Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías de Telecomunicación

Intensificación Electrónica

Entorno de simulación Hardware-In-the-Loop para estudios de tolerancia a fallos en sistemas electrónicos complejos

Autor: Daniel López Gómez

Tutores: Hipólito Guzmán Miranda

Miguel Ángel Aguirre Echánove

Departamento de Ingeniería Electrónica

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2015

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías de Telecomunicación

Entorno de simulación Hardware-In-the-Loop para estudios de tolerancia a fallos en sistemas electrónicos complejos

Autor:
Daniel López Gómez
(danlopgom@gmail.com)

Tutores:
Hipólito Guzmán Miranda
Profesor Ayudante Doctor

Miguel Ángel Aguirre Echánove
Catedrático de Universidad

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Trabajo Fin de Grado: Entorno de simulación Hardware-In-the-Loop para estudios de tolerancia a fallos en sistemas electrónicos complejos

Autor: Daniel López Gómez

Tutores: Hipólito Guzmán Miranda

Miguel Ángel Aguirre Echánove

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

"Conocimiento es poder.
Conocimiento compartido
es poder multiplicado."

Robert Noyce

Agradecimientos

"Yo soy yo y mi circunstancia y si no la salvo a ella no me salvo yo"

José Ortega y Gasset

La realización de este Trabajo Fin de Grado ha sido posible gracias a la influencia de muchas personas, pero quisiera aprovechar las siguientes líneas para dar las GRACIAS especialmente a:

Mis tutores, **Dr. Hipólito Guzmán Miranda** y **Dr. Miguel Ángel Aguirre Echánove**, por ofrecerme la posibilidad de realizar un Trabajo Fin de Grado que me ilusione y por dedicarme parte de su tiempo siempre que lo he necesitado.

A mis compañeros y amigos de clase, especialmente a **Francisco Jesús Marchal Cebador** y **Francisco José Pérez Díaz**, gracias a ellos se han hecho más llevaderas las dificultades del día a día universitario durante el cual hemos compartido más que libros y apuntes.

A los **profesores** que he tenido a lo largo de mi vida, desde el colegio hasta la universidad. Todos ellos me han aportado en mayor o menor medida conocimientos y valores tanto académicos como personales que me han servido y servirán durante el resto de mi vida.

Y por último y no menos importante, a mis padres, **Antonio e Isabel**. Me han hecho la vida mucho más fácil de lo que es. Nunca podré agradecerles lo suficiente el esfuerzo que vienen realizando desde hace años.

Daniel López Gómez

Sevilla, 2015

Resumen

Lo que el lector se encontrará en este documento es la descripción de un nuevo método de análisis de la tolerancia a fallos en sistemas electrónicos, dándole un enfoque diferente al que ya se le ha dado en los estudios existentes. Detallando lo que se encontrará en cada capítulo:

A lo largo de los dos primeros capítulos se hará un estudio del estado del arte en el ámbito de la inyección de fallos y un breve resumen de lo que ha caracterizado hasta ahora a estos estudios.

Los capítulos 3 y 4 contienen la descripción de la técnica que se quiere mostrar, de modo que se especificarán tanto sus características como los pasos y las herramientas necesarias para su realización.

En el capítulo 5 se encuentran los resultados obtenidos a partir de los varios análisis que se han llevado a cabo durante la realización de este trabajo, además de también analizarlos y arrojar conclusiones sobre ellos.

Por último, los capítulos 6 y 7 recogen la interpretación personal del autor acerca de si se han alcanzado los objetivos propuestos inicialmente y los posibles trabajos que podrían realizarse en el futuro como continuación del aquí realizado.

Abstract

What the reader will find in this document is the description of a new method of analysis of fault tolerance in electronic systems, giving it a different approach than has already been given in existing studies. Detailing what can be found in each chapter:

In the first two chapters, the state of the art in the field of fault injection will be detailed and also a brief summary of the characteristics of the studies existing.

Chapters 3 and 4 contain the description of the technique to be shown, so that their characteristics, and the steps and tools necessary for implementation will be specified.

In Chapter 5 the results obtained from the various analyses that have been carried out during the course of this work are described. Additionally, results are analyzed in order to draw conclusions from them.

Finally, chapters 6 and 7 collect the personal interpretation of the author on whether the goals initially planned have been achieved and the possible work that could be done in the future as a continuation of what has been done here.

Introducción

"Lo que hacemos en la vida
tiene su eco en la eternidad"

Gladiator

Sería difícil poder entender el nivel de vida alcanzado hoy en día en nuestra sociedad sin la aparición de la gran cantidad de sistemas electrónicos con los que convivimos: teléfonos móviles, sistemas de navegación y guiado en aviones, ordenadores, sistemas de posicionamiento local (GPS)… Aunque la motivación de este proyecto no viene solo por los sistemas electrónicos que ya conocemos, sino de los que todavía quedan por conocer. En las últimas décadas estamos viviendo una auténtica revolución tecnológica que probablemente tenga como consecuencia la aparición de cada vez más y más dispositivos cuyo centro de operaciones sea un circuito electrónico.

Si bien podemos llegar a pensar que un circuito electrónico afectado por uno o varios fallos puede quedar sin utilidad, esto no tiene por qué ser siempre cierto. Y menos mal, pues como ya predijo Gordon Moore en su famosa ley de 1965, a lo largo de los últimos años se han conseguido integrar cada vez más y más transistores en un mismo área de silicio y ello ha sido gracias a la constante búsqueda de la miniaturización de los llamados "tamaños característicos" de los dispositivos semiconductores, haciendo así que éstos sean cada vez más sensibles a las hostilidades que les rodean como puede ser la acción de la radiación ionizante. Hostilidad (la radiación ionizante) que ya era considerada en el sector espacial pero que desde hace unos años viene siendo preocupante también en el sector aeronáutico e incluso en aplicaciones terrestres cuyo nivel se encuentre próximo al del mar.

Es por ello por lo que se hace necesario el estudio del comportamiento de los circuitos electrónicos ante los posibles fallos que en ellos puedan ocurrir. De esta forma, si conseguimos cuantificar la tolerancia que los circuitos presentan, abriremos la posibilidad de conseguir ahorrar las protecciones que se suponían necesarias antes del estudio y que presentan algunas desventajas como el aumento del coste de fabricación del circuito o el uso de recursos adicionales como área o potencia.

Índice

Índice de Tablas

Índice de Figuras

1 ESTADO DEL ARTE EN INYECCIÓN DE FALLOS

1.1 Técnicas de inyección de fallos

Puesto que la variedad de técnicas de inyección de fallos es muy amplia, merece la pena situar la utilizada en el presente trabajo dentro de las posibilidades existentes. Una forma de clasificar las técnicas de inyección de fallos es la siguiente:

- **Simulada**: el circuito objeto de estudio será simulado por completo mediante ordenador, por lo que la velocidad de simulación dependerá fuertemente de la capacidad computacional de éste.

- **Emuladas sobre FPGA**: en esta técnica el circuito estará implementado en una FPGA (Field Programmable Gate Array) y por ello será necesario que sea descrito mediante un lenguaje de descripción hardware (HDL, de sus siglas en inglés, Hardware Description Language). La prueba contará con la gran velocidad intrínseca de las FPGA, pero por el contrario tradicionalmente ha presentado inconvenientes a la hora de inyectar los fallos y de observar como éstos se propagan por el circuito.

- **Emuladas sobre prototipos**: en este caso se utilizarán dispositivos fabricados en silicio. Presenta la ventaja de contar con el mismo circuito real que finalmente será expuesto a la radiación. No obstante, la configuración del entorno requerirá de mucho más trabajo que en los casos anteriores puesto que cada circuito requerirá una plataforma hardware específica.

La solución que en este documento se expone permitirá el uso de las dos primeras técnicas de inyección de fallos.

A continuación veremos una tabla en la que se comparan los métodos vistos

Técnica		Velocidad	Alcance[a]	Coste	Precisión[b]	Info necesaria[c]	Precisión SW	Alcance Temporal	Tipos de circuito[d]	Notas
Simulada		☹	😐	😊😊😊	😐	☹	☹	😊😊😊	😐	e
Emulada FPGA	Instrumentada	😊😊😊	😐	😊😊	☹☹	☹	😊😊	😊😊😊	😐	f
	No Instrumentada	😊😊	😊😊😊	😊😊	😊😊😊	😐	😊😊	😊😊😊	😊😊😊	
Emulada sobre Prototipo	Pin-Level	😊😊	☹☹	😊😊	😊😊😊	😊😊😊	😊😊😊	☹	😊😊😊	g
	SWIFI	😊😊	☹	😊😊	😊😊😊	😊😊😊	😐	☹	☹☹	h
	Láser	😊😊	😊😊😊	☹	😊😊😊	☹	😊😊	😐[i]	😊😊😊	j

[a] ¿Podemos atacar a todos los bits del diseño?
[b] ¿El circuito sobre el que inyectamos los fallos, es exactamente el mismo que el circuito real? ¿Si no es exactamente igual, difiere mucho o poco de éste?
[c] ¿Necesitamos información específica sobre el diseño (Por ejemplo su descripción VHDL, una netlist post-síntesis o incluso su layout)?
[d] Tipos de circuito que podemos probar. Por ejemplo, si utilizamos SWIFI sólo podremos probar circuitos microprocesador.
[e] La precisión del análisis depende de la precisión del modelo de simulación.
[f] Al instrumentar el MUT estamos realmente atacando un circuito distinto.
[g] Pequeño alcance, muy baja controlabilidad para insertar el fallo.
[h] El circuito a probar debe ser un microprocesador.
[i] Teóricamente el alcance temporal es ilimitado. En la práctica, está limitado por el determinismo y la observabilidad del experimento láser.
[j] La velocidad puede ser muy elevada si se ha utilizado previamente un emulador para localizar las áreas sensibles.

Tabla 1. Comparativa de las distintas técnicas de inyección de fallos. Fuente: [1]

1.2 Modelo de fallo

Como se comentó en la introducción de este documento, no todos los fallos que afectan a un circuito electrónico tienen por qué tener como consecuencia que su funcionamiento no pueda considerarse aceptable. En este apartado veremos los distintos tipos de fallos que pueden afectar a un circuito, su gravedad y especificaremos los utilizados en el presente trabajo[1]:

- **Errores físicos**: también llamados "hard errors". Son los fallos más graves puesto que afectan a la parte física del dispositivo (hardware) de forma permanente y pueden destruir el dispositivo en su totalidad. En estos casos las protecciones deben añadirse mediante tecnología de fabricación.

- **Errores lógicos**: también llamados "soft errors". Son los fallos más subsanables, pues únicamente afectan de forma transitoria a la información del interior del dispositivo (software).

- **El modelo Bit-Flip**: un bit-flip no es más que la inversión de un bit, es decir, si su valor era 0 pasará a ser 1 y viceversa. Este modelo permite estudiar las consecuencias que pueden tener en el circuito los errores lógicos.

En este trabajo se utilizará el modelo Bit-Flip puesto que se pretende analizar el sistema cuando es afectado por la acción de errores lógicos.

[1] Para profundizar puede verse [1] o [7]

2 ANTECEDENTES EN EL ESTUDIO DE TOLERANCIA A FALLOS

"Picasso had a saying. He said good artists copy great artists steal. And we have always been shameless about stealing great ideas"

Steve Jobs

Tradicionalmente, las herramientas utilizadas para el estudio de tolerancia a fallos en circuitos electrónicos han trabajado según el siguiente esquema:

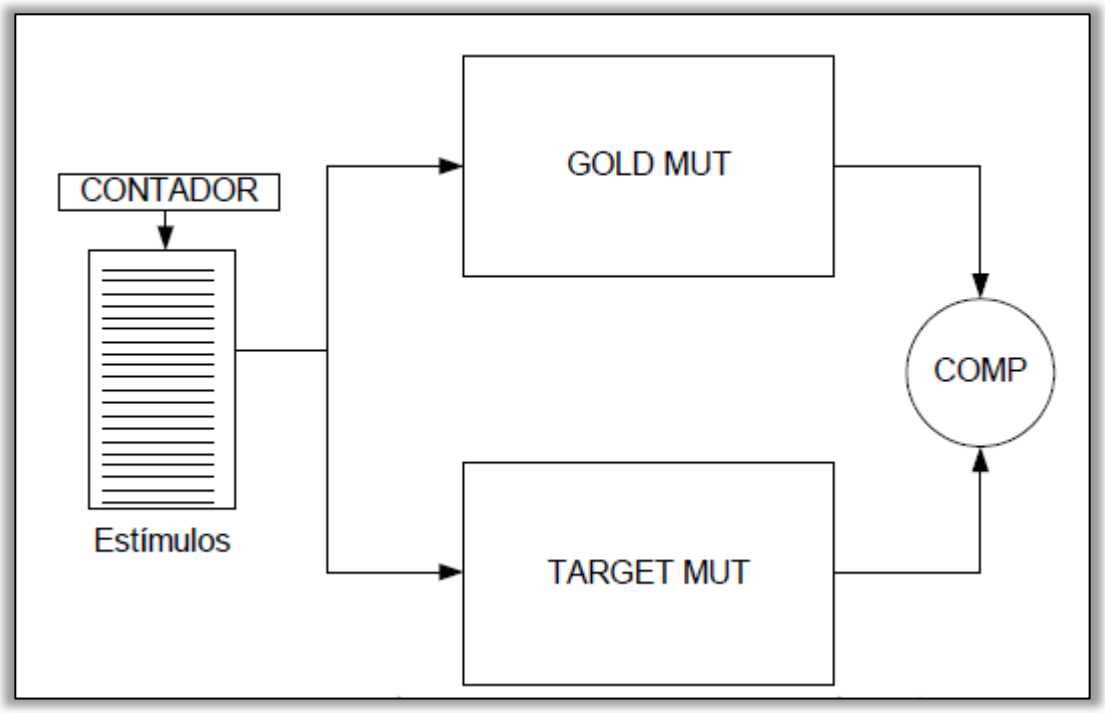


Figura 1. Arquitectura de implementación tradicional. Fuente: [1]

Como puede observarse, durante la realización del estudio se ejecutarán dos diseños en paralelo. Ambos serán prácticamente iguales, con la diferencia de que uno será afectado por la inyección de fallos (TARGET MUT, Module Under Test) y el otro se ejecutará sin ningún tipo de fallo inducido (GOLD MUT).

Este modelo de estudio basa su criterio de análisis en la comparación ciclo a ciclo de ambas salidas, dando por inválido el diseño si éstas no coinciden tanto en tiempo

como en valor. Además, es considerado únicamente el circuito eléctronico y no todo el sistema del que forma parte, por lo que fallos que podrían ser corregidos por el conjunto del sistema son declarados inasumibles por el simple hecho de estudiar solo el circuito. Es por ello que este criterio puede ser considerado demasiado restrictivo para determinadas aplicaciones.

2.1 Relajación en tiempo: Modelo de Tabla Inteligente

Algunos trabajos han surgido con el objetivo de lograr una mayor relajación de las restricciones provocadas por las herramientas tradicionales. Uno de ellos es el Modelo de Tabla Inteligente[2], el cual añade una cierta libertad temporal al criterio anteriormente descrito.

Este modelo consiste en almacenar en una tabla los datos de salida que obtendría el diseño GOLD MUT y el instante en el que éstos ocurren. Una vez se tiene creada dicha tabla, el usuario deberá establecer el parámetro más importante, el Tiempo de Recuperación Crítico, que es el tiempo máximo que se permite al circuito para que saque el valor correcto a la salida.

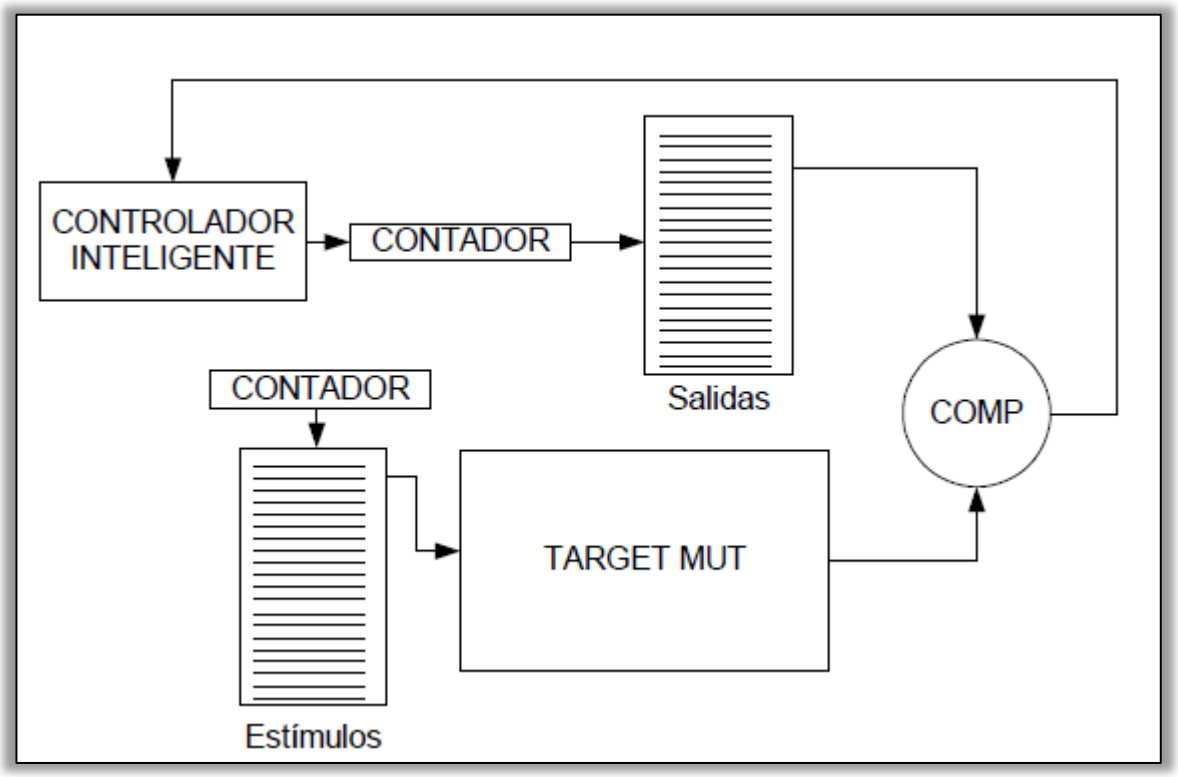


Figura 2. Arquitectura de implementación con modelo de Tabla Inteligente. Fuente: [1]

[2] Para más información puede leerse [1] o [6]

2.2 Relajación en tiempo y en valor: Este trabajo

Lo que en este documento se propone va un paso más allá y consigue añadir un grado más de libertad a la hora de comparar el modelo con fallos y el modelo sin fallos, de modo que el valor que se obtenga a la salida pueda no ser exactamente el esperado pero aun así pueda resultar válido para el correcto funcionamiento del sistema.

