Diseño e implementación en FPGA de un filtro Kalman para aplicaciones biomédicas

Trabajo Fin de Grado
Grado en Ingeniería de Computadores

Fernando Capellán Pizarroso
Simona Florina Puica
Daniel Sánchez Huerta

Dirigido por

Oscar Garnica Alcázar
Juan Lanchares Dávila

Departamento de Arquitectura de Computadores y Automática
Facultad de Informática
Universidad Complutense de Madrid

Junio 2015

Documento maquetado con TEXIS v.1.0+.

Diseño e implementación en FPGA de un filtro Kalman para aplicaciones biomédicas

Trabajo fin de grado

Grado en Ingeniería de Computadores

Autores

Fernando Capellán Pizarroso
Simona Florina Puica
Daniel Sánchez Huerta

Directores

Oscar Garnica Alcázar
Juan Lanchares Dávila

Departamento de Arquitectura de Computadores y Automática
Facultad de Informática
Universidad Complutense de Madrid

Junio 2015

Autorización

Los autores de este proyecto autorizan a la Universidad Complutense de Madrid a difundir y utilizar el presente trabajo de investigación, tanto la aplicación como la memoria, únicamente con fines académicos, no comerciales y mencionando expresamente a sus autores. También autorizan a la Biblioteca de la UCM a depositar el trabajo en el Archivo Institucional E-Prints Complutense.

Autores

Fernando Capellán Pizarroso

Simona Florina Puica

Daniel Sánchez Huerta

La mejor forma de predecir el futuro
es implementarlo.
David Heinemeier Hansson

Agradecimientos

No nos habría sido posible realizar este proyecto de no haber contado con todo el apoyo y cariño de todas las personas que nos rodean. En primer lugar queremos agradecer a nuestros directores Juan y Óscar por habernos propuesto este proyecto y por habernos guiado y aconsejado a lo largo de su desarrollo. En segundo lugar dar las gracias a todos los profesores que, a lo largo de tantos años de carrera, nos han transmitido los conocimientos necesarios para poder llevarlo a cabo. Por último y no menos importante, a nuestros familiares, parejas, amigos y compañeros, los cuales nos han acompañado a lo largo de esta odisea.

A todos vosotros. Gracias.

Resumen

La Diabetes Mellitus tipo 1 es una enfermedad crónica caracterizada por la incapacidad del páncreas de producir insulina. Esta hormona regula la absorción de la glucosa del torrente sanguíneo por parte de las células. Debido a la ausencia de insulina en el cuerpo, la glucosa se acumula en el torrente sanguíneo provocando problemas a corto y largo plazo, como por ejemplo deterioro celular.

Los pacientes con esta enfermedad necesitan controlar su glucemia (concentración de glucosa en sangre) midiendo la misma de forma regular e inyectándose insulina subcutánea de por vida. Para conocer la glucemia se pueden utilizar Monitores Continuos de Glucosa (MCG), que proporcionan el valor de la glucosa intersticial cada 1-5 minutos. Los MCG actuales presentan los siguientes problemas:

- El sensor que llevan incorporado introduce ruidos asociados a la medición y se degrada a lo largo de su vida útil, lo que dificulta la interpretación de los datos obtenidos.
- Debido al transporte de la glucosa desde el torrente sanguíneo hacia el fluido intersticial, el valor de la glucosa en este último presenta cierto retraso frente al valor de la glucosa en el primero.
- Es necesaria una calibración frecuente, usando como referencia la glucosa en sangre muestreada del propio paciente.
- La variabilidad de la glucosa entre distintos pacientes y a lo largo de la vida de un mismo paciente dificulta el ajuste de los parámetros de los filtros.

Una solución propuesta actualmente es la utilización de filtros Kalman. Éstos se describen mediante algoritmos recursivos capaces de identificar el estado oculto (glucosa en sangre) a partir de medidas indirectas del mismo (glucosa intersticial), y al mismo tiempo predicen su comportamiento futuro y eliminan el ruido de la señal.

El principal problema de esta propuesta es que su implementación se realiza en software, y por tanto, trabajan con datos ya procesados por el

sensor. Esto puede agravar alguno de los problemas listados con anterioridad e impide sacar el máximo provecho a la gran capacidad de estos filtros.

Este proyecto surgió con la idea de dejar atrás los filtros Kalman software y sustituir los filtros digitales hardware con los que actualmente cuentan los sensores por un filtro de Kalman. En este proyecto se implementa sobre una FPGA un filtro Kalman de tres estados (glucosa, velocidad de variación de la glucosa y aceleración de la velocidad de dicha variación), teniendo como entrada medidas reales proporcionadas por un MCG.

En los resultados se puede comprobar que las predicciones realizadas por el filtro Kalman implementado en este proyecto se ajustan perfectamente al comportamiento de la glucosa en sangre. Se han realizado simulaciones del filtro para observar su respuesta ante variaciones bruscas de la glucosa y la fiabilidad de la misma durante un largo periodo de tiempo. Para ello se han utilizado medidas de cuatro pacientes reales. En aquellos cuya glucosa cambia de forma brusca se puede observar que el filtro estima el valor de ésta erróneamente. A pesar de ello, utiliza este resultado incorrecto para corregir la siguiente predicción y ofrecer resultados correctos en el futuro.

Palabras clave: FPGA, Filtro de Kalman, Páncreas Artificial, Diabetes Mellitus, Diseño Hardware, Algoritmo recursivo, Procesado de señal.

Abstract

Diabetes Mellitus type 1 is a chronic disease characterised by the inability of the pancreas to produce insulin. This hormone controls the absorption of glucose from the bloodstream by the cells. As a result of the absence of insulin in the body, glucose builds up in the bloodstream causing short and long term problems. Cellular degeneration serves as an example.

Patients with this disease need to keep their glycemia under control by measuring it regularly and injecting subcutaneous insulin for the rest of their lives. An option to know the blood glucose concentration is the use of a continuous glucose monitoring, as the provide the interstitial glucose value every 1 to 5 minutes. Current CGMs present the following problems:

- The sensor they have embedded adds measurement-related noises and deteriorates during its lifespan, rendering the interpretation of the gathered data more difficult.
- There is a delay between the concentration of blood glucose and interstitial glucose due to the fact that it has to be transported from one environment to another.
- Frequent calibration is required, using the blood glucose measured from the patient as reference.
- Glucose variability between patients and in a sole patient's lifetime makes difficult to adjust the filter parameters.

A proposed solution is the use of Kalman filters. They are described by recursive algorithms capable of identifying the hidden state (blood glucose) from indirect measurements of itself (interstitial glucose), predicting, at the same time, its future behavior and removing the noise from the signal.

The main problem these proposals share is that the implementation is done in software, and thus, they work with data that has already been processed by the sensor. This can worsen some of the problems listed above, and prevents the filters from delivering their full potential.

This project's goal is to leave behind Kalman filters implemented in software and to replace the digital hardware filters that sensors currently equip

with a Kalman filter. In this project, a Kalman filter with three states (glucose, glucose variation speed and acceleration of the speed of said variation) is implemented on an FPGA, with real measurements provided by a CGM.

Experimental results show that the estimations made by the Kalman filter implemented throughout this project match precisely the behavior of the blood glucose. Simulations of the filter have been done to observe its response in the presence of sharp variations of glucose and its reliability during long periods of time. For this, measurements from four real patients have been used. When glucose experienced a sudden change, the filter can be seen mispredicting the next value. Despite this, it uses this incorrect result to correct the next estimation and deliver precise results in the future.

Keywords: FPGA, Kalman filter, Artificial Pancreas, Diabetes Mellitus, Hardware design, Recursive algorithm, Signal processing.

Índice

Índice de figuras

Índice de Tablas

Parte I

Estado del arte

Capítulo 1

Motivaciones del trabajo

La Diabetes Mellitus tipo 1 (DM1) es una enfermedad crónica autoinmune caracterizada por la destrucción de las células beta del páncreas, lo que le imposibilita producir insulina. Los pacientes con esta enfermedad no tienen la capacidad de generar suficiente insulina para llevar la glucosa del torrente sanguíneo a las células. Esto da lugar a un aumento de la glucosa en sangre (GS). Esta situación se conoce como hiperglucemia, y puede provocar problemas sanguíneos deteriorando la hemoglobina o los vasos sanguíneos. Si no se trata a tiempo, los efectos a largo plazo sobre el organismo pueden llevar al paciente a un estado de coma o incluso provocar su muerte.

Para poder llevar una vida normal, los pacientes de DM1 necesitan administrarse insulina subcutánea de por vida para suplir su déficit y evitar la hiperglucemia. Por otro lado, un exceso de insulina puede dar lugar a una situación de hipoglucemia, donde los niveles de GS son muy bajos, que es asimismo peligroso. El control glucémico es un punto crítico en este tipo de pacientes, que deben encontrar un punto de balance entre ambos extremos.

Para obtener el control glucémico autónomo es necesario que el paciente mida de forma regular su nivel de GS. Esto se puede hacer utilizando Monitores Continuos de Glucosa (MCG). Estos dispositivos no miden la GS directamente, sino que trabajan con la glucosa intersticial (GI), es decir, la que se encuentra en el espacio intercelular, siendo menos intrusivos para el paciente. Los MCG permiten medir la GI del paciente a intervalos de tiempo regulares, del orden de minutos.

El MCG por sí solo no basta para llevar a cabo un buen control glucémico. Necesita trabajar conjuntamente con un algoritmo que sea capaz de anticiparse al desarrollo del estado del paciente. Debe ser capaz de realizar predicciones con los datos proporcionados por dicho monitor. Una bomba de insulina proporciona esta sustancia al paciente según lo considere necesario el MCG. Juntos, estos elementos formarían un sistema conocido como Páncreas Artificial (PA).

Sin embargo el PA aún no se ha desarrollado plenamente, presentando

todavía ciertos problemas y dificultades. A continuación se listan una serie de problemas relacionados con los MCG:

- El sensor de glucosa que llevan incorporado introduce ruidos asociados a la medición. Además, se degradan progresivamente a lo largo de su vida útil. Esto dificulta la interpretación de los datos obtenidos y genera la necesidad de procesarlos primero.

- La glucosa se distribuye por el organismo a través del torrente sanguíneo y, antes de llegar a las células pasa por el fluido intersticial, que es donde el MCG efectúa las medidas para ser lo menos invasivo posible para el paciente. Debido a que el transporte de la glucosa de un medio a otro no es instantáneo el valor de la GI presenta cierto retraso frente al valor de la GS, por lo que los valores recogidos por el sensor no son los correspondientes a la GS del paciente en ese momento.

- El MCG necesita una calibración frecuente. Para ello se usa como referencia un muestreo de la GC del propio paciente.

- Es necesario ajustar de forma personalizada los parámetros de los filtros incluidos en el MCG para adaptarlos a las características individuales de cada paciente. Además, para un mismo paciente, factores como la edad, la alimentación, los hábitos de salud y la actividad diaria hacen que varíe el comportamiento de la glucosa en su organismo.

Una solución propuesta actualmente para estos problemas es la utilización de filtros Kalman. Éstos hacen uso de algoritmos recursivos que permiten identificar el estado oculto de un sistema utilizando medidas indirectas del mismo, al mismo tiempo que predicen su comportamiento futuro y eliminan el ruido de la señal. Las equivalencias con el monitor de glucosa son:

- El estado oculto del sistema es la glucosa en sangre, ya que se quiere medir sin actuar directamente sobre ella.

- La glucosa intersticial será la medida indirecta para identificar la GS debido a la relación que tiene con la misma.

El principal problema de esta propuesta es que la implementación de este tipo de filtros se hace mediante software. De esta manera, no pueden trabajar con otros datos que no estén procesados y tratados en primer lugar por el sensor. A causa de esto, los problemas listados anteriormente pueden verse agravados y el filtro no puede mostrar todo su potencial.

Para ofrecer una solución viable y factible a los problemas de los MCG existentes y de las implementaciones software del filtro Kalman, en este proyecto se implementa sobre una FPGA un filtro Kalman de tres estados (glucosa, velocidad de variación de la glucosa y aceleración de la velocidad de

dicha variación). Éste podrá utilizar como entrada las medidas reales de GI proporcionadas por un MCG sin que estén adulteradas por el tratamiento previo de los filtros digitales hardware de los sensores.

Capítulo 2

Teoría del filtro Kalman

El filtro de Kalman es un algoritmo propuesto por Rudolf E. Kalman en 1960 como solución recursiva al problema del filtrado de datos lineales discretos. Desde su introducción, debido a su robustez y simplicidad, ha sido usado en muchos ámbitos como en predicciones financieras, rastreo de misiles o navegación autónoma y asistida.

Está formado por un conjunto de ecuaciones matemáticas. Su objetivo es estimar de manera óptima el estado oculto de un sistema a partir de medidas indirectas del mismo.

El filtro de Kalman es útil para el filtrado de ruidos blancos y gausianos. Los ruidos blancos son aquellos cuyo valor no está correlacionado en el tiempo. Los ruidos gausianos son aquellos cuya amplitud en un instante de tiempo sigue una distribución de Gauss.

Existen dos tipos de FK:

- Discreto: Las ecuaciones de los modelos de proceso y de medición son lineales.

- Extendido: Las ecuaciones de los modelos de proceso y/o medición no son lineales.

En este trabajo se implementa el FK discreto, bajo la suposición de que las ecuaciones del modelo de la glucosa son lineales. Como modelo general se han utilizado las ecuaciones del Filtro de Kalman discreto propuestas por Welch y Bishop (2001). Toda la terminología del trabajo se ha adaptado a la nomenclatura de dicho artículo.

2.1. Ecuaciones generales para el filtro de Kalman discreto

2.1.1. Ecuaciones de los modelos de proceso y medición

Es importante fijar estos modelos para la extracción posterior de las ecuaciones del filtro.

- La ecuación del modelo del *proceso* es

$$x_k = Ax_{k-1} + Bu_k + w_{k-1} \tag{2.1}$$

  donde:

  - x_k es el vector del estado del proceso en el instante actual (k).
  - x_{k-1} es el vector del estado del proceso en el instante anterior $(k-1)$.
  - A es la matriz que relaciona el estado anterior con el estado actual (k).
  - B es la matriz que relaciona el estado actual con la entrada de control.
  - u_k es una entrada de control en el instante actual (k).
  - w_{k-1} es el ruido asociado al proceso en el instante anterior $(k-1)$.

- La ecuación del modelo de *medición* es

$$z_k = Hx_k + v_k \tag{2.2}$$

  donde:

  - z_k es la medición en el instante actual (k).
  - H es la matriz que relaciona la medición con el estado actual.
  - v_k es el ruido asociado a la medición en el instante actual (k).

Las variables v y w representan ruidos del sistema. Estas señales se modelan como ruido blanco gausiano aditivo y siguen distribuciones normales de la siguiente forma:

$$p(w) \sim N(0, Q)$$

$$p(v) \sim N(0, R)$$

Los parámetros Q y R representan las covarianzas de dichos ruidos. Q es la covarianza del ruido del proceso mientras que R es la covarianza del ruido de la medición. En la práctica, estas variables pueden cambiar a lo largo del tiempo pero en este proyecto se asumirá que son constantes.

2.1.2. Ecuaciones del Filtro de Kalman

El algoritmo consta de dos etapas: una de predicción, en la que proporciona una estimación del estado del sistema previa a la llegada de la medida, y otra de corrección, en la que utiliza la medición tomada por el sensor para corregir el estado del sistema y mejorar las predicciones futuras. Esto hace que las ecuaciones del FK se dividan en dos subconjuntos.

- Ecuaciones de la etapa de predicción: estima el valor del estado en un instante futuro basándose en el estado corregido con la ultima medición tomada.

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k \tag{2.3}$$

donde:

  - $\hat{x}_k^-$ representa el estado *a priori*, estimado antes de la medición.
  - $\hat{x}_{k-1}$ representa el estado corregido *a posteriori*, calculado después de la medición.
  - A, B y u representan los mismos parámetros que en el modelo del proceso.

y

$$P_k^- = AP_{k-1}A^T + Q \tag{2.4}$$

donde

  - P_k^- es una medida de la precisión de la estimación del estado, también llamada covarianza del error de estimación. En este caso, la covarianza del error de estimación *a priori*, calculada en el instante actual (k).
  - P_{k-1} representa la covarianza del error *a posteriori*, calculada en el instante anterior ($k-1$).
  - Q representa la matriz de covarianza del ruido del proceso.
  - A representa el mismo parámetro que en el modelo del proceso.

- Ecuaciones de la etapa de corrección: utilizan la medición y los datos *a priori* para calcular los valores *a posteriori*.

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{2.5}$$

donde:

  - K_k representa la *ganancia de Kalman*.
  - P_k^- representa la covarianza del error de estimación *a priori*.
  - H es la misma matriz descrita en el modelo de medición.
  - R es la covarianza del ruido de la medición.

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{2.6}$$

donde:

- $\hat{x}_k$ representa el estado *a posteriori*.
- $\hat{x}_k^-$ representa el estado *a priori*.
- K_k representa la ganancia de Kalman.
- z_k representa la medida real tomada.

y

$$P_k = (I - K_k H)P_k^- \tag{2.7}$$

donde:

- P_k representa la covarianza del error de estimación *a posteriori*.
- I es la matriz *identidad*.
- K_k representa la ganancia de Kalman.
- H es la misma matriz descrita en el modelo de medición.
- P_k^- representa la covarianza del error de estimación *a priori*.

2.1.3. Algoritmo

El filtro de Kalman sigue una dinámica del tipo predicción-corrección, en la que se proyecta el estado estimado para posteriormente corregirse con la medida real. El filtro de Kalman se describe por las ecuaciones mostradas en la figura 2.1, extraída de Welch y Bishop (2001).

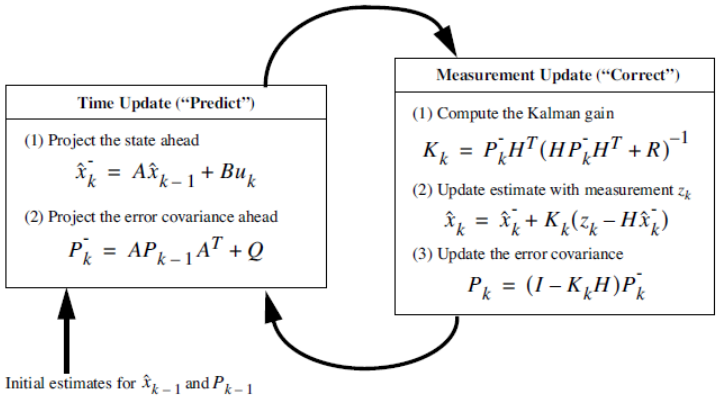


Figura 2.1: Sistema de predicción-corrección con las ecuaciones de cada etapa

El algoritmo comienza con unas estimaciones iniciales que sirven para que el sistema tenga unos datos con los que poder empezar a funcionar. A partir de ahí:

- Predicción:
  - Cálculo del estado del sistema x *a priori* a partir del estado *a posteriori* y la matriz constante A.
  - Cálculo de la covarianza del error de estimación P *a priori*, a partir de la misma *a posteriori*, la matriz constante A y la matriz de covarianza del ruido del proceso Q.
- Corrección:
  - Comienza con el cálculo de la ganancia una vez se tenga calculada la covarianza del error *a priori*, necesaria para calcular tanto el estado como la covarianza del error *a posteriori*.
  - Se incorpora la medición y, junto con la ganancia, se calcula el estado *a posteriori*.
  - Cálculo de la covarianza del error *a posteriori* a partir de la predicción y la ganancia.

Capítulo 3

Modelo del Filtro de Kalman propuesto por Palerm

Como se ha indicado en la sección 2.1.1, es importante fijar los modelos de proceso y de medición. Estos modelos son los que determinan si el filtro será discreto o extendido, las dimensiones de todas las matrices, así como los valores de las matrices constantes A y H. En este proyecto se han usado como referencia los artículos Palerm et al. (2005) y Palerm y Bequette (2007).

Según estos artículos, el *modelo de proceso* queda definido por:

$$g_{k+1} = g_k + d_k$$

$$d_{k+1} = d_k + f_k$$

$$f_{k+1} = f_k + w_k$$

donde

- g es la glucosa intersticial
- d es la velocidad de variación de la glucosa intersticial, es decir, la derivada primera de g
- f es la velocidad de variación de d, por tanto la derivada primera de d o, lo que es lo mismo, la derivada segunda de g
- w es el ruido blanco gaussiano aditivo del proceso, con covarianza Q

El *modelo de medición* descrito en los mismos artículos es:

$$y_k = g_k + v_k$$

donde

- y es el valor medido

- g es la glucosa intersticial
- v es el ruido blanco gaussiano aditivo de la medición, con covarianza R

Los subíndices k y $k+1$ indican el instante de tiempo actual y el instante de tiempo siguiente, respectivamente.

La primera información que extraemos de estos modelos es que ambos son lineales, por lo tanto el filtro que se tiene que implementar es discreto y no extendido. A continuación representamos estos modelos en forma matricial:

$$\begin{bmatrix} g_{k+1} \\ d_{k+1} \\ f_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} g_k \\ d_k \\ f_k \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} w_k$$

$$y_k = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} g_k \\ d_k \\ f_k \end{bmatrix} + v_k$$

Se puede observar que en estas expresiones no aparece el término de Bishop Bu_k. Esto se debe a que el modelo de Bishop es un modelo general y supone que existe una entrada de control (u_k). Sin embargo, en los filtros aplicados al control de glucosa no se utiliza dicha entrada, por lo tanto el término Bu_k desaparece de la expresión. Si se ponen estas ecuaciones en forma compacta, adquieren la siguiente forma

$$x_k = Ax_{k-1} + w_k$$

$$z_k = Hx_k + v_k$$

, que concuerda con las expresiones de Bishop, salvo por el término correspondiente a la entrada de control que se ha eliminado.

Por consiguiente tenemos que x es una matriz de 3x1 que contiene los 3 estados del proceso:

$$x_k = \begin{bmatrix} g_k \\ d_k \\ f_k \end{bmatrix}$$

A es una matriz 3x3:

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

H es una matriz 1x3:

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

Variable	Descripción	Dimensiones	Tamaño
x	Vector de estados	nx1	3x1
A	Matriz que relaciona el estado actual con el anterior	nxn	3x3
B	Matriz que relaciona el control con el estado	-	no se usa
H	Matriz que relaciona la medición con el estado	mxn	1x3
z	Medición	mx1	1x1
K	Ganancia de Kalman	nxm	3x1
Q	Covarianza del ruido del proceso	nxn	3x3
P	Covarianza del error asociada a la estimación	nxn	3x3
R	Covarianza del ruido de la medición	mxm	1x1

Tabla 3.1: Variables del FK particularizado

En la tabla 3.1 se particularizan las dimensiones de los elementos del FK propuesto por Palerm.

Volviendo al modelo genérico de Bishop que se muestra en la figura 2.1 pero aplicando el modelo particularizado de Palerm, las ecuaciones del sistema nos quedarían como en la figura 3.1. Nótese que la única diferencia es la ausencia de los parámetros Bu_k, puesto que en este sistema no se usan las entradas de control.

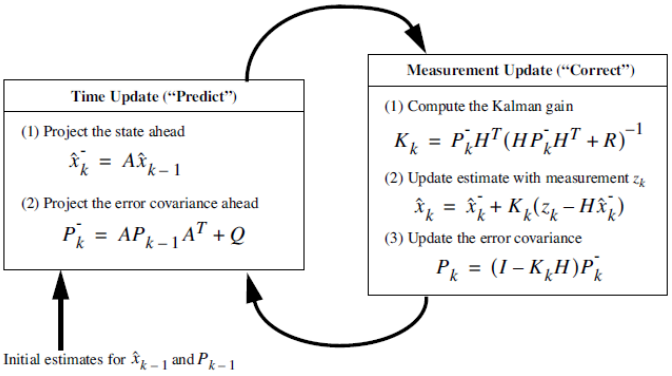


Figura 3.1: Sistema de predicción-corrección con las ecuaciones del modelo de Palerm

En el resto del documento, para hablar de las variables de estas ecuaciones, se va a usar la nomenclatura que se muestra en la tabla 3.2. Además, en la tabla 3.3se definen los convenios de estilo que se han utilizado para representar diferentes elementos en el documento.

Variable	Nombre	Descripción
A	A	Constante A, matriz de dimensiones 3x3.
A^T	A^T	Traspuesta de A, matriz de dimensiones 3x3.
H	H	Constante H, matriz de dimensiones 1x3.
H^T	H^T	Traspuesta de H, matriz de dimensiones 3x1.
I	I	Matriz *identidad*, de dimensiones 3x3.
K_k	K	Ganancia de Kalman, matriz de 3x1.
$\hat{x}_k^-$	XPR	Estado estimado *a priori*, matriz de dimensiones 3x1.
$\hat{x}_k$	XPO	Estado corregido *a posteriori*, matriz de dimensiones 3x1.
P_k^-	PPR	Covarianza del error *a priori*, matriz de dimensiones 3x3.
P_k	PPO	Covarianza del error *a posteriori*, matriz de dimensiones 3x3.
Q	Q	Covarianza del ruido del proceso, matriz de dimensiones 3x3.
R	R	Covarianza del ruido de la medida, escalar.
z	z	Medida que entra al sistema, escalar.

