Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de la Telecomunicación
Intensificación de Sistemas Electrónicos

Contraste de un experimento de radiación ionizante frente a su reproducción bajo inyección de fallos

Autor: Daniel Vela Calderón
Tutor: Hipólito Guzmán Miranda

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Septiembre, 2018

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Contraste de un experimento de radiación ionizante frente a su reproducción bajo inyección de fallos

Autor:
Daniel Vela Calderón

Tutor:
Hipólito Guzmán Miranda
Profesor contratado doctor

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Septiembre, 2018

Proyecto Fin de Carrera: Contraste de un experimento de radiación ionizante frente a su reproducción bajo inyección de fallos

Autor: Daniel Vela Calderón

Tutor: Hipólito Guzmán Miranda

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mi familia y amigos

A mis maestros

A mis compañeros

Agradecimientos

En primer lugar, me gustaría agradecer al tutor de este trabajo, Hipólito Guzmán Miranda, por poner siempre sus grandes conocimientos de la materia a disposición del alumnado. Muchas gracias por sacar tiempo, muchas veces de donde no había, para atender cualquier tipo de duda o sugerencia de mejora, y por supuesto por su enorme paciencia conmigo. También agradecer la oportunidad de poder trabajar con él y su grupo de investigación. No me quiero olvidar de los compañeros del grupo de investigación, en especial Luis, por compartir su enorme manejo de la informática sin importar cuándo ni cómo, y a María, por su ayuda y paciencia constante. He aprendido muchísimo de vosotros.

No me quiero olvidar de todos los profesores que me han instruido durante todos estos años de carrera, y en especial me gustaría nombrar a Juan Antonio Becerra González, por confiar en mí, y en mis capacidades desde el primer día, y por su ayuda tanto en la escuela como en las prácticas de empresa.

Por último, y apartándome del ámbito académico, me gustaría agradecer a mi familia, por su apoyo constante, a mis padres por mostrarme su confianza desde el día que comencé mis estudios, a mi hermano por su ejemplo, y a mis dos hermanas por aguantarme continuamente. También me quiero acordar de esos compañeros de clase, que han llegado a convertirse en amigos después compartir juntos estos años de carrera. Y no quiero olvidarme de esos amigos, que han estado ahí durante estos años, y en especial mencionar a Abi, Dani, Miguel y Ana.

A todos muchas gracias, ya que sin vosotros no hubiera podido llegar hasta aquí.

Daniel Vela Calderón

Sevilla, 2018

Resumen

En este trabajo se ha llevado a cabo la comparación de ciertos datos de radiación con otros datos obtenidos a partir de la plataforma de inyección de fallos de la Universidad de Sevilla FT-UNSHADES 2. El fin de dicha comparación no es más que la de poder demostrar la mayor o menor exactitud en los resultados de la plataforma mencionada.

Para llevar a cabo dicha comparación se ha hecho uso de la estadística, pudiéndose demostrar que los resultados obtenidos a partir de la plataforma son los esperados por los datos reales de radiación, dentro de un margen de error calculado.

Se han partido de datos reales de radiación, que han sido procesados de manera que puedan ser comparados con los otros datos obtenidos a partir de la plataforma de inyección de fallos.

Abstract

This work assesses the accuracy of a fault injection platform FT-UNSHADES 2 develops by Universidad de Sevilla by comparing its output with the results of a radiation experiment.

For this comparison we are going to use the statistic, and we will prove that the results of both approaches are within the statistically expected error margin.

The radiation results have been processed a way of can be compare with the results obtained since the fault injection platform.

A real radiation results has been processed so they can be compared with the results obtained with the fault injection platform.

Índice

Índice de Tablas

Índice de Figuras

Notación

SEE	Single Event Effects
SEU	Single Event Upsets
TMR	Triple Modular Redundacy
SEL	Single Event Latchup
SEGR	Single Event Gate Rupture
SEB	Single Event Burnout
FPGA	Field Programmable Gate Arrays
DUT	Device Under Test
LANSCE	Los Álamos Science Center
AVF	Architecture Vulnerabity Factor
CNA	Centro Nacional de Aceleradores

1 INTRODUCCIÓN

"La educación es el pasaporte hacia el futuro, el mañana pertenece a aquello que se preparan para él en el día de hoy".

- Malcolm X -

Es bien sabido que uno de los principales inventos llevados a cabo por el ser humano, y que han ayudado al estado de bienestar de éstos es el de los satélites artificiales. Se llama satélite a todo aquello que orbita alrededor de un cuerpo celeste. Nuestro caso de estudio se basa en satélites artificiales, creados por el ser humano y que orbitan alrededor de la Tierra.

Una de las particularidades a tener en cuenta de los satélites es que están sometidos a unas condiciones que nada tiene que ver con las de la Tierra, cuya atmósfera protege de la radiación procedente del espacio. Por tanto, cabe destacar, que necesita de una electrónica que se adapte a dichas condiciones, que bien son extremas. Investigadores e ingenieros invierten una gran cantidad de esfuerzos en la mejora de las características de los mencionados sistemas electrónicos. Éstos intentan reducir, en la medida de lo posible, el efecto de la radiación ionizante proveniente de vientos solares (que mueven electrones, protones e iones pesados) y de radiación cósmica.

La radiación ionizante puede llegar a provocar distintos tipos de daños en los sistemas electrónicos. En nuestro caso de estudio, nos vamos a centrar en los llamados Efectos producidos por eventos singulares, o en ingles, Single Event Effects (SEE). Estos casos de SEE, no son más que protones, neutrones o iones pesados con suficiente carga como para producir un "evento", como puede ser un cambio de estado en un biestable o incluso un cambio en la polarización de un transistor. Podemos llegar a distinguir cuatro tipos de SEE, que numeramos a continuación:

1. Single Event Upsets (SEU): que no es más que un cambio de estado provocado por una partícula de radiación ionizante [1]. Dicho cambio de estado es provocado cuando la partícula golpea un nodo importante de un elemento lógico. Es un efecto que no llega a corromper el circuito, si no que es un error que se percibe a la salida. Este efecto es contrarrestado por los diseñadores de circuitos a través de la llamada Triple Modular Redundacy (TMR), que explicaremos más adelante.
2. Single Event Latchup (SEL): cuando la partícula con elevada energía pasa a través de una región sensible, puede llegar a provocar una situación de elevada corriente, cuya consecuencia es la pérdida de la funcionalidad del dispositivo [2]. Con el fin de evitar este tipo de fallos en el dispositivo, éstos cuentan con limitadores de corriente, que se encargar de reiniciar el dispositivo en el caso de que el exceso supere el umbral.
3. Single Event Gate Rupture (SEGR): es un evento producido cuando el golpeo de una partícula da como resultado una descomposición, y como consecuencia, una conducción de corriente a través de la puerta de un transistor MOSFET [3].
4. Single Event Burnout (SEB): este evento se produce cuando una partícula tiene la suficiente energía

como para hacer que un transistor pase a su estado de conducción. Si se da este efecto de manera prolongada puede llegar a provocar que el circuito no funcione de la manera esperada además de poder llegar a destruir el circuito en algunos casos.

Por tanto, habrá que buscar distintas formas de evitar, o de saber responder a estos efectos sobre los circuitos electrónicos. En un entorno espacial usamos circuitos integrados programables, o Field Programmable Gate Arrays (FPGA). La tecnología de las FPGA, que puede ser SRAM, flash o antifusible, es la que nos permite que sea reprogramable todas las veces que deseemos. Además, para protegerse frente a fallos, la memoria de configuración puede estar continuamente rescribiéndose (scrubbing). Es por ello que el uso de una FPGA puede funcionar como respuesta a la radiación, pero es insuficiente.

Por tanto, existen técnicas para dotar el circuito de una mayor protección. Una de las técnