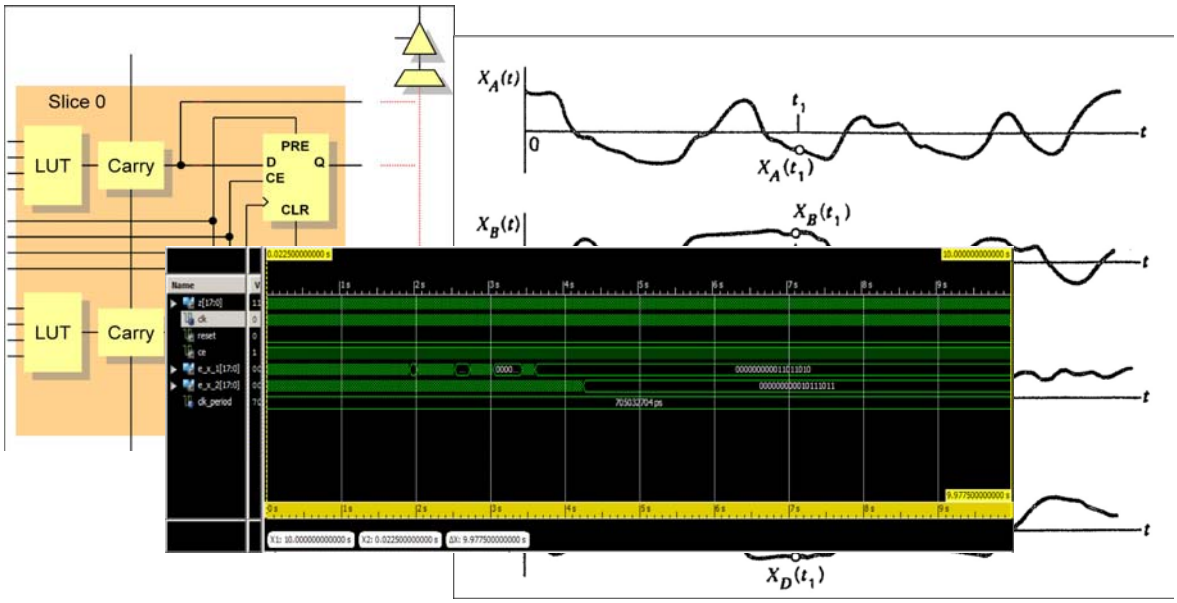
Editorial de la Universidad
Tecnológica Nacional

# Implementación de filtro de Kalman en FPGA




Trabajo final de carrera:

**Darío J. Dematties Y Francisco A. Iglesias**

Jurado evaluador:

**Ing. Rodrigo Gonzalez – Ing. Eduardo Serdoch**

Departamento de Electrónica  Cátedra de Proyecto Final

Facultad Regional Mendoza
Universidad Tecnológica Nacional – U.T.N.
Argentina

Editorial de la Universidad Tecnológica Nacional – edUTecNe
http://www.edutecne.utn.edu.ar
edutecne@utn.edu.ar

# Índice

# Resumen

En el presente proyecto se implementó el algoritmo de Bierman-Thornton del Filtro de Kalman (KF) en un Arreglo de Compuertas Programables en Campo (FPGA) operando en punto fijo.

El KF procesa mediciones ruidosas de un sistema lineal perturbado por ruido aleatorio Gaussiano a fin de estimar su estado interno. Este algoritmo de estimación, constituye en la actualidad una herramienta fundamental en una amplia cantidad de aplicaciones en la industria, que van desde sistemas de navegación inercial hasta procesamiento de imágenes.

La implementación del KF en FPGA es de relevancia debido a su gran capacidad de cómputo paralelo y bajo costo de desarrollo. El presente trabajo posee un valor estratégico debido a que desarrolla el Know-How necesario para permitir la implementación de este tipo de conocimiento avanzado en desarrollos e investigaciones tecnológicas.

Se elaboró documentación detallada sobre los fundamentos teóricos del KF y las técnicas numéricas que permiten su funcionamiento en sistemas digitales. Se desarrolló software que permite realizar el análisis de estabilidad numérica del filtro operando en punto fijo. Se sintetizó el KF en una FPGA Xilinx y se evaluó su desempeño temporal y numérico, obteniéndose un factor de mejora de 11.28 y una tasa de procesamiento de 2.29 [MSamples/s]

Palabras clave: Procesamiento de señales - Filtro de Kalman - Análisis Numérico - FPGA.

# 1 Abreviaturas

**CLB** Bloque lógicos configurable, del Inglés Configurable Logic Block

**FP** Punto Fijo, del Inglés Fixed Point

**KF** Filtro de Kalman, del Inglés Kalman Filter

**KF-BT** Kalman Filter - Bierman Thornton

**KF-OA** Filtro de Kalman-Algoritmo Original, del Inglés Kalman Filter - Original Algorithm

**FPGA** Arreglo de compuertas programable en campo, del Inglés Field Programable Gates Array

**KG** Ganancia de Kalman, del Inglés Kalman Gain

**IC** Condiciones Iniciales, del Inglés Initial Conditions

**LTI** Lineal invariante en el tiempo, del Inglés Linear Time Invariant

**LTV** Lineal variante en el tiempo, del Inglés Linear Time Variant

**LDE** Ecuación diferencial lineal, del Inglés Linear Differential Equation

**LS** Sistema dinámico lineal, del Inglés Linear System

**LUT** Tabla de entrada-salida, del Inglés Look-Up Table

**MSE** Error cuadrático medio, del Inglés Mean Squared Error

**PSD** Densidad espectral de potencia, del Inglés Power Spectral Density

**PLD** Dispositivo lógico programable, del Inglés Programable Logic Device

**RP** Proceso aleatorio, del Inglés Random Process

**RV** Variable aleatoria, del inglés Rnadom Variable

**STM** Matriz de Transición de estados, del Inglés State Trasition Matrix

**VHDL** Lenguaje de descripción de hardware para muy grandes escalas de integración, del Inglés Very large scale Hardware Description Language

# 2 Simbología matemática y definición de constantes

## 2.1 Notación estadística

$x(t)$ Proceso aleatorio de tiempo continuo.

$x[n]$ Proceso aleatorio de tiempo discreto o secuencia aleatoria.

$X(t_0)$ Variable aleatoria continua correspondiente al muestreo del proceso $x(t)$ en $t = t_0$.

$p[X(t_0)]$ Función de densidad de probabilidad de la RV $X(t_0)$.

$p[X(t_0), X(t_1)]$ Función de densidad de probabilidad conjunta de las variables $X(t_0)$ y $X(t_1)$.

$R_x(t_1, t_2)$ Función de autocorrelación del RP $x(t)$.

$N(\mu, \sigma)$ Función de densidad de probabilidad Normal o Gaussiana de media $\mu$ y desviación estándar $\sigma$.

## 2.2 Notación especial en espacio de estados

$T$ Intervalo de muestreo

$x$ Vector de estado.

$x_k$ Escalar, que es el *k*-ésimo componente del vector $x$.

$x_k$ Vector que es el *k*-ésimo elemento de la secuencia ... , $x_{k-1}, x_k, x_{k+1}$, ... de vectores.

$\hat{x}$ Estimación del valor de $x$.

$\hat{x}_k^-$ Estimación a priori del vector $x_k$, condicionada al total de las mediciones anteriores con excepción de la obtenida en el tiempo $t_k$.

$\hat{x}_k$ Estimación a posteriori del vector $x_k$, condicionada al total de las mediciones disponibles al tempo $t_k$.

$\dot{x}$ Derivada de $x$ con respecto al tiempo $t$.

## 2.3 Simbología estándar en filtrado de Kalman

$F$ Matriz de coeficientes dinámicos de una LDE que define un sistema dinámico.

$G$ Matriz de acoplamiento entre el ruido aleatorio del proceso y el estado de un LS.

$H$ Matriz de sensibilidad de medición que define la relación lineal entre el estado de un LS y las mediciones que pueden ser realizadas.

$\overline{K}$ Matriz de la KG.

$P$ Matriz de covarianza del error de estimación.

$Q$ Matriz de covarianza de ruido del proceso.

$R$ Matriz de covarianza del error de medición u observación.

$x$ Vector de estados internos de un LS.

$z$ Vector (o escalar) de los valores medidos.

$\Phi$ Matriz de transición de estados de un LS discreto.

### 2.3.1 Dimensiones de matrices y vectores.

Vectores

| | | |
|---|---|---|
| $x$ | Estado del sistema | $n$ |
| $w$ | Ruido de proceso | $r$ |
| $u$ | Entrada de control | $r$ |
| $z$ | Medición | $l$ |
| $v$ | Ruido de medición | $l$ |

Matrices

| | | |
|---|---|---|
| Φ | Transición de estado | $nxn$ |
| $G$ | Acoplamiento del ruido de proceso | $nxr$ |
| $Q$ | Covarianza del ruido del proceso | $rxr$ |
| $H$ | Sensibilidad de medición | $lxn$ |
| $R$ | Covarianza del ruido de medición | $lxl$ |

## 2.4 Notación especial en aritmética de punto fijo

Para mantener una compatibilidad con las fuentes referenciadas se adopta el punto como indicador de la posición de la unidad decimal.

$O(IL, FL)$ Palabra binaria interpretada en una aritmética de punto fijo (con signo) con parte entera de longitud $IL$ y parte fraccional de longitud $FL$.

$C(L)$ Contenido de una palabra binaria de longitud $L$.

$D(IL, FL)$ Valor en decimal de una palabra binaria con parte entera de longitud $IL$ y parte fraccional de longitud $FL$.

# 3 Marco teórico

Para el desarrollo del presente capitulo se consultó principalmente la bibliografía dada en [**1**] y en [**2**].

## 3.1 Fundamentos estadísticos

### 3.1.1 Procesos aleatorios

Un enfoque intuitivo, que facilita la descripción de los procesos *aleatorios (RPs)*, se logra desde la diferenciación de estos con los procesos *determinísticos*.

En este contexto se entiende por proceso a toda regla que asigna una cantidad a una variable indeterminada en un instante de tiempo dado. Sin embargo, por razones de nomenclatura se conviene en llamar *proceso* a toda variable cuyos valores son asignados mediante la ejecución de dicha regla en el tiempo.

En los procesos determinísticos, su valor es determinado directamente por una regla de asignación y una o más variables no aleatorias. Un ejemplo de proceso determinístico es:

$$x(t) = \sin(2\pi t)$$

En este caso en particular, el proceso $x$ varía con el tiempo. Otro ejemplo en el cual la evolución del proceso es independiente con respecto al tiempo es:

$$x(t) = 17$$

Por otro lado en los RPs (o *estocásticos*), su valor es determinado mediante una regla de asignación y una o más variables aleatorias. Debido a esto solo se conocen de manera determinística las propiedades estadísticas del proceso aleatorio. Un ejemplo de un RP en el que su estadística varía con el tiempo es:

$$x(t) = A\sin(2\pi t) + B$$

Donde tanto $A$ como $B$ son variables aleatorias con densidades de probabilidad conocidas. Esta regla establece que para un instante de tiempo dado, tanto la fase como la amplitud del proceso $x$ están gobernadas por los valores de $A$ y $B$ resultantes del experimento estadístico realizado en ese instante.

Un ejemplo de RP que evoluciona de manera independiente del tiempo es:

$$x(t) = \mathrm{A}$$

**Ecuación 3-1**

Notar que esto no implica que el valor de $x$ sea constante con el timepo pues, para cada instante se debe ejecutar un experimento distinto por lo que la variable aleatoria (RV) $A$ puede tomar valores distintos. En la Figura 3-1 se muestran diferentes ejecuciones de este proceso, llamadas *realizaciones.* Estas deben ser ejecutadas al mismo tiempo. En la figura también se grafican diferentes muestras tomadas para el instante de tiempo $t_1$, estas son:

$X_a(t_1), X_b(t_1), X_c(t_1)$ *y* $X_d(t_1)$. Las mismas constituyen diferentes muestras de una misma RV $X(t_1)$ (pues su densidad de probabilidad es la misma), si elegimos otro instante tendremos una RV diferente digamos $X(t_2)$. Con esto podemos visualizar como un *RP constituye un conjunto infinito de variables aleatorias* relacionadas por un mismo experimento estadístico conceptual. La densidad de probabilidad conjunta de dos variables aleatorias de un mismo proceso nos da información sobre la variabilidad temporal del mismo. Esto puede verse al tomar dos variables en dos instantes de tiempo muy cercanos $X(t_1)$ y $X(t_1 + \Delta t)$, si el proceso tiene una alta variabilidad respecto a $\Delta t$ es esperable que ambas variables sean estadísticamente independientes o sea que: $p[X(t_1), X(t_2)] = p[X(t_1)]\, p[X(t_2)]$.

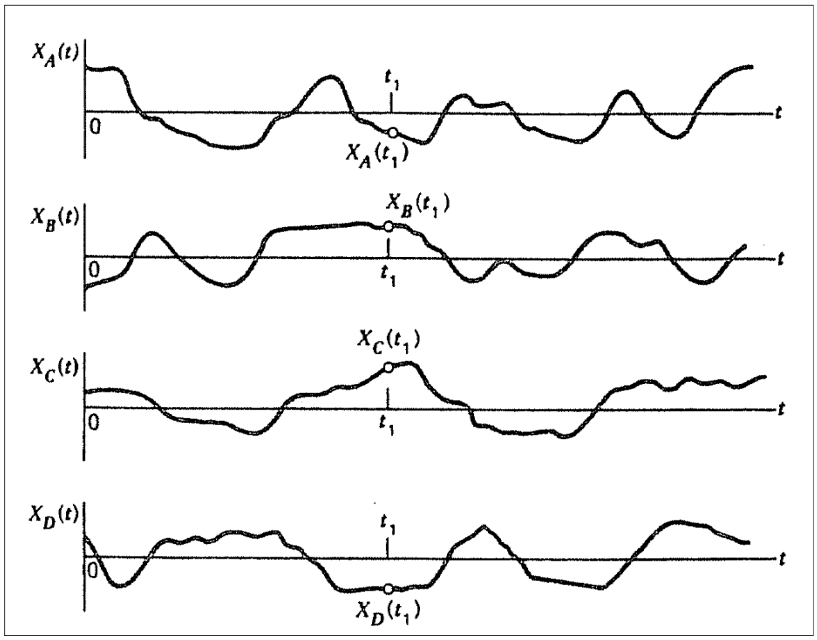



**Figura 3-1: Diferentes realizaciones del RP establecido por la Ecuación 3-1**

Todos los RPs arriba ejemplificados son *procesos llamados de tiempo continuo*, pues la variable tiempo evoluciona de manera continua.

Antagónicamente existen los llamados *RPs discretos o secuencias discretas*, para los cuales la variable tiempo solo adopta valores dados por un múltiplo entero de una cantidad fija (usualmente el tiempo de muestreo).

Un RP se dice *estacionario* si sus funciones de probabilidad no cambian con el tiempo. Esto es si tomamos un conjunto de variables aleatorias: $X(t_1), X(t_2), \dots, X(t_k)$ y un conjunto de variables desplazadas en el tiempo: $X(t_1 + \Delta t), X(t_2 + \Delta t), \dots, X(t_k + \Delta t)$. Entonces las funciones de densidad del primer conjunto deben ser iguales a las del segundo. Conceptualmente esto implica que la evolución estadística del proceso es independiente del momento en el que se comience su ejecución.

Un RP se dice *ergódico* si los parámetros estadísticos de su evolución temporal son iguales a los parámetros estadísticos obtenidos mediante sus muestras en un único instante de tiempo. Esto es que la densidad de probabilidad del proceso calculada temporalmente es la misma que la densidad de cualquiera de sus variables aleatorias.

Las definiciones anteriores son extrapolables a los procesos aleatorio vectoriales. Estos son procesos de carácter vectorial donde cada componente es un RP escalar. Por lo tanto su función de densidad de probabilidad es también vectorial.

NOTA: De aquí en adelante la notación utilizada para los RPs escalares $x(t)$, se utilizara para indicar un *RP vectorial.* El mismo criterio se aplicará a $p[x(t)]$.

### 3.1.2 Media

Se define al momento estadístico de primer orden o media del RP $x(t)$ a la cantidad dada por:

$$E\langle x(t)\rangle = \int_{-\infty}^{\infty} x(t)p[x(t)]\ dx(t)$$

**Ecuación 3-2**

Si el RP en cuestión, es ergódico, entonces puede calcularse la media sin necesidad de disponer de la función $p[x(t)]$ explícitamente, si no, computando el promedio temporal:

$$E\langle x(t)\rangle = \lim_{T\to\infty} \frac{1}{T}\int_{0}^{T} x(t)\ dt$$

**Ecuación 3-3**

### 3.1.3 Autocorrelación y correlación cruzada

Se define la *autocorrelación* de un RP $x(t)$ como:

$$R_x(t_1, t_2) = E\langle x(t_1)x^T(t_2)\rangle = \iint_{-\infty}^{\infty} x(t_1)\ p[x(t_1), x(t_2)]\ x^T(t_2)\ dx(t_1)\ dx(t_2)$$

**Ecuación 3-4**

Si el RP es estacionario, su autocorrelación no depende de $t_1$ ni de $t_2$ si no únicamente de su diferencia $\tau = t_2 - t_1$:

$$R_x(\tau) = E\langle x(t)x^T(t+\tau)\rangle$$

**Ecuación 3-5**

Si además el RP es ergódico, su autocorrelación puede obtenerse de:

$$R_x(\tau) = \lim_{T\to\infty} \frac{1}{T}\int_{0}^{T} x(t)x(t+\tau)\ dt$$

**Ecuación 3-6**

Sin embargo si el RP es no ergódico, será necesario distinguir entre la función de autocorrelación temporal dada por la Ecuación 3-7 y la función de autocorrelación definida por la Ecuación 3-4.

Se define además, la *correlación cruzada* de dos RPs $x(t)$ e $y(t)$:

$$R_{xy}(t_1, t_2) = E\langle x(t_1) y^T(t_2)\rangle = \iint_{-\infty}^{\infty} x(t_1)\, p[x(t_1), y(t_2)]\, y^T(t_2)\, dx(t_1)\, dy(t_2)$$

**Ecuación 3-7**

### 3.1.4 Autocovarianza

En algunos desarrollos, incluido este trabajo, se utiliza la *autocovarianza* del RP $x(t)$ en vez de su autocorrelación, definida como:

$$Cov(t_1, t_2) = E\langle [x(t_1) - E\langle x(t_1)\rangle][x(t_2) - E\langle x(t_2)\rangle]^T\rangle$$

**Ecuación 3-8**

Notar que si el RP es de media nula, entonces:

$$Cov(t_1, t_2) = R_x(t_1, t_2)$$

**Ecuación 3-9**

### 3.1.5 Densidad espectral de potencia

Del estudio de la relación que existe entre un RP $x(t)$ y su función de autocorrelación $R_x(\tau)$, puede verse como esta ultima nos brinda gran cantidad de información útil sobre la variabilidad temporal de $x(t)$. Sin embargo, $R_x(\tau)$ posee la ventaja de ser una cantidad determinística. Esta relación, promueve utilizar a $R_x(\tau)$ para analizar y tratar formalmente problemas en los que interviene $x(t)$.

Debido a lo anterior, resulta de interés estudiar el contenido armónico de la función de autocorrelación de un RP. Esto se hace mediante el cálculo de la transformada de Fourier de $R_x(\tau)$, cantidad denominada Densidad Espectral de Potencia o PSD:

$$S_x(jw) = \mathcal{F}[R_x(\tau)] = \int_{-\infty}^{\infty} R_x(\tau)\, e^{-jw\tau}\, d\tau$$

**Ecuación 3-10**

La semejanza en el nombre proviene de la analogía con la transformada de una señal temporal. Sin embargo debe tenerse cierta precaución pues, si tenemos un RP estacionario este no es absolutamente integrable y por ende su transformada no está definida. Si aplicamos sin embargo una ventana temporal de ancho $T$ a $x(t)$ para obtener su versión truncada $x_T(t)$, es posible estimar su densidad espectral y demostrar el siguiente teorema fundamental:

$$\lim_{T\to\infty} E\,\langle \frac{1}{T} |\mathcal{F}[\, x_T(t)\,]|^2\rangle = S_x(jw)$$

**Ecuación 3-11**

Esta importante relación refuerza la utilización de $R_x(\tau)$ y $S_x(jw)$ como herramientas con las cuales es posible operar de manera determinística en los problemas donde intervienen $x(t)$ y $\mathcal{F}[\, x_T(t)\,]$.

Este teorema constituye la piedra angular de la teoría de Shaping filters. En esta se utiliza la Ecuación 3-11 para demostrar que para un sistema LTI con respuesta en frecuencia $G(jw)$ excitado por un RP con PSD $S_x(jw)$, la PSD del RP de salida $S_y(jw)$ es:

$$S_y(jw) = |G(jw)|^2 S_x(jw)$$

Mediante este resultado y sabiendo que la PSD del ruido blanco es 1, puede utilizarse un sistema LTI excitado por ruido blanco para generar un RP con la PSD que se desee $S_y(jw)$ con tal de hacer:

$$|G(jw)|^2 = S_y(jw)$$

NOTA: Todas las expresiones desarrolladlas para RPs de tiempo continuo en la sección 3.1 tienen su par análogo para el caso de RPs de tiempo discreto.

## 3.2 Modelado por estados de sistemas lineales con entradas estocásticas

### 3.2.1 Tiempo continuo

Como es sabido, la herramienta matemática natural para el modelado de sistemas lineales en tiempo continuo es la ecuación diferencial lineal o LDE. Así, en general un sistema LTV de $n$-esimo orden será modelado por una LDE de orden $n$ a coeficientes variables.

Sin embargo para los fines de control del sistema, resulta más práctico (en especial para sistemas de un orden elevado) el modelado del mismo mediante un sistema de $n$ LDE de primer orden (para el caso de una única salida).

Considerarse un sistema LTV con $r$ entradas de control ruidosas, modeladas por la suma de una señal de control determinística $u(t)$ más una señal de ruido aleatoria $w(t)$ y $l$ salidas $z(t)$, modelable en un espacio de $n$ estados. Su modelo en variables de estados es:

$$\dot{x}(t) = F(t)x(t) + C(t)u(t) + G(t)w(t)$$

**Ecuación 3-12**

$$z(t) = H(t)x(t) + v(t) + D(t)u(t)$$

**Ecuación 3-13**

La Ecuación 3-12 se denomina ecuación de estados y la Ecuación 3-13 se denomina ecuación de salida, en ellas:

$x(t)$ Vector de estados internos, con dimensiones $nx1$.

$u(t)$ Vector de entrada de control determinístico, con dimensiones $rx1$.

$z(t)$ Vector de salida o de medición, con dimensiones $lx1$.

$v(t)$ RP blanco de ruido de medición, con media nula y dimensiones $lx1$.

$R(t)$ Covarianza de $v(t)$. Tal que, $E\langle v(t_1)v(t_2)^T\rangle = R(t_1)\delta(t_2 - t_1)$. Con dimensión $lxl$

$w(t)$ RP blanco de ruido de la planta, con media nula y con dimensiones $rx1$.

$Q(t)$ Covarianza de $w(t)$. Tal que, $E\langle w(t_1)w(t_2)^T\rangle = Q(t_1)\delta(t_2 - t_1)$. Con dimensión $rxr$

$F(t)$ Matriz de coeficientes dinámicos variante en el tiempo, con dimensiones $nxn$.

$C(t)$ Matriz de acoplamiento de entrada variante en el tiempo, con dimensiones $nxr$.

$G(t)$ Matriz de acoplamiento de ruido del proceso variante en el tiempo, con dimensión $nxr$

$H(t)$ Matriz de sensibilidad de medición variante en el tiempo, con dimensión $lxn$

$D(t)$ Matriz de acoplamiento de salida variante en el tiempo, con dimensión $lxr$

Por razones de sencillez, se elimina la variable independiente $t$ de la notación, con esto se supone todas las cantidades variantes con el tiempo salvo que se indique lo contrario. Además se considerará nula a la matriz $D(t)$, por lo que se operará en base a las expresiones:

$$\dot{x} = Fx + Cu + Gw$$

**Ecuación 3-14**

$$z = Hx + v$$

**Ecuación 3-15**

### 3.2.2 Tiempo discreto

Análogamente, un sistema LTV de tiempo discreto de orden $n$ podrá ser modelado ya sea mediante una ecuación en diferencias de $n$-esimo orden o mediante un sistema de $n$ ecuaciones en diferencia de primer orden.

Mediante el segundo paradigma el modelo de estados de un sistema LTV con $r$ entradas de control ruidosas, modeladas por la suma de una señal de control determinística $u[k]$ más una señal de ruido aleatoria $w[k]$ y $l$ salidas $z[k]$, es:

$$x[k+1] = \Phi[k]x[k] + C[k]u[k] + G[k]w[k]$$

**Ecuación 3-16**

$$z[k] = H[k]x[k] + v[k] + D[k]u[k]$$

**Ecuación 3-17**

Donde:

$x[k]$ Vector de estados internos, con dimensiones $nx1$.

$u[k]$ Vector de entrada de control determinístico, con dimensiones $rx1$.

$z[k]$ Vector de salida o de medición, con dimensiones $lx1$.

$v[k]$ Secuencia blanca de ruido de medición, con dimensiones $lx1$.

$R[k]$ Covarianza de $v[k]$. Tal que, $E\langle v[k_1]v[k_2]^T\rangle = R[k_1]\delta[k_2 - k_1]$. Con dimensión $lxl$

$w[k]$ Secuencia blanca de ruido de proceso, con dimensiones $rx1$.

$Q[k]$ Covarianza de $w[k]$. Tal que, $E\langle w[k_1]w[k_2]^T\rangle = Q[k_1]\delta[k_2 - k_1]$. Con dimensión $rxr$

$\Phi[k]$ Matriz de transición de estados variante en el tiempo, con dimensiones $nxn$.

$C[k]$ Matriz de acoplamiento de entrada variante en el tiempo, con dimensiones $nxr$.

$G[k]$ Matriz de acoplamiento de ruido del proceso variante en el tiempo, con dimensiones $nxr$.

$H[k]$ Matriz de sensibilidad de medición variante en el tiempo, con dimensiones $lxn$.

$D[k]$ Matriz de acoplamiento de salida variante en el tiempo, con dimensiones $lxr$.

Por razones de sencillez, se denotará al valor de cualquiera de estas cantidades en el instante $k$-esimo mediante el subíndice $k$ (por ejemplo $x_k$). *Aquí la notación será ambigua entre, la $k$-esima componente de un vector $x_k$ y su valor en el $k$-esimo instante, sin embargo, dicha ambigüedad desaparece al considerar el contexto de cada expresión en particular.*

Por razones que se explicarán en la sección 3.2.3, es conveniente renombrar los términos $C[k]u[k]\ y\ G[k]w[k]$, como $u_k y\, w_k$ respectivamente. Ademas, se considerará nula a la matriz $\mathsf{D(t)}$ por lo que se operará en base a las expresiones:

$$x_{k+1} = \Phi_k x_k + u_k + w_k$$

**Ecuación 3-18**

$$z_k = H_k x_k + v_k$$

**Ecuación 3-19**

### 3.2.3 Discretización del modelo de estados

Discretización de la ecuación de estados

En el proceso de modelado, es común obtener el modelo de tiempo discreto de un sistema dado mediante el muestreo temporal de su modelo en tiempo continuo.

Este proceso de discretización se realiza en general en el domino de la frecuencia compleja. Así, luego de obtener la transformada de Laplace de la ecuación diferencial de estados, se aplica una transformación $(Tr)$ para mapear $s \rightarrow (Tr) \rightarrow z$. Luego, mediante $Z^{-1}$ se obtendrá el modelo en tiempo discreto. Según la forma de $Tr$ (Bilinear, Zero Order Hold, etc.) se consiguen diferentes aproximaciones con distintas propiedades, más o menos útiles, según la aplicación de interés.

Este método de discretización tiene su dual en el dominio del tiempo, este si bien es menos sintético nos permitirá obtener las expresiones de uso corriente en la bibliografía del KF que son las que aplicaremos en este trabajo.

Partiremos de la ecuación diferencial de estados dada por la Ecuación 3-14 para obtener la ecuación en diferencia del modelo de estados discreto dado por la Ecuación 3-18.

En primer lugar hallamos la solución de la Ecuación 3-14 utilizando un factor integrante:

$$\dot{x} = Fx + Cu + Gw$$

$$\dot{x} - Fx = Cu + Gw$$

Considerando como instante inicial a $t$ y como final a $t + T$, donde $T$ es el periodo de muestreo, el factor integrante queda: $e^{-\int_t^{t+T} F\, d\tau}$.

Premultiplicando m.a.m por el factor integrante propuesto:

$$e^{-\int_t^{t+T} F\, d\tau}\dot{x} - e^{-\int_t^{t+T} F\, d\tau}Fx = e^{-\int_t^{t+T} F\, d\tau}(Cu + Gw)$$

$$\frac{d}{dt}(e^{-\int_t^{t+T} F\, d\tau}x) = e^{-\int_t^{t+T} F\, d\tau}(Cu + Gw)$$

$$e^{-\int_t^{t+T} F\, d\tau}x = \int_t^{t+T} e^{-\int_t^{t+T} F\, d\tau}(Cu + Gw)d\zeta + K$$

Donde la constante de integración vale $K = x(t)$, esto es, el valor inicial del vector de estados.

$$x(t + T) = e^{\int_t^{t+T} F(\tau)\, d\tau}\int_t^{t+T} e^{-\int_t^{\zeta} F(\tau)d\tau}[C(\zeta)u(\zeta) + G(\zeta)w(\zeta)]d\zeta + e^{\int_t^{t+T} F(\tau)\, d\tau}x(t)$$

$$x(t + T) = e^{\int_t^{t+T} F(\tau)d\tau}x(t) + \int_t^{t+T} e^{\int_t^{t+T} F(\tau)\, d\tau}e^{-\int_t^{\zeta} F(\tau)d\tau}[C(\zeta)u(\zeta) + G(\zeta)w(\zeta)]d\zeta$$

**Ecuación 3-20**

En ausencia de términos forzantes la expresión anterior queda:

$$x(t + T) = e^{\int_t^{t+T} F(\tau)d\tau}x(t)$$

Aquí puede interpretarse el factor $e^{\int_t^{t+T} F(\tau)d\tau}$ como un operador que permite obtener el estado del sistema en el instante $t + T$ a partir del instante anterior $t$. Esto significa que $e^{\int_t^{t+T} F(\tau)d\tau}$ modela la evolución temporal del sistema entre dos instantes dados suponiendo entradas nulas. Por esto se lo denomina *matriz de transición de estados* o STM y se designa:

$$\Phi(t + T, t) = e^{\int_t^{t+T} F(\tau)d\tau}$$

**Ecuación 3-21**

Notar además, que el factor $e^{\int_t^{t+T} F(\tau)\, d\tau}e^{-\int_t^{\zeta} F(\tau)d\tau}$ de la Ecuación 3-20 puede ahora reescribirse como:

$$e^{\int_t^{t+T} F(\tau)\,d\tau} e^{-\int_t^{\xi} F(\tau)d\tau} = \Phi(t+T,t)\Phi^{-1}(\xi,t) = \Phi(t+T,\xi)$$

Con esto, la Ecuación 3-20 queda:

$$x(t+T) = \Phi(t+T,t)\,x(t) + \int_t^{t+T} \Phi(t+T,\xi)\,C(\xi)u(\xi)d\xi + \int_t^{t+T} \Phi(t+T,\xi)\,G(\xi)w(\xi)d\xi$$

**Ecuación 3-22**

Si restringimos la variable temporal a los instantes dados por $t = kT$ dónde $k = 0,1,2,3\ldots$, la Ecuación 3-21 queda:

$$\Phi(t+T,t) = \Phi(kT+T,kT)$$

Normalizando el tiempo respecto a $T$, se obtiene una versión en tiempo discreto de la STM:

$$\Phi[k+1,k] = \Phi_k$$

Con esto puede reescribirse la Ecuación 3-22 como:

$$x_{k+1} = \Phi_k x_k + u_k + w_k$$

Que es la Ecuación 3-18, donde:

$$u_k = \int_t^{t+T} \Phi(t+T,\xi)\,C(\xi)u(\xi)d\xi$$

**Ecuación 3-23**

Es la versión discretizada del término asociado a la entrada de control determinística $C(\xi)u(\xi)$ del modelo continuo y se denomina *término forzante.*

Por otro lado:

$$w_k = \int_t^{t+T} \Phi(t+T,\xi)\,G(\xi)w(\xi)d\xi$$

**Ecuación 3-24**

Es la versión discretizada del término asociado al ruido de planta $G(\xi)w(\xi)$ del modelo continuo. Debido a que $w(\xi)$ es un RP Gaussiano, $w_k$ constituye una secuencia también Gaussiana de media nula. Sin embargo su covarianza se verá modificada según lo que se expondrá a continuación.

Discretización de la ecuación de salida

Puesto que la Ecuación 3-15 del modelos de estados continuo no presenta ninguna operación de derivación, uno pretendería discretizarla directamente. Sin embargo esto representa un problema ya que al ser $v(t)$ un RP blanco, si tomamos muestras de $z(t)$ obtendremos una

secuencia aleatoria de varianza infinita. Debido a esto, se toma en el proceso de muestreo un promedio de $z(t)$ sobre el intervalo de muestreo $T$. Así:

$$z_k = \frac{1}{T}\int_t^{t+T} z(t)dt = \frac{1}{T}\int_t^{t+T} [H(t)x(t) + v(t)]dt$$

Para T lo suficientemente pequeño, se puede considerar a la cantidad determinística $H(t)x(t)$como constantes, luego:

$$z_k \approx H_k x_k + \frac{1}{T}\int_t^{t+T} v(t)dt$$

Con lo que se obtiene:

$$z_k = H_k x_k + v_k$$

Que es la Ecuación 3-19, aquí:

$H_k$ Es la matriz de sensibilidad de medición obtenida del muestreo de $H(t)$ en el instante $t = kT$.

Además:

$$v_k = \frac{1}{T}\int_t^{t+T} v(\tau)d\tau$$

**Ecuación 3-25**

Es la versión discreta del ruido de medición y resulta ser una secuencia aleatoria Gaussiana también de media nula pero de distinta covarianza, como se explica en la sección siguiente.

Discretización de las covarianzas

Cuando se trabaja con entradas estocásticas, además de discretizar las ecuaciones del modelo de estados, se hace necesario calcular las varianzas de los ruidos discretizados.

En primer lugar, para el ruido del proceso se tiene:

$$Q_k = E\langle w_k w_k^T \rangle$$

**Ecuación 3-26**

Reemplazando la Ecuación 3-24 en la Ecuación 3-26, obtenemos la siguiente expresión:

$$Q_k = E\left\langle \int_t^{t+T} \Phi(t+T,\xi)\, G(\xi) w(\xi) d\xi \left[\int_t^{t+T} \Phi(t+T,\alpha)\, G(\alpha) w(\alpha) d\alpha\right]^T \right\rangle$$

$$Q_k = \int_t^{t+T}\int_t^{t+T} \Phi(t+T,\xi)\, G(\xi) E\langle w(\xi) w(\alpha)^T \rangle G(\alpha) \Phi(t+T,\alpha) d\xi d\alpha$$

**Ecuación 3-27**

Si el período de muestreo $T$ es lo suficientemente pequeño como para considerar que:

$$\Phi(t+T,t) \approx I$$

La Ecuación 3-27 puede aproximarse con la siguiente expresión:

$$Q_k \approx \int_t^{t+T} \int_t^{t+T} G(\xi) E\langle w(\xi) w(\alpha)^T \rangle G(\alpha) d\xi d\alpha$$

**Ecuación 3-28**

La covarianza de $w(t)$ como se introdujo en la sección 3.2.1 es:

$$E\langle w(t_1), w(t_2)^T \rangle = Q(t_1)\delta(t_2 - t_1)$$

Reemplazando esta última expresión en la Ecuación 3-28 y aplicando la propiedad de muestreo de la covarianza impulsiva se obtiene:

$$Q_k = GQG^T T$$

**Ecuación 3-29**

En segundo lugar, para el ruido de medición se tiene:

$$R_k = E\langle v_k v_k^T \rangle$$

**Ecuación 3-30**

Reemplazando la Ecuación 3-25 en la expresión anterior y reordenando se tiene para un $T$ pequeño:

$$R_k = E\left\langle \left[\frac{1}{T}\int_t^{t+T} v(\xi) d\xi\right] \left[\frac{1}{T}\int_t^{t+T} v(\alpha) d\alpha\right]^T \right\rangle$$

$$R_k = \frac{1}{T^2}\iint_t^{t+T} E\langle v(\xi) v^T(\alpha) \rangle \, d\xi d\alpha$$

**Ecuación 3-31**

La covarianza de $v(t)$ como se introdujo en la sección 3.2.1 es:

$$E\langle v(t_1), v(t_2)^T \rangle = R(t_1)\delta(t_2 - t_1)$$

Reemplazando la última expresión en la Ecuación 3-31 y aplicando la propiedad de muestreo de la covarianza impulsiva se obtiene:

$$R_k = \frac{R}{T}$$

**Ecuación 3-32**

## 3.3 Problema general del filtrado y KF

Así se denota al problema de estimar el estado interno de un sistema lineal estocástico mediante la utilización únicamente de las salidas del sistema, o sea mediciones que están linealmente relacionadas a los estados internos.

NOTA: De aquí en adelante procederemos a trabajar únicamente en tiempo discreto, salvo que se especifique lo contrario. Esto es motivado por el hecho de que los desarrollos subsecuentes son presentados únicamente como soporte teórico para la implementación final del KF en sistemas digitales. Sin embargo, es necesario advertir al lector que la mayoría de los desarrollos que siguen tienen su dual para tiempo continuo.

En la siguiente figura se presenta gráficamente el problema general del filtrado adecuado al contexto de este trabajo

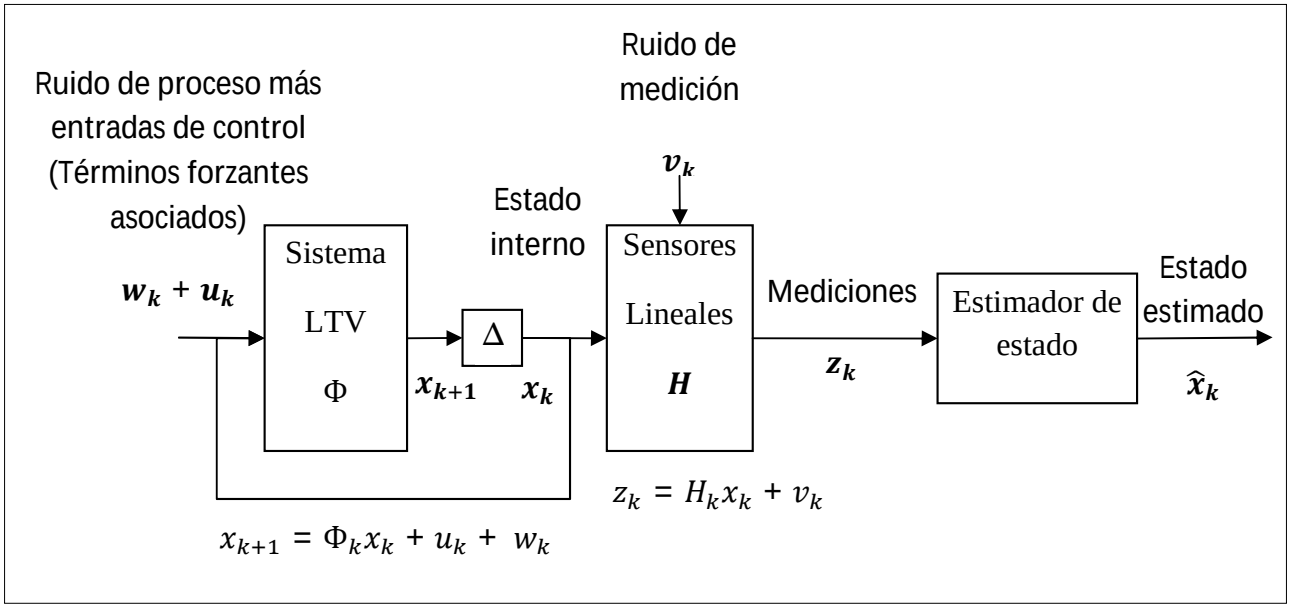



**Figura 3-2: Problema general del filtrado adecuado al contexto de este trabajo.**

Como se ilustra, el problema se reduce a hallar un sistema estimador que permita obtener el estado estimado $\hat{x}_k$, utilizando las mediciones del sistema $z_k$ y un criterio en particular.

En principio será posible obtener diversos estimadores que utilicen diferentes criterios. Como se verá más adelante el criterio elegido está asociado a la característica estadística de $x_k$ (media, moda, etc.) que se desea estimar con $\hat{x}_k$.

Filtros de mínimo cuadrado

Los estimadores, en donde el criterio que se utiliza para hallar $\hat{x}_k$, es la minimización del error cuadrático medio (MSE) de estimación, son llamados filtro de mínimo cuadrado. El MSE de estimación está dado por:

$$MSE = E\langle (x_k - \hat{x}_k)(x_k - \hat{x}_k)^T \rangle$$

Filtro de Kalman

Cada uno de los posibles estimadores que resuelven el problema del filtrado, tendrá diferentes propiedades pudiendo ser tanto sistemas lineales como no lineales, sin embargo, para el caso particular en que:

- El sistema es LTV

- Los ruidos de proceso $w_k$ y de medición $v_k$ son RPs blancos Gaussianos de media nula y en principio no correlacionados entre sí (Notar que la blancura del RP implica que no hay dependencia estadística entre sus variables aleatorias, pero no restringe la forma de la función de densidad de probabilidad de las mismas. Estos requerimientos estadísticos iniciales para los ruidos, se verán justificados y posiblemente modificados en los desarrollos del KF de la siguiente sección.).

Puede demostrarse que existe un estimador óptimo y que es uno lineal. Esto quiere decir que no puede hallarse un sistema estimador mejor (sea lineal o no lineal) en relación a la minimización del MSE.

El sistema referido en el párrafo anterior es el que se conoce como *Filtro de Kalman* (KF).

En este punto cabe aclarar que, si bien a estos sistemas estimadores se les llama filtros en el sentido de que separan una señal deseada (estado) de una indeseada (ruidos), su operación y filosofía difiere sustancialmente de la asociada a los filtros lineales que operan en el domino de la frecuencia. En los primeros se aplica un criterio estadístico (minimizar MSE de estimación) para calcular algún parámetro libre del sistema (por ejemplo la ganancia), por esto, no es necesaria ninguna relación especial entre el contenido armónico del ruido y la respuesta en frecuencia del estimador.

Debido a lo expuesto en esta sección, es que se enmarca el desarrollo del KF en el área conocida como *teoría de estimación estadística.* Esta es una disciplina que halla sus fundamentos tanto en la teoría de probabilidades como en la teoría de sistemas dinámicos. Lo anterior lo presenta de manera sencilla [**2**] mediante la siguiente figura:

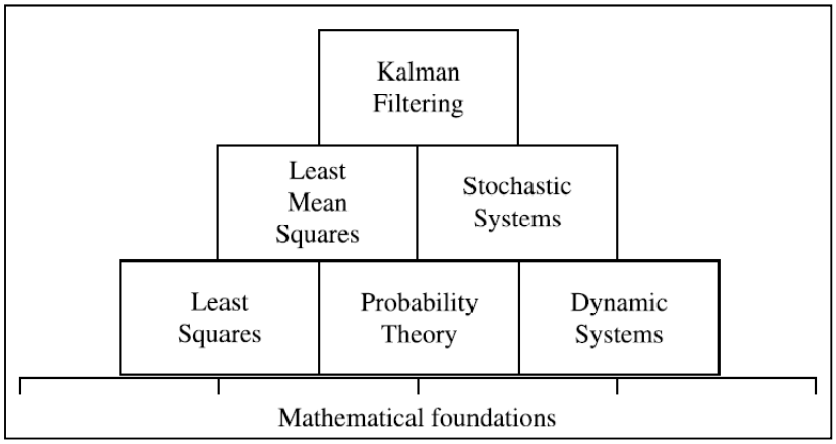



**Figura 3-3:Fundamentos teóricos del KF según [2]**

## 3.4 Filtro de Kalman

### 3.4.1 Historia y alcance

El KF es el resultado de un proceso evolutivo de ideas de muchos y diversos pensadores a lo largo de varios siglos. Los principales contribuyentes pueden visualizarse en la selección dada en [**2**] mediante la siguiente figura:

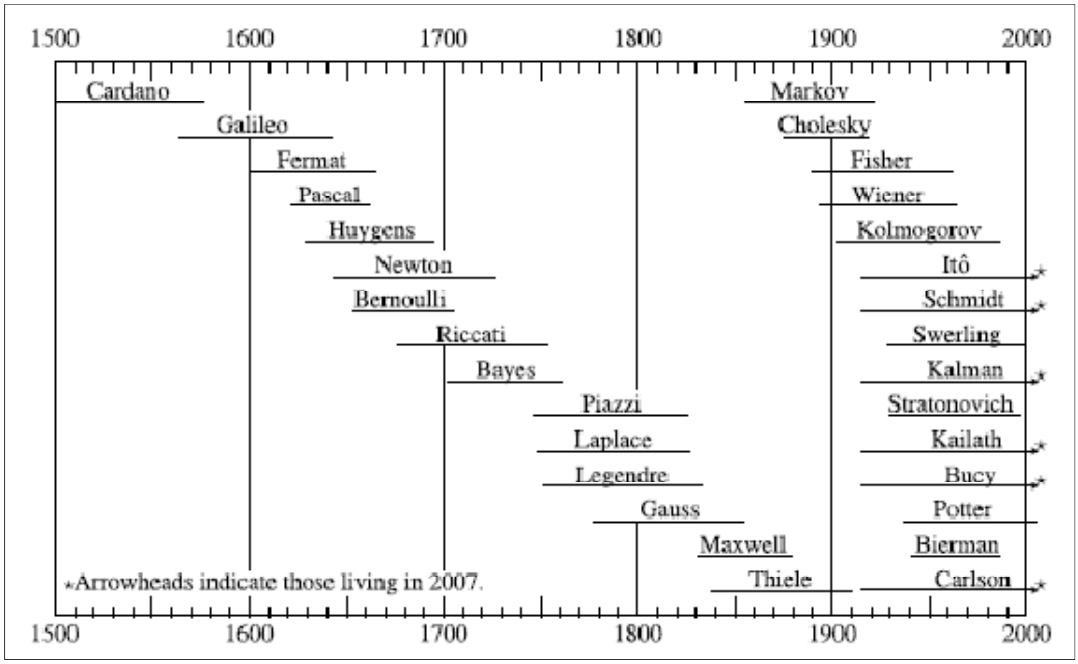



**Figura 3-4: Líneas de vida de los principales contribuyentes a la teoría de la estimación según [2].**

Como puede apreciarse, un desarrollo histórico exhaustivo no está dentro de los límites de este trabajo. Por esto nos contentaremos con una breve descripción del aporte realizado por el desarrollador del KF, a saber Rudolf Emil Kalman (1930-Presente).

En 1958, Kalman obtuvo financiamiento por parte de la Oficina de investigación científica de la fuerza área de EE.UU, para realizar investigación avanzada en estimación y control en el Instituto de investigación para estudios avanzados de la compañía Glenn L. Martin en Baltimore EE.UU.

Para ese momento el fondo teórico pertinente del que Kalman disponía era básicamente:

- Filtrado de Wiener-Kolmogorov: Durante los primeros años de la segunda guerra mundial, Norbert Wiener (1894-1964) desarrollo la teoría de estimadores óptimos principalmente, mediante el uso de la PSD en el domino de la frecuencia para caracterizar la dinámica y propiedades estadísticas de un sistema dinámico. Derivó así la solución del problema general del filtrado minimizando el MSE en términos de un operador integral que podía ser sintetizado en circuitos analógicos. Todo esto asumiendo un modelo del proceso invariante en el tiempo.

- Modelos estocásticos de estados: Conocido como la teoría de Shaping Filters (Ver el final de la sección 3.1.5), este enfoque fusiona el paradigma de modelado por estados

de un sistema lineal con la teoría de RPs al establecer que puede generarse un RP con una PSD deseada, al excitar un filtro lineal con una respuesta en frecuencia especifica mediante ruido blanco.

Kalman extendió la teoría de Wiener-Kolmogorov a sistemas variantes en el tiempo y a sistemas de tiempo discreto mediante la aplicación del modelado por estados.

Las ideas de Kalman fueron tomadas con cierto escepticismo entre sus pares por lo que, decidió publicar su primer trabajo en el tema en una revista de ingeniería mecánica en vez de una en ingeniería eléctrica. A pesar de esto, Kalman persistió en presentar su filtro el cual pronto se convirtió en tema de estudio en numerosas universidades de EE.UU.

Aplicaciones

Debido a la compatibilidad del desarrollo de Kalman con los sistemas eléctricos digitales, las primeras aplicaciones prácticas no tardaron en aparecer. Tal vez la más emblemática, sea la aplicación del KF en el sistema de navegación inercial utilizado para el cálculo de parte de la trayectoria en el proyecto Apolo durante los años 60.

El desempeño exitoso en el programa Apolo, fue el puntapié inicial de una fusión permanente entre los sistemas de navegación inercial y el KF que se remonta hasta la actualidad. Aquí, este permite además, resolver los problemas de fusión de sensores y el de rechazo de datos erróneos de manera elegante y sintética.

Desde su introducción, gracias al avance de la computación digital y sucesivos desarrollos en materia de estabilidad numérica, el KF ha ganado considerable terreno en aplicaciones tan diversas como: procesamiento de imágenes, navegación satelital, aviónica, robótica, predicción del clima espacial, redes de computadoras, etc.

No es exagerado decir que, debido a su impacto tecnológico, puede considerarse al KF el mayor desarrollo dentro del área de la estimación estadística del siglo veinte.

### 3.4.2 Derivación de la ganancia de Kalman

La derivación de la ganancia del KF, se basa en la selección del criterio de estimación adecuado del vector de estado $x$, el cual, tendrá como consecuencia la minimización del MSE de la estimación $\hat{x}$:

$$P_k = E\langle (x_k - \hat{x}_k)(x_k - \hat{x}_k)^T \rangle$$

**Ecuación 3-33**

La estimación elegida será:

$$\hat{x}_k = E\langle x_k | z_k^* \rangle$$

**Ecuación 3-34**

El valor en el que se estimará el vector de estado, se tomará del valor esperado de la variable aleatoria $x_k$ condicionada a la ocurrencia de la cadena de mediciones $z_k^*=z_0, z_1, \dots, z_k$.

Primero escribimos el error cuadrático medio de la estimación de $x_k$ condicionado a $z_k^*$:

$$E\langle(x_k - \hat{x}_k)^T(x_k - \hat{x}_k)|z_k^*\rangle$$

Expandiendo la factorización obtenemos:

$$E\langle(x_k^T x_k - x_k^T\hat{x}_k - \hat{x}_k^T x_k + \hat{x}_k^T\hat{x}|z_k^*)\rangle$$

Distribuyendo la esperanza y la condición:

$$E\langle x_k^T x_k|z_k^*\rangle - E\langle x_k^T|z_k^*\rangle\hat{x}_k - \hat{x}_k^T E\langle x_k|z_k^*\rangle + \hat{x}_k^T\hat{x}_k$$

El hecho de haber sacado la estimación $\hat{x}_k$ como factor fuera de los operadores de esperanza, se justifica porque dicha estimación es función de la cadena $z_k^*$ puesta (Ecuación 3-34), como condición de la variable aleatoria $x_k$.para su obtención.

Completando el cuadrado de los últimos tres términos, se obtiene:

$$E\langle(x_k - \hat{x}_k)^T(x_k - \hat{x}_k)|z_k^*\rangle = E\langle x_k^T x_k|z_k^*\rangle + [\hat{x}_k - E\langle x_k|z_k^*\rangle]^T[\hat{x}_k - E\langle x_k|z_k^*\rangle] - E\langle x_k^T|z_k^*\rangle E\langle x_k|z_k^*\rangle$$

Solamente los factores del término central del miembro derecho dependen de $\hat{x}_k$ y por ende de la elección del criterio de estimación. Por lo tanto, el afán de reducir el MSE todo lo posible nos lleva a escoger el criterio de estimación propuesto en la Ecuación 3-34, que anulará dichos factores.

Además de minimizar el MSE, el criterio de estimación elegido junto con la restricción de tratar con RPs cuyas RVs tengan función de densidad de probabilidad Gaussiana, nos permite escribir una expresión *explícita* para la estimación óptima en forma *recursiva.*

NOTA: Debemos mencionar que si bien se utilizarán RPs blancos cuyas RVs tengan una función de densidad de probabilidad Gaussiana, no es condición necesaria la blancura de dichos procesos para el desarrollo de la teoría del KF. De la no blancura de dichos procesos surge la necesidad de un tratamiento del sistema en el espacio de los estados con múltiples entradas y salidas que, aquí no desarrollamos.

Asumiendo que, se dispone de una estimación previa (*a priori*) óptima $\hat{x}_k^-$ y su matriz de covarianza del error $P_k^-$ y considerando que $x_k$ denota la variable aleatoria $x$ en el tiempo $t_k$ condicionada a la cadena de mediciones $z_{k-1}^*$, se sabe que la función de densidad de probabilidad de $x_k$ es:

$$p[x_k] \sim N(\hat{x}_k^-, P_k^-)$$

**Ecuación 3-35**

De la ecuación de salida del sistema discretizado, repetida aquí por conveniencia:

$$z_k = H_k x_k + v_k$$

**Ecuación 3-36**

Podemos escribir de manera inmediata la función de densidad de probabilidad de $z_k$ como sigue:

$$E\langle H_k x_k + v_k \rangle = H_k E\langle x_k \rangle + E\langle v_k \rangle = H_k \hat{x}_k^- + 0$$

$$\begin{aligned} E\langle (H_k x_k + v_k)(H_k x_k + v_k)^T \rangle - (H_k \hat{x}_k^-)(H_k \hat{x}_k^-)^T \\ = E\langle H_k x_k x_k^T H_k^T + H_k x_k v_k^T + v_k x_k^T H_k^T + v_k v_k^T \rangle - (H_k \hat{x}_k^- \hat{x}_k^{-T} H_k^T) \\ = H_k [E\langle x_k x_k^T \rangle - \hat{x}_k^- \hat{x}_k^{-T}] H_k^T + H_k E\langle x_k v_k^T \rangle + E\langle v_k x_k^T \rangle H_k^T + E\langle v_k v_k^T \rangle \\ = H_k P_k^- H_k^T + 0 + 0 + R_k \end{aligned}$$

$$p[z_k] \sim N(H_k \hat{x}_k^-, H_k P_k^- H_k^T + R_k)$$

**Ecuación 3-37**

De la Ecuación 3-36, podemos obtener la función de densidad de probabilidad condicional de la variable aleatoria $z_k$ dada $x_k$:

$$E\langle H_k x_k + v_k | x_k \rangle = H_k E\langle x_k | x_k \rangle + E\langle v_k | x_k \rangle = H_k x_k + 0$$

$$\begin{aligned} E\langle (H_k x_k + v_k)(H_k x_k + v_k)^T | x_k \rangle - (H_k x_k)(H_k x_k)^T \\ = E\langle H_k x_k x_k^T H_k^T + H_k x_k v_k^T + v_k x_k^T H_k^T + v_k v_k^T | x_k \rangle - (H_k x_k x_k^T H_k^T) \\ = H_k [E\langle x_k x_k^T | x_k \rangle - x_k x_k^T] H_k^T + H_k E\langle x_k v_k^T | x_k \rangle + E\langle v_k x_k^T | x_k \rangle H_k^T \\ + E\langle v_k v_k^T | x_k \rangle = 0 + 0 + 0 + R_k \end{aligned}$$

$$p[z_k | x_k] \sim N(H_k x_k, R_k)$$

**Ecuación 3-38**

Los resultados obtenidos anteriormente, se basan en el hecho de que $v_k$ y $x_k$ son RVs independientes, ya que la función de densidad de probabilidad de $x_k$ está condicionada a la cadena de mediciones $z_{k-1}^*$, la cual no incluye la medición $z_k$ afectada por $v_k$.

Finalmente, mediante la fórmula de Bayes podemos escribir:

$$p[x_k | z_k] = \frac{p[z_k | x_k] p[x_k]}{p[z_k]}$$

**Ecuación 3-39**

Haciendo referencia a la Ecuación 3-38, Ecuación 3-37 y la Ecuación 3-35, reemplazándolas en la Ecuación 3-39 podemos escribir la siguiente expresión:

$$p[x_k | z_k] \sim \frac{N(H_k x_k, R_k) N(\hat{x}_k^-, P_k^-)}{N(H_k \hat{x}_k^-, H_k P_k^- H_k^T + R_k)}$$

**Ecuación 3-40**

Se puede apreciar entonces que, la función de densidad condicional de probabilidad $p[x_k | z_k]$, está condicionada a la cadena $z_k^*$ completa de mediciones.

A través de la adecuada inspección de los términos exponenciales de las funciones de densidad de probabilidad normal, se pueden determinar la media y la covarianza de la Ecuación 3-40.

$$Media = \hat{x}_k^- + P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1} (z_k - H_k \hat{x}_k^-)$$

**Ecuación 3-41**

$$Covarianza = [(P_k^-)^{-1} + H_k^T R_k^{-1} H_k]^{-1}$$

**Ecuación 3-42**

Las propiedades de la función de densidad de probabilidad normal, permiten obtener expresiones explícitas recursivas para lograr las sucesivas estimaciones que minimicen el MSE en cada iteración.

Nótese además que, se ha arribado a una forma lineal del estimador (Ecuación 3-42). Esta característica no se impuso, por lo que, se deduce que el mejor estimador posible es uno lineal y por ende no es necesario investigar otras posibles opciones no lineales.

### 3.4.3 Algoritmo iterativo

En un comienzo, para emprender la ejecución del algoritmo iterativo no se dispone de mediciones previas con las cuales realizar la estimación inicial $\hat{x}_k^-$, es por ello que se toma como estimación inicial $\hat{x}_0^-$, a la media del proceso $x$. Además, la covarianza del error $P_0^-$ se tomará igual a la matriz de covarianza del proceso $x$.

Con el fin de actualizar la estimación anterior se hace uso de la medición actual $z_k$ a través de la Ecuación 3-41, la cual, constituye una *combinación lineal* entre la medición ruidosa y la estimación inicial de la siguiente manera:

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - H_k \hat{x}_k^-)$$

**Ecuación 3-43**

Donde:

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}$$

**Ecuación 3-44**

Es la ganancia de Kalman, la cual surge del criterio de estimación seleccionado en la Ecuación 3-34, junto al requisito de fijar que las funciones de densidad de probabilidad de las RVs de los RPs participantes sean normales de media nula.

Sabiendo que el error de la estimación a priori $\hat{x}_k^-$ es:

$$e_k^- = x_k - \hat{x}_k^-$$

La matriz de covarianza asociada a dicho error será:

$$P_k^- = E\langle e_k^- e_k^{-T} \rangle = E\langle (x_k - \hat{x}_k^-)(x_k - \hat{x}_k^-)^T \rangle$$

Por otra parte de la Ecuación 3-43, el error de la estimación *a posteriori* $\hat{x}_k$ es:

$$e_k = x_k - \hat{x}_k$$

Haciendo referencia a la Ecuación 3-33, se puede expresar la matriz de covarianza asociada a dicho error como:

$$P_k = E\langle e_k e_k^T \rangle = E\langle (x_k - \hat{x}_k)(x_k - \hat{x}_k)^T \rangle$$

**Ecuación 3-45**

Sustituyendo la Ecuación 3-36 en la Ecuación 3-43 y esta última en la Ecuación 3-45, se obtiene:

$$P_k = E\langle [(x_k - \hat{x}_k^-) - K_k(H_k x_k + v_k - H_k \hat{x}_k^-)][(x_k - \hat{x}_k^-) - K_k(H_k x_k + v_k - H_k \hat{x}_k^-)]^T \rangle$$

Expandiendo la factorización, introduciendo el operador de esperanza y teniendo en cuenta la independencia entre el error de la estimación a priori $e_k^-$ y el ruido de medición $v_k$, se obtiene que:

$$P_k = P_k^- - K_k H_k P_k^- - P_k^- H_k^T K_k^T + K_k(H_k P_k^- H_k^T + R_k)K_k^T$$

**Ecuación 3-46**

Sustituyendo la Ecuación 3-44 en la Ecuación 3-46, se llega a:

$$P_k = (I - K_k H_k)P_k^-$$

**Ecuación 3-47**

Se puede demostrar la equivalencia entre la Ecuación 3-42 y la Ecuación 3-47.

A través de la Ecuación 3-43, se hace posible la asimilación de una nueva medición en el instante $t_k$ para producir una corrección en la estimación según la historia completa que precede a la evolución del sistema. Es importante notar la necesidad de $\hat{x}_k^-$ en la Ecuación 3-43 y de $P_k^-$ en la Ecuación 3-47, para poder obtener $\hat{x}_k$ y $P_k$ respectivamente, se puede anticipar la misma necesidad en la siguiente iteración con el fin de hacer un uso óptimo de la medición en el instante $t_{k+1}$.

Debido a esto, es necesaria que la estimación actualizada $\hat{x}_k$ sea proyectada a la siguiente iteración como si esta atravesara el sistema mismo. Por lo tanto haciendo referencia a la Ecuación 3-18, este proceso se llevará a cabo de la siguiente manera:

$$\hat{x}_{k+1}^- = \Phi_k \hat{x}_k + E\langle w_k \rangle + u_k$$

**Ecuación 3-48**

Como se distingue en esta expresión, se reemplazo $w_k$ por su valor medio $E\langle w_k \rangle$. Esto es así debido a que al ser el valor de $w$ desconocido en cada instante, la mejor estimación de su influencia neta sobre el proceso a lo largo de sucesivas iteraciones es un desbalance (offset) en el valor del estado igual a $E\langle w_k \rangle$.

Para RPs con media nula, como se consideran aquí, la Ecuación 3-48 queda:

$$\hat{x}_{k+1}^{-} = \Phi_k \hat{x}_k + u_k$$

**Ecuación 3-49**

La matriz de covarianza del error cometido por aceptar esta proyección en el valor del vector de estado se computa de la siguiente manera:

$$\begin{aligned} e_{k+1}^{-} &= x_{k+1} - \hat{x}_{k+1}^{-} = (\Phi_k x_k + w_k + u_k) - (\Phi_k \hat{x}_k + u_k) = \Phi_k (x_k - \hat{x}_k) + w_k \\ &= \Phi_k e_k + w_k \end{aligned}$$

$$\begin{aligned} P_{k+1}^{-} &= E\langle e_{k+1}^{-} {e_{k+1}^{-}}^T \rangle = E\langle (\Phi_k e_k + w_k)(\Phi_k e_k + w_k)^T \rangle \\ &= E\langle \Phi_k e_k e_k^T \Phi_k^T + \Phi_k e_k w_k^T + w_k e_k^T \Phi_k^T + w_k w_k^T \rangle \\ &= \Phi_k E\langle e_k e_k^T \rangle \Phi_k^T + \Phi_k E\langle e_k w_k^T \rangle + E\langle w_k e_k^T \rangle \Phi_k^T + E\langle w_k w_k^T \rangle \\ &= \Phi_k P_k \Phi_k^T + 0 + 0 + Q_k \end{aligned}$$

Se llegó a este resultado por el hecho de que las RVs $w_k$ y $e_k$ son independientes debido a que, el error cometido en la estimación actualizada $e_k$ se debe al ruido $w_{k-1}$, y por lo tanto no tiene ninguna vinculación estadística con $w_k$. Es así que:

$$P_{k+1}^{-} = \Phi_k P_k \Phi_k^T + Q_k$$

**Ecuación 3-50**

Haciendo referencia a la Ecuación 3-44, Ecuación 3-43, Ecuación 3-47, Ecuación 3-49 y la Ecuación 3-50, hemos elaborado la siguiente figura, en la cual se hace explícita la recursividad del algoritmo del KF así como otras propiedades a las que se hará referencia en las próximas secciones:

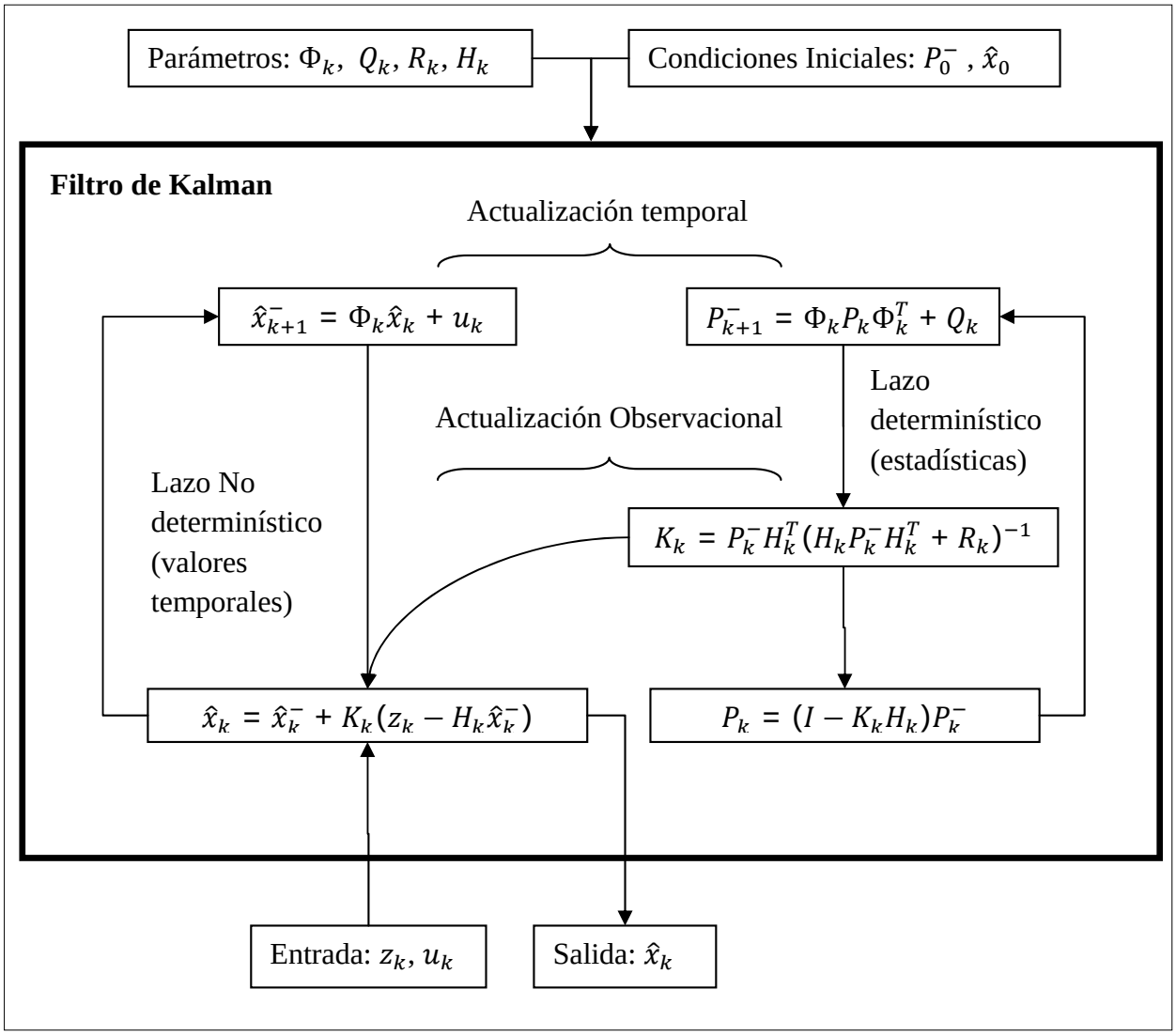



**Figura 3-5: Ecuaciones iterativas del algoritmo original del KF y su relación causal expresadas en un diagrama de flujo que ilustra los pasos a seguir para ejecutar el mismo.**

### 3.4.4 KF como solución del Problema general del filtrado

En la Figura 3-6 se grafica el KF como el estimador de estados que viene a solucionar el problema general del filtrado presentado en la sección 3.3. Se ha detallado además, toda la trayectoria de datos y los componentes dentro del filtro.

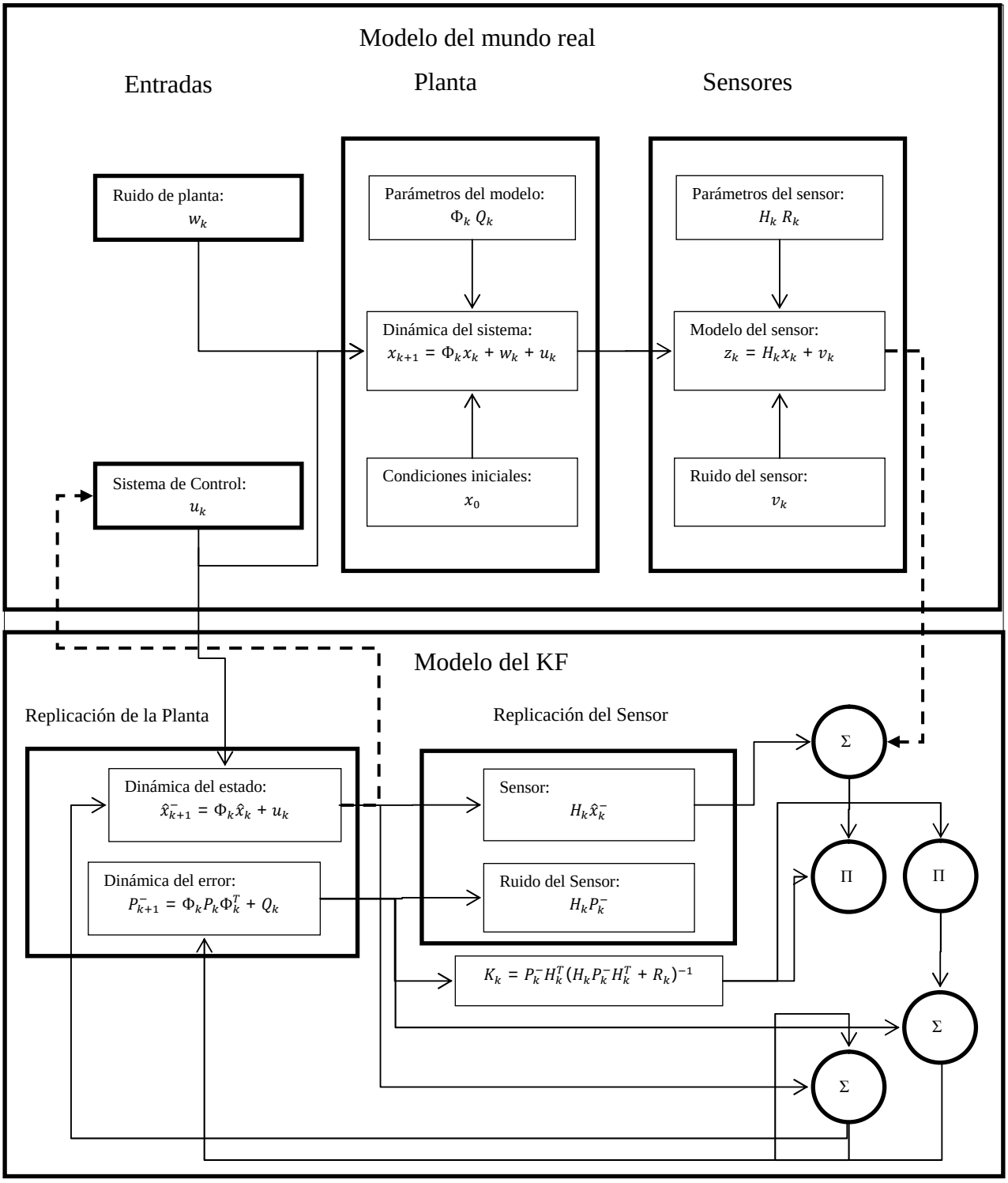



**Figura 3-6: Esquema detallado que ilustra al KF como solución del problema del filtrado.**

A continuación, se presentan algunos puntos importantes, para concluir con el desarrollo de la teoría del KF:

- Las ecuaciones de actualización temporal dentro del KF constituyen una réplica interna de la dinámica de la planta y lo sensores. Nótese además que debido a la imposibilidad de conocer el valor instantáneo de los ruidos estas replicas internas no son exactas pues estiman el efecto neto de los ruidos mediante sus valores medios.

- Para el caso de ruidos Gaussianos, nótese que el usar la media condicionada como criterio de estimación (Ecuación 3-34) es el más "adecuado" en términos de que el pico de densidad de probabilidad coincide con la media. Además puede demostrarse que en este caso el criterio de la media condicional también minimiza la esperanza de casi cualquier funcion no decreciente del error de estimación (como el MSE).

- Para casos prácticos en donde no se conoce en detalle la distribución de probabilidades de los ruidos, salvo tal vez la estructura de la covarianza, es posible suponer que estos son Gaussianos (con la covarianza conocida). Lógicamente esto conlleva un cierto riesgo pero permite al menos tener un modelo operativo.

### 3.4.5 KF para sistemas invariantes en el tiempo

En este trabajo implementamos el KF para el caso en que tanto el proceso como los sensores pueden modelarse como sistemas LTI. Adicionalmente se considera que las características estadísticas de ambos ruidos son también constantes así como la entrada de control determinística.

Para este caso, las siguientes cantidades permanecen invariantes para todas las iteraciones del filtro:

$$\Phi_k \to \Phi;\ H_k \to H;\ R_k \to R;\ Q_k \to Q;\ u_k \to u$$

Con esto, las ecuaciones iterativas del FK quedan:

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$$

**Ecuación 3-51**

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - H\hat{x}_k^-)$$

**Ecuación 3-52**

$$P_k = (I - K_k H) P_k^-$$

**Ecuación 3-53**

$$\hat{x}_{k+1}^- = \Phi \hat{x}_k + u$$

**Ecuación 3-54**

$$P_{k+1}^- = \Phi P_k \Phi^T + Q$$

**Ecuación 3-55**

Un punto importante a discutir aquí son las dos diferentes opciones que existen para implementar el algoritmo dado por las ecuaciones arriba expresadas.

Si analizamos la Figura 3-5 , el lazo de computo que se realiza para actualizar los valores de $P_k$ es un lazo determinístico, esto significa que no es influenciado por los valores de $z_k$. Ó sea que, si se conociesen todos los valores de $\Phi_k, H_k, R_k \ y\ Q_k$ es posible computar este lazo antes

de que de hecho se reciban o generen las mediciones. Esta característica abre la posibilidad de lo que se denomina un análisis Off-Line del KF haciendo referencia a que se computan todos los valores de la ganancia antes de que la planta este de hecho funcionando. En el caso de sistemas LTI que tratamos aquí, los valores de las cantidades mencionadas son todos conocidos y constantes, por ende en principio es viable una implementación del KF en donde no sea necesario computar las ganancias en tiempo de ejecución (On-Line).

De esta forma el problema de implementar el FK tiene en este punto, dos soluciones en términos de cuando se computen las $K_k$, estas se detallan en la siguiente tabla:

| Opción de implementación | Ventajas | Desventajas |
|---|---|---|
| Off-Line | • Menor complejidad computacional: No es necesario computar la Ecuación 3-51, Ecuación 3-53, ni la Ecuación 3-55 en tiempo de ejecución. | • Es necesario computar previamente el lazo determinístico para todo el tiempo e interés.<br>• Este KF no puede funcionar bajo cambios en las IC, Φ, o H. |
| On-Line | • El KF puede funcionar con otras IC o readaptarse fácilmente e incluso en campo para otra planta o tipo de sensor.<br>• No es necesario un cálculo previo. | • Mayor complejidad computacional. |

**Tabla 3-1: Dos opciones diferentes a la hora de implementar el KF para sistemas LTI**

De lo expuesto en la tabla de arriba, y principalmente debido al hecho de que implementar la opción On-Line permite desarrollar un sistema digital que este mucho más "cerca" del necesario para poder computar el caso de una planta variante en el tiempo (que es el de mayor interés practico y se discutirá en la sección 7.6), es que se decide por esta opción. Esta decisión es también impulsada por los objetivos iniciales de este trabajo que son los de: elaborar el Know-How necesario para que el KF pueda ser utilizado en futuros trabajos e investigaciones regionales y, satisfacer ciertos requerimientos dados por el Laboratorio de Computación Reconfigurable de la Universidad Tecnológica Nacional FRM.

# 4 Field Programable Gates Array

## 4.1 Introducción

Los Arreglos de Compuertas Programables en Campo o Field Programable Gates Array (FPGA), son dispositivos semiconductores programables basados en un arreglo matricial de Bloques Lógicos Configurables o CLBs conectados mediante interconexiones programables. Las FPGAs pueden ser programadas para que cumplan requerimientos funcionales específicos de una aplicación [**3**].

### 4.1.1 Historia y alcance

Las FPGA son el resultado de la convergencia de dos tecnologías diferentes, los dispositivos lógicos programables o PLDs y los circuitos integrados de aplicación específica o ASIC.

La historia de los PLDs comenzó con los primeros dispositivos PROM a los que luego se les añadió versatilidad con los Arreglos Lógicos Programables o PAL que permitieron un mayor número de entradas y la inclusión de registros de desplazamiento. Esos dispositivos han continuado creciendo en tamaño y performance.

Por otro lado, los ASIC siempre han poseído gran performance, pero su uso ha requerido tradicionalmente una considerable inversión tanto de tiempo como de dinero. Intentos de reducir esta característica han provenido de la modularización de los elementos de los circuitos, como los ASIC basados en celdas, y de la estandarización de las máscaras de diseño.

Las dos estrategias anteriores se combinaron finalmente mediante un mecanismo de interconexión que pudiese programarse utilizando fusibles, anti fusibles o celdas RAM y celdas ROM, resultando esto en la aparición de las FPGAs.

La primera FPGA comercial fue desarrollada por los cofundadores de la empresa Xilinx, Ross Freeman y Bernard Vonderschmitt en 1985, denominada la XC2064. Esta incluía 64 CLBs con dos tablas de entrada-salida o LUTs de tres entradas.

Debido la gran mejora en el desempeño y flexibilidad de las FPGA, que las nuevas tecnologías de fabricación de semiconductores han proveído en los últimos años y a su efecto positivo en la relación costo de desarrollo versus performance (ver la siguiente sección), el uso de las FPGAs se ha extendido a diversas áreas. Principalmente en el procesamiento digital de señales, estas incluyen: telecomunicaciones, procesamiento de video y audio, módems de alta velocidad, criptografía, control industrial, tecnología aeroespacial y militar, entre otras.

En los siguientes gráficos, se muestran las distribución de ventas de FPGAs y PLDs: geográficamente, por mercado (Figura 4-1) y por fabricante (Figura 4-2) según lo expresado por Xilinx, el fabricante líder en FPGA a nivel mundial, en [**4**].

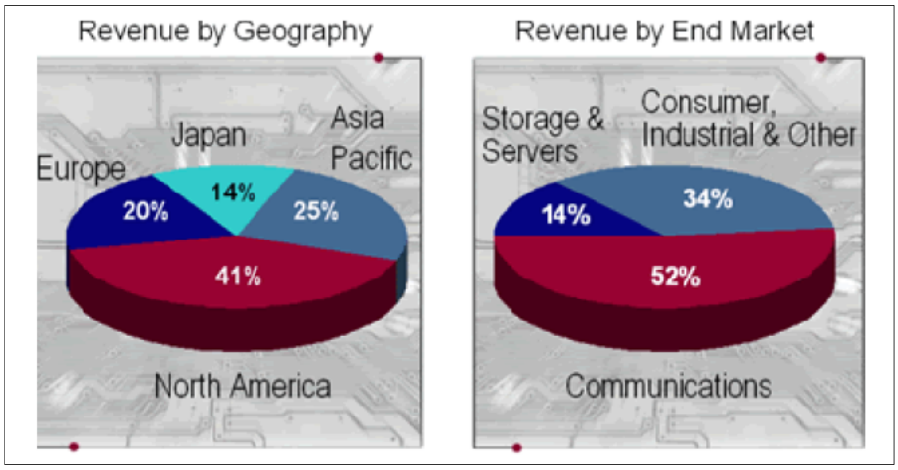



**Figura 4-1: Distribución de las ventas de FPGAs y PLDs geográficamente (izquierda) y por mercado (derecha)**

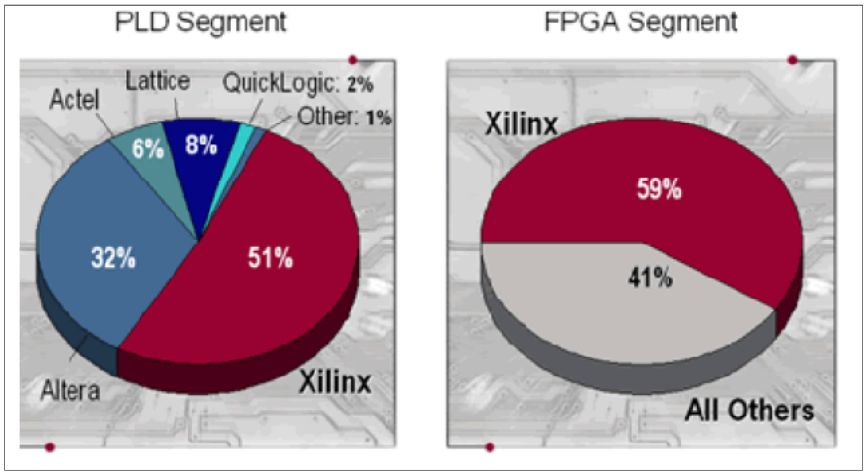



**Figura 4-2: Distribución de las ventas de FPGAs (derecha) y PLDs (izquierda) según el fabricante.**

### 4.1.2 Ventajas y desventajas de los sistemas basados en FPGA

A la hora de implementar un sistema digital para procesamiento de señales, las opciones más comunes del mercado actual son:

- Desarrollar un Circuito Integrado de Aplicación Especifica o ASIC
- Utilizar un Circuito Integrado para Procesamiento de Señales o DSP
- Desarrollar el sistema en una FPGA
- Desarrollar el software que cumpla con la misma tarea en un sistema Microprocesado

Entre las características más relevantes de estas cuatro plataformas, podemos establecer una comparación básica presentada en la siguiente tabla:

| Tecnología | Performance/ Costo | Tiempo de desarrollo | Velocidad de operación | Flexibilidad del diseño |
|---|---|---|---|---|
| ASIC | Muy alto | Muy largo | ⇧ | ⇩ |
| DSP | Medio | Largo | | |
| FPGA | Medio-Bajo | Corto | | |
| Microprocesador | Medio-Bajo | Corto | | |

**Tabla 4-1: Comparación de los aspectos relevantes, en términos de su impacto económico, entre diferentes alternativas tecnológicas a la hora de implementar un sistema digital.**

La característica más relevante a la hora de seleccionar una tecnología específica para utilizar en un proyecto de ingeniería, es sin duda el efecto global que esta va a tener sobre el precio del producto final (más específicamente en las ganancias). Las cuatro propiedades presentadas en la tabla son de las que más influyen en este aspecto y son por ende, las que guiarán la selección de la tecnología a utilizar.

El problema a la hora de comparar las diferentes opciones es entonces de carácter técnico-económico y un análisis detallado debería incluir entre otros aspectos: la influencia del tiempo de desarrollo en el costo del producto en relación a su ciclo de vida en el mercado, las tendencias del mercado de semiconductores, disponibilidad de proveedores, ciclo de vida esperado del producto, costos de desarrollo, etc.

Aquí nos contentaremos con detallar solo algunos de los aspectos técnicos del problema:

<u>El costo de desarrollo:</u>

El costo de desarrollo del sistema es uno de los elementos que tiene mayor influencia en el costo final de los productos de mediana y alta tecnología.

Hay numerosos aspectos que influyen en el costo de desarrollo, tal vez el más importante es el del *tiempo de desarrollo*. En la siguiente grafica se comparan el tiempo de diseño entre sistemas basados en ASIC y en FPGA según [**4**]:

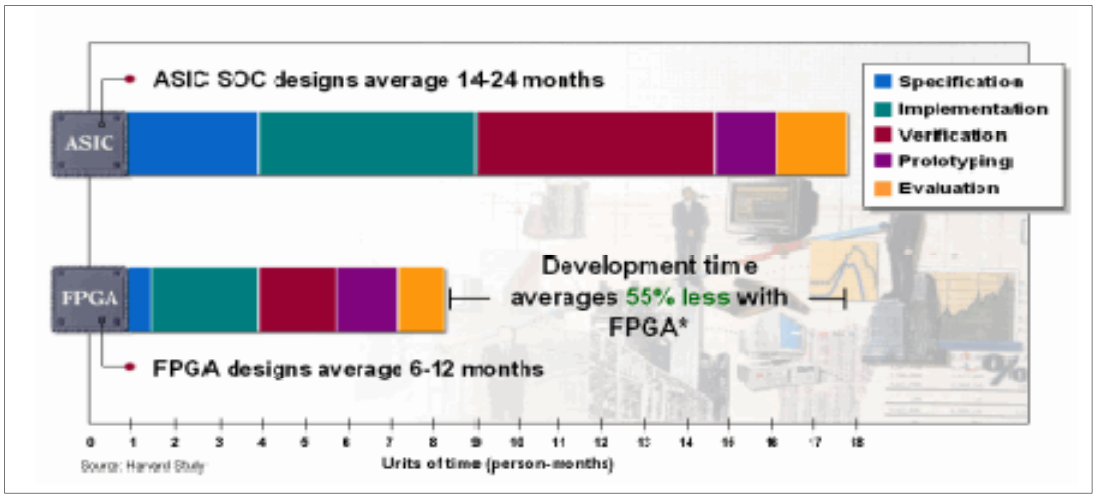



**Figura 4-3: Comparación entre los tiempos de desarrollo para sist. basados en ASIC y FPGA.**

Como puede apreciarse, existe una reducción promedio de al menos un 55% en el tiempo de desarrollo entre ambas plataformas.

Flexibilidad del sistema:

En caso de productos que tengan un tiempo de vida considerable, el problema de la flexibilidad en términos de su reconfiguración o actualización cobra importancia.

Para el caso de los ASIC esta característica está de lleno excluida debido a su arquitectura fija luego de la manufactura.

Para el caso de las FPGA las posibilidades de reconfiguración son muy amplias requiriendo esto solo de un costo de ingeniería para desarrollar las modificaciones necesarias.

Los sistemas basados en procesadores, son los que en principio tienen un costo de reconfiguración menor, pero no deben olvidarse los costos crecientes del desarrollo de software en relación al precio del hardware.

Performance:

Debido a la especificidad en la arquitectura del hardware, los sistemas basados en ASIC son los que presentan una mayor performance en comparación a los niveles de procesamiento que pueden lograrse con las FPGA. Sin embargo estos últimos presentan mejoras sustanciales respecto a los sistemas microprocesados debido principalmente a que la flexibilidad en la arquitectura permite realizar operaciones en paralelo.

Mercado de las FPGA:

Dadas las características técnicas y económicas actuales de los sistemas basados en FPGA, puede establecerse como su principal mercado de aplicación a todos los proyectos de ingeniería orientados a la producción de un reducido número de unidades (de otra manera el costo de desarrollo no tendría un efecto importante sobre el precio final) pero que, los requerimientos técnicos y su relación con los costos de desarrollo impongan la necesidad de un sistema con una performance más elevada que la de los basados en microprocesadores o microcontroladores.

## 4.2 Estructura interna de una FPGA

Tomaremos como referencia las arquitecturas introducidas por la empresa Xilinx debido a que la FPGA utilizada en este trabajo pertenece a este fabricante. Aquí se tomó como referencia al trabajo [**5**], un desarrollo detallado de este tema puede hallarse en [**3**] consultando las hojas de datos de cada dispositivo en particular. La correspondiente a la FPGA que utilizamos en este trabajo se encuentra en el anexo 11.3.

Los tres elementos básicos que componen prácticamente toda FPGA de Xilinx en la actualidad son:

CLB:

Los CLB, son bloques lógicos configurables en donde se implementan las funciones lógicas dentro de la FPGA. Cada CLB contiene un número reducido de Slices (Capas), siendo estas los elementos lógicos básicos de la FPGA pues, son capaces de implementar completamente sistemas digitales tanto combinacionales como secuenciales.

Una Slice tiene una estructura similar a la mostrada en la Figura 4-5 Figura 4-5, aquí pueden apreciarse sus elementos básicos:

- Las Tablas de entrada salida o LUTs, donde se implementan las funciones combinacionales en memorias de un bit permitiendo así retardos constantes atreves de la LUT.

- Los bloques con la lógica de conexionado o Carry, permiten interconectar las Slices dentro de un CLB (para imlementar sumadores eficientes) como se muestra en la Figura 4-4. Notar que en este caso hay 4 Slices conectadas mediante 2 cadenas que conforman las entradas (parte inferior de la figura) y salidas (parte superior) independientes de el CLB

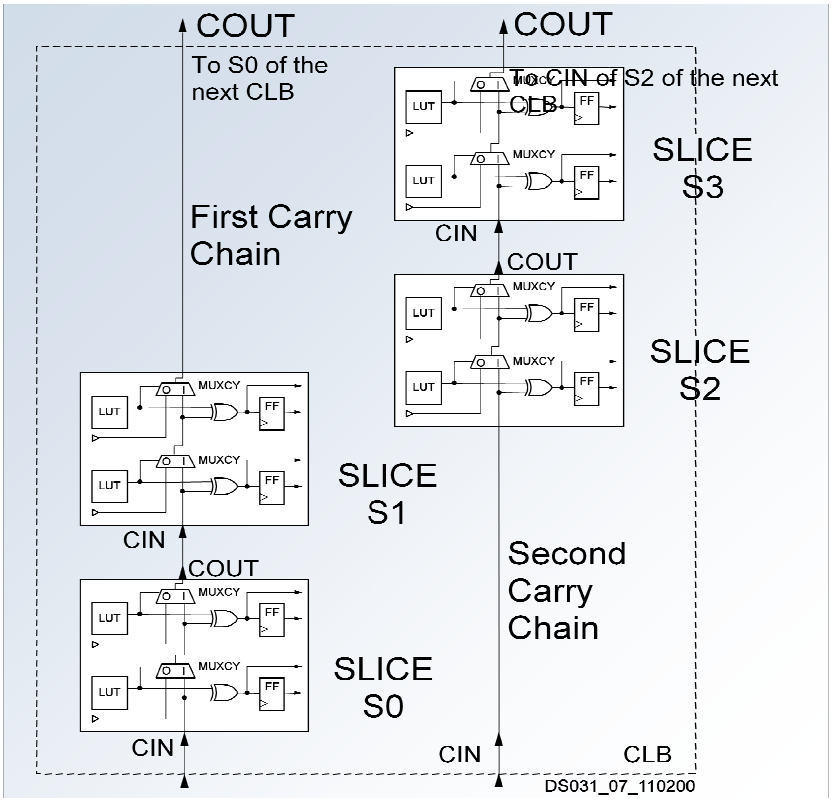



**Figura 4-4: Detalle del conexionado entre las Slices que conforman un CLB**

- Un juego de salidas registradas.

- Un juego de salidas no registradas.

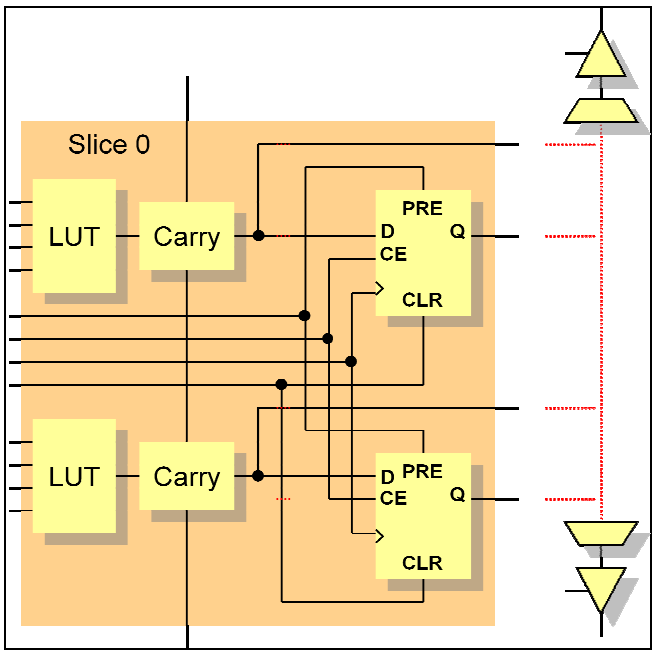



**Figura 4-5: Estructura interna simplificada de una de las Slices que conforman un CLB.**

Bloques de I/O:

Los Bloques de Entrada y Salida cumplen la función de conectar la lógica interna de la FPGA a los pines de salida de la misma. Poseen una estructura como la mostrada en la Figura 4-6, en la misma se pueden distinguir:

- Como lógica de entrada, a dos registros DDR multiplexados
- Como drivers de salida, a dos registros DDR multiplexados y dos registros DDR de 3-estados
- Los anteriores tiene fuentes de reloj (Clk) y habilitación de reloj (CE) separadas

Los Bloques de I/O, permiten conexiones mediante diferentes estándares como son: LVTTL, LVCMOS, PCI-X, GTL, etc. Además incluyen:

- Sistema de Impedancia Controlada Digitalmente, que permite implementar las terminaciones de líneas en el mismo chip.
- Resistencias de Pull-Up y Pull-Down integradas.
- Salidas rápida o lenta seleccionables.
- Etc.

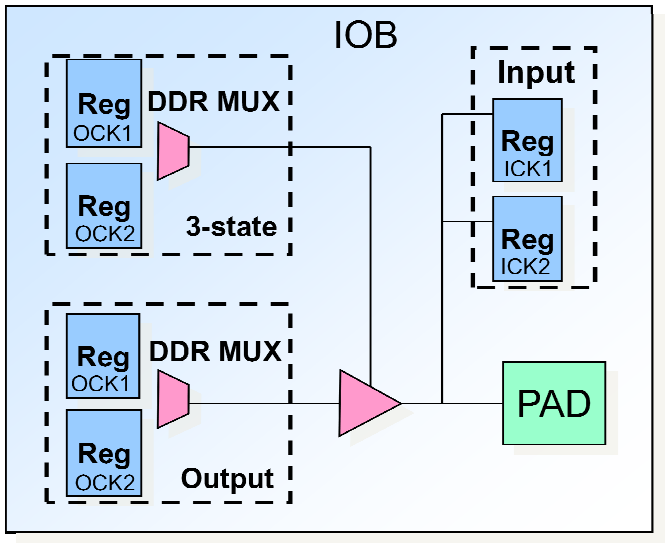



**Figura 4-6: Diagrama de un Bloque de I/O básico de una FPGA**

Recursos de Conexionado:

Con esto nos referimos a todos los mecanismos configurables que permiten interconectar los CLB entre sí de manera de poder implementar el sistema deseado.

Como puede apreciarse en la Figura 4-7 , cada CLB está conectado mediante una matriz de conexión configurable a los recursos de ruteo que corren tanto vertical como horizontalmente entre las filas y columnas de CLB.

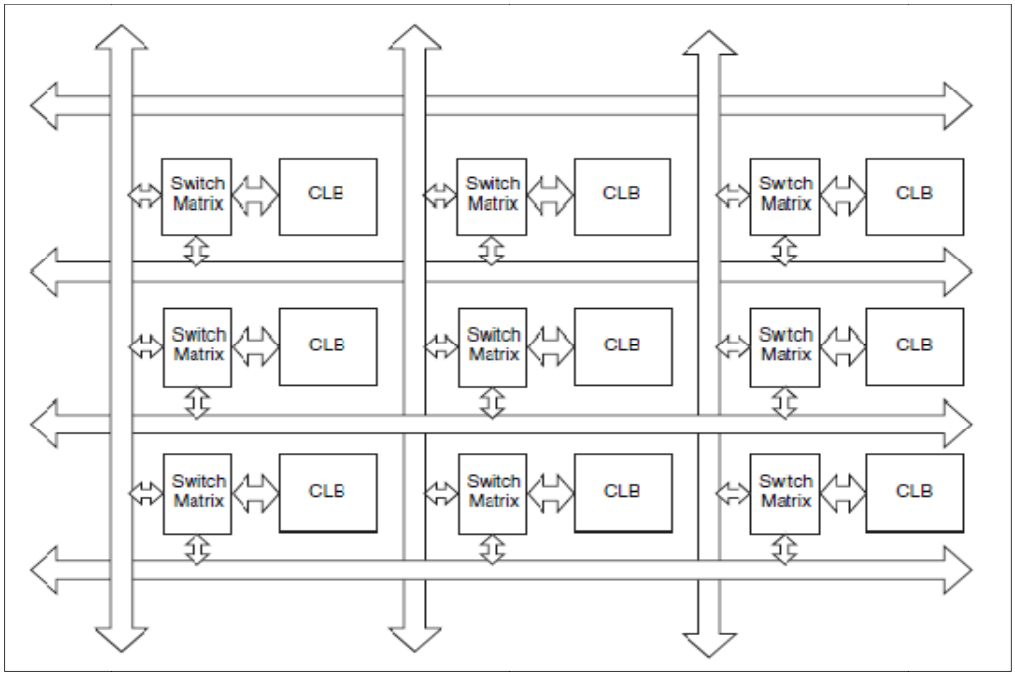



**Figura 4-7: Recursos de interconexión entre los CLB de una FGPA.**

Los diferentes recursos de conexión están definidos principalmente por su longitud. En el caso de la familia Spartan 6 de Xilinx estos son los mostrados en la Figura 4-8:

- Conexiones rápidas (Fast): Realimentan las salidas de un CLB con sus entradas.

- Conexiones simples (Single): Conectan CLB contiguos vertical u horizontalmente.

- Conexiones dobles (Double): Conectan CLB por medio en cualquier dirección.

- Conexiones cuádruples (Quad): Conectan uno de cada cuatro CLB en cualquier dirección. Este recurso reemplaza el de las Conexiones largas de las antiguas arquitecturas.

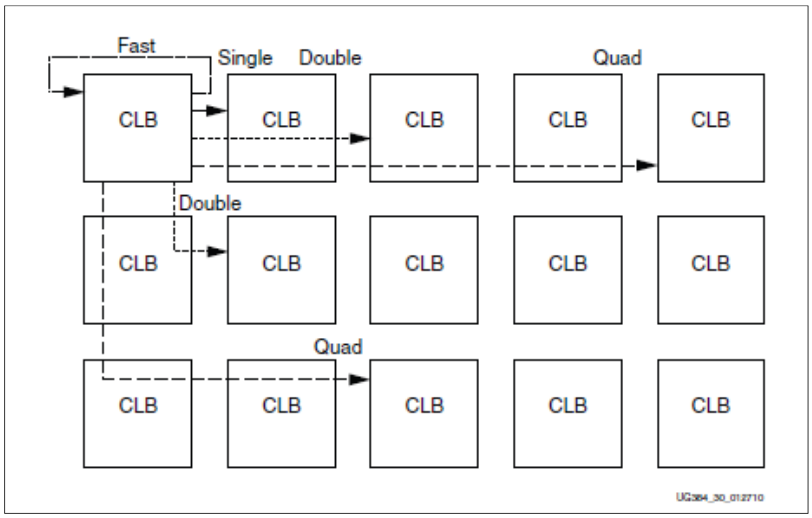



**Figura 4-8: Diferentes recursos de conexión para las FPGAs de la familia Spartan 6 de Xilinx.**

Otros recursos de Hardware:

Los elementos básicos nombrados arriba, son los mínimos necesarios que constituyen una FPGA. Sin embargo, su utilidad práctica es impulsada principalmente por el incremento sustancial de su performance, obtenido gracias a la implementación de otros recursos (de uso típicos en sistemas digitales) mediante módulos de hardware específicos embebidos.

Los recursos más comunes que pueden encontrarse actualmente son:

- Memoria RAM: Esta incluye tanto la memoria *RAM Distribuida*, que es la asociada a cada LUT, como la memoria *RAM en bloques*, implementadas en sectores especificos de la pastilla.

- Slices DSP: Estos son módulos especializados en realizar operaciones aritméticas típicas de las aplicaciones orientadas al DSP. Una discusión detallada de los módulos DSP de la FPGA en su relación a este trabajo se da en la sección 7.1.1.

- Administración del reloj: Aquí se incluyen los Digital Clock Managers o DCM, que tienen la capacidad de entregar salidas de frecuencia múltiplo o una fracción del reloj de entrada. Además proveen un control del desfasamiento mediante una realimentación PLL y un control de defesado configurable cada 90° Adicionalmente, se cuenta con múltiples buses de clk independientes.

- Interfaz para puertos: Es común también encontrar como recursos embebidos, la interfaz que implementa la comunicación con un puerto estándar, en general de alta velocidad. Por ejemplo puertos PCI express, Puertos Ethernet Gigabit, etc..

## 4.3 Ventajas de la implementación del KF en FPGA

El algoritmo iterativo del KF que en principio se desea implementar se presentó en la sección 3.4.5. Como puede verse allí las ecuaciones que deben computarse son de la Ecuación 3-51 a la Ecuación 3-55. Las repetimos aquí por conveniencia agrupando las que pueden ser computadas en paralelo:

Paso 1

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

$$P_k = (I - K_k H)P_k^-$$

Paso 2

$$\hat{x}_{k+1}^- = \Phi\hat{x}_k + u$$

$$P_{k+1}^- = \Phi P_k \Phi^T + Q$$

Paso 3

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$$

Esta es una división preliminar pues, un análisis más detallado revela aun una mayor posibilidad de paralelizar el algoritmo (que aquí no desarrollamos porque, como se verá más adelante, esta no es la versión final del algoritmo que se implementará).

De todas formas, puede justificarse la implementación del KF en una FPGA debido principalmente a que la capacidad de procesamiento en paralelo intrínseca de la FPGA permite obtener una mayor performance gracias a que:

- Permite efectuar la multiplicación (operación de gran complejidad computacional) de Matrices en paralelo y no elemento a elemento.

- Permite computar las dos expresiones dentro del paso 1 en paralelo (ídem para el Paso 2)

Además la capacidad de reconfigurabilidad de la FPGA junto a la flexibilidad de diseño del lenguaje de descripción de hardware permiten implementar diferentes plantas, sensores o IC sin necesidad de rediseñar completamente el filtro (ver sección 3.4.5).

Con todo, se obtiene un FK con un desempeño temporal mayor que el logrado en sistemas microprocesados equivalentes y que es más flexible y barato de desarrollar que uno basado en ASIC.

# 5 KF en sistemas de aritmética finita

NOTA: Todo el código presentado en la secciones 5 y 6, se realizó utilizando el software de cálculo MATLAB versión R2008a de la compañía The MathWorks Inc.

## 5.1 Conceptos fundamentales de aritmética finita

La principal característica con la que uno debe lidiar al intentar implementar el KF en un sistema computacional real, es la aritmética y memoria finitas. Esto produjo que, históricamente fuese necesario primero analizar la performance del KF bajo estas condiciones. Fruto del análisis del *comportamiento numérico* del KF, surgieron formas alternativas al algoritmo original planteado por Kalman.

En esta sección discutimos los conceptos necesarios que permiten hacer un tratamiento inicial de este problema.

### 5.1.1 Cuantización

En toda aritmética finita (ℕ) el número de elementos existentes con los que es posible operar es finito. Debido a esto siempre que se desee representar una cantidad continúa en ℕ es esperable la aparición de un *error de cuantización.*

Para el caso de sistemas digitales el error de cuantización o de redondeo depende directamente del la longitud de la palabra con la que se trabaje y del sistema de representación numérica adoptado (punto fijo, punto flotante, etc).

En base a lo anterior es que, se define el *error de redondeo* $\varepsilon$, como el máximo número que cumple con:

$$1 + \varepsilon = 1 \; en \; precisión \; de \; maquina$$

Por ejemplo, para el caso de los sistemas que cumplen el estándar IEEE de 64 bits de precisión, como es en las mayorías de las PCs actuales: $\varepsilon = 2^{-52}$.

### 5.1.2 Número de condición

El estudio inicial del comportamiento de los algoritmos computacionales en aritméticas finitas, se realiza a través del concepto de *número de condición* (CN), este surge de analizar la susceptibilidad de la solución de un sistema a pequeñas variaciones en sus entradas (vector de términos independientes o la matriz de coeficientes).

Así, dada una matriz $A$ y un problema lineal $Ax = B$, la sensibilidad de las soluciones $x$ a variaciones en $B$ está cuantificada por el CN de $A$ denotado como $Cond(A)$:

Si definimos el error relativo de la entrada como ($\|B\|$ denota la norma de B.):

$$er_B = \frac{\|\Delta B\|}{\|B\|}$$

y el error relativo de la salida como:

$$er_x = \frac{\|\Delta x\|}{\|x\|} = \frac{\|A^{-1}\Delta B\|}{\|A^{-1}B\|}$$

Entonces el CN de A es:

$$Cond(A) = \frac{\max(er_x)}{min(er_B)} = \|A\|\,\|A^{-1}\|$$

Como puede apreciarse, $Cond(A)$ cuantifica la sensibilidad de la solución del sistema a variaciones en la entrada. Debe notarse sin embargo que esto es una propiedad de la matriz de coeficientes $A$ y no fruto de un error de redondeo o del método utilizado para resolver el problema. Sin embargo cuando aparezcan los errores de redondeo (manifestados como $er_B$ por ejemplo) es el CN el que me permitirá prever el comportamiento del algoritmo (en este caso la solución del problema lineal). En definitiva expresa cuán confiable es el resultado de un algoritmo que opera con entradas susceptibles de tener error.

Puede hacerse un desarrollo análogo considerando cambios a la matriz $A$.

Nótese que el valor de $Cond(A)$ está directamente determinado por cuan "cerca" está la matriz de ser singular. Aquí nos referimos a las variaciones que son necesarias en los valores de los elementos de $A$, para que esta sea singular, en relación a la precisión de trabajo.

De esta forma, cuando se opera en una aritmética finita con un $\varepsilon$ dado, puede calcularse el máximo error relativo a la salida como el producto $\varepsilon\, Cond(A)$. Si este error es mayor que el deseado, se dice que la matriz y el sistema de ecuaciones (o algoritmo asociados) están *mal condicionados*. Si la exactitud de la salida es satisfactoria se dice en cambio que el sistema está *bien condicionado*.

## 5.2 Estabilidad numérica del KF

El concepto de *estabilidad numérica,* se refiere a la *robustez* de un algoritmo frente a los errores de redondeo. Con robustez nos referimos a la insensibilidad de un algoritmo frente a cambios en sus entradas o parámetros.

Cundo se intentó implementar el KF en las primeras computadoras digitales, se encontró que la performance del sistema se veía severamente deteriorada. Esto es que los MSE de estimación eran mucho mayores que los teóricamente predichos. Eventualmente se descubrió que este problema era producido por los errores de redondeo inherentes a la aritmética utilizada.

Para resolver este problema se comenzó a buscar diferentes implementaciones (expresiones aritméticamente equivalentes) del algoritmo original del KF que fuesen numéricamente más estables.

### 5.2.1 Factorización de matrices aplicada al KF

Una herramienta muy eficaz, cuando se intenta buscar versiones numéricamente más estables de algoritmos en los que intervienen operaciones matriciales, es utilizar el concepto de

factorización. Aquí factorizar una matriz se refiere a expresarla en términos de un producto de factores con propiedades especiales.

El uso de la factorización de matrices en algoritmos computacionales tiene varias ventajas importantes:

- Puede demostrarse que los factores de una matriz tienen un CN menor que la matriz original. Por lo que, operar con los factores producirá algoritmos numéricamente más estables.

- En general reduce el tiempo de cómputo y la cantidad de memoria requerida debido a que los factores de una matriz tienen simetrías que pueden ser explotadas en el código.

Existen diversos métodos de factorizaciones que pueden aplicarse a diferentes tipos de matrices y que, producirán factores con diferentes propiedades. Nombraremos aquí los más utilizados en el desarrollo de implementaciones más estables del KF:

- Triangularización
  - Por rotaciones de Givens
  - Por reflexiones de Householder

- Descomposición de Cholesky.

- Descomposición de Cholesky modificada, también llamada factorización UD.

- Modificaciones de Rango-1.

- Ortogonalización de Gram–Schmidt.

- Ortogonalización de Gram–Schmidt sopesada.

La aplicación de estos métodos ha dado lugar a diferentes implementaciones del algoritmo del KF, entre ellas, las más exitosas se basan en la descomposición de la matriz $P$ en términos de productos simétricos de factores triangulares. En general estos enfoques pueden agruparse dentro de tres métodos:

- Filtros de raíz-cuadrada (Square-root filters): Aquí se descompone $P$ en un producto simétrico de factores de Cholesky.
$$P = CC^T$$
- Filtros de covarianza UD: Aquí se descompone $P$ en un producto simétrico de factores de Cholesky modificados.
$$P = UDU^T$$
- Filtros de raíz-cuadrada de información: Aquí se descompone la inversa de $P$, denominada la matriz de información. Esta se utiliza en una versión alternativa del algoritmo original de Kalman denominado filtro de información (ver sección 5.3.5).

Estos métodos se aplican luego a las ecuaciones de actualización temporal y a las de actualización observacional por separado o de manera combinada. Como se verá en la sección 5.4, la implementación que utilizaremos en este trabajo, denominada de Bierman-Thornton, *está dentro del segundo tipo de los nombrados arriba*. Por esto, desarrollaremos a continuación solo los métodos de factorización necesarios para este enfoque. Para una breve descripción del resto de los métodos, explicados en el contexto de otra importante implementación denominada de Carlson-Schmidt, véase el anexo 11.4.2

Cholesky modificada o UD:

Toda matriz, simétrica y definida positiva $M_{mxm}$, puede factorizarse como sigue:

$$M = UDU^T$$

**Ecuación 5-1**

Donde:

$U_{nxn}$ es una matriz triangular superior (o inferior) unitaria. Esto es, una matriz triangular con todos los elementos de su diagonal principal iguales a 1.

$D_{nxn}$ es una matriz diagonal.

Los valores de $U$ y $D$, se hallan resolviendo el sistema de ecuaciones simultaneas planteado por la Ecuación 5-1 con los elementos de las matrices explicitados.

La principal ventaja, de este método respecto al de Cholesky original (en el que $U$ no requiere ser unitaria), es que aquí no es necesario calcular raíces cuadradas. Esto último produce un algoritmo de descomposición numéricamente más estable (notar que los factores de Cholesky, al ser triangulares unitarios, tienen determinante 1 y por ende siempre tienen inversa).

En la siguiente figura se presenta un algoritmo para computar la descomposición UD de una matriz $M_{nxn}$ dado en [**2**]:

```
    for j=n:-1:1,
        for i=j:-1:1,
            alfa=M(i,j);
            for s=j+1:n,
                alfa=alfa-U(i,s)*D(s,s)*U(j,s);
            end
            if i==j
                D(j,j)=alfa;
                U(j,j)=1;
            else
                U(i,j)=alfa/D(j,j);
            end
        end
    end
```

**Figura 5-1: Código para computar la descomposición en factores de Cholesky modificados, también llamada descomposición UD, de una matriz cuadrada simétrica y definida positiva $M_{nxn}$.**

Ortogonalización de Gram-Schmidt sopesada:

El proceso de ortogonalización de Gram-Schmidt se usa para encontrar un conjunto ortogonal de vectores $\{b_i\}_1^n$, que son una combinación lineal de un conjunto de partida de vectores linealmente independientes $\{a_i\}_1^n$. Este método se utiliza comúnmente para, dada una base de un espacio vectorial, obtener una nueva base ortonormal.

Para adecuarlo a nuestro caso, tomamos una matriz $A_{nxm}$, y la escribimos como un conjunto de $n$ vectores $m$ dimensionales, $\{a_i\}_1^n$. Entonces el algoritmo de Gram-Schmidt asocia a $A$ una matriz $L_{nxn}$ tal que:

$$A = BL$$

Donde los vectores que definen $B$, a saber $\{b_i\}_1^n$ son ortogonales entre sí con respecto a funciones de peso definidas en la diagonal de la matriz diagonal de peso. $D_w$.

De esta manera:

$$b_i^T D_w b_j = \begin{cases} \|b_i\|^2 \; si \; i = j \\ 0 \quad si \; i \neq j \end{cases}$$

Entonces $B$ se denomina una *matriz ortogonal* y cumple con:

$$B^T D_w B = D_{\|b\|^2}$$

Donde $D_{\|b\|^2}$es una matriz diagonal de $nxn$ con los elementos de su diagonal principal en forma ascendente igual a las normas $\|b_1\|,^2 \|b_2\|^2, \dots, \|b_n\|^2$ respectivamente. Para $B$ ortonormal $D_{\|b\|^2} = I$.

En la ortogonalización de Gram-Schmidt no sopesada o normal $D_w = I$.

Una explicación detallada del procedimiento de cálculo necesario para hallar $B$ y $L$, así como una explicación más desarrollada del proceso de ortogonalización se hallan en el anexo 11.4.1.

La ortogonalización de Gram-Schmidt modificada que se nombra aquí no es numéricamente estable en la práctica. Es por esto que en realidad se utilizará una versión modificada del algoritmo de la misma atribuida a Björck [**6**] (La cual no presenta modificaciones conceptuales importantes).

## 5.3 Problemas de la implementación del KF en aritmética finita

En la presente sección, introduciremos los problemas que surgen al tratar de implementar el algoritmo original del KF en un sistema real con aritmética finita tal como la presentada en la sección 5.1. Como se verá hay dos grandes impedimentos que impulsarán luego el desarrollo de algoritmos alternativos (presentados en la sección 5.4) al original planteado por Kalman. Estos son, así como los aspectos que trataremos de ellos, los siguientes:

- Comportamiento numérico:

- Fuentes de mal condicionamiento del KF
- No asociatividad de las operaciones aritméticas fundamentales
- Perdidas de exactitud en algoritmos iterativos y oscilaciones indeseadas

- Cantidad de recursos computacionales requeridos:
  - Calculo de la matriz inversa en la ecuación de la ganancia
  - Procesamiento del vector de mediciones una componente a la vez
  - Tratamiento del ruido de medición correlacionado

Estos problemas impiden que se cumplan los requisitos necesarios bajo los cuales se derivó la ganancia optima de Kalman. En otras palabras el funcionamiento del KF en un sistema práctico se aleja del teórico y por esto, ya no puede afirmarse que la estimación siga siendo óptima.

Estos inconvenientes son los que tratan de minimizar las implementaciones alternativas del algoritmo del KF. Si bien el desarrollo original de Kalman se lo consideró como ideal para su implementación en sistemas digitales, es gracias a estos avances en materia de estabilidad numérica que el KF ha llegado a tener el gran impacto tecnológico que presenta en la actualidad.

### 5.3.1 Fuentes de mal condicionamiento del KF

Algunas de las fuentes que contribuyen al mal condicionamiento del algoritmo del KF son:

- Grandes incertezas en los valores de las matrices: $\Phi, Q, H$ o $R$.

- Grandes rangos de variación de los valores de las matrices y vectores: $\Phi, Q, H, R, z$ o $x$, que, pueden resultar por ejemplo de una mala selección del escalado o las unidades utilizadas.

- Mal condicionamiento de la matriz intermedia: $(HPH^T + R)$, que, debe ser invertida en la ecuación de la KG.

- Grandes dimensiones de las matrices, que se reflejan en un incremento potencial (al cuadrado o al cubo) de la cantidad de operaciones aritméticas y, por ende, de la cantidad de redondeos a efectuar.

- Una precisión de maquina deficiente ($\varepsilon$ grande).

### 5.3.2 No asociatividad de las operaciones aritméticas fundamentales

Cuando se trabaja en aritmética finita, la suma algebraica, la multiplicación de números y la división de números, no cumplen en general con la ley de asociatividad.

Este es un tema no muy abordado en la literatura introductoria del KF directamente pero se vuelve importante al trabajar con sistemas mal condicionados, como son las implementaciones en aritmética de punto fijo del KF utilizadas en este trabajo.

Un análisis de esta característica, en el contexto de la aplicación del KF en aritmética de punto fijo, se establece en la sección 6.2.5

### 5.3.3 Pérdida de exactitud en algoritmos iterativos

Claramente la degradación en la performance del KF que introducen los errores de cuantización, se ve incrementada por el hecho de que su algoritmo es iterativo.

Así, como se verá en la sección 6.2, el hecho de trabajar con un tamaño de palabra fijo hace que deba readaptarse el resultado en cada operación básica (multiplicación, suma, etc.) que se realiza, resultando esto en una degradación acumulativa de la exactitud del resultado.

Ruido de cuantización:

Debe tenerse en claro que la operación de cuantización es una operación netamente no lineal y por esto, al implementar cualquier algoritmo en una aritmética finita, hace que este pase a comportarse también de esta manera. Debido a esto, pueden presentarse comportamientos extraños incluso en los algoritmos más dóciles.

Estos efectos se denotan en conjunto como *Ruido de Cuantización* debido a que, su tratamiento se realiza en forma estadística (por más que el proceso es netamente determinístico), introduciéndose una fuente de ruido con densidad de probabilidad uniforme asociada a cada operación que se realiza y propagando los resultados a través de todo el algoritmo.

Uno de los efectos más característicos del ruido de cuantización son las *oscilaciones de ciclo limitado.* Estas pueden producir por ejemplo que un sistema LTI tenga salida de estado estacionario, incluso en ausencia de entradas.

Por ejemplo considere el sistema discreto con entrada $u[k]$ y salida $y[k]$ dado por:

$$y[k] = -0.625\, y[k-1] + u[k]$$

Si se lo hace operar en aritmética infinita, su respuesta con entrada nula y IC igual a $y[0] = -\frac{3}{8}$ es:

$$y[k] = \frac{3}{8}, -\frac{1}{4}, \frac{1}{8}, 0, 0, 0, \ldots$$

Si ahora se lo implementa en una aritmética de 4 bits con redondeo, la salida es:

$$y[k] = \frac{3}{8}, -\frac{1}{4}, \frac{1}{8}, -\frac{1}{8}, \frac{1}{8}, -\frac{1}{8}, \ldots$$

Estudio del comportamiento numérico del KF:

A la hora de implementar un algoritmo en una aritmética finita, es la relación entre:

- El $\varepsilon$ del sistema utilizado.

- El máximo y el mínimo número representables

- El rango de variación de los operandos (entradas).

La que determinará la manifestación de los todos los efectos arriba descriptos y en consecuencia el tratamiento que debe realizarse para una implementación exitosa.

Debido a que en general los datos de entrada son desconocidos (salvo su estadística), habrá típicamente dos opciones para el análisis del comportamiento de un algoritmo:

1. Tratamiento teórico: Este implica aplicar la teoría de tratamiento estadístico del ruido de cuantización. Presenta la ventaja de que se garantiza el correcto funcionamiento para todo el rango de los valores de entrada. Pero tiene la contra de ser altamente complejo en especial para algoritmos complicados, además este análisis es específico para cada aplicación. Para un desarrollo detallado en el tema ver [**7**].

2. Tratamiento experimental: Otra posibilidad muy utilizada actualmente, debido a la gran cantidad de herramientas computacionales que la facilitan, es estudiar el comportamiento de nuestro algoritmo funcionando en una aritmética finita en particular, mediante pruebas (corridas numéricas) exhaustivas que incluyan una buena cantidad de todos los posibles casos. Es viable incluso en algunas situaciones probar el desempeño del algoritmo para todos los casos que admiten las entradas.

En este trabajo adoptamos un tratamiento basado en 2 (el cual se desarrolla en la sección 1) debido principalmente a que un análisis como el requerido por 1 no se justifica dado los objetivos propuestos.

### 5.3.4 Calculo de la matriz inversa en la ecuación de la ganancia

Un punto crítico en el cálculo de una iteración del KF, es sin duda la computación de la inversa que se presenta en la Ecuación 3-51 de la KG que aquí repetimos por practicidad:

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$$

Para un sistema con $n$ estados internos de los cuales se miden $l$, la cantidad $(H P_k^- H^T + R)$ resulta ser una matriz cuadrada $R^*{}_{lxl}$.

Aquí dos importantes dificultades a sortear son:

- Complejidad computacional: La complejidad de la inversión de $R^*$ es $O(l^3)$ utilizando el método de eliminación Gauss-Jordan (Hay otros métodos que reducen la complejidad pero no por debajo de $O(l^{2.376})$). Esto es un claro impedimento debido a la gran cantidad de operaciones a realizar cuando $l$ crece.

- Mal condicionamiento de $R^*{}_{lxl}$: En general los algoritmos típicos para invertir matrices no poseen buena estabilidad numérica como para ser implementados en la práctica debido principalmente a la cantidad de operaciones (en especial las divisiones) que intervienen.

### 5.3.5 Procesamiento del vector de mediciones una componente a la vez

Una solución óptima para resolver los problemas anteriores, solo aplicable cuando los ruidos de medición de cada estado no están correlacionados entre sí, es la de procesar el vector de mediciones una componente a la vez y de esta manera eliminar la inversión de la matriz $R^*$. Para desarrollar esta metodología primero debe introducirse una versión alternativa del algoritmo original del KF.

KF de información:

Existe una forma algebraicamente equivalente al algoritmo original de Kalman que se denomina, por razones que veremos más adelante, filtro de información. La derivación detallada de este filtro puede encontrarse en [**1**] aquí solo necesitamos las expresiones finales alternativas del algoritmo iterativo, estas se presentan en la Figura 5-2.

Las principales características de la versión alternativa del KF son:

- No se necesita $K_k$ para actualizar la $P_k^-$.
- Se trabaja con la inversa de $R_k$.
- Se actualiza observacionalmente solo $P_k^{-1}$, por lo que son necesarias 2 inversiones de $nxn$ por cada iteración. Esto hace que esta versión no sea adecuada para sistemas de gran orden.

Si bien a primera vista esta versión del KF no parece tener ninguna ventaja, resulta muy útil en los casos en los que la estimación inicial tiene gran incertidumbre. En el caso extremo en que no se conoce la estimación inicial y por ende se asume $P_0 \to \infty$, el algoritmo original del KF no podría aplicarse mientras que esta versión si pues, operaría con la inversa de la incertidumbre (la información)${P_0}^{-1} \to 0$.

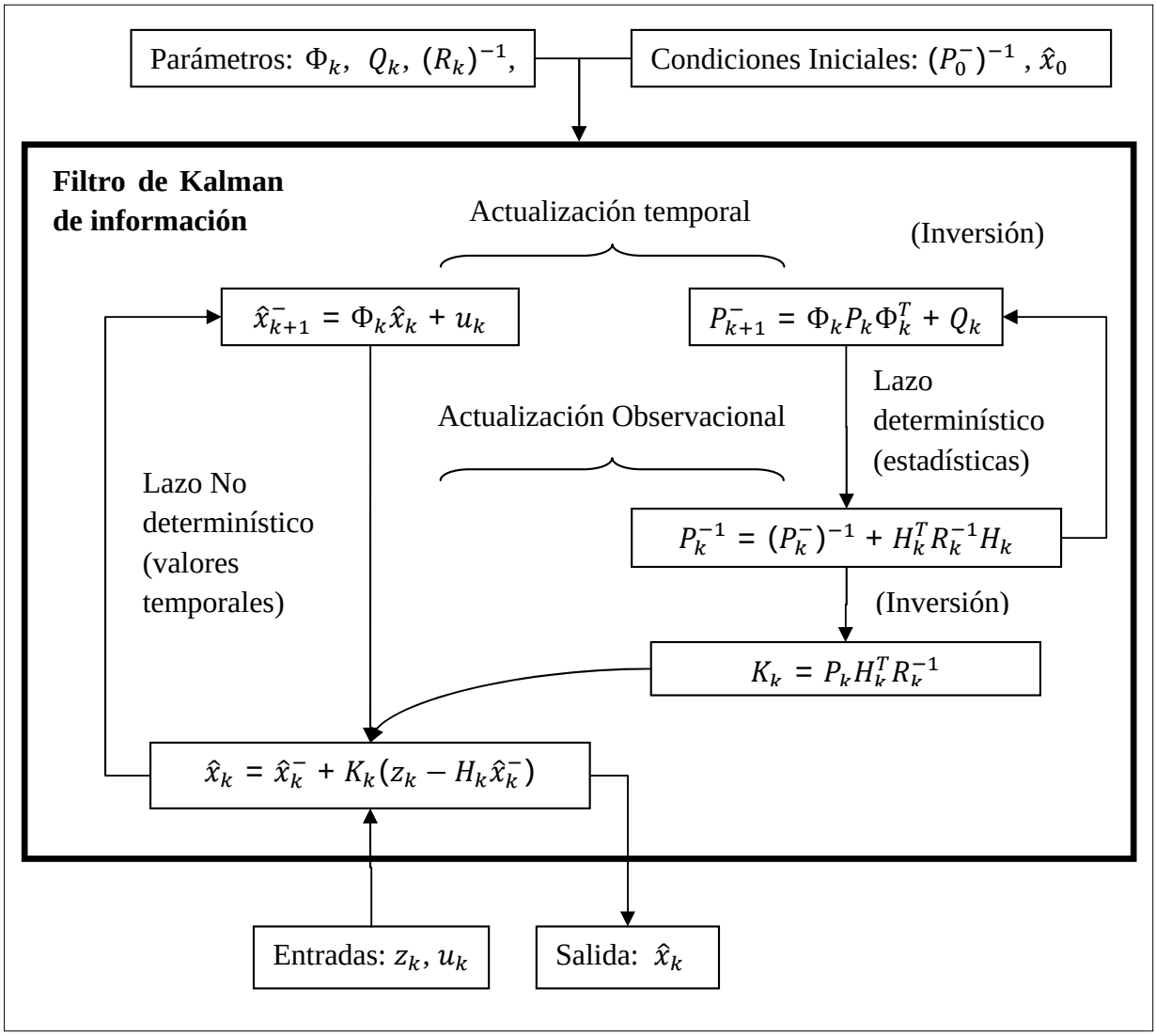



**Figura 5-2: Algoritmo alternativo (aunque algebraicamente equivalente) del KF denominado también Filtro de información**

Procesamiento Secuencial de las mediciones en el KF de información:

Ahora si, estamos en condiciones de derivar el método de procesamiento escalar del vector de mediciones. Para esto, utilizaremos la expresión de actualización temporal de la matriz de información:

$$P_k^{-1} = (P_k^-)^{-1} + H_k^T R_k^{-1} H_k$$

Luego, asumiendo ruido no correlacionado ($R$ diagonal) y explicitando las matrices, tenemos:

$$P_k^{-1} = (P_k^-)^{-1} + \begin{bmatrix} h_{11} & \cdots & h_{l1} \\ \vdots & \ddots & \vdots \\ h_{1n} & \cdots & h_{ln} \end{bmatrix} \begin{bmatrix} {r_{11}}^{-1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & {r_{ll}}^{-1} \end{bmatrix} \begin{bmatrix} h_{11} & \cdots & h_{1n} \\ \vdots & \ddots & \vdots \\ h_{l1} & \cdots & h_{ln} \end{bmatrix}$$

Reescribimos esta expresión particionado la $H_k$ en vectores fila que denotaremos $H_{k,1}, H_{k,2}, \ldots, H_{k,l}$:

$$P_k^{-1} = (P_k^-)^{-1} + \begin{bmatrix} H_{k,1}^T & \cdots & H_{k,l}^T \end{bmatrix} \begin{bmatrix} {r_{11}}^{-1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & {r_{ll}}^{-1} \end{bmatrix} \begin{bmatrix} H_{k,1} \\ \vdots \\ H_{k,l} \end{bmatrix}$$

Luego, desarrollando el producto, se tiene:

$$P_k^{-1} = (P_k^-)^{-1} + H_{k,1}^T {r_{11}}^{-1} H_{k,1} + \cdots + H_{k,l}^T {r_{ll}}^{-1} H_{k,l}$$

**Ecuación 5-2**

En esta expresión, puede interpretarse como que $P_k^{-1}$ se va *construyendo gradualmente* a medida que se van procesando sucesivamente las mediciones correspondientes a los ruidos $r_{11}, r_{22}, \ldots r_{ll}$ (Esto refuerza la interpretación de $P_k^{-1}$como la matriz de información). Al procesar $l$ mediciones se obtiene la versión completa de $P_k^{-1}$

Tomemos los sucesivos términos en la Ecuación 5-2 para definir un conjunto de covarianzas (inversas) parciales:

$$P_{k,1}^{-1} = (P_k^-)^{-1} + H_{k,1}^T {r_{11}}^{-1} H_{k,1}$$

$$P_{k,2}^{-1} = P_{k,1}^{-1} + H_{k,2}^T {r_{22}}^{-1} H_{k,2}$$

$$\vdots$$

$$P_{k,l}^{-1} = P_{k,l-1}^{-1} + H_{k,l}^T {r_{ll}}^{-1} H_{k,l} = P_k^{-1}$$

De esta forma, dividiremos el computo de cada actualización observacional completa del filtro, en $l$ *Pasos* (en cada uno se utilizará una covarianza parcial), solo luego de completar todos los Pasos es que se obtendrá la estimación completa del estado en un instante $k$ determinado.

Este procedimiento se lleva a cabo de la siguiente manera, detallaremos el primero de los Pasos:

1. Se computa la primera $P_k^{-1}$ parcial:

$$P_{k,1}^{-1} = (P_k^-)^{-1} + H_{k,1}^T {r_{11}}^{-1} H_{k,1}$$

   - Aquí se utiliza solo la primera fila de $H_k$: $H_{k,1}$ y la covarianza a priori completa obtenida en el último Paso de la iteración anterior del filtro: $P_k^-$

2. Siguiendo los pasos del algoritmo en la Figura 5-2, se puede calcular una *ganancia parcial* $K_{k,1}$:

$$K_{k,1} = P_{k,1} H_{k,1}^T r_{11}^{-1}$$

   - Aquí se utiliza solo el primer elemento de la diagonal de $R_k^{-1}$: $r_{11}^{-1}$

3. Esta ganancia es utilizada para realizar una actualización temporal *parcial* de la estimación:

$$\hat{x}_{k,1} = \hat{x}_k^- + K_{k,1}\left(z_1 - H_{k,1}\hat{x}_k^-\right)$$

   - Aquí se utiliza solo el primer elemento del vector de mediciones de $z_k$: $z_{k,1}$ y la estimación a priori completa obtenida en el último Paso de la iteración anterior: $\hat{x}_k^-$. Además como antes se usa $H_k^1$.

4. Finalmente se realiza una proyección temporal trivial (pues se está operando siempre para conseguir la estimación completa en el mismo instante k), mediante:

$$\left(P_{k,2}^-\right)^{-1} = P_{k,1}^{-1}$$

$$\hat{x}_{k,2}^- = \hat{x}_{k,1}$$

Luego se comienza de nuevo desde el punto 1 y se efectúa el Paso 2 .Se sigue de esta forma hasta completar los $l$ Pasos. En el último de estos no se realiza la actualización temporal trivial pues ya se tienen los valores completos para el instante actual:

$$\hat{x}_k = \hat{x}_{k,l}$$

$$P_k^{-1} = P_{k,l}^{-1}$$

$$K_k = K_{k,l}$$

Por lo que se procede a realizar la actualización temporal común de estos valores para luego pasar a la siguiente iteración.

Procesamiento Secuencial de las mediciones en el algoritmo original de Kalman:

Si bien el procedimiento se desarrolló para el KF de información, los conceptos vistos son validos también para la versión original del KF que es la de mayor uso.

Por esto colocamos aquí, respetando la notación introducida en el aparatado anterior, el procedimiento necesario para efectuar el procesamiento del vector de mediciones de a una componente a la vez, utilizando el algoritmo original del KF. El mismo se presenta, de una forma análoga a la Figura 3-5, en la Figura 5-3.

Nótese que, en la ecuación de cálculo de la ganancia, el termino a invertir $(H_{k,m}P_{k,m}^-H_{k,m}^T + r_{mm})$ es ahora un escalar eliminando así los problemas descriptos en la sección 5.3.4.

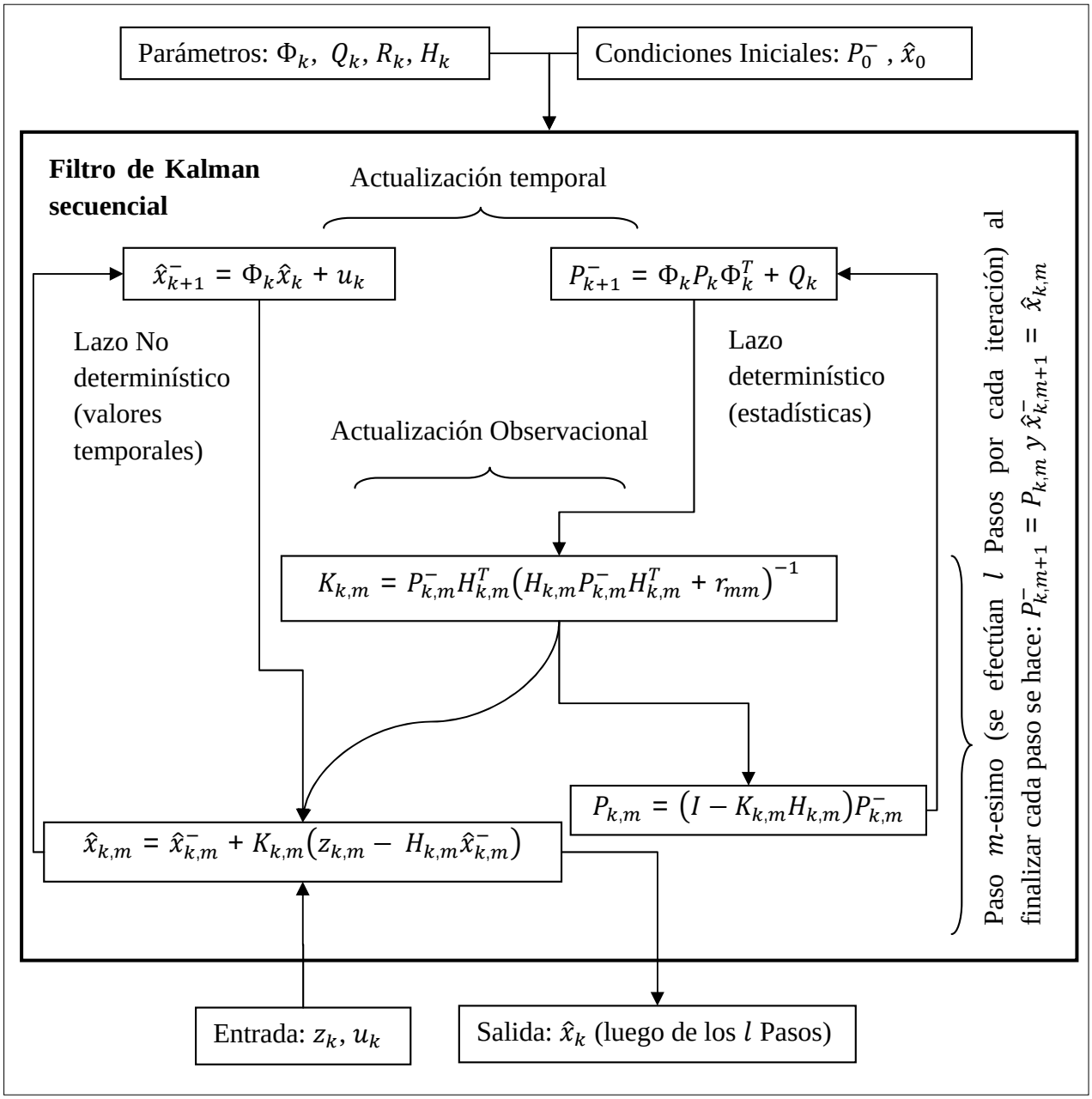



**Figura 5-3: Procedimiento para operar en el algoritmo original del KF procesando secuencialmente el vector de mediciones.**

A fin de obtener una formulación más cercana a la implementación, se reemplazan las actualizaciones temporales triviales mostradas en la Figura 5-3: $P_{k,m+1}^- = P_{k,m}$ y $\hat{x}_{k,m+1}^- = \hat{x}_{k,m}$ en las ecuaciones de actualización observacional. Con esto el algoritmo recursivo para computar los $l$ de los Pasos queda:

$$K_{k,m} = P_{k,m-1} H_{k,m}^T \left(H_{k,m} P_{k,m-1} H_{k,m}^T + r_{mm}\right)^{-1}$$

$$P_{k,m} = \left(I - K_{k,m} H_{k,m}\right) P_{k,m-1}$$

$$\hat{x}_{k,m} = \hat{x}_{k,m-1} + K_{k,m}\left(z_{k,m} - H_{k,m}\hat{x}_{k,m-1}\right)$$

Como comentario final diremos que, si bien el desarrollo anterior fue presentado para el procesamiento de un elemento del vector de mediciones a la vez (o sea un escalar), es también extensible al caso en el que la matriz $R_k$ es diagonal por bloques:

$$R_k = \begin{bmatrix} R_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & R_{ll} \end{bmatrix}$$

Donde los $R_{11}, R_{22}, \dots R_{ll}$ son bloques cuadrados de elementos en general no nulos. Este caso se da físicamente cuando solo un grupo de mediciones tienen ruidos correlacionados (por ejemplo al tomar mediciones redundantes de una misma cantidad).

### 5.3.6 Tratamiento del ruido de medición correlacionado.

La principal limitación del método para procesar secuencialmente las mediciones es que, para poder implementarlo, los ruidos de estas no deben estar correlacionados. Esto es, la matriz $R$ debe ser diagonal (al menos por bloques)

Existe sin embargo, un procedimiento que puede aplicarse al caso general con ruido correlacionado de manera de decorrelacionarlos y obtener así una versión equivalente del sistema que me permita aplicar el procesamiento secuencial de la mediciones.

Partamos de un caso en el que $R$ es no diagonal:

$$z_k = H_k x_k + v_k$$

**Ecuación 5-3**

Factorizemos $R$ mediante el método de Cholesky modificado:

$$R_k = U_k D U_k^T$$

Aquí como sabemos $D_k$ es una matriz diagonal y es la que utilizaremos en reemplazo de $R_k$. Para esto, deberá operarse con un juego modificado de mediciones:

$$\acute{z}_k = U_k^{-1} z_k$$

Reemplazando la Ecuación 5-3 en esta expresión:

$$\acute{z}_k = U_k^{-1}(H_k x_k + v_k)$$

$$\acute{z}_k = U_k^{-1} H_k x_k + U_k^{-1} v_k$$

$$\acute{z}_k = \acute{H}_k x_k + \acute{v}_k$$

En esta nueva ecuación de salida, el ruido de medición no está correlacionado, pues:

$$\acute{R}_k = E\langle \acute{v}_k \acute{v}_k^T \rangle$$

$$\acute{R}_k = E\langle (U_k^{-1} v_k)(U_k^{-1} v_k)^T \rangle$$

$$\acute{R}_k = U_k^{-1} E\langle v_k v_k^T \rangle (U_k^T)^{-1}$$

$$\acute{R}_k = U_k^{-1} R_k (U_k^T)^{-1}$$

$$\acute{R}_k = U_k^{-1} U_k D U_k^T (U_k^T)^{-1} = D$$

Entonces, dado un sistema con ruido correlacionado lo único que debe hacerse es:

1. Computar: $R_k = U_k D U_k^T$

2. Computar: $\acute{H}_k = U_k^{-1} H_k$. Notar que no hace falta hallar $U_k^{-1}$ pues al ser esta triangular unitaria puede hallarse $\acute{H}_k$ de manera más rápida y más estable resolviendo el sistema $U_k \acute{H}_k = H_k$ por métodos alternativos (como el de sustitución reversa)

3. Computar: $\acute{z}_k = U_k^{-1} z_k$. Ídem al caso anterior.

4. Reemplazar la ecuación de salida por: $\acute{z}_k = \acute{H}_k x_k + \acute{v}_k$ y utilizar $\acute{R}_k = D$. Notar que $\acute{v}_k = U_k^{-1} v_k$ sigue siendo una secuencia aleatoria con la misma media que $v_k$.

## 5.4 Diferentes implementaciones del KF

Con el objetivo de disminuir los efectos indeseados producidos por el mal condicionamiento de un problema, es que surgen dos grandes corrientes dentro de las implementaciones alternativas al algoritmo original del KF. Estas se diferencian básicamente en el método de factorización de matrices que utilizan.

Como se mencionó en la sección 5.2.1, por un lado aparecen los *filtros de raíz cuadrada* y por el otro los *filtros UD*. Ambos filtros basan su implementación en una reformulación de las ecuaciones de actualización mediante la expresión factorizada de la matriz *P*

Aquí trataremos dos de las diversas implementaciones que se han desarrollado del KF. Estas son las que han resultado más exitosas luego de ser aplicadas en numerosos casos prácticos [**2**].

Cada una de estas implementaciones se compone de un par de algoritmos, cada uno de los cuales implementan una de las actualizaciones (temporal u observacional). Esto se detalla en la siguiente tabla.

Notese que tanto el algoritmo de Bierman como el de Carlson, utilizan el método de procesamiento secuencial del vector de mediciones a fin de mejorar aun más la estabilidad numérica y reducir la complejidad computacional neta.

| **Algoritmos de Factorización** / **Tipo de filtro** | **Actualización observacional** $(P_k^- \rightarrow P_k)$ $(\hat{x}_k^- \rightarrow \hat{x}_k)$ | **Actualización temporal** $(P_k \rightarrow P_{k+1}^-)$ $(\hat{x}_k \rightarrow \hat{x}_{k+1}^-)$ |
|---|---|---|
| **Raíz cuadrada** (Cholesky) | **Carlson** | **Schmidt** (Triangularización: Rotaciones de Givens o Reflexiones de Householder) |
| **UD** (Cholesky modificado) | **Bierman** | **Thornton** (Ortogonalización: Gram-Schmidt sopesada) |

**Tabla 5-1: Detalle de dos implementaciones diferentes del filtro del KF así como los métodos de factorización de matrices que utilizan.**

Selección de la implementación a utilizar:

La principal diferencia entre las dos implementaciones del KF que aquí desarrollamos es el método de factorización que utilizan. Así en la Carlson-Schmidt será necesario computar raíces cuadradas. Esto es un problema tanto desde el punto de vista de la estabilidad numérica como de la complejidad computacional. Carlson-Schmidt tiene sin embargo, algunos beneficios si se dispone de capacidad de computo paralelo (debido a que la triangularización de Givens puede paralelizarse).

En este trabajo se pretende sintetizar el KF en un dispositivo de lógica reconfigurable (con capacidad de computo paralelo) operando con una aritmética de punto fijo (con un gran $\varepsilon$). Como se asume que debido a la aritmética a utilizar la estabilidad numérica de la implementación a elegir es un factor más crítico que la performance temporal del sistema, es que decide utilizar la opción de Bierman-Thotnon.

### 5.4.1 Implementación de Carlson-Schmidt

Un desarrollo detallado de esta implementación, que hemos elaborado en base a lo expuesto en [**2**], se encuentra en el anexo 11.4.2.

### 5.4.2 Implementación de Bierman-Thornton

Actualización observacional de Bierman:

NOTA: En el desarrollo de este apartado se omitirán los subíndices que indican el tiempo discreto por fines de claridad. Notar que esto no dificulta el entendimiento de las expresiones pues Bierman realiza la actualización observacional, por lo que todas las cantidades intervinientes están evaluadas en el mismo instante $k$.

La forma convencional para lograr la actualización en la matriz de covarianza del error cometido de estimación a priori $P^-$ es:

$$P = (I - KH)P^-$$

Distribuyendo $P^-$ y reemplazando $K$:

$$P = P^- - \frac{P^- H^T H P^-}{r + H P^- H^T}$$

**Ecuación 5-4**

Se debe tener en cuenta que $P^{-T} = P^-$ ya que dicha matriz es simétrica, en tanto que $r$ es una de las componentes de la diagonal de la matriz $R$ de covarianza del ruido de la medición $v$ (ver sección 5.3.5).

Las matrices de covarianza en las incertidumbres a posteriori y a priori se pueden factorizar de la siguiente manera:

$$P = UDU^T$$

**Ecuación 5-5**

$$P^- = U^- D^- U^{-T}$$

**Ecuación 5-6**

Reemplazando la Ecuación 5-5 y la Ecuación 5-6 en la Ecuación 5-4:

$$UDU^T = U^- D^- U^{-T} - \frac{U^- D^- U^{-T} H^T H U^- D^- U^{-T}}{r + H U^- D^- U^{-T} H^T}$$

**Ecuación 5-7**

Tomando la siguiente igualdad:

$$v = U^{-T} H^T$$

**Ecuación 5-8**

Se deduce entonces que:

$$v^T = H U^-$$

**Ecuación 5-9**

Donde $v$ es un vector que tiene la dimensión del vector de estado $n$ (no confundir con el vector de ruido de medición que también se denota $v$).

Reemplazando la Ecuación 5-9 y la Ecuación 5-8 en la Ecuación 5-7 se tiene:

$$UDU^T = U^- D^- U^{-T} - \frac{U^- D^- v v^T D^- U^{-T}}{r + v^T D^- v}$$

Sacando factor común la premultiplicación de $U^-$ y la postmultiplicación de $U^{-T}$ se obtiene:

$$UDU^T = U^- \left[ D^- - \frac{D^- v v^T D^-}{r + v^T D^- v} \right] U^{-T}$$

**Ecuación 5-10**

La Ecuación 5-10 contiene el término no factorizado siguiente:

$$D^- - D^- v\,[r + v^T D^- v]^{-1}\, v^T D^-$$

**Ecuación 5-11**

Aplicando la factorización de Cholesky modificada $(UD)$ a la Ecuación 5-11 se obtiene:

$$B\acute{D}B^T = D^- - D^- v\,[r + v^T D^- v]^{-1}\, v^T D^-$$

**Ecuación 5-12**

Donde $B$ es un factor triangular unitario.

Reemplazando la Ecuación 5-12 en la Ecuación 5-10 se obtiene:

$$UDU^T = U^-\left[B\acute{D}B^T\right]U^{-T} = [U^- B]\acute{D}[B^T U^{-T}]$$

**Ecuación 5-13**

De donde se puede deducir que:

$$U = U^- B$$

$$D = \acute{D}$$

Son los factores de la matriz de covarianza de incertidumbre de estimación a posteriori.

Por lo tanto, para la actualización observacional de los factores $UD$ de la matriz de covarianza $P$ es suficiente encontrar un método que sea numéricamente estable y eficiente para la factorización $UD$ de la expresión matricial $D^- - D^- v\,[r + v^T D^- v]^{-1}\, v^T D^-$ donde $v = U^{-T} H^T$ es un vector columna.

La actualización del vector de estados es bastante directa, se realiza utilizando la expresión del algoritmo original (Ecuación 3-43):

$$\hat{x} = \hat{x}^- + K(z - H\hat{x}^-)$$

Donde la ganancia se obtienen de remplazar en su expresión original (Ecuación 3-44) la $P_k^-$ factorizada (Ecuación 5-5) y expresar esto utilizando la Ecuación 5-8:

$$K = U^- D^- U^{-T} H^T (HU^- D^- U^{-T} H^T + r)^{-1}$$

$$K = U^- D^- v(v^T D^- v + r)^{-1}$$

Actualización temporal de Thornton:

De la Ecuación 3-29 se sabe que:

$$Q_k = GQG^T$$

**Ecuación 5-14**

En general, se tiene que el ruido $w$ es autocorrelacionado, consecuentemente la matriz de covarianza de dicho ruido $Q$ es no diagonal y se debe proceder a su diagonalización a través de la factorización $UD$ de la siguiente manera:

$$UD_qU^T = Q$$

**Ecuación 5-15**

Reemplazando la Ecuación 5-15 en la Ecuación 5-14 se llega a:

$$Q_k = GUD_qU^TG^T$$

Y siendo $\acute{G} = GU$:

$$Q_k = \acute{G}D_q\acute{G}^T$$

**Ecuación 5-16**

Se conoce que la actualización temporal de la matriz de covarianza de incertidumbre en la estimación a posteriori $P$ es:

$$P_{k+1}^- = \Phi_k P_k \Phi_k^T + Q_k$$

**Ecuación 5-17**

Aplicando las factorizaciones dadas en la Ecuación 5-5 y la Ecuación 5-6 y reemplazando la Ecuación 5-16 en la Ecuación 5-17 se obtiene:

$$U_{k+1}^- D_{k+1}^- U_{k+1}^{-T} = \Phi_k U_k^- D_k^- U_k^{-T} \Phi_k^T + \acute{G}_k D_q \acute{G}_k^T$$

**Ecuación 5-18**

Se puede llegar a la Ecuación 5-18 conformando las siguientes matrices:

$$A = \begin{bmatrix} U_k^T \Phi_k^T \\ \acute{G}_k^T \end{bmatrix}_{2nxn}$$

$$D_w = \begin{bmatrix} D_k & 0 \\ 0 & D_{Q_k} \end{bmatrix}_{2nx2n}$$

Y realizando el siguiente producto con estas:

$$\begin{aligned} A^T D_w A &= \begin{bmatrix} U_k^T \Phi_k^T \\ \acute{G}_k^T \end{bmatrix}^T \begin{bmatrix} D_k & 0 \\ 0 & D_{Q_k} \end{bmatrix} \begin{bmatrix} U_k^T \Phi_k^T \\ \acute{G}_k^T \end{bmatrix} = \begin{bmatrix} \Phi U_k & \acute{G}_k \end{bmatrix} \begin{bmatrix} D_k & 0 \\ 0 & D_{Q_k} \end{bmatrix} \begin{bmatrix} U_k^T \Phi_k^T \\ \acute{G}_k^T \end{bmatrix} \\ &= \begin{bmatrix} \Phi_k U_k^- D_k^- & \acute{G} D_q \end{bmatrix} \begin{bmatrix} U_k^T \Phi_k^T \\ \acute{G}_k^T \end{bmatrix} = \Phi_k U_k^- D_k^- U_k^{-T} \Phi_k^T + \acute{G}_k D_q \acute{G}_k^T \\ &= U_{k+1}^- D_{k+1}^- U_{k+1}^{-T} \end{aligned}$$

Con lo cual se llega a:

$$U_{k+1}^{-} D_{k+1}^{-} U_{k+1}^{-T} = A^T D_w A$$

**Ecuación 5-19**

Ortogonalizando la matriz $A$ de la siguiente manera:

$$A = BL$$

**Ecuación 5-20**

Reemplazando la Ecuación 5-20 en la Ecuación 5-19:

$$U_{k+1}^{-} D_{k+1}^{-} U_{k+1}^{-T} = (BL)^T D_w BL = L^T B^T D_w BL$$

Sabiendo que el término $B^T D_w B$ en la ecuación anterior corresponde a una ortogonalización sopesada en la cual se redefine el producto interno como:

$$b_i^T D_w b_j = \begin{cases} \|b_i\|^2 & si\ i = j \\ 0 & si\ i \neq j \end{cases}$$

Y teniendo en cuenta que:

$$B^T D_w B = D_{\|b\|^2}$$

Se llega a la siguiente igualdad:

$$U_{k+1}^{-} D_{k+1}^{-} U_{k+1}^{-T} = L^T D_{\|b\|^2} L$$

$$U_{k+1}^{-} = L^T$$

$$D_{k+1}^{-} = D_{\|b\|^2}$$

La actualización temporal del vector de estados se realiza utilizando la expresión original, Ecuación 3-49.

### 5.4.3 Algoritmo iterativo de la implementación del KF- BT

En la Figura 5-4 se presentan en forma sintética las expresiones de la implementación completa de KF-BT desarrolladas en la sección anterior.

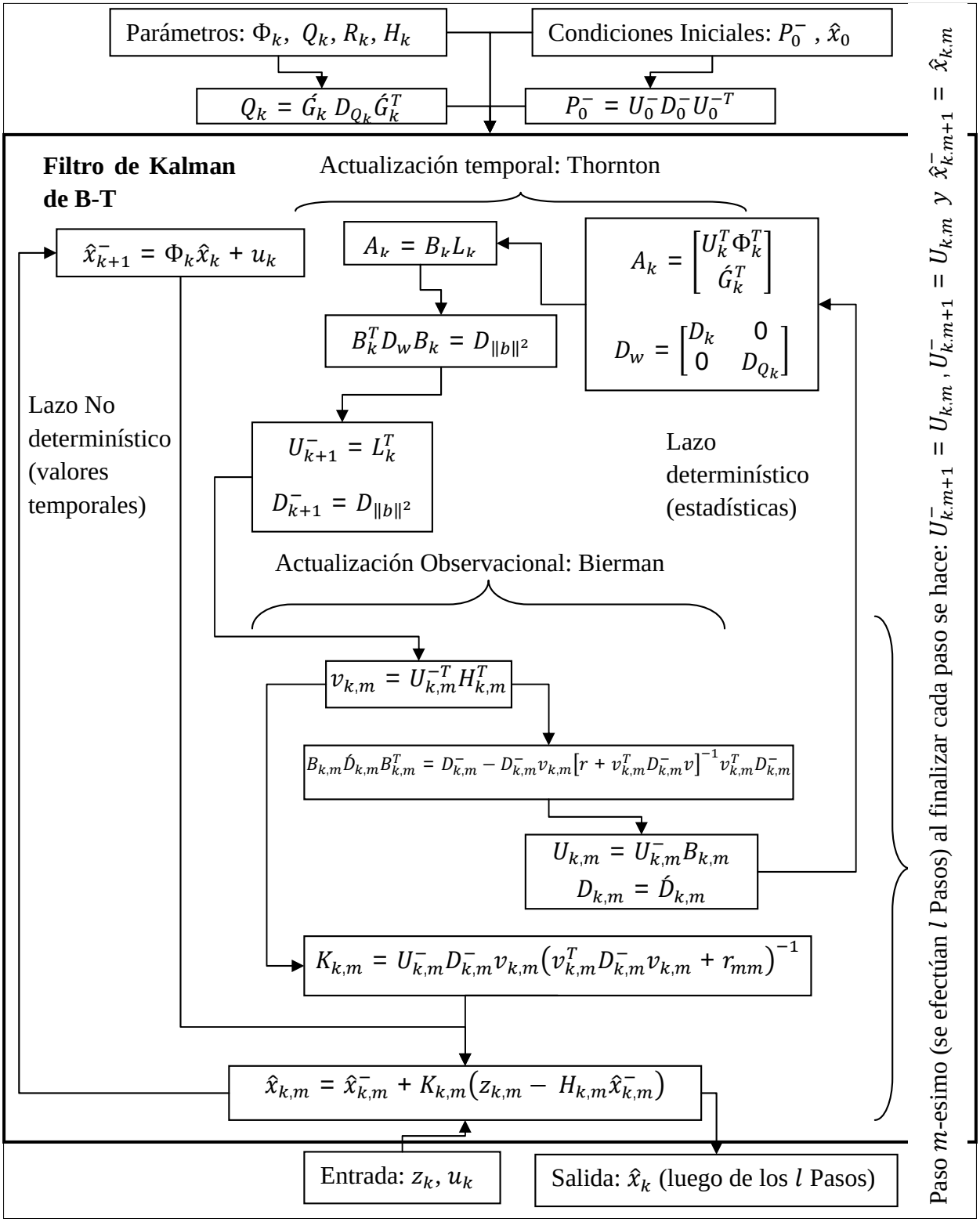



**Figura 5-4: Esquema completo con todas las ecuaciones y el flujo de datos de la implementación de BT del KF.**

# 6 KF-BT en aritmética de punto fijo

## 6.1 Definición de la aritmética de punto fijo

Introducción:

La aritmética de punto fijo (FP) es una aritmética finita en la que la posición del punto decimal es fija dentro de la palabra de trabajo. La principal desventaja de este tipo de aritmética es que para un tamaño de palabra dado, el $\varepsilon$ que se logra es grande en relación a otros sistemas de representación numérica como por ejemplo el de punto flotante.

Este gran error de redondeo hace que los problemas mencionados en la sección 1 cobren relevancia, principalmente el de la estabilidad numérica del algoritmo a implementar. Como se verá, para controlar esto, se recurre entre otras cosas al escalado de las variables intervinientes. Sin embargo, el proceso de escalado es laborioso y consume mucho tiempo, es por esto que históricamente en cuanto el hardware lo permitió se migraron eventualmente las aplicaciones digitales a punto flotante.

No obstante, hay un gran número de problemas en la actualidad (celulares, DSP, OpenGL, etc.) en los que se sigue aplicando FP, debido principalmente a que este requiere mucho menos hardware para ser sintetizado que otros sistemas de representación numérica. [**8**]

La aritmética de FP fue la primera en el que se implemento exitosamente el algoritmo del KF cuando se trabajó en el proyecto Apolo, invirtiéndose cerca del 30% del tiempo de programación en el escalado de los algoritmos.

Definición Formal:

Los elementos necesarios para definir completamente una aritmética de PF son:

1. La longitud de la palabra
2. La posición del punto decimal
3. El método para tratar el Overflow
4. El método para tratar el Redondeo
5. Escalado

### 6.1.1 Longitud de la Palabra y posición del punto

Como es sabido, una palabra de $WL$ bits puede adoptar $2^{WL}$ estados distintos. Según el sistema de representación numérica que se adopte podrán representarse con esta misma palabra diferentes grupos de números.

En la aritmética de FP, para una longitud de la parte fraccional dada por $FL$, el grupo de números representables $T$, es el definido por:

$$T = \left\{ \frac{p}{2^{FL}} / -2^{WL-1} \leq p \leq 2^{WL-1} - 1, p \in \mathbb{Z} \right\}$$

**Ecuación 6-1**

Esto es, serán representables los números racionales en los que el numerador y el denominador cumplan con las exigencias descriptas arriba. Nótese que la forma especial que adopta el denominador implica que para todos los números representables la posición del punto decimal será la misma, de aquí el nombre de aritmética de punto fijo.

Con esto los $WL$ bits ($b$) de la palabra de trabajo quedarán, asumiendo una representación de los números negativos en complemento a 2, interpretados según se detalla en la siguiente figura:

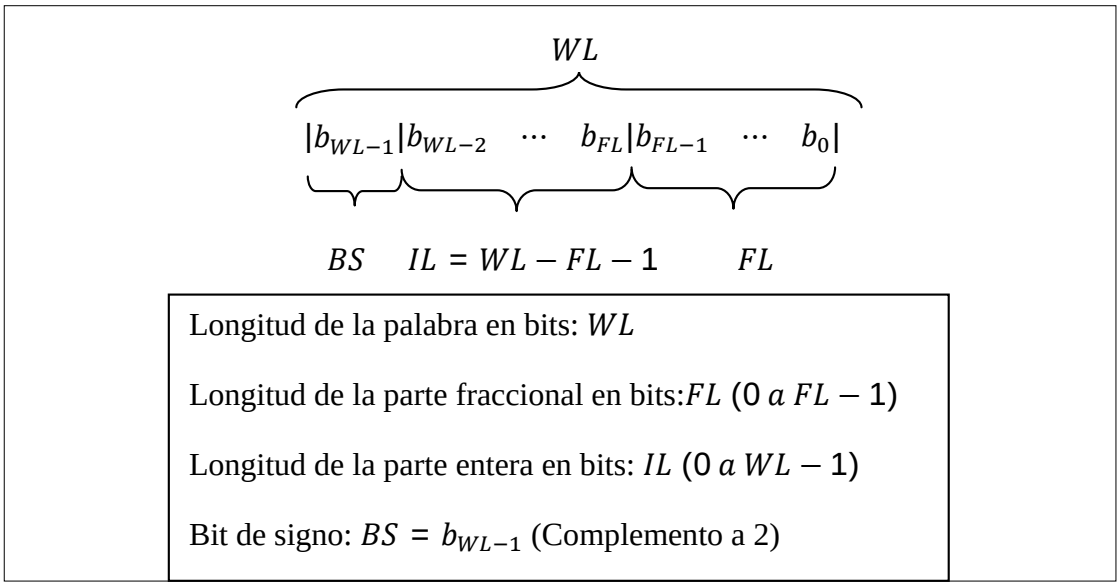



**Figura 6-1: Interpretación de todos los bits de una palabra de longitud *WL* asumiendo una aritmética de FP con signo y con parte fraccional de longitud *FL*.**

La distribución de los bits de la palabra mostrada en la Figura 6-1 se denotará en forma sintética mediante la notación:

$$O(IL, FL)$$

Además se denotará al contenido en bits de una palabra o fracción de palabra de longitud $L$ como $C(L)$. Por ejemplo, el contenido de la parte fraccional de la palabra de la Figura 6-1 es:

$$C(FL) = b_{FL-1} \quad \cdots \quad b_0$$

Finalmente, se expresará el valor en decimal correspondiente a una palabra o fracción de palabra de longitud L como $D(L)$. Por ejemplo, el valor decimal de $O(IL, FL)$, es:

$$D(IL, FL)$$

Estas notaciones abreviadas, facilitarán el análisis de la modificación de la aritmética que se presenta en las operaciones algebraicas básicas en FP (Sección 6.1.2).

Error de redondeo:

El error de redondeo $\varepsilon$ es igual al menor número positivo representable:

$$\varepsilon = \frac{1}{2^{FL}}$$

**Ecuación 6-2**

Máximos y Mínimos:

Cualquier palabra binaria interpretada como en la Figura 6-1 tendrá un valor decimal (elemento de $T$) asociado definido por:

$$D(IL, FL) = \frac{1}{2^{FL}}\left[-2^{WL-1}BS + \sum_{k=0}^{WL-2} 2^k b_k\right]$$

**Ecuación 6-3**

El valor decimal máximo posible que puede adoptar la palabra, será igual a la cantidad de combinaciones para los números positivos multiplicado por $\varepsilon$:

$$max[\,D(IL, FL)] = (2^{WL-1} - 1)\varepsilon = 2^{IL} - \frac{1}{2^{FL}}$$

**Ecuación 6-4**

Análogamente, el valor mínimo será la cantidad de combinaciones para los números negativos multiplicado por $\varepsilon$:

$$min[D(IL, FL)] = -2^{WL-1}\varepsilon = -2^{IL}$$

**Ecuación 6-5**

Notar que el valor mínimo es absolutamente más grande que el máximo, esto se debe a que la combinación correspondiente al valor 0 ocupa una de las combinaciones asignadas a los números positivos. Esta característica, la posición del 0 en la escala de valores representables, se denomina el *Bias* de la aritmética y puede ser modificado al aplicar un método de *Escalado* (ver la sección 6.1.4).

### 6.1.2 Tratamiento del Overflow

Como se verá más adelante (Sección 6.2), existen diversas situaciones en donde será necesario convertir un número expresado en una aritmética de FP en particular a otra.

Tomemos el caso en que se desea convertir entre aritméticas que tienen la misma longitud de la parte fraccional $FL_1 = FL_2$, pero diferentes de la parte entera:

$$O_1(IL_1, FL_1) \rightarrow O_2(IL_2, FL_2)$$

1) Para el caso en que $IL_1 < IL_2$, la conversión es directa pues, no hay posibilidad de desbordamiento u Overflow. Simplemente se copian los bits y se hace el rellenado (Padding) según lo especificado en la siguiente figura:

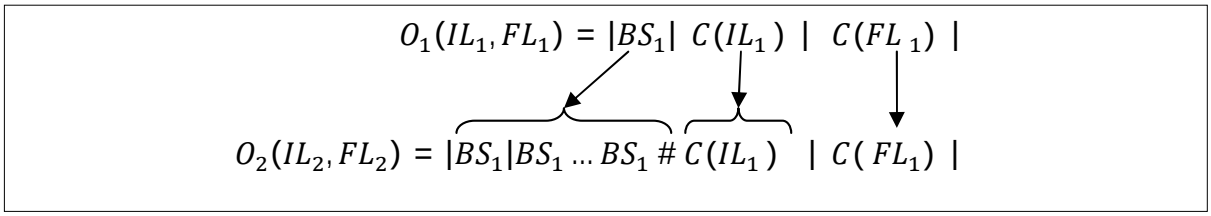



**Figura 6-2: Procedimiento para expresar un número dado en una aritmética con $O_1(IL_1, FL_1)$ a otra con $O_2(IL_2, FL_2)$ para el caso en que $FL_1 = FL_2$ y $IL_1 < IL_2$. O sea sin posibilidad de Overflow.**

Aquí el operador # implica la unión o concatenación de dos cadenas de bits.

Al procedimiento expresado en la figura anterior lo escribiremos como una única ecuación de transformación de la siguiente forma:

$$O_1(IL_1, FL_1) \rightarrow O_2(IL_2, FL_2)$$

$$BS_2 = BS_1$$

$$C(IL_2) = BS_1 \dots BS_1 \# C(IL_1)$$

$$C(FL_2) = C(FL_1)$$

**Ecuación 6-6**

2) Para el caso en que $IL_1 > IL_2$, tendremos dos opciones más según sea:
   a) $D(IL_1) \leq \max [D(IL_2)]$ y $D(IL_1) \geq \min [D(IL_2)]$, entonces no hay Overflow y la conversión es directa:

$$O_1(IL_1, FL_1) \rightarrow O_2(IL_2, FL_2)$$

$$BS_2 = BS_1$$

$$C(IL_2) = C(IL_1)_{IL_2}$$

$$C(FL_2) = C(FL_1)$$

**Ecuación 6-7**

Aquí $C(IL_1)_{IL_2}$ significa tomar los $IL_2$ bits menos significativos de $C(IL_1)$

   b) $D(IL_1) > max\ [D(IL_2)]$ ó $D(IL_1) < min\ [D(IL_2)]$, por lo que hay Overflow y el número no puede ser representado en la aritmética de destino. En este caso debe decidirse con que método se tratará el Overflow, los más comunes son:

      I. Saturar (Saturate): Aquí simplemente se toma como representación al máximo o mínimo valor posible (de la aritmética destino) según sea el numero original positivo o negativo respectivamente, así:

Si $BS_1 = 0$ (positivo) entonces se toma $max[\ D(IL_2, FL_2)]$:

$$O_1(IL_1, FL_1) \rightarrow O_2(IL_2, FL_2)$$

$$BS_2 = 0$$

$$C(IL_2) = 1 \dots 1$$

$$C(FL_2) = 1 \dots 1$$

**Ecuación 6-8**

Si $BS_1 = 1$ (negativo) entonces se toma $min[\,D(IL_2, FL_2)]$:

$$O_1(IL_1, FL_1) \rightarrow O_2(IL_2, FL_2)$$

$$BS_2 = 1$$

$$C(IL_2) = 0 \dots 0$$

$$C(FL_2) = 0 \dots 0$$

**Ecuación 6-9**

II. Recortar (Wrap): Aquí se toma como representación al número original recortado para que "entre" en la palabra destino. Este método requiere menos hardware que el de saturar pues solo hay que desechar los bits más significativos de la parte entera. Notar sin embargo que en general el valor numérico que se obtiene diferirá considerablemente del de origen y además el comportamiento es menos "predecible" que si se satura:

$$O_1(IL_1, FL_1) \rightarrow O_2(IL_2, FL_2)$$

$$BS_2 = BS_1$$

$$C(IL_2) = C(IL_1)_{IL_2}$$

$$C(FL_2) = C(FL_1)$$

**Ecuación 6-10**

### 6.1.3 Tratamiento del Redondeo

Tomemos ahora el caso en que se desea convertir entre aritméticas con la misma longitud de la parte entera $IL_1 = IL_2$, pero diferente de la parte fraccional:

$$O_1(IL_1, FL_1) \rightarrow O_2(IL_2, FL_2)$$

1. Para el caso en que $FL_1 < FL_2$, no hay necesidad de redondeo pues el numero puede ser representado en la aritmética destino simplemente rellenando los bits menos significativos (aunque esto no es una práctica común debido a que los bits rellenados dejan de ser cifras significativas):

$$O_1(IL_1, FL_1) \rightarrow O_2(IL_2, FL_2)$$

$$BS_2 = BS_1$$

$$C(IL_2) = C(IL_1)$$

$$C(FL_2) = C(FL_1)\#\mathrm{BS}_1 \ldots \mathrm{BS}_1$$

**Ecuación 6-11**

2. Para el caso en que $FL_1 > FL_2$, tendremos dos opciones más según sea:
   a. El valor asociado al digito no nulo menos significativo de $D(FL_1)$ es $> \varepsilon_2$, entonces no hay necesidad de un método de redondeo y el número se representa en la aritmética destino simplemente tomando una versión recortada del mismo (de todas maneras se pierde exactitud por perderse los ceros significativos a la derecha del numero original):

$$O_1(IL_1, FL_1) \rightarrow O_2(IL_2, FL_2)$$

$$BS_2 = \mathrm{BS}_1$$

$$C(IL_2) = C(IL_1)$$

$$C(FL_2) = C(FL_1)^{FL_2}$$

**Ecuación 6-12**

Donde $C(FL_1)^{FL_2}$ implica tomar los $FL_2$bits más significativos de $C(FL_1)$.

   b. El valor asociado al digito no nulo menos significativo de $D(FL_1)$ es $< \varepsilon_2$, por lo que el número no puede ser representado en la aritmética de destino y es necesario aplicar algún método de redondeo. Cada uno de ellos presentará una relación performance numérica vs hardware necesario para síntesis diferente [**9**]:

      I. Round: Este método es el que introduce menos error (igual a la mitad del $\varepsilon$) pero el que requiere de un hardware más complejo para su implementación.
      Cada número se aproxima al valor representable más cercano, el valor obtenido será mayor o menor que el original dependiendo del caso (no hay una dirección de redondeo predeterminada). En la siguiente figura se muestra este procedimiento:

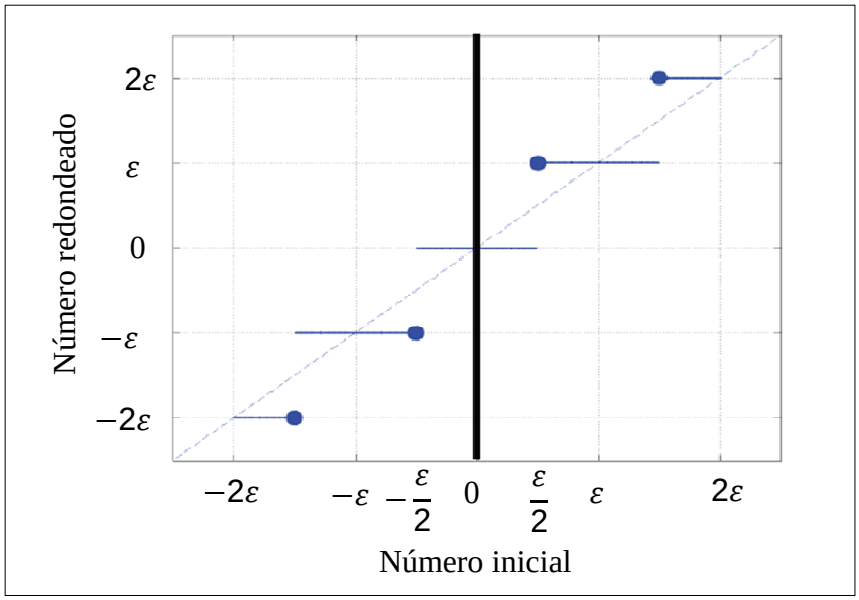



**Figura 6-3: Método de redondeo Round**

II. Fix: Aquí se realiza el redondeo aproximando cada valor al más cercano en dirección hacia el cero. Este método requiere menos hardware que el anterior pero, como se ve en la siguiente grafica, el error es mayor (igual a $\varepsilon$):

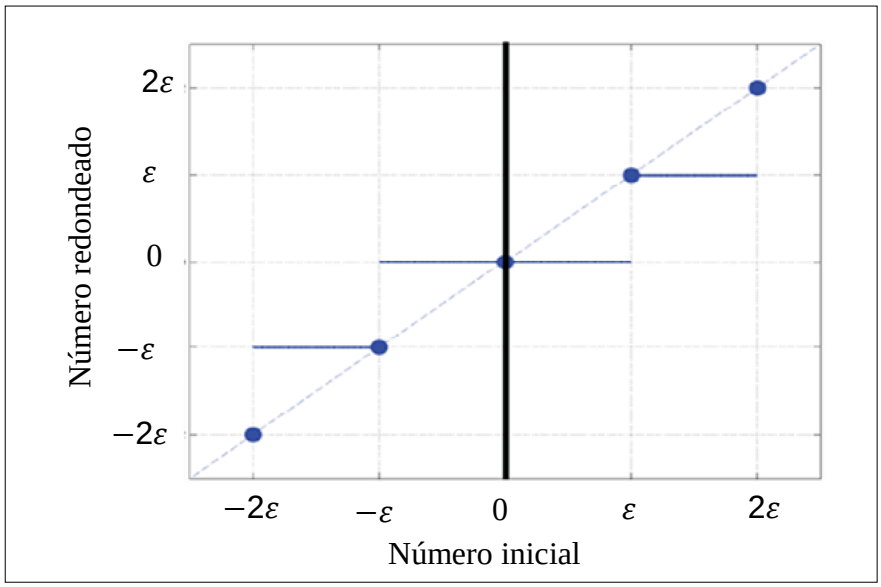



**Figura 6-4: Método de redondeo Fix**

III. Floor (o truncado): Aquí se redondea cada número hacia el valor más cercano en dirección hacia menos infinito. Nótese que esto da un redondeo asimétrico (con offset) que puede traer problemas en ciertas aplicaciones.

*Sin embargo, este método es el que menor hardware requiere pues solo se necesita desechar (truncar) los bits menos significativos (si se trabaja en complemento a 2) para redondear*.

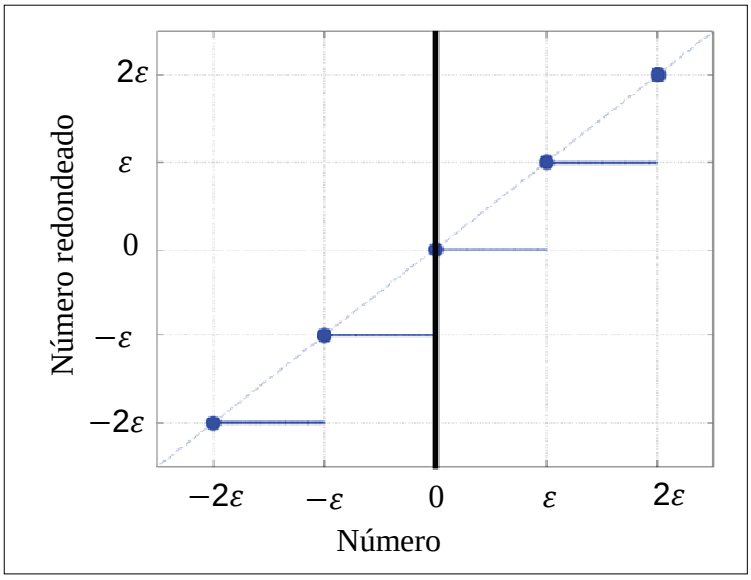



**Figura 6-5: Método de redondeo Floor o Truncado**

Utilizando nuestra notación, para aplicar este método de redondeo debe hacerse:

$$O_1(IL_1, FL_1) \rightarrow O_2(IL_2, FL_2)$$

$$BS_2 = \mathrm{BS}_1$$

$$C(IL_2) = C(IL_1)$$

$$C(FL_2) = C(FL_1)^{FL_2}$$

**Ecuación 6-13**

IV. Ceil: Aquí se redondea cada número hacia el valor más cercano en dirección hacia más infinito. Nótese que esto da un redondeo asimétrico (con offset) que puede traer problemas en ciertas aplicaciones.

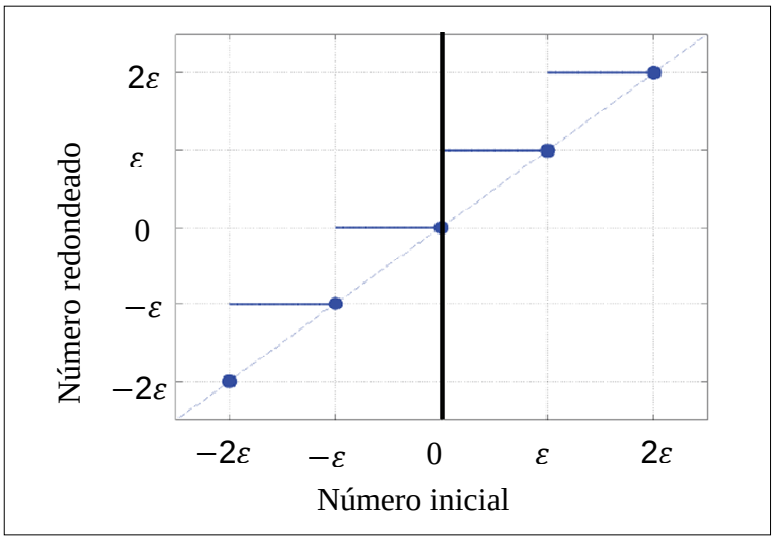



**Figura 6-6: Método de redondeo Ceil**

### 6.1.4 Escalado

Si se conocen los rangos de valores que puede adoptar las variables intervinientes en el funcionamiento de un algoritmo, es posible que se de el caso en que utilizar una aritmética centrada en el cero (Sin Bias) desaproveche hardware. Por ejemplo si se sabe que los valores numéricos positivos que pueden presentarse no van a superar el 30% del máximo número positivo representable, entonces puede aprovecharse ese 70% en desuso para poder manejar números negativos más grandes:

Otro punto importante, es la elección de las unidades de las variables a manejar. Esto permite también ahorrar hardware. Por ejemplo si todos los valores de una variable de longitud rondan los miles de metros, entonces, es mejor usar el km como unidad de medida.

Las dos modificaciones de la aritmética mencionadas en los párrafos anteriores, se implementan mediante el escalado. Aplicar un escalado significa aplicar una transformación lineal antes de almacenar una cantidad en una variable (también deben escalarse las constantes), o sea que el valor almacenado y la cantidad que representa ya no son iguales. Luego, cuando se va interpretar el valor almacenado en una variable debe aplicarse la transformación inversa.

La relación entre una cantidad $\alpha$ y su valor almacenado en la maquina $\beta$ es:

$$\alpha = Scale\ \beta + Bias$$

**Ecuación 6-14**

Donde la constante $Bias$ define la posición del 0 de la aritmética.

Por otro lado, la constante $Scale$ define las unidades a utilizar (modifica el rango de alcance de la aritmética) y es a su vez establecida por dos factores:

$$Scale = F2^{E}$$

**Ecuación 6-15**

Con $F$ estandarizada a los valores: $F \in R / 1 \leq F < 2$. Nótese que toda aritmética de punto fijo es en realidad una escalada pues la relación entre el número entero (que es en definitiva lo único físicamente almacenable) $\beta 1$ y su interpretación como un numeró con punto fijo $\alpha 1$ es $\alpha = 2^{-FL}\ \beta$. Sin embargo con la aplicación del escalado dado por la Ecuación 6-14, se considera el uso de un factor de escala más general dado por la Ecuación 6-15.

En la Figura 6-7, se muestra un ejemplo de escalado en el que se logra cubrir todo el rango de valores de una variable de interés dada mediante la aplicación a una aritmética de FP (en principio insuficiente) de un escalado. El mismo es determinado en este caso por las constantes $Scale = 4\ (F = 1, E = 2)$ y $Bias = -0.7 \max [D(IL, FL)]$.

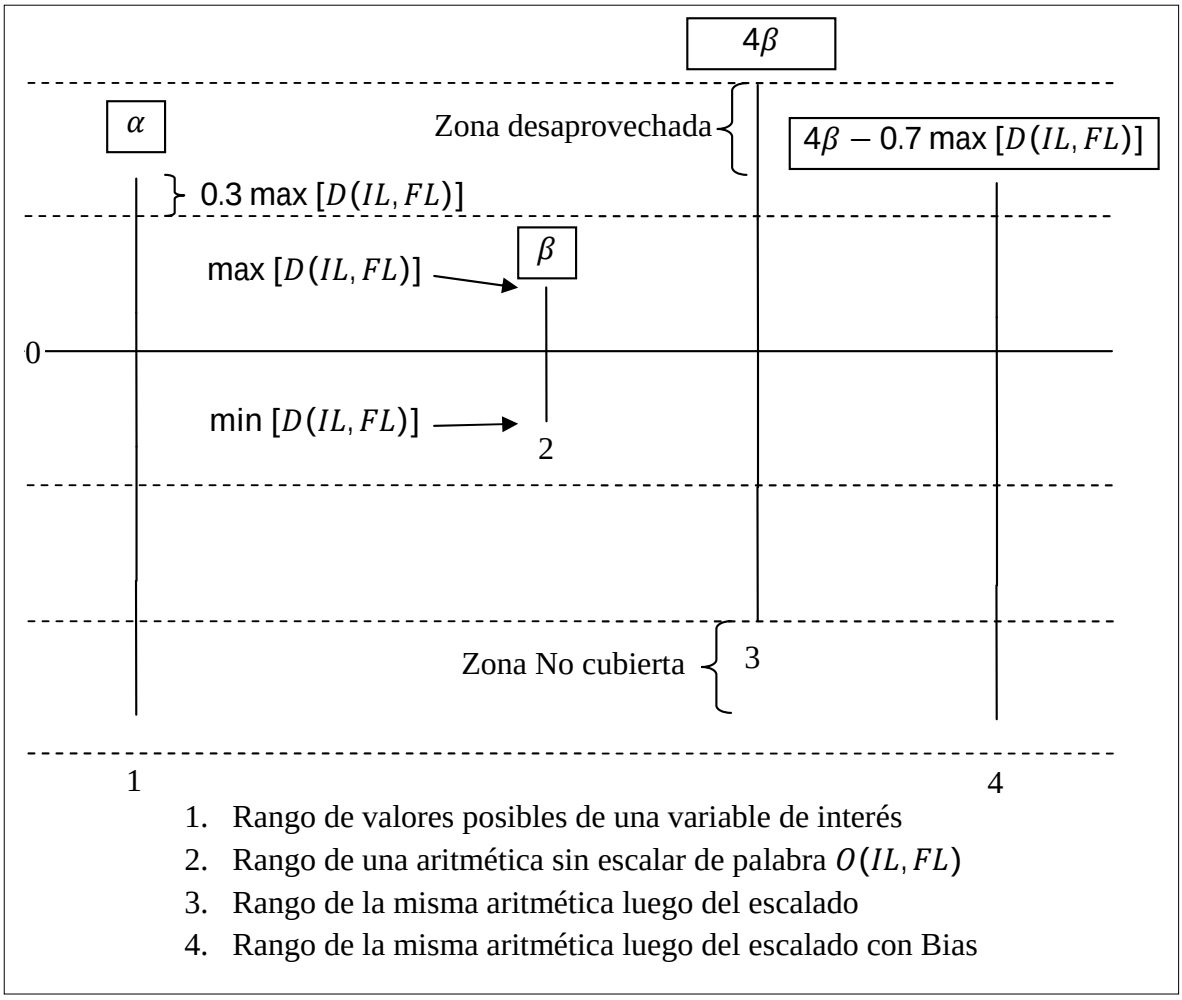



**Figura 6-7: Ejemplo de la aplicación de un escalado con Bias para poder representar todo el rango de una variable de interés en una aritmética en principio insuficiente.**

La aplicación del escalado es una herramienta casi indispensable para el correcto funcionamiento de un algoritmo en una aritmética tan comprometida como la de punto fijo. Sin embargo su implementación requiere de hardware adicional pues todas las cantidades deben ser escaladas antes de ser utilizadas en el algoritmo. Además debe contemplarse la modificación del error de redondeo que surge del escalado. De las dos versiones de escalado la que requiere de menos hardware es la que no tienen Bias pues implica una operación de adición menos.

Para una descripción más detallada del proceso de escalado ver [**9**].

## 6.2 Operaciones aritméticas fundamentales en punto fijo

El caso más común, en algoritmos iterativos como es el KF, es que todos los operandos tengan la misma aritmética. Debido a que cuando se realiza una operación con dos operandos de las mismas características se obtiene un resultado con una aritmética diferente, será necesario readecuarlo de nuevo a la aritmética original. Para esto, se aplican *los métodos de tratamiento del Overflow y del redondeo explicados en la sección anterior*. Realizar esto implica sin duda una pérdida de exactitud, pero al ser el algoritmo iterativo es la única manera de mantener en un tamaño práctico la palabra de trabajo.

### 6.2.1 Suma

Para el caso de la suma el resultado tendrá una parte entera con un bit más que el de los operandos debido a la posibilidad de Overflow:

$$O(\mathrm{IL},\mathrm{FL}) + O(\mathrm{IL},\mathrm{FL}) = O_{\mathrm{res}}(\mathrm{IL}+1,\mathrm{FL})$$

**Ecuación 6-16**

Por lo que luego de operar debe hacerse:

$$O_{\mathrm{res}}(\mathrm{IL}+1,\mathrm{FL}) \rightarrow O(\mathrm{IL},\mathrm{FL})$$

**Ecuación 6-17**

### 6.2.2 Complemento

El complemento daría en principio un resultado siempre con la misma aritmética que el operando, sin embargo hay un caso en que esto no es así.

Si se computa el complemento a 2 (en complemento a 1 esto no sucede) de $\min[D(IL,FL)]$ se producirá un Overflow por ser $|\min[D(IL,FL)]| > |\max[D(IL,FL)]|$. Debido a esto es que debe considerarse para el caso general que el resultado de la operación de complemento tiene una parte entera con un bit de más:

$$-O(\mathrm{IL},\mathrm{FL}) = O_{\mathrm{res}}(\mathrm{IL}+1,\mathrm{FL})$$

**Ecuación 6-18**

Por lo que luego de operar debe hacerse:

$$O_{\mathrm{res}}(\mathrm{IL}+1,\mathrm{FL}) \rightarrow O(\mathrm{IL},\mathrm{FL})$$

**Ecuación 6-19**

### 6.2.3 Multiplicación

La multiplicación es una operacion en la que se produce gran pérdida de exactitud cuando se readapta el resultado a la aritmética de los operandos. Esto es así porque la longitud de la palabra del resultado de una multiplicación es del doble de la de los operandos (o sea, $2WL = 2 + 2IL + 2FL$).

$$O(\mathrm{IL},\mathrm{FL}) \,.\, O(\mathrm{IL},\mathrm{FL}) = O_{\mathrm{res}}(2\mathrm{IL}+1,2\mathrm{FL})$$

**Ecuación 6-20**

Por lo que luego de operar debe hacerse:

$$O_{\mathrm{res}}(2\mathrm{IL}+1,2\mathrm{FL}) \rightarrow O(\mathrm{IL},\mathrm{FL})$$

**Ecuación 6-21**

### 6.2.4 División

El caso de la división en FP es un poco más complejo que el de las operaciones anteriores. Mientras que en aquellas no son demasiadas las diferencias entre los algoritmos alternativos de computo (salvo tal vez para la multiplicación), en el caso de la división existen varios métodos de cálculo que presentan diferentes propiedades numéricas y de síntesis.

El mayor esfuerzo en obtener algoritmos alternativos para la división, se debe principalmente a que esta es una operación con una mayor complejidad computacional y que requiere de mucho más hardware para ser implementada que las otras tres. Además el operador de división "/" implementado en las Librerías estándar de VHDL solo acepta divisores que sean una potencia de 2 (o sea que solo realiza un desplazamiento).

El algoritmo que aplicamos en este proyecto es uno de los llamados algoritmos de división lenta (pero de cálculo exacto) derivados del algoritmo de división manual y es presentado en [**10**]. Existen también, otros llamados de división rápida, que utilizan métodos numéricos para realizar una estimación del resultado con la exactitud deseada. Sin embargo, requieren en general del computo de una estimación inicial más o menos certera para facilitar la convergencia de algoritmo. El cálculo de senda estimación requiere de hardware adicional y debe diseñarse con cierto conocimiento del problema a tratar (algo que como se verá tratamos de evitar en este trabajo).

La principal ventaja del algoritmo que adoptamos es que, es directamente sintetizable en FPGA, tiene gran capacidad de paralelización y puede implementarse completamente de manera combinacional (lo que permite la creación de un operador o librería que facilita la realización de algoritmos en los que intervienen gran cantidad de divisiones). Además como se explica en [**11**] la no utilización de registros y líneas de reloj produce en general una reducción del consumo y la latencia.

En la sección 7.4.3, en el apartado que trata sobre la función Reciprocal, se presenta en detalle este algoritmo que se basa en el presentado en [**10**] y [**11**] salvo algunas modificaciones importantes que se tratan en esa sección.

Aquí analizaremos las características del resultado:

$$\frac{O(\mathrm{IL},\mathrm{FL})}{O(\mathrm{IL},\mathrm{FL})} = O_{\mathrm{res}}(\mathrm{IL}+\mathrm{FL}+1,\mathrm{IL}+\mathrm{FL})$$

**Ecuación 6-22**

Por lo que luego deberá hacerse:

$$O_{\mathrm{res}}(\mathrm{IL}+\mathrm{FL}+1,\mathrm{IL}+\mathrm{FL}) \rightarrow O(\mathrm{IL},\mathrm{FL})$$

**Ecuación 6-23**

### 6.2.5 Análisis de la secuencia de operación en el algoritmo del KF- BT

Como se mencionase en la sección 5.3.2, las operaciones en aritmética finita no son en general asociativas. Esto significa que la secuencia de computo afecta el resultado.

Tomemos por ejemplo el caso de la suma algebraica de tres operandos positivos con la misma aritmética:

$$O_1(IL, FL)\ , O_2(IL, FL)\ y\ O_3(IL, FL)$$

Tales que, la suma de los dos primeros produzca un Overflow:

$$D[O_1(IL, FL) + O_2(IL, FL)] > \max\,[D(IL, FL)]$$

Entonces, en general será:

$$[O_1(IL, FL) + O_2(IL, FL)] - O_3(IL, FL) \neq O_1(IL, FL) + [O_2(IL, FL) - O_3(IL, FL)]$$

Pues el resultado de $O_2(IL, FL) - O_3(IL, FL)$ al ser más pequeño que $O_2(IL, FL)$ puede hacer que la suma ya no de un Overflow.

Así, en el miembro derecho de la expresión anterior se produce Overflow y en el miembro izquierdo no. Esto lógicamente arroja resultados numéricos distintos.

Esta es una característica que no se presenta en aritméticas infinitas o que no es muy relevante si la aritmética que se usa tiene un rango varios órdenes de magnitud mayor al de los operandos.

Lamentablemente este no es el caso del FP por lo que, surge la pregunta de cómo deben asociarse las operaciones de un algoritmo dado en aritmética infinita o de punto flotante cuando se lo desea implementar en FP. Aquí, tal como ocurre para el caso del análisis del ruido de cuantización, habrá dos opciones básicas:

- Si se conocen los rangos de variación de los operandos en cada operación, se puede realizar un análisis teórico que determine las asociaciones óptimas que minimizarán las posibilidades de Overflow y redondeo. Sin embargo, este tipo de análisis se dificulta para el caso de algoritmos con entradas aleatorias.

- El otro paradigma es el de la experimentación y el aprovechamiento de las herramientas computacionales actuales para evaluar el comportamiento numérico del algoritmo en punto fijo con diferentes asociaciones. De todas formas, este acercamiento también se beneficia mucho del conocimiento numérico del problema a tratar.

En este trabajo debido a que no se desarrolló el KF para su funcionamiento con una planta especifica, no se efectuó un análisis de las asociaciones que deberían realizarse para que el funcionamiento sea numéricamente óptimo. Dicho análisis sobrepasaría además los límites de este trabajo pues requiere realizar un seguimiento de la propagación, a través del algoritmo del KF-BT, de los rangos de variación de cada una de las variables utilizando sus propiedades estadísticas.

De todas formas, citamos aquí dos recaudos que resultaron de utilidad en varias oportunidades:

- Si en las operaciones intervienen constantes es probable que sea posible un análisis rápido de la asociación óptima.

- Al comparar dos algoritmos en FP, sus operaciones deben estar *asociadas de la misma manera* para que arrojen el mismo resultado.

En la sección 6.5.2 detallamos a modo de referencia las operaciones sensibles al problema de la secuencia de operación dentro del algoritmo del KF-BT.

## 6.3 Sobre la herramienta Fixed-Point ToolBox de MATLAB

Fixed-Pint ToolBox es una herramienta de MATLAB que se utiliza para el manejo de números en punto fijo.

Con la finalidad de poder especificar el formato del número en FP con el que se trabajará en la ejecución de los algoritmos del KF se procede a la definición de la siguiente estructura:

```
N=numerictype('Signed',Signe,'WordLength',WordLength,'FractionLength',
FractionLength);
```

Dicha estructura permite la especificación de diversos parámetros asociados con el formato del número en punto fijo a utilizar [**9**]. Aquí fijamos los valores de estos parámetros como ejemplo:

- 'Signed': Si el número tiene signo (BS=1) o no (BS=0). En nuestro caso se trabaja con números con signo.

- 'WordLength': La longitud de la palabra en número de bits. Trabajamos con números de 18 bits.

- 'FractionLength': La longitud de la parte fraccional de la palabra en número de bits. De los 18 bits, se reservan 6 bits para la parte fraccional.

Una vez creada esta estructura, se pueden declarar e inicializar variables que hagan uso de dicha estructura por medio de la creación de objetos llamados “fi” de la herramienta:

```
Uno=fi(1,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',SumM,
'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',ProductM,'ProductWordLength',ProductWordLength,'ProductFra
ctionLength',ProductFractionLength);
```

A través de este objeto “fi” se ha declarado la variable llamada “Uno” e inicializado con un valor de 1. Dicho objeto hace referencia a la estructura N para especificar el formato de la variable en punto fijo.

En este tipo de objetos se pueden especificar diversos parámetros [**9**], pero para nuestros objetivos se especificaron los siguientes:

- 'OverflowMode': Modo de tratamiento del desbordamiento. En nuestro caso utilizamos el modo ‘saturate’ que es consistente con los criterios de actuación exhibidos en la sección 6.1.2.

- 'RoundMode': Modo de tratamiento del redondeo. En nuestro caso utilizamos el modo ‘floor’ que es consistente con lo exhibido en la sección 6.1.3.

- 'SumMode': Especifica el formato de la palabra en la que se almacenará el resultado de la suma. En nuestro caso inicializamos el parámetro con ‘SpecifyPrecision’ lo cual da lugar a la elección del formato de la palabra que se desee para almacenar el resultado de la suma a través de los siguientes parámetros relacionados:
    - 'SumWordLength': Longitud de la palabra en donde se almacenará el resultado de las sumas entre variables declaradas como objetos “fi”, en número de bits. Para nuestro caso se escogió una longitud igual a la del parámetro 'WordLength'.
    - 'SumFractionLength': Longitud de la parte fraccional de la palabra donde se almacenará el resultado de las sumas entre variables declaradas como objetos “fi”, en número de bits. Para nuestro caso se escogió una longitud igual a la del parámetro 'FractionLength'.

- 'ProductMode': Especifica el formato de la palabra en la que se almacenará el resultado de la multiplicación. En nuestro caso inicializamos el parámetro con ‘SpecifyPrecision’ lo cual da lugar a la elección del formato de la palabra que se desee para almacenar el resultado de la multiplicación a través de los siguientes parámetros relacionados:
    - 'ProductWordLength': Longitud de la palabra en donde se almacenará el resultado de las multiplicaciones entre variables declaradas como objetos “fi”, en número de bits. Para nuestro caso se escogió una longitud igual a la del parámetro 'WordLength'.
    - 'ProductFractionLength': Longitud de la parte fraccional de la palabra donde se almacenará el resultado de las multiplicaciones entre variables declaradas como objetos “fi”, en número de bits. Para nuestro caso se escogió una longitud igual a la del parámetro 'FractionLength'.

## 6.4 Oscilador de segundo orden ruidoso

En este trabajo hemos tratado en lo posible de mantener siempre el desarrollo de los conceptos involucrados en la implementación del KF lo más alejado posible de un problema especifico pues, como recalcamos en varias oportunidades, el objetivo principal es el desarrollo del Know-how que permita aplicar el KF en futuras investigaciones o problemas específicos regionales.

Debido a esto, todos los desarrollos teóricos así como el software de alto nivel y el hardware descripto trataron de mantenerse generales en lo que respecta a los valores y dimensiones de las matrices que definen el sistema, los sensores y los ruidos.

Sin embargo, a fin de probar el funcionamiento del hardware desarrollado y de explicar algunos conceptos que no pueden ser tratados independientemente de la aplicación, se decide utilizar como planta de prueba un simple sistema oscilador de segundo orden ruidoso. Esta elección se basa en el hecho de que tener muchos estados internos extendería demasiado los desarrollos numéricos propiciando que se pierdan de vista los conceptos centrales que se tratan de explicar. Además, las plantas de orden superior tienen en general matrices definidas en función de parámetros físicos que requerirían ahondar en la etapa de modelado de un sistema específico y esto, como se mencionó, trata de evitarse aquí.

### 6.4.1 Definición del sistema

El modelo matemático del oscilador armónico de segundo orden ruidoso (LTI) queda especificado a través de la ecuación diferencial:

$$\ddot{y}(t) + 2\zeta\omega\dot{y}(t) + \omega^2 y(t) = u(t) + w(t)$$

**Ecuación 6-24**

Donde:

$y(t)$ $[ft]$ Representa el desplazamiento en pies (del Inglés feet).

$\zeta$ Representa la relación de amortiguamiento.

$\omega$ Representa la frecuencia natural no amortiguada.

$u(t)$ Representa la entrada de control determinística.

$w(t)$ Representa el ruido (blanco Gaussiano de media nula) de entrada del sistema.

Sus valores para nuestro caso son:

$$\zeta = 0.2$$

$$\omega = 5 \left[\frac{rad}{s}\right]$$

$$u(t) = 12 \left[\frac{ft}{s^2}\right]$$

$$Q(t) = E\langle w(t)^2 \rangle = 0.02 \left[\frac{ft^2}{s^3}\right]$$

### 6.4.2 Modelo de estados continuo

Para construir el modelo en el espacio de los estados del sistema descrito en la sección 3.2.1, se define primero el vector de estados mediante el procedimiento que deriva en el modelo de variables de fase.

$$x_1(t) = y(t)$$

$$x_2(t) = \dot{x}_1(t) = \dot{y}(t)$$

$$\dot{x}_2(t) = \ddot{x}_1(t) = \ddot{y}(t) = u(t) + w(t) - 2\zeta\omega\dot{y}(t) - \omega^2 y(t)$$

$$\dot{x}_2(t) = u(t) + w(t) - 2\zeta\omega x_2(t) - \omega^2 x_1(t)$$

De esta manera se ha transformado un modelo descrito por una ecuación diferencial de *segundo orden* en uno descrito por un sistema de *dos ecuaciones* de primer orden:

$$\dot{x}_1(t) = x_2(t)$$

$$\dot{x}_2(t) = -2\zeta\omega x_2(t) - \omega^2 x_1(t) + u(t) + w(t)$$

Que, matricialmente queda:

$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix} = \begin{bmatrix}0 & 1\\ -\omega^2 & -2\zeta\omega\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + \begin{bmatrix}0\\ 1\end{bmatrix}u(t) + \begin{bmatrix}0\\ 1\end{bmatrix}w(t)$$

$$z(t) = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + v(t)$$

Expresemos numéricamente este modelo teniendo en cuenta los datos específicos del sistema y que:

$$R(t) = E\langle v(t)^2\rangle = 0{,}01\ [ft^2]$$

Para obtener:

$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix} = \begin{bmatrix}0 & 1\\ -25 & -2\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + \begin{bmatrix}0\\ 1\end{bmatrix}12 + \begin{bmatrix}0\\ 1\end{bmatrix}w(t)$$

$$z(t) = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + v(t)$$

De esta manera, las matrices correspondientes al modelo de estados definido en la sección 3.2.1 quedan:

$$x(t) = \begin{bmatrix}x_1(t)[ft]\\ x_2(t)\left[\frac{ft}{s}\right]\end{bmatrix}$$

$$u(t) = 12\left[\frac{ft}{s^2}\right]$$

$$C(t) = \begin{bmatrix}0\\ 1\end{bmatrix}$$

$$G(t) = \begin{bmatrix}0\\ 1\end{bmatrix}$$

$$F(t) = \begin{bmatrix}0\left[\frac{rad}{s}\right]^2 & 1\left[\frac{rad}{s}\right]\\ -25\left[\frac{rad}{s}\right]^2 & -2\left[\frac{rad}{s}\right]\end{bmatrix}$$

$$H(t) = [1 \quad 0]$$

$$w(t)\ \left[\frac{ft}{s^2}\right] es\ un\ RP\ Gaussiano\ de\ media\ nula\ con\ Q(t) = 0.02\ \left[\frac{ft^2}{s^3}\right]$$

$$v(t)[ft]\ es\ un\ RP\ Gaussiano\ de\ media\ nula\ con\ R(t) = 0.01\ [ft^2 s]$$

**Ecuación 6-25**

### 6.4.3 Modelo de estados discreto

Discretización de la ecuación de estados:

Se procede inicialmente a la discretización de la ecuación de estados como se especificó en la sección 3.2.3:

Tomando $T = 0{,}01\ [s]$:

- Se calcula la STM haciendo referencia a la Ecuación 3-21:

$$\Phi(t + T, \xi) = e^{\int_t^{t+T} F d\tau} = e^{FT} = I + FT + \frac{1}{2!}F^2T^2 + \cdots + \frac{1}{n!}F^nT^n + \cdots$$

$$\Phi(t + T, \xi) = e^{0{,}01F}$$

$$\Phi_k = \begin{bmatrix} 0.9988 & 0.0099 \\ -0.2474 & 0.9790 \end{bmatrix}$$

- El término forzante asociado a la entrada determinística se obtiene de la Ecuación 3-23:

$$u_k = \int_t^{t+T} \Phi(t + T, \xi)\ C(\xi)\ u(\xi)\ d\xi$$

$$u_k = \int_0^{0{,}01} \left\{ \begin{bmatrix} 0.9988 & 0.0099 \\ -0.2474 & 0.9790 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} 12 \left[\frac{ft}{s^2}\right] \right\} d\xi = \begin{bmatrix} 0.0012 \\ 0.1175 \end{bmatrix} \left[\frac{ft}{s}\right]$$

- Como se expresó en la sección 3.2.3, el término forzante asociado al ruido de proceso resulta en una secuencia aleatoria también Gaussiana de igual media pero de distinta covarianza:

$$w_k\ \left[\frac{ft}{s}\right] RP\ Gaussiano\ de\ media\ nula\ con\ covarianza\ Q_k$$

- La covarianza del ruido discretizado se obtiene haciendo referencia a la Ecuación 3-29 que, como se explicó en la sección 3.2.3, da lugar al cómputo aproximado de la misma:

$$Q_k = G(t)Q(t)G^T(t)T$$

Sabiendo que:

$$Q(t) = 0.02 \left[\frac{ft^2}{s^3}\right]$$

Sin embargo se hace referencia a la Ecuación 3-27 (que es la ecuación exacta de discretización de $Q(t)$) con el fin de deducir las dimensiones:

$$Q_k = \begin{bmatrix}0\\1\end{bmatrix}(0.02)\,[0 \quad 1](0.01) = \begin{bmatrix}0 & 0\\0 & 0.0002\end{bmatrix}\left[\frac{ft^2}{s}\right]$$

Discretización de la ecuación de salida:

- Como se explicó en la sección 3.2.3, $H_k$ es obtenida del muestreo de $H(t)$ en el instante $t = kT$. Como aquí $H(t)$ es invariante en el tiempo:

$$H_k = [1 \quad 0]$$

- Como se expresó en la sección 3.2.3, el ruido de medición discretizado resulta en una secuencia aleatoria también Gaussiana de igual media pero de distinta covarianza:

$$v_k\,[ft]\ RP\ Gaussiano\ de\ media\ nula\ con\ covarianza\ R_k$$

- La covarianza del ruido discretizado se obtiene haciendo referencia a la Ecuación 3-30 como se explicó en la sección 3.2.3:

$$R_k = \frac{R(t)}{T}$$

Sabiendo que:

$$R(t) = 0.01\,[ft^2\ s]$$

Y, haciendo referencia a la Ecuación 3-31 (que es la ecuación exacta de discretización de $R(t)$) para deducir la dimensión de $R_k$, se obtiene:

$$R_k = \frac{0.01}{0.01} = 1[ft^2\ s]$$

De esta manera, las matrices correspondientes al modelo de estados definido en la sección 3.2.2 quedan:

$$x_k = \begin{bmatrix}x_{1_k}[ft]\\x_{2_k}\left[\frac{ft}{s}\right]\end{bmatrix}$$

$$u = \begin{bmatrix}0.0012\,[ft]\\0.1175\left[\frac{ft}{s}\right]\end{bmatrix}$$

$$\Phi = \begin{bmatrix} 0.9988 & 0.0099 \\ -0.2474 & 0.9790 \end{bmatrix}$$

$$H = [1 \quad 0]$$

$$w_k \left[\frac{ft}{s}\right] RP\ Gaussiano\ de\ media\ nula\ con\ covarianza\ Q = \begin{bmatrix} 0 & 0 \\ 0 & 0.0002 \end{bmatrix} \left[\frac{ft^2}{s}\right]$$

$$v_k\ [ft]\ RP\ Gaussiano\ de\ media\ nula\ con\ covarianza\ R = 1[ft^2\ s]$$

### 6.4.4 Cálculo de las matrices del KF-BT

Con el fin de llegar a la obtención de las matrices necesarias para la ejecución del algoritmo iterativo, ilustrado en la Figura 5-4, se procede a aplicar el algoritmo de factorización $UD$ a la matriz de covarianza de la incertidumbre de estimación a priori inicial:

$$P_0^- = \begin{bmatrix} 2[ft^2 s] & 0[ft^2] \\ 0[ft^2] & 2\left[\frac{ft^2}{s}\right] \end{bmatrix}$$

Obteniendo:

$$U_0^- = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, D_0^- = \begin{bmatrix} 2[ft^2 s] & 0[ft^2] \\ 0[ft^2] & 2\left[\frac{ft^2}{s}\right] \end{bmatrix}$$

Para el inicio de la ejecución, se necesita también el valor inicial de la estimación a priori del vector de estados:

$$\hat{x}_0^- = \begin{bmatrix} 0[ft] \\ 0\left[\frac{ft}{s}\right] \end{bmatrix}$$

## 6.5 Desempeño del KF- BT en aritmética de punto fijo

En esta sección se evalúa:

- El desempeño numérico del algoritmo del KF-BT corriendo en aritmética de punto flotante de doble precisión (que denotaremos KF-BT-DP).

- El desempeño numérico del algoritmo del KF-BT corriendo en diferentes aritméticas de FP (que denotaremos KF-BT-FP).

- El desempeño numérico del algoritmo original del KF (KF-OA) corriendo en diferentes aritméticas de FP (que denotaremos KF-OA-FP).

- Comparación entre los desempeños de los algoritmos KF-BT-DP y KF-BT-FP en diferentes aritméticas de FP. Esto se hace para poder mostrar la degradación de la estabilidad numérica al pasar de doble precisión a FP.

- Comparación entre los desempeños de los algoritmos KF-BT-FP y KF-OA-FP en diferentes aritméticas de FP. Esto se hace para poder mostrar la mejora en la estabilidad numérica que introducen las implementaciones del KF basadas en la factorización matricial (como es BT).

Para esto, desarrollamos diversos programas en el software de cálculo *MATLAB* haciendo uso de la herramienta *Fixed-Point ToolBox*.

La planta utilizada para estas pruebas es la desarrollada en las secciones 6.4.3 y 6.4.4. Esto es un punto importante a considerar a la hora de generalizar los resultados pues, una prueba exhaustiva de la estabilidad numérica de cualquier implementación del KF, implicaría probar su desempeño en plantas con diferente nivel de condicionamiento, ya que los resultados están altamente influenciados por las características del problema.

Código Fuente

A continuación presentamos los archivos de MATLAB en los que se encuentran codificadas las funciones que desarrollamos para obtener los resultados presentados en esta sección:

- *KF_comp.m,* disponible en el anexo 11.1.1. Este archivo implementa la función KF_comp (descripta en el encabezado del archivo) que básicamente se encarga de correr para una misma realización (conjunto de mediciones) y una misma aritmética de FP, tres implementaciones distintas del KF: KF-BT-DP, KF-BT-FP y la KF-OA-FP. Para cada una de estas, la función calcula también los errores de estimación medios, tiempos de convergencia, etc.
  *La función KF_comp es utilizada por todos los otros archivos.*

- *Single_implementation.m,* disponible en el anexo 11.1.2. Este archivo se encarga de presentar para las tres implementaciones la siguiente información de cada uno de los estados estimados: el error de estimación rms, tiempo de convergencia y Factor de Mejora. Para el caso que se elija KF-BT-DP, todas estas cantidades se muestran para un gran número de realizaciones distintas. En caso de elegirse KF-BT-FP ó KF-OA-FP, todas estas cantidades se muestran promediadas para 10 realizaciones y en función del parámetro FL de la aritmética de FP.

- *Comparison.m* disponible en el anexo 11.1.3. Este archivo se encarga de comparar las diferentes implementaciones del KF. Para esto muestra los cocientes (KF-BT-DP) / (KF-BT-FP) y (KF-BT-FP) / (KF-OA-FP) de la siguientes cantidades: el error de estimación rms, tiempo de convergencia, Factor de Mejora, error de estimación rms del estado transitorio y error de estimación rms del estado estacionario. Estas cantidades son un promedio para 10 realizaciones y se computan además en función del parámetro FL de la aritmética de FP.

- *Basic_Plots:* Disponible en el anexo 11.1.4. Este archivo se encarga de graficar para las tres implementaciones del KF y, para cada estado, el desarrollo temporal de las siguientes cantidades: el estado estimado, el estado exacto, las mediciones y la KG.

Criterios de comparación

Para evaluar el desempeño del funcionamiento de cada implementación del KF, así como para compararlas entre sí, se definen las siguientes cantidades:

- Error de estimación rms:

$$rms\_Err(e_x) = \sqrt{\frac{T}{sT}\sum_{k=1}^{sT/T}(e_{x_k})^2} = \sqrt{\frac{T}{sT}\sum_{k=1}^{sT/T}(x_k - \hat{x}_k)^2}$$

  El tiempo de simulación se toma en todas las corridas como $sT = 10\ [s]$

- Factor de Mejora: Con este nombre hemos denotado al cociente:

$$IMP_{Fact} = \frac{rms\ Err(e_z)}{rms\ Err(e_x)}$$

  Donde $e_z$ es el error que se comete si se estima el estado interno directamente con las mediciones, o sea, sin aplicar el KF: $e_{z_k} = x_k - z_k$. Este factor da una idea de la mejora relativa producida al utilizar el KF como estimador.

- Tiempo de convergencia: El tiempo de convergencia (tc) del KF se define mediante la KG. Si se inspecciona la Ecuación 3-43, de actualización temporal de la estimación, se puede ver como cada una de las columnas de la ganancia $K_k$ son las que escalan a cada uno de los errores de los distintos estados. Debido a esto, es que se establece que la ganancia representativa del estado $n$ está dada por la norma de la $n-$ésima columna de $K_k$ ($K_{k,n}$):

$$Ganancia\ del\ estado\ n = \|K_{k,n}\|$$

  Al ser el sistema invariante en el tiempo, al llegar al estado estacionario, la estimación del KF tiende a mejorar (la matriz $P$ tiende a 0). Esto produce también que la KG entre en un estado estacionario y tienda a disminuir acercándose a cero.

  Definiremos el $tc[s]$ como el instante en que la ganancia entra en el régimen estacionario. Para estimar esto en una secuencia discreta finita, establecemos un criterio de convergencia dado por:

$$con\_K = 0.005$$
$$con\_t = 10\%\ de\ sT = 1\ [s]$$

Entonces, el $\mathrm{tc}[s]$ es el instante en el cual la ganancia se hace menor que $con\ K$ y se mantiene luego debajo de este valor, al menos $con\_t$ segundos.

*Para el caso en que el filtro no converja en el $sT$ adoptado, se define $tc[s] = 0$. Nótese que el hecho de que el filtro no converja no quiere decir que este diverja (convergencia a ∞), sino que la ganancia no se estabilizó debajo de un valor deseado (por ejemplo si la KG se estabiliza a un valor mayor a con_K, según este criterio no converge).*

- Error de estimación rms de estado transitorio: Es el error de estimación rms considerando sólo el intervalo transitorio, $[0, ct]$:

$$rms\_trss\_Err(e_x) = \sqrt{\frac{T}{ct}\sum_{k=1}^{ct/T}(x_k - \hat{x}_k)^2}$$

- Error de estimación rms de estado estacionario: Es el error de estimación rms considerando sólo el intervalo en el que el filtro se considera estabilizado, $[ct, sT]$:

$$rms\_steady\_Err(e_x) = \sqrt{\frac{T}{(sT - ct)}\sum_{k=ct/T}^{sT/T}(x_k - \hat{x}_k)^2}$$

### 6.5.1 Algoritmo del KF-BT en doble precisión

El algoritmo del KF-BT operando en doble precisión se encuentra codificado entre las líneas 194 y 381 del archivo KF_comp.m.

En la siguiente figura se muestran las estimaciones, obtenidas para una corrida (realización) del KF-BT-DP, de los dos estados de la planta en cuestión. Además, se muestran los valores exactos de los estados así como la mediciones del estado 1 (pues es el único estado que se mide):

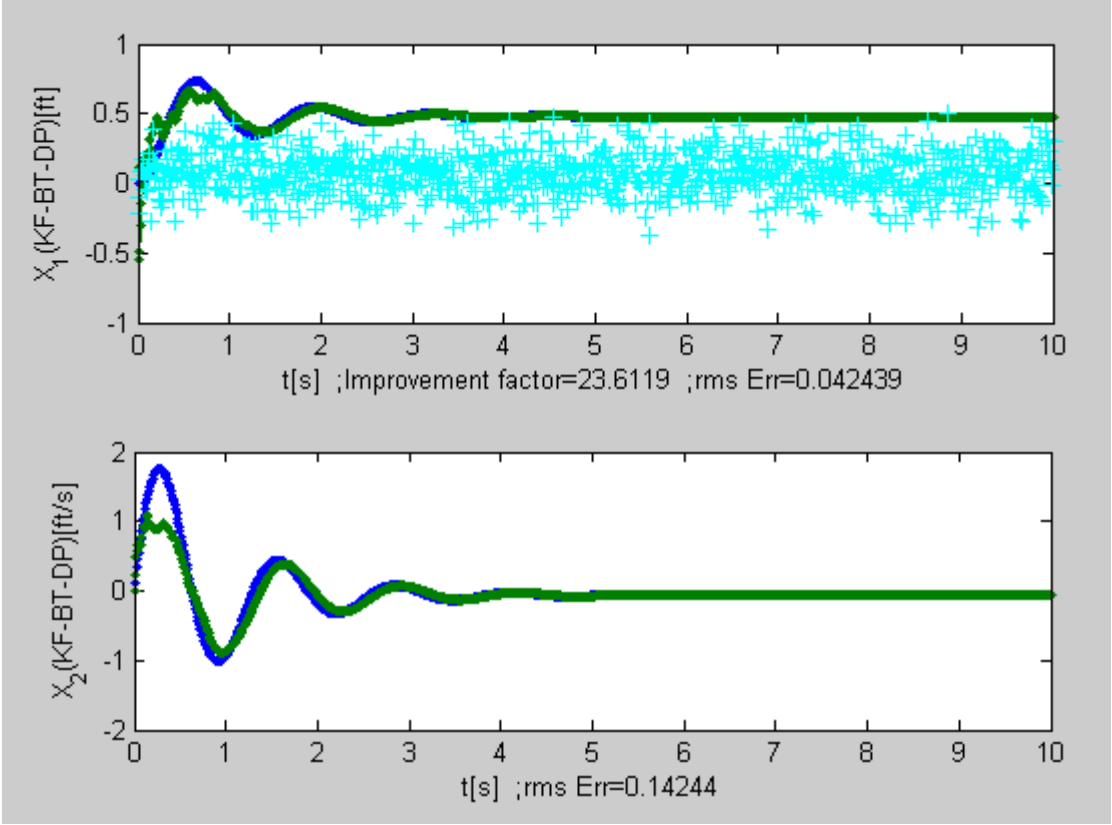



**Figura 6-8: Estados ($x_1$superior y $x_2$ inferior) estimados con el KF-BT-DP (en verde) en función del tiempo. Además los estados exactos (en azul) y las mediciones escaladas al 15% (en cian).**

La Figura 6-9, se muestran los valores que adopta la KG para este caso. También se muestra acotado el valor final de la misma y el $ct[s]$:

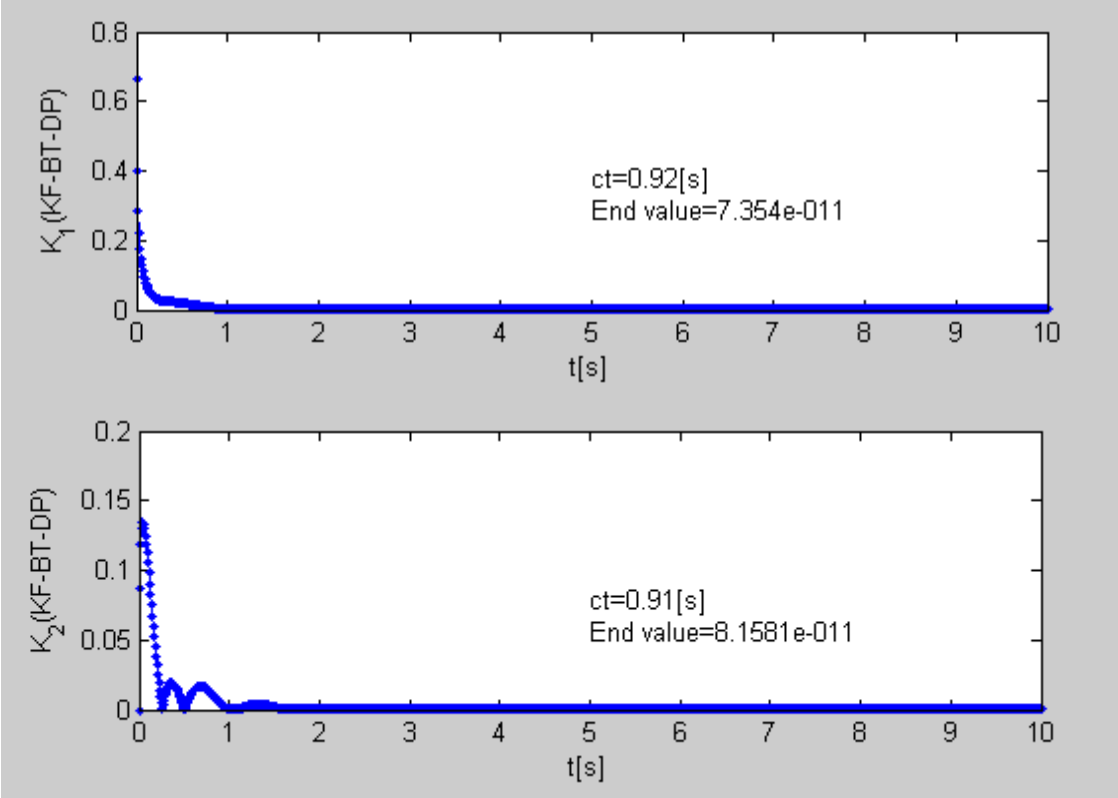



**Figura 6-9: Valores de la KG en función del tiempo para el estado $x_1$(superior) y $x_2$ (inferior) del KF-BT-DP.**

Hemos computado además los valores del error rms de estimación para 170 realizaciones distintas. Estos se muestran en la Figura 6-10 que, para mayor simpleza, se han agrupado en promedios de a 10 realizaciones:

De la misma manera que para el caso anterior, se muestran a continuación los valores de los factores de mejora (Figura 6-11) y de los ct (Figura 6-12). Dado que para nuestro caso se mide únicamente el estado 1, el factor de mejora sólo puede computarse para este. Nótese además que, como la KG se calcula en el lazo determinístico del filtro, el ct no varía de una realización a otra (pues no depende de $z_k$), sin embargo incluimos esta figura para una futura comparación:

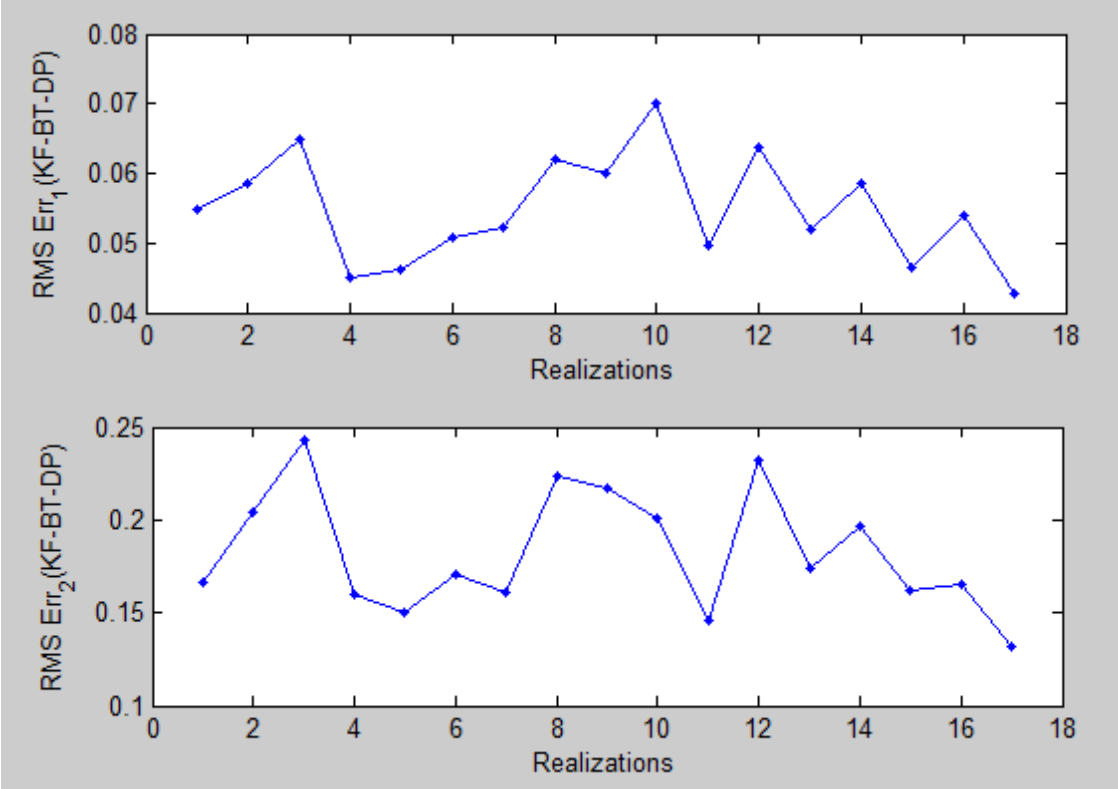



**Figura 6-10: Errores rms de estimación para varias realizaciones del algoritmo KF-BT-DP para el estado $x_1$(superior) y $x_2$(inferior).**

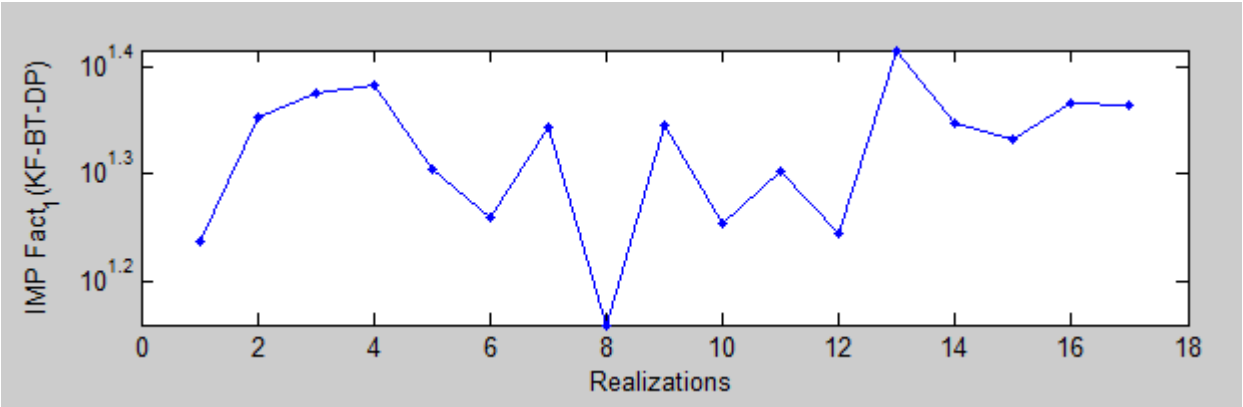



**Figura 6-11: Factor de mejora del estado $x_1$(el único que se mide) para varias realizaciones del KF-BT-DP. Nótese que el eje de ordenadas está en escala logarítmica.**

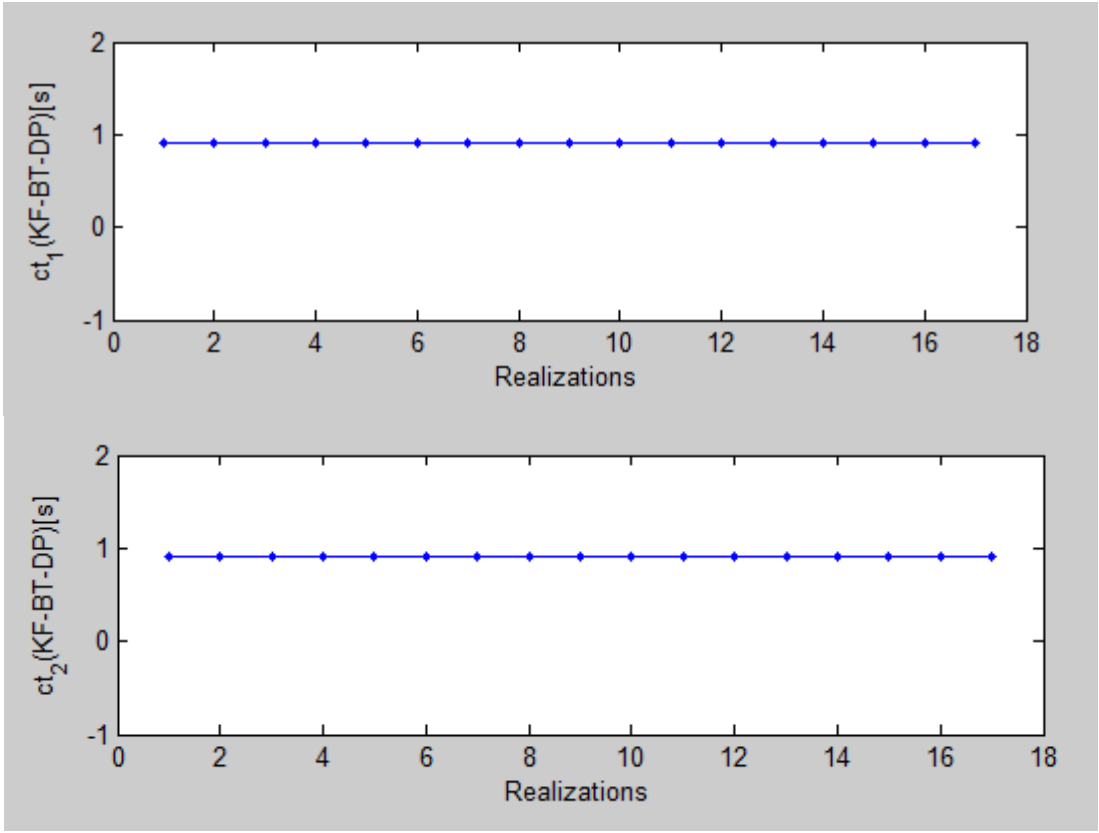



**Figura 6-12: ct para diferentes realizaciones del estado $x_1$ (superior) y $x_2$ (inferior) usando el KF-BT-DP. Se incluye este gráfico solo para resaltar el hecho de que el ct no varía con las realizaciones y para comparaciones futuras.**

### 6.5.2 Algoritmo del KF-BT en punto fijo

El algoritmo del KF-BT-FP se encuentra codificado entre las líneas 482 y 624 del archivo KF_comp.m

En la siguiente figura se muestran las estimaciones temporales, obtenidas para una realización del KF-BT-FP, de los dos estados de la planta en cuestión. Además, se muestran los valores exactos de los estados y las mediciones del estado 1 (escaladas con un factor de 0.15 para facilitar la visualización) para todo el $sT$. Todo esto computado con la siguiente aritmética:

$$WL = 18$$

$$FL = 11$$

$$Round\ Mode = Floor$$

$$Overflow\ Mode = Saturate$$

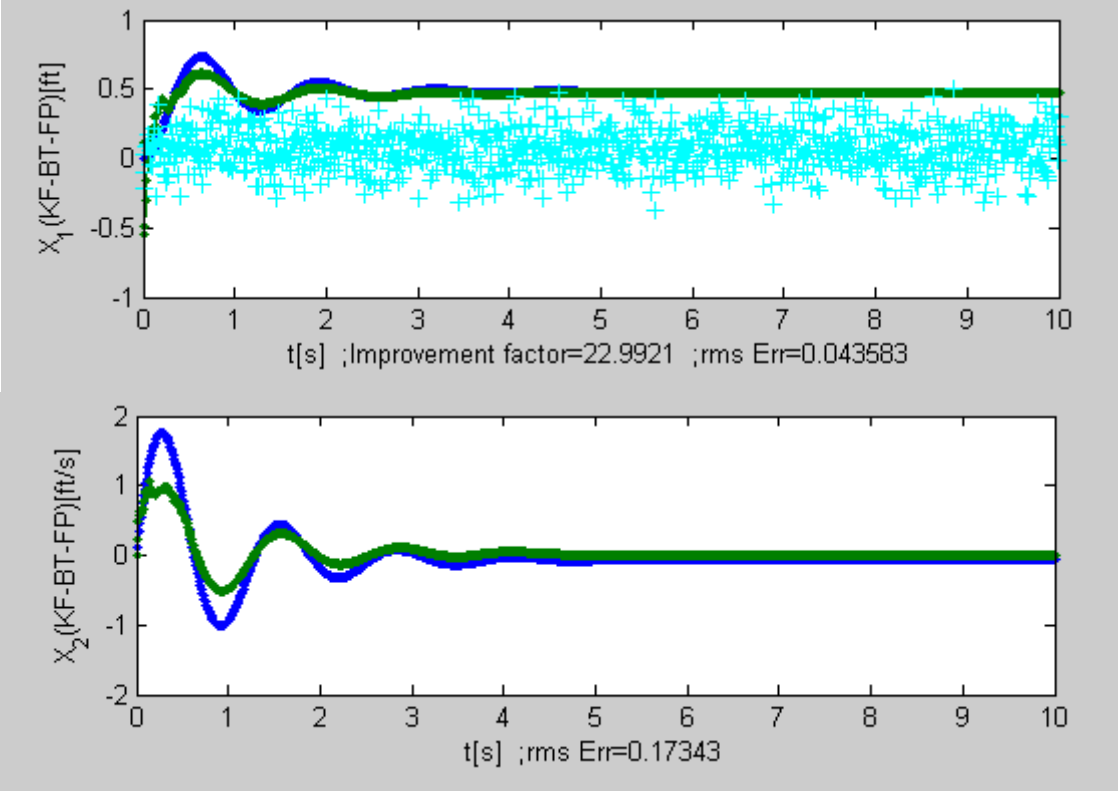



**Figura 6-13: Estados ($x_1$superior y $x_2$ inferior) estimados con el KF-BT-FP (en verde) en función del tiempo. Además los estados exactos (en azul) y las mediciones escaladas al 15% (en cian).**

La Figura 6-14 muestra los valores que adopta la KG para este caso. También se muestra acotado el valor final de la misma y el $ct[s]$.

Hemos computado también (ver Figura 6-15) los valores del error rms de estimación promediados para 10 realizaciones y para diferentes valores del parámetro FL, que define la longitud en bits de la parte fraccional de la palabra de trabajo. El resto de los parámetros que definen la aritmética se tomaron (por razones que veremos en la sección 6.5.4) como:

$$WL = 18$$

$$Round\ Mode = Floor$$

$$Overflow\ Mode = Saturate$$

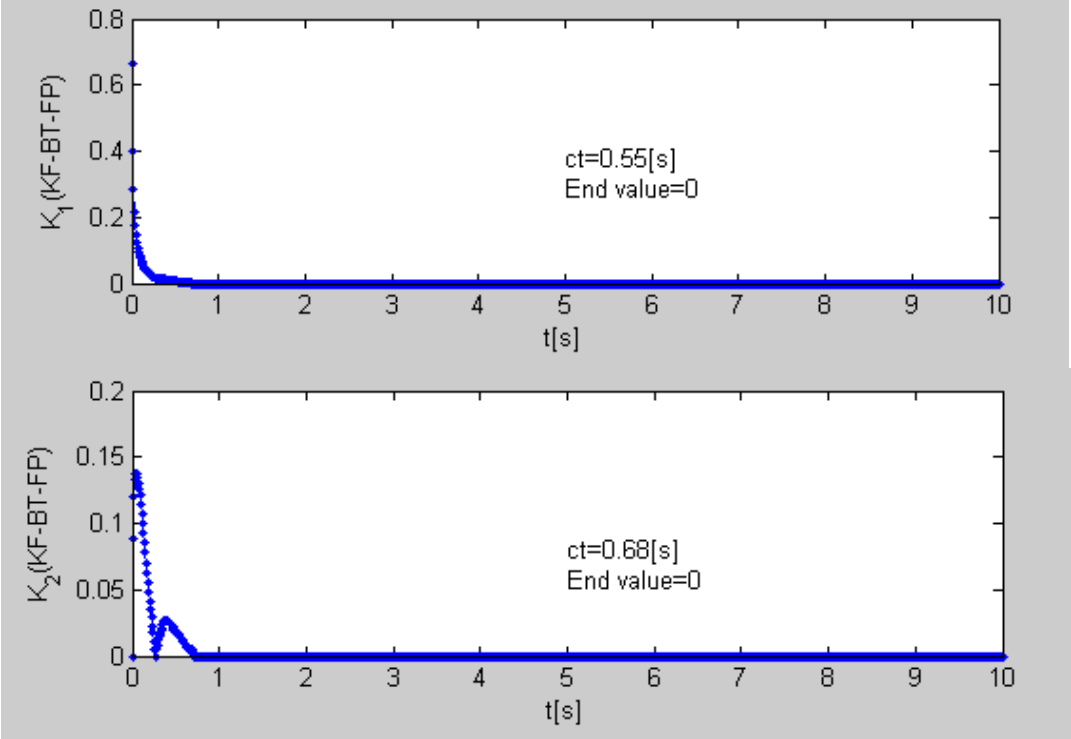



**Figura 6-14: KG en función del tiempo para los estados $x_1$(superior) y $x_2$ (inferior) del KF-BT-FP.**

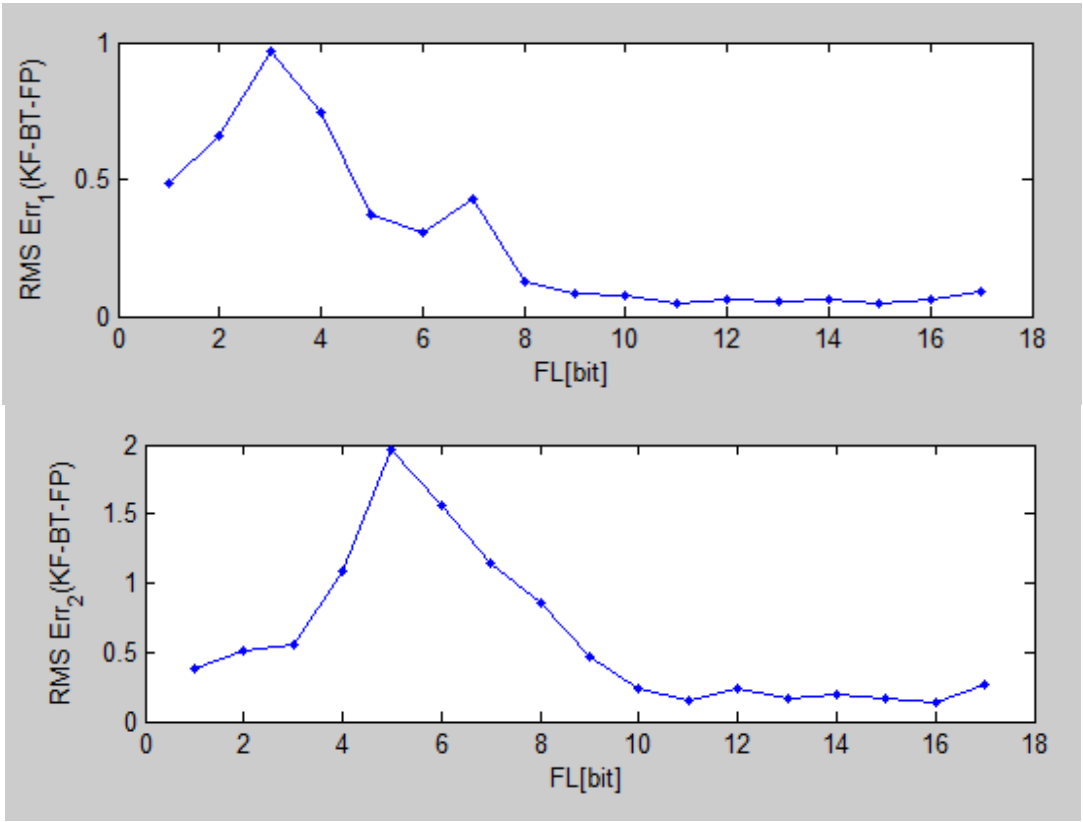



**Figura 6-15: Errores rms de estimación del algoritmo KF-BT-FP para el estado $x_1$(superior) y $x_2$(inferior) en función de FL. Nótese que el eje de ordenadas está en escala logarítmica.**

De la misma manera que para el caso anterior, se muestran a continuación, los valores de los factores de mejora (Figura 6-16) y de los ct (Figura 6-17). Nótese que aquí los ct se modifican debido a que se modifica la aritmética de trabajo:

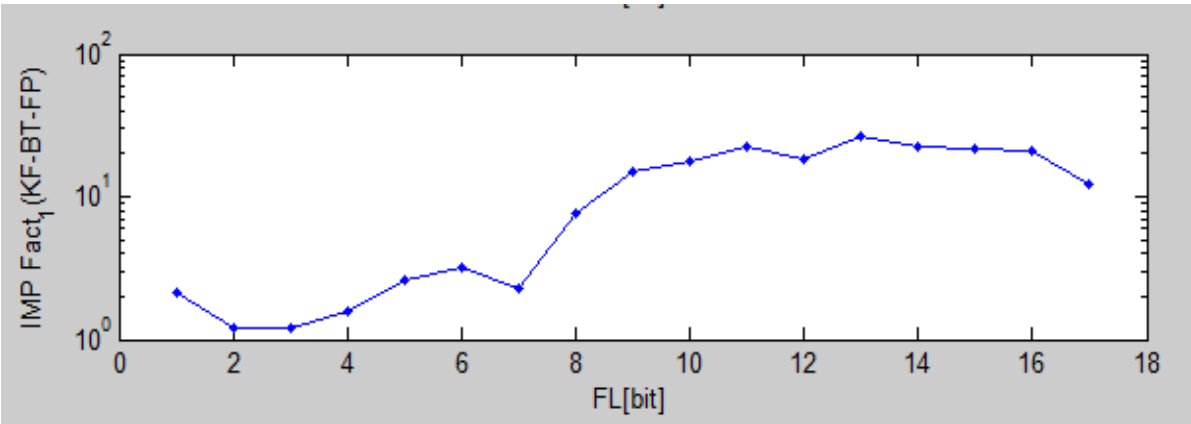



**Figura 6-16: Factor de Mejora para el estado $x_1$(el único que se mide) usando el KF-BT-FP en función de FL. Nótese que el eje de ordenadas está en escala logarítmica.**

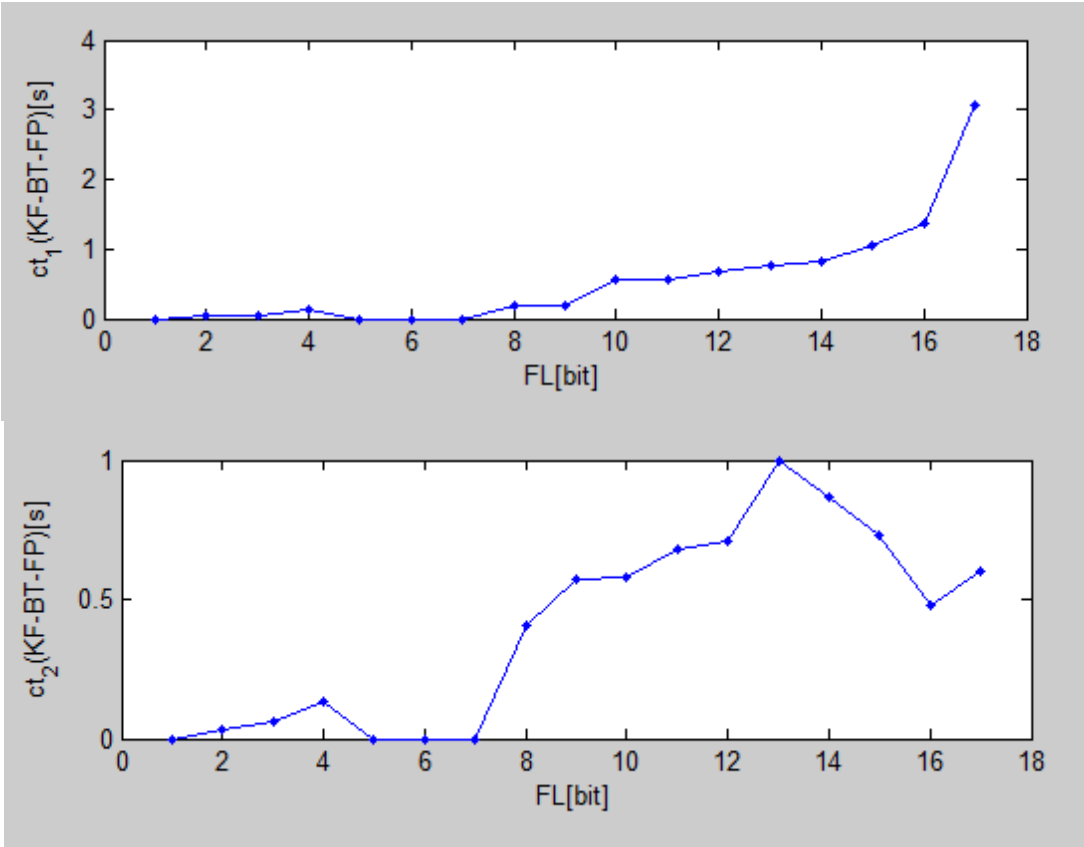



**Figura 6-17: ct del estado $x_1$ (superior) y $x_2$ (inferior) para el KF-BT-FP en función del FL.**

Operaciones en las que debe cuidarse la asociatividad

Como se explicó en la sección 6.2.5, la secuencia de operación (precedencia de las operaciones aritméticas) en las ecuaciones del KF son importantes si se trabaja en aritmética de FP.

Aquí detallaremos, para el caso del algoritmo KF-BT-FP, cuáles son las ecuaciones en las que puede modificarse la asociación de las operaciones de manera de obtener un resultado numérico diferente.

*Incluimos aquí esta información porque es relevante a la hora de querer utilizar el algoritmo en MATLAB para verificar el correcto funcionamiento del mismo codificado en VHDL. Para que dos algoritmos sean iguales en FP deben ser "exactamente iguales", incluyendo las asociaciones de las operaciones. Es un error común obviar esto debido a la costumbre de trabajar en aritméticas infinitas o de gran rango en relación al problema, como la de punto flotante.*

Las operaciones en las que deben tomarse estos recaudos son las localizadas en las siguientes líneas del archivo KF_comp.m: 520; 537; 583; 585; 597 y 601

### 6.5.3 Algoritmo Original del KF en punto fijo

El algoritmo del KF-OA-FP se encuentra codificado entre las líneas 415 y 437 del archivo KF_comp.m.

En la siguiente figura se muestran las estimaciones temporales, obtenidas para una realización del KF-OA-FP, de los dos estados de la planta en cuestión. Además se muestran los valores exactos de los estados y las mediciones del estado 1 (escaladas con un factor de 0.15 para facilitar la visualización) para todo el $sT$. Todo esto computado con la siguiente aritmética:

$$WL = 18$$

$$FL = 11$$

$$Round\ Mode = Floor$$

$$Overflow\ Mode = Saturate$$

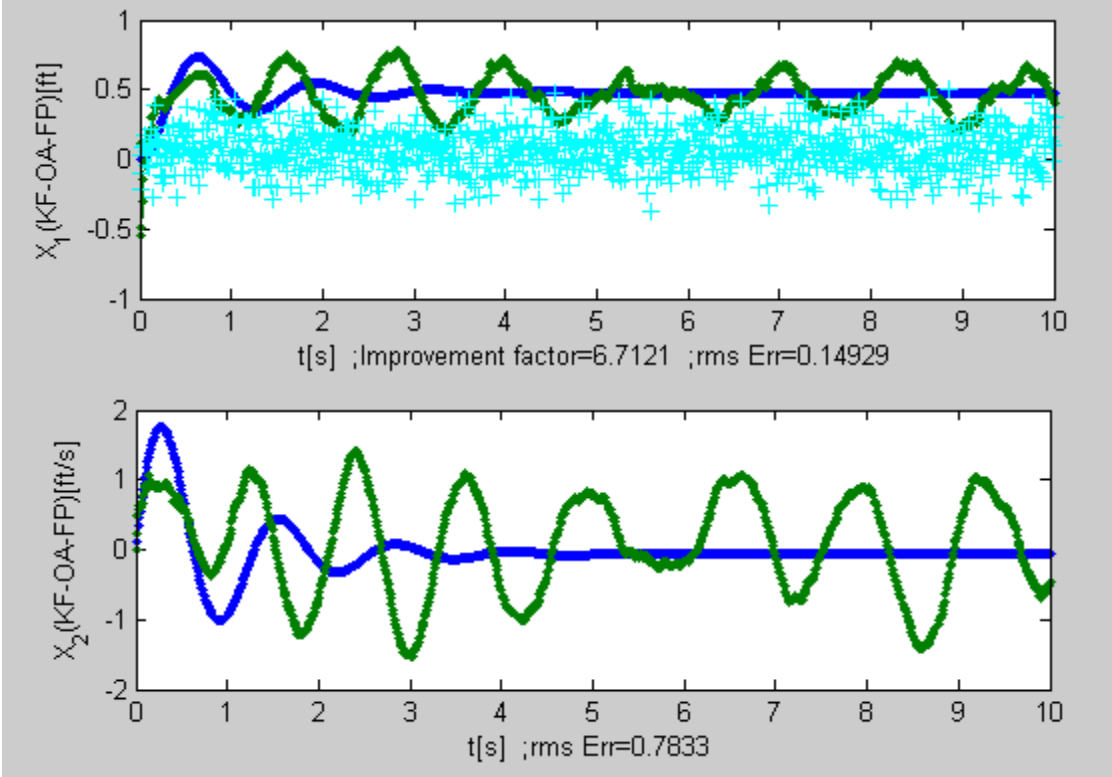



**Figura 6-18: Estados ($x_1$superior y $x_2$ inferior) estimados con el KF-OA-FP (en verde) en función del tiempo. Además los estados exactos (en azul) y las mediciones escaladas al 15% (en cian).**

La Figura 6-19 muestra los valores que adopta la KG para este caso. También se muestra acotado el valor final de la misma y el $ct[s]$.