A continuación veremos una tabla en la que se clasifican los distintos modelos de comparación según sean estrictos o flexibles tanto en tiempo como en valor:

		Tiempo	
		Estricto	Flexible
Valor	Estricto	Comparación Ciclo a Ciclo	Modelo de Tabla Inteligente
	Flexible	Comparación Ciclo a Ciclo teniendo en cuenta CRC[3]. EDAC[4]	**Este trabajo**

Tabla 2. Clasificación de técnicas según restricción en tiempo y valor

[3] CRC (Cyclic Redundancy Checking) puede corregir errores además de detectarlos.
[4] Acrónimo de Error Detection And Correction

2.3 Comparación gráfica entre herramientas tradicionales y este trabajo

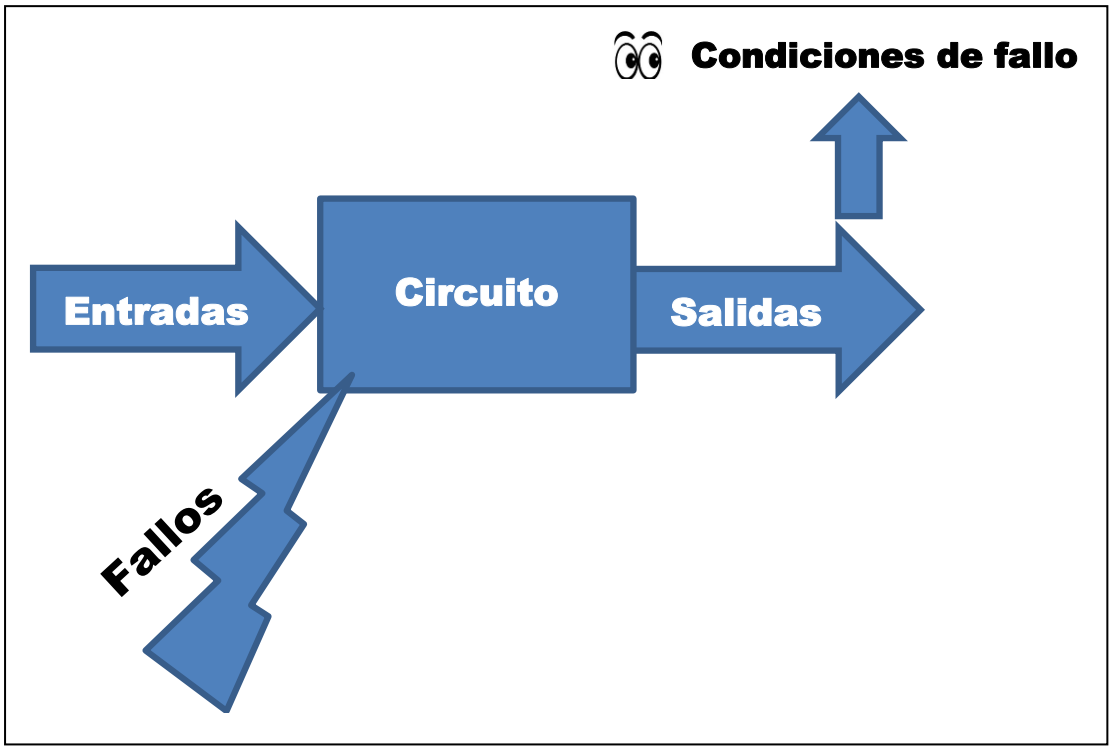


Figura 3. Estructura seguida en herramientas tradicionales

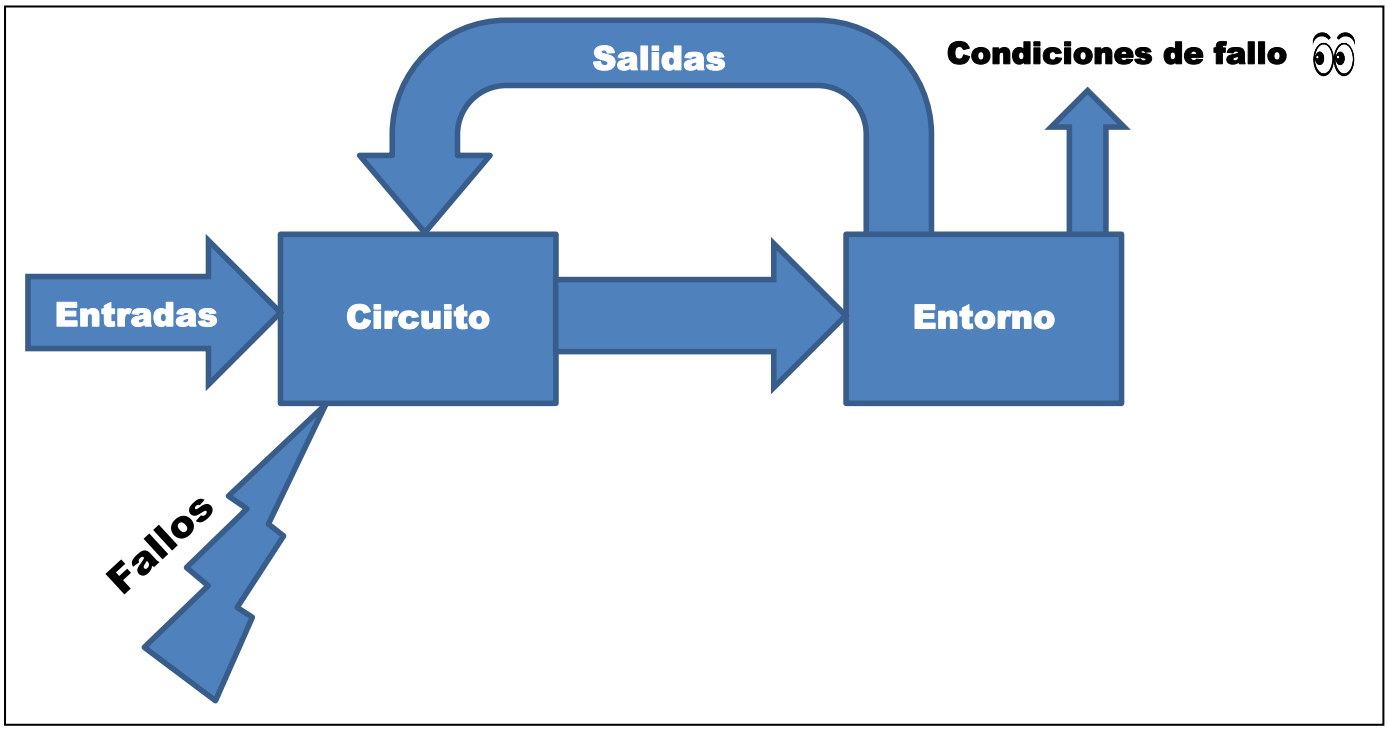


Figura 4. Estructura Hardware-In-the-Loop utilizada en este trabajo

3 ALCANCE DEL PRESENTE TRABAJO

El principal objetivo con el que nace este trabajo es el de reproducir y estudiar el comportamiento de un sistema electrónico completo cuando se ve afectado bajo los efectos de la radiación ionizante. Cabe destacar que cuando se habla de un sistema electrónico completo ello se refiere a que no solo se analizará cómo responde un circuito electrónico al inyectarle fallos, pues en este caso existen ya estudios al respecto, sino que en el presente trabajo el circuito será solo una parte del sistema que se analiza. Es por ello que fallos que podrían ser declarados como intolerables si consideramos solo el circuito, pueden ser declarados tolerables si analizamos el sistema en su conjunto.

Para llevar a cabo dicha idea se va a montar un entorno sencillo pero muy práctico e ilustrativo que estará formado por un controlador (el cual será el circuito al que se inyectarán los fallos) y que se conectará con un péndulo invertido formando un sistema en bucle cerrado.

El controlador estará descrito en el lenguaje de descripción hardware VHDL[5] y podrá ser simulado por ordenador o emulado en una FPGA. Por otro lado, el péndulo invertido será modelado por software gracias a la herramienta Simulink. De esta forma se montará un entorno Hardware-In-the-Loop (HIL) en el que el hardware de la FPGA y el software del ordenador se unen creando un sistema híbrido con gran potencial para el estudio de la tolerancia a fallos en circuitos electrónicos.

Queda fuera del alcance de este trabajo (y se sugiere como un trabajo futuro) la integración del sistema con herramientas de inyección de fallos ya existentes.

[5] Acrónimo resultado de la combinación de VHSIC (Very High Speed Integrated Circuit) y HDL (Hardware Description Language)

4 SOLUCIÓN PROPUESTA

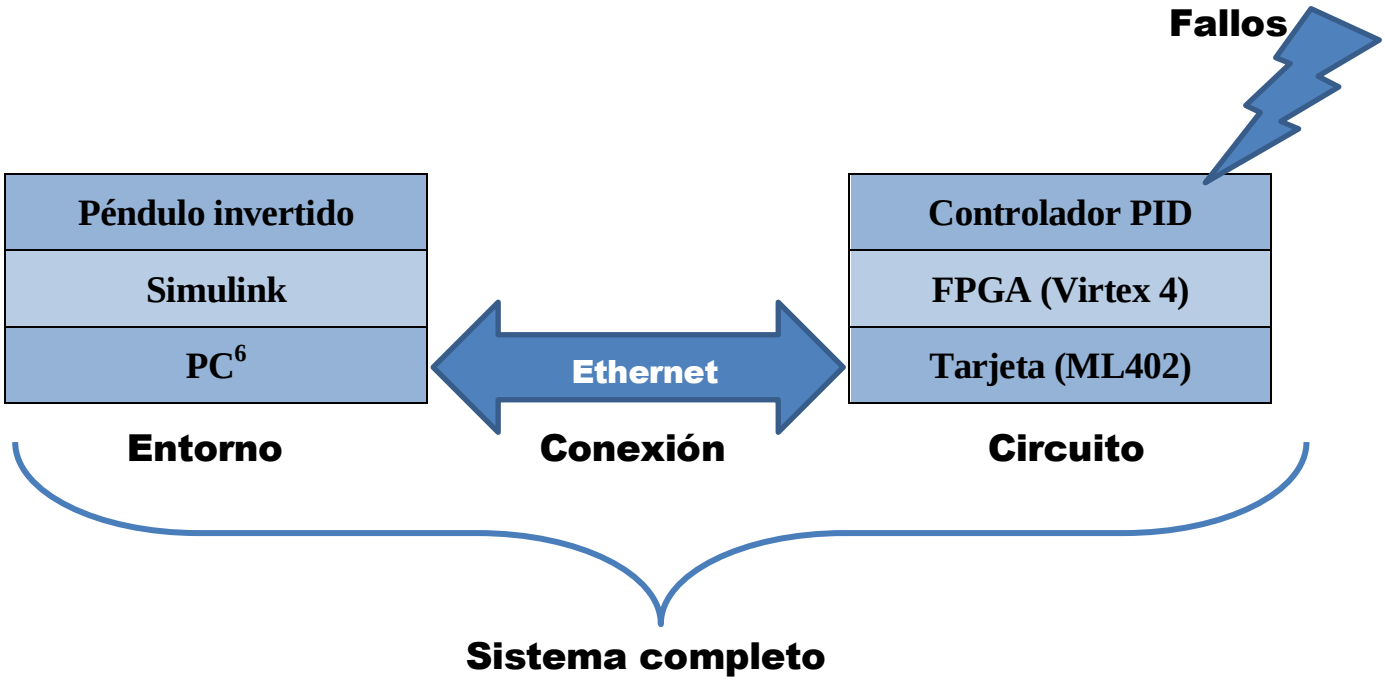
"If I had asked my customers what they wanted, they would have said: a faster horse"

Henry Ford

Lo que veremos a continuación será cómo se ha conseguido materializar la idea anteriormente descrita. En el presente capítulo comentaremos los pasos a seguir para llevar a cabo el estudio del sistema electrónico así como el método de inyección de fallos utilizado, las herramientas que han servido de apoyo y la descripción a fondo de todos los componentes que forman parte del entorno con el que haremos dicho estudio.

4.1 Esquema Hardware-In-the-Loop

El siguiente esquema representa la estructura y componentes que conforman la solución que se ha llevado a cabo en este trabajo:



[6] Procesador Intel Core i7-3610QM @ 2.3GHz y 8 GB de RAM

4.2 Procedimiento de simulación

Los pasos que se deben seguir para realizar el análisis mediante esta solución son los siguientes:

1. **Ejecutar el Script de Configuración**. Es un script realizado en MATLAB (lenguaje M) que presenta un menú mediante el cual se permite al usuario elegir el tipo de análisis que se quiere realizar. Una vez se ha ejecutado correctamente, se habrán configurado las variables del entorno y quedará listo para ser simulado.

2. **Correr la simulación**. Una vez tenemos configurado el entorno, ya podemos ejecutar la simulación en el modelo de Simulink. Durante la realización de este paso se podrá ir viendo en tiempo real el comportamiento que va teniendo el sistema ante los fallos que se le van inyectando. Una vez terminado este paso ya tendremos todos los datos guardados en el "Worskpace" de MATLAB, listos para ser analizados.

3. **Ejecutar el Script de Análisis**. Es otro script en MATLAB. Con él se estudiarán los datos obtenidos en el Paso 2 mediante gráficas y análisis estadísticos.

4.3 Herramientas utilizadas

Para la realización del presente trabajo ha sido indispensable el uso de las siguientes herramientas:

4.3.1 MATLAB/Simulink

Desde que fuese creado en 1984 por Cleve Moler, MATLAB (nombre abreviado de MATrix LABoratory) ha conseguido posicionarse como uno de los productos software de cálculo numérico más usados dentro tanto del ámbito académico como del empresarial. Cuenta con su propio lenguaje de programación (lenguaje M) el cual se caracteriza por ser interpretado y bastante intuitivo. Dentro del paquete MATLAB podemos encontrar una herramienta adicional, Simulink, la cual consiste en un entorno de programación visual con mucho más alto nivel de abstracción que el mencionado lenguaje M.

Ambas herramientas, MATLAB y Simulink, serán la base del presente proyecto puesto que será en Simulink donde se encontrará el entorno objeto de estudio y donde se llevarán a cabo las simulaciones y, por otro lado, será gracias a varios scripts de MATLAB que se podrá hacer la configuración del mencionado entorno y el análisis de los datos que sus simulaciones registran.

La versión de MATLAB/Simulink utilizada en este trabajo ha sido la R2013b.

4.3.2 Xilinx ISE

Xilinx ISE (Integrated Synthesis Environment) es un software proporcionado por la empresa norteamericana de semiconductores "Xilinx, Inc" para la síntesis y análisis de diseños en lenguaje de descripción hardware (HDL). En este trabajo se ha utilizado por dos motivos: el primero porque es necesario para poder utilizar correctamente System Generator for DSP (se describe en el siguiente apartado) junto con MATLAB y el segundo motivo porque se ha utilizado su editor de texto para desarrollar el código VHDL del controlador PID. Si bien para desarrollar dicho código podría haberse optado por otra herramienta, como por ejemplo el editor de texto que incorpora MATLAB, se ha elegido finalmente el de Xilinx ISE por la razón principal de incorporar la verificación de sintaxis y la capacidad de sintetizar el diseño proporcionando así más información acerca del circuito.

La versión de Xilinx ISE utilizada en este trabajo ha sido la 14.7[7]

4.3.3 System Generator for DSP

Es la herramienta característica y diferenciadora de este proyecto.

System Generator for DSP (o simplemente System Generator) es un software también creado por "Xilinx, Inc" y que para su uso debe ser añadido a modo de plugin a Simulink. Entre sus principales utilidades están la creación de diseños en lenguaje HDL con un alto nivel de abstracción o también lo que es llamado Hardware Co-Simulation (o Hardware-In-the-Loop), que consiste en la simulación conjunta de Simulink y una plataforma hardware.

Para el presente trabajo, System Generator se ha utilizado para emular el código VHDL del controlador PID junto con el modelo en Simulink del péndulo invertido. Por lo tanto ha sido gracias a esta herramienta que se ha podido crear un entorno híbrido caracterizado por la comunicación en tiempo real de una FPGA junto con un sistema modelado por software.

Existen varias alternativas a System Generator para la creación de entornos Hardware-In-the-Loop entre las cuales se encuentran:

- FPGA-in-the-loop: herramienta proporcionada por MATLAB con características similares a System Generator.
- Código MATLAB: aunque infinitamente más tedioso, la creación de una interfaz FPGA-MATLAB/Simulink basado en únicamente líneas de código MATLAB es posible.

Una vez vistas las herramientas, vamos a proceder a describir los distintos componentes que forman el entorno de estudio.

[7] Aun existiendo versiones posteriores del software en el momento de realizar el trabajo, se ha utilizado esta versión por ser la última que proporciona soporte a la tarjeta utilizada (ML402)

4.4 Modelo del péndulo

Como ya se ha comentado anteriormente, el péndulo será modelado mediante software utilizando la herramienta Simulink.

4.4.1 Origen del péndulo

Puesto que el desarrollo de un modelo que simule el comportamiento de un péndulo invertido no es objetivo del presente trabajo, se ha optado por utilizar uno ya existente. Concretamente el modelo que se ha utilizado es proporcionado por la propia herramienta Simulink y se puede encontrar con sólo escribir el comando "penddemo" en la línea de comandos de MATLAB.

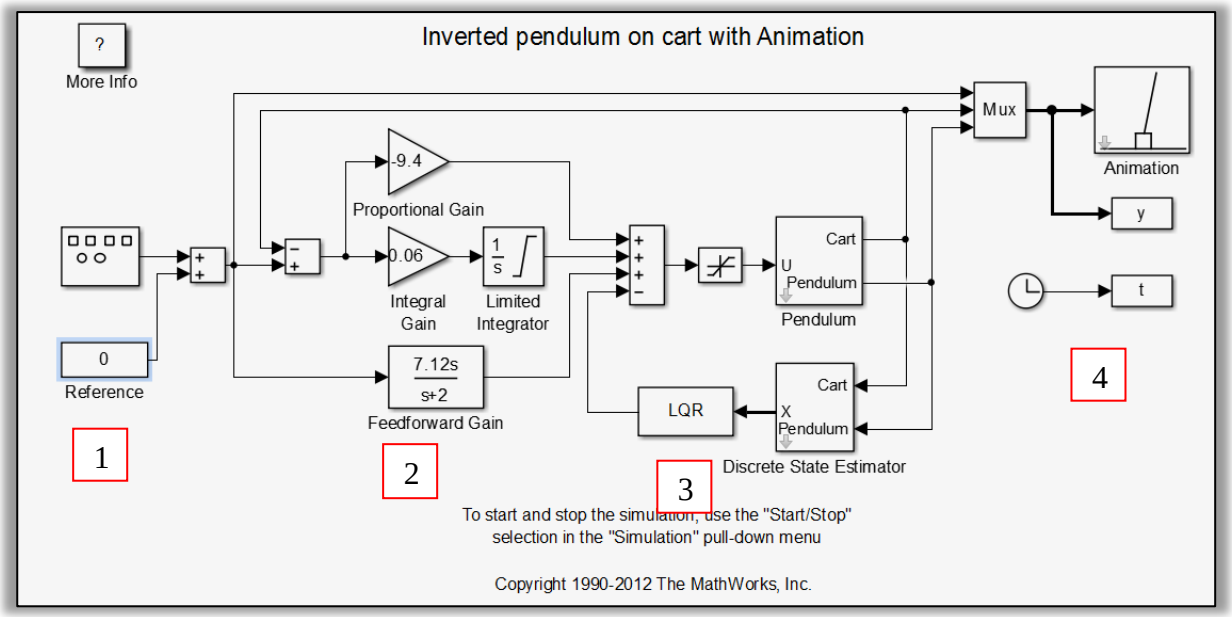


Figura 5. Ejemplo "penddemo" de MATLAB. Por orden: entrada, controlador, péndulo y salida

Aunque a priori pueda parecer un modelo simple, en él encontramos una buena base de la que partir para la creación del entorno que queremos puesto que ambos cuentan con la misma estructura: datos de entrada del sistema, controlador, péndulo invertido y datos de salida.

También cabe destacar que es en este ejemplo donde encontramos la representación gráfica de la simulación del sistema.

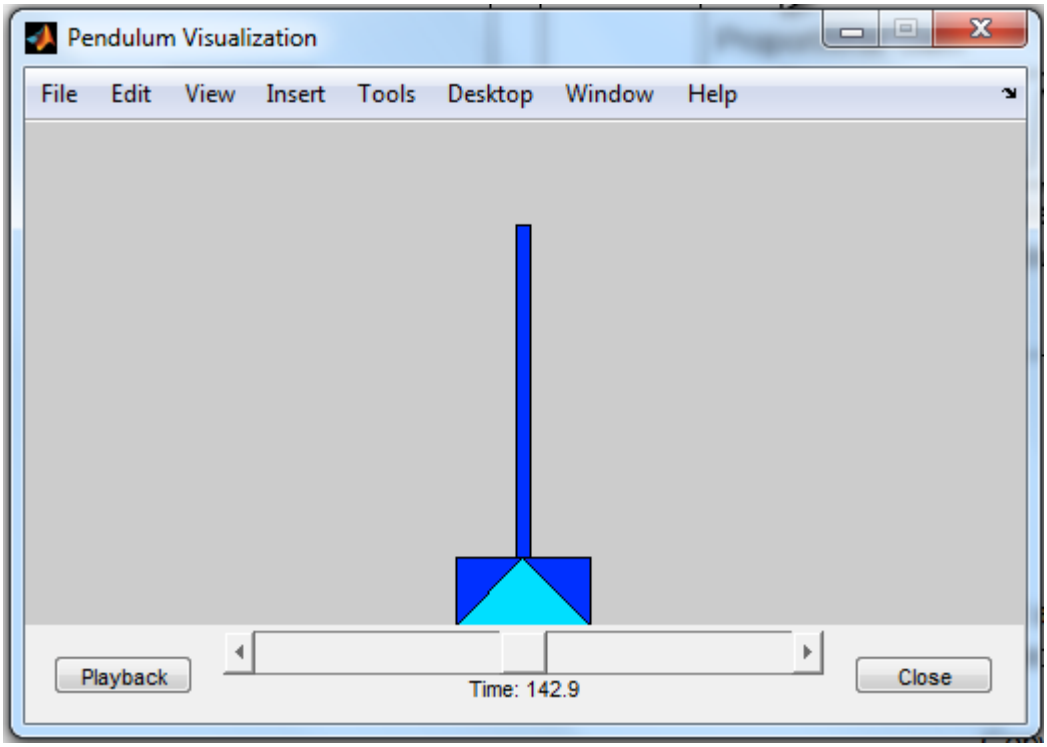


Figura 6. Ventana que muestra el comportamiento del sistema.En azul claro la referencia. En azul oscuro el carrito más el péndulo

Gracias a ello podemos ver **en tiempo real** el comportamiento que tiene el sistema ante los fallos que se le van inyectando.

