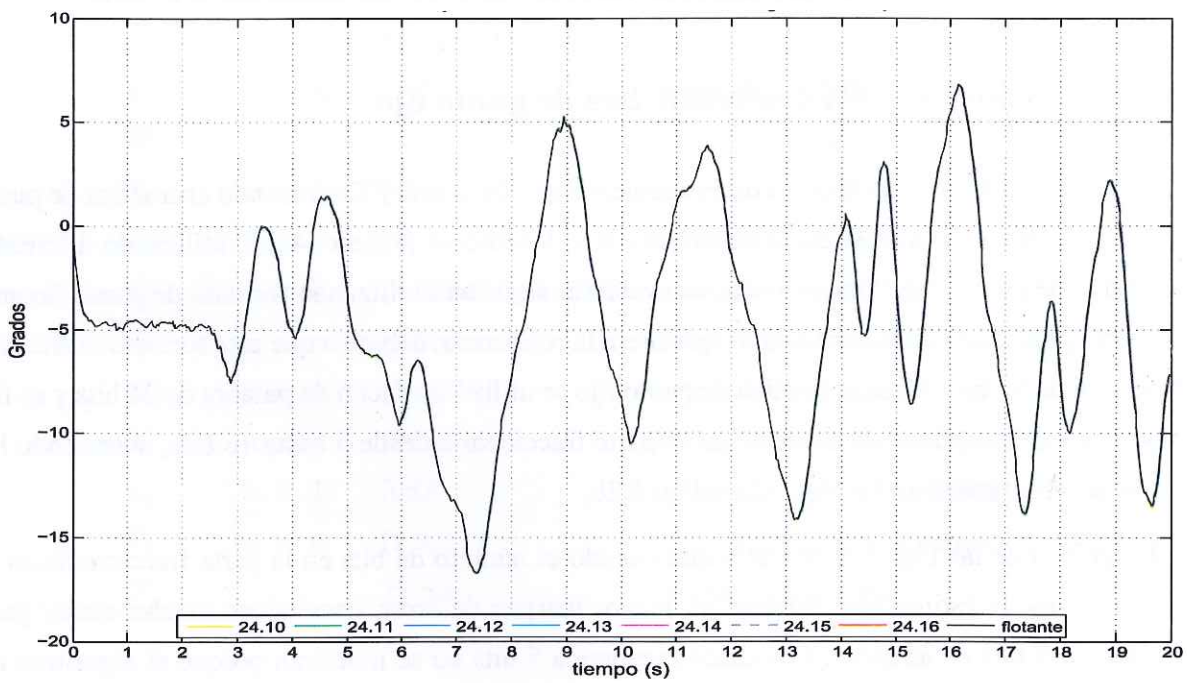
5.1.1. Análisis del FK con aritmética de punto fijo

Con la finalidad de analizar el comportamiento del algoritmo FK utilizando aritmética de punto fijo. Se modificó el modelo en Simulink-Matlab, descrito en la Secc. 4.2.3, utilizando diferentes formatos de punto fijo. Primeramente de evaluó el algoritmo utilizando formato de punto flotante de doble precisión con la finalidad de que fuera la referencia, debido a que este formato utiliza una precisión de 53 bits. En la aritmética de punto fijo se utilizó un ancho de palabra de 24 bits y se fue modificando la cantidad de dígitos para la parte fraccionaria desde 5 hasta 16 bits, obteniendo las respuestas mostradas en las Fig. 5.2a y Fig. 5.2b.

La gráfica de las Fig. 5.2a muestra que cuando el número de bits en la parte fraccionaria es 5, 6, 7 y 8 bits, la estimación contiene un mayor margen de error, haciéndose mucho menor para valores de 9 bits en adelante. Los casos menores a 5 bits no se muestran porque el algoritmo no realiza la estimación correctamente, esto es debido a la falta de precisión inherente al número de bits utilizados (para más detalle ver Apéndice D). Debido a que las estimaciones para valores de 10 bits en adelante son muy semejantes, como se muestra en la gráfica de la Fig. 5.2b, fue necesario llevar a cabo un análisis estadístico de estos resultados que nos llevaran a determinar los margenes de error producidos al modificar la precisión utilizada en los formatos de punto fijo.



(a) Respuesta utilizando de 5 a 9 bits en la parte fraccionaria



(b) Respuesta utilizando de 10 a 16 bits en la parte fraccionaria

Figura 5.2: Estimación de posición con diferente precisión.

Análisis estadístico del error.

El análisis del error se realizó con un conjunto de datos formado con 2000 muestras obtenidas experimentalmente con la plataforma descrita en la Secc. 4.1.3. Estos datos fueron tomados en un tiempo de 40 segundos con un periodo de muestreo de 20 milisegundos. Se simuló el algoritmo para cada uno de los diferentes formatos de precisión utilizando el mismo ancho de palabra (24 bits) y solo modificando la cantidad de bits asignada a la parte fraccionaria (bits menos significativos).

A partir de la respuesta de estimación de posición y bias del FK se obtuvieron vectores a los cuales se les calculó la desviación estándar, la media, el SNR y el error típico. Los resultados del análisis de error para las estimaciones de posición y bias se observan en las Tablas. 5.2 y 5.3. La primera columna indica el número de bits de precisión utilizados en el cálculo del algoritmo, y se puede identificar que conforme se aumenta el número de bits en la parte fraccionaria se decrementa la desviación estándar y en consecuencia el error típico. De la Fig. 5.3a se aprecia como el error en la estimación con 6, 7, 8 y 9 bits es considerablemente mayor que para los demás casos, en consecuencia cualquier formato por encima de 10 dígitos de precisión se considera satisfactorio. Sin embargo para una mayor definición se puede observar el orden de magnitud del error en la gráfica de la Fig. 5.3b, la disminución de un orden de magnitud se da aproximadamente por cada 3 bit de incremento en la parte faccionaria.

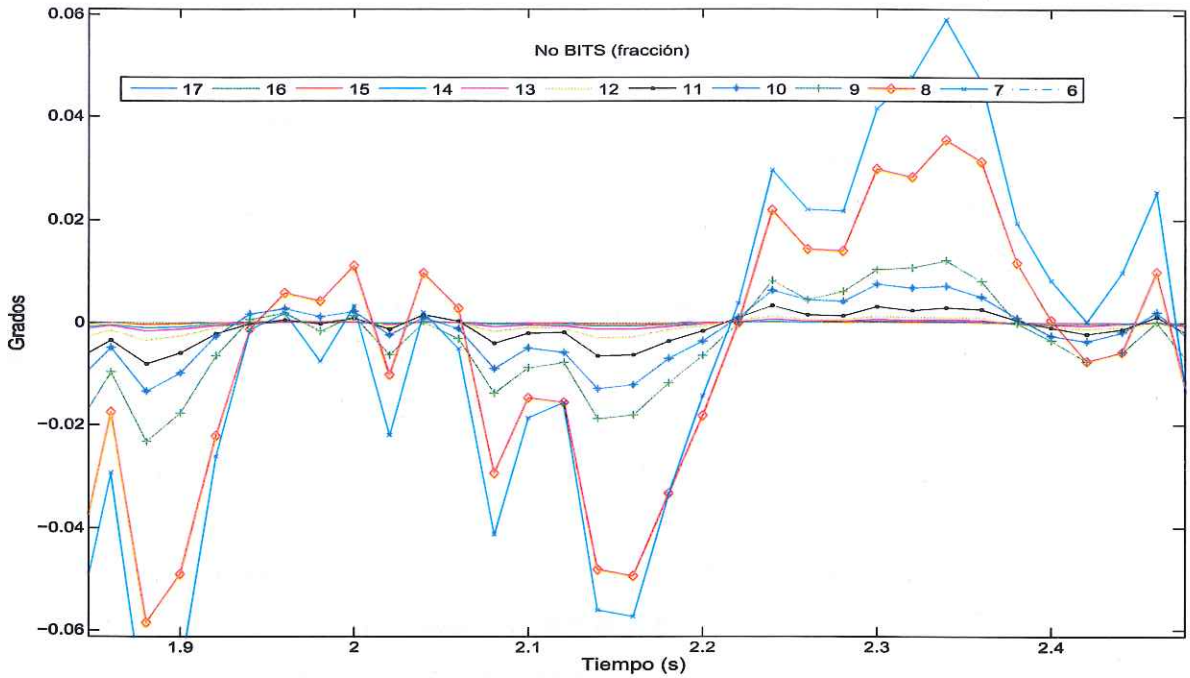
Los resultados de éste análisis mostraron que los formatos que tenían menos de 9 bits para la parte fraccionaria causaban problemas de convergencia, y en la parte entera se requieren al menos 10 bits para evitar problemas de desbordamiento. Se encontró que para que el algoritmo tenga una respuesta similar a la solución con formato de punto flotante, se requiere al menos un formato 20.9, esto es, 9 dígitos para la parte fraccionaria, 10 para la parte entera y uno para el signo. En base a estos resultados se decidió, de manera conservadora, utilizar un formato de 24.14 para la implementación VLSI del FK.

Tabla 5.2: Análisis estadístico del error de posición.

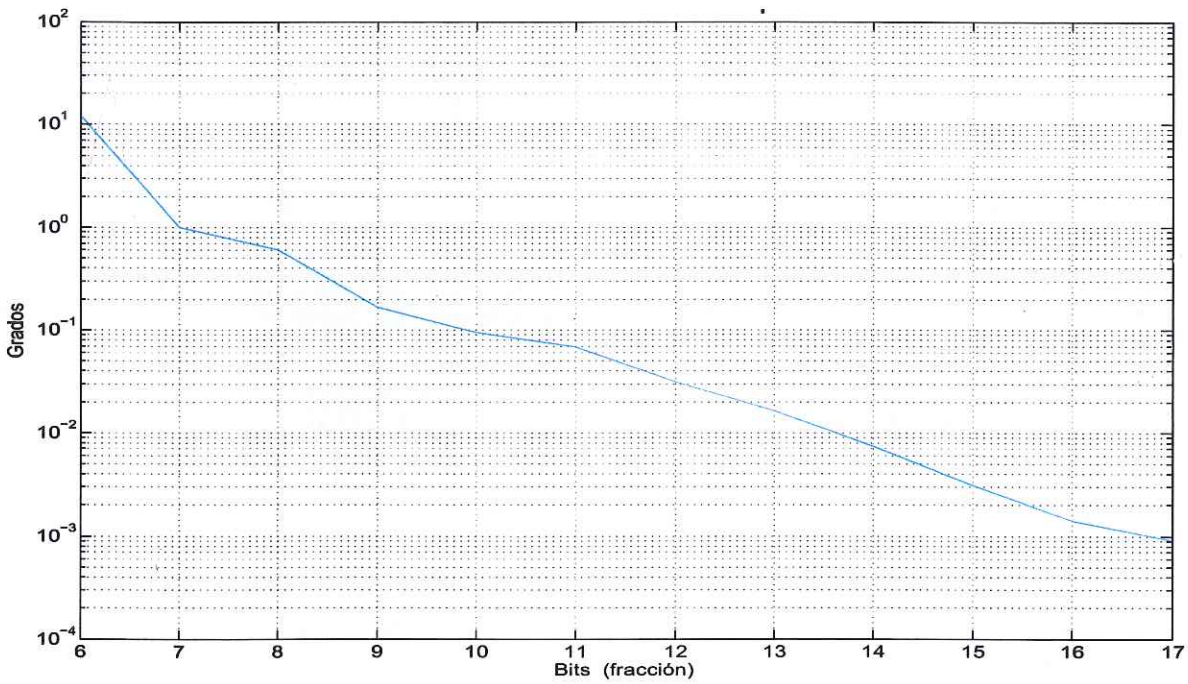
Bits	Std	Media	SNR	Error típico
6	12.60785	-1.64564	-0.13053	0.39850
7	1.03147	0.00653	0.00633	0.03260
8	0.61476	0.01144	0.01861	0.01943
9	0.16862	-0.00029	-0.00172	0.00533
10	0.09645	-0.00046	-0.00481	0.00305
11	0.06981	-0.00010	-0.00139	0.00221
12	0.03175	0.00008	0.00266	0.00100
13	0.01665	-0.00004	-0.00257	0.00053
14	0.00749	0.00000	-0.00001	0.00024
15	0.00317	-0.00003	-0.00793	0.00010
16	0.00144	0.00002	0.01607	0.00005
17	0.00092	-0.00002	-0.01711	0.00003

Tabla 5.3: Análisis estadístico del error del bias.

Bits	Std	Media	SNR	Error típico
6	1.47655	-0.02657	-0.01799	0.04667
7	0.55400	-0.01085	-0.01958	0.01751
8	0.37136	-0.00117	-0.00314	0.01174
9	0.13577	0.00025	0.00184	0.00429
10	0.08993	-0.00064	-0.00712	0.00284
11	0.03008	-0.00070	-0.02337	0.00095
12	0.02833	-0.00096	-0.03387	0.00090
13	0.01557	-0.00072	-0.04646	0.00049
14	0.01287	-0.00071	-0.05477	0.00041
15	0.01230	-0.00078	-0.06316	0.00039
16	0.01179	-0.00072	-0.06096	0.00037
17	0.00086	0.00000	-0.00477	0.00003



(a) Error para diferentes formatos.



(b) Desviación estándar del error de posición.

Figura 5.3: Error por efectos de precisión.

5.1.2. Implementación del FK por software utilizando la plataforma UDB5

Con la finalidad de poder realizar una comparación entre los resultados de la ejecución del algoritmo por software contra la ejecución en hardware en la etapa final de validación, fue necesario experimentar con otra plataforma para la adquisición de datos, siendo esta la tarjeta UDB5 de la compañía Sparkfun mostrada en la Fig. 4.1, la cual contiene la unidad de medición inercial MPU-6000, con 3 giroscopios y 3 acelerómetros de tipo MEMS. La tarjeta UDB5 también dispone de un procesador digital dsPIC33FJ256 de Microchip para acondicionamiento de los datos provenientes del MPU-6000 y su transmisión hacia una terminal en la PC a través del bus UART.

Los datos que se muestran en la Fig. 5.4 fueron obtenidos con la tarjeta UDB5 aplicando movimientos angulares y utilizando un tiempo de muestreo de 20 ms. Se observan 1000 muestras equivalentes a un periodo de 20 segundos. La Fig. 5.5 representa las aceleraciones en los ejes x, z en magnitud de gravedades. Debido a que la posición entre acelerómetros es ortogonal, por la colocación el acelerómetro z está sobre cero, mientras que el acelerómetro x le afecta la gravedad y se acerca a 1 g.

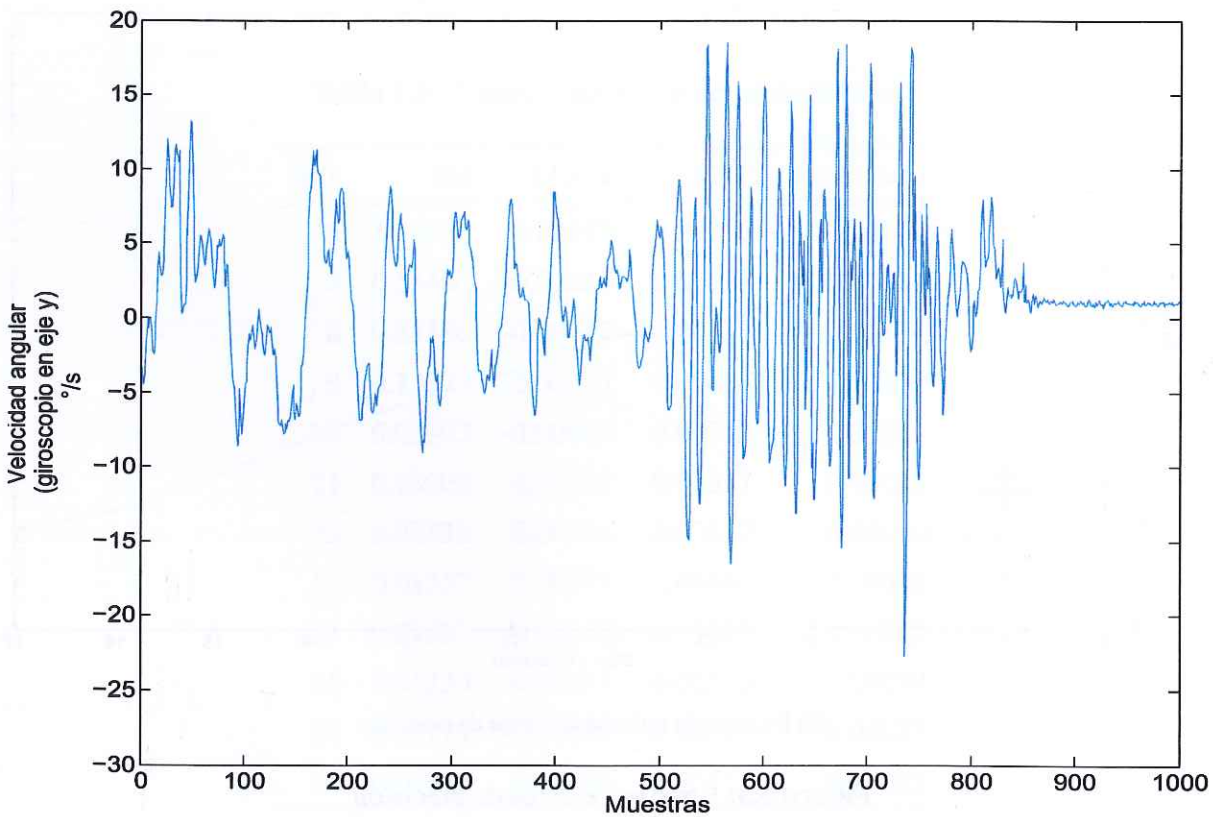


Figura 5.4: Velocidad angular (plataforma UDB5).

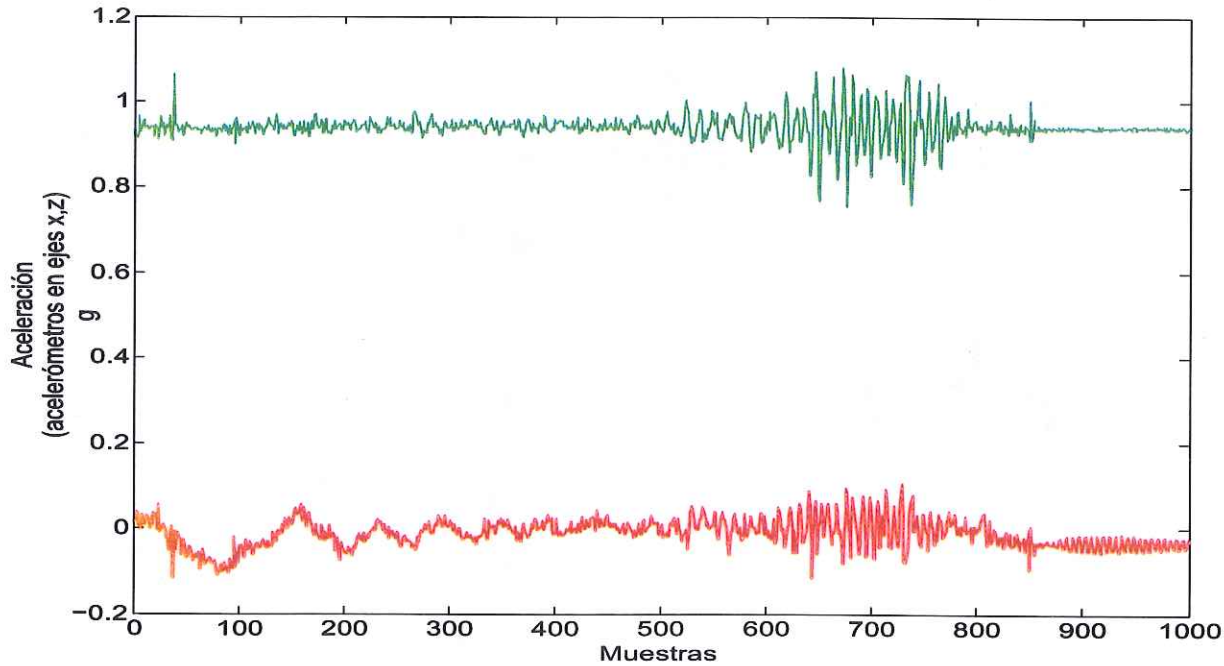


Figura 5.5: Aceleración (plataforma UDB5).

Los datos de esta plataforma se aplican como estímulo en la simulación del filtro de Kalman del modelo descrito en la Secc. 4.2.3, obteniendo los resultados mostrados en la Fig. 5.6 donde se distingue el efecto que produce el bias del giroscopio generando un error considerable respecto al tiempo. Aun cuando el cálculo de la posición angular utilizando la aceleración es precisa, presenta un efecto muy ruidoso, por lo que la respuesta del FK elimina el efecto del bias provocado por el giroscopio, y el efecto ruidoso de los acelerómetros.

5.1.3. Implementación del FK convencional en tiempo real

En la implementación del algoritmo FK convencional sobre un sistema embebido se empleó el procesador digital dsPIC30F5011 de Microchip© en conjunto con el sensor inercial ADIS16354 de Analog Device^{TM}(Fig. 5.7a). El objetivo de esta implementación implicó el análisis del tiempo de ejecución de un ciclo de iteración del algoritmo FK convencional bajo un esquema de programación secuencial sobre una plataforma embebida.

En la Fig. 5.7b se observan las señales SPI que contienen la información de los sensores provenientes de la ADIS16354. La primer señal contiene una serie de 6 pulsos que representan el instante de la lectura de cada uno de los sensores inerciales (3 acelerómetros y 3 giroscopios),

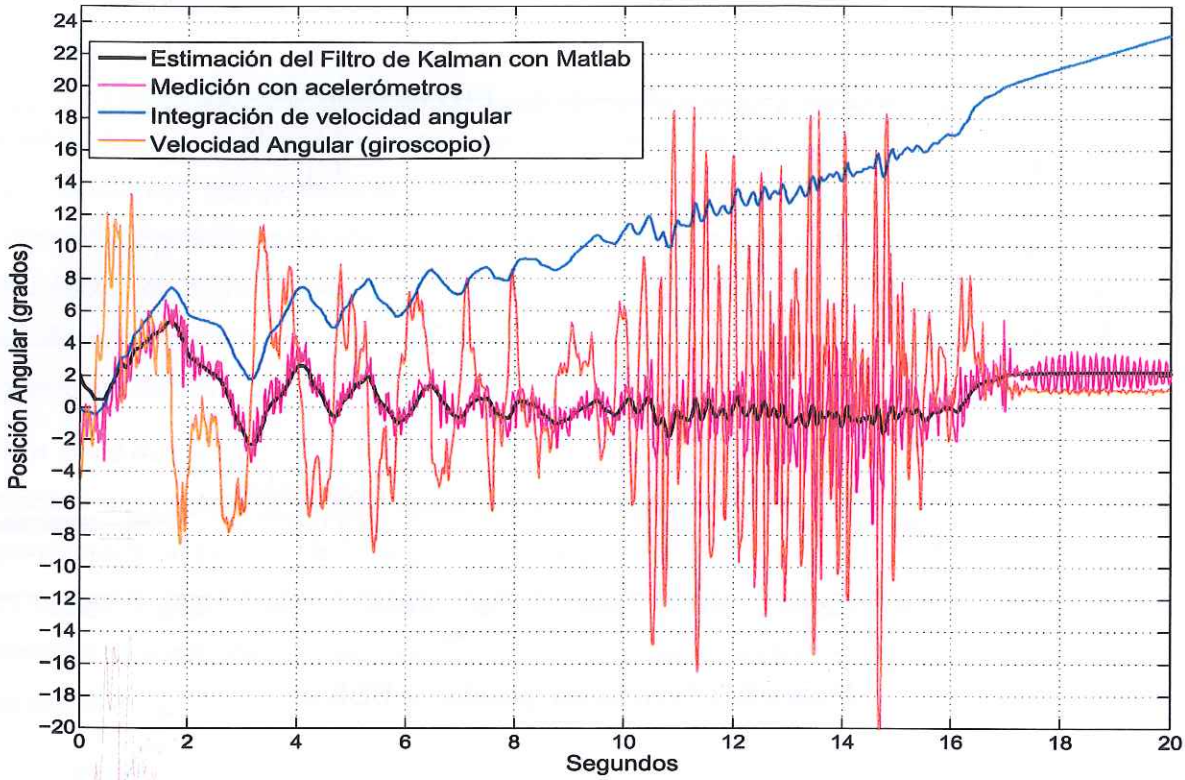
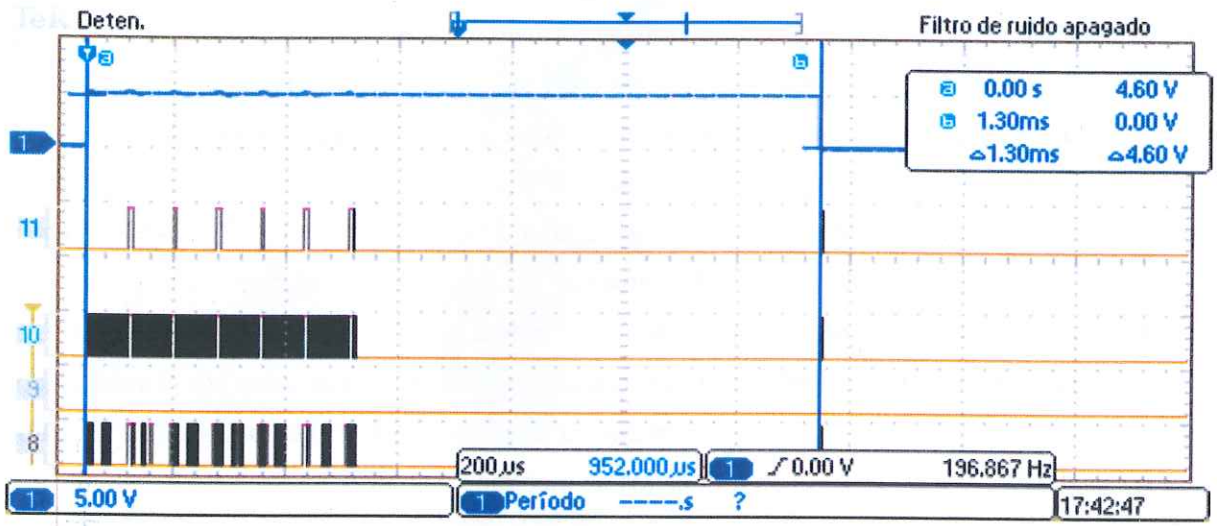


Figura 5.6: Resultado de simulación en Matlab del FK.

esto se realiza durante los primeros 500 μs. El siguiente pulso generado es el momento en que se termina de evaluar el algoritmo FK convencional, las mediciones realizadas reflejaron un tiempo de ejecución de 1.3 ms bajo condiciones de una estimación de 2 estados, un periodo de muestreo de 5 ms, una variable medida y una variable de entrada. Como podemos apreciar el tiempo de ejecución se encuentra en el rango de los milisegundos. Tomando como referencia el análisis de complejidad computacional llevado a cabo en la Secc. 2.5 se obtuvo la Tabla 5.4 la cual nos muestra claramente que si el FK fuera de 6 estados requeriríamos un promedio de 25.36 ms, lo cual para aplicaciones de tiempo real no es adecuado ya que durante una iteración (típicamente entre 5 y 20 ms) no solo se debe resolver el FK, sino otras tareas adicionales en función de la frecuencia de muestreo y de la aplicación en particular. Por lo tanto un sistema embebido como tal, no siempre es la opción más indicada para la implementación del algoritmo FK.

(a) Platafoma



(b) Señales

Figura 5.7: Sistema embebido con sensor ADIS16354.

Tabla 5.4: Tiempo de ejecución del algoritmo FK en un sistema embebido.

n=r	$O(n^3)$	$t(ms)$
2	212	1.3
3	611	3.74
4	1334	8.18
5	2477	15.18
6	4136	25.36
7	6407	39.28
8	9368	57.44
9	13169	80.75
10	17852	109.46

5.2. Validación de la arquitectura implementada

La etapa de validación experimental requirió de la construcción de un banco de pruebas que permitiera estimular al chip, ejecutar el algoritmo FK y recuperar los resultados de la estimación para su verificación y análisis. El esquema del banco de pruebas se muestra en la Fig. 5.8 y una descripción detallada se encuentra en el Apéndice B. La plataforma UDB5 contiene los sensores inerciales como acelerómetros y giroscopios que generan un conjunto de datos los cuales son enviados al sistema embebido que los acondiciona al formato numérico aceptado por el chip PEMFK (representación de magnitud y signo). El sistema embebido utilizado fue una tarjeta de evaluación de Microchip con un microcontrolador de 32 bits (PIC32 starter kit), este sistema actúa como un controlador maestro (MCU) cargando en el chip PEMFK los datos provenientes de los sensores inerciales. El chip PEMFK realiza la función de esclavo. Enseguida el controlador maestro genera la señal *START* que indica al esclavo que comience la ejecución del algoritmo. Cuando el chip PEMFK termina la ejecución de una iteración, éste emite una señal en la terminal de *READY*, la cual da aviso para que los resultados de la estimación pueden ser recuperados por el controlador maestro. Estos resultados se transfieren por un bus serial a una computadora, quien recibe los datos para ser analizados y/o utilizados con propósitos de medición y control.

5.2.1. Integración del MCU con el CI PEMFK

Con base en el diseño del sistema MCU PIC32 se implementó el código para la comunicación entre el MCU y el CI PEMFK. Las rutinas implementadas realizan las siguientes tareas:

- Escritura y lectura de los buses y señales del PEMFK en los puertos del MCU.
- Algoritmo de carga y lectura de una dirección del banco de registros del PEMFK.
- Algoritmo de inicialización del banco de registros completo del PEMFK.
- Verificación de los valores de inicialización del banco de registros del PEMFK.
- Algoritmo para carga de datos de sensores en los registros del PEMFK correspondientes.
- Algoritmo de adquisición de la estimación realizada por el PEMFK.
- Algoritmo integral de operación del PEMFK por muestra; incluye carga de la muestra, filtrado, recuperación de la estimación y envío de resultado por comunicación serial.

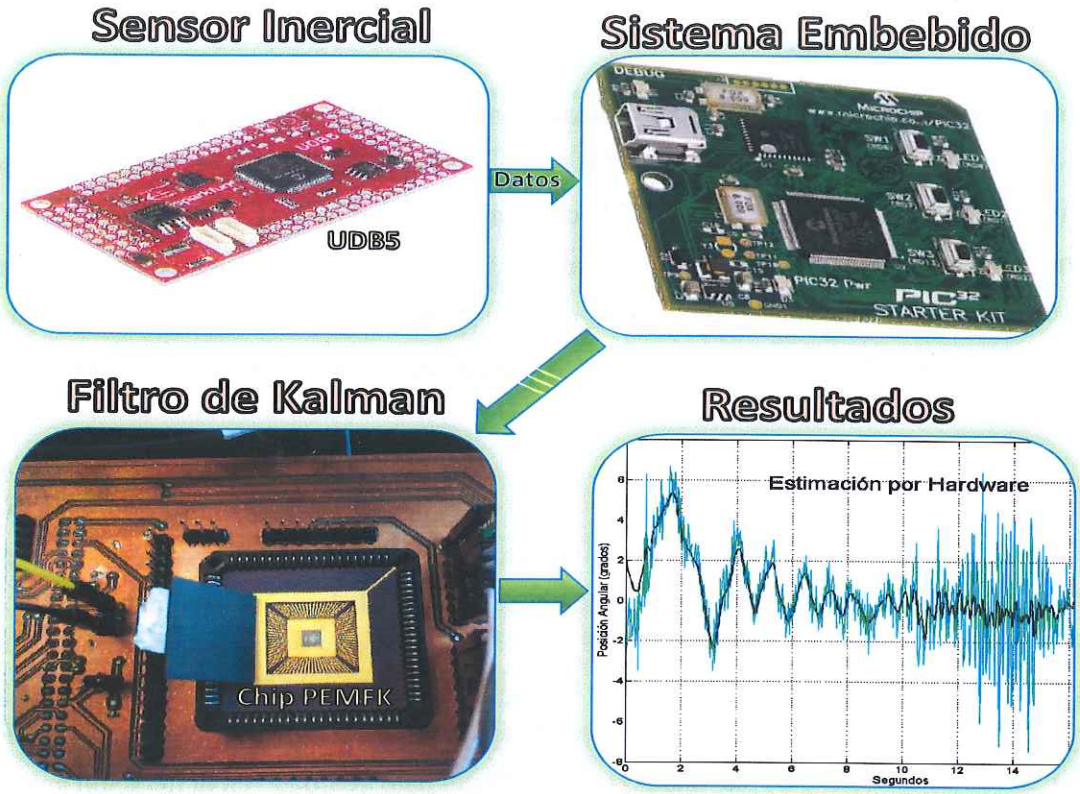


Figura 5.8: Esquema del banco de pruebas para validación del chip PEMFK.

Filtro de Kalman embebido en el PIC32

Se implementó el FK en el controlador maestro PIC32, con el fin de realizar la comparación contra la respuesta del CI PEMFK. Como resultado de la implementación, el FK fue ejecutado en la unidad de procesamiento maestro PIC32 y requirió de 245 μs para concluir una iteración.

Pruebas realizadas al sistema

El sistema MCU PIC32 fue sometido a pruebas de generación de señales para asegurar la correcta interacción eléctrica entre el CI y la unidad de control maestro del sistema. Para dichas pruebas se utilizó el osciloscopio de señales mixtas MSO 4034 de la empresa Tektronix©.

Antes de comenzar con pruebas del CI, se comprobó que los puertos del MCU estuvieran correctamente configurados. La comprobación se realizó con el analizador de estados lógicos y se procedió a realizar las pruebas de funcionamiento del CI. Para éstas, se utilizó en un inicio una señal de reloj (*CLK*) de 10 kHz generada por el MCU, incrementándola posteriormente a 100 kHz.

Algoritmo de escritura en una dirección del banco de registros (*loadReg*)

En el Apéndice B se encuentran las rutinas principales utilizadas en la implementación del banco de pruebas para la validación del PEMFK. La prueba consistió en cargar iterativamente una dirección del banco de registros del PEMFK con cinco datos diferentes entre sí únicamente en la parte menos significativa. Posteriormente, se realizó la lectura del registro para comprobar que los datos se almacenaron correctamente, 2 ciclos de reloj la parte alta y 2 ciclos de reloj la parte baja. El algoritmo que realizó la rutina *loadReg* es visualizado en la Fig. 5.9 y consiste en los siguientes pasos:

a) Se carga en el registro temporal de la parte alta (*RH*) los bits más significativos (*MSB*) [23:12] durante 3 ciclos de *CLK*. Esta Secc. del algoritmo es una escritura temporal, por lo que en el bus de salida *DOUT* se visualizan los *MSB* del dato guardado en la iteración anterior.

b) Se carga en el registro temporal de la parte baja (*RL*) los bits menos significativos (*LSB*) [11:0] durante 3 ciclos de *CLK*. Esta fase del algoritmo es una escritura temporal, por lo que en el bus de salida *DOUT* se visualizan los *LSB* del dato guardado en la iteración anterior.

c) La dirección del registro a cargar se manda en el bus *EDIR* y se habilita la señal *WRITE_B* en alto durante un ciclo de *CLK*.

d) Se mantiene la señal *WRITE_B* en bajo durante un ciclo de *CLK*.

Los datos de la Tabla 5.5 se utilizaron en la escritura de registros con la finalidad de verificar que estuvieran almacenando la información cargada a los mismos. Al analizar los resultados obtenidos en las señales recuperadas del circuito integrado, se concluyó que con esta información el PEMFK estaba respondiendo correctamente a los impulsos eléctricos aplicados. Adicionalmente, utilizando un osciloscopio, se determinó el periodo de tiempo necesario para ejecutar dicho procedimiento a distintas frecuencias de reloj. La Tabla 5.6 contiene los resultados obtenidos en dichas mediciones.

Algoritmo de inicialización del banco de registros

La prueba consistió en inicializar cada localidad del banco de registros del PEMFK utilizando la rutina *loadReg*. Posteriormente, con el objetivo de comprobar que los datos estuvieran correctamente almacenados en el circuito integrado, se utilizó la rutina *readReg*, para obtener iterativamente cada una de las localidades del banco de registros. La información obtenida del chip se envió a

Tabla 5.5: Datos utilizados en la escritura de registros.

Dato	*MSB* [23:12]	*LSB* [11:0]
1	0xAAA	0x111
2	0xBBB	0x222
3	0xCCC	0x333
4	0xDDD	0x444
5	0xDDD	0x555

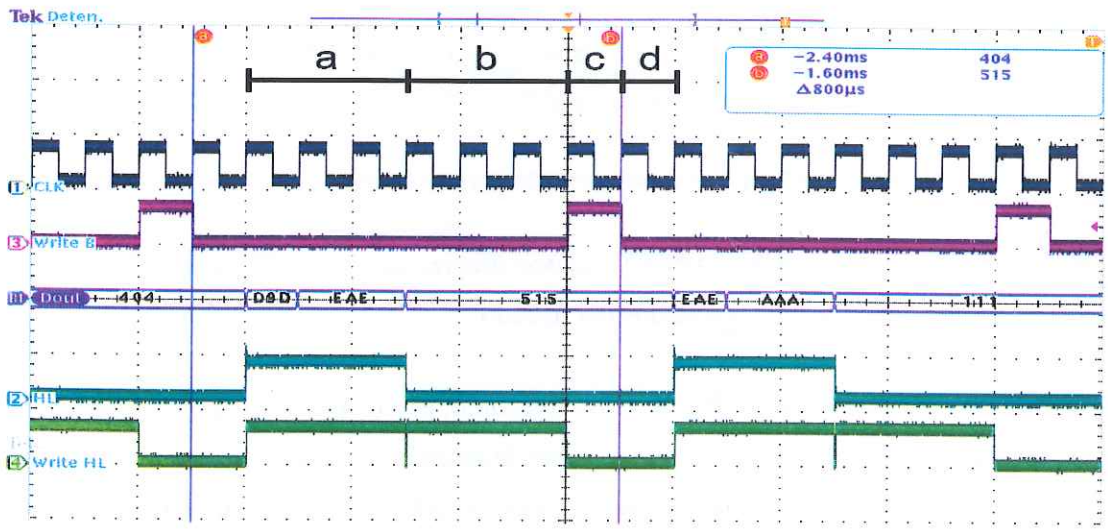


Figura 5.9: Resultados del algoritmo de escritura *loadreg* con $F_{clk} = 100KHz$.

Tabla 5.6: Tiempo de escritura de un registro a distintas frecuencias de reloj.

F_{clk}	Tiempo de escritura
$10\,KHz$	$800\mu s$
$100\,KHz$	$80\mu s$
$200\,KHz$	$40\mu s$

través del módulo UART1 (RS-232) utilizando la rutina *sendPacket*. El cable TTL-232R fue la opción utilizada para conectar el sistema a la terminal serial. Durante la prueba se utilizó una frecuencia de reloj del circuito integrado de 10 kHz.

Verificación del secuenciador

El secuenciador con el que cuenta el PEMFK se encarga de ejecutar cada instrucción del algoritmo del FK. Para iniciar una iteración del proceso es necesario que la terminal de *READY* esté en alto para aplicar la señal de *START*. La prueba consistió en utilizar rutinas de comunicación para aplicar las señales necesarias y activar el secuenciador de instrucciones del PEMFK; verificando de esta manera el comportamiento de las señales analizadas. La señal de reloj utilizada en la primera ejecución fue de 10 kHz. El algoritmo ejecutado en esta prueba es visualizado en las Figs. 5.10a y 5.10b consiste en los siguientes pasos:

a) Se aplica una señal de *RESET* en alto durante un ciclo, esperando uno adicional para asegurar que el CI esté en un estado listo para operar.

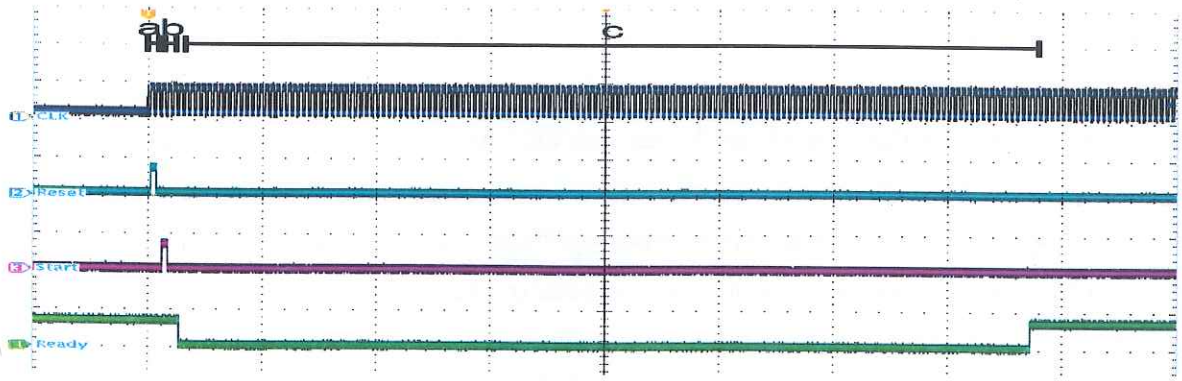
b) Se aplica una señal de *START* durante un ciclo para iniciar el secuenciador.

c) El PEMFK realiza las instrucciones y cuando finaliza activa en alto la terminal de *READY* permitiendo el acceso a los resultados del filtrado.

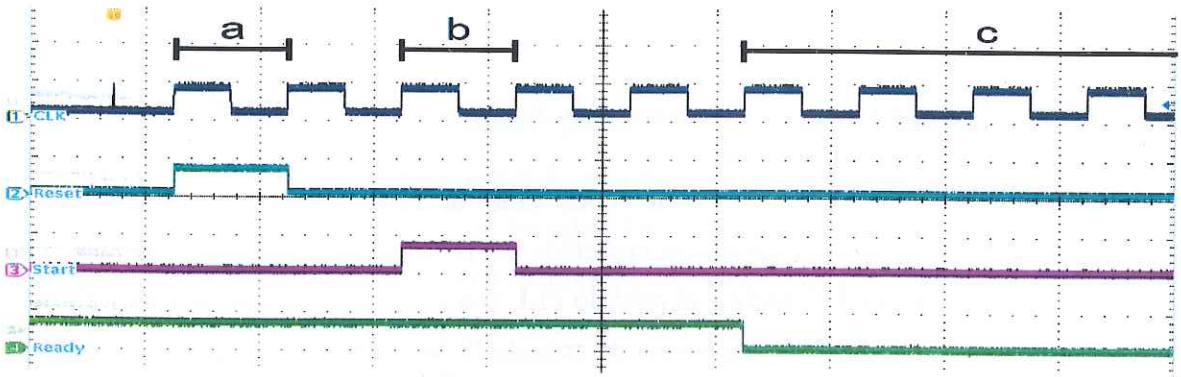
A partir de los resultados obtenidos en el osciloscopio, es posible afirmar que el secuenciador funciona correctamente. Previo a ejecutar la prueba, la terminal de *READY* ya se encontraba en un nivel lógico alto, sin embargo la aplicación del pulso de *RESET* asegura que el CI se encuentre listo para operar. Es importante mencionar que el camino de la señal *START* cuenta internamente con un circuito de sincronización. Debido a esto la instrucción de inicio es realmente reconocida 2 ciclos después de haberla activado externamente. Utilizando el mismo algoritmo de pruebas, se realizaron distintas iteraciones del mismo a distintas frecuencias de reloj. Esto con el fin de determinar el periodo de tiempo que le toma al PEMFK ejecutar el filtro. Asimismo, se puede comprobar la linealidad en tiempo del CI. La Tabla 5.7 contiene los resultados obtenidos de dichas pruebas.

5.2.2. Sistema UDB5

El sistema UDB5 (Fig. 4.1) consta de una tarjeta electrónica que contiene un microcontrolador de la familia dsPIC33 de Microchip y una unidad de medición inercial tipo MEMS modelo MPU-6000 con 3 giroscopios y 3 acelerómetros. El microcontrolador dsPIC33 es considerado la unidad de control secundario (SCU por sus siglas en inglés) del banco de pruebas.



(a) Vista completa del análisis del inicio del secuenciador.



(b) Vista parcial del análisis del inicio del secuenciador.

Figura 5.10: Resultados de la verificación del secuenciador.

Tabla 5.7: Relación de frecuencia y periodo de filtrado a distintas frecuencias de reloj.

F_{clk}	Ciclos de reloj	Periodo de filtrado
$10\,KHz$	150	$15\,ms$
$50\,KHz$	150	$3\,ms$
$100\,KHz$	150	$1{,}5\,ms$
$200\,KHz$	150	$0{,}75\,ms$

El sistema UDB5 integra el MPU-6000 con el microcontrolador dsPIC33 por medio del protocolo SPI. El SCU cuenta igualmente con un módulo UART1 configurado para comunicación serial utilizando el protocolo RS-232 con un *baud rate* de 230,400 bps. Este módulo es dedicado para la comunicación de los datos de los sensores inerciales entre el sistema UDB5 y el sistema MCU PIC32, sin embargo también puede ser utilizado para transferir información directamente a una computadora en caso de ser necesario.

Es importante tener en cuenta que el PEMFK utiliza una arquitectura de representación binaria decimal de punto fijo de 24 bits con magnitud y signo, es decir, el bit más significativo del vector representa el signo del dato, siendo un bit en alto una magnitud negativa y un bit en bajo una magnitud positiva. Los siguientes 9 bits más significativos después del signo, representan la parte entera de la magnitud, mientras los 14 bits menos significativos representan la parte fraccional de la misma. El SCU del sistema UDB5 es utilizado para realizar la conversión de representación binaria que debe ser utilizada en el banco de pruebas. Con base en lo anterior, se desarrolló el código para efectuar la conversión de un dato de representación flotante, a la representación binaria que requiere el PEMFK y viceversa.

Configuración de la unidad de control secundario (SCU)

La unidad de control secundario es un microcontrolador de 16 bits dsPIC33FJ256GP710A de la empresa Microchip, el cual se encuentra embebido en la tarjeta UDB5, es configurado para utilizar un oscilador externo de 8 MHz y el módulo PLL con el que cuenta el dispositivo. El PLL es configurado para incrementar la frecuencia de entrada (8 MHz) en un factor de ocho. Por lo tanto, la frecuencia de operación del microcontrolador es de 64 MHz. De acuerdo con la hoja de especificaciones, éste tiene un valor de 32 MIPS (Millones de Instrucciones Por Segundo).

Las rutinas necesarias para la inicialización del SCU realizan las siguientes tareas:

- Configuración del oscilador @ 64 MHz.
- Configuración de las entradas y salidas del microcontrolador.
- Configuración del módulo UART1 @ 230400 bps.

Sensor inercial MPU-6000

El sensor inercial MPU-6000 es un dispositivo que contiene un acelerómetro y un giroscopio, de tecnología MEMS, cada uno con 3 ejes de información. Se cuenta con un convertidor analógico-digital (ADC) de 16 bits de resolución para cada eje de medición, es decir 6 convertidores en total. El sensor inercial le permite al usuario configurar la escala de medición tanto del acelerómetro, como del giroscopio, de acuerdo a la dinámica que se requiere analizar. La Tabla 5.8 muestra las diferentes configuraciones de escala de medición disponibles en el sensor inercial.

Tabla 5.8: Configuraciones del rango de medición del sensor inercial MPU-6000.

Rango giroscopio (± °/seg)	Rango acelerómetro (± g)
200	2
500	4
1000	8
2000	16

El sensor MPU-6000 cuenta también con un módulo I2C (@ 400 kHz typ.) y otro SPI (@ 1 MHz typ.) para la transmisión de los parámetros. Es posible configurar el sensor para generar señales de interrupción periódicas a un periférico, así este último solicita al sensor los valores requeridos al ejecutarse la rutina de interrupción. Los valores de los sensores se encuentran almacenados en un banco de registros con palabras de 8 bits. Se recomienda que la frecuencia de muestreo de datos no exceda los 200 muestras por segundo.

Integración del SCU y el sensor inercial MPU-6000

Los puertos de comunicación SPI del sensor inercial embebido se encuentran conectados al módulo SPI2 del microcontrolador dsPIC33. Por medio de este protocolo de comunicación, es como el SCU configura y adquiere datos del MPU-6000. El módulo UART1 es utilizado para transferir información a otro dispositivo externo a la tarjeta.