Tabla 3.2: Nomenclatura utilizada para referirse a las variables del filtro

Elemento	Estilo	Ejemplo
módulos principales	Small Caps	Gain
entidades	**negrita**	**fifo_write**
instancias de una entidad	*cursiva*	*i_calc_x*
recepción de señal en una FSM	$\leftarrow$	done_mul $\leftarrow$ 1
asignación de señal en una FSM	$\rightarrow$	start_fifo_read $\rightarrow$ 1

Tabla 3.3: Convenio de representación

Parte II

Implementación

Capítulo 4

Módulo Filtro de Kalman

Como se puede observar en la figura 4.1, el módulo *top-level* del diseño se descompone en tres submódulos principales, y en cada uno de ellos se implementa una parte del sistema de ecuaciones vistas en el capítulo 3. En la tabla 4.1 se puede consultar su interfaz.

En la implementación de estos módulos se han utilizado una serie de elementos comunes a los que se dedica un capítulo, puesto que es necesario explicarlos para entender mejor la implementación de los otros módulos. En los siguientes capítulos se procede a explicar con mayor profundidad cada uno de estos módulos:

- **Módulos Comunes**: Elementos comunes instanciados en los módulos principales. (Capítulo 5).
- **Prediction**: Implementa las ecuaciones 2.3 y 2.4. (Capítulo 6).
- **Gain**: Implementa la ecuación 2.5. (Capítulo 7).
- **Correction**: Implementa las ecuaciones 2.6 y 2.7. (Capítulo 8).

Por otro lado, en las distintas secciones de este capítulo se va a explicar:

- **Representación de los datos**: es importante saber cómo se representan los datos para operar con ellos correctamente. (Sección 4.1)
- **El funcionamiento general del filtro**: antes de entrar a ver el funcionamiento interno de cada submódulo, es conveniente entender cómo interaccionan éstos a alto nivel. (Sección 4.2)
- **Datos intercambiados y constantes**: qué datos se intercambian o usan los módulos. (Sección 4.3)
- **Las FIFOs**: Se han utilizado para desacoplar el intercambio de información entre los módulos principales. (Sección 4.4)

- **El protocolo de comunicación**: Para que los módulos tuvieran un comportamiento homogéneo a la hora de transmitir sus resultados, se diseñó un protocolo para la comunicación entre ellos. (Sección 4.5)

Nombre	Tamaño	Sentido	Comentario
imsr	16 bits	entrada	Medición recibida del monitor continuo de glucosa.
imsr_ready	1 bit	entrada	Indica que imsr está listo para ser utilizado.
r	16 bits	entrada	Covarianza asociada a la medida (matriz 1x1 R).
q_addr	4 bits	entrada	Proporciona la dirección de lectura de Q.
q_data	16 bits	entrada	Entrada de datos de la memoria externa Q.
start	1 bit	entrada	Inicio global del sistema.
xpo_data_out	16 bits	salida	Salida del resultado del filtro.
xpo_write_fifo	1 bit	salida	Notifica la salida del resultado.

Tabla 4.1: Interfaz del FK

4.1. Representación de los datos

Las medidas generadas por el sensor tienen que poder ser interpretadas por el filtro. Para ello se convierten a un formato digital que el sistema pueda manejar. El sistema trabaja con vectores de 16 bits. Para la representación que se la coma fija, ya que es más sencilla de implementar que la coma flotante, y además no es necesaria la potencia y precisión de esta última representación.

Para concretar el formato es necesario tener en cuenta los valores más altos que se necesitan representar. Los valores entre los que suele oscilar la glucemia están comprendidos en el intervalo de 100 a 300 mg/dl. Para representar estos valores es necesaria como mínimo una parte entera de 9 bits, con la que se puedan representar $2^9 = 512$ valores.

Contando con el bit de signo y el probable desbordamiento que se pueda producir en los cálculos, se ha decidido utilizar una parte entera de 12 bits y una parte decimal de 4 (en notación Q corresponde a $Q12{,}4$). Se sacrifica precisión por un mayor rango de valores, pero en este caso 4 bits proporcionan una precisión suficiente.

La representación en coma fija se explica más en detalle en el apéndice A.

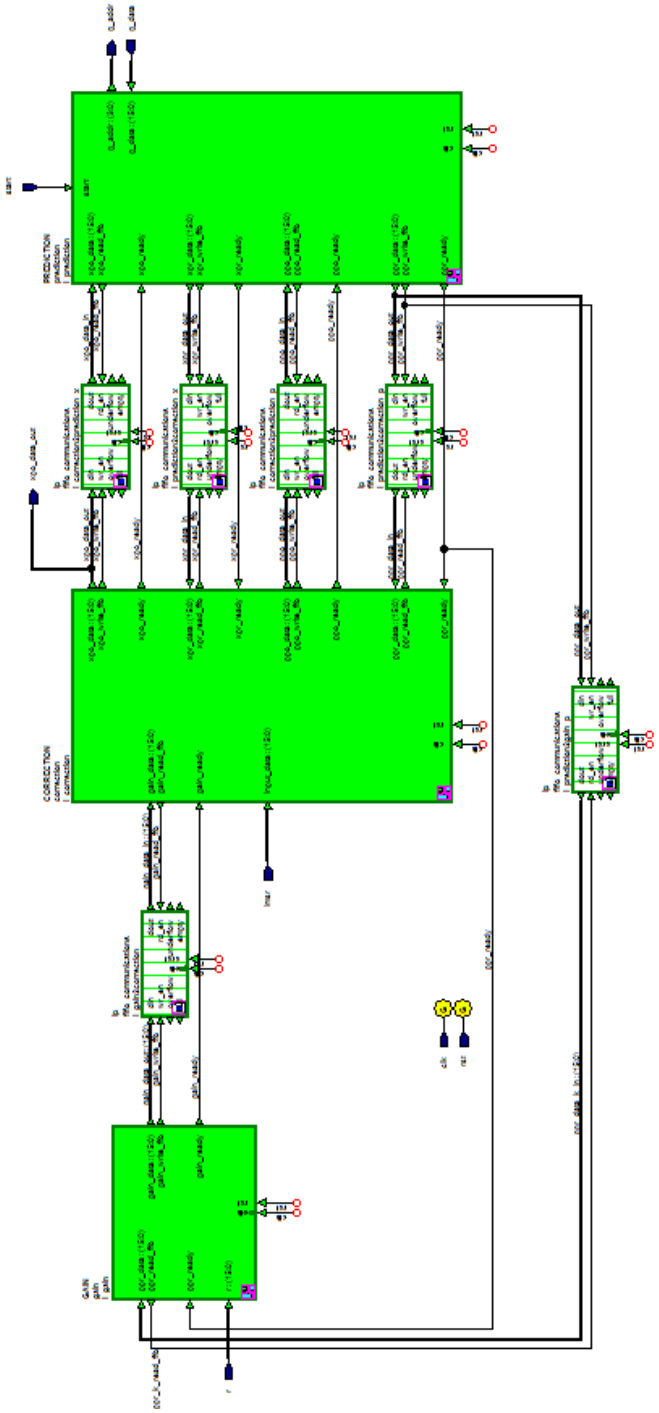


Figura 4.1: Módulo Filtro de Kalman a nivel top

4.2. Funcionamiento general

Sistema cíclico

El Filtro de Kalman es un sistema recursivo, que se retroalimenta para conseguir estimar los estados de forma óptima. Esto implica la existencia de un ciclo en el cual cada módulo, en una iteración dada, realiza las siguientes tareas básicas:

1. Lee el dato proporcionado por el módulo emisor.
2. Procesa el dato.
3. Pasa a ser módulo emisor y proporciona el dato al módulo receptor.

Este ciclo se repite de forma indefinida mientras el sistema no sea *reseteado*.

Inicio del sistema

El hecho de que el sistema sea cíclico genera un problema a la hora de iniciar los cálculos, puesto que no existe ningún módulo emisor que genere datos iniciales del sistema. Por ello, es necesario que en la primera iteración se carguen estos datos iniciales con los que realizar los cálculos. Considerando esto como una fase de inicialización, el sistema entra en el ciclo de filtrado inmediatamente después. De esta forma:

1. El sistema empieza a funcionar cuando se activa la señal start.
2. Esta señal activa el módulo Prediction, que es el encargado de llevar a cabo dicha fase de inicialización del sistema.
3. Una vez que el módulo Prediction termina los cálculos con los valores iniciales, los pasa a los módulos receptores.
4. Las siguientes iteraciones siguen según los puntos explicados en la sección 4.2.

4.3. Datos intercambiados y constantes

Datos salientes y entrantes

En la tabla 4.2 se detalla, para cada módulo, la información de entrada y de salida, sus dimensiones, su origen o destino y el nombre que se le ha dado en la interfaz del módulo.

			PREDICTION		
Entrada		Dimensiones	Origen	Nombre	Observaciones
	XPO	3x1	CORRECTION	xpo_data	Estado *a posteriori*
	PPO	3x3	CORRECTION	ppo_data	Covarianza P *a posteriori*
	Q	3x3	Exterior	q_data	Dato generado por adaptación
Salida		Dimensiones	Destino	Nombre	Observaciones
	XPR	3x1	CORRECTION	xpr_data	Estado *a priori*
	PPR	3x3	CORRECTION y GAIN	ppr_data	Covarianza P *a priori*
			GAIN		
Entrada		Dimensiones	Origen	Nombre	Observaciones
	PPR	3x3	PREDICTION	ppr_data	Covarianza P *a priori*
	R	1x1	Exterior	r	Dato generado por adaptación
Salida		Dimensiones	Destino	Nombre	Observaciones
	K	3x1	CORRECTION	gain_data	Ganancia
			CORRECTION		
Entrada		Dimensiones	Origen	Nombre	Observaciones
	XPR	3x1	PREDICTION	xpr_data	Estado *a priori*
	K	3x1	GAIN	gain_data	Ganancia
	z	escalar	Lectura	imsr	Viene de una medida exterior
	PPR	3x3	PREDICTION	ppr_data	Covarianza P *a priori*
Salida		Dimensiones	Destino	Nombre	Observaciones
	XPO	3x1	PREDICTION	xpo_data	Estado *a posteriori*
	PPO	3x3	PREDICTION	ppo_data	Covarianza P *a posteriori*

Tabla 4.2: Entradas y salidas de los distintos módulos

Datos constantes

Además de los datos que intercambian los módulos entre sí o reciben desde el exterior, existen una serie de datos que son constantes, equivalentes a las matrices constantes de las ecuaciones del filtro. Estas constantes se han implementado mediante memorias ROM que se han instanciado dentro de cada módulo. En la tabla 4.3 se detallan sus tamaños y dónde se usan.

Constante	Dimensiones	Uso	Nombre matriz
A	3x3	Prediction	c_rom_a
H	1x3	Gain y Correction	c_rom_H
I	3x3	Correction	c_rom_I

Tabla 4.3: Datos constantes

4.4. FIFOs

Como se ha visto en el apartado anterior, los módulos principales tienen que intercambiar información entre sí. El mecanismo de intercambio de datos se basa en el uso de memorias FIFO. En la imagen 4.2 se muestra la interfaz de las mismas. Es importante hacer notar que en este diseño no se han utilizado los puertos *overflow*, *full*, *underflow* y *empty*, puesto que el protocolo elaborado en la sección 4.5 se encarga del control del flujo de datos.

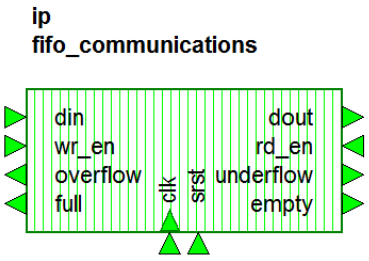


Figura 4.2: Interfaz de las FIFOs utilizadas en el diseño

El tamaño de las FIFOs que se han utilizado es de 16 posiciones. Sin embargo los datos que se intercambian entre los módulos no llegan a tener tantos elementos, por lo que es importante saber cuántos elementos se van a pasar a través de la misma para definir correctamente el protocolo de intercambio de datos.

En el *top-level* del FK se han instanciado 6 FIFOs, cuyos nombres y el número de elementos que se intercambian a través de ellas se detallan en la tabla 4.4.

Nombre	Nº elementos
i_correction2prediction_x	3
i_prediction2correction_x	3
i_correction2prediction_p	9
i_prediction2correction_p	9
i_prediction2gain_p	9
i_gain2correction	3

Tabla 4.4: FIFOs instanciadas en el *top-level*

4.5. Protocolo de comunicación

Una vez resuelto el problema del intercambio de datos mediante la incorporación de FIFOs al diseño, surgió la necesidad de coordinar los tres módulos para las escrituras y lecturas de las mismas. Llamaremos *módulo emisor* al que genera el dato y *módulo receptor* al que lo recibe. En el ejemplo de la figura 4.3 se puede apreciar que el módulo GAIN es el emisor de datos y CORRECTION es el módulo receptor. El protocolo de comunicación es el siguiente:

- Cuando un módulo emisor ha terminado el cálculo de un dato, lo escribe en la FIFO.

- Una vez que ha concluido la escritura, activa una señal (gain_ready en el ejemplo) durante un **único ciclo de reloj** para indicar al módulo receptor que el dato que necesita está disponible.

- Esta señal activa la lectura de la FIFO por el módulo receptor.

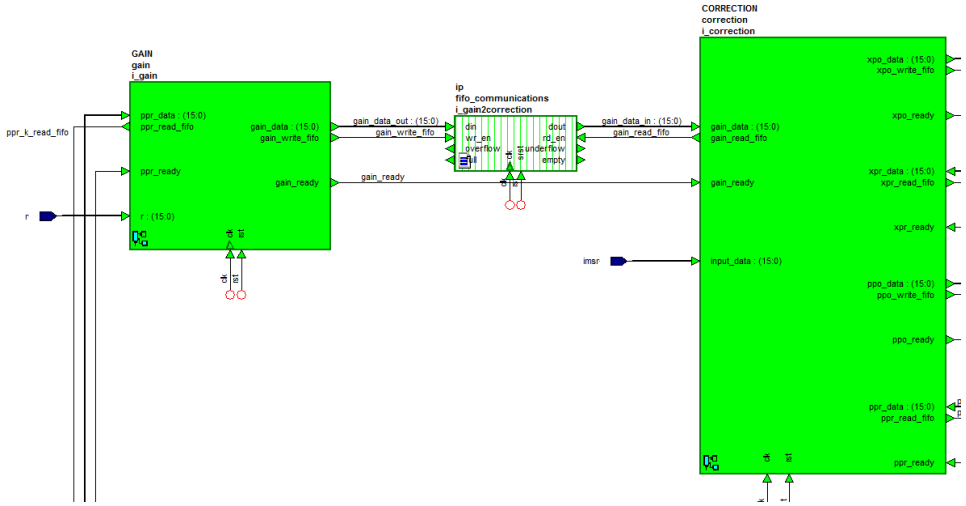


Figura 4.3: Protocolo de comunicación mediante FIFOs

Escritura de la FIFO

El módulo encargado de esta tarea es **fifo_write**, cuyo diagrama de bloques de puede observar en la figura 4.4. El control lo realiza el módulo emisor. Se controla mediante una máquina de estados (*i_fifo_access_wr*) que tiene un estado inicial y un estado de espera de n ciclos, siendo n el número de datos que tiene que escribir ($n = 9$ para matrices 3x3 y $n = 3$ para matrices 3x1). Los flip-flops *i_wr_en_delay* y *i_wr_en_delay1* se utilizan para la correcta sincronización de este módulo con los módulos con los que interactúa.

En la tabla 4.5 se describen las señales necesarias para el control.

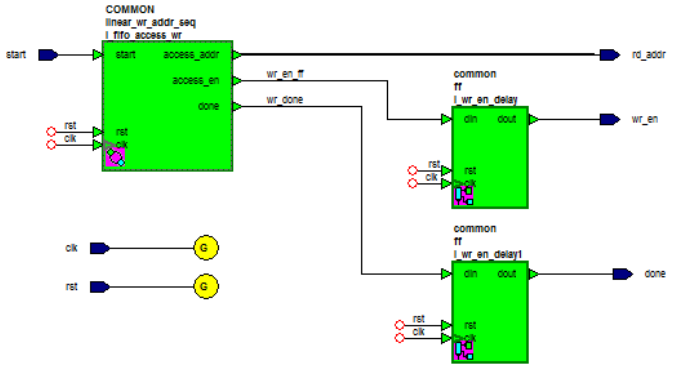


Figura 4.4: Módulo encargado de la escritura de las FIFOs

Nombre	Tamaño	Sentido	Propósito
start	1 bit	entrada	Activa el proceso de escritura
done	1 bit	salida	Fin del proceso de escritura
rd_addr	2/4 bits	salida	Dirección de la memoria RAM del dato que se va a escribir en la FIFO
wr_en	1 bit	salida	Habilita la escritura en la FIFO

Tabla 4.5: Tabla con las señales del módulo de escritura de FIFOs

El cronograma de la figura 4.5 muestra la secuencia de acciones que se llevan a cabo durante la escritura.

- El proceso de escritura comienza cuando se recibe la señal start.
- En el ciclo siguiente se empiezan a proporcionar direcciones para leer desde la RAM que se va a escribir en la FIFO.
- Un ciclo más tarde se habilita la escritura en FIFO. Esto es porque la RAM tarda un ciclo en proporcionar el dato pedido.
- Cuando se ha escrito el último dato, se activa la señal done.

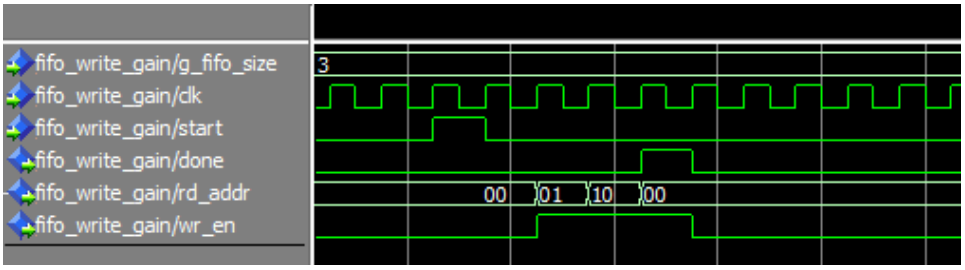


Figura 4.5: Cronograma de la escritura de las FIFOs

Lectura de la FIFO

El módulo encargado de esta tarea es **fifo_read**, cuyo diagrama de bloques se puede observar en la figura 4.6. El control lo realiza el módulo receptor. Se realiza mediante un estado inicial y un estado de espera de n ciclos, siendo n el número de datos que tiene que leer ($n = 9$ para matrices 3x3 y $n = 3$ para matrices 3x1). En la tabla 4.6 se describen las señales necesarias para el control.

Nombre	Tamaño	Sentido	Propósito
start	1 bit	entrada	Activa el proceso de lectura
done	1 bit	salida	Fin del proceso de lectura
rd_en	1 bit	salida	Activa la FIFO para que proporcione los datos que contiene
wr_addr	2/4 bits	salida	Dirección de la memoria RAM sobre la que se almacena el dato leído de la FIFO
wr_en	1 bit	salida	Habilita la escritura en la memoria RAM

Tabla 4.6: Tabla con las señales del módulo de lectura de FIFOs

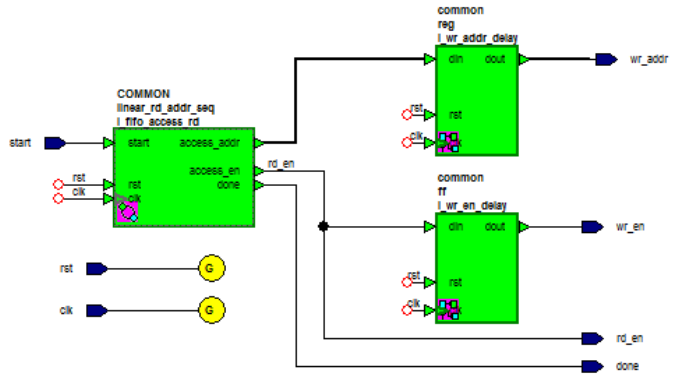


Figura 4.6: Módulo encargado de la lectura de las FIFOs

En el cronograma de la figura 4.7 se puede observar la secuencia de acciones que se llevan a cabo durante la lectura.

- El proceso de lectura comienza cuando se activa la señal start.
- En el ciclo siguiente el módulo envía la habilitación de lectura a la FIFO.
- Un ciclo más tarde activa la escritura en la RAM y empieza a generar

direcciones. Esto se debe a que la FIFO tarda un ciclo en proporcionar el dato.

- Coincidiendo con la escritura en RAM el último dato, se activa la señal done para avisar de que se han leído todos los datos de la FIFO.

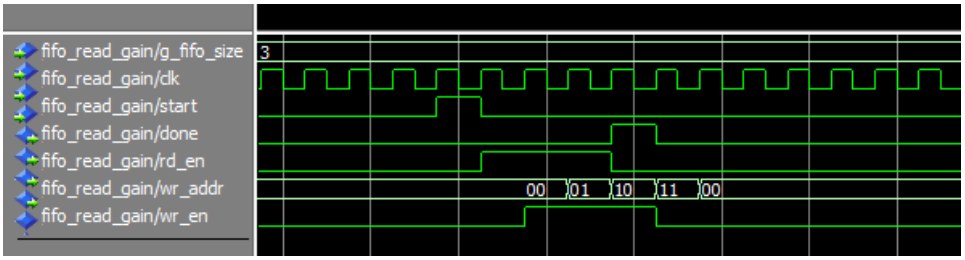


Figura 4.7: Cronograma de la lectura de las FIFOs

Capítulo 5

Módulos Comunes

Las tareas complejas de los módulos principales se pueden descomponer en tareas más simples. Éstas son realizadas por módulos más sencillos que se pueden reutilizar. En este capítulo se clasificarán y se detallarán por tipos dicho módulos.

5.1. Generadores de direcciones

Las matrices del diseño se almacenan por filas en memorias RAM lineales. Para operar con ellas es necesario poder acceder en el orden correcto a sus componentes. Los módulos generadores de direcciones se encargan de esta tarea, proporcionando para cada operación las direcciones adecuadas para cada uno de los operandos. Para la escritura de resultados se usan otros generadores de direcciones que permiten la escritura lineal de resultados sobre las RAMs.

Estos son todos los módulos generadores de direcciones:

- Suma
  - **adder_3x1_addr_seq**
  - **adder_3x3_addr_seq**
- Resta
  - **sub_3x3_addr_seq**
- Multiplicación
  - **mult_1x3_3x1_addr_seq**
  - **mult_3x1_1x3_addr_seq**
  - **mult_3x1_scalar_addr_seq**
  - **mult_3x3_3x1_addr_seq**

- **mult_addr_seq** (para dos matrices de dimensiones 3x3)
- **mult_addr_seq_trasp**

- Escritura lineal
  - **linear_1x3_addr_cntr**
  - **linear_3x3_addr_cntr**

Para explicar estos módulos se ha tomado como ejemplo el secuenciador de la multiplicación de 3x1_1x3 dado que el diseño del resto de los secuenciadores es muy similar.

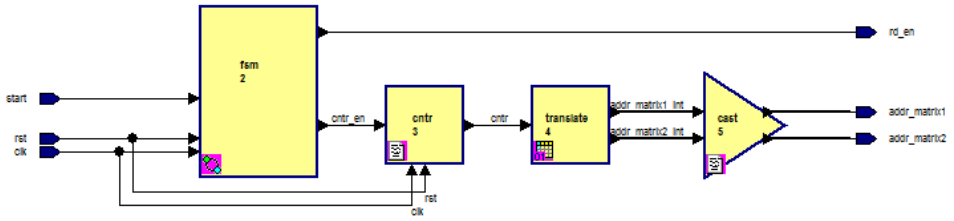


Figura 5.1: Ejemplo de módulo generador de direcciones

- La máquina de estados *fsm* activa una señal de capacitación para el contador que permanece activa un número de ciclos igual al número de elementos que hay que leer de la memoria. En este ejemplo la señal estará activa durante 9 ciclos.
- El contador *cntr* cuenta hasta 9 mientras esté habilitado.
- La tabla de traducción *translate*, implementada mediante una tabla de verdad, traduce cada número generado por el contador en las direcciones de las 2 memorias de las que se va a leer (en la tabla 5.1 se puede ver la traducción de este ejemplo). Esta forma nos permite tener una sola copia de las matrices constantes, ya que cambiando la tabla de traducción podemos leer una matriz de forma traspuesta.
- El módulo *cast* realiza una conversión de entero a vector.

cntr	addr1	addr2	cntr	addr1	addr2
0	0	0	5	1	2
1	0	1	6	2	0
2	0	2	7	2	1
3	1	0	8	2	2
4	1	1			

Tabla 5.1: Tabla de traducción de **mult_3x1_1x3_addr_seq**

5.2. Operadores

Suma de matrices

Estos módulos efectúan la suma de dos matrices componente a componente. Hay dos tipos de módulos sumadores:

- **adder_3x1**
- **adder_3x3**

Los sumadores de matrices se descomponen en un sumador, un registro para sincronizar el dato con la señal que indica que es válido y una máquina de estados (FSM). Su diagrama de bloques se puede ver en la figura 5.2. Los estados de la FSM se explican en la tabla 5.2.