Hemos computado también (Figura 6-20) los valores del error rms de estimación promediados para 10 realizaciones y además para diferentes valores del parámetro FL. El resto de los parámetros que definen la aritmética se tomaron como antes:

$$WL = 18$$

$$Round\ Mode = Floor$$

$$Overflow\ Mode = Saturate$$

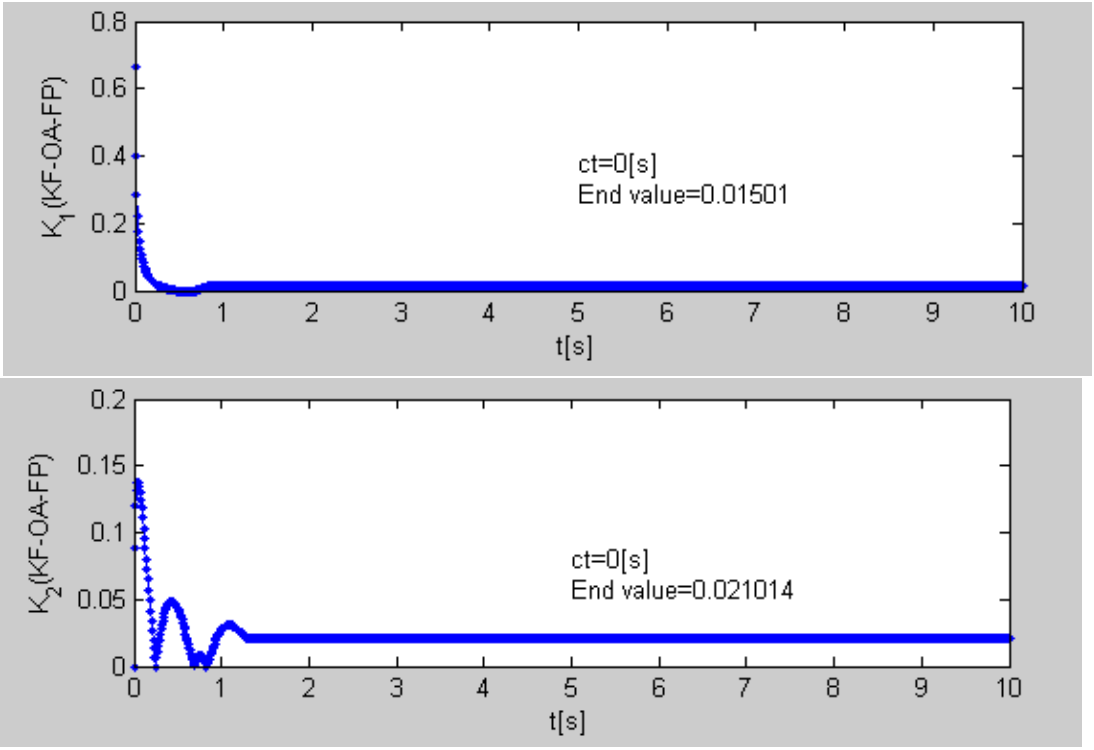



**Figura 6-19: KG en función del tiempo para el estado $x_1$(superior) y $x_2$ (inferior) del KF-OA-FP. El ct para este caso es 0, lo que implica que el filtro no converge en relación al criterio adoptado.**

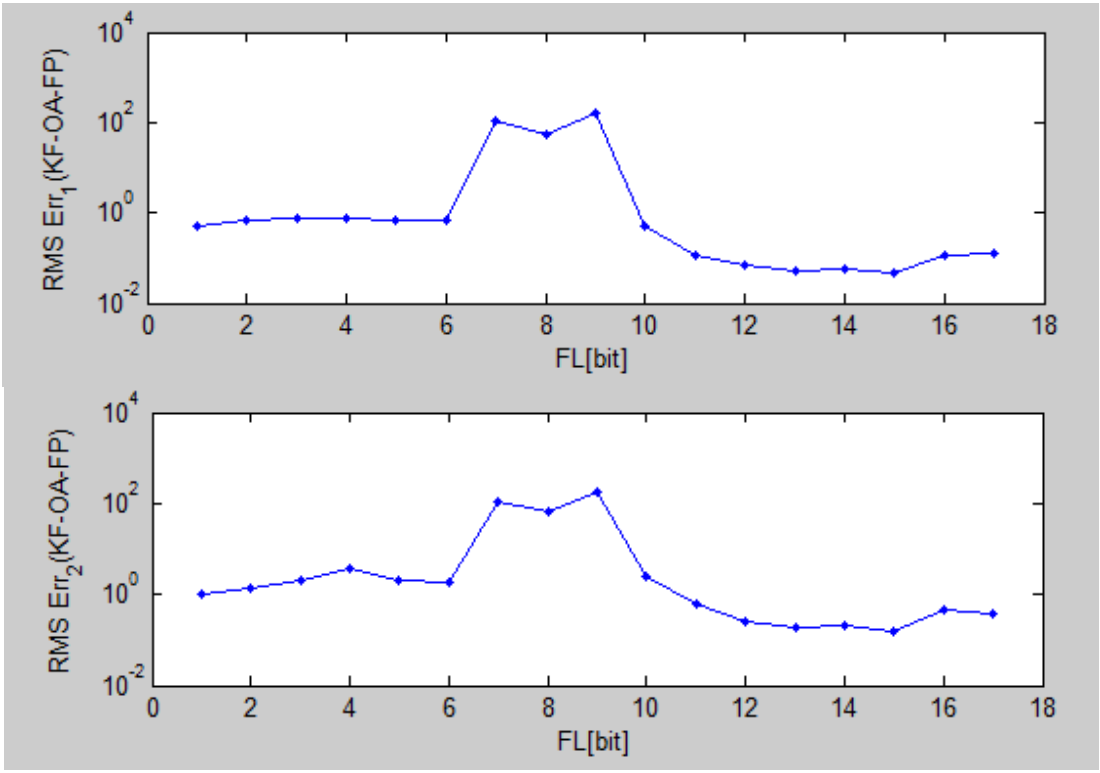



**Figura 6-20: Errores rms de estimación del algoritmo KF-OA-FP para el estado $x_1$(superior) y $x_2$(inferior) en función de FL. Nótese que el eje de ordenadas está en escala logarítmica.**

De la misma manera que para el caso anterior, se muestran a continuación los valores de los factores de mejora (Figura 6-21) y de los ct (Figura 6-22).

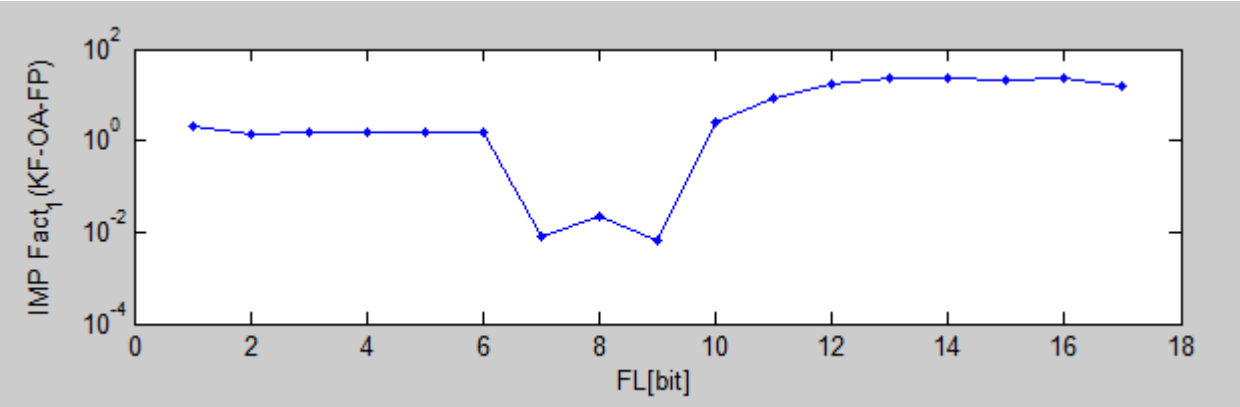



**Figura 6-21: Factor de Mejora para el estado $x_1$(el único que se mide) usando el KF-OA-FP en función de FL. Nótese que el eje de ordenadas está en escala logarítmica.**

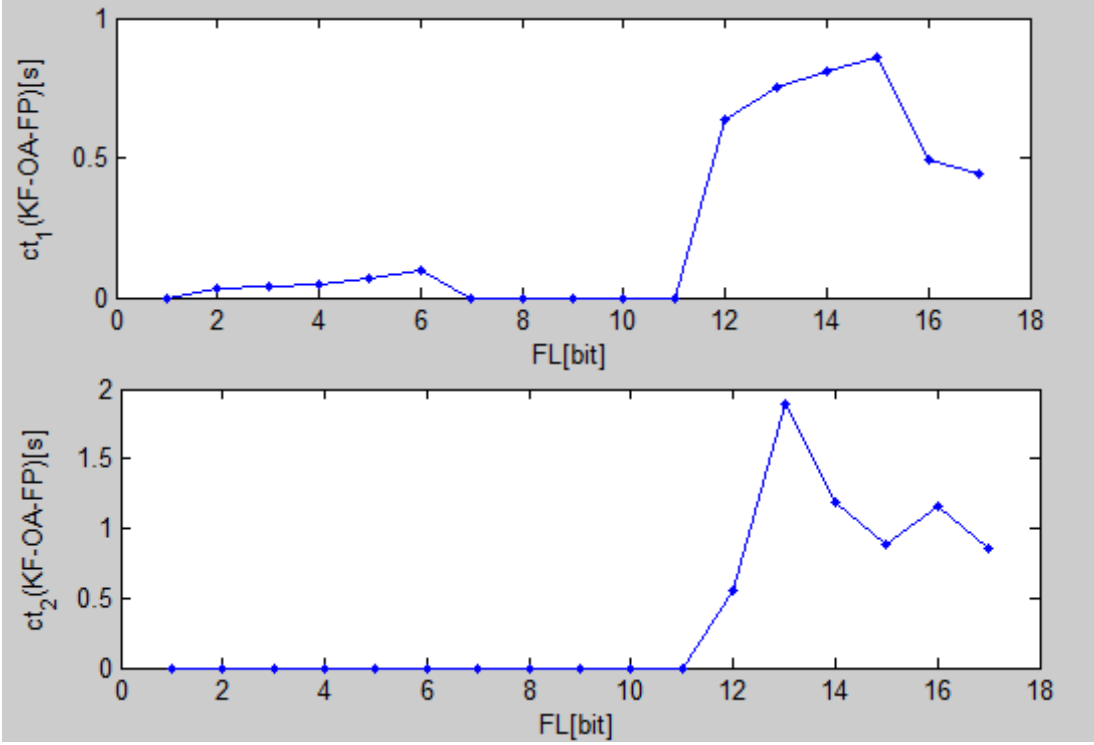



**Figura 6-22: ct del estado $x_1$ (superior) y $x_2$ (inferior) para diferentes FL del KF-OA-FP.**

### 6.5.4 Comparación del desempeño de BT en doble precisión y punto fijo

A fin de ejemplificar las diferencias en el comportamiento numérico del KF-BT operando en una aritmética como la de punto flotante de doble precisión (bien condicionado) y en una de mucho menor rango como la de FP (mal condicionado), realizamos las siguientes pruebas.

En primer lugar se compararon los errores rms de estimación para diferentes valores de FL. Como antes, para cada valor de FL se computaron 10 realizaciones y se promediaron los resultados. El resto de los parámetros que definen la aritmética se tomaron:

$$WL = 18$$

$$Round\ Mode = Floor$$

$$Overflow\ Mode = Saturate$$

En la siguiente figura se muestran los cocientes correspondientes a los errores rms de estimación para cada estado:

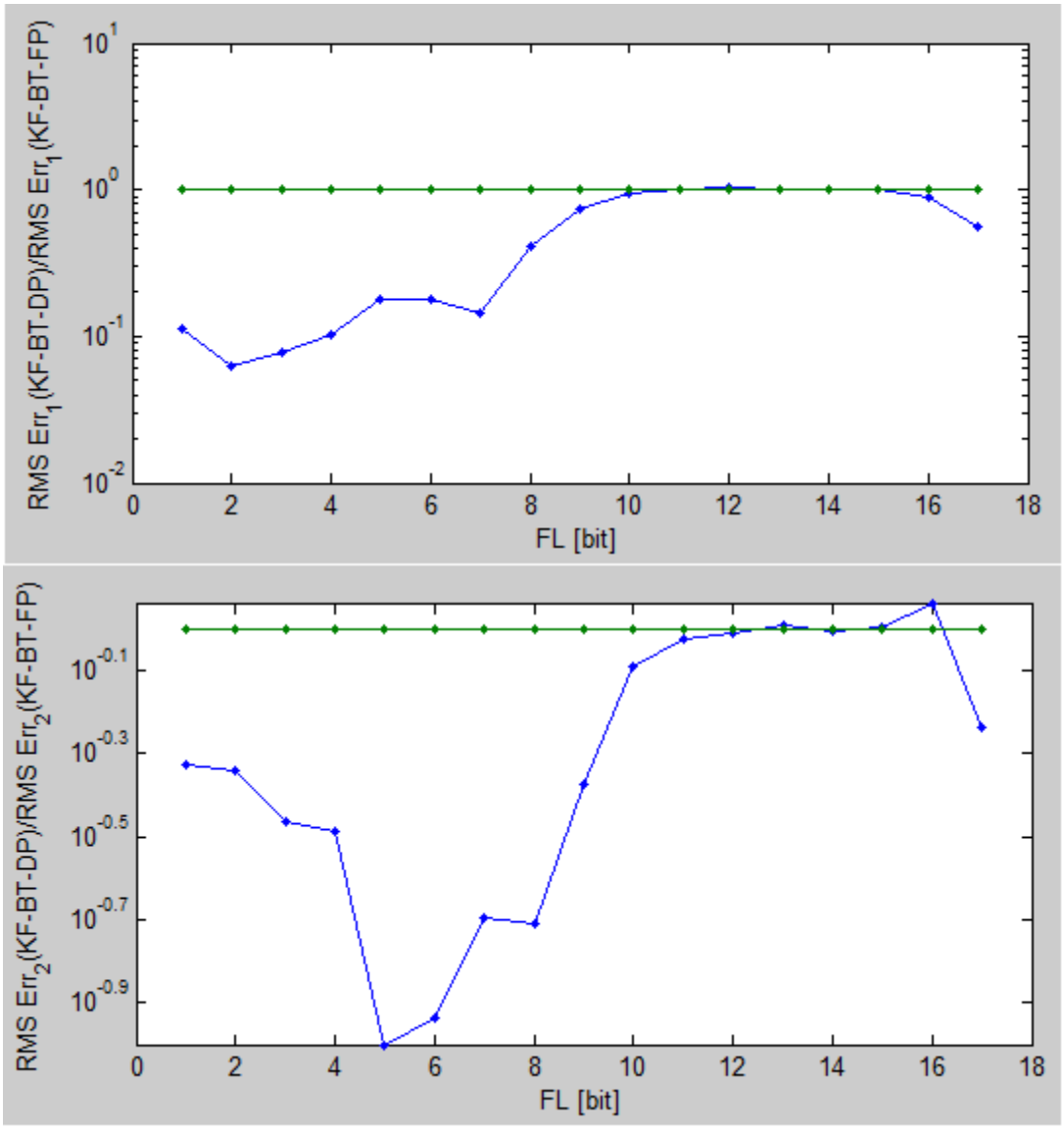



**Figura 6-23: Cocientes entre el error rms (en azul) de KF-BT-DP y KF-BT-FP para el estado $x_1$ (superior) y $x_2$ (inferior). Se muestra también la línea unitaria (en verde).**

Como era de esperarse los cocientes se mantienen casi siempre por debajo de la línea unitaria, reafirmando así la superioridad de la aritmética de doble precisión con respecto a la de FP.

Es importante resaltar los comportamientos que se observan para el estado 1 en $FL = 11, 12, 13\ y\ 15\ bits$ donde se aprecian errores rms levemente mayores con la aritmética de doble precisión; lo mismo sucede en el estado 2 para $FL = 13, 15\ y\ 16\ bits$. Esto es atribuible a las propiedades numéricas de la planta utilizada.

En la siguiente figura se muestran los cocientes correspondientes a los errores rms transitorios de estimación para cada estado:

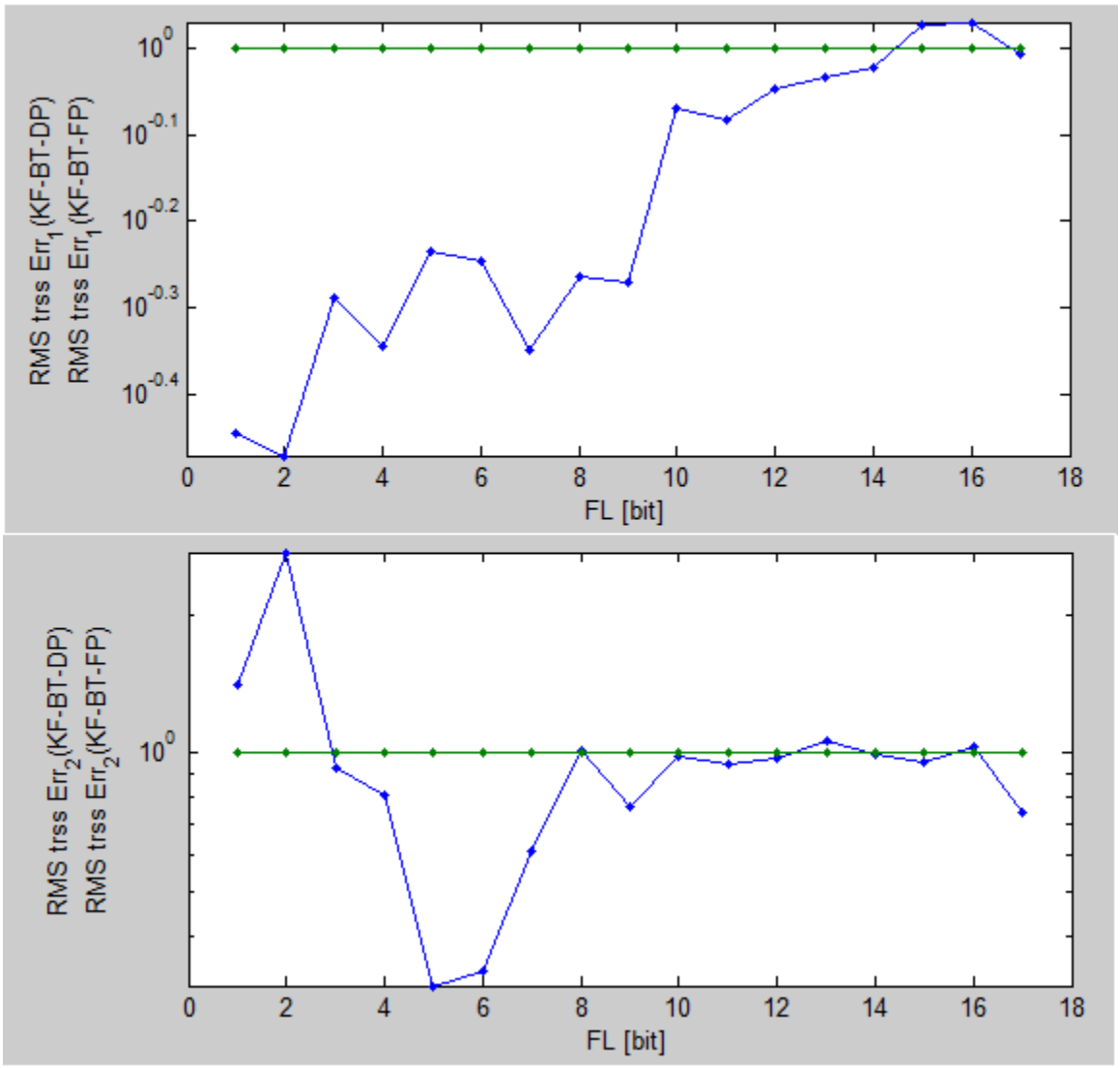



**Figura 6-24: Cocientes entre el error rms transitorio (en azul) de KF-BT-DP y KF-BT-FP para el estado $x_1$ (superior) y $x_2$ (inferior). Se muestra también la línea unitaria (en verde).**

En general se observa un comportamiento coherente con el hecho de la superioridad de la aritmética de doble precisión en comparación con la de FP. Sin embargo, en el estado 1 para una $FL = 15\, y\, 16$ y en el estado 2 para una $FL = 1, 2, 8, 13\, y\, 16$, se aprecian comportamientos inconsistentes con esto atribuibles a las propiedades numéricas de la planta utilizada.

En el estado 1 se observa una clara tendencia de decrecimiento en el error cometido por el sistema con aritmética de FP con respecto a la de doble precisión a medida que crece la $FL$. Se puede decir que, para este estado, las diferencias impuestas por la aritmética se atenúan con el crecimiento de FL. Esto último puede estar relacionado con el hecho de que el rango numérico del ejemplo desarrollado podría presentar números que son de valor absoluto pequeño.

En las siguientes figuras se muestran los cocientes correspondientes a los errores rms estacionarios de estimación para cada estado:

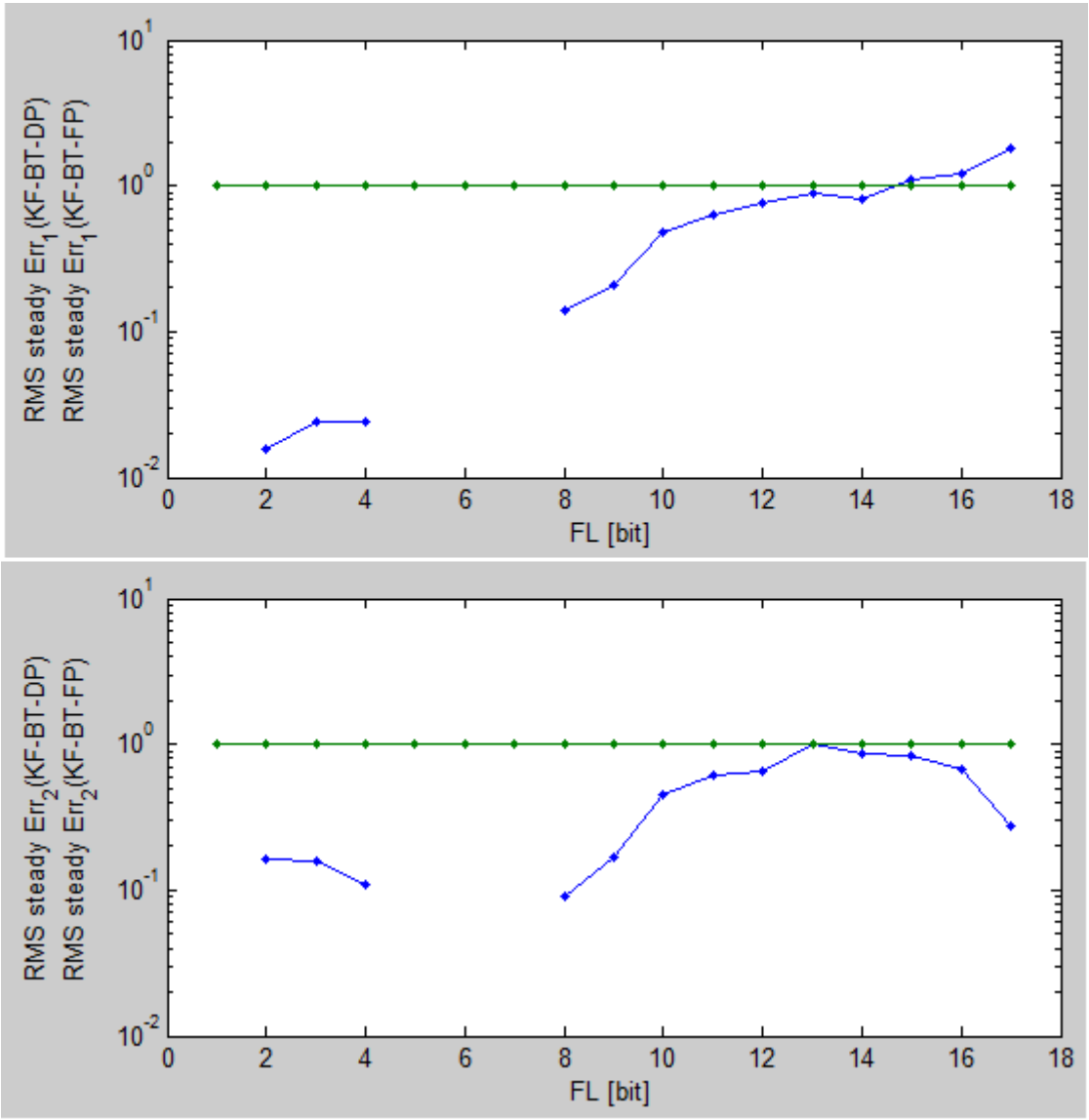



**Figura 6-25: Cocientes entre el error rms estacionario (en azul) de KF-BT-DP y KF-BT-FP para el estado $x_1$ (superior) y $x_2$ (inferior). Se muestra también la línea unitaria (en verde). Los puntos no mostrados corresponden a los valores de FL para los que el KF-BT-FP no convergió.**

Los cocientes se encuentran, en su gran mayoría, debajo de la línea unitaria, lo cual convalida la superioridad en la aritmética de doble precisión con respecto a la de FP. Se encuentran situaciones anómalas sólo en el estado 1 para $FL = 15, 16\ y\ 17\ bits$ y en el estado 2 para $FL = 13$.

Nuevamente, en el estado 1 se presenta una clara tendencia de disminución del error cometido en aritmética de FP al aumentar el FL. La ausencia de puntos en ambas gráficas para $5 \leq FL \leq 7$ implica la no convergencia en la aritmética de FP.

En la siguiente figura se muestran los cocientes correspondientes a los tiempos de convergencia para cada estado:

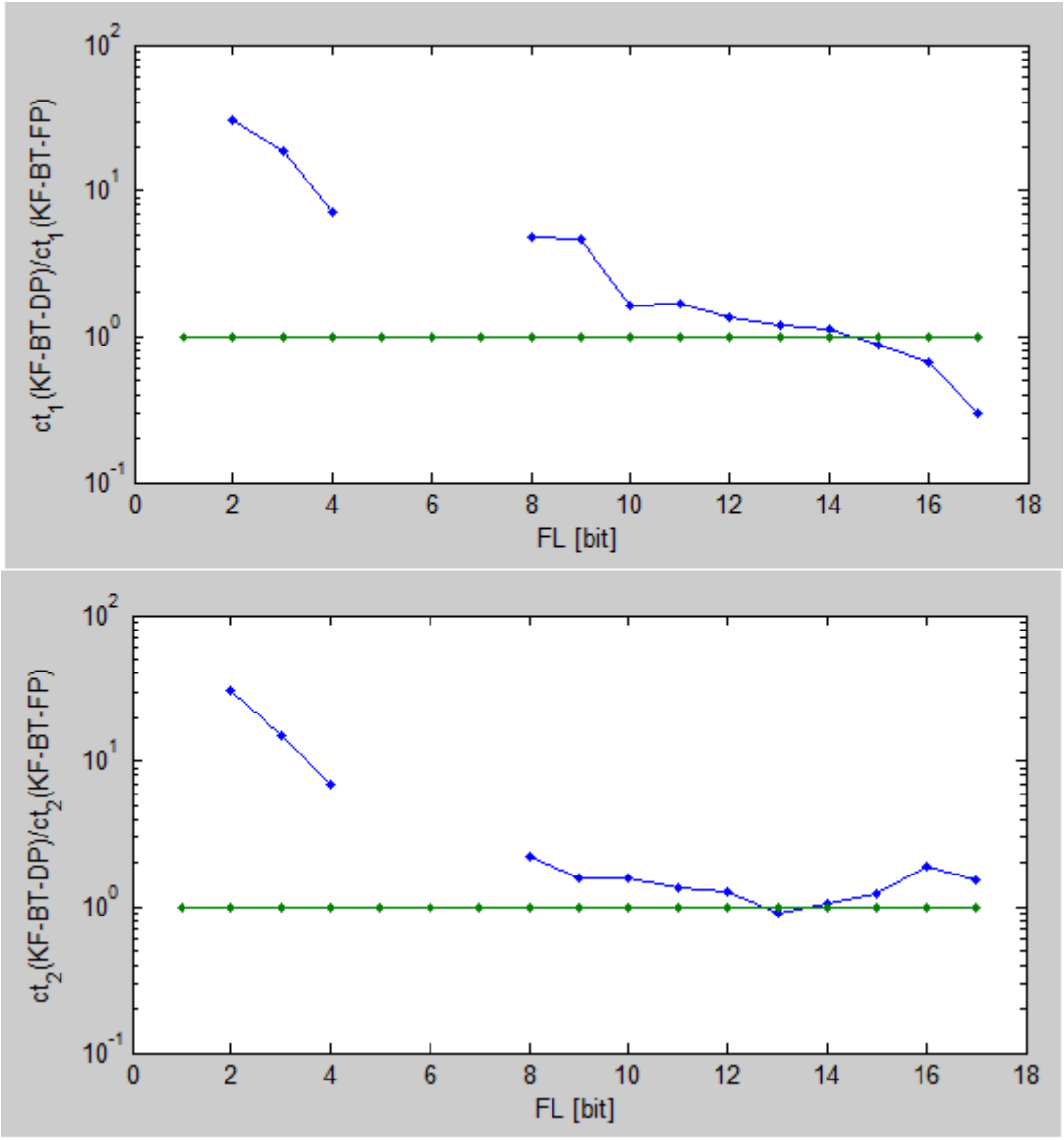



**Figura 6-26: Cocientes entre el ct (en azul) de KF-BT-DP y KF-BT-FP para el estado $x_1$ (superior) y $x_2$ (inferior). Se muestra también la línea unitaria (en verde). Los puntos no mostrados corresponden a los valores de FL para los que el KF-BT-FP no convergió.**

En su gran mayoría los tiempos de convergencia en doble precisión son mayores que en FP.

En general se observan tendencias de crecimiento en los ct de KF-BT-FP, para valores crecientes de FL.

Se observa aquí, en concordancia con la Figura 6-25 la ausencia de convergencia para $5 \leq FL \leq 7$.

En la siguiente figura se muestra el cociente correspondiente al Factor de Mejora:

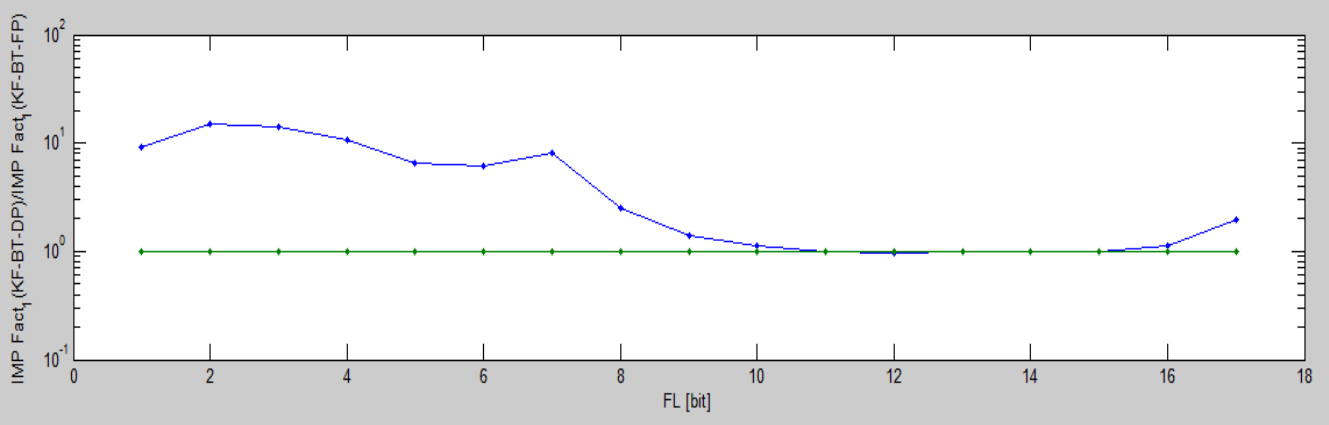



**Figura 6-27: Cocientes entre el Factor de Mejora (en azul) de KF-BT-DP y KF-BT-FP para el estado $x_1$. Se muestra también la línea unitaria (en verde).**

Como era de esperarse, los factores de mejora para el algoritmo ejecutándose con la aritmética de doble precisión son mayores que en FP para la mayoría de los casos

### 6.5.5 Comparación del desempeño de BT y el Algoritmo Original en punto fijo

Lo que se pretende aquí es resaltar las diferencias que surgen, entre ambos algoritmos, como consecuencia de la aplicación de la factorización.

En primer lugar se compararon los errores rms de estimación para diferentes valores de FL. Como antes para cada valor de FL se computaron 10 realizaciones y se promediaron los resultados. El resto de los parámetros que definen la aritmética se tomaron:

$$WL = 18$$

$$Round\ Mode = Floor$$

$$Overflow\ Mode = Saturate$$

En la siguiente figura se muestran los cocientes correspondientes a los errores rms de estimación para cada estado:

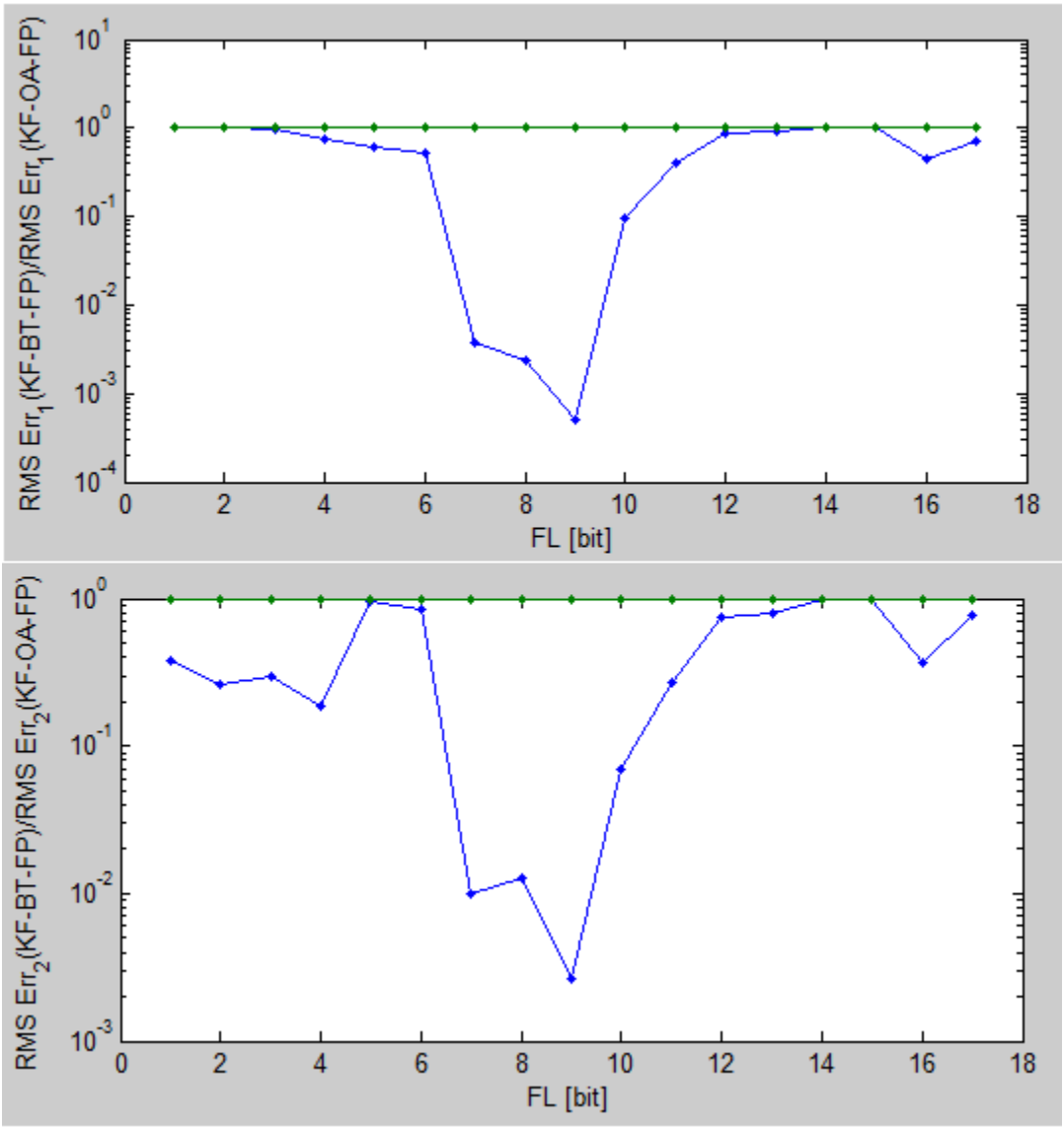



**Figura 6-28: Cocientes entre los errores rms de estimación (en azul) de KF-BT-FP y KF-OA-FP para el estado $x_1$ (superior) y $x_2$ (inferior). Se muestra también la línea unitaria (en verde).**

Es apreciable, claramente, la superioridad presentada por el algoritmo de KF-BT con respecto al KF-OA en cuanto a la inferioridad en sus errores RMS. Esto muestra que con la misma aritmética se consigue un mejor condicionamiento del sistema a través de la factorización de la matriz de covarianza del error que ofrece KF-BT. El único caso en el cual el error RMS del KF-BT es mayor al del KF-OA es en el estado 2 para una $FL = 15$ y se puede decir que la superioridad en dicho error es prácticamente nula comparada con los demás puntos del experimento. Los menores cocientes se ubican en la zona entre $7 \leq FL \leq 10$.

Nótese que si bien existen valores de FL para los que en principio la mejora en el error no es demasiada, debe recordarse que estos resultados dependen completamente de la planta utilizada. Estos resultados, mas lo expresado en al bibliografía, justifica la utilización del algoritmo de KF-BT-FP por sobre el KF-OA-FP dado que, según los objetivos de este trabajo, no se apunta a desarrollar el filtro mas optimo para una planta en particular.

En la siguiente figura se muestran los cocientes correspondientes a los errores rms transitorios de estimación para cada estado:

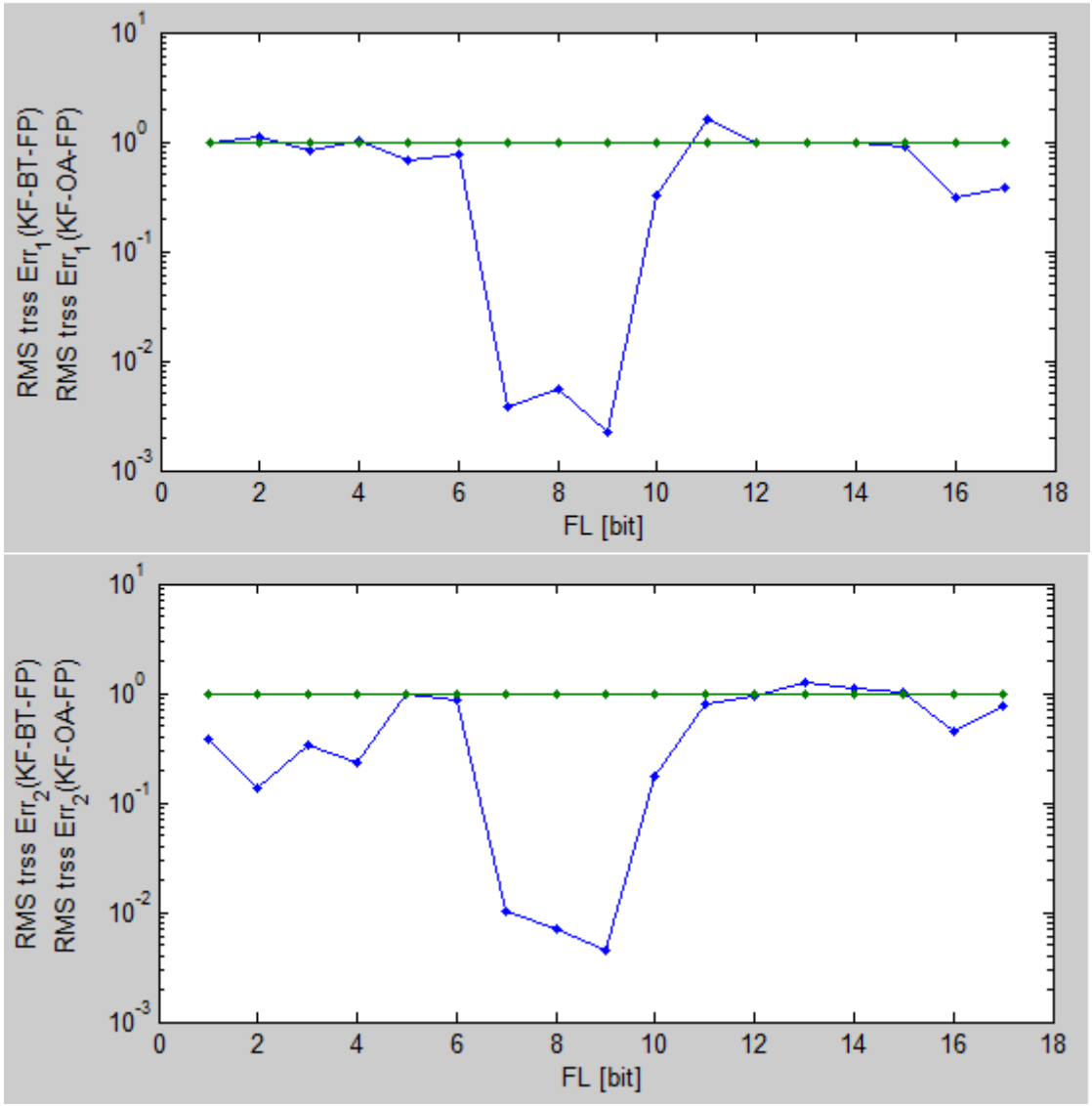



**Figura 6-29: Cocientes entre los errores rms de estimación transitorios (en azul) de KF-BT-FP y KF-OA-FP para el estado $x_1$ (superior) y $x_2$ (inferior). Se muestra también la línea unitaria (en verde).**

Se observan comportamientos coherentes con el mejor condicionamiento del KF-BT con respecto al KF-OA para la misma aritmética de punto fijo. Sin embargo en el estado 1 para una $FL = 2,4$ y $10\ bits$ y en el estado 2 para una $FL = 13,14$ y $15\ bits$ se aprecian errores transitorios mayores para el KF-BT. Los menores cocientes se encuentran en la zona $7 \leq FL \leq 10$.

En la siguiente figura se muestran los cocientes correspondientes a los errores rms estacionarios de estimación para cada estado:

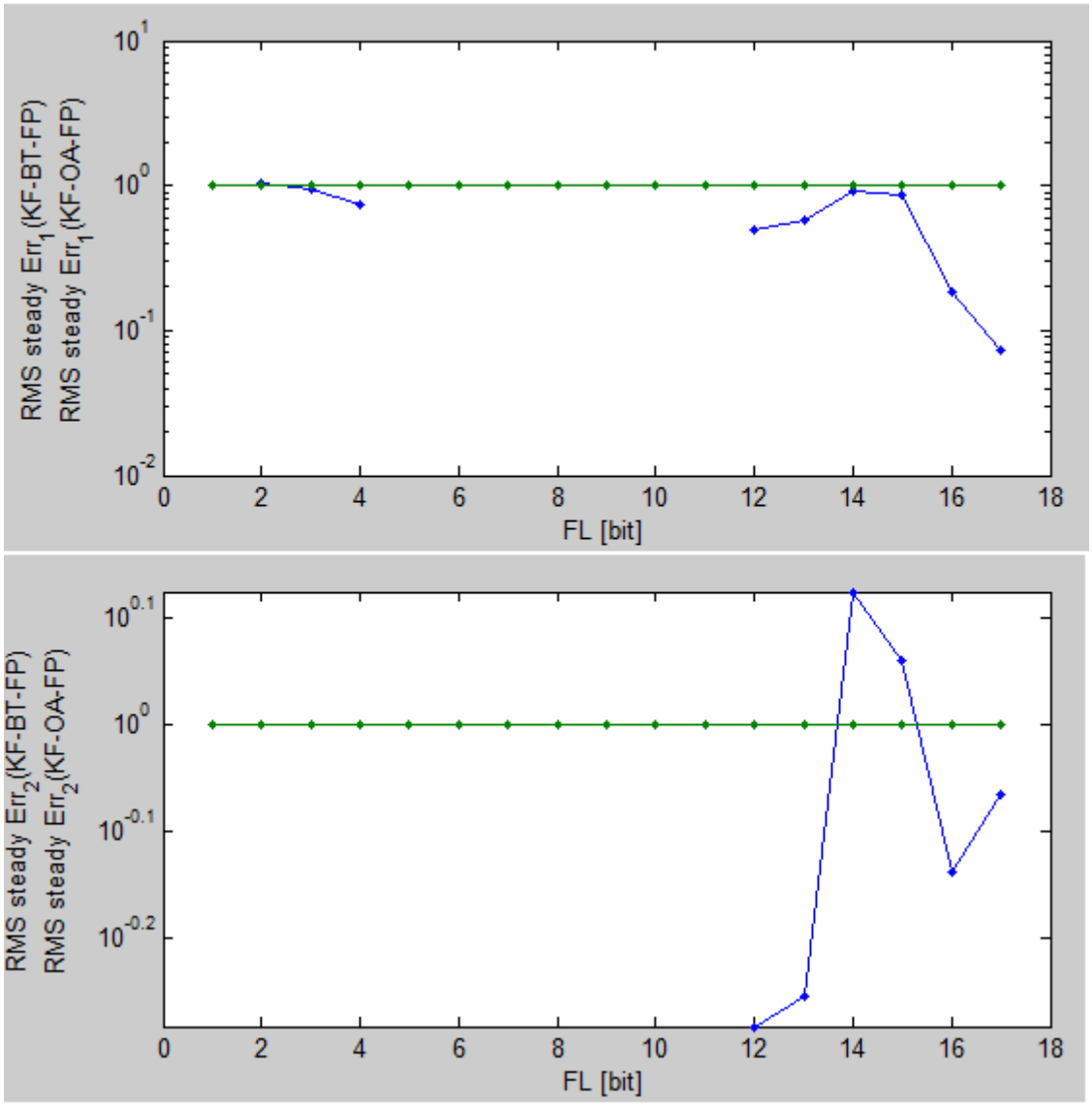



**Figura 6-30: Cocientes entre los errores rms de estimación estacionarios (en azul) de KF-BT-FP y KF-OA-FP para el estado $x_1$ (superior) y $x_2$ (inferior). Se muestra también la línea unitaria (en verde).**

En el estado 1 los errores correspondientes a la ejecución del KF-BT están, en su gran mayoría, por debajo de los correspondientes a la ejecución del KF-OA. Esto indica una situación de coherencia con el hecho del mejor condicionamiento de KF-BT. En el estado 2, si bien la tendencia no es tan marcada, hay cuatro aritméticas contra dos en las cuales el error cometido por la ejecución del KF-BT es menor que en la ejecución del KF-OA.

Se observa una gran cantidad de puntos ($5 \leq FL \leq 11$ para el estado 1 y $1 \leq FL \leq 11$ para el estado 2) en los cuales no se verifica una situación de convergencia. Teniendo en cuenta que el algoritmo KF-BT no converge sólo para los casos en los cuales $5 \leq FL \leq 7$, se observa que las no convergencias observadas en la figura se deben al KF-OA excepto para el estado 1 en $FL = 5\ y\ 6\ bits$, en cuyos casos el KF-OA converge para dicho estado mientras que el KF-BT no.

En la Figura 6-31, se muestran los cocientes correspondientes a los tiempos de convergencia para cada estado.

Los tiempos de convergencia para el KF-BT son mayores para el KF-OA para el estado 1, mientras que para el estado 2 cambia la tendencia.

Se observa ausencia de puntos para los casos en los cuales no se tiene una situación de convergencia en la ejecución de los algoritmos ($5 \leq FL \leq 11$ para el estado 1 y $1 \leq FL \leq 11$ para el estado 2). Situación consistente con la presentada en la Figura 6-30.

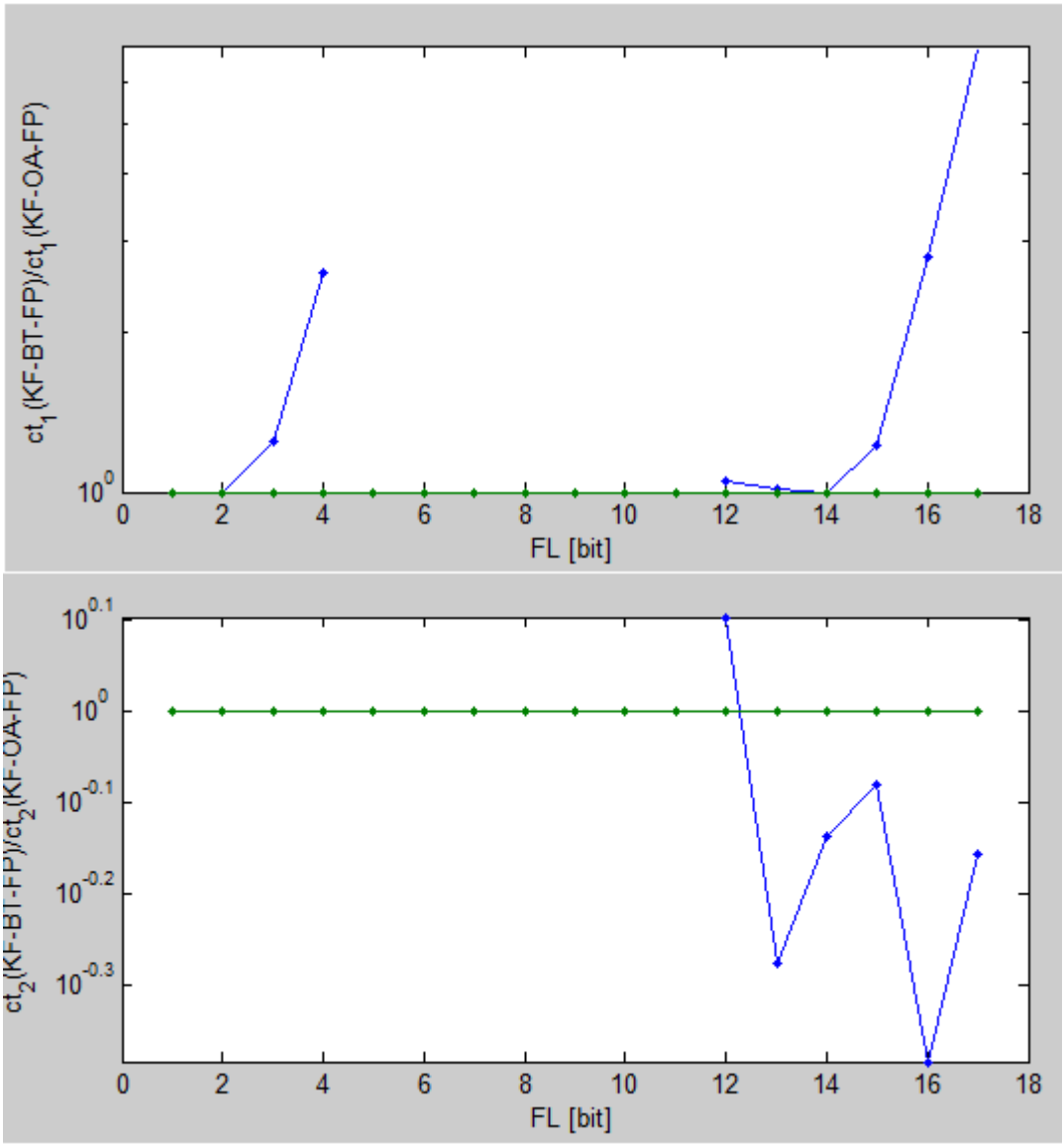



**Figura 6-31: Cocientes entre los ct (en azul) de KF-BT-FP y KF-OA-FP para el estado $x_1$ (superior) y $x_2$ (inferior). Se muestra también la línea unitaria (en verde). Los puntos no mostrados corresponden a situaciones en que alguno de los dos algoritmos no convergió.**

En la Figura 6-32, se muestra el cociente correspondiente al factor de mejora.

Excepto para $FL = 1, 2\ y\ 3\ bits$ se presenta un factor de mejora superior para el KF-BT que para el KF-OA, lo cual era de esperarse basándose en el mejor condicionamiento del sistema para el KF-BT debido a la factorización.

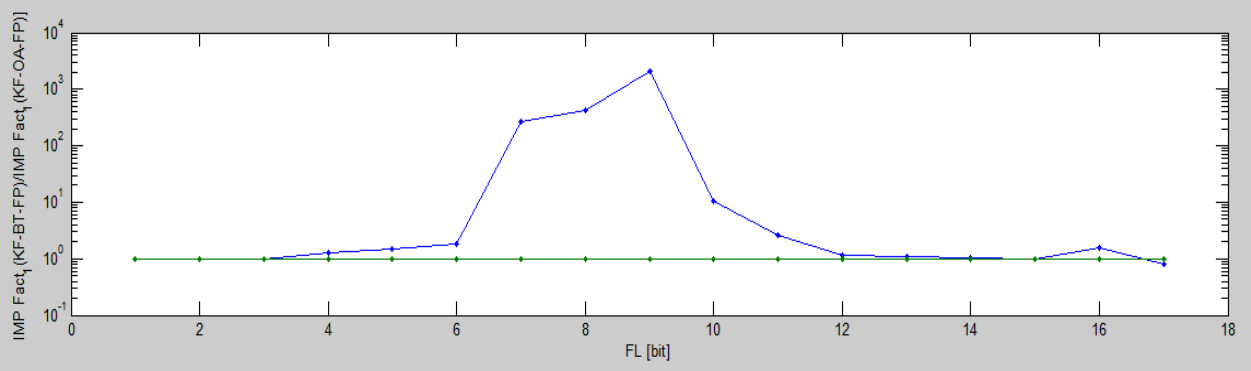



**Figura 6-32: Cocientes entre el Factor de Mejora (en azul) de KF-BT-FP y KF-OA-FP para el estado $x_1$. Se muestra también la línea unitaria (en verde).**

## 6.6 Selección de los parámetros óptimos de la aritmética de punto fijo

En la sección 6.5 se justificó la decisión de utilizar el algoritmo del KF-BT-FP mediante un análisis de su desempeño en relación a su implementación en punto flotante de doble precisión (KF-BT-DP) y a la implementación original de Kalman en punto fijo (KF-OA-FP).

Luego de seleccionada la implementación, se presenta el problema de elegir los parámetros óptimos de la aritmética de FP a utilizar. Primero debemos advertir que este es un problema complejo en el que intervendrán variables de diversa índole, pues, la elección adecuada de sendos parámetros influirá no solo en la performance del sistema si no también en el hardware requerido y por ende en el costo final del sistema.

Aquí nos limitaremos a elaborar algunos criterios técnicos de selección a fin de establecer una guía preliminar para determinar los parámetros de la aritmética de FP.

Longitud de la palabra (WL):

La longitud de palabra óptima se determinaría por ejemplo para el caso de la implementación del KF en un ASIC con el fin de minimizar tanto el error de estimación (y posiblemente el tiempo de convergencia) como el costo del sistema [**12**]. Esto se muestra en la Figura 6-33.

Sin embargo para el caso de la implementación en FPGA, el hardware utilizado no puede variar de manera continua como en un ASIC (Figura 6-33). Por esto la utilización de los recursos de hardware embebidos en la FPGA, tendrá gran relevancia en el desempeño final del sistema y por lo tanto afectará en la elección del WL óptimo.

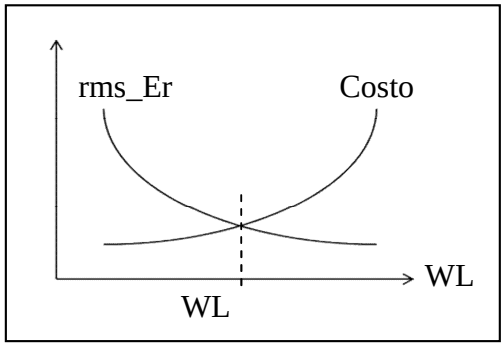



**Figura 6-33: Criterio inicial para seleccionar la longitud de la palabra de trabajo.**

Debido a que en cada iteración del algoritmo de KF-BT-FP se computan gran cantidad de multiplicaciones de números en punto fijo con longitud de palabra de WL, es de particular interés el uso de los módulos DSP embebidos en la FPGA para efectuar sendas multiplicaciones. Estos módulos se estudiarán en detalle en la sección 7.1.1, aquí nos interesa el hecho de que la utilización de los DSP implica una reducción del tiempo de computo del filtro así como del consumo. Esto sumado al hecho de que este recurso viene de fábrica embebido en la FPGA (por lo que no utilizarlos sería un desperdicio de hardware) determina que el WL se elija en base a la palabra que los DSP pueden manejar.

Para el caso de la FPGA utilizada en este trabajo, los módulos DSP trabajan con operandos de 18 bits en complemento a 2. Por esto hacemos:

$$WL = 18[bit]$$

**Ecuación 6-26**

Modo de Overflow

Sobre los dos modos de tratamiento del Overflow que se discutieron en la sección 6.1.2 se sabe que el de *Saturar* tiene mucho mejor comportamiento numérico que el de Recortar. Aquí adoptaremos el primero a pesar de que su utilización requiere un poco más de hardware:

$$Overflow\ Mode = Saturate$$

**Ecuación 6-27**

De todas formas incluimos a modo de referencia, la Figura 6-34, en la que se muestra el Factor de Mejora obtenido utilizando estas dos opciones de tratamiento de Overflow y para diversos valores de FL. Esta figura fue obtenida mediante el programa Arithmetic_selection.m codificado para tal fin y que se halla adjunto en el anexo 11.1.5.

Modo de Redondeo

Como se vio en la sección 6.1.3, existen diversos métodos de redondeo que en general tendrán distintos comportamientos numéricos y requerimientos de hardware. Aquí utilizaremos:

$$Round\ Mode = Floor$$

**Ecuación 6-28**

Motivados por el hecho de que este es el método que requiere de menos hardware para ser sintetizado, pues solo se deben truncar (desechar) los bits como se explicó en la sección 6.1.3. Sin embargo, dado los objetivos de este trabajo, incluimos a modo de referencia la siguiente figura, en la que se muestra el valor del Factor de Mejora en función del FL para dos modos de redondeo distintos (Fix de mejor comportamiento numérico y Floor).

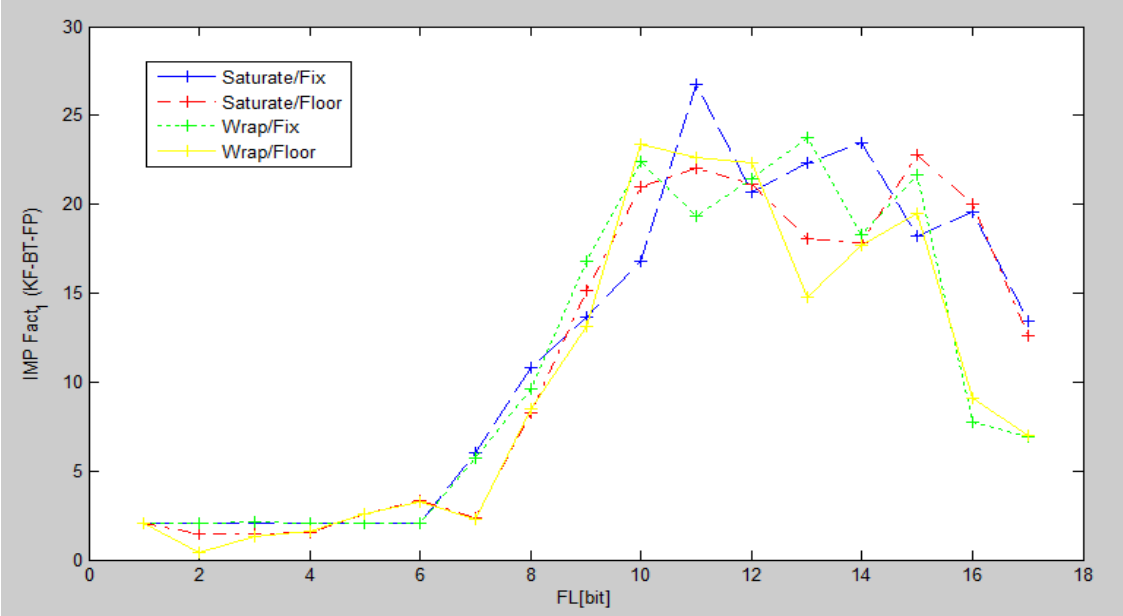



**Figura 6-34: Factor de Mejora promedio (para 10 realizaciones) del KF-BT-FP en función del FL y para 4 combinaciones de los modos de redondeo y Overflow especificadas en la leyenda.**

Como puede verse el desempeño de las diferentes opciones varía según el FL de manera bastante compleja, sin quedar determinada la superioridad numérica absoluta de ninguna de ellos.

Longitud de la parte fraccional (FL)

Para seleccionar la posición del punto decimal en la palabra (FL), debe establecerse un criterio de ponderación. Aquí utilizaremos la comparación de los Factores de Mejora y tiempos de convergencia (nótese que no incluimos el hardware requerido para sintetizar cada opción aunque este puede resultar determinante dependiendo de los objetivos del proyecto que se lleve a cabo. Esto se debe a que es necesario conocer en detalle la relación entre el hardware utilizado y el costo final del sistema si se desea realizar un análisis cuantitativo como el que presentamos en esta sección). Repetiremos aquí la Figura 6-16 y la parte correspondiente al estado 1 de la Figura 6-17 en una misma presentación dada en la Figura 6-35.

Analizaremos el primero de los estados (pues es el único para el que el Factor de Mejora está definido) aunque esto no es una regla general, el peso relativo que el desempeño del filtro en cada estado tiene sobre lo que se considere el desempeño general depende de la aplicación (por ejemplo puede ser más critico mantener un bajo error en la estimación de las aceleraciones debido a sus efectos mecánicos nocivos que en la posición o la velocidad para un vehículo determinado).

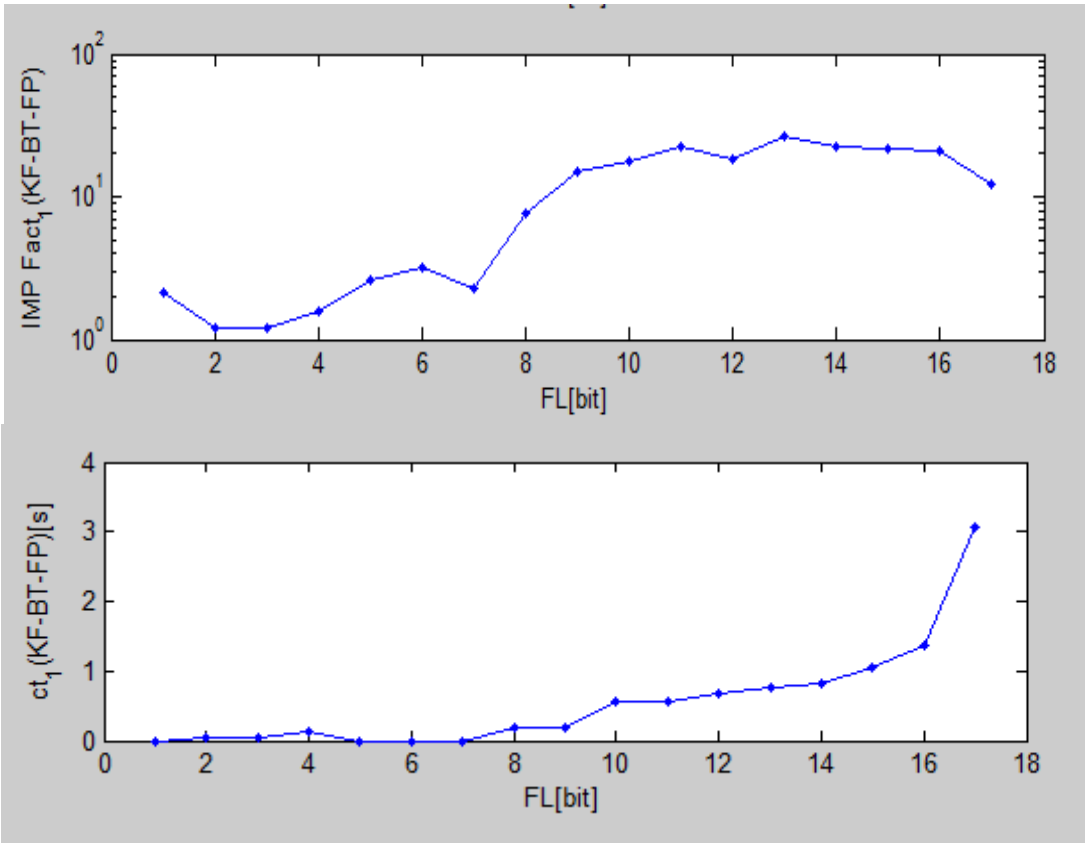



**Figura 6-35: Factores de Mejora (superior) y ct (inferior) en función del FL para el KF-BT-FP.**

Como se ve en la Figura 6-35, tanto el $ct$ como el Factor de Mejora ($IMP\_Fact$) crecen al crecer $FL$. Esto también puede verse en la Tabla 6-1 donde presentamos los valores numéricos de ambas cantidades (columnas 2 y 4) así como el valor normalizado de $ct$ (columna 3), el reciproco del $IMP\_Fact$ (columna 5) una suerte de factor de desmejora y su valor normalizado (columna 6).

| FL [bit] | ct [s] | ct_norm | IMP_Fact | 1/IMP_Fact_norm | (1/IMP_Fact)_norm |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 2.057530676 | 0.486019485 | 0.597640923 |
| 2 | 0.03 | 0.009803922 | 1.681341239 | 0.594763262 | 0.731359288 |
| 3 | 0.05 | 0.016339869 | 1.229664532 | 0.813229929 | 1 |
| 4 | 0.13 | 0.04248366 | 1.547351761 | 0.646265462 | 0.794689717 |
| 5 | 0 | 0 | 2.710548027 | 0.368929084 | 0.453659009 |
| 6 | 0 | 0 | 3.334437303 | 0.299900676 | 0.368777224 |
| 7 | 0 | 0 | 2.361326057 | 0.423490859 | 0.520751689 |
| 8 | 0.19 | 0.062091503 | 7.828811101 | 0.127733316 | 0.157069128 |
| 9 | 0.2 | 0.065359477 | 13.23147564 | 0.07557736 | 0.092934799 |
| 10 | 0.57 | 0.18627451 | 15.56977596 | 0.064227 | 0.078977664 |
| 11 | 0.55 | 0.179738562 | 23.23997853 | 0.0430293 | 0.052911604 |
| 12 | 0.67 | 0.218954248 | 18.94123859 | 0.052794858 | 0.064919964 |
| 13 | 0.76 | 0.248366013 | 20.82340234 | 0.048022892 | 0.059052047 |
| 14 | 0.81 | 0.264705882 | 17.76639398 | 0.056286042 | 0.06921295 |
| 15 | 1.05 | 0.343137255 | 25.8947616 | 0.038617849 | 0.047487 |
| 16 | 1.37 | 0.447712418 | 19.79554572 | 0.050516415 | 0.062118244 |

| | | | | | |
|---|---|---|---|---|---|
| 17 | 3.06 | 1 | 11.3690043 | 0.08795845 | 0.108159387 |

**Tabla 6-1: Valores numéricos correspondientes a la Figura 6-35: *ct* e *IMP_Fact* (columnas 2 y 4), además el valor normalizado de *ct* (columna 3), el reciproco del *IMP_Fact* (columna 5) una suerte de factor de desmejora y su valor normalizado (columna 6).**

Si se prioriza únicamente el Factor de Mejora, como se ve en la parte superior de la Figura 6-35, debería hacerse: $FL = 15\,[bit]$ correspondiente al máximo $IMP\_Fact = 25.89$ . Sin embargo para este punto el $ct = 1.05\,[s]$ uno de los peores (ver Tabla 6-1). *Recordar que* $ct = 0\,[s]$ *implica la no convergencia.*

Si en cambio se busca el $FL$ que maximice él $IMP\_Fact$ y a la vez minimice el $ct$, debemos buscar el punto en donde las curvas del reciproco del primero se cruzan con las del segundo.

Esto se muestra en la Figura 6-36, elaborada con los datos de la Tabla 6-1. Como puede verse, el valor optimo se da para $9\,[bit] < FL < 10[bit]$.

Nosotros adoptaremos:

$$FL = 9\,[bit]$$

**Ecuación 6-29**

Nótese que, en base a lo expuesto en la Figura 6-34 para el valor del $FL$ elegido, la combinación de métodos de redondeo y Overflow seleccionada, satúrate y floor respectivamente, es la que tiene el segundo Factor de Mejora más alto.

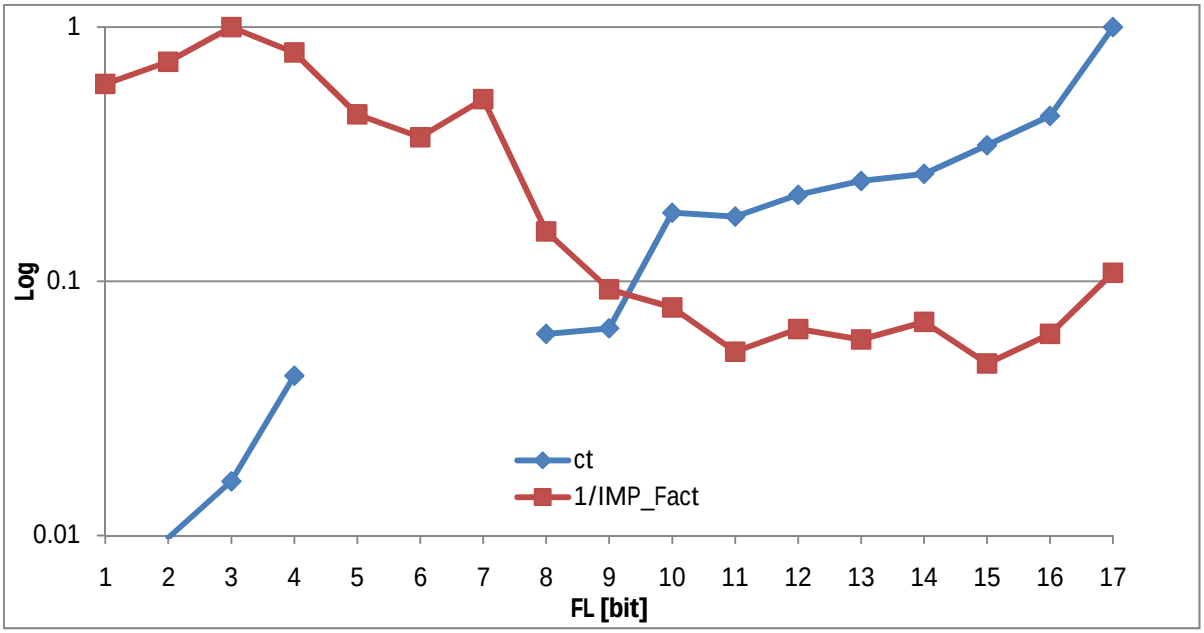



**Figura 6-36: Curvas normalizadas de $1/IMP\_Fact$ (en rojo) y *ct* (en azul) mostradas con una escala vertical logarítmica para poder localizar el valor de *FL* óptimo.**

# 7 Implementación del KF-BT en FPGA

## 7.1 Selección y especificaciones del hardware utilizado

La descripción del KF-BT-FP en VHDL se realizó de manera general y escalable. Esto permite que puedan describirse filtros de mayor orden con solo modificar algunos parámetros. Esto se verá limitado si se desea sintetizar luego el código por los recursos de hardware disponibles en la FPGA (ver sección 8.2).

La síntesis del KF-BT-FP se realizo en una FPGA de la empresa XILINX (introducidas en la sección 1). Dado de que el presente proyecto no tiene como objetivo ninguna familia de dispositivos en particular, la selección está limitada por la filosofía de diseño del hardware que se adopte. Aquí como se verá en la 7.4 se optó por una implementación de ejecución paralela, esto determinó que la FPGA a utilizar tuviese que contar con la cantidad de módulos DSP adecuada para que el software de síntesis pudiese operar correctamente (este problema se comenta en la sección 7.6). Debido a esto, utilizamos para realizar las pruebas de implementación la FPGA XC6SLX45-3CSG324 de la familia Spartan 6. En el anexo 11.3 se encuentra la hoja de datos de esta familia de dispositivos.

Las principales características de este dispositivo (fabricado con tecnología de $45\ [nm]$), se hallan en la tabla 1 del citado anexo, de la cual repetimos en la siguiente figura la entrada correspondiente a la FPGA seleccionada.

**Spartan-6 FPGA Feature Summary**

*Table 1:* **Spartan-6 FPGA Feature Summary by Device**

| Device | Logic Cells(1) | Configurable Logic Blocks (CLBs) | | | DSP48A1 Slices(3) | Block RAM Blocks | | CMTs(5) | Memory Controller Blocks (Max)(6) | Endpoint Blocks for PCI Express | Maximum GTP Transceivers | Total I/O Banks | Max User I/O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Slices(2) | Flip-Flops | Max Distributed RAM (Kb) | | 18 Kb(4) | Max (Kb) | | | | | | |
| XC6SLX45 | 43,661 | 6,822 | 54,576 | 401 | 58 | 116 | 2,088 | 4 | 2 | 0 | 0 | 4 | 358 |

**Figura 7-1: Características más relevantes de la FPGA XC6SLX45-3CSG324 que utilizamos para realizar las pruebas de implementación.**

### 7.1.1 Módulos DSP

Como se introdujo en la sección 6.5.4 una de las principales características del hardware disponible en una FPGA en lo que respecta a la implementación del KF son los módulos DSP embebidos. Estos permiten implementar diversas operaciones y en especial la multiplicación de números de 18 bits en complemento a 2 disminuyendo sustancialmente el tiempo de cómputo en relación a la implementación de los multiplicadores en los CLBs.

Para el caso de la familia Spartan 6 los bloques DPS (o DSP Slices) están constituidos como se muestra en la siguiente figura [**13**]:

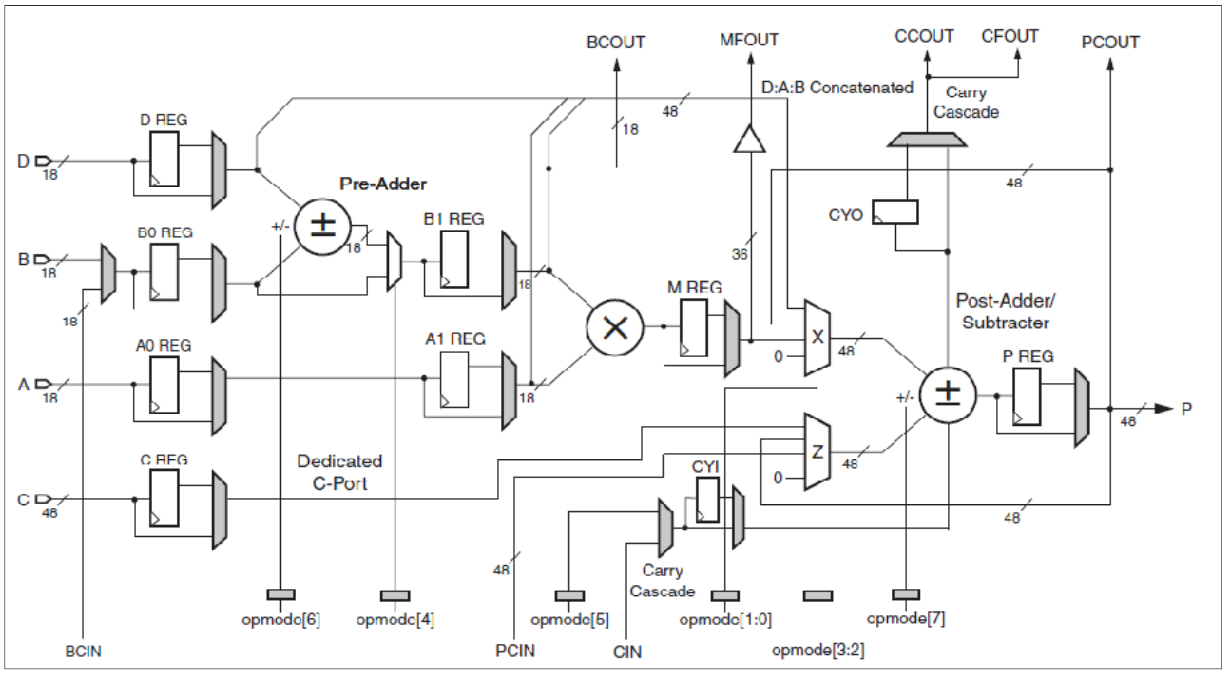



**Figura 7-2: DSP slice de la familia de FPGAs Spartan 6 de Xilinx, denotado DSP48A1.**

Como se muestra en la Tabla 6-1 la FPGA seleccionada tiene 58 DSP48A1 embebidos distribuidos en dos columnas iguales. Los elementos sombreados en gris en la Figura 7-2 son multiplexores configurables mediante los archivos de User Constraint (UCF) que permiten la implementación de una gran cantidad funciones aritméticas distintas entre los cuatro operandos de entrada (A, B, C y D). Estas incluyen tanto versiones registradas (para aprovechar al máximo la capacidad de implementar Pipeline) como la operación asíncrona (no registrada).

En la Figura 7-3 se muestran todas las posibles operaciones que puede sintetizar el DSP48A1 segun [**13**]. *Para la implementación del KF-BT-FP emplearemos el módulo DSP como un multiplicador asíncrono en complemento a 2, esto corresponde a la operación "Multiply" de la Figura 7-3 que como se ve, ejecuta el producto entre las dos entradas de 18 bits A y B*.

Como veremos en la sección 7.4.3, si se codifica en VHDL la operación de multiplicación de la forma adecuada el sintetizador inferirá automáticamente la configuración correcta para cada DSP48A1, de manera de que se ejecute en cada uno de ellos una multiplicación de $18x18$ bits por cada ciclo de reloj.

*Table 1-7:* **DSP48A1 Operating Modes**

| OPMODE | OPMODE[7:0] | | | | | | | | CARRYINSEL | Output |
|---|---|---|---|---|---|---|---|---|---|---|
| | Post-Adder/ Subtracter | Pre_Adder/ Subtracter | CIN | Pre-Adder/ Subtracter Select | Z | | X | | | |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |
| Zero + CIN | 0 | 0/1 | 0/1 | 0/1 | 0 | 0 | 0 | 0 | 0 | + CIN |
| Zero – CIN | 1 | 0/1 | 0/1 | 0/1 | 0 | 0 | 0 | 0 | 0 | – CIN |
| Zero + OPMODE[5] | 0 | 0/1 | 0/1 | 0/1 | 0 | 0 | 0 | 0 | 1 | + OPMODE[5] |
| Zero – OPMODE[5] | 1 | 0/1 | 0/1 | 0/1 | 0 | 0 | 0 | 0 | 1 | – OPMODE[5] |
| Hold P | 0/1 | 0/1 | 0/1 | 0/1 | 0 | 0 | 1 | 0 | 0/1 | ±(P + CIN) |
| D:A:B Select | 0/1 | 0/1 | 0/1 | 0 | 0 | 0 | 1 | 1 | 0/1 | ± (D:A:B + CIN) |
| D:A:B Select: with PreAdd | 0/1 | 0/1 | 0/1 | 1 | 0 | 0 | 1 | 1 | 0/1 | ± (D:A:(D ±B) + CIN) |
| Multiply | 0/1 | 0/1 | 0/1 | 0 | 0 | 0 | 0 | 1 | 0/1 | ±(A × B + CIN) |
| PreAdd-Multiply | 0/1 | 0 | 0/1 | 1 | 0 | 0 | 0 | 1 | 0/1 | ±(A × (D + B) + CIN) |
| PreSubtract-Multiply | 0/1 | 1 | 0/1 | 1 | 0 | 0 | 0 | 1 | 0/1 | ± (A × (D – B) + CIN) |
| P Cascade Select | 0/1 | 0/1 | 0/1 | 0/1 | 0 | 1 | 0 | 0 | 0/1 | PCIN ±CIN |
| P Cascade Feedback Add | 0/1 | 0/1 | 0/1 | 0/1 | 0 | 1 | 1 | 0 | 0/1 | PCIN ±(P + CIN) |
| P Cascade Add | 0/1 | 0/1 | 0/1 | 0 | 0 | 1 | 1 | 1 | 0/1 | PCIN ±(D:A:B + CIN) |
| P Cascade Add with PreAdd | 0/1 | 0/1 | 0/1 | 1 | 0 | 1 | 1 | 1 | 0/1 | PCIN ± (D:A:(D ± B) + CIN |
| P Cascade Multiply Add | 0/1 | 0/1 | 0/1 | 0 | 0 | 1 | 0 | 1 | 0/1 | PCIN ±(A × B + CIN) |
| Cascade PreAdd-Multiply | 0/1 | 0 | 0/1 | 1 | 0 | 1 | 0 | 1 | 0/1 | PCIN ± (A × (D + B) + CIN) |
| Cascade PreSubtract-Multiply | 0/1 | 1 | 0/1 | 1 | 0 | 1 | 0 | 1 | 0/1 | PCIN ± (A × (D – B) + CIN) |
| Feedback Carryin Add | 0/1 | 0/1 | 0/1 | 0/1 | 1 | 0 | 0 | 0 | 0/1 | P ± CIN |
| Double Feedback Add | 0/1 | 0/1 | 0/1 | 0/1 | 1 | 0 | 1 | 0 | 0/1 | P ± (P + CIN) |
| Feedback Add | 0/1 | 0/1 | 0/1 | 0 | 1 | 0 | 1 | 1 | 0/1 | P ± (D:A:B + CIN) |
| Feedback Addwith PreAdd | 0/1 | 0/1 | 0/1 | 1 | 1 | 0 | 1 | 1 | 0/1 | P ± (D:A:(D ± B) + CIN) |
| Multiply-Accumulate | 0/1 | 0/1 | 0/1 | 0 | 1 | 0 | 0 | 1 | 0/1 | P ± (A × B + CIN) |
| Feedback PreAdd-Multiply | 0/1 | 0 | 0/1 | 1 | 1 | 0 | 0 | 1 | 0/1 | P ± (A × (D + B) + CIN) |
| Feedback PreSubtract-Multiply | 0/1 | 1 | 0/1 | 1 | 1 | 0 | 0 | 1 | 0/1 | P ± (A × (D – B) + CIN) |
| C Select | 0/1 | 0/1 | 0/1 | 0/1 | 1 | 1 | 0 | 0 | 0/1 | C ± CIN |
| Feedback Add | 0/1 | 0/1 | 0/1 | 0/1 | 1 | 1 | 1 | 0 | 0/1 | C ± (P + CIN) |
| 48-Bit Adder | 0/1 | 0/1 | 0/1 | 0 | 1 | 1 | 1 | 1 | 0/1 | C ± (D:A:B + CIN) |
| Multiply-Add | 0/1 | 0/1 | 0/1 | 0 | 1 | 1 | 0 | 1 | 0/1 | C ± (A × B + CIN) |
| C PreAdd-Multiply | 0/1 | 0 | 0/1 | 1 | 1 | 1 | 0 | 1 | 0/1 | C ± (A × (D + B) + CIN) |
| C PreSubtract-Multiply | 0/1 | 1 | 0/1 | 1 | 1 | 1 | 0 | 1 | 0/1 | C ± (A × (D – B) + CIN) |
| 48-Bit Adder with PreAdd | 0/1 | 0/1 | 0/1 | 1 | 1 | 1 | 1 | 1 | 0/1 | C ± (D:A:(D ± B) + CIN) |

**Notes:**
1. For the remaining functions, CIN = the CIN output from the upstream DSP48A1 block or OPMODE[5] depending upon programming of CARRY SEL shown in this section.
2. Output mux must be selecting the register output.
3. Added for all 1s case.

**Figura 7-3: Resumen de todas las posibles funciones aritméticas que puede sintetizar un DSP48A1.**

## 7.2 Entorno de desarrollo y lenguaje utilizado

Para la descripción del hardware se utilizo el lenguaje VHDL versión 200X. Trabajamos con el entorno de desarrollo ISE V12.4 (nt) de Xilinx que utiliza XST como tecnología de síntesis.

La licencia del programa fue provista por el Laboratorio de Computación Reconfigurable de la UTN-FRM.

Para efectuar las simulaciones en todas las instancias se utilizó el simulador de Xilinx, Isim V12.4 (nt).

## 7.3 Librerías de terceros utilizadas

Las librerías VHDL de terceros que utilizamos son, las recomendadas por la IEEE (contenidas en la librería IEEE [**14**]) y la UNSIM desarrollada por Xilinx para poder simular módulos instaneados como black box (contenidas en la librería UNSIM), ambas vienen por defecto con ISE. De las dos nombradas utilizamos en particular:

- IEEE.STD_LOGIC_1164
- IEEE.NUMERIC_STD
- UNISIM.VCOMPONENTS

## 7.4 Packages desarrollados

Las funciones, parámetros y variables necesarias para la implementación del KF-BT-FP se distribuyeron en tres packages que nombramos aquí y se detallan en las siguientes secciones:

1. Fixed_Point_Pkg.vhd: Que implementa las cuatro operaciones matemáticas fundamentales (suma, complemento, producto y reciproco) en la aritmética de FP especificada. Se halla en el archivo del anexo 11.2.1

2. Parameters.vhd. En el que se especifican los parámetros de la planta que se va a utilizar, la aritmética de FP y se inicializan las variables correspondientes. Se halla en el archivo del anexo 11.2.2

3. System_def.vhd: En el que se inicializan las matrices del problema como la planta a utilizar, los ruidos, etc. Se halla en el archivo del anexo 11.2.3

### 7.4.1 Parameters.vhd

Aquí se definen automáticamente todas las variables que van a intervenir en el diseño. Para esto es necesario establecer manualmente las siguientes constantes:

- Aritmética de FP:
  - WORD_LENGTH: Longitud de la palabra de trabajo (WL)
  - FRACTION_LENGTH: Longitud de la parte fraccional (FL)
- Planta y sensores:
  - n: Dimensión del vector de estados interno.
  - l: Dimensión del vector de mediciones.
  - p: Dimensión del vector de entrada.

Los tipos y subtipos de variables básicos con los que se operará son:

Palabras (vectores de bits):

1. STD_LOGIC_VECTOR: (para comunicar entre módulos)
    a. WORD: Palabra de trabajo de ongitud WORD_LENGTH.
    b. WORD_x2: Palabra de longitud 2*WORD_LENGTH
2. SIGNED: (para operar internamente)
    a. S_WORD: Palabra de trabajo.
    b. S_WORD_p1: Palabra de longitud WORD_LENGTH +1 para almacenar el resultado de la suma.
    c. S_WPRD_x2: Palabra de longitud 2* WORD_LENGTH para almacenar el resultado de la multiplicación.

Vectores (vectores de vectores de bits):

3. Vectores de STD_LOGIC_VECTOR: (para comunicar entre módulos)
    a. WORDV_n: Vector $n$-dimensional de WORD
    b. WORDV_l: Vector$l$-dimensional de WORD
    c. WORDV_UT: Vector de WORD capaz de almacenar todos los elementos (del triangulo superior sin contar la diagonal principal) de una matriz triangular superior unitaria de dimensión $n$.
    Puede demostrarse que para almacenar los elementos de la matriz:

$$\begin{bmatrix} 1 & a_{12} & \cdots & a_{1(n-1)} & a_{1n} \\ 0 & 1 & \cdots & a_{2(n-1)} & a_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & a_{(n-1)n} \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix}$$

En un vector de dimensión $r$:

$$WORDV\_UT = [v_1 \quad \dots \quad v_r]$$

Con el siguiente orden:

$$WORDV\_UT = [a_{12} \quad \dots \quad a_{1n} \quad a_{23} \quad \dots \quad a_{2n} \quad \dots \quad a_{(n-1)n}]$$

La dimensión de este debe valer:

$$r = \frac{n^2 - n}{2}$$

Y para acceder al elemento $a_{ij}$ de la matriz dentro del vector debe utilizarse la siguiente transformación:

$$v_s = a_{ij}$$

Donde el índice es:

$$s = j - i + \sum_{k=1}^{i-1}(n-k)$$

$$s = j - \frac{i}{2}(1 + i - 2n) - n$$

4. Vectores de SIGNED: (para operar internamente)
    a. S_WORDV_n: Vector $n$-dimensional de S_WORD
    b. S_WORDV_l: Vector$l$-dimensional de S_WORD
    c. S_WORDV_UT: Vector de S_WORD capaz de almacenar todos los elementos (del triangulo superior sin contar la diagonal principal) de una matriz triangular superior unitaria de dimensión $n$.

Matrices (arreglos de vectores de bits):

5. Matrices de STD_LOGIC_VECTORS: (para comunicar entre módulos)
    a. WORDM_nxn: Matriz cuadrada de WORD con dimensión $n$.
6. Matrices de SIGNED: (para operar internamente)
    a. S_WORDM_nxn: Matriz cuadrada de S_WORD con dimensión $n$.
    b. S_WORDM_lxn: Matriz de S_WORD con dimensiones $lxn$.
    c. S_WORDM_nxp: Matriz de S_WORD con dimensiones $nxp$.

### 7.4.2 System_def.vhd

La interpretación de este archivo es bastante directa. Aquí se definen las constantes que establecen los valores de los siguientes parámetros invariantes en el tiempo:

1. De la planta:
    a. $\Phi$
    b. $\acute{G}$
    c. $D_Q$
    d. $u$
2. Los sensores:
    a. $H$
    b. R
3. El filtro (Condiciones Iniciales):
    a. $x_0$
    b. $U_0$
    c. $D_0$

Para cargar las constantes debe cuidarse de utilizar los números adecuadamente interpretados , según la aritmética que se especificó en el archivo Parameters.vhd.

### 7.4.3 Fixed_Point_Pkg.vhd

En este package se describe todo el hardware necesario para poder implementar una aritmética de FP teniendo en cuenta los puntos cubiertos en la secciónes 6.1 y 6.2. Para resolver este problema hay dos enfoques básicos.

- Desarrollar módulos específicos que implementen cada operación en FP. Esto requiere luego una descripción estructural del KF que especifique la correcta interconexión de todos estos bloques para que se efectúen las operaciones de cada modulo de mayor jerarquía. La principal desventaja de esta opción es el tiempo de desarrollo y la complejidad de la descripción final. La mayor ventaja es que permite tener un mayor control sobre el hardware sintetizado.

- La otra posibilidad (adoptada aquí) es la de codificar cada operación en una función que luego sea utilizada en el resto de los módulos. La mayor ventaja de esta opción es el tiempo reducido de desarrollo y la simpleza de la descripción elaborada. Sin embargo esta metodología deriva en un menor control del hardware sintetizado y en la elevada exigencia para XST de sintetizar bloques asíncronos de elevada complejidad (ver sección 7.6).

Funciones implementadas:

resize_p1

Localización

Esta función se encuentra codificada entre las líneas 84 y 100 del archivo Fixed_Point_Pkg.vhd.

Entradas

- Operando tipo S_WORD_p1: $O_{p1}(IL_{p1}, FL_{p1}) = O_{p1}(IL + 1, FL)$

Salidas

- Resultado tipo S_WORD: $O(IL, FL)$

Funcionamiento

Esta función se utiliza para readecuar el resultado de una suma de nuevo a la aritmética de trabajo utilizando Saturate en caso de que se presente Overflow (ver la secciónes 6.1.2 y 6.1.3).

Por lo tanto, esta rutina ejecutará las operaciones:

Si no hay Overflow:

$$O_{p1}(IL_{p1}, FL_{p1}) \rightarrow O(IL, FL)$$

$$BS = BS_{p1}$$

$$C(IL) = C(IL_{p1})_{IL}$$

$$C(FL) = C(FL_{p1})$$

Si hay Overflow:

Si $BS_{p1} = 0$:

$$O_{p1}(IL_{p1}, FL_{p1}) \rightarrow O(IL, FL)$$

$$BS = 0$$

$$C(IL) = 1 \dots 1$$

$$C(FL) = 1 \dots 1$$

*Si* $BS_{p1} = 1$:

$$O_{p1}(IL_{p1}, FL_{p1}) \rightarrow O(IL, FL)$$

$$BS = 1$$

$$C(IL) = 0 \dots 0$$

$$C(FL) = 0 \dots 0$$

Para *verificar que no hay Overflow* al tratar de readecuar una palabra cualquiera más larga que la de trabajo $O_A(IL + A, FL)$, a la de trabajo $O(IL, FL)$, solo debe comprobarse que todos los $A$ bits extras tengan el mismo valor que el bit de signo de $O_A$. En nuestra notación:

$$Si\ C(A) = BS_A, BS_A, \dots, BS_A \rightarrow NO\ hay\ Overflow$$

resize_x2

Localización

Esta función se encuentra codificada entre las líneas 102 y 127 del archivo Fixed_Point_Pkg.vhd.

Entradas

- Operando tipo S_WORD_x2: $O_{x2}(IL_{x2}, FL_{x2}) = O_{x2}(2IL + 1,2FL)$

Salidas

- Resultado tipo S_WORD: $O(IL, FL)$

Funcionamiento

Esta función se utiliza para readecuar el resultado de una multiplicación de nuevo a la aritmética de trabajo utilizando Saturate en caso de que se presente Overflow y Floor para efectuar el redondeo (ver secciónes 6.1.2 y 6.1.3).

Por lo tanto, esta rutina ejecutará las operaciones:

Si no hay Overflow:

$$O_{x2}(IL_{x2}, FL_{x2}) \rightarrow O(IL, FL)$$

$$BS = BS_{p1}$$

$$C(IL) = C(IL_{x2})_{IL}$$

$$C(FL) = C(FL_{x2})^{FL}$$

Si hay Overflow:

Si $BS_{x2} = 0$:

$$O_{x2}(IL_{x2}, FL_{x2}) \rightarrow O(IL, FL)$$

$$BS = 0$$

$$C(IL) = 1 \dots 1$$

$$C(FL) = 1 \dots 1$$

*Si* $BS_{x2} = 1$:

$$O_{x2}(IL_{x2}, FL_{x2}) \rightarrow O(IL, FL)$$

$$BS = 1$$

$$C(IL) = 0 \dots 0$$

$$C(FL) = 0 \dots 0$$

*No se realizó una función específica para realizar la multiplicación* porque esta viene implementada adecuadamente en el operador "*" de la librería aritmética de la IEEE. O sea que para multiplicar dos números $A$ *y* $B$ correctamente solo debe hacerse (ver sección 6.2.3):

$$resize\_x2[O_A(IL, FL) * O_B(IL, FL)]$$

fp_add

Localización

Esta función se encuentra codificada entre las líneas 127 y 137 del archivo Fixed_Point_Pkg.vhd.

Entradas

- Operando tipo S_WORD:$O_A(IL_A, FL_A) = O_A(IL, FL)$
- Operando tipo S_WORD: $O_B(IL_B, FL_B) = O_B(IL, FL)$

Salidas

- Resultado de la suma tipo S_WORD: $O(IL, FL)$

Funcionamiento

Esta función efectúa la suma $O_A(IL, FL) + O_B(IL, FL)$ y devuelve el resultado readecuado a la aritmética de trabajo $O(IL, FL)$.

Es importante destacar que el operador "+" implementado en las librería aritmética de la IEEE *no tiene en cuenta el Overflow*, o sea que si sumamos los operandos sin ninguna modificación tendríamos:

$$O_A(IL, FL) + O_B(IL, FL) = O(IL, FL)$$

Lo cual es erróneo pues si ocurre un Overflow no es detectado (ver sección 6.2.1). Por esto se hace un padding de 1 bit a la izquierda de cada operando (se añaden ceros no significativos) para obtener el resultado con toda la exactitud posible:

$$BS_A \& O_A(IL, FL) + BS_B \& O_B(IL, FL) = O(1 + IL, FL)$$

Luego se utiliza la función resize_p1 para readecuar el resultado a la aritmética de trabajo (y así en caso de Overflow se saturará el resultado):

$$O(1 + IL, FL) \rightarrow O(IL, FL)$$

fp_comp

Localización

Esta función se encuentra codificada entre las líneas 137 y 148 del archivo Fixed_Point_Pkg.vhd.

Entradas

- Operando tipo S_WORD:$O_A(IL_A, FL_A) = O_A(IL, FL)$

Salidas

- Resultado del complemento tipo S_WORD: $O(IL, FL)$

Funcionamiento

Esta función devuelve el complemento a 2 del operando de entrada readecuado a la aritmética de trabajo.

Como se explicó en la sección 6.2.2 si se intenta complementar el valor mínimo de la aritmética se produce un Overflow por ser $|\min[D(IL, FL)]| > |\max[D(IL, FL)]|$.

Es importante destacar que el operador "-" implementado en la librería aritmética de la IEEE no contempla este caso pues hace:

$$-O_A(IL, FL) = O(IL, FL)$$

Que es erróneo por qué no puede detectarse si se produce el mencionado Overflow. Debido a esto es necesario hacer un padding a la izquierda del número a complementar para no perder exactitud:

$$-BS_A\&O_A(IL, FL) = O(IL + 1, FL)$$

Luego se utiliza la función resize_p1 para readecuar el resultado a la aritmética de trabajo (y así en caso de Overflow se saturará el resultado):

$$O(1 + IL, FL) \rightarrow O(IL, FL)$$

fp_reciprocal

Localización

Esta función se encuentra codificada entre las líneas 148 y 214 del archivo Fixed_Point_Pkg.vhd.

Entradas

- Operando tipo S_WORD: $O(IL, FL)$

Salidas

- Resultado tipo S_WORD: $O(IL, FL)$

Funcionamiento

El cociente entre dos números en punto fijo se puede ejecutar siempre como un cociente entre enteros con tal de tener en cuenta la posición del punto de cada uno de los operandos.

Para la realización de esta función se aplica la filosofía algorítmica de restas sucesivas del divisor al dividendo.

Se toma el ejemplo siguiente en el que se detalla la filosofía de manera más sencilla. Si se quiere realizar la siguiente división:

$$7.8/3.6$$

Con esta división y haciendo referencia a la Ecuación 6-22 se obtiene que el resultado de la misma debe tener 4 dígitos.

$$? \quad ? \quad ? \quad ?$$

Se procede expresando los números en cifras enteras equivalentes, con lo cual se tiene:

$$78/36$$

Luego se toma el dividendo, de un dígito a la vez introduciéndolo en un registro de igual número de dígitos que el resultado y se lo compara con el divisor:

$$0 \quad 0 \quad 0 \quad 7 < 36$$

Como en este caso $7 < 36$ no se puede restar $36$ de $7$ con resultado positivo. Esta situación secuela en el primer dígito de información del resultado de la división: $0$.

$$0 \quad ? \quad ? \quad ?$$

Se toma el siguiente dígito del dividendo y se compara con el divisor:

$$0 \quad 0 \quad 7 \quad 8 > 36$$

Esta comparación lleva a que se posibiliten las dos restas siguientes:

$$0 \quad 0 \quad 7 \quad 8 - 36 = 0 \quad 0 \quad 4 \quad 2,$$

$$0 \quad 0 \quad 4 \quad 2 - 36 = 0 \quad 0 \quad 0 \quad 6$$

Con lo cual, el segundo dígito de información en el cociente es: $2$.

$$0 \quad 2 \quad ? \quad ?$$

Con esto se ha obtenido la parte entera de la división, lo cual se logra enfrentando los puntos fraccionales de ambos operandos, o sea, el simple hecho de obtener el número de veces que entra el divisor en el dividendo.

Se desplaza hacia la izquierda el resultado de la última resta y se introduce el próximo dígito del dividendo (el cual es $0$). Luego se realiza la comparación:

$$0 \quad 0 \quad 6 \quad 0 > 36$$

La cual da como resultado la realización de la siguiente resta:

$$0 \quad 0 \quad 6 \quad 0 - 36 = 0 \quad 0 \quad 2 \quad 4$$

Y el cociente se establece en:

$$0 \quad 2 \quad 1 \quad ?$$

Se realiza un nuevo desplazamiento en este último resultado y se introduce el próximo dígito del dividendo (el cual es $0$). Luego se realiza la comparación:

$$0 \quad 2 \quad 4 \quad 0 > 36$$

La cual da lugar a las siguientes 6 restas:

$$0 \quad 2 \quad 4 \quad 0 - 36 = 204 - 36 = 168 - 36 = 132 - 36 = 96 - 39 = 60 - 36 = 0 \quad 0 \quad 2 \quad 4$$

Estas 6 restas dejan el siguiente cociente:

0 2 1 6

Lo cual arroja el resultado que se esperaba: 2.16.

Se pueden seguir realizando desplazamientos en los resultados de las restas dependiendo del número de dígitos significativos que se necesiten.

En el algoritmo de implementación se declaran las siguientes variables:

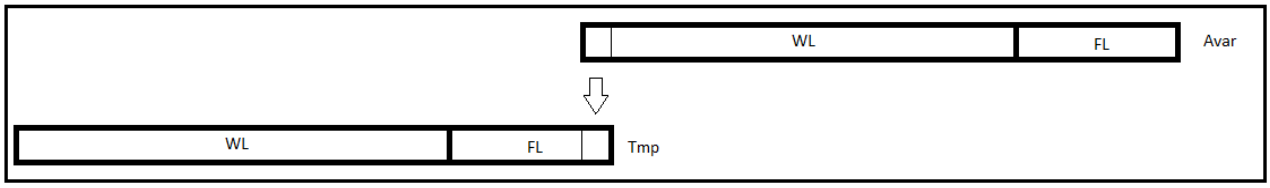
$$Avar, Tmp, Tmp3 \; y \; Quot = O_{pFL}(IL, FL_{pFL}) = O_{pFL}(IL, 2FL)$$

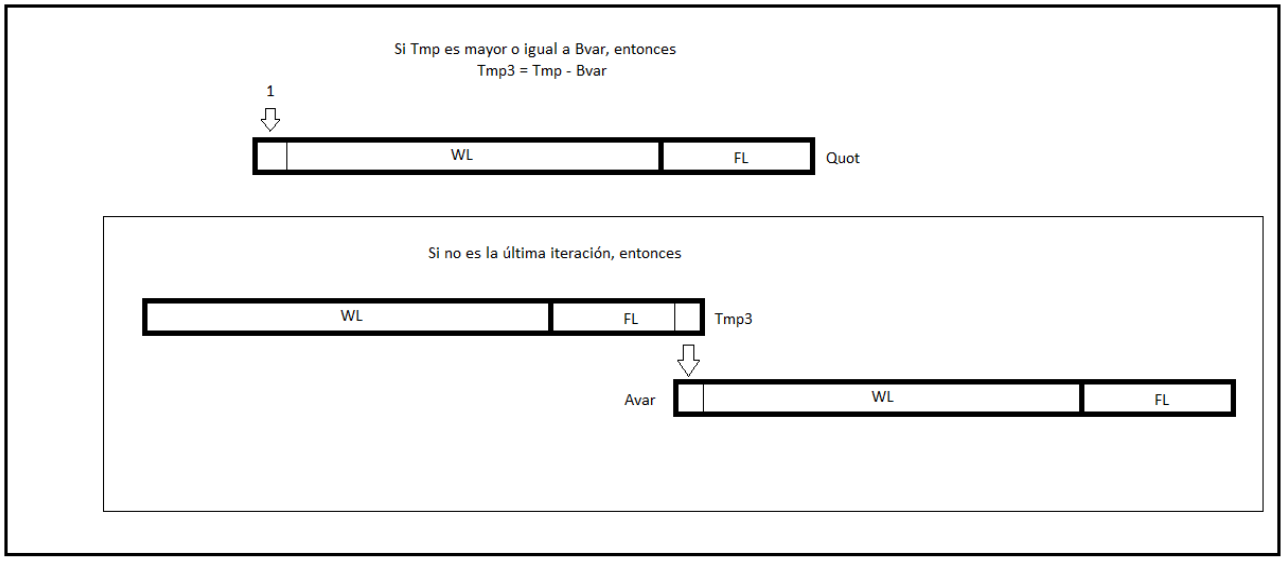
$$Result \; y \; Bvar = O(IL, FL)$$

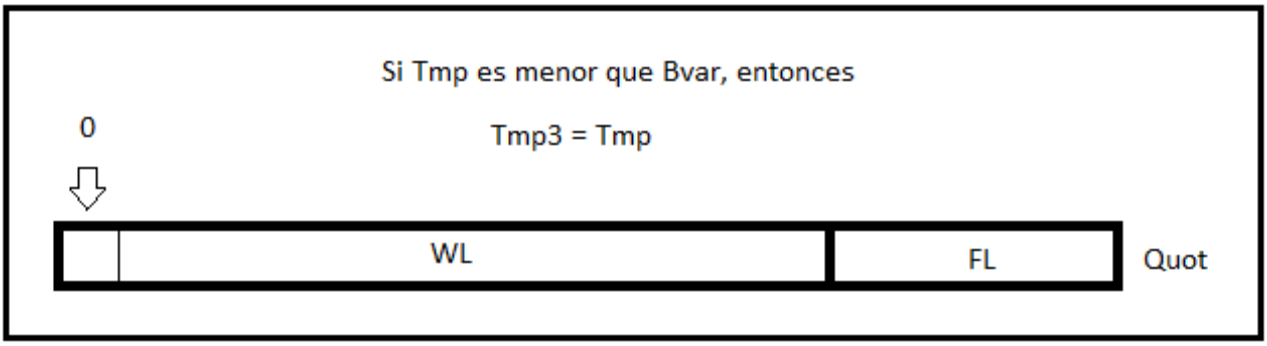
El dividendo es una constante de valor unitario que se guarda en $Avar$.

$$Avar = 1,$$

Luego se entra en el siguiente ciclo, en el cual se realizan las operaciones ilustradas:



En la siguiente iteración:

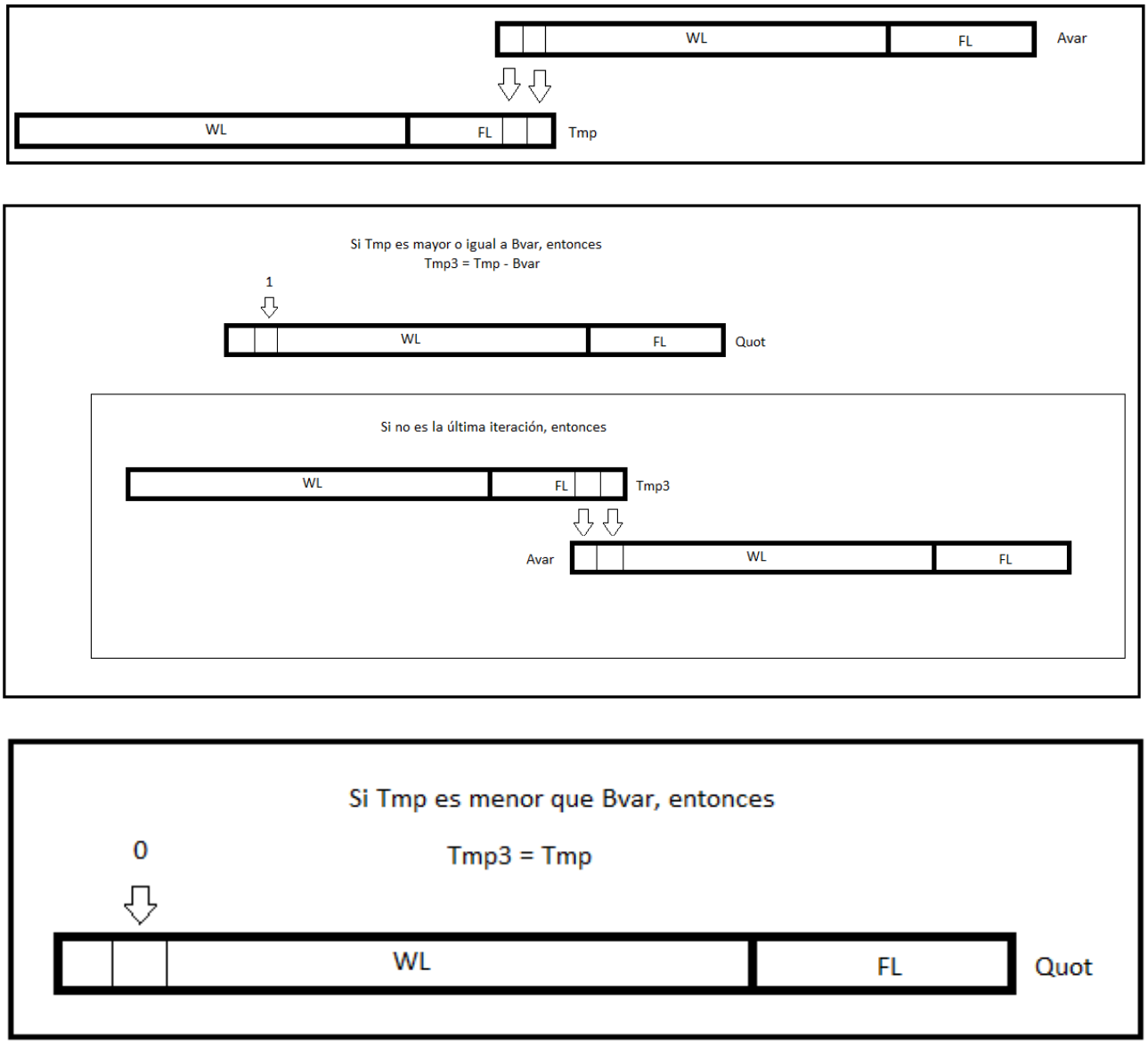



**Figura 7-4: Algoritmo de división entre números positivos, a través de restas sucesivas.**

Y así sigue hasta finalizar con el llenado de la variable $Quot$, tras $WL + FL$ iteraciones.

El algoritmo mostrado en la Figura 7-4 detalla el procedimiento en la obtención del cociente entre números positivos este es idéntico al de división presentado en [**10**] y [**11**] salvo algunas modificaciones.

A continuación se detallan las características agregadas a dicho algoritmo para adaptarlo a las necesidades del presente trabajo:

- Teniendo en cuenta lo desarrollado en la sección 6.2.4 y haciendo referencia a la Ecuación 6-22, que se repite aquí por conveniencia:

$$\frac{O(\mathrm{IL},\mathrm{FL})}{O(\mathrm{IL},\mathrm{FL})} = O_{\mathrm{res}}(\mathrm{IL}+\mathrm{FL}+1,\mathrm{IL}+\mathrm{FL})$$

Se aprecia que en la parte entera del resultado de la división se disponen de $IL + FL + 1\,bits$ mientras que en su parte fraccional se disponen de $IL + FL\,bits$. Este resultado se debe asignar en una palabra de $IL + FL\,bits$, con lo cual se debe realizar una reasignación entre las partes enteras de $IL + FL + 1 \rightarrow IL\,bits$ y entre las partes fraccionales de $IL + FL \rightarrow FL\,bits$.

La información contenida en la parte entera de $O_{res}$ a reasignar en la parte entera de $O$ es imprescindible para la consulta y tratamiento del desbordamiento. Se debe tener en cuenta que para este caso el dividendo debe ser almacenado en una palabra de $IL + FL + 1\,bits$ de parte entera, lo cual conlleva a una cantidad de $IL + FL + 1\,iteraciones$ en el algoritmo para poder acceder a toda la información disponible en la porción entera del resultado de la división, la cual, como se acaba de expresar, es indispensable para la consulta y el tratamiento de posibles desbordamientos. Esto último da para pensar que el hecho de haber escogido un método de tratamiento del Overflow como (saturate) nos arrastra a esta restricción pero, por el contrario aún escogiendo el método (wrap) lleva a la misma limitación, ya que aunque se realice el truncamiento de los bits superiores para la reasignación $IL + FL + 1 \rightarrow IL$ es necesaria la ejecución de $IL + FL + 1$ iteraciones en el algoritmo para llegar a la obtención de la información no truncada contenida en la parte entera del resultado de la división. Esto último es propio del algoritmo de división elegido para la implementación, en el cual la obtención de los bits menos significativos dependen de la ejecución para el cómputo de los bits más significativos. Se concluye con esto que para la implementación del algoritmo de división, no influye la elección del tratamiento del overflow en cuanto a la complejidad final del algoritmo.

Por el contrario se puede afirmar que la elección del método de redondeo si influye con las posibilidades de disminuir la complejidad final en la implementación del algoritmo de división. Debido a la reasignación de la parte fraccional ($IL + FL \rightarrow FL\,bits$), no es necesaria la realización de $IL + FL$ iteraciones adicionales luego de la obtención de la parte entera. Los bits obtenidos en cada iteración subsecuente en el algoritmo dependen de las iteraciones anteriores y no a la inversa, por lo tanto y según el método de redondeo elegido (floor) no se hace necesario el cómputo de dichos bits y se procede a la realización de sólo $FL$ iteraciones para la obtención de toda la información necesaria contenida en la parte fraccional del resultado a reasignar. Se disminuye la complejidad por el hecho de haber realizado $IL$ iteraciones menos las cuales tienen el reflejo de una implementación combinacional directa en el hardware.

- Teniendo en cuenta el dispositivo del que se dispone para la implementación de los algoritmos del KF-BT, se aprecia la conveniencia en la utilización de los multiplicadores embebidos que se detallan en la sección 7.1.1. Esto último sumado a la reducción en la complejidad del algoritmo de división que se consigue programándolo como una función que devuelve el recíproco de un número, nos inclina por dicha opción. Por lo tanto, en vez de computar una división se computa el reciproco del denominador. Luego deberá computarse aparte la multiplicación por el numerador: $\frac{n}{d} = n * \frac{1}{d}$. Esto hace que el tiempo total de cómputo sea menor (y presumiblemente también el consumo) ya que para efectuar la multiplicación se cuenta en la FPGA con recursos de hardware embebidos específicos para tal fin.

- Se debe tener en cuenta que el algoritmo básico ilustrado en la Figura 7-4 realiza sólo divisiones entre números positivos. Otra modificación incluida al mismo fue la de agregar la capacidad del tratamiento de números negativos en complemento a 2.

- El algoritmo original arroja el cociente en $Quot$ y el resto en $Tmp3$. En nuestro caso no se necesita la disponibilidad del resto.

- Se adaptó el algoritmo original para posibilitar la utilización de las librerías recomendadas por la IEEE. Para lo cual se realizaron modificaciones pertinentes en el código. En el anexo 11.2.1 se encuentra el código en VHDL para la implementación de la aritmética de punto fijo, mientras que se puede hacer referencia a [**11**] para su comparación.

- Se modificó el algoritmo para que se produjera la saturación en caso de darse la necesidad tener que computar el recíproco de cero.

- Se incluyó una rutina que implementa la función resize de la siguiente manera, dadas las necesidades patentes en el tipo de variable de salida de la función reciprocal:

  La serie de instrucciones a las que se hace referencia sirven para readecuar el resultado de la función reciprocal de nuevo a la aritmética de trabajo utilizando Saturate en caso de que se presente Overflow y Floor para efectuar el redondeo (ver sección 6.2.3).

  Por lo tanto, esta rutina ejecutará las operaciónes:

  Si no hay Overflow:

$$O_{pFL}(IL_{pFL}, FL_{pFL}) = O_{pFL}(IL, 2FL) \rightarrow O(IL, FL)$$

$$BS = BS_{pFL}$$

$$C(IL) = C\left(IL_{pFL}\right)_{IL}$$

$$C(FL) = C\left(FL_{pFL}\right)^{FL}$$

Si hay Overflow:

Si $BS_{pFL} = 0$:

$$O_{pFL}(IL_{pFL}, FL_{pFL}) \rightarrow O(IL, FL)$$

$$BS = 0$$

$$C(IL) = 1 \dots 1$$

$$C(FL) = 1 \dots 1$$

*Si* $BS_{pFL} = 1$:

$$O_{pFL}(IL_{pFL}, FL_{pFL}) \rightarrow O(IL, FL)$$

$$BS = 1$$

$$C(IL) = 0 \dots 0$$

$$C(FL) = 0 \dots 0$$

## 7.5 Módulos de Hardware desarrollados

### 7.5.1 KF_BT

Entradas

- Z Entrada de tipo WORD
- clk Entrada de tipo STD_LOGIC
- Reset Entrada de tipo STD_LOGIC
- CE Entrada de tipo STD_LOGIC

Salidas

- E_X_1 Salida de tipo WORD
- E_X_2 Salida de tipo WORD

Funcionamiento

La estructura interna de nuestra implementación del KF-BT consta de una organización formada por dos módulos combinacionales y uno secuencial síncrono activado por flancos de subida, por lo tanto el KF-BT queda constituido en un sistema secuencial síncrono activado por flancos de subida.

La ruta de datos viene a cumplir la función del control de realimentación presente en los flip-flops secuenciales síncronos activados por flacos más algunas características funcionales propias del multiplexado de los datos, ya que este sistema de control está compuesto por dos circuitos combinacionales.

La implementación estructural del módulo que constituye el KF-BT es totalmente escalable a través de la modificación de los parámetros $n, l, p, (WL) y\ (FL)$ definidos en la sección 7.4.1 excepto por el hecho de tener que modificar el código para la implementación de las salidas escalares necesarias $(E\_X\_n)$. Esta última limitación se produce por problemas de compatibilidad presentados por el software de simulación, al no aceptar este último, trabajar con variables que constituyen vectores de STD_LOGIC_VECTOR. La Figura 7-5 ilustra la estructura del módulo.

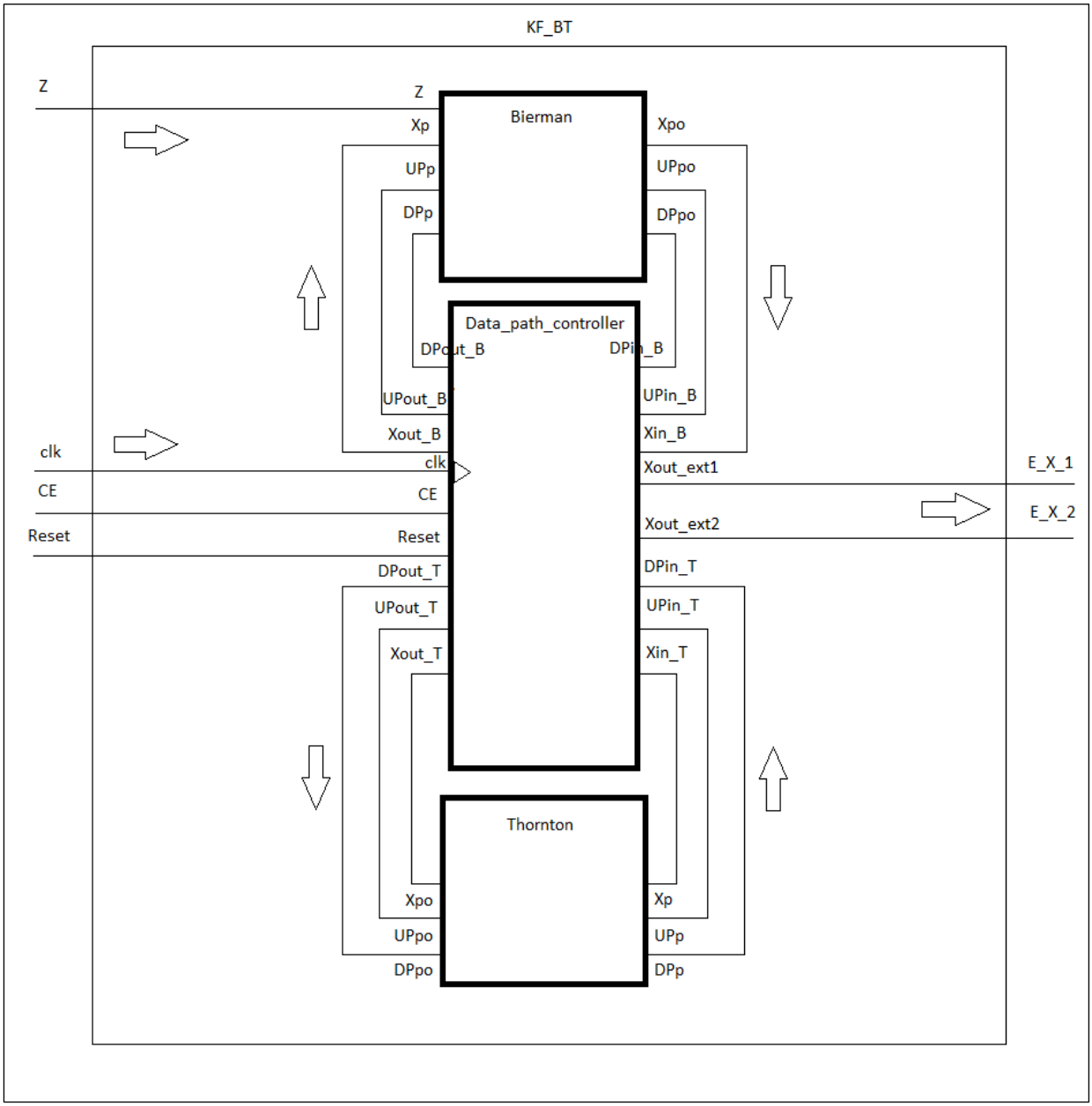



**Figura 7-5: Estructura interna del KF-BT.**

Especificación temporal

La especificación temporal del módulo se ilustra en la Figura 7-6, esta requiere que previo a la finalización en el reseteo $(Reset = 1)$ deba estar estabilizada la primer componente escalar del vector de medición $Z$. Una vez estabilizadas las salidas del módulo Data_path_controller correspondientes a las condiciones iniciales del KF-BT, el módulo combinacional Bierman tomará un ciclo de reloj para propagar su salida y presentarla estable.

Cuando se haya deshabilitado el reset $(Reset = 0)$, la entrada correspondiente a la primera componente del vector de mediciones $Z$ deben presentarse estable y se puede habilitar el funcionamiento del KF-BT mediante la mantención de la señal $CE = 1$. Se debe también establecer la siguiente componente escalar del vector de medición $Z$ para cada flanco de subida en la señal $clk$ luego de que se estableció la señal $CE = 1$. Una vez introducidas las $l$ componentes del vector de medición $Z$, en el siguiente flanco de subida no se exije la mantención de una componente en $Z$ determinada, un ciclo posterior de la señal de $clk$ se debe presentar en la entrada del KF-BT la primer componente del vector de medición $Z$ siguiente. Luego de esto, el ciclo se repite, como se muestra en la Figura 7-6.

Las salidas del KF-BT estarán disponibles a partir del flanco de subida en la señal $clk$ inmediatamente consecutivo a la introducción de la última componente del vector de medición $Z$. Estas salidas corresponden a la estimación a posteriori del vector de estados.

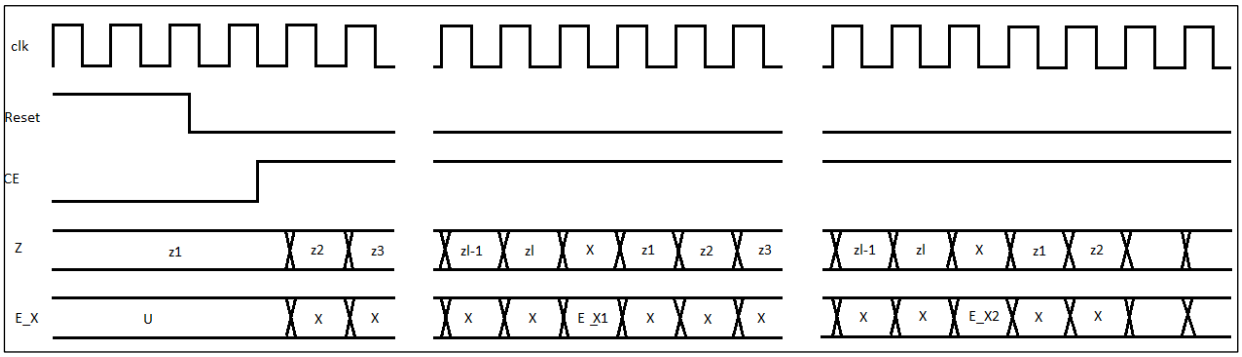



**Figura 7-6: Diagrama de especificaciones temporales del módulo KF-BT.**

### 7.5.2 Bierman_module

Entradas

- Z Entrada de tipo WORD
- Xp Entrada de tipo WORDV_n
- UPp Entrada de tipo WORDV_UT
- DPp Entrada de tipo WORDV_n

Salidas

- Xpo Salida de tipo WORDV_n
- UPpo Salida de tipo WORDV_UT
- DPpo Salida de tipo WORDV_n

Funcionamiento

Como se muestra en la Figura 7-5, este es un módulo que ejecuta el algoritmo de actualización observacional de Bierman (sección 5.4.3) de manera combinacional a través de las operaciones aritméticas implementadas.

La característica que atañe a dichas operaciones aritméticas (Sección 7.4.3) es la de que son funciones implementables combinacionalmente.

Si bien funciones como fp_add tienen la realimentación que constituye el acarreo (lo cual daría a las mismas aparentes características de secuencialidad) se puede argumentar que, independientemente del instante en el que se presente la entrada estable, la salida se estabilizará, en el valor correcto, cuando el acarreo finalice su propagación en toda la función. Es así que la salida depende sólo del valor de sus entradas y no del instante en el que se presentan las mismas.

Dicho aquello, el sintetizador utilizado (Sección 7.2) permite la implementación de estas funciones de manera combinacional y será este, el encargado de cumplir las exigencias antes descriptas con tal de conseguir un funcionamiento coherente del módulo con la especificación del algoritmo.

### 7.5.3 Thornton_module

Entradas

- Xpo Entrada de tipo WORDV_n
- UPpo Entrada de tipo WORDV_UT
- DPpo Entrada de tipo WORDV_n

Salidas

- Xp Salida de tipo WORDV_n
- UPp Salida de tipo WORDV_UT
- DPp Salida de tipo WORDV_n

Funcionamiento

Como muestra la Figura 7-5, este es un módulo que implementa el algoritmo de actualización temporal de Thornton (sección 5.4.3) de manera combinacional a través de las operaciones aritméticas utilizadas.

Además de lo ya explicado en el apartado anterior en cuanto a la implementación combinacional de las operaciones aritméticas fundamentales, se puede agregar que para el algoritmo de Thornton es necesaria la implementación de la ortogonalización de Gram Schmidt sopesada la cual consta de un algoritmo en el que no se pueden explotar las características paralelas del dispositivo de lógica reconfigurable debido a que se presenta secuencialidad en el mismo como bien se expone en el anexo 11.4.1 y en la sección 5.4. Sin embargo la herramienta de síntesis permite la implementación de dicha ortogonalización de

manera combinacional debido a que independientemente del momento en el que se presenten sus entradas estables, el algoritmo confluirá a un estado estable determinado por su especificación.

### 7.5.4 Data_path_controller

Entradas

- Xin_B Entrada de tipo WORDV_n
- UPin_B Entrada de tipo WORDV_UT
- DPin_B Entrada de tipo WORDV_n
- Xin_T Entrada de tipo WORDV_n
- UPin_T Entrada de tipo WORDV_UT
- DPin_T Entrada de tipo WORDV_n
- clk Entrada de tipo STD_LOGIC
- CE Entrada de tipo STD_LOGIC
- Reset Entrada de tipo STD_LOGIC

Salidas

- Xout_B Salida de tipo WORDV_n
- UPout_B Salida de tipo WORDV_UT
- DPout_B Salida de tipo WORDV_n
- Xout_T Salida de tipo WORDV_n
- UPout_T Salida de tipo WORDV_UT
- DPout_T Salida de tipo WORDV_n
- Xout_ext1 Salida de tipo WORD
- Xout_ext2 Salida de tipo WORD

Funcionamiento

Como se muestra en la Figura 7-5, este es un módulo secuencial síncrono activado por flancos de subida.

Dicho módulo se encarga de controlar la ruta de datos en la realimentación del estado interno del KF-BT.

Este módulo cumple las funciones llevadas a cabo por los flip-flops síncronos en todo sistema secuencial síncrono de control con el agregado de funciones de multiplexado para controlar la ruta de los datos entre ambos módulos combinacionales.

El control de la ruta de datos deriva en el cómputo del algoritmo de Bierman a través de $l$ iteraciones para que luego se produzca la conmutación de dicha ruta y se realice el cómputo de Thornton, esto último deviene de la ejecución del algoritmo para el KF-BT procesando el vector de mediciones una componente a la vez como se explica oportunamente en la sección 5.3.5.

Para nuestro caso $l = 1$ y el control en la ruta de datos presenta una alternancia estricta. Para el caso en que se hagan necesarias implementaciones del KF-BT con $l > 1$ el código prevé instanciaciones condicionales de módulos alternativos para tales situaciones (apéndices 11.2.4, 11.2.8 y 11.2.9). Entre dichos módulos se genera una nueva línea que comunica la iteración $l$, desde el módulo Data_path_controller al módulo de Bierman, correspondiente previa a la ejecución del algoritmo de Thornton. Con esta información, el módulo de Bierman indexa correctamente la matriz $H$ para su ejecución correcta procesando el vector de mediciones una componente a la vez (Sección 5.3.5).

## 7.6 Indicaciones para la implementación de sistemas LTV

Coherentemente a los objetivos de este trabajo, se deja la especificación estructural inicial pensando en la implementación de un KF-BT-FP que realice la estimación del vector de estados de un sistema LTV, procesando el vector de mediciones una componente a la vez (Sección 5.3.5). Esta estructura se ilustra en la Figura 7-8.

Las entradas adicionales correspondientes son:

Para el módulo de Bierman

Como se explicó en la sección 5.3.5 se sabe que cuando se procesa el vector de mediciones una componente a la vez se requiere en cada iteración del módulo uno de los $l$ vectores fila de la matriz $H$, los cuales tienen dimensión $n$. En cada una de las $l$ iteraciones se necesitará también una de las componentes de la diagonal principal de la matriz $R$ de covarianza del ruido en la medición. Para el caso en que la matriz $R$ no sea una matriz diagonal, (ruido de medición correlacionado sección 5.3.6) el procedimiento de factorización de la misma se deberá llevar a cabo cada $l + 1$ iteraciones del módulo KF-BT, o sea, cuando se actualiza $R$.

- H Entrada de tipo WORDV_n
- R Entrada de tipo WORD

Tanto $H$ como $R$ deben ser cargadas con la misma especificación que $Z$.

Para el módulo de Thornton

Se debe tener en cuenta que en caso de que el ruido de la planta sea correlacionado, la matriz de covarianza de dicho ruido $Q$ no será diagonal y se deberá realizar su factorización $Q = UD_QU^T$, como así también el cómputo de $G' = GU$ cada $l + 1$ ciclos en el módulo KF-BT. (Sección 5.4.2)

- Φ Entrada de tipo WORDV_nxn
- G' Entrada de tipo WORDV_nxp
- $D_Q$ Entrada de tipo WORDV_p
- u Entrada de tipo WORDV_n

La especificación para la carga de dichas entradas establece que las mismas deben estar estabilizadas en el valor correcto previo al flanco de subida que sucede en el ciclo posterior al que se realiza la carga de la componente $l$ del vector de medición $Z$.

Tanto el vector Φ como el $G'$ son tratados como matrices en el interior del módulo combinacional que realiza el cómputo del algoritmo de Thornton. Es así necesaria la toma de algunas convenciones en cuanto a la carga de dichos vectores con el fin de no afectar la lógica del algoritmo.

Se tomará convencionalmente que los vectores provenientes desde el exterior serán cargados desde las matrices respectivas fila por fila de la siguiente manera:

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} = [a_{11} \quad \cdots \quad a_{1n} \quad a_{21} \quad \cdots \quad a_{2n} \quad \cdots \quad a_{n1} \quad \cdots \quad a_{nn}]$$

Por lo tanto, el subíndice $s$ del elemento del vector $Phi_{(nxn)}$ desde la matriz $Phi_{nxn}$ está dado por:

$$s = f(i,j) = (i-1)n + j$$

$$Phi_s = Phi_{ij}$$

Y el subíndice $s$ del elemento del vector $G_{(nxp)}$desde la matriz $G_{nxp}$ está dado por:

$$s = f(i,j) = (i-1)p + j$$

$$G_s = G_{ij}$$

Con el fin evitar la modificación del algoritmo de Thornton se debe proceder al agregado de las siguientes sentencias de las cuales resultarán los valores de los subíndices $i\ j$ desde el subíndice $s$. Se tomará como caso general el de la matriz $G_{nxp}$. El algoritmo a seguir se muestra en la Figura 7-7.

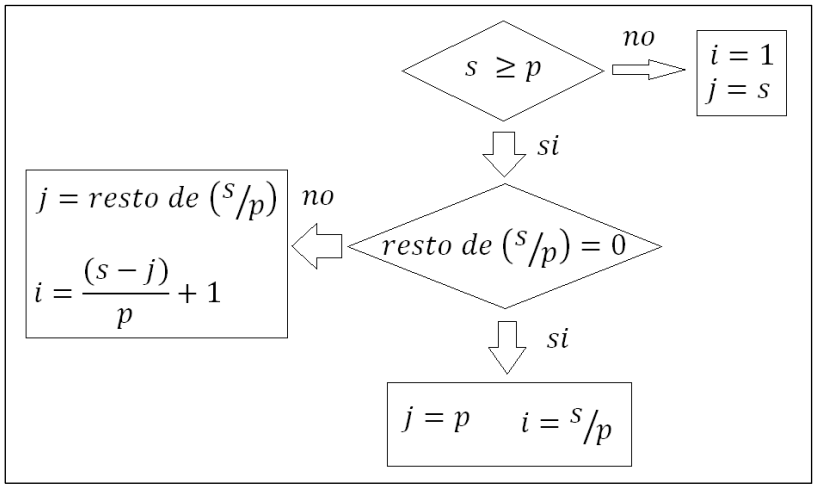



**Figura 7-7: Algoritmo de carga de vectores a matrices.**

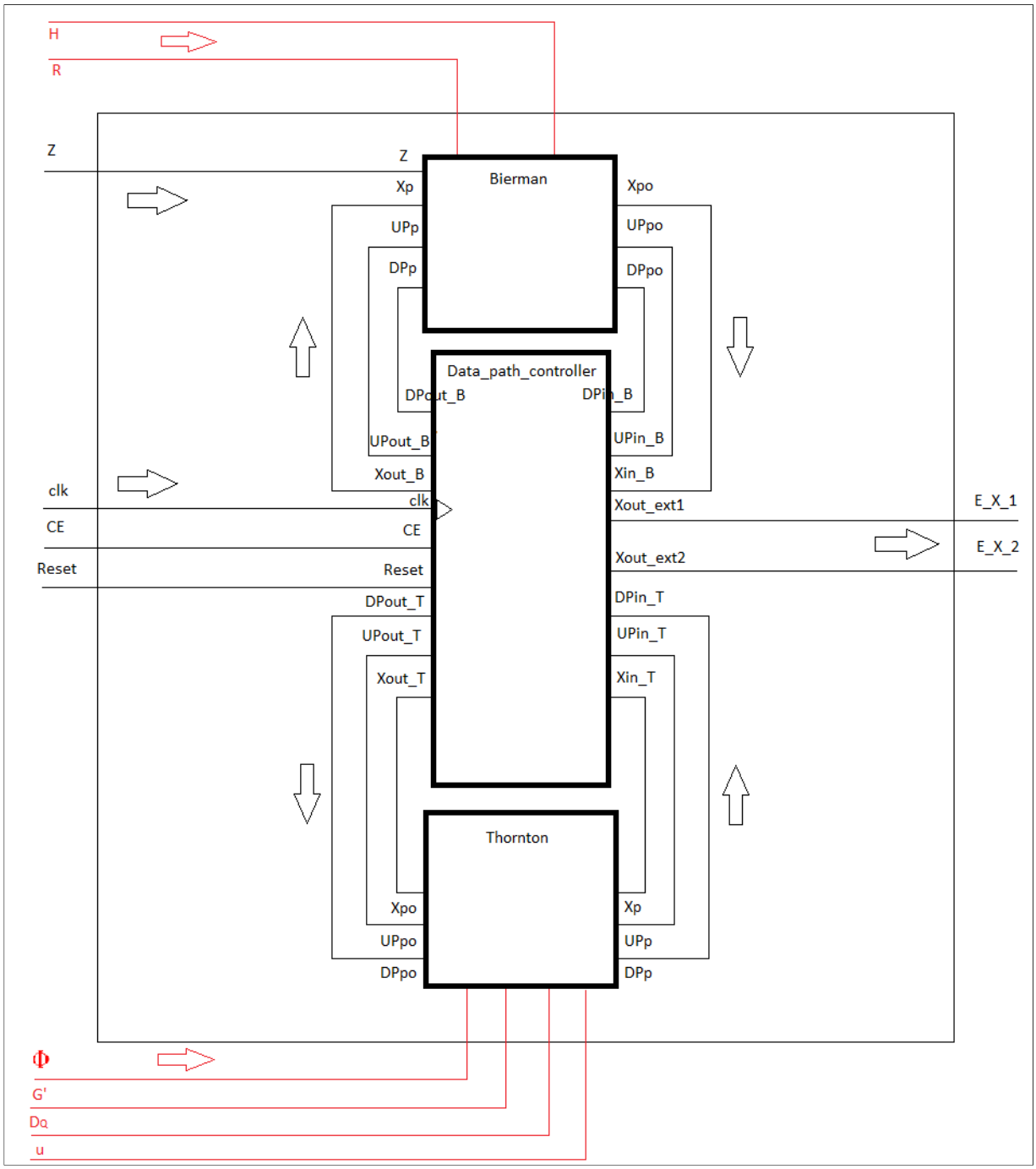



**Figura 7-8: Diagrama estructural de la implementación del KF-BT para el caso de una planta LTV.**

## 7.7 Comentario sobre problemas de síntesis en las familias Spartan 6 y Virtex 5

Como se indicase en la sección 7.4.3, la filosofía de descripción del hardware aquí adoptada (definición de las operaciones aritméticas en una librería en vez de implementarlas en módulos independientes) derivó en la síntesis de dos grandes bloques asíncronos (Bierman y Thornton).

Si bien la descripción resultante es sintética, organizada, de fácil lectura, requiere de un menor tiempo de desarrollo y está más acorde a los objetivos de este trabajo (pues facilita el diseño escalable y es más independiente de la aplicación), no resultó ser la más adecuada para

sintetizar el hardware descripto en la FPGA. Esto es debido a que no permite tener un control detallado sobre el hardware que se va sintetizar y a que las tecnologías de síntesis (basadas en algoritmos heurísticos) están orientadas a la optimización de sistemas síncronos, pues en estos, se cuenta con mayores restricciones que facilitan el análisis temporal y por ende la convergencia del algoritmo.

Esto provocó que, a la hora de sintetizar el KF-BT-FP descripto como en la sección 7.5 se presentasen problemas de funcionamiento. En particular sucedió que el hardware descripto funcionaba correctamente a nivel lógico (en la simulación Behavioral) pero luego de ser sintetizado, el funcionamiento de los bloques combinacionales de Bierman y Thorton era erróneo (manifestado en la simulación Post-Synthesis). Puesto que cuando se probaba el funcionamiento de los módulos por separado este era correcto y que el error surgió cuando se sintetizaban todos los bloques juntos, es probable que este problema se deba a la incapacidad del sintetizador (fruto una descripción inadecuada para la síntesis en FPGAs de esta tecnología) de manejar la complejidad de los bloques de Bierman y Thornton además de la temporización de la realimentación, simultáneamente.

Esta dificultad se presentó cuando se pretendió sintetizar tanto en las familias Virtex 5 como Spartan 6.

Luego de varias etapas de pruebas, en las que se buscó modificar el comportamiento de XST mediante la modificación de las opciones de síntesis, la única solución que se vislumbró al problema (que no requiriese modificar sustancialmente la descripción realizada) fue la de sintetizar cada bloque por separado y luego utilizar las netlists de cada uno para instanearlos como black box modules en el modulo del KF_BT.

# 8 Validación de Campo

Las siguientes pruebas se realizaron entre los días 1 y 30 del mes de Mayo de 2012 utilizando el software, librerías y FPGA descriptos en la sección 7. Las mismas fueron realizadas por los alumnos que desarrollaron el proyecto Dematties Darío e Iglesias Francisco con el asesoramiento del tutor Ing. Rodrigo Gonzalez del Laboratorio de Computación Reconfigurable de la Universidad Tecnológica Nacional Facultad Regional Mendoza.

## 8.1 Condiciones de prueba y desempeño numérico

En el esquema de diseño para FPGA propuesto por Xilinx existen varias instancias en las que se utilizan simuladores de alto nivel para verificar el correcto funcionamiento del sistema bajo desarrollo. Los dos tipos de verificación (utilizando simulación de alto nivel) más comúnmente usados son los que comprueban:

- El código VHDL desarrollado: Mediante una *simulación Behavioral*. Esta solo contempla el funcionamiento lógico del hardware sintetizado a partir del código en VHDL, por lo que no se utiliza para depurar errores relativos a temporización o ruteo. La ventaja de esta simulación es que requiere poca capacidad computacional para llevarse a cabo.

- El hardware sintetizado y ubicado en la FPGA destino: Mediante una *simulación Post-Route*. Esta contempla todos los aspectos relevantes que hacen al funcionamiento real del hardware sintetizado en la FPGA destino. Para esto utiliza modelos de propagación que estiman los tiempos de retardo en todas las líneas de transmisión en función de su longitud, temperatura y otros parametros. Lo mismo se aplica para las compuertas y dispositivos lógicos. Aquí cabe destacar que estos modelos, generados por el fabricante de la FPGA, son pesimistas por lo que *si el sistema funciona correctamente en la simulación Post-Route se garantiza su funcionamiento en el hardware bajo las condiciones de simulación* (basicamente temperatura y tensión de alimentación). La desventaja de este método es que requiere de gran capacidad computacional si desea simularse el funcionamiento del hardware por muchos ciclos de reloj.

Para verificar el correcto funcionamiento del hardware descripto se contrastó los resultados del KF-BT-FP codificado en MATLAB con los valores arrojados por las simulaciones.

Para facilitar esta comparación elaboramos el programa localizado en el archivo KF_test_for_implementation.m del anexo 11.1.6. Este programa computa el KF-BT-FP (codificado numéricamente igual que el VHDL) y devuelve los resultados en binario de todos los parámetros necesarios en cada iteración para poder compararlos directamente con los valores arrojados por la simulación.

Además, el Test Bench de VHDL que prueba el filtro en la FPGA se programó para que tome las mediciones de un archivo de texto “data_in.dat” y que escriba los valores de las

estimaciones en otro “data_out.dat”. El código del Test Bench se encuentra localizado en el archivo Kalman_Filter_Test_Bench.vhd del anexo 11.2.10.

### 8.1.1 Simulaciones Post-Route

Primero verificaremos el correcto funcionamiento del filtro en la FPGA mediante una simulación Post-Route para establecer principalmente que no hay problemas de temporización, sincronismo u otros asociados al hardware. Dado que el algoritmo es iterativo esto puede comprobarse con solo ejecutar unas pocas iteraciones.

Detallaremos primero algunos puntos importantes a tener en cuenta a la hora de ejecutar una simulación Post-Route:

- Los packages personales no son reconocidas cuando se elabora el archivo de la unidad bajo prueba (UUT). Estas deben ser agregadas, mediante la especificación del path correspondiente, al archivo Kalman_Filter_Test_Bench_par.prj.

- Además el Isim no encuentra por defecto el modelo de al UUT (archivo KF_BT_timesim) por lo que también debe especificarse su path en el archivo Kalman_Filter_Test_Bench_par.prj.

  Teniendo en cuenta estos dos cambios, el archivo en cuestión Kalman_Filter_Test_Bench_par.prj (poner All files para visualizarlo) queda:

  vhdl work "netgen/par/KF_BT_timesim.vhd"

  vhdl work "Kalman_Filter_Test_Bench.vhd"

  vhdl work "parameters.vhd"

  vhdl work "system_def.vhd"

- El tipo de señal buffer no es soportado en simulaciones Post-Route.

- Cualquier variable que sea un vector de vectores no es soportado en simulaciones Post-Route si se utiliza en los puertos de entrada o salida del bloque de mayor jerarquía (el que será la UUT del test bench). Esto sucede para el tipo WORDV_n, por esto se modificó la salida para que sean dos salidas tipo WORD individuales.

En la siguiente figura se presentan los resultados (estados estimados) de la simulación Post-Route para las primeras iteraciones del filtro mostrados mediante el diagrama de tiempo de las señales de salida:

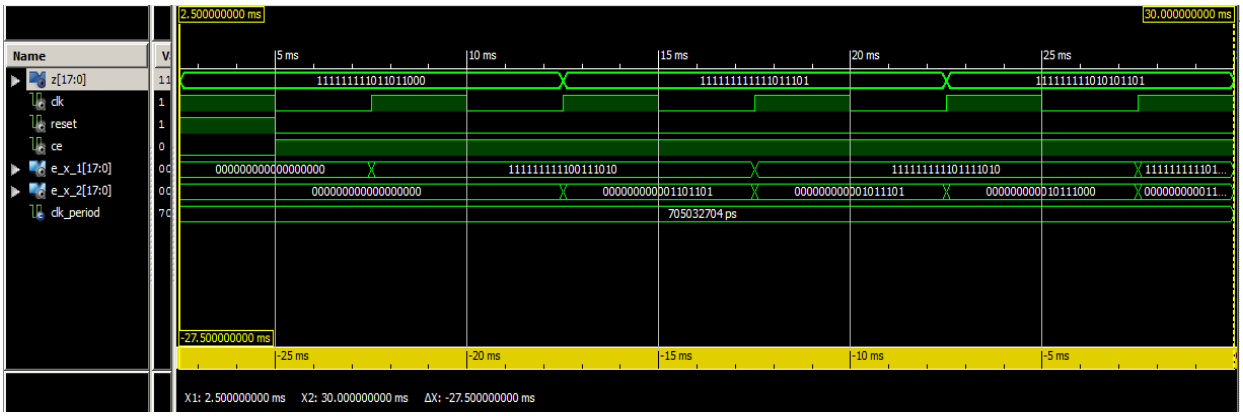



**Figura 8-1: Diagrama temporal con las salidas (especificadas en la columna de la izquierda) de la simulación Post-Route del KF-BT-FP.**

En la siguiente tabla se muestran las 7 iteraciones del experimento de la Figura 8-1, ademas se colocan al lado de los resultados arrojados por MATLAB para la misma realización. Como puede verse los resultados son idénticos.

| KF-BT-FP (WL=18; FL=9) | | | | | | |
|---|---|---|---|---|---|---|
| Z | E_X MATLAB | | E_X FPGA | | Diferencia ? | |
| WL=18; FL=9 | E_X1 | E_X2 | E_X1 | E_X2 | E_X1 | E_X2 |
| Prueba | 1 | 1 | 0 | 0 | ERR | ERR |
| 111111111011011000 | 111111111100111010 | 000000000000000000 | 111111111100111010 | 000000000000000000 | | |
| 111111111111011101 | 111111111101111010 | 000000000001011101 | 111111111101111010 | 000000000001011101 | | |
| 111111111010101101 | 111111111101000000 | 000000000011010001 | 111111111101000000 | 000000000011010001 | | |
| 000000000011001000 | 111111111110010101 | 000000000011111111 | 111111111110010101 | 000000000011111111 | | |
| 111111111110110101 | 111111111110011011 | 000000000101001010 | 111111111110011011 | 000000000101001010 | | |
| 111111111001101110 | 111111111101110010 | 000000000111000100 | 111111111101110010 | 000000000111000100 | | |
| 000000000000101110 | 111111111110001100 | 000000000111111101 | 111111111110001100 | 000000000111111101 | | |

**Tabla 8-1: Se comparan (columnas 6 y 7) los resultados de las estimaciones del KF-BT-FP funcionando en MATLAB (columnas 2 y 3) y en la FPGA (columnas 4 y 5) para 200 mediciones (columna 1) de una misma realización.**

### 8.1.2 Simulaciones Behavioral

Luego de demostrar que el filtro implementado en la FPGA no posee problemas de temporización (sección anterior), pueden utilizarse las simulaciones Behavioral para establecer que el funcionamiento numérico del mismo es idéntico al del filtro funcionando en MATLAB. Hacemos esto porque las simulaciones Behavioral requieren de menos tiempo de cómputo permitiendo realizar fácilmente una mayor cantidad de pruebas como se requiere para demostrar la igualdad numérica de ambas implementaciones (FPGA y MATLAB).

En la siguiente tabla se muestran y comparan las estimaciones de cada estado para 200 iteraciones ($2\ [s]$ para el periodo de muestreo utilizado) de la misma realización en las dos plataformas:

| KF-BT-FP (WL=18; FL=9, RM=Floor, OM=Saturate) | | | | | | |
|---|---|---|---|---|---|---|
| Z | E_X MATLAB | | E_X FPGA | | Diferencia? | |
| WL=18; FL=9 | E_X1 | E_X2 | E_X1 | E_X2 | E_X1 | E_X2 |
| Prueba | 1 | 1 | 0 | 0 | ERR | ERR |
| 000000001101000100 | 111111111011111010 | 000000000000000000 | 111111111011111010 | 000000000000000000 | | |
| 000000000110100111 | 000000000001111011 | 000000000000100010 | 000000000001111011 | 000000000000100010 | | |
| 000000000001110111 | 000000000011010010 | 000000000000010110 | 000000000011010010 | 000000000000010110 | | |
| 000000000101011011 | 111111111111100011 | 000000000010111001 | 111111111111100011 | 000000000010111001 | | |
| 111111111100000010 | 111111111100111100 | 000000000110001000 | 111111111100111100 | 000000000110001000 | | |
| 000000000110111111 | 000000000001101010 | 000000000010110001 | 000000000001101010 | 000000000010110001 | | |
| 000000000010010110 | 000000000001001001 | 000000000011110101 | 000000000001001001 | 000000000011110101 | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 000000000101010000 | 000000000001011000 | 000000000100000011 | 000000000001011000 | 000000000100000011 | | |
| 111111110111111101 | 000000000001100100 | 000000000100010000 | 000000000001100100 | 000000000100010000 | | |
| 000000001100101100 | 000000000001001001 | 000000000101011100 | 000000000001001001 | 000000000101011100 | | |
| 000000000011111110 | 000000000001000001 | 000000000110001111 | 000000000001000001 | 000000000110001111 | | |
| 111111110001010000 | 000000000001110001 | 000000000101001101 | 000000000001110001 | 000000000101001101 | | |
| 000000000100011101 | 000000000001111111 | 000000000101000110 | 000000000001111111 | 000000000101000110 | | |
| 000000001011000000 | 000000000010001101 | 000000000100111010 | 000000000010001101 | 000000000100111010 | | |
| 000000000110000010 | 000000000010011101 | 000000000100100010 | 000000000010011101 | 000000000100100010 | | |
| 000000001100000101 | 000000000010101101 | 000000000011111101 | 000000000010101101 | 000000000011111101 | | |
| 000000001011101110 | 000000000010100110 | 000000000100100011 | 000000000010100110 | 000000000100100011 | | |
| 111111111010000001 | 000000000010011110 | 000000000101010001 | 000000000010011110 | 000000000101010001 | | |
| 111111101110111111 | 000000000010011101 | 000000000101101110 | 000000000010011101 | 000000000101101110 | | |
| 000000001111111000 | 000000000010011101 | 000000000110001110 | 000000000010011101 | 000000000110001110 | | |
| 000000000100110101 | 000000000010011111 | 000000000110100010 | 000000000010011111 | 000000000110100010 | | |
| 000000000010000000 | 000000000010100010 | 000000000110100010 | 000000000010100010 | 000000000110100010 | | |
| 111111111110101111 | 000000000010100101 | 000000000110110011 | 000000000010100101 | 000000000110110011 | | |
| 111111110101110110 | 000000000010101000 | 000000000111000001 | 000000000010101000 | 000000000111000001 | | |
| 000000000100000011 | 000000000010101011 | 000000000111001001 | 000000000010101011 | 000000000111001001 | | |
| 111111110111011011 | 000000000010101110 | 000000000111010010 | 000000000010101110 | 000000000111010010 | | |
| 111111111011110110 | 000000000010110001 | 000000000111100110 | 000000000010110001 | 000000000111100110 | | |
| 000000000001000010 | 000000000010110100 | 000000001000001001 | 000000000010110100 | 000000001000001001 | | |
| 000000001000111010 | 000000000010111000 | 000000001000010101 | 000000000010111000 | 000000001000010101 | | |
| 000000000110000000 | 000000000010111100 | 000000001000001010 | 000000000010111100 | 000000001000001010 | | |
| 111111111000011000 | 000000000011000000 | 000000001000000011 | 000000000011000000 | 000000001000000011 | | |
| 000000001111001000 | 000000000011000100 | 000000001000101111 | 000000000011000100 | 000000001000101111 | | |
| 000000000011101011 | 000000000011001000 | 000000001001000000 | 000000000011001000 | 000000001001000000 | | |
| 000000010010100001 | 000000000011001011 | 000000001000110010 | 000000000011001011 | 000000001000110010 | | |
| 111111111010000001 | 000000000011010000 | 000000001001001010 | 000000000011010000 | 000000001001001010 | | |
| 111111111101100011 | 000000000011010101 | 000000001001101000 | 000000000011010101 | 000000001001101000 | | |
| 111111111101110011 | 000000000011011011 | 000000001010000011 | 000000000011011011 | 000000001010000011 | | |
| 000000000011001101 | 000000000011011111 | 000000001001110101 | 000000000011011111 | 000000001001110101 | | |
| 111111111100011000 | 000000000011100010 | 000000001001011100 | 000000000011100010 | 000000001001011100 | | |
| 111111111100100110 | 000000000011100101 | 000000001001000101 | 000000000011100101 | 000000001001000101 | | |
| 111111110001011001 | 000000000011101010 | 000000001001010000 | 000000000011101010 | 000000001001010000 | | |
| 111111111101110101 | 000000000011101110 | 000000001001000101 | 000000000011101110 | 000000001001000101 | | |
| 000000000110101101 | 000000000011110000 | 000000001000101000 | 000000000011110000 | 000000001000101000 | | |
| 000000000011010010 | 000000000011111000 | 000000001001011001 | 000000000011111000 | 000000001001011001 | | |
| 000000000011100000 | 000000000011111100 | 000000001001010001 | 000000000011111100 | 000000001001010001 | | |
| 000000000100000000 | 000000000100000000 | 000000001001000111 | 000000000100000000 | 000000001001000111 | | |
| 000000000110011000 | 000000000100000111 | 000000001001100001 | 000000000100000111 | 000000001001100001 | | |
| 000000000001000111 | 000000000100010001 | 000000001010011001 | 000000000100010001 | 000000001010011001 | | |
| 111111111111100011 | 000000000100011000 | 000000001010011001 | 000000000100011000 | 000000001010011001 | | |
| 000000001001010101 | 000000000100011011 | 000000001001110101 | 000000000100011011 | 000000001001110101 | | |
| 000000000001111110 | 000000000100011110 | 000000001001001101 | 000000000100011110 | 000000001001001101 | | |
| 000000000111001111 | 000000000100100011 | 000000001001001001 | 000000000100100011 | 000000001001001001 | | |
| 111111111101110010 | 000000000100100110 | 000000001000100010 | 000000000100100110 | 000000001000100010 | | |
| 000000000100010110 | 000000000100101001 | 000000001000001000 | 000000000100101001 | 000000001000001000 | | |
| 000000001011100001 | 000000000100101110 | 000000000111111101 | 000000000100101110 | 000000000111111101 | | |
| 111111111110011111 | 000000000100101111 | 000000000111010001 | 000000000100101111 | 000000000111010001 | | |
| 000000000110110000 | 000000000100110000 | 000000000110101001 | 000000000100110000 | 000000000110101001 | | |
| 000000000011110010 | 000000000100110011 | 000000000110011010 | 000000000100110011 | 000000000110011010 | | |
| 000000000101000110 | 000000000100110100 | 000000000101101110 | 000000000100110100 | 000000000101101110 | | |
| 000000000010110111 | 000000000100110110 | 000000000101011010 | 000000000100110110 | 000000000101011010 | | |
| 111111110101001000 | 000000000100111000 | 000000000101000100 | 000000000100111000 | 000000000101000100 | | |
| 000000000100100111 | 000000000100111010 | 000000000100110000 | 000000000100111010 | 000000000100110000 | | |
| 000000010000111011 | 000000000100111011 | 000000000100010010 | 000000000100111011 | 000000000100010010 | | |
| 000000001011000000 | 000000000100111100 | 000000000011111000 | 000000000100111100 | 000000000011111000 | | |
| 000000000010110001 | 000000000100111101 | 000000000011100000 | 000000000100111101 | 000000000011100000 | | |
| 000000000000011111 | 000000000100111110 | 000000000011010001 | 000000000100111110 | 000000000011010001 | | |
| 000000000111110000 | 000000000100111111 | 000000000010111010 | 000000000100111111 | 000000000010111010 | | |
| 000000000010101100 | 000000000100111111 | 000000000010100011 | 000000000100111111 | 000000000010100011 | | |
| 000000000001100000 | 000000000100111111 | 000000000010001011 | 000000000100111111 | 000000000010001011 | | |
| 000000000110001010 | 000000000100111111 | 000000000001110101 | 000000000100111111 | 000000000001110101 | | |
| 000000001111101110 | 000000000100111111 | 000000000001011110 | 000000000100111111 | 000000000001011110 | | |
| 111111110001011111 | 000000000100111110 | 000000000001000111 | 000000000100111110 | 000000000001000111 | | |
| 000000001010010011 | 000000000100111101 | 000000000000110010 | 000000000100111101 | 000000000000110010 | | |
| 111111111101100001 | 000000000100111100 | 000000000000011101 | 000000000100111100 | 000000000000011101 | | |
| 000000001101000110 | 000000000100111011 | 000000000000001001 | 000000000100111011 | 000000000000001001 | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 111111110010000110 | 000000000100111010 | 111111111111110101 | 000000000100111010 | 111111111111110101 | | |
| 000000000011101010 | 000000000100111000 | 111111111111100011 | 000000000100111000 | 111111111111100011 | | |
| 000000000101110111 | 000000000100110110 | 111111111111010001 | 000000000100110110 | 111111111111010001 | | |
| 000000000001111100 | 000000000100110100 | 111111111111000001 | 000000000100110100 | 111111111111000001 | | |
| 000000000011101010 | 000000000100110010 | 111111111110110001 | 000000000100110010 | 111111111110110001 | | |
| 111111111001000111 | 000000000100110000 | 111111111110100010 | 000000000100110000 | 111111111110100010 | | |
| 111111111100001010 | 000000000100101110 | 111111111110010100 | 000000000100101110 | 111111111110010100 | | |
| 111111111110100101 | 000000000100101011 | 111111111110000111 | 000000000100101011 | 111111111110000111 | | |
| 000000000111111110 | 000000000100101000 | 111111111101111010 | 000000000100101000 | 111111111101111010 | | |
| 000000000000010010 | 000000000100100101 | 111111111101101110 | 000000000100100101 | 111111111101101110 | | |
| 000000010001001101 | 000000000100100010 | 111111111101100100 | 000000000100100010 | 111111111101100100 | | |
| 000000000101111101 | 000000000100011111 | 111111111101011011 | 000000000100011111 | 111111111101011011 | | |
| 111111110101110111 | 000000000100011100 | 111111111101010010 | 000000000100011100 | 111111111101010010 | | |
| 000000000001000100 | 000000000100011001 | 111111111101001010 | 000000000100011001 | 111111111101001010 | | |
| 111111111111011010 | 000000000100010110 | 111111111101000011 | 000000000100010110 | 111111111101000011 | | |
| 000000000101011000 | 000000000100010011 | 111111111100111110 | 000000000100010011 | 111111111100111110 | | |
| 000000000010101110 | 000000000100010000 | 111111111100111001 | 000000000100010000 | 111111111100111001 | | |
| 000000000010001011 | 000000000100001101 | 111111111100110101 | 000000000100001101 | 111111111100110101 | | |
| 111111111110011110 | 000000000100001010 | 111111111100110010 | 000000000100001010 | 111111111100110010 | | |
| 000000000011110000 | 000000000100000110 | 111111111100110000 | 000000000100000110 | 111111111100110000 | | |
| 111111111000101001 | 000000000100000010 | 111111111100101111 | 000000000100000010 | 111111111100101111 | | |
| 000000001001000010 | 000000000011111110 | 111111111100101111 | 000000000011111110 | 111111111100101111 | | |
| 111111111001011110 | 000000000011111010 | 111111111100101111 | 000000000011111010 | 111111111100101111 | | |
| 111111111110101101 | 000000000011110110 | 111111111100110000 | 000000000011110110 | 111111111100110000 | | |
| 000000000010110000 | 000000000011110010 | 111111111100110010 | 000000000011110010 | 111111111100110010 | | |

**Tabla 8-2: Se comparan (columnas 6 y 7) los resultados de las estimaciones del KF-BT-FP funcionando en MATLAB (columnas 2 y 3) y en la FPGA (columnas 4 y 5) para 200 mediciones (columna 1) de una misma realización.**

Como puede verse, el filtro en la FPGA responde de manera idéntica en cada iteración que su par en MATLAB. En la siguiente figura se muestra el diagrama de tiempo con las señales de los primeros valores de la Tabla 8-2 obtenidos desde Isim. (Recuérdese que a partir del primer flanco de subida en que CE=1, los valores a la salida surgen cada un ciclo en la secuencia Thornton-Bierman-Thornton-Bierman-…):

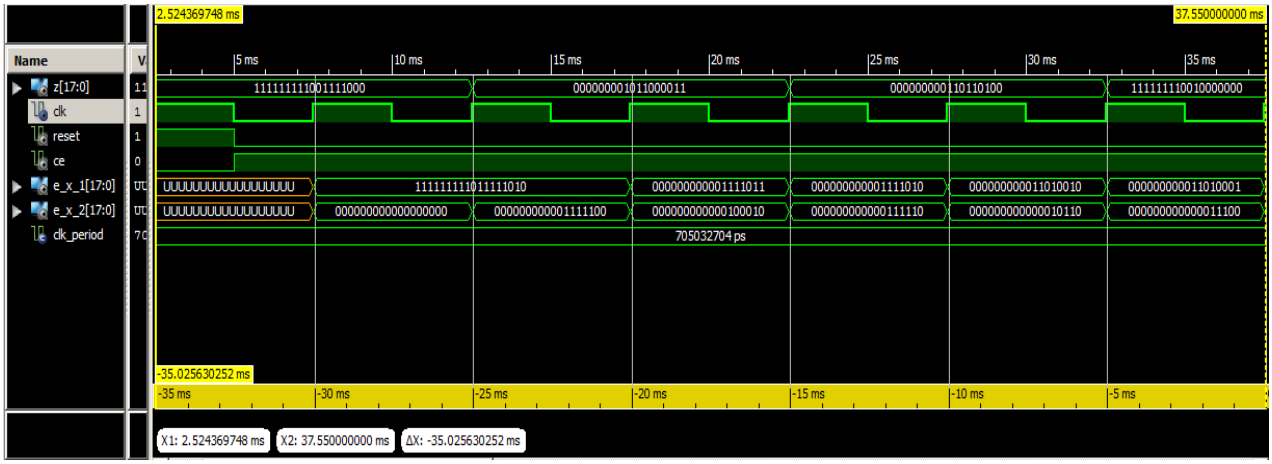



**Figura 8-2: Diagrama temporal con algunas de las salidas del experimento de la Tabla 8-2. A la izquierda puede apreciarse el nombre de cada señal.**

En la siguiente tabla se resumen otros experimentos realizados que al igual que el de la Tabla 8-2 tampoco arrojaron diferencias entre las dos plataformas, verificando así su equivalencia numérica:

| Numero de Prueba | Cant. de iteraciones | Tiempo de simulación[s] | Errores detectados |
|---|---|---|---|
| 1 | 200 | 2 | No |
| 2 | 200 | 2 | No |
| 3 | 1000 | 10 | No |

**Tabla 8-3: Otros experimentos realizaos para verificar que el filtro se comporta numéricamente igual en MATLAB que en el hardware (FPGA)**

### 8.1.3 Desempeño numérico

La ventaja de haber realizado un estudio detallado, como el de las secciones 1, 1 y 9, sobre los problemas de implementación del KF-BT en aritmética de FP es que las pruebas del funcionamiento numérico del hardware se simplifican. Esto se debe a que uno tiene un control pleno (a nivel de bit) del funcionamiento del algoritmo sintetizado. Dado que en la sección anterior se mostró que el KF-BT-FP en la FPGA funciona de manera idéntica que el de MATLAB, todo lo analizado en las secciones 6.5.2, 6.5.5 y 6.6 respecto del comportamiento numérico del filtro es válido también para el hardware funcionando en la FPGA.

Adicionalmente añadimos aquí a modo de conclusión, la Figura 8-3 y la Figura 8-4 correspondiente al último experimento de la Tabla 8-3, en la que se muestran el diagrama de tiempo con las señales obtenidas de Isim y una grafica con los estados estimados, exactos y las mediciones.

Notese la diferencia (para la aritmetica seleccionada en la sección 6.6) en el funcionamiento del KF-BT-FP mostrado, comparado con el del KF-OA-FP mostrado en la Figura 8-5 (ver sección 6.5.5):

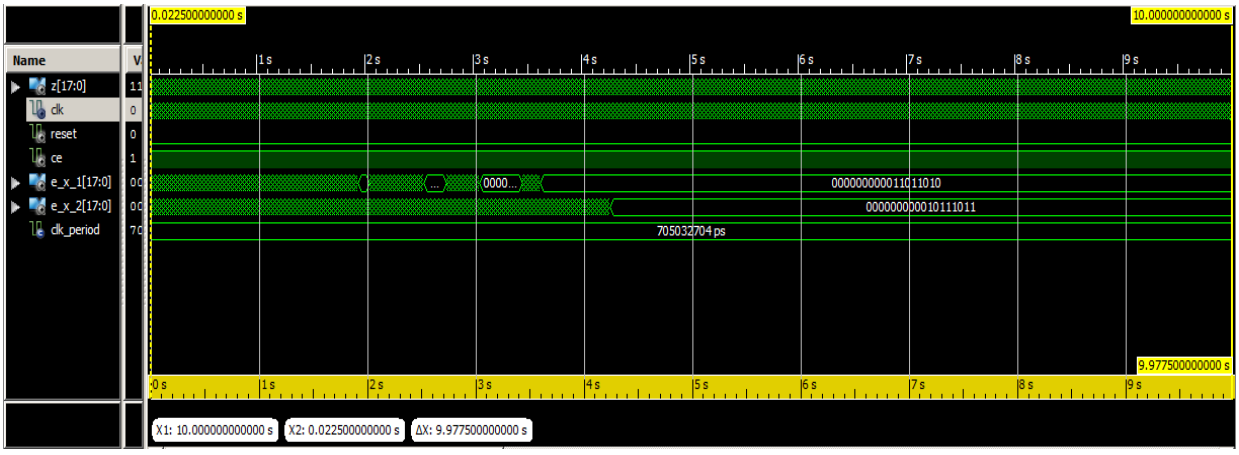



**Figura 8-3: KF-BT-FP funcionando en la FPGA, se muestra el diagramas de tiempo de la señales de salida.**

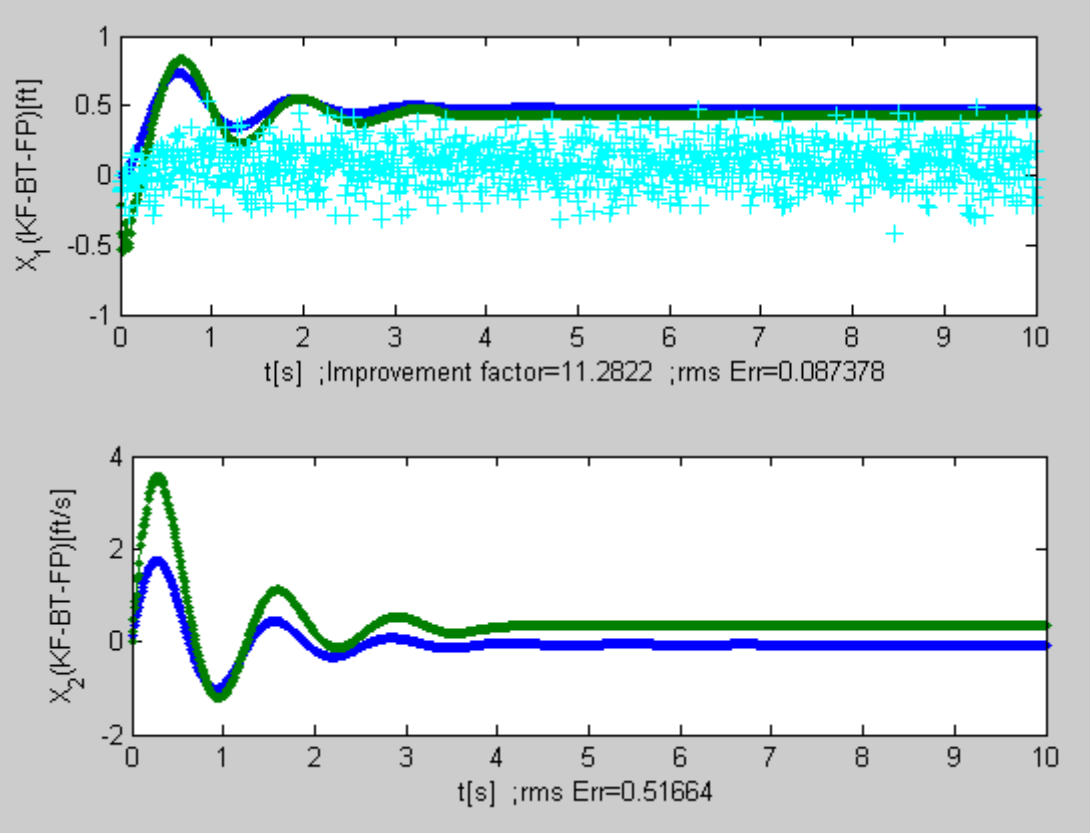



**Figura 8-4: KF-BT-FP funcionando en la FPGA, se muestran las estimaciones (en Verde) , los estados exactos (en Azul) y las mediciones (en Cian).**

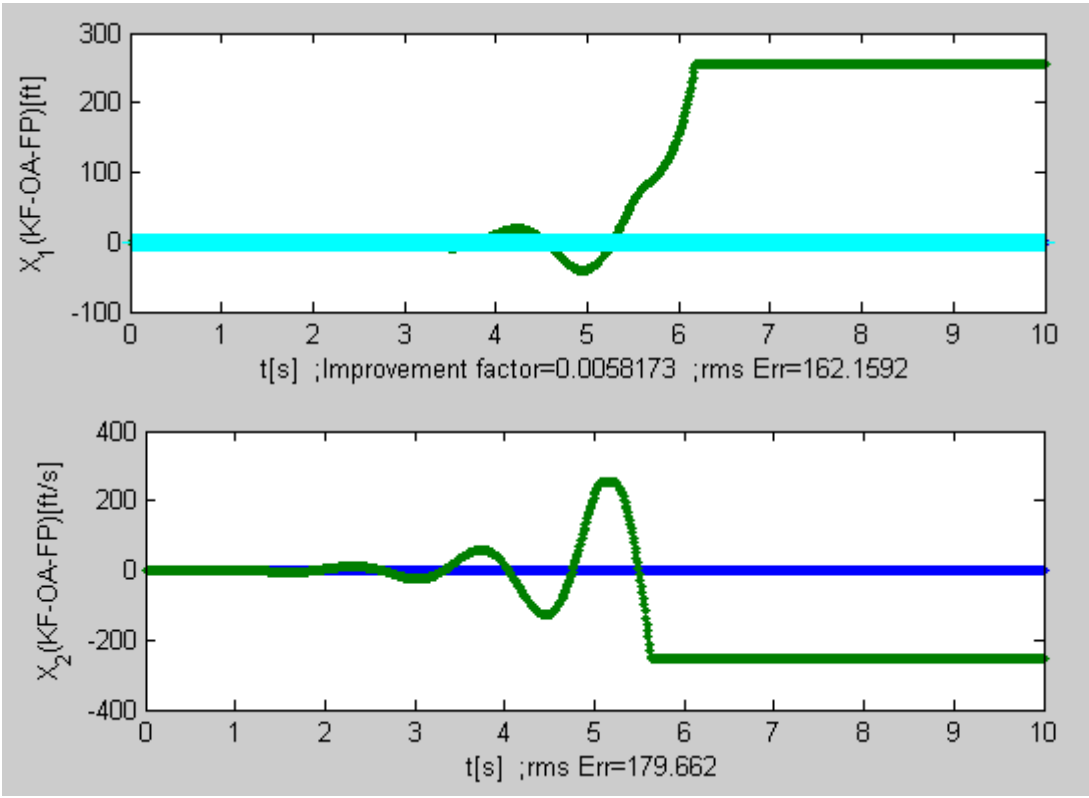



**Figura 8-5: KF-OA-FP (en MATLAB) funcionando para la misma realización y con la misma aritmetica que el KF-BT-FP de la Figura 8-4.**

## 8.2 Recursos de Hardware

En la siguiente tabla se presentan, discriminadas por modulo, las primitivas de hardware inferidas por el sintetizador XST en la implementación del KF-BT-FP (siempre con la aritmética seleccionada en la sección 6.6) en la FPGA especificada en la sección 7.1.

| Modulo de Bierman | Modulo de Thornton | Data Path Controller |
|---|---|---|
| # Multipliers : 11<br>18x18-bit multiplier : 11 | # Multipliers : 23<br>18x10-bit multiplier : 4<br>18x18-bit multiplier : 11<br>18x4-bit multiplier : 1<br>18x5-bit multiplier : 1<br>18x8-bit multiplier : 6 | |
| # Adders/Subtractors : 65<br>1-bit subtractor : 2<br>10-bit subtractor : 2<br>11-bit subtractor : 2<br>12-bit subtractor : 2<br>13-bit subtractor : 2<br>14-bit subtractor : 2<br>15-bit subtractor : 2<br>16-bit subtractor : 2<br>17-bit subtractor : 2<br>18-bit subtractor : 2<br>19-bit adder .. : 7<br>19-bit subtractor : 8<br>2-bit subtractor : 2<br>20-bit subtractor : 2<br>21-bit subtractor : 2<br>22-bit subtractor : 2<br>23-bit subtractor : 2<br>24-bit subtractor : 2<br>25-bit subtractor : 2<br>26-bit subtractor : 2<br>3-bit subtractor : 2<br>4-bit subtractor : 2<br>5-bit subtractor : 2<br>6-bit subtractor : 2<br>7-bit subtractor : 2<br>8-bit subtractor : 2<br>9-bit subtractor : 2 | # Adders/Subtractors : 43<br>1-bit subtractor : 1<br>10-bit subtractor : 1<br>11-bit subtractor : 1<br>12-bit subtractor : 1<br>13-bit subtractor : 1<br>14-bit subtractor : 1<br>15-bit subtractor : 1<br>16-bit subtractor : 1<br>17-bit subtractor : 1<br>18-bit subtractor : 1<br>19-bit adder : 12<br>19-bit subtractor : 6<br>2-bit subtractor : 1<br>20-bit subtractor : 1<br>21-bit subtractor : 1<br>22-bit subtractor : 1<br>23-bit subtractor : 1<br>24-bit subtractor : 1<br>25-bit subtractor : 1<br>26-bit subtractor : 1<br>3-bit subtractor : 1<br>4-bit subtractor : 1<br>5-bit subtractor : 1<br>6-bit subtractor : 1<br>7-bit subtractor : 1<br>8-bit subtractor : 1<br>9-bit subtractor : 1 | # Adders/Subtractors : 1<br>1-bit adder : 1 |
| # Comparators : 54<br>27-bit comparator greater : 18<br>27-bit comparator lessequal : 36 | # Comparators : 27<br>27-bit comparator greater : 9<br>27-bit comparator lessequal : 18 | |
| # Multiplexers : 739<br>1-bit 2-to-1 multiplexer : 696<br>18-bit 2-to-1 multiplexer : 39<br>27-bit 2-to-1 multiplexer : 4 | # Multiplexers : 405<br>1-bit 2-to-1 multiplexer : 348<br>18-bit 2-to-1 multiplexer : 55<br>27-bit 2-to-1 multiplexer : 2 | # Multiplexers : 3<br>1-bit 2-to-1 multiplexer : 1<br>18-bit 2-to-1 multiplexer :2 |
| # Xors : 15<br>1-bit xor2 : 15 | # Xors : 27<br>1-bit xor2 : 27 | |
| | | # Registers : 217<br>Flip-Flops : 217 |

**Tabla 8-4: Primitivas inferidas por el sintetizador XST al implementar el KF-BT-FP separadas por modulo.**

En la siguiente tabla se muestran los recursos del hardware de la FPGA utilizados para implementar las primitivas de la Tabla 8-4 y su interconexión:

| Device Utilization Summary | | | | [-] |
|---|---|---|---|---|
| **Slice Logic Utilization** | **Used** | **Available** | **Utilization** | **Note(s)** |
| Number of Slice Registers | 223 | 54,576 | 1% | |

| | | | | |
|---|---|---|---|---|
| Number used as Flip Flops | 223 | | | |
| Number used as Latches | 0 | | | |
| Number used as Latch-thrus | 0 | | | |
| Number used as AND/OR logics | 0 | | | |
| Number of Slice LUTs | 3,943 | 27,288 | 14% | |
| Number used as logic | 3,938 | 27,288 | 14% | |
| Number using O6 output only | 2,931 | | | |
| Number using O5 output only | 37 | | | |
| Number using O5 and O6 | 970 | | | |
| Number used as ROM | 0 | | | |
| Number used as Memory | 0 | 6,408 | 0% | |
| Number used exclusively as route-thrus | 5 | | | |
| Number with same-slice register load | 0 | | | |
| Number with same-slice carry load | 5 | | | |
| Number with other load | 0 | | | |
| Number of occupied Slices | 1,281 | 6,822 | 18% | |
| Number of LUT Flip Flop pairs used | 3,945 | | | |
| Number with an unused Flip Flop | 3,722 | 3,945 | 94% | |
| Number with an unused LUT | 2 | 3,945 | 1% | |
| Number of fully used LUT-FF pairs | 221 | 3,945 | 5% | |
| Number of unique control sets | 5 | | | |
| Number of slice register sites lost to control set restrictions | 17 | 54,576 | 1% | |
| Number of bonded IOBs | 57 | 218 | 26% | |
| Number of RAMB16BWERs | 0 | 116 | 0% | |
| Number of RAMB8BWERs | 0 | 232 | 0% | |
| Number of BUFIO2/BUFIO2_2CLKs | 0 | 32 | 0% | |
| Number of BUFIO2FB/BUFIO2FB_2CLKs | 0 | 32 | 0% | |
| Number of BUFG/BUFGMUXs | 1 | 16 | 6% | |
| Number used as BUFGs | 1 | | | |
| Number used as BUFGMUX | 0 | | | |
| Number of DCM/DCM_CLKGENs | 0 | 8 | 0% | |
| Number of ILOGIC2/ISERDES2s | 0 | 376 | 0% | |
| Number of IODELAY2/IODRP2/IODRP2_MCBs | 0 | 376 | 0% | |
| Number of OLOGIC2/OSERDES2s | 0 | 376 | 0% | |

| | | | | |
|---|---|---|---|---|
| Number of BSCANs | 0 | 4 | 0% | |
| Number of BUFHs | 0 | 256 | 0% | |
| Number of BUFPLLs | 0 | 8 | 0% | |
| Number of BUFPLL_MCBs | 0 | 4 | 0% | |
| Number of DSP48A1s | 33 | 58 | 56% | |
| Number of ICAPs | 0 | 1 | 0% | |
| Number of MCBs | 0 | 2 | 0% | |
| Number of PCILOGICSEs | 0 | 2 | 0% | |
| Number of PLL_ADVs | 0 | 4 | 0% | |
| Number of PMVs | 0 | 1 | 0% | |
| Number of STARTUPs | 0 | 1 | 0% | |
| Number of SUSPEND_SYNCs | 0 | 1 | 0% | |
| Average Fanout of Non-Clock Nets | 4.23 | | | |

**Tabla 8-5: Recursos de hardware ocupados por el KF-BT-FP en la FPGA Spartan 6 detallada en la sección 7.1.**

Adicionalmente, mostramos en la siguiente tabla, únicamente a modo indicativo, los recursos de hardware consumidos por el KF-BT-FP para diferentes dimensiones del vector de estado (valores de $n$):

| Cantidad de estados | Slices | DSP48A |
|---|---|---|
| 2 | 4786 | 33 |
| 4 | 8899 | 78 |
| 6 | 12815 | 118 |
| 8 | 16615 | 158 |
| 10 | 20485 | 198 |

**Tabla 8-6: Recursos de hardware de la FPGA consumidos por el KF-BT-FP para diferentes dimensiones del vector de estados.**

Estos resultados son acordes a la implementación de KF-BT dada en [**15**]

## 8.3 Desempeño temporal

La máxima frecuencia de reloj obtenida para el KF-BT es de: $\frac{1}{217.709\,[ns]} = 4.59\,[MHz]$. Dado que para $l = 1$, se procesa un vector de mediciones completo cada 2 ciclos de reloj esto equivale a una frecuencia de muestreo máxima del filtro de $2.296\,x10^6\,[\frac{muestras}{s}]$ o una muestra cada aproximadamente $T = 436\,[ns]$.

En la Figura 8-5, se muestra el diagrama de tiempo con las señales de salida del filtro operando a su máxima frecuencia de reloj (el experimento completo constó de 5 interaciones en las que no se detecto ningún error).

Dado este rendimiento temporal, el sistema desarrollado no presentó ninguna dificultad en ejecutar el caso que implementamos con $T = 0.01\,[s]$.

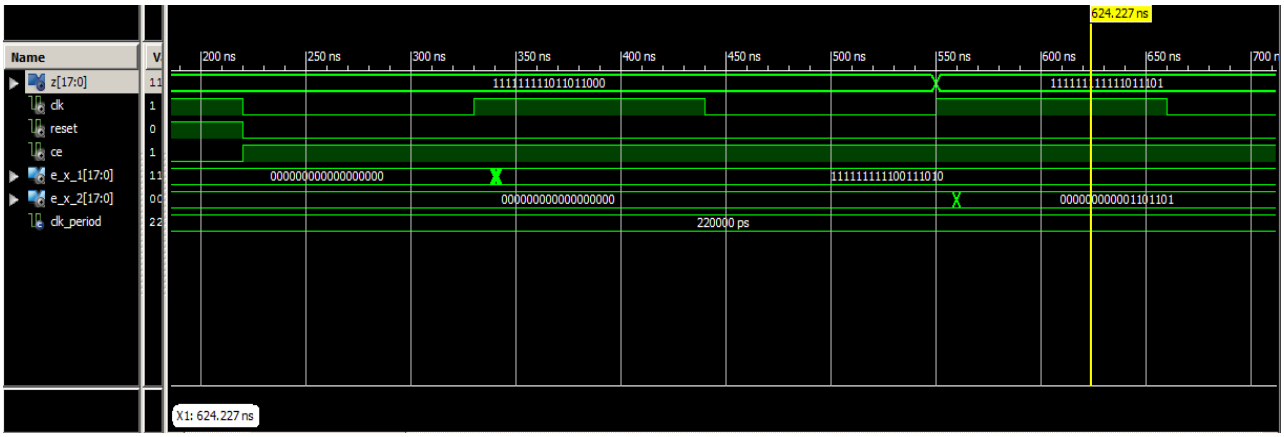



**Figura 8-6: Diagrama temporal con algunas de las salidas de una simulación Post-Route con 5 iteraciones operando a la máxima frecuencia de reloj. A la izquierda se detalla el nombre de cada señal.**

## 8.4 Consumo eléctrico estimado

En las siguientes tablas se muestra el consumo estimado de la implementación del KF-BT-FP (con la aritmética seleccionada en la sección 6.6) en la FPGA especificada en la sección 7.1. Los datos fueron extraídos del programa Xilinx XPower Analyzer V.12.4 (nt) y constituyen estimaciones del consumo eléctrico basadas en modelos desarrollados por el fabricante de la FPGA. Estos pueden diferir del consumo real según se modifiquen las condiciones de simulación (temperatura ambiente, tensión de alimentación, tasas de conmutación, etc.).

Aquí definimos el reloj a 4 [MHz] y las tasas de conmutación como se detallan en la siguiente figura (nótese que se utilizo la máxima posible para los DSP dado que es un componente crítico del diseño):

| | | |
|---|---|---|
| FF Toggle Rate (%) | 12.5 | 0.0 to 200.0 |
| I/O Toggle Rate (%) | 12.5 | 0.0 to 200.0 |
| Output Load (pF) | 5.0 | 0.0 to 1000000.0 |
| I/O Enable Rate (%) | 100.0 | 0.0 to 100.0 |
| BRAM Write Rate (%) | 50.0 | 0.0 to 100.0 |
| BRAM Enable Rate (%) | 25.0 | 0.0 to 100.0 |
| DSP Toggle Rate (%) | 200.0 | 0.0 to 200.0 |

**Figura 8-7: Tasas de conmutación adoptadas para la simulación de consumo eléctrico de XPower Analyzer.**

El resto de las condiciones de simulación son las detalladas en la siguiente figura:

| Device | |
|---|---|
| Family | Spartan6 |
| Part | xc6slx45 |
| Package | csg324 |
| Grade | C-Grade |
| Process | Typical |
| Speed Grade | -2 |

| Environment | |
|---|---|
| Ambient Temp (C) | 25.0 |
| Use custom TJA? | No |
| Custom TJA (C/W) | NA |
| Airflow (LFM) | 0 |

| Thermal Properties | Effective TJA (C/W) | Max Ambient (C) | Junction Temp (C) |
|---|---|---|---|
| | 22.6 | 84.2 | 25.8 |

**Figura 8-8: Condiciones de simulación adoptadas para el análisis de consumo de XPower Analyzer.**

Los resultados de la simulación de consumo efectuada por XPower Analyzer pueden verse en la siguiente figura:

| On-Chip | Power (W) | Used | Available | Utilization (%) |
|---|---|---|---|---|
| Clocks | 0.000 | 1 | --- | --- |
| Logic | 0.002 | 3446 | 27288 | 13 |
| Signals | 0.003 | 4427 | --- | --- |
| DSPs | 0.000 | 27 | 58 | 47 |
| IOs | 0.001 | 57 | 218 | 26 |
| Leakage | 0.031 | | | |
| Total | 0.037 | | | |

**Figura 8-9: Resultados de la simulación de consumo eléctrico de XPower Analyzer.**

# 9 Análisis de factibilidad económica

La evaluación económica de proyectos de cooperación tiene por objetivo identificar las ventajas y desventajas asociadas a la inversión en un proyecto antes de la implementación del mismo.

La evaluación económica es un método de análisis útil para adoptar decisiones racionales ante diferentes alternativas. Es un análisis comparativo de las acciones alternativas tanto en términos de costes como de beneficios con el fin de demostrar la factibilidad y rentabilidad económica de un proyecto.

Es necesario mencionar que sólo se evaluará la etapa del diseño y de la producción del proyecto sin incluir la etapa de comercialización del producto.

Para evaluar la factibilidad económica del proyecto se utilizará el método simplificado de formulación de proyectos, con lo que se pretende determinar el Valor Actual Neto (VAN), la Tasa Interna de Retorno (TIR) y la curva de sensibilidad.

Definición de objetivos:

Creación de los contenidos de conocimientos (know-how) necesarios que permitan emprender investigaciones o desarrollos regionales en el área de la implementación del filtro de Kalman en dispositivos de lógica reconfigurable.

El producto final consta de:

- Material necesario que cree los conocimientos de base orientados específicamente a encarar el proceso de investigación o desarrollo.
- Código fuente del software necesario para la realización de un perfil de pruebas a llevar a cabo en el proceso de investigación.
- Código fuente del software necesario para la descripción del hardware en el que se realizarán las implementaciones del algoritmo del filtro de Kalman.

El mercado al cual apunta el proyecto es la venta del material producido a laboratorios, instituciones educativas de nivel superior y empresas de servicios de ingeniería.

## 9.1 Evaluación del proyecto

Tanto los costes como los beneficios se miden en dinero. Se trata del tipo de análisis más extendido en los estudios de viabilidad de inversiones productivas.

Definición del nivel de tecnología y los costos correspondientes:

Inversiones iniciales:

| Inversiones iniciales | Valor unitario ($) | Cantidad | Valor ($) |
|---|---|---|---|
| Notebooks | 3200 | 2 | 6400 |

| | | | |
|---|---|---|---|
| Manuales VHDL | 300 | 2 | 600 |
| Libros de teoría sobre el KF. | 300 | 3 | 900 |
| Entorno de desarrollo Xilinx ISE | 11000 | 1 | 11000 |
| Programa de cálculo Matlab R2008a | 16000 | 1 | 16000 |
| Horas de investigación | 70 | 190 | 13300 |
| Horas de desarrollo de software | 70 | 230 | 16100 |
| | | | 64300 |

Costos corrientes anuales:

Se han estipulado gastos concernientes a la confección del producto terminado.

| Costos corrientes | Valor unitario ($) | Cantidad | Valor ($) |
|---|---|---|---|
| Impresión del material teórico. | 40 | 4 | 160 |
| Grabación del software en el soporte tecnológico. | 2 | 4 | 8 |
| Packaging | 5 | 4 | 20 |
| | | | 188 |

Egresos corrientes anuales:

Estos se computan por medio de las anualidades de los costos. Con esto se tiene en cuenta la inversión inicial en la formación de los egresos corrientes a través del factor de recuperación de la inversión.

El factor de recuperación para un interés del $i = 10\%$ y para un lapso de tiempo de $T = 5\ años$ años es:

$$FR(i,T) = 0.264$$

En el presente proyecto se considerará una liquidación de los bienes adquiridos en un 10% de su valor inicial, por lo tanto $L = \$3490$. Esto se debe a que el material utilizado es de rápida obsolescencia.

| Egresos corrientes | Valor unitario ($) | Cantidad | Valor ($) |
|---|---|---|---|
| Producto terminado | 16449,84 | 4 | 65799,36 |

Precio de venta neto del producto terminado:

Para la formación del precio, se incurrirá en un beneficio del 45%.

| Precio de venta neto ($) | Beneficios esperados |
|---|---|
| 23852,268 | 0,45 |

Ingresos corrientes anuales:

Son los que resultan de las ventas del material.

| Ingresos corrientes | Valor unitario ($) | Cantidad | Valor ($) |
|---|---|---|---|
| Producto terminado. | 23852,268 | 4 | 95409,072 |

Beneficios netos anuales o flujo de caja:

| Flujo de caja | | | |
|---|---|---|---|
| Año | Costo($) | Ingreso($) | FC($) |
| 0 | 64300 | 0 | -64300 |
| 1 | 65799,36 | 95409,072 | 29609,712 |
| 2 | 65799,36 | 95409,072 | 29609,712 |
| 3 | 65799,36 | 95409,072 | 29609,712 |
| 4 | 65799,36 | 95409,072 | 29609,712 |
| 5 | 65799,36 | 95409,072 | 29609,712 |

| | | | |
|---|---|---|---|
| L | 0 | 3490 | 3490 |
| Total | 393296,8 | 480535,36 | 87238,56 |

## 9.2 Flujo de fondos

VAN: valor presente de los beneficios netos que genera un proyecto a lo largo de su vida útil, descontados a la tasa de interés que refleja el costo de oportunidad para él o la inversionista que tiene el capital que piensa invertir en el proyecto.

En la tabla siguiente se expresan los valores de los flujos de caja actualizados a una tasa de descuento del 10%.

| Flujo de fondos | 0 | 1 | 2 | 3 | 4 | 5 | L |
|---|---|---|---|---|---|---|---|
| I = Inv. Inicial [$] | 64300 | | | | | | 3490 |
| Egresos Corrientes Anuales [$] | | 65799.36 | 65799.36 | 65799.36 | 65799.36 | 65799.36 | |
| Ingresos Corrientes Anuales [$] | | 95409.072 | 95409.072 | 95409.072 | 95409.072 | 95409.072 | |
| Retornos Anuales [$] | 0 | 29609.712 | 29609.712 | 29609.712 | 29609.712 | 29609.712 | |
| Qn = Beneficios Netos Anuales [$] | -64300 | 29609.712 | 29609.712 | 29609.712 | 29609.712 | 29609.712 | |
| 1/(1+r)^n | 1 | 0.90909091 | 0.82644628 | 0.7513148 | 0.68301346 | 0.62092132 | |
| VN [$] | -64300 | 26917.92 | 24470.8364 | 22246.2149 | 20223.8317 | 18385.3016 | 2167 |

| | | |
|---|---|---|
| VAN | 50111.1 | >>>0 |
| TIR | 36.88% | >30% |

El flujo de caja actualizado a un 10% asciende a algo más de 50 mil pesos al cabo de 5 años. Este dato es muy importante para valorar la conveniencia o no de realizar la inversión.

La regla de decisión es que si el VAN es positivo el proyecto se aprueba, pues será posible obtener ganancias respecto a la mejor inversión alternativa, si es igual a cero es indiferente y si es negativo se rechaza. O sea:

- VAN > 0 Se aprueba
- VAN = 0 Indiferente
- VAN < 0 Se rechaza

Se debe notar que este resultado se ha obtenido aplicando una tasa de descuento del 10% de manera arbitraria. Se ha fijado un 10% porque ha parecido razonable de acuerdo a la tendencia de los tipos de interés y de la inflación en un país y en un momento determinado.

La Tasa Interna de Retorno de un proyecto mide la rentabilidad promedio anual que genera el capital que permanece invertido en él. Se define como aquella tasa de descuento que iguala a cero el Valor Actual Neto. Es un valor que nos remite a un determinado tipo de interés para el cual realizar o no la inversión sería indiferente. Cuanta más alta sea la TIR más alta será la rentabilidad esperada del negocio y, al revés, cuanto más baja la TIR más riesgo corremos al realizar la inversión. Si la TIR es mayor que el coste de oportunidad del capital, entonces el capital del proyecto evaluado genera una rentabilidad mayor que la que puede ser generada por la mejor alternativa de inversión. En ese caso es recomendable apostar por el proyecto.

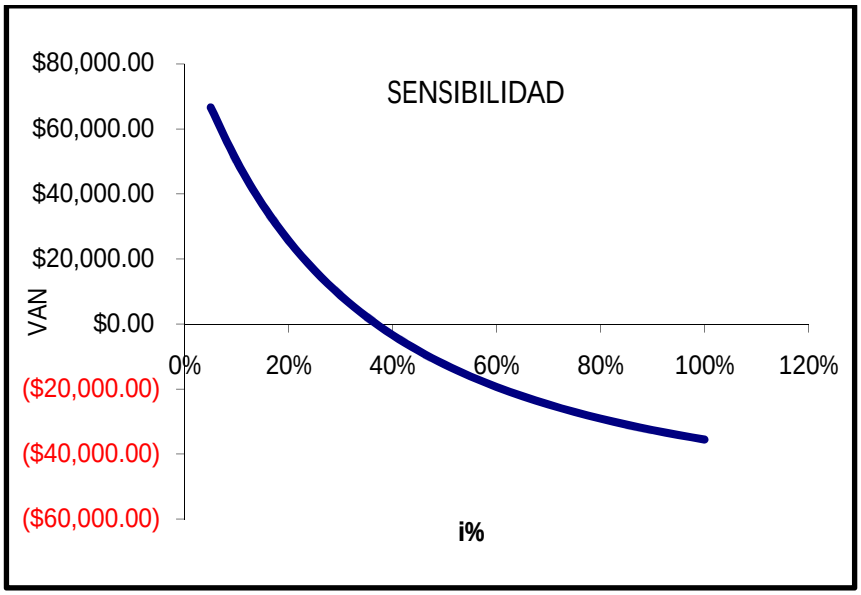
## 9.3 Análisis de sensibilidad

Un requerimiento en los proyectos de inversión es medir el riesgo que se corre cuando alguna de las magnitudes de los factores financieros del proyecto pueda tener variaciones. De igual manera el encargado de tomar la decisión deberá saber la sensibilidad del resultado ante las variaciones de esas magnitudes, como lo sería por ejemplo el cambio en el programa de inversiones o del flujo de caja, la tasa de interés también pudiese variar a lo largo de la vida útil del proyecto. Si hay inestabilidad económica o incluso política, la moneda del país está sometida a frecuentes devaluaciones y no se puede controlar la inflación, entonces se tendría que elegir una tasa de descuento más alta, es decir se pedirían flujos de caja más elevados, reduciendo así el número de negocios que sería rentable poner en marcha en ese país y en esas condiciones.

Mediante el análisis de la sensibilidad, es posible identificar las variaciones máximas permisibles de las magnitudes de factores que en nuestro caso serían la inversión inicial, flujos de caja o bien la tasa de interés. Los cambios en los factores pueden modificar los resultados de decisión. Este análisis constituye una forma de introducir el riesgo en la selección de inversiones. Con el manejo de la sensibilidad, es posible establecer los intervalos de comportamiento aceptables para que un proyecto continúe siendo rentable.

El siguiente gráfico presenta el resultado del análisis de sensibilidad, donde es posible ver como al incrementar la tasa de interés, los valores del VAN, se van reduciendo, hasta llegar al punto en que iguala su valor a cero y deja de ser rentable, ya que, posteriormente se presentan valores negativos, lo que significa que los costos actualizados son superiores a los beneficios que, por supuesto están actualizados.

| Tasa | VAN |
|---|---|
| 5% | $ 66.629,06 |
| 10% | $ 50.111,12 |
| 15% | $ 36.691,49 |
| 20% | $ 25.653,72 |
| 25% | $ 16.472,41 |
| 30% | $ 8.756,48 |
| 35% | $ 2.210,74 |
| 40% | $ -3.390,47 |
| 45% | $ -8.221,69 |
| 50% | $ -12.419,43 |
| 55% | $ -16.091,53 |
| 60% | $ -19.323,98 |
| 65% | $ -22.186,01 |
| 70% | $ -24.733,76 |
| 75% | $ -27.013,12 |
| 80% | $ -29.061,92 |
| 85% | $ -30.911,51 |
| 90% | $ -32.588,06 |
| 95% | $ -34.113,55 |
| 100% | $ -35.506,53 |



# 10 Bibliografía

[1] Hwang, Robert Grover Brown & Patrick Y. C., *Introduction to random signlas and applied Kalman filtering*.: JOHN WILEY & SONS, 1997.

[2] Andrews, Mohinder S. Grewall y Angus P., *Kalman Filtering*.: WILEY, 2008.

[3] Xilinx. http://www.xilinx.com.

[4] Karen Parnell & Roger Bruner, "Comparing and Contrasting FPGA and Microprocessor System Design and Development," vol. WP213, 2004.

[5] University Program Xilinx. (2005) http://www.xilinx.com.

[6] A. Björck, "Solving least squares problems by orthogonalization," vol. 7, no. pp. 1–21, 1967.

[7] Bernard Widrow and István Kollár, *Quantization Noise*.: Cambridge University Press, 2008.

[8] Jetro Lauha, "The neglected art of Fixed Point arithmetic," , 2006.

[9] MathWorks. Fixed-Point ToolBoox.

[10] Volnei A. Pedroni, *Circuit Design with VHDL*.: MIT Press., 2004.

[11] Zbyněk Fedra and Jaromír Kolouch, "VHDL Procedure for Combinational Divider," 2011.

[12] Miodrag Bolic, "Fixed-point design,".

[13] Xilinx. (2009) Spartan-6 FPGA DSP48A1 Slice User Guide.

[14] QUALIS. (1995) 1164 PACKAGES QUICK REFERENCE CARD.

[15] Gang Chen and Li Guo, "The FPGA implementation of Kalman Filter," , Malta, 2005.

[16] Randy Yates. (2009) http://www.digitalsignallabs.com.

# 11 Anexos

## 11.1 Código fuente en MATLAB

### 11.1.1 KF_comp.m

```
1 function[IMP_Fact,RMS_Err,tc,RMS_trssErr,RMS_steadyErr] =
KF_comp(OverflowMode,
RoundMode,W_length,WF_length,St,con_KG,con_t,plots)
2
3 %KF different implementations comparison for a second order oscilator.
4 %F.A.Igleisas & D.Dematties - UTN-FRM 2011.
5
6 %Perform for a second order system (the code is general though):
7 %BT implementation in Double precision
8 %The kalman original algorithm in signed Fixed point
9 %The BT implementation in signed Fixed point
10 %INPUTS
11 %OverflowMode: Mode to deal with overflows
12 %RoundMode: Rounding mode
13 %W_length: Word lentgth for the fixed point implementation
14 %WF_length: Number of bits in the fractional part of the word
15 %St: Simulation time
16 %con_KG: Defines the convergence criterion, i.e the value of KG
17 %con_t: Defines the convergence criterion, i.e the time lapse that the
18 %system KG needs to be under the value of con_KG to consider that has
19 %converged.
20 %plots: If 1, plots the Estimated Xs , KGs and Ps. If 0, doesn't
21 %OUTPUTS
22 % RMS_err(d1,d2): overall rms errors for each algorithm (dimension 1,
the 4 element
is the measurement's rms_error without KF)
23 % and each state (dimension 2)
24 % tc (d1,d2): X1 convergence time for each algorithm (dimension 1)and
each state
(dimension 2)
25 % RMS_trssErr(d1,d2): rms error of the transient part for each
algorithm(dimension
1) and each state (dimension 2)
26 % the transient part is obtained from the KG of each state.
27 % RM_S_steadyErr(d1,d2): rms error of the steady part for each
algorithm(dimension
1) and each state (dimension 2)
28 % the steady state error is 0 if the filter never converges in the given
29 % simulation time
30 % PLOTS: If plots=1,the program also plots X and E_X vs t, P vs t and KG
vs t for
each
31 % algorithm and each state
32
33 %%%%%%%%%%%%%%%%Definition of the Fixed Point
arithmetic%%%%%%%%%%%%%%%%%%
34
35 Signe=1; %signed arithmetic "1".
36 OverflowM=OverflowMode;
37 RoundM=RoundMode;
38 SumM='SpecifyPrecision';
39 ProductM='SpecifyPrecision';
40
41 %Arithmetic Sum operation result parameters.
42 SumWordLength=W_length; %sum result word length.
```

```
43 SumFractionLength=WF_length; %sum result fraction length.
44
45 %Arithmetic Product operation result parameters.
46 ProductWordLength=W_length; %product result word length.
47 ProductFractionLength=WF_length; %product result fraction length.
48
49 %Arithmetic variables' parameters.
50 WordLength=W_length; %word length.
51 FractionLength=WF_length; %fraction length.
52
53 %A numerictype struct specification.
54 N=numerictype('Signed',Signe,'WordLength',WordLength,'FractionLength',
FractionLength);
55
56 %%%%%%%%%%%%%%%%Simulation variables and
parameters%%%%%%%%%%%%%%%%%%%%%%%%
57
58 n=2; %state vector dimention.
59 l=1; %mesurement vector dimention.
60 p=1; %process noise vector dimention.
61 T=0.01; %Sampling time in seconds.
62 Dt=floor(St/T+1); %Simulation discrete time.
63 M_period= 20 ; % Number of averaged samples (taken backwards from the
end time) used
to compute the end values.
64
65 Time=1:Dt;
66 Time=T*(Time-1); %Time vector in seconds (0,T,2T,...,St).
67
68 %%%%%%%%%%%%%%%%Definition of Double precision
variables%%%%%%%%%%%%%%%%%%
69 %Note the last dimention denote the discrete time variable. Used to
store
70 %the time variant quantities.
71 %The initial step is 1 corresponing to de initial conditions
72
73 X=zeros(n,Dt); %Exact State vectors matrix.
74 E_X=zeros(n,Dt); %Estimated State vectors matrix.
75 Z=zeros(l,Dt); %Measurent vectors matrix.
76 F=zeros(n,n); %Coeficients' Matrix.
77 Phi=zeros(n,n); %State transition matrix.
78 G=zeros(n,p); %Process noise coumpling matrix.
79 P=zeros(n,n,Dt); %Uncertainty covariance matrix. One matrix at each
instant.
80 H=zeros(l,n); %Measurement matrix.
81 R=zeros(l,l); %Discrete measurement noise covariance matrix.
82 V=zeros(l,Dt); %Noise vectors matrix.
83 Q=zeros(n,n); %Discrete process noise covariance matrix.
84 W=zeros(n,Dt); %Noise vectors matrix.
85 KG=zeros(n,l,Dt);%KG matrix.
86 u=zeros(n,1); %Control input (C*u).
87 Qc=zeros(p,1); %Variances of white-noise secuences of W.
88 Rc=zeros(l,1); %Variances of white-noise secuences of V.
89
90 %%%%%%Numeric Problem Definition: From example 4.4 Grewal and
Andrews%%%%%%
91
92 %Variances of white-noise secuences.
93 Rc(1,1)= 0.01; % 10%
94 Qc(1,1)= 0.02; % 14%
95
96 %Coeficients' Matrix
```

```
97 F(1,1)=0;
98 F(1,2)=1;
99 F(2,1)=-25;
100 F(2,2)=-2;
101
102 %State transition matrix.
103 Phi=expm(F*T);
104
105 %Process noise coupling matrix.
106 G(1,1)=0;
107 G(2,1)=1;
108
109 %Sensors matrix.
110 H(1,1)=1;
111 H(1,2)=0;
112
113 %System noise covariance matrix.
114 Q=T*(G*Qc*G'); % First order aproximation to obtain the discrete Q.
115
116 %Sensors noise covariance matrix.
117 R(1,1)= Rc/T;
118
119 %Control input (C*u).
120 u(1,1)=0;
121 u(2,1)=12;
122
123 %Forcing term.
124 uc = (Phi*u)*T;
125
126 %%%%%%%%%%%%%%%%%%%%%%%%%Initial
Conditions%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
127
128 %Estimation error covariance matrix.
129 P(1,1,1)=2;
130 P(2,1,1)=0;
131 P(1,2,1)=0;
132 P(2,2,1)=2;
133
134 %State vector.
135 X(1,1,1)=0;
136 X(2,1,1)=0;
137
138 %Estimated State vector.
139 E_X(:,:,1)=X(:,:,1);
140
141 %%%%%%%%%%%%%%%%%%Auxiliar variables definition
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
142
143 %NOTE: We save the a posteriori values of P and E_X
144
145 %(ic), to save the initial conditions because the algorithms are
146 %destructive
147 Pic=P;
148 E_Xic=E_X;
149
150 %To store the Double and Fixed precision measurements
151 ZDP=Z;
152 ZFP=Z;
153
154 %To store the BT Algorithm: Double Precision results
155 PbtDP=P;
156 E_XbtDP=E_X;
```

```
157 KGbtDP=KG;
158
159 %To store the Original Algorithm: Fixed Point results
160 PoaFP=P;
161 E_XoaFP=E_X;
162 KGoaFP=KG;
163
164 %To store the BT Algorithm: Fixed Point results
165 PbtFP=P;
166 E_XbtFP=E_X;
167 KGbtFP=KG;
168
169 %%%%%%%%%%%%%Measurements generation: Double
Precision%%%%%%%%%%%%%%%%%%%%%%
170
171 for i=1:Dt,
172 V(:,i)=normrnd(zeros(l,1),sqrt(diag(R)));
173 end
174 for i=1:Dt,
175 W(:,i)=normrnd(zeros(n,1),sqrt(diag(Q)));
176 end
177
178 %Noisy measurement generation.
179 for k=1:Dt-1,
180 X(:,k+1)=Phi*X(:,k)+W(:,k)+uc;
181 Z(:,k)=H*X(:,k)+V(:,k);
182 end
183 Z(:,Dt)=H*X(:,Dt)+V(:,Dt);
184 %In the last iteration, X(:,k+1) does not need to be computed.
185
186 %Exact States generation.
187 for k=1:Dt-1,
188 X(:,k+1)=Phi*X(:,k)+uc;
189 end
190
191 %Save the double precision Z for future plots
192 ZDP=Z ;
193
194 %%%%%%%%%%%%%KF Bierman-Thornton algorithm: Double
precision%%%%%%%%%%%%%
195
196 %Defines the extra matrices for the BT algorithm.
197 % P=U*D*U'
198 U=zeros(n,n,Dt);
199 Uv=zeros((n^2-n)/2,Dt);
200 D=zeros(n,n,Dt);
201 %Q=G*Dq*G' Where G*Qc*G'=G*U*Dq*U'*G'. So:
202 %Dq can come from a Qc's UD factorization and G=G*U if necesary.
203 Dq=zeros(n,n);
204 G=zeros(n,n);
205 PhiU=zeros(n,n);
206 %Generates the initials U and D from P by UD factorization.
207 for j=n:-1:1,
208 for i=j:-1:1,
209 alfa=P(i,j,1);
210 for s=j+1:n,
211 alfa=alfa-U(i,s,1)*D(s,s,1)*U(j,s,1);
212 end
213 if i==j
214 D(j,j,1)=alfa;
215 U(j,j,1)=1;
216 else
```

```
217 U(i,j,1)=alfa/D(j,j,1);
218 end
219 end
220 end
221
222 %Transfers the U matrix's upper triangle to the Uv vector. To work like
the vhdl
code
223 for j=1:n
224 for i=1:n
225 if j>i
226 Uv(j-n+(i*(2*n-i-1))/2,1)=U(i,j,1);
227 end
228 end
229 end
230
231 %Bierman-Thornton algorithm
232 for k=1:Dt,
233
234 for r=1:l, %Sequential measurement processing loop
235
236 %Bierman observational update:
237 %Based on "bierman.m" by Grewal and Andrews
238 dz = Z(r,k);
239 for j=1:n
240 s_aux = H(r,j)*E_X(j,k) ;
241 dz = dz - s_aux ; % computes Z-H*X
242 a(j) = H(r,j);
243 i = 1;
244 while 1 % equivalent to: for in 1 to (j-1) (iterates j-1 times)
245 if i == j
246 break;
247 end
248 index_aux = j-n+(i*(2*n-i-1))/2 ; % to find the element Uij inside
the U vector
249 s_aux1 = Uv(index_aux,k)*H(r,i) ;
250 a(j) = a(j) + s_aux1; % computes V=U'*H'
251 i = i+1 ;
252 end
253 end
254
255 for j=1:n
256 b(j) = D(j,j,k)*a(j); % computes the b factor in the Grewall's Bierman
matlab algorithm (called the unescaled kalman gain)
257 end
258
259 alpha=R(r,r);
260 gamma=1/alpha;
261
262 for j=1:n, %Within these two for, all the operations are scalar
263 beta=alpha;
264 s_aux= a(j)*b(j);
265 alpha=alpha +s_aux;
266 lambda= -(a(j)*gamma);
267 gamma =1/alpha;
268 D(j,j,k)=beta*(gamma*D(j,j,k));
269 i=1;
270 while 1
271 if i == j
272 break;
273 end
274 index_aux = j-n+(i*(2*n-i-1))/2 ; % to find the element Uij inside
```

```
the U vector
beta=Uv(index_aux,k);
s_aux=b(i)*lambda;
Uv(index_aux,k)=beta + s_aux;
s_aux=b(j)*beta;
b(i)=b(i) + s_aux;
i=i+1;
end
end

for j=1:n
aux =(gamma*dz)*b(j);
E_X(j,k)=E_X(j,k)+aux;
end

%Save the KG just to plot it !!!
KG(:,r,k)=gamma*b;
end

% Restores the U matrix from Uv vector to compute the a posteriori P
% in a matricial way
for j=1:n
for i=1:n
if j>i
U(i,j,k)=Uv(j-n+(i*(2*n-i-1))/2,k);
end
end
end

%Save the a posteriori P just to plot it !!!
P(:,:,k)=U(:,:,k)*D(:,:,k)*U(:,:,k)';


%Temporal update, not computed in the last iteration
if k<Dt

%Thornton temporal update: Based on "thornton.m" by Grewal and Andrews.
%PhiU = Phi*U(:,:,k);
G_aux=G; %To avoid G destruction
PhiU(:,:) = 0;
for i=1:n,
for j=1:n,
r=1;
while 1
if r==j % equivalent to: for r=1 to (j-1) but vhdl
implementable
break;
end
index_aux = j-n+(r*(2*n-r-1))/2;
PhiU(i,j)= PhiU(i,j)+ Phi(i,r)*Uv(index_aux,k);
r=r+1;
end
PhiU(i,j)=PhiU(i,j)+Phi(i,j);
end
end

for i=n:-1:1,
sigma = 0;
for j=1:n,
sigma = sigma + PhiU(i,j)*(PhiU(i,j)*D(j,j,k)) ;
if (j <= p)
sigma = sigma + G_aux(i,j)*(G_aux(i,j)*Dq(j,j));
```

```
335 end
336 end
337 D(i,i,k+1) = sigma;
338
339 j=1;
340 while 1 % equivalent to: for j=1 to (i-1) but vhdl implementable
341 if j == i
342 break;
343 end
344 sigma = 0;
345 for s=1:n
346 sigma = sigma + PhiU(i,s)*(D(s,s,k)*PhiU(j,s));
347 end
348
349 for s=1:p,
350 sigma = sigma + G_aux(i,s)*(Dq(s,s)*G_aux(j,s));
351 end
352 index_aux = i-n+(j*(2*n-j-1))/2 ; % to find the element Uji (note
the changed index in this case) inside the U vector
353 Uv(index_aux,k+1) =sigma/D(i,i,k+1) ;
354 for s=1:n,
355 PhiU(j,s) = PhiU(j,s) - Uv(index_aux,k+1)*PhiU(i,s);
356 end
357 for s=1:p,
358 G_aux(j,s) = G_aux(j,s) - Uv(index_aux,k+1)*G_aux(i,s);
359 end
360 j = j+1;
361 end
362 end
363
364 %State estimation temporal update
365 E_X(:,k+1)= 0 ;
366 for i=1:n,
367 for r=1:n,
368 E_X(i,k+1)=E_X(i,k+1)+Phi(i,r)*E_X(r,k);
369 end
370 end
371 E_X(:,k+1)=E_X(:,k+1)+uc;
372
373 end
374 end
375
376 %Stores the results for future plots
377 PbtDP=P;
378 E_XbtDP=E_X;
379 KGbtDP=KG;
380
381 %%%%%%%%%%%%%%%%%%%%%%%%Definition of fixed point
variables%%%%%%%%%%%%%%%%%%
382 %NOTE: Takes the numeric values from the double precision definitions
383
384 %Time variant matrices and vectors
385 E_X=fi(E_Xic,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
386 P=fi(Pic,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
```

```
387
KG=fi(zeros(n,l,Dt),N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode'
,
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
388
389 %Forcing Term
390 uc=fi(uc,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
391
392 %Measurents vector matrix.
393 Z=fi(ZDP,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
394 %ZFP=Z; % store de fixed point Z for future plots
395
396 %Time invariant system's matrices.
397 Phi=fi(Phi,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
398
H=fi(H,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',SumM,'SumWor
dLength',
SumWordLength,'SumFractionLength',SumFractionLength,'ProductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
399
Q=fi(Q,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',SumM,'SumWor
dLength',
SumWordLength,'SumFractionLength',SumFractionLength,'ProductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
400
R=fi(R,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',SumM,'SumWor
dLength',
SumWordLength,'SumFractionLength',SumFractionLength,'ProductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
401
402 %Auxiliary matrices.
403 I_ll=fi(eye(l),N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
404 I_nn=fi(eye(n),N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
405
Aux_ll=fi(zeros(l,l),N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode
',
```

```
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
406
Aux_nn=fi(zeros(n,n),N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode
',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
407
408 %Auxiliary scalars.
409 Uno=fi(1,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
410
411 %Saves the fixed point initial conditions.
412 E_Xic=E_X;
413 Pic=P;
414
415 %%%%%%%%%%%%%%%%%%KF, original algorithm: Fixed
Point%%%%%%%%%%%%%%%%%%%%%%
416
417 for k=1:Dt,
418
419 %Observational update
420 Aux_ll=divide(N,I_ll,(H*P(:,:,k)*H'+R));
421 KG(:,:,k)=P(:,:,k)*H'*Aux_ll;
422 E_X(:,k)=E_X(:,k)+KG(:,:,k)*(Z(:,k)-H*E_X(:,k));
423 P(:,:,k)=P(:,:,k)-KG(:,:,k)*H*P(:,:,k);
424 P(:,:,k)=0.5*(P(:,:,k)+P(:,:,k)'); %Force simetry
425
426 %%Temporal update, not computed in the last iteration
427 if k<Dt
428 P(:,:,k+1)=Phi*P(:,:,k)*Phi'+Q;
429 E_X(:,k+1)=Phi*E_X(:,k)+uc;
430 end
431
432 end
433
434 %Stores the results for future plots
435 PoaFP=P;
436 E_XoaFP=E_X;
437 KGoaFP=KG;
438
439 %%%%%%%%%%KF, Bierman-Thornton algorithm: Fixed
Point%%%%%%%%%%%%%%%%%%%%%%
440
441 %Defines the extra matrices for the BT algorithm.
442
443 % P=U*D*U'
444
U=fi(zeros(n,n,Dt),N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
445 Uv=fi(zeros((n^2-
n)/2,Dt),N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
```

```
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
446
D=fi(zeros(n,n,Dt),N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
447 %Q=G*Dq*G' Where G*Qc*G'=G*U*Dq*U'*G'. So:
448 %Dq can come from a Qc's UD factorization and G=G*U if necesary.
449
G=fi(zeros(n,n),N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
450
Dq=fi(zeros(n,n,Dt),N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode'
,
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
451
PhiU=fi(zeros(n,n),N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',SumFractionLength,'P
roductMode',
ProductM,'ProductWordLength',ProductWordLength,'ProductFractionLength',
ProductFractionLength);
452
453 %Recovers the initial conditions.
454 E_X=E_Xic;
455 P=Pic;
456
457 %Generates the initials U and D from P by UD factorization.
458 for j=n:-1:1,
459 for i=j:-1:1,
460 alfa=P(i,j,1);
461 for s=j+1:n,
462 alfa=alfa-U(i,s,1)*D(s,s,1)*U(j,s,1);
463 end
464 if i==j
465 D(j,j,1)=alfa;
466 U(j,j,1)=1;
467 else
468 U(i,j,1)=divide(N,alfa,D(j,j,1));
469 end
470 end
471 end
472
473 %Transfers the U matrix's upper triangle to the Uv vector. To work like
the vhdl
code
474 for j=1:n
475 for i=1:n
476 if j>i
477 Uv(j-n+(i*(2*n-i-1))/2,1)=U(i,j,1);
478 end
479 end
480 end
```

```
481
482 %Bierman-Thornton algorithm
483 for k=1:Dt,
484
485 for r=1:l, %Sequential measurement processing loop
486
487 %Bierman observational update:
488 %Based on "bierman.m" by Grewal and Andrews
489 dz = Z(r,k);
490 for j=1:n
491 s_aux = H(r,j)*E_X(j,k) ;
492 dz = dz - s_aux ; % computes Z-H*X
493 a(j) = H(r,j);
494 i = 1;
495 while 1 % equivalent to: for in 1 to (j-1) (iterates j-1 times)
496 if i == j
497 break;
498 end
499 index_aux = j-n+(i*(2*n-i-1))/2 ; % to find the element Uij inside
the U vector
500 s_aux1 = Uv(index_aux,k)*H(r,i) ;
501 a(j) = a(j) + s_aux1; % computes V=U'*H'
502 i = i+1 ;
503 end
504 end
505
506 for j=1:n
507 b(j) = D(j,j,k)*a(j); % computes the b factor in the Grewall's Bierman
matlab algorithm (called the unescaled kalman gain)
508 end
509
510 alpha=R(r,r);
511 gamma=divide(N,Uno,alpha);
512
513 for j=1:n, %Within these two for, all the operations are scalar
514 beta=alpha;
515 s_aux= a(j)*b(j);
516 alpha=alpha +s_aux;
517 lambda= -(a(j)*gamma);
518 gamma =divide(N,Uno,alpha);
519 D(j,j,k)=beta*(gamma*D(j,j,k));
520 i=1;
521 while 1
522 if i == j
523 break;
524 end
525 index_aux = j-n+(i*(2*n-i-1))/2 ; % to find the element Uij inside
the U vector
526 beta=Uv(index_aux,k);
527 s_aux=b(i)*lambda;
528 Uv(index_aux,k)=beta + s_aux;
529 s_aux=b(j)*beta;
530 b(i)=b(i) + s_aux;
531 i=i+1;
532 end
533 end
534
535 for j=1:n
536 aux =(gamma*dz)*b(j);
537 E_X(j,k)=E_X(j,k)+aux;
538 end
539
```

```
%Save the KG just to plot it !!!
KG(:,r,k)=gamma*b;
end

% Restores the U matrix from Uv vector to compute the a posteriori P
% in a matricial way
for j=1:n
for i=1:n
if j>i
U(i,j,k)=Uv(j-n+(i*(2*n-i-1))/2,k);
end
end
end

%Save the a posteriori P just to plot it !!!
P(:,:,k)=U(:,:,k)*D(:,:,k)*U(:,:,k)';

%Temporal update, not computed in the last iteration
if k<Dt

%Thornton temporal update: Based on "thornton.m" by Grewal and Andrews.
%PhiU = Phi*U(:,:,k);
G_aux=G; %To avoid G destruction
PhiU(:,:) = 0;
for i=1:n,
for j=1:n,
r=1;
while 1
if r==j % equivalent to: for r=1 to (j-1) but vhdl
implementable
break;
end
index_aux = j-n+(r*(2*n-r-1))/2;
PhiU(i,j)= PhiU(i,j)+ Phi(i,r)*Uv(index_aux,k);
r=r+1;
end
PhiU(i,j)=PhiU(i,j)+Phi(i,j);
end
end

for i=n:-1:1,
sigma = 0;
for j=1:n,
sigma = sigma + PhiU(i,j)*(PhiU(i,j)*D(j,j,k)) ;
if (j <= p)
sigma = sigma + G_aux(i,j)*(G_aux(i,j)*Dq(j,j));
end
end
D(i,i,k+1) = sigma;

j=1;
while 1 % equivalent to: for j=1 to (i-1) but vhdl implementable
if j == i
break;
end
sigma = 0;
for s=1:n
sigma = sigma + PhiU(i,s)*(D(s,s,k)*PhiU(j,s));
end

for s=1:p,
sigma = sigma + G_aux(i,s)*(Dq(s,s)*G_aux(j,s));
```

```
601 end
602 index_aux = i-n+(j*(2*n-j-1))/2 ; % to find the element Uji (note
the changed index in this case) inside the U vector
603 Uv(index_aux,k+1) = divide(N,Uno,D(i,i,k+1))*sigma ;
604 for s=1:n,
605 PhiU(j,s) = PhiU(j,s) - Uv(index_aux,k+1)*PhiU(i,s);
606 end
607 for s=1:p,
608 G_aux(j,s) = G_aux(j,s) - Uv(index_aux,k+1)*G_aux(i,s);
609 end
610 j = j+1;
611 end
612 end
613
614 %State estimation temporal update
615 E_X(:,k+1)= 0 ;
616 for i=1:n,
617 for r=1:n,
618 E_X(i,k+1)=E_X(i,k+1)+Phi(i,r)*E_X(r,k);
619 end
620 end
621 E_X(:,k+1)=E_X(:,k+1)+uc;
622
623 end
624 end
625
626 %Stores the results for future plots
627 PbtFP=P;
628 E_XbtFP=E_X;
629 KGbtFP=KG;
630
631
632 %%%%%%%%%%%%%%%%%%Plot, Error and Convergence time
calculation%%%%%%%%%%%%%%%%%
633
634 Kz=1; %To scale the value of z in the plots. (Just a visualization
matter)
635
636 %%%%%%%%%%%%%%%%%%%%%%X and E_X vs time%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
637 if plots==1
638 figure('Name','X (blue) & E_X(Green) vs time for btDP (1 Row), the aoFP
(2Row)
and btFP (3Row)','NumberTitle','off');
639 end
640 for i=1:n
641 %%%%%%%% for BT DP
642
643 RMS_V=0; % computes the estimation RMS error
644 ERRbtDP = X - E_XbtDP ;
645 for k=1:Dt
646 RMS_V = RMS_V + ERRbtDP(i,k)*ERRbtDP(i,k);
647 end
648
649 RMS_V = sqrt(RMS_V/Dt);
650 RMS_Err(1,i)= RMS_V;
651
652 %%%%%%%% for OA FP
653
654 RMS_V=0; % computes the estimation RMS error
655 ERRoaFP = X - double(E_XoaFP) ;
656 for k=1:Dt
657 RMS_V=RMS_V + double(ERRoaFP(i,k))*double(ERRoaFP(i,k));
```

```
658 end
659 RMS_V = sqrt(RMS_V/Dt);
660 RMS_Err(2,i)= RMS_V;
661
662 %%%%%%%% for BT FP
663
664 RMS_V=0; % computes the estimation RMS error
665 ERRbtFP = X - double(E_XbtFP) ;
666 for k=1:Dt
667 RMS_V=RMS_V + double(ERRbtFP(i,k))*double(ERRbtFP(i,k));
668 end
669 RMS_V = sqrt(RMS_V/Dt);
670 RMS_Err(3,i)= RMS_V;
671
672 %%%%%%%% for System's measurement
673
674 RMS_V=0; % computes the estimation RMS error
675 ERRmeaure(i,:) = X(i,:) - ZDP ;
676 for k=1:Dt
677 RMS_V = RMS_V + ERRmeaure(i,k)*ERRmeaure(i,k);
678 end
679
680 RMS_V = sqrt(RMS_V/Dt);
681 RMS_Err(4,i)= RMS_V;
682
683 %Computes the Improvement factor.
684 if i<l+1
685 for k=1:3
686 IMP_Fact(k,i)=RMS_Err(4,i)/RMS_Err(k,i);
687 end
688 end
689
690
691 %%%%%%% Plotsssss
692 if plots==1
693 if i<l+1
694
695 subplot(3,n,i),plot(Time,X(i,:),Time,E_XbtDP
(i,:),'Marker','*','MarkerSize',3),xlabel(['KF-BT-DP rmsErr:
',num2str(RMS_Err(1,i))]),
ylabel(['Improvement factor: ',num2str(IMP_Fact(1,i))])
696 hold
697 plot(Time,Kz*Z(i,:),'c+')
698
699 subplot(3,n,i+n),plot(Time,X(i,:),Time,E_XoaFP
(i,:),'Marker','*','MarkerSize',3),xlabel(['KF-OA-FP rmsErr:
',num2str(RMS_Err(2,i))]),
ylabel(['Improvement factor: ',num2str(IMP_Fact(2,i))])
700 hold
701 plot(Time,Kz*Z(i,:),'c+')
702
703 subplot(3,n,i+2*n),plot(Time,X(i,:),Time,E_XbtFP
(i,:),'Marker','*','MarkerSize',3),xlabel(['KF-BT-FP rmsErr:
',num2str(RMS_Err(3,i))]),
ylabel(['Improvement factor: ',num2str(IMP_Fact(3,i))])
704 hold
705 plot(Time,Kz*Z(i,:),'c+')
706 else
707 subplot(3,n,i),plot(Time,X(i,:),Time,E_XbtDP
(i,:),'Marker','*','MarkerSize',3),xlabel(['KF-BT-DP rmsErr:
',num2str(RMS_Err(1,i))])
708 subplot(3,n,i+n),plot(Time,X(i,:),Time,E_XoaFP
```

```
(i,:),'Marker','*','MarkerSize',3),xlabel(['KF-OA-FP rmsErr:
',num2str(RMS_Err(2,i))])
709 subplot(3,n,i+2*n),plot(Time,X(i,:),Time,E_XbtFP
(i,:),'Marker','*','MarkerSize',3),xlabel(['KF-BT-FP rmsErr:
',num2str(RMS_Err(3,i))])
710 end
711 end
712 end
713
714 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%P vs time%%%%%%%%%%%%%%%%%%%%%%%%%%
715 if plots==1
716 figure('Name','P vs time for btDP (1 Row), the aoFP (2Row) and btFP
(3Row)','NumberTitle','off');
717 end
718 if plots==1
719 for i=1:n
720 for k=1:Dt
721 Paux2(k)=PbtDP(i,i,k);
722 end
723
subplot(3,n,i),plot(Time,Paux2,'Marker','*','MarkerSize',3),xlabel(['PbtDP'
,
num2str(i)])
724 for k=1:Dt
725 Paux2(k)=PoaFP(i,i,k);
726 end
727 subplot(3,n,i+n),plot(Time,Paux2,'Marker','*','MarkerSize',3),xlabel
(['PoaFP',num2str(i)])
728 for k=1:Dt
729 Paux2(k)=PbtFP(i,i,k);
730 end
731 subplot(3,n,i+2*n),plot(Time,Paux2,'Marker','*','MarkerSize',3),xlabel
(['PbtFP',num2str(i)])
732 end
733 end
734 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%KG vs time%%%%%%%%%%%%%%%%%%%%%%%%%%
735 if plots==1
736 figure('Name','KG vs time for btDP (1 Row), the aoFP (2Row) and btFP
(3Row)','NumberTitle','off');
737 end
738 for i=1:n
739 %%%%%%%% for BT DP
740 for k=1:Dt % The most representative KG for Xn state is the norm of n
row
741 KGaux2(k)=norm(KGbtDP(i,:,k));
742 end
743
744 MS_V=0; % computes the end values of KG
745 for k=Dt:-1:Dt-M_period
746 MS_V= MS_V + double(KGaux2(k))*double(KGaux2(k));
747 end
748 MS_V=MS_V/(M_period);
749 MS_V=sqrt(MS_V);
750 KGend(1,i)=MS_V;
751
752 cont=0;
753 tc_aux =0;% computes the convergence time
754 for k=1:Dt
755 if (KGaux2(k)< con_KG)
756 if tc_aux == 0
757 tc_aux=k*T;
758 else
```

```
759 cont=cont+1 ;
760 if cont == con_t*T;
761 break ;
762 end
763 end
764 else
765 tc_aux=0;
766 cont=0;
767 end
768 end
769 if tc_aux == T
770 tc_aux = 0 ; % for the KG allwas =0 case
771 end
772 tc(1,i)=tc_aux ;
773
774 tc_discrete= floor(tc_aux/T);
775 RMS_V=0; % computes the estimation RMS error of the transient part
776 if tc_discrete == 0 % When the KG doesn't converge tc_aux=0
777 tc_discrete= Dt;
778 end
779 for k=1:(tc_discrete)
780 RMS_V=RMS_V + ERRbtDP(i,k)*ERRbtDP(i,k);
781 end
782 RMS_V = sqrt(RMS_V/(tc_discrete));
783 RMS_trssErr(1,i)= RMS_V;
784
785 RMS_V=0; % computes the estimation RMS error of the permament part
786 for k=(tc_discrete) :Dt
787 RMS_V=RMS_V + ERRbtDP(i,k)*ERRbtDP(i,k);
788 end
789 RMS_V = sqrt(RMS_V/(Dt-tc_discrete));
790 RMS_steadyErr(1,i)= RMS_V;
791
792 if plots==1
793
subplot(3,n,i),plot(Time,KGaux2,'Marker','*','MarkerSize',3),xlabel(['End
value : ',num2str( KGend(1,i))]),ylabel(['ct KF-BT-DP: ',num2str(
tc(1,i))])
794 end
795 %%%%%%%% for OA FP
796 for k=1:Dt % computes the norm for Fixed point values
797 KGaux2(k)=0;
798 for j=1:l
799 KGaux2(k)= KGaux2(k)+ double(KGoaFP(i,j,k))*double(KGoaFP(i,j,k));
800 end
801 KGaux2(k)=sqrt(KGaux2(k));
802 end
803
804 MS_V=0; % computes the end values of KG
805 for k=Dt:-1:Dt-M_period
806 MS_V=MS_V+double(KGaux2(k))*double(KGaux2(k));
807 end
808 MS_V=MS_V/(M_period);
809 MS_V=sqrt(MS_V);
810 KGend(2,i)=MS_V;
811
812 cont=0;
813 tc_aux =0;% computes the convergence time
814 for k=1:Dt
815 if (double(KGaux2(k))< con_KG)
816 if tc_aux == 0
817 tc_aux=k*T;
```

```
818 else
819 cont=cont+1 ;
820 if cont == con_t*T;
821 break ;
822 end
823 end
824 else
825 tc_aux=0;
826 cont=0;
827 end
828 end
829 if tc_aux == T
830 tc_aux = 0 ; % for the KG allwas =0 case
831 end
832 tc(2,i)=tc_aux ;
833
834 tc_discrete= floor(tc_aux/T);
835 RMS_V=0; % computes the estimation RMS error of the transient part
836 if tc_discrete == 0 % When the KG doesn't converge tc_aux=0
837 tc_discrete= Dt;
838 end
839 for k=1:tc_discrete
840 RMS_V=RMS_V + double(ERRoaFP(i,k))*double(ERRoaFP(i,k));
841 end
842 RMS_V = sqrt(RMS_V/(tc_discrete));
843 RMS_trssErr(2,i)= RMS_V;
844
845 RMS_V=0; % computes the estimation RMS error of the permament part
846 for k=tc_discrete :Dt
847 RMS_V=RMS_V + double(ERRoaFP(i,k))*double(ERRoaFP(i,k));
848 end
849 RMS_V = sqrt(RMS_V/(Dt-tc_discrete));
850 RMS_steadyErr(2,i)= RMS_V;
851
852
853 if plots==1
854
subplot(3,n,i+n),plot(Time,KGaux2,'Marker','*','MarkerSize',3),xlabel(['End
value: ',num2str(KGend(2,i))]),ylabel(['ct KF-OA-FP: ',num2str( tc(2,i))])
855 end
856 %%%%%%%% for BT FP
857 for k=1:Dt % computes the norm for Fixed point values
858 KGaux2(k)=0;
859 for j=1:l
860 KGaux2(k)= KGaux2(k)+double(KGbtFP(i,j,k))*double(KGbtFP(i,j,k));
861 end
862 KGaux2(k)=sqrt(KGaux2(k));
863 end
864
865 MS_V=0; % computes the end values of KG
866 for k=Dt:-1:Dt-M_period
867 MS_V=MS_V+double(KGaux2(k))*double(KGaux2(k));
868 end
869 MS_V=MS_V/(M_period);
870 MS_V=sqrt(MS_V);
871 KGend(3,i)=MS_V;
872
873
874 cont=0;
875 tc_aux =0; % computes the convergence time, tc=0 -> not convergece
876 for k=1:Dt
877 if (double(KGaux2(k))< con_KG)
```

```
878 if tc_aux == 0
879 tc_aux=k*T;
880 else
881 cont=cont+1 ;
882 if cont == con_t*T;
883 break ;
884 end
885 end
886 else
887 tc_aux=0;
888 cont=0;
889 end
890 end
891 if tc_aux == T
892 tc_aux = 0 ; % for the KG allways =0 case
893 end
894 tc(3,i)=tc_aux ;
895
896 tc_discrete = floor(tc_aux/T);
897 RMS_V=0; % computes the estimation RMS error of the transient part
898 if tc_discrete == 0 % When the KG doesn't converge tc_aux=0
899 tc_discrete= Dt;
900 end
901 for k=1:(tc_discrete)
902 RMS_V=RMS_V + double(ERRbtFP(i,k))*double(ERRbtFP(i,k));
903 end
904 RMS_V = sqrt(RMS_V/(tc_discrete));
905 RMS_trssErr(3,i)= RMS_V;
906
907
908 RMS_V=0; % computes the estimation RMS error of the permament part
909 for k=tc_discrete :Dt
910 RMS_V=RMS_V + double(ERRbtFP(i,k))*double(ERRbtFP(i,k));
911 end
912 RMS_V = sqrt(RMS_V/(Dt-tc_discrete));
913 RMS_steadyErr(3,i) = RMS_V;
914
915 if plots==1
916 subplot(3,n,i+2*n),plot(Time,KGaux2,'Marker','*','MarkerSize',3),xlabel
([‘End value: ',num2str(KGend(3,i))]),ylabel(['ct KF-BT-FP: ',num2str(
tc(3,i))])
917 end
918
919 end
```

### 11.1.2 single_implementation.m

```
1 %Compration of 3 diferent implementations of the KF: KF-BT-DP ,KF-OA-FP,
2 %KF-BT-FP.
3 %
4 %Authors: F.A.Igleisas & D.Dematties - UTN-FRM 2011 - Argentina.
5 %franciscoiglesias@frm.utn.edu.ar ; dariodematties@yahoo.com.ar
6
7 %The programs plots for each algorithm:
8 % -The rms Err of the estimation
9 % -The Improvement factors
10 % -The convergence times
11 %The values ploted are average over several number of realizations and
are
12 %shown as function of FL (The fraction part's lenght of the working
word)
13
```

```
%Requires function KF_comp

clf
clear all;
close all;

n=2; %States number for the system implemented inside KF_comp function
l=1; %Measurement vextor dimention.
real=10; % Number of realizations to average the rmsErr and ct
WordLength = 18;
Overflow_Mode = 'saturate';
Round_Mode = 'floor';
con_KG = 0.005 ; % Defines the convergence criterion: This is the end KG
value
con_t = 1 ; % Defines the convergence criterion: time lapse with
KG<con_KG to
consider convergence
simulationTime = 10 ; % Simulation time in seconds.
Plot = 0 ;

RMS_Err = zeros(4,n,WordLength-1,4); % to save the overall rms error
RMS_Err_aux = RMS_Err ;
IMP_Fact= zeros(3,l,WordLength-1);
IMP_Fact_aux=IMP_Fact;

RMS_trss_Err_aux= zeros(3,n,WordLength-1);
RMS_steady_Err_aux= zeros(3,n,WordLength-1);
tc = zeros (3,n,WordLength-1); % to save the convergence time for each
point
position and each state
% NOTE: The convergence time doesn't change with the realizations, so we
don't
compute an average value
FL = 1:WordLength-1 ; % generates the x axis for the plots

warning ('off', 'MATLAB:divideByZero') ;

for i=1 : WordLength-1
for k=1 : real
[IMP_Fact_aux(:,:,i),RMS_Err_aux(:,:,i),tc(:,:,i),RMS_trss_Err_aux
(:,:,i),RMS_steady_Err_aux(:,:,i)] =
KF_comp(Overflow_Mode,Round_Mode,WordLength,i,
simulationTime,con_KG,con_t,Plot);
%Average RMS_Err
RMS_Err(:,:,i)= RMS_Err(:,:,i) + RMS_Err_aux(:,:,i);
%Average Improvement factor
IMP_Fact(:,:,i)= IMP_Fact(:,:,i) + IMP_Fact_aux(:,:,i);
end
RMS_Err(:,:,i) = RMS_Err(:,:,i)/real ;
IMP_Fact(:,:,i)= IMP_Fact(:,:,i)/real ;
end


%%%%%Convergence time

figure('Name','mean ct vs FL for btDP (1 Row), the aoFP (2 Row) and btFP
(3
Row)','NumberTitle','off');

for i=1:n
for k=1 : WordLength-1
plot1(k) = tc(1,i,k);
```

```
64 end
65 subplot(3,n,i),plot(FL,plot1,'Marker','*','MarkerSize',3),xlabel
('Realizations'),ylabel(['ct_',num2str(i),'(KF-BT-DP)[s]'])
66 for k=1 : WordLength-1
67 plot1(k) = tc(2,i,k);
68 end
69 subplot(3,n,i+n),plot(FL,plot1,'Marker','*','MarkerSize',3),
xlabel('FL[bit]'),ylabel(['ct_',num2str(i),'(KF-OA-FP)[s]'])
70 for k=1 : WordLength-1
71 plot1(k) = tc(3,i,k);
72 end
73 subplot(3,n,i+2*n),plot(FL,plot1,'Marker','*','MarkerSize',3),
xlabel('FL[bit]'),ylabel(['ct_',num2str(i),'(KF-BT-FP)[s]'])
74 end
75
76 %%%%%RMS error
77
78 figure('Name','mean RMS error vs FL for btDP (1 Row), the aoFP (2 Row)
and btFP
(3 Row)','NumberTitle','off');
79
80 for i=1:n
81 for k=1 : WordLength-1
82 plot1(k) = RMS_Err(1,i,k);
83 end
84 subplot(3,n,i),plot(FL,plot1,'Marker','*','MarkerSize',3),xlabel
('Realizations'),ylabel(['RMS Err_',num2str(i),'(KF-BT-DP)'])
85 for k=1 : WordLength-1
86 plot1(k) = RMS_Err(2,i,k);
87 end
88 subplot(3,n,i+n),semilogy(FL,plot1,'Marker','*','MarkerSize',3),
xlabel('FL[bit]'),ylabel(['RMS Err_',num2str(i),'(KF-OA-FP)'])
89 for k=1 : WordLength-1
90 plot1(k) = RMS_Err(3,i,k);
91 end
92 subplot(3,n,i+2*n),plot(FL,plot1,'Marker','*','MarkerSize',3),
xlabel('FL[bit]'),ylabel(['RMS Err_',num2str(i),'(KF-BT-FP)'])
93 end
94
95 %%%%%Improvement factor
96
97 figure('Name','mean Improvement factor vs FL for btDP (1 Row), the aoFP
(2 Row)
and btFP (3 Row)','NumberTitle','off');
98
99 for i=1:l
100 for k=1 : WordLength-1
101 plot1(k) = IMP_Fact(1,i,k);
102 end
103 subplot(3,l,i),semilogy(FL,plot1,'Marker','*','MarkerSize',3),
xlabel('Realizations'),ylabel(['IMP Fact_',num2str(i),'(KF-BT-DP)'])
104 for k=1 : WordLength-1
105 plot1(k) = IMP_Fact(2,i,k);
106 end
107 subplot(3,l,i+l),semilogy(FL,plot1,'Marker','*','MarkerSize',3),
xlabel('FL[bit]'),ylabel(['IMP Fact_',num2str(i),'(KF-OA-FP)'])
108 for k=1 : WordLength-1
109 plot1(k) = IMP_Fact(3,i,k);
110 end
111 subplot(3,l,i+2*l),semilogy(FL,plot1,'Marker','*','MarkerSize',
3),xlabel('FL[bit]'),ylabel(['IMP Fact_',num2str(i),'(KF-BT-FP)'])
112 end
```

113

### 11.1.3 Comparison.m

```
1 %Compration of 3 diferent implementations of the KF: KF-BT-DP ,KF-OA-FP,
2 %KF-BT-FP.
3 %Authors: F.A.Igleisas & D.Dematties - UTN-FRM 2011 - Argentina.
4 %franciscoiglesias@frm.utn.edu.ar ; dariodematties@yahoo.com.ar
5
6 %The programs plots:
7 % -The rms Err quotients: KF-BT-DP/KF-BT-FP and KF-BT-FP/KF-OA-FP
8 % -The Improvement factors quotients: KF-BT-DP/KF-BT-FP and KF-BTFP/
KF-OA-FP
9 % -The convergence times quotient: KF-BT-DP/KF-BT-FP and KF-BTFP/
KF-OA-FP
10
11 %The values ploted are average over several number of realizations and
are
12 %shown as function of FL (The fraction part's lenght of the working
word)
13
14 %Requires function KF_comp
15
16 clf
17 clear all;
18 close all;
19
20 n=2; %States number for the system implemented inside KF_comp function
21 l=1; %Measurement vextor dimention.
22 real=10; % Number of realizations to average the rmsErr and ct
23 WordLength = 18;
24 Overflow_Mode = 'saturate';
25 Round_Mode = 'floor';
26 con_KG = 0.005 ; % Defines the convergence criterion: This is the end KG
value
27 con_t = 1 ; % Defines the convergence criterion: time lapse with
KG<con_KG to
consider convergence
28 simulationTime = 10 ; % Simulation time in seconds.
29 Plot = 0 ;
30
31 RMS_Err = zeros(4,n,WordLength-1); % to save the overall rms error
32 RMS_Err_aux = RMS_Err ;
33 IMP_Fact= zeros(3,l,WordLength-1);
34 IMP_Fact_aux=IMP_Fact;
35 RMS_trss_Err= zeros(3,n,WordLength-1);
36 RMS_trss_Err_aux= RMS_trss_Err;
37 RMS_steady_Err= zeros(3,n,WordLength-1);
38 RMS_steady_Err_aux= RMS_steady_Err;
39 tc = zeros (3,n,WordLength-1); % to save the convergence time for each
point
position and each state
40 % NOTE: The convergence time doesn't change with the realizations, so we
don't
compute an average value
41 FL = 1:WordLength-1 ; % generates the x axis for the plots
42 One=ones(1,WordLength-1);
43
44 warning ('off', 'MATLAB:divideByZero') ;
45
46 for i=1 : WordLength-1
47 for k=1 : real
```

```
48 [IMP_Fact_aux(:,:,i),RMS_Err_aux(:,:,i),tc(:,:,i),RMS_trss_Err_aux
(:,:,i),RMS_steady_Err_aux(:,:,i)] =
KF_comp(Overflow_Mode,Round_Mode,WordLength,i,
simulationTime,con_KG,con_t,Plot);
49 %Sum of the RMS_Err
50 RMS_Err(:,:,i)= RMS_Err(:,:,i) + RMS_Err_aux(:,:,i);
51 %Sum of the RMS_trss_Err
52 RMS_trss_Err(:,:,i)= RMS_trss_Err(:,:,i) + RMS_trss_Err_aux(:,:,i);
53 %Sum of the RMS_steady_Err
54 RMS_steady_Err(:,:,i)= RMS_steady_Err(:,:,i) + RMS_steady_Err_aux
(:,:,i);
55 %Sum of the Improvement factor
56 IMP_Fact(:,:,i)= IMP_Fact(:,:,i) + IMP_Fact_aux(:,:,i);
57 end
58 %Average RMS_Err
59 RMS_Err(:,:,i) = RMS_Err(:,:,i)/real ;
60 %Average RMS_trss_Err
61 RMS_trss_Err(:,:,i) = RMS_trss_Err(:,:,i)/real ;
62 %Average RMS_steady_Err
63 RMS_steady_Err(:,:,i) = RMS_steady_Err(:,:,i)/real ;
64 %Average Improvement factor
65 IMP_Fact(:,:,i)= IMP_Fact(:,:,i)/real ;
66 end
67
68
69 %%%%%Convergence time
70
71 figure('Name','mean ct quotient % vs FL for btDP vs btFP (1 Row) and
btFP vs aoFP
(2 Row)','NumberTitle','off');
72
73 for i=1:n
74 for k=1 : WordLength-1
75 plot1(k) = (tc(1,i,k)/tc(3,i,k));
76 plot2(k) = (tc(3,i,k)/tc(2,i,k));
77 end
78 subplot(2,n,i),semilogy(FL,plot1,FL,
One,'Marker','*','MarkerSize',3),xlabel('FL [bit]'),ylabel('ct(KF-BT-
DP)/ct(KF-BTFP)')
79 subplot(2,n,i+n),semilogy(FL,plot2,FL,
One,'Marker','*','MarkerSize',3),xlabel('FL [bit]'),ylabel('ct(KF-BT-
FP)/ct(KF-OAFP)')
80 end
81
82 %%%%%RMS error
83
84 figure('Name','mean RMS error quotient % vs FL for btDP vs btFP (1 Row)
and btFP
vs aoFP (2 Row)','NumberTitle','off');
85
86 for i=1:n
87 for k=1 : WordLength-1
88 plot1(k) = (RMS_Err(1,i,k)/RMS_Err(3,i,k));
89 plot2(k) = (RMS_Err(3,i,k)/RMS_Err(2,i,k));
90 end
91 subplot(2,n,i),semilogy(FL,plot1,FL,
One,'Marker','*','MarkerSize',3),xlabel('FL [bit]'),ylabel('RMS Err(KF-BT-
DP)/RMS Err
(KF-BT-FP)')
92 subplot(2,n,i+n),semilogy(FL,plot2,FL,
One,'Marker','*','MarkerSize',3),xlabel('FL [bit]'),ylabel('RMS Err(KF-BT-
FP)/RMS Err
```

```
(KF-OA-FP)')
93 end
94
95 %%%%%RMS transient error
96
97 figure('Name','mean RMS transient error quotient % vs FL for btDP vs
btFP (1 Row)
and btFP vs aoFP (2 Row)','NumberTitle','off');
98
99 for i=1:n
100 for k=1 : WordLength-1
101 plot1(k) = (RMS_trss_Err(1,i,k)/RMS_trss_Err(3,i,k));
102 plot2(k) = (RMS_trss_Err(3,i,k)/RMS_trss_Err(2,i,k));
103 end
104 subplot(2,n,i),semilogy(FL,plot1,FL,
One,'Marker','*','MarkerSize',3),xlabel('FL [bit]'),ylabel({'RMS trss
Err(KF-BTDP)';'
RMS trss Err(KF-BT-FP)'})
105 subplot(2,n,i+n),semilogy(FL,plot2,FL,
One,'Marker','*','MarkerSize',3),xlabel('FL [bit]'),ylabel({'RMS trss
Err(KF-BTFP)';'
RMS trss Err(KF-OA-FP)'})
106 end
107
108 %%%%%RMS steady error
109
110 figure('Name','mean RMS steady error quotient % vs FL for btDP vs btFP
(1 Row)
and btFP vs aoFP (2 Row)','NumberTitle','off');
111
112 for i=1:n
113 for k=1 : WordLength-1
114 plot1(k) = (RMS_steady_Err(1,i,k)/RMS_steady_Err(3,i,k));
115 plot2(k) = (RMS_steady_Err(3,i,k)/RMS_steady_Err(2,i,k));
116 end
117 subplot(2,n,i),semilogy(FL,plot1,FL,
One,'Marker','*','MarkerSize',3),xlabel('FL [bit]'),ylabel({'RMS steady
Err(KF-BTDP)';'
RMS steady Err(KF-BT-FP)'})
118 subplot(2,n,i+n),semilogy(FL,plot2,FL,
One,'Marker','*','MarkerSize',3),xlabel('FL [bit]'),ylabel({'RMS steady
Err(KF-BTFP)';'
RMS steady Err(KF-OA-FP)'})
119 end
120
121 %%%%%Improvement factor
122
123 figure('Name','mean Improvement factor quotient % vs FL for btDP vs
btFP (1 Row)
and btFP vs aoFP (2 Row)','NumberTitle','off');
124
125 for i=1:l
126 for k=1 : WordLength-1
127 plot1(k) = (IMP_Fact(1,i,k)/IMP_Fact(3,i,k));
128 plot2(k) = (IMP_Fact(3,i,k)/IMP_Fact(2,i,k));
129 end
130 subplot(2,l,i),semilogy(FL,plot1,FL,
One,'Marker','*','MarkerSize',3),xlabel('FL [bit]'),ylabel('IMP Fact(KF-BT-
DP)/IMP
Fact(KF-BT-FP)')
131 subplot(2,l,i+l),semilogy(FL,plot2,FL,
```

```
One,'Marker','*','MarkerSize',3),xlabel('FL [bit]'),ylabel('IMP Fact(KF-BT-
FP)/IMP
Fact(KF-OA-FP)]')
132 end
```

### 11.1.4 Basic_Plots.m

```
1 %The program generates the plots showing the estimation results and the
2 %gains for 3 different KF implementations ( KF-BT-DP ,KF-OA-FP and
3 %KF-BT-FP) running for the same problem.
4
5 %Authors: F.A.Igleisas & D.Dematties - UTN-FRM 2011 - Argentina.
6 %franciscoiglesias@frm.utn.edu.ar ; dariodematties@yahoo.com.ar
7
8 %The plots are for each state of the system:
9 %1-The exact state, the estimated state and the measurements vs time
10 %2-The P vs time
11 %3-The KG vs time
12
13 %Requires function KF_comp
14
15 clf
16 clear all;
17 close all;
18
19 n=2; %States number for the system implemented inside KF_comp function
20 l=1; %Measurement vextor dimention.
21 real=1; % Number of realizations to average the rmsErr and ct
22 WordLength = 18 ;
23 FractionLength = 9;
24 Overflow_Mode = 'saturate';
25 Round_Mode = 'floor';
26 con_KG = 0.005 ; % Defines the convergence criterion: This is the end KG
value
27 con_t = 1 ; % Defines the convergence criterion: time lapse with
KG<con_KG to
consider convergence
28 simulationTime = 10 ; % Simulation time in seconds.
29 Plot = 1 ;
30
31 RMS_steady_Err = zeros(3,n); % to save the rms error of the steady part
for each
point position and each state
32 RMS_trss_Err=RMS_steady_Err; % to save the rms error of the transient
part for
each point position and each state
33 RMS_Err = zeros(4,n); % to save the overall rms error
34 RMS_trss_Err_aux= RMS_trss_Err ;
35 RMS_steady_Err_aux= RMS_trss_Err ;
36 tc = zeros (3,n,WordLength-1); % to save the convergence time for each
point
position and each state
37 IMP_Fact= zeros(3,l);
38 % NOTE: The convergence time doesn't change with the realizations, so we
don't
compute an average value
39
40 warning ('off', 'MATLAB:divideByZero') ;
41 for i=1 : real
42 [IMP_Fact(:,:),RMS_Err(:,:,i),tc(:,:,i),RMS_trss_Err_aux(:,:,i),
RMS_steady_Err_aux(:,:,i)] = KF_comp(Overflow_Mode,Round_Mode,WordLength,
FractionLength,simulationTime,con_KG,con_t,Plot);
```

```
43 end
44
```

### 11.1.5 Arithmetic_Selection.m

```
1 %Compration of the KF Bierman-Thornton implementation running in
different Fixed
Point arithemtics.
2 %Authors: F.A.Igleisas & D.Dematties - UTN-FRM 2011 - Argentina.
3 %franciscoiglesias@frm.utn.edu.ar ; dariodematties@yahoo.com.ar
4
5 %NOTE: An arithemtic is defined by:
6 % 1-Word Length
7 % 2-Fractional part Length
8 % 3-Overflow Mode
9 % 4-Round Mode
10
11 %The program, plots the IMP_Fact for 4 combinations of the Overflow and
12 %Round modes:
13 %1-Saturate/Fix
14 %2-Saturate/Floor
15 %3-Wrap/Fix
16 %4-Wrap/Floor
17
18 %The ploted values are the averags over several number of realizations
19
20 %Requires the function KF_comp
21
22 clf
23 clear all;
24 close all;
25
26 n=2; %States number for the system implemented inside KF_comp function
27 l=1; % measurement vector lenght
28 real=10; % Number of realizations to average the rmsErr and ct
29 WordLength = 18;
30 con_KG = 0.005 ; % Defines the convergence criterion: This is the end KG
value
31 con_t = 1 ; % Defines the convergence criterion: time lapse with
KG<con_KG to
consider convergence
32 simulationTime = 10 ; % Simulation time in seconds.
33 Plot = 0 ;
34
35 RMS_Err = zeros(4,n,WordLength-1,4); % to save the rms error of the
steady part
for each point position, each state and each of the
36 %4 modes: Overflow Mode: Saturate or Wrap ; Round Mode: Fix or Round
37 RMS_Err_aux=RMS_Err;
38 RMS_trss_Err=zeros(3,n,WordLength-1,4); % to save the rms error of the
transient
part for each point position and each state
39 RMS_steady_Err = RMS_trss_Err; % to save the overall rms error
40 RMS_trss_Err_aux= RMS_trss_Err ;
41 RMS_steady_Err_aux= RMS_steady_Err ;
42 tc = zeros (3,n,WordLength-1,4); % to save the convergence time for each
point
position and each state
43 % NOTE: The convergence time doesn't change with the realizations, so we
don't
compute an average value
44 point_pos = 1:WordLength-1 ; % generates the x axis for the plots
```

```
45 IMP_Fact= zeros(3,l,WordLength-1,4);
46 IMP_Fact_aux=IMP_Fact;
47
48 warning ('off', 'MATLAB:divideByZero') ;
49
50 for m=1 :4
51 switch m
52 case 1 % 1 arithmetic
53 Overflow_Mode = 'saturate';
54 Round_Mode = 'fix';
55 case 2 % 2 arithmetic
56 Overflow_Mode = 'saturate';
57 Round_Mode = 'floor';
58 case 3 % 3 arithmetic
59 Overflow_Mode = 'wrap';
60 Round_Mode = 'fix';
61 case 4 % 4 arithmetic
62 Overflow_Mode = 'wrap';
63 Round_Mode = 'floor';
64 end
65
66 for i=1 : WordLength-1
67 for k=1 : real
68 [IMP_Fact_aux(:,:,i,m),RMS_Err_aux(:,:,i,m),tc(:,:,i,m),
RMS_trss_Err_aux(:,:,i,m),RMS_steady_Err_aux(:,:,i,m)] =
KF_comp(Overflow_Mode,
Round_Mode,WordLength,i,simulationTime,con_KG,con_t,Plot);
69 %Average RMS_steady_Err
70 IMP_Fact(:,:,i,m)= IMP_Fact(:,:,i,m) + IMP_Fact_aux(:,:,i,m);
71 end
72 IMP_Fact(:,:,i,m)= IMP_Fact(:,:,i,m)/real ;
73 end
74 end
75
76 %%%%%Plots the btFP IMP_Fact( average of all the realizations) vs fixed
point
position, for each arithemtic
77 %State 1
78 figure('Name','X1 IMP Fact for btFP vs Fractional part length: Saturate-
Round
(blue) , Saturate-Floor(red), Wrap-Round (green), Wrap-Floor
(yellow)','NumberTitle','off');
79 for j=1:4
80 for k=1 : WordLength-1
81 plot1(k) = IMP_Fact(3,1,k,j);
82 end
83 switch j
84 case 1
85 plot(point_pos,plot1,'--b+');
86 case 2
87 hold on;
88 plot(point_pos,plot1,'-.r+');
89 case 3
90 hold on;
91 plot(point_pos,plot1,':g+');
92 case 4
93 hold on;
94 plot(point_pos,plot1,'-y+'),xlabel('FL[bit]'),ylabel( 'IMP Fact_1 (KFBT-
FP)')
95 end
96 end
```

### 11.1.6 KF_test_for_implementation.m

```
1 %KF execution for VHDL implementations comparison for a second order
oscilator.
2 %F.A.Igleisas & D.Dematties - UTN-FRM 2011.
3
4 %Perform for a second order system (the code is general though):
5 %The BT implementation in signed Fixed point
6 %INPUTS
7 %Dt Ejecutation's iterations' number.
8 %OB When is "1", only Bierman algorithm executation.
9 %OT When is "1",only Thornton algorithm executation.
10 %FR When is "1", shows only the results of the las iteration
11 %OverflowMode: Mode to deal with overflows
12 %RoundMode: Rounding mode
13 %W_length: Word lentgth for the fixed point implementation
14 %WF_length: Number of bits in the fractional part of the word
15 %OUTPUTS
16 % For each iteration each of these values is shown in binary:
17 %Dpost:a posteriori D matrix
18 %Upost:a posteriori U matrix
19 %Xpost:a posteriori X vector
20 %Dprior:a priori D matrix
21 %Uprior:a priori U matrix
22 %Xprior:a priori X vector
23
24 clf
25 clear all;
26 close all;
27
28 %%%%%%%%%%%%%%%%%Definition of the executation
mode%%%%%%%%%%%%%%%%%%%%%%%%%
29
30 Dt=100; %Ejecutation's iterations' number.
31 OB=0; %When is "1", only Bierman algorithm executation.
32 OT=0; %When is "1",only Thornton algorithm executation.
33 FR=0; %When is "1", shows only the results of the las iteration
34 BT=0; %Data to burn the fpga (Burn Test)
35
36 %%%%%%%%%%%%%%%%%Definition of the Fixed Point
arithmetic%%%%%%%%%%%%%%%%%%
37
38 OverflowMode='saturate';
39 RoundMode='floor';
40 W_length=18;
41 WF_length=9;
42
43
44 Signe=1; %signe consideration "1" or not consideration "0".
45 OverflowM=OverflowMode;
46 RoundM=RoundMode;
47 SumM='SpecifyPrecision';
48 ProductM='SpecifyPrecision';
49 SumCast=0;
50
51 %Arithmetic Sum operation result parameters.
52 SumWordLength=W_length; %sum result word length.
53 SumFractionLength=WF_length; %sum result fraction length.
54
55 %Arithmetic Product operation result parameters.
56 ProductWordLength=W_length; %product result word length.
57 ProductFractionLength=WF_length; %product result fraction length.
```

```
58
59 %Arithmetic variables' parameters.
60 WordLength=W_length; %word length.
61 FractionLength=WF_length; %fraction length.
62
63 %Scaling rules
64 ProductSlope=2^(-6);
65 ProductFixedExponent=-6;
66
67 %A numerictype object specification.
68 N=numerictype('Signed',Signe,'WordLength',WordLength,'FractionLength',
FractionLength);
69
70
71 %%%%%%%%%%%%%%%%Simulation variables and
parameters%%%%%%%%%%%%%%%%%%%%%%%%
72
73 n=2; %state vector dimention.
74 l=1; %mesurement vector dimention.
75 p=1; %process noise vector dimention.
76 T=0.01; %Sampling time in seconds.
77
78 %Note thah the last dimention denote the time variable.
79 %The initial step is 1 corresponing to de initial conditions
80
81 %%%%%%%%%%%%%%%%Definition of Double precision
variables%%%%%%%%%%%%%%%%%%
82
83 X=zeros(n,Dt); %Exact State vectors matrix
84 E_X=zeros(n,Dt); %Estimated State vectors matrix
85 Z=zeros(l,Dt); %Measurent vectors matrix
86 Phi=zeros(n,n);
87 G=zeros(n,p); %Process noise coumpling matrix
88 P=zeros(n,n,Dt); %One matrix at each instant.
89 H=zeros(l,n);
90 R=zeros(l,l); %Discrete measurement noise covariance matrix
91 V=zeros(l,Dt); %Noise vectors matrix
92 Q=zeros(n,n); %Discrete process noise covariance matrix
93 W=zeros(n,Dt); %Noise vectors matrix
94 KG=zeros(n,l,Dt);
95 uc=zeros(n,1); %Forcing term of the control input uc
96 Qc=zeros(p,p); %Variances of white-noise secuences of W.
97 Rc=zeros(l,l); %Variances of white-noise secuences of V.
98
99 %%%%%%Numeric Problem Definition: From example 4.4 Grewal and
Andrews%%%%%%
100
101 %Variances of white-noise secuences.
102 Rc(1,1)= 0.01; % 10%
103 Qc(1,1)= 0.02; % 14%
104
105 %Coeficients' Matrix
106 F(1,1)=0;
107 F(1,2)=1;
108 F(2,1)=-25;
109 F(2,2)=-2;
110
111 %State transition matrix.
112 Phi=expm(F*T);
113
114 %Process noise coupling matrix.
115 G(1,1)=0;
```

```
116 G(2,1)=1;
117
118 %Sensors matrix.
119 H(1,1)=1;
120 H(1,2)=0;
121
122 %System noise covariance matrix.
123 Q=T*(G*Qc*G'); % First order aproximation to obtain the discrete Q.
124
125 %Sensors noise covariance matrix.
126 R(1,1)= Rc/T;
127
128 %Control input (C*u).
129 u(1,1)=0;
130 u(2,1)=12;
131
132 %Forcing term.
133 uc = (Phi*u)*T;
134
135 %%%%%%%%%%%%%%%%%%%%%%%%%Initial
Conditions%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
136
137 %Estimation error covariance matrix.
138 P(1,1,1)=2;
139 P(2,1,1)=0;
140 P(1,2,1)=0;
141 P(2,2,1)=2;
142
143 %State vector.
144 X(1,1,1)=0;
145 X(2,1,1)=0;
146
147 %Estimated State vector.
148 E_X(:,:,1)=X(:,:,1);
149
150 %%%%%%%%%%%%%Measurements generation: Double
Precision%%%%%%%%%%%%%%%%%%%%%%
151
152 for i=1:Dt,
153 V(:,i)=normrnd(zeros(l,1),sqrt(diag(R)));
154 end
155 for i=1:Dt,
156 W(:,i)=normrnd(zeros(n,1),sqrt(diag(Q)));
157 end
158
159 %Noisy measurement generation.
160 for k=1:Dt-1,
161 X(:,k+1)=Phi*X(:,k)+W(:,k)+uc;
162 Z(:,k)=H*X(:,k)+V(:,k);
163 end
164 Z(:,Dt)=H*X(:,Dt)+V(:,Dt);
165 %In the last iteration, X(:,k+1) does not need to be computed.
166
167 %Exact States generation.
168 for k=1:Dt-1,
169 X(:,k+1)=Phi*X(:,k) + uc ;
170 end
171
172 %%%%%%%%%%%%%%%%%%%%%%Definition of fixed point
variables%%%%%%%%%%%%%%%%%%
173 %NOTE: Takes the numeric values from the double precision definitions
174
```

```
175 %Time variant matrices and vectors
176 E_X=fi(E_X,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
177 P=fi(P,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
178
KG=fi(zeros(n,l,Dt),N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode'
,
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
179
180 %Forcing Term
181 uc=fi(uc,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
182
183 %Measurents vector matrix.
184 Z=fi(Z,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
185
186 %Time invariant system's matrices.
187 Phi=fi(Phi,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
188 H=fi(H,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
189 Q=fi(Q,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
190 R=fi(R,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
191
192 %Auxiliary variables
193 %a=fi(a,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
194 %b=fi(b,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
195
196 %Auxiliary scalars.
197 Uno=fi(1,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
198
```

```
199 %%%%%%%%%%KF, Bierman-Thornton algorithm: Fixed
Point%%%%%%%%%%%%%%%%%%%%%%
200
201 %Defines the extra matrices for the BT algorithm.
202
203 % P=U*D*U'
204
U=fi(zeros(n,n,Dt),N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
205 Uv=fi(zeros((n^2-n)/2,Dt),N,'OverflowMode',OverflowM,'RoundMode',
RoundM,'SumMode',SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
206
D=fi(zeros(n,n,Dt),N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
207 % Q=G*Dq*G' Dq can come from a UD factorization if necesary.
208 G=fi(G,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
209 Dq=fi(Qc,N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
210
PhiU=fi(zeros(n,n),N,'OverflowMode',OverflowM,'RoundMode',RoundM,'SumMode',
SumM,'SumWordLength',SumWordLength,'SumFractionLength',
SumFractionLength,'ProductMode',ProductM,'ProductWordLength',
ProductWordLength,'ProductFractionLength',ProductFractionLength);
211
212
213 %Generates the initials U and D from P by UD factorization.
214 for j=n:-1:1,
215 for i=j:-1:1,
216 alfa=P(i,j,1);
217 for s=j+1:n,
218 alfa=alfa-U(i,s,1)*D(s,s,1)*U(j,s,1);
219 end
220 if i==j
221 D(j,j,1)=alfa;
222 U(j,j,1)=1;
223 else
224 U(i,j,1)=divide(N,alfa,D(j,j,1));
225 end
226 end
227 end
228
229 %Transfers the U matrix's upper triangle to the Uv vector. To work like
the vhdl
code
230 for j=1:n
231 for i=1:n
232 if j>i
233 Uv(j-n+(i*(2*n-i-1))/2,1)=U(i,j,1);
234 end
235 end
236 end
```

```
237
238 if BT==0
239 %Shows the initial conditions and initializations
240 disp('Matrices'' initializations:')
241 disp(' ')
242 disp('H0 =')
243 disp(bin(H(1,:)))
244 disp('Q =')
245 disp(bin(Q))
246 disp('R =')
247 disp(bin(R))
248 disp('Phi')
249 disp(bin(Phi))
250 disp(' ')
251 disp('Dq =')
252 disp(bin(Dq))
253 disp('G =')
254 disp(bin(G))
255 disp(' ')
256 disp('uc =')
257 disp(bin(uc))
258 disp(' ')
259 disp(' ')
260 disp('Initial conditions')
261 disp(' ')
262 disp('P0 =')
263 disp(bin(P(:,:,1)))
264 disp('D0 =')
265 disp(bin(D(:,:,1)))
266 disp('U0 =')
267 disp(bin(Uv(:,1)))
268 disp('X0 =')
269 disp(bin(E_X(:,1)))
270 disp('Z0 =')
271 disp(bin(Z(:,1)))
272 disp(' ')
273 disp(' ')
274 end
275
276 %Bierman-Thornton algorithm
277 for k=1:Dt,
278
279 if OT==0
280 for r=1:l, %Sequential measurement processing loop
281
282 %Bierman observational update:
283 %Based on "bierman.m" by Grewal and Andrews
284 % a=U(:,:,k)'*H(r,:)'; %Non scalar
285 % b=D(:,:,k)*a; %Non scalar
286 % dz=Z(r,k) - H(r,:)*E_X(:,k);%Non scalar
287 dz = Z(r,k);
288 for j=1:n
289 s_aux = H(r,j)*E_X(j,k) ;
290 dz = dz - s_aux ; % computes Z-H*X
291 a(j) = H(r,j);
292 i = 1;
293 while 1 % equivalent to: for in 1 to (j-1) (iterates j-1 times)
294 if i == j
295 break;
296 end
297 index_aux = j-n+(i*(2*n-i-1))/2 ; % to find the element Uij
inside the U vector
```

```
298 s_aux1 = Uv(index_aux,k)*H(r,i) ;
299 a(j) = a(j) + s_aux1; % computes V=U'*H'
300 i = i+1 ;
301 end
302 end
303
304 for j=1:n
305 b(j) = D(j,j,k)*a(j); % computes the b factor in the Grewall's
Bierman matlab algorithm (called the unescaled kalman gain)
306 end
307
308 alpha=R(r,r);
309 gamma=divide(N,Uno,alpha);
310
311 for j=1:n, %Within these two for, all the operations are scalar
312 beta=alpha;
313 s_aux= a(j)*b(j);
314 alpha=alpha +s_aux;
315 lambda= -(a(j)*gamma);
316 gamma =divide(N,Uno,alpha);
317 D(j,j,k)=beta*(gamma*D(j,j,k));
318 i=1;
319 while 1
320 if i == j
321 break;
322 end
323 index_aux = j-n+(i*(2*n-i-1))/2 ; % to find the element Uij
inside the U vector
324 beta=Uv(index_aux,k);
325 s_aux=b(i)*lambda;
326 Uv(index_aux,k)=beta + s_aux;
327 s_aux=b(j)*beta;
328 b(i)=b(i) + s_aux;
329 i=i+1;
330 end
331 end
332
333 for j=1:n
334 aux =(gamma*dz)*b(j);
335 E_X(j,k)=E_X(j,k)+aux;
336 end
337
338 % Save the KG just to plot it !!!
339 KG(:,r,k)=gamma*b;
340
341
342 %Saves the a posteriori D, U and X.
343 Dpost=D(:,:,k);
344 Upost=Uv(:,k);
345 Xpost=E_X(:,k);
346
347 if OB==1
348 D(:,:,k+1)=D(:,:,k);
349 Uv(:,k+1)=Uv(:,k);
350 E_X(:,k+1)=E_X(:,k);
351 end
352
353 if BT==0
354 %Shows the a posteriori D, U and X.
355 if FR==1
356 if k==Dt
357 disp('Bierman k iteration = ')
```

```
358 disp(k)
359 disp(' ')
360 if l > 1
361 disp('Measurement component number = ')
362 disp(r)
363 end
364 disp('Z =')
365 disp(bin(Z(r,k)))
366 disp(' ')
367 disp('Dpost = ')
368 disp(bin(Dpost))
369 disp('Upost =')
370 disp(bin(Upost))
371 disp('Xpost =')
372 disp(bin(Xpost))
373 disp(' ')
374 disp(' ')
375 disp(' ')
376 disp(' ')
377 end
378 else
379 disp('Bierman k iteration = ')
380 disp(k)
381 disp(' ')
382 if l > 1
383 disp('Measurement component number = ')
384 disp(r)
385 end
386 disp('Z =')
387 disp(bin(Z(r,k)))
388 disp(' ')
389 disp('Dpost = ')
390 disp(bin(Dpost))
391 disp('Upost =')
392 disp(bin(Upost))
393 disp('Xpost =')
394 disp(bin(Xpost))
395 disp(' ')
396 disp(' ')
397 disp(' ')
398 disp(' ')
399 end
400 end
401 end
402
403 end
404
405 %Temporal update.
406 if OB==0 & k<Dt
407
408 %Thornton temporal update: Based on "thornton.m" by Grewal and Andrews.
409 %PhiU = Phi*U(:,:,k);
410 G_aux=G; %To avoid G destruction
411 PhiU(:,:) = 0;
412 for i=1:n,
413 for j=1:n,
414 r=1;
415 while 1
416 if r==j % equivalent to: for r=1 to (j-1) but vhdl
implementable
417 break;
418 end
```

```
index_aux = j-n+(r*(2*n-r-1))/2;
PhiU(i,j)= PhiU(i,j)+ Phi(i,r)*Uv(index_aux,k);
r=r+1;
end
PhiU(i,j)=PhiU(i,j)+Phi(i,j);
end
end

for i=n:-1:1,
sigma = 0;
for j=1:n,
sigma = sigma + PhiU(i,j)*(PhiU(i,j)*D(j,j,k)) ;
if (j <= p)
sigma = sigma + G_aux(i,j)*(G_aux(i,j)*Dq(j,j));
end
end
D(i,i,k+1) = sigma;

j=1;
while 1 % equivalent to: for j=1 to (i-1) but vhdl implementable
if j == i
break;
end
sigma = 0;
for s=1:n
sigma = sigma + PhiU(i,s)*(D(s,s,k)*PhiU(j,s));
end

for s=1:p,
sigma = sigma + G_aux(i,s)*(Dq(s,s)*G_aux(j,s));
end
index_aux = i-n+(j*(2*n-j-1))/2 ; % to find the element Uji (note
the changed index in this case) inside the U vector
Uv(index_aux,k+1) = divide(N,Uno,D(i,i,k+1))*sigma ;
for s=1:n,
PhiU(j,s) = PhiU(j,s) - Uv(index_aux,k+1)*PhiU(i,s);
end
for s=1:p,
G_aux(j,s) = G_aux(j,s) - Uv(index_aux,k+1)*G_aux(i,s);
end
j = j+1;
end
end

%State estimation temporal update
E_X(:,k+1)= 0 ;
for i=1:n,
for r=1:n,
E_X(i,k+1)=E_X(i,k+1)+Phi(i,r)*E_X(r,k);
end
end

E_X(:,k+1)=E_X(:,k+1)+uc;

%Save the a priori D, U and X.
Dprior= D(:,:,k+1);
Uprior= Uv(:,k+1);
Xprior= E_X(:,k+1);

if BT==0
%Shows the a priori D, U and X.
if FR==1
```

```
480 if k==Dt
481 disp('Thornton k iteration =')
482 disp(k)
483 disp(' ')
484 disp('Dprior = ')
485 disp(bin(Dprior))
486 disp('Uprior =')
487 disp(bin(Uprior))
488 disp('Xprior =')
489 disp(bin(Xprior))
490 disp(' ')
491 disp(' ')
492 disp(' ')
493 disp(' ')
494 end
495 else
496 disp('Thornton k iteration =')
497 disp(k)
498 disp(' ')
499 disp('Dprior = ')
500 disp(bin(Dprior))
501 disp('Uprior =')
502 disp(bin(Uprior))
503 disp('Xprior =')
504 disp(bin(Xprior))
505 disp(' ')
506 disp(' ')
507 disp(' ')
508 disp(' ')
509 end
510 end
511 end
512 end
513 if BT==1
514 disp('For Z:')
515 for i=1:l
516 for k=1:Dt
517 if k==1
518 fprintf('%d => ( %d => "%s",', i,k,bin(Z(i,k)))
519 elseif k==Dt
520 fprintf('%d => "%s"),\n', k,bin(Z(i,k)))
521 else
522 fprintf('%d => "%s",', k,bin(Z(i,k)))
523 end
524 if rem(k,100)==0
525 disp(' ')
526 end
527 end
528 end
529
530 disp(' ')
531 disp('For E_X:')
532
533 for i=1:n
534 for k=1:Dt
535 if k==1
536 fprintf('%d => ( %d => "%s",', i,k,bin(E_X(i,k)))
537 elseif k==Dt
538 fprintf('%d => "%s"),\n', k,bin(E_X(i,k)))
539 else
540 fprintf('%d => "%s",', k,bin(E_X(i,k)))
541 end
```

```
542 if rem(k,100)==0
543 disp(' ')
544 end
545 end
546 end
547
548 disp(' ')
549 end
550
551 %Plots all the Z and the E_X together
552 disp('All the Z together:')
553 for k=1:Dt
554 disp(bin(Z(:,k)));
555 end
556
557 disp('All the E_X together:')
558 for k=1:Dt
559 disp([bin(E_X(1,k)),' ',bin(E_X(2,k))]);
560 end
561
```

## 11.2 Código fuente en VHDL

### 11.2.1 Fixed_Point_pkg.vhd

```
1 --Kalman Filter, Bierman-Thornton implementation in FPGA
2 --F.A.Iglesias (Leg:27310) and D.Dematties (Leg:22547)
3 --UTN-FRM 2012
4 ------------------------------------------------------------
5 --Package Description:
6 --Defines the fixed point arithmetic functions
7 --1)Resize function:
8 -- Transform the input number, that can be one of the two types
(overloaded operator):
9 -- S_WORD_p1: Used to store the signed fixed point number:
Sign+(INTEGER_LENGTH+1).FRACTION_LENGTH
10 -- S_WORD_x2: Used to store the signed fixed point number:
Sign+(2*INTEGER_LENGTH+1).(2*FRACTION_LENGTH)
11 -- ,in to the working number type:
12 -- S_WORD : Used to store the signed fixed point number:
Sign+INTEGER_LENGTH.FRACTION_LENGTH
13 -- This is done by reducing both the integer and the fractional part.
14 -- The applied criteria are:
15 -- Overflow Mode: Saturate
16 -- Round Mode: Fix (Truncate)
17
18 --2)Reciprocal:
19 --Returns the reciprocal of the S_WORD argument.
20 -- Works with arguments in 2's complement representation
21 -- When the argument is 0 returns the larges positive number
22 -- INPUTS:
23 -- S_WORD: Defines the Divisor
24 -- OUTPTS:
25 -- S_WORD_x2: Returns the reciprocal = 1/Divisor
26
27 --3)Add function:
28 --Returns the addition of the two S_WORD arguments.
29 -- Works with arguments in 2's complement representation
30 -- Returns the result in S_WORD by applying the resize function
31 -- INPUTS:
32 -- S_WORD: The two arguments A and B
33 -- OUTPTS:
```

```
34 -- S_WORD: The result of A+B with the proper arithmetic parameters
35
36 --4)Complement function:
37 --Returns the negative of the S_WORD argument.
38 -- Works with argument in 2's complement representation
39 -- Returns the result in S_WORD by applying the resize function
40 -- INPUTS:
41 -- S_WORD: The arguments A
42 -- OUTPTS:
43 -- S_WORD: The result of -A with the proper arithmetic parameters
44
45 ----------------------
46 --Comments:
47 --yaaaaaaaaaaaahooooooo!!!
48 ------------------------------------------------------------
49 --Libraries:
50 library IEEE;
51 use IEEE.STD_LOGIC_1164.ALL;
52 use ieee.numeric_std.all;
53 --Our packages
54 use work.parameters.ALL; 55 use work.system_def.ALL;
56 ------------------------------------------------------------
57
58 package Fixed_Point_pkg is
59
60 -----------Functions declaration-----------------------
61
62 --Resize function, input : S_WORD_p1
63 function resize_p1 (sfixed_add: S_WORD_p1) return S_WORD;
64
65 --Resize function, input : S_WORD_x2
66 function resize_x2 (sfixed_mult: S_WORD_x2 ) return S_WORD;
67
68 --Fixed point division
69 function fp_reciprocal(divisor: S_WORD)return S_WORD;
70
71 --Fixed point Addition
72 function fp_add(A, B: S_WORD)return S_WORD;
73
74 --Fixed point Complement
75 function fp_comp( A: S_WORD)return S_WORD;
76
77 end Fixed_Point_pkg ;
78
79 --------------------Functions'-body---------------------------
80 package body Fixed_Point_pkg is
81
82 -------Resize function, input : S_WORD_p1----------------------
83 --Note that Rounding is not necesary since S_WORD and S_WORD_p1 have the
same
fractional length
84 function resize_p1 (
85 sfixed_add: S_WORD_p1) return S_WORD is
86 variable sfixed : S_WORD;
87 begin
88 if sfixed_add(WORD_LENGTH) = sfixed_add(WORD_LENGTH-1) then
89 -- No overflow, then: sfixed <- sfixed_add
90 sfixed:= sfixed_add((WORD_LENGTH-1) downto 0) ;-- trafers the valid bits
91 else
92 -- Overflow, then: sfixed <- Saturate
93 if sfixed_add(WORD_LENGTH) = '0' then
94 sfixed := max_S_WORD ; -- Positive saturate
```

```
95 else
96 sfixed := min_S_WORD ; -- Negative saturate
97 end if;
98 end if;
99 return sfixed ;
100 end function resize_p1;
101
102 --Resize function, input : S_WORD_x2----------------------------
103 function resize_x2 (
104 sfixed_mult: S_WORD_x2) return S_WORD is
105 variable sfixed : S_WORD;
106 subtype S_AUX is signed ((INTEGER_LENGTH+1) downto 0);-- Subtype,
length = IL + 2
to check the padding in sfixed_mult
107 constant all_1_aux : S_AUX := (others => '1');
108 constant all_0_aux : S_AUX := (others => '0');
109 begin 110
111 if sfixed_mult((2*WORD_LENGTH-1) downto
(2*FRACTION_LENGTH+INTEGER_LENGTH)) =
all_1_aux or
112 sfixed_mult((2*WORD_LENGTH-1) downto
(2*FRACTION_LENGTH+INTEGER_LENGTH)) =
all_0_aux then
113 -- No overflow, then: sfixed <- sfixed_mult
114 sfixed (WORD_LENGTH-1 downto 0) :=
sfixed_mult((2*FRACTION_LENGTH+INTEGER_LENGTH
) downto FRACTION_LENGTH) ; -- transfers the valid bits
115 else
116 -- Overflow, then: sfixed <- Saturate
117 if sfixed_mult(2*WORD_LENGTH-1) = '0' then
118 sfixed := max_S_WORD ; -- Positive saturate
119 else
120 sfixed := min_S_WORD ; -- Negative saturate
121 end if;
122 end if;
123
124 return sfixed ;
125 end function resize_x2;
126
127 ----------------------Addition function-------------------------------
--
128 --Perform A+B=C with the proper fixed point considerations
129 function fp_add(
130 A, B: S_WORD)return S_WORD is
131 variable C : S_WORD;
132 begin
133 C := resize_p1((A(WORD_LENGTH-1) & A) + (B(WORD_LENGTH-1) & B)) ;
134 return C ;
135 end function fp_add;
136
137 ----------------------Complement function-----------------------------
----
138 --Perform Res = -A with the proper fixed point considerations
139
140 function fp_comp(
141 A: S_WORD)return S_WORD is
142 variable Res : S_WORD;
143 begin
144 Res := resize_p1(-(A(WORD_LENGTH-1) & A)); -- complements the A operand
145 return Res ;
146 end function fp_comp;
147
```

```
148 ---------Reciprocal division-----------------------------
149 --Note that we scaled the dividend in order to do the fixed point
division by an
integer division
150
151 function fp_reciprocal (
152 divisor: S_WORD)return S_WORD is
153
154 subtype S_WORD_pFL is signed ((WORD_LENGTH + FRACTION_LENGTH-1) downto
0);
--Signed variable of length WORD_LENGTH + FRACTION LENGTH to save Division
results
155 subtype S_AUX is signed ((FRACTION_LENGTH) downto 0);-- Subtype, length
= FL+1 to
check the padding in the result
156 subtype S_AUX2 is signed ((FRACTION_LENGTH-1) downto 0);-- Subtype,
length = FL to
padding the divisor
157
158 VARIABLE Tmp,Avar,Tmp3, Quot: S_WORD_pFL ;
159 VARIABLE Result ,Bvar : S_WORD ; 160
161 constant sP1_0_pFL : S_WORD_pFL :=(2*FRACTION_LENGTH=>'1',
others=>'0'); -- Used
to do the reciprocal division (1/S_WORD) in Bierman and Thornton's modules
162 constant all_1_aux : S_AUX := (others => '1');
163 constant all_0_aux : S_AUX := (others => '0');
164 constant all_0_aux2: S_AUX2 := (others => '0');
165
166 begin
167
168 if divisor = null_S_WORD then
169 Result := max_S_WORD ; -- When divideing by 0 returns positive saturate
170 else
171 if divisor(WORD_LENGTH-1)='1' then -- if divisor is negative must be
complemented
172 Bvar := fp_comp(divisor) ;
173 else
174 Bvar := divisor ;
175 end if;
176
177 Avar := sP1_0_pFL; -- Dividend, constant for reciprocal
178
179 FOR i IN (WORD_LENGTH + FRACTION_LENGTH-1) DOWNTO 0 LOOP
180 Tmp := (others=>'0');
181 Tmp((WORD_LENGTH + FRACTION_LENGTH-1)-i DOWNTO 0) := Avar((WORD_LENGTH
+
FRACTION_LENGTH-1) DOWNTO i) ;
182 IF Tmp >= Bvar THEN
183 Quot(i) := '1';
184 Tmp3 := Tmp - Bvar;
185 IF i /= 0 THEN
186 Avar((WORD_LENGTH + FRACTION_LENGTH-1) DOWNTO i) := Tmp3((WORD_LENGTH +
FRACTION_LENGTH-1)-i DOWNTO 0);
187 Avar(i-1) := sP1_0_pFL(i-1);
188 END IF;
189 ELSE
190 Quot(i) := '0';
191 Tmp3 := Tmp;
192 END IF;
193 END LOOP;
194
195
```

```
196 --resize the result
197
198 if Quot((WORD_LENGTH + FRACTION_LENGTH-1) downto WORD_LENGTH-1) =
all_1_aux or
199 Quot((WORD_LENGTH + FRACTION_LENGTH-1) downto WORD_LENGTH-1) =
all_0_aux then
200 -- No overflow, then: sfixed <- sfixed_mult
201 Result(WORD_LENGTH-1 downto 0) := Quot((WORD_LENGTH-1) downto 0) ; --
transfers
the valid bits
202 else
203 -- Overflow, then: sfixed <- Saturate
204 if Quot(WORD_LENGTH + FRACTION_LENGTH-1) = '0' then
205 Result := max_S_WORD ; -- Positive saturate
206 else
207 Result := min_S_WORD ; -- Negative saturate
208 end if;
209 end if;
210
211 if divisor(WORD_LENGTH-1)='1' then -- if divisor is negative, the
result must be
complemented 212 Result := fp_comp(result) ;
213 end if;
214
215 end if ;
216
217 return Result;
218 end function fp_reciprocal;
219
220 end Fixed_Point_pkg ;
221
```

### 11.2.2 Parameters.vhd

```
1 --Kalman Filter, Bierman-Thornton implementation in FPGA
2 --F.A.Iglesias (Leg:27310) and D.Dematties (Leg:22547)
3 --UTN-FRM 2012
4 ----------------------------------------------------------
5 --Description:
6 --
7 --INPUTS:
8 --
9 --OUTPUTS:
10 --
11 --Comments:
12 --
13 ----------------------------------------------------------
14 --Libraries:
15 library IEEE;
16 use IEEE.STD_LOGIC_1164.ALL;
17 use ieee.numeric_std.all;
18 library UNISIM; -- Used to simulate when using componetes of the library
19 use UNISIM.VComponents.all;
20 ----------------------------------------------------------
21 --Package:
22 --Defines the systems arithemtic and other logic paramateres
23
24 package parameters is
25
26 constant clk_div : integer := 12 ;-- To compute internal clk from the 50
MHz
digilent external clk.
```

```
27 constant n : integer := 2;-- Dimension of the state vector X
28 constant l : integer := 1;-- Dimension of the measurement vector Z
29 constant p : integer := 1;-- Dimension of the input vector u
30 constant WORD_LENGTH : integer := 18;-- Word lentgth =
1+INTEGER_LENGTH+FRACTION_LENGTH
31 constant FRACTION_LENGTH : integer := 6;-- Fraction length
32 constant INTEGER_LENGTH : integer := WORD_LENGTH - FRACTION_LENGTH - 1;
33 constant log2_l : integer := 2;
34
35 -- Automatic definitions, do not change this !!:
36
37 --WORDS
38 Subtype l_STATE is std_logic_vector ((log2_l-1) downto 0);--Controller's
internal
state signal.
39 Subtype WORD is std_logic_vector ((WORD_LENGTH-1) downto 0);-- std_logic
vector of
length WORD_LENGTH
40 Subtype WORD_x2 is std_logic_vector ((2*WORD_LENGTH-1) downto 0);--
std_logic
vector of length 2*WORD_LENGTH
41 constant Z_WORD : WORD := (others => 'Z') ; -- WORD vector of 'Z' (high
impedance)
42 constant NULL_WORD : WORD := (others => '0') ; -- WORD vector of '0'
43 Subtype S_WORD is signed (WORD_LENGTH-1 downto 0);-- Signed variable of
length
WORD_LENGTH for internal operation
44 constant null_S_WORD : S_WORD := (others => '0') ; -- null S_WORD
45 constant min_S_WORD : S_WORD := (WORD_LENGTH-1 => '1', others => '0') ;
--
Smallest negative number in S_WORD
46 constant max_S_WORD : S_WORD := ( (WORD_LENGTH-1) => '0', others => '1')
;
--Largest positive number in S_WORD
47 Subtype S_WORD_p1 is signed (WORD_LENGTH downto 0);-- Signed variable of
length
WORD_LENGTH+1 to save addition results
48 Subtype S_WORD_x2 is signed((2*WORD_LENGTH-1) downto 0);-- Signed
variable of length 2*WORD_LENGTH to save multiplication results
49 Subtype counter_lp1 is integer range l downto 0;-- To count the l+1
states in the
Controller module.
50 --VECTORS
51 Subtype counter_l is natural range 1 to l ; -- variable type used as
counter up to l
52 Subtype counter_n is natural range 1 to n ; -- variable type used as
counter up to n
53 Subtype counter_UT is natural range 1 to ((n*n-n)/2) ; -- variable type
used as
counter up to (n*n-n)/2
54 Type WORDV_n is array (1 to n) of WORD;-- Vector of WORDs, length n
55 constant Z_WORDV_n : WORDV_n := (others => Z_WORD) ; -- WORDV_n vector
of Z_WORD,
this is all 'Z'
56 constant NULL_WORDV_n : WORDV_n := (others => NULL_WORD) ; -- WORDV_n
vector of
NULL_WORD, this is all '0'
57 Type WORDV_l is array (1 to l) of WORD;-- Vector of WORDs, length l
58 Type WORDV_UT is array (1 to ((n*n-n)/2)) of WORD;-- Vector of WORDs,
length
(n^2-2)/2 to transport unit tringular matrices
```

```
59 constant Z_WORDV_UT : WORDV_UT := (others => Z_WORD) ; -- WORDV_UT
vector of
Z_WORD, this is all 'Z'
60 constant NULL_WORDV_UT : WORDV_UT := (others => NULL_WORD) ; -- WORDV_UT
vector of
NULL_WORD, this is all '0'
61 Type S_WORDV_n is array (1 to n) of S_WORD ; -- Vector of S_WORDs,
length n
62 constant NULL_S_WORDV_n : S_WORDV_n := (others=>null_S_WORD); -- Null
S_WORDV_n
63 Type S_WORDV_l is array (1 to l) of S_WORD ; -- Vector of S_WORDs,
length l
64 Type S_WORDV_p is array (1 to p) of S_WORD ; -- Vector of S_WORDs,
length p
65 Type S_WORDV_UT is array (1 to ((n*n-n)/2)) of S_WORD ; -- Vector of
S_WORDs,
length (n^2-2)/2 to transport unit tringular matrices
66
67 --MATRICES
68 Type WORDM_nxn is array (1 to n, 1 to n) of WORD; -- Matrix of nxn WORDS
69 Type S_WORDM_lxn is array (1 to l, 1 to n) of S_WORD;-- Matrix of lxn
S_WORDS
70 Type S_WORDM_nxn is array (1 to n, 1 to n) of S_WORD;-- Matrix of nxn
S_WORDS
71 constant NULL_S_WORDM_nxn : S_WORDM_nxn :=
(others=>(others=>null_S_WORD)); --
Null S_WORDM_nxn
72 Type S_WORDM_nxp is array (1 to n, 1 to p) of S_WORD;-- Matrix of nxp
S_WORDS
73
74 end parameters;
```

### 11.2.3 System_Def.vhd

```
1 --Kalman Filter, Bierman-Thornton implementation in FPGA
2 --F.A.Iglesias (Leg:27310) and D.Dematties (Leg:22547)
3 --UTN-FRM 2012
4 ----------------------------------------------------------
5 --Description:
6 --
7 --INPUTS:
8 --
9 --OUTPUTS:
10 --
11 --Comments:
12 --
13 ----------------------------------------------------------
14 --Libraries:
15 library IEEE;
16 use IEEE.STD_LOGIC_1164.ALL;
17 use ieee.numeric_std.all;
18 library UNISIM; -- Used to simulate when using componetes of the library
19 use UNISIM.VComponents.all;
20 use IEEE.math_real.ALL;
21 use work.parameters.ALL;
22 ----------------------------------------------------------
23 --Package:
24 -- Initializes the case to be implemented:
25 -- IMPORTANT: Only the non zero and non unit values of each matrix must
be initialized
26 -- Time inveriant matrices of the system, sensors, and noises
27
```

```
28 package system_def is
29
30 --Decimal Numbers in Fixed point format (must be compatible with the
arithemtic
definition given in "parameters" pkg:
31 --NUMBERS VALID FOR: WORD_LENGTH=18 and FRACTION_LENGTH=6
32
33 constant s0_0 : signed := ("0000000000000000"); -- This group is used
in the
system definition
34 constant sP1_0 : signed := ("0000000000001000000");
35
36
37 -- System Matrices definition and initialization
38 constant H0 : S_WORDM_lxn := (1=>(1=>
"0000000000001000000",others=>s0_0),others => (
others=> s0_0));-- Measurement matrix (H)
39 constant R0 : S_WORDV_l := (1=> "0000000000001000000",others=> s0_0);--
Diagonal of
the Measurement noise covariance matrix (R)
40 constant Phi0: S_WORDM_nxn := (1=>(1=> "0000000000000111111",
2=>"0000000000000000000",
others=> s0_0),
2=>(1=>"1111111111110000",2=>"0000000000000111110",others=> s0_0),
others=>(others=>s0_0));-- State transition matrix (Phi)
41 constant G0 : S_WORDM_nxp := (1=> (others=>s0_0),2=>(others => sP1_0));-
- Coupling
factor. G0= G*Uq where G=Input coupling matrix and Uq => Qc=Uq*DQ0*Uq' (for
non
diagonal continuous Q )
42 constant DQ0 : S_WORDV_p := (1=>"0000000000100011110",others => s0_0);--
Diagonal
par of Qc. Qd=G*DQ0*G
43 constant uc : S_WORDV_n :=
(1=>"0000000000000000000",2=>"0000000000000000111",others
=>s0_0);-- Discrete Forcing term
44 ---initial conditions
45 constant X0 : S_WORDV_n := (others =>s0_0);-- Initial value of X
46 constant UP0 : S_WORDV_UT:= (others => s0_0);-- Initial value of U
(modified Cholesky
factor of P), only the upper triangle's values. 47 constant DP0 : S_WORDV_n
:= (1=>"0000000000010000000",2=> "0000000000010000000",others
=> s0_0);-- Initial value of D (modified Cholesky factor of P)
48
49
50 end system_def;
51
```

### 11.2.4 KF_BT.vhd

```
1 --Kalman Filter, Bierman-Thornton implementation in FPGA
2 --F.A.Iglesias (Leg:27310) and D.Dematties (Leg:22547)
3 --UTN-FRM 2012
4 ----------------------------------------------------------
5 --Description:
6 --
7 --INPUTS:
8 --
9 --OUTPUTS:
10 --
11 --Comments:
12 --
```

```
13 ----------------------------------------------------------
14 --Libraries:
15 library IEEE;
16 use IEEE.STD_LOGIC_1164.ALL;
17 use ieee.numeric_std.all;
18 library UNISIM; -- Used to simulate when using componetes of the library
19 use UNISIM.VComponents.all;
20
21 use work.parameters.ALL; -- Our packages
22 use work.system_def.ALL;
23 ----------------------------------------------------------
24
25 entity KF_BT is
26 Port ( Z : in WORD;
27 clk : in STD_LOGIC;
28 Reset : in STD_LOGIC;
29 CE : in STD_LOGIC;
30 E_X_1 : out WORD ; -- Two outputs beacause in post-route simulation
WORDV_n type is not supported
31 E_X_2 : out WORD );
32 end KF_BT;
33
34 architecture Structural of KF_BT is
35
36 --components declarations
37 component Bierman is
38 Port ( Z : in WORD;
39 Xp : in WORDV_n;
40 UPp : in WORDV_UT;
41 DPp : in WORDV_n;
42 Xpo : out WORDV_n;
43 UPpo : out WORDV_UT;
44 DPpo : out WORDV_n);
45 end component Bierman;
46
47 component Bierman_l is
48 Port ( Z : in WORD;
49 state : in l_STATE;
50 Xp : in WORDV_n;
51 UPp : in WORDV_UT;
52 DPp : in WORDV_n;
53 Xpo : out WORDV_n;
54 UPpo : out WORDV_UT;
55 DPpo : out WORDV_n);
56 end component Bierman_l;

57
58 component Thornton is
59 Port ( Xpo : in WORDV_n;
60 UPpo : in WORDV_UT;
61 DPpo : in WORDV_n;
62 Xp : out WORDV_n;
63 UPp : out WORDV_UT;
64 DPp : out WORDV_n);
65 end component Thornton;
66
67 component Data_path_controller is
68 Port ( Xin_B : in WORDV_n;
69 Xin_T : in WORDV_n;
70 UPin_B : in WORDV_UT;
71 UPin_T : in WORDV_UT;
72 DPin_B : in WORDV_n;
```

```
73 DPin_T : in WORDV_n;
74 CE : in STD_LOGIC;
75 clk : in STD_LOGIC;
76 Reset : in STD_LOGIC;
77 Xout_B : out WORDV_n;
78 Xout_T : out WORDV_n;
79 Xout_ext1 : out WORD ; -- Two outputs beacause in post-route simulation
WORDV_n type is not supported
80 Xout_ext2 : out WORD ;
81 UPout_B : out WORDV_UT;
82 UPout_T : out WORDV_UT;
83 DPout_B : out WORDV_n;
84 DPout_T : out WORDV_n);
85 end component Data_path_controller;
86
87 component Data_path_controller_l is
88 Port ( Xin_B : in WORDV_n;
89 Xin_T : in WORDV_n;
90 UPin_B : in WORDV_UT;
91 UPin_T : in WORDV_UT;
92 DPin_B : in WORDV_n;
93 DPin_T : in WORDV_n;
94 CE : in STD_LOGIC;
95 clk : in STD_LOGIC;
96 Reset : in STD_LOGIC;
97 Xout_B : out WORDV_n;
98 Xout_T : out WORDV_n;
99 Xout_ext1 : out WORD ; -- Two outputs beacause in post-route simulation
WORDV_n type is not supported
100 Xout_ext2 : out WORD ;
101 UPout_B : out WORDV_UT;
102 UPout_T : out WORDV_UT;
103 DPout_B : out WORDV_n;
104 DPout_T : out WORDV_n;
105 int_state : out l_STATE);
106 end component Data_path_controller_l;
107
108
109 --internal signals
110 signal Xp_B, Xpo_B, Xp_T, Xpo_T, DPp_B, DPpo_B, DPp_T, DPpo_T : WORDV_n
;
111 signal UPp_B, UPpo_B, UPp_T, UPpo_T: WORDV_UT ;

112 signal int_state : l_STATE;
113
114 begin
115
116 COND1: IF l=1 GENERATE
117 BEGIN
118 Bierman_module: Bierman
119 port map (Z,Xp_B,UPp_B,DPp_B, Xpo_B ,UPpo_B, DPpo_B);
120 END GENERATE COND1;
121
122 COND2: IF l>1 GENERATE
123 BEGIN
124 Bierman_module_l: Bierman_l
125 port map (Z,int_state,Xp_B,UPp_B,DPp_B, Xpo_B ,UPpo_B, DPpo_B);
126 END GENERATE COND2;
127
128
129 Thornton_module: Thornton
130 port map (Xpo_T ,UPpo_T, DPpo_T,Xp_T, UPp_T, DPp_T);
```

```
131
132
133 COND3: IF l=1 GENERATE
134 BEGIN
135 Data_Path: Data_path_controller
136 port map (Xpo_B, Xp_T, UPpo_B, UPp_T, DPpo_B, DPp_T, CE, clk, Reset,
Xp_B, Xpo_T,
E_X_1,E_X_2, UPp_B, UPpo_T, DPp_B, DPpo_T);
137 END GENERATE COND3;
138
139 COND4: IF l>1 GENERATE
140 BEGIN
141 Data_Path_l: Data_path_controller_l
142 port map (Xpo_B, Xp_T, UPpo_B, UPp_T, DPpo_B, DPp_T, CE, clk, Reset,
Xp_B, Xpo_T,
E_X_1,E_X_2, UPp_B, UPpo_T, DPp_B, DPpo_T,int_state);
143 END GENERATE COND4;
144
145 end Structural;
146
```

### 11.2.5 Bierman.vhd

```
1 --Kalman Filter, Bierman-Thornton implementation in FPGA
2 --F.A.Iglesias (Leg:27310) and D.Dematties (Leg:22547)
3 --UTN-FRM 2012
4 ----------------------------------------------------------
5 --Description:
6 --Implements the Bierman observational Update
7 --INPUTS:
8 --All the a priori values of the last iteration
9 --Xp, UPp, DPp
10 --And the measurements
11 --Z
12 --OUTPUTS:
13 --All the a posteriori values of the last iteration
14 --Xpo, UPpo, DPpo
15 --Comments:
16 --
17 ----------------------------------------------------------
18 --Libraries:
19 library IEEE;
20 use IEEE.STD_LOGIC_1164.ALL;
21 use ieee.numeric_std.all;
22 library UNISIM; -- Used to simulate when using componetes of the library
23 use UNISIM.VComponents.all;
24 use work.parameters.ALL; -- Our packages
25 use work.system_def.ALL;
26 use work.Fixed_Point_pkg.ALL;
27 ----------------------------------------------------------
28
29 entity Bierman is
30 Port ( Z : in WORD;
31 Xp : in WORDV_n;
32 UPp : in WORDV_UT;
33 DPp : in WORDV_n;
34 Xpo : out WORDV_n;
35 UPpo : out WORDV_UT;
36 DPpo : out WORDV_n);
37 end Bierman;
38
39 architecture Mixed of Bierman is
```

```
40 begin
41
42 scalar_bierman : process(Z, Xp, UPp, DPp) is
43
44 variable dz, alpha, gamma, beta, lambda, R_int, s_aux, s_aux1: S_WORD; -
- Internal
aux variable.
45 variable X, H_int, V, D, b : S_WORDV_n; -- Internal aux vectors.
46 variable U : S_WORDV_UT; -- Internal aux vector to store the unit upepr
triangular
matrix U.
47 variable i : counter_n ;-- used in the variable loops (while)
48 variable index_aux : counter_UT ; -- used as index of UT vector
49 variable index : counter_l;
50
51 begin
52
53 -----------load the input signals-------------------------
54
55 for k in 1 to n loop -- load and converts Xp to a internal variable
56 X(k) := signed(Xp(k));
57 end loop;
58
59 for k in 1 to ((n*n-n)/2) loop -- load and converts UPp to a internal
variable
60 U(k) := signed(UPp(k));
61 end loop;
62
63 for k in 1 to n loop -- load and converts DPp to a internal variable
64 D(k) := signed(DPp(k));
65 end loop;
66
67 for k in 1 to n loop -- load and converts H to a internal variable
68 H_int(k) := signed(H0(1,k));
69 end loop;
70
71 R_int := signed(R0(1));
72
73 dz := signed(Z);
74
75 -------------starts the Bierman algorithm-------------------------
76
77 for j in 1 to n loop
78 dz := fp_add(dz,fp_comp(resize_x2(H_int(j)*X(j))));-- computes Z-H*X
79 V(j) := H_int(j);
80 i := 1;
81 while true loop -- equivalent to: for in 1 to (j-1) (iterates j-1 times)
82 if i = j then
83 exit;
84 end if;
85 index_aux := j-n+(i*(2*n-i-1))/2 ; -- to find the element Uij inside the
U vector
86 V(j) := fp_add(V(j),resize_x2(U(index_aux)*H_int(i)));-- computes
V=U'*H'
87 i := i+1 ;
88 end loop;
89 end loop;
90
91 for k in 1 to n loop
92 b(k) := resize_x2(D(k)*V(k)) ; -- computes the b factor in the Grewall's
Bierman matlab algorithm (called the unescaled kalman gain)
93 end loop;
```

```
94
95 alpha := R_int ;
96 gamma := fp_reciprocal(alpha);
97
98
99 for j in 1 to n loop
100 beta := alpha;
101 alpha:= fp_add(alpha,resize_x2(V(j)*b(j))) ;
102 lambda:=fp_comp(resize_x2(V(j)*gamma)); --value s_aux complemanted
103 gamma := fp_reciprocal(alpha);
104 D(j):= resize_x2(beta*resize_x2(gamma*D(j)));
105 i := 1;
106 while true loop -- equivalent to: for in 1 to (j-1) (iterates j-1
times)
107 if i = j then
108 exit;
109 end if;
110 index_aux := j-n+(i*(2*n-i-1))/2 ; -- to find the element Uij inside
the
U vector
111 beta := U(index_aux);
112 U(index_aux):= fp_add(beta,resize_x2(b(i)*lambda));
113 b(i) := fp_add(b(i),resize_x2(b(j)*beta));
114 i := i+1 ;
115 end loop;
116 end loop;
117
118 for k in 1 to n loop
119 X(k):= fp_add(X(k),resize_x2(resize_x2(gamma*dz)*b(k)));
120 end loop;
121
122 for k in 1 to n loop -- assign the internal X to the output signal Xpo
123 Xpo(k) <= std_logic_vector(X(k)) ;
124 end loop;
125
126 for k in 1 to ((n*n-n)/2) loop -- assign the internal U to the output
signal UPpo
127 UPpo(k) <= std_logic_vector(U(k)) ;
128 end loop;
129
130 for k in 1 to n loop -- assign the internal D to the output signal DPpo
131 DPpo(k) <= std_logic_vector(D(k)) ;
132 end loop;
133
134 end process scalar_bierman;
135 end Mixed;
136
```

### 11.2.6 Thronton.vhd

```
1 --Kalman Filter, Bierman-Thornton implementation in FPGA
2 --F.A.Iglesias (Leg:27310) and D.Dematties (Leg:22547)
3 --UTN-FRM 2012
4 ----------------------------------------------------------
5 --Description:
6 --Implements the Thornton temporal update
7 --INPUTS:
8 --All the a posteriori values of this iteration
9 --Xpo, UPpo, DPpo
10 --OUTPUTS:
11 ----All the a priori values of the next iteration
12 --Xp, UPp, DPp
```

```
13 --Comments:
14 --
15 ----------------------------------------------------------
16 --Libraries:
17 library IEEE;
18 use IEEE.STD_LOGIC_1164.ALL;
19 use ieee.numeric_std.all;
20 library UNISIM; -- Used to simulate when using componetes of the library
21 use UNISIM.VComponents.all;
22 use work.parameters.ALL; -- Our packages
23 use work.system_def.ALL;
24 use work.Fixed_Point_pkg.ALL;
25 ----------------------------------------------------------
26
27 entity Thornton is
28 Port ( Xpo : in WORDV_n;
29 UPpo : in WORDV_UT;
30 DPpo : in WORDV_n;
31 Xp : out WORDV_n;
32 UPp : out WORDV_UT;
33 DPp : out WORDV_n);
34 end Thornton;
35
36 architecture Behavioral of Thornton is
37 begin
38
39 scalar_thornton: process (Xpo,UPpo, DPpo) is
40
41 variable sigma, s_aux : S_WORD; -- Internal aux variable.
42 variable X, Xout, V, D, Dout, b, uc_int : S_WORDV_n; -- Internal aux
vectors.
43 variable U, Uout : S_WORDV_UT; -- Internal aux vector to store the unit
upepr
triangular matrix U.
44 variable Phi_int, PhiU : S_WORDM_nxn;-- Internal aux nxn matrices
45 variable DQ_int : S_WORDV_p;-- Internal aux pxp matrices
46 variable G_int : S_WORDM_nxp;-- Internal aux pxp matrices
47 variable j,r : counter_n ;-- used in the variable loops (while)
48 variable index_aux, index_aux1 : counter_UT ; -- used as index of UT
vector
49
50 begin
51
52 -----------load the input signals-------------------------
53
54 for k in 1 to n loop -- load and converts Xp to a internal variable
55 X(k) := signed(Xpo(k));
56 end loop;
57
58 for k in 1 to ((n*n-n)/2) loop -- load and converts UPp to a internal
variable
59 U(k) := signed(UPpo(k));
60 end loop;
61
62 for k in 1 to n loop -- load and converts DPp to a internal variable
63 D(k) := signed(DPpo(k));
64 end loop;
65
66 for i in 1 to n loop
67 for j in 1 to n loop -- load and converts Phi to a internal variable
68 Phi_int(i,j) := Phi0(i,j);
69 end loop;
```

```
70 end loop;
71
72 for i in 1 to p loop -- load and converts DQ to a internal variable
73 DQ_int(i) := DQ0(i);
74 end loop;
75
76 for i in 1 to n loop -- load and converts G to a internal variable
77 for j in 1 to p loop
78 G_int(i,j) := G0(i,j);
79 end loop;
80 end loop;
81
82 for k in 1 to n loop -- load and converts c to a internal variable
83 uc_int(k) := uc(k);
84 end loop;
85
86 -------------starts the Thornton algorithm-------------------------
87
88 PhiU := NULL_S_WORDM_nxn ;
89 for i in 1 to n loop --PhiU = Phi*U(:,:,k)
90 for j in 1 to n loop
91 r:=1;
92 while true loop -- equivalent to: for r=1 to (j-1) but vhdl
implementable
93 if r = j then
94 exit;
95 end if;
96 index_aux := j-n+(r*(2*n-r-1))/2;-- to find the element Uij
97 PhiU(i,j):= fp_add(PhiU(i,j),resize_x2(Phi_int(i,r)*U(index_aux)));
98 r:=r+1;
99 end loop;
100 PhiU(i,j):= fp_add(PhiU(i,j),Phi_int(i,j));
101 end loop;
102 end loop;
103
104 for i in n downto 1 loop
105 sigma := null_S_WORD;
106 for j in 1 to n loop
107 sigma := fp_add(sigma, resize_x2(PhiU(i,j)*resize_x2(PhiU(i,j)*D(j))))
;
108 if (j < p+1) then
109 sigma := fp_add(sigma,resize_x2(G_int(i,j)*resize_x2(G_int(i,j)*
DQ_int(j)))) ;
110 end if;
111 end loop;
112 Dout(i) := sigma;
113 j:=1;
114 while true loop -- equivalent to: for j=1 to (i-1) but vhdl
implementable
115 if j = i then
116 exit;
117 end if;
118 sigma := null_S_WORD;
119 for s in 1 to n loop
120 sigma := fp_add(sigma,resize_x2(PhiU(i,s)*resize_x2(D(s)*PhiU(j,s))));
121 end loop;
122
123 for s in 1 to p loop
124 sigma := fp_add(sigma, resize_x2(G_int(i,s)*resize_x2(DQ_int(s)*G_int
(j,s))));
125 end loop;
```

```
126 index_aux := i-n+(j*(2*n-j-1))/2 ; -- to find the element Uji inside
the
U vector
127 Uout(index_aux) := resize_x2(fp_reciprocal(Dout(i))*sigma);
128 for s in 1 to n loop
129 PhiU(j,s) := fp_add(PhiU(j,s),fp_comp(resize_x2(Uout(index_aux)*PhiU(
i,s))));
130 end loop;
131 for s in 1 to p loop
132 G_int(j,s) := fp_add(G_int(j,s),fp_comp(resize_x2(Uout(index_aux)*
G_int(i,s)))) ;
133 end loop;
134 j := j+1;
135 end loop;
136 end loop;
137
138 --State estimation temporal update
139
140 Xout := NULL_S_WORDV_n;
141 for i in 1 to n loop
142 for r in 1 to n loop
143 Xout(i):= fp_add(Xout(i),resize_x2(Phi_int(i,r)*X(r)));
144 end loop;
145 Xout(i):= fp_add(Xout(i),uc_int(i));
146 end loop;
147
148 for k in 1 to n loop -- assign the internal X to the output signal Xpo
149 Xp(k) <= std_logic_vector(Xout(k)) ;
150 end loop;
151
152 for k in 1 to ((n*n-n)/2) loop -- assign the internal U to the output
signal UPpo
153 UPp(k) <= std_logic_vector(Uout(k)) ;
154 end loop;
155
156 for k in 1 to n loop -- assign the internal D to the output signal DPpo
157 DPp(k) <= std_logic_vector(Dout(k)) ;
158 end loop;
159
160 end process scalar_thornton;
161 end Behavioral;
162
```

### 11.2.7 Data_path_controller.vhd

```
1 --Kalman Filter, Bierman-Thornton implementation in FPGA
2 --F.A.Iglesias (Leg:27310) and D.Dematties (Leg:22547)
3 --UTN-FRM 2012
4 ----------------------------------------------------------
5 --Description:
6 --Redirects the values of X, UP and DP in 3 diff paths:
7 -- 1)From Bierman to Bierman : To compute the l iterations in the
sequential
measurement process.
8 -- 2)From Bierman to Thornton : To compute the temporal update
9 -- 3)From Thornton to Bierman : To compute the observational update
10
11 --INPUTS:
12 --Bierman output values: Xpo, DPpo, UPpo (Or the partial values in the l
Steps)
13 --Thornton output values: Xp, DPp, UPp
14 --Clk: Clock input
```

```
15 --CE:Chip Enable input
16 --Reset: Reset Input
17
18 --OUTPUTS:
19 --Bierman input values: Xp, DPp, UPp (Or the partial values in the l
Steps)
20 --Thornton inptu values: Xpo, DPpo, UPpo
21 --The KF-BT output: Xpo
22 --Comments:
23 --For more information refer to the user guide
24 ----------------------------------------------------------
25 --Libraries:
26 library IEEE;
27 use IEEE.STD_LOGIC_1164.ALL;
28 use ieee.numeric_std.all;
29 library UNISIM; -- Used to simulate when using componetes of the library
30 use UNISIM.VComponents.all;
31
32 use work.parameters.ALL; -- Our packages
33 use work.system_def.ALL;
34 ----------------------------------------------------------
35
36 entity Data_path_controller is
37 Port ( Xin_B : in WORDV_n;
38 Xin_T : in WORDV_n;
39 UPin_B : in WORDV_UT;
40 UPin_T : in WORDV_UT;
41 DPin_B : in WORDV_n;
42 DPin_T : in WORDV_n;
43 CE : in STD_LOGIC;
44 clk : in STD_LOGIC;
45 Reset : in STD_LOGIC;
46 Xout_B : out WORDV_n;
47 Xout_T : out WORDV_n;
48 Xout_ext1 : out WORD ; -- Two outputs beacause in post-route simulation
WORDV_n type is not supported
49 Xout_ext2 : out WORD ;
50 UPout_B : out WORDV_UT;
51 UPout_T : out WORDV_UT;
52 DPout_B : out WORDV_n;
53 DPout_T : out WORDV_n);
54 end Data_path_controller;
55
56 architecture Behavioral of Data_path_controller is
57 begin
58
59 secuential: process (clk,CE,Reset) is
60 variable state: counter_lp1 ;
61 variable X_int , DP_int : WORDV_n ;
62 variable UP_int : WORDV_UT ;
63 begin
64
65 If Reset='1' then
66
67 state := 1 ;
68
69 for k in 1 to n loop
70 Xout_B(k) <= std_logic_vector (X0(k)) ; -- Initial conditions
71 end loop;
72
73 for k in 1 to n loop
74 DPout_B(k) <= std_logic_vector (DP0(k));
```

```
75 end loop;
76
77 for k in 1 to ((n*n-n)/2) loop
78 UPout_B(k) <= std_logic_vector (UP0(k));
79 end loop;
80
81 elsif clk'event and clk='1' and CE='1' then
82
83 case state is
84
85 when 0 => -- Bierman Operation (1 time)
86 X_int := Xin_T;
87 DP_int := DPin_T;
88 UP_int := UPin_T;
89
90 Xout_ext1 <= X_int(1);
91 Xout_ext2 <= X_int(2) ;
92
93 Xout_B <= X_int ;
94 DPout_B <= DP_int;
95 UPout_B <= UP_int;
96
97 state := state + 1 ;
98
99 when l => -- Thornton Operation (1 time)
100
101 X_int := Xin_B;
102 DP_int := DPin_B;
103 UP_int := UPin_B;
104
105 Xout_ext1 <= X_int(1);
106 Xout_ext2 <= X_int(2) ;
107
108 Xout_T <= X_int ;
109 DPout_T <= DP_int;
110 UPout_T <= UP_int;
111
112 state := 0;
113 end case;
114
115 end if;
116 end process secuential;
117
118 end Behavioral;
119
```

### 11.2.8 Bierman_l.vhd

```
1 --Kalman Filter, Bierman-Thornton implementation in FPGA
2 --F.A.Iglesias (Leg:27310) and D.Dematties (Leg:22547)
3 --UTN-FRM 2012
4 ----------------------------------------------------------
5 --Description:
6 --
7 --INPUTS:
8 --
9 --OUTPUTS:
10 --
11 --Comments:
12 --
13 ----------------------------------------------------------
14 --Libraries:
```

```
15 library IEEE;
16 use IEEE.STD_LOGIC_1164.ALL;
17 use ieee.numeric_std.all;
18 library UNISIM; -- Used to simulate when using componetes of the library
19 use UNISIM.VComponents.all;
20 use work.parameters.ALL; -- Our packages
21 use work.system_def.ALL;
22 use work.Fixed_Point_pkg.ALL;
23 ----------------------------------------------------------
24
25
26
27
28 entity Bierman_l is
29 Port ( Z : in WORD;
30 state : in l_STATE;
31 Xp : in WORDV_n;
32 UPp : in WORDV_UT;
33 DPp : in WORDV_n;
34 Xpo : out WORDV_n;
35 UPpo : out WORDV_UT;
36 DPpo : out WORDV_n);
37 end Bierman_l;
38
39
40
41
42 architecture Mixed of Bierman_l is
43 begin
44
45 --scalar_bierman : process(Z, R, H, Xp, UPp, DPp) is
46 scalar_bierman : process(Z, Xp, UPp, DPp, state) is
47
48 variable dz, alpha, gamma, beta, lambda, R_int, s_aux, s_aux1: S_WORD; -
- Internal
aux variable.
49 variable X, H_int, V, D, b : S_WORDV_n; -- Internal aux vectors.
50 variable U : S_WORDV_UT; -- Internal aux vector to store the unit upepr
triangular
matrix U.
51 variable i : counter_n ;-- used in the variable loops (while)
52 variable index_aux : counter_UT ; -- used as index of UT vector
53 variable index : counter_l;
54
55 begin

56
57 -----------load the input signals-------------------------
58
59 for k in 1 to n loop -- load and converts Xp to a internal variable
60 X(k) := signed(Xp(k));
61 end loop;
62
63 for k in 1 to ((n*n-n)/2) loop -- load and converts UPp to a internal
variable
64 U(k) := signed(UPp(k));
65 end loop;
66
67 for k in 1 to n loop -- load and converts DPp to a internal variable
68 D(k) := signed(DPp(k));
69 end loop;
70
```

```
71 index := to_integer(unsigned(state)); -- Load the Step control
1<=index<=l
72
73 for k in 1 to n loop -- load and converts H to a internal variable
74 H_int(k) := signed(H0(index,k));
75 end loop;
76
77 R_int := signed(R0(index));
78
79 dz := signed(Z);
80
81 ------------starts the Bierman algorithm-------------------------
82
83 for j in 1 to n loop
84 dz := fp_add(dz,fp_comp(resize_x2(H_int(j)*X(j))));-- computes Z-H*X
85 V(j) := H_int(j);
86 i := 1;
87 while true loop -- equivalent to: for in 1 to (j-1) (iterates j-1 times)
88 if i = j then
89 exit;
90 end if;
91 index_aux := j-n+(i*(2*n-i-1))/2 ; -- to find the element Uij inside the
U vector
92 V(j) := fp_add(V(j),resize_x2(U(index_aux)*H_int(i)));-- computes
V=U'*H'
93 i := i+1 ;
94 end loop;
95 end loop;
96
97 for k in 1 to n loop
98 b(k) := resize_x2(D(k)*V(k)) ; -- computes the b factor in the Grewall's
Bierman matlab algorithm (called the unescaled kalman gain)
99 end loop;
100
101 alpha := R_int ;
102 gamma := fp_reciprocal(alpha);
103
104
105 for j in 1 to n loop
106 beta := alpha;
107 alpha:= fp_add(alpha,resize_x2(V(j)*b(j))) ;
108 lambda:=fp_comp(resize_x2(V(j)*gamma)); --value s_aux complemanted
109 gamma := fp_reciprocal(alpha);
110 D(j):= resize_x2(beta*resize_x2(gamma*D(j)));

111 i := 1;
112 while true loop -- equivalent to: for in 1 to (j-1) (iterates j-1
times)
113 if i = j then
114 exit;
115 end if;
116 index_aux := j-n+(i*(2*n-i-1))/2 ; -- to find the element Uij inside
the
U vector
117 beta := U(index_aux);
118 U(index_aux):= fp_add(beta,resize_x2(b(i)*lambda));
119 b(i) := fp_add(b(i),resize_x2(b(j)*beta));
120 i := i+1 ;
121 end loop;
122 end loop;
123
124 for k in 1 to n loop
```

```
125 X(k):= fp_add(X(k),resize_x2(resize_x2(gamma*dz)*b(k)));
126 end loop;
127
128 for k in 1 to n loop -- assign the internal X to the output signal Xpo
129 Xpo(k) <= std_logic_vector(X(k)) ;
130 end loop;
131
132 for k in 1 to ((n*n-n)/2) loop -- assign the internal U to the output
signal UPpo
133 UPpo(k) <= std_logic_vector(U(k)) ;
134 end loop;
135
136 for k in 1 to n loop -- assign the internal D to the output signal DPpo
137 DPpo(k) <= std_logic_vector(D(k)) ;
138 end loop;
139
140 end process scalar_bierman;
141 end Mixed;
```

### 11.2.9 Data_path_controller_l.vhd

```
1 ----------------------------------------------------------------------------
---------
2 -- Company:
3 -- Engineer:
4 --
5 -- Create Date: 18:45:58 04/20/2012
6 -- Design Name:
7 -- Module Name: Data_path_controller_l - Behavioral
8 ---Kalman Filter, Bierman-Thornton implementation in FPGA
9 --F.A.Iglesias (Leg:27310) and D.Dematties (Leg:22547)
10 --UTN-FRM 2012
11 ----------------------------------------------------------
12 --Description:
13 --
14 --INPUTS:
15 --
16 --OUTPUTS:
17 --
18 --Comments:
19 --
20 ----------------------------------------------------------
21 --Libraries:
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use ieee.numeric_std.all;
25 library UNISIM; -- Used to simulate when using componetes of the library
26 use UNISIM.VComponents.all;
27
28 use work.parameters.ALL; -- Our packages
29 use work.system_def.ALL;
30 ----------------------------------------------------------
31
32 entity Data_path_controller_l is
33 Port ( Xin_B : in WORDV_n;
34 Xin_T : in WORDV_n;
35 UPin_B : in WORDV_UT;
36 UPin_T : in WORDV_UT;
37 DPin_B : in WORDV_n;
38 DPin_T : in WORDV_n;
39 CE : in STD_LOGIC;
40 clk : in STD_LOGIC;
```

```
41 Reset : in STD_LOGIC;
42 Xout_B : out WORDV_n;
43 Xout_T : out WORDV_n;
44 Xout_ext1 : out WORD ; -- Two outputs beacause in post-route simulation
WORDV_n type is not supported
45 Xout_ext2 : out WORD ;
46 UPout_B : out WORDV_UT;
47 UPout_T : out WORDV_UT;
48 DPout_B : out WORDV_n;
49 DPout_T : out WORDV_n;
50 int_state : out l_STATE);
51 end Data_path_controller_l;
52
53 architecture Behavioral of Data_path_controller_l is
54 begin
55
56 secuential: process (clk,CE,Reset) is

57 variable state: counter_lp1 ;
58 variable X_int , DP_int : WORDV_n ;
59 variable UP_int : WORDV_UT ;
60 begin
61
62 If Reset='1' then
63
64 state := 1 ;
65 int_state <= std_logic_vector(to_unsigned(state,log2_l));
66
67 for k in 1 to n loop
68 Xout_B(k) <= std_logic_vector (X0(k)) ; -- Initial conditions
69 end loop;
70
71 for k in 1 to n loop
72 DPout_B(k) <= std_logic_vector (DP0(k));
73 end loop;
74
75 for k in 1 to ((n*n-n)/2) loop
76 UPout_B(k) <= std_logic_vector (UP0(k));
77 end loop;
78
79 elsif clk'event and clk='1' and CE='1' then
80
81 case state is
82
83 when 0 => -- Bierman Operation (1 time)
84 X_int := Xin_T;
85 DP_int := DPin_T;
86 UP_int := UPin_T;
87
88 Xout_ext1 <= X_int(1);
89 Xout_ext2 <= X_int(2) ;
90
91 Xout_B <= X_int ;
92 DPout_B <= DP_int;
93 UPout_B <= UP_int;
94
95 state := state + 1 ;
96 int_state <= std_logic_vector(to_unsigned(state,log2_l));
97
98 when 1 to (l-1) => -- Bierman Operation (l-1 times)
99 X_int := Xin_B;
100 DP_int := DPin_B;
```

```
101 UP_int := UPin_B;
102
103 Xout_B <= X_int ; -- Bierman Feedback
104 DPout_B <= DP_int;
105 UPout_B <= UP_int;
106
107 state := state + 1 ;
108 int_state <= std_logic_vector(to_unsigned(state,log2_l));
109
110 when l => -- Thornton Operation (1 time)
111 X_int := Xin_B;
112 DP_int := DPin_B;
113 UP_int := UPin_B;

114
115 Xout_ext1 <= X_int(1);
116 Xout_ext2 <= X_int(2) ;
117
118 Xout_T <= X_int ;
119 DPout_T <= DP_int;
120 UPout_T <= UP_int;
121
122 state := 0;
123 end case;
124
125 end if;
126 end process secuential;
127
128 end Behavioral;
```

### 11.2.10 Kalman_Filter_Test_Bench.vhd

```
1 --Kalman Filter, Bierman-Thornton implementation in FPGA
2 --F.A.Iglesias (Leg:27310) and D.Dematties (Leg:22547)
3 --UTN-FRM 2012
4 ----------------------------------------------------------
5 --Description:
6 --
7 --INPUTS:
8 --
9 --OUTPUTS:
10 --
11 --Comments:
12 --
13 --Libraries:
14 use work.parameters.ALL; -- Our packages
15 use work.system_def.ALL;
16 library IEEE;
17 use IEEE.STD_LOGIC_1164.ALL;
18 use ieee.numeric_std.all;
19
20 use ieee.std_logic_textio.all;
21 --library UNISIM; -- Used to simulate when using componetes of the
library
22 --use UNISIM.VComponents.all;
23
24 library std;
25 use std.textio.all;
26
27
28 ENTITY Kalman_Filter_Test_Bench IS
29 END Kalman_Filter_Test_Bench;
```

```
30
31 ARCHITECTURE behavior OF Kalman_Filter_Test_Bench IS
32
33 -- Component Declaration for the Unit Under Test (UUT)
34
35 COMPONENT KF_BT
36 Port ( Z : in WORD;
37 clk : in STD_LOGIC;
38 Reset : in STD_LOGIC;
39 CE : in STD_LOGIC;
40 E_X_1 : out WORD ; -- Two outputs beacause in post-route simulation
WORDV_n type is not supported
41 E_X_2 : out WORD );
42 END COMPONENT;
43
44
45 --Inputs
46 signal Z : WORD := (others => '0');
47 signal clk : std_logic := '0';
48 signal Reset : std_logic := '0';
49 signal CE : std_logic := '0';
50
51 --Outputs
52 signal E_X_1, E_X_2 : WORD;
53
54 -- Clock period definitions
55 constant clk_period : time := 250 ns;
56
57 BEGIN
58
59 -- Instantiate the Unit Under Test (UUT)
60 uut: KF_BT PORT MAP (
61 Z => Z,
62 clk => clk,
63 Reset => Reset,
64 CE => CE,
65 E_X_1 => E_X_1,
66 E_X_2 => E_X_2
67 );
68
69 -- Clock process definitions
70 clk_process :process
71 begin
72 clk <= '0';
73 wait for clk_period/2;
74 clk <= '1';
75 wait for clk_period/2;
76 end process;
77
78 -- read file process
79 read_input_file:process
80 variable inline, outline:line;
81 variable ReadV: STD_LOGIC;
82 variable end_of_line:boolean;
83 variable count: Integer;
84 file input_file:text is "data_in.dat";
85 file output_file:text is out "data_out.dat";
86 begin
87 count :=1;
88 CE<='0';
89 Reset<='1';
90 Z<="0000000000000000";
```

```
91 wait for clk_period*1/2;
92
93 while not endfile(input_file) loop
94 readline(input_file,inline);
95 for k in WORD_LENGTH-1 downto 0 loop
96 read(inline,ReadV,end_of_line);
97 Z(k)<=ReadV; -- read one STD_LOGIC value from file an load a complete Z
98 end loop;
99
100 if count=1 then -- the first time only
101 wait for clk_period*1/2;
102 CE<='1';
103 Reset<='0';
104 wait for clk_period*1/2;
105 wait for clk_period*(l+1/2);
106 count:=count+1;
107 write(outline,string'("---KF-BT-FP: Iglesias.F & Demmaties.D-UTN-FRM
2012---"));
108 writeline(output_file,outline);
109 write(outline,E_X_1);-- write the E_X to file
110 write(outline," ");
111 write(outline,E_X_2);
112 writeline(output_file,outline);
113 else
114 wait for clk_period*(l+1);-- one Z each 2 periods
115 write(outline,E_X_1);-- write the E_X to file
116 write(outline," ");
117 write(outline,E_X_2);
118 writeline(output_file,outline);
119 end if;
120 end loop;
121 wait;
122 end process read_input_file;
123
124 END;
125
```

## 11.3 Hoja de datos de la familia Spartan 6 de Xilinx

XILINX®

# Spartan-6 Family Overview

DS160 (v2.0) October 25, 2011 **Product Specification**

## General Description

The Spartan®-6 family provides leading system integration capabilities with the lowest total cost for high-volume applications. The thirteen-member family delivers expanded densities ranging from 3,840 to 147,443 logic cells, with half the power consumption of previous Spartan families, and faster, more comprehensive connectivity. Built on a mature 45 nm low-power copper process technology that delivers the optimal balance of cost, power, and performance, the Spartan-6 family offers a new, more efficient, dual-register 6-input look-up table (LUT) logic and a rich selection of built-in system-level blocks. These include 18 Kb (2 x 9 Kb) block RAMs, second generation DSP48A1 slices, SDRAM memory controllers, enhanced mixed-mode clock management blocks, SelectIO™ technology, power-optimized high-speed serial transceiver blocks, PCI Express® compatible Endpoint blocks, advanced system-level power management modes, auto-detect configuration options, and enhanced IP security with AES and Device DNA protection. These features provide a low-cost programmable alternative to custom ASIC products with unprecedented ease of use. Spartan-6 FPGAs offer the best solution for high-volume logic designs, consumer-oriented DSP designs, and cost-sensitive embedded applications. Spartan-6 FPGAs are the programmable silicon foundation for Targeted Design Platforms that deliver integrated software and hardware components that enable designers to focus on innovation as soon as their development cycle begins.

## Summary of Spartan-6 FPGA Features

- Spartan-6 Family:
  - Spartan-6 LX FPGA: Logic optimized
  - Spartan-6 LXT FPGA: High-speed serial connectivity
- Designed for low cost
  - Multiple efficient integrated blocks
  - Optimized selection of I/O standards
  - Staggered pads
  - High-volume plastic wire-bonded packages
- Low static and dynamic power
  - 45 nm process optimized for cost and low power
  - Hibernate power-down mode for zero power
  - Suspend mode maintains state and configuration with multi-pin wake-up, control enhancement
  - Lower-power 1.0V core voltage (LX FPGAs, -1L only)
  - High performance 1.2V core voltage (LX and LXT FPGAs, -2, -3, and -3N speed grades)
- Multi-voltage, multi-standard SelectIO™ interface banks
  - Up to 1,080 Mb/s data transfer rate per differential I/O
  - Selectable output drive, up to 24 mA per pin
  - 3.3V to 1.2V I/O standards and protocols
  - Low-cost HSTL and SSTL memory interfaces
  - Hot swap compliance
  - Adjustable I/O slew rates to improve signal integrity
- High-speed GTP serial transceivers in the LXT FPGAs
  - Up to 3.2 Gb/s
  - High-speed interfaces including: Serial ATA, Aurora, 1G Ethernet, PCI Express, OBSAI, CPRI, EPON, GPON, DisplayPort, and XAUI
- Integrated Endpoint block for PCI Express designs (LXT)
- Low-cost PCI® technology support compatible with the 33 MHz, 32- and 64-bit specification.
- Efficient DSP48A1 slices
  - High-performance arithmetic and signal processing
  - Fast 18 x 18 multiplier and 48-bit accumulator
  - Pipelining and cascading capability
  - Pre-adder to assist filter applications
- Integrated Memory Controller blocks
  - DDR, DDR2, DDR3, and LPDDR support
  - Data rates up to 800 Mb/s (12.8 Gb/s peak bandwidth)
  - Multi-port bus structure with independent FIFO to reduce design timing issues
- Abundant logic resources with increased logic capacity
  - Optional shift register or distributed RAM support
  - Efficient 6-input LUTs improve performance and minimize power
  - LUT with dual flip-flops for pipeline centric applications
- Block RAM with a wide range of granularity
  - Fast block RAM with byte write enable
  - 18 Kb blocks that can be optionally programmed as two independent 9 Kb block RAMs
- Clock Management Tile (CMT) for enhanced performance
  - Low noise, flexible clocking
  - Digital Clock Managers (DCMs) eliminate clock skew and duty cycle distortion
  - Phase-Locked Loops (PLLs) for low-jitter clocking
  - Frequency synthesis with simultaneous multiplication, division, and phase shifting
  - Sixteen low-skew global clock networks
- Simplified configuration, supports low-cost standards
  - 2-pin auto-detect configuration
  - Broad third-party SPI (up to x4) and NOR flash support
  - Feature rich Xilinx Platform Flash with JTAG
  - MultiBoot support for remote upgrade with multiple bitstreams, using watchdog protection
- Enhanced security for design protection
  - Unique Device DNA identifier for design authentication
  - AES bitstream encryption in the larger devices
- Faster embedded processing with enhanced, low cost, MicroBlaze™ soft processor
- Industry-leading IP and reference designs

## Spartan-6 FPGA Feature Summary

*Table 1:* **Spartan-6 FPGA Feature Summary by Device**

| Device | Logic Cells(1) | Configurable Logic Blocks (CLBs) | | | DSP48A1 Slices(3) | Block RAM Blocks | | CMTs(5) | Memory Controller Blocks (Max)(6) | Endpoint Blocks for PCI Express | Maximum GTP Transceivers | Total I/O Banks | Max User I/O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Slices(2) | Flip-Flops | Max Distributed RAM (Kb) | | 18 Kb(4) | Max (Kb) | | | | | | |
| XC6SLX4 | 3,840 | 600 | 4,800 | 75 | 8 | 12 | 216 | 2 | 0 | 0 | 0 | 4 | 132 |
| XC6SLX9 | 9,152 | 1,430 | 11,440 | 90 | 16 | 32 | 576 | 2 | 2 | 0 | 0 | 4 | 200 |
| XC6SLX16 | 14,579 | 2,278 | 18,224 | 136 | 32 | 32 | 576 | 2 | 2 | 0 | 0 | 4 | 232 |
| XC6SLX25 | 24,051 | 3,758 | 30,064 | 229 | 38 | 52 | 936 | 2 | 2 | 0 | 0 | 4 | 266 |
| XC6SLX45 | 43,661 | 6,822 | 54,576 | 401 | 58 | 116 | 2,088 | 4 | 2 | 0 | 0 | 4 | 358 |
| XC6SLX75 | 74,637 | 11,662 | 93,296 | 692 | 132 | 172 | 3,096 | 6 | 4 | 0 | 0 | 6 | 408 |
| XC6SLX100 | 101,261 | 15,822 | 126,576 | 976 | 180 | 268 | 4,824 | 6 | 4 | 0 | 0 | 6 | 480 |
| XC6SLX150 | 147,443 | 23,038 | 184,304 | 1,355 | 180 | 268 | 4,824 | 6 | 4 | 0 | 0 | 6 | 576 |
| XC6SLX25T | 24,051 | 3,758 | 30,064 | 229 | 38 | 52 | 936 | 2 | 2 | 1 | 2 | 4 | 250 |
| XC6SLX45T | 43,661 | 6,822 | 54,576 | 401 | 58 | 116 | 2,088 | 4 | 2 | 1 | 4 | 4 | 296 |
| XC6SLX75T | 74,637 | 11,662 | 93,296 | 692 | 132 | 172 | 3,096 | 6 | 4 | 1 | 8 | 6 | 348 |
| XC6SLX100T | 101,261 | 15,822 | 126,576 | 976 | 180 | 268 | 4,824 | 6 | 4 | 1 | 8 | 6 | 498 |
| XC6SLX150T | 147,443 | 23,038 | 184,304 | 1,355 | 180 | 268 | 4,824 | 6 | 4 | 1 | 8 | 6 | 540 |

**Notes:**

1. Spartan-6 FPGA logic cell ratings reflect the increased logic cell capability offered by the new 6-input LUT architecture.
2. Each Spartan-6 FPGA slice contains four LUTs and eight flip-flops.
3. Each DSP48A1 slice contains an 18 x 18 multiplier, an adder, and an accumulator.
4. Block RAMs are fundamentally 18 Kb in size. Each block can also be used as two independent 9 Kb blocks.
5. Each CMT contains two DCMs and one PLL.
6. Memory Controller Blocks are not supported in the -3N speed grade.

## Spartan-6 FPGA Device-Package Combinations and Available I/Os

Spartan-6 FPGA package combinations with the available I/Os and GTP transceivers per package are shown in Table 2. Due to the transceivers, the LX and LXT pinouts are not compatible.

*Table 2:* **Spartan-6 Device-Package Combinations and Maximum Available I/Os**

| Package | CPG196[1] | TQG144[1] | CSG225[2] | FT(G)256[3] | CSG324 | | FG(G)484[3,4] | | CSG484[4] | | FG(G)676[3] | | FG(G)900[3] | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Body Size (mm) | 8 x 8 | 20 x 20 | 13 x 13 | 17 x 17 | 15 x 15 | | 23 x 23 | | 19 x 19 | | 27 x 27 | | 31 x 31 | |
| Pitch (mm) | 0.5 | 0.5 | 0.8 | 1.0 | 0.8 | | 1.0 | | 0.8 | | 1.0 | | 1.0 | |
| Device | User I/O | User I/O | User I/O | User I/O | GTPs | User I/O | GTPs | User I/O | GTPs | User I/O | GTPs | User I/O | GTPs | User I/O |
| XC6SLX4 | 106 | 102 | 132 | | | | | | | | | | | |
| XC6SLX9 | 106 | 102 | 160 | 186 | NA | 200 | | | | | | | | |
| XC6SLX16 | 106 | | 160 | 186 | NA | 232 | | | | | | | | |
| XC6SLX25 | | | | 186 | NA | 226 | NA | 266 | | | | | | |
| XC6SLX45 | | | | | NA | 218 | NA | 316 | NA | 320 | NA | 358 | | |
| XC6SLX75 | | | | | | | NA | 280 | NA | 328 | NA | 408 | | |
| XC6SLX100 | | | | | | | NA | 326 | NA | 338 | NA | 480 | | |
| XC6SLX150 | | | | | | | NA | 338 | NA | 338 | NA | 498 | NA | 576 |
| XC6SLX25T | | | | | 2 | 190 | 2 | 250 | | | | | | |
| XC6SLX45T | | | | | 4 | 190 | 4 | 296 | 4 | 296 | | | | |
| XC6SLX75T | | | | | | | 4 | 268 | 4 | 292 | 8 | 348 | | |
| XC6SLX100T | | | | | | | 4 | 296 | 4 | 296 | 8 | 376 | 8 | 498 |
| XC6SLX150T | | | | | | | 4 | 296 | 4 | 296 | 8 | 396 | 8 | 540 |

**Notes:**
1. There is no memory controller on the devices in these packages.
2. Memory controller block support is x8 on the XC6SLX9 and XC6SLX16 devices in the CSG225 package. There is no memory controller in the XC6SLX4.
3. These devices are available in both Pb and Pb-free (additional G) packages as standard ordering options.
4. These packages support two of the four memory controllers in the XC6SLX75, XC6SLX75T, XC6SLX100, XC6SLX100T, XC6SLX150, and XC6SLX150T devices.

## Configuration

Spartan-6 FPGAs store the customized configuration data in SRAM-type internal latches. The number of configuration bits is between 3 Mb and 33 Mb depending on device size and user-design implementation options. The configuration storage is volatile and must be reloaded whenever the FPGA is powered up. This storage can also be reloaded at any time by pulling the PROGRAM_B pin Low. Several methods and data formats for loading configuration are available.

Bit-serial configurations can be either master serial mode, where the FPGA generates the configuration clock (CCLK) signal, or slave serial mode, where the external configuration data source also clocks the FPGA. For byte-wide configurations, master SelectMAP mode generates the CCLK signal while slave SelectMAP mode receives the CCLK signal for the 8- and 16-bit-wide transfer. In master serial mode, the beginning of the bitstream can optionally switch the clocking source to an external clock, which can be faster or more precise than the internal clock. The available JTAG pins use boundary-scan protocols to load bit-serial configuration data.

The bitstream configuration information is generated by the ISE® software using a program called BitGen. The configuration process typically executes the following sequence:

- Detects power-up (power-on reset) or PROGRAM_B when Low.
- Clears the whole configuration memory.
- Samples the mode pins to determine the configuration mode: master or slave, bit-serial or parallel.
- Loads the configuration data starting with the bus-width detection pattern followed by a synchronization word, checks for the proper device code, and ends with a cyclic redundancy check (CRC) of the complete bitstream.
- Starts a user-defined sequence of events: releasing the internal reset (or preset) of flip-flops, optionally waiting for the DCMs and/or PLLs to lock, activating the output drivers, and transitioning the DONE pin to High.

The Master Serial Peripheral Interface (SPI) and the Master Byte-wide Peripheral Interface (BPI) are two common methods used for configuring the FPGA. The Spartan-6 FPGA configures itself from a directly attached industry-standard SPI serial flash PROM. The Spartan-6 FPGA can configure itself via BPI when connected to an industry-standard parallel NOR flash. Note that BPI configuration is not supported in the XC6SLX4, XC6SLX25, and XC6SLX25T nor is BPI available when using Spartan-6 FPGAs in TQG144 and CPG196 packages.

Spartan-6 FPGAs support MultiBoot configuration, where two or more FPGA configuration bitstreams can be stored in a single configuration source. The FPGA application controls which configuration to load next and when to load it.

Spartan-6 FPGAs also include a unique, factory-programmed Device DNA identifier that is useful for tracking purposes, anti-cloning designs, or IP protection. In the largest devices, bitstreams can be copy protected using AES encryption.

### Readback

Most configuration data can be read back without affecting the system's operation.

## CLBs, Slices, and LUTs

Each configurable logic block (CLB) in Spartan-6 FPGAs consists of two slices, arranged side-by-side as part of two vertical columns. There are three types of CLB slices in the Spartan-6 architecture: SLICEM, SLICEL, and SLICEX. Each slice contains four LUTs, eight flip-flops, and miscellaneous logic. The LUTs are for general-purpose combinatorial and sequential logic support. Synthesis tools take advantage of these highly efficient logic, arithmetic, and memory features. Expert designers can also instantiate them.

### SLICEM

One quarter (25%) of Spartan-6 FPGA slices are SLICEMs. Each of the four SLICEM LUTs can be configured as either a 6-input LUT with one output, or as dual 5-input LUTs with identical 5-bit addresses and two independent outputs. These LUTs can also be used as distributed 64-bit RAM with 64 bits or two times 32 bits per LUT, as a single 32-bit shift register (SRL32), or as two 16-bit shift registers (SRL16s) with addressable length. Each LUT output can be registered in a flip-flop within the CLB. For arithmetic operations, a high-speed carry chain propagates carry signals upwards in a column of slices.

### SLICEL

One quarter (25%) of Spartan-6 FPGA slices are SLICELs, which contain all the features of the SLICEM except the memory/shift register function.

### SLICEX

One half (50%) of Spartan-6 FPGA slices are SLICEXs. The SLICEXs have the same structure as SLICELs except the arithmetic carry option and the wide multiplexers.

## Clock Management

Each Spartan-6 FPGA has up to six CMTs, each consisting of two DCMs and one PLL, which can be used individually or cascaded.

## DCM

The DCM provides four phases of the input frequency (CLKIN): shifted 0°, 90°, 180°, and 270° (CLK0, CLK90, CLK180, and CLK270). It also provides a doubled frequency CLK2X and its complement CLK2X180. The CLKDV output provides a fractional clock frequency that can be phase-aligned to CLK0. The fraction is programmable as every integer from 2 to 16, as well as 1.5, 2.5, 3.5 . . . 7.5. CLKIN can optionally be divided by 2. The DCM can be a zero-delay clock buffer when a clock signal drives CLKIN, while the CLK0 output is fed back to the CLKFB input.

### Frequency Synthesis

Independent of the basic DCM functionality, the frequency synthesis outputs CLKFX and CLKFX180 can be programmed to generate any output frequency that is the DCM input frequency ($F_{IN}$) multiplied by M and simultaneously divided by D, where M can be any integer from 2 to 32 and D can be any integer from 1 to 32.

### Phase Shifting

With CLK0 connected to CLKFB, all nine CLK outputs (CLK0, CLK90, CLK180, CLK270, CLK2X, CLK2X180, CLKDV, CLKFX, and CLKFX180) can be shifted by a common amount, defined as any integer multiple of a fixed delay. A fixed DCM delay value (fraction of the input period) can be established by configuration and can also be incremented or decremented dynamically.

### Spread-Spectrum Clocking

The DCM can accept and track typical spread-spectrum clock inputs, provided they abide by the input clock specifications listed in the *Spartan-6 FPGA Data Sheet: DC and Switching Characteristics*. Spartan-6 FPGAs can generate a spread-spectrum clock source from a standard fixed-frequency oscillator.

## PLL

The PLL can serve as a frequency synthesizer for a wider range of frequencies and as a jitter filter for incoming clocks in conjunction with the DCMs. The heart of the PLL is a voltage-controlled oscillator (VCO) with a frequency range of 400 MHz to 1,080 MHz, thus spanning more than one octave. Three sets of programmable frequency dividers (D, M, and O) adapt the VCO to the required application.

The pre-divider D (programmable by configuration) reduces the input frequency and feeds one input of the traditional PLL phase comparator. The feedback divider (programmable by configuration) acts as a multiplier because it divides the VCO output frequency before feeding the other input of the phase comparator. D and M must be chosen appropriately to keep the VCO within its controllable frequency range.

The VCO has eight equally spaced outputs (0°, 45°, 90°, 135°, 180°, 225°, 270°, and 315°). Each can be selected to drive one of the six output dividers, O0 to O5 (each programmable by configuration to divide by any integer from 1 to 128).

## Clock Distribution

Each Spartan-6 FPGA provides abundant clock lines to address the different clocking requirements of high fanout, short propagation delay, and extremely low skew.

### Global Clock Lines

In each Spartan-6 FPGA, 16 global-clock lines have the highest fanout and can reach every flip-flop clock. Global clock lines must be driven by global clock buffers, which can also perform glitchless clock multiplexing and the clock enable function. Global clocks are often driven from the CMTs, which can completely eliminate the basic clock distribution delay.

### I/O Clocks

I/O clocks are especially fast and serve only the localized input and output delay circuits and the I/O serializer/deserializer (SERDES) circuits, as described in the I/O Logic section.

## Block RAM

Every Spartan-6 FPGA has between 12 and 268 dual-port block RAMs, each storing 18 Kb. Each block RAM has two completely independent ports that share only the stored data.

### Synchronous Operation

Each memory access, whether read or write, is controlled by the clock. All inputs, data, address, clock enables, and write enables are registered. The data output is always latched, retaining data until the next operation. An optional output data pipeline register allows higher clock rates at the cost of an extra cycle of latency.

During a write operation in dual-port mode, the data output can reflect either the previously stored data, the newly written data, or remain unchanged.

### Programmable Data Width

- Each port can be configured as 16K × 1, 8K × 2, 4K × 4, 2K × 9 (or 8), 1K × 18 (or 16), or 512 x 36 (or 32).
- The x9, x18, and x36 configurations include parity bits. The two ports can have different aspect ratios.
- Each block RAM can be divided into two completely independent 9 Kb block RAMs that can each be configured to any aspect ratio from 8K x 1 to 512 x 18, with 256 x 36 supported in simple dual-port mode.

## Memory Controller Block

Most Spartan-6 devices include dedicated memory controller blocks (MCBs), each targeting a single-chip DRAM (either DDR, DDR2, DDR3, or LPDDR), and supporting access rates of up to 800 Mb/s.

The MCB has dedicated routing to predefined FPGA I/Os. If the MCB is not used, these I/Os are available as general purpose FPGA I/Os. The memory controller offers a complete multi-port arbitrated interface to the logic inside the Spartan-6 FPGA. Commands can be pushed, and data can be pushed to and pulled from independent built-in FIFOs, using conventional FIFO control signals. The multi-port memory controller can be configured in many ways. An internal 32-, 64-, or 128-bit data interface provides a simple and reliable interface to the MCB.

The MCB can be connected to 4-, 8-, or 16-bit external DRAM. The MCB, in many applications, provides a faster DRAM interface compared to traditional internal data buses, which are wider and are clocked at a lower frequency. The FPGA logic interface can be flexibly configured irrespective of the physical memory device. The MCB functionality is not supported in the -3N speed grade.

## Digital Signal Processing—DSP48A1 Slice

DSP applications use many binary multipliers and accumulators, best implemented in dedicated DSP slices. All Spartan-6 FPGAs have many dedicated, full-custom, low-power DSP slices, combining high speed with small size, while retaining system design flexibility.

Each DSP48A1 slice consists of a dedicated 18 × 18 bit two's complement multiplier and a 48-bit accumulator, both capable of operating at up to 390 MHz. The DSP48A1 slice provides extensive pipelining and extension capabilities that enhance speed and efficiency of many applications, even beyond digital signal processing, such as wide dynamic bus shifters, memory address generators, wide bus multiplexers, and memory-mapped I/O register files. The accumulator can also be used as a synchronous up/down counter. The multiplier can perform barrel shifting.

## Input/Output

The number of I/O pins varies from 102 to 576, depending on device and package size. Each I/O pin is configurable and can comply with a large number of standards, using up to 3.3V. The *Spartan-6 FPGA SelectIO Resources User Guide* describes the I/O compatibilities of the various I/O options. With the exception of supply pins and a few dedicated configuration pins, all other package pins have the same I/O capabilities, constrained only by certain banking rules. All user I/O is bidirectional; there are no input-only pins.

All I/O pins are organized in banks, with four banks on the smaller devices and six banks on the larger devices. Each bank has several common $V_{CCO}$ output supply-voltage pins, which also powers certain input buffers. Some single-ended input buffers require an externally applied reference voltage ($V_{REF}$). There are several dual-purpose $V_{REF}$-I/O pins in each bank. In a given bank, when I/O standard calls for a $V_{REF}$ voltage, each $V_{REF}$ pin in that bank must be connected to the same voltage rail and can not be used as an I/O pin.

## I/O Electrical Characteristics

Single-ended outputs use a conventional CMOS push/pull output structure, driving High towards $V_{CCO}$ or Low towards ground, and can be put into high-Z state. Many I/O features are available to the system designer to optionally invoke in each I/O in their design, such as weak internal pull-up and pull-down resistors, strong internal split-termination input resistors, adjustable output drive-strengths and slew-rates, and differential termination resistors. See the *Spartan-6 FPGA SelectIO Resources User Guide* for more details on available options for each I/O standard.

## I/O Logic

### Input and Output Delay

This section describes the available logic resources connected to the I/O interfaces. All inputs and outputs can be configured as either combinatorial or registered. Double data rate (DDR) is supported by all inputs and outputs. Any input or output can be individually delayed by up to 256 increments (except in the -1L speed grade). This is implemented as IODELAY2. The identical delay value is available either for data input or output. For a bidirectional data line, the transfer from input to output delay is automatic. The number of delay steps can be set by configuration and can also be incremented or decremented while in use.

Because these tap delays vary with supply voltage, process, and temperature, an optional calibration mechanism is built into each IODELAY2:

- For source synchronous designs where more accuracy is required, the calibration mechanism can (optionally) determine dynamically how many taps are needed to delay data by one full I/O clock cycle, and then programs the IODELAY2 with 50% of that value, thus centering the I/O clock in the middle of the data eye.
- A special mode is available only for differential inputs, which uses a phase-detector mechanism to determine whether the incoming data signal is being accurately sampled in the middle of the eye. The results from the phase-detector logic can be used to either increment or decrement the input delay, one tap at a time, to ensure error-free operation at very high bit rates.

### ISERDES and OSERDES

Many applications combine high-speed bit-serial I/O with slower parallel operation inside the device. This requires a serializer and deserializer (SerDes) inside the I/O structure. Each input has access to its own deserializer (serial-to-parallel converter) with programmable parallel width of 2, 3, or 4 bits. Where differential inputs are used, the two serializers can be cascaded to provide parallel widths of 5, 6, 7, or 8 bits. Each output has access to its own serializer (parallel-to-serial converter) with programmable parallel width of 2, 3, or 4 bits. Two serializers can be cascaded when a differential driver is used to give access to bus widths of 5, 6, 7, or 8 bits.

When distributing a double data rate clock, all SerDes data is actually clocked in/out at single data rate to eliminate the possibility of bit errors due to duty cycle distortion. This faster single data rate clock is either derived via frequency multiplication in a PLL, or doubled locally in each IOB by differentiating both clock edges when the incoming clock uses double data rate.

## Low-Power Gigabit Transceiver

Ultra-fast data transmission between ICs, over the backplane, or over longer distances is becoming increasingly popular and important. It requires specialized dedicated on-chip circuitry and differential I/O capable of coping with the signal integrity issues at these high data rates.

All Spartan-6 LXT devices have 2–8 gigabit transceiver circuits. Each GTP transceiver is a combined transmitter and receiver capable of operating at data rates up to 3.2 Gb/s. The transmitter and receiver are independent circuits that use separate PLLs to multiply the reference frequency input by certain programmable numbers between 2 and 25, to become the bit-serial data clock. Each GTP transceiver has a large number of user-definable features and parameters. All of these can be defined during device configuration, and many can also be modified during operation.

### Transmitter

The transmitter is fundamentally a parallel-to-serial converter with a conversion ratio of 8, 10, 16, or 20. The transmitter output drives the PC board with a single-channel differential current-mode logic (CML) output signal.

TXOUTCLK is the appropriately divided serial data clock and can be used directly to register the parallel data coming from the internal logic. The incoming parallel data is fed through a small FIFO and can optionally be modified with the 8B/10B algorithm to guarantee a sufficient number of transitions. The bit-serial output signal drives two package pins with complementary CML signals. This output signal pair has programmable signal swing as well as programmable pre-emphasis to compensate for PC board losses and other interconnect characteristics.

### Receiver

The receiver is fundamentally a serial-to-parallel converter, changing the incoming bit serial differential signal into a parallel stream of words, each 8, 10, 16, or 20 bits wide. The receiver takes the incoming differential data stream, feeds it through a programmable equalizer (to compensate for the PC board and other interconnect characteristics), and uses the $F_{REF}$ input to initiate clock recognition. There is no need for a separate clock line. The data pattern uses non-return-to-zero (NRZ) encoding and optionally guarantees sufficient data transitions by using the 8B/10B encoding scheme. Parallel data is then transferred into the FPGA logic using the RXUSRCLK clock. The serial-to-parallel conversion ratio can be 8, 10, 16, or 20.

## Integrated Endpoint Block for PCI Express Designs

The PCI Express standard is a packet-based, point-to-point serial interface standard. The differential signal transmission uses an embedded clock, which eliminates the clock-to-data skew problems of traditional wide parallel buses.

The PCI Express Base Specification 1.1 defines bit rate of 2.5 Gb/s per lane, per direction (transmit and receive). When using 8B/10B encoding, this supports a data rate of 2.0 Gb/s per lane.

The Spartan-6 LXT devices include one integrated Endpoint block for PCI Express technology that is compliant with the PCI Express Base Specification Revision 1.1. This block is highly configurable to system design requirements and operates as a compliant single lane Endpoint. The integrated Endpoint block interfaces to the GTP transceivers for serialization/de-serialization, and to block RAMs for data buffering. Combined, these elements implement the physical layer, data link layer, and transaction layer of the protocol.

Xilinx provides a light-weight (<200 LUT), configurable, easy-to-use LogiCORE™ IP that ties the various building blocks (the integrated Endpoint block for PCI Express technology, the GTP transceivers, block RAM, and clocking resources) into a compliant Endpoint solution. The system designer has control over many configurable parameters: maximum payload size, reference clock frequency, and base address register decoding and filtering.

More information and documentation on solutions for PCI Express designs can be found at:

http://www.xilinx.com/technology/protocols/pciexpress.htm

## Spartan-6 FPGA Ordering Information

Table 3 shows the speed and temperature grades available in the different Spartan-6 devices. Some devices might not be available in every speed and temperature grade.

*Table 3:* **Speed Grade and Temperature Ranges**

| Device Family | Speed Grade and Temperature Range | |
|---|---|---|
| | Commercial (C) 0°C to +85°C | Industrial (I) –40°C to +100°C |
| Spartan-6 LX | -3, -3N, -2, -1L | -3, -3N, -2, -1L |
| Spartan-6 LXT | -3, -3N, -2 | -3, -3N, -2 |

The Spartan-6 FPGA ordering information shown in Figure 1 applies to all packages, including Pb-Free. Refer to the Package Marking section of UG385, *Spartan-6 FPGA Packaging and Pinouts* for a more detailed explanation of the device markings.

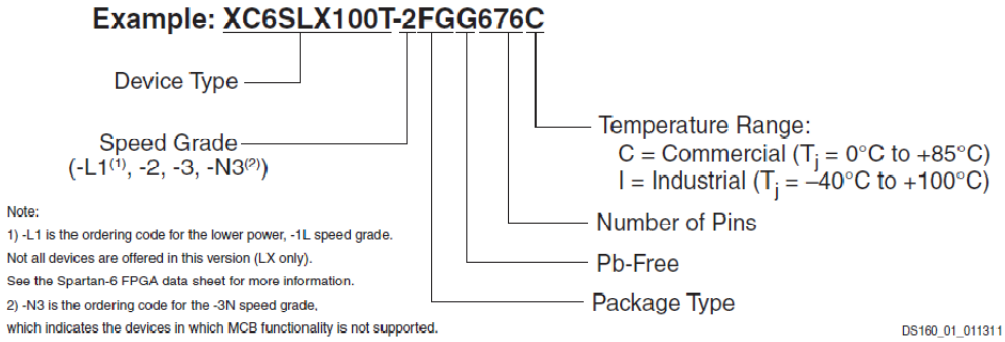



*Figure 1:* **Spartan-6 FPGA Ordering Information**

## Revision History

The following table shows the revision history for this document:

| Date | Version | Description of Revisions |
|---|---|---|
| 02/02/09 | 1.0 | Initial Xilinx release. |
| 05/05/09 | 1.1 | Updated and simplified Designed for low cost, Multi-voltage, multi-standard SelectIO™ interface banks, and Integrated Memory Controller blocks sections on page 1. Clarified PCI support on page 1is only for the 33 MHz specification. Revised number of logic cells, slices, and maximum user I/O, and added number of flip-flops to Table 1. In Table 2, revised user I/O counts, removed the XC6SLX25 in the CSG225 package and the XC6SLX45T in the FGG676 package, added XC6SLX9 in the FT(G)256 package and XC6SLX45 in the CSG324 package, and added notes. Clerical edits to the following sections: Dynamic Reconfiguration Port, Readback, CLBs, Slices, and LUTs, Frequency Synthesis, PLL, Programmable Data Width, and Memory Controller Block. Clarified I/O pin range, $V_{REF}$ banks, and electrical characteristics in the Input/Output section. |
| 06/24/09 | 1.2 | Updated device/package combinations in Table 1 and Table 2 including adding the XC6SLX75 and XC6SLX75T devices. Added ordering information and FPGA documentation sections. Removed partial reconfiguration discussion from the Readback section. |
| 11/05/09 | 1.3 | Updated Figure 1, page 9 to show -4 speed grade. Added 64-bit PCI support on page 1. Updated User I/O numbers in Table 1and Table 2. Clarifying edits to these sections: Configuration, Digital Signal Processing—DSP48A1 Slice, Input/Output, and PCI Express documentation. |

| Date | Version | Description of Revisions |
|---|---|---|
| 03/03/10 | 1.4 | Updated the slice counts for the LX25 and LX25T in Table 1. Revised the Dynamic Reconfiguration Port section. Added to the Spread-Spectrum Clocking section. Changed the PLL VCO maximum frequency to 1080 MHz and the DSP48A1 slice maximum frequency to 320 MHz due to the addition of the -4 speed specification. Clarified configurations in the Programmable Data Width section. Updated Low-Power Gigabit Transceiver operating rate. |
| 08/02/10 | 1.5 | Updated data transfer rate per differential I/O from 1,050 Mb/s to 1,080 Mb/s in Summary of Spartan-6 FPGA Features. Added the -3N speed grade to appropriate section throughout the document, including Figure 1. Updated category in Table 2 from Size to Body Size. Updated the Configuration section with SPI and BPI interface information. Removed the Dynamic Reconfiguration Port section. Updated the operating speed of the DSP48A1 slice multiplier and accumulator to 390 MHz in Digital Signal Processing—DSP48A1 Slice. Updated Input and Output Delay. |
| 11/05/10 | 1.6 | In Summary of Spartan-6 FPGA Features and in Low-Power Gigabit Transceiver, updated GTP serial transceiver data rate to 3.2 Gb/s. Updated the notes in Figure 1. Added DS170, *XA Spartan-6 Automotive FPGA Family Overview* to the Spartan-6 FPGA Documentation. |
| 03/21/11 | 1.7 | Updated from Advance to Preliminary Specification. Removed -4 speed grade from Summary of Spartan-6 FPGA Features and Figure 1 per XCN11008. Updated Integrated Endpoint Block for PCI Express Designs and Input and Output Delay. |
| 10/25/11 | 2.0 | Updated from Preliminary Specification to Production Specification. Updated Configuration, including the range of configuration bits to 3 Mb and 33 Mb, and Input and Output Delay. Updated Spartan-6 FPGA Ordering Information and added Table 3. Added Defense-Grade Spartan-6Q Family Overview (DS172) to list of documentation. |

## Notice of Disclaimer

## Spartan-6 FPGA Documentation

Complete and up-to-date documentation of the Spartan-6 family of FPGAs is available on the Xilinx website at http://www.xilinx.com/support/documentation/spartan-6.htm. In addition to the most recent *Spartan-6 Family Overview*, the following files are also available for download:

***Spartan-6 FPGA Data Sheet: DC and Switching Characteristics*** (DS162)

This data sheet contains the DC and Switching Characteristic specifications for the Spartan-6 family.

***Spartan-6 FPGA Packaging and Pinout Specifications*** (UG385)

These specifications includes the tables for device/package combinations and maximum I/Os, pin definitions, pinout tables, pinout diagrams, mechanical drawings, and thermal specifications.

***Spartan-6 FPGA Configuration Guide*** (UG380)

This all-encompassing configuration guide includes chapters on configuration interfaces (serial and parallel), multi-bitstream management, bitstream encryption, boundary-scan and JTAG configuration, and reconfiguration techniques.

***Spartan-6 FPGA SelectIO Resources User Guide*** (UG381)

This guide describes the SelectIO™ resources available in all the Spartan-6 devices.

***Spartan-6 FPGA Clocking Resources User Guide*** (UG382)

This guide describes the clocking resources available in all Spartan-6 devices, including the DCMs and the PLLs.

***Spartan-6 FPGA Block RAM Resources User Guide*** (UG383)

This guide describes the Spartan-6 device block RAM capabilities.

***Spartan-6 FPGA Configurable Logic Blocks User Guide*** (UG384)

This guide describes the capabilities of the configurable logic blocks (CLB) available in all Spartan-6 devices.

***Spartan-6 FPGA GTP Transceivers User Guide*** (UG386)

This guide describes the GTP transceivers available in all the Spartan-6 LXT FPGAs.

***Spartan-6 FPGA DSP48A1 Slice User Guide*** (UG389)

This guide describes the architecture of the DSP48A1 slice in Spartan-6 FPGAs and provides configuration examples.

***Spartan-6 FPGA Memory Controller User Guide*** (UG388)

This guide describes the Spartan-6 FPGA memory controller block, a dedicated, embedded multi-port memory controller that greatly simplifies interfacing Spartan-6 FPGAs to the most popular memory standards.

***Spartan-6 FPGA PCB Design and Pin Planning Guide*** (UG393)

This guide provides information on PCB design for Spartan-6 devices, with a focus on strategies for making design decisions at the PCB and interface level.

***Spartan-6 FPGA Power Management User Guide*** (UG394)

This document provides information on the various hardware methods of power management in Spartan-6 FPGAs, primarily focusing on the suspend mode.

***XA Spartan-6 Automotive FPGA Family Overview*** (DS170)

This overview outlines the features and product selection of the Xilinx Automotive (XA) Spartan-6 family.

***Defense-Grade Spartan-6Q Family Overview*** (DS172)

This overview outlines the features and product selection of the Defense-Grade Spartan-6Q family.

## 11.4 Explicaciones teóricas adicionales

### 11.4.1 Ortogonalización de Gram-Schmidt

Las matrices cuyas columnas forman un conjunto ortonormal son muy importantes para una diversidad de aplicaciones incluidas los algoritmos computacionales que operan con las mismas. Este tipo de matrices goza de diversas propiedades, aquí se hace referencia a la siguiente:

- Si la matriz $U_{nxm}$ tiene $m$ columnas ortonormales entre si $u_1, u_2, \dots, u_m$de dimensión $n$:

$$U = [u_1 \quad u_2 \quad \cdots \quad u_m]$$

$$U^T = [u_1 \quad u_2 \quad \cdots \quad u_m]^T = \begin{bmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_m^T \end{bmatrix}$$

$$U^T U = \begin{bmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_m^T \end{bmatrix} [u_1 \quad u_2 \quad \cdots \quad u_m] = \begin{bmatrix} u_1^T u_1 & u_1^T u_2 & \cdots & u_1^T u_m \\ u_2^T u_1 & u_2^T u_2 & \cdots & u_2^T u_m \\ \vdots & \vdots & \ddots & \vdots \\ u_m^T u_1 & u_m^T u_2 & \cdots & u_m^T u_m \end{bmatrix}$$

  Todos los elementos de la última matriz son productos internos entre vectores. Sabiendo que las columnas que forman $U_{nxm}$ son ortonormales:

$$u_i^T u_j = u_j u_i^T = \begin{cases} 0 \; si \; i \neq j \\ 1 \; si \; i = j \end{cases}$$

$$U^T U = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

  Por lo tanto, una matriz ortonormal es una matriz $U$ que es invertible y cuadrada tal que $U^{-1} = U^T$.
  Resumiendo, se puede decir que cualquier matriz con columnas ortonormales es una matriz ortogonal y lo más extraño es que dicha matriz debe tener filas ortonormales también.

El proceso de ortogonalización de Gram-Schmidt se usa para encontrar un conjunto ortogonal de vectores $\{b_i\}_1^n$ de dimensión $m$ que son una combinación lineal de un conjunto de partida de vectores linealmente independientes $\{a_i\}_1^n$ de dimensión $m$. Este método se utiliza comúnmente para encontrar una base ortogonal u ortonormal para cualquier subespacio que es diferente de cero y que tiene dimensión $m$.

Por ejemplo si $W$ es un subespacio generado $(G)$ de dimensión $m = 3$:

$$W = G\{a_1, a_2\}$$

Donde:

$$a_1 = \begin{bmatrix} 3 \\ 6 \\ 0 \end{bmatrix}$$

$$a_2 = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$$

Si se quisiera construir una base ortogonal $\{b_1, b_2\}$ para $W$ como la que aparece en la Figura 11-1:

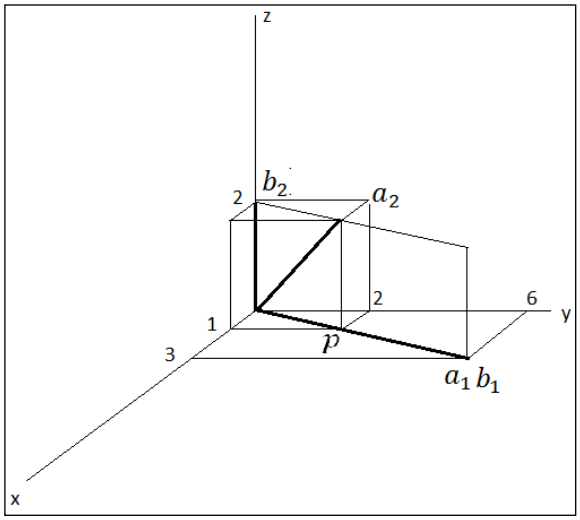



**Figura 11-1**

La componente de $a_2$ que es ortogonal a $a_1$ es $a_2 - p$ donde $p$ es la proyección de $a_2$ sobre $a_1$. El vector resultante de $a_2 - p$ está íntegramente contenido por $W$ debido a que aquel está formado completamente por $a_2$ y un múltiplo de $a_1$.

Por lo tanto si $b_1 = a_1$ se tiene que:

$$b_2 = a_2 - p = a_2 - \frac{a_2 a_1}{a_1 a_1} a_1 = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} - \frac{15}{45} \begin{bmatrix} 3 \\ 6 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$$

Por lo tanto $\{b_1, b_2\}$ es un conjunto de vectores ortogonales cuyos vectores son distintos de cero. Se concluye entonces que $\{b_1, b_2\}$ es una base para $W$.

El siguiente ejemplo clarificará la generalización del proceso:

Si $\{a_1, a_2, a_3\}$ es linealmente independiente y, por lo tanto, constituye una base para un subespacio $W$ de dimensión $m = 4$ tal que:

$$a_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$a_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$a_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Si se quiere construir una base ortogonal para $W$ se debe proceder de la siguiente manera:

- Se debe tomar $b_1 = a_1$. De esta forma se genera el subespacio $W_1 = G\{a_1\} = G\{b_1\}$
- Se debe construir $v_2$ substrayendo de $x_2$ su proyección sobre $W_1$ esto es:

$$b_2 = a_2 - Proy_{W_1}(a_2)$$

$$b_2 = a_2 - \frac{a_2 b_1}{b_1 b_1} b_1 \; siendo \; b_1 = b_1$$

$$b_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \frac{3}{4}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -3/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}$$

Por lo tanto $b_2$ es la componente de $a_2$ que es ortogonal a $a_1$ y $\{b_1, b_2\}$ es una base ortogonal para el subespacio $W_2$ que es generado por $a_1$ y $a_2$, $W_2 = G\{a_1, a_2\} = G\{b_1, b_2\}$.

- Debido a que $b_2$ tiene elementos fraccionales se hace conveniente su escalado por un factor de 4 y reemplazar $\{b_1, b_2\}$ por:
-

$$b_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\acute{b}_2 = \begin{bmatrix} -3 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

- Se debe construir $b_3$ sustrayendo de $a_3$ su proyección sobre el subespacio $W_2$, para ello se debe utilizar la base $\{b_1, \acute{b}_2\}$.

$$b_3 = a_3 - Proy_{W_2}(a_3)$$

$$b_3 = a_3 - \frac{a_3 b_1}{b_1 b_1} b_1 - \frac{a_3 \acute{b}_2}{\acute{b}_2 \acute{b}_2} \acute{b}_2$$

$$v_3 = \begin{bmatrix}0\\0\\1\\1\end{bmatrix} - \frac{2}{4}\begin{bmatrix}1\\1\\1\\1\end{bmatrix} - \frac{2}{12}\begin{bmatrix}-3\\1\\1\\1\end{bmatrix} = \begin{bmatrix}0\\-2/3\\1/3\\1/3\end{bmatrix}$$

Se concluye que $\{b_1, \acute{b}_2, b_3\}$ es una base ortogonal para $W$.

A los efectos de generalizar el proceso de la ortogonalización de Gram-Schmidt se puede enunciar que si se dispone de una base $\{a_i\}_1^n = \{a_1, a_2, \dots, a_n\}$ de dimensión $m$.para un subespacio $W$, se puede definir mediante el siguiente proceso:

$$b_1 = a_1$$

$$b_2 = a_2 - \frac{a_2 b_1}{b_1 b_1} b_1$$

$$b_3 = a_3 - \frac{a_3 b_1}{b_1 b_1} b_1 - \frac{a_3 b_2}{b_2 b_2} b_2$$

$$\vdots$$

$$b_n = a_n - \frac{a_n b_1}{b_1 b_1} b_1 - \frac{a_n b_2}{b_2 b_2} b_2 - \cdots - \frac{a_n b_{n-1}}{b_{n-1} b_{n-1}} b_{n-1}$$

**Ecuación 11-1**

Concluyendo, de esta manera, que $\{b_i\}_1^n = \{b_1, b_2, \dots, b_n\}$ es una base ortogonal para $W$.

Para adecuarlo a nuestro caso, tomamos una matriz $A_{mxn}$, y la escribimos como un conjunto de $n$ vectores $m$ dimensionales, $\{a_i\}_1^n$. Entonces el algoritmo de Gram-Schmidt asocia a $A_{mxn}$ una matriz $B_{mxn}$ tal que:

$$A_{mxn} = \begin{bmatrix}\acute{a}_{11} & \cdots & \acute{a}_{1n}\\ \vdots & \ddots & \vdots\\ \acute{a}_{m1} & \cdots & \acute{a}_{mn}\end{bmatrix} = [a_1 \quad \cdots \quad a_n]$$

$$B_{mxn} = \begin{bmatrix}\acute{b}_{11} & \cdots & \acute{b}_{1n}\\ \vdots & \ddots & \vdots\\ \acute{b}_{m1} & \cdots & \acute{b}_{mn}\end{bmatrix} = [b_1 \quad \cdots \quad b_n]$$

$$b_i^T b_j = \begin{cases}\|b_i\|^2 \; si \; i = j\\ 0 \quad si \; i \neq j\end{cases}$$

**Ecuación 11-2**

Donde $L$ es una matriz triangular inferior unitaria tal que:

$$A_{mxn} = B_{mxn} L_{nxn}$$

$$[a_1 \quad a_2 \quad \cdots \quad a_n] = [b_1 \quad b_2 \quad \cdots \quad b_n]\begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{bmatrix}$$

$$= [(b_1 + b_2 l_{21} + \cdots + b_n l_{n1}) \quad (b_2 + \cdots + b_n l_{n2}) \quad \cdots \quad (b_n)] = A_{mxn}$$

$$A_{mxn} = \begin{bmatrix} (b_1 + b_2 l_{21} + \cdots + b_n l_{n1}) \\ (b_2 + \cdots + b_n l_{n2}) \\ \vdots \\ (b_n) \end{bmatrix}^T = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}^T$$

De donde despejando las igualdades escalares se tiene:

$$b_1 = a_1 - b_n l_{n1} - \cdots - b_2 l_{21}$$

$$b_2 = a_2 - b_n l_{n2} - \cdots$$

$$\vdots$$

$$b_n = a_n$$

Donde según la Ecuación 11-1 las componentes de la matriz triangular superior unitaria *L* son:

$$l_{n1} = \frac{a_1 b_n}{b_n b_n}$$

$$l_{21} = \frac{a_1 b_2}{b_2 b_2}$$

$$l_{n2} = \frac{a_2 b_n}{b_n b_n}$$

Se tiene de la Ecuación 11-2 que los vectores que constituyen las columnas de la matriz $B$ son ortogonales, por lo tanto:

$$B^T B = [b_1 \quad b_2 \quad \cdots \quad b_n]^T [b_1 \quad b_2 \quad \cdots \quad b_n] = \begin{bmatrix} {b_1}^T \\ {b_2}^T \\ \vdots \\ {b_n}^T \end{bmatrix} [b_1 \quad b_2 \quad \cdots \quad b_n]$$

$$= \begin{bmatrix} \|b_1\|^2 & 0 & \cdots & 0 \\ 0 & \|b_2\|^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \|b_n\|^2 \end{bmatrix} = D_{\|b\|^2}$$

Ortogonalización de Gram-Schmidt sopesada:

Si los vectores que definen $B$, a saber $\{b_i\}_1^n$ son ortogonales entre sí con respecto a funciones de peso definidas en la diagonal de la matriz diagonal de peso. $D_w$.

De esta manera:

$$B^T D_w B = [b_1 \quad b_2 \quad \cdots \quad b_n]^T \begin{bmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_m \end{bmatrix} [b_1 \quad b_2 \quad \cdots \quad b_n]$$

$$= \begin{bmatrix} {b_1}^T \\ {b_2}^T \\ \vdots \\ {b_n}^T \end{bmatrix} \begin{bmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_m \end{bmatrix} [b_1 \quad b_2 \quad \cdots \quad b_n]$$

$$= \begin{bmatrix} \|b_1\|^2 & 0 & \cdots & 0 \\ 0 & \|b_2\|^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \|b_n\|^2 \end{bmatrix} = D_{\|b\|^2}$$

$$b_i^T D_w b_j = \begin{cases} \|b_i\|^2 \; si \; i = j \\ 0 \quad si \; i \neq j \end{cases}$$

Entonces $B$ se denomina una *matriz ortogonal* y cumple con:

$$B^T D_w B = D_{\|b\|^2}$$

Donde $D_{\|b\|}$es una matriz diagonal de $nxn$ con los elementos de su diagonal principal en forma ascendente igual a las normas $\|b_1\|, \|b_2\|, \ldots, \|b_n\|$ respectivamente. Para $B$ ortonormal $D_{\|b\|} = I$.

En la ortogonalización de Gram-Schmidt no sopesada o normal $D_w = I$.

### 11.4.2 Implementación de Carlson-Schmidt

Actualización observacional de Carlson, “Triangular rápida”

Este es un algoritmo de modificación de rango 1 que genera un factor de Cholesky triangular superior $W$.

La $W$ es triangular superior de $nxn$ tal que:

$$WW^T = I - \frac{vv^T}{r + |v|^2}$$

Entonces si $W$ es una matriz de $3x3$:

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ 0 & w_{22} & w_{23} \\ 0 & 0 & w_{33} \end{bmatrix}$$

Se tiene que:

$$WW^T = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ 0 & w_{22} & w_{23} \\ 0 & 0 & w_{33} \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ 0 & w_{22} & w_{23} \\ 0 & 0 & w_{33} \end{bmatrix}^T = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ 0 & w_{22} & w_{23} \\ 0 & 0 & w_{33} \end{bmatrix} \begin{bmatrix} w_{11} & 0 & 0 \\ w_{12} & w_{22} & 0 \\ w_{13} & w_{23} & w_{33} \end{bmatrix}$$

$$= \begin{bmatrix} (w_{11}^2 + w_{12}^2 + w_{13}^2) & (w_{12}w_{22} + w_{13}w_{23}) & w_{13}w_{33} \\ (w_{12}w_{22} + w_{13}w_{23}) & (w_{22}^2 + w_{23}^2) & w_{23}w_{33} \\ w_{13}w_{33} & w_{23}w_{33} & w_{33}^2 \end{bmatrix}$$

Esta multiplicación de matrices corresponde a la siguiente sumatoria para el elemento $i\ m$:

$$\sum_{k=m}^{3} w_{ik}w_{mk}$$

En general se tiene:

$$\sum_{k=m}^{j} w_{ik}w_{mk}$$

Por otro lado si definimos el vector $v$ tal que es de tres elementos, tenemos:

$$v = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

Así, se tiene que:

$$vv^T = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} = \begin{bmatrix} v_1^2 & v_1v_2 & v_1v_3 \\ v_1v_2 & v_2^2 & v_2v_3 \\ v_1v_3 & v_2v_3 & v_3^2 \end{bmatrix}$$

Y la norma al cuadrado de $v$ es:

$$|v|^2 = v_1^2 + v_2^2 + v_3^2$$

Para el caso de tres dimensiones se puede desarrollar:

$$I - \frac{vv^T}{r + |v|^2} = \begin{bmatrix} 1 - \frac{v_1^2}{r + v_1^2 + v_2^2 + v_3^2} & -\frac{v_1v_2}{r + v_1^2 + v_2^2 + v_3^2} & -\frac{v_1v_3}{r + v_1^2 + v_2^2 + v_3^2} \\ -\frac{v_1v_2}{r + v_1^2 + v_2^2 + v_3^2} & 1 - \frac{v_2^2}{r + v_1^2 + v_2^2 + v_3^2} & -\frac{v_2v_3}{r + v_1^2 + v_2^2 + v_3^2} \\ -\frac{v_1v_3}{r + v_1^2 + v_2^2 + v_3^2} & -\frac{v_2v_3}{r + v_1^2 + v_2^2 + v_3^2} & 1 - \frac{v_3^2}{r + v_1^2 + v_2^2 + v_3^2} \end{bmatrix}$$

Esta operación entre vectores corresponde a la siguiente sumatoria para el elemento $i\ m$:

$$\Delta_{im} - \frac{v_i v_m}{r + \sum_{k=1}^{3} v_k^2}$$

En general se tiene:

$$\Delta_{im} - \frac{v_i v_m}{r + \sum_{k=1}^{j} v_k^2}$$

Con lo cual se aprecia que:

$$\sum_{k=m}^{j} w_{ik} w_{mk} = \Delta_{im} - \frac{v_i v_m}{r + \sum_{k=1}^{j} v_k^2}$$

**Ecuación 11-3**

Se puede demostrar que la Ecuación 11-3 se cumple para todo $i, m, j$ tal que $1 \leq i \leq m \leq j \leq n$.

Aquí no se elabora una demostración formal pero se ejemplifica el uso de la Ecuación 11-3 mediante un caso particular:

Si $i = m = j = 1$:

$$w_{11}^2 = 1 - \frac{v_1^2}{r + v_1^2}$$

**Ecuación 11-4**

Por otro lado cuando $i = 1$ y $m = j = 2$:

$$w_{12} w_{22} = -\frac{v_1 v_2}{r + v_1^2 + v_2^2}$$

$$(w_{12} w_{22})^2 = \left(\frac{v_1 v_2}{r + v_1^2 + v_2^2}\right)^2$$

**Ecuación 11-5**

Para cuando $i = m = j = 2$

$$w_{22}^2 = 1 - \frac{v_2^2}{r + v_1^2 + v_2^2} = \frac{R + v_1^2}{r + v_1^2 + v_2^2}$$

**Ecuación 11-6**

Cuando $i = m = 1$ y $j = 2$:

$$w_{11}^2 + w_{12}^2 = 1 - \frac{v_1^2}{r + v_1^2 + v_2^2} = \frac{R + v_2^2}{r + v_1^2 + v_2^2}$$

**Ecuación 11-7**

Se puede deducir de la Ecuación 11-5 y la Ecuación 11-6 que:

$$w_{12}^2 = \frac{(w_{12}w_{22})^2}{w_{22}^2} = \left(\frac{v_1 v_2}{r + v_1^2 + v_2^2}\right)^2 \left(\frac{r + v_1^2 + v_2^2}{r + v_1^2}\right) = \frac{(v_1 v_2)^2}{(r + v_1^2 + v_2^2)(r + v_1^2)}$$

**Ecuación 11-8**

Con la Ecuación 11-8 y la Ecuación 11-4 se llega a:

$$\begin{aligned} w_{11}^2 + w_{12}^2 &= 1 - \frac{v_1^2}{r + v_1^2} + \frac{(v_1 v_2)^2}{(r + v_1^2 + v_2^2)(r + v_1^2)} \\ &= \frac{(r + v_1^2 + v_2^2)(r + v_1^2) - (r + v_1^2 + v_2^2)v_1^2 + (v_1 v_2)^2}{(r + v_1^2 + v_2^2)(r + v_1^2)} \\ &= \frac{(r + v_1^2 + v_2^2)(r + v_1^2) - (r + v_1^2)v_1^2}{(r + v_1^2 + v_2^2)(r + v_1^2)} = 1 - \frac{v_1^2}{r + v_1^2 + v_2^2} = \frac{r + v_2^2}{r + v_1^2 + v_2^2} \end{aligned}$$

Resultado que concuerda con la Ecuación 11-7.

En el caso que $m = j$ la sumatoria en el miembro izquierdo tiene sólo un término y:

$$w_{ij}w_{jj} = \Delta_{ij} - \frac{v_i v_j}{r + \sum_{k=1}^{j} v_k^2}$$

Si $i = j$ se tiene:

$$w_{jj}^2 = \Delta_{jj} - \frac{v_j^2}{r + \sum_{k=1}^{j} v_k^2} = 1 - \frac{v_j^2}{r + \sum_{k=1}^{j} v_k^2} = \frac{r + \sum_{k=1}^{j} v_k^2 - v_j^2}{r + \sum_{k=1}^{j} v_k^2} = \frac{r + \sum_{k=1}^{j-1} v_k^2}{r + \sum_{k=1}^{j} v_k^2}$$

$$w_{jj} = \sqrt{\frac{r + \sum_{k=1}^{j-1} v_k^2}{r + \sum_{k=1}^{j} v_k^2}}$$

**Ecuación 11-9**

Si $i < j$ se tiene:

$$w_{ij}w_{jj} = -\frac{v_i v_j}{r + \sum_{k=1}^{j} v_k^2}$$

$$w_{ij} = -\frac{v_i v_j}{r + \sum_{k=1}^{j} v_k^2}\frac{1}{w_{jj}} = -\frac{v_i v_j}{r + \sum_{k=1}^{j} v_k^2}\sqrt{\frac{r + \sum_{k=1}^{j} v_k^2}{r + \sum_{k=1}^{j-1} v_k^2}} = -\frac{v_i v_j}{\sqrt{r + \sum_{k=1}^{j} v_k^2}\sqrt{r + \sum_{k=1}^{j-1} v_k^2}}$$

**Ecuación 11-10**

Y si $i > j$ entonces:

$$w_{ij} = 0$$

Se puede demostrar que si $C^-$ es una matriz triangular superior que constituye un factor de Cholesky de la matriz de covarianza del error a priori $P^-$, entonces:

$$C = C^{-}W$$

**Ecuación 11-11**

Donde, en este caso $C$ es un factor de Cholesky de la matriz de covarianza del error a posteriori $P$.

Si se toma:

$$v = C^{-T}H^{T}$$

Entonces sabiendo que:

$$WW^{T} = I - \frac{vv^{T}}{r + |v|^{2}}$$

Y teniendo en cuenta que la norma al cuadrado:

$$|v|^{2} = v^{T}v$$

$$WW^{T} = I - \frac{vv^{T}}{r + v^{T}v} = I - \frac{C^{-T}H^{T}HC^{-}}{r + HC^{-}C^{-T}H^{T}} = I - \frac{C^{-T}H^{T}HC^{-}}{r + HP^{-}H^{T}}$$

Tomando:

$$C^{-}WW^{T}C^{-T} = C^{-}IC^{-T} - \frac{C^{-}C^{-T}H^{T}HC^{-}C^{-T}}{r + HP^{-}H^{T}} = P^{-} - \frac{P^{-}H^{T}HP^{-}}{r + HP^{-}H^{T}}$$
$$= P^{-} - \frac{P^{-}H^{T}}{r + HP^{-}H^{T}}HP^{-} = P^{-} - KHP^{-} = (I - KH)P^{-} = P = CC^{T}$$

Con esto queda demostrada la Ecuación 11-11:

Con la Ecuación 11-9, y la Ecuación 11-10 es posible derivar una expresión para los elementos de:

$$C = C^{-}W$$

Debido a que $C$ y $W$ son matrices triangulares superiores los elementos $C_{ik} = 0$ para $k < i$ y los elementos $W_{kj} = 0$ para $k > j$.

Con el objeto de obtener una mejor visualización de las conclusiones a sacar se toma como ejemplo un caso en el que $n = 3$ así:

$$C^{-} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ 0 & c_{22} & c_{23} \\ 0 & 0 & c_{33} \end{bmatrix}$$

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ 0 & w_{22} & w_{23} \\ 0 & 0 & w_{33} \end{bmatrix}$$

La matriz producto será:

$$C = C^{-}W = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ 0 & c_{22} & c_{23} \\ 0 & 0 & c_{33} \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ 0 & w_{22} & w_{23} \\ 0 & 0 & w_{33} \end{bmatrix}$$
$$= \begin{bmatrix} c_{11}w_{11} & c_{11}w_{12} + c_{12}w_{22} & c_{11}w_{13} + c_{12}w_{23} + c_{13}w_{33} \\ 0 & c_{22}w_{22} & c_{22}w_{23} + c_{23}w_{33} \\ 0 & 0 & c_{33}w_{33} \end{bmatrix}$$

Consecuentemente, para $1 \leq i \leq m \leq j \leq n$, significa que $c_{ij}$ son los elementos de la parte triangular superior así como de la diagonal de la matriz $C$. Por lo tanto en la $i$-ésima fila y la $j$-ésima columna de la matriz que resulta del producto $C = C^{-}W$ se encuentran los elementos dados por:

$$c_{ij} = \sum_{k=i}^{j} \bar{c_{ik}} w_{kj} + t\acute{e}rminos\ nulos$$

Mediante la Ecuación 11-9 y la Ecuación 11-10 se puede obtener una fórmula para los elementos de $C = C^{-}W$ de la siguiente manera:

$$c_{ij} = \sum_{k=i}^{j} \bar{c_{ik}} w_{kj} = \bar{c_{ij}} w_{jj} + \sum_{k=i}^{j-1} \bar{c_{ik}} w_{kj}$$
$$= \bar{c_{ij}} \sqrt{\frac{r + \sum_{k=1}^{j-1} v_k^2}{r + \sum_{k=1}^{j} v_k^2}} - \sum_{k=i}^{j-1} \frac{\bar{c_{ik}} v_k v_j}{\sqrt{r + \sum_{k=1}^{j} v_k^2} \sqrt{r + \sum_{k=1}^{j-1} v_k^2}}$$
$$= \frac{1}{\sqrt{r + \sum_{k=1}^{j} v_k^2}} \left[ \bar{c_{ij}} \sqrt{r + \sum_{k=1}^{j-1} v_k^2} - v_j \sum_{k=i}^{j-1} \frac{\bar{c_{ik}} v_k}{\sqrt{r + \sum_{k=1}^{j-1} v_k^2}} \right]$$

Esta es la fórmula general para obtener los elementos del factor de Cholesky triangular superior de la matriz de covarianza de la incertidumbre en la estimación a posteriori, utilizando los elementos del factor de Cholesky triangular superior de la matriz de covarianza de la incertidumbre en la estimación a priori junto con el vector:

$$v = C^{-T} H^T$$

Donde $H$ es la matriz de medición (un vector columa) y $r$ es un elemento de la diagonal principal de la matriz $R$.

Actualización temporal de Schmidt

Si $C_p$ es un factor de Cholesky de la matriz de covarianza de la incertidumbre en la estimación a posteriori $P_k$ y $C_q$ es un factor de Cholesky de la matriz de covarianza del ruido de entrada $Q_k$, entonces la matriz particionada:

$$A = \left[ G_k C_q | \Phi_k C_p \right]_{nx(n+q)}$$

Es un factor de Cholesky no cuadrado y no triangular de la matriz de covarianza de la incertidumbre en la estimación a priori $P_{k+1}^{-}$ que tiene un producto cuadrado de $nxn$ como el siguiente:

$$AA^T = \left[G_k C_q | \Phi_k C_p\right]_{nx(n+q)} \left[G_k C_q | \Phi_k C_p\right]_{nx(n+q)}^T = \left[G_k C_q | \Phi_k C_p\right]_{nx(n+q)} \left[\frac{C_q^T G_k^T}{C_p^T \Phi_k^T}\right]_{(n+q)xn}$$

$$= G_k C_q C_q^T G_k^T + \Phi_k C_p C_p^T \Phi_k^T = \Phi_k P_k \Phi_k^T + G_k Q_k G_k^T = P_k^-$$

$$AA^T = P_k^-$$

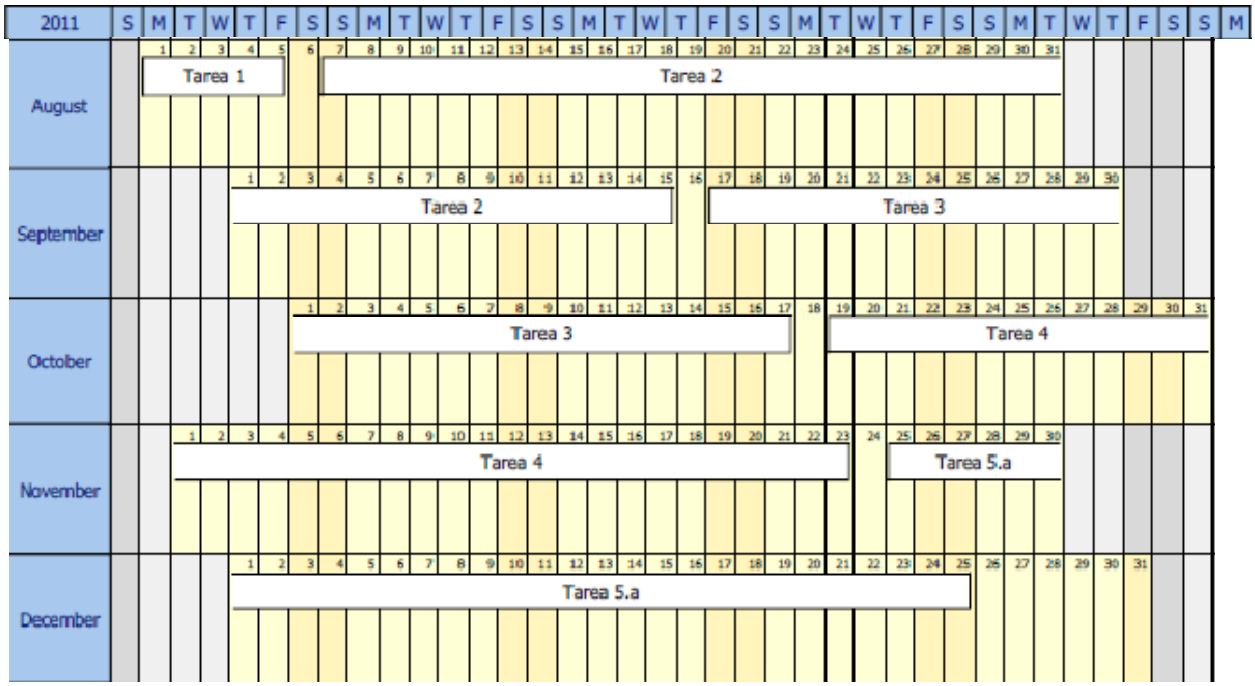
Así es como queda lo que se quería constatar; $A$ es un factor de Cholesky no cuadrado y no triangular de $P_k^-$. Pero en un algoritmo iterativo en el que intervengan ambas actualizaciones de la matriz $P$ de covarianza de error se busca que $A$ sea cuadrado y triangular. Sin embargo existen procedimientos algorítmicos que modifican $A$ para que cumpla con las expectativas necesarias de cuadratura y triangularidad. Las rotaciones de Givens como las reflexiones de Householder constituyen dos de los métodos con los cuales solucionar las disposiciones inadecuadas en la matriz $A$. Dichos métodos pueden ser encontrados en [**2**].

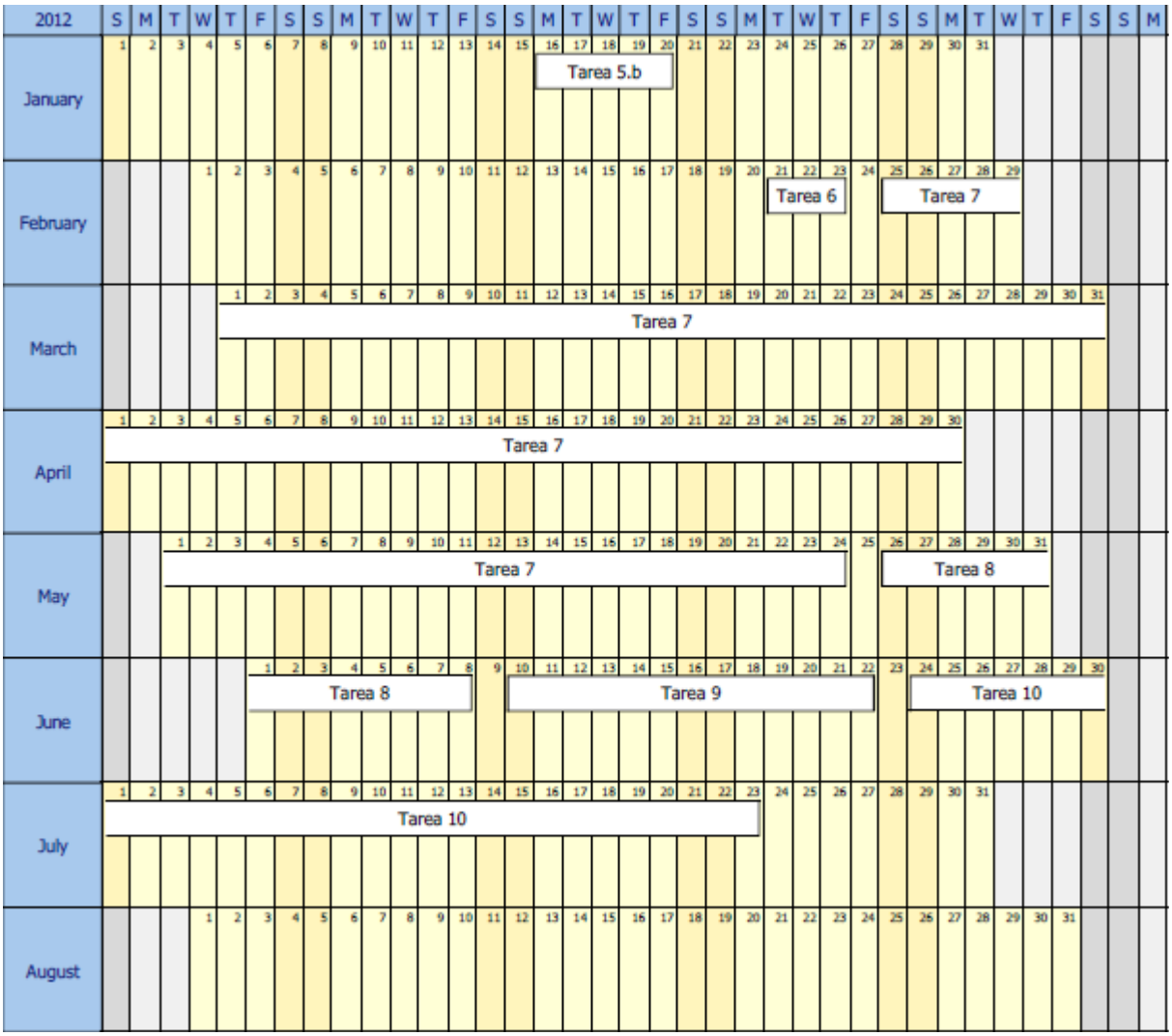
## 11.5 Plan de Trabajo

Tareas propuestas en el plan de trabajo del anteproyecto:

1. Recolección de fuentes de información específicas.
2. Estudio y practica del lenguaje VHDL
3. Estudio y análisis del soporte teórico del FK y sus problemas de computo
4. Planteamiento detallado de los requerimientos técnicos del problema
5. Estudio de la resolución del algoritmo del FK mediante FPGA y generación del sistema lógico implementable. Esto implica:
    a. Elaboración de la arquitectura del sistema lógico.
    b. Determinación de las especificaciones de los bloques lógicos (Sumadores, Multiplicadores, Memorias, Unidades de control) necesarios.
6. Elaboración de un plan de trabajo en base a los bloques lógicos que surgen del análisis realizado en 5.
7. Resolución progresiva de todas las tareas asignadas en 6 (bloques lógicos), controlando que se satisfagan los requerimientos especificados en 5.b
8. Integración de las partes funcionales en una unidad operativa final
9. Implementación de un control de calidad exhaustivo de la unidad final
10. Elaboración del informes técnico y toda la documentación correspondiente

Diagrama de GANT correspondiente:


2011
August
Tarea 1
Tarea 2
September
Tarea 2
Tarea 3
October
Tarea 3
Tarea 4
November
Tarea 4
Tarea 5.a
December
Tarea 5.a


2012
January
Tarea 5.b
February
Tarea 6
Tarea 7
March
Tarea 7
April
Tarea 7
May
Tarea 7
Tarea 8
June
Tarea 8
Tarea 9
Tarea 10
July
Tarea 10
August

Planilla de actividades realizadas:

| Fecha | Actividad | Duración [h] | Firma Tutor |
|---|---|---|---|
| | | | |
| 30/06/2011 | Bibliografía y orientación | 2 | FRAN - DARÍO |
| 7/07/2011 | Intro VHDL | 2 | FRAN - DARÍO |
| 9/07/2011 | Teoría KF | 5 | FRAN - DARÍO |
| 14/07/2011 | Intro VHDL | 2 | FRAN - DARÍO |
| 16/07/2011 | Teoría KF | 5 | FRAN - DARÍO |
| 21/07/2011 | Mult Matrices | 2 | FRAN - DARÍO |
| 23/07/2011 | Teoría KF | 5 | FRAN - DARÍO |
| 28/07/2011 | Mult Matrices | 2 | FRAN - DARÍO |
| 29/07/2011 | Teoría KF | 5 | FRAN - DARÍO |
| 4/8/2011 | Mult Matrices | 2 | FRAN - DARÍO |
| 5/8/2011 | Teoría KF | 5 | FRAN - DARÍO |
| 10/8/2011 | Teoría KF | 2 | FRAN - DARÍO |
| 11/8/2011 | Teoría KF | 5 | FRAN - DARÍO |
| 17/08/2011 | Programación VHDL | 2 | FRAN - DARÍO |
| 24/08/2011 | Programación VHDL | 2 | FRAN - DARÍO |
| 26/08/2011 | Programación VHDL | 3 | FRAN - DARÍO |
| 31/08/2011 | Programación VHDL | 3 | FRAN - DARÍO |
| 5/9/2011 | Teoría KF | 3 | FRAN |
| 15/09/2011 | Teoría KF | 2 | FRAN |
| 19/09/2011 | Documentación | 1 | FRAN |
| 6/10/2011 | Teoría KF | 2 | FRAN |
| 13/10/2011 | Teoría KF | 4 | FRAN |
| 16/10/2011 | Teoría KF | 6 | FRAN |
| 21/10/2011 | Teoría KF | 6 | FRAN |
| 22/10/2011 | Teoría KF | 7 | FRAN |
| 25/10/2011 | Teoría KF | 5 | FRAN |
| 7/11/2011 | Teoría KF | 4 | FRAN |
| 8/11/2011 | Practica KF en Mathematica | 6 | FRAN |
| 9/11/2011 | Practica KF en Mathematica | 6 | FRAN |
| 10/11/2011 | Practica KF en Mathematica | 7 | FRAN |
| 11/11/2011 | Practica KF en Mathematica | 3 | FRAN |
| 12/11/2011 | Practica KF en Mathematica | 2 | FRAN |
| 13/11/2011 | Practica KF en Mathematica | 5 | FRAN |
| 14/11/2011 | Practica KF en Mathematica | 4 | FRAN |
| 15/11/2011 | Practica KF en Mathematica | 6 | FRAN |
| 16/11/2011 | Practica KF en Mathematica | 5 | FRAN - DARÍO |
| 8/29/2011 | Teoría KF | 3 | DARIO |
| 8/29/2011 | Programación | 2 | DARIO |
| 8/30/2011 | Programación | 3 | DARIO |
| 8/31/2011 | Programación | 3.5 | DARIO |
| 9/1/2001 | Programación | 3 | DARIO |
| 9/4/2011 | Teoría KF | 2.5 | DARIO |

| | | | |
|---|---|---|---|
| 9/5/2011 | Teoría KF | 3 | DARIO |
| 9/7/2011 | Teoría KF | 4 | DARIO |
| 9/12/2011 | Teoría KF | 2 | DARIO |
| 9/13/2011 | Teoría KF | 2 | DARIO |
| 9/15/2011 | Teoría KF | 5 | DARIO |
| 9/27/2011 | Teoría KF | 6 | DARIO |
| 10/24/2011 | Teoría KF | 4 | DARIO |
| 10/26/2011 | Teoría KF | 6 | DARIO |
| Del 1/12/2011 al 20/12/2011 | Documentación y Teoría KF | 5 promedio p/día | FRAN - DARÍO |
| Del 1/1/2012 al 23/01/2012 | Teoría KF, Documentación y Programación MATLAB | 6 promedio p/día | FRAN - DARÍO |
| Mes 2 2012 | Documentación y Programación MATLAB | 6 promedio p/día | FRAN - DARÍO |
| Mes 3 2012 | Programación VHDL | 7 promedio p/día | FRAN - DARÍO |
| Mes 4 2012 | Programación VHDL y MATLAB | 8 promedio p/día | FRAN - DARÍO |
| Mes 5 2012 | Programación VHDL, MATLAB y elaboración del Informe final | 8 promedio p/día | FRAN - DARÍO |

# Agradecimientos

Los autores quieren agradecer la colaboración en el desarrollo de este trabajo a las siguientes personas o instituciones: Al departamento de Electrónica de la UTN Facultad Regional Mendoza; Al Laboratorio de Computación Reconfigurable de la UTN-FRM; A los directores del proyecto Ing. Rodrigo Gonzales e Ing. Eduardo Serdoch; Al Ing. Pablo Salvadeo y al personal de la cátedra de Proyecto Final.

Darío Dematties:

A la conciencia, que da nota de su propia existencia; aquella, aún incomunicable, capacidad valorativa de lo bello que arriba a la percepción y de lo sombrío que contrasta realzando aún más su belleza.

A Mabel García Herrero. Que la conciencia se valga.

Iglesias Francisco:

8 años y medio…fiu me cansa solo pensarlo. Dada la naturaleza polifacética del ser humano, difícilmente pueda una persona atribuirse solitariamente el éxito en una empresa de la envergadura que esta carrera ha representado para mí. Por esto quisiera agradecer a todos aquellos que, directa o indirectamente, han contribuido a que pueda alcanzar este objetivo. Primero mi gran familia, comenzando por mi querido padre Mario, laburador incansable que ha dedicado su vida a sus hijos. Este logro es en gran parte suyo también. A mis hermanos Pablo, Pedro, Mercedes, Angeles, Rosario, Juan y Alfredo, que me han enseñado, acompañado y soportado siempre. A Belén por acompañarme y enseñarme a ser una mejor y más completa persona. A mis amigos Fernando, Pablo, Francisco, Abel, Sebastián, Nicolas y otros más que hicieron de esta una etapa para recordar. Dentro del ambiente académico, aunque sin duda sus enseñanzas trascendieron esta esfera, quisiera agradecer a Hebe Cremades, Eduardo Serdoch, Pompilio Zigrino y mi compañero de trabajo Darío Dematties. Quisiera también expresar mi gratitud al estado de la Republica Argentina por proveerme de nada menos que 22 años ininterrumpidos de educación pública y accesible en todos sus niveles, esta es una deuda social que me comprometo a saldar. Finalmente dedico por mi parte este trabajo a la memoria de mi querida madre Magdalena. “Que el destino, tirano, no permitiese que disfrutaras todos los resultados de tu esfuerzo, no modifica ni un poco la magnífica labor que realizaste con tus hijos”