ASPECTOS NUMÉRICOS EN LA IMPLEMENTACIÓN DE CONTROLADORES DIGITALES UTILIZANDO "HARDWARE" EN EL LAZO DE SIMULACIÓN

Sánchez Gómez Diego Felipe [1]
Sarria Villa César Alberto [2]

UNIVERSIDAD DEL CAUCA
FACULTAD DE CIENCIAS NATURALES, EXACTAS Y DE LA EDUCACIÓN
INGENIERÍA FÍSICA
POPAYÁN
2005

[1]dfsanchez@unicauca.edu.co
[2]casarria@unicauca.edu.co

ASPECTOS NUMÉRICOS EN LA IMPLEMENTACIÓN DE CONTROLADORES DIGITALES UTILIZANDO "HARDWARE" EN EL LAZO DE SIMULACIÓN

Sánchez Gómez Diego Felipe
Sarria Villa César Alberto

Trabajo de grado para optar al título de
Ingeniero Físico

Director
M.Sc Carlos Felipe Rengifo

UNIVERSIDAD DEL CAUCA
FACULTAD DE CIENCIAS NATURALES, EXACTAS Y DE LA EDUCACIÓN
INGENIERÍA FÍSICA
POPAYÁN
2005

Nota de aceptación

_____

_____

_____

_____

_____

_____

Director
M.Sc. Carlos Felipe Rengifo Rodas
Departamento de Electrónica
Instrumentación y Control
Universidad del Cauca

_____

Jurado
Ing. Víctor Hugo Mosquera Leyton
Departamento de Electrónica
Instrumentación y Control
Universidad del Cauca

_____

Jurado
Ing. Mario Andrés Córdoba Gonzáles
Departamento de Física
Universidad del Cauca

Fecha de sustentación: Popayán; Diciembre 6 de 2005.

CONTENIDO

pag.

Índice de figuras

Introducción

Vivimos en una era que los sociólogos han llamado la revolución de las computadoras. Como cualquier otra revolución, tiene amplia difusión, es penetrante y tendrá un efecto duradero en la sociedad. Es tan fundamental para la economía y el orden social actuales como lo fue la revolución industrial del siglo XIX. Afectara los patrones de pensamiento y el estilo de vida de cada individuo. Mientras que el principal efecto de la Revolución Industrial fue el aumento de nuestras posibilidades físicas, la revolución de las computadoras ampliará nuestro poder mental. Poco a poco hemos ido integrando las computadoras a nuestra vida diaria; éstas, realizan tareas con mayor efectividad que cualquier humano, y en consecuencia se ha generado un nuevo estilo de vida, en el cual la tecnología es el eje central de nuestra civilización actual.

A medida que integramos los dispositivos digitales a diferentes procesos industriales y aplicaciones comerciales, se presentan nuevos desafíos en el procesamiento de señal; los cuales implican, altas velocidades de procesamiento, gran capacidad de memoria, fácil programación, portabilidad, etc. Implementar algoritmos de control sobre estas tecnologías presenta ciertas dificultades; una ley de control consiste en un procedimiento matemático que calcula la salida del controlador a partir del estado actual del proceso. Las arquitecturas digitales afectan la representación de los parámetros del controlador y las operaciones realizadas por éste; ya que, presentan efectos de redondeo y cuantización debido a la representación numérica mediante cadenas binarias finitas. La implementación de algoritmos de control sobre un computador requiere un algoritmo bien condicionado y numéricamente estable; en el cual, una pequeña perturbación en los parámetros del controlador o un calculo mal realizado, debido a efectos de cuantización y/o redondeo, no modifiquen el comportamiento de lazo cerrado del sistema.

Cuando se diseñan controladores digitales, es posible obtener diferentes realizaciones en espacio de estados para este. Si se asume una precisión infinita en las operaciones aritméticas, dichas representaciones son equivalentes. Sin embargo; al implementar la ley de control, es el dispositivo quien establece la precisión: en los cálculos, en la representación de los coeficientes del controlador y el rango de valores que es posible operar sin restricción. Las consecuencias de la longitud de palabra finita afectan de manera diferente a cada una de las realizaciones del controlador y por lo tanto no son equivalentes, ni todas presentan un desempeño adecuado.

Una vez definido el controlador, todo procedimiento computacional que permita determinar la ley de control a partir del error, constituye una realización; entre las cuales se encuentran las de forma directa, Serie, Paralela entre otras. Obtener la realización del controlador que genere la respuesta deseada al incorporarla al proceso no es sencillo; pues está limitado por la precisión numérica y las aproximaciones consideradas por el diseñador. Para muchos la estabilidad numérica consiste en utilizar "hardware" de sofisticadas características, elevando así los costos de implementación, y en ocasiones sin lograr el resultado esperado. Los estudios recientemente desarrollados en éste tema, abordan el problema calculando la sensibilidad de los polos del controlador a los efectos de cuantización en sus coeficientes, y proponen la realización de menor índice de sensibilidad como la de mejor desempeño de lazo cerrado, considerando los efectos de la longitud de palabra finita.

Cuando se implementa la ley de control sobre alguna arquitectura digital implica la validación de su comportamiento de lazo cerrado en el sistema real. Condición que puede acarrear consecuencias irreversibles sobre la planta y elevar los costos de implementación si se prueba un controlador que no presente un buen desempeño; motivo por el cual la simulación se ha convertido en una herramienta fundamental en la construcción de prototipos, debido a su relación directa con el sistema, minimizando los costos y el tiempo de ejecución del proyecto. En los últimos años, las herramientas software para el diseño, implementación y validación de sistemas de control, han incorporado "hard-

ware" en el lazo de simulación HIL (hardware in the loop). Al involucrar "hardware" y "software" como componentes de la simulación, ésta es mucho más verídica, pues en estos procesos de validación se pueden tener conectadas partes reales que interactuaran con la simulación; considerando aspectos de realización esenciales, que los métodos tradicionales no tienen en cuenta.

En este trabajo se estudiara experimentalmente, mediante simulaciones HIL, el comportamiento de lazo cerrado de las realizaciones serie, paralela, delta y nominal del controlador de posición de un péndulo invertido, implementadas sobre una FPGA. A cada realización se le calculo la sensibilidad polo-coeficiente por medio del índice propuesto por James F. Whindborne, Robert S. H. Istepanian y Jun Wu; luego, se comparan los resultados teóricos, con los obtenidos de las simulaciones HIL. De los cuales se espera que las realizaciones que presenten un adecuado comportamiento de lazo cerrado, posean un índice de sensibilidad bajo. El documento esta conformado por cuatro capítulos:

En el capitulo uno, se introducen los conceptos de cuantización, redondeo, y sensibilidad; además, se explica el procedimiento matemático para obtener diferentes realizaciones del controlador en tiempo discreto.

En el capitulo dos, se da una noción general del método de validación HIL (Hardware-in-the-Loop Simulation), se comentan sus posibilidades tanto a nivel industrial como académico y las ventajas que presenta sobre los métodos clásicos de validación de controladores digitales. Además, se ilustra el procedimiento de implementación de una plataforma de validación de controladores digitales HIL utilizando el "toolbox" de MATLAB, xPC Target.

En el capitulo tres, se presenta el proceso de implementación de un sistema de control embebido (ECS) sobre una FPGA, para el posicionamiento de un péndulo invertido; También, se ilustra el procedimiento por el cual se obtienen diferentes realizaciones de éste, usando funciones de ***Matlab***. Finalmente se comentan las dificultades que se presentaron al diseñar una arquitectura dedicada en VHDL, y como las nuevas herra-

mientas para el diseño hardware en FPGAs facilitaron este proceso.

En el capitulo cuatro, se presenta los resultados de las simulaciones HIL de cada realización y se analiza tanto el comportamiento dinámico del sistema realimentado como las características algorítmicas que presenta cada una de ellas. También, se presentan las conclusiones de la investigación.

Capítulo 1

Implementación en Espacio de Estados de Controladores de un Grado de Libertad

1.1. Introducción

Un computador es un dispositivo capaz de resolver problemas o controlar información según una secuencia algorítmica, mediante un proceso mecánico o eléctrico. Desde que las personas empezaron a resolver problemas hace miles de años, se han buscado diversas formas de simplificar las tareas para solucionarlos. La automatización de las operaciones aritméticas ha sido de interés primordial en los últimos años. El surgimiento de las computadoras proporciono una forma económica de realizar aritmética sencilla, y conforme maduró la tecnología las técnicas de computación se ampliaron con rapidez para resolver problemas numéricos en diversas áreas científicas o implementaciones industriales que controlan robots, maquinaria, automóviles, juegos, fabricas y una amplia gama de procesos. Sin embargo, el uso de computadoras presenta nuevas dificultades, debido a que no almacenan números como $\frac{2}{3}$, $7\frac{3}{7}$, $\sqrt{2}$ y π. En su lugar utilizan cadenas binarias de longitud finita; que muchas veces no son suficientes para representar exactamente un valor en particular [1].

Las múltiples posibilidades que presentan las arquitecturas digitales tales como: fácil programación, portabilidad, bajo costo etc; han llevado a los Ingenieros en Control a implementar sus algoritmos sobre estos dispositivos. Para calcular una ley de control por medio de un computador, se deben considerar los efectos de redondeo y cuantización como parámetros de diseño. El redondeo en las operaciones implícitas en éste, produce respuestas inexactas; mientras la cuantización afecta la representación de sus coeficientes, generando un controlador totalmente diferente al diseñado. Estos problemas de implementación, pueden generar un comportamiento inestable del sistema de lazo cerrado, y en algunas situaciones causar daños irreparables sobre el mismo [2], [3].

Cuando se diseñan controladores digitales, es posible obtener diferentes realizaciones en espacio de estados. Si se asume una aritmética de precisión infinita, dichas representaciones son equivalentes; sin embargo, las consecuencias de longitud de palabra finita (FLW), afectan de manera diferente a cada una de estas y por lo tanto no son equivalentes, ni todas presentan un optimo desempeño. Todo procedimiento computacional que permita determinar la ley de control a partir del error, constituye una realización; entre las cuales se destacan la directa, paralela y cascada. Elegir una estructura adecuada que minimice la sensibilidad del controlador ante efectos FLW requiere de un conocimiento profundo de los problemas anteriormente planteados y sus implicaciones en el sistema [1], [4].

En este capitulo se introducen los conceptos de cuantización, redondeo, y sensibilidad; además, se explica el procedimiento matemático para obtener diferentes realizaciones del controlador en tiempo discreto.

1.2. Condicionamiento un problema matemático

Un problema matemático se dice que es bien condicionado si pequeños cambios en los datos producen una variación proporcional en la solución. Si estos tienen el potencial de provocar grandes alteraciones en la solución, el problema se dice que es mal condicionado. A continuación se muestra que calcular los autovalores de una matriz u obtener

las raices de un polinomio puede conllevar a problemas matemáticos mal condicionados [5], [6].

Ejemplo 1.2.1 Considere la matriz:

$$A = \begin{bmatrix} -128 & 127 & 3 \\ 653{,}1 & 138 & -15 \\ 287 & 63 & -4 \end{bmatrix} \tag{1.1}$$

Cuyos autovalores son: 3, $\frac{3}{2} \pm \frac{7}{10}\sqrt{5}i$. Si se perturba el coeficiente $A(2,1)$ de 653,1 a 652,9 los nuevos autovalores son: 3, $\frac{3}{10} \pm \frac{1}{10}\sqrt{295}$. Una variación porcentual de

$$\frac{|653{,}1 - 652{,}9|}{|653{,}1|} \times 100 = 3{,}0623 \times 10^{-3}\,\%, \tag{1.2}$$

en el coeficiente $A(2,1)$ de la matriz A produjo un cambio porcentual de

$$\frac{\left|\left(\frac{3}{2} + \frac{7}{10}\sqrt{5}i\right) - \left(\frac{3}{10} + \frac{1}{10}\sqrt{295}\right)\right|}{\left|\frac{3}{2} + \frac{7}{10}\sqrt{5}i\right|} \times 100 = 76{,}044\,\%, \tag{1.3}$$

Este ejemplo muestra que un minúsculo cambio en un coeficiente de una matriz, tiene el potencial de provocar un gran cambio en los autovalores.

Ejemplo 1.2.2 Considere el polinomio:

$$P(x) = x^4 - 4x^3 + 6x^2 - 4x + 1$$

Cuyas raices son: 1,0, 1,0, 1,0 y 1,0. Si se perturba el término independiente de 1,0 a 0,999 las nuevas raices serán: 0,9, 1,1, $1 \pm 0{,}1i$.

Los ejemplos anteriores muestran el mal condicionamiento debido a problemas inherentes de sensibilidad de las matrices y los polinomios ante variaciones en sus coeficientes. Estos problemas son independientes de redondeo y cuantización [2], [3].

1.3. Efectos de la longitud de palabra finita

En un computador digital los números reales se representan utilizando secuencias binarias de longitud finita, lo que conlleva a cuantización en los coeficientes del controlador y a errores de redondeo en las operaciones aritméticas. La representación más común

de los números en un computador digital conocida como aritmética de punto flotante (ANSI-IEEE 754), utiliza un formato de 32 bits; donde el bit más significativo determina el signo, los siguientes 8 bits determinan el exponente, y los 23 bits menos significativos la mantisa.

Signo	Exponente	Mantisa
s	$e_7\ e_6\ \ldots\ e_0$	$m_{22}\ m_{21}\ \ldots\ m_0$

El signo S se calcula según:

$$S = (-1)^s$$

El exponente E se calcula de acuerdo con la siguiente expresión:

$$E = e_7 \times 2^7 + e_6 \times 2^6 + \cdots + e_1 \times 2^1 + e_0 \times 2^0$$

El valor mínimo que puede tomar E es 0, y el valor máximo es 255.

La mantiza M se calcula según:

$$M = m_{22} \times 2^{-1} + m_{21} \times 2^{-2} + \cdots + m_1 \times 2^{-21} + m_0 \times 2^{-23}$$

El valor mínimo que puede tomar M es 0, y el valor máximo es $M = 1 - 2^{-23}$.

Con el signo S, el exponente E y la mantiza M se determina el número N representado por la cadena binaria. Si E es mayor que 0 y menor que 255 ($0 < E < 255$), N se calcula con la expresión 1.4, [2].

$$N = (-1)^S \times (1 + M) \times 2^{E-127} \tag{1.4}$$

1.3.1. Cuantización de los coeficientes del controlador

La cuantización en los coeficientes del controlador genera cambios en su comportamiento dinámico, ya que, al variar los coeficientes de un polinomio se alteran sus raíces, de igual manera, al perturbar los coeficientes de una matriz se modifican sus autovalores [3], [4].

Ejemplo 1.3.1 Considere el controlador:

$$G(q) = \frac{q^3}{q^3 - 2{,}95q^2 + 2{,}9q - 0{,}95} \tag{1.5}$$

Cuyos polos son: 1,0, 1,0 y 0,95.
Si la implementación se realiza en aritmetica de punto flotante con 5 bits de mantisa y 2 de exponente, ninguno de los coeficientes de (1.5) se podrá representar exactamente. Los nuevos valores de los coeficientes del controlador serán:

$$G(q) = \frac{q^3}{q^3 - 2{,}9375q^2 + 2{,}875q - 0{,}953125} \tag{1.6}$$

Cuyos polos son: $1{,}23081$, $0{,}85335 \pm 0{,}21492i$. El controlador diseñado (1.5) es marginalmente estable y con doble acción integral. El controlador implementado tiene un comportamiento dinámico totalmente diferente,(1.6) es inestable y no tiene acción integral [2].

Sensibilidad autovalor - coeficiente

Como se mostro en el ejemplo1.2.1 los autovalores de algunas matrices son demasiado sensibles a perturbaciones en los coeficientes. Si $A \in \Re^{n \times n}$ y sus n autovalores son distintos, la sensibilidad del k-esimo autovalor con respecto a variaciones en los coeficientes de A puede determinarse utilizando la expresión propuesta por [4]:

$$S_k = \frac{\partial \lambda_k}{\partial A} = \left(R_k L_k^H\right)^T \tag{1.7}$$

Siendo:

- R_k un autovector derecho asociado al autovalor λ_k de la matriz A. R_k satisface la ecuación: $AR_k = \lambda_k R_k$.
- L_k es un autovector izquierdo asociado al autovalor λ_k de la matriz A. L_k satisface la ecuación: $L_k^H A = L_k^H \lambda_k$. Para que la expresión (1.7) sea válida L_k debe ser la

k-esima columna de la matriz $L = R^{-H}$. Siendo $R = \begin{bmatrix} R_1 & R_2 & \dots & R_n \end{bmatrix}$ la matriz de autovectores derechos de A.

A^T denota la transpuesta de la matriz A, A^H denota la transpuesta del conjugado de la matriz A y A^{-H} denota la la inversa de A^H.

$$S_k = \begin{bmatrix} \frac{\partial \lambda_k}{\partial a_{1,1}} & \frac{\partial \lambda_k}{\partial a_{1,2}} & \dots & \frac{\partial \lambda_k}{\partial a_{1,n}} \\ \frac{\partial \lambda_k}{\partial a_{2,1}} & \frac{\partial \lambda_k}{\partial a_{2,2}} & \dots & \frac{\partial \lambda_k}{\partial a_{2,n}} \\ \vdots & \dots & \ddots & \vdots \\ \frac{\partial \lambda_k}{\partial a_{n,1}} & \frac{\partial \lambda_k}{\partial a_{n,2}} & \dots & \frac{\partial \lambda_k}{\partial a_{n,n}} \end{bmatrix} \tag{1.8}$$

El elemento i, j de la matriz de sensibilidad S_k cuantifica el efecto de una pertubación en el coeficiente $a_{i,j}$ sobre el autovalor λ_k. Una variación infinitesimal en el coeficiente $a_{i,j}$, de $a_{1,2}$ a $a_{1,2} + \Delta a_{1,2}$ conlleva una variación en el autovalor de λ_1 a $\lambda_1 + \Delta\lambda_1$ siendo $\Delta\lambda_1 = S_k(i,j)\Delta a_{1,2}$.

Indice de sensibilidad

Cuando se quiere realizar una comparación de la sensibilidad autovalor coeficiente entre diferentes matrices, resulta más sencillo utilizar un índice de valor escalar. Para lo cual [4] propone el siguiente índice:

$$\sum_{k=1}^{n} \left\| \frac{\partial \lambda_k}{\partial \Phi} \right\|_F^2 \tag{1.9}$$

Si $A \in \Re^{n \times n}$, $\|A\|_F$ denota la norma de *Frobenius* de la matriz A, definida como:

$$\|A\|_F = \sqrt{\mathrm{trace}\left(\mathrm{A^H A}\right)} = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{i,j}|^2} \tag{1.10}$$

Ejemplo 1.3.2 Cálculo de la sensibilidad de los autovalores de la matriz A ante variaciones en sus coeficientes.

$$A = \begin{bmatrix} -3 & 2 \\ 10 & 5 \end{bmatrix} \tag{1.11}$$

Para encontrar los autovectores se requiere calcular los autovalores de la matriz.

$$
\begin{aligned}
|A-\lambda I| &= \left| \begin{bmatrix} -3 & 2 \\ 10 & 5 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right| \\
&= \left| \begin{bmatrix} -3-\lambda & 2 \\ 10 & 5-\lambda \end{bmatrix} \right| \\
&= -(3+\lambda)(5-\lambda) - 20 \\
&= \lambda^2 - 2\lambda - 35 \\
&= (\lambda - 7)(\lambda + 5)
\end{aligned}
$$

Se calcula el autovector derecho asociado al autovalor $\lambda_1 = 7$.

$$
\begin{aligned}
(A - 7I)\, R_1 &= 0 \\
\left(\begin{bmatrix} -3 & 2 \\ 10 & 5 \end{bmatrix} - \begin{bmatrix} 7 & 0 \\ 0 & 7 \end{bmatrix} \right) \begin{bmatrix} r_{1,1} \\ r_{1,2} \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
\begin{bmatrix} -10 & 2 \\ 10 & -2 \end{bmatrix} \begin{bmatrix} r_{1,1} \\ r_{1,2} \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\end{aligned}
$$

De donde se obtiene el siguiente sistema de ecuaciones:

$$
\begin{aligned}
-10 r_{1,1} + 2 r_{1,2} &= 0 \\
10 r_{1,1} - 2 r_{1,2} &= 0
\end{aligned}
$$

El sistema anterior tiene infinitas soluciones. Una de ellas es $r_{1,1} = 1$ y $r_{1,2} = 5$. El autovector derecho asociado al autovalor $\lambda_1 = 7$ es $R_1 = \begin{bmatrix} 1 & 5 \end{bmatrix}^T$. Con un procedimiento similar se obtiene el autovector derecho asociado al autovalor $\lambda_2 = -5$. Dicho autovector es $R_2 = \begin{bmatrix} 1 & -1 \end{bmatrix}^T$. Con R_1 y R_2 se conforma la matriz de autovectores derechos R:

$$
R = \begin{bmatrix} 1 & 1 \\ 5 & -1 \end{bmatrix}. \tag{1.12}
$$

Se calcula la matriz de autovectores izquierdos utilizando la expresión $L = R^{-H}$:

$$\begin{aligned} L &= \begin{bmatrix} 1 & 1 \\ 5 & -1 \end{bmatrix}^{-H} \\ &= \left(\frac{1}{-6} \begin{bmatrix} -1 & -5 \\ -1 & 1 \end{bmatrix}^{T} \right)^{H} \\ &= \frac{1}{-6} \begin{bmatrix} -1 & -5 \\ -1 & 1 \end{bmatrix} \\ &= \frac{1}{6} \begin{bmatrix} 1 & 5 \\ 1 & -1 \end{bmatrix} \end{aligned}$$

De donde se obtiene: $L_1 = \begin{bmatrix} 1/6 & 1/6 \end{bmatrix}^T$ y $L_2 = \begin{bmatrix} 5/6 & -1/6 \end{bmatrix}^T$.
La sensibilidad del autovalor $\lambda_1 = 7$ es:

$$\begin{aligned} \frac{\partial \lambda_1}{\partial A} &= \left(R_1 L_1^H \right)^T \\ &= \left(\begin{bmatrix} 1 \\ 5 \end{bmatrix} \begin{bmatrix} \frac{1}{6} & \frac{1}{6} \end{bmatrix} \right)^T \\ &= \frac{1}{6} \begin{bmatrix} 1 & 5 \\ 1 & 5 \end{bmatrix} \end{aligned}$$

La sensibilidad del autovalor $\lambda_2 = -5$ es:

$$\begin{aligned} \frac{\partial \lambda_2}{\partial A} &= \left(R_2 L_2^H \right)^T \\ &= \left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} \begin{bmatrix} \frac{5}{6} & -\frac{1}{6} \end{bmatrix} \right)^T \\ &= \frac{1}{6} \begin{bmatrix} 5 & -5 \\ -1 & 1 \end{bmatrix} \end{aligned}$$

Aplicando (1.9) a la matriz A, se obtiene:

$$\begin{aligned} S &= \left\| \frac{1}{6}\begin{bmatrix} 1 & 5 \\ 1 & 5 \end{bmatrix} \right\|_F^2 + \left\| \frac{1}{6}\begin{bmatrix} 5 & -5 \\ -1 & 1 \end{bmatrix} \right\|_F^2 \\ &= \left(\frac{1}{36} + \frac{25}{36} + \frac{1}{36} + \frac{25}{36} \right) + \left(\frac{25}{36} + \frac{25}{36} + \frac{1}{36} + \frac{1}{36} \right) \\ &= \frac{26}{9} \end{aligned}$$

1.3.2. Errores de redondeo

El efecto del redondeo en las operaciones se introducirá por medio del ejemplo 1.3.3:

Ejemplo 1.3.3 considere los vectores:

$$a = \begin{bmatrix} 100000000 & 1 & 100000000 \end{bmatrix}$$

$$b = \begin{bmatrix} 100000000 & 1 & -100000000 \end{bmatrix}$$

Su producto escalar es $a \cdot b = 1$. Si éste se calcula con aritmética de punto flotante de 32 bits (8 de mantisa y 23 de exponente) el resultado es cero.

Este problema se puede evitar si se cambia el orden de realizar la operación.

1.4. Transformaciones de similaridad

Considere :

$$\begin{aligned} x(kh+k) &= \Phi x(kh) + \Gamma u(kh) \\ y(kh) &= Cx(kh) + Du(kh). \end{aligned}$$

Si T es una matriz no singular, y partir de esta se genera un nuevo vector de estados aplicando la transformación lineal $w = Tx$, se obtiene el siguiente modelo en espacio de estados:

$$T^{-1}w(kh+k) = \Phi T^{-1}w(kh) + \Gamma u(kh)$$
$$y(kh) = CT^{-1}w(kh) + Du(kh)$$

Despejando $w(kh+h)$,

$$w(kh+k) = T\Phi T^{-1}w(kh) + T\Gamma u(kh)$$
$$y(kh) = CT^{-1}w(kh) + Du(kh)$$

El nuevo sistema en espacio de estados es similar al sistema original en el sentido de que tienen los mismos autovalores y la misma relación entrada salida [2], [4]. En adelante las matrices del sistema transformado se denotarán utilizando el subindice T, así:

$$w(kh+k) = \Phi_T w(kh) + \Gamma_T u(kh)$$
$$y(kh) = C_T w(kh) + D_T u(kh)$$

Siendo:

$$\Phi_T = T\Phi T^{-1}$$
$$\Gamma_T = T\Gamma$$
$$C_T = CT^{-1}$$
$$D_T = D$$

1.4.1. Invarianza de los autovalores

A continuación se mostrará que los autovalores de la matriz de transición de estado permanecen invariantes cuando se aplica una transformación de similaridad.

$$
\begin{aligned}
|T\Phi T^{-1} - \lambda I| &= |T\Phi T^{-1} - \lambda I T T^{-1}| \\
&= |T\Phi T^{-1} - \lambda T I T^{-1}| \\
&= |T\left(\Phi - \lambda I\right) T^{-1}| \\
&= |T| \times |\Phi - \lambda I| \times |T^{-1}|
\end{aligned}
$$

Dado que $|T^{-1}| = \frac{1}{|T|}$ se tiene:

$$
|T\Phi T^{-1} - \lambda I| = |\Phi - \lambda I| \tag{1.13}
$$

Lo anterior indica que el polinomio caraterístico de la matriz Φ no cambia al aplicar la transformación T, y por tanto sus raices, que son los autovalores de Φ, tampoco varian [4], [2].

1.4.2. Invarianza de la función de transferencia

Al igual que los autovalores la funcón de transferencia permanece invariante al aplicar una transformación de similaridad. Si se aplica transformada Z al modelo :

$$
\begin{aligned}
w(kh + k) &= T\Phi T^{-1} w(kh) + T\Gamma u(kh) \\
y(kh) &= CT^{-1} w(kh) + Du(kh)
\end{aligned}
$$

Se obtiene:

$$
\begin{aligned}
Z\{w(kh + k)\} &= T\Phi T^{-1} Z\{w(kh)\} + T\Gamma Z\{u(kh)\} \\
Z\{y(kh)\} &= CT^{-1} Z\{w(kh)\} + DZ\{u(kh)\}
\end{aligned}
$$

$$Z\{w(kh+k)\} = T\Phi T^{-1} Z\{w(kh)\} + T\Gamma Z\{u(kh)\}$$
$$Z\{y(kh)\} = CT^{-1} Z\{w(kh)\} + DZ\{u(kh)\}$$

$$zW(z) = T\Phi T^{-1} W(z) + T\Gamma U(z)$$
$$Y(z) = CT^{-1} W(z) + DU(z)$$

$$W(z) = \left(zI - T\Phi T^{-1}\right)^{-1} T\Gamma U(z)$$
$$Y(z) = CT^{-1} W(z) + DU(z)$$

Se despeja $W(z)$ para obtener la relación entrada salida del sistema.

$$\begin{aligned}
Y(z) &= CT^{-1}\left(zI - T\Phi T^{-1}\right)^{-1} T\Gamma U(z) + DU(z) \\
&= CT^{-1}\left(zTIT^{-1} - T\Phi T^{-1}\right)^{-1} T\Gamma U(z) + DU(z) \\
&= CT^{-1}\left(T\left(zI - \Phi\right)T^{-1}\right)^{-1} T\Gamma U(z) + DU(z) \\
&= CT^{-1}\left(T\left(zI - \Phi\right)^{-1} T^{-1}\right) T\Gamma U(z) + DU(z) \\
&= CT^{-1} T\left(zI - \Phi\right)^{-1} T^{-1} T\Gamma U(z) + DU(z) \\
&= C\left(zI - \Phi\right)^{-1} \Gamma U(z) + DU(z)
\end{aligned}$$

Es evidente entonces que la relación entrada salida del sistema es independiente de la transformación de similaridad T. De lo cual se concluye que una función de transferencia tiene múltiples representaciones en espacio de estados [2], [4].

1.4.3. Transformación de los autovectores

Cuando se aplica una transformación lineal a un modelo en espacio de estados los valores propios de la matriz de transición permanecen invariantes, sin embargo los

vectores propios no. Los nuevos vectores propios pueden calcularse directamente de la matriz transformada, o bien en términos de los vectores propios de la matriz original y la transformación [4].

Transformación de los vectores propios derechos

Si R_k es un vector propio derecho de la matriz Φ asociado con el valor propio λ_k, este satisface:

$$\Phi R_k = \lambda_k R_k \tag{1.14}$$

Se reemplaza Φ por $T^{-1}\Phi_T T$

$$\begin{aligned} T^{-1}\Phi_T T R_k &= \lambda_k R_k \\ \Phi_T (T R_k) &= \lambda_k (T R_k) \end{aligned}$$

La ecuación anterior indica que TR_k es un vector propio derecho de Φ_T.

Transformación de los vectores propios izquierdos

Si L_k es un vector propio izquierdo de la matriz Φ, asociado con el valor propio λ_k, este satisface:

$$L_k^H \Phi = \lambda_k L_k^H \tag{1.15}$$

Se reemplaza Φ por $T^{-1}\Phi_T T$

$$\begin{aligned} L_k^H T^{-1}\Phi_T T &= \lambda_k L_k^H \\ \left(L_k^H T^{-1}\right)\Phi_T &= \lambda_k \left(L_k^H T^{-1}\right) \\ \left(T^{-H} L_k\right)^H \Phi_T &= \lambda_k \left(T^{-H} L_k\right)^H \end{aligned}$$

La ecuación anterior indica que $T^{-H}L_k$ es un vector propio izquierdo de Φ_T.

El hecho de que los autovectores no permanezcan invariantes al aplicar una transformación de similaridad indica que la sensibilidad de los autovalores con respecto a perturbaciones en los coeficientes depende de la representación en espacio de estados utilizada [2], [4].

El anterior resultado es el eje central de este trabajo.

Ejemplo 1.4.1 En este ejemplo se muestra como cambia la sensibilidad autovalor coeficiente al aplicar una transformación de similaridad.

$$\Phi = \begin{bmatrix} 0 & 0 & 0 & -24 \\ 1 & 0 & 0 & 50 \\ 0 & 1 & 0 & -35 \\ 0 & 0 & 1 & 10 \end{bmatrix} \tag{1.16}$$

Cuyo indice de sensibilidad es $527{,}7634$: Si se aplica la transformación de similaridad definida por:

$$T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 4 & 16 & 64 \end{bmatrix} \tag{1.17}$$

Se obtiene que $\Phi_T = TAT^{-1}$ es:

$$\Phi_T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix} \tag{1.18}$$

Cuyo indice de sensibilidad es: 4. Los índices de sensibilidad se calcularon utilizando el programa mostrado en el listado1.4.1 [2].

Listado 1.4.1 Cálculo del índice (1.9).

```
% function [S]    = msensitivity(A)
% % Matrix sensitivity. SENSI(A)  measures eigenvalue
% % coeficient sensitivity.
% %
% %  S   =  sum || (Rk * Lk^H )^T ||
% %                                  frobenius
% % Being:
% %
% %  d lk
% % ----- = (Rk * Lk^H )^T
% %  dA
% %
% % lk : Eigenvalue.
% % Rk : Right eigenvector.
% % Lk : Left eigenvector.
% A = sym(A);
% [R,D] = eig(A);
% if double(rank(A)) < double(length(A)),
%   error('Right eigenvector are not linearly independent', ...
%     'You can not apply this index to the given matrix ');
% end
% L = inv(R)';
% S = 0;
% for i = 1:length(A),
%         S = S + norm( double ( R(:,i) * L(:,i)' ) , 'fro' )^2;
% end
%
```

1.5. Realización de controladores de tiempo discreto

Todo procedimiento computacional que permita determinar la ley de control a partir de las entradas del controlador constituyen una realización [7].

Para ilustrar los métodos de cálculo de las diferentes realizaciones se tomará, como base el controlador definido por:

$$G_c(q) = \frac{n_4 q^4 + n_3 q^3 + n_2 q^2 + n_1 q + n_0}{q^4 + d_3 q^3 + d_2 q^2 + d_1 q + d_0}, \tag{1.19}$$

Este se supone que opera en el sistema en lazo cerrado de la figura1.1. El sistema dado

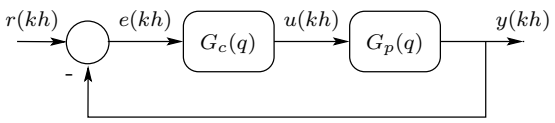

Figura 1.1: Sistema en lazo cerrado.

por (1.19) tiene como entrada la señal de error $e(kh)$ y como salida a $u(kh)$. Es decir, la ley de control se obtiene de:

$$u(kh) = G_c(q)e(kh). \tag{1.20}$$

Donde q es el operador de desplazamiento; tal que: $qx(kh) = x(kh + h)$.

1.5.1. Realización directa.

La realización directa, como su nombre lo indica se obtiene de la función de transferencia del controlador; A partir de (1.20) se tiene:

$$\begin{aligned} u(kh) =& \frac{n_4q^4 + n_3q^3 + n_2q^2 + n_1q + n_0}{q^4 + d_3q^3 + d_2q^2 + d_1q + d_0}e(kh) \\ =& \frac{n_4 + n_3q^{-1} + n_2q^{-2} + n_1q^{-3} + n_0q^{-4}}{1 + d_3q^{-1} + d_2q^{-2} + d_1q^{-3} + d_0q^{-4}}e(kh), \end{aligned}$$

reorganizando se transforma en:

$$\begin{aligned} u(kh)\big(1 + d_3q^{-1} + d_2q^{-2} + d_1q^{-3} + d_0q^{-4}\big) = \\ \big(n_4 + n_3q^{-1} + n_2q^{-2} + n_1q^{-3} + n_0q^{-4}\big)e(kh), \end{aligned}$$

con la cual, resolviendo para $u(kh)$ la ley de control es:

$$\begin{aligned} u(kh) =& -d_3u(kh - h) - d_2u(kh - 2h) - d_1u(kh - 3h) - d_0u(kh - 4h) \\ &+ n_4e(kh) + n_3e(kh - h) + n_2e(kh - 2h) + n_1e(kh - 3h) + n_0e(kh - 4h) \end{aligned} \tag{1.21}$$

En la figura 1.2, se muestra el diagrama de la realización directa del controlador (1.19); del cual se observa que la ley de control es obtenida a partir de valores ponderados de

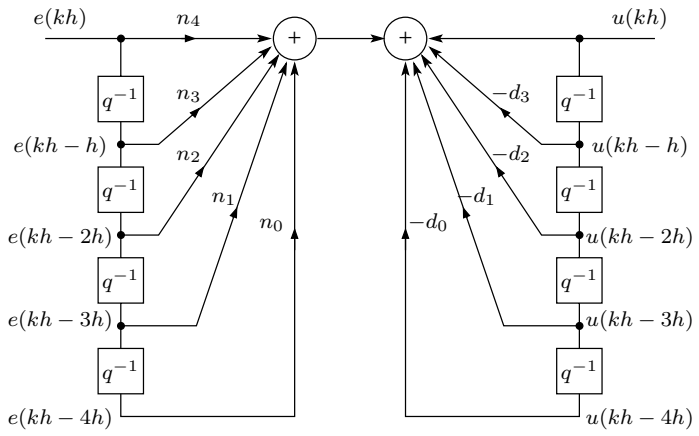

Figura 1.2: Sistema en lazo cerrado.

Listado 1.5.1 Realización directa de un controlador 1DOF.

```
%double  ley_control(double ek)
%{
%        static double  ek1 = 0, ek2 = 0, ek3 = 0, ek4 = 0;
%        static double  uk1 = 0, uk2 = 0, uk3 = 0, uk4 = 0;
%
%        // Ecuacion de salida
%        uk      = -d3*uk1 -d2*uk2 -d1*uk3 -d0uk4 +
%                     n4*ek +n3*ek1 +n2*ek2 +n1*ek3 +n0ek4;
%
%        // Actualizacion de variables.
%        ek4     = ek3;
%        ek3     = ek2;
%        ek2     = ek1;
%        ek1     = ek;
%
%        uk4     = uk3;
%        uk3     = uk2;
%        uk2     = uk1;
%        uk1     = uk;
%
%return uk;
%}
%
```

salidas y entradas anteriores. el listado 1.5.1 muestra el algoritmo en lenguaje C de la ecuación (1.21).

Las variables `ek1, ek2, ek3` y `ek4` denotan respectivamente los instantes $e(kh-h)$,

$e(kh-2h)$, $e(kh-3h)$ y $e(kh-4h)$ de la señal $e(kh)$. Así mismo, `uk1, uk2, uk3` y `uk4` denotan respectivamente los instantes $u(kh-h)$, $u(kh-2h)$, $u(kh-3h)$ y $u(kh-4h)$ de la señal $u(kh)$. Las constantes del controlador ($n_0, \ldots, n_4$ y $d_0, \ldots, d_3$) se asume que se declararán como variables globales. Esta realización depende directamente del modelo dinámico del controlador, esto lo hace extremadamente sensible a errores computacionales si es de alto orden.

1.5.2. Realización canónica.

Para obtener esta realización, (1.19) se representa como:

$$G_c(q) = n_4 + \frac{a_3q^3 + a_2q^2 + a_1q + a_0}{q^4 + d_3q^3 + d_2q^2 + d_1q + d_0}, \tag{1.22}$$

donde $a_i = n_i - n_4d_i$. El objetivo de esta factorización es que la función de transferencia sea estrictamente propia, si esto no se cumple el procedimiento aquí descrito no será aplicable. Nuevamente de (1.20),

$$u(kh) = n_4e(kh) + \frac{a_3q^3 + a_2q^2 + a_1q + a_0}{q^4 + d_3q^3 + d_2q^2 + d_1q + d_0}e(kh) \tag{1.23}$$

representa la ley de control, la cual se reescribe como:

$$u(kh) = \left(c_3q^3 + c_2q^2 + c_1q + c_0\right)x(kh) + n_4e(kh), \tag{1.24}$$

donde

$$x(kh) = \frac{1}{q^4 + d_3q^3 + d_2q^2 + d_1q + d_0}e(kh) \tag{1.25}$$

es una variable auxiliar a partir de la cual se generarán los estados del sistema. Representando $x(kh)$ como la ecuación en diferencias

$$\begin{aligned} e(kh) &= \left(q^4 + d_3q^3 + d_2q^2 + d_1q + d_0\right)x(kh) \\ &= x(kh+4h) + d_3x(kh+3h) + d_2x(kh+2h) + d_1x(kh+h) + d_0x(kh) \end{aligned}$$

y resolviendo para $x(kh+4h)$ se tiene:

$$x(kh+4h) = -d_3x(kh+3h) - d_2x(kh+2h) - d_1x(kh+h) - d_0x(kh) + e(kh). \tag{1.26}$$

Para la selección de los estados se procede de la siguiente manera. Se definen las variables

$$\begin{aligned} x_1(kh) &= x(kh) \\ x_2(kh) &= x(kh+h) \\ x_3(kh) &= x(kh+2h) \\ x_4(kh) &= x(kh+3h), \end{aligned} \tag{1.27}$$

las cuales, desplazadas un instante se convierten en:

$$\begin{aligned} x_1(kh+h) &= x(kh+h) \\ x_2(kh+h) &= x(kh+2h) \\ x_3(kh+h) &= x(kh+3h) \\ x_4(kh+h) &= x(kh+4h). \end{aligned} \tag{1.28}$$

Ahora, de (1.28), (1.27) y (1.26) se obtiene

$$\begin{aligned} x_1(kh+h) &= x_2(kh) \\ x_2(kh+h) &= x_3(kh) \\ x_3(kh+h) &= x_4(kh) \\ x_4(kh+h) &= -d_0x_1(kh) - d_1x_2(kh) - d_2x_3(kh) - d_3x_4(kh) + e(kh), \end{aligned} \tag{1.29}$$

que expresada en forma matricial queda:

$$\begin{bmatrix} x_1(kh+h) \\ x_2(kh+h) \\ x_3(kh+h) \\ x_4(kh+h) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -d_0 & -d_1 & -d_2 & -d_3 \end{bmatrix} \begin{bmatrix} x_1(kh) \\ x_2(kh) \\ x_3(kh) \\ x_4(kh) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} e(kh). \tag{1.30}$$

Se representa la ley de control (1.24) en términos de los estados obtenidos:

$$\begin{aligned} u(kh) &= \big(a_3q^3 + a_2q^2 + a_1q + a_0\big)x(kh) + n_4e(kh) \\ &= a_3x(kh+3h) + a_2x(kh+2h) + a_1x(kh+h) + a_0x(kh) + n_4e(kh) \end{aligned} \tag{1.31}$$

y reemplazando (1.26) se obtiene:

$$u(kh) = a_0x_1(kh) + a_1x_2(kh) + a_2x_3(kh) + a_3x_4(kh) + n_4e(kh). \tag{1.32}$$

Reescribiendo en forma matricial la ley de control es:

$$u(kh) = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} x_1(kh) \\ x_2(kh) \\ x_3(kh) \\ x_4(kh) \end{bmatrix} + n_4 e(kh). \tag{1.33}$$

Del modelo en espacio de estados descrito por las ecuaciones (1.30) y (1.33) se obtiene la función en lenguaje C mostrada en el listado 1.5.2.