Estado	Acción
init_st	Espera que la señal de start $\leftarrow 1$ para empezar.
add_st	Espera durante tantos ciclos como sumas se deban efectuar y activa valid_data $\rightarrow 1$.
done_st	Indica la conclusión de la suma y activa la señal done $\rightarrow 1$.

Tabla 5.2: Estados de la FSM del sumador

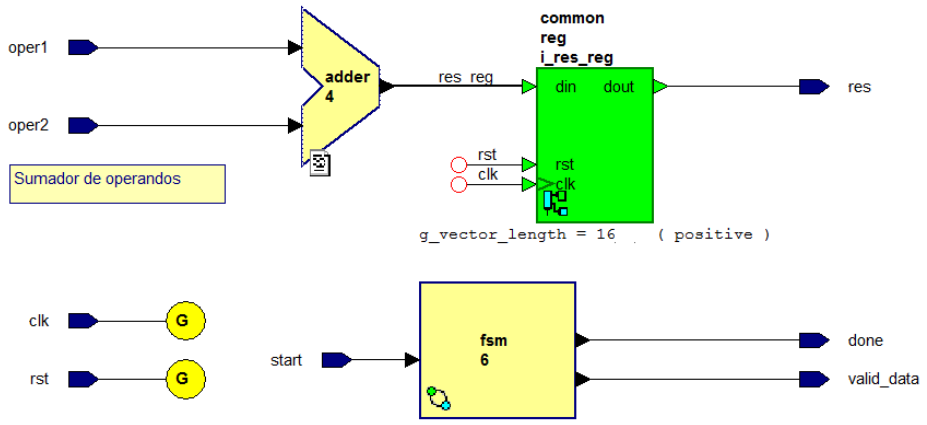


Figura 5.2: Módulo sumador 3x3

Resta de matrices

Al igual que la suma, la resta se efectúa elemento a elemento. Hay un módulo para esta operación:

- **subs_3x3**

Subs_3x3 lleva a cabo la resta entre dos matrices de tamaño 3x3. Para ello usa una instancia de *substractor*[1] como restador, un registro para retardar el resultado un ciclo (para la sincronización del dato con la señal que indica que es válido) y una máquina de estados, similar a la de la suma (tabla 5.3), para controlar su funcionamiento.

Estado	Acción
init_st	Espera que la señal de start ← 1 para empezar.
sub_st	Espera durante tantos ciclos como restas se deban efectuar y activa valid_data → 1.
done_st	Indica la conclusión de la resta y activa la señal done → 1.

Tabla 5.3: Estados de la FSM del restador

En la figura 5.3 se puede observar el diagrama de bloques de este módulo.

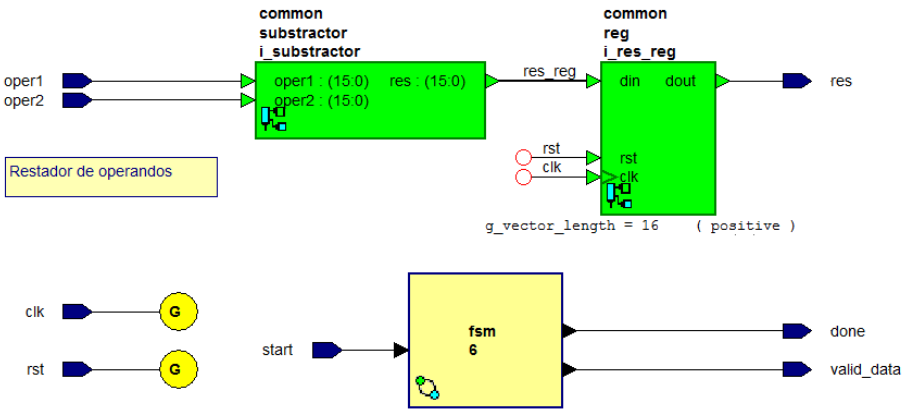


Figura 5.3: Módulo restador 3x3

[1] **substractor** es un restador simple de escalares

Multiplicación de matrices

Estos módulos realizan la multiplicación de una fila por una columna, es decir $a_{11} * b_{11} + a_{12} * b_{21} + a_{13} * b_{31}$. Debido a las peculiaridades de la multiplicación de matrices, es especialmente importante la generación de direcciones para leer en el orden adecuado las componentes de las matrices que se van a multiplicar. Es por ello que los multiplicadores están estrechamente vinculados a los generadores de direcciones. Existen las siguientes variantes, que se diferencian en el número de ciclos:

- **mult_1x3_3x1**
- **mult_1x3_3x3**
- **mult_3x1_1x3**
- **mult_3x1_scalar**
- **mult_3x3_3x1**
- **mult_3x3_3x3**

El diagrama de bloques de este módulo se puede observar en la figura 5.4.

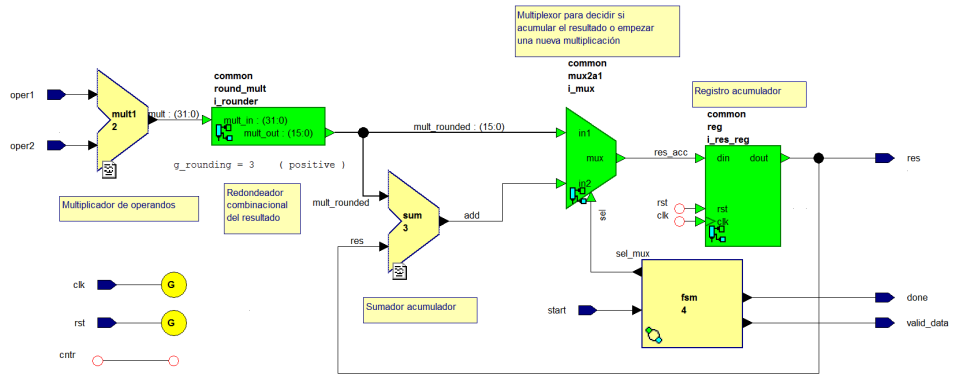


Figura 5.4: Módulo multiplicador 3x3 con 3x3

- La señal start se debe recibir a la vez que los primeros datos a multiplicar ($a_{11} * b_{11}$). Tras multiplicarlos, se les aplica el redondeo (explicado más abajo) y se envían al registro acumulador seleccionando la primera entrada del multiplexor.
- De esta forma cuando la FSM pasa al siguiente estado ya tiene la primera multiplicación en el registro y se puede acumular al resultado de las siguientes (a $a_{11} * b_{11}$ que ya estaba en el registro se le suma $a_{12} * b_{21}$).

- En el estado de acumulación se selecciona la segunda entrada del multiplexor para dejar paso al resultado del sumador, que lleva acumulado el resultado anterior con el que acaba de salir del multiplicador $(a_{11} * b_{11} + a_{12} * b_{21})$. Esto se hace durante dos ciclos en este ejemplo.

- Al terminar se manda una señal para avisar de que el dato que sale es un resultado válido, puesto que cada elemento de la nueva matriz se almacena en la RAM nada más estar disponible. El registro acumulador se borra y se vuelve a empezar.

- En este ejemplo, este proceso se repite 9 veces. Al final se activa otra señal que indica la finalización de la multiplicación de las matrices.

La máquina de estados que controla el multiplicador tiene cuatro estados, detallados en la tabla 5.4. En el segundo estado de la tabla n establece el número de ciclos en función de los operandos que se tengan que acumular.

Estado	Acción
init_st	Espera que la señal de start $\leftarrow 1$ para empezar.
mult_st	Espera n ciclos y selecciona la entrada del mux para acumular.
wr_st	Indica que el dato que llega es el resultado de la acumulación.
done_st	Indica la conclusión de la multiplicación.

Tabla 5.4: Estados de la FSM del multiplicador

Para que el resultado de la operación de multiplicación sea el correcto, se debe redondear debido a la aritmética en punto fijo[2]. Para ello se utiliza un módulo de redondeo, en el cual se suma un 1 al bit 4 debido a que es el bit más significativo de la parte que se quiere redondear. Después, se realiza un desplazamiento a la derecha de 4 bits. Por último, se restablece el signo del número inicial.

Poniendo un ejemplo, supóngase que se quieran multiplicar los números 3,6875 y 4,25. Su representación en notación $Q_{12,4}$ es:

$$3,6875 = 0000000000111011 = 59_{(10)}$$
$$4,25 = 0000000001000100 = 68_{(10)}$$

Por lo tanto,

$$3,6875 * 4,25 = 15,671875$$

que corresponde a

[2]Ver apendice A.

$$\begin{array}{r} 0000000000111011 \\ *0000000001000100 \\ \hline 00000000000000000000111110101100 \end{array}$$

Nótese que el resultado binario tiene el doble de bits. Esto es porque al multiplicar dos vectores de 16 bits da como resultado uno de 32 bits.

Interpretándolo como números enteros, la multiplicación corresponde a

$$59 * 68 = 4012$$

El resultado de la multiplicación binaria no representa el resultado de los números reales. Para que corresponda es necesario realizar una serie de operaciones sobre aquél. Los pasos a realizar son:

1. El resultado anterior requiere ser redondeado. En este ejemplo se van a redondear 4 bits. Para ello se debe sumar un 1 al bit más significativo que se va a redondear (el cuarto en este caso).

$$\begin{array}{r} 00000000000000000000111110101100 \\ +1000 \\ \hline 00000000000000000000111110110100 \end{array}$$

2. Se eliminan los 4 últimos bits haciendo un desplazamiento a la derecha.

$$00000000000000000000111110110\cancel{1100}$$

3. Se mantiene el bit de signo y se extraen los 15 bits menos significativos.

$$15,6875 = 0000000011111011 = 251_{(10)}$$

El resultado ahora sí es el correcto, aunque se puede observar que no coincide exactamente con el resultado de la operación en números reales. Sin embargo 15,6875 es la representación más próxima y precisa que se puede hacer de 15,671875 en notación $Q_{12,4}$.

División o inversión

En el módulo GAIN se debe efectuar una división de una matriz por otra. Para el caso particular que se ha dado en este proyecto, la matriz divisor tiene unas dimensiones de 1x1, es decir, es un escalar, por lo que el módulo **inversion** realiza una división simple de una matriz entre un escalar. En la figura 5.5 se puede observar el diagrama de este módulo.

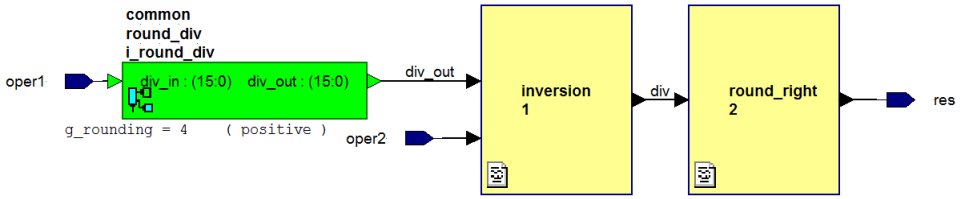


Figura 5.5: Módulo que realiza la división

- Una vez recibidos los dos operandos, el dividendo entra en el módulo *i_round_div* que desplazará hacia la izquierda 5 bits.
- Tras el desplazamiento, se realiza la operación de división en el bloque *inversion*, con la particularidad de sumar 1 al bit 0 debido a que sólo en el caso de que haya acarreo, el bit 1 pasa a valer 1.
- Después el bloque *round_right* realiza un desplazamiento de 1 bit hacia de la derecha. Esto se ha implementado de esta forma debido a que se opera en punto fijo[3].

Poniendo un ejemplo del funcionamiento de la división, supóngase que se quieran dividir los números 4.125 y 5.125. Su representación en notación $Q_{12,4}$ es:

$$4{,}125 = 0000000001000010 = 66_{(10)}$$
$$5{,}125 = 0000000001010010 = 82_{(10)}$$

Por lo tanto,

$$4{,}125/5{,}125 = 0{,}804878$$

que corresponde a

$$\begin{array}{r} 0000000001000010 \\ /0000000001010010 \\ \hline 0000000000001101 \end{array}$$

Interpretándolo como números enteros, la división corresponde a

$$66/82 = 0{,}804878$$

El problema que surge es que este valor no existe con esta representación, y con redondear mediante el método utilizado en la multiplicación no se consigue un valor correcto. Por lo tanto, es necesario buscar el valor más cercano realizando las siguientes operaciones:

[3] Ver apéndice A

1. Se realiza el desplazamiento al dividendo: 1000010 desplazado 5 bits a la izquierda nos da 100001000000.
2. Se realiza la división: 100001000000 / 1010010 = 11001.
3. Se sumam 1 al resultado: 11001 + 1 = 11010.
4. Se desplaza 1 bit hacia la derecha: el resultado es 1101, que en aritmética de punto fijo es 0.8125, por lo que se redondea al par más cercano.

5.3. Memorias RAM

Estas memorias de lectura y escritura permiten almacenar las diferentes matrices necesarias para los cálculos.

En el proyecto se han utilizado 3 tipos de memorias RAM:

- **ram_3x1**
- **ram_3x3**

Estas memorias son de doble puerto:

- El primero para la escritura, con las señales
  - addra: dirección de escritura
  - dina: dato de entrada
  - wea: habilitación de escritura
- El segundo para la lectura, con las señales
  - addrb: dirección de lectura
  - doutb: dato de salida

No es necesaria una señal de capacitación de lectura de las RAM, ya que el hecho de proporcionar una dirección por el puerto de lectura hace que esta entregue los datos que contenga. El dato estará en la salida doutb un ciclo después de la llegada de la dirección.

La escritura se realiza mientras wea esté activada. En cada ciclo que esta señal esté activa se almacena el dato que esté en la entrada dina en la dirección proporcionada en la entrada addra.

El cronograma de la figura 5.6 ilustra las secuencias de acciones durante una escritura y posteriormente una lectura de la misma.

Como la **ram_3x1** puede almacenar hasta 3 elementos, no necesita más de 2 bits de direcciones para poder acceder a ellos. Por ello, todos los componentes con los que interactúa usan buses de 2 bits para la dirección, mientras que los componentes que interactúan con las **ram_3x3** usan buses de 4 bits.

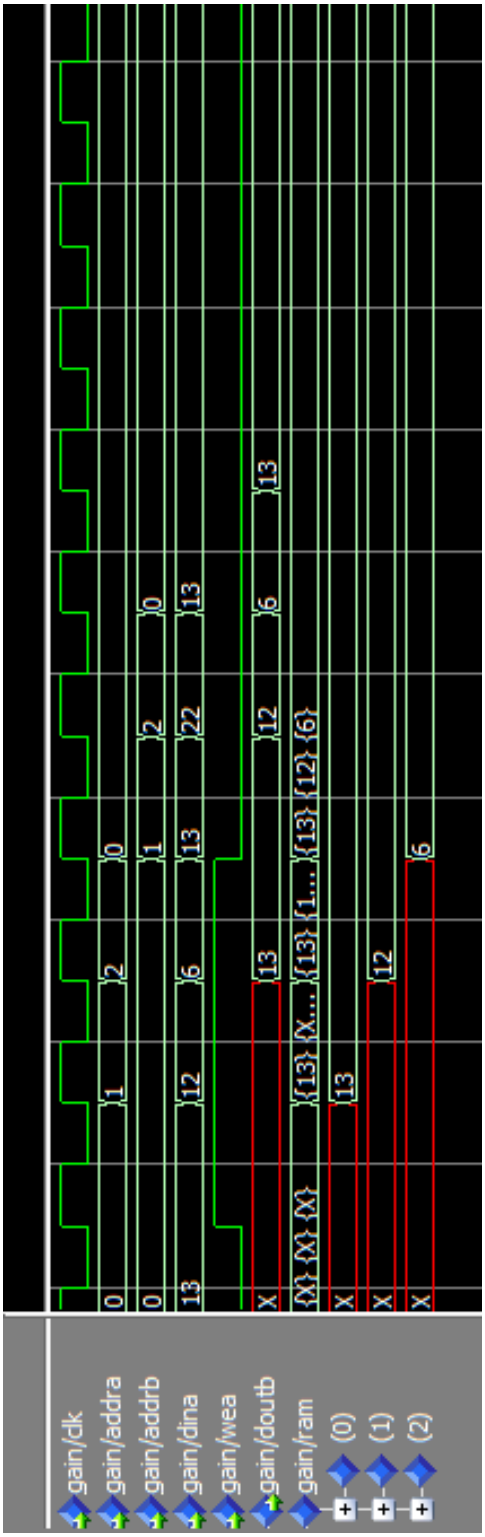


Figura 5.6: Cronograma de la escritura/lectura en una RAM

Capítulo 6

Módulo Prediction

Este módulo implementa las ecuaciones del filtro correspondientes a la etapa de predicción:

$$\hat{x}_k^- = A\hat{x}_{k-1} \tag{6.1}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{6.2}$$

Puesto que las dos ecuaciones son independientes y se pueden calcular en paralelo, se ha descompuesto el módulo en dos submódulos (ver figura 6.1):

- *i_ calc_ x*: implementa la ecuación 6.1.
- *i_ calc_ p*: implementa la ecuación 6.2.

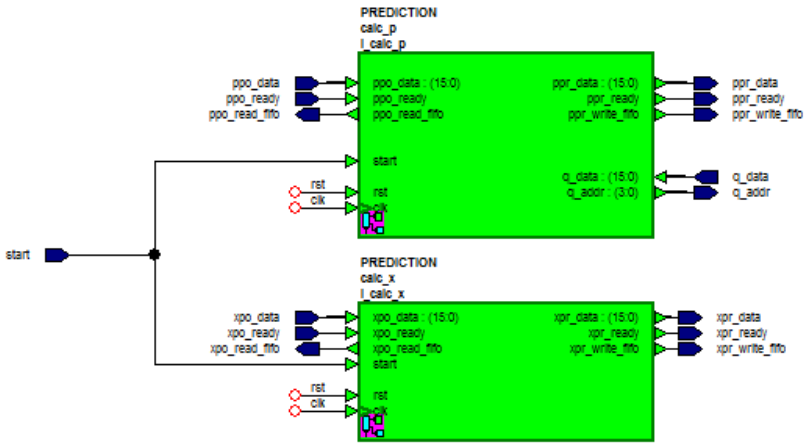


Figura 6.1: *Top-level* del módulo Prediction

En la tabla 6.1 se decribe la interfaz del módulo. La forma de agrupar estas señales en la tabla corresponde a las necesitadas para el correcto intercambio de información mediante la FIFO: para cada matriz intercambiada existe un bus de datos, una señal que avisa de la disponibilidad del dato y una señal de petición de lectura o escritura, según el caso. Las últimas tres señales, que no corresponden a información intercambiada, se propagan desde este módulo hasta el exterior del FK.

Nombre	Tamaño	Sentido	Comentario
xpo_data	16 bits	entrada	Viene de la FIFO *i_correction2prediction_x*.
xpo_read_fifo	1 bit	salida	Activa lectura de datos de la FIFO.
xpo_ready	1 bit	entrada	Enviado por CORRECTION cuando XPO está en la FIFO.
xpr_data	16 bits	salida	Va a la FIFO *i_prediction2correction_x*.
xpr_write_fifo	1 bit	salida	Activa la escritura en la FIFO.
xpr_ready	1 bit	salida	Enviado a CORRECTION cuando XPR está en la FIFO.
ppo_data	16 bits	entrada	Viene de la FIFO *i_correction2prediction_p*.
ppo_read_fifo	1 bit	salida	Activa lectura de datos de la FIFO.
ppo_ready	1 bit	entrada	Enviado por CORRECTION cuando PPO está en la FIFO.
ppr_data	16 bits	salida	Va a las FIFOs *i_prediction2correction_p* y *i_prediction2gain_p*.
ppr_write_fifo	1 bit	salida	Activa la escritura en las FIFOs.
ppr_ready	1 bit	salida	Enviado a CORRECTION y GAIN cuando PPR está en la FIFO.
q_addr	4 bits	salida	Proporciona la dirección de lectura de Q.
q_data	16 bits	entrada	Viene de la memoria externa Q.
start	1 bit	entrada	Inicia el sistema.

Tabla 6.1: Interfaz del módulo PREDICTION

6.1. Inicio de la predicción

Es importante hacer notar que es en este módulo donde se realiza la inicialización del sistema mencionada en la sección 4.2. Para ello:

- se multiplexa la entrada de las memorias *i_mem_xpo* y *i_mem_ppo* para que se pueda elegir entre los datos iniciales o los datos provenientes de las FIFOs.

- las unidades de control tienen unas fases de inicialización y posteriormente entran en un bucle infinito.

Fase de inicialización

Empieza al activarse la entrada start, que marca el inicio global del sistema. Durante esta fase se procede a cargar unos datos iniciales en las memorias para que se puedan empezar a hacer cálculos. En esta fase las unidades de control seleccionan la entrada del dato inicial.

Resto de iteraciones

Una nueva iteración empieza cuando desde el módulo CORRECTION se reciben las señales xpo_ready y ppo_ready. En estas iteraciones ya existen datos en las FIFOs, previamente calculados por el módulo CORRECTION. Por ello las unidades de control seleccionan la entrada del multiplexor correspondiente al dato proveniente de la FIFO.

6.2. Cálculo de XPR: módulo *i_calc_x*

Implementa la expresión $\hat{x}_k^- = A\hat{x}_{k-1}$, que predice el estado *a priori* a partir del estado estimado *a posteriori* y la matriz A. El proceso se realiza en varias fases:

1. Cuando se recibe desde CORRECTION la señal xpo_ready que indica que XPO está listo, se procede a leerlo desde la fifo *i_correction2prediction_x* para almacenarlo en la memoria local *i_mem_xpo*.

2. Una vez leídos todos los datos y guardados en la memoria, se empieza a hacer la multiplicación de la matriz constante A con la matriz que acabamos de guardar en *i_mem_xpo*.

3. A medida que salen los datos del módulo de multiplicación se van almacenando en la memoria *i_mem_xpr*.

4. Cuando se hayan almacenado todos los datos en la memoria, se escriben éstos en la FIFO *i_prediction2correction_x* y se avisa al módulo CORRECTION de que XPR está listo activando xpr_ready.

En la figura 6.2 tenemos el diagrama de bloques de este módulo.

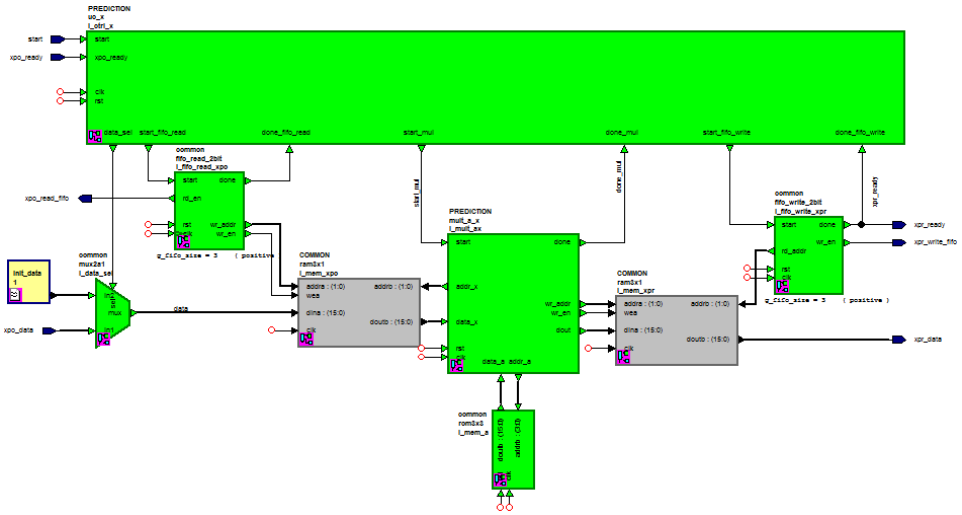


Figura 6.2: Diagrama de bloques del módulo *i_calc_x*

Almacenamiento de los datos

Para almacenar los datos leídos de las FIFOs o los resultados de las operaciones intermedias se usan memorias RAM y para aquellos datos que son constantes se han usado memorias ROM inicializadas con valores constantes.

En la tabla 6.2 se detallan todas las memorias instanciadas en este módulo.

Instancia	Tipo	Origen	Destino	Almacena
i_mem_a	ROM 3x3	-	*i_mult_ax*	*A*
i_mem_xpo	RAM 3x1	*i_correction2-prediction_x*	*i_mult_ax*	XPO
i_mem_xpr	RAM 3x1	*i_add*	*i_prediction2-correction_x*	XPR

Tabla 6.2: Instancias de memorias usadas en el módulo *i_calc_x*

Unidad de control: módulo *i_ ctrl_ x*

Es una instancia de la entidad **uc_x**, que implementa una máquina de estados para controlar la secuencia de operaciones que este módulo debe llevar a cabo. En la figura 6.3 se muestra el diagrama de transición de estados y en la tabla 6.3 se detallan los estados de esta unidad de control.