4.4.2 Características del modelo

A continuación analizaremos más en profundidad el modelo del péndulo que estamos utilizando y veremos cómo se ha realizado para así poder entender su funcionamiento.

Si entramos dentro del péndulo podremos observar lo siguiente:

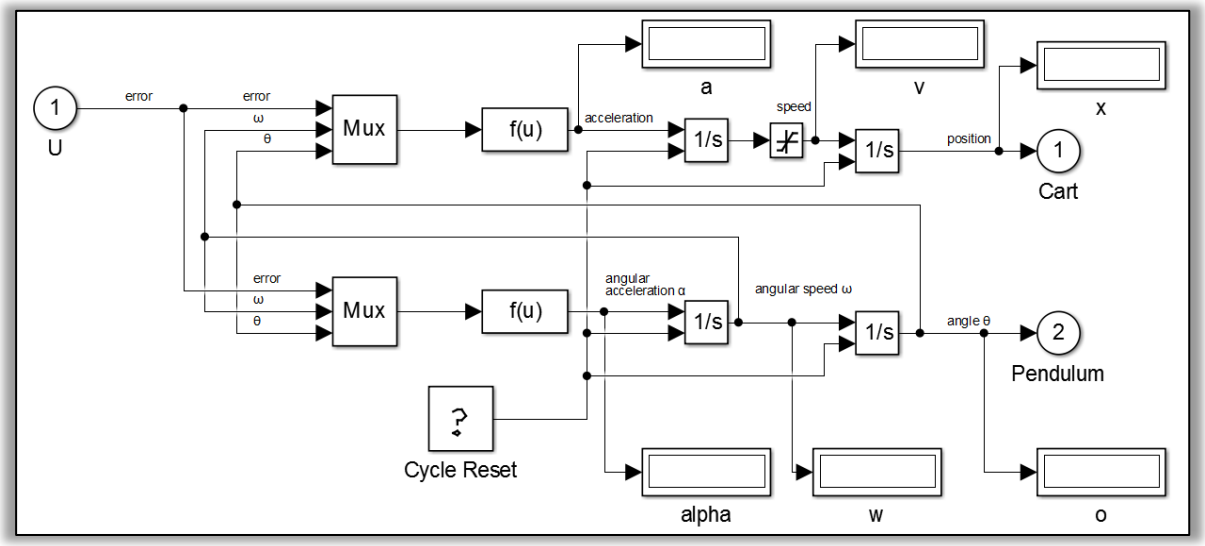


Figura 7. Vista interna del péndulo

Como podemos ver, es un sistema en bucle cerrado que cuenta con una entrada y dos salidas: a la entrada recibe la salida del controlador y en las salidas se obtienen la posición del péndulo invertido y del carrito respectivamente. Para simular el comportamiento de ambos se sigue la misma estructura: la salida del controlador junto con otras señales internas se pasan como entradas a un bloque Simulink que calcula la aceleración del péndulo y del carrito a partir de una ecuación para cada uno.

4.4.2.1 Ecuaciones para el péndulo y carrito

Ecuación para la aceleración del péndulo:

$$\alpha = \frac{\frac{-e\cos(\theta)}{m} + \frac{(M_{cart}+m)*g*\sin(\theta)}{m} - l*\omega^2*\sin(\theta)*\cos(\theta)}{l*\left(\frac{M_{cart}}{m}+\sin^2(\theta)\right)} \quad (3\text{-}1)$$

Ecuación para la aceleración del carrito:

$$a = \frac{\left(\frac{e}{m} - g*\sin(\theta)*\cos(\theta) + l*\omega^2*\sin(\theta)\right)}{\frac{M_{cart}}{m}+\sin^2(\theta)} \quad (3\text{-}2)$$

Variables

- e: entrada
- l: distancia del péndulo al centro de masas del carrito
- M_{cart}: masa del carrito
- m: masa del péndulo
- g: gravedad
- θ: posición angular del péndulo
- ω: velocidad angular del péndulo

Una vez obtenida la aceleración, ésta se hace pasar por un par de integradores con el fin de obtener el valor de la posición de ambos componentes.

También cabe mencionar el rango de valores posibles en cuanto a la posición tanto del péndulo como del carrito.

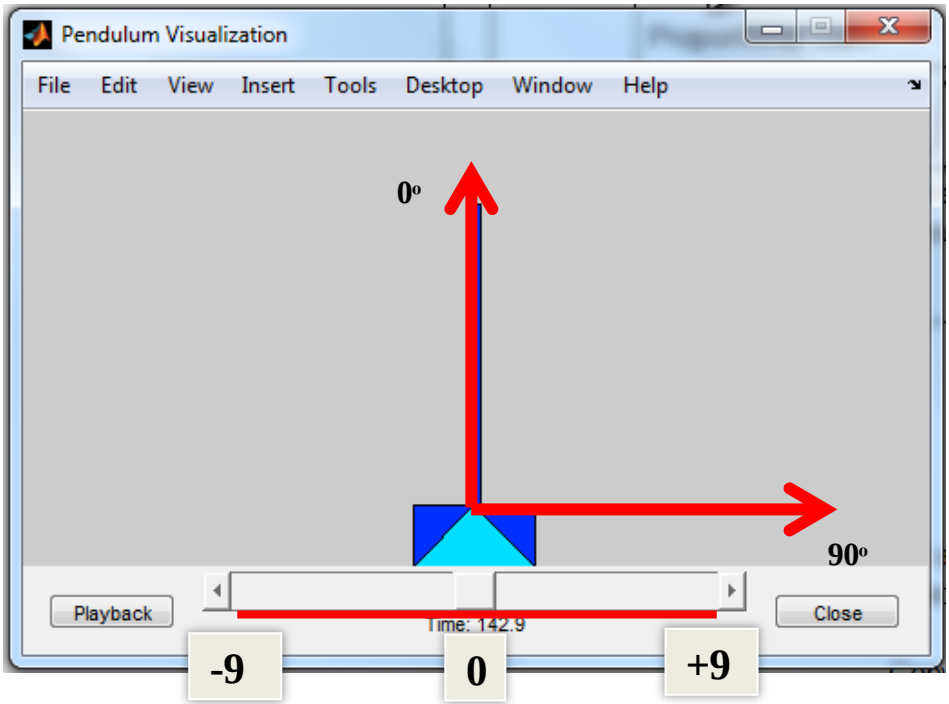


Figura 8. Rango de valores que muestra la ventana de simulación

En cuanto al péndulo su posición de partida es el eje vertical y puede tomar valores desde 0 a 360º. Con respecto al movimiento horizontal del carrito, éste puede desplazarse desde menos infinito hasta más infinito, aunque la ventana que muestra la simulación abarca desde -9 hasta +9.

4.5 Controlador PID

Una vez que sabemos el péndulo que se ha utilizado, es turno de hablar del otro componente que forma nuestro entorno: el controlador.

El controlador será un circuito electrónico digital que será simulado por ordenador o emulado en una FPGA y será aquí donde realicemos la inyección de fallos a nuestro sistema.

4.5.1 Origen del controlador

Por su versatilidad, facilidad de implementación y buen rendimiento en la mayoría de sistemas, el controlador elegido para este trabajo ha sido de tipo PID. Puesto que dicho controlador será implementado en una FPGA, se necesitará de un lenguaje de descripción hardware para su desarrollo, habiendo sido elegido el lenguaje VHDL y cuyo código se adjunta en Anexo A - Código VHDL del Controlador PID.

A diferencia del péndulo, el controlador se ha realizado desde cero, puesto que los controladores encontrados en varias páginas web[8] resultaron estar optimizados para aplicaciones concretas sin apenas poder ser configurables por el usuario.

[8] OpenCores y GitHub

4.5.2 Características

4.5.2.1 Representación

En cuanto a la aritmética, ha sido elegido el punto fijo y se debe principalmente por dos razones: la primera a que el punto flotante, la otra alternativa, es más difícil de implementar (aunque existen librerías que lo facilitan, ver [2]) y la segunda y más importante se debe que el punto flotante puede provocar una pérdida de precisión en los cálculos (para demostración ejecutar código de ejemplo encontrado en Anexo B – Ejemplo de pérdida de precisión en punto flotante)

4.5.2.2 Entidad

El controlador consta de 10 entradas y de 4 salidas que a continuación se describen.

Entradas:

Nombre	Bits	Comentario
clk	1	Reloj
ce	1	Clock enable, activo a nivel alto
rst	1	Reset, activo a nivel alto
error	16	Diferencia entre referencia y carrito
kp	16	Ganancia del elemento proporcional
ki	16	Ganancia del elemento integral
kd	16	Ganancia del elemento derivativo
inject	1	Determina si se inyectan fallos • '0' → No inyecta • '1' →Inyecta
reg_id	16	Registro al que se inyectan los fallos • 0 → Proporcional • 1 → Integral • e.o.c → Derivativo
index	16	Posición del bit que se le hará el bit-flip

Tabla 3. Puertos de entrada del controlador

Salidas:

Nombre	Bits	Comentario
data_out	32	Datos de salida
inject_out	1	Mismo valor que inject
reg_id_out	16	Mismo valor que reg_id
index_out	16	Mismo valor que index

Tabla 4. Puertos de salida del controlador

4.5.2.3 Arquitectura

Como se ha comentado anteriormente, el controlador será de tipo PID por lo que su implementación será bastante sencilla. Contará únicamente con dos procesos: uno asíncrono y otro síncrono.

En el proceso asíncrono se realizarán los cálculos propios del controlador, y, aunque hecho de cero, para su creación ha sido de gran importancia la lectura de varias referencias como [3] o [4]. En la siguiente imagen podemos ver el sencillo algoritmo que sigue el funcionamiento de un controlador de tipo PID:

```
previous_error = 0
integral = 0
start:
  error = setpoint - measured_value
  integral = integral + error*dt
  derivative = (error - previous_error)/dt
  output = Kp*error + Ki*integral + Kd*derivative
  previous_error = error
  wait(dt)
  goto start
```

Figura 9. Algoritmo para implementar un controlador de tipo PID. Fuente: [4]

En el proceso síncrono será donde se haga el cambio de valor a la salida del controlador, implementando así biestables que serán a los que se les hará la operación de “bit-flip” ya comentado.

4.5.3 Instrumentación del controlador

En este apartado se va a describir el código que se ha añadido al controlador para la inyección de fallos. Puesto que tanto como en simulación con ordenador como con emulación con FPGA el controlador estará descrito en VHDL, la forma de instrumentarlo será la misma en los dos casos.

4.5.3.1 Código VHDL

El siguiente recuadro contiene las líneas de código principales con las que se consigue la inyección de fallos en el controlador.

```
-- Faults injection
   if (inject = '1') then
      case to_integer(signed(reg_id)) is
         when 0 =>       -- inject in yp
            yp(index2) <= not(p_yp(index2));
         when 1 =>       -- inject in yi
            yi(index2) <= not(p_yi(index2));
         when others =>  -- inject in yd
            yd(index2) <= not(p_yd(index2));
      end case;
    end if;
```

Figura 10. Código VHDL para inyectar fallos al controlador

Como se ha comentado anteriormente, puesto que el fallo que queremos inyectar consiste en cambiar el valor del bit de un biestable, las líneas anteriores deben ser añadidas en la parte final del proceso síncrono de nuestro código.

4.6 Entorno completo

Aunque el ejemplo del péndulo proporcionado por MATLAB es una gran base de la que partir, las características del presente trabajo hacen que se deban hacer modificaciones las cuales se pasan a detallar en esta sección.

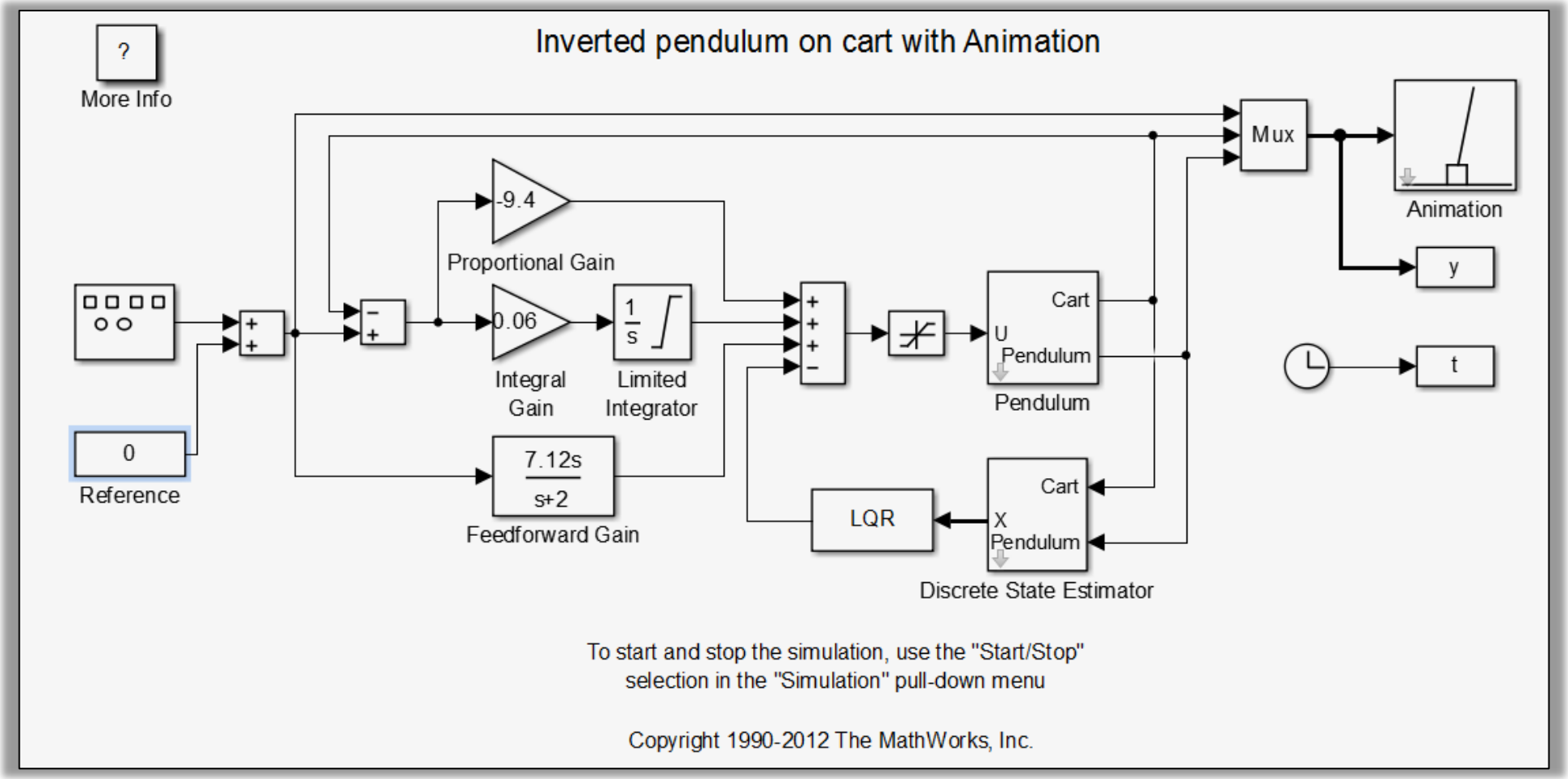


Figura 11. Ejemplo “penddemo” de MATLAB

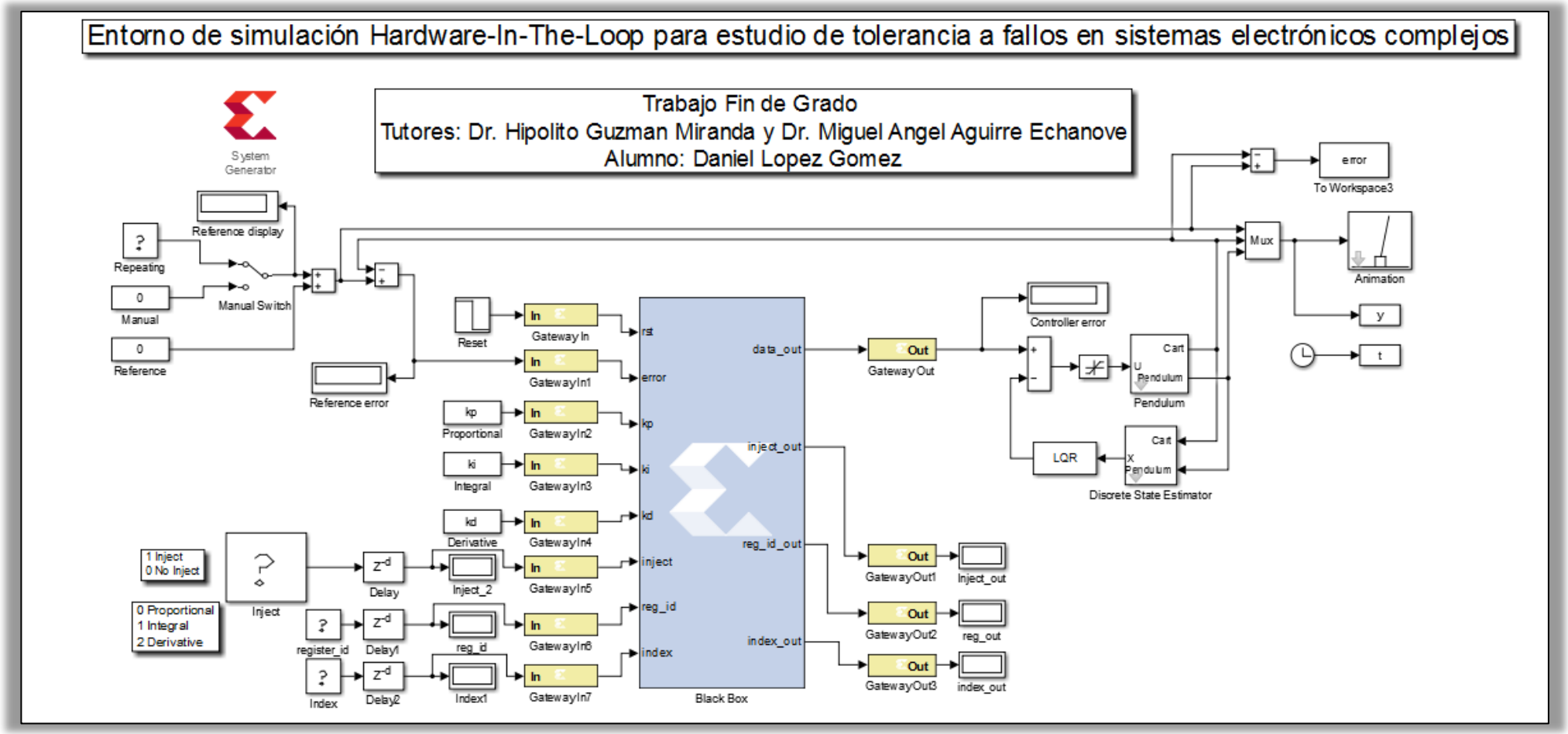
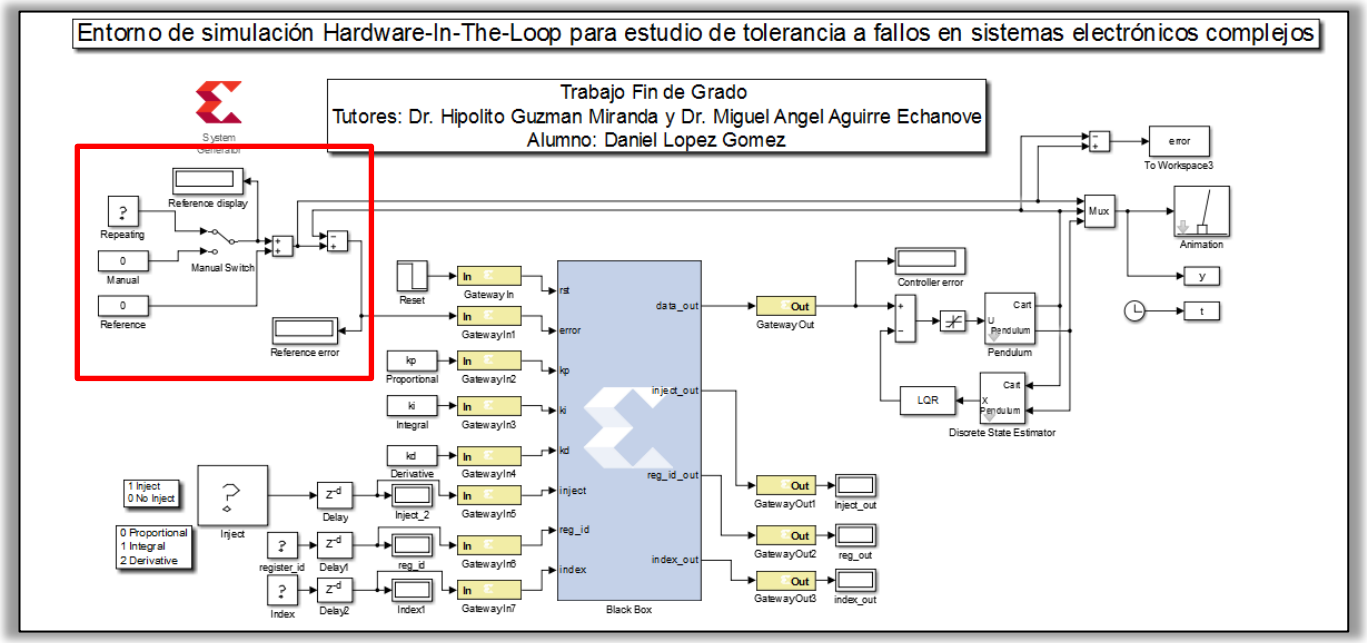


Figura 12. Entorno Simulink para la simulación del sistema

4.6.1 Entrada

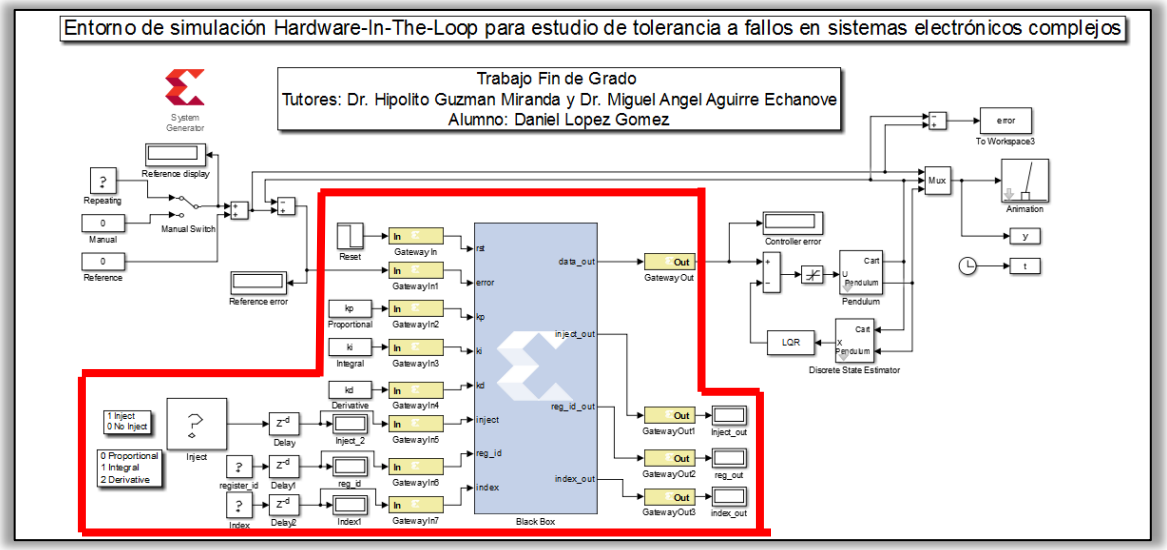
A la entrada del sistema se proporciona un vector con una serie de valores que serán las referencias de posición que debe alcanzar el péndulo en cada Run[9]. Tanto los valores como la longitud del mencionado vector serán modificables por el usuario gracias al script MATLAB de configuración del entorno.



4.6.2 Controlador

El método para incluir el código VHDL del controlador dentro del entorno con el que estamos trabajando es bastante fácil puesto que la herramienta System Generator incluye un bloque Simulink destinado a tal fin. Dicho bloque es denominado "Black Box" y permite incluir VHDL, Verilog o EDIF[10] dentro de un diseño. Una vez que hemos asociado la Black Box al código que queremos incluir, System Generator crea un script de configuración que permite la correcta interpretación del mismo. Aunque dicho script es generado en buena parte automáticamente, dependiendo de la complejidad del código deberá ser manipulado por el diseñador para especificar parámetros tales como: lenguaje incluido, tipo de puertos de entrada y salida, reloj del sistema...

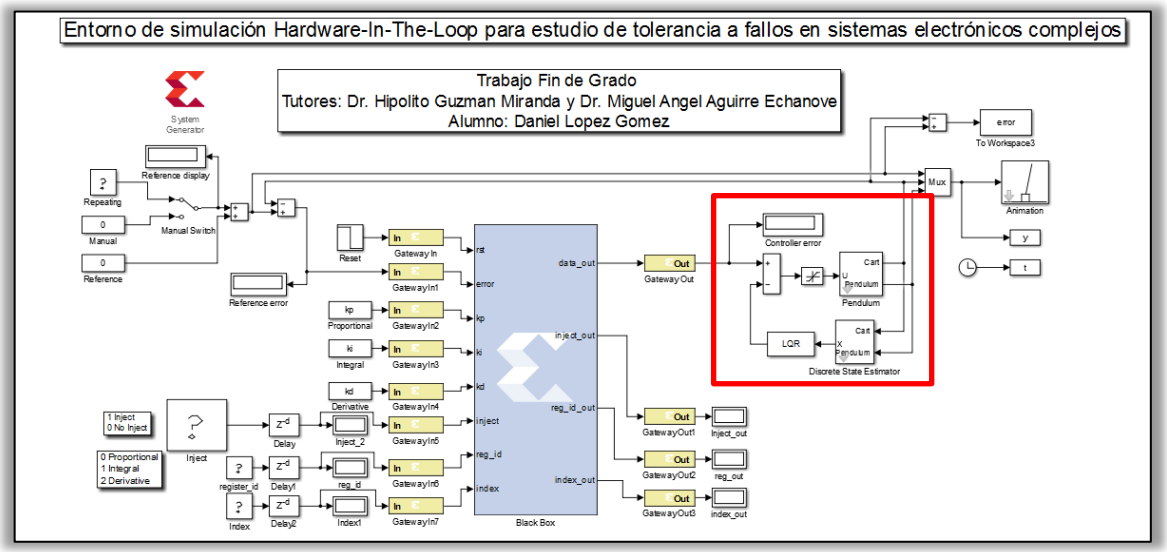
[9] Un Run es una única ejecución del diseño durante los vectores de test completos (en este caso, los vectores de test es el conjunto de referencias que el péndulo debe alcanzar) mientras la cual pueden inyectarse fallos. A un conjunto de Runs se le llama Campaña.
[10] Acrónimo de Electronic Design Interchange Format



4.6.3 Péndulo

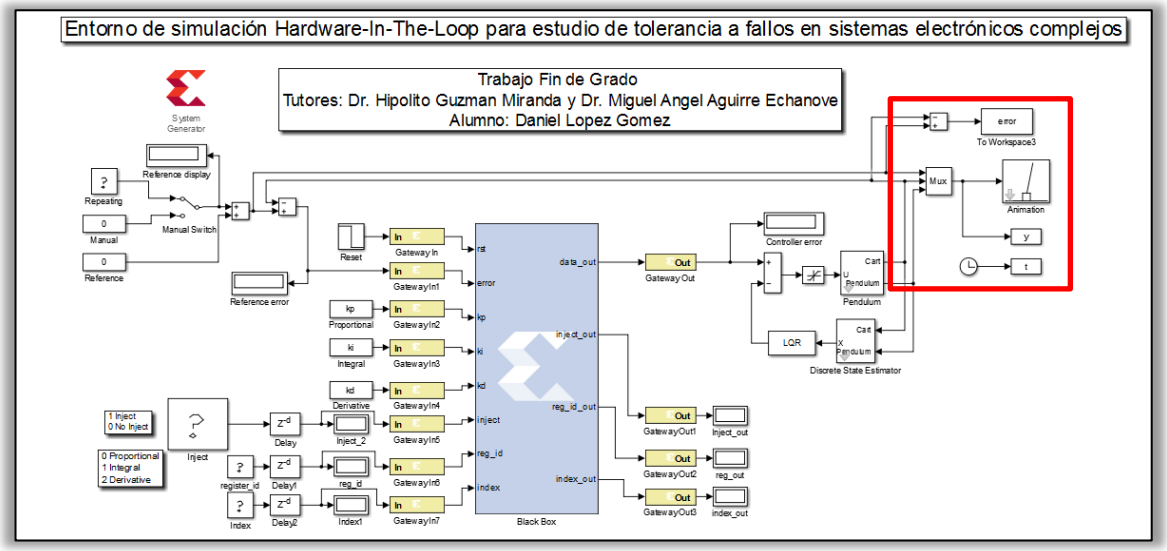
Como se ha comentado en capítulos anteriores, el péndulo que se ha utilizado para este trabajo es el proporcionado por MATLAB mediante el comando "penddemo". Solo se han realizado dos pequeñas modificaciones al mismo y han sido:

- Añadir un reset al principio de cada Run puesto que de no hacerlo el péndulo no vuelve nunca a su estado de reposo una vez descontrolado.
- Añadir un limitador a la velocidad del carrito ya que, en caso de que se descontrole, alcanza valores muy altos en muy poco tiempo, generando así una situación poco realista.



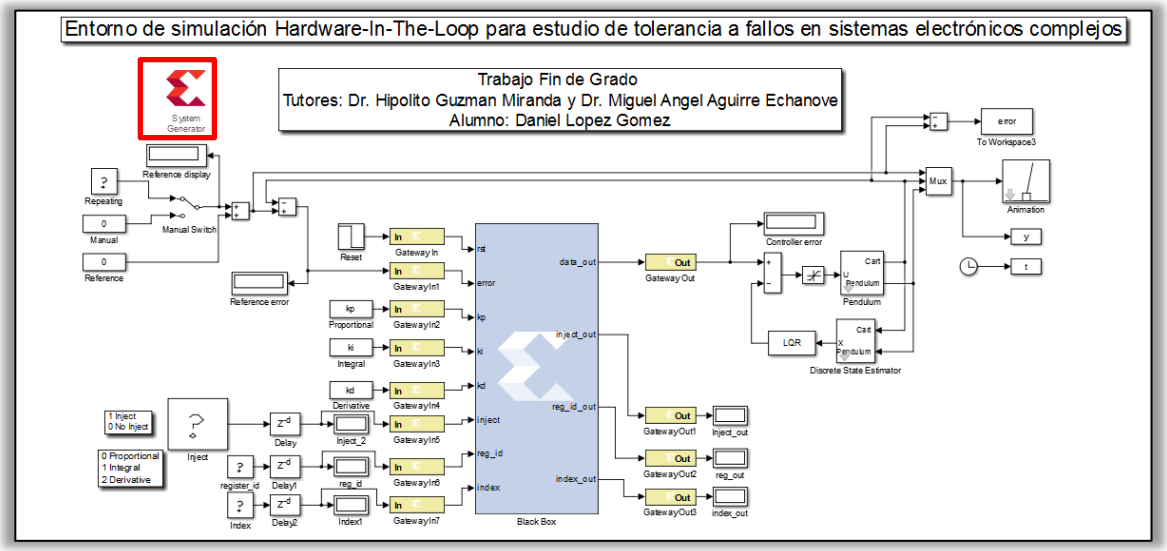
4.6.4 Salida

Para mejorar el análisis del sistema se ha añadido un bloque "to_workspace" de la librería de Simulink para así obtener los datos en una estructura que facilitará el manejo de los datos obtenidos.



4.6.5 System Generator Token

System Generator Token es un bloque que debe estar presente en todos los diseños en los que se utilice System Generator. Su cometido es configurar parámetros del diseño tales como: tarjeta en la que se implementará el diseño, lenguaje HDL utilizado, frecuencia de reloj del sistema…



4.7 Scripts MATLAB

4.7.1 Script de configuración del entorno: config_v2.m

Con el objetivo de facilitar la configuración del entorno se ha creado un script MATLAB el cual presenta un menú al usuario con el que se inicializan las variables para realizar el análisis que se desee.

Al ejecutar el mencionado script aparece un menú que permite elegir:

- **Tipo de análisis**
  - **Single Run:** también llamado modo Campaña. En este modo el sistema realizará un Run sin fallos (denominado Golden Run) e inmediatamente después realizará más (la cantidad depende de lo que el usuario seleccione) en los que el sistema se verá afectado por fallos. Al comienzo de cada Run el entorno será reseteado, de modo que todos los Runs se ejecutarán con las mismas condiciones iniciales, evitando así que se afecten entre ellos. El modo Single Run sirve para analizar por separado los efectos de cada fallo y, gracias a realizar Runs con y sin fallos, en este modo se podrán realizar comparaciones entre ellos.
  - **Endless:** este tipo de análisis tiene por objetivo emular un entorno de radiación en el que los fallos puedan acumularse. Es por ello que en este modo no se realizarán comparativas (de hecho no se ejecutará el Golden Run). Por el contrario, se ejecutarán una serie de Runs (la cantidad es elegida por el usuario) en el que se irán inyectando fallos de forma aleatoria.
- **Registro(s)**
  - **Proporcional**
  - **Integral**
  - **Derivativo**
  - **(Todos):** Solo puede elegirse en el modo Endless
- **Bit(s)**
  - **Un solo bit**
  - **Parte entera:** bits desde la posición 16 hasta la 31
  - **Parte decimal:** bits desde la posición 0 hasta la 15
  - **Todo el registro:** bits desde la posición 0 hasta la 31
- **Tipo de fallo en cada Run**
  - **Constante**: el bit elegido será siempre invertido
  - **Probabilidad**: el usuario elige probabilidad con la que aparecerán fallos
  - **Número exacto de fallos**: el usuario elige el número exacto de fallos y estos se repartirán aleatoriamente en cada Run

Aunque también si se edita el mencionado script permite al diseñador cambiar parámetros como:

- Valores de referencia en cada Run para el péndulo
- Tiempo entre referencias
- Parámetros del controlador
- …

4.7.2 Script de análisis de los resultados: analysis_v2.m

Una vez que se ha realizado la simulación del sistema, se puede comprobar el comportamiento que ha tenido el mismo gracias a este script. Para ello hace uso de otros scripts, cuya funcionalidad veremos a continuación:

4.7.2.1 Datos estadísticos: statistical_data.m

Se encarga de mostrar por pantalla parámetros estadísticos, más concretamente la media y desviación de la diferencia entre el Golden Run y el Run con fallos en cada bit, además de una gráfica bastante ilustrativa en la que se puede ver la influencia de cada bit en la estabilidad del entorno.

```
Bit=0   |   Mean=6.082959e-07   |   Deviation=5.784885e-07
Bit=1   |   Mean=1.216594e-06   |   Deviation=1.156979e-06
Bit=2   |   Mean=2.433183e-06   |   Deviation=2.313957e-06
Bit=3   |   Mean=4.866371e-06   |   Deviation=4.627915e-06
Bit=4   |   Mean=9.732739e-06   |   Deviation=9.255829e-06
Bit=5   |   Mean=1.946548e-05   |   Deviation=1.851165e-05
Bit=6   |   Mean=3.893093e-05   |   Deviation=3.702328e-05
Bit=7   |   Mean=7.786181e-05   |   Deviation=7.404646e-05
Bit=8   |   Mean=4.093907e-04   |   Deviation=6.299116e-04
Bit=9   |   Mean=4.362396e-04   |   Deviation=8.155429e-04
Bit=10  |   Mean=4.839911e-04   |   Deviation=7.711288e-04
Bit=11  |   Mean=6.995601e-05   |   Deviation=3.085635e-04
Bit=12  |   Mean=8.444647e-04   |   Deviation=1.275219e-03
Bit=13  |   Mean=4.092665e-04   |   Deviation=1.038468e-03
Bit=14  |   Mean=6.957982e-04   |   Deviation=1.801882e-03
Bit=15  |   Mean=1.445011e-03   |   Deviation=3.684865e-03
Bit=16  |   Mean=3.185559e-03   |   Deviation=9.993166e-03
Bit=17  |   Mean=5.926989e-03   |   Deviation=1.978389e-02
Bit=18  |   Mean=1.167931e-02   |   Deviation=3.987283e-02
Bit=19  |   Mean=2.345110e-02   |   Deviation=8.037686e-02
Bit=20  |   Mean=4.619350e-02   |   Deviation=1.607274e-01
Bit=21  |   Mean=9.095334e-02   |   Deviation=3.197819e-01
Bit=22  |   Mean=1.573313e-01   |   Deviation=5.606262e-01
Bit=23  |   Mean=2.478969e-01   |   Deviation=7.039999e-01
Bit=24  |   Mean=2.478969e-01   |   Deviation=7.039999e-01
Bit=25  |   Mean=2.478969e-01   |   Deviation=7.039999e-01
Bit=26  |   Mean=2.478969e-01   |   Deviation=7.039999e-01
Bit=27  |   Mean=2.478969e-01   |   Deviation=7.039999e-01
Bit=28  |   Mean=2.478969e-01   |   Deviation=7.039999e-01
Bit=29  |   Mean=2.478969e-01   |   Deviation=7.039999e-01
Bit=30  |   Mean=2.478969e-01   |   Deviation=7.039999e-01
Bit=31  |   Mean=2.863773e-01   |   Deviation=8.091196e-01
```

Figura 13. Parámetros estadísticos obtenidos en un análisis

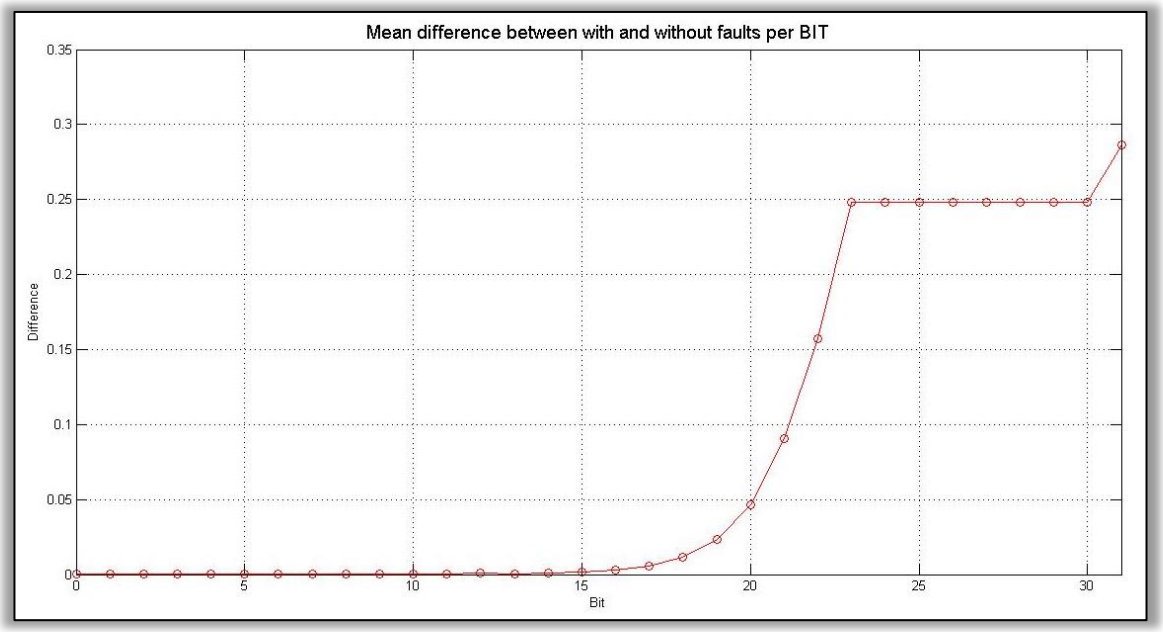


Figura 14. Diferencia media por bit entre el Golden Run y el Run con fallos

4.7.2.2 Evolución del error con y sin fallos: error_graph.m

Permite comparar el comportamiento del error en el caso del sistema con fallos y del sistema sin fallos.

- La línea en color verde representa la evolución del error en el sistema SIN fallos.
- La línea en color rojo representa la evolución del error en el sistema CON fallos.
- Los círculos azules son los fallos que han sido inyectados.
- Los segmentos verticales negros representan un cambio de referencia, de ahí que ocurran oscilaciones.

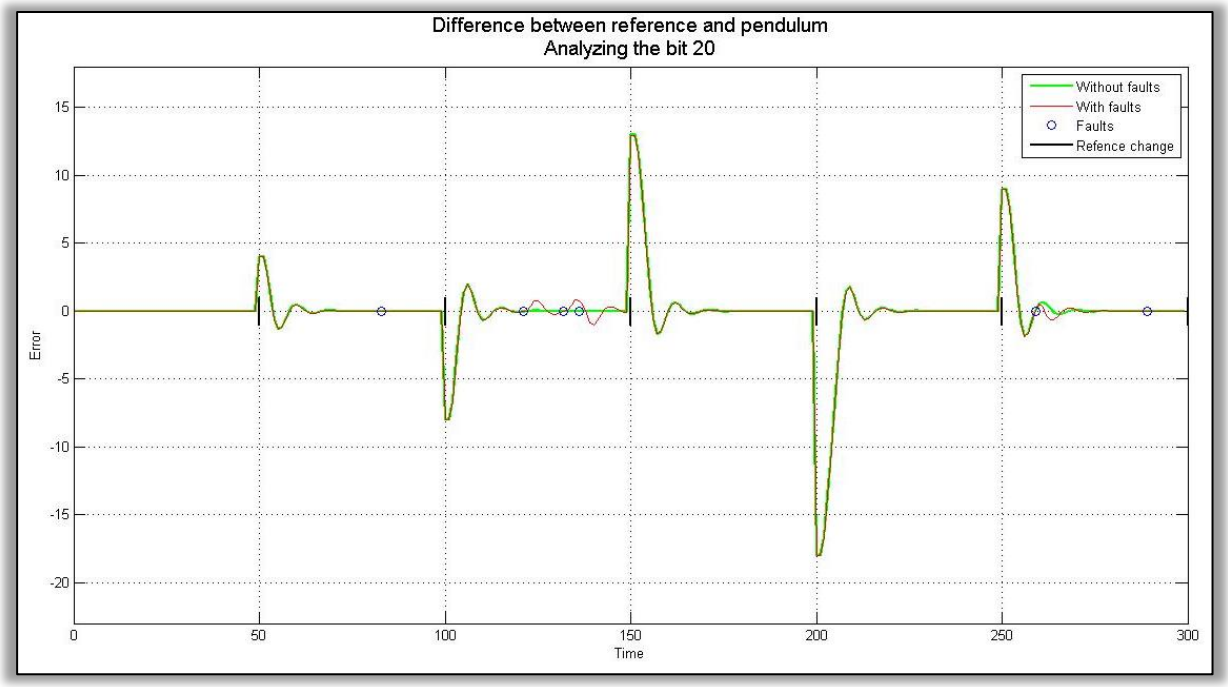


Figura 15. Comportamiento del péndulo al analizar el bit 20

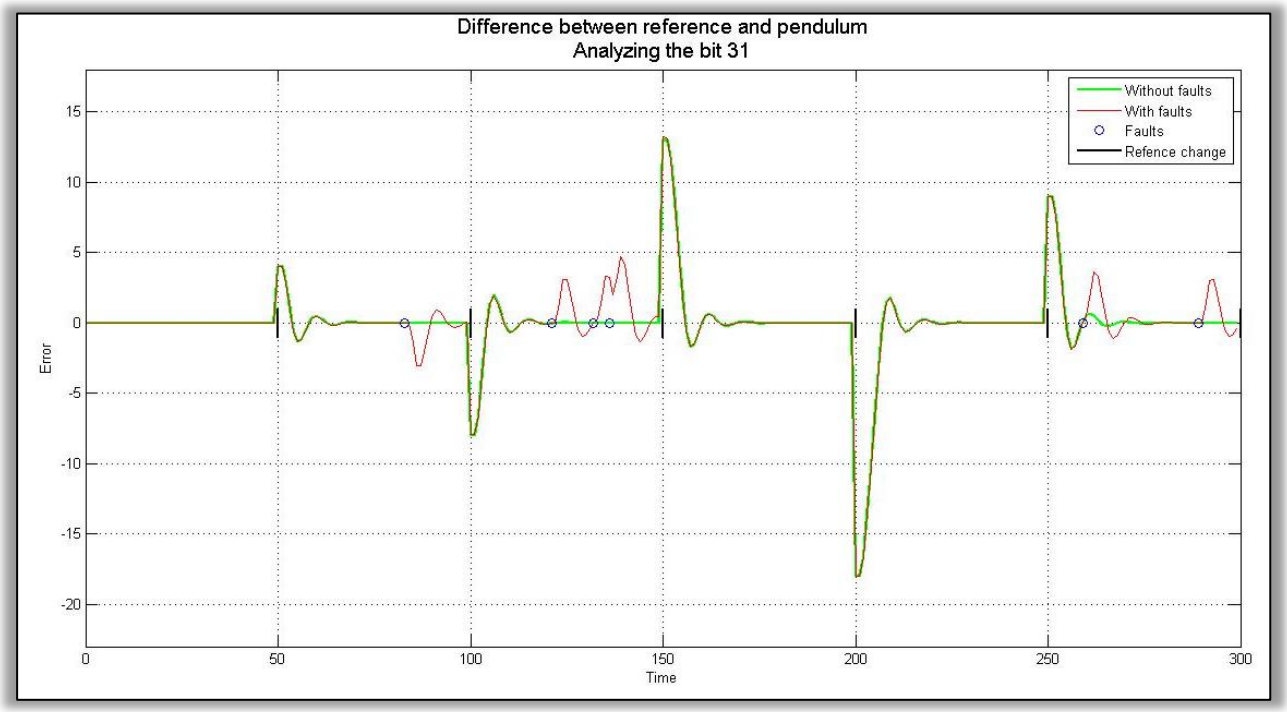


Figura 16. Comportamiento del péndulo al analizar el bit 31

4.7.2.3 Evolución de la diferencia entre con y sin fallos: difference_graph.m

Gracias a estas gráficas podrá cuantificarse la diferencia en el comportamiento entre con y sin fallos.

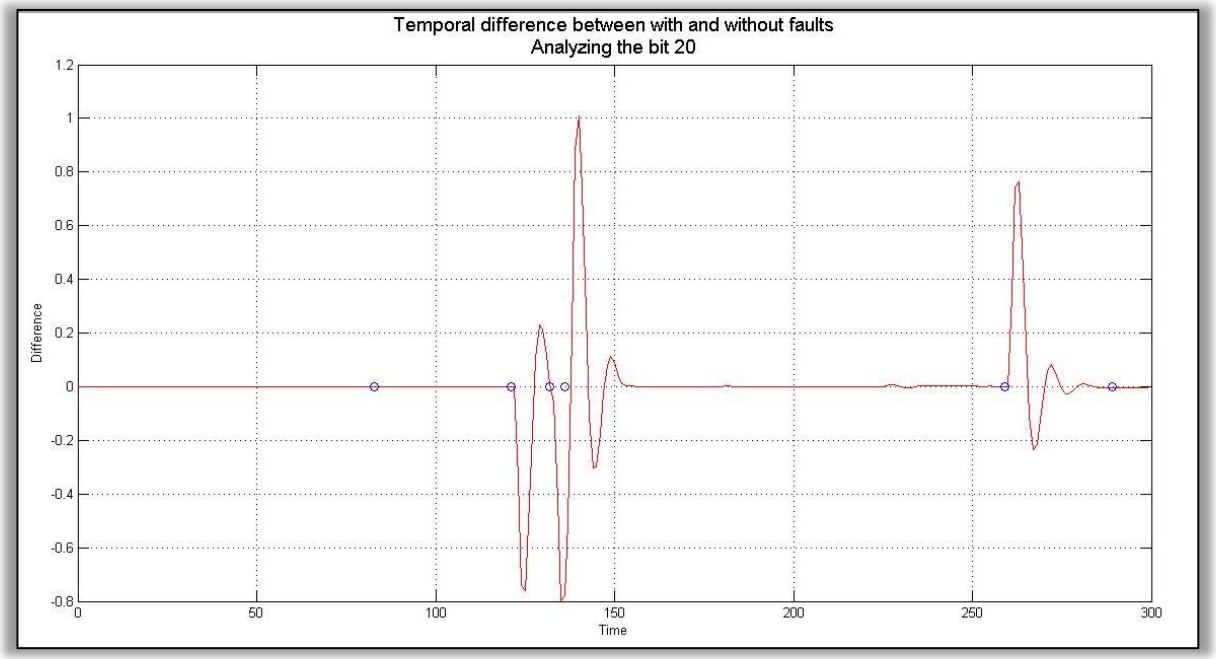


Figura 17. Diferencia entre el Golden Run y el Run con fallos durante el bit 20

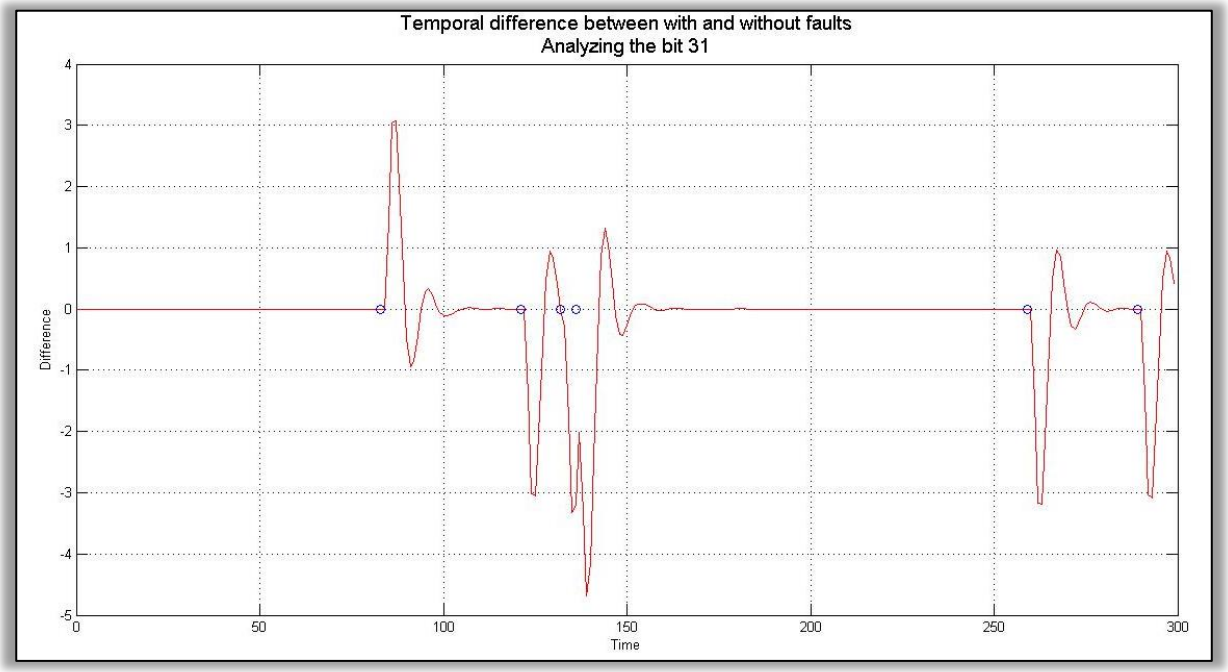
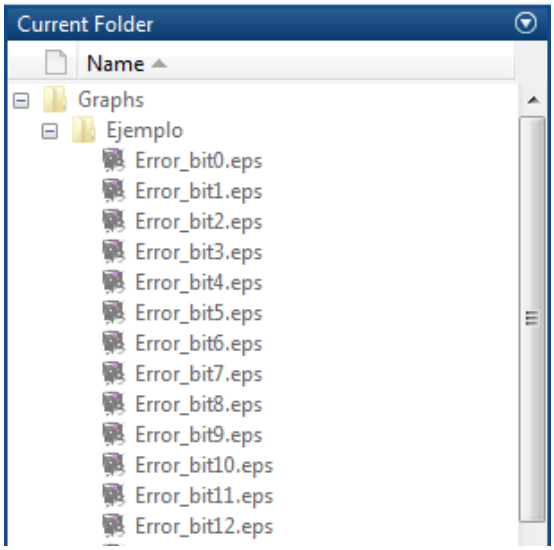


Figura 18. Diferencia entre el Golden Run y el Run con fallos durante el bit 31

4.7.2.4 Guardar localmente datos: readme.m

Gracias a este script se podrán guardar las gráficas y datos anteriores localmente en el ordenador en el que se esté realizando el estudio.




```
1  [Analysis data]
2  Register = Proportional
3  Index = all the register [0-31].
4  Injection mode = Number of faults (6)
5  Injected faults = 6.
6
7  [Analysis results]
8  Bit=0   |   Mean=6.082959e-07   |   Deviation=5.784885e-07
9  Bit=1   |   Mean=1.216594e-06   |   Deviation=1.156979e-06
10 Bit=2   |   Mean=2.433183e-06   |   Deviation=2.313957e-06
11 Bit=3   |   Mean=4.866371e-06   |   Deviation=4.627915e-06
12 Bit=4   |   Mean=9.732739e-06   |   Deviation=9.255829e-06
13 Bit=5   |   Mean=1.946548e-05   |   Deviation=1.851165e-05
14 Bit=6   |   Mean=3.893093e-05   |   Deviation=3.702328e-05
15 Bit=7   |   Mean=7.786181e-05   |   Deviation=7.404646e-05
```

A la izquierda podemos ver como en el subdirectorio "Ejemplo" que se encuentra dentro de "Graphs" se han guardado las imágenes del comportamiento del péndulo de la evolución de la media por bit en cada Run.

A la derecha el archivo "readme.txt" que resume el tipo de análisis realizado y la evolución de la media y desviación en cada bit.

4.8 Conexión PC-FPGA

Los dos tipos de conexión a los que la herramienta System Generator da soporte para la comunicación PC-FPGA son Ethernet y JTAG, por lo tanto dependiendo de la tarjeta que se utilice se podrá realizar uno o ambos análisis dependiendo de los puertos con los que cuente. El modelo utilizado para la realización del presente trabajo ha sido Virtex®-4 SX ML402 de Xilinx, el cual cuenta con puertos Ethernet y JTAG.

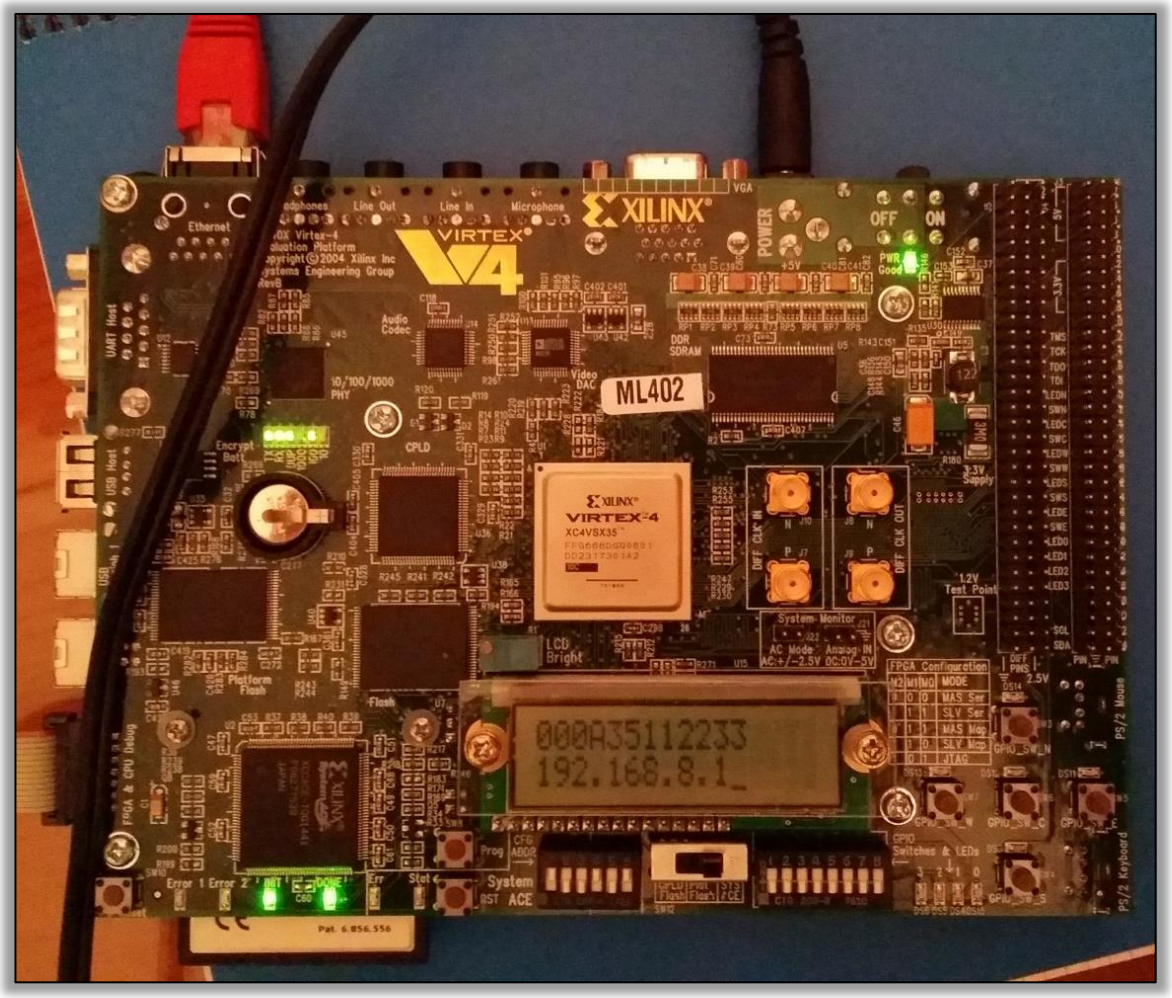


Figura 19. Tarjeta utilizada para la realización de este trabajo. Modelo ML402 de Xilinx

4.8.1 Conexión Ethernet

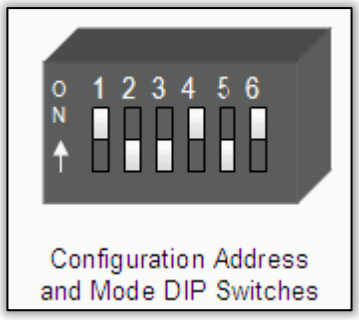
En el caso de la tarjeta ML402, para la conexión mediante Ethernet será necesario contar con:

- Tarjeta Virtex®-4 SX ML402
- Cable de alimentación de la tarjeta
- Memoria CompactFlash y su lector para el PC
- Network Interface Card (NIC) Ethernet en el PC
- Cable Ethernet

Además será necesario realizar una serie de ajustes tanto en la tarjeta como en el PC para poder llevar a cabo este análisis.

4.8.1.1 Configuración de la FPGA

Para configurar la FPGA se utilizará una memoria CompactFlash en la que se cargarán desde el PC los archivos necesarios. En dichos archivos se establecerán las direcciones MAC e IP de la tarjeta. Además, también será necesario colocar los botones con los que cuenta la tarjeta para que la configuración sea la referente a una comunicación Ethernet:



4.8.1.2 Configuración del entorno

Para configurar el entorno será necesario especificar en el System Generator Token que la simulación la queremos hacer con la FPGA y con conexión Ethernet.