Listado 1.5.2 Realización canónica de un controlador 1DOF.

```
double  ley_control(double ek)
{
        static double  x1 = 0, x2 = 0, x3 = 0, x4 = 0;
               double  x1s, x2s, x3s, x4s, uk;

        // Ecuaciones de estado.
        x1s     = x2;
        x2s     = x3;
        x3s     = x4;
        x4s     = -d0*x1 -d1*x2 -d2*x3 -d3*x4 +ek;

        // Ecuacion de salida.
        uk      = a0*x1 + a1*x2 + a2*x3 + a3*x4 + n4*ek;

        // Actualizacion de variables.
        x4      = x4s;
        x3      = x3s;
        x2      = x2s;
        x1      = x1s;

return uk;
}
```

Note que la ecuación (1.30) corresponde a la denominada *forma canónica observable* de un modelo en espacio de estados. Esta forma canónica es obtenida a traves de la

transformación de similaridad $z = T_o x$. Donde

$$T_o = \begin{bmatrix} C \\ C\Phi \\ C\Phi^2 \\ \cdots \\ C\Phi^{n-1} \end{bmatrix}, \tag{1.34}$$

corresponde a la *matríz de observabilidad* del sistema.

1.5.3. Realización serie.

Consiste en descomponer un controlador de alto orden en módulos de primer y segundo grado; los cuales se ejecutan secuencialmente. Para esta realización, el denominador del controlador se supondrá como un polinomio con dos raices reales y distintas que se denotarán como r_1 y r_2 y dos raices complejas conjugadas que se denotadas por r_3 y r_4, así el controlador

$$G_c(q) = k_c \frac{(q-c_1)(q-c_2)(q-c_3)(q-c_4)}{(q-r_1)(q-r_2)(q-r_3)(q-r_4)} \tag{1.35}$$

puede ser expresado como el producto de dos funciones de transferencia de primer orden asociadas a los polos reales r_1 y r_2, y a una función de transferencia de segundo orden asociada a los polos complejos conjugados r_3 y r_4, es decir,

$$G_c(q) = k_c G_1(q) G_2(q) G_3(q), \tag{1.36}$$

donde:

$$\begin{aligned} G_1(q) &= \frac{q-c_1}{q-r_1} \\ G_2(q) &= \frac{(q-c_3)(q-c_4)}{(q-r_3)(q-r_4)} \\ G_3(q) &= \frac{q-c_2}{q-r_2}. \end{aligned}$$

Modelo en espacio de estados del componente $G_1(q)$.

La señal $e(kh)$ de entrada al controlador es también la señal de entrada a $G_1(q)$, y su salida se denotará como $u_1(kh)$, que constituirá la entrada de $G_2(q)$. Por lo tanto:

$$\begin{aligned} G_1(q) &= \frac{u_1(kh)}{e(kh)} \\ &= \frac{q - c_1}{q - r_1} \\ &= 1 + \frac{r_1 - c_1}{q - r_1}, \end{aligned} \tag{1.37}$$

resolviendo para $u_1(kh)$, la salida del bloque $G_1(q)$ es:

$$\begin{aligned} u_1(kh) &= e(kh) + \frac{r_1 - c_1}{q - r_1} e(kh) \\ &= e(kh) + x(kh) \end{aligned} \tag{1.38}$$

con

$$x(kh) = \frac{r_1 - c_1}{q - r_1} e(kh). \tag{1.39}$$

Reorganizando (1.38) y (1.39), el modelo en espacio de estados del componente $G_1(s)$ es:

$$x(kh + h) = r_1 x(kh) + (r_1 - c_1)\, e(kh) \tag{1.40}$$

$$u_1(kh) = x(kh) + e(kh). \tag{1.41}$$

Del modelo en espacio de estado descrito por (Ec.1.40) se obtiene la función en lenguaje C mostrada en el listado 1.5.3.

Listado 1.5.3 Realización serie de un controlador 1DOF (Módulo I).

```
double  modulo_I(double ek)
{
        static double  x1 = 0;
               double  x1s, u1k;

        // Ecuaciones de estado.
        x1s     = r1*x1 + (r1-c1)*ek;

        // Ecuacion de salida.
        u1k     = x1 + ek;

        // Actualizacion de variables.
        x1      = x1s;

return u1k;
}
```

Modelo en espacio de estados del componente del componente $G_2(q)$.

La señal $u_1(kh)$ es la señal de entrada a $G_2(q)$, y La salida se denotará como $u_2(kh)$, que constituirá la entrada de $G_3(q)$. Por lo tanto:

$$\begin{aligned} G_2(q) &= \frac{u_2(kh)}{u_1(kh)} \\ &= \frac{(q-c_3)(q-c_4)}{(q-r_3)(q-r_4)} \\ &= 1 + \frac{(r_3+r_4-c_3-c_4)\,q + (c_3c_4 - r_3r_4)}{q^2 - (r_3+r_4)\,q + r_3r_4}, \end{aligned} \tag{1.42}$$

resolviendo para $u_2(kh)$, la salida del bloque $G_2(q)$ es:

$$\begin{aligned} u_2(kh) &= u_1(kh) + \frac{(r_3+r_4-c_3-c_4)\,q + (c_3c_4 - r_3r_4)}{q^2 - (r_3+r_4)\,q + r_3r_4} u_1(kh) \\ &= u_1(kh) + \Big[(r_3+r_4-c_3-c_4)\,q + (c_3c_4 - r_3r_4)\Big] x(kh) \end{aligned} \tag{1.43}$$

donde:

$$x(kh) = \frac{1}{q^2 - (r_3+r_4)\,q + r_3r_4} u_1(kh), \tag{1.44}$$

con la cual, luego de ser reorganizada se tiene:

$$x(kh+2h) = (r_3 + r_4)\, x(kh+h) - r_3 r_4 x(kh) + u_1(kh). \tag{1.45}$$

Se seleccionan como variables de estado $x_1(kh) = x(kh)$ y $x_2(kh) = x(kh+h)$. Reemplazando kh por $kh+h$, se tienen por ecuaciones de estado:

$$\begin{aligned} x_1(kh+h) &= x(kh+h) \\ &= x_2(kh), \end{aligned} \tag{1.46}$$

y de (1.45)

$$\begin{aligned} x_2(kh+h) &= x(kh+2h) \\ &= -r_3 r_4 x_1(kh) + (r_3 + r_4)\, x_2(kh) + u_1(kh). \end{aligned} \tag{1.47}$$

Ahora, retomando la ecuación de salida (1.43), y de (1.46) se tiene:

$$\begin{aligned} u_2(kh) &= (c_3 c_4 - r_3 r_4)\, x(kh) + (r_3 + r_4 - c_3 - c_4)\, x(kh+h) + u_1(kh) \\ &= (c_3 c_4 - r_3 r_4)\, x_1(kh) + (r_3 + r_4 - c_3 - c_4)\, x_2(kh) + u_1(kh), \end{aligned} \tag{1.48}$$

reagrupando, finalmente se obtiene la representación en espacio de estados

$$\begin{aligned} \begin{bmatrix} x_1(kh+h) \\ x_2(kh+h) \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ -r_3 r_4 & r_3 + r_4 \end{bmatrix} \begin{bmatrix} x_1(kh) \\ x_2(kh)s \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_1(kh) \\ u_2(kh) &= \begin{bmatrix} c_3 c_4 - r_3 r_4 & r_3 + r_4 - c_3 - c_4 \end{bmatrix} \begin{bmatrix} x_1(kh) \\ x_2(kh) \end{bmatrix} + u_1(kh). \end{aligned} \tag{1.49}$$

Del modelo en espacio de estado descrito por (Ec.1.49) se obtiene la función en lenguaje C mostrada en el listado 1.5.4.

Listado 1.5.4 Realización serie de un controlador 1DOF (Módulo II).

```
double  modulo_II(double u1k)
{
        static double  x1 = 0, x2 = 0;
               double  x1s, x2s, u2k;

        // Ecuaciones de estado.
        x1s     = x2;
        x2s     = -r_3*r_4*x1 + (r_3+r_4)*x2 + u1k;

        // Ecuacion de salida.
        u2k     = (c_3c_4-r_3r_4)*x1 + (r_3+r_4-c_3-c_4)*x2 + u1k;

        // Actualizacion de variables.
        x1      = x1s;
        x2      = x2s;

return u2k;
}
```

Modelo en espacio de estados del componente $G_3(q)$

Con un procedimiento similar al utilizado para $G_1(q)$ se obtiene la representación en espacio de estados de $G_3(q)$

$$\begin{aligned} x(kh+h) &= r_2 x(kh) + (r_2 - c_2)\, u_2(kh) \\ u_3(kh) &= x(kh) + u_2(kh), \end{aligned} \tag{1.50}$$

cuya función de implementación en lenguaje C es mostrada en el listado 1.5.5.
Para la integración de los componentes el problema se reduce a calcular la salida de cada uno de los módulos e incertarla en su sucesor, este procedimiento es resumido en el listado 1.5.6.

Listado 1.5.5 Realización serie de un controlador 1DOF (Módulo III).

```
double  modulo_III(double u2k)
{
        static double  x1 = 0;
               double  x1s, u3k;

        // Ecuaciones de estado.
        x1s     = r2*x1 + (r2-c2)*u2k;

        // Ecuacion de salida.
        u3k     = x1 + u2k;

        // Actualizacion de variables.
        x1      = x1s;

return u3k;
}
```

Listado 1.5.6 Realización serie de un controlador 1DOF (Integración Componentes).

```
double  controlador_RST(double ek)
{
         double  uk, u1k, u2k, u3k;

        // Calculo de la ley de control.
        u1k     = modulo_I( ek );
        u2k     = modulo_II( u1k );
        u3k     = modulo_III( u2k );
        uk      = kc * u3k;
return uk;
}
```

Integración de los componentes de la realización serie

Con el fin de obtener un modelo en espacio de estados que incluya: el estado de $G_1(q)$, los dos estados de $G_2(q)$ y el estado de $G_3(q)$ se encontrará una expresión general que combine en un solo modelo el producto de las ecuaciones (Ec.1.51), (Ec.1.52) y (Ec.1.53).

$$\begin{aligned} z_1(kh+h) &= A_1 z_1(kh) + B_1 e(kh) \\ u_1(kh) &= C_1 z_1(kh) + D_1 e(kh) \end{aligned} \tag{1.51}$$

$$\begin{aligned} z_2(kh+h) &= A_2 z_2(kh) + B_2 u_1(kh) \\ u_2(kh) &= C_2 z_2(kh) + D_2 u_1(kh) \end{aligned} \tag{1.52}$$

$$\begin{aligned} z_3(kh+h) &= A_3 z_3(kh) + B_3 u_2(kh) \\ u_3(kh) &= C_3 z_3(kh) + D_3 u_2(kh) \end{aligned} \tag{1.53}$$

Se elimina la variable $u_1(kh)$ de los modelos (Ec.1.51) y (Ec.1.52) y se obtiene:

$$\begin{aligned} z_2(kh+h) &= A_2 z_2(kh) + B_2 \left(C_1 z_1(kh) + D_1 e(kh)\right) \\ &= B_2 C_1 z_1(kh) + A_2 z_2(kh) + B_2 D_1 e(kh) \\ u_2(kh) &= C_2 z_2(kh) + D_2 \left(C_1 z_1(kh) + D_1 e(kh)\right) \\ &= D_2 C_1 z_1(kh) + C_2 z_2(kh) + D_2 D_1 e(kh) \end{aligned} \tag{1.54}$$

Se elimina la variable $u_2(kh)$ de los modelos (Ec.1.54) y (Ec.1.53) obteniendo:

$$\begin{aligned} z_3(kh+h) &= A_3 z_3(kh) + B_3 \left(D_2 C_1 z_1(kh) + C_2 z_2(kh) + D_2 D_1 e(kh)\right) \\ &= B_3 D_2 C_1 z_1(kh) + B_3 C_2 z_2(kh) + A_3 z_3(kh) + B_3 D_2 D_1 e(kh) \\ u_3(kh) &= C_3 z_3(kh) + D_3 \left(D_2 C_1 z_1(kh) + C_2 z_2(kh) + D_2 D_1 e(kh)\right) \\ &= D_3 D_2 C_1 z_1(kh) + D_3 C_2 z_2(kh) + C_3 z_3(kh) + D_3 D_2 D_1 e(kh) \end{aligned}$$

De lo anterior se obtienen las siguientes ecuaciones de estado:

$$\begin{aligned} z_1(kh+h) &= A_1 z_1(kh) + B_1 e(kh) \\ z_2(kh+h) &= B_2 C_1 z_1(kh) + A_2 z_2(kh) + B_2 D_1 e(kh) \\ z_3(kh+h) &= B_3 D_2 C_1 z_1(kh) + B_3 C_2 z_2(kh) + A_3 z_3(kh) + B_3 D_2 D_1 e(kh) \end{aligned} \tag{1.55}$$

Se reescribe la ecuación anterior en notación matricial,

$$\begin{bmatrix} z_1(kh+h) \\ z_2(kh+h) \\ z_3(kh+h) \end{bmatrix} = \begin{bmatrix} A_1 & 0 & 0 \\ B_2C_1 & A_2 & 0 \\ B_3D_2C_1 & B_3C_2 & A_3 \end{bmatrix} \begin{bmatrix} z_1(kh+h) \\ z_2(kh+h) \\ z_3(kh+h) \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2D_1 \\ B_3D_2D_1 \end{bmatrix} e(kh) \tag{1.56}$$

Se expresa $u_3(kh)$ en notación matricial.

$$u_3(kh) = \begin{bmatrix} D_3D_2C_1 & D_3C_2 & C_3 \end{bmatrix} \begin{bmatrix} z_1(kh) \\ z_2(kh) \\ z_3(kh) \end{bmatrix} + D_3D_2D_1e(kh) \tag{1.57}$$

Para obtener un modelo en espacio de estados de la realización serie se reemplazan A_i, B_i, C_i, D_i y z_i en (Ec.1.56) y (Ec.1.57). De este reemplazo se obtienen (Ec.1.58) y Ec.1.59).

$$\begin{bmatrix} v_1(kh+h) \\ v_2(kh+h) \\ v_3(kh+h) \\ v_4(kh+h) \end{bmatrix} = \begin{bmatrix} r_1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & -r_3r_4 & r_3+r_4 & 0 \\ r_2-c_2 & \lambda_1 & \lambda_2 & r_2 \end{bmatrix} \begin{bmatrix} v_1(kh) \\ v_2(kh) \\ v_3(kh) \\ v_4(kh) \end{bmatrix} + \begin{bmatrix} r_1-c_1 \\ 0 \\ 1 \\ r_2-c_2 \end{bmatrix} e(kh) \tag{1.58}$$

Siendo:

$$\lambda_1 = (r_2 - c_2)(c_3c_4 - r_3r_4)$$
$$\lambda_2 = (r_2 - c_2)(r_3 + r_4 - c_3 - c_4)$$

$$u_3(kh) = \begin{bmatrix} 1 & c_3c_4 - r_3r_4 & r_3 + r_4 - c_3 - c_4 & 1 \end{bmatrix} \begin{bmatrix} v_1(kh) \\ v_2(kh) \\ v_3(kh) \\ v_4(kh) \end{bmatrix} + e(kh) \tag{1.59}$$

Del modelo en espacio de estado descrito por las ecuaciones (Ec.1.58) y Ec.1.59) se obtiene la función en lenguaje C mostrada en el listado 1.5.7.

Listado 1.5.7 Realización serie de un controlador 1DOF.

```
#define L1  (r2-c2)*(c3*c4-r3*r4)
#define L2  (r2-c2)*(r3+r4-c3-c4)
double  ley_control(double ek)
{
        static double  v1 = 0, v2 = 0, v3 = 0, v4 = 0;
               double  v1s, v2s, v3s, v4s, uk;

        // Ecuaciones de estado.
        v1s     = r1*v1 + (r1-c1)*ekh;
        v2s     = v3;
        v3s     = v1 -r3*r4*v2 + (r3+r4)*v3 + ekh;
        v4s     = (r2-c2)*v1 + L1*v2 + L2*v3 + r2*v4 + (r2-c2)*ekh;

        // Ecuacion de salida.
        uk      = v1 + (c3*c4-r3*r4)*v2 + (r3+r4-c3-c4)*v3 + v4 + ek;

        // Actualizacion de variables.
        v4      = v4s;
        v3      = v3s;
        v2      = v2s;
        v1      = v1s;

return uk;
}
```

La realización serie a diferencia de las realizaciones anteriores tiene como ventaja que los coeficientes del programa se escriben directamente en términos de los valores numéricos de los ceros y los polos del controlador [2].

1.6. Realización diagonal o paralela

Para encontrar una realización paralela la función de transferencia del controlador se debe expresar como una suma de funciones de transferencia de primer y segundo orden. En el caso del controlador (Ec.1.35) se obtienen dos funciones de transferencia de primer orden que corresponden a los polos reales r_1 y r_2, y una función de transferencia de segundo orden que corresponden a los polos complejos conjugados r_3 y r_4. A continuación se muestra la factorización del controlador (Ec.1.35):

$$G(q) = n_4 + \frac{\alpha_1}{q - r_1} + \frac{\alpha_2 q + \alpha_3}{(q - r_3)(q - r_4)} + \frac{\alpha_4}{q - r_2}.$$

Mediante una expansión en fracciones parciales se expresa $G(q)$ como $G_1(q) + G_2(q) + G_3(q)$.

$$\begin{aligned} u(kh) &= G(q)e(kh) \\ &= k_\alpha e(kh) + G_1(q)e(kh) + G_2(q)e(kh) + G_3(kh)e(kh) \\ &= u_0(kh) + u_1(kh) + u_2(kh) + u_3(kh) \end{aligned}$$

Siendo:

- $$u_0(kh) = n_4 e(kh)$$

- $$\begin{aligned} u_1(kh) &= G_1(q)e(kh) \\ &= \frac{\alpha_1}{q - r_1} e(kh) \end{aligned}$$

- $$u_2(kh) = G_2(q)e(kh) = \frac{\alpha_2 q + \alpha_3}{(q - r_3)(q - r_4)}e(kh)$$

- $$u_3(kh) = G_3(q)e(kh) = \frac{\alpha_4}{q - r_2}e(kh)$$

Cada una de las anteriores funciones de transferencia se convertira en un modelo en espacio de estados.

1.6.1. Modelo en espacio de estados del componente $G_1(q)$

$$\begin{aligned} u_1(kh) &= G_1(q)e(kh) \\ &= \frac{\alpha_1}{q - r_1}e(kh) \\ &= \alpha_1 x(kh) \end{aligned}$$

Siendo $x(kh)$ una variable auxiliar definida por (Ec.1.60), a partir de la cual se encuentra la ecuación de estado de $G_1(q)$:

$$\begin{aligned} x(kh) &= \frac{1}{q - r_1}e(kh) \\ (q - r_1)\, x(kh) &= e(kh) \\ x(kh + h) - r_1 x(kh) &= e(kh) \\ x(kh + h) &= r_1 x(kh) + e(kh) \end{aligned} \tag{1.60}$$

Equivalente a:

$$\begin{aligned} x(kh+h) &= r_1 x(kh) + e(kh) \\ u_1(kh) &= \alpha_1 x(kh) \end{aligned} \tag{1.61}$$

Del modelo anterior se obtiene la función en lenguaje C mostrada en el listado 1.6.1.