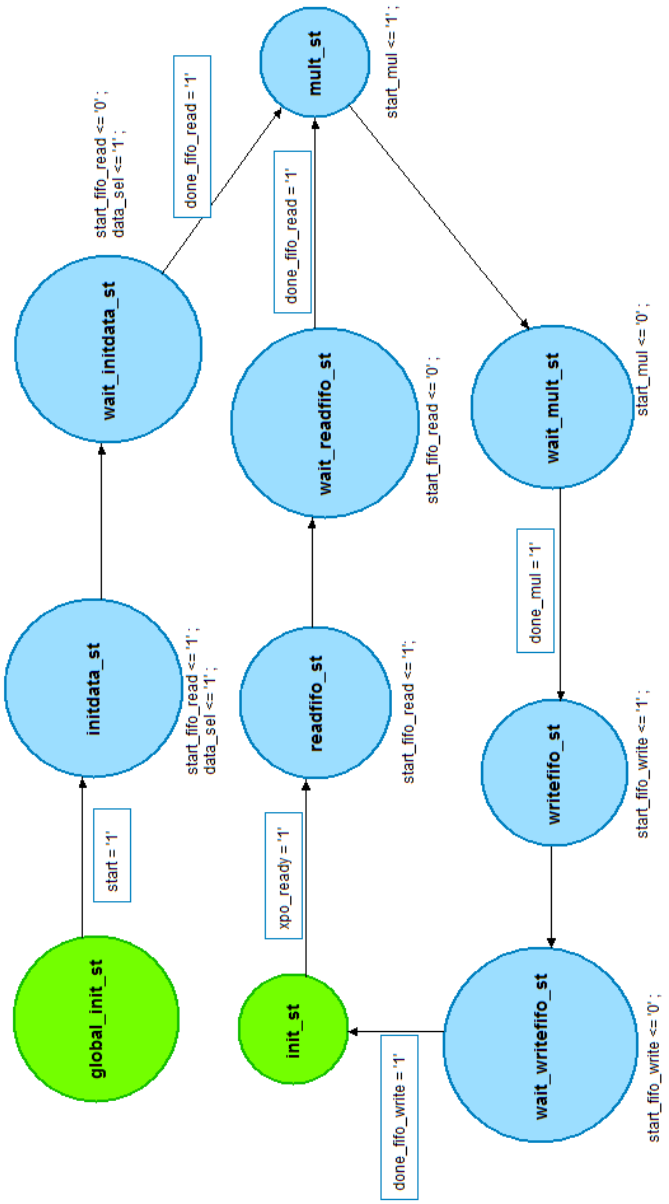


Figura 6.3: Unidad de control del módulo *i_ calc_ x*

Estado	Acción
global_init_st	El sistema está a la espera del inicio global, hasta que señal start ← 1.
initdata_st	Inicia la escritura de las memorias con los valores iniciales en vez de leer datos de la FIFO, ya que está vacía. Para ello activa la señal start_fifo_read → 1. Selecciona la segunda entrada del multiplexor *i_data_sel* que está en la entrada de datos de la memoria *i_mem_xpo* (data_sel → 1). De esta forma se aprovecha la generación de direcciones del módulo *i_fifo_read_xpo* para ir almacenando los valores.
wait_initdata_st	Desactiva la señal de inicio start_fifo_read → 0 y espera a que termine la inicialización, hasta que done_fifo_read ← 1. Mantiene seleccionada la segunda entrada del multiplexor *i_data_sel* (data_sel → 1). En este estado se terminan de inicializar las memorias.
mult_st	Activa la señal de inicio de la multiplicación start_mul → 1.
wait_mult_st	Desactiva la señal de inicio start_mul → 0. En este estado se termina de realizar la multiplicación hasta que se avise de su conclusión, recibiendo done_mul ← 1.
writefifo_st	Inicia el módulo *i_fifo_write_xpr* para que escriba XPR en FIFO poniendo start_fifo_write → 1.
wait_writefifo_st	Desactiva la señal de inicio start_fifo_write → 0. En este estado sigue escribiendo en la FIFO hasta que se recibe el aviso de que se han escrito todos los datos, cuando done_fifo_write ← 1.
init_st	Permanece a la espera de que XPO esté listo para iniciar otra iteración del sistema, cuando la señal xpo_ready ← 1.
read_fifo_st	Inicia el módulo de lectura de la FIFO *i_fifo_read_xpo* activando la señal start_fifo_read → 1. Se selecciona la primera entrada del multiplexor *i_data_sel* (data_sel → 0).
wait_readfifo_st	Desactiva la señal de inicio de lectura de FIFO start_fifo_read → 0 y mantiene seleccionada la primera entrada del multiplexor *i_data_sel* (data_sel → 0). En este estado se sigue leyendo de la FIFO hasta que se recibe el aviso de que se han leído todos los datos, cuando la señal done_fifo_read ← 1

Tabla 6.3: Estados de la unidad de control del módulo *i_calc_x*

Lectura de la FIFO

Lo primero que se hace al activarse la señal xpo_ready es empezar a leer XPO desde la FIFO *i_correction2prediction_x* para almacenarlo en la memoria *i_mem_xpo*.

El módulo encargado de esta tarea es *i_fifo_read_xpo*, que es una instancia del módulo **fifo_read** explicado en la sección 4.5. En este caso el valor de n es 3, puesto que XPO es de 3 elementos.

Multiplicación: módulo *i_mult_ax*

Realiza la multiplicación $A\hat{x}$. En la figura 6.4 se puede observar el diagrama de bloques de la entidad **mult_a_x** que ha sido instanciada para esta tarea, y en la tabla 6.4 se detallan los módulos de los que está formado.

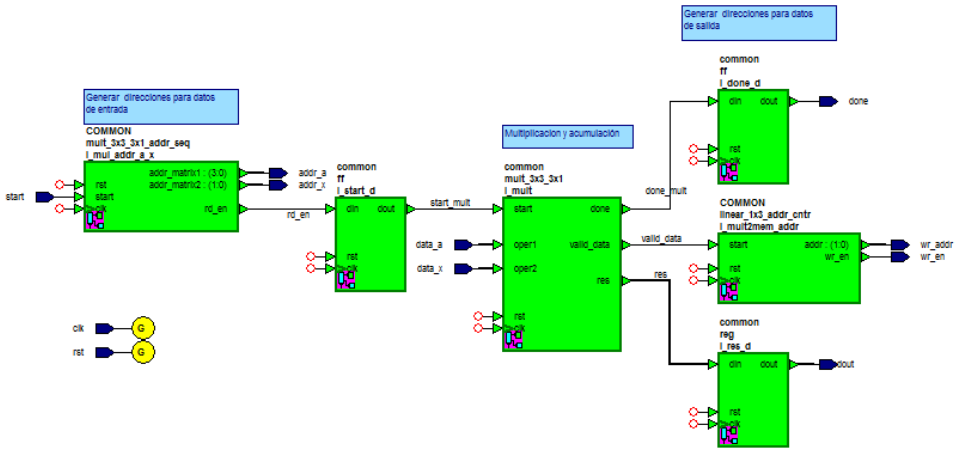


Figura 6.4: Diagrama de bloques del módulo *i_mult_ax*

Este módulo empieza su actividad en cuanto se activa la señal start. Ésta entra al módulo *i_mul_addr_a_x* que empieza a sacar, cada ciclo, las direcciones de lectura para pedirle los componentes en el orden correcto a las memorias. También activa una señal para que el módulo *i_mult* comience a multiplicar y acumular datos. Esta señal se retrasa un ciclo mediante un *flip-flop* para sincronizar el inicio de la multiplicación con la llegada de los datos correctos, debido a que las memorias *i_mem_a* y *i_mem_xpo* tardan un ciclo en proporcionar el dato pedido.

Una vez que el multiplicador empieza a sacar resultados válidos, activa el módulo *i_mult2mem_addr*, que genera las direcciones para el almacenamiento de las componentes del resultado. De esta forma los resultados se van almacenando en la memoria *i_mem_xpr* a medida que van saliendo del multiplicador. Debido a que el generador tarda un ciclo en proporcionar la

dirección, es necesario sincronizar el dato con la dirección a la que se tiene que almacenar, para lo que se utiliza un registro.

Cuando se han terminado de multiplicar todos los datos el multiplicador activa la señal done. Ésta señal se debe retrasar un ciclo para sincronizarla con el último dato que se va a escribir, pues de lo contrario se avisaría antes de haber guardado resultado completo.

Instancia	Entidad	Función
i_mul_addr_a_x	**mult_3x3_3x1_addr_seq**	Genera las direcciones de lectura de los datos.
i_mult	**mult_3x3_3x1**	Realiza la multiplicación acumulada.
i_mult2mem_addr	**linear_1x3_addr_cntr**	Genera las direcciones de escritura de los datos.
i_start_d	**ff**	Sincroniza el inicio de la multiplicación con la llegada de los primeros datos desde la memoria.
i_done_d	**ff**	Sincroniza la señal done con el úlitmo dato guardado en memoria.
i_res_d	**reg**	Sincroniza el dato con la dirección en la que se almacena.

Tabla 6.4: Elementos del módulo *i_mult_ax*

Escritura en la FIFO

Una vez que se hayan almacenado todos los datos resultantes de la multiplicación en la memoria *i_mem_xpr*, se procede a escribir éstos en la FIFO *i_correction2prediction_x*. Cuando estén todos almacenados en la FIFO se avisa al módulo CORRECTION de que XPR está listo activando la señal xpr_ready. De esto se encarga *i_fifo_write_xpr*, que es una instancia de la entidad **fifo_write** explicada en la sección 4.5. En este caso también tenemos que el valor de n es 3, puesto que PPR es de 3 elementos.

6.3. Cálculo de PPR: módulo *i_ calc_ p*

Implementa la expresión $P_k^- = AP_{k-1}A^T + Q$, que calcula la covarianza del error de estimación *a priori* a partir de la misma *a posteriori*, A y el parámetro externo Q. Este proceso se realiza en varias fases:

1. Cuando el módulo recibe desde CORRECTION la señal ppo_ready que indica que PPO está lista, se lee desde la fifo *i_ correction2prediction_ p* para almacenarlo en la memoria local *i_ mem_ ppo*.

2. Una vez guardado en la memoria, se empieza a hacer la primera multiplicación, cuyos factores son la matriz constante A y la matriz que acabamos de guardar en *i_ mem_ ppo*.

3. A medida que salen los datos del módulo de multiplicación se van almacenando en la memoria auxiliar *i_ mem_ ap* para que se pueda usar este resultado en la siguiente multiplicación.

4. Al terminar la primera multiplicación empieza la segunda, que multiplica el resultado de la anterior (almacenado en *i_ mem_ ap*) por la matriz constante A^T. En *i_ mem_ a* está almacenada A, por lo que para esta operación tendrá que ser leída de forma traspuesta.

5. Los datos resultado de esta segunda multiplicación se almacenan en una segunda memoria auxiliar *i_ mem_ apa*, para facilitar su uso en la tercera operación.

6. Por último, a la matriz resultado de la segunda multiplicación se le suma la matriz Q.

7. El resultado de esta operación se almacena directamente en la memoria *i_ mem_ ppr* para que sea escrito posteriormente en la FIFO.

8. Cuando se hayan almacenado todos los datos en la memoria, se escriben estos en la FIFO *i_ prediction2correction_ p* y se avisa al módulo CORRECTION de que PPR está lista, activando la señal ppr_ready.

En la figura 6.5 tenemos el diagrama de bloques de este módulo.

Almacenamiento de los datos

Igual que en el módulo *i_ calc_ x*, se usan memorias RAM para los datos y ROM para las constantes. En la tabla 6.5 se detallan las instancias de dichas memorias.

Debido a que el contenido de la memoria *i_ mem_ a* se utiliza tanto en *i_ mult1* como en *i_ mult2* se ha tenido que multiplexar el acceso de lectura

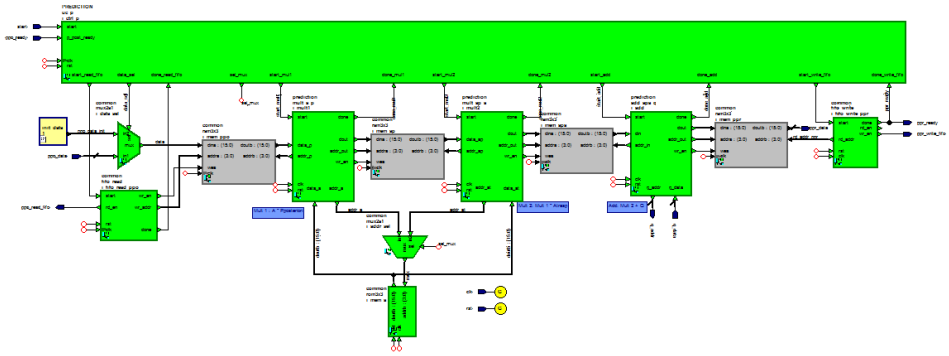


Figura 6.5: Diagrama de bloques del módulo *i_calc_p*

de la misma, estando la unidad de control a cargo de seleccionar de qué módulo llega la dirección de lectura de la misma en un momento dado.

Instancia	Tipo	Origen	Destino	Almacena
i_mem_a	ROM 3x3	-	*i_mult1*, *i_mult2*	A
i_mem_ppo	RAM 3x3	*i_correction2-prediction_p*	*i_mult1*	PPO
i_mem_ap	RAM 3x3	*i_mult1*	*i_mult2*	AP
i_mem_apa	RAM 3x3	*i_mult2*	*i_add*	$(AP)A^T$
i_mem_ppr	RAM 3x3	*i_add*	*i_prediction2-correction_p*, *i_prediction-2gain_p*	PPR

Tabla 6.5: Instancias de memorias usadas en el módulo *i_calc_p*

Unidad de control: módulo *i_ctrl_p*

Es una instancia de la entidad **uc_p**, implementada mediante una máquina de estados, y su misión es controlar la secuencia de operaciones que se llevan a cabo en este módulo. En la figura 6.6 se muestra el diagrama de estados y sus transiciones. En la tabla 6.6 se detallan los estados de esta unidad de control.

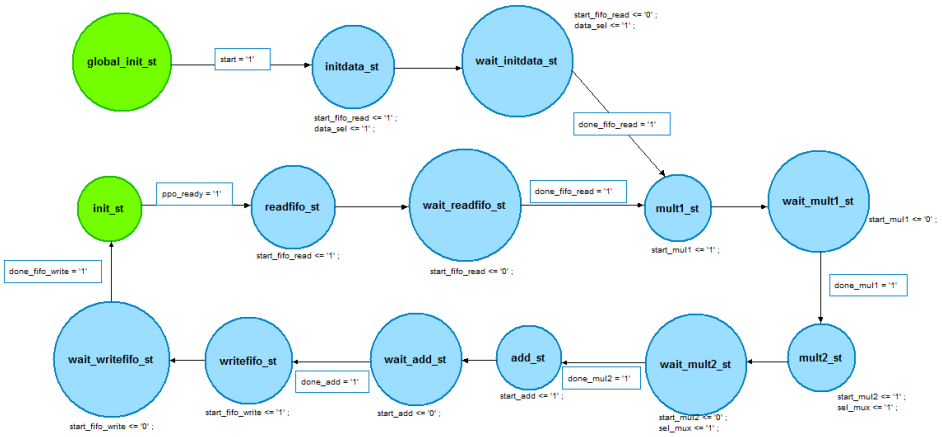


Figura 6.6: Unidad de control del módulo *i_calc_p*

Tabla 6.6: Estados de la unidad de control del módulo *i_calc_p*

Estado	Acción
global_init_st	El sistema está a la espera del inicio global, hasta que la señal start → 1.
initdata_st	Inicia la escritura de las memorias con los valores iniciales en vez de leer datos de la FIFO, ya que está vacía. Para ello activa la señal start_fifo_read → 1. Selecciona la segunda entrada del multiplexor *i_data_sel* que está en la entrada de datos de la memoria *i_mem_ppo* (data_sel → 1). De esta forma se aprovecha la generación de direcciones del módulo *i_fifo_read_ppo* para ir almacenando los valores.
wait_initdata_st	Desactiva la señal de inicio start_fifo_read → 0 y espera a que termine la inicialización, hasta que done_fifo_read ← 1. Mantiene seleccionada la segunda entrada del multiplexor *i_data_sel* (data_sel → 1). En este estado se terminan de inicializar las memorias.
mult1_st	Activa la señal de inicio de la primera multiplicación start_mul1 → 1. Selecciona la primera entrada del multiplexor *i_addr_sel* poniendo sel_mux → 0 para enviar a la ROM *i_mem_a* las direcciones de lectura provenientes desde *i_mult1*.

Continúa en la página siguiente

Unidad de control del módulo *i_calc_p* (continuación)

wait_mult1_st	Desactiva la señal de inicio start_mul1 → 0. Mantiene seleccionada la primera entrada del multiplexor *i_addr_sel* (sel_mux → 0). En este estado se termina de realizar la primera multiplicación hasta que se avisa de que ha concluido, cuando done_mul1 ← 1.
mult2_st	Activa la señal de inicio de la segunda multiplicación start_mul2 → 1. Selecciona la segunda entrada del multiplexor *i_addr_sel* (sel_mux → 1) para enviar a la ROM *i_mem_a* las direcciones de lectura provenientes desde *i_mult2*.
wait_mult2_st	Desactiva la señal de inicio start_mul2 → 0. Mantiene seleccionada la segunda entrada del multiplexor *i_addr_sel* (sel_mux → 1). En este estado se termina de realizar la segunda multiplicación hasta que se avisa de que ha concluido, cuando done_mul2 ← 1.
add_st	Activa la señal de inicio del módulo sumador start_add → 1.
wait_add_st	Desactiva la señal de inicio del módulo sumador start_add → 0. En este estado se termina de realizar la suma hasta que se recibe el aviso de su conclusión done_add ← 1.
writefifo_st	Inicia el módulo *i_fifo_write_ppr* para que escriba PPR en la FIFO poniendo start_fifo_write → 1.
wait_writefifo_st	Desactiva la señal de inicio start_fifo_write → 0. En este estado sigue escribiendo en la FIFO hasta que se recibe el aviso de que se han escrito todos los datos, cuando done_fifo_write ← 1.
init_st	Permanece a la espera de que PPO esté listo para iniciar otra iteración del sistema, cuando la señal ppo_ready ← 1.
readfifo_st	Inicia el módulo de lectura de la FIFO *i_fifo_read_ppo* activando la señal start_fifo_read → 1. Se selecciona la primera entrada del multiplexor *i_data_sel* (data_sel → 0).
wait_readfifo_st	Desactiva la señal de inicio de lectura de FIFO start_fifo_read → 0 y mantiene seleccionada la primera entrada del multiplexor *i_data_sel* (data_sel → 0). En este estado se sigue leyendo de la FIFO hasta que se recibe el aviso de que se han leído todos los datos, cuando done_fifo_read ← 1.

Lectura de la FIFO

Al activarse la señal ppo_ready, lo primero que se tiene que hacer es leer PPO desde la FIFO *i_correction2prediction_p* para almacenarla en la memoria *i_mem_ppo*. El módulo que se encarga de esta tarea es *i_fifo_read_ppo*, que es una instancia del módulo **fifo_read** explicado en la sección 4.5. En este caso el valor de n es 9, puesto que PPO es de 9 elementos.

Primera multiplicación: módulo *i_mult1*

La primera operación de este módulo tras cargar los datos de la FIFO consiste en multiplicar la matriz A por la matriz P. Esta operación se realiza mediante un módulo específico **mult_a_p**, cuyo diseño es muy similar al módulo de multiplicación explicado en la sección 6.2.

Las diferencias con respecto al módulo explicado en dicha sección consisten en el uso de máquinas generadoras de direcciones y el multiplicador particularizados para el tamaño de las matrices con las que se opera en este módulo (matrices de dimensiones 3x3). Este módulo lee los datos de las memorias *i_mem_a* y *i_mem_ppo* y almacena los resultados en la memoria *i_mem_ap*.

En la tabla 6.7 se detallan los módulos que han sido particularizados para esta operación y en la tabla 6.10 (izquierda) se pueden consultar la tabla de verdad que particulariza el orden las direcciones de acceso para cada las matrices que se multiplican en esta operación.

Instancia	Entidad	Función
i_mult_seq	**mult_addr_seq**	Genera las direcciones de lectura de los datos.
i_mult	**mult_3x3_3x3**	Multiplica A por P.
i_addr_cntr	**linear_3x3_addr_cntr**	Genera las direcciones de escritura de los datos.

Tabla 6.7: Elementos particularizados para el módulo *i_mult1*

Segunda multiplicación: módulo *i_mult2*

La segunda operación consiste en multiplicar el resultado de la multiplicación anterior por la matriz A^T. Esta operación la realiza un módulo similar al anterior llamado **mult_ap_a**, pero con un generador de direcciones diferente. La principal particularidad de este generador se debe a que no existe una instancia de una memoria con el equivalente traspuesto de la matriz A. Sin embargo se puede acceder de forma traspuesta a sus elementos

simplemente variando el orden de acceso a los mismos en la tabla de verdad del generador de direcciones. Por otro lado, se debe leer esta matriz como segundo operando de la multiplicación, por lo que además de leerla de forma traspuesta se tiene que leer por columnas.

Este módulo lee los datos de las memorias *i_mem_ap* y *i_mem_a* y almacena el resultado en la memoria intermedia *i_mem_apa*.

En la tabla 7.8 se detallan los módulos que han sido particularizados para esta operación. Y en la tabla 6.10 (centro) se puede consultar la tabla de verdad que particulariza las direcciones de acceso para esta operación. Si se compara con la tabla de la izquierda se puede ver la diferencia en el orden de acceso a los elementos de A para su lectura de forma traspuesta.

Instancia	Entidad	Función
i_mult_seq	**mult_addr_seq_trasp**	Generador de direcciones para que lea A^T.
i_mult	**mult_3x3_3x3**	Multiplica (AP) por A^T.
i_addr_cntr	**linear_3x3_addr_cntr**	Genera las direcciones de escritura de los datos.

Tabla 6.8: Elementos particularizados para el módulo *i_mult2*

Suma: módulo *i_add*

Después de haber hecho las dos multiplicaciones solo queda sumar al resultado la matriz Q. El módulo encargado de esta tarea es una instancia de la entidad **add_apa_q**, cuyo diseño es similar a los módulos de multiplicación, variando los generadores de direcciones y cambiando el multiplicador por un sumador de matrices. Este módulo coge por un lado los datos de la memoria *i_mem_apa*, y por otro lado la matriz Q, que está almacenada en una memoria externa al Filtro de Kalman y que se accede a través de una interfaz de lectura que se propaga hasta el exterior. El resultado se almacena en la memoria *i_mem_ppr*.

En la tabla 6.9 se detallan los módulos específicos de la suma y en la tabla 6.10 se puede consultar la tabla de verdad que particulariza las direcciones de acceso para esta operación. Nótese que al ser una operación de suma, el acceso es lineal.

Instancia	Entidad	Función
i_add_seq	**adder_3x3_addr_seq**	Genera las direcciones de lectura de los datos.
i_adder	**adder_3x3**	Suma APA^T y Q.
i_add_cntr	**linear_3x3_addr_cntr**	Genera las direcciones de escritura de los datos.

Tabla 6.9: Elementos particularizados para el módulo *i_add*

i_mult1		
cntr	A	P
0	0	0
1	1	3
2	2	6
3	0	1
4	1	4
5	2	7
6	0	2
7	1	5
8	2	8
9	3	0
10	4	3
11	5	6
12	3	1
13	4	4
14	5	7
15	3	2
16	4	5
17	5	8
18	6	0
19	7	3
20	8	6
21	6	1
22	7	4
23	8	7
24	6	2
25	7	5
26	8	8

i_mult2		
cntr	AP	A^T
0	0	0
1	1	1
2	2	2
3	0	3
4	1	4
5	2	5
6	0	6
7	1	7
8	2	8
9	3	0
10	4	1
11	5	2
12	3	3
13	4	4
14	5	5
15	3	6
16	4	7
17	5	8
18	6	0
19	7	1
20	8	2
21	6	3
22	7	4
23	8	5
24	6	6
25	7	7
26	8	8

i_add		
cntr	APA^T	Q
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8

Tabla 6.10: Tablas de verdad para el acceso a datos de diferentes operaciones

Escritura en la FIFO

Cuando se ha almacenado el resultado de la suma en *i_mem_ppr*, se procede a su escritura en la FIFO *i_prediction2correction_p*. Una vez estén todos los datos escritos en la FIFO se avisa al módulo CORRECTION de que PPR ya está lista para ser usada. Para ello se activa la señal ppr_ready. El módulo encargado de esta tarea es *i_fifo_write_ppr*, instancia del módulo **fifo_write** explicado en la sección 4.5. En este caso el valor de n es 9, puesto que el tamaño de la matriz de PPR es de 9 elementos.