Tomando como referencia el proyecto MatrixPilot desarrollado por Bill Premerlaniet et al. (para mayor información consultar https://github.com/MatrixPilot/MatrixPilot/wiki), se implementó el código que con las rutinas necesarias para la configuración del módulo SPI2 (@ 667 kHz). Asimismo, dicho código permite la solicitud de información por parte del maestro al esclavo mediante una rutina de interrupción. Éste último utiliza las rutinas del primero para inicializar y configurar el sensor inercial. Esta configuración incluye la señal periódica que le indica al SCU, por interrupción, que la lectura de los registros del acelerómetro y giroscopio debe realizarse. De igual manera, fueron implementadas las rutinas de adquisición y procesamiento de datos del sensor inercial para recuperar los datos de velocidad angular, aceleración y temperatura, sin embargo éstos son procesados de distinta manera. El código implementado contiene una función llamada

KalmanFilter, que recibe de los sensores la aceleración lineal, la velocidad angular y estima la posición angular. El filtro de Kalman por software es ejecutado en la unidad de procesamiento secundario en aproximadamente 737 μs.

Transmisión de valores flotantes y valores binarios escalados.

La prueba consistió en ejecutar en la tarjeta UDB5 el proceso de inicialización del SCU, configurando los módulos de comunicación UART1 y SPI2. El cable TTL-232R fue la opción utilizada para conectar el sistema a la terminal serial. Asimismo, el sensor inercial se configuró para generar un muestreo de datos cada 20 ms, es decir a una frecuencia de 50 Hz.

Para la transmisión de valores flotantes primero se recuperan los datos del sensor inercial, se escalan de acuerdo a los parámetros definidos por el fabricante para manejar las magnitudes reales, posteriormente se ejecuta por software el algoritmo de Kalman, y el resultado de la estimación se transmite por UART con formato flotante de 3 decimales de precisión.

Para la transmisión de valores enteros escalados primero se recuperan datos del sensor inercial, se escalan, se determina la posición angular realizando la operación arco tangente entre las aceleraciones, luego se convierten los resultados a una representación de magnitud y signo de 24 bits con punto fijo; y finalmente transmitir un vector de 48 bits de información como una cadena de 12 caracteres hexadecimales. La Tabla 5.9 contiene los resultados de temporización de los procesos más relevantes en la operación de la tarjeta UDB5.

Tabla 5.9: Temporización de procesos ejecutados en el sistema UDB5.

Proceso ejecutado	Tiempo de ejecución
Adquisición de 7 palabras de 16 bits, utilizando comunicación SPI.	229 μs
Una iteración del FK por software en el dsPIC33 @ 32 MIPS.	737 μ
Dos iteración del FK por software en el dsPIC33 @ 32 MIPS.	1,49 ms
Transmisión de 4 datos flotantes con 3 decimales, en el protocolo RS-232.	10,9 ms
Escalamiento a decimal, operación arco tangente entre las aceleraciones y conversión a representación binaria.	5,78 ms
Transmisión de cadena de 12 caracteres hexadecimales, con protocolo RS-232.	573 μs

5.2.3. Filtrado iterativo del circuito integrado PEMFK

El banco de pruebas implementado para la validación del circuito PEMFK, consiste de su integración con los sistemas MCU PIC32 y UDB5. El primero tiene la tarea de controlar el proceso de pruebas en su totalidad; adquiriendo datos de posición y velocidad angular del segundo, inyectarlos al CI, recuperar el resultado del algoritmo del chip y enviarlo a una terminal serial. Este proceso se realiza periódicamente con información del sensor inercial en tiempo real. Por otro lado, el mismo banco de pruebas permite probar el circuito integrado en modo fuera de línea (no tiempo real).

Operación del circuito integrado fuera de línea (offline)

Este modo de operación consiste en la declaración constante por código, de una matriz de información de sensores previamente recuperada y procesada. De esta manera, el sistema MCU PIC32 utiliza los datos de esta matriz para inyectarlos al circuito integrado y realizar el algoritmo de filtrado. Los resultados del filtro son transferidos a una terminal serial.

Operación del circuito integrado en tiempo real (online)

El modo de operación online del banco de pruebas, realiza el algoritmo FK en tiempo real. Primero se recuperan los datos del sistema UDB5 en determinados intervalos de tiempo. Cada una de las muestras es procesada tanto por el sistema MCU PIC32 como por el circuito integrado PEMFK. Finalmente los resultados de ambos sistemas son transmitidos por via serial con el protocolo RS-232 a una terminal para su procesamiento y análisis posterior. La trama de datos recuperada de 12 caracteres hexadecimales contiene la muestra original en formato flotante, proveniente del sensor inercial, el resultado en formato flotante de la muestra filtrada por software, y el resultado recuperado del PEMFK. Esta información es recuperada del banco de pruebas con el objetivo de analizar la eficiencia del CI con respecto a otras plataformas ejecutando el mismo algoritmo.

La Tabla 5.10 contiene los resultados de temporización de los procesos más relevantes en la operación de la tarjeta UDB5.

Tabla 5.10: Temporización de procesos ejecutados en el banco de pruebas.

Proceso ejecutado	Tiempo ejecución
Cambio de representación de caracteres hexadecimales (ASCII), a dos variables enteras long sin signo: posición y velocidad angular.	37 μs
Conversión de representación entera de 24 bits de magnitud y signo con punto fijo, a dos variables flotantes:	43 μ
Transmisión de dos variables flotantes con 3 decimales de precisión, utilizando el protocolo de comunicación RS-232	850 μs
Iteración de una implementación del FK por software en el PIC32 @ 80 MHz.	245 μs
Acondicionamiento y escritura de muestra del sensor inercial en el circuito integrado PEMFK @ 100 kHz.	240 μs
Iteración del FK en el PEMFK @ 100 kHz.	1,5 ms
Iteración del FK en el PEMFK @ 200 kHz.	750 μs
Iteración del FK en el PEMFK @ 1 MHz.	150 μs
Iteración del FK en el PEMFK @ 10 MHz.	15 μs
Iteración del FK en el PEMFK @ 15 MHz.	10 μs
Iteración del FK en el PEMFK @ 20 MHz.	7,5 μs
Iteración del FK en el PEMFK @ 30 MHz.	5 μs

5.2.4. Experimentación y pruebas.

Para verificar la respuesta del chip bajo condiciones controladas, se tomaron muestras de datos por diferentes periodos de tiempo a una tasa de muestreo de 20 ms. Al mismo tiempo, se implementó por software la solución del algoritmo sobre el controlador maestro (PIC32) con la finalidad de comparar los resultados contra el chip PEMFK. Posteriormente se realizó el post-procesamiento de la información muestreada para hacer un análisis comparativo de la respuesta del PIC32 contra la respuesta del chip PEMFK. La Tabla 5.11 muestra el conjunto de valores utilizados en la experimentación de una serie de pruebas variando los parámetros de sintonía del filtro (covariazas de ruido de proceso (**Q**) y covarianzas de ruido de medición (**R**). La frecuencia de reloj aplicada al chip PEMFK fue de 100KHz y los resultados obtenidos se muestran en las gráficas de la Fig. 5.12. Otro parámetro que se manipuló durante la experimentación fue la dinámica del movimiento sobre los sensores.

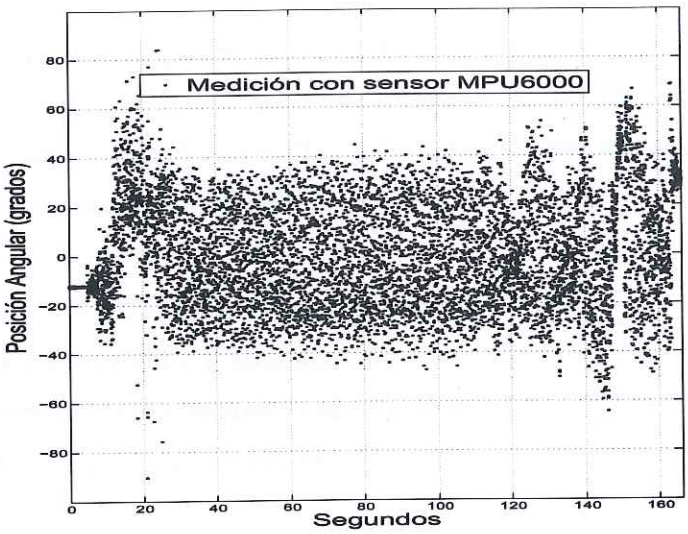
Para la experimentación con los parámetros de sintonía (R,Q) se instaló la tarjeta UDB5 sobre una plataforma experimental con un helicóptero no tripulado [64] (Fig. 5.11a). El ruido inducido a los sensores se ve reflejado en las gráficas de la Fig. 5.11b como una nube de puntos muy densa. Si estos valores se usaran en el control de un sistema causaría severos problemas en su control. Es por ello la necesidad del uso del FK que toma en consideración el ruido del sistema y de las mediciones.

Para mejorar la respuesta en la estimación y obtener un ajuste óptimo en los parámetros de sintonía, se modificaron los valores de R y los resultados de observan en la Fig. 5.12. La respuesta del PIC32 y el PEMFK son tan cercandas a la referencia que no se distinguen, en la Fig. 5.12d se muestra un acercamiento de la respuesta para apreciar la magnitud del defasamiento en un instante en particular.

Tabla 5.11: Parámetros utilizados para validación del chip PEMFK.

Frecuencia PEMFK 100KHz		Consumo de corriente promedio	R	Q	
Aplicación manual	Estático	336	0.6	0.02247	0
				0	0.03
	Dinámica suave	339	0.6	0.02247	0
				0	0.03
	Dinámica rápida	340	0.6	0.02247	0
				0	0.03
Aplicación helicóptero	Estático	335	4	0.02247	0
				0	0.03
	Variación de R	352	4	0.02247	0
				0	0.03
		339	7	0.02247	0
				0	0.03
		345	10	0.02247	0
				0	0.03
		344	20	0.02247	0
				0	0.03
	Variación de Q	339	10	0.0002247	0
				0	0.0003
		335	10	0.002247	0
				0	0.003
		345	10	0.02247	0
				0	0.03
		339	10	0.2247	0
				0	0.3

(a) Plataforma experimental con helicóptero.



(b) Medición de posición angular con motor encendido.

Figura 5.11: Mediciones con sensores instalados en una plataforma con helicóptero.

Cálculo de ESR (ERROR TO SIGNAL RATIO)

El cálculo de la razón de error a señal (ESR) se realizó con el propósito de caracterizar el margen de error y observar su comportamiento con los diferentes parámetros de sintonía. El análisis del ESR se obtiene mediante las ecuaciones (5.1) - (5.4).

$$error = Sref - pemfk \tag{5.1}$$

$$ENEref = sum(Sref.^2) \tag{5.2}$$

$$ENEerr = sum(error.^2) \tag{5.3}$$

$$ESR = 10 * log10(ENEerr/ENEref)(dB) \tag{5.4}$$

Donde:

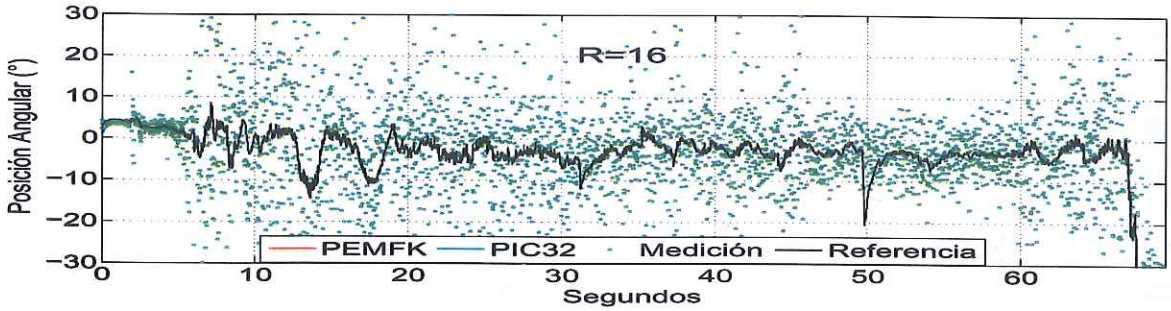
Sref se define como el vector de referencia (Simulación de Matlab).

ENEref es el valor de energía de la señal de referencia (Matlab).

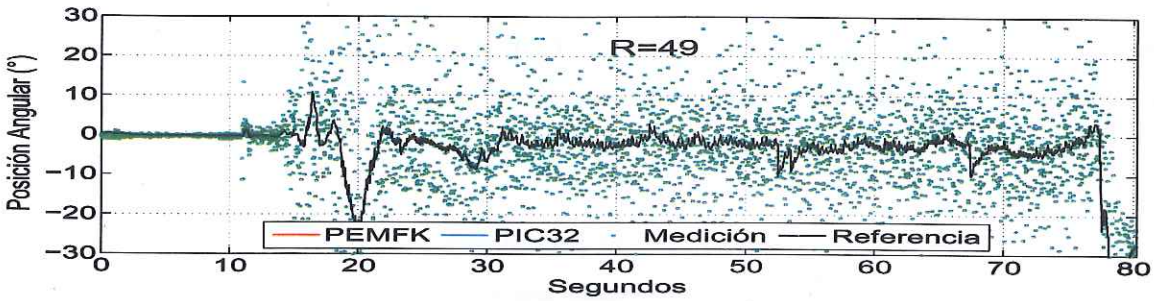
pemfk es el vector de estimación del PEMFK.

ENEerr es la energía de la señal de error del PEMFK.

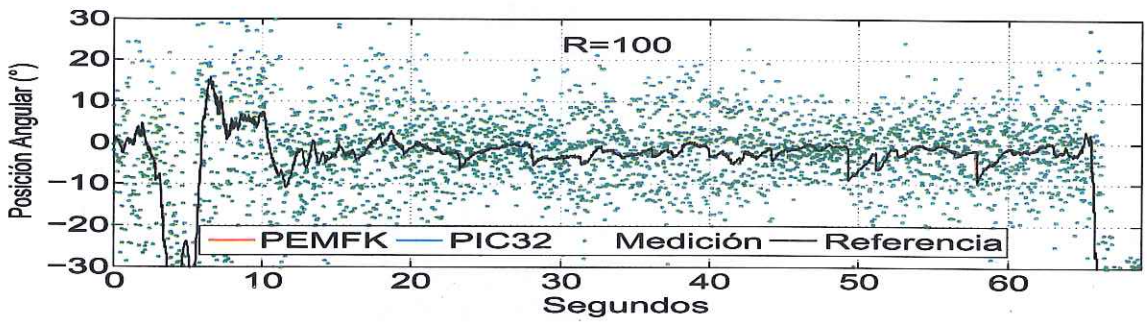
Como parte de la experimentación se hizo el análisis de ESR para las diferentes condiciones tanto de la dinámica del sistema como de los parámetros de sintonía descritas en la Secc. 5.2.4



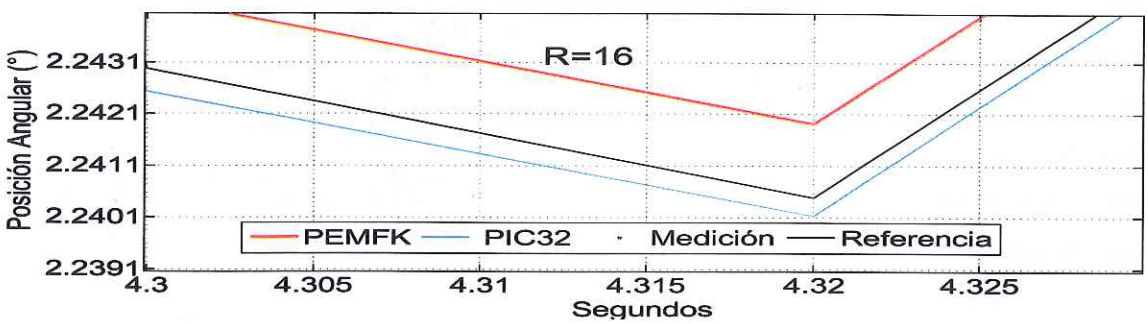
(a) R=16



(b) R=49



(c) R=100



(d) Acercamiento a la estimación.

Figura 5.12: Estimación con chip PEMFK vs. PIC32.

(experimentos a,b,...,m). En la Tabla 5.12 se observa que el valor más alto en decibelios se obtuvo en el experimento *a*, cuando el sistema está estático, y no hay mucha influencia de ruido del proceso (inducido por el efecto del motor del helicóptero).

En el experimento *a* la respuesta del chip PEMFK fue mejor comparada con la respuesta entregada por el sistema embebido PIC32. También se puede apreciar que en la mayoría de los casos los valores son superiores a los 40 *dB*. lo cual se considera un bajo margen de error. En los experimentos j, k, l, m y n se aprecia una mejor respuesta en el PIC32, esto se atribuye a que los parámetros de sintonía tienen valores muy pequeños, lo cual influye en que la arquitectura del PEMFK es de punto fijo comparada contra la arquitectura de punto flotante en el PIC32, sin embargo los resultados del PEMFK se consideran bastante aceptables y con un bajo margen de error.

Tabla 5.12: Cálculo del ESR para diferentes experimentos.

Experimento	σ	PEMFK (*dB*)	σ	PIC32 (*dB*)
a	0.00018	-85.62	0.00288	-61.62
b	0.00204	-78.61	0.01556	-60.95
d	0.02668	-56.13	0.25356	-36.57
f	0.00566	-64.28	0.00627	-63.39
g	0.00331	-63.62	0.00287	-64.85
h	0.00849	-59.27	0.00208	-71.51
i	0.01151	-49.28	0.00110	-69.72
j	0.00644	-54.99	0.00027	-82.69
k	0.04209	-41.44	0.00017	-89.51
l	0.00802	-56.74	0.00060	-79.20
i	0.01151	-49.28	0.00110	-69.72
m	0.00330	-60.09	0.00453	-57.36

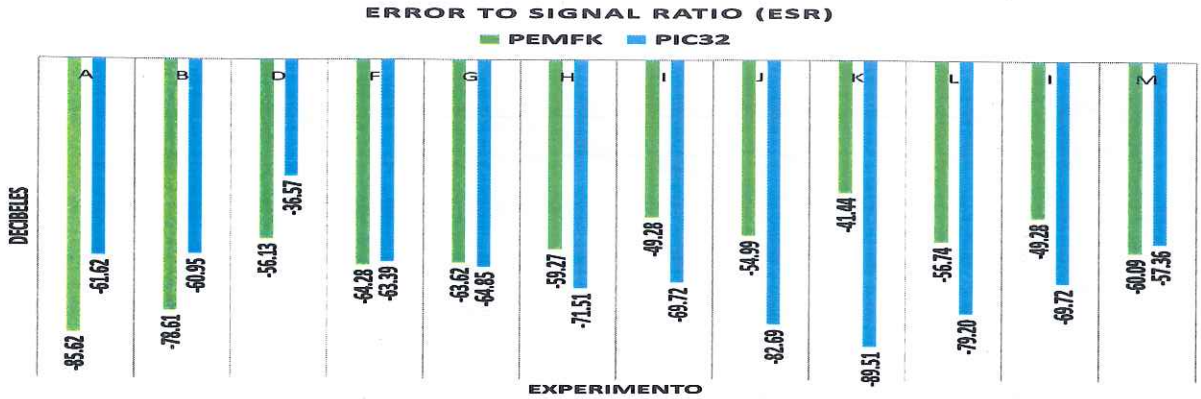
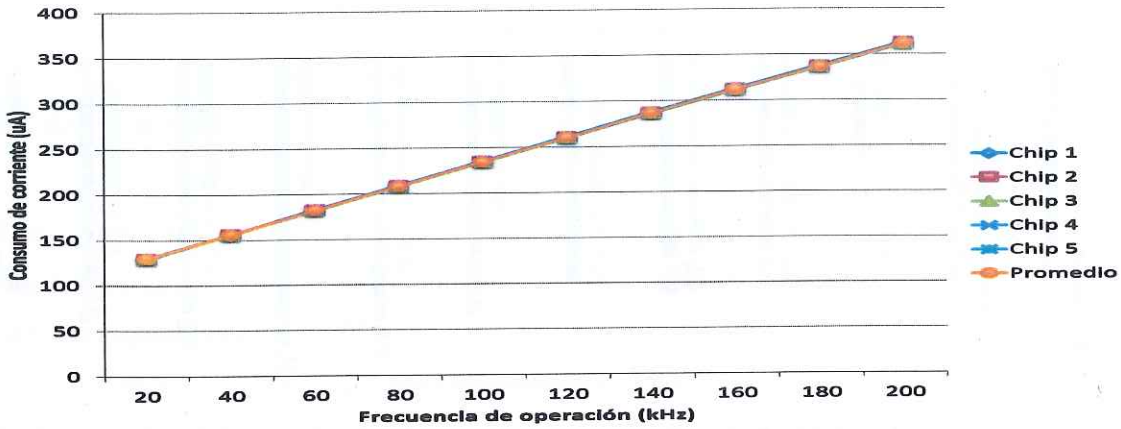


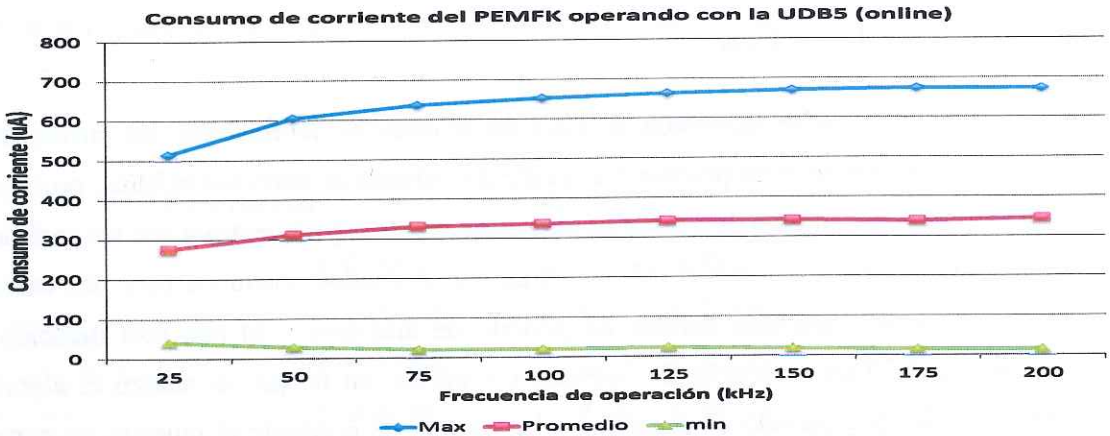
Figura 5.13: Estimación de la Razón de Error a Señal (ESR).

Pruebas de consumo de corriente

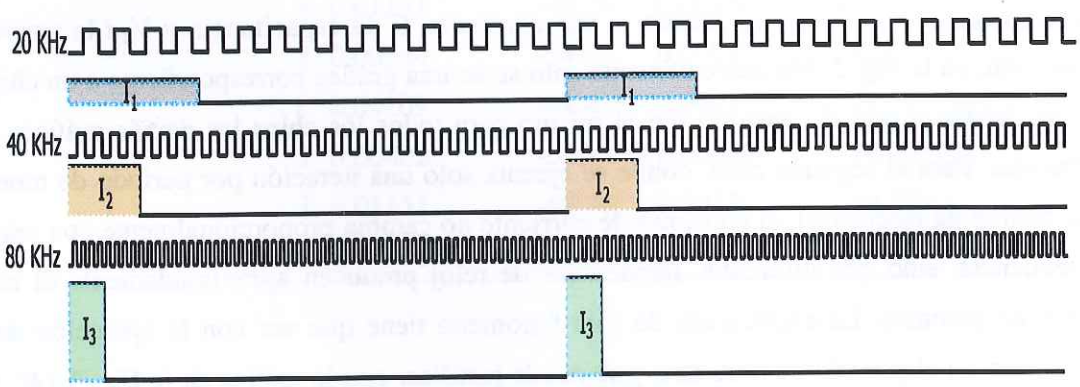
Un total de cinco chips se recibieron después de la etapa de fabricación, los cuales fueron probados uno a uno. Una de estas pruebas fue medir el consumo de corriente mínima, promedio y máxima en periodos de medición de 5 minutos para cada uno de experimentos y con los parámetros de la Tabla 5.11. Las Figs. 5.14a y 5.14b muestran los resultados obtenidos para dos casos: a) operación continua del algoritmo durante un periodo de muestreo y b) una sola iteración por periodo de muestreo. Para la prueba de operación continua, en la que se realizó el algoritmo ininterrumpidamente por periodo de muestreo, el consumo de corriente si muestra un aumento lineal con respecto a la frecuencia de reloj, validando lo que la literatura establece, que el consumo de potencia dinámica (debida a la frecuencia del reloj-F_{ck}) es proporcional a la frecuencia [50], esto es $P_{dinamica} = \alpha C V_{dd} F_{ck}$, siendo α una constante, C la capacitancia y V_dd la tensión de alimentación; en la Fig. 5.14a aparentemente solo se ve una gráfica correspondiente a un chip, sin embargo, debido a que el consumo fue el mismo para todos los chips las demás gráficas están sobrepuestas. Para el segundo caso, donde se ejecuta solo una iteración por periodo de muestreo (forma común de ejecución), el consumo de corriente no cambia proporcionalmente con respecto a la frecuencia, sino que diferentes frecuencias de reloj producen aproximadamente el mismo consumo de potencia. La explicación de este fenómeno tiene que ver con la ejecución de una sola iteración en el periodo de muestreo y se puede justificar con la gráfica de la Fig. 5.14c, en la cual se representa el efecto sobre el consumo de corriente cuando varía la frecuencia aplicada, se observa que al aumentar la frecuencia también aumenta la corriente instantánea, pero el periodo de tiempo se reduce proporcionalmente, manteniendo similar el consumo para los diferentes valores de frecuencia.



(a) operación contínua



(b) 1 iteración por cada periodo de muestreo



(c) Consumo de corriente vs. frecuencia

Figura 5.14: Consumo de corriente.

5.2.5. Pruebas a frecuencia mayores a 1 MHz

Las pruebas a frecuencias por encima de 1 MHz se realizaron con la finalidad de medir los limites de respuesta del chip, antes de que comience a degradar su operación. La Tabla 5.13 resume los tiempos de ejecución del algoritmo bajo determinadas frecuencias. 20 MHz es la frecuencia donde el chip PEMFK responde sin degradar la estimación (Fig. 5.15f). Por arriba de esta frecuencia la respuesta del chip comienza a tener efectos de alejamiento sobre el valor simulado (ver Fig. 5.15h). Este problema en parte se debe al tiempo de retardo en cada uno de los bloques en la arquitectura, ya que puede darse el caso de no tener el tiempo requerido para resolver las operaciones con la precisión suficiente, provocando que el error aumente conforme se incrementa la frecuencia de operación. Otro efecto que ocurre es debido al diseño del banco de pruebas, ya que no se consideró la problemática de los efectos parásitos del PCB en altas frecuencias, haciendo que los limites de frecuencia estén dependiendo más de los efectos parásitos que de la arquitectura propia del PEMFK.

Tabla 5.13: Tiempo de ejecución del algoritmo en chip PEMFK.

frecuencia MHz	t_{pemfk} μs	Rendimiento actualizaciones/segundo
12	12.43	80,000
15	9.93	100,000
20	7.4	133,333
25	5.9	166,667
26	5.7	173,333
27	5.5	180,000
28	5.3	186,667
29	5.1	193,333
30	5.0	200,000

5.2.6. Comparativa contra una arquitectura de un procesador de 16 bits

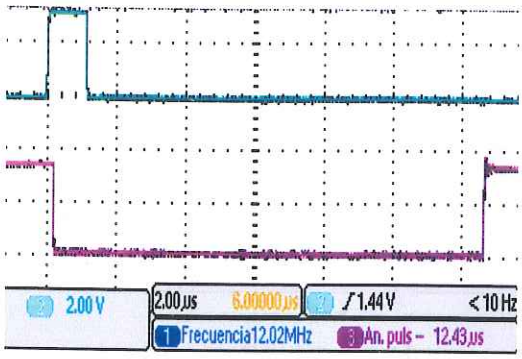
La UDB5 en la cual se encuentran los sensores inerciales, también incluye un procesador de 16 bits de la familia dsPIC33 de Microchip en el cual se implementó el algoritmo FK con la finalidad de hacer una comparativa con una arquitectura más parecida a la del PEMFK. Los resultados que se muestran en la Tabla comparativa 5.14 nos indican que el PEMFK, el cual implementa por hardware el algoritmo FK, ejecuta el algoritmo en el mismo tiempo que lo hace la implementación por software (dsPIC33) pero con las grandes diferencias en la frecuencia de reloj y energía consumida.

Otra comparativa se muestra en la gráfica de la Fig. 5.16, que representa el tiempo de ejecución del FK con el PEMFK dentro de un rango de frecuencias. Si consideramos que la frecuencia de la implementación por software opera a 64 MHz, el chip PEMFK resolvería el algoritmo en un periodo de 2,3 μs, inclusive a una frecuencia de 20 MHz el tiempo de 7,5μs está muy por debajo de la implementación por software, haciéndolo más eficiente que su contraparte el dsPIC33.

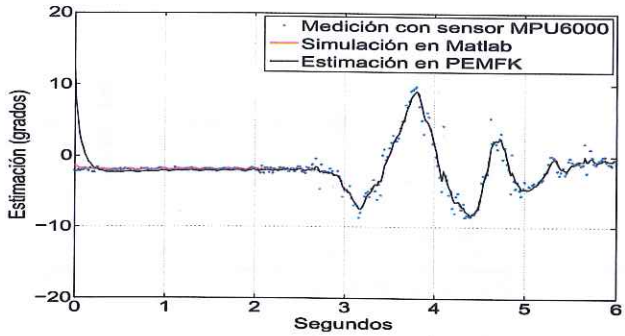
En conclusión, el PEMFK es capaz de operar a más altas frecuencias, como fue demostrado, a tan solo 200 kHz su tiempo de ejecución ya se encuentra por debajo de lo que tarda un sistema comercial, esta capacidad de utilización en bajas frecuencias lo hace muy robusto a problemas por interferencias electromagnéticas. Además los recursos de energía y área han sido optimizados con la finalidad de poder lograr implementaciones en aplicaciones móviles.

Tabla 5.14: Comparativa entre PEMFK y dsPIC33.

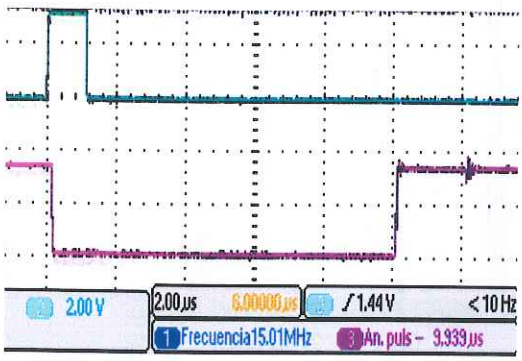
Parámetro	PEMFK	dsPIC33
Ciclos de reloj / iteración	150	47,000
Frecuencia de reloj	200 KHz	64 MHz
Corriente promedio	340 μA	56 mA
Potencia promedio	1,1 mW	280 mW
tiempo de ejecución	750μs	737 μs



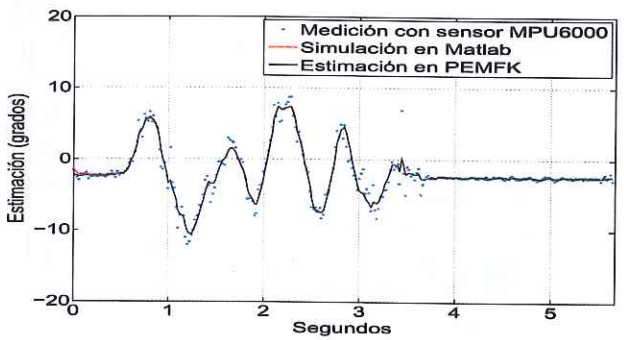
(a) $T_{pemfk} = 12{,}43\mu s$



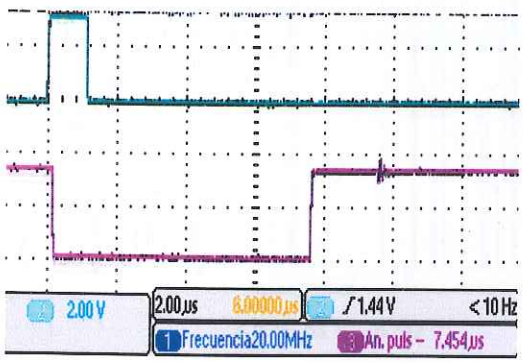
(b) $F_c = 12\,MHz$



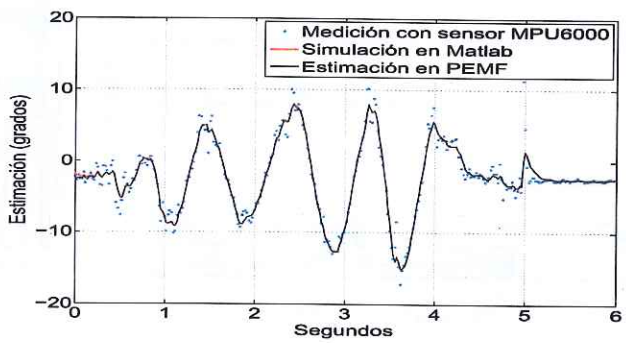
(c) $T_{pemfk} = 9{,}93\mu s$



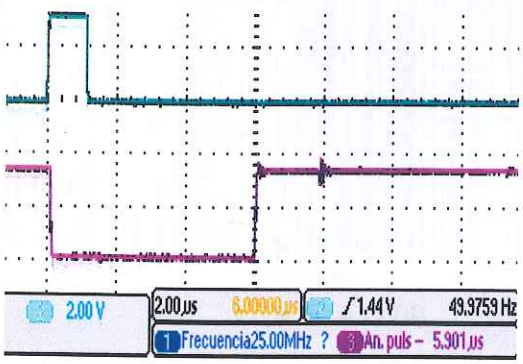
(d) $F_c = 15\,MHz$



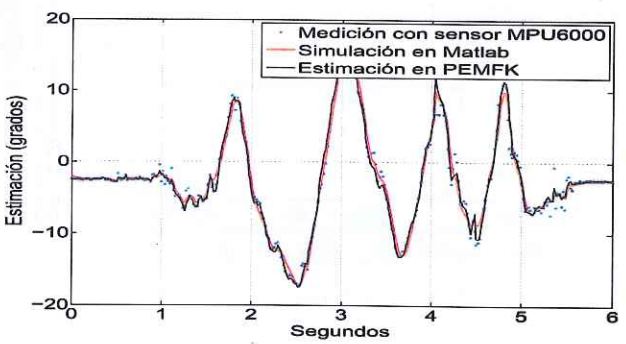
(e) $T_{pemfk} = 7{,}4\mu s$



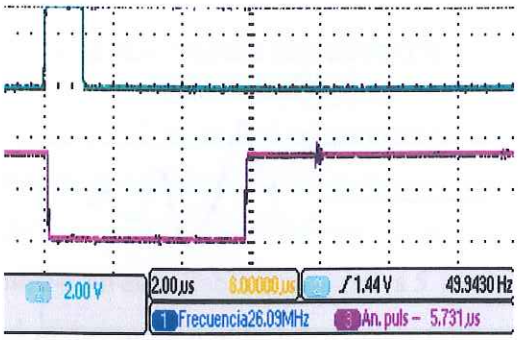
(f) $F_c = 20\,MHz$



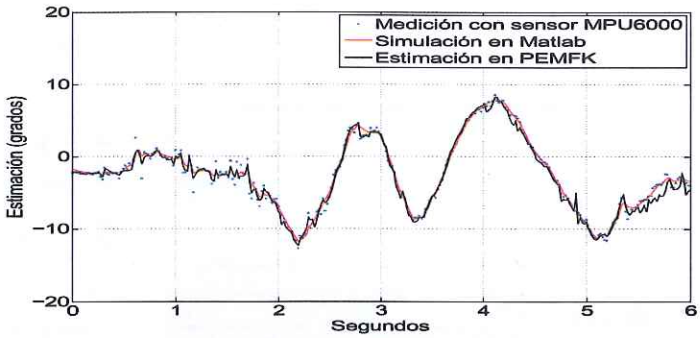
(g) $T_{pemfk} = 5{,}9\mu s$



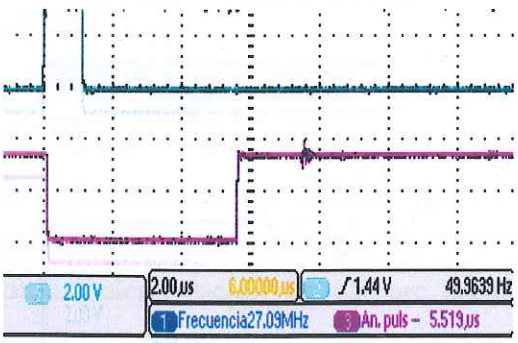
(h) $F_c = 25\,MHz$



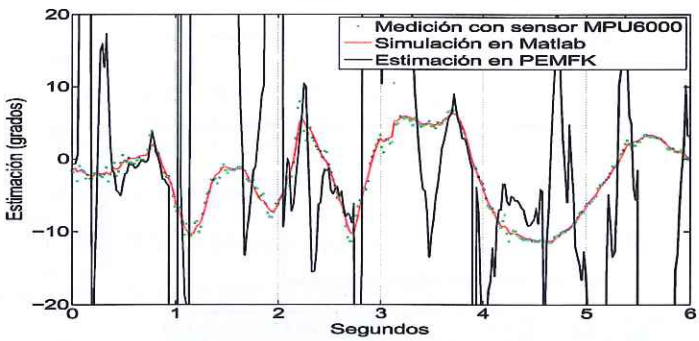
(a) $T_{pemfk} = 5{,}7\mu s$



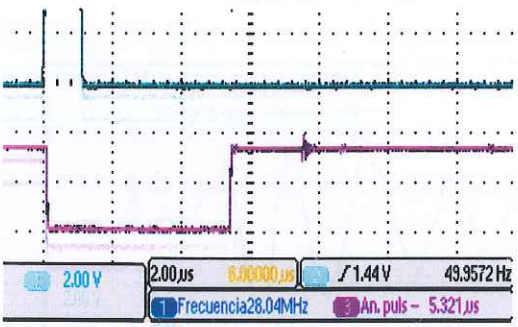
(b) $F_c = 26\,MHz$



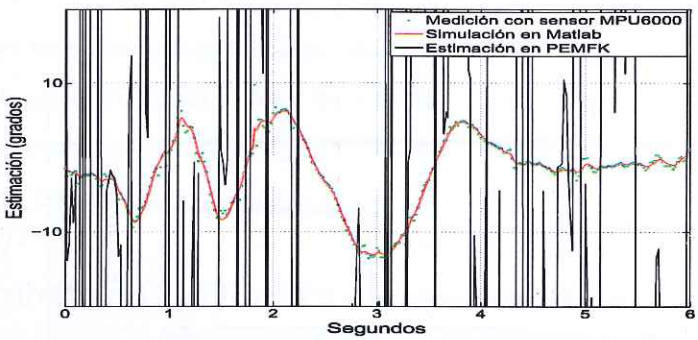
(c) $T_{pemfk} = 5{,}5\mu s$



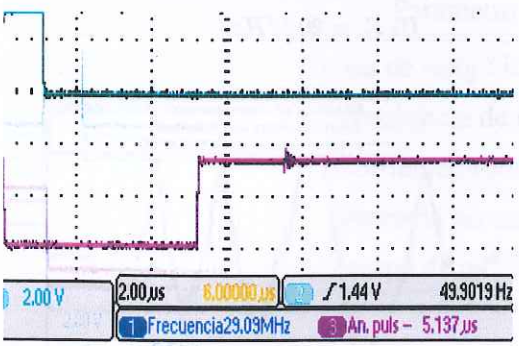
(d) $F_c = 27\,MHz$



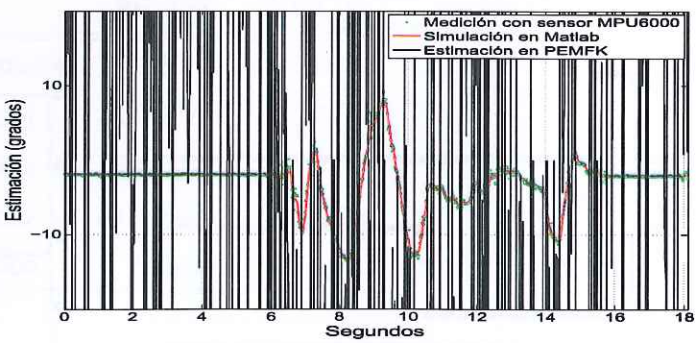
(e) $T_{pemfk} = 5{,}3\mu s$



(f) $F_c = 28\,MHz$



(g) $T_{pemfk} = 5{,}1\mu s$



(h) $F_c = 29\,MHz$

Figura 5.15: Tiempo de ejecución de una iteración y gráfica de estimación.

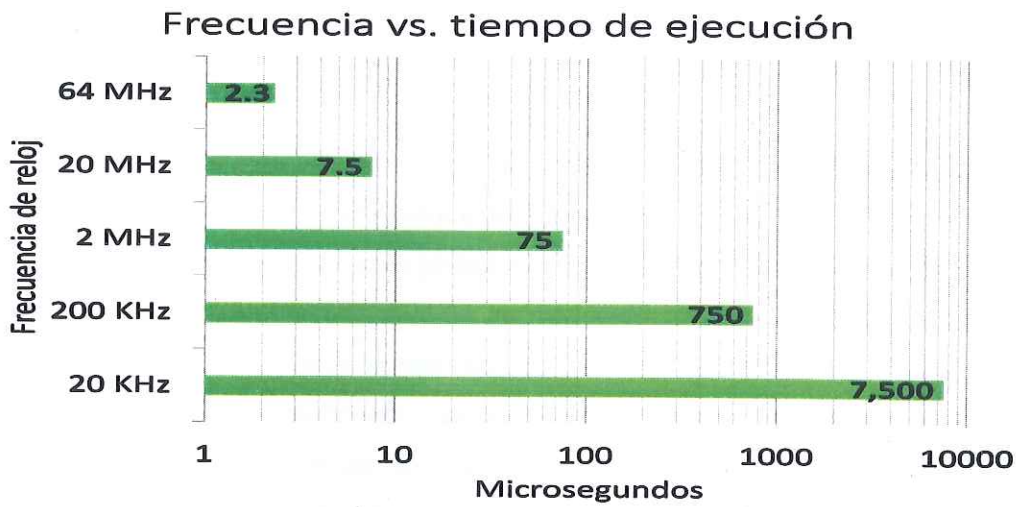


Figura 5.16: Tiempo de ejecución del FK en hardware.

Capítulo 6

Discusión, conclusiones y contribuciones.

6.1. Discusión

El FK es un estimador para una gran clase de problemas, utilizado en aplicaciones como la microcirugía, misiones espaciales y vehículos autónomos entre muchas otras. Su algoritmo presenta una complejidad computacional que crece polinómicamente en función de los estados a estimar. La tendencia de muchos autores en un principio fue lograr disminuir la complejidad computacional con diferentes formulaciones, como la de la raíz cuadrada, el filtro de información, factorización U-D y la descomposición Q-R principalmente. Las implementaciones en VLSI del FK se comenzaron a proponer en los años 80's, con el uso de técnicas de arreglos sistólicos, donde la base es el uso de múltiples procesadores, que si bien logran grandes desempeños, el uso de una gran cantidad de recursos computacionales siempre está presente. Otra tendencia han sido las implementaciones con sistemas embebidos ya sea mediante microcontroladores, procesadores digitales de señal (DSPs) y arreglos de compuertas lógicas programables (FPGA's). Sin embargo en la mayoría de los trabajos presentados no interesa la optimización de recursos computacionales ni el consumo de energía, peso y espacio, que son restricciones importantes en las aplicaciones de navegación inercial.

Para aplicaciones en tiempo real, se han preferido las implementaciones en computadoras personales con potentes microprocesadores. Algunos autores han desarrollado implementaciones sobre plataformas con FPGAs como García-Qunchía [65], integrando aplicaciones de navegación inercial con sistemas de posicionamiento global. Sakkay [42] implementó en VLSI la corrección de mediciones en espectrofotometría a través de una arquitectura sistólica que utiliza una área de 150 mm^2 y una frecuencia de 15 MHz, Youmin [38] llevó a cabo simulaciones de arquitecturas paralelas con

sistemas de computo paralelo. Otros autores han propuesto implementaciones VLSI del algoritmo FK como [43, 66, 67, 68]. Estos trabajos han sido propuestos para aplicaciones de alta velocidad tales como ecualización adaptiva de canales, ecualización en sistemas cdma, espectofotometría y reconstrucción de señales respectivamente. Estas implementaciones usan arquitecturas paralelas y arreglos sistólicos con el propósito de mejorar la velocidad de las actualizaciones en las iteraciones del FK. Sin embargo, nuestro objetivo no es maximizar la velocidad, sino mantener lo más bajo posible el área de silicio y el consumo de energía en aplicaciones de vehículos autónomos con velocidades por debajo de las 300 actualizaciones por segundo.

Por otro lado, la mayoría de los trabajos publicados sobre implementaciones de arquitecturas del FK en VLSI han sido presentadas solamente a nivel de ecuación o nivel de diagrama de bloques. De los trabajos encontrados solo dos aseguran haber realizado la síntesis VLSI y layout [43, 69]. Específicamente, [43] reporta un chip sintetizado (no fabricado) hecho con 750 k transistores que ejecutan un algoritmo FK de tres estados con una palabra de punto fijo en 20-bits usando un arreglo sistólico paralelo. A pesar de que esta implementación puede ser excelente para aplicaciones de alta velocidad, utilizando la arquitectura presentada en este trabajo se puede fabricar un chip que requiere sólo 140 K transistores, para una representación de datos con una ancho de 24-bits en punto fijo, para ejecutar un algoritmo FK de tres estados para velocidades de actualizaciones bajas requeridas en las aplicaciones de navegación inercial. La estimación de parámetros de navegación como posición y velocidad angular en sistemas inerciales rotacionales son importantes para los vehículos no tripulados como helicópteros y drones. Para estas aplicaciones, las velocidades máximas de procesamiento están de 50 a 300 actualizaciones por segundo, de acuerdo al ancho de banda de los sensores utilizados como acelerómetros y giroscopios.

Nuestra propuesta en esta Tesis Doctoral se enfocó en el diseño de una arquitectura VLSI optimizada en recursos computacionales y su implementación con un dispositivo ASIC para la solución del FK y poderse emplear en aplicaciones de navegación inercial con altas restricciones de consumo de energía, peso, volumen y manteniendo una alta eficiencia. Los resultados de la investigación presentada en la tesis demuestran que aún cuando la implementación en hardware del algoritmo FK utiliza la formulación convencional que presenta una complejidad de orden cúbico, el tiempo de cómputo de una iteración se puede lograr en el orden de los microsegundos lo que permite ser implementado en las aplicaciones de tiempo real para las cuales fue diseñado. Con propósitos de validar las características de la arquitectura se sintetizó el ASIC que se denominó PEMFK para resolver un FK de dos estados con una medición. En este caso particular fue fabricado finalmente con tecnología CMOS de 0.5 μm donde se llevaron a cabo las mediciones finales. Los resultados

obtenidos en la etapa de validación nos llevan a demostrar que es posible mantener un consumo de energía promedio del orden de 1.1 mW operando a 200 KHz comparado contra 280 mW en su implementación con un procesador dsPIC33 a 64 MHz como se mostró en la Secc. 5.2.6. Como se puede apreciar en esta comparativa, la operación a una baja frecuencia resultó un rendimiento similar al de un procesador digital en altas frecuencias. Sin embargo, nuestro chip PEMFK puede operar en frecuencias de hasta 20 MHz lo cual se traduce en un rendimiento de 133,333 actualizaciones por segundo. Estos resultados se pueden trasladar hacia una implementación de un FK de 10 estados estimados con 10 mediciones. En base a nuestra arquitectura optimizada, la cual requeriría de 110,101 ciclos de reloj para este caso, según el análisis presentado en la Secc. 2.5, siendo el tiempo de cómputo estimado de 5.5 ms, el cual es factible de implementar para una aplicación de navegación inercial.

Finalmente, dentro de los logros obtenidos están la adquisición de un expertise en el área de diseño de circuitos integrados digitales, que nos permite establecer bases sólidas en esta línea de investigación, para en forma continua poder obtener más productos tecnológicos con alta tecnología diseñada en México. A la vez se tuvo la oportunidad de preparar dos tesistas de maestría, vinculados con esta línea de investigación, aportando otro granito de arena en este gran mar de conocimientos.

6.2. Conclusiones

1. Se definió una arquitectura simplificada y diseñada en forma parametrizable (Secc. 4.4), esto es que puede crecer dependiendo del número de estados del sistema y del número de mediciones, quedando un sistema abierto y expandible.