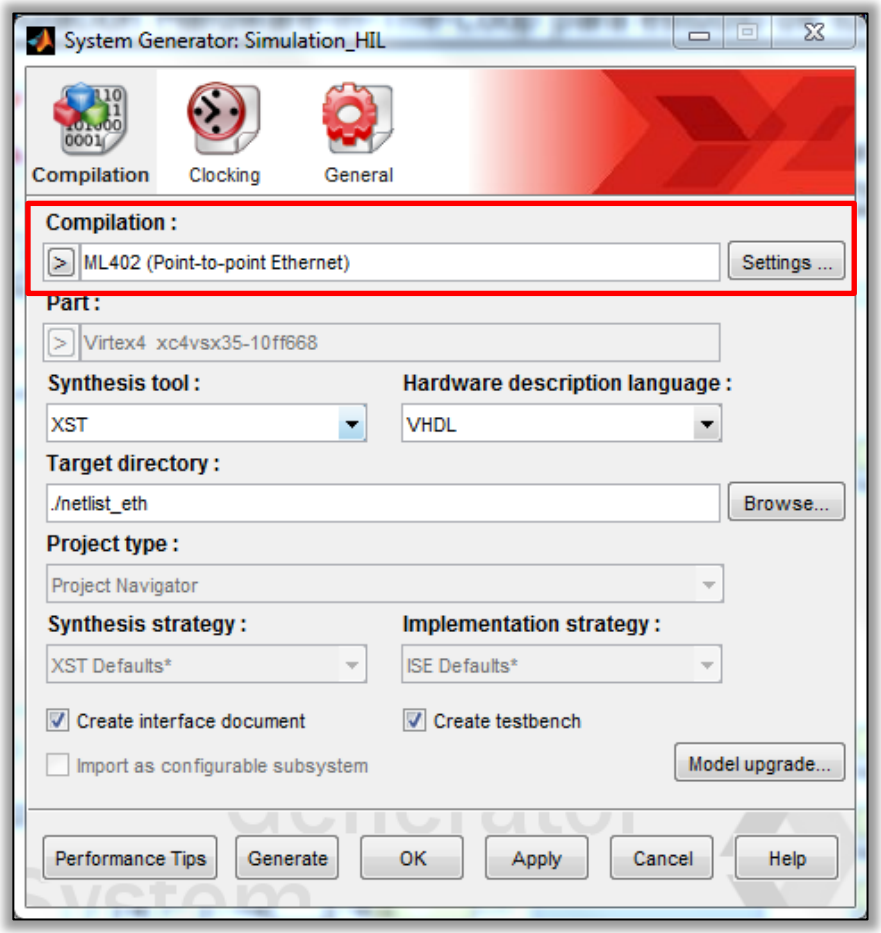


Figura 20. Configuración del System Generator Token para emulación con la FPGA

Tras seleccionar la configuración correcta y darle a “Generate”, System Generator se encargará de compilar nuestro diseño y posteriormente nos aparecerá una nueva ventana en la que aparecerá un bloque parecido a la Black Box que teníamos anteriormente. Para montar nuestro nuevo entorno Hardware-In-the-Loop solo habrá que copiar el modelo anterior cambiando la Black Box que teníamos por el nuevo bloque que hemos creado, quedando de la siguiente manera:

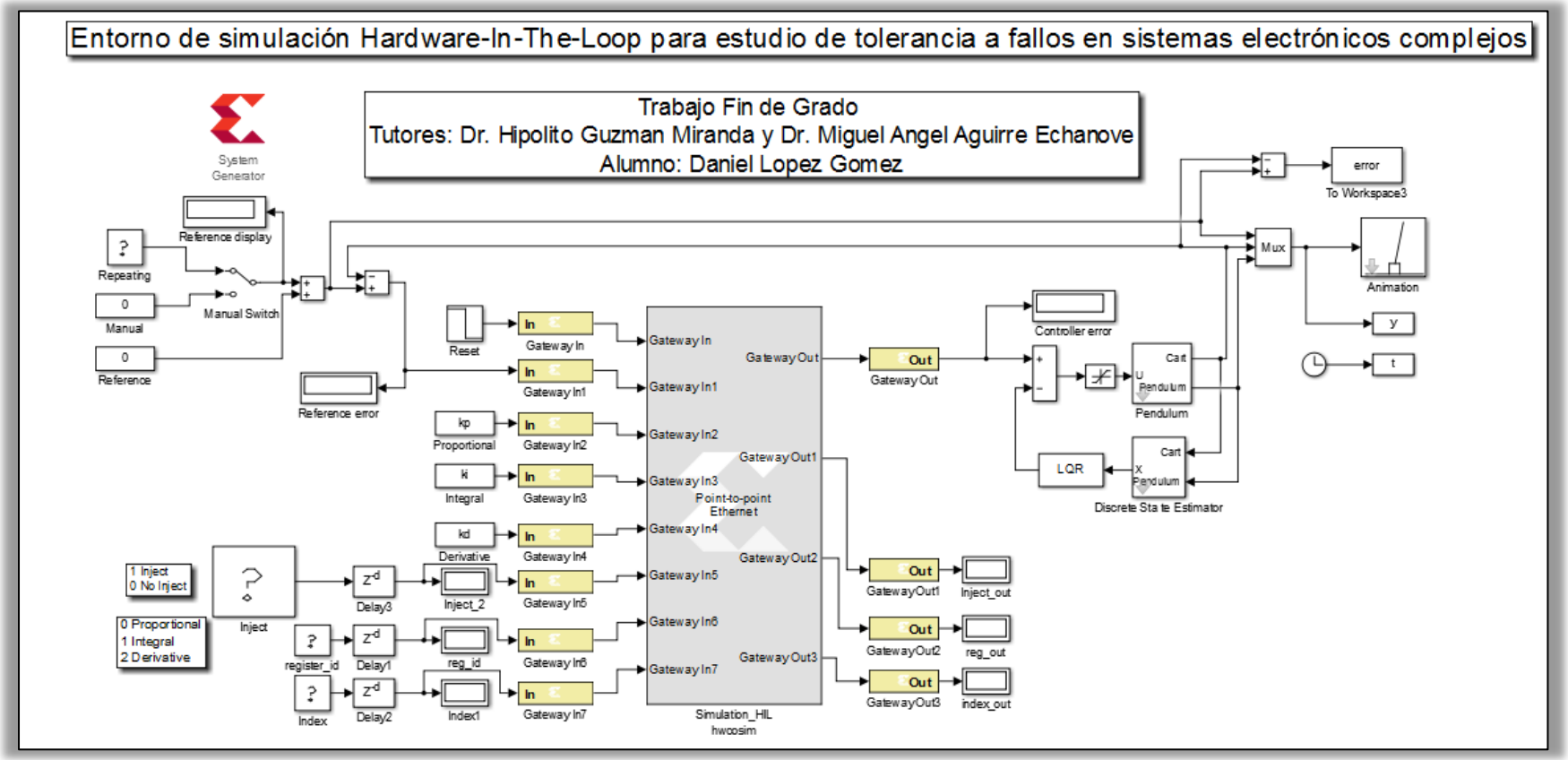


Figura 21. Entorno Simulink utilizado para la emulación con la FPGA

Por último, haciendo doble click en el nuevo bloque, habrá que establecer la interfaz en la cual conectaremos el cable Ethernet, el modo de reloj que queremos (Single stepped o Free running) y especificaremos que la configuración se haga mediante el mismo cable Ethernet, puesto que si no, será necesario un cable adicional.

4.8.2 Conexión JTAG

Para la configuración mediante JTAG serán necesarios los siguientes componentes:

- Tarjeta Virtex®-4 SX ML402
- Cable de alimentación de la tarjeta
- Memoria CompactFlash y su lector para el PC
- Puerto USB en el PC
- Cable "Xilinx Parallel Cable IV"

Los siguientes pasos a seguir para llevar a cabo la simulación son muy similares a los vistos para el caso de la conexión Ethernet.

4.8.3 Modos de reloj

El modo de reloj que se elija influirá en la forma de comunicarse que tendrán Simulink y la FPGA. Existen dos modos posibles:

- **Single-Step Clock:** el reloj que utilizará la FPGA será proporcionado por la propia herramienta Simulink estando ambos sincronizados. Es por ello que es posible que se limite el rendimiento alcanzable por la FPGA si se utiliza este reloj.
- **Free-Running Clock:** en este modo la FPGA correrá con un reloj interno pudiendo alcanzar velocidades mucho mayores que en el caso anterior.

5 RESULTADOS EXPERIMENTALES

5.1 Clasificación de fallos

Como ya se ha comentado, no todos los fallos que afectan a un sistema electrónico deben de resultar fatales para éste y es en este apartado donde profundizaremos en dicho concepto. En el presente trabajo clasificaremos los fallos según la consecuencia que tenga sobre el sistema, de este modo contemplaremos dos tipos de fallos:

- **Recupera**: el péndulo es capaz de corregir el error introducido por el fallo siendo posible que continúe siguiendo las referencias que se le van introduciendo.
- **No recupera**: el péndulo se descontrola de modo que solo puede volver al estado inicial mediante un reseteo del sistema.

5.2 Factores que contribuyen al descontrol del sistema

A partir de las diferentes simulaciones y sus respectivos análisis que se han ido haciendo durante el transcurso de este proyecto se han podido apreciar diferentes factores que contribuyen a que los fallos introducidos afecten de una manera más negativa al sistema. Dichos factores son los siguientes:

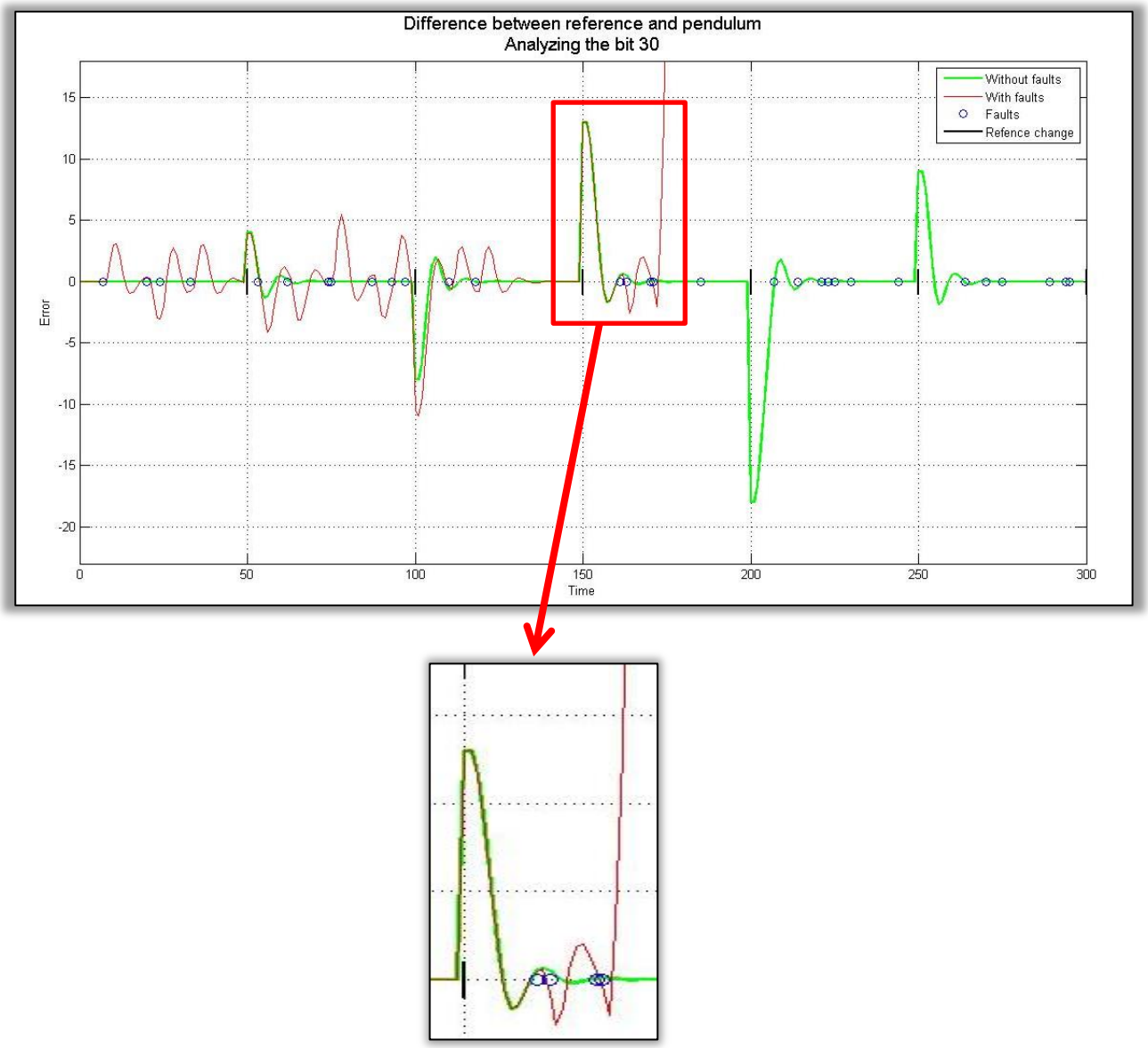
- **Posición y estado del bit afectado**: estas dos características deben ser consideradas conjuntamente puesto que el error final que se introduce al cambiar un bit depende de la combinación de ambas. Para su mejor comprensión, lo veremos con un ejemplo:

Ejemplo 4-1. *A la entrada del péndulo usaremos un byte (8 bits). Recordemos que cuanto mayor sea el valor pasado al péndulo, mayor será el desplazamiento de éste. El número del que partiremos será 50 (00110010), el cual será proporcionado al péndulo para que éste calcule su aceleración.*

Si el bit afectado va a pasar de 0 a 1, cuanto más significativo sea, mayor será el error introducido, puesto que al valor inicial le estaremos sumando 2^X siendo X la posición del bit. De este modo, si por ejemplo, cambiamos el bit que está en la posición 6, el nuevo número será 114 (01110010), lo que provocará un mayor desplazamiento del péndulo y por lo tanto una mayor oscilación en el sistema.

Sin embargo, si el bit afectado va a pasar de 1 a 0, cuanto más significativo sea, menor será el valor final que se pasa al péndulo y por lo tanto menor será la oscilación que soportará el sistema.

- **Diferencia temporal entre fallos**: puesto que un fallo provoca oscilaciones cuando es introducido en el sistema, esto hace que al ocurrir fallos consecutivos sea más probable que el sistema se descontrole, ya que la oscilación del segundo se unirá a las oscilaciones provocadas por el primero.



- **Momento del fallo**: debido a que la simulación consiste en ir proporcionando referencias al péndulo que éste debe seguir, ello provoca que necesariamente existan oscilaciones que ocurrirán cuando se cambia dicha referencia. Por lo tanto, al igual que en el punto anterior, en el caso de que ocurra un fallo durante una oscilación éste afectará de forma más negativa al sistema ya que la oscilación aumentará.

5.3 Análisis de cada bit/registro

En este apartado podremos ver cómo varía la gravedad de los fallos dependiendo de dónde sean introducidos.

5.3.1 Bits

Para estudiar la importancia de cada bit en la estabilidad del sistema veremos varias gráficas en las que se va variando tanto el registro afectado como el tipo de fallo introducido:

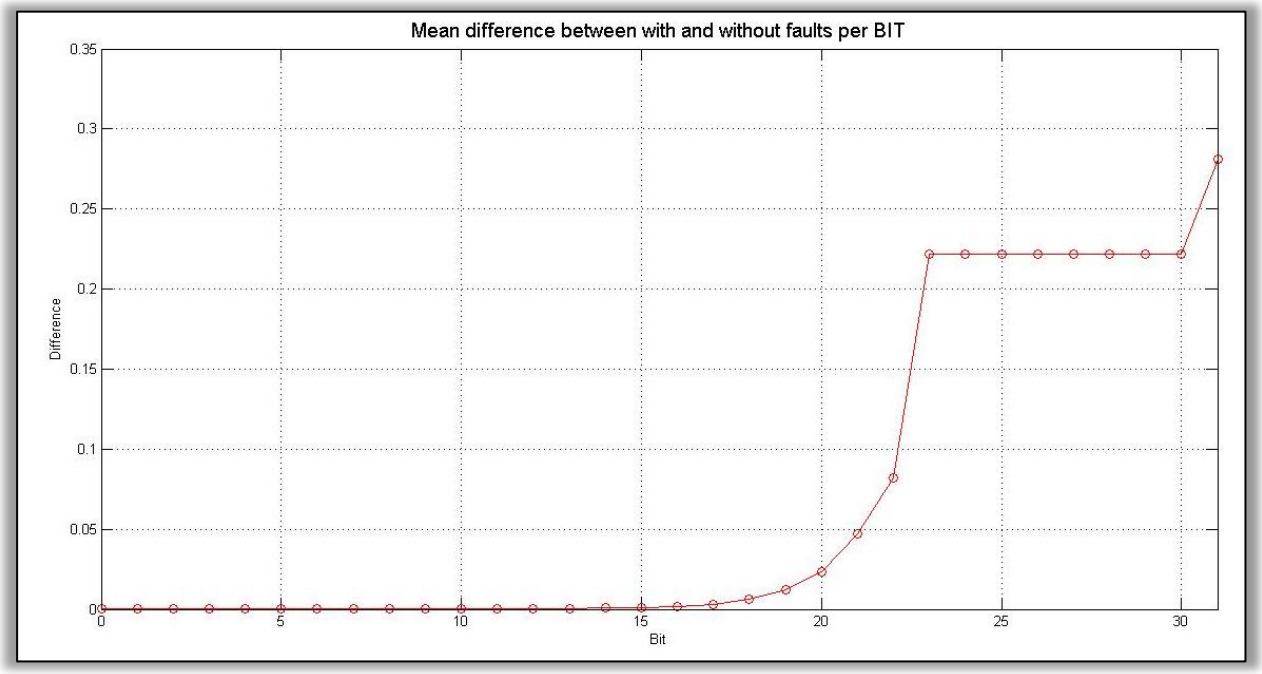


Figura 22. Error medio por bit. Registro: proporcional. Fallos inyectados: 6.