Capítulo 7

Módulo GAIN

Este módulo calcula la ganancia de Kalman, que se calcula para minimizar el error asociado a PPR. Implementa la expresión

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1} \tag{7.1}$$

La tabla 7.1 y la figura 7.1 muestran la interfaz y el diagrama de bloques de este módulo respectivamente.

Nombre	Tamaño	Sentido	Comentario
ppr_data	16 bits	entrada	Viene de la FIFO *i_prediction2gain_p*.
ppr_read_fifo	1 bit	salida	Activa lectura de datos de la FIFO.
ppr_ready	1 bit	entrada	Enviado por PREDICTION cuando XPR está en la FIFO.
gain_data	16 bits	salida	Va a la FIFO *i_gain2correction*.
gain_write_fifo	1 bit	salida	Activa la escritura en la FIFO.
gain_ready	1 bit	salida	Enviado a CORRECTION cuando XPR está en la FIFO.
r	16 bits	entrada	Covarianza asociada a la medida, R.

Tabla 7.1: Interfaz del módulo GAIN

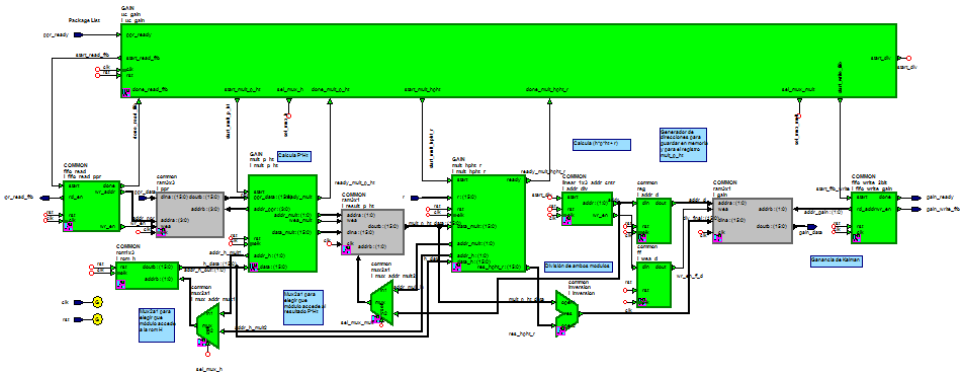


Figura 7.1: *Top-level* del módulo GAIN

7.1. Cálculo de la ganancia

El proceso del cálculo de la ganancia se realiza en varias fases:

1. Cuando se recibe la señal ppr_ready, que indica que PPR está lista, se procede a leerla desde la FIFO *i_prediction2gain_p* para almacenarla en la memoria local *i_ppr*.

2. Una vez guardada en la memoria *i_ppr*, se procede a hacer la primera multiplicación, cuyos factores son la matriz constante H^T y PPR, que acabamos de guardar en memoria.

3. A medida que salen los datos del módulo de multiplicación, se van almacenando en la memoria auxiliar *i_result_p_ht*, ya que este resultado será necesario en la siguiente multiplicación.

4. Al terminar la primera multiplicación empieza la segunda, que multiplica la matriz constante H por el resultado de la anterior multiplicación, y da como resultado un escalar.

5. Al resultado calculado en el punto anterior se le suma la covarianza asociada a la medida, es decir, el escalar R.

6. Por último, la matriz resultado de la primera multiplicación se divide entre el escalar calculado en el paso anterior.

7. El resultado de esta operación se almacena directamente en la memoria *i_gain* para que sea escrito posteriormente en la FIFO.

8. Cuando se hayan almacenado todos los datos en la memoria, se escriben en la FIFO *i_gain2correction* y se avisa al módulo CORRECTION de que la ganancia está lista activando la señal gain_ready.

En la tabla 7.2 se listan todos los módulos instanciados en GAIN junto con la función que desempeñan en el cálculo de la ganancia.

Instancia	Entidad	Función
i_addr_div	**linear_1x3_addr_cntr**	Genera las direcciones para la división de PH^T entre $HPH^T + R$
i_addr_d	**reg**	Retrasa un ciclo la dirección de escritura en la memoria *i_gain*
i_inversion	**inversion**	Realiza la división entre PH^T entre $HPH^T + R$
i_mult_p_ht	**mult_p_ht**	Multiplica P y H^T
i_mult_hpht_r	**mult_hpht_r**	Calcula $HPH^T + R$
i_mux_addr_mult1	**mux2a1**	Permite seleccionar diferentes orígenes para las direcciones de acceso a *i_rom_h*
i_mux_addr_mult2	**mux2a1**	Permite elegir si la dirección de lectura para la memoria *i_result_p_ht* viene desde *i_mult_hpht_r* o *i_addr_div*.
i_wea_d	**ff**	Retrasa un ciclo la señal de habilitación de escritura en la memoria *i_gain*.

Tabla 7.2: Elementos a nivel *top* del módulo GAIN

Almacenamiento de los datos

Los datos se almacenan en memorias RAM, y los datos constantes en memorias ROM. En la tabla 7.3 se detallan las memorias utilizadas en este módulo.

Unidad de control: módulo *i_uc_gain*

Este módulo es una instancia de la entidad **uc_gain**. Éste implementa una FSM que controla todas las operaciones que debe realizar este módulo.

Instancia	Tipo	Origen	Destino	Almacena
i_ppr	RAM 3x3	*i_prediction-2gain*	*i_mult_p_ht*	PPR
i_rom_h	ROM 3x1	-	*i_mult_p_ht*, *i_mult_hpht_r*	H
i_result_p_ht	RAM 3x1	*i_mult_p_ht*	*i_mult_hpht_r*	PH^T
i_gain	RAM 3x1	*i_mult_hpht_r*	*i_gain2-correction*	K

Tabla 7.3: Instancias de memorias usadas en el módulo GAIN

Para calcular la ganancia no hacen falta los estados de inicialización como ocurre en las unidades de control del módulo PREDICTION, ya que aquí no se lleva a cabo dicha fase.

Su diagrama de estados se puede observar en la figura 7.2. En la tabla 7.4 se detallan los estados de esta unidad de control.

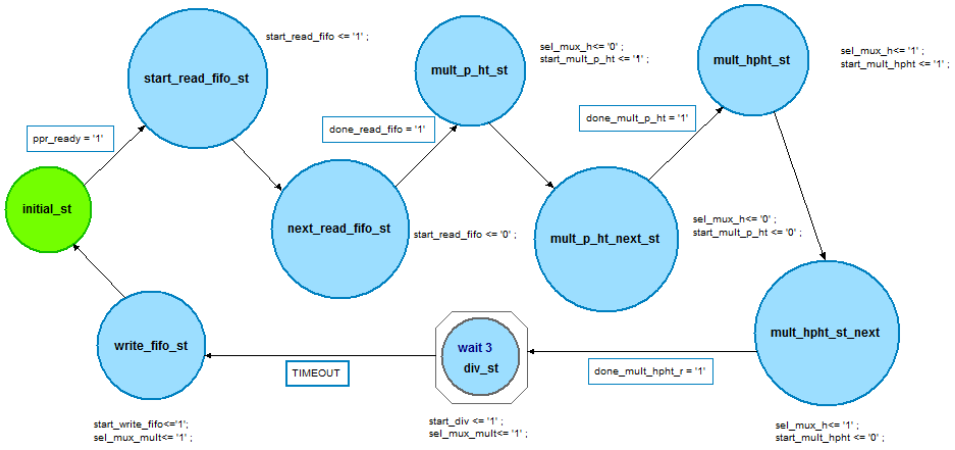


Figura 7.2: Unidad de control del módulo GAIN

Tabla 7.4: Estados de la unidad de control del módulo GAIN

Estado	Acción
initial_st	El sistema está a la espera de que PPR esté listo para iniciar otra iteración, hasta que la señal ppr_ready ← 1.

Continúa en la página siguiente

Unidad de control del módulo GAIN (continuación)

start_read_fifo_st	Inicia el módulo de lectura de FIFO *i_fifo_read_ppr* activando la señal start_read_fifo → 1.
next_read_fifo_st	Desactiva la señal de inicio de lectura de FIFO start_read_fifo → 0. En este estado se sigue leyendo de la FIFO hasta que se recibe el aviso de que se han leído todos los datos, cuando done_read_fifo ← 1.
mult_p_ht_st	Activa la señal de inicio de la primera multiplicación start_mult_p_ht → 1. Selecciona la primera entrada del multiplexor *i_mux_addr_mult1* (sel_mux_h → 0) para habilitar la lectura de H desde el módulo *i_mult_p_ht*.
mult_p_ht_next_st	Desactiva la señal de inicio start_mult_p_ht → 0. Mantiene seleccionada la primera entrada del multiplexor *i_mux_addr_mult1* (sel_mux_h → 0). En este estado se termina de realizar la primera multiplicación hasta que se avisa de que ha concluido, cuando done_mult_p_ht ← 1.
mult_hpht_st	Activa la señal de inicio de la segunda multiplicación start_mult_hpht → 1. Selecciona la segunda entrada del multiplexor *i_mux_addr_mult1* (sel_mux_h → 1) para habilitar la lectura de H desde el módulo *i_mult_hpht_r*. También selecciona la primera entrada del multiplexor *i_mux_addr_mult2* (sel_mux_mult → 0) para habilitar la lectura de *i_result_p_ht* por parte del mismo módulo de multiplicación.
mult_hpht_next_st	Desactiva la señal de inicio start_mult_hpht → 0. Mantiene seleccionada la segunda entrada del multiplexor *i_mux_addr_mult1* (sel_mux_h → 1) y la primera entrada del multiplexor *i_mux_addr_mult2* (sel_mux_mult → 0). En este estado se termina de realizar la segunda multiplicación hasta que se avisa de que ha concluido, cuando done_mult_hpht_r ← 1.

Continúa en la página siguiente

Unidad de control del módulo GAIN (continuación)

div_st	Durante tres ciclos habilita la división activando start_div → 1. Selecciona la segunda entrada del multiplexor *i_mux_addr_mult2* (sel_mux_mult → 1) para habilitar la lectura de *i_result_p_ht* por parte del módulo *i_inversion*.
write_fifo_st	Inicia el módulo de escritura en FIFO *i_fifo_write_gain* activando la señal start_fifo_write → 1.

Inicio de la ganancia

El módulo GAIN necesita PPR para realizar sus cálculos. Para ello, el sistema espera la señal ppr_ready desde el módulo PREDICTION para comenzar. Esta señal va dirigida hacia la unidad de control para que el sistema comience a realizar los cálculos.

Lectura de la FIFO

Cuando el módulo GAIN recibe la señal ppr_ready que indica que PPR está lista, se procede a leerla desde la FIFO *i_prediction2gain_p* para almacenarla en la memoria local *i_ppr*. De esta tarea se encarga el módulo *i_fifo_read_ppr*, que es una instancia del módulo **fifo_read** explicado en la sección 4.5. En este caso el valor de n es 9, puesto que PPR tiene 9 elementos.

Primera multiplicación: módulo *i_mult_p_ht*

Este módulo realiza la multiplicación de PPR por la constante H^T, representando la operación $P_k^- H^T$. El diseño de este módulo es similar al módulo **mult_a_x** descrito en la sección 6.2, pero particularizado para las matrices con las que se opera en este caso. En la figura 7.3 se puede observar su diagrama de bloques. La tabla 7.6 recoge las particularidades y la tabla 7.5 recoge la interfaz de este módulo.

Una vez realizados los cálculos, guarda los resultados en la memoria *i_result_p_ht* (externa a este módulo) y activa la señal ready_mult para indicar que ha finalizado.

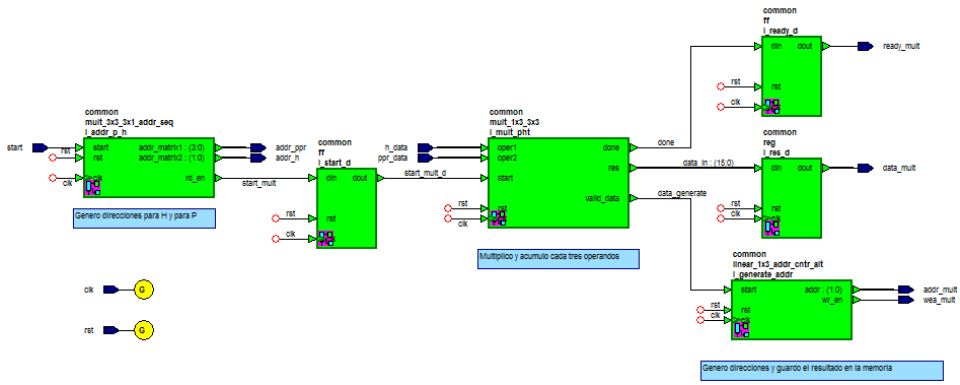


Figura 7.3: Diagrama de bloques del módulo *i_mult_p_ht*

Nombre	Tamaño	Sentido	Comentario
h_data	16 bits	entrada	Datos leídos de *i_rom_h*.
ppr_data	16 bits	entrada	Datos leídos de *i_ppr*.
start	1 bit	entrada	Inicia el módulo.
addr_ppr	4 bits	salida	Dirección para leer desde la RAM *i_ppr*.
addr_h	2 bits	salida	Dirección para leer desde la ROM *i_rom_h*.
addr_mult	2 bits	salida	Dirección para escribir en la RAM *i_result_p_ht*.
data_mult	16 bits	salida	Datos escritos en la RAM *i_result_p_ht*.
wea_mult	1 bit	salida	Habilita la escritura en *i_result_p_ht*.
ready_mult	1 bit	salida	Señala la finalización del cálculo de este módulo.

Tabla 7.5: Interfaz del módulo *i_mult_p_ht*

Instancia	Entidad	Función
i_addr_p_h	**mult_3x3_3x1_addr_seq**	Genera las direcciones de lectura de los datos.
i_mult_pht	**mult_1x3_3x3**	Multiplica P por H^T.
i_generate_addr	**linear_1x3_addr_cntr**	Genera las direcciones de escritura de los datos.

Tabla 7.6: Elementos particularizados para el módulo *i_mult_p_ht*

Segunda multiplicación: módulo *i_mult_hpht_r*

Este módulo realiza la multiplicación de la constante H y el resultado de la multiplicación anterior $P_k^- H^T$, y además suma R, es decir, la covarianza asociada a la medida. En conjunto, representa la operación $HP_k^- H^T + R$. Una vez realizados los cálculos, el sistema activa la señal ready para avisar de que ha acabado.

El diagrama de bloques de este módulo se puede ver en la figura 7.4, y en las tablas 7.7 y 7.8 se detallan la interfaz y la descripción de los módulos que lo componen respectivamente.

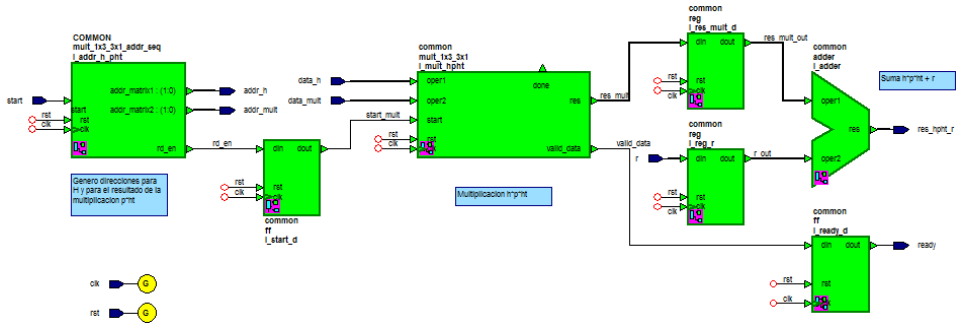


Figura 7.4: Diagrama de bloques del módulo *i_mult_hpht_r*

Nombre	Tamaño	Sentido	Comentario
r	16 bits	entrada	Datos de la covarianza de la medida.
start	1 bit	entrada	Inicia el módulo.
data_mult	16 bits	entrada	Datos leídos de *i_result_p_ht*.
data_h	16 bits	entrada	Datos leídos de *i_rom_h*.
addr_mult	2 bits	salida	Dirección para leer desde la RAM *i_result_p_ht*.
addr_h	2 bits	salida	Dirección para leer desde la ROM *i_rom_h*.
res_hpht_r	16 bits	salida	Resultado de la operación.
ready	1 bit	salida	Señala la finalización del cálculo de este módulo.

Tabla 7.7: Interfaz del módulo *i_mult_hpht_r*

Instancia	Entidad	Función
i_addr_h_pht	**mult_1x3_3x1_addr_seq**	Genera las direcciones de lectura de los datos.
i_adder	**adder**	Realiza la suma final.
i_mult_hpht	**mult_1x3_3x1**	Multiplica H por PH^T.
i_ready_d	**ff**	Retrasa un ciclo la señal de ready debido a que se tarda un ciclo en guardar el último elemento en memoria.
i_reg_r	**reg**	Registro que guarda R (covarianza de la medida).
i_start_d	**ff**	Retrasa un ciclo el inicio de la multiplicación ya que es lo que tardan en llegar los datos de memoria.
i_res_mult_d	**reg**	Retrasa durante un ciclo el primer operando de la suma.

Tabla 7.8: Elementos del módulo *i_mult_hpht_r*

División: módulo *i_inversion*

En el módulo GAIN se debe efectuar una división de una matriz por otra. Para el caso particular que se ha dado en este proyecto, la matriz divisor tiene unas dimensiones de 1x1, por tanto es un escalar, y la matriz dividendo es de 3x1. En esta operación intervienen dos módulos que se detallan en la tabla 7.9.

Las direcciones generadas para leer desde la memoria *i_result_p_ht* se aprovechan como direcciones de escritura en la memoria *i_gain* y se sincronizan con el dato mediante un registro. También se genera una señal para habilitar la escritura en *i_gain*, que se sincroniza con el resto mediante un *flip-flop*.

Instancia	Entidad	Función
i_addr_div	**linear_1x3_addr_cntr**	Genera direcciones para leer el contenido de la RAM *i_result_p_ht*.
i_inversion	**inversion**	Realiza la división.

Tabla 7.9: Elementos de la división

Escritura en la FIFO

Tras almacenar la ganancia completa en la memoria *i_gain*, se puede proceder a su escritura en la FIFO *i_gain2correction*. Una vez completada la escritura, se avisa al módulo CORRECTION de que la ganancia ya está disponible, activando la señal gain_ready. El módulo que se encarga de estas tareas es *i_fifo_write_gain*, instancia del módulo **fifo_write** explicado en la sección 4.5. En este caso el valor de n es 3, puesto que el tamaño de la matriz de ganancia es de 3 elementos.

Capítulo 8

Módulo Correction

Este módulo implementa las ecuaciones del filtro correspondientes al cálculo de XPO y PPO de la etapa de corrección:

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{8.1}$$

$$P_k = (I - K_k H)P_k^- \tag{8.2}$$

En la tabla 8.1 se detalla la interfaz de este módulo y en la figura 8.1 se muestra el diagrama de bloques del mismo.

Se realiza en las siguientes fases:

1. Espera hasta haber recibido la notificación de que están disponibles todos los datos que necesita.
2. Inicia las lecturas de las FIFOs por orden y espera a su conclusión.
3. Activa el módulo de corrección XPO. Cuando éste termina, manda escribir el dato sobre la FIFO.
4. Activa el módulo de corrección PPO. Cuando éste termina, manda escribir el dato sobre la FIFO.

Almacenamiento de datos

Los datos se almacenan en memorias RAM y las constantes utilizados en este módulo se guardan en memorias ROM a nivel top. En este módulo las memorias RAM están encapsuladas en diferentes submódulos. En la tabla 8.2 se detallan todas las memorias instanciadas en este módulo, comentando además para cada una el módulo que la encapsula.

Nombre	Tamaño	Sentido	Comentario
gain_data	16 bits	entrada	Viene de la FIFO *i_gain2correction*.
gain_read_fifo	1 bit	salida	Activa lectura de datos de la FIFO.
gain_ready	1 bit	entrada	Enviado por Gain cuando K está listo.
input_data	16 bits	entrada	Medición real necesaria para corregir las predicciones y mejorar la calidad de la señal filtrada.
input_data_ready	1 bit	entrada	Avisa de que se puede leer una nueva medición.
xpo_data	16 bits	salida	Va a la FIFO *i_correction2prediction_x*.
xpo_write_fifo	1 bit	salida	Activa la escritura en la FIFO y avisa al exterior de que el dato es válido.
xpo_ready	1 bit	salida	Enviado a Prediction cuando XPO está en la FIFO.
xpr_data	16 bits	entrada	Viene de la FIFO *i_prediction2correction_x*.
xpr_read_fifo	1 bit	salida	Activa la lectura de la FIFO.
xpr_ready	1 bit	entrada	Enviado por Prediction cuando XPR está en la FIFO.
ppo_data	16 bits	salida	Va a la FIFO *i_correction2prediction_p*.
ppo_write_fifo	1 bit	salida	Activa la escritura en la FIFO.
ppo_ready	1 bit	salida	Enviado a Prediction cuando PPO está en la FIFO.
ppr_data	16 bits	entrada	Viene de la FIFO *i_prediction2correction_p*.
ppr_read_fifo	1 bit	salida	Activa la lectura de la FIFO.
ppr_ready	1 bit	entrada	Enviado por Prediction cuando PPR está en la FIFO.

Tabla 8.1: Interfaz del módulo Correction

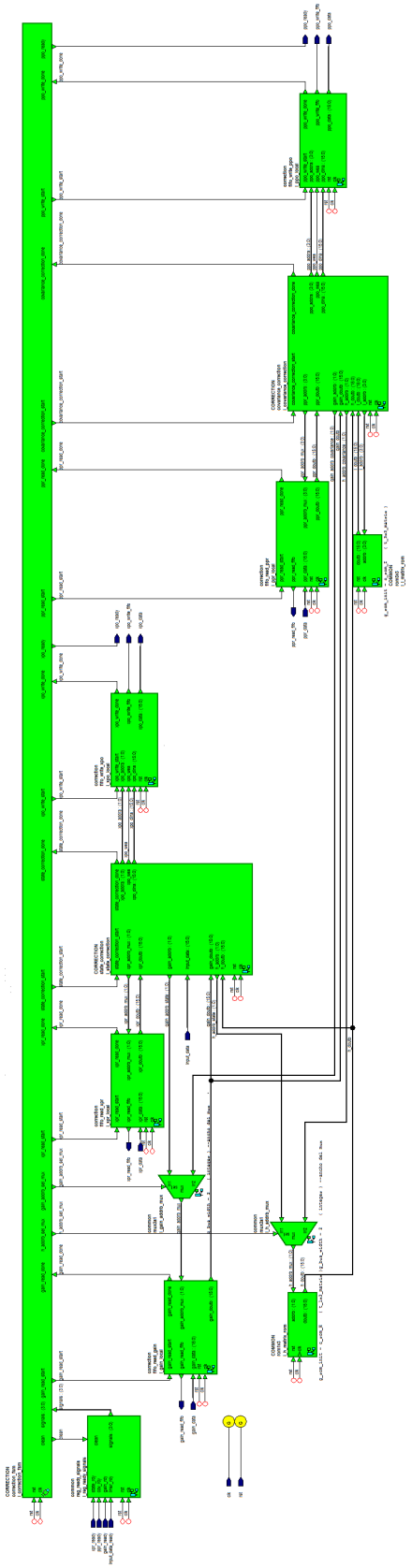


Figura 8.1: *Top-level* del módulo CORRECTION

Módulo	Tipo	Origen	Destino	Contiene
		Correction		
i_h_matrix_rom	ROM 1x3	-	*i_state_correction*, *i_covariance_correction*	H
i_i_matrix_rom	ROM 3x3	-	*i_covariance_correction*	I
		i_gain_local		
i_gain_ram	RAM 3x1	*i_gain2correction*	*i_state_correction*, *i_covariance_correction*	K
		i_xpr_local		
i_xpr_ram	RAM 3x1	*i_prediction2correction_x*	*i_state_correction*	XPR
		i_ppr_local		
i_ppr_ram	RAM 3x3	*i_prediction2correction_p*	*i_covariance_correction*	PPR
		i_state_correction		
i_h_mult_xpr_res_ram	RAM reg	*i_h_mult_xpr*	*i_z_sub_hxpr*	$H\hat{x}^-$
i_z_sub_hxpr_res_ram	RAM reg	*i_z_sub_hxpr*	*i_k_mult_zhxpr*	$z - H\hat{x}^-$
i_k_mult_zhxpr_res_ram	RAM 3x1	*i_k_mult_zhxpr*	*i_xpr_plus_kzhxpr*	$K(z - H\hat{x}^-)$
		i_covariance_correction		
i_k_mult_h_res_ram	RAM 3x3	*i_k_mult_h*	*i_i_sub_kh*	KH
i_i_sub_kh_res_ram	RAM 3x3	*i_i_sub_kh*	*i_ikh_mult_ppr*	$I - KH$
		i_xpo_local		
i_xpo_ram	RAM 3x1	*i_state_correction*	*i_correction2prediction_x*	XPO
		i_ppo_local		
i_ppo_ram	RAM 3x3	*i_covariance_correction*	*i_correction2prediction_p*	PPO

Tabla 8.2: Memorias utilizadas en el módulo Correction

Unidad de control maestra: *i_correction_fsm*

CORRECTION tiene dos niveles de control: una unidad de control maestra y dos esclavas. La unidad maestra lleva el control general y activa cada una de las unidades de control esclavas cuando sea su turno realizar las operaciones.

El módulo *i_correction_fsm* es una instancia de la entidad **correction_fsm**. Esta entidad implementa la FSM maestra que controla la secuencia de operaciones que debe realizar este módulo. La figura 8.2 muestra el diagrama de transición de estados y la tabla 8.3 describe en detalle la función de cada estado.

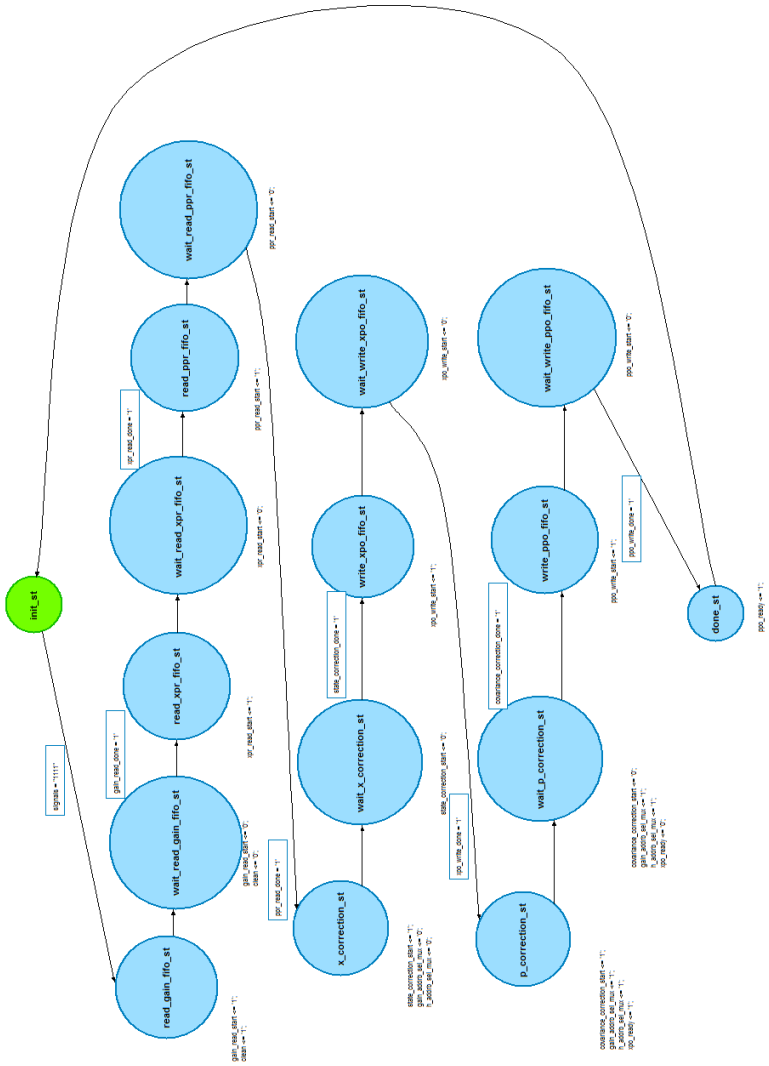


Figura 8.2: Diagrama de estados del módulo *i_correction_fsm*

Tabla 8.3: Estados de la unidad de control maestra del módulo CORRECTION

Estado	Acción
init_st	El sistema permanece a la espera de las señales ready (signals ← "1111")
read_gain_fifo_st	Limpia *i_reg_ready_signals* activando clean → 1 e inicia la lectura de la ganancia de la FIFO poniendo gain_read_start → 1.
wait_read_gain_fifo_st	Desactiva la señal de inicio gain_read_start → 0. Se sigue leyendo de la FIFO hasta la recepción de la señal gain_read_done ← 1.
read_xpr_fifo_st	Inicia la lectura de XPR de la FIFO activando xpr_read_start → 1.
wait_read_xpr_fifo_st	Desactiva la señal de inicio xpr_read_start → 0. Se sigue leyendo de la FIFO hasta la recepción de la señal xpr_read_done ← 1.
read_ppr_fifo_st	Inicia la lectura de PPR de la FIFO activando ppr_read_start → 1.
wait_read_ppr_fifo_st	Desactiva la señal de inicio ppr_read_start → 0. Se sigue leyendo de la FIFO hasta la recepción de la señal ppr_read_done ← 1.
x_correction_st	Selecciona la primera entrada de *i_gain_addrb_mux* y *i_h_addrb_mux* para que sea *i_state_correction* quien lea las memorias *i_gain_ram* y *i_h_matrix_rom*. Inicia dicho módulo poniendo state_correction_start → 1.
wait_x_correction_st	Desactiva la señal state_correction_start → 0. Mantiene seleccionadas las entradas de los multiplexores. Espera a que el módulo *i_state_correction* termine los cálculos, hasta que state_correction_done ← 1.
write_xpo_fifo_st	Inicia la escritura de XPO en la FIFO activando xpo_write_start → 1.
wait_write_xpo_fifo_st	Desactiva la señal xpo_write_start → 0. Se sigue escribiendo en la FIFO hasta que llega el aviso de conclusión, cuando xpo_write_done ← 1.

Continúa en la página siguiente

Unidad de control maestra del módulo CORRECTION (continuación)

p_correction_st	Selecciona la segunda entrada de *i_gain_addrb_mux* y *i_h_addrb_mux* para que sea *i_covariance_correction* quien lea las memorias *i_gain_ram* y *i_h_matrix_rom*. Inicia dicho módulo poniendo covariance_correction_start → 1. Notifica que XPO está listo activando xpo_ready → 1.
wait_p_correction_st	Desactiva las señales xpo_ready → 0 y covariance_correction_start → 0. Espera a que el módulo *i_covariance_correction* termine los cálculos, hasta que covariance_correction_done ← 1.
write_ppo_fifo_st	Inicia la escritura de PPO en la FIFO activando ppo_write_start → 1.
wait_write_ppo_fifo_st	Desactiva la señal ppo_write_start → 0. Se sigue escribiendo en la FIFO hasta que llega el aviso de conclusión, cuando ppo_write_done ← 1.
done_st	Notifica a PREDICTION que PPO está en la FIFO activando ppo_ready → 1.

Espera de las señales *ready*

El módulo CORRECTION necesita tres datos para poder comenzar sus operaciones (K, XPO y PPR) y la señal de que la medida real z está lista.

Debido a que el sensor sólo proporciona la medida z cada 1-5 minutos, es necesario esperar a que éste avise de la llegada de dicha medida. Para ello existe la señal de entrada input_data_ready, que recibe desde el *top-level* la entrada imsr_ready. El módulo CORRECTION se queda inactivo hasta que llega una nueva z.

El módulo *i_reg_ready_signals* es el encargado de registrar la recepción de las señales *ready* que envían los módulos PREDICTION y GAIN, así como de la señal input_data_ready. Una vez registradas las cuatro señales este módulo avisa a la unidad de control principal *i_correction_fsm* para que el comience el flujo de cálculos.

Lectura de las FIFOs

Cuando se reciben todas las señales de *ready*, la unidad de control principal activa los módulos de lectura de las FIFOs para almacenar los datos entrantes.

Como el módulo CORRECTION tiene que leer datos desde tres FIFOs, se han utilizado para esta tarea tres módulos propios. Cada uno de ellos encapsula una instancia de un módulo **fifo_read** explicado en la sección 4.5 y una instancia de memoria RAM del tamaño más adecuado al dato que va a almacenar. La tabla 8.4 recoge los detalles de cada uno de ellos. En la figura 8.3 se puede observar el diagrama de bloques y en la tabla 8.5 se detalla la interfaz del primero, puesto que los otros dos son similares.

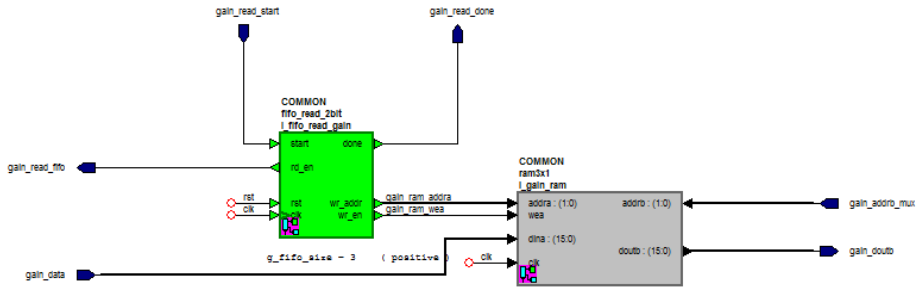


Figura 8.3: Diagrama de bloques del módulo *i_gain_local*

Instancia	FIFO origen	Instancia **read_fifo**	Instancia RAM
i_gain_local	*i_gain2correction*	*i_fifo_read_gain*	*i_gain_ram*
i_xpr_local	*i_prediction2correction_x*	*i_fifo_read_xpr*	*i_xpr_ram*
i_ppr_local	*i_prediction2correction_p*	*i_fifo_read_ppr*	*i_ppr_ram*

Tabla 8.4: Módulos lectores de FIFO usados en CORRECTION

8.1. Cálculo de XPO: módulo *i_state_correction*

Implementa la expresión $\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$, que corrige el estado estimado calculado en la etapa de predicción utilizando para ello la ganancia de Kalman y la medida real. El resultado de esta operación es XPO. Este proceso de realiza en las siguientes fases:

1. El módulo espera recibir la señal state_correction_start desde la unidad de control principal del módulo CORRECTION (*i_correction_fsm*) para poder empezar. Ésta indica que los datos que va a utilizar están disponibles en las RAMs correspondientes: el estado estará almacenado en *i_xpr_ram* y la ganancia en *i_gain_ram*. También indica que la entrada z está disponible.

Puerto	Tamaño	Sentido	Comentario
gain_read_start	1 bit	entrada	Al activarse comienza el proceso de lectura.
gain_read_fifo	1 bit	salida	Habilita la lectura de datos de la FIFO.
gain_data	16 bits	entrada	Datos leídos de la FIFO *i_gain2correction*.
gain_read_done	1 bit	salida	Señala la finalización de la lectura de la FIFO.
gain_addrb_mux	2 bits	entrada	Dirección para lectura de la memoria desde el exterior.
gain_doutb	16 bits	salida	Salida de datos hacia el exterior.

Tabla 8.5: Interfaz del módulo *i_gain_local*

2. La unidad de control secundaria *i_state_correction_fsm* activa el módulo *i_h_mult_xpr* para realizar la primera operación, que consiste en multiplicar la matriz H por XPR. El resultado se guarda en la memoria intermedia *i_h_mult_xpr_res_ram*.

3. Al finalizar la primera operación, la unidad de control activa el módulo *i_z_sub_hxpr*. Éste resta de la medida real z el resultado obtenido en la operación anterior y guarda el nuevo resultado en la memoria *i_z_sub_hxpr_res_ram*.

4. Cuando finaliza la segunda operación la unidad de control activa el módulo *i_k_mult_zhxpr* que multiplica la ganancia de Kalman por el resultado de la operación anterior. El resultado de la operación se guarda en la memoria *i_k_mult_zhxpr_res_ram*.

5. La unidad de control selecciona la entrada de *i_xpr_addrb_mux* para que el módulo *i_xpr_plus_kzhxpr* pueda leer desde la memoria *i_xpr_ram* y comienza la última operación, es decir, la suma de XPR al resultado obtenido en la operación anterior. El resultado se guarda a la memoria externa a este módulo *i_xpo_ram*.

6. Por último, se avisa a la unidad de control principal de que el módulo ha terminado los cálculos.

En la figura 8.4 se puede observar el diagrama de bloques de este módulo.

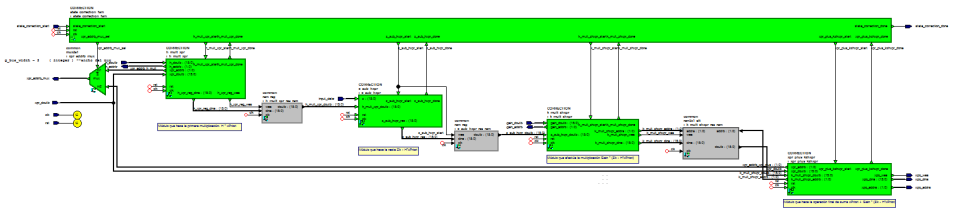


Figura 8.4: *Top-level* del módulo *i_state_correction*

Unidad de control esclava: módulo *i_state_correction_fsm*

El módulo *i_state_correction_fsm* es una instancia de la entidad **state_correction_fsm**. Esta entidad implementa una FSM que controla la secuencia de operaciones que debe realizar este módulo. La figura 8.5 muestra el diagrama de transición de estados y la tabla 8.6 describe en detalle la función de cada estado.

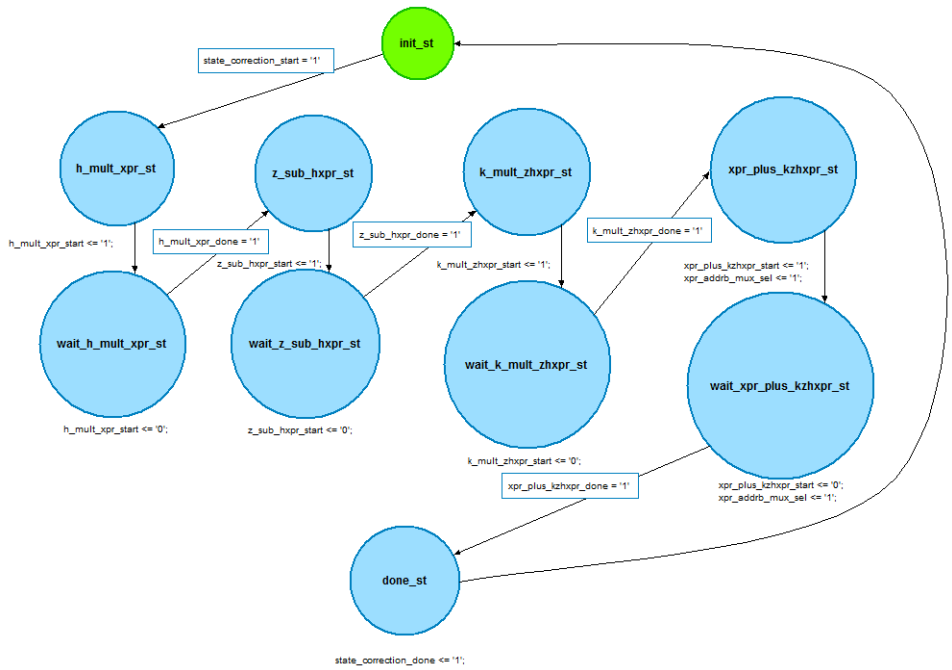


Figura 8.5: Diagrama de estados del módulo *i_state_correction_fsm*

Estado	Acción
init_st	Espera que la señal state_correction_start ← 1 para empezar los cálculos.
h_mult_xpr_st	Activa la señal de inicio de la primera multiplicación h_mult_xpr_start → 1.
wait_h_mult_xpr_st	Desactiva la señal de inicio h_mult_xpr_start → 0. En este estado se termina de realizar la primera multiplicación, hasta que h_mult_xpr_done ← 1.
z_sub_hxpr_st	Activa la señal de inicio del módulo restador z_sub_hxpr_start → 1.
wait_z_sub_hxpr_st	Desactiva la señal z_sub_hxpr_start → 0. En este estado se termina de realizar la resta, hasta que z_sub_hxpr_done ← 1.
k_mult_zhxpr_st	Activa la señal de inicio de la segunda multiplicación k_mult_zhxpr_start → 1.
wait_k_mult_zhxpr_st	Desactiva la señal de inicio k_mult_zhxpr_start → 0. En este estado se termina de realizar la segunda multiplicación, hasta que k_mult_zhxpr_done ← 1.
xpr_plus_kzhxpr_st	Activa la señal de inicio de la suma xpr_plus_kzhxpr_start → 1. Selecciona la entrada del multiplexor *i_xpr_addrb_mux* para que *i_xpr_plus_kzhxpr* pueda leer *i_xpr_ram*, poniendo xpr_addrb_mux_sel → 1.
wait_xpr_plus_kzhxpr_st	Desactiva la señal de inicio xpr_plus_kzhxpr_start → 0. Mantiene seleccionada la entrada del multiplexor del estado anterior. En este estado se termina de realizar la suma, hasta que xpr_plus_kzhxpr_done ← 0.
done_st	Activa la señal state_correction_done → 1 para indicar a la unidad de control principal la conclusión de los cálculos.

Tabla 8.6: Estados de la unidad de control del módulo *i_state_correction*

Primera multiplicación: módulo *i_h_mult_xpr*

Este módulo realiza la multiplicación $H\hat{x}_k^-$. En la figura 8.6 se puede observar el diagrama de bloques de este módulo, y en la tabla 8.7 se detallan los módulos que lo forman.

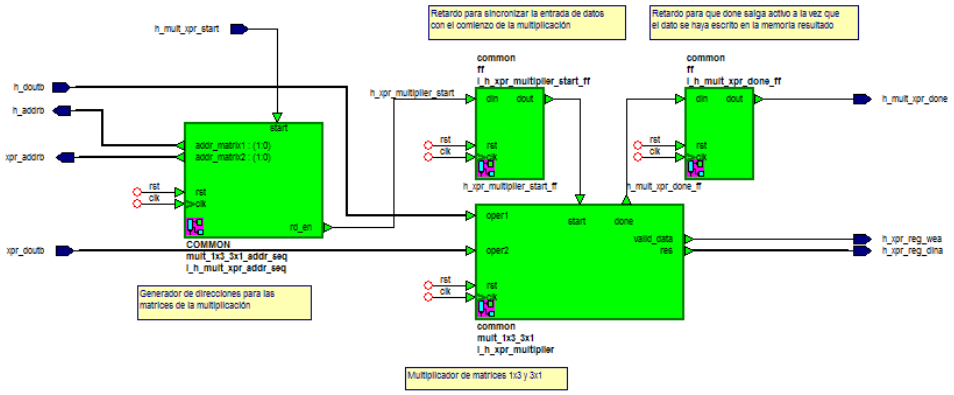


Figura 8.6: Diagrama de bloques del módulo *i_h_mult_xpr*

Instancia	Entidad	Función
i_h_mult_xpr_addr_seq	**mult_1x3_3x1_addr_seq**	Genera las direcciones de lectura de los datos.
i_h_xpr_multiplier_start_ff	**ff**	Sincroniza el inicio de la multiplicación con la llegada de los datos de la memoria.
i_h_xpr_multiplier	**mult_1x3_3x1**	Realiza la multiplicación.
i_h_xpr_multiplier	**ff**	Sincroniza la señal done con el último dato guardado en memoria.

Tabla 8.7: Elementos del módulo *i_h_mult_xpr*

Resta: módulo *i_z_sub_hxpr*

Este módulo realiza la resta $z - H\hat{x}_k^-$. En la figura 8.7 se puede observar el diagrama de bloques de este módulo, y en la tabla 8.8 se detallan los módulos que lo forman.

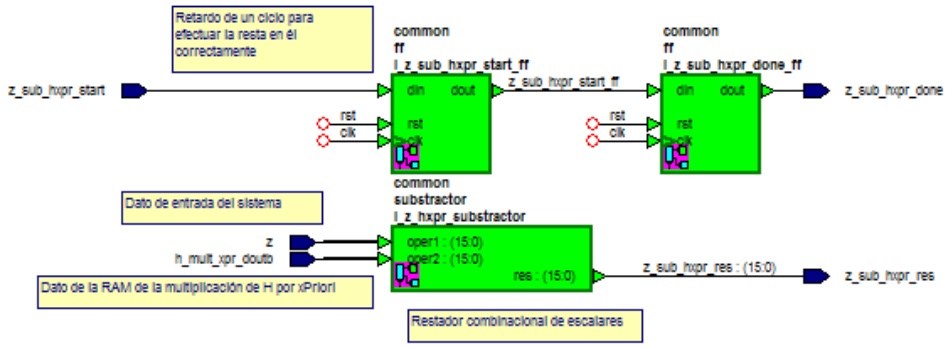


Figura 8.7: Diagrama de bloques del módulo *i_z_sub_hxpr*

Instancia	Entidad	Función
i_z_sub_hxpr_start_ff	**ff**	Retarda la señal de start para usarla como señal de done.
i_z_hxpr_subtractor	**substractor**	Resta dos escalares.
i_z_sub_hxpr_done_ff	**ff**	Sincroniza la señal de done con la escritura del dato en memoria.

Tabla 8.8: Elementos del módulo *i_z_sub_hxpr*

Segunda multiplicación: módulo *i_k_mult_zhxpr*

Este módulo realiza la multiplicación $K(z - H\hat{x}_k^-)$. El diseño de este módulo es similar al explicado en la sección 8.1, pero particularizado para las matrices con las que se opera en este caso.

Suma: módulo *i_xpr_plus_kzhxpr*

Este módulo realiza la suma $\hat{x}_k^- + K(z - H\hat{x}_k^-)$. En la figura 8.8 se puede observar el diagrama de bloques de este módulo, y en la tabla 8.9 se detallan los módulos que lo forman.

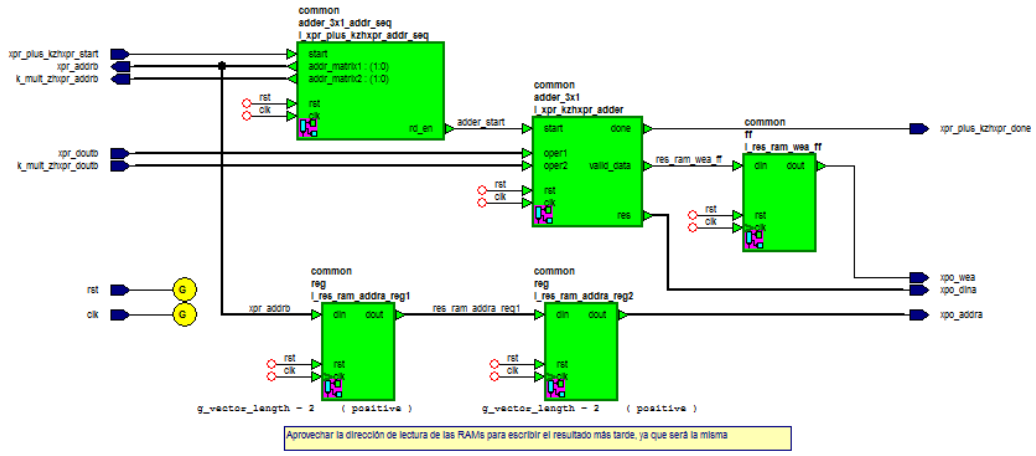


Figura 8.8: Diagrama de bloques del módulo *i_xpr_plus_kzhxpr*

Instancia	Entidad	Función
i_xpr_plus_kzhxpr_addr_seq	**addr_3x1_addr_seq**	Genera las direcciones de lectura de los datos.
i_xpr_kzhxpr_adder	**addr_3x1**	Realiza la suma.
i_res_ram_wea_ff	**linear_1x3_addr_cntr**	Sincroniza la habilitación de escritura de la memoria de resultado con la dirección de escritura.
i_res_ram_addra_reg1 y *i_res_ram_addra_reg2*	**reg**	Retardan la dirección de lectura para reutilizarla como dirección de escritura del resultado.

Tabla 8.9: Elementos del módulo *i_xpr_plus_kzhxpr*

8.2. Cálculo de PPO: módulo *i_ covariance_ correction*

Implementa la expresión $P_k = (I - K_k H)P_k^-$, que calcula PPO utilizando para ello PPR calculada por el módulo PREDICTION, la matriz *Identidad*, y K. Este proceso se realiza en varias fases:

1. El módulo espera la señal covariance_correction_start desde unidad de control principal del módulo CORRECTION (*i_ correction_ fsm*) para poder empezar. Esta le indica que los datos que va a utilizar están disponibles en las RAMs correspondientes: PPR estará almacenado en *i_ ppr_ ram* y K en *i_ gain_ ram*.

2. La unidad de control secundaria *i_ covariance_ correction* activa el módulo *i_ k_ mult_ h* para realizar la primera operación, que consiste en multiplicar K por H. El resultado se guarda en la memoria intermedia *i_ k_ mult_ h_ res_ ram*.

3. Al finalizar la primera operación, la unidad de control activa el módulo *i_ i_ sub_ kh* para realizar la segunda operación, que consiste en restar de la matriz I el resultado obtenido en la operación anterior. El resultado se guarda en la memoria intermedia *i_ i_ sub_ kh_ res_ ram*.

4. Al finalizar la segunda operación, la unidad de control activa el módulo *i_ ikh_ mult_ ppr* para realizar la tercera operación, que consiste en multiplicar el resultado obtenido en la operación anterior por PPR. El resultado se guarda en la memoria *i_ ppo_ ram*, externa a este módulo.

5. Por último, se avisa a la unidad de control principal que el módulo ha terminado los cálculos.

En la figura 8.9 se puede observar el diagrama de bloques de este módulo.

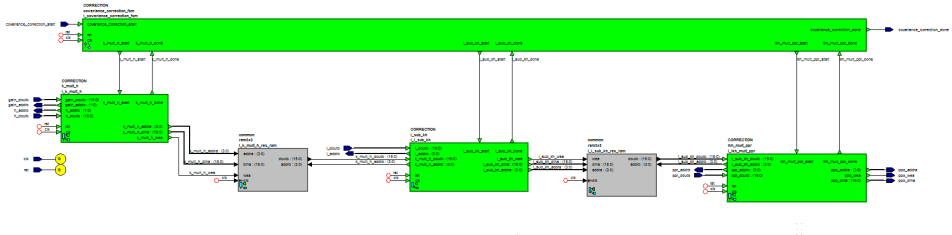


Figura 8.9: *Top-level* del módulo *i_ covariance_ correction*

Unidad de control esclava: módulo *i_covariance_correction_fsm*

El módulo *i_covariance_correction_fsm* es una instancia de la entidad **covariance_correction_fsm**. Esta entidad implementa una FSM que controla la secuencia de operaciones que debe realizar este módulo. La figura 8.10 muestra el diagrama de transición de estados, y en la tabla 8.10 se describe con detalle la función de cada estado.

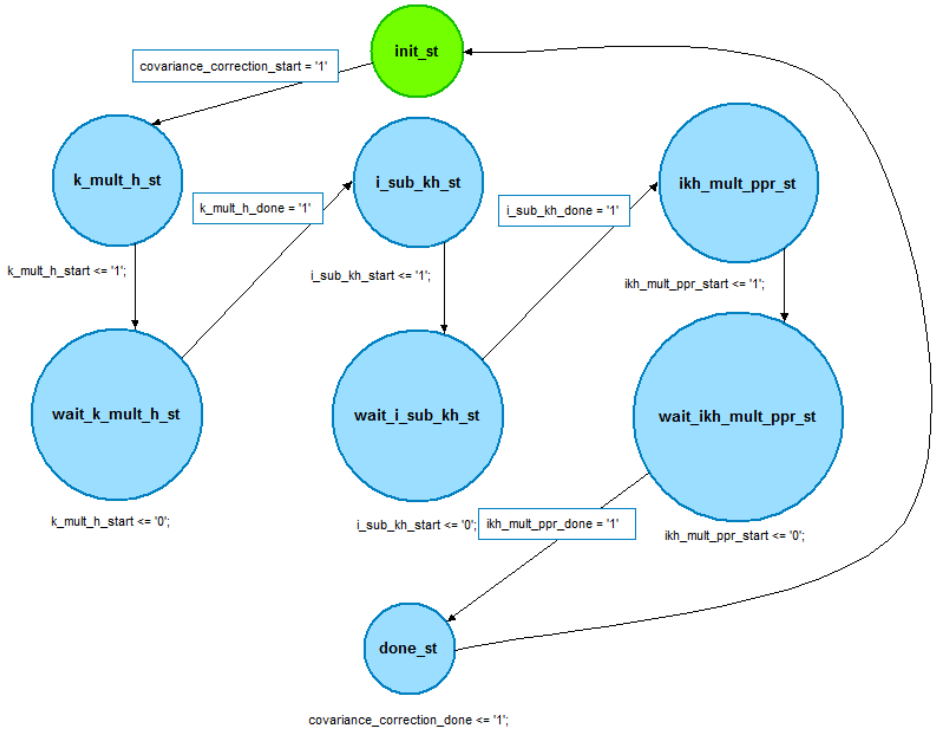


Figura 8.10: Diagrama de estados de *i_covariance_correction_fsm*

Tabla 8.10: Estados de la unidad de control de *i_covariance_correction*

Estado	Acción
init_st	Espera que la señal covariance_correction_start ← 1 para empezar los cálculos.
k_mult_h_st	Activa la señal de inicio de la primera multiplicación k_mult_h_start → 1.

Continúa en la página siguiente

Unidad de control de *i_covariance_correction* (cont.)

wait_k_mult_h_st	Desactiva la señal de inicio k_mult_h_start → 0. En este estado se termina de realizar la primera multiplicación, hasta que k_mult_h_done ← 1.
i_sub_kh_st	Activa la señal de inicio del módulo restador i_sub_kh_start → 1.
wait_i_sub_kh_st	Desactiva la señal i_sub_kh_start → 0. En este estado se termina de realizar la resta, hasta que i_sub_kh_done ← 1.
ikh_mult_ppr_st	Activa la señal de inicio de la segunda multiplicación ikh_mult_ppr_start → 1.
wait_ikh_mult_ppr_st	Desactiva la señal ikh_mult_ppr_start → 0. En este estado se termina de realizar la segunda multiplicación, hasta que ikh_mult_ppr_done ← 1.
done_st	Activa la señal covariance_correction_done → 1 para indicar a la unidad de control principal la conclusión de los cálculos.

Primera multiplicación: módulo *i_k_mult_h*

Este módulo realiza la primera operación, que corresponde a la multiplicación KH. El diseño de este módulo es similar al módulo *i_h_mult_xpr*, explicado en la sección 8.1, pero con el módulo generador de direcciones y el módulo multiplicador particularizados para las dimensiones de las matrices con las que tiene que operar.

Resta: módulo *i_i_sub_kh*

Este módulo realiza la segunda operación, que corresponde a la resta $I - KH$. El diagrama de bloques de este módulo se puede observar en la figura 8.11. En la tabla 8.11 se detallan los módulos que lo componen.

Segunda multiplicación: módulo *i_ikh_mult_ppr*

Este módulo realiza la tercera operación que corresponde a la multiplicación $(I-KH)P^{-}$. En la figura 8.12 se puede observar el diagrama de bloques de este módulo y en la tabla 8.12 se detallan los módulos que lo componen.

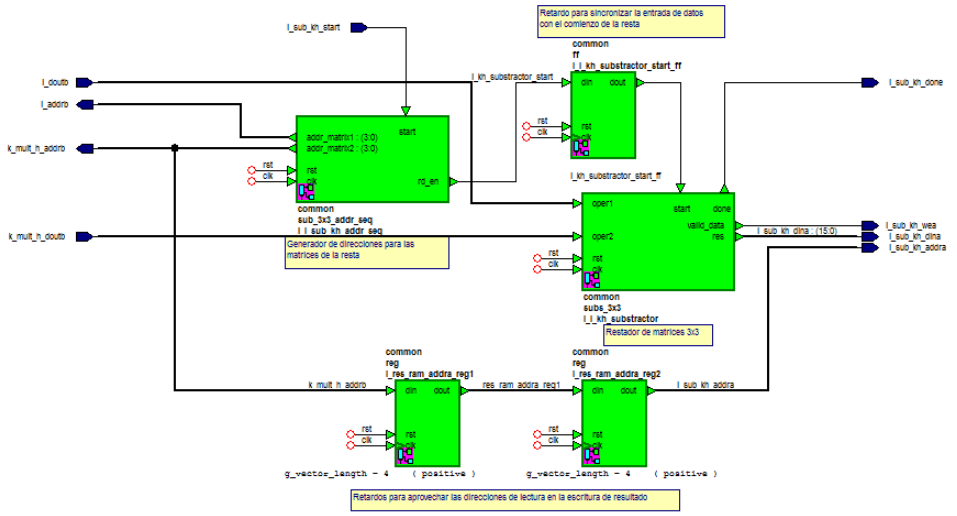


Figura 8.11: Diagrama de bloques del módulo *i_i_sub_kh*

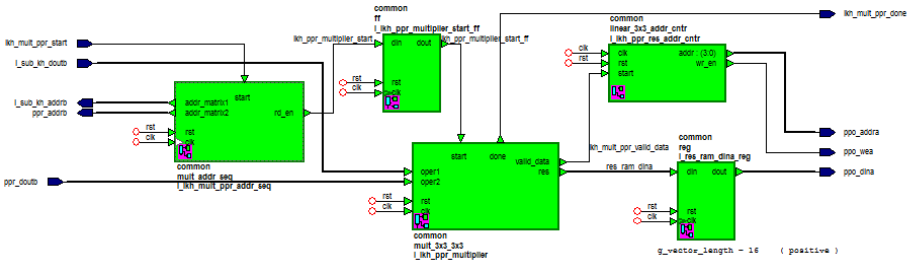


Figura 8.12: Diagrama de bloques del módulo *i_ikh_mult_ppr*

Escritura en las FIFOs

Cuando terminan todos los cálculos y los resultados quedan almacenados en las memorias, la unidad de control principal activa los módulos de escritura en las FIFOs para enviarlos a las FIFOs.

Igual que para la lectura, el módulo CORRECTION tiene para esta tarea módulos propios. En este caso en vez de **fifo_read** tienen una instancia del módulo **fifo_write** explicado en la sección 4.5. La tabla 8.13 recoge los detalles de estos módulos. En la figura 8.13 se puede observar el diagrama de bloques y en la tabla 8.14 se detalla la interfaz del primero, puesto que el otro es similar.

Instancia	Entidad	Función
i_i_sub_kh_addr_seq	**sub_3x3_addr_seq**	Genera las direcciones de lectura de datos.
i_i_kh_substractor_start_ff	**ff**	Sincroniza el inicio de la resta con la llegada de los datos de memoria.
i_i_kh_subtractor	**subs_3x3**	Realiza la resta.
i_res_ram_addra_reg1 y *i_res_ram_addra_reg2*	**reg**	Retardan la dirección de lectura para reutilizarls como dirección de escritura del resultado.

Tabla 8.11: Elementos del módulo *i_i_sub_kh*

Instancia	Entidad	Función
i_ikh_mult_ppr_addr_seq	**mult_addr_seq**	Genera las direcciones de lectura de datos.
i_ikh_ppr_multiplier_start_ff	**ff**	Sincroniza el inicio de la multiplicación con la llegada de los datos de memoria.
i_ikh_ppr_multiplier	**mult_3x3_3x3**	Realiza la multiplicación.
i_ikh_mult_ppr_res_addr_cntr	**linear_3x3_addr_cntr**	Genera las direcciones y habilita la escritura del resultado en memoria.
i_res_ram_dina_reg	**reg**	Sincroniza el dato con la dirección de escritura.

Tabla 8.12: Elementos del módulo *i_ikh_mult_ppr*

Instancia	FIFO destino	Instancia read_fifo	Instancia RAM
i_xpo_local	*i_correction2prediction_x*	*i_fifo_write_xpo*	*i_xpo_ram*
i_ppo_local	*i_correction2prediction_p*	*i_fifo_write_ppo*	*i_ppo_ram*

Tabla 8.13: Módulos escritores en FIFO usados en CORRECTION

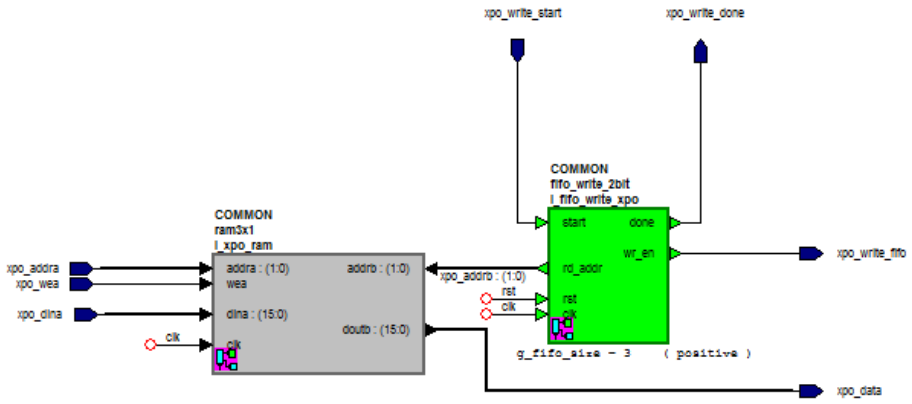


Figura 8.13: Diagrama de bloques del módulo *i_xpo_local*

Puerto	Tamaño	Sentido	Comentario
xpo_write_start	1 bit	entrada	Al activarse comienza el proceso de escritura en FIFO.
xpo_write_fifo	1 bit	salida	Habilita la escritura de datos sobre la FIFO.
xpo_data	16 bits	salida	Datos enviados hacia la FIFO *i_correction2prediction_x*.
xpo_write_done	1 bit	salida	Señala la finalización de la escritura en FIFO.
xpo_addra	2 bits	entrada	Dirección de escritura para la memoria interna.
xpo_wea	1 bit	entrada	Habilita la escritura sobre la memoria interna.
xpo_dina	16 bits	entrada	Entrada de datos hacia la memoria interna.

Tabla 8.14: Interfaz del módulo *i_xpo_local*

Capítulo 9

Resultados experimentales

En este capítulo se detallan los resultados experimentales obtenidos mediante las simulaciones del sistema utilizando los datos de diferentes pacientes reales. Para crear las representaciones gráficas se ha seguido el siguiente procedimiento:

En primer lugar, con la herramienta HDL Designer se ha creado un testbench. En él hay dos arquitecturas (y además se dispone de las matrices de covarianza Q y R):

- Filtro de Kalman.
- Analizador de ficheros externos de los cuales extrae los valores de la medición de la glucosa de un paciente y los introduce en el filtro.

En segundo lugar, el filtro de Kalman genera datos de salida que son recogidos en otro fichero de texto para su futuro tratamiento.

Por último, se analiza el fichero de texto en Matlab y representamos la glucosa del paciente generada por el monitor continuo de glucosa frente a los datos generados por el filtro.

Tal y como se ha establecido en el capítulo 3, el vector del estado del sistema tiene tres componentes. La componente de interés es la primera, que es la correspondiente a la glucosa estimada.

9.1. Ajuste de los parámetros Q y R

Los resultados que proporciona el filtro dependen del ajuste de dos parámetros:

- Q: covarianza del ruido de proceso
- R: covarianza del ruido de la medición

En teoría, el filtro se guiará más por la predicción o por la medición en función de la relación entre estos dos parámetros.

- Un valor de Q mayor que R indica que las mediciones son más fiables que las estimaciones del filtro.
- Análogamente, un valor de R más alto que Q indica que las mediciones llegan con mucho ruido, por lo que el filtro se fía más de sus propias estimaciones.

Para comprobarlo, se han hecho simulaciones con diferentes relaciones Q/R. El resultado de estas simulaciones se refleja en la figura 9.1, donde la línea azul representa la medida de la glucosa y

- la línea roja representa $R = 64$ y $Q = 2$. La salida del filtro con este ajuste se aleja de la medida, de lo que se extrae que el resultado se basa más en la predicción del filtro.
- la línea verde representa $R = 2$ y $Q = 2$. La salida del filtro con este ajuste se acerca a la glucosa, pero no da los valores exactos.
- la línea de puntos azul claro representa $R = 2$ y $Q = 64$. La salida en este caso coincide exactamente con la medida de la glucosa. Esto es así porque el filtro tiene más en cuenta la medida de glucosa que le entra como entrada que la propia predicción que hace de ella.

Tras el estudio de los parámetros R y Q podemos confirmar que el valor de estos parámetros afecta a los resultados que proporciona el sistema:

- si la relación Q/R >1, el filtro tiene más en cuenta las medidas de entrada que su propia predicción.
- si la relación Q/R <1, el filtro tiene más en cuenta la estimación que las medidas de entrada.

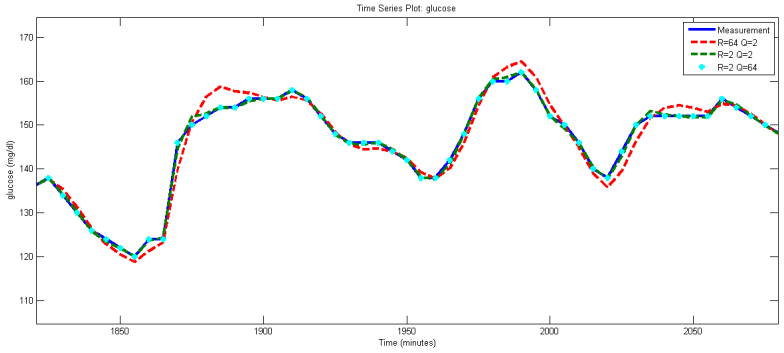


Figura 9.1: Simulación con diferentes relaciones Q/R

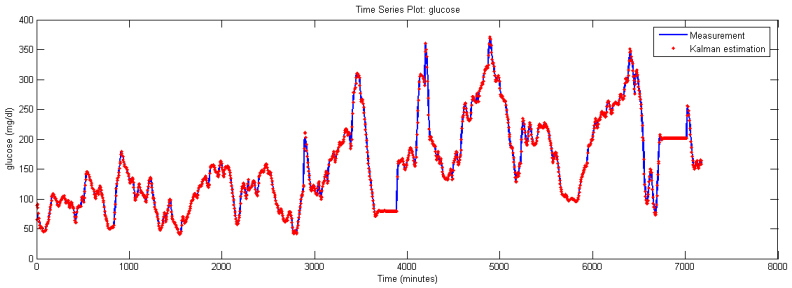
En este caso se ha elegido fijar $R = 16$ y $Q = 2$. Sin embargo, debido a que se recibe una medida de glucosa cada 1-5 minutos, ese intervalo de tiempo se podrá aprovechar para que se ajusten estos parámetros (externamente al FK) para minimizar los efectos de la degeneración del sensor, y también para adaptar el filtro a las circunstancias particulares de cada paciente. Con ellos se conseguiría paliar dos de los principales problemas que tienen en la actualidad los CGMs.

9.2. Simulaciones con datos reales

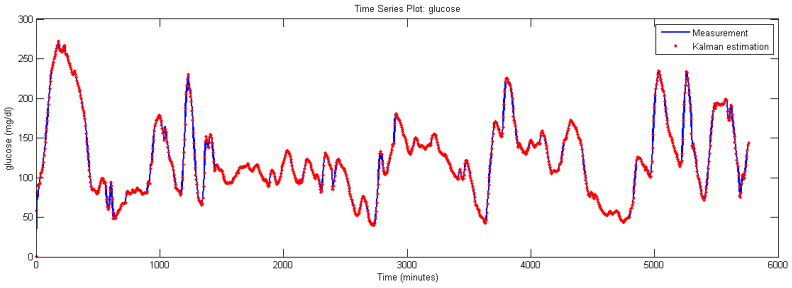
Para realizar las simulaciones se cuenta con datos reales de glucosa 4 pacientes con Diabetes. Para diferenciarlos se les adjudicó identificadores a cada paciente: BG, CL, JB y ME.

Los resultados del filtro se han representado en gráficas (figura 9.2) para su comparación con las mediciones reales de los pacientes. En dichas gráficas se representa cómo la glucosa (en mg/dl) varía en función del tiempo. La línea azul continua corresponde a la medida real, mientras que los puntos en rojo corresponden a las estimaciones realizadas por el filtro de Kalman.

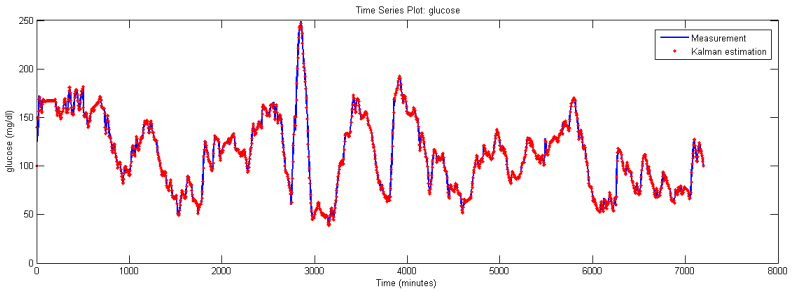
En las gráficas se puede observar que, mientras la glucosa varíe poco, el filtro de Kalman devuelve estimaciones muy precisas. En cambio cuando la glucosa varía de forma brusca las estimaciones del filtro son erróneas. Sin embargo, el FK utiliza dichos resultados erróneos corregir las siguientes predicciones y vuelve a realizar predicciones correctas.



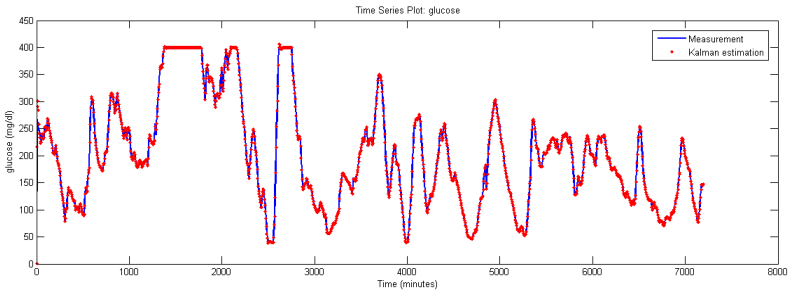
(a) Simulación con los datos del paciente BG



(b) Simulación con los datos del paciente CL



(c) Simulación con los datos del paciente JB



(d) Simulación con los datos del paciente ME

Figura 9.2: Resultados de las simulaciones con diferentes pacientes.

Capítulo 10

Síntesis

10.1. Configuración de la herramienta

La configuración utilizada para sintetizar el diseño en HDL Designer es:

- FPGA Vendor: Xilinx
- Family: Virtex 6
- Device: 6vcx75t
- Package: ff484

10.2. Análisis del informe de síntesis

En la tabla 10.1 se muestra la información resultante de hacer la síntesis, sacada del HDL Synthesis Report.

En el mismo se detalla el resumen de la temporización (Timing summary) y el uso total de memoria del diseño:

- Periodo mínimo: 33,662ns (Frecuencia máxima: 29,707MHz)
- Tiempo mínimo de llegada de entradas antes del reloj: 1,660ns
- Tiempo máximo necesario para las salidas después del reloj: 2,475ns
- Retardo del camino combinacional máximo (camino crítico): 0,668ns
- El uso total de memoria es de 296832 KB $\approx$ 290 MB, representando un 3 % del área disponible en la FPGA.

Debido a que los datos sólo se reciben cada 1-5 minutos, no es necesaria una frecuencia de trabajo muy alta. El módulo CORRECTION queda parado hasta que llega una nueva medición con la que corregir las estimaciones.

Módulo	Número de Componentes
RAMs	38
Multiplicadores	9
Sumadores/Restadores	81
Registros	610
Comparadores	17
Multiplexores	387
Máquinas de Estados	29
Xors	1
FIFO	6
Contadores	33

Tabla 10.1: Módulos utilizados en la síntesis del proyecto

Esto lleva a que el resto del circuito también se quede parado debido a la dependencia existente entre los módulos.

Habiendo tanto tiempo disponible entre dato y dato, se podría cuestionar la necesidad de implementar el filtro de Kalman en hardware, ya que al parecer la mayor parte del tiempo está en espera. Sin embargo:

- A diferencia de su variante software, que recibe las señales ya procesadas por los filtros digitales que el sensor tiene incorporado, en la implementación en hardware recibirá las señales brutas recogidas por el sensor, aumentando la calidad del filtrado.

- Por otro lado, tal y como se comenta en el capítulo anterior, mientras el filtro está en espera se pueden ir calculando nuevos valores para los parámetros Q y R con la finalidad de ajustar mejor el filtro a la degeneración del sensor por un lado, y buscar un ajuste más personalizado para un paciente concreto.

- Con esta última finalidad, también se podrían adaptar las matrices constantes A y H.

En caso de que durante los periodos de espera del filtro no diese tiempo para el cálculo de los nuevos parámetros Q y R, podría ser necesario aumentar la frecuencia de trabajo del filtro, por ejemplo, mediante segmentación. Sin embargo, en caso contrario no convendría aumentar la frecuencia, ya que una elevada frecuencia implica un mayor consumo por parte del circuito.

10.3. Camino crítico

El informe de síntesis establece la siguiente ruta como camino crítico:

```
Data Path: i_gain/i_mult_hpht_r/i_res_mult_d/dout_0 (FF)
to i_gain/i_gain/Mram_ram (RAM)
                              Gate     Net
  Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
  ----------------------------------------  ------------
   FDR:C->Q              1   0.375   0.491  dout_0 (dout_0)
   end scope: 'i_gain/i_mult_hpht_r/i_res_mult_d:dout<0>'
   begin scope: 'i_gain/i_mult_hpht_r/i_adder:oper1<0>'
   end scope: 'i_gain/i_mult_hpht_r/i_adder:res<3>'
   end scope: 'i_gain/i_mult_hpht_r:res_hpht_r<3>'
   begin scope: 'i_gain/i_inversion:oper2<3>'
   begin scope: 'i_gain/i_inversion/div_out[15]_oper2[15]_div_0:b<3>'
   end scope: 'i_gain/i_inversion/div_out[15]_oper2[15]_div_0:o<14>'
   end scope: 'i_gain/i_inversion:res<15>'
   begin scope: 'i_gain/i_gain:dina<15>'
   RAMB18E1:DIADI15          0.261          Mram_ram
  ----------------------------------------
  Total                     33.662ns (12.478ns logic, 21.184ns route)
                                     (37.1% logic, 62.9% route)
```

El inicio de éste es la salida del registro *i_res_mult_d*, que retrasa la propagación del resultado de la multiplicación un ciclo. Esta salida entra como primer operando al sumador combinacional *i_adder* y, al salir de éste, sale también del módulo que los contiene por el puerto res_hpht_r.

Una vez fuera, el resultado de estas operaciones entra como segundo operando al módulo *i_inversion*, que consta de varios módulos combinacionales, como el redondeo de la división, la propia operación de división y el desplazamiento a la derecha (ver sección 5.2). Por último, este resultado se transmite a la RAM *i_gain*, ya que es el resultado del cálculo de la ganancia que se transmitirá a continuación a la FIFO que comunica Gain con Correction.

Como se puede apreciar, el tiempo de operaciones no es excesivo, 12.5 ns, mientras que lo que más tiempo consume es la transmisión de los datos entre módulos, 21.2 ns. En total, 33.662 ns, que coincide con el periodo mínimo de la señal de reloj. En caso de ser necesario aumentar la frecuencia del sistema, la distribución de este módulo habría que modificarla o segmentarla para que el periodo del camino fuera menor. En la figura 10.1 se ilustran los módulos que intervienen en el camino crítico.

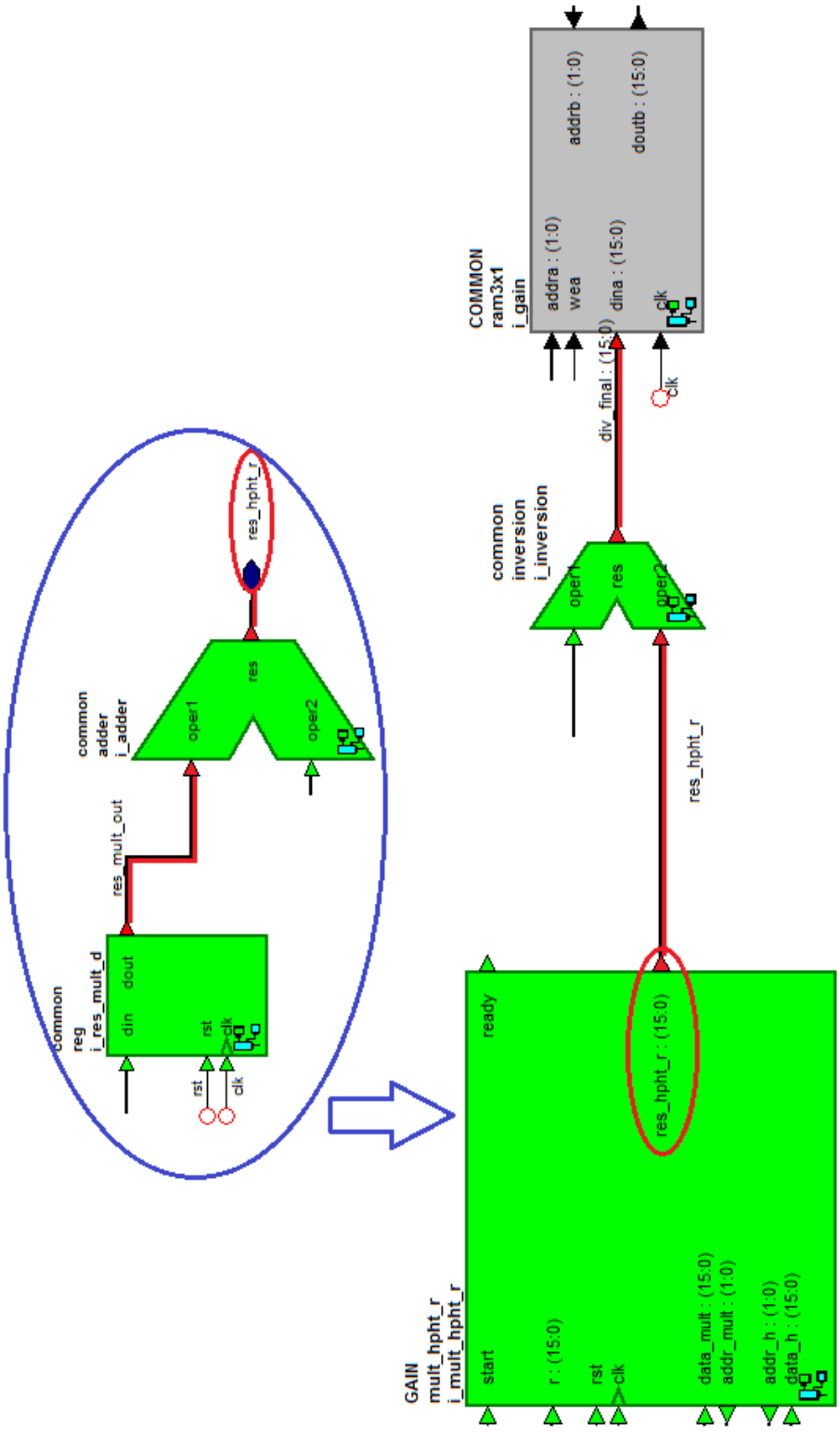


Figura 10.1: Diagrama del camino crítico del FK

Parte III

Conclusiones y trabajo futuro

Capítulo 11

Conclusiones

Este proyecto forma parte de un proyecto de mayor magnitud que tiene como finalidad mejorar el control de la glucemia en pacientes con diabetes. Esto se lleva a cabo mediante bombas de insulina y monitores continuos de glucosa. Éstos últimos tienen incorporados sensores que generan la señal de la medición de la glucosa. Pero los sensores sufren degeneración a lo largo de su vida útil, lo que añade ruidos que es preciso eliminar. El filtro de Kalman es un candidato ideal para realizar esta tarea.

El filtro de Kalman estima el estado de un sistema dinámico a partir de un conjunto de medidas indirectas relacionadas con el mismo. En este proyecto se ha partido de unos modelos de proceso y medición discretos y lineales. Para los ruidos se ha supuesto que son blancos y gaussianos. A partir de los supuestos anteriores se han obtenido las ecuaciones que describen el comportamiento del filtro que se han utilizado posteriormente para la implementación.

El filtro consta de dos etapas, una de predicción y otra de corrección. La predicción estima los resultados del filtro para poder corregirlos en la etapa de corrección. Para ello usa las medidas tomadas del propio paciente para que las predicciones futuras sean más fiables y precisas.

El sistema se compone en tres módulos: uno que realiza la estimación, otro el cálculo de la ganancia de Kalman y por último la corrección. Cada uno de estos módulos ha sido completamente diseñado en VHDL por los miembros del proyecto. Las transmisión de los datos entre los distintos módulos se ha realizado mediante memorias FIFO que permiten la comunicación unidireccional de un módulo a otro. Estas memorias han sido generadas con la herramienta Core Generator de Xilinx.

El correcto funcionamiento del sistema ha sido verificado mediante

1. la simulación del código VHDL usando datos de glucosa de pacientes reales y

2. la comparación de los resultados con los datos generados por modelos

en coma fija de MatLab que también han sido desarrollados por los miembros del proyecto.

La concordancia de ambas simulaciones nos permiten asegurar que el Filtro de Kalman diseñado en este proyecto funciona tal como se esperaba.

El filtro ha sido sintetizado y se ha generado un bitstream para su volcado sobre una FPGA Virtex-6 modelo 6vcx75t. El área que ocupa la implementación supone 3 % del área disponible de la FPGA con lo que el filtro aquí diseñado deja suficiente área de la FPGA disponible para la implementación de sistemas más complejos. Podría por lo tanto usarse este filtro como un bloque de propiedad intelectual, IP, en el diseño de sistemas on chip.

Por último, el análisis estático de tiempos estima que la frecuencia de trabajo máxima del filtro es 29,707MHz. Esta frecuencia es muy superior a la frecuencia requerida para el procesado de datos provenientes de una monitor continuo de glucosa y posibilita su utilización en sistemas con requisitos de temporización más exigentes.

Capítulo 12

Conclusions

This project is part of a greater project whose purpose is to improve the glycemic control on patients of Diabetes. This is carried out with insulin pumps and continuous glucose monitors. These last ones have sensors incorporated that generate the glucose measurement signal. But sensors suffer degeneration during its lifespan. Due to this, a white noise is incorporated into the signal, so it becomes necessary to remove it. The Kalman filter is an ideal candidate to perform this task.

The Kalman filter estimates the state of a dynamic system from a set of indirect measurements related to itself. In this case, we started from discrete and linear model of process and measurement. It was assumed that the noise was white and gaussian. The equations that describe the filter have been extracted from the mentioned model, and have been used to implement it afterwards.

The filter has two stages, a prediction one and a correction one. Prediction estimates the results of the filter so it can correct them in the next stage, correction. To do this, it uses the measurements taken from the patient so that future estimations are more accurate and reliable.

The system has three main modules: one that carries out the estimation, another that calculates the Kalman gain and the last one for the correction. Every member of the team has designed one of these modules each, completely in VHDL. The data transmission between modules has been done using FIFO memories that allow unidirectional communication from a module to another. These memories have been generated using the Xilinx Core Generator tool.

The proper functioning of the system has been verified by:

1. simulating the VHDL code using glucose data obtained from real patients and

2. comparing those results with data generated with fixed-point models designed in Matlab by the team members.

The consistency between both simulations show that the Kalman Filter designed in this project works just as expected.

The filter has been synthesized and a bitstream has been generated to be dumped over a Virtex-6 FPGA, 6vcx75t model. The implementation takes up only 3 % of the available area of the FPGA. Thus, the filter designed in this project leaves enough free area on the FPGA which allows the implementation of systems with a greater complexity. Therefore, it could be used as an Intellectual Property (IP) block within System-on-Chip designs.

Finally, static time analysis estimates that the maximum working frequency is 29.707MHz. This frequency is much higher that the required frequency to process the data that comes from a continuous glucose monitor. This makes possible its incorporation into systems that have stricter timing requirements.

Capítulo 13

Futuras líneas de trabajo

En este proyecto se ha abordado la primera etapa del proyecto principal, el diseño y la implementación en hardware del filtro Kalman, para eliminar el ruido y mejorar la calidad de la señal de entrada. El futuro de este proyecto pasará por una serie de etapas, a saber:

1. Implementar técnicas de clock-gating para reducir el consumo de energía del sistema.

2. Definir e implementar conjuntos de simulaciones más extensas que aseguren una cobertura del código del 100 %.

3. Debido a que el sensor envía una medida al filtro cada 1-5 minutos, no es necesaria una frecuencia de trabajo muy elevada. Sin embargo la mayor parte del tiempo el filtro está parado, a la espera de recibir el siguiente dato. Este intervalo de tiempo en los que el FK está parado se podrá aprovechar para buscar nuevos parámetros del filtro que ajusten y mejoren su comportamiento.

4. Como los datos se reciben cada 1-5 minutos, el FK tiene suficiente tiempo para realizar sus cálculos hasta recibir la siguiente medida. Pero si en este intervalo de tiempo no diese tiempo a encontrar los nuevos parámetros Q y R del filtro, se podría segmentar para aumentar la frecuencia a la que éste trabaja.

5. Desarrollo del System on Chip(SoC) evolutivo sobre el que se implementará el filtro desarrollado en este proyecto. El SoC constará de los siguientes componentes entre otros: un microprocesador para ejecutar el algoritmo evolutivo, el filtro que será gestionado por el microprocesador, un módulo que evaluará la validez del filtro, memoria, buses y periféricos.

6. Estudio de la adaptabilidad del filtro. Se determinará qué parámetros del filtro deben evolucionar para que se adapte dinámicamente a las ne-

cesidades del paciente y a las características del sensor. Posteriormente se implementará en el hardware del filtro.

7. Sustituir el microprocesador por hardware que implemente el algoritmo evolutivo dentro del SoC. Esto se debe a que un microprocesador no está pensado para ser utilizado por el paciente en su vida cotidiana sino para hacer pruebas de desarrollo. El uso de hardware específico permite mejorar la portabilidad y durabilidad del sistema.

8. Debido al objetivo anterior de sustituir el microprocesador, la gestión del filtro también debe ser revisada. Por ello se buscará hacerla autónoma y que se autoadapte a los fallos y la degeneración del sensor, modificando su configuración.

Además, en este proyecto se ha tenido en cuenta el formato de unos datos concretos a usar, lo que ha condicionado el ancho de los buses entre módulos, el tamaño de las memorias y FIFOs y las dimensiones de las matrices y vectores entre otros elementos. Como mejora se pueden usar tamaños genéricos en la definición de los módulos para que el filtro pueda utilizarse para cualquier modelo y datos, no unos concretos.

Parte IV

Apéndices

Apéndice A

Representación Aritmética

En este proyecto se trabaja con números binarios expresados en punto fijo. En este apéndice se explican los aspectos más relevantes de dicha notación.

A.1. ¿Qué es punto fijo?

Es una representación de números reales que cuenta con la ventaja de poder hacer conversiones a y desde números naturales de forma prácticamente inmediata. Como desventaja, el dominio de los valores que puede representar es reducido e impreciso comparado con la notación científica. Sin embargo, a igualdad de bits, el punto fijo tiene mayor precisión.

Una representación en punto fijo se basa en una representación de números naturales en la que los valores reales han sido multiplicados por un factor antes de ser expresado como número natural. Para recuperar el valor real, deberá dividirse por dicho factor. Si prefijamos un factor de 1000, el valor real de 0,0034 será expresado como $0,0034 * 1000 = 3$, por lo que la precisión máxima está determinada directamente por el factor utilizado.

A.2. Operar en punto fijo

Debido a que cada valor lo hemos multiplicado por una constante, se debe tener en cuenta a la hora de realizar las operaciones, es decir, si a y b son números reales y f el factor de precisión usado, sea $A = a \cdot f$ y $B = b \cdot f$. A continuación se ejemplifican las operaciones aritméticas básicas:

- Suma: $A + B = a \cdot f + b \cdot f = (a + b) \cdot f \rightarrow a + b \sim A + B$
- Multiplicación: $A \cdot B = a \cdot f \cdot b \cdot f = (a \cdot b) \cdot f^2 \rightarrow a \cdot b \sim (A \cdot B)/f$
- División: $A/B = a \cdot f/b \cdot f = a/b \rightarrow a/b \sim (A \cdot f)/B$

Las operaciones anteriores las podemos realizar las veces que sean necesarias, pero debemos tener en cuenta que para obtener la variable real o la fija se ha de hacer lo siguiente:

- Dada la variable real a, para obtener la variable en coma fija A debemos hacer $A = a \cdot f$
- Dada la variable en coma fija A, para obtener la variable real a debemos hacer $a = A/f$
- Los errores por redondeo cometidos.

A.3. Notación Q

Dado que el factor de escala puede variar, la "notación Q" se utiliza para especificar el número de dígitos a la derecha del punto. Una notación Q_n indica que el número de bits a la derecha del punto es n. En general la notación Q no especifica la longitud de la palabra. Cuando es necesario, se utiliza una notación más compleja $Q_{m.n}$ donde m son los bits para la parte entera y n los bits para la fracción. Además, se reserva un bit para el signo, luego $n = m + n + 1$. Para calcular el valor decimal de un número binario de n bits con notación $Q_{m.n}$ se realiza la operación que se muestra en la figura A.1.

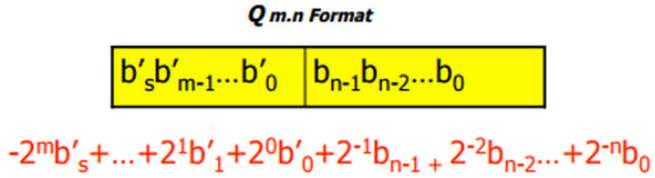


Figura A.1: Formato $Q_{m.n}$

A.3.1. Determinación de m y n en una notación $Q_{m.n}$

Un problema a la hora de utilizar la notación $Q_{m.n}$ es fijar, para un determinado número de n bits, el valor de m y el valor de n. Para ello hay que definir el rango dinámico del problema a tratar. El rango dinámico depende precisamente de los valores de m seleccionado y viene dado por la expresión $[-(2^m), 2^m - 2^{-n}]$, siendo 2^{-n} la precisión para la notación $Q_{m.n}$.

Vamos a ver un ejemplo, en números representados en complemento a dos y notación $Q_{2,7}$ el rango es $[-(2^2), 2^2 - 2^{-7}] = [-4, 3{,}9921875]$ y la precisión $2^{-7} = 0{,}00781$.

Debido al número limitado de bits, el rango y la precisión de los valores del sistema variarán según se asignen más o menos dígitos a m o a n, por lo que cuanto mayor sea el rango, menor precisión tendrán los números. Por lo tanto, la forma de determinar la relación entre m y n es determinar cuál va a ser el mayor valor positivo que se contemplará para el sistema y resolver las ecuaciones

$$2^m - 2^{-n} = MAX \tag{A.1}$$

$$n = m + n + 1 \tag{A.2}$$

Los valores adecuados entre los que debe oscilar la glucemia en el sistema deben estar comprendidos en el intervalo de 100 a 300. Para representar estos valores es necesaria como mínimo una parte entera de 9 bits, con la que se puedan representar $29 = 512$ valores. Contando con el bit de signo y el probable desbordamiento que se pueda producir en los cálculos, se ha decidido utilizar una parte entera de 12 bits y una parte decimal de 4 ya que no es imprescindible una gran precisión, sino un mayor rango de valores. Utilizando la notación Q anteriormente explicada, se ha hecho uso de $Q_{12,4}$.

A.3.2. Aritmética en notación Q

Los números $Q_{m.n}$ se pueden ver como la división entre dos enteros, el numerador es el número almacenado en el registro y el denominador es 2^n. Al tener que mantener la posición del punto, hay que conservar el denominador constante. La figura A.2 muestra las fórmulas de cómo se deben ejecutar las operaciones matemáticas para mantener el denominador constante.

$$\begin{aligned}
\frac{N_1}{d} + \frac{N_2}{d} &= \frac{N_1 + N_2}{d} \\
\frac{N_1}{d} - \frac{N_2}{d} &= \frac{N_1 - N_2}{d} \\
\left(\frac{N_1}{d} \times \frac{N_2}{d}\right) \times d &= \frac{N_1 \times N_2}{d} \\
\left(\frac{N_1}{d} / \frac{N_2}{d}\right) / d &= \frac{N_1 / N_2}{d}
\end{aligned}$$

Figura A.2: Fórmulas para operar con notación Q manteniendo el denominador constante

Dado que el denominador es una potencia de dos, la multiplicación se implementa mediante un desplazamiento a la izquierda y la división hacia

la derecha. Para mantener la precisión, los resultados intermedios de multiplicaciones y divisiones deben ser de doble precisión y se debe realizar el redondeo del resultado intermedio antes de convertirlo al numero Q deseado.

Apéndice B

Herramientas utilizadas

Es este apartado se listarán las herramientas empleadas por el equipo en las diferentes etapas del proyecto.

- Simulación de modelos: Mathworks Matlab R2014a
- Entorno Unix: Cygwin 1.7.34
- Clientes Git: SyntEvo SmartGit 6.5 y TortoiseGit
- Diseño de hardware: Mentor Graphics HDL Designer Series 2012.1
- Simulación de módulos y sistemas: Mentor Graphics Questa Sim 10.1
- Documentación del proyecto: Google Drive
- Maquetación de la memoria del proyecto: MiKTeX 2.9.5105, TeXnic-Center y TeXiS v1.0

Apéndice C

Constantes del sistema

En el este apartado describimos el código utilizado para declarar las constantes para el Filtro de Kalman, se definen dimensiones, tipos y constantes de las matrices.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  package constants is
5  constant c_row_matrix_width  : natural := 3;
6  constant c_square_matrix_width  : natural := 3;
7  constant c_square_matrix_height : natural := 3;
8
9  ------------------------------------------------
10 -- Data types
11 ------------------------------------------------
12
13 constant c_data_width : natural := 16;
14 constant c_addr_width : natural := 4;
15
16
17 subtype t_data is std_logic_vector(c_data_width-1 downto 0);
18 subtype t_data_int is integer range 0 to (2**c_data_width)-1;
19
20
21 subtype t_addr is std_logic_vector(c_addr_width-1 downto 0);
22 subtype t_addr_int is integer range 0 to (2**c_addr_width)-1;
23
24
25 type t_3x3_matrix is array (0 to c_square_matrix_height *
26 c_square_matrix_width-1) of t_data;
27 type t_1x3_matrix is array (0 to c_row_matrix_width-1)
28 of t_data;
29
30
```

```
31 constant c_rom_a : t_3x3_matrix := ("0000000000010000",
32                                     "0000000000010000",
33                                     "0000000000000000",
34                                     "0000000000000000",
35                                     "0000000000010000",
36                                     "0000000000010000",
37                                     "0000000000000000",
38                                     "0000000000000000",
39                                     "0000000000010000");
40
41 constant c_rom_H : t_1x3_matrix := ("0000000000010000",
42                                     "0000000000000000",
43                                     "0000000000000000");
44
45 constant c_rom_I : t_3x3_matrix := ("0000000000010000",
46                                     "0000000000000000",
47                                     "0000000000000000",
48                                     "0000000000000000",
49                                     "0000000000010000",
50                                     "0000000000000000",
51                                     "0000000000000000",
52                                     "0000000000000000",
53                                     "0000000000010000");
54
55 constant c_rom_q : t_3x3_matrix := ("0000000000010000",
56                                     "0000000000000000",
57                                     "0000000000000000",
58                                     "0000000000000000",
59                                     "0000000000010000",
60                                     "0000000000000000",
61                                     "0000000000000000",
62                                     "0000000000000000",
63                                     "0000000000010000");
64 end constants;
```

Apéndice D

Abreviaturas y acrónimos

En el este apartado se listan las abreviaturas y acrónimos utilizados a lo largo de este documento.

DM1 Diabetes Mellitus de tipo 1

FK filtro de Kalman

ff flip-flop

FIFO first in - first out

FPGA Field Programmable Gate Array

FSM finite state machine

GI glucosa intersticial

GS glucosa en sangre

MCG monitor continuo de glucosa

mux multiplexor

PA páncreas artificial

PPO covarianza del error del filtro *a posteriori*

PPR covarianza del error del filtro *a priori*

RAM random-access memory

reg registro

ROM read-only memory

SoC System on Chip

VHDL Very high speed circuit Hardware Description Language

XPO estado del proceso *a posteriori*

XPR estado del proceso *a priori*

Bibliografía

Ashenden, P. J. y Lewis, J. *The Disegner's Guide to VHDL*, vol. 3. Morgan Kaufmann, 2010.

Bequette, B. W. Optimal estimation applications to continuous glucose monitoring. *Proceeding of the 2004 American Control Conference*, página 5, 2004.

Bequette, B. W. Continuous glucose monitoring: Real-time algorithms for calibration, filtering, and alarms. *Journal of Diabetes Science and Technology*, vol. 4(2), página 15, 2010.

Bishop, D. Fixed point package userѕ guide. *Packages and bodies for the IEEE*, páginas 1076–2008, 2006.

Facchinetti, A., Del Favero, S., Sparacino, G., Castle, J. R., Ward, W. K. y Cobelli, C. Modeling the glucose sensor error. *IEEE Transactions on Biomedical Engineering*, vol. 61(3), página 10, 2014.

Facchinetti, A., Sparacino, G. y Cobelli, C. An online self-tunable method to denoise cgm sensor data. *IEEE Transactions on Biomedical Engineering*, vol. 57(3), página 8, 2010.

Freckmann, G., Pleus, S., Link, M., Zschornack, E., Klotzer, H.-M. y Haug, C. Performance evaluation of three continuous glucose monitoring systems: Comparison of six sensors per subject in parallel. *Journal of Diabetes Science and Technology*, vol. 7(4), página 12, 2013.

Kleeman, L. Understanding and applying kalman filtering. PDF, 1996.

Kuure-Kinsey, M., Palerm, C. C. y Bequette, B. W. A dual-rate kalman filter for continuous glucose monitoring. *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual Conference*, página 4, 2006.

Lacey, T. Tutorial: The kalman filter. *Georgia Institute of Technology*, 1998.

LEWIS, J. Vhdl math tricks of the trade. PDF, 2003.

PALERM, C. C. y BEQUETTE, B. W. Hypoglycemia detection and prediction using continuous glucose monitoring; a study on hypoglycemic clamp data. *Journal of Diabetes Science and Technology*, vol. 1(5), páginas 624–629, 2007.

PALERM, C. C., WILLIS, J. P., DESEMONE, J. y BEQUETTE, B. W. Hypoglycemia prediction and detection using optimal estimation. *Diabetes Technology & Therapeutics*, vol. 7(1), página 13, 2005.

WELCH, G. y BISHOP, G. An introduction to the kalman filter. página 81, 2001.