2. Para optimizar el área en el chip, se diseñó el multiplicador con un esquema jerárquico, el cual para la síntesis utiliza celdas primitivas de sumadores que reducen considerablemente el área de silicio requerida en la implementación, reduciendo considerablemente el número de transistores así como la energía consumida. Cabe señalar que la operación de multiplicación es una de las más utilizadas y su ejecución se realiza en un solo ciclo de reloj.

3. La implementación para el inverso multiplicativo que se requiere en la solución de la matriz inversa hace uso del multiplicador, por lo que se estructuró la solución del inverso multiplicativo mediante la técnica de aproximaciones sucesivas, requiriendo de 25 ciclos de reloj para completar el resultado. Aún cuando esta operación es la más lenta de todas, solo se evalúa una sola vez por cada muestra.

4. La respuesta y eficiencia del FK convencional se ve afectada considerablemente por el ancho de la palabra y los bits usados en la parte fraccional de la representación aritmética empleada.

   - Los formatos estándar de punto flotante para 32 y 64 bits utilizan una precisión de 24 y 53 bits respectivamente, es por ello que la estabilidad y convergencia del algoritmo no se ve alterada por efectos de redondeo.
   - Para evitar problemas en la estabilidad y convergencia debido a efectos de redondeo debemos evaluar el algoritmo con al menos 5 dígitos en la parte fraccionaria, 9 para la parte entera mas un dígito para el signo (Secc. 5.1.1).
   - Si el número de bits para la fracción es pequeña, la respuesta en la estimación puede presentar una mayor incertidumbre. Es por ello que un mayor número de dígitos para la parte fraccionaria beneficia a la estimación del filtro.
   - Los resultados mostraron que al utilizar más de 9 bits para la fracción en la implementación VLSI con aritmética de punto fijo, los parámetros como SNR y error típico son muy cercanos a los parámetros obtenidos con el formato estándar de punto flotante.
   - Para la implementación de la arquitectura en CI se eligió el ancho de palabra de 24 bits de manera conservadora, de los cuales los 14 menos significativos representan la parte fraccionaria, que equivalen a 4 dígitos decimales. Dando así un margen de error de 10^{-2} en la estimación (Fig. 5.3), lo cual lo hace un formato balanceado y a la vez permite optimizar el área y retardo en su implementación VLSI.

5. Para la solución del FK es necesario dar solución a la inversa de una matriz cuadrada cuya dimensión depende del número de mediciones del sistema. Para lograr esto se utiliza la formulación del FK secuencial, el cual evita la solución de la matriz inversa realizando un cálculo escalar con cada una de las mediciones (Secc. 2.4).

6. Se realizó un análisis del tiempo de ejecución en ciclos de reloj para el algoritmo FK convencional y de la formulación secuencial incluyendo la reformulación de Joseph para mejorar la estabilidad. El máximo tiempo computacional para la ejecución del algoritmo FK de dos estados es de 113 ciclos de reloj para la formulación convencional y 150 ciclos para la formulación de Joseph. En las pruebas de validación se determinó que la frecuencia máxima de operación es de 20 MHz. Bajo estas condiciones el tiempo de ejecución del FK de dos estados se ejecuta en 5.5 μs para la convencional y 7.7 μs para la formulación de Joseph. El estudio mostró que para un caso de 10 estados con 10 variables medidas se requieren

110,101 ciclos de reloj que se ejecutarían en 304 μs. Esto valida la gran ventaja de ejecutar el algoritmo en hardware sobre aplicaciones que requieren estimaciones en tiempo real.

7. Debido a que el ancho de banda de los sensores inerciales es baja (del orden de 50 a 300 actualizaciones por segundo) las frecuencias con que puede operar el chip están en el orden de KHz, a diferencia de las implementaciones en sistemas embebidos que hacen uso de frecuencias de decenas de MHz. Esta características hace a nuestra propuesta una solución robusta contra problemas debidos a efectos por interferecias electromagnéticas.

8. Podemos comprobar una de las hipótesis de la tesis, donde se establece disminuir el tiempo de ejecución del algoritmo del orden de milisegundos a microsegundos revisando los tiempos de ejecución logrados con la implementación por software descrita en la Secc. 5.1.3 contra la implementación por hardware del presente trabajo. El tiempo de ejecución en la implementación por software fue de 1.3 ms, mientras que en la implementación VLSI se ejecuta en 5 μs con una frecuencia de reloj de 20 MHz. Un tiempo de ejecución equivalente a 1.5 ms en el CI se puede desarrollar si se opera a una frecuencia de 100 KHz.

9. La validación de la arquitectura propuesta fue realizada con un banco de pruebas bajo un entorno controlado, con información en tiempo real generada por sensores inerciales comerciales. Los resultados obtenidos en la la Secc. 5.2 reflejan la veracidad en los resultados de la estimación tanto con el CI denominado PEMFK así como el mismo algoritmo ejecutado por software en un procesador digital 16 bits (dsPIC33). La referencia se obtuvo con la simulación por Matlab utilizando los mismos parámetros y muestras pero evaluado con un modelo de punto flotante para lograr la mejor estimación y evitar el problema del redondeo.

10. El CI PEMFK presentado en esta Tesis Doctoral está implementado con la mínima cantidad de 70,192 transistores (Secc. 4.7). La corriente máxima de consumo estimada es de 840 μA y la corriente promedio medida en la etapa de validación fue de 340 μA. La potencia máxima estimada fue de 27 mW/MHz, y medida para una frecuencia de reloj de 200 KHz con una alimentación de 3.3 V fue de 1.1 mW. Estos resultados nos validan otra de las hipótesis del proyecto.

11. La frecuencia de operación máxima validada con el banco de pruebas es de 25 MHz con un rendimiento de 166,667 actualizaciones por segundo y una latencia de 5.9 μS.

12. Se presenta una implementación VLSI optimizada para un FK convencional de dos estados, con dimensiones de 5.6 mm^2.

6.3. Contribuciones

- Se reporta una arquitectura VLSI optimizada donde:
  - El diseño lógico del multiplicador estructural descrito en la Secc. 4.4.2 permitió lograr una optimización en área de silicio minimizando el área requerida. Una implementación de 2 estados requirió de 70,192 transistores comparada contra implementaciones que reportan hasta 10 veces más transistores [13, 28, 35, 66, 68, 69].
  - La implementación VLSI reduce el tiempo de ejecución del algoritmo al orden de microsegundos, validando otra de las hipótesis establecidas, (por un factor de 313 veces menor que un sistema embebido de 32 bits).
  - Es posible operar a frecuencias de reloj del orden de los KHz haciendo al sistema más inmune contra problemas de interferencia electromagnética comparado con los sistemas que operan en rangos de frecuencias de decenas de MHz. A una frecuencia de 200 KHz se desarrolla una eficiencia similar que un sistema embebido con mayores recursos computacionales a 64 MHz.
  - Se puede mejorar sustancialmente la eficiencia utilizando frecuencias mayores de reloj sin un aumento en el consumo energético. Esto permite la implementación de filtros con un número de estados considerable, en aplicaciones de tiempo real.
  - En la implementación VLSI de 2 estados, la energía requerida es del orden de 1mW, con un consumo de corriente promedio de $340\mu A$ Aquí se valida la hipótesis sobre la disminución de la potencia requerida al orden de los micro-watts. Pudiendo en un futuro ser energizado con sistemas tipo *harvesting*.
  - Debido a que la tendencia actual es el desarrollo de sistemas embebidos portables, donde su alimentación es por baterías, este desarrollo aporta un gran beneficio para este tipo de sistemas, facilitando su integración con tecnología MEMS para desarrollar sensores inteligentes.

- Los productos obtenidos durante la investigación han sido:

  - Una arquitectura VLSI parametrizable del algoritmo del FK. En dicha arquitectura se diseñó un multiplicador optimizado en área de silicio con una metodología que es patentable, por lo cual ya se presentó la documentación requerida en la oficina de transferencia de tecnología del CIDESI para realizar el registro.

  - Informe técnico sobre la síntesis de la arquitectura de un FK de 2 estados y su fabricación en chip.

  - Reporte sobre la metodología empleada en las pruebas de los chips fabricados a través de los servicios de MOSIS y su validación en tiempo real .

  - Se generaron 3 artículos de revista, de los cuales dos son indizadas y una arbitrada. Dos artículos ya están publicados y uno en proceso de revisión.

  - Se participó con 3 artículos en congresos internacionales y 1 simposium nacional.

  - Se registró ante INDAUTOR el *Software para la generación de trazado de circuito integrado de comunicaciones digitales y operaciones aritméticas* con número de registro 03-2014-051611225300-01 en mayo de 2014.

  - Se realizó ante el IMPI una solicitud de registro de esquema de trazado de circuito integrado con fecha 22 de septiembre de 2014 y número de expediente MX/t/2014/000001, folio MX/E/2014/069648 la cual ya fue otorgada.

  - Durante el periodo de investigación se asesoraron 2 tesis de maestría, una de las cuales se concluyó en diciembre de 2013 y la segunda está en proceso.

    - García López R. Itzel, Tesis: "Síntesis del protocolo de comunicación SPI en un módulo VLSI", CIDESI, 2013.
    - Rodríguez Romano Claudia S., Tesis: "Síntesis de la arquitectura de un microprocesador tipo RISC para aplicaciones en sistemas embebidos ", CIDESI, en proceso.

6.4. Publicaciones

1. Arbitrada

   - Chávez-Bracamontes Ramón, et al, "VLSI Design with Alliance Free CAD Tools: an Implementation Example", Revista: Ingeniería Investigación y Tecnología, volumen XVI (número 3), julio-septiembre 2015: 441-452, ISSN 1405-7743, México.

2. Indizada

   - Chávez-Bracamontes Ramón, et al, "VLSI architecture of a Kalman filter optimized for real-time applications", Revista: IEICE Electronics Express (ELEX), enviado junio 2015, ISSN:1349-2543, Japón, Factor de impacto: 0.320 (2014-2015).
   - Manuel Bandala, et al, "Multi-rate sensor fusion for underwarer heading estimation", Revista: Industrial Robot: An International Journal, 41/4 (2014) 347–350, ISSN: 0143-991X, [DOI 10.1108/IR-04-2014-0321], Factor de impacto: 0.635 (2014-2015).

3. Congreso internacional

   - Chávez-Bracamontes Ramón, et al, "Análisis de la complejidad computacional del Filtro de Kalman convencional aplicado a navegación inercial", ROPEC, 2012, Colima, México.
   - García López R. Itzel, et al, "Arquitectura abierta del protocolo de comunicación SPI en un módulo VLSI", ROPEC 2012, Noviembre 2012, Colima, México.
   - Rodríguez Romano Claudia S., et al, "Síntesis de un procesador digital elemental", CIINDET 2015, Morelos, México.

4. Simposium Nacional

   - Rodríguez Romano Claudia S., et al, "Diseño de un microprocesador RISC para la creación de ASICs", SIRCOM, 2015, Colima, México.

6.5. Recomendaciones futuras

EL diseño y desarrollo de circuitos integrados representa una oportunidad tecnológica para México lo cuál no debe perder continuidad, sino por el contrario fortalecer sus capacidades tanto tecnológicas como científicas en ésta materia. Con este trabajo queda de manifiesto que es factible el desarrollo de este tipo de sistemas así como la generación de conocimiento y de personal especialista. Las siguientes recomendaciones se hacen para continuar la investigación y hacer nuevos avances en este campo.

Recomendaciones específicas sobre el algoritmo

1. Pueden ser implementadas formulaciones diferentes a la convencional haciendo uso de una memoria externa, sustituyendo el microcódigo de la ROM interna definido en la Secc. 4.4.3 en una memoria externa.

2. La versión implementada del FK incluyó la modificación por Joseph en el cálculo de la covarianza del error, este caso ayuda a la estabilización en la respuesta numérica. Pueden omitirse los ciclos del 94 al 129 reduciendo un 25 % el tiempo de ejecución del algoritmo, pudiéndose llevar a cabo en solo 115 ciclo de reloj en lugar de 150.

Recomendaciones específicas sobre la arquitectura

1. Una primera acción se enfocaría sobre la optimización de área y tiempos de ejecución de cada uno de los bloques definidos en la arquitectura.

2. Modificar la arquitectura para un caso de 3 o 4 estados, esto requiere del rediseño del banco de datos que es el único módulo que se incrementaría en área.

3. Rediseñar un banco de pruebas que considere los efectos de capacitancias parásitas con la finalidad de subir las frecuencias de reloj y mejorar los límites.

4. Incluir en la unidad aritmética la operación de raíz cuadrada, con la finalidad de poder implementar algoritmos como los definidos por Potter [15] y Kaminski [22].

Referencias

[1] M. Bandala, "Supervisan aeropuertos nacionales con tecnología de CIDESI." Blog del Sistema de Centros Públicos de Investigación Conacyt, Abril 2014.

[2] T. S.-J. et al., "Design of ROVs for the Mexican Power and oil industries," *1rst International Conference on Applied Robotics for the Power Industry*, October 2010.

[3] C. Rubio-González and J. A. Monterrubio-López, "Diseño mecánico de Diablo Ultrasónico para la medición de espesores," No. ISBN: 978-607-95309-3-8, SOMIM, September 2010.

[4] R. Baker, *CMOS Circuit Design, Layout, and Simulation.* IEEE Press Series on Microelectronic Systems, IEEE Press, 3d. ed., 2010.

[5] M. S. Grewal and A. P. Andrews, *Kalman Filtering Theory and Practice Using MATLAB.* Wiley-IEEE Press, 4th ed., September 2015.

[6] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, pp. 35–45, March 1960.

[7] A. H. Jazwinski, *Stochastic Processes and Filtering Theory.* New York: Elsevier, 2007.

[8] L. A. Mcgee and S. F. Schmidt, "Discovery of the Kalman Filter as a practical tool for aerospace and industry," tech. rep., National Aeronautics and Space Administration, 1985.

[9] T. Soong, "On a priori statistics in minimum variance estimation problems," *J. Basic Eng.*, vol. 87D, pp. 109–112, 1965.

[10] T. Nishimurs and M. Nead, "On the apriori information in sequential estimation problems," *IEEE Trans. Automatic Control*, vol. AC11, no. 2, pp. 197–204, 1966.

[11] R. E. Griffin and A. P. Sage, "Sensitivity analysis of discrete filtering and smooting algorithms," *AIAA J.*, vol. 7, no. 10, pp. 1890–1897, 1969.

[12] N. F. Toda, F. Shlee, and P. Obsharsky, "Region of Kalman Filter convergence for several autonomous navigation modes," *AIAA J.*, vol. 7, pp. 622–627, 1969.

[13] K. D. Rao, "Parallel implementation of radar tracking Extended Kalman Filters on transputer networks," *IEEE Transactions on Aerospace and Electronic Systems,*, vol. 31, pp. 857 – 862, April 1995.

[14] R. Bucy and P. D. Joseph, *Filtering for stochastic processes with applications to guidance.* American Mathematical Society Chelsea, 2005.

[15] J. Potter and R. Stern, *Statistical Filtering of Space Navigation Measurements.* Guidance and Control Conference, 1963.

[16] A. Andrews, "A square root formulation of the Kalman covariance equations," *AIAA J.*, vol. 6, pp. 1165–1166, 1968.

[17] G. H. Golub, "Numerical methods for solving linear least-squares problems," *Math*, vol. 7, pp. 206–216, 1965.

[18] R. J. Hanson and C. L. Lawson, "Extension and application of the Householder algorithm for solving linear least-squares problems," *Math Comp*, vol. 23, no. 108, pp. 787–812, 1969.

[19] A. H. Householder, "Unitary triangularization of nonsymmetric matrix," *J. Assoc. Comp. Mach.*, vol. 5, pp. 339–342, 1958.

[20] H. C. Cox, "Estimation of state variables and parameters for noisy dynamic systems," *IEEE Trans. Aero. Elect. Systems*, vol. AC-9, pp. 5–12, 1964.

[21] P. Dyer and McReynolds, "Extension of Square-Root filtering to include process noise," *J. Opt. Theory and aplication*, vol. 3, no. 6, pp. 444–459, 1969.

[22] P. G. Kaminski, *Square Filtering Root and Smoothing For Discrete Processes.* PhD thesis, Stanford University, July 1971.

[23] P. G. Kaminski and A. Bryson, "Discrete square root smoothing," *Proc. AIAA*, pp. 72–877, 1972.

[24] M. Morf and T. Kailath, "Square-Root algorithms for Least-Squares estimation," *IEEE Transactions on Automatic Control,*, vol. AC-20, pp. 487–497, August 1975.

[25] G. J. Bierman, "Measurement updating using the U-D factorization," in *IEEE Conf. on Decision and Control*, pp. 337–346, 1975.

[26] H. Kung, "Why systolic architectures?," *Computer*, vol. 15, pp. 37–46, 1982.

[27] J. Graham and T. Kadela, "Parallel algorithms and architectures for optimal state estimation," *IEEE Transactions on Computers*, vol. C-34, pp. 1061 – 1068, Nov. 1985.

[28] S. T. and H. Y., "Parallel VLSI implementation of the Kalman Filter," *IEEE Trans. on Aerospace and Electronic Systems*, vol. AES23, pp. 215–224, March 1987.

[29] L. L. Scharf and S. Sigurdsson, "Fixed point implementation of Fast Kalman Predictors," *IEEE Tran. on Automatic Control*, vol. AC-29, pp. 850–852, 1984.

[30] J. M. Jover and T. Kailath, "A parallel architecture for Kalman Filter measurement update and parameter estimation," *Automatica*, vol. 22, no. 1, pp. 43–57, 1986.

[31] F. El-Hawary and Ravindranath, "Application of array processing for parallel linear recursive Kalman filtering in underwater acoustic exploration," *IEEE*, pp. 336–340, 1986.

[32] S. Kung and J. Hwang, "Systolic designs for state space models: Kalman filtering and neural network," *Decision and Control IEEE*, vol. 26, pp. 1461–1467, 1987.

[33] P. K. Behera, "Implementation of factorized filter in systolic arrays," *IEEE*, pp. 2065–2068, 1988.

[34] F. Gaston, G. Irwin, and J. Mcwhirter, "Systolic square root covariance Kalman filtering," *Journal of VLSI Signal Processing*, vol. 2, pp. 37–49, 1990.

[35] P. Rao and M. A. Bayoumi, "An efficient VLSI implementation of Real-Time Kalman Filter," *IEEE*, pp. 2353–2356, 1990.

[36] S.-Y. Kung and J.-N. Hwang, "Systolic array designs for Kalman filtering," *IEEE Transactions on Signal Processing*, vol. 39, pp. 171 – 182, Jan 1991.

[37] M. A. Bayoumi, P. Rao, and B. Alhalabi, "VLSI Parallel architecture for Kalma Filter an algorithm specific approach," *Journal of VLSI Signal Processing*, vol. 4, pp. 147–163, 1992.

[38] Z. Youmin, P. Quan, Z. Hongcai, and D. Guanzhong, "A parallel decoupled Kalman filtering algorithm and systolic architecture," *Proceedings of the 32nd Conference on Decision and Control*, 1993.

[39] M. Moonen, "Implementing the Square-Root Information Kalman Filter on a Jacobi-Type systolic array," *Journal of VLSI Signal Processing*, vol. 8, pp. 283–291, 1994.

[40] C. J. Fayomi, M. Sawan, and S. Bennis, "Paralel VLSI implementation of a new simplified architecture of Kalman Filter," *CCECE*, no. 0-7803-276-7-9, 1995.

[41] D. Massicotte, R. Z. Morawsk, and A. Barwicz, "Kalman-filter-based algorithms of spectrometric data correction: Part 1 an iterative algorithm of deconvolution," *IEEE Tran. on Instrumentation and Measurement*, vol. 46, pp. 678–684, June 1997.

[42] K. Sakkay, D. Massicotte, and A. Barwicz, "A systolic architecture for spectrometric data correction based on Kalman-spline and LMS filters," *IEEE*, pp. 357–360, 1998.

[43] A. Mozipo, "A parallel architecture for adaptive channel equalization based on Kalman Filter using MMAlpha," in *Electrical and Computer Engineering, 1999 IEEE Canadian Conference on*, 1999.

[44] T. Kailath, A. Sayed, and B. Hassibi., *Linear Estimation*. Tom Robbins, 2000.

[45] K. Santha and V. Vaidehi, "Parallel-Pipelined architecture for the Kalman based adaptive equalizer," *IEEE - ICSCN*, pp. 172–177, 2007.

[46] K. Yao, "Systolic algorithms and architectures for high-throughput processing applications," *Journal of Signal Processing Systems*, vol. 53, pp. 15–34, 2008.

[47] M. S. Grewal and J. Kain, "Kalman Filter implementation with improved numerical properties," *IEEE Tran. On Automatic Control*, vol. 55, pp. 2058–2068, 2010.

[48] B. Anderson and J. B. Moore, *Optimal filtering*. Information and System Science, Prentice Hall, 1979.

[49] G. J. Bierman, "A comparison of discrete linear filtering algorithms," *IEEE Trans. Aero. Elect. Systems*, vol. AES-9, no. 1, pp. 28–37, 1973.

[50] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design A Circuits and System Perspective*. Addison-Wesley, 2011.

[51] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann, 2004.

[52] C. A. Piña, "MOSIS: IC Prototyping and low volume production service," *IEEE*, vol. 1, pp. 4–5, 2001.

[53] J. Staudhammer, "Educational use of MOSIS," *IEEE*, vol. 1, pp. 147–148, 1997.

[54] MOSIS, "The MOSIS Service." Retrieve on http://www.mosis.com/pages/products/mep/mep-about, 09/10/14.

[55] O. Semiconductor, "ON Semiconductor processes [on line]." Retrieve on http://www.onsemi.com/PowerSolutions/content.do?id=16558&lctn=home, 06/12/2015.

[56] LIP6, "Alliance VLSI CAD System [on line]." Retrieve on https://soc-extras.lip6.fr/en/alliance-abstract-en/, 01/20/15.

[57] K.-S. Lam and F. Ak, "Alliance tutorial, Part 1 VHDL Modeling and simulation [on line]." Retrieve on https://soc-extras.lip6.fr/en/alliance-abstract-en/, 01/20/15.

[58] K.-S. Lam and F. Ak, "Alliance tutorial, Part 2 Logic synthesis [on line]." Retrieve on http://www-asim.lip6.fr/pub/alliance/distribution/latest/, 01/20/15.

[59] K.-S. Lam and F. Ak, "Alliance tutorial, Part 3 Place and Route [on line]." Retrieve on http://www-asim.lip6.fr/pub/alliance/distribution/latest/, 01/20/15.

[60] C. Silva, T. Yoshida, and A. Palacios, "Introduction to VLSI CMOS circuits design." Ministry of Education and Science of Japan and the Toin University of Yokohama, 2006.

[61] J.-P. Chaput and F. Pétrot, *GenLib User's Manual [on line].* Pierre & Marie Curie University, LIP6 ASIM Department, Retrieve on ftp://ftp.lip6.fr/lip6/softs/alliance/latest-checkout/alliance/src/genlib/doc/genlib.pdf.

[62] J. Esfandyari, R. D. Nuccio, and G. Xu, "Solutions for MEMS sensor fusion." Solid State Technology, STMicroelectronics, Coppell, TX USA, 2011.

[63] M. S. Grewal and A. P. Andrews, *Kalman Filtering Theory and Practice Using MATLAB.* Wiley-IEEE Press, September 2008.

[64] F. Espinoza, "Plataforma experimental con helicóptero no tripulado," Master's thesis, CIDE-SI, Querérato, México., 2015.

[65] A. García-QunchÍa, "A system-on-chip (soc) platform to integrated navigation systems & gpps," *IEEE International Symposium on Industrial Electronics*, pp. 603–608, 2009.

[66] Y. Guo, J. Zhang, D. McCain, and J. R. Cavallaro, "Structured parallel architecture for displacement MIMO Kalman equalizer in CDMA systems," *IEEE Tran. On Circuits and Systems*, vol. 54, pp. 122–126, February 2007.

[67] A. Barwicz, D. Massicotte, Y. Savaria, P. A. Pango, and R. Z. Morawski, "An application-specific processor dedicated to Kalman-Filter-based correction of spectrometric data," *IEEE Tran. on Instrumentation and Measurement*, vol. 44, June 1995.

[68] D. Massicotte, "A systolic VLSI implementation of Kalman-Filter-based algorithms for signal reconstruction," *IEEE*, vol. 1, pp. 3029–3032, 1998.

[69] A. Chacón-Rodríguez, P. Julián, and F. Masson, "Fast and low power integrated circuit for impulsive sound localisation using Kalman Filter approach," *Electronics Letters*, vol. 46, April 2010.

[70] K. Gilleo, *MEMS/MOEMS Packing concepts, designs, materials and process, Nanoscience and Tecnology*. Mc Graw Hill, 2005.

[71] M. S. Grewal, L. R. Weill, and A. P. Andrews, *Global Positioning System, Inertial Navigation, and Integration*. John Wiley & Sons, 2007.

[72] O. J. Woodman, "An introduction to inertial navigation," tech. rep., University of Cambridge, August 2007.

[73] A. Sripad and D. Snyder., "A necessary and sufficient condition for quantization error to be uniform and white," *IEEE Tran. On Acoustic, Signal, and Speech Processing*, vol. 25(5), pp. 442–448, October 1997.

[74] L. Wanhammar, *DSP Integrated Circuits*. Academic Press, 1999.

[75] M. S. Committee, *IEEE Standar for Floating-Point Arithmetic*. IEEE Computer Society, 2008.

[76] C. Mead and L. Conway, *Introduction to VLSI Systems*. MA: Addison-Wesley, 1980.

[77] L. Xiu, *VLSI Circuit Design Methology Demystified A Conceptual Taxonomy*. Wiley-Interscience, 2008.

Apéndice A

Código

Scripts bloque Secuenciador

flag_script

```
 #!/bin/bash
# Script para sintetizar flag.vst
nombre=flag
echo "PROYECTO ACTUAL: ${nombre}"
vasy -apo -I vhdl ${nombre}  &&
boom -VA ${nombre} ${nombre}_boom &&
boog ${nombre}_boom ${nombre}_boog &&
loon ${nombre}_boog ${nombre} &&
ocp -v -rows 1 ${nombre} ${nombre} &&
genpat -v ${nombre}_datos
asimut ${nombre} ${nombre}_datos ${nombre}_simula  &&
xpat -l ${nombre}_simula
exit 0
```

rom_script

```
#!/bin/bash
nombre=rom
boom -A -V -l 2 -j ${nombre} ${nombre}_boom &&
boog ${nombre}_boom ${nombre}_boog &&
loon ${nombre}_boog ${nombre} &&
ocp -v -rows 9 ${nombre} ${nombre} &&
genpat -v rom_datos &&
asimut   ${nombre} rom_datos rom_simula &&
xpat -l rom_simula
exit 0
```

sec_script

```
#!/bin/bash
nombre=sec
export MBK_OUT_LO=vst &&
export VH_MAXERR=1 &&
genlib -v ${nombre}
ocp -v -partial ${nombre} ${nombre} ${nombre}_ocp &&
genpat -v sec_datos
asimut -zd ${nombre} sec_datos ${nombre}_simula  &&
xpat -l ${nombre}_simula
exit 0
```

Scripts bloque Banco de Datos

banco_regcore_script

```
#!/bin/bash
# Script para sintetizar el bloque banco_reg_core
genlib -v banco_reg_dpt &&
genlib -v banco_reg_ctl &&
genlib -v banco_reg_core &&
ocp -v -partial banco_reg_core_place  banco_reg_core banco_reg_core &&
# Simulacion
genpat -v banco_reg_core_datos &&
asimut  banco_reg_core banco_reg_core_datos banco_reg_core_simula   &&
xpat -l banco_reg_core_simula
exit 0
```

banco_decod_script

```
#!/bin/bash
# Script para sintetizar el decodificador decod del banco de memoria
nombre=banco_decod
echo "PROYECTO ACTUAL: ${nombre}"
vasy -apo -I vhdl ${nombre}  &&
boom -VA ${nombre} ${nombre}_boom &&
boog ${nombre}_boom ${nombre}_boog &&
loon ${nombre}_boog ${nombre} &&
ocp -v -rows 1 ${nombre} ${nombre} &&
genpat -v ${nombre}_datos
asimut -zd ${nombre} ${nombre}_datos ${nombre}_simula &&
xpat -l ${nombre}_simula
exit 0
```

banco_sinc_write_script

```
#!/bin/bash
nombre=banco_sinc_write
export MBK_OUT_LO=vst &&
export VH_MAXERR=1 &&
vasy -apo -I vhdl ${nombre} ${nombre}_vasy &&
boom -VA ${nombre}_vasy ${nombre}_boom &&
boog ${nombre}_boom ${nombre}_boog &&
loon ${nombre}_boog ${nombre} &&
ocp -v -rows 5  ${nombre} ${nombre} &&
genpat -v ${nombre}_datos
asimut  -zd ${nombre} ${nombre}_datos ${nombre}_simula &&
xpat -l ${nombre}_simula &&
exit 0
```

banco_script

```
#!/bin/bash
# Script para sintetizar el modulo de banco de datos (banco)
nombre=banco
export MBK_OUT_LO=vst &&
export VH_MAXERR=1 &&
genlib -v ${nombre}  &&
ocp -v -partial ${nombre}_place ${nombre} ${nombre}_ocp &&
genpat -v ${nombre}_datos
asimut -zd ${nombre} ${nombre}_datos ${nombre}_simula  &&
xpat -l ${nombre}_simula
exit 0
```

Scripts bloque Unidad Aritmética

Script para sub-módulo comparador del inverso multiplicativo

inv_comp_script

```
#!/bin/bash
#  Script para sintetizar el comparador de 48 bits para el modulo del inverso multiplicativo
nombre=inv_comp
vasy -apo -C 2 -I vhdl ${nombre} ${nombre}_vasy &&
boom -VA ${nombre}_vasy ${nombre}_boom &&
boog ${nombre}_boom ${nombre}_boog &&
loon ${nombre}_boog ${nombre} &&
genpat -v ${nombre}_datos &&
asimut  -zd ${nombre} ${nombre}_datos ${nombre}_simula &&
xpat -l ${nombre}_simula &&
exit 0
```

inv_controla_script

```
#!/bin/bash
#  Script para la sintesis del modulo de control del  inverso multiplicativo
nombre=inv_controla
syf -CVE -a inv_control &&
# Antes de ejecutar los siguientes comandos es necesario agregar  las siguientes lineas
# al archivo inv_control.vbe generado por  la herramienta syf despues de la linea
#  clear24 : out bit_vector(23 DOWNTO 0);   -- clear24
#  vdd : in BIT;    -- vdd
#  vss : in BIT -- vss
#  );
boom -V ${nombre} ${nombre}_boom &&
boog ${nombre}_boom ${nombre}_boog &&
loon ${nombre}_boog ${nombre} &&
genpat -v ${nombre}_datos
asimut -zd ${nombre} ${nombre}_datos ${nombre}_simula &&
xpat -l ${nombre}_simula
exit 0
```

inv_meta_script

```
#!/bin/bash
# Script para sintetizar el modulo de meta del inverso multiplicativo
nombre=inv_meta
vasy -apo -I vhdl ${nombre} &&
boom -V ${nombre} ${nombre}_boom &&
boog ${nombre}_boom ${nombre}_boog &&
loon ${nombre}_boog ${nombre} &&
genpat  -v ${nombre}_datos
asimut ${nombre} ${nombre}_datos ${nombre}_simula &&
xpat -l ${nombre}_simula
exit 0
```

Script para sub-módulo regy del inverso multiplicativo

inv_regy_script

```
#!/bin/bash
nombre=inv_regy
vasy -apo -I vhdl ${nombre} &&
boom -V ${nombre} ${nombre}_boom &&
boog ${nombre}_boom ${nombre}_boog &&
loon ${nombre}_boog ${nombre} &&
genpat -v ${nombre}_datos
asimut -b ${nombre} ${nombre}_datos ${nombre}_simula &&
xpat -l ${nombre}_simula
exit 0
```

ua_inv_script

```
#!/bin/bash
# Script para sintetizar el modulo de inverso multiplicativo
nombre=ua_inv
genlib -v ${nombre} &&
flatlo -r ${nombre} ua_inversa
ocp -v -rows 76 ua_inversa ua_inversa &&
exit 0
```

sinc_sr_script

```
#!/bin/bash
nombre=sinc_sr
vasy -apo -I vhdl ${nombre} ${nombre}_vasy &&
boom -VA ${nombre}_vasy ${nombre}_boom &&
boog ${nombre}_boom ${nombre}_boog &&
loon ${nombre}_boog ${nombre} &&
ocp -v -rows 3 ${nombre} ${nombre} &&
genpat -v ${nombre}_datos
asimut  -zd ${nombre} ${nombre}_datos ${nombre}_simula &&
xpat -l ${nombre}_simula &&
exit 0
```

core_script

```
#!/bin/bash
nombre=core
genlib -v ${nombre}
ocp -v -ioc ${nombre} -partial ${nombre}_p ${nombre} ${nombre} &&
nero -V -L -2 -p ${nombre} ${nombre}  pem24 &&
genpat -v core_datos
asimut   ${nombre} core_datos ${nombre}_simula   &&
xpat -l ${nombre}_simula
export MBK_OUT_LO=al &&
cougar -v -t pem24 ${nombre}_trans &&
cougar -v pem24 ${nombre}_cougar &&
export MBK_OUT_LO=vst &&
lvx vst al ${nombre}  ${nombre}_cougar -f &&
export MBK_OUT_LO=spi &&
cougar -t pem24 pem24 &&
export MBK_OUT_LO=vst &&
graal -l pem24
exit 0
```

Apéndice B

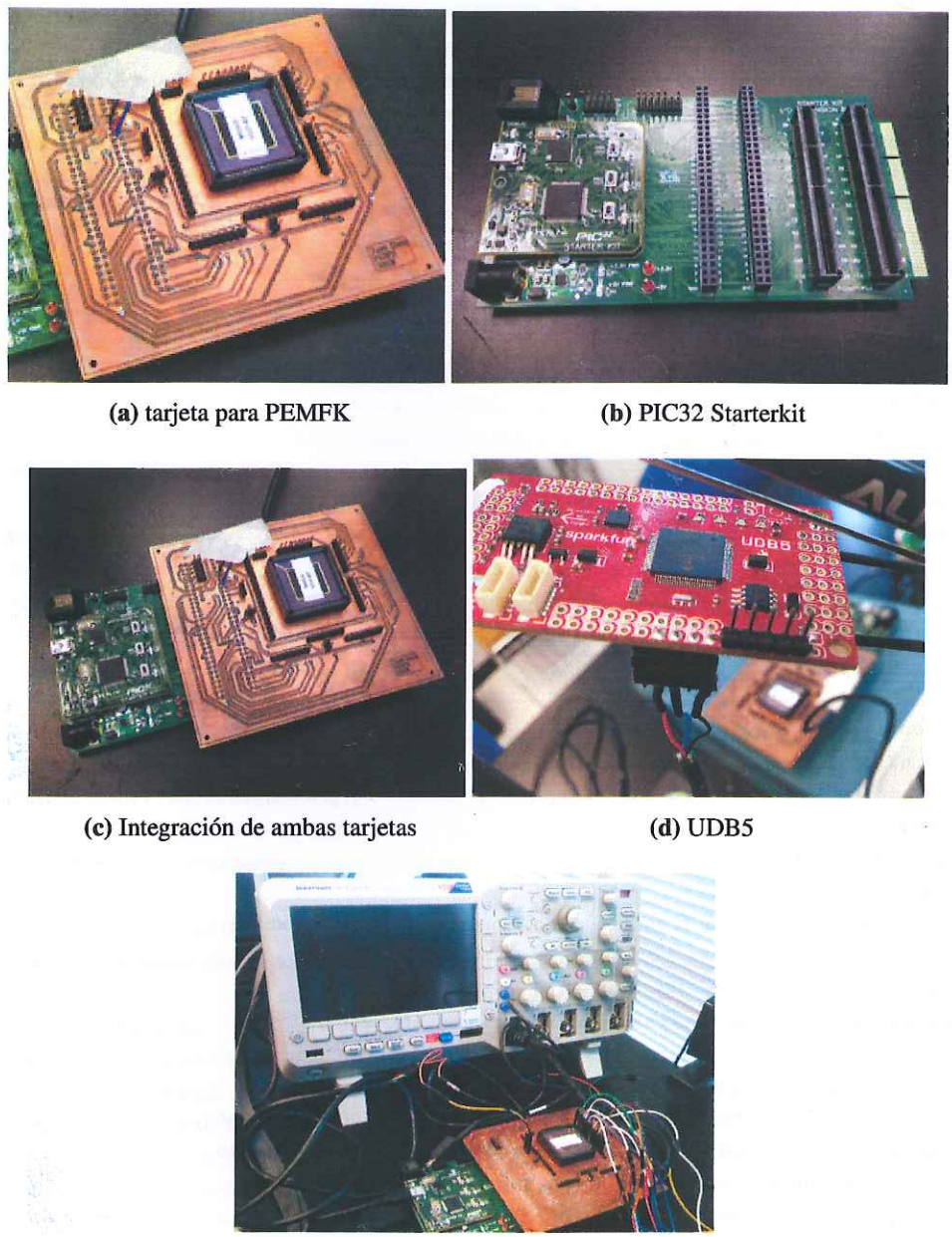
Banco de prueba para validación del chip PEMFK

El banco de pruebas (software y hardware) fue diseñado para comprobar el correcto funcionamiento del circuito integrado PEMFK en el que se implementa el algoritmo del filtro de Kalman.

El banco de pruebas consta de tres sistemas distintos. El primero es una tarjeta electrónica diseñada exclusivamente para el montaje del PEMFK (Fig. B.1a). El segundo sistema es una tarjeta de desarrollo comercial que utiliza un microcontrolador de 32 bits de la compañía Microchip© (PIC32 starterkit, Fig. B.1b). De tal forma que la interconexión de ambas permite la comunicación y recuperación de datos vía puerto serial a una terminal de computadora. Con este banco de pruebas se pueden realizar experimentación en tiempo real y fuera de línea. Un tercer sistema se requiere para las pruebas en tiempo real, el cual está basado en la tarjeta desarrollada y comercializada por Sparkfun: UDB5© (Fig. B.1d). Ésta permite adquirir datos desde el sensor inercial MPU6000 de Invensense© (Fig. B.1d).

El microcontrolador PIC32 es considerado la unidad de control maestro (MCU por sus siglas en inglés) del banco de pruebas. La tarjeta de la Fig. B.1a fue diseñada para ser integrada directamente al puerto de expansión del PIC32 Starter Kit como se muestra en la Fig. B.1c. El sistema PIC32 MCU cuenta con dos módulos UART configurados para comunicación serial utilizando el protocolo RS-232 con un baud rate de 230400 bps. Uno de ellos (UART2) es dedicado para la comunicación entre el sistema PIC32 MCU y el sistema UDB5. El otro módulo (UART1) es utilizado para la comunicación entre el sistema PIC32 MCU y una computadora, la cual a través de una terminal serial, recupera los resultados del banco de pruebas. Es importante tener en cuenta que el PEMFK utiliza una arquitectura de representación binaria decimal de punto fijo de 24 bits con magnitud

y signo, es decir, el bit más significativo del vector representa el signo del dato, siendo un bit en alto una magnitud negativa y un bit en bajo una magnitud positiva. Los siguientes 9 bits más significativos después del signo, representan la parte entera de la magnitud, mientras los 14 bits menos significativos representan la parte fraccional de la misma.



(a) tarjeta para PEMFK

(b) PIC32 Starterkit

(c) Integración de ambas tarjetas

(d) UDB5

(e) Banco de pruebas

Figura B.1: Banco de Pruebas.

Rutinas principales para validación con el banco de pruebas

```
/*****************************************************************************
 * CODIGO EMBEBIDO PARA VALIDAR EL CIRCUITO INTEGRADO PEMFK QUE IMPLEMENTA EL FK
 * Tituto: pemfk.h
 * Proyecto: PIC32 MCU
 * Descripcion:
 *      Este codigo contiene las rutinas de comunicacion y
 *      las funciones necesarias para filtrar datos usando el ASIC PEMFK.
*****************************************************************************/
#include <plib.h>
#include "../KalmanParam.h"
#include "conv.h"
// Prototipo de rutinas de comuncacion.
void writeBusD(unsigned int data);
void writeBusEDIR(unsigned char dir);
void setHL();
void clearHL();
void setW_HL();
void clearW_HL();
void setWrite_B();
void clearWrite_B();
void setReset();
void clearReset();
void setStart();
void clearStart();
unsigned int readBusY();
unsigned char readReady();
// Prototipos de las rutinas de procesamiento.
void setInitConfig();
void loadReg(unsigned long int data, unsigned char dir);
unsigned long int readReg(unsigned char dir);
void initPEMFK();
unsigned char verifyInit();
void verifyInit2();
unsigned char feedData(unsigned long int pang, unsigned long int wang);
unsigned long int getX1();
unsigned long int getX2();
void filterData(unsigned long int pang, unsigned long int wang);
// Inicializacion de array
long unsigned int initConfig[32] =
{0x000000,0x000000,0x000000,0x000000,0x000000,0x000000,0x000000,0x000000,
0x000000,0x000000,0x000000,0x000000,0x000000,0x000000,0x000000,0x000000,
0x000000,0x000000,0x000000,0x000000,0x000000,0x000000,0x000000,0x000000,
0x000000,0x000000,0x000000,0x000000,0x000000,0x000000,0x000000,0x000000};
// Rutina para convertir los valores de parametros del sistema a valores binarios usando "conv.h".
void setInitConfig()
{
    initConfig[0x06] = decimal_binario_fraccion(sn*sn,10,14);
    initConfig[0x07] = decimal_binario_fraccion(P11,10,14);
    initConfig[0x08] = decimal_binario_fraccion(P12,10,14);
```

```
    initConfig[0x09] = decimal_binario_fraccion(P21,10,14);
    initConfig[0x0A] = decimal_binario_fraccion(P22,10,14);
    initConfig[0x13] = decimal_binario_fraccion(Q11,10,14);
    initConfig[0x14] = decimal_binario_fraccion(Q12,10,14);
    initConfig[0x15] = decimal_binario_fraccion(Q21,10,14);
    initConfig[0x16] = decimal_binario_fraccion(Q22,10,14);
    initConfig[0x17] = decimal_binario_fraccion(A11,10,14);
    initConfig[0x18] = decimal_binario_fraccion(A12,10,14);
    initConfig[0x19] = decimal_binario_fraccion(A21,10,14);
    initConfig[0x1A] = decimal_binario_fraccion(A22,10,14);
    initConfig[0x1B] = decimal_binario_fraccion(B11,10,14);
    initConfig[0x1C] = decimal_binario_fraccion(B21,10,14);
    initConfig[0x1D] = decimal_binario_fraccion(C11,10,14);
    initConfig[0x1E] = decimal_binario_fraccion(C12,10,14);
    initConfig[0x1F] = decimal_binario_fraccion(1.0,10,14);
}

/*****************************************************************************
 *      Rutinas de comunicacion  entre el MCU y el PEMFK
*****************************************************************************/
// Rutina que escribe el bus D [11:0] usando los bits [13:7,4:0] del PORTD.
void writeBusD(unsigned int data)
{
    unsigned int dataTemp=0;
    dataTemp=dataTemp|((data&0x3)<<8);       // D[1:0] -> RD[9:8]
    dataTemp=dataTemp|((data&0x4)<<9);       // D[2] -> RD[11]
    dataTemp=dataTemp|((data&0x18)>>3);      // D[4:3] -> RD[1:0]
    dataTemp=dataTemp|((data&0x20)>>1);      // D[5] -> RD[4]
    dataTemp=dataTemp|((data&0x40)>>3);      // D[6] -> RD[3]
    dataTemp=dataTemp|((data&0x80)>>5);      // D[7] -> RD[2]
    dataTemp=dataTemp|((data&0x100)<<2);     // D[8] -> RD[10]
    dataTemp=dataTemp|((data&0x200)>>2);     // D[9] -> RD[7]
    dataTemp=dataTemp|((data&0xC00)<<2);     // D[11:10] -> RD[13:12]
    PORTD=(PORTD&0xC000)|dataTemp;
}
// Rutina que escribe el bus EDIR [4:0] usando los bits [9,7,5,3:2] del PORTA.
void writeBusEDIR(unsigned char dir)
{
    unsigned int dirTemp=0;
    dirTemp=dirTemp|((dir&0x1)<<5);      // EDIR[0] -> RA[5]
    dirTemp=dirTemp|((dir&0x2)<<6);      // EDIR[1] -> RA[6]
    dirTemp=dirTemp|((dir&0x4)<<7);      // EDIR[2] -> RA[5]
    dirTemp=dirTemp|((dir&0x18)>>1);     // EDIR[4:3] -> RA[3:2]
    PORTA=(PORTA&0x40)|dirTemp;
}
// Rutina que pone la terminal HL usando el bit [14] del PORTD.
void setHL()
{
    PORTDbits.RD14=1;
}
```

```
// Rutina que limpia la terminal  HL usando el bit [14] de PORTD.
void clearHL()
{
    PORTDbits.RD14=0;
}
// Rutina que pone la the terminal W_HL usando el bit [6] del PORTA.
void setW_HL()
{
    PORTAbits.RA6=1;
}
// Rutina que limpia la terminal  W_HL usando el bit [6] de PORTA.
void clearW_HL()
{
    PORTAbits.RA6=0;
}
// Rutina que pone la the terminal write_B usando el bit [15] de PORTD.
void setWrite_B()
{
    PORTDbits.RD15=1;
}
// Rutina que limpia la terminal  write_B usando el bit [15] of PORTD.
void clearWrite_B()
{
    PORTDbits.RD15=0;
}
// Rutina que pone la the terminal Reset usando el bit [0] of PORTF.
void setReset()
{
    PORTFbits.RF0=1;
}
// Rutina que limpia la terminal  Reset usando el bit [0] of PORTF.
void clearReset()
{
    PORTFbits.RF0=0;
}
// Rutina que pone la the terminal Start usando el bit [1] of PORTF.
void setStart()
{
    PORTFbits.RF1=1;
}
// Rutina que limpia la terminal  Start usando el bit [1] of PORTF.
void clearStart()
{
    PORTFbits.RF1=0;
}
// Rutina que lee el bus Y [11:0] desde el PEMFK usando los bits [11:0] de PORTB.
unsigned int readBusY()
{
    return PORTB&0xFFF;
}
```

```
// Rutina que lee  la terminal READY desde el PEMFK usando el bit [3] de PORTC.
unsigned char readReady()
{
    return PORTCbits.RC3&0x1;
}
/*****************************************************************************
 *
 *              Rutinas de procesamiento de datos para el PEMFK
 *
*****************************************************************************/

// Rutina que carga un vector de bits [23:0] (data) en un registro
// direccion [4:0] (dir) en el rango of 0 to 31.
// Nota: el proceso de carga consiste de tres pasos...
//      1. Los 12 bits MSBs son almacenados temporalmente en el registro RH ajustando HL y ...
    W_HL en alto.
//      2. Los 12 bits LSBs son almacenados temporalmente en el registro RL con HL en bajo y ...
    W_HL en alto.
//      3. El almacenamiento permanente de los 24 bits del bus RHL usando la terminal Write_B ...
    en alto.
void loadReg(unsigned long int data, unsigned char dir)
{
    // 12 MSBs
    writeBusD((data&0xFFF000)>>12);
    setHL();
    setW_HL();
    waitCycles(3);
    clearW_HL();
    clearHL();

    // 12 LSBs
    writeBusD(data&0x000FFF);
    setW_HL();
    waitCycles(3);
    clearW_HL();

    // Escrituta permanente de los datos en la direccion apuntada por dir.
    writeBusEDIR(dir);
    setWrite_B();
    waitCycles(1);
    clearWrite_B();
    waitCycles(1);
}

// Rutine que lee un registro completo (dir) del PEMFK.
// Nota: El proceso consiste de dos pasos...
//      1. Leer la parte alta del registro (Bits MSBs).
//      2. Leer la parte baja (LSBs) del registro y concatenar a los bits  MSBs.
// Regresa un vector de 32 bits que contiene la lectura del registro completo.
unsigned long int readReg(unsigned char dir)
```

```
{
    unsigned long int dataTemp=0;
    writeBusEDIR(dir);
    waitCycles(1);

    setHL();            // MSBs
    waitCycles(1);
    dataTemp=(readBusY()<<12);

    clearHL();          // LSBs
    waitCycles(1);
    dataTemp=dataTemp|readBusY();
    return dataTemp&0xFFFFFF;
}
// Rutine que inicializa los registros del PEMFK con los valores definidos en initConfig[].
// Adicionalmente, la teminal de reset se ajusta a uno para dejar listo al PEMFK.
void initPEMFK()
{
    unsigned char i;
    setReset();
    waitCycles(5);
    clearReset();
    for(i=0;i<32;i++)
    {
        loadReg(initConfig[i],i);
    }
    setReset();
    waitCycles(5);
    clearReset();
}
// Rutine que verifica el contenido de los registros del PEMFK.
// Se ejecurta una lectura iterativa de cada registro, que es comparada
// con los valores esperados de acuerdo a initConfig[]
// Regresa 1 cuando la verificacion es valida, y 0 cuando no es valida.
// Adicionalmente, un mensaje de la verificacion se envia atraves de UART.
unsigned char verifyInit()
{
    unsigned char i;
    for(i=0;i<32;i++)
    {
        if(readReg(i)!=initConfig[i])
        {
            INTDisableInterrupts();
            putsUART1("**********INVALID PEMFK INITIALIZATION**********\n\r");
            INTEnableSystemMultiVectoredInt();
            return 0;
        }
    }
    INTDisableInterrupts();
    putsUART1("**********VALID PEMFK INITIALIZATION**********\n\r");
```

```
    INTEnableSystemMultiVectoredInt();
    return 1;
}
// Rutina que lee el contenido de los registros del PEMFK.
// Una lectura iterativa de cada registro se ejecuta, la cual es enviada
// via UART a una terminal serial  en formato hexadecimal de 12 caracteres
// Los primeros 6 caracteres (MSBs) representan el valor de la direccion dir
// del registro, mientras los LSBs representan el valor contenido en el PEMFK.
void readInit()
{
    unsigned char i;
    unsigned long int dataTemp;
    writeBusD(initConfig[0x0]>>12);
    INTDisableInterrupts();
    putsUART1("**********INICIO CONTENIDO INICIALIZACION**********\n\r");
    INTEnableSystemMultiVectoredInt();
    for(i=0;i<32;i++)
    {
        dataTemp=readReg(i);
        INTDisableInterrupts();
        sendPacket(i,dataTemp);
        INTEnableSystemMultiVectoredInt();
    }
    INTDisableInterrupts();
    putsUART1("**********FIN CONTENIDO INICIALIZACION**********\n\r");
    INTEnableSystemMultiVectoredInt();
}
// Rutina que alimentala posicion angular y el valor del giroscopio en formato de
// magnitud y signo de 24 bits en los registros 0x00 y 0x01 respectivamente.
// Se genera un alto en la termial START  para iniciar el calculo del filtro.
// La rutina espera que la terminal READY se vaya a alto, cual significa que se ha terminado ...
    de ejecutar una iteracion.
// Ahora se pueden leer las estimaciones en los registros X1 y X2 (reg. 0x02 y 0x03 ...
    respectivamente).
// Un 1 se retorna cuando la rutina se ha terminado.
unsigned char feedData(unsigned long int pang, unsigned long int wang)
{
    loadReg(pang,0x00);
    loadReg(wang,0x01);
    setStart();
    waitCycles(1);
    clearStart();
    waitCycles(2);
    while(READY!=1);
    return 1;
}
// Rutina que lee el registro X1. Regresa un vector entero de 32 bits
// que contiene los 24 bits mas relevantes del valor estimado.
unsigned long int getX1()
{
```

```
    return readReg(0x02);
}
// Rutina que lee el registro X2. Regresa un vector entero de 32 bits
// que contiene los 24 bits mas relevantes del valor estimado.
unsigned long int getX2()
{
    return readReg(0x03);
}
// Rutina que recibe la posicion angular (pang) y la velocidad angular (wang)
// y filtra los datos. El resultado estimado por el PEMFK es recuperado y enviado a traves de ...
    UART a una terminal por puerto serial.
void filterData(unsigned long int pang, unsigned long int wang)
{
    INTEnableSystemMultiVectoredInt();
    unsigned long int x1Temp, x2Temp;
    feedData(pang,wang);        // Carga el registro 0x00 y 0x01, y filtra el dato.
    x1Temp=getX1();
    x2Temp=getX2();             // Recupera el resultado.
    INTDisableInterrupts();
    PORTSetBits(IOPORT_G, BIT_15);
    sendPacket(x1Temp,x2Temp);  // Encia a traves de UART la estimacion obtenida.    ...
    PORTClearBits(IOPORT_G, BIT_15);
}
```

Apéndice C

Sistemas inerciales

Navegación inercial

Se puede definir *parámetros de navegación* al conjunto de valores numéricos que describen la posición, la velocidad y la orientación de un vehículo respecto de un sistema de referencia dado con unidades específicas. Con base en esta denominación, definimos navegación como *el arte y la ciencia que permiten determinar los parámetros de navegación de un vehículo con información disponible a bordo del mismo.*

La navegación inercial se sustenta en un principio básico de la cinemática: *Conocidos en un instante inicial la velocidad, la orientación y la posición de un móvil así como los valores instantáneos presentes y futuros de su aceleración lineal y su velocidad angular relativas a un sistema de referencia dado, es posible calcular la posición, la velocidad y la orientación del vehículo en todo instante futuro.* Cabe destacar que, la aceleración lineal y la velocidad angular de un cuerpo, contrariamente a la posición, la velocidad y la orientación, pueden ser medidas sin información exterior al mismo.

Un sistema de navegación inercial ejecuta las siguientes funciones:

- Mide fuerzas específicas.
- Mide velocidades angulares.
- Modela el campo gravitacional.
- Integra las fuerzas específicas para obtener la posición y la velocidad.

Sensores inerciales

Los sensores inerciales basan las mediciones de aceleración y velocidad angular en los principios de navegación inercial. Los sensores inerciales más comunes son los acelerómetros y los giroscopios.

Acelerómetro

Un simple acelerómetro se puede construir con una masa conectada a un resorte y midiendo su deflexión como se observa en la figura C.1. Los acelerómetros también miden la gravitación, y al total de fuerzas medidas se les conoce como fuerza específica. Usando 3 o más acelerómetros se puede construir un medidor de fuerza específica tridimensional, f_{IB}^{B}, esto significa la fuerza específica de un cuerpo (B) relativa al espacio inercial (I).

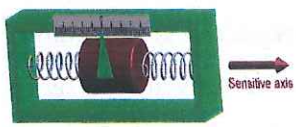


Figura C.1: Acelerómetro.

Giroscopio (*Gyro*)

Los giroscopios miden velocidad angular relativa a un espacio inercial (ω_{IB}^{B}). Los principios de medición son:

- Giroscopio mecánico (*Spinning wheel*)
- Efecto Sagnac
  - Giroscopio de Anillo Láser (RLG)
  - Giroscopio de fibra óptica (FOG)
- Efecto Coriolis
  - MEMS
  - Tuning fork
  - Wine glass

Actualmente hay muchas empresas dedicadas a la fabricación de sensores inerciales [70], como por ejemplo Analog Device, AGNC-coremicro-UNCU, Aeron y muchas más que han logrado desarrollar unidades de medición inercial a precios reducidos y a la vez han mejorado sus características de rango, sensibilidad, resolución y ancho de banda; sin embargo los avances continúan a nivel de dispositivo.

El control de sistemas dinámicos como es el caso de un vehículo autobalanceable, requiere del empleo acertado de giroscopios y acelerómetros, quienes proveen variables de velocidad angular y aceleración; señales que en la mayoría de los casos presentan características de error como la deriva, inestabilidad en el factor de escala y ruido aleatorio, haciendo difícil la tarea de control del sistema.

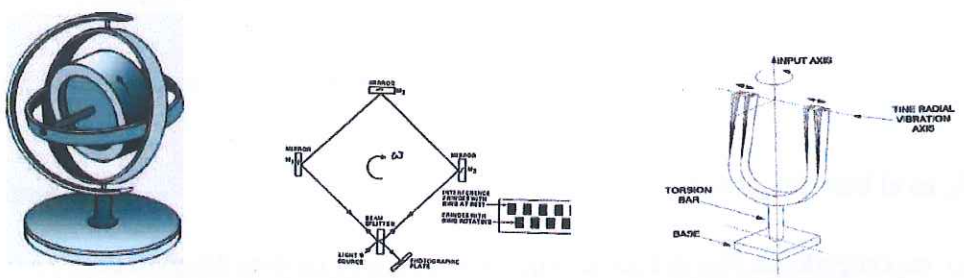
Terminología de sensores inerciales

Aceleración

Es el cambio en la velocidad lineal

Bias

El bias se puede definir como un offset del valor correcto de la señal de salida y se puede separar en dos términos: bias estático y bias dinámico.



(a) Giroscopio mecánico (Caplex, 2000)

(b) Efecto Sagnac (Bose, 1998)

(c) Efecto coriolis (Titterton y Weston, 1997)

Figura C.2: Tipos de giroscopios.

Usualmente se especifica en mg. para acelerómetros (b_a) y en $^o/h$ para giroscopios (b_g)

$$\begin{aligned} b_a &= b_{as} + b_{ad} \\ b_g &= b_{gs} + b_{gd} \end{aligned}$$

El bias estático (b_{as}, b_{gs}) es una constante durante toda la operación, pero difiere en cada corrida de la IMU.

El bias dinámico (b_{ad}, b_{gd}) el cual algunas veces es llamado *bias instability* que es un bias que varía y que puede cambiar sobre un rango específico de tiempo.

Un bias no compensado puede introducir error en un cálculo de velocidad o posición. Un acelerómetro sin compensación de bias introduce un error proporcional al tiempo (t) en velocidad y proporcional a (t^2) en posición.

$$v = \int b_f dt = b_f t \Rightarrow p = \int v dt = \int b_f dt = \frac{1}{2} b_f t^2$$

donde: b_f bias del acelerómetro
p posición
v velocidad

Un giroscopio sin compensación de bias introduce un error angular, $\delta\theta$ proporcional al tiempo.

$$\delta\theta = \int b_u dt = b_u t$$

donde b_u es el bias del giroscopio.

Ejemplo: suponiendo un bias del acelerómetro de $1m/s^2$ y un bias del gyro de $1^o/s$ (aprox. 0.02 rad/s). Si estos bias no son compensados en las mediciones, el bias del acelerómetro generará un error de posición de 50m solamente en 10 segundos, y de 5Km después de 100s. En contraste, el bias del giroscopio generará un error de 27.8 Km en posición después de 100s. Esto muestra que el bias del gyro en el error de posición es más pronunciado que el bias del acelerómetro a medida que introduce un error cúbico creciendo el error de posición.

Factor de escala

EL factor de escala es la razón de la entrada a la salida del sensor. Un error de factor de escala es el error en esta razón después de la conversión de unidades. Puede ser causado por la imperfección en el sensor dentro del ensamblado de la IMU, a menudo se especifica en ppm. Para las IMU de tipo MEM puede llegar a ser tan alto como 2000 ppm. $(0{,}2 \times 10^{-2})$ ó 0.2 % de la salida verdadera.

Ruido

La salida de las IMU MEMS son perturbadas por varias fuentes de ruido, tal como ruido térmico y ruido eléctrico [72]. El ruido de un giroscopio es integrado para producir Angular Random Walk (ARW), y el ruido de los acelerómetros es integrado para producir Random Walk (RW) sobre su solución de velocidad.

Usualmente los fabricantes especifican el ruido en términos de ARW con unidades de grados por hora $(^o/\sqrt{hr})$, algunos la especifican como Power Spectral Density $(^o/hr)^2/Hz$ ó densidad de ruido FFT $(^o/hr/\sqrt{Hz})$.

Ejemplo: El Honeywell HG G1700 tiene un ruido de 0.5 $^o/\sqrt{hr}$, lo cual significa que después de 1 hr la desviación estándar del error de posición será de $0{,}5^o$, después de 2 hrs cerca de $0{,}5^o/\sqrt{2} = 0{,}71^o$, después de 3 hrs. de $0{,}5^o/\sqrt{3} = 0{,}87^o$

Desplazamiento angular

Es el cambio relativo en posición angular.

Velocidad angular

Es la velocidad de cambio de desplazamiento angular con respecto al tiempo.

Heading

Es el cambio en la velocidad lineal. Es la dirección hacia la cual un objeto en movimiento está apuntando.

Inercia

Es la resistencia de un objeto para cambiar su estado de movimiento.

Navegación inercial

Para navegar en un espacio inercial (sin presencia de gravedad), y dada una f_{IB}^{B} y ω_{IB}^{B} conociendo los valores iniciales de velocidad, posición y orientación se puede:

- Obtener velocidad, integrando la aceleración sensada.
- Obtener la posición con una segunda integración .
- Para integrar en la dirección correcta, se necesita la orientación. Ésta se obtiene integrando la velocidad angular sensada.

Micron

Micrómetro, es una millonésima de metro.

Quantization

La operación matemática de truncamiento de un valor decimal de precisión arbitraria a un valor binario fijo con cierta resolución.

Tilt inclination

Es el ángulo de desplazamiento desde una referencia definida, En muchos casos la referencia definida es el horizonte de la Tierra.

Apéndice D

Aritmética computacional

Representaciones numéricas en sistemas digitales

Los sistemas digitales utilizan sistemas numéricos binarios para representar y procesar la información del mundo real, esta información se almacenan en palabras binarias. Una palabra binaria es una secuencia binaria de longitud finita de 1's y 0's. La manera en la cual los componentes de hardware interpretan esta secuencia de 1's y 0's está descrita por un tipo de dato. Los tipos de datos se pueden dividir en datos de punto fijo y datos de punto flotante. Un dato de punto fijo se caracteriza por el tamaño de la palabra en bits, el punto binario y el signo, el punto binario es la forma de como los valores de punto fijo son escalados. Mientras que el tipo de dato de punto flotante se caracteriza por un bit de signo, una fracción (mantisa) y el exponente (IEEE 754-1985). La selección del tipo de dato depende de la aplicación específica, la arquitectura usada y el costo de desarrollo entre otros. Cuando se escoge un tipo de datos se debe considerar los siguientes factores:

- El rango numérico del resultado.
- La precisión requerida del resultado.
- El error de cuantización asociado.
- Las condiciones aritméticas excepcionales.

Aritmética de punto fijo

La aritmética de punto fijo es muy usada en la implementación de sistemas de procesamiento de señal debido a su simplicidad y eficiencia-área comparada con la aritmética de punto flotante. Las

principales ventajas de la aritmética de punto fijo son su alta frecuencia de operación, pequeña área, y bajo consumo de energía. Sin embargo, el rango dinámico de la representación de punto fijo es mucho menor que su equivalente en punto flotante con la misma longitud de palabra. Por lo tanto, para evitar el sobreflujo, los valores de punto fijo deben escalarse. Esto implica que su precisión numérica es reducida debido a la cuantización.

La representación de punto fijo es vulnerable al sobreflujo, sin embargo, con una buena elección del factor de escala y longitud de palabra este problema puede ser evitado. Por lo tanto, una buena elección de estos factores son importantes cuando se están optimizando aplicaciones en punto fijo.

El uso de aritmética de punto fijo introduce ruido inevitable en forma cuantizable (quantization noise). El ruido cuantizable a menudo es modulado por la suma de su señal y una variable aleatoria [73]. Este ruido aditivo es un ruido blanco uniformemente distribuido que no está correlacionado con la señal.

El mínimo número de bits para la parte entera de una señal está determinado por el rango de la señal y los bits para la fracción determinar el grado de precisión de la misma. La suma del número de bits de la parte entera y la fraccionaria determinan el ancho de palabra que satisface la eficiencia de algoritmo en formato de punto fijo. En consecuencia la eficiencia del sistema se puede ver degradada por el efecto de la cuantización del formato de punto fijo elegido.

La representación de números en punto fijo [74] en una palabra binaria, contienen una parte entera y una fraccionaria separadas por un punto denominado binario. La posición fija del punto binario define el rango del valor entero y la fracción. Por ejemplo en un formato 16.8 con signo, la parte entera tiene un rango de -128 a 127 ya que el bit mas significativo es el de signo y la parte fraccionaria contiene 8 dígitos, siendo 2^{-8} el valor mas pequeño de representar.

Ademas para el manejo del signo se pueden clasificar en: magnitud con signo, complemento a 1, complemento a 2 y offset binario. La representación de complemento a 2 es una de las más utilizadas, conocida también como representación de complemento Radix, en donde el valor de una palabra binaria w_d normalizada en representación de complemento a 2 es:

$$x = -x_0 + \Sigma_{i=1}^{w_d-1} x_i 2^{-i} \tag{D.1}$$

Los valores quedan en el rango de $-1 \leq x \leq (1 - Q)$, donde $Q = 2^{w_d+1}$. Hay un numero más negativo que números positivos y el valor +1 no puede ser representado. El valor negativo de un número en representación de complemento a 2 puede obtenerse del correspondiente número positivo agregándose Q al número complementado.

Ejemplo:

$$(+0{,}828125)_{10} = (0{,}110101)_{2C} \tag{D.2}$$

$$(-0{,}828125)_{10} = (1{,}001010)_{2C} + (0{,}000001)_{2C} \tag{D.3}$$

$$= (1{,}001011)_{2C} \tag{D.4}$$

$$(0)_{10} = (0{,}000000)_{2C} \tag{D.5}$$

Características de la aritmética de punto fijo:

- La multiplicación de dos números binarios enteros de n bits genera un resultado con una longitud de $2 * n$ bits.
- La multiplicación de dos números de punto fijo fraccionarios de n bits normalizados da como resultado un valor con longitud de n bits.
- En la representación de complemento a dos la suma y resta pueden ejecutarse sin considerar el signo de los operandos.
- Si la suma cae en rango correcto, se pueden sumar varios números en complemento a 2 a pesar de que la suma pueda temporalmente tener un desbordamiento del rango numérico disponible.
- En la aritmética de punto fijo fraccionaria las oscilaciones parásitas son más fácilmente suprimidas que en la aritmética de punto flotante.
- La aritmética de punto fijo también requiere menos área de chip y es mucho más rápida que la aritmética de punto flotante.
- En la multiplicación con aritmética fraccional de complemento a dos nunca ocurre sobreflujo, sin embargo los errores por redondeo pueden tener efectos de inestabilidad en sistemas retroalimentados.

Aritmética de punto flotante [75]

EL formato de punto flotante es usado para representar un conjunto finito de números reales. En esta representación se puede manejar un rango dinámico mas amplio que en la aritmética de punto fijo. La IEEE ha establecido lineamientos o estándares bajo la norma IEEE Std 754-2008. Las palabras de longitud fija mas usadas son típicamente de 32, 64 y 128 bits.

Un número x de punto flotante binario se representa por el producto de dos números con signo, la mantisa M y el exponente E: $x = M \times 2^E$. Donde 2 es la base del sistema binario. El exponente determina el rango de números que pueden ser representados, y la mantisa la precisión del número. Por ejemplo, si el exponente y la mantisa son representados por 8 y 16 bits respectivamente, el rango de números de punto flotante que puede ser representado en este caso es desde $0{,}5 \times 2^{-128}$ hasta $1 - (2^{-15}) \times 2^{128}$. De los 16 bits usados para representar la mantisa, un bit se usa para el signo.

La precisión de números de punto flotante es 1 en $2^{14}(0{,}61 \times 10^4)$ que es cerca de 4 dígitos decimales. Para mayor información refiérase a [75].

En aritmética de punto flotante, ocurren errores por redondeo tanto en la suma como en la multiplicación mientras los errores por redondeo en punto fijo solo ocurren en la multiplicación.

Comparación de la precisión en una representación decimal contra punto fijo

Dada una fracción decimal, x, consistente de 'd' dígitos, su precisión es $\pm 0{,}5 \times 10^{-d}$. Si representamos el mismo número en binario con B bits, su precisión se convierte en $\pm 0{,}5 \times 2^{-B}$.

Para mantener la misma precisión para las dos representaciones se requiere:

$$0{,}5 \times 10^{-d} = 0{,}5 \times 2^{-B}, \tag{D.6}$$

esto es:

$$B = d\, log_2 10 \approx 3{,}3d\, bits. \tag{D.7}$$

Por ejemplo: suponiendo que el número 0.23456 tiene que ser representado en binario, entonces se requieren $3{,}3 \times 5 = 17\ bits$ para representar la misma precisión. La siguiente tabla muestra la relación entre el número de bits y la precisión en dígitos decimales.

Número de bits	Precisión (número de dígitos decimales)
7	2.1
8	2.4
10	3.0
12	3.6
14	4.2
15	4.5
16	4.8
18	5.4
20	6.1
23	7.0
24	7.3
64	19.4

Complejidad computacional

La complejidad computacional estudia la complejidad inherente a la resolución de un problema computable. Los recursos comúnmente estudiados son el tiempo (mediante una aproximación al número y tipo de pasos de ejecución de un algoritmo para resolver un problema) y el espacio (mediante una aproximación a la cantidad de memoria utilizada para resolver un problema). Los problemas que tienen una solución con orden de complejidad lineal son los problemas que se resuelven en un tiempo que se relaciona linealmente con su tamaño. Los sistemas de cómputo de hoy en día resuelven problemas mediante algoritmos que tienen como máximo una complejidad computacional polinómico lo que significa que la relación entre el tamaño del problema y su tiempo de ejecución es polinómico. La notación $O(f(n))$ se utiliza para indicar la cantidad de operaciones a ejecutar de un algoritmo. Cuando el tiempo de ejecución de un algoritmo es menor que un cierto valor calculado a partir del número de variables implicadas usando una fórmula polinómica, se dice que dicho algoritmo se puede resolver en un tiempo polinómico. Problemas con una complejidad computacional como $3n^2+5n$, o $4n^6+7n^4-2n^2$ son polinómicos pero 2^n no lo es, haciendo al problema intratable computacionalmente para valores grandes de n. Dentro de los tiempos polinómicos, podemos distinguir los logarítmicos $(log(n))$, los lineales (n), los cuadráticos (n^2) y cúbicos (n^3).

Apéndice E

Análisis numérico

Error por redondeo

El error por redondeo es la diferencia entre el valor verdadero de los resultados y el valor aproximado en un sistemas digital.

Estabilidad

Se refiere a la tendencia del cambio del error en un esquema iterativo.

- Es *estable* si el error inicial o errores pequeños en cualquier tiempo permanecen pequeños cuando progresa la iteración.
- Es *inestable* si el error inicial o error pequeño se hace más y más grande en cualquier momento o eventualmente se hace ilimitado.

Estabilidad numérica y robustez

Estos términos son usados para describir propiedades cualitativas de los métodos de solución a problemas aritméticos. La robustez se refiere a la relativa insensibilidad de la solución a errores de algún tipo, y la estabilidad numérica se refiere a la robustez en contra de errores de redondeo.

La solución numérica de la ecuación de Riccati tiende a ser mas robusta en contra de los errores por redondeo si se usa como variable dependiente los factores de Cholesky de la matriz de covarianza [63]. Los métodos numéricos para resolver la ecuación de Riccati en términos de factores de Cholesky son llamados métodos de factorización y su implementación en el filtro de Kalman es llamado "square-root filtering"(metodo de la raíz cuadrada).

Precisión contra estabilidad numérica

Los errores por redondeo se pueden reducir usando más precisión (más bits en la mantisa o en la parte fraccionaria del formato de datos), pero la exactitud del resultado es también influenciada por la exactitud de los parámetros iniciales usados y los métodos de implementación. Matemáticamente, métodos de implementación equivalentes pueden tener estabilidades numéricas muy diferentes con la misma precisión, pero en algunos casos puede depender de las propiedades intrínsecas del problema a resolver.

La asimetría de la matriz de covarianza de la incertidumbre en la estimación de los estados de un sistema es un síntoma de degradación numérica y una causa de inestabilidad numérica, y las medidas para simetrizar el resultado pueden ser beneficiosas [63].

Convergencia y divergencia

En la solución de un algoritmo con un esquema iterativo, la convergencia significa que cuando la iteración está en progreso la solución se acerca al valor verdadero. En cambio si la solución se aleja se le llama divergencia.

Divergencia numérica

La divergencia numérica del filtro de Kalman es a menudo asociada con el cálculo de la matriz de covarianza que pierde su no-negatividad. Por lo tanto es una práctica común intentar preservar su no-negatividad delimitando la diagonal de abajo (para prevenir que el cálculo de la covarianza se haga muy pequeño) y limitar la correlación entre pares de variables.