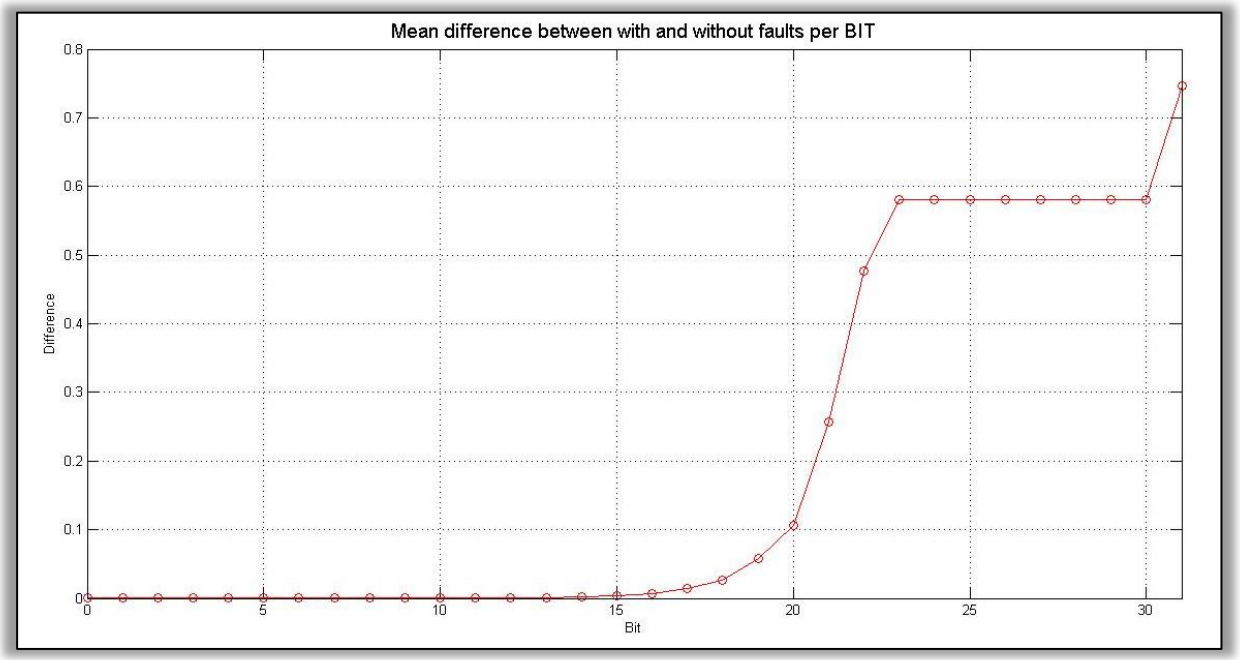


Figura 23. Error medio por bit. Registro: proporcional. Probabilidad de fallo: 6%

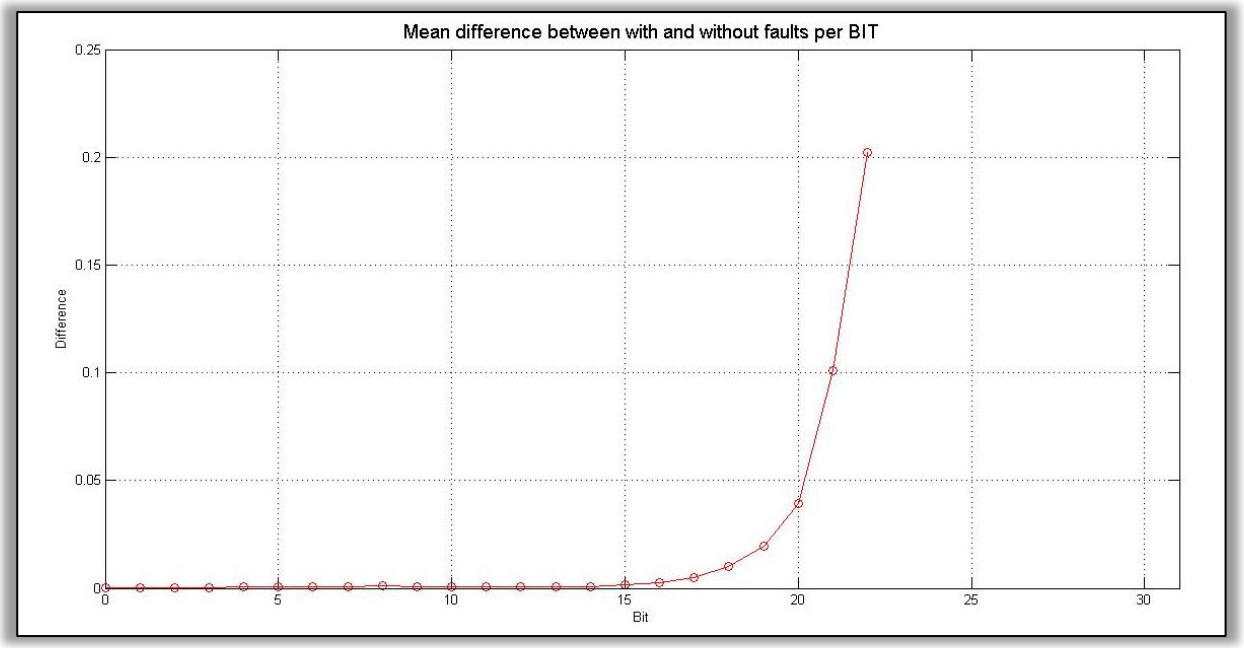


Figura 24. Error medio por bit. Registro: integral. Fallos inyectados: 6

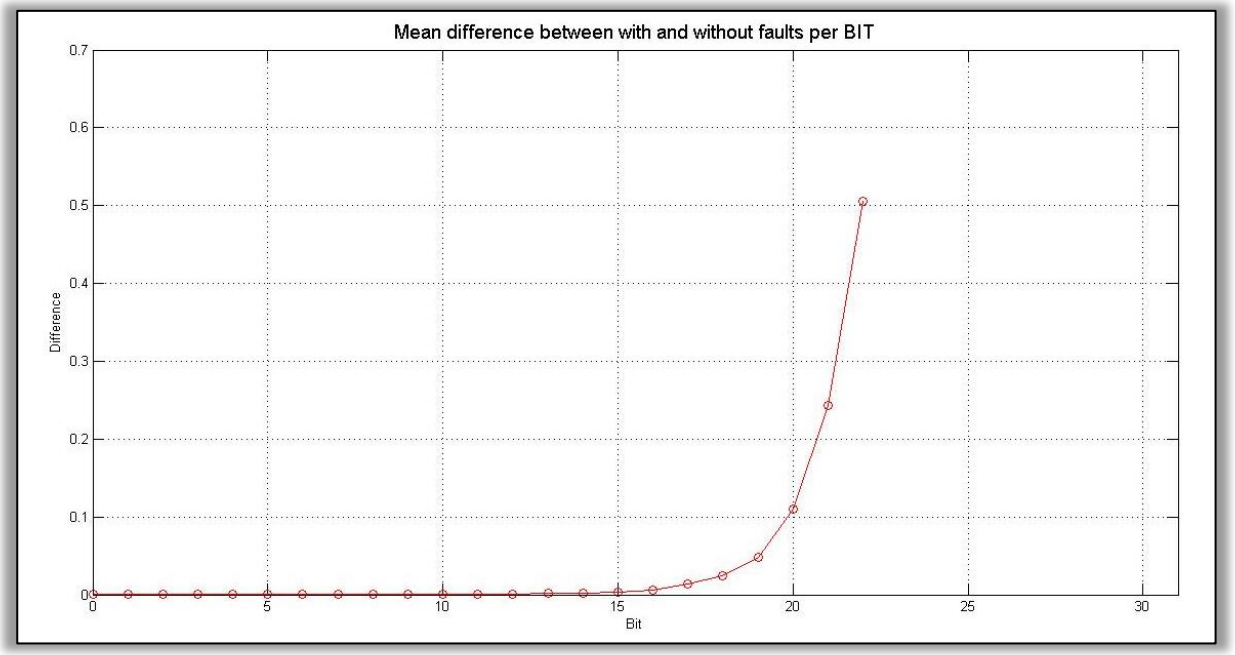


Figura 25. Error medio por bit. Registro: integral. Probabilidad de fallo: 6%

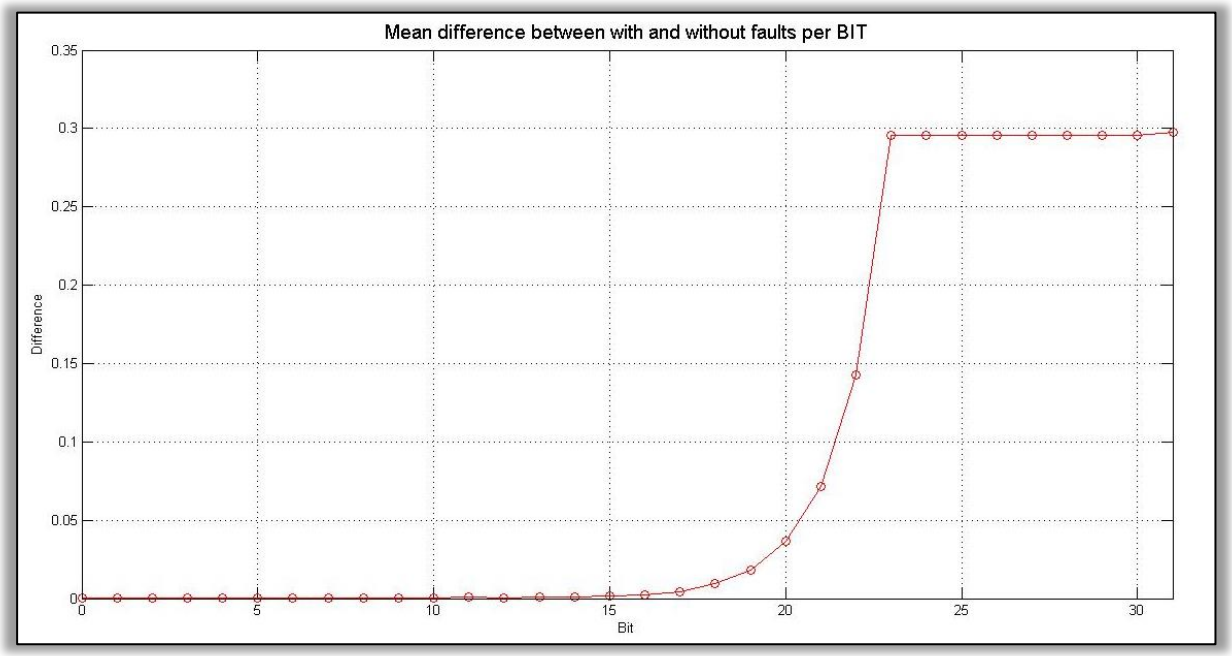


Figura 26. Error medio por bit. Registro: derivativo. Fallos inyectados: 6

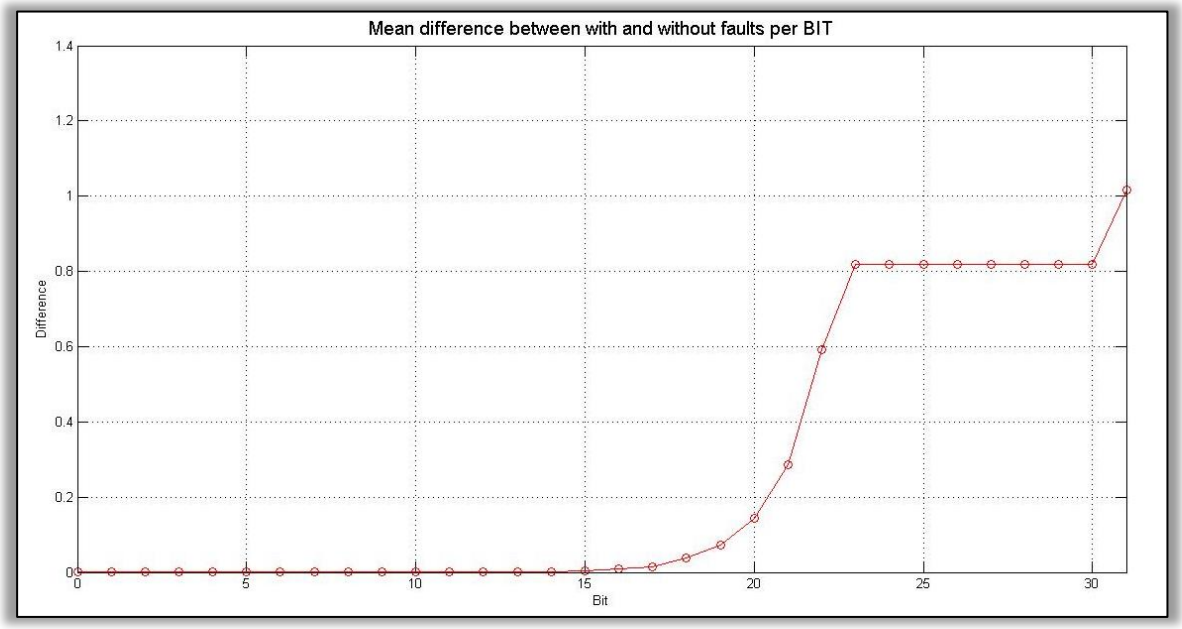


Figura 27. Error medio por bit. Registro: derivativo. Probabilidad de fallo: 6%

A partir de las gráficas podemos ver como existe un patrón común a todas:

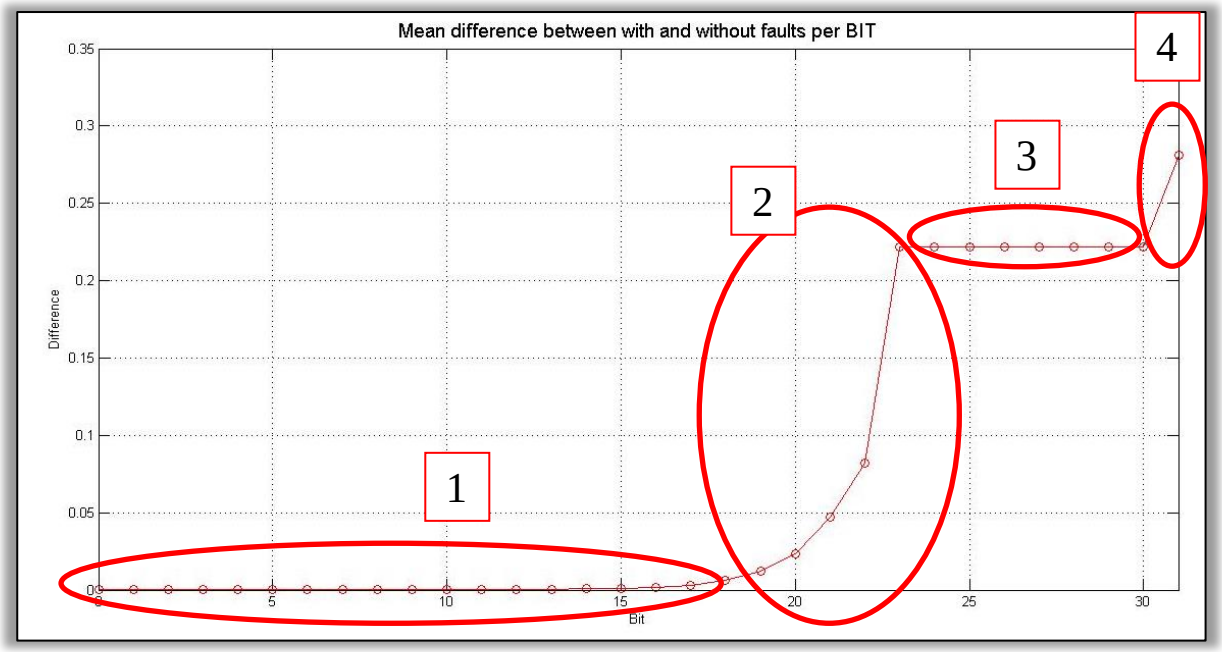


Figura 28. Patrón común en las gráficas de error medio por bit.

Comentaremos cada zona:

1. Desde, aproximadamente, la posición 0 hasta la 17, son los bits menos significativos por lo que parece lógico pensar que si son alterados no cambia en gran medida el comportamiento del sistema.

2. Desde, aproximadamente, la posición 18 hasta la 23, se produce un aumento exponencial del error. Como se ha podido apreciar en gráficas anteriores, cuando estos bits son atacados es bastante probable que el sistema se descontrole por lo que pueden ser considerados bits críticos.

3. Desde, aproximadamente, la posición 24 hasta la 30, el error se mantiene constante, de lo que se puede deducir que si el sistema se mantiene estable mientras el bit 24 es alterado, lo será también en los consecutivos hasta el 30.

4. El bit 31 es el bit de signo (recordemos que el registro se representa en complemento a 2) lo que supone que al realizarle el bit flip el péndulo llevará a cabo un movimiento opuesto al que realizaría sin fallos. Es por ello que es el bit más crítico del registro y en el que será más probable que el sistema se descontrole.

5.3.2 Registros

A continuación veremos diferentes gráficas con las que podremos comparar los efectos de los fallos en los diferentes registros. Para poder realizar la comparación en las mismas condiciones, el vector de fallos[11] será común a los diferentes registros dentro del mismo análisis.

5.3.2.1 Análisis 1

Número de fallos: 8. Posiciones[12]: 5, 38, 87, 163, 173, 209, 226 y 272

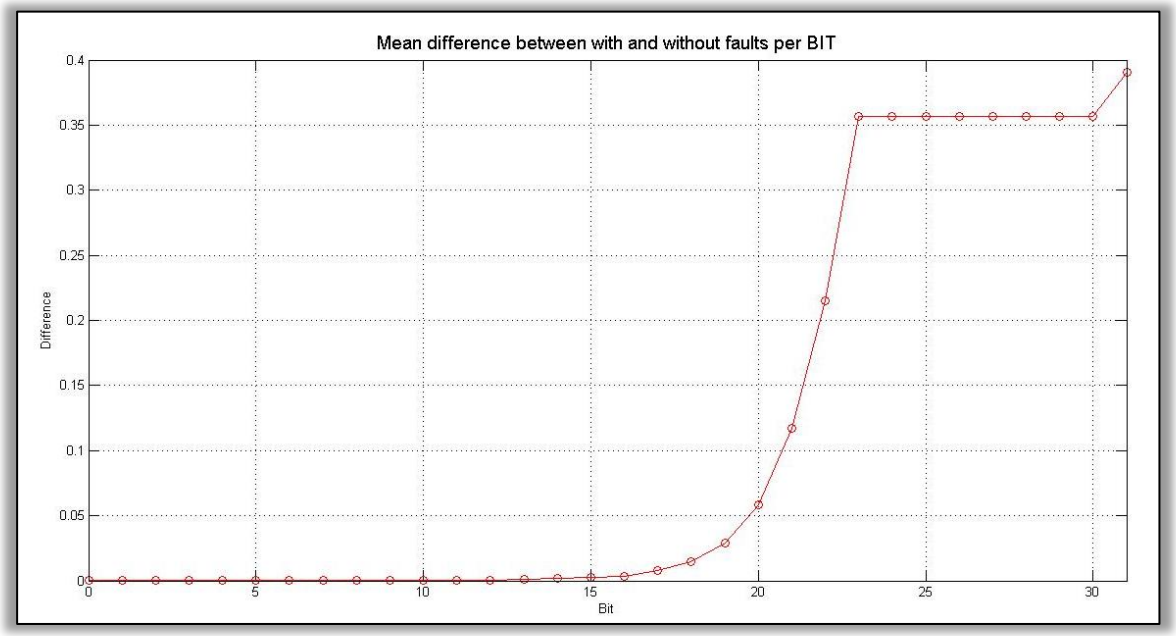


Figura 29. Análisis de registros 1. Registro: proporcional

[11] Es un vector con la longitud de un Run y que contiene los valores '0' y '1'. Los fallos se inyectarán en el Run en la posición que este vector contenga los valores '1'

[12] Elegidas aleatoriamente

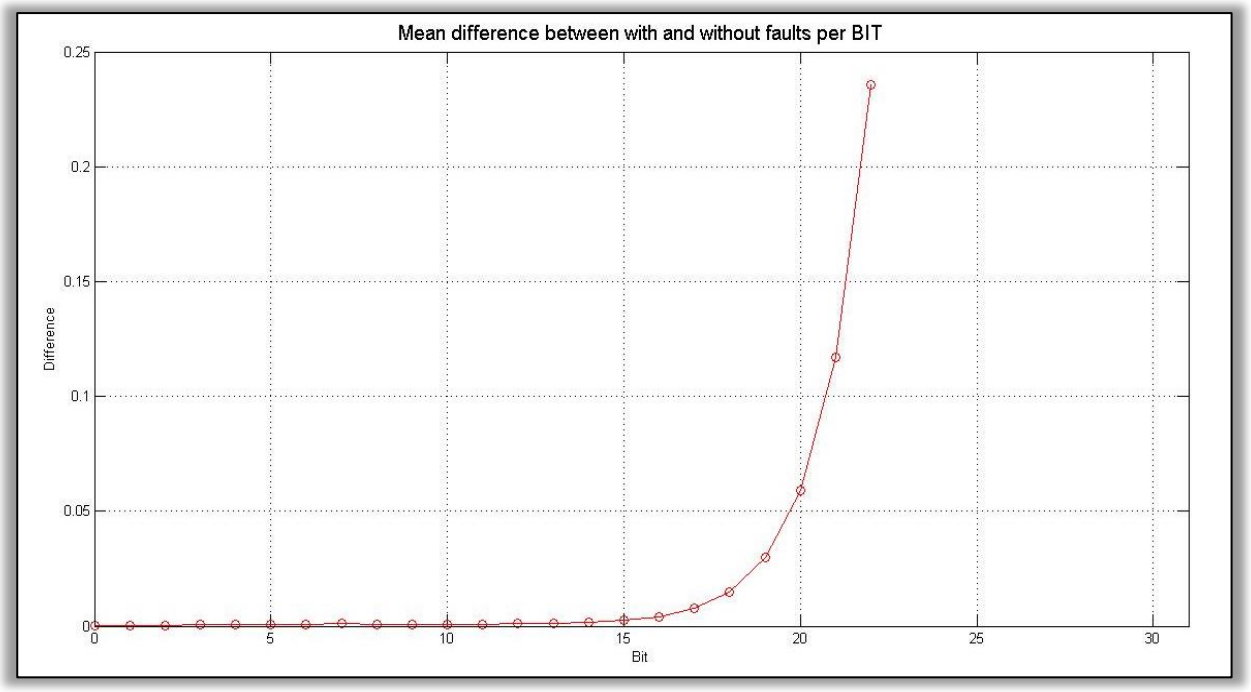


Figura 30. Análisis de registros 1. Registro: integral

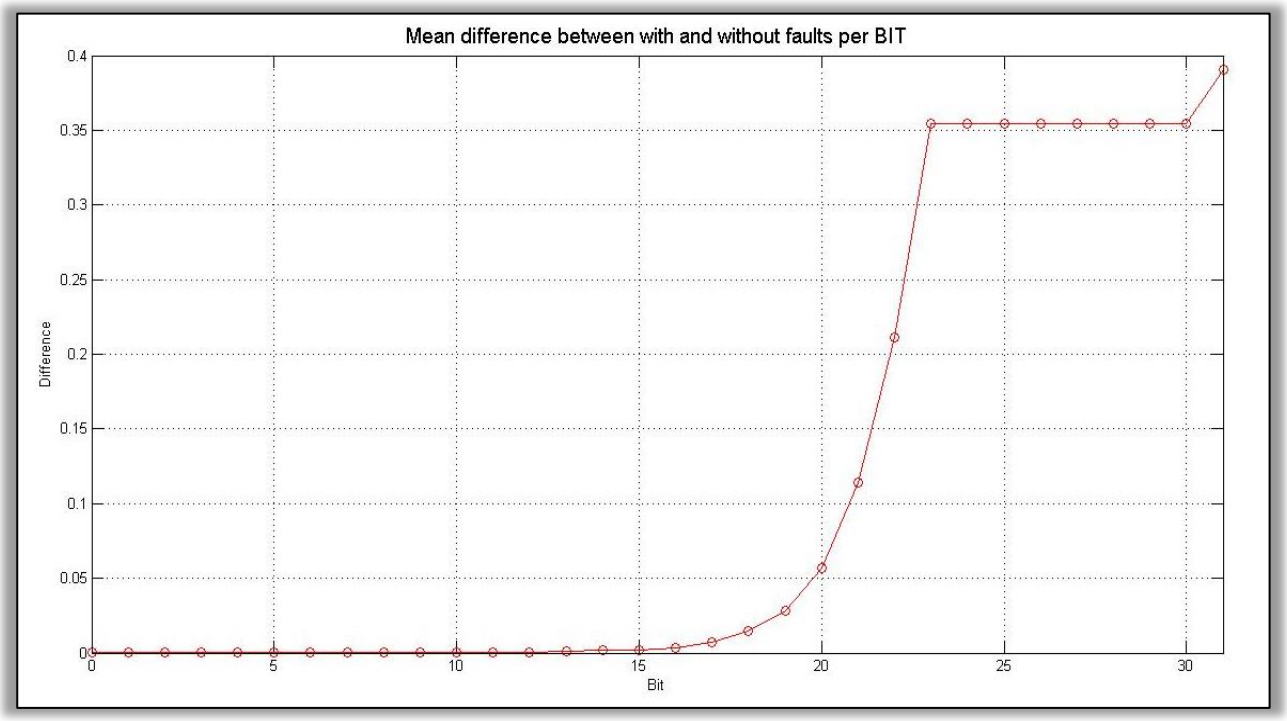


Figura 31. Análisis de registros 1.Registro: derivativo

5.3.2.2 Análisis 2

Número de fallos: 5. Posiciones[13]: 22, 74, 168, 231 y 285 (elegidas al azar)