Listado 1.6.1 Realización paralelo de un controlador 1DOF (Modulo I).

```
double  modulo_I(double ek)
{       static double  x1 = 0;
        double  x1s, u1k;
        // Ecuaciones de estado.
        x1s     = r1*x1 + ek;
        // Ecuacion de salida.
        u1k     = alpha1*x1;
        // Actualizacion de variables.
        x1      = x1s;
return u1k;
}
```

1.6.2. Modelo en espacio de estados del componente $G_2(q)$

$$\begin{aligned} u_2(kh) &= G_2(q)e(kh) \\ &= \frac{\alpha_2 q + \alpha_3}{(q-r_3)(q-r_4)} e(kh) \\ &= \frac{\alpha_2 q + \alpha_3}{q^2 - (r_3+r_4)\,q + r_3 r_4} e(kh) \\ &= (\alpha_2 q + \alpha_3)\, x(kh) \end{aligned}$$

A partir de la variable auxiliar $x(kh)$ definida a continuación se encuentra la ecuación

de estado de $G_2(q)$:

$$x(kh) = \frac{1}{q^2 - (r_3 + r_4)\, q + r_3 r_4} u_1(kh)$$

$$q^2 x(kh) - (r_3 + r_4)\, q x(kh) + r_3 r_4 x(kh) = u_1(kh)$$

$$x(kh + 2h) - (r_3 + r_4)\, x(kh + h) + r_3 r_4 x(kh) = u_1(kh)$$

Se despeja $x(kh + 2h)$.

$$x(kh + 2h) = (r_3 + r_4)\, x(kh + h) - r_3 r_4 x(kh) + u_1(kh)$$

Se escogen como variables de estado: $x_1(kh) = x(kh)$ y $x_2(kh) = x(kh+h)$. y se obtiene las siguientes ecuaciones de estado:

$$\begin{aligned}
x_1(kh + h) &= x(kh + h) \\
&= x_2(kh) \\
x_2(kh + h) &= x(kh + 2h) \\
&= (r_3 + r_4)\, x(kh + h) - r_3 r_4 x(kh) + u_1(kh) \\
&= (r_3 + r_4)\, x_2(kh) - r_3 r_4 x_1(kh) + u_1(kh) \\
&= -r_3 r_4 x_1(kh) + (r_3 + r_4)\, x_2(kh) + u_1(kh)
\end{aligned}$$

Se retoma la ecuación de salida.

$$\begin{aligned}
u_2(kh) &= (\alpha_2 q + \alpha_3)\, x(kh) \\
&= \alpha_2 x(kh + h) + \alpha_3 x(kh) \\
&= \alpha_3 x_1(kh) + \alpha_2 x_2(kh)
\end{aligned}$$

Finalmente se obtiene (Ec.1.62) que es la representación en espacio de estados de $G_2(q)$.

$$\begin{aligned}
\begin{bmatrix} x_1(kh + h) \\ x_2(kh + h) \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ -r_3 r_4 & r_3 + r_4 \end{bmatrix} \begin{bmatrix} x_1(kh) \\ x_2(kh) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} e(kh) \\
u_2(kh) &= \begin{bmatrix} \alpha_3 & \alpha_2 \end{bmatrix} \begin{bmatrix} x_1(kh) \\ x_2(kh) \end{bmatrix}
\end{aligned} \tag{1.62}$$

Del modelo anterior se obtiene la función en lenguaje C mostrada en el listado 1.6.2.

Listado 1.6.2 Realización paralelo de un controlador 1DOF (Modulo II).

```
double  modulo_II(double ek)
{       static double  x1 = 0, x2 = 0;
        double  x1s, x2s, u2k;
        // Ecuaciones de estado.
        x1s     = x2;
        x2s     = -r_3*r_4*x1 + (r_3+r_4)*x2 + ek;
        // Ecuacion de salida.
        u2k     = alpha3*x1 + alpha2*x2;
        // Actualizacion de variables.
        x1      = x1s;
        x2      = x2s;
return u2k;
}
```

1.6.3. Modelo en espacio de estados del componente $G_3(q)$

Con un procedimiento similar al utilizado para $G_1(q)$ se obtiene (Ec.1.63) que es la representación en espacio de estados de $G_3(q)$.

$$\begin{aligned} x(kh+h) &= r_2 x(kh) + e(kh) \\ u_3(kh) &= \alpha_4 x(kh) \end{aligned} \tag{1.63}$$

Del modelo en espacio de estado descrito por (Ec.1.63) se obtiene la función en lenguaje C mostrada en el listado 1.6.3.

Listado 1.6.3 Realización paralelo de un controlador 1DOF (Modulo III).

```
double  modulo_III(double ek)
{       static double  x1 = 0;
        double  x1s, u3k;
        // Ecuaciones de estado.
        x1s     = r2*x1 + ek;
        // Ecuacion de salida.
        u3k     = alpha4*x1;
        // Actualizacion de variables.
        x1      = x1s;
return u3k;
}
```

1.6.4. Integración de componentes en la realización paralelo

En el listado 1.6.4 se muestra el código utilizado para integrar las funciones `modulo_I`, `modulo_II` y `modulo_III`.

Listado 1.6.4 Realización paralelo de un controlador 1DOF (Integración Componentes).

```
double  controlador_RST(double ek)
{
         double  uk, u0k, u1k, u2k, u3k;

        // Ecuaciones de estado.
        u0k     = n4 * ek;
        u1k     = modulo_I( ek );
        u2k     = modulo_II( ek );
        u3k     = modulo_III( ek );
        uk      = u0k + u1k + u2k + u3k;
return uk;
}
```

1.6.5. Modelo en espacio de estados de la realización paralela

Con el fin de obtener un modelo en espacio de estados que incluya: el estado de $G_1(q)$, los dos estados de $G_2(q)$ y el estado de $G_3(q)$ se encontrará una expresión general que combine en un solo modelo la suma de los modelos (Ec.1.64), (Ec.1.65) y (Ec.1.66).

$$\begin{aligned} z_1(kh+h) &= A_1 z_1(kh) + B_1 e(kh) \\ u_1(kh) &= C_1 z_1(kh) + D_1 e(kh) \end{aligned} \tag{1.64}$$

$$\begin{aligned} z_2(kh+h) &= A_2 z_2(kh) + B_2 e(kh) \\ u_2(kh) &= C_2 z_2(kh) + D_2 e(kh) \end{aligned} \tag{1.65}$$

$$\begin{aligned} z_3(kh+h) &= A_3 z_3(kh) + B_3 e(kh) \\ u_3(kh) &= C_3 z_3(kh) + D_3 e(kh) \end{aligned} \tag{1.66}$$

A diferencia de la realización serie en la realización paralelo los submodelos estan desacoplados y por tanto la combinación de ecuaciones puede realizarse de manera directa.

$$\begin{bmatrix} z_1(kh+h) \\ z_2(kh+h) \\ z_3(kh+h) \end{bmatrix} = \begin{bmatrix} A_1 & 0 & 0 \\ 0 & A_2 & 0 \\ 0 & 0 & A_3 \end{bmatrix} \begin{bmatrix} z_1(kh) \\ z_2(kh) \\ z_3(kh) \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix} e(kh) \tag{1.67}$$

Se expresa $u(kh) = u_1(kh) + u_2(kh) + u_3(kh)$ en notación matricial.

$$u_3(kh) = \begin{bmatrix} C_1 & C_2 & C_3 \end{bmatrix} \begin{bmatrix} z_1(kh) \\ z_2(kh) \\ z_3(kh) \end{bmatrix} + n_4 e(kh) \tag{1.68}$$

Para obtener un modelo en espacio de estados de la realización paralelo se reemplazan A_i, B_i, C_i, D_i y z_i en (Ec.1.67) y (Ec.1.68). De este reemplazo se obtienen (Ec.1.69) y Ec.1.70).

$$\begin{bmatrix} v_1(kh+h) \\ v_2(kh+h) \\ v_3(kh+h) \\ v_4(kh+h) \end{bmatrix} = \begin{bmatrix} r_1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -r_3 r_4 & r_3 + r_4 & 0 \\ 0 & 0 & 0 & r_2 \end{bmatrix} \begin{bmatrix} v_1(kh) \\ v_2(kh) \\ v_3(kh) \\ v_4(kh) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} e(kh) \tag{1.69}$$

$$u(kh) = \begin{bmatrix} \alpha_1 & \alpha_3 & \alpha_2 & \alpha_4 \end{bmatrix} \begin{bmatrix} v_1(kh) \\ v_2(kh) \\ v_3(kh) \\ v_4(kh) \end{bmatrix} + n_4 e(kh) \tag{1.70}$$

Del modelo en espacio de estado descrito por las ecuaciones (Ec.1.69) y Ec.1.70). se obtiene la función en lenguaje C mostrada en el listado 1.6.5.

Listado 1.6.5 Realización paralelo de un controlador 1DOF.

```
double  ley_control(double ek)
{
        static double  v1 = 0, v2 = 0, v3 = 0, v4 = 0;
               double  v1s, v2s, v3s, v4s, uk;

        // Ecuaciones de estado.
        v1s     =  r1*v1 + ekh;
        v2s     =  v3;
        v3s     = -r3*r4*v2 + (r3+r4)*v3 + ekh;
        v4s     =  r2*v4 + ekh;

        // Ecuacion de salida.
        uk      = alpha1*v1 + alpha3*v2 + alpha2*v3 + alpha4*v4 + n4*ek;

        // Actualizacion de variables.
        v4      = v4s;
        v3      = v3s;
        v2      = v2s;
        v1      = v1s;

return uk;
}
```

1.7. Realización en el operador δ

Esta realización se obtiene a partir del operador δ, el cual se define por:

$$\delta = \frac{q-1}{h} \tag{1.71}$$

Expresando (1.19) en terminos del operador δ tenemos que:

$$\begin{aligned} G_c(q) &= \frac{n_4(\delta h+1)^4 + n_3(\delta h+1)^3 + n_2(\delta h+1)^2 + n_1(\delta h+1) + n_0}{q^4 + d_3(\delta h+1)^3 + d_2 q(\delta h+1)^2 + d_1(\delta h+1) + d_0} \\ &= \frac{b_4(\delta h)^4 + b_3(\delta h)^3 + b_2(\delta h)^2 + b_1(\delta h) + b_0}{(\delta h)^4 + a_3(\delta h)^3 + a_2(\delta h)^2 + a_1(\delta h) + a_0} \end{aligned} \tag{1.72}$$

Se reescribe (Ec.1.72) como (Ec.1.73).

$$G_c(q) = b_4 + \frac{c_3(\delta h)^3 + c_2(\delta h)^2 + c_1(\delta h) + c_0}{(\delta h)^4 + a_3(\delta h)^3 + a_2(\delta h)^2 + a_1(\delta h) + a_0} \tag{1.73}$$

De donde se obtiene (Ec.1.74).

$$u(kh) = b_4 e(kh) + \frac{c_3(\delta h)^3 + c_2(\delta h)^2 + c_1(\delta h) + c_0}{(\delta h)^4 + a_3(\delta h)^3 + a_2(\delta h)^2 + a_1(\delta h) + a_0} e(kh) \tag{1.74}$$

Se conforma la ecuación de salida del sistema.

$$u(kh) = \left(c_3(\delta h)^3 + c_2(\delta h)^2 + c_1(\delta h) + c_0\right) x(kh) + b_4 e(kh) \tag{1.75}$$

Siendo la señal auxiliar $x(kh)$ la descrita por (Ec.1.76). A partir de esta se generarán los estados del sistema.

$$x(kh) = \frac{1}{(\delta h)^4 + a_3(\delta h)^3 + a_2(\delta h)^2 + a_1(\delta h) + a_0} e(kh) \tag{1.76}$$

Se reescribe (Ec.1.76) como una ecuación en diferencias y se obtiene (Ec.1.77).

$$\begin{aligned} \left((\delta h)^4 + a_3(\delta h)^3 + a_2(\delta h)^2 + a_1(\delta h) + a_0\right) x(kh) &= e(kh) \\ (\delta h)^4 x(kh) + a_3(\delta h)^3 x(kh) + a_2(\delta h)^2 x(kh) + a_1(\delta h) x(kh) + a_0 x(kh) &= e(kh) \end{aligned} \tag{1.77}$$

Se despeja $(\delta h)^4 x(kh)$ de (Ec.1.77).

$$(\delta h)^4 x(kh) = -a_3(\delta h)^3 x(kh) - a_2(\delta h)^2 x(kh) - a_1(\delta h) x(kh) - a_0 x(kh) + e(kh)$$

Se hace la selección de estados descrita por (Ec.1.78).

$$\begin{aligned} x_1(kh) &= x(kh) \\ x_2(kh) &= (\delta h) x(kh) \\ x_3(kh) &= (\delta h)^2 x(kh) \\ x_4(kh) &= (\delta h)^3 x(kh) \end{aligned} \tag{1.78}$$

En (Ec.1.78) se aplica el operador δh.

$$\begin{aligned}
(\delta h)x_1(kh) &= x_2(kh) \\
(\delta h)x_2(kh) &= x_3(kh) \\
(\delta h)x_3(kh) &= x_4(kh) \\
(\delta h)x_4(kh) &= -a_0x_1(kh) - a_1x_2(kh) - a_2x_3(kh) - a_3x_4(kh) + e(kh)
\end{aligned} \tag{1.79}$$

Se reescribe (Ec.1.79) en forma matricial.

$$\delta h \begin{bmatrix} x_1(kh) \\ x_2(kh) \\ x_3(kh) \\ x_4(kh) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -a_0 & -a_1 & -a_2 & -a_3 \end{bmatrix} \begin{bmatrix} x_1(kh) \\ x_2(kh) \\ x_3(kh) \\ x_4(kh) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} e(kh) \tag{1.80}$$

En (Ec.1.80) se reemplaza δh por $q - 1$ y se obtiene en (Ec.1.81).

$$
\begin{aligned}
(q-1)\begin{bmatrix} x_1(kh) \\ x_2(kh) \\ x_3(kh) \\ x_4(kh) \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -a_0 & -a_1 & -a_2 & -a_3 \end{bmatrix} \begin{bmatrix} x_1(kh) \\ x_2(kh) \\ x_3(kh) \\ x_4(kh) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} e(kh) \\
q\begin{bmatrix} x_1(kh) \\ x_2(kh) \\ x_3(kh) \\ x_4(kh) \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -a_0 & -a_1 & -a_2 & -a_3 \end{bmatrix} \begin{bmatrix} x_1(kh) \\ x_2(kh) \\ x_3(kh) \\ x_4(kh) \end{bmatrix} + \begin{bmatrix} x_1(kh) \\ x_2(kh) \\ x_3(kh) \\ x_4(kh) \end{bmatrix} \\
&\quad + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} e(kh) \\
\begin{bmatrix} x_1(kh+h) \\ x_2(kh+h) \\ x_3(kh+h) \\ x_4(kh+h) \end{bmatrix} &= \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ -a_0 & -a_1 & -a_2 & 1-a_3 \end{bmatrix} \begin{bmatrix} x_1(kh) \\ x_2(kh) \\ x_3(kh) \\ x_4(kh) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} e(kh)
\end{aligned}
\tag{1.81}
$$

Se retoma la ecuación de salida del sistema (Ec.1.75).

$$
\begin{aligned}
u(kh) &= \left(c_3(\delta h)^3 + c_2(\delta h)^2 + c_1(\delta h) + c_0\right) x(kh) + b_4 e(kh) \\
&= c_3(\delta h)^3 x(kh) + c_2(\delta h)^2 x(kh) + c_1(\delta h) x(kh) + c_0 x(kh) + b_4 e(kh)
\end{aligned}
\tag{1.82}
$$

En (Ec.1.82) se reemplazan $x(kh)$, $(\delta h)x(kh)$, $(\delta h)^2 x(kh)$, $(\delta h)^3 x(kh)$ por $x_1(kh)$, $x_2(kh)$, $x_3(kh)$, $x_4(kh)$ y se obtiene (Ec.1.83).

$$
u(kh) = c_0 x_1(kh) + c_1 x_2(kh) + c_2 x_3(kh) + c_3 x_4(kh) + b_4 e(kh) \tag{1.83}
$$

Se reescribe (Ec.1.83) en forma matricial.

$$u(kh) = \begin{bmatrix} c_0 & c_1 & c_2 & c_3 \end{bmatrix} \begin{bmatrix} x_1(kh) \\ x_2(kh) \\ x_3(kh) \\ x_4(kh) \end{bmatrix} + b_4 e(kh) \tag{1.84}$$

Del modelo en espacio de estado descrito por las ecuaciones (Ec.1.81) y (Ec.1.84) se obtiene la función en lenguaje C mostrada en el listado 1.7.1.

Listado 1.7.1 Realización en el operador δ de un controlador 1DOF.

```
double  ley_control(double ek)
{
        static double  x1 = 0, x2 = 0, x3 = 0, x4 = 0;
               double  x1s, x2s, x3s, x4s, uk;

        // Ecuaciones de estado.
        x1s     = x1 + x2;
        x2s     = x2 + x3;
        x3s     = x3 + x4;
        x4s     = -a0*x1 -a1*x2 -a2*x3 +(1-a3)*x4 +ek;

        // Ecuacion de salida.
        uk      = c0*x1 + c1*x2 + c2*x3 + c3*x4 + b4*ek;

        // Actualizacion de variables.
        x4      = x4s;
        x3      = x3s;
        x2      = x2s;
        x1      = x1s;

return uk;
}
```

Bibliografía

[1] Karl J. Astrom, Bjorn Wittenmark: ***Computer-Controlled Systems,*** Prentice Hall, 1997

[2] Rengifo Rodad Carlos Felipe ***Notas de clase, Control Digital.*** Universidad del Cauca, Popayán 2005.

[3] Dare Afolabi: ***Catastrophes in Control Systems,*** Darsaky Publications, 2002.

[4] James F. Whindborne, Robert S. H. Istepanian y Jun Wu: ***Reduction of controller Fragility by Pole Sensitivity,*** IEEE Transactions on automatic control,Vol 46 ,N2, 2001.

[5] Alan J. Laub. Senior Member, IEEE: ***Numerical linear Algebra Aspects of Control Design Computation,*** IEEE Transactions on automatic control, Vol 1 AC 30,N2, 1985.

[6] MATLAB Control System Toolbox: ***User Guide. Reliable Computations.***

[7] John G. Proakis, Dimitris G. Manolakis: ***Digital Signal Procesing,*** Prentice Hall, 1998.

Capítulo 2

Hardware-in-the-Loop Simulation

2.1. Introducción

El proceso de diseño de un controlador involucra varias etapas. Inicia con el modelado matemático del sistema y finaliza con la verificación experimental. En la Academia muchas veces los proyectos de control quedan en la etapa de modelado y simulación, debido al alto costo, tiempo de implementación y dificultad en la construcción de una planta. Por otro lado, la verificación experimental clásica, requiere una conexión directa del sistema de control con la planta; lo cual implica un alto riesgo de destrucción de los equipos involucrados e incluso exponer vidas humanas, cuando se prueba un controlador defectuoso en situaciones criticas [1], [2].

"Hardware-in-the-Loop", (HIL) es un método de simulación capaz de dar solución a los anteriores problemas, ya que las entradas y salidas del controlador son conectadas a una simulación lo más real posible de la planta. Este método permite tener un mejor entorno de prueba y sin necesidad de exponer los equipos, validando el controlador en condiciones extremas de funcionamiento que en prototipos de prueba reales son difíciles de obtener. Por esta razón, esta técnica de simulación se ha integrado al proceso de diseño e implementación de controladores tanto a nivel industrial como académico [3].

En este capitulo se da una noción general del método de validación HIL (Hardware-

in-the-Loop Simulation), se comentan sus posibilidades tanto a nivel industrial como académico y las ventajas que presenta sobre los métodos clásicos de validación de controladores digitales. Además, se ilustra el procedimiento de implementación de una plataforma de validación de controladores digitales HIL utilizando el "toolbox" de MATLAB, xPC Target. Por medio del cual se hizo la descripción algorítmica utilizando bloques de SIMULINK, del modelo matemático no lineal para un péndulo invertido y el protocolo de comunicación a través del puerto paralelo con su sistema de control embebido en una FPGA.

2.2. Pasos en el diseño e implementación de controladores

Los pasos típicos en el diseño e implementación de controladores son representados en la Figura No. 2.1.

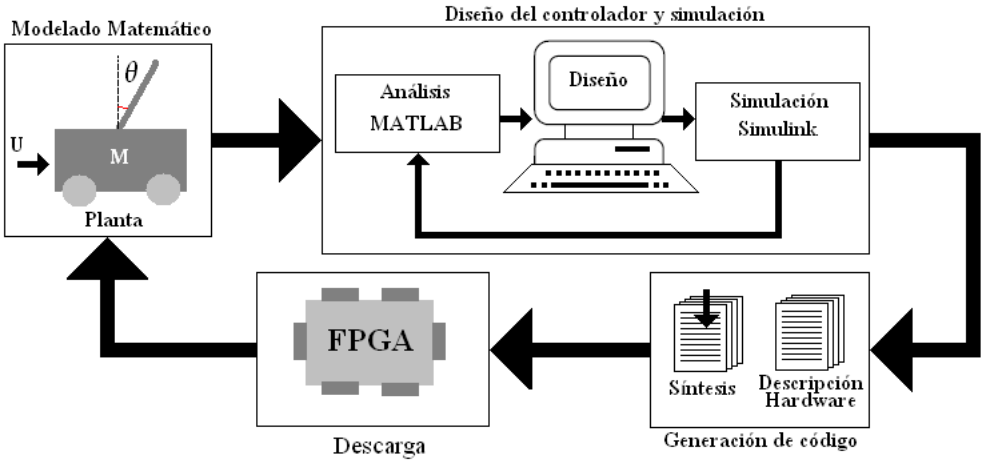

Figura 2.1: Pasos en el diseño e implementación de controladores.

El ciclo de diseño de un controlador comienza con el modelado de la planta y finaliza con la verificaron experimental. El modelado es una formulación matemática que describe la dinámica del proceso controlado a través de ecuaciones diferenciales; este se puede obtener de dos formas, por medio de leyes físicas o realizando una identificación.

La identificación consiste en obtener una aproximación del comportamiento dinámico de la planta, a partir de un análisis estadístico de las señales de entrada y salida del sistema [4].

Una vez obtenido el modelo de la planta, por medio de herramientas CACSD (Computer-aided control system design) como MATLAB, se procede a diseñar una estrategia de control, la cual se pone a punto por medio de simulaciones "off - line". Cuando se obtiene el comportamiento deseado del sistema, se continua con la fase de implementación; que consiste, en transcribir la estrategia de control en un algoritmo soportado por una arquitectura digital. Dependiendo del tipo de hardware que se use para la implementación del controlador, se escoge la herramienta generadora de código. Estas, a partir de un modelo abstracto como un diagrama en SIMULINK, generan código en un lenguaje de implementación como C, liberando al diseñador de la tediosa tarea de escribir y depurar el código, permitiendo optimizar el tiempo de implementación de la estrategia de control [5].

El principal inconveniente que surge entre la fase de simulación e implementación, son los recursos limitados del dispositivo hardware que calculará la ley de control. En una simulación "off-line" se cuenta con una gran cantidad de recursos como memoria y alta resolución aritmética, que algunos dispositivos no soportan, obligando al diseñador a cambiar la estrategia de control por una menos sensible a errores de precisión aritmética, liviana computacionalmente, o bien a adquirir un dispositivo con muchos más recursos que incrementa los costos del producto final. [5].

Por ultimo, la forma tradicional de probar el diseño implementado sobre el dispositivo embebido, consiste en integrar este directamente a la planta como se muestra en la Figura No. 2.2, en la cual se evalúa el comportamiento del sistema realimentado; si este, no cumple con los requerimientos planteados inicialmente, se regresa a la etapa de diseño y simulación. Todas las etapas descritas, se deben realizar cíclicamente hasta obtener un comportamiento adecuado del sistema de lazo cerrado. Este método de prueba de conexión directa a la planta, puede tener efectos irreversibles sobre esta, e incluso,

exponer la integridad física de los operarios; si el controlador, no es correctamente diseñado.

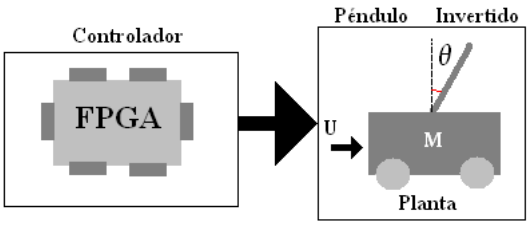

Figura 2.2: Conexión directa entre el controlador y la planta.

2.2.1. Hardware en lazo de simulación

El riesgo que implica la comprobación directa del ECS (Embedded Control Systems) sobre la planta, la necesidad de desarrollar productos en corto tiempo y el alto costo que involucra construir prototipos de prueba, hizo que la ingeniería se preocupara por desarrollar un método de prueba confiable y de bajo costo. Una de las primeras compañías en buscar la solución de este problema fue dSPACE GmbH, la cual creo un nuevo concepto de simulación denominado Hardware-in-the-Loop Simulation (HIL), como método de validación de sistemas de control embebidos encargados de la seguridad y confort en vehículos [1],[6]

Una configuración HIL tiene la capacidad de modelar el sistema completamente como una simulación "off-line", así como, la posibilidad de usar componentes reales ejecutándose en paralelo ("on-line") con la simulación de los otros componentes. Esta característica permite validar el comportamiento del ECS en lazo cerrado y establecer las condiciones de operación antes de ser probados sobre la planta real [7].

En una configuración HIL la planta de las Figuras No. 2.1 y 2.2, y algunos componentes del lazo de control, son reemplazados por la simulación en tiempo real de su comportamiento, como se muestra en la Figura No. 2.3.

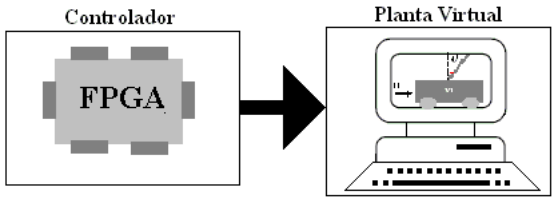

Figura 2.3: Configuración HIL.

Un sistema de control en tiempo real depende de la lógica del programa, y del tiempo en que se generen los resultados. Éste interactúa con la planta basado en la información de sensores que miden su estado actual, el cual debe ser consistente con el estado del entorno, o de lo contrario la acción de control podría ser defectuosa [7], [8].

Las posibles arquitecturas de una simulación HIL son mostrados en la Figura No. 2.4. Estas diferentes configuraciones dan la flexibilidad de remplazar componentes virtuales por reales, a medida que avanza la ejecución del proyecto [9]. La posibilidad de probar la unidad de control embebido sobre un entorno donde, actuadores, sensores y planta son simulados permite evaluar el desempeño del controlador en condiciones extremas, sin ningún riesgo, las cuales son difíciles de reproducir sobre una plataforma de prueba con componentes reales.

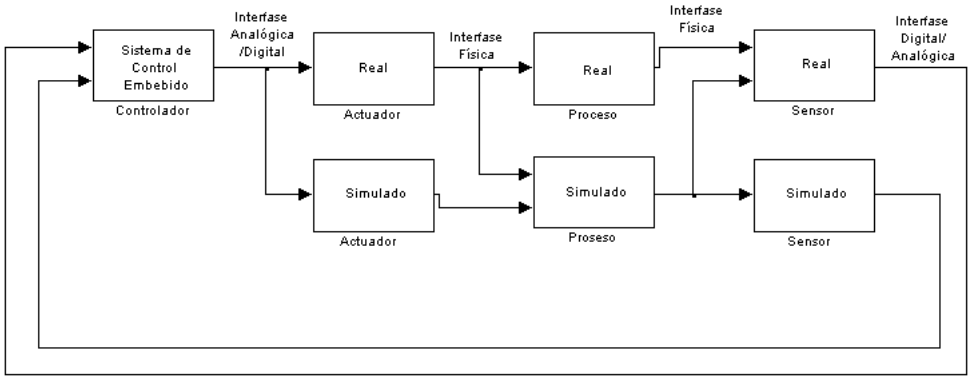

Figura 2.4: Arquitecturas HIL.

Otra de las ventajas que ofrece el método de simulación HIL es obtener plantas muy complejas en entornos virtuales, de bajo costo y en corto tiempo de implementación, lo que hace de la técnica de simulación HIL no solo una herramienta útil para probar sistemas de control embebidos, también puede ser usada en la enseñanza, permitiendo a los estudiantes validar controladores de diferentes plantas industriales sin ningún riesgo.

2.3. Implementación de la planta sobre la plataforma de tiempo real

Un péndulo invertido montado en un carro que se desplaza linealmente es mostrado en la Figura No. 2.5. El objetivo del controlador es desplazar linealmente el carro, conservando el péndulo invertido en posición vertical. Esta planta puede girar en cualquier momento y en cualquier dirección sobre el plano x, y; a menos que se aplique una fuerza de control (u) conveniente.

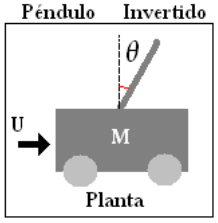

Figura 2.5: Diagrama de un Péndulo Invertido.

Uno de los objetivos planteados al realizar este trabajo, fue observar el comportamiento en lazo cerrado de un péndulo invertido, para el cual, su controlador de posición es implementado sobre una FPGA, utilizando como herramienta de validación HIL. El modelo matemático de la planta y otros componentes del lazo de control se implementaron sobre un "kernel" de tiempo real utilizando el "toolbox" de MATLAB, xPC Target, esta herramienta permite desarrollar prototipos de prueba que se ejecutan en tiempo real fácilmente.

xPC Target utiliza dos computadores, un PC "target" administrado a través de una comunicación serial por un PC "host", como se muestra en la Figura No. 2.6; este

ultimo debe ejecutar una plataforma Microsoft Windows soportada por MathWorks, MATLAB, SIMULINK y VISUAL C. El PC "target" es cualquier computador con un procesador desde Intel 386/486/Pentium o AMD K5/K6/Athlon en adelante, que cuente con una unidad de disco de 3,5 y un puerto serial libre, éste no requiere DOS, Windows, Linux ni ningún otro sistema operativo instalado, solo es necesario inicializar el PC "target" con un disquete que carga el "kernel"de tiempo real y configura la comunicación serial entre los dos equipos. El disco de inicio se crea en el PC "host" mediante la instrucion *xpcbootdisk* desde la ventana de comandos de MATLAB [10].

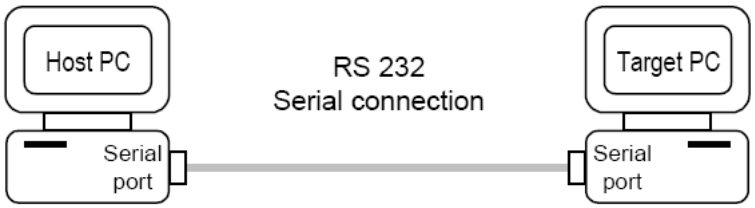

Figura 2.6: Interfaz de comunicación entre el computador "host" y "target".

La construcción de una aplicación que se ejecute en el "kernel" xPC Target, es una tarea relativamente simple, solo basta crear un modelo en SIMULINK del algoritmo que se desea ejecutar sobre el "kernel" de tiempo real. Luego, se cambia el paso de simulación de variable a fijo desde las opciones de simulación. Por ultimo, Real-Time Workshop, xPC Target y un compilador de C (en este caso Visual C) se encargan de generar a partir del modelo en SIMULINK, el código de tiempo real que es descargado desde el computador "host" al "target" a través de la comunicación serial entre estos.

En la Figura No. 2.7 se muestra el diagrama de simulación "off-line" del péndulo invertido y su sistema de control distribuido realizada en SIMULINK, en la cual fueron tenidas en cuenta diferentes consideraciones reales, como efectos de cuantización, ruido en la medición, además de modelar el comportamiento de sensores, actuadores y conversores. El controlador del péndulo invertido fue distribuido en dos partes, un regulador de ángulo de segundo orden y un controlador de posición de octavo orden. De este ultimo se obtuvieron las diferentes realizaciones descritas en el capitulo 1; cada una de las realizaciones fue probada "off-line" con este diagrama y posteriormente implementadas

sobre una FPGA.

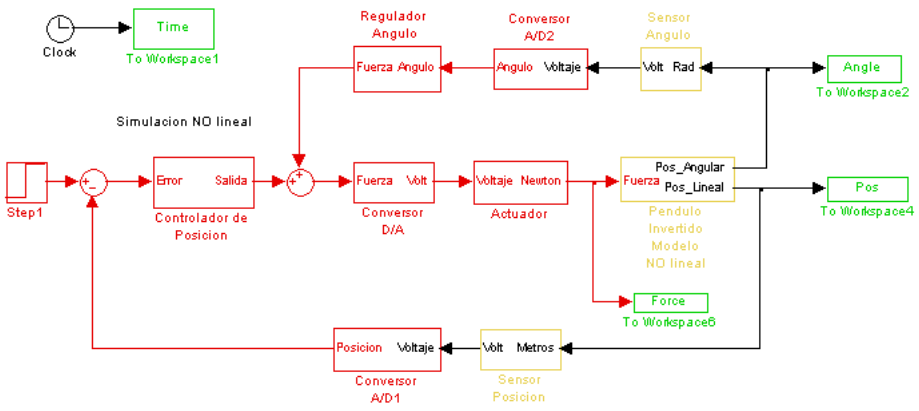

Figura 2.7: Simulación no lineal del Péndulo invertido y su controlador.

La construcción de la plataforma HIL (donde la planta, el regulador de ángulo, los actuadores y sensores son simulados y el controlador de posición es implementado físicamente sobre una FPGA), es obtenida reemplazando el controlador de el diagrama de la Figura No. 2.7 por un bloque de comunicación entre el "target" PC y la FPGA a través del puerto paralelo, como se muestra en el diagrama de la Figura 2.8. La FPGA recibe la señal de error, calcula los estados y retorna la ley de control, tal como lo hacia el bloque de control de la Figura No. 2.7.

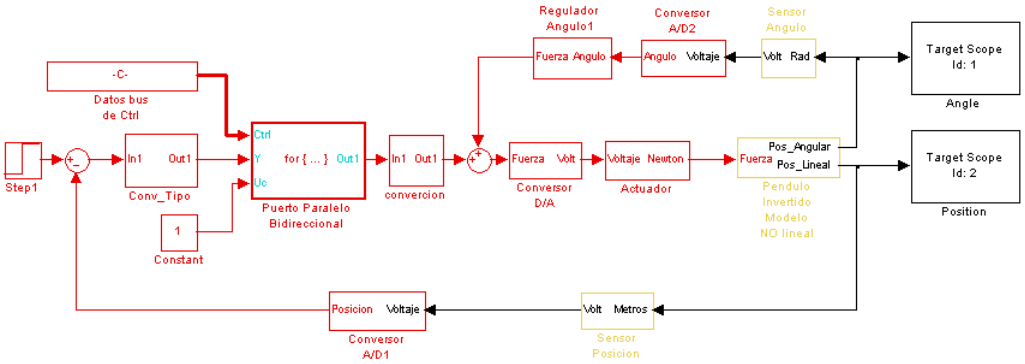

Figura 2.8: Diagrama de bloques de la implementación en el PC "target".

2.3.1. Comunicación entre la planta virtual y el controlador embebido

Este bloque (ver Figura No. 2.8) tiene la función de comunicar la FPGA y el PC "target" a través del puerto paralelo configurado en modo bidireccional. La FPGA recibe la señal de error proveniente del "kernel" de tiempo real y retorna la ley de control, con una transferencia de datos de 16 bits. El dato de 16 bits se dividide en dos, parte alta y parte baja cada una de 8 bits, enviadas una a la vez por el bus de datos; por medio del bus de control se informa a la FPGA que parte del dato le será enviado, de igual forma indica cuando la FPGA debe transmitir cada parte del dato hacia el PC. Una de las características más importantes de este bloque, es la necesidad de ejecutar cada una de estas tareas internas, en solo un paso de simulación.

El montaje experimental integrando: los computadores "target", "host" y el sistema de control embebido en la FPGA es mostrado en la siguiente Figura.

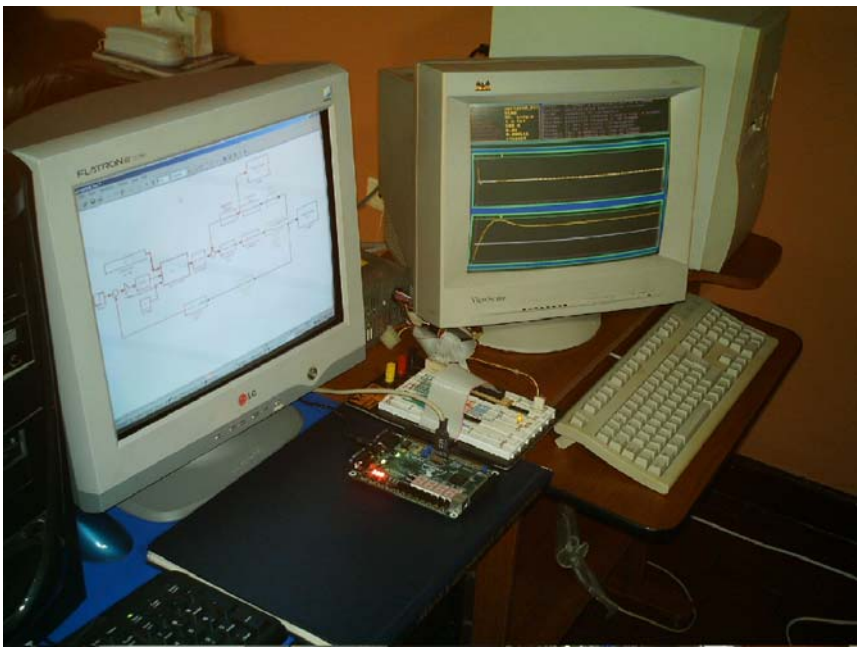

Figura 2.9: Montaje experimental de la configuración HIL.

Mediante esta configuración se probó el comportamiento de varias realizaciones del

controlador de posición del péndulo invertido embebido en una FPGA, permitiendo analizar su comportamiento real. Esta plataforma puede ser utilizada en la validación y puesta a punto de controladores implementados sobre diferentes arquitecturas digitales (FPGA, DSP, PLC, etc.), siempre y cuando el se satisfaga el protocolo de comunicación entre el controlador y el PC "target". Con lo cual se abre la posibilidad de utilizar esta configuración en una gran variedad de aplicaciones industriales, reduciendo los costos y tiempos de implementación; además, de los riesgos operacionales. Esta herramienta también tiene amplias posibilidades académicas; pues, permite afianzar los conceptos teóricos de control sobre aplicaciones reales, en el cual los estudiantes pueden llevar a cabo todo el proceso de diseño, implementación y verificación de algoritmos de control.

Bibliografía

[1] Grega Wojciech. ***Hardware-in-the-loop Simulation and application in control education,*** *ASEE/IEEE Frontiers in Education Conference*, 1999, sección 12*b*6, pg. 7-12.

[2] Harald Schludermann, Thomas Kirchmair. ***Soft-Commissioning: Hardware-in-the-loop-Based Verification of Controller Software,*** *Proceedings of the* 2000 *winter Simulation Conference,* pg. 893 – 899.

[3] NATIONAL INSTRUMENTS^{TM}. ***LabVIEW FPGA in Hardware-in-the-Loop Simulation Application,*** julio del 2003.

[4] Carlos Felipe Rengifo Rodas. ***Notas de clase, Identificación y Simulación de Sistemas,*** Universidad del Cauca, Popayán 2004.

[5] Lee Wootaik, Shin Minsuk, Sunwoo Myoungho. ***Target-identical rapid control prototyping platform for model-based engine control,*** South Korea 2004.

[6] Daniel Lemp, Adam Opel AG, Susanne Köhl, dSPACE GmbH, Markus Plöger, dSPACE GmbH. ***ECU Network Testing by Hardware-in-the-Loop Simulation,*** dSPACE 2003.

[7] Mansoor Sa'ad: ***Behaviour and operation of pump storage hydroplants,*** Universidad de Wales, Bangor 2000.

[8] Hanselmann Herbert. ***Hardware-in-the-Loop Simulation Testing and its Integration into a CACSD Toolset,*** *IEEE International Symposium on Computer-Aided Control System Design,* Michigan USA 1996.

[9] Sanvido Marco. ***Hardware in the loop simulation framework,*** Swiss federal institute the tecnology (ETH), Zurich 2002.

[10] MATLAB xPC Target Toolbox: ***Getting Started Version*** 2***.***

Capítulo 3

Implementación del controlador en la FPGA

3.1. Introducción

En esta sección se presenta el proceso de implementación de un sistema de control embebido (ECS) sobre una FPGA, para el posicionamiento de un péndulo invertido; éste fue diseñado por el director del trabajo de grado, mediante técnicas avanzadas de control; su modelo en espacio de estados se muestra en (3.2). También, se ilustra el procedimiento por el cual se obtienen diferentes realizaciones de éste, usando funciones de ***Matlab***. Finalmente se comentan las dificultades que se presentaron al diseñar una arquitectura dedicada en VHDL, y como las nuevas herramientas para el diseño hardware en FPGAs facilitaron este proceso.

3.2. Realizaciones del controlador de posición del péndulo invertido

La representación matemática de un controlador digital, en espacio de estados se muestra en(3.1).

$$x(kh+h) = Ax(kh) + Be(kh)$$
$$u(kh) = Cx(kh) + De(kh) \tag{3.1}$$

Siendo h el periodo de muestreo y $k = 0, 1, 2, ...$

Donde A, B, C y D son: la matriz de estado, la matriz de entrada, la matriz de salida y la matriz de transmisión directa respectivamente; además, $x(kh)$, $u(kh)$ y $e(kh)$ son: el vector de estados, la señal de control y el vector de entrada en el instante k. Donde; $e(kh)$, representa la diferencia entre la salida del proceso y la referencia.

La metodología utilizada para obtener las realizaciones del controlador, mediante instrucciones optimizadas de ***Matlab***, se ajusta para cualquier controlador. Los modelos en espacio de estados que se presentan, fueron los que finalmente se implementaron.

3.2.1. Realización Nominal

Las matrices que se muestran en (3.2), conforman el modelo en espacio de estados de la realización nominal. El código en C de la implementación en punto fijo de 32 bits se muestra en el listado 3.2.1.

$$A = \begin{bmatrix} 0{,}9216 & 0{,}0331 & 0{,}0045 & -0{,}0029 & 0{,}0077 & -0{,}0041 & 0{,}6439 & -3{,}9455 \\ 0{,}1729 & 0{,}8640 & 0{,}0390 & -0{,}0308 & -0{,}1553 & -0{,}4066 & -15{,}700 & 95{,}853 \\ 0{,}5974 & -0{,}5569 & 0{,}9135 & -0{,}0372 & -0{,}1954 & -0{,}1739 & -18{,}468 & 112{,}96 \\ 0{,}2192 & -0{,}2072 & -0{,}0302 & 0{,}9803 & -0{,}0792 & -0{,}0692 & -7{,}0340 & 43{,}023 \\ 0{,}0003 & -0{,}0005 & -0{,}0017 & 0{,}0042 & 1{,}0038 & 0{,}0061 & 0{,}1493 & -0{,}9497 \\ 0{,}0007 & -0{,}0000 & 0{,}0034 & -0{,}0090 & -0{,}0268 & 0{,}9211 & -2{,}7808 & 16{,}969 \\ 0{,}0001 & 0 & 0{,}0004 & -0{,}0010 & -0{,}0020 & 0{,}0016 & 0{,}9175 & 0{,}5026 \\ 0{,}0001 & 0 & 0{,}0005 & -0{,}0014 & -0{,}0032 & -0{,}0000 & -0{,}0047 & 0{,}9675 \end{bmatrix}$$

$$B^T = \begin{bmatrix} -1{,}159 & 29{,}35 & 34{,}81 & 12{,}5800 & -0{,}838 & 5{,}1820 & 0{,}4074 & 0{,}6427 \end{bmatrix}$$

$$C = \begin{bmatrix} 0{,}001 & 0{,}0014 & -0{,}0004 & 0{,}0008 & 0{,}0026 & 0{,}0041 & 0{,}2521 & -1{,}54 \end{bmatrix}$$

$$D = -0{,}472 \tag{3.2}$$

Listado 3.2.1 Realización Nominal

```
int  ley_control(int ek)
{
static int  x1 = 0, x2 = 0, x3 = 0, x4 = 0, x5 = 0, x6 = 0, x7 = 0, x8 = 0;
int         x1s, x2s, x3s, x4s, x5s, x6s, x7s, x8s ,uk;

const int a11=60400,a12=2166, a13=294,  a14=190,  a15=506,  a16=270,  a17=42197,  a18=258572;
const int a21=11330,a22=56625,a23=2557, a24=2015, a25=10180,a26=26646,a27=1028948,a28=6281821;
const int a31=39153,a32=36494,a33=59869,a34=2440, a35=12807,a36=11399,a37=1210289,a38=7402724;
const int a41=14362,a42=13576,a43=1978, a44=64247,a45=5191, a46=4536, a47=460978, a48=2819530;
const int a51=16,   a52=35,   a53=109,a54=273,  a55=65785,a56=402,  a57=9784,   a58=62237;
const int a61=48,   a62=0,    a63=225,  a64=592,  a65=1757, a66=60367,a67=182241, a68=1112123;
const int a71=5,    a72=9,    a73=24,   a74=64,   a75=128,  a76=105,  a77=60130,  a78=32938;
const int a81=7,    a82=4,    a83=34,   a84=91,   a85=208,  a86=3,    a87=305,    a88=63405;

const int b1=75964,b2=1923452,b3=2281569,b4=824198,b5=54953,b6=339616,b7 =26702, b8=42118;
const int c1=64,c2=93,c3=26,c4=50,c5=170,c6=267,c7=16518,c8=100956,d=30913;

        // Ecuaciones de estado.
        x1s = a11*x1 + a12*x2 + a13*x3 - a14*x4 + a15*x5 - a16*x6 - a17*x7 - a18*x8 - b1*ek;
        x2s = a21*x1 + a22*x2 + a23*x3 - a24*x4 - a25*x5 - a26*x6 - a27*x7 + a28*x8 + b2*ek;
        x3s = a31*x1 - a32*x2 + a33*x3 - a34*x4 - a35*x5 - a36*x6 - a37*x7 + a38*x8 + b3*ek;
        x4s = a41*x1 - a42*x2 - a43*x3 + a44*x4 - a45*x5 - a46*x6 - a47*x7 + a48*x8 + b4*ek;
        x5s = a51*x1 - a52*x2 - a53*x3 + a54*x4 + a55*x5 + a56*x6 + a57*x7 - a58*x8 - b5*ek;
        x6s = a61*x1 + a62*x2 + a63*x3 - a64*x4 - a65*x5 + a66*x6 - a67*x7 + a68*x8 + b6*ek;
        x7s = a71*x1 - a72*x2 + a73*x3 - a74*x4 - a75*x5 + a76*x6 + a77*x7 + a78*x8 + b7*ek;
        x8s  = a81*x1 + a82*x2 + a83*x3 - a84*x4 - a85*x5 - a86*x6 - a87*x7 + a88*x8 + b8*ek;

        // Ecuación de salida.
        uk  = c1*x1 + c2*x2 + c3*x3 + c4*x4 + c5*x5 + c6*x6 + c7*x7 + c8*x8 +d*ek;

        // Actualización de variables.
        x1      = x1s;
        x2      = x2s;
        x3      = x3s;
        x4      = x4s;
        x5      = x5s;
        x6      = x6s;
        x7      = x7s;
        x8      = x8s;

return uk;
}
```

3.2.2. Realización Paralela

La realización paralela o diagonal, se obtiene transformando la realización nominal a la forma canónica de Jordan. Esta se calcula utilizando la instrucción ***cannon*** de ***Matlab***. La cual tiene como argumentos de entrada el modelo en espacio de estados del controlador y retorna el nuevo modelo diagonalizado. El controlador obtenido y su correspondiente código en C de la implementación de punto fijo de 32 bits se muestran en (3.3) y en el listado 3.2.2 respectivamente.

$$
\begin{aligned}
A &= \begin{bmatrix}
0{,}864 & 0{,}098 & 0 & 0 & 0 & 0 & 0 & 0 \\
-0{,}098 & 0{,}864 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0{,}951 & 0{,}075 & 0 & 0 & 0 & 0 \\
0 & 0 & -0{,}075 & 0{,}951 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0{,}904 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0{,}100 & 0{,}0003 & 0 \\
0 & 0 & 0 & 0 & 0 & -0{,}0002 & 0{,}100 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0{,}9567
\end{bmatrix} \\
B^T &= \begin{bmatrix} 4{,}179 & 1{,}985 & -1{,}339 & 1{,}161 & -2{,}78 & -1{,}077 & -35{,}49 & -35{,}55 \end{bmatrix} \\
C &= \begin{bmatrix} -0{,}160 & -0{,}231 & -0{,}056 & 0{,}124 & -0{,}012 & 0 & 0 & -0{,}001 \end{bmatrix} \\
D &= -0{,}472
\end{aligned}
\tag{3.3}
$$

Listado 3.2.2 Realización Paralela

```
int  ley_control(int ek)
{
static int  x1 = 0, x2 = 0, x3 = 0, x4 = 0, x5 = 0, x6 = 0, x7 = 0, x8 = 0;
int         x1s, x2s, x3s, x4s, x5s, x6s, x7s, x8s ,uk;

const int a1=56625, a2=6408, a3=62316, a4=4951, a5=59219, a6=65518, a7=19, a8=62697;
const int b1=273872,b2=130101,b3=87734,b4=76098,b5=182202,b6=70564,b7=2326175,b8=2329587;
const int c1=10482,c2=15114,c3=3657,c4=8116,c5=794,c8 = 77;
const int d = 30913;

        // Ecuaciones de estado.
        x1s = a1*x1  + a2*x2  + b1*ek;
        x2s = a2*x1  + a1*x2  + b2*ek;
        x3s = a3*x3  + a4*x4  - b3*ek;
        x4s = -a4*x3 + a3*x4  + b4*ek;
        x5s = a5*x5  - b5*ek;
        x6s = a6*x6  + a7*x7  - b6*ek;
        x7s = -a7*x6 + a6*x7  - b7*ek ;
        x8s  = a8*x8 - b8*ek;
        // Ecuación de salida.
        uk  = -c1*x1 - c2*x2 - c3*x3 + c4*x4 - c5*x5 - c8*x8 +d*ek;
        // Actualización de variables.
        x1      = x1s;
        x2      = x2s;
        x3      = x3s;
        x4      = x4s;
        x5      = x5s;
        x6      = x6s;
        x7      = x7s;
        x8      = x8s;
return uk;
}
```

3.2.3. Realización Serie

La realización serie se obtiene mediante el comando *ss2sos* de ***Matlab***, el cual retorna la función de transferencia de cada uno de los bloques de segundo orden que conforman la realización. Luego, se obtiene la representación de cada bloque en espacio de estados con la instrucción *ss*. Los modelos en espacio de estados de cada modulo y su correspondiente código en C de la implementación en punto fijo de 32 bits se muestran en (3.4),(3.5),(3.6),(3.7) y en los listados 3.2.3, 3.2.4, 3.2.5, 3.2.6 respectivamente.

- Modulo I

$$\begin{bmatrix} x_1 1(kh+h) \\ x_1 2(kh+h) \end{bmatrix} = \begin{bmatrix} 0{,}9567 & 0 \\ 0 & 0{,}9036 \end{bmatrix} \begin{bmatrix} x_1 1(kh) \\ x_1 2(kh) \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix} e(kh)$$

$$u1(kh) = \begin{bmatrix} -0{,}7381 & 1{,}67 \end{bmatrix} \begin{bmatrix} x_1 1(kh) \\ x_1 2(kh) \end{bmatrix} + e(kh) \tag{3.4}$$

Listado 3.2.3 Realización Serie: Modulo I

```
int modulo1 (int ek)
{
const int a11 = 62697 , a12 = 59218;
const int b11 = 131072;
const int c11 = 48372 , c12 = 109437;
const int d = 65536;

static int X11 = 0, X12 = 0;
int X11s = 0,X12s = 0, u1 = 0;

//calculo de los estados
X11s = a11 * X11 + b11*ek;
X12s = a12 * X12 + b11*ek;
//calculo de la ley del primer modulo
u1 = -c11*X11 + c12*X12 + d*ek;
//actualizacion de los estados
return u1;
}
```

- Modulo II

$$\begin{bmatrix} x_2 1(kh+h) \\ x_2 2(kh+h) \end{bmatrix} = \begin{bmatrix} 0{,}864 & 0{,}0978 \\ -0{,}0977 & 0{,}864 \end{bmatrix} \begin{bmatrix} x_2 1(kh) \\ x_2 2(kh) \end{bmatrix} + \begin{bmatrix} 0 \\ 0{,}5 \end{bmatrix} e(kh)$$

$$u1(kh) = \begin{bmatrix} -0{,}1681 & -0{,}1636 \end{bmatrix} \begin{bmatrix} x_2 1(kh) \\ x_2 2(kh) \end{bmatrix} + u1(kh) \tag{3.5}$$

Listado 3.2.4 Realización Serie: Modulo II

```
int modulo2 (int u1)
{
const int a21 = 56625 , a22 = 6407;
const int b22 = 32768;
const int c21 = 11013 , c22 = 10722;
const int d = 65536;

static int X21 = 0, X22 = 0;
int X21s = 0,X22s = 0, u2 = 0;

//calculo de los estados
X21s =  a21*X21 + a22*X22;
X22s = -a22*X21 + a21*X22 + b22*u1;
//calculo de la ley del primer modulo
u2 = -c21*X21 - c22*X22 + d*u1;
//actualizacion de los estados
return u2;
}
```

- Modulo III

$$\begin{bmatrix} x_3 1(kh+h) \\ x_3 2(kh+h) \end{bmatrix} = \begin{bmatrix} 0{,}951 & 0{,}0754 \\ -0{,}0755 & 0{,}9509 \end{bmatrix} \begin{bmatrix} x_3 1(kh) \\ x_3 2(kh) \end{bmatrix} + \begin{bmatrix} 0 \\ 0{,}125 \end{bmatrix} u2(kh)$$

$$u3(kh) = \begin{bmatrix} -0{,}3186 & -0{,}1373 \end{bmatrix} \begin{bmatrix} x_3 1(kh) \\ x_3 2(kh) \end{bmatrix} + u2(kh) \tag{3.6}$$

Listado 3.2.5 Realización Serie: Modulo III

```
int modulo3 (int u2)
{
const int a31 = 62315 , a32 = 4950 ;
const int b32 = 8192;
const int c31 = 20881 , c32 = 8998;
const int d = 65536;

static int X31 = 0, X32 = 0;
int X31s = 0,X32s = 0, u3 = 0;

//calculo de los estados
X31s =   a31*X31 + a32*X32;
X32s =  -a32*X31 + a31*X32 + b32*u2;
//calculo de la ley del primer modulo
u3 = -c31*X31 - c32*X32 + d*u2;
//actualizacion de los estados
return u3;
}
```

- Modulo IV

$$\begin{bmatrix} x_4 1(kh+h) \\ x_4 2(kh+h) \end{bmatrix} = \begin{bmatrix} 0{,}951 & 0{,}0754 \\ -0{,}0755 & 0{,}9509 \end{bmatrix} \begin{bmatrix} x_4 1(kh) \\ x_4 2(kh) \end{bmatrix} + \begin{bmatrix} 0 \\ 0{,}125 \end{bmatrix} u3(kh)$$

$$u4(kh) = \begin{bmatrix} -0{,}3186 & -0{,}1373 \end{bmatrix} \begin{bmatrix} x_4 1(kh) \\ x_4 2(kh) \end{bmatrix} + u3(kh) \tag{3.7}$$

El codigo en lenguaje C que enlaza todas los modulos se muestra en el listado 3.2.7

Listado 3.2.6 Realización Serie: Modulo IV

```
int modulo4 (int u3)
{
const int a41 = 65517 , a42 = 18;
const int b42 = 2048;
const int c41 = 23 , c42 = 1174;
const int d = 65536;

static int X41 = 0, X42 = 0;
int X41s = 0,X42s = 0, u4 = 0;

//calculo de los estados
X41s =   a41*X41 - a42*X42;
X42s =   a42*X41 + a41*X42 + b42*u3;
//calculo de la ley del primer modulo
u4 = c41*X41 - c42*X42 + d*u3;
//actualizacion de los estados
return u4;
}
```

Listado 3.2.7 Ley de control: Realización serie

```
int Ley_control (int ek)
{
    const int K = -30913;
    int u_tem,U;

    u_tem = modulo1 (ek);
    u_tem = modulo2(u_tem);
    u_tem = modulo3(u_tem);
    u_tem = modulo4(u_tem);
    U = K * u_tem;
    return U;
    }
```

3.2.4. Realización con operador delta δ

Para obtener la realización en operador delta primero se debe llevar el modelo en espacio de estados del controlador a la forma de (1.35), esto se hace con el comando ***zpk*** de ***Matlab***; a continuación se definen los nuevos polos y ceros del controlador en el operador δ, como se ilustro en (1.72) y finalmente se retorna a la definición del controlador en términos del operador de desplazamiento, utilizando la expresión (1.71). El código

de la función en ***Matlab*** que obtiene esta realización se muestra en el listado 3.2.8. El controlador obtenido y su correspondiente código en C de la implementación en punto fijo de 32 bits se muestran en 3.8 y en el listado 3.2.9 respectivamente.

Listado 3.2.8 Algoritmo en Matlab para obtener la realización δ

```
function [Hd_SS] = delta_operator(Hq_SS)


% Convertion to a ZPK object
Hq_ZPK                              = zpk(Hq_SS);

% The zeros in shift operator are converted to zeros in delta operator
Z                                   = Hq_ZPK.Z{1};
Hq_ZPK.Z{1}                         = Z - ones(length(Z),1);

% The poles in shift operator are converted to poles in delta operator
P                                   = Hq_ZPK.P{1};
Hq_ZPK.P{1}                         = P - ones(length(P),1);

% Convertion to state-space object
Hd_SS                               = ss(Hq_ZPK);
I                                   = eye(size(Hd_SS.A));
Hd_SS.A                             = I + Hd_SS.A;
Hd_SS.B                             = Hd_SS.B;
```

$$A = \begin{bmatrix} 0{,}864 & 0{,}125 & -0{,}0685 & 0 & -0{,}0398 & 0 & 0{,}0005 & 0{,}0003 \\ -0{,}0765 & 0{,}864 & 0{,}0062 & 0 & 0{,}0036 & 0 & 0{,}0001 & 0{,}0001 \\ 0 & 0 & 0{,}9509 & 0{,}125 & -0{,}0571 & 0 & 0{,}0007 & 0{,}0005 \\ 0 & 0 & -0{,}0456 & 0{,}9509 & -0{,}0153 & 0 & 0{,}0002 & 0{,}0001 \\ 0 & 0 & 0 & 0 & 0{,}9997 & 0{,}125 & -0{,}0253 & -0{,}0181 \\ 0 & 0 & 0 & 0 & 0 & 0{,}9997 & -0{,}0216 & -0{,}0155 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0{,}9036 & -0{,}0140 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0{,}9567 \end{bmatrix}$$

$$B^T = \begin{bmatrix} -1{,}159 & 29{,}35 & 34{,}81 & 12{,}5800 & -0{,}838 & 5{,}1820 & 0{,}4074 & 0{,}6427 \end{bmatrix}$$

$$C = \begin{bmatrix} 0{,}001 & 0{,}0014 & -0{,}0004 & 0{,}0008 & 0{,}0026 & 0{,}0041 & 0{,}2521 & -1{,}54 \end{bmatrix}$$

$$D = -0{,}4717 \tag{3.8}$$

Listado 3.2.9 Realización en operador δ

```
int Ley_control (int ek)
{
static int  x1 = 0, x2 = 0, x3 = 0, x4 = 0, x5 = 0, x6 = 0, x7 = 0, x8 = 0;
int         x1s, x2s, x3s, x4s, x5s, x6s, x7s, x8s ,uk;

const int a11=56625,a12=8192,a13=4488,a15=2608,a17=31,a18=22,a21=5012,a22=56625,a23=407;
const int a25=237,a27=2,a28=2,a33=62315,a34=8192,a35=3742,a37=44,a38=32,a43=2991,a44=62315;
const int a45=1002,a47=12,a48=8,a55=65517,a56=8192,a57=1656,a58=1188,a66=65517,a67=1416;
const int a68=1015,a77=59218,a78=915,a88=62697;
const int b1=1228,b2=111,b3=1763,b4=472,b5=65236,b6=55785,b7=50246,b8=144117;
const int c1=314607,c3=112912,c5=65608,c7=785,c8=562,d=30913;

       // Ecuaciones de estado.
        x1s = a11*x1 + a12*x2 - a13*x3 - a15*x5 + a17*x7 + a18*x8 - b1*ek;
        x2s = -a21*x1 + a22*x2 + a23*x3 + a25*x5 - a27*x7 - a28*x8 + b2*ek;
        x3s = a33*x3 + a34*x4 - a35*x5 + a37*x7 + a38*x8 - b3*ek;
        x4s = -a43*x3 + a44*x4 - a45*x5 + a47*x7 + a48*x8 - b4*ek;
        x5s = a55*x5 + a56*x6 - a57*x7 - a58*x8 + b5*ek;
        x6s = a66*x6 - a67*x7 - a68*x8 + b6*ek;
        x7s = a77*x7 - a78*x8 + b7*ek;
        x8s  = a88*x8 + b8*ek;
        // Ecuación de salida.
        uk  = -c1*x1 - c3*x3 - c5*x5 + c7*x7 + c8*x8 - d*ek;
        // Actualización de variables.
        x1      = x1s;
        x2      = x2s;
        x3      = x3s;
        x4      = x4s;
        x5      = x5s;
        x6      = x6s;
        x7      = x7s;
        x8      = x8s;
return uk;
    }
```

3.3. Una FPGA: la arquitectura ideal para diseño de controladores de tiempo discreto

El continuo desarrollo de la microelectrónica con avances en la tecnología de alta escala de integración VLSI (Very large scale integrate), ha ampliado exponencialmente las posibilidades de procesamiento de información en las ultimas dos décadas, siendo el DSP (Digital Signal Procesor) su principal representante. Los DSPs son usados para implementar muchas aplicaciones de procesamiento de señal, tales como: protocolos de voz, comunicaciones inalámbricas, sistemas de radar y satélite, procesamiento de imágenes, etc. Sin embargo, al ampliarse la capacidad de procesamiento, se generan nuevas necesidades que no se han podido cubrir; pues a pesar de ser programados por software, son limitados debido a la arquitectura fija del hardware, que no los faculta para hacerle frente a los desafíos presentes en el procesamiento de señal, de la actualidad.