Estadística de señales con ruido

La probabilidad y estadística se utiliza para caracterizar las señales y los procesos que las generan. Los conceptos más importantes para el análisis estadístico de señales son la media (μ), la desviación estándar (σ), el SNR (signal to noise ratio) el error típico. La variable, N, se usa para representar el número total de muestras en una señal. Para el cálculo de los parámetros media (μ), desviación estándar (σ), razón señal ruido (SNR) y error típico se utilizan las ecuaciones siguientes:

El valor medio (μ) de una señal es la suma de todas las muestras dividido entre el número de ellas (N)

$$\mu = \frac{1}{N}\sum_{i=0}^{N-1} x_i \tag{E.1}$$

La expresión $|x_i - \mu|$ describe que tan lejos la muestra i se desvía de la media. La desviación promedio de una señal se encuentra sumando las desviaciones de todas las muestras individuales, y luego dividiéndolas por el número de muestras, N. Sin embargo este parámetro no es muy usado estadísticamente sino la potencia representada por la desviación de la media, a esto se le llama desviación estándar (σ). Esta se obtiene con la siguiente ecuación:

$$\sigma^2 = \frac{1}{N-1}\sum_{i=0}^{N-1}(x_i - \mu)^2 \tag{E.2}$$

El término, σ^2, en estadística se le conoce comúnmente como varianza. La desviación estándar es una medida de que tan lejos la señal fluctúa de la media. La varianza representa la potencia de ésta fluctuación.

La relación señal-ruido (SNR) es otro término muy usado para caracterizar una señal con ruido y se calcula dividiendo la media por la desviación estándar. Un valor alto de SNR significa mejores datos.

$$SNR = \frac{\mu}{\sigma} \tag{E.3}$$

Para señales aleatorias, el error típico está dado por:

$$error\ tipico = \frac{\sigma}{N^{\frac{1}{2}}} \tag{E.4}$$

Apéndice F

Reglas de diseño en VLSI

Reglas de diseño de layouts [50]

El principal objetivo de las reglas de diseño es construir circuitos funcionales confiables en un área tan pequeña como sea posible. En general las reglas de diseño representan un compromiso entre *eficiencia* y *rendimiento*. Entre más conservadoras sean las reglas de diseño lo más probable es que el circuito funcionará. Sin embargo, mientras más agresivas sean las reglas, mayor será la oportunidad para mejorar la eficiencia y tamaño.

Las reglas de diseño especifican al diseñador ciertas restricciones geométricas en el diseño del layout, para que los patrones procesados en la oblea preserven la topología y geometría del diseño. Es importante tener en cuenta que las reglas de diseño no representan la frontera entre una fabricación correcta e incorrecta. Mas bien, representan una tolerancia que asegura una alta probabilidad de una correcta fabricación y operación subsecuente.

Las reglas de diseño describen las características más pequeñas que pueden ser y qué tan cerca de ellas pueden estar sin fallos en un proceso de manufactura. Las reglas de diseño industriales se especifican usualmente en micrones. Esto hace difícil que se pueda migrar de un proceso a otro más avanzado ó de diferentes procesos de fabricas porque no todas las reglas escalan en la misma forma.

Mead y Conway [76] popularizaron las reglas de diseño escalables basadas en un simple parámetro λ que caracteriza la resolución del proceso. λ es generalmente la mitad de la longitud mínima del canal de un transistor. Esta longitud es la distancia entre la fuente y el drenaje de un transistor y

está definido por el ancho mínimo del hilo de polisilicio. Por ejemplo, un proceso de 180 nm tiene un ancho mínimo de polisilicio de $0{,}18\mu$ y usa reglas de diseño con $\lambda = 0{,}09\mu m$.

Las reglas basadas en lambda son necesariamente conservadoras porque ellas redondean hacia arriba dimensiones para un múltiplo entero de λ. Sin embargo, ellas hacen un escalado de las capas trivial; el mismo layout puede cambiarse a un nuevo proceso simplemente por la especificación de un nuevo valor de λ.

Algunos diseñadores describen a menudo un proceso por su *"feature size"*, que se refiere a la longitud mínima del transistor, así que λ es la mitad del *feature size*.

Las reglas describen el mínimo ancho para evitar rupturas en una línea, mínimo espaciado, sobre (*overlaps*) para asegurar que dos capaz se encimen completamente.

Ejemplo:

- Ancho mínimo y espaciado del metal y difusión de 4 λ
- Contactos son de $2\lambda \times 2\lambda$ y deben ser rodeadas por 1λ sobre la capa superior e inferior.
- El polisilicio debe tener un ancho de 2λ.
- El polisilicio cubre la difusión por 2λ donde se desea que un transistor tenga un espaciado de 1λ de igual forma donde no se desee un transistor.
- El polisilicio y contactos tienen un espaciado de 3λ de otros polisilicio o contactos.
- Los pozos *Nwell* rodean un transistor *pMOS* por 6λ y evaden a los transistores *NMOS* por 6λ

En un proceso de 3 metales, el ancho del tercer metal es típicamente 6λ y el espaciado 4λ. En general, procesos con más layers proveen capas de metal más gruesas y anchas que tienen resistencias más bajas. Las dimensiones de los transistores se especifican a menudo por su ancho/largo(W/L. Por ejemplo, el transistor *NMOS* de la fig. F.1 tiene un W/L de 4/2.

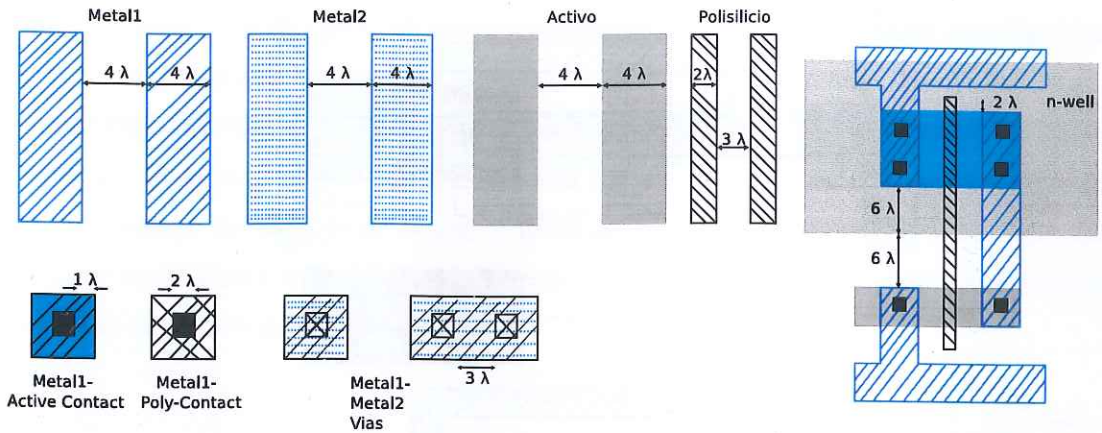


Figura F.1: Reglas de diseño basadas en λ.

Los transistores *PMOS* son más anchos que los *NMOS* debido a que los hoyos se mueven más lentamente que los electrones, esto es, el transistor es más ancho para trasferir la misma corriente. La fig. F.2 muestra un layout de un inversor *CMOS*.

Reglas para la capa nwell

La capa *nwell* (pozo n) es usualmente un implante más profundo que los implantes en el drenaje y fuente del transistor, y por lo tanto, es necesario proveer suficiente espacio entre bordos del *nwell* y las difuciones n+ adyacentes. La separación entre el bordo del pozo y la difusión está determinada por la transición del óxido de campo a través del límite del pozo.

Debido a que la resistencia del pozo n puede ser de varios Kohms por cuadrado, es necesario aterrizar el pozo al fondo permitiendo un número suficiente de contactos de pozo (*well taps*). Esto previene excesivas caídas de tensión debido a corrientes de pozo.

Reglas para transistores

Los transistores *CMOS* son definidos generalmente por al menos cuatro máscaras. Estas son *activo* (también llamada *difussion*, *diff*, *thinox*, *OD*, o *RX*), *n-select* (también llamada *n-implant*, *nimp*, o *nplus*), *p-select* (también llamado *p-implant*, *pimp*, o *pplus*) y *polisilicio* (también llamado *poly*, *polyg*, *PO*, ó *PC*). La máscara de activo define todas las áreas donde se colocarán difusiones tipo p- o n- ó donde se colocarán las compuertas de los transistores. Las compuertas del transistor

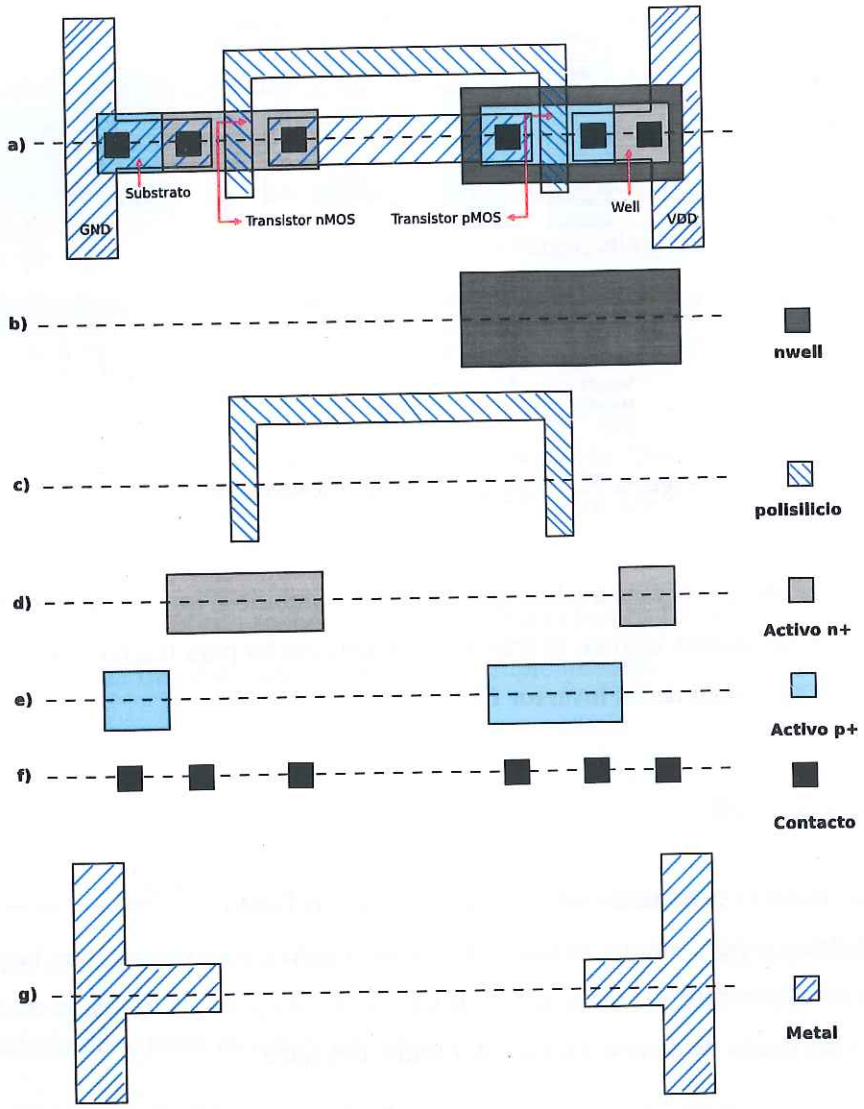


Figura F.2: Inversor *CMOS*.

están definidas por las operaciones AND de la máscara del polisilicio y la activa, donde cruza el polisilicio y la difusion).

Las capas *Select* define que tipo de difusión se requiere. *n-select* redondea regiones activas donde se requiere una difusión tipo n y *p-select* donde se requiere una tipo p.

Una área *n-diffusion* dentro de una región *p-well* define un transistor n-MOS. El área de difusión n dentro de un pozo n define un contacto n-well. Además las áreas *p-diffusion* dentro de los *n-well* definen los contactos de substrato (*pwell contact*). Finalmente, los diseñadores de sistemas definen solamente la difusion n (*ndif*) y difusión p (*pdiff*) para reducir la complejidad del proceso.

Es esencial para el *poly* cruzar el activo completamente; de otra forma el transistor que se crea se cortocircuitará por una ruta de la difusión entre fuente y drenaje. Por lo tanto, el *Poly* se requiere extenderlo más allá de los límites de la área activa. Este aumento es referenciado como *gate extension*. El Activo debe extenderse más allá del *poly* para que así las regiones de la fuente y el drenaje existen llevando carga dentro y fuera del canal. Las regiones activo y poly que no deben formar un transistor deben mantenerse separadas; esto resulta de una regla de espaciamiento (*spacing rule*) del activo al *poly* (fig. F.3).

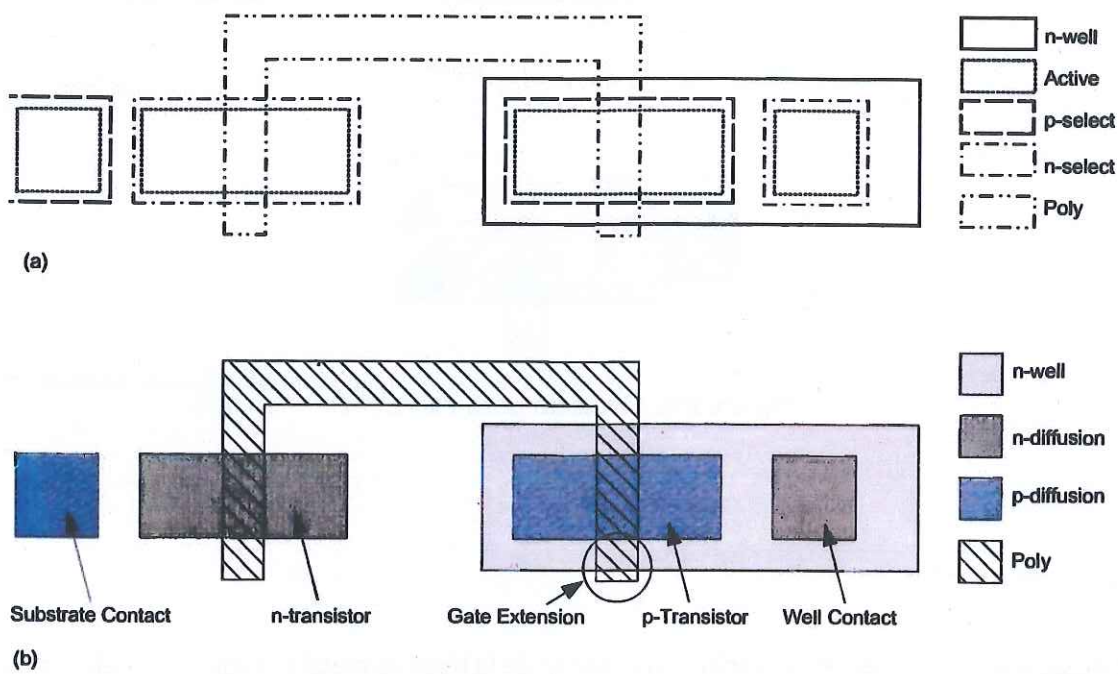


Figura F.3: Proceso de construcción de un transistor en un pozo n.

Reglas para contactos

Hay varios contactos disponibles:

- Metal a *p-active* (*p-diffusion*)
- Metal a *n-active* (*n-diffusion*)
- Metal a *polysilicon*
- Metal a *well* o *substrato*

Dependiendo del proceso, otros contactos como *buried polysilicon-active contacts* pueden permitirse para interconexiones locales. Debido a que el substrato esta dividido dentro de regiones pozo, cada pozo aislado debe conectarse a la alimentación de tensión apropiada, por ejemplo, el pozo n debe conectarse a VDD y el substrato a GND con contactos *well* o de substrato. Un metal hace una pobre conexión con un substrato o pozo ligeramente dopado, por lo tanto, una región activa altamente dopada se coloca debajo del contacto fig. F.4 Cuando sea posible, use más de un contacto en cada conexión. Esto mejora significativamente el rendimiento en muchos procesos debido a las conexiones hechas aún cuando haya malformaciones en contactos.

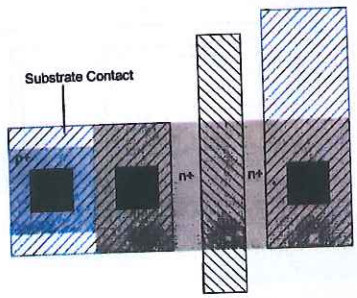


Figura F.4: Contacto pozo-substrato.

Reglas para metales

El espaciado de metales puede variar con el ancho de la línea de metal (también llamado *fat-metal* rules). Esto es sobre los anchos de hilos de metal, el mínimo espaciado puede incrementarse. Esto es debido a características del proceso *etch* de hilos pequeños contra largos. También puede haber reglas para el ancho máxima del metal. Esto es, hilos de metal simples no pueden ser más grandes que cierto ancho. Si se necesitan hilos más anchos y agregando enlaces del tablero. Adicionalmente puede haber reglas de espaciado que se aplican a longitud, cercanía de lineas paralelas de metal.

Reglas para vías

Los procesos pueden variar si permiten vías *stacked* que sean colocadas sobre regiones de polisilicio y difusion. Algunos procesos permiten que las vías sean colocadas dentro de estas áreas, pero no permiten al vías espaciarse dentro del límite del polisilicio o difusión. Esto resulta de variaciones topológicas repentinas que ocurren en los límites de subcapas. Procesos modernos de

planarizado permiten vías apiladas, lo cual reduce el área requerida para pasar de un metal de un nivel bajo a uno más alto.

Otras reglas

La pasivación o capa de *overglass* es una capa protectora de SiO_2(vidrio) que cubre el chip final. Se requieren tamaños apropiados de apertura en pads y cualquier otro punto interno de prueba.

Algunas reglas adicionales que pueden presentar algunos procesos son los siguientes:

- Extensión del polisilicio o metal más allá de un contacto o vía.
- Diferentes extensiones al *poly* de puerta dependiendo de la longitud del dispositivo.
- Máximo ancho de un *feature*.
- Mínimo ancho de un *feature*
- Mínimo tamaño muesca.

Apéndice G

Proceso de fabricación

Características del proceso C5

On Semiconductor es una empresa de semiconductores que ofrece servicios de fabricación de circuitos integrados, la Fig. G.1 muestra la variedad de procesos que actualmente ofrece.

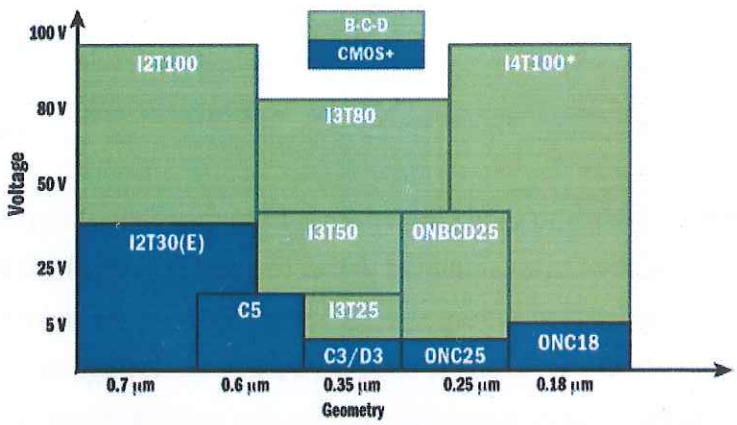


Figura G.1: Representación de los procesos de *ON Semi*.

Capas del proceso *ON Semiconductor C5*

En la Tabla G.1 se presentan las capas (o layers) que incorporan el diseño de un chip en el proceso *ON Semi* C5 con dimensión característica de canal de 0.5 μm al fabricarse (y al dibujarse de 0.6 μm), la cual permite hasta 3 capas de metal, y 2 capas de *polisilicio*.

Tabla G.1: Capas principales del proceso *On Semi*.

No.	*Capas (Layers)*	No.	*Capas (Layers)*
1	Pozo-N (N-WELL)	10	VIA 1 (Conexión Metal1-Metal2)
2	Región activa tipo n o tipo p) (ACTIVE)	11	METAL 2
3	Contacto (CONTACT)	12	VIA 2 (Conexión Metal2-Metal3)
4	Polisilicio (POLY)	13	METAL 3
5	Contacto a polisilicio (POLY CONTACT)	14	POLY 2
6	Dopado tipo n (N-SELECT)	15	POLY 2 CONTACT
7	Dopado tipo p (P-SELECT)	16	HIGH RESISTENCE (HRP)
8	Contacto a activo (ACTIVE CONTACT)	17	OVERGLASS
9	METAL 1	18	Ubicación del pad (PAD COMMENT)

Las características más importantes del proceso C5 se definen en la Fig. G.2. Dentro de las más importantes se encuentra un voltaje de operación de 5 y 12V, un espesor del oxido de compuerta de 13.5 nm, 3 metales, densidad de 4.2K $gates/mm^2$ con un consumo de 1.58 $\mu W/MHz/gate$.

Features

- 2 or 3 metal layers
- Poly to poly capacitors
- EEPROM
- Schottky diodes
- High voltage I/O – 12/20 V
- High-resistance poly
- Low-voltage modules

Process Characteristics

Operating Voltage	5, 12 V
Substrate Material	P-Type, Bulk
Drawn Transistor Length	0.6 µm
Gate Oxide Thickness	13.5 nm
Contact/Via Size	0.5 µm
Contacted Gate Pitch	3.9 µm
Top Metal Thickness	675 nm
Contacted Metal Pitch	
Metal 1	1.5 µm
Metal 2, 3	1.6 µm
Metal Composition	TiN/AlCu/TiN

Digital Design
High Performance Core
4.2 K gates/mm² *
1.58 µW/MHz/gate
103 ps gate delay (2 Input NAND, fanout = 2)
Tall Pads for high I/O count designs
86 µm in-line pad pitch
60 µm staggered pad pitch
558 µm pad height

Figura G.2: Características del proceso C5 de *ON Semiconductor*.

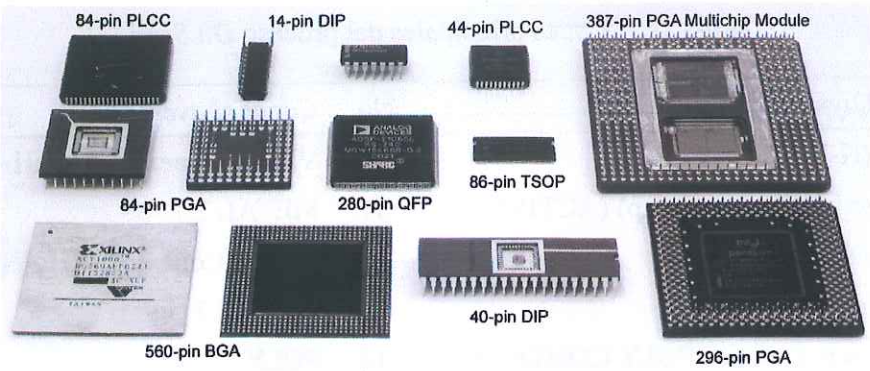
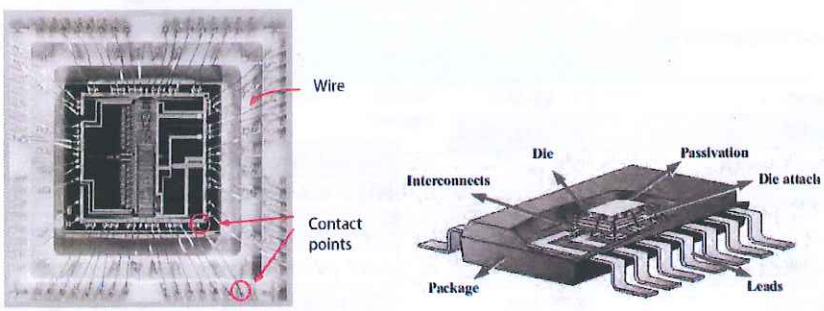


Figura G.3: Tipos de encapsulados de CI's.

Los tipos más comunes de encapsulados de encapsulados de CI se muestran en la Fig. G.4a. Dentro de estos encontramos los formatos de tecnología through-hole como son los DIP, PGA. También existen una variedad de formatos para tecnología de montaje superficial como son PLCC, BGA y TSOP entre los más usuales.

Hay dos maneras de realizar las inter-conexiones del circuito integrado, *wire bonding* y *flip-chip bonding* para este trabajo solo se detalla la técnica *wire bonding*. Unión de alambre (en inglés *wire bonding*) [77], es la técnica que utiliza alambre delgado y una combinación de calor, presión, y/o energía ultrasónica para hacer la interconexión entre el dado y el empaquetado. Los alambres (interconexiones) están hechos de oro, aluminio o cobre aproximadamente de un diámetro de 15 μm. En la Fig. G.4 se muestra el ejemplo de este tipo de técnica.



(a) Dado en el interior del encapsulado. **(b)** conexiones de oro.

Figura G.4: Bonding.