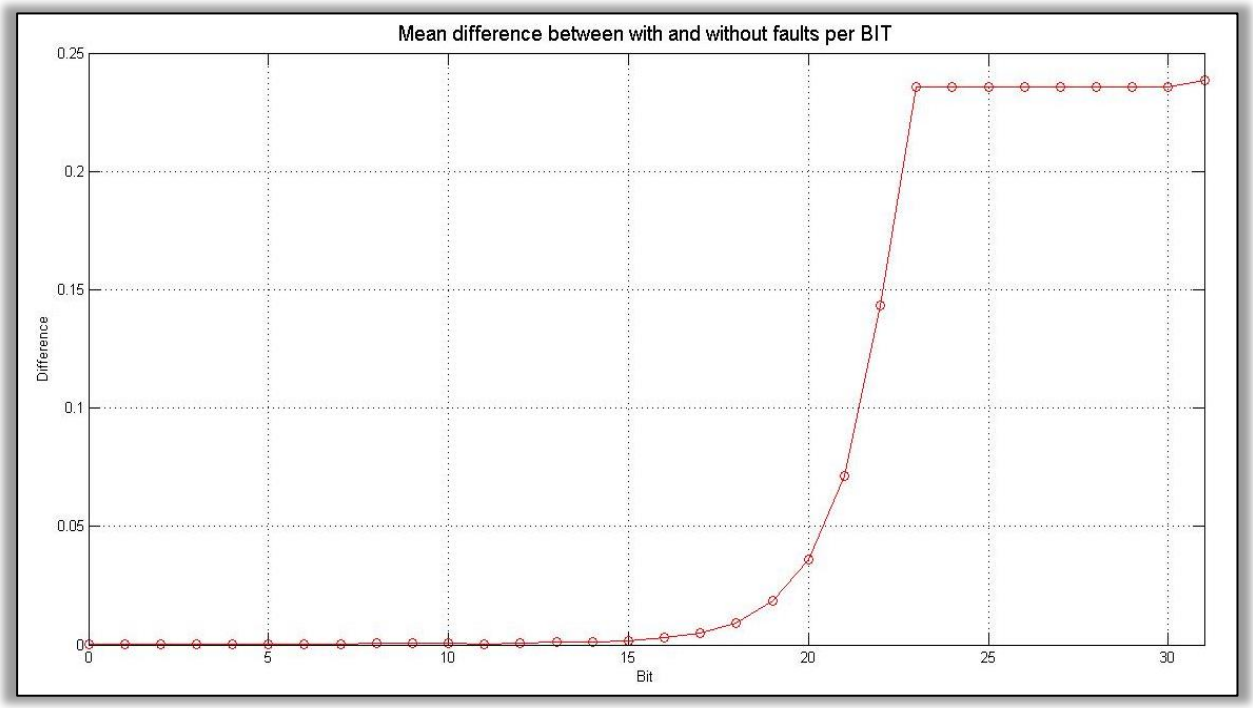


Figura 32. Análisis de registros 2. Registro: proporcional

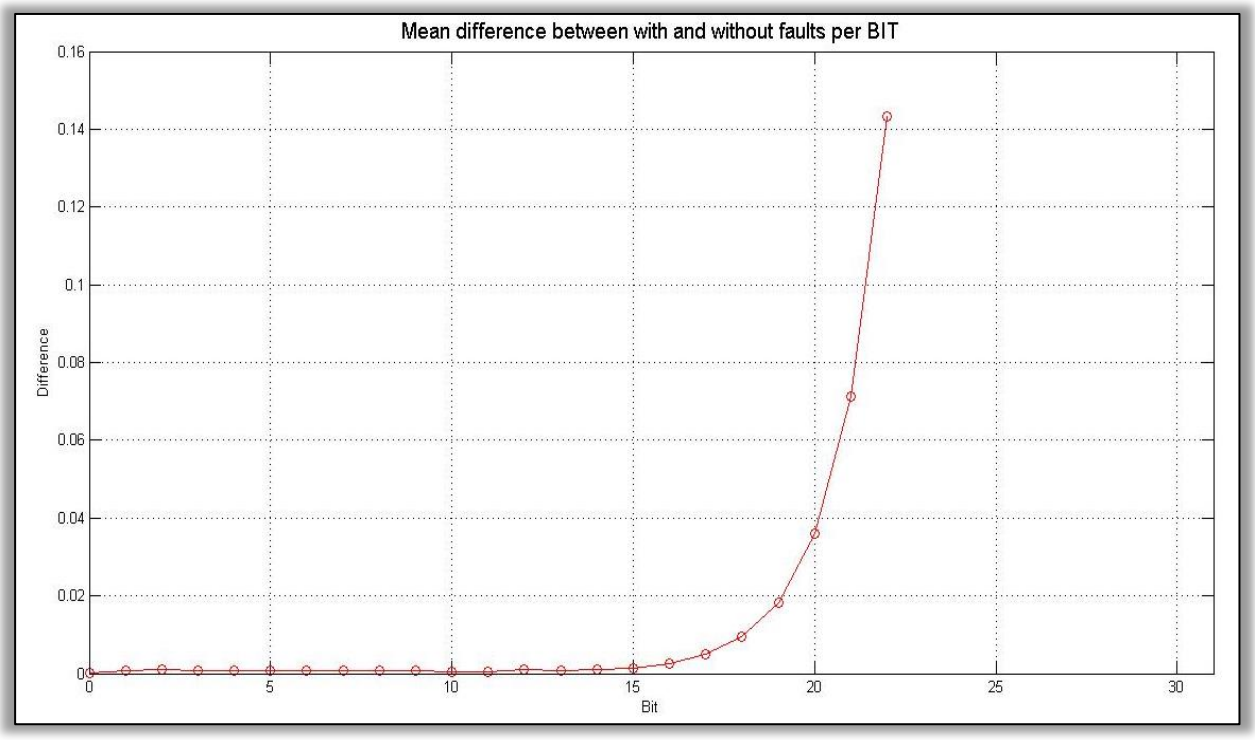


Figura 33. Análisis de registros 2. Registro: integral

[13] Elegidas aleatoriamente

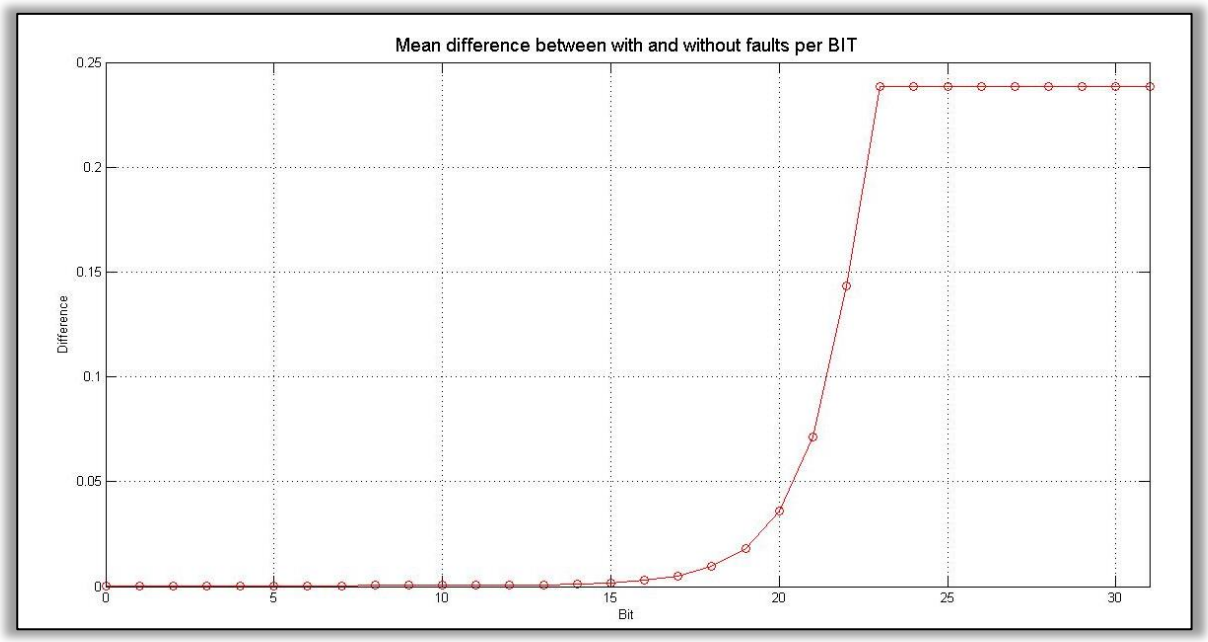


Figura 34. Análisis de registros 2. Registro: derivativo

Como podemos observar a partir de los dos análisis realizados, el registro más crítico es el integral, que es el que ha provocado el descontrol del sistema en ambos casos. Los dos registros restantes (proporcional y derivativo) soportan mejor la inyección de fallos, teniendo ambos un comportamiento similar.

6 CONCLUSIONES

6.1 Nueva técnica de análisis

A lo largo de este documento se ha expuesto un nuevo método caracterizado por ser Hardware-In-the-Loop con el cual se ofrece un nuevo enfoque al estudio de la tolerancia a fallos en sistemas electrónicos. Este nuevo método es consecuencia de la unión de varios elementos: scripts MATLAB, entorno Simulink, FPGA, código VHDL, System Generator for DSP... y permite el análisis de cualquier sistema siempre que el circuito que contenga sea descrito en un lenguaje de descripción hardware (HDL) y que se consiga modelar el resto del sistema mediante la herramienta Simulink.

6.2 Mejora en los sistemas de inyección de fallos tradicionales

La realización de este trabajo ha supuesto además la demostración del margen de mejora que todavía existe en las herramientas de inyección de fallos en circuitos electrónicos ya que se ha conseguido relajar las exigentes restricciones que las rigen. Ello ha sido posible gracias a que se ha enfocado el estudio desde un punto de vista del sistema global que se quiere analizar y no únicamente del circuito electrónico que éste contiene.

7 TRABAJOS FUTUROS

"Stay hungry, stay foolish"

Steve Jobs

7.1 Circuitos y entornos más complejos

Lo que en este documento ha querido mostrarse ha sido el método de análisis utilizado, y no el sistema electrónico a estudiar. Es por ello que se ha utilizado un entorno sencillo pero que a su vez es muy típico a la hora de realizar estudios en el ámbito de Ingeniería de Control.

Por lo tanto, se propone como un posible trabajo futuro el utilizar el método de análisis descrito en este documento para estudiar circuitos y entornos más complejos y representativos de situaciones reales que los que aquí han sido utilizados.

7.2 Integración con FT-UNSHADES2

Investigadores del Grupo de Ingeniería Electrónica del Departamento de Ingeniería Electrónica de la Universidad de Sevilla, bajo la dirección del Profesor Miguel Ángel Aguirre Echánove, y entre los que se encuentra el Profesor Hipólito Guzmán Miranda, han desarrollado para la Agencia Espacial Europea (ESA) un sistema de evaluación de los efectos de la radiación en circuitos digitales llamado FT-UNSHADES2.

El hecho de integrar la herramienta utilizada en este documento junto a FT-UNSHADES2 puede suponer una mejora sustancial de esta última.

7.3 Comparación con resultados obtenidos de una herramienta tradicional

Por último, otro posible trabajo que puede llevarse a cabo en un futuro es realizar un estudio en el que se compare un mismo sistema electrónico mediante varias técnicas entre las que se encuentren la técnica de este trabajo y las técnicas tradicionales, pudiendo así obtener diferentes puntos de vista y cuantificando las diferencias existentes entre ambos enfoques.

8 REFERENCIAS

[1] H. Guzmán, Aportaciones a las técnicas de emulación y protección de sistemas microelectrónicos complejos bajo efectos de la radiación, Sevilla, 2010.

[2] D. Bishop, Floating point package user's guide.

[3] D. Bagni y D. Mackay, Floating-Point PID Controller Design with Vivado HLS and System Generator for DSP, 2013.

[4] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/PID_controller. [Último acceso: Agosto 2015].

[5] H. Guzmán, VHDL para procesado de señal (Apuntes del Master en Ingeniería de Telecomunicaciones).

[6] H. Guzmán, M. A. Aguirre y J. Tombs, Noninvasive Fault Classification, Robustness and Recovery Time Measurement in Microprocessor-Type Architectures Subjected to Radiation-Induced Errors, 2009.

[7] H. Guzmán, Implementación de un procesador empotrado para su estudio bajo inyección de fallos, Sevilla, 2008.

[8] H. M. Quinn, W. H. Robinson, D. A. Black y S. P. Buchner, Fault Simulation and Emulation Tools to Augment Radiation-Hardness Assurance Testing, 2013.

[9] Xilinx, System Generator for DSP, User Guide, 2012.

[10] Xilinx, Vivado Design Suite User Guide, Model-Based DSP Design using System Generator, 2014.

9 ANEXOS

9.1 Anexo A - Código VHDL del Controlador PID

```
-- Autor: Daniel López Gómez (danlopgom@gmail.com)
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.math_real.all;

entity controller is
   generic(
    data_width: integer := 16
    );

port(
        clk         : in std_logic; -- Clock
        ce          : in std_logic; -- Clock Enable
        rst         : in std_logic; -- Reset
        error       : in std_logic_vector(data_width-1 downto 0);   -- Error
        kp          : in std_logic_vector(data_width-1 downto 0);   --
Proportional
        ki          : in std_logic_vector(data_width-1 downto 0);   --
Integral
        kd          : in std_logic_vector(data_width-1 downto 0);   --
Derivative
        inject      : in std_logic;
        reg_id      : in std_logic_vector(data_width-1 downto 0);
        index       : in std_logic_vector(data_width-1 downto 0);
        data_out    : out std_logic_vector(data_width*2-1 downto 0);
        inject_out  : out std_logic;
        reg_id_out  : out std_logic_vector(data_width-1 downto 0);
        index_out   : out std_logic_vector(data_width-1 downto 0)
  );
end controller;

architecture Behavioral of controller is
signal error_1      : std_logic_vector(data_width-1 downto 0);      --
Previous error
signal yp, p_yp     : std_logic_vector(data_width*2-1 downto 0);    --
Proportional out
signal yi, p_yi     : std_logic_vector(data_width*2-1 downto 0);    --
Integral out
signal yd, p_yd     : std_logic_vector(data_width*2-1 downto 0);    --
Derivative out

signal index2: integer range 0 to 31;
begin

-- Testing Hardware Co-Sim
inject_out<=inject;
reg_id_out<=reg_id;
index_out<=index;

index2 <= to_integer(signed(index));

comb: process(yp,yi,yd,error_1,error,kp,ki,kd)
begin
```

```
-- Proportional
p_yp <= std_logic_vector(signed(kp)*signed(error));

-- Integral, limited to [-5,5]
if ((signed(ki)*signed(error)) + signed(yi) > 5) then
    p_yi <= std_logic_vector(to_signed(5,p_yi'length));
elsif ((signed(ki)*signed(error)) + signed(yi) < -5) then
    p_yi <= std_logic_vector(to_signed(-5,p_yi'length));
else
    p_yi <= std_logic_vector((signed(ki)*signed(error)) + signed(yi));
end if;

-- Derivative
p_yd <= std_logic_vector((signed(kd)*signed(error)) -
(signed(kd)*signed(error_1)));

-- Out, limited to [-64,64]
if (signed(yp(31 downto 16)) + signed(yd(31 downto 16)) > 64) then
    data_out(31 downto 16) <= std_logic_vector(to_signed(64,data_out(31
downto 16)'length));
    data_out(15 downto 0) <=(others=>'0');
elsif (signed(yp(31 downto 16)) + signed(yd(31 downto 16)) < -64) then
    data_out(31 downto 16) <= std_logic_vector(to_signed(-64,data_out(31
downto 16)'length));
    data_out(15 downto 0) <=(others=>'0');
elsif (signed(error(7 downto 1)) = 0 and signed(error(15 downto 8)) = 0) then
-- here's the final error
    data_out <= (others=>'0');
else
    data_out <= std_logic_vector(signed(yp) + signed(yi) + signed(yd));
end if;
end process;

sinc: process(rst,clk,ce)
begin
    if (rst='1') then
        yp <=(others=>'0');
        yi <=(others=>'0');
        yd <=(others=>'0');
        error_1 <=(others=>'0');

    elsif(clk='1' and clk'event and ce='1') then
        yp <= p_yp;
        yi <= p_yi;
        yd <= p_yd;
        error_1<=error;

        -- Faults injection
        if (inject = '1') then
            case to_integer(signed(reg_id)) is
                when 0 =>       -- inject in yp
                    yp(index2) <= not(p_yp(index2));
                when 1 =>       -- inject in yi
                    yi(index2) <= not(p_yi(index2));
                when others =>  -- inject in yd
                    yd(index2) <= not(p_yd(index2));
            end case;
        end if;
    end if;
end process;
end Behavioral;
```

9.2 Anexo B – Ejemplo de pérdida de precisión en punto flotante

Para una rápida ejecución del código se recomienda utilizar un compilador online como por ejemplo codepad

```
// Autor: Hipólito Guzmán
#include <stdio.h> // for printf
#include <time.h> // for time
#include <stdlib.h> // for rand
#define MAX 10e10 // max rand number range
int main (void)
{
float table [1000];
float accfwd = 0;
float accrev = 0;
srand(time(NULL)); // set random seed to time
int i;
// Initialize table with random floats
for (i=0; i<1000; i++)
{
table[i] = ((float)rand()/(float)(RAND_MAX)) * MAX;
}
// Sum from 0 to 999
for (i=0; i<1000; i++)
{
accfwd += table[i];
}
// Sum from 999 to 0
for (i=999; i>= 0; i--)
{
accrev += table[i];
}
// Compare and print
printf ("sum (fwd): %f\n", accfwd);
printf ("sum (rev): %f\n", accrev);
printf ("difference: %f\n", accfwd-accrev);
return 1;
}
```