Las FPGAs presentan una solución flexible y eficiente para implementar aplicaciones DSP; debido a que pueden ser reconfigurados en hardware, ofreciendo una arquitectura completamente dedicada o especifica, para cada aplicación. Inicialmente las FPGAs, fueron concebidas para implementar lógica digital compleja, sobre áreas reducidas. Con la introducción de las FPGAs de alta densidad, tal como Stratix de Altera, Virtex 4 y Spartan 3 de Xilinx, que incorporan varios bloques funcionales embebidos de alto nivel, los diseñadores pueden crear sistemas completos sobre un chip programable (SOPC System On a Programable Chip). Estas características hacen de las FPGAs la plataforma hardware ideal para implementar controladores digitales, ya que permiten la personalización de algoritmos computacionales e implementar múltiples ciclos de control, que se ejecutan en paralelo sobre un solo chip de bajo costo.

El nivel de sofisticación que presentan las FPGAs en la actualidad, ha forzado un cambio en las herramientas de desarrollo y puesta a punto de hardware. Las compañías diseñadoras de software, ponen a nuestra disposición una amplia gama de tools de desarrollo; que permiten generar código VHDL optimizando para cada dispositivo, a partir de lenguajes de alto nivel, tales como C/C++ y diagramas de bloques en Simulink.

3.3.1. Diseño de la Arquitectura Hardware

Un controlador es un sistema dinámico, cuya representación en espacio de estados y su algoritmo en código C, se muestran en (3.2) y en el listado 3.2.1 respectivamente; de los cuales se observa que el calculo de los estados y la ley de control, se realiza mediante una suma de productos.

La realización nominal del controlador de posición del péndulo invertido, es el modelo mas pesado computacionalmente hablando, que se implemento durante este trabajo. Este requiere nueve multiplicaciones y ocho sumas para cada estado, e igual cantidad al calcular la ley de control; demandando 81 multiplicaciones y 72 sumas, o un total de 81 operaciones MAC (multiplicación, suma y acumula) para obtener todos los cómputos. Implementar una aplicación numéricamente intensiva como esta sobre una FPGA, implica la escogencia de varios parámetros de diseño, tales como:

- El tipo de aritmética con la cual se realizarán las operaciones. Los cálculos en punto flotante son mas precisos; pero, son computacionalmente mas exigentes y se debe proceder cuidadosamente, cuando se opera sobre números de orden de magnitud diferente. Los cálculos en punto fijo son computacionalmente mas livianos, sin embargo son menos precisos y se deben hacer excepciones para controlar el sobre flujo; a demás, es necesario establecer cual es la mínima resolución para representar adecuadamente los coeficientes del controlador (8, 16 o 32 bits).
- El número de módulos embebidos dedicados para aplicaciones DSP disponibles en la FPGA; pues establece la cantidad de operaciones que se pueden ejecutar en paralelo, sobre el dispositivo.
- El tiempo máximo de respuesta del hardware dedicado, quien determina el número de operaciones que se deben ejecutar en paralelo según el algoritmo.

Para establecer el tipo de aritmética, se hicieron varias simulaciones en ***Matlab*** y ***Simulink*** del controlador, en punto fijo de diferente resolución y en punto flotante de 32 bits. En la figura 3.1 se muestran los resultados de estas simulaciones, de la cual se concluye que las implementaciones en 32 bits de punto fijo y punto flotante son las mas

convenientes.

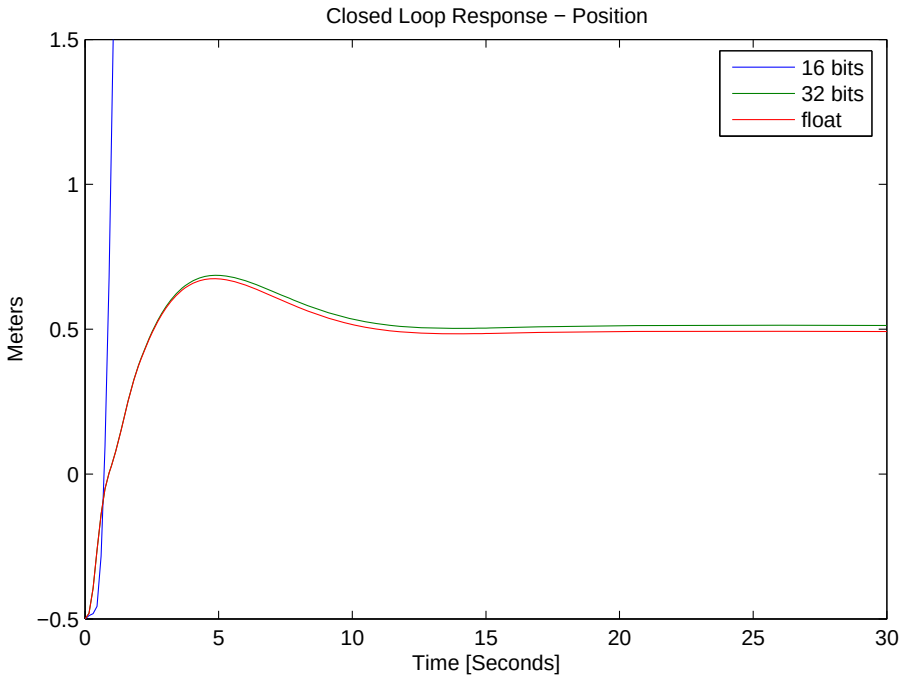

Figura 3.1: Simulaciones con diferente precisión

A continuación se hizo la descripción hardware de las unidades MAC de punto flotante, y en punto fijo de 32 bits, que soportan las condiciones de acarreo y sobre flujo. La descripción hardware de los módulos MAC se hizo por medio del ***ISE 6.2i*** de ***Xilinx***, para una FPGA ***Spartan 3***; usando como herramienta de síntesis ***XST*** y el ***Modelsim 5.7g*** para simular el Hardware. Posteriormente se hizo una simulación HIL con la cual se comprobó el correcto funcionamiento de los modulos MAC.

Del reporte de síntesis del XST para cada modulo, se tiene que: una operación MAC de 32 bits de punto fijo, consume el 33 % de los recursos DSP, y tarda 33 ns. mientras, que en punto flotante tarda 536 ns, y consume igual cantidad de recursos DSP disponibles en la ***Spartan 3***. Lo cual abre la posibilidad de utilizar hasta tres módulos MAC en paralelo; pero, se observa también, que la velocidad de procesamiento no es un parámetro critico, pues el tiempo de muestreo del controlador es Td = 0,01 s, y realizar todas

las operaciones sobre la MAC con mayor latency (tiempo en generar una respuesta) tarda menos de 0,5 ms. De lo cual se concluye que una arquitectura hardware adecuada para calcular la ley de control con estas MAC, debe centrarse en optimizar el consumo de recursos de la FPGA y no en la velocidad de procesamiento.

La arquitectura hardware que se propuso inicialmente para calcular la ley de control, buscando la optimización de recursos, se muestra en la figura 3.2; esta compuesta por cuatro componentes fundamentales: una unidad MAC, sobre la cual se realizaran todos los cálculos, puesto que no es necesario el procesamiento en paralelo; dos memorias, una ROM bidimensional y una RAM, las cuales contienen los coeficientes y todos los estados del controlador respectivamente. Finalmente se encuentra la unidad de control, que se encarga de determinar los ciclos de lectura y escritura de la RAM, indexar la ROM, la RAM y resetear el acumulador de la MAC para efectuar un nuevo cálculo.

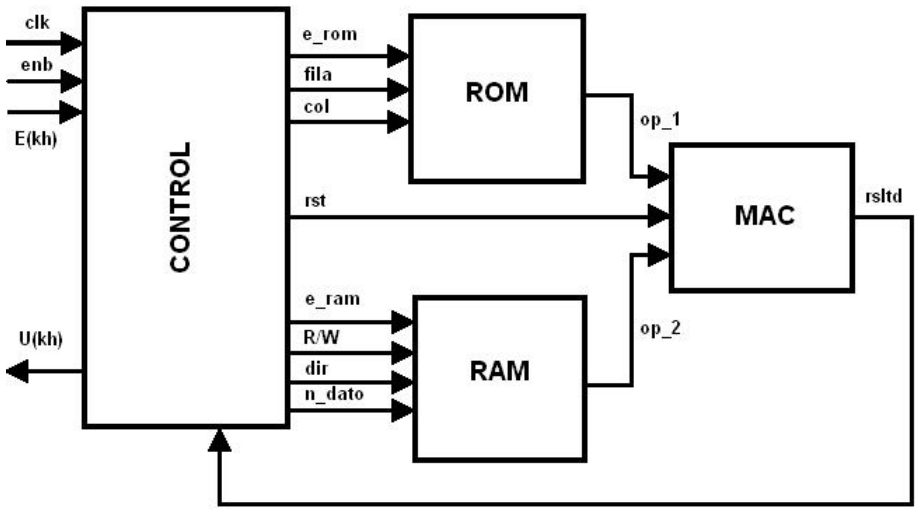

Figura 3.2: Arquitectura Hardware

La arquitectura funciona de la siguiente manera: Control es una unidad sincrona que coordina todas las componentes de la arquitectura, *enb* es un puerto de entrada al modulo de control, éste maneja el tiempo td de muestreo del controlador, y se controla desde el kernel de tiempo real. Para cada cambio de *enb* a estado activo se lee $e(k)$, actualiza éste valor sobre la RAM y procede a calcular la ley de control, indexando los

coeficientes en la primera fila de la ROM y los estados de la RAM, cuyas salidas son los operandos de la MAC; luego toma el resultado final de la MAC y actualiza la ley de control. De forma similar calcula cada estado, con la diferencia que para cada uno indexa su correspondiente fila de coeficientes sobre la ROM, y al terminar de calcular todos los estados siguientes, actualiza estos valores sobre la RAM.

La arquitectura se puso a punto con un controlador de bajo orden (2^{do}), utilizando una ROM de 3×3 y una RAM de 3 posiciones, con el fin de reducir la complejidad de la estructura y facilitar el seguimiento de las señales sobre esta, en las simulaciones HIL. El controlador es un RST con acción integral para una planta de primer orden; cuyo propósito es seguir la referencia y soportar los disturbios de carga que se generan en la entrada de la planta, su modelo en espacio de estados y el de la planta se muestra en (3.9) y (3.10), respectivamente. En la figura 3.6 se muestra el diagrama de bloques en simulink con el cual se hicieron las simulaciones HIL, y en las figuras 3.3, 3.4 y 3.5 se muestran la salida de la planta, la entrada a esta y la ley de control respectivamente. Estas graficas se obtuvieron del xPC-Target, de las cuales se observa que el sistema de lazo cerrado efectivamente esta siguiendo la referencia y soporta el disturbio de carga en la entrada de la planta, con lo cual se demuestra el correcto funcionamiento de la estructura hardware implementada para controladores de bajo orden.

$$\begin{bmatrix} x1(kh+h) \\ x2(kh+h) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x1(kh) \\ x2(kh) \end{bmatrix} + \begin{bmatrix} 0{,}26 & -1{,}3 \\ 0 & 1{,}04 \end{bmatrix} \begin{bmatrix} Uc(kh) \\ Y(kh) \end{bmatrix}$$

$$u(kh) = x1(kh) + 0{,}2Uc(kh) - 1{,}3Y(kh) \tag{3.9}$$

$$G(z) = \frac{1}{z^2 - 0{,}8z} \tag{3.10}$$

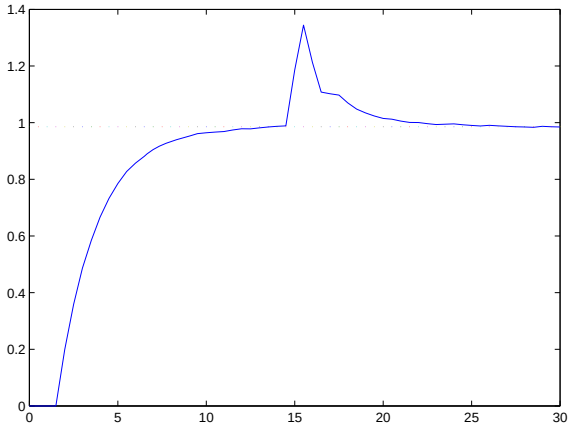

Figura 3.3: Salida de la planta

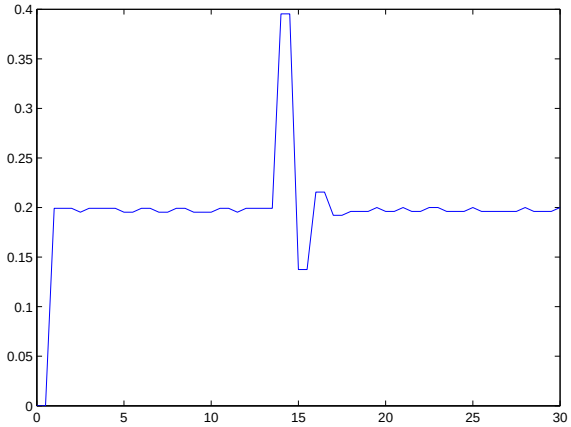

Figura 3.4: Entrada a la planta

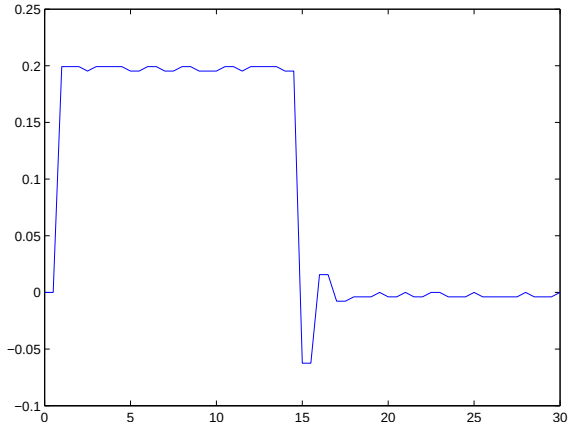

Figura 3.5: Ley de Control

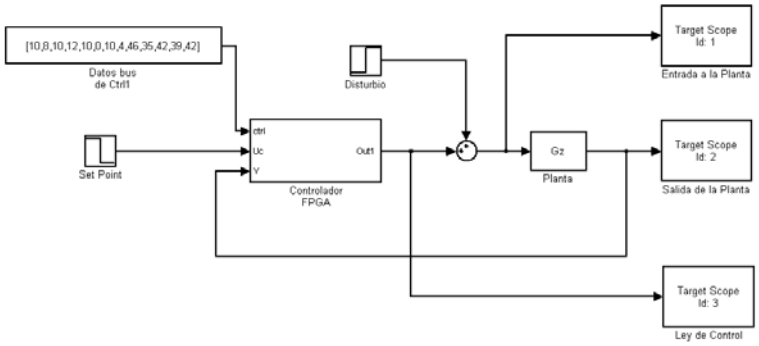

Figura 3.6: Modelo en simulink de la planta de segundo orden

Luego de corroborar el correcto funcionamiento de la arquitectura, se continuo con la implementación de la estructura para el controlador de octavo orden; se hizo la descripción en VHDL de la ROM bidimensional de 9 x 9, la RAM de 9 posiciones y la unidad de control que administre estos módulos. Cada unidad se simulo con ***Modelsim*** y se le hicieron pruebas HIL, con el fin de asegurar su adecuado funcionamiento individual.

Luego, se integraron uno por uno cada componente a la unidad de control. Empezando por la ROM, como se ilustra en la figura 3.7. Mediante pruebas HIL se hizo el seguimiento de todas las señales de entrada, de salida e intermedias con el fin de validar su adecuado comportamiento. Las simulaciones HIL fueron determinantes para analizar el funcionamiento de la arquitectura hardware, tanto globalmente como el de cada modulo al interactuar con los demás.

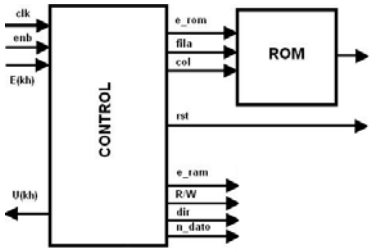

Figura 3.7: Integración de la ROM al modulo de control.

Después de asegurarse que los coeficientes del controlador se estaban indexando correctamente desde la ROM por medio de la unidad de control, se integro la RAM y se le hicieron pruebas HIL utilizando el esquema que se muestra en la figura 3.8. Con las pruebas se pretendía validar los ciclos de lectura y escritura de la RAM al llenarla con los coeficientes del controlador provenientes de la ROM. Una vez se comprobó el adecuado comportamiento de estos módulos, se integro la unidad MAC, obteniendo la arquitectura que se muestra en la figura 3.2. Momento en el cual la estructura empezó a fallar, debido a la gran cantidad de procesos que se sincronizaban a tan alta frecuencia (50 MHz).

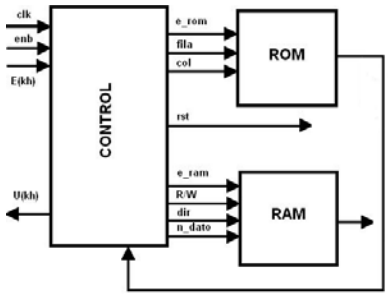

Figura 3.8: Integración de la ROM y la RAM al modulo de control.

Inicialmente se pensó que el problema estaba en la unida MAC; pero, mediante pruebas HIL se encontró que la MAC estaba operando bien, de acuerdo a los valores que provenían de la memorias ROM y RAM; entonces, se centro la atención en estos módulos. Encontrando el problema en la RAM, la cual mostraba deficiencias de sincronización en los tiempos de lectura y escritura. Este problema se corrigió convirtiendo la RAM en una unidad síncrona. A pesar de solucionar el problema en la RAM, la arquitectura continuo mostrando un comportamiento global deficiente para el controlador de octavo orden. Ahora el problema se ubico en la señal de error; el valor que se pasaba por el puerto, era diferente del que se actualizaba sobre la RAM; momento en el cual se pensó en cambiar la frecuencia de procesamiento de la arquitectura, pues se tenían que coordinar muchos mas procesos, que cuando se implemento el controlador de bajo orden, y para sorpresa nuestra la arquitectura presentaba un comportamiento diferente dependiente de la frecuencia. También se hicieron pruebas con dos frecuencias de sincronización en

la arquitectura; una para el modulo de control y otra del doble de la anterior para los otros módulos, con el fin de asegurarse que cada modulo complete las ordenes del modulo de control antes que éste cambie su estado. Con esta modificación se obtuvo un mejor funcionamiento de la arquitectura; pero, aun así no fue lo suficientemente confiable como para utilizarla en nuestro estudio; puesto que, no se comportaba bien durante el tiempo necesario. A pesar de todos los esfuerzos involucrados en la puesta a punto de esta arquitectura para controladores de alto orden, se decidió cambiar el enfoque en el proceso de implementación; pues esta etapa de la investigación había tomado cerca de la mitad del cronograma establecido para realizar la totalidad del proyecto, y aun no se obtenían los resultados esperados.

Debido a los inconvenientes encontrados en la descripción en VHDL de la arquitectura, se busco una herramienta de diseño hardware sobre FPGAs que facilitara este proceso. Existe en el mercado una amplia variedad de software para el desarrollo de hardware en FPGAs, los cuales presentan diferentes posibilidades de acuerdo a las necesidades de implementación. Para las FPGAs de Xilinx es posible utilizar ***System Generator, Spark, Impulse C, Nucleos, Catapult C, EDK (Embedded Development Kit),*** entre otros. Finalmente se decidió utilizar ***EDK***, pues como se explico antes la velocidad de procesamiento no era un parámetro critico, y por consiguiente no fue necesario el procesamiento en paralelo; además, se accedió con relativa facilidad a una versión demo por seis meses. El ***EDK*** como su nombre lo indica, permite el desarrollo y puesta a punto de un sistema embebido sobre FPGAs de gama alta y media de ***Xilinx***, el cual describe un procesador (para las ***Virtex*** 4 pueden ser hasta 4 procesadores) de 32 bits (***Microblaze*** o ***PowerPC*** según el dispositivo), con unidad aritmética de punto flotante que soporta el estándar IEEE 754.

Este tool tiene una interfaz grafica de usuario que permite, con unos cuantos click, crear una aplicación preliminar de cierto grado de complejidad, para alguna tarjeta de desarrollo soportada; en nuestro caso el ***Starter Board*** de ***Digilent*** para la ***Spartan 3***, a partir de la cual se inicia la construcción del embebido. La incorporación y diseño de nuevo hardware se hace también mediante una interfaz grafica de usuario, la cual

genera el código base en verilog o VHDL del periférico; que consiste en la descripción hardware del protocolo de comunicación del procesador embebido y el hardware personalizado, dejando al diseñador en libertad de describir el comportamiento del periférico.

La programación y puesta a punto del codigo en C del procesador embebido, se realiza mediante un compilador ***GNU*** (gcc). El manejo de los periféricos del procesador se realiza mediante apuntadores y soporta programación orientada a objetos. El ***EDK*** crea todas las librerías, controladores y archivos necesarios para el diseño del procesador. Además, da la posibilidad de bajarle al sistema un "kernel" de tiempo real.

Finalmente, a pesar de la potencialidad del ***EDK***, se presentaron algunos problemas con los tiempos de respuesta de la unidad aritmética de punto flotante, y con las multiplicaciones en punto fijo; ya que, retorna los 32 bits menos significativos del acumulador de 64 bits y no da la posibilidad de tomar los bits a nuestra conveniencia; lo cual nos obligo a utilizar hardware co-procesador para realizar las operaciones de multiplicación en 32 bits de punto fijo y solo utilizamos la unidad aritmética para efectuar sumas. En cuanto a las operaciones de punto flotante, no se pudo implementar hardware co-procesador; pues los recursos de la FPGA no lo permitieron y en definitiva se opto por abordar la implementación de las realizaciones en esta aritmética.

Bibliografía

[1] Ahmed Amine Jerraya, ***Multiprocessor Systems-on-Chips***, *Princeton University*, Morgan Kaufmann Publishers, Elsevier ,2005.

[2] Behrooz Parhami, ***Introduction to Parallel Processing Algorithms and Architectures***, *Princeton University*, University of California, Santa Barbara, Kluwer Academic Publishers, 2002.

[3] Uwe Mayer Baese, ***Digital Signal Processing with Field Programable Gate Arrays***, *Florida State University*, Editorial Springer, 2001.

[4] Xilinx, Advanced Designe Guide ***DSP: Designing for optimals results***, *Xilinx, DSP Products*, 2005.

[5] David A. Gwaltney, Kenneth D. King and Keary J. Smith, ***Implementation of Adaptative Digital Controllers on Programmable Logic Devices***, *Nasa Marshall Space Flight Center*,Huntsville.

[6] The MathWorks inc, ***Using the Control System Toolbox,Version*** 1, 2000.

[7] Miguel A. Vega, Juan M. Sánchez, Juan A. Gómez ***Advances in FPGA tools and techniques***, *Microprocessors and Microsystems*,Elsevier, 2005.

[8] Mario Vera Lizcano, Gustavo Vejarano, Jaime Velasco, ***Diseño de Funciones DSP usando VHDL y CPLDs-FPGAs***, *Universidad del Valle.*

Capítulo 4

Análisis de resultados y conclusiones

Integrando el ECS (Embedded Control System) sobre el cual se implementaron las realizaciones anteriormente expuestas, con el modelo matemático no lineal del péndulo invertido, ejecutado sobre el "kernel"de tiempo real xPC Target; se valido el comportamiento de lazo cerrado del sistema ante una entrada escalón, durante un tiempo de simulación de 300 segundos. Cada realización se obtuvo de los modelos descritos en el capitulo I, cuyas representaciones en espacio de estados y sus algoritmos en lenguaje C se muestran en el capitulo III, Para cada una de estas realizaciones se calculo el índice de sensibilidad mediante la ecuación (*) propuesta por (*). Cada implementación se realizo sobre una FPGA utilizando EDK como herramienta de desarrollo. A continuación se presenta los resultados de las simulaciones HIL de cada realización y se analiza tanto el comportamiento dinámico del sistema realimentado como las características algorítmicas que presenta cada una de ellas.

4.1. Realización Nominal

El modelo en espacio de estados de esta realización y su correspondiente código en C se muestran en (3.2) y el listado 3.2.1 respectivamente. Comparando los algoritmos de todas las realizaciones se observa, que esta es la realización computacionalmente mas

pesada, ya que demanda 81 multiplicaciones, 72 sumas y 17 asignaciones. Su índice de sensibilidad es uno de los mas bajos con un valor de: 4′686,884,800.

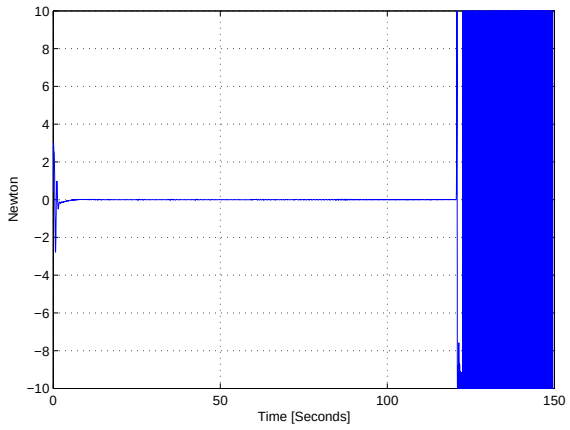

Figura 4.1: Ley de Control; Realización Nominal.

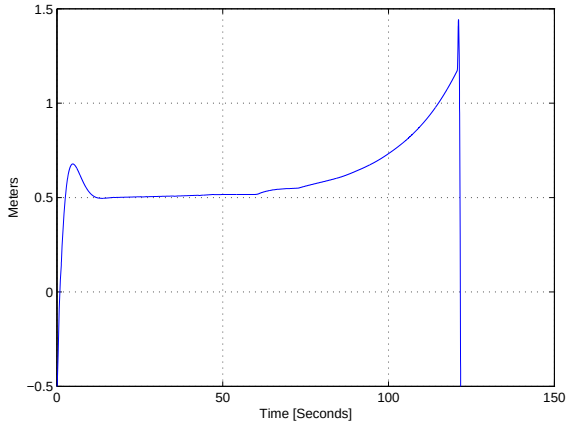

Figura 4.2: Respuesta de Lazo Cerrado (Posición); Realización Nominal.

En las figuras 4.1, 4.2 y 4.3 se muestra la ley de control, la respuesta de lazo cerrado del sistema para la posición y el ángulo del péndulo invertido; en ellas claramente se observa que el sistema se desestabiliza a pesar de presentar un índice de sensibilidad bajo.

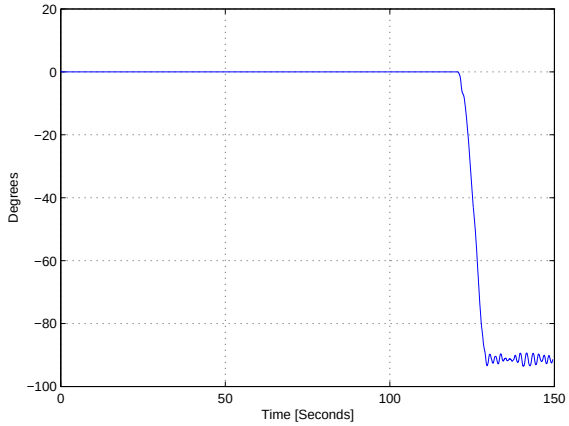

Figura 4.3: Respuesta de Lazo Cerrado (Ángulo); Realización Nominal.

4.2. Realización Delta

El modelo en espacio de estados de esta realización y su correspondiente código en C se muestran en (3.8) y el listado 3.2.9 respectivamente. El algoritmo es mas liviano que el de la realización nominal, demanda 46 multiplicaciones, 37 sumas y 17 asignaciones; su índice de sensibilidad es el mas alto con un valor de: 2810′287,350,546.

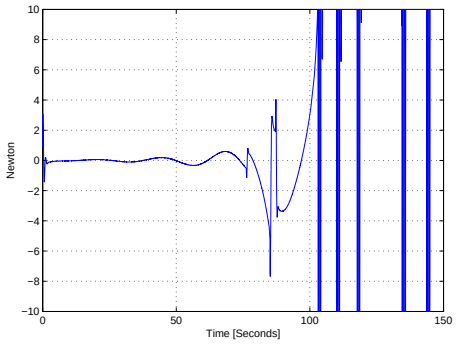

Figura 4.4: Ley de Control; Realización Delta.

En las figuras 4.4, 4.5 y 4.6 se muestra la ley de control, la respuesta de lazo cerrado del sistema para la posición y el ángulo del péndulo invertido; en ellas se observa que el sistema se desestabiliza.

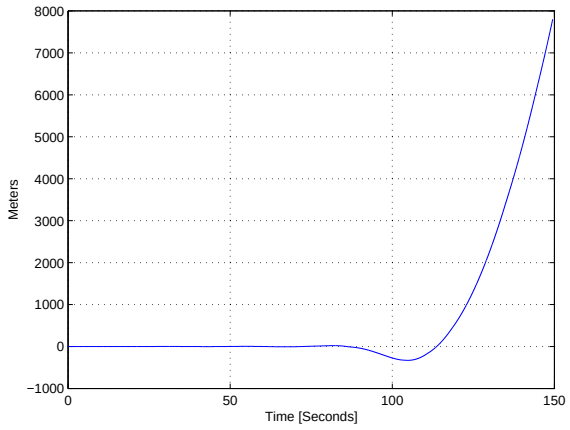

Figura 4.5: Respuesta de Lazo Cerrado (Posición); Realización Delta.

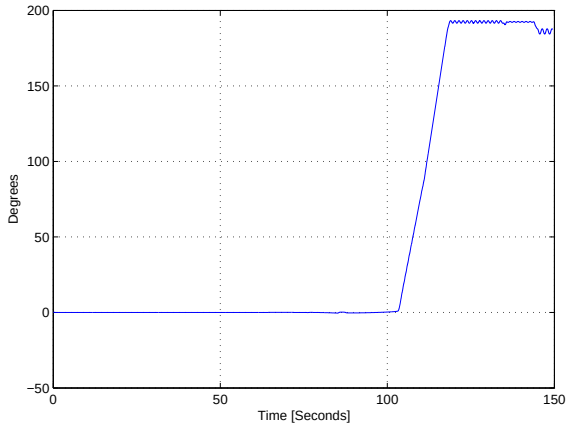

Figura 4.6: Respuesta de Lazo Cerrado (Ángulo); Realización Delta.

4.3. Realización Serie

El modelo en espacio de estados de cada modulo de esta realización y su correspondiente código en C se muestran en (3.4), (3.5), (3.6), (3.7) y los listados 3.2.3, 3.2.4, 3.2.5, 3.2.6 respectivamente. El algoritmo es el más liviano que se obtuvo, demanda 28 multiplicaciones, 19 sumas y 17 asignaciones; su índice de sensibilidad es de los más altos con un valor de: =147′289,728,253.

En las figuras 4.7, 4.8 y 4.9 se muestra la ley de control, la respuesta de lazo cerrado del sistema para la posición y el ángulo del péndulo invertido; en las cuales se observa que el sistema es estable.

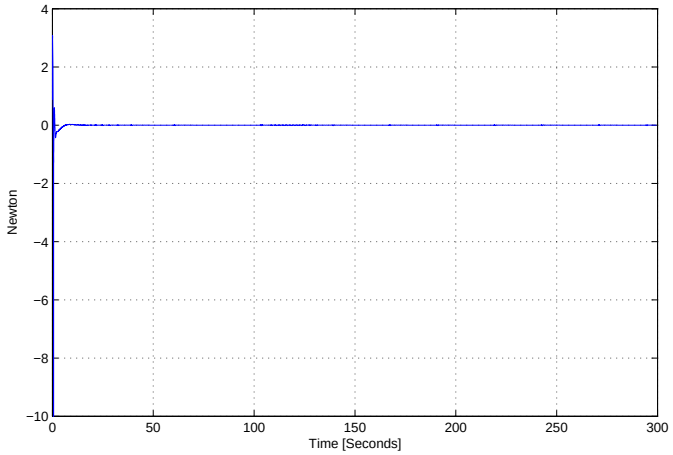

Figura 4.7: Ley de Control; Realización Serie.

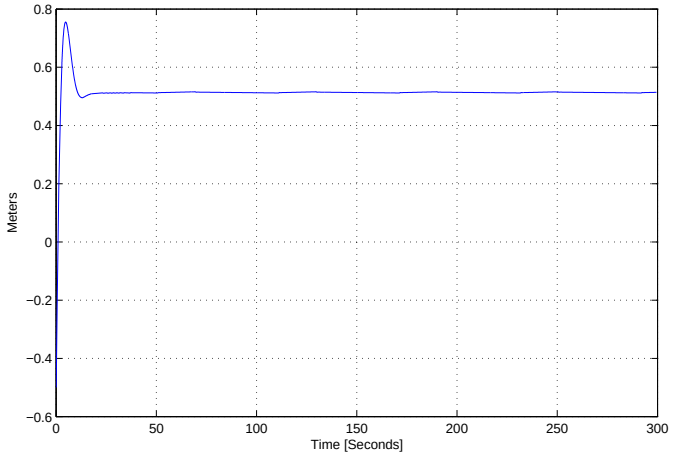

Figura 4.8: Respuesta de Lazo Cerrado (Posición); Realización Serie.

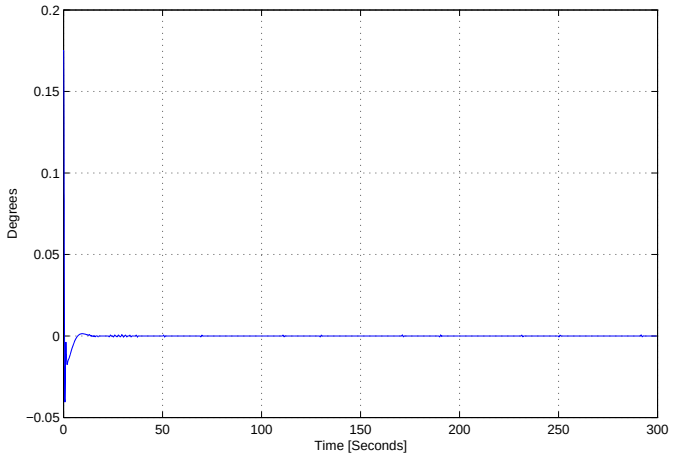

Figura 4.9: Respuesta de Lazo Cerrado (Ángulo); Realización Serie

4.4. Realización Paralela

El modelo en espacio de estados de esta realización y su correspondiente código en C se muestran en (3.3) y el listado 3.2.2 respectivamente. Su algoritmo, en cuanto a numero de operaciones es muy similar al de la realización cascada, ya que demanda 29 multiplicaciones, 20 sumas y 17 asignaciones; su índice de sensibilidad es: 18'836.021.703.

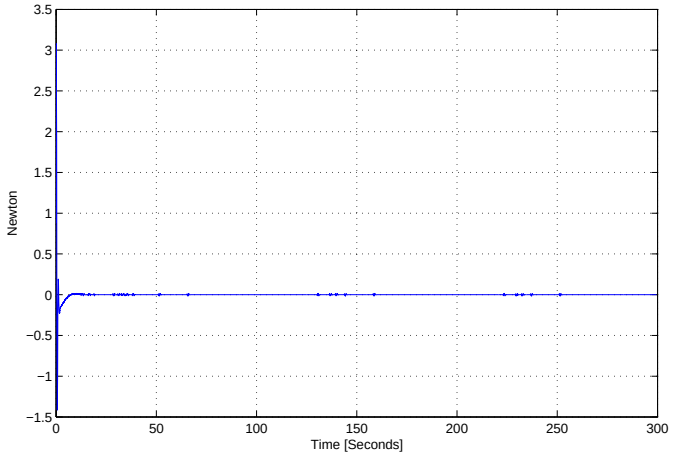

Figura 4.10: Ley de Control; Realización Paralela

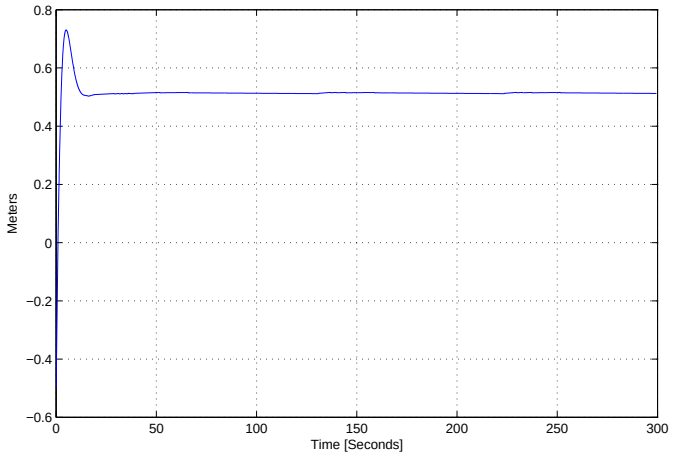

Figura 4.11: Respuesta de Lazo Cerrado (Posición); Realización Paralela

En las figuras 4.10, 4.11 y 4.12 se muestra la ley de control, la respuesta de lazo cerrado del sistema para la posición y el ángulo del péndulo invertido; en las cuales se aprecia

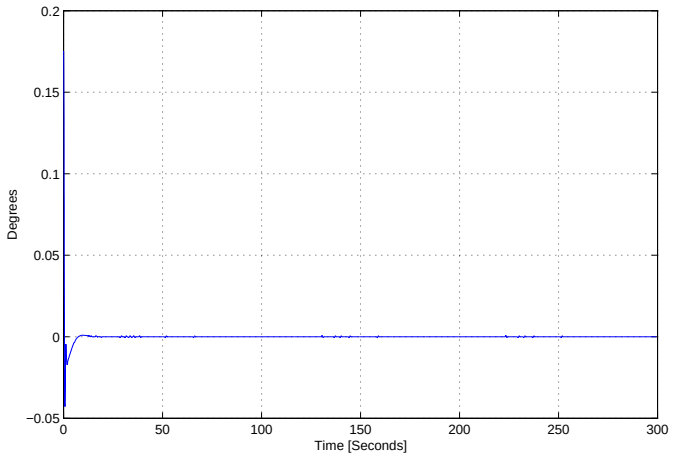

Figura 4.12: Respuesta de Lazo Cerrado (Angulo); Realización Paralela

que el sistema es estable.

En el proceso usual de diseño de controladores digitales, se asume que éste puede ser implementado exactamente, y que la mayor fuente de incertidumbre del sistema de lazo cerrado se encuentra en la planta; debido a las aproximaciones consideradas por el diseñador y la dificultad en la determinación exacta de los parámetros de ésta. Comparando las respuestas de lazo cerrado de cada realización implementada, claramente se observa que la dinámica del sistema depende de la realización. Demostrando que el controlador, también presenta un nivel de incertidumbre que puede desestabilizar el sistema realimentado; dicha incertidumbre proviene de la cuantización en los parámetros del controlador y del redondeo en las operaciones aritméticas.

De las graficas obtenidas de las respuestas de lazo cerrado para cada realización y su correspondiente índice de sensibilidad se observa que algunas realizaciones presentan un comportamiento de lazo cerrado inestable a pesar de tener un índice de sensibilidad bajo; comparado con otras, cuya respuesta de lazo cerrado fue estable. Por otro lado, a partir del número de operaciones aritméticas se infiere que las realizaciones con menos cálculos presentan un mejor comportamiento en lazo cerrado, minimizando los efectos de longitud de palabra finita y reducen la complejidad algorítmica.

4.5. Conclusiones

Comparando los índices de sensibilidad de cada realización y sus correspondientes respuestas de lazo cerrado; claramente se observa, que la dinámica del sistema depende de la realización implementada; además, obtener el modelo de menor sensibilidad polo coeficiente, no garantiza la estabilidad del sistema ante efectos de longitud de palabra finita; mientras que, una realización con pocas operaciones y con una adecuada representación de sus coeficientes en la precisión elegida, minimiza sus efectos. Además facilita su implementación y mejoran los tiempos respuesta del controlador.

Cuando se realiza aritmética intensiva sobre FPGAs, el tiempo de respuesta del hardware dedicado y los recursos disponibles sobre el dispositivo, son los parámetros fundamentales que determinan las condiciones de procesamiento (Secuencial o Paralelo). Tiempo corto de respuesta implica procesamiento en paralelo y un alto consumo de recursos; mientras que, el procesamiento secuencial optimiza el consumo de recursos, pero podría no satisfacer la velocidad de procesamiento. Hacer que estos dos parámetros de diseño converjan y sincronizar toda la arquitectura, son los retos fundamentales que enfrenta el diseñador de hardware sobre FPGAs.

Los enormes avances que presentan las herramientas software para el diseño de hardware en FPGAs, reducen dramáticamente el tiempo de implementación y facilitan el proceso de desarrollo de estructuras complejas, basados en lenguajes de alto nivel como C, C++, diagramas de bloques en Simulink, entre otros . En la actualidad no es necesario que el diseñador centre sus esfuerzos en la descripción de hardware mediante código VHDL, ni posea conocimientos avanzados de diseño sobre estas arquitecturas; sino, en determinar las características esenciales del hardware dedicado; tales como: tipo de procesamiento (secuencial o paralelo), tiempo de respuesta y el objetivo de la arquitectura (Procesamiento de señal o lógica digital); para, posteriormente elegir la herramienta de desarrollo que mejor se ajusta a los requerimientos de la arquitectura y facilite su implementación.

Utilizando ayudas computacionales de Matlab se logro construir una potente herramienta de validación de controladores HIL. Esta herramienta presenta la ventaja de ser

fácilmente implementada, de bajo costo y con resultados altamente confiables. Además permite probar el controlador sobre un entorno virtual en condiciones extremas de operación sin ningún riesgo. Lo que hace de esta herramienta un método útil, de simulación, validación y puesta a punto de sistemas de control embebido, permitiendo un estudio detallado de problemas de implementación; tanto a nivel industrial como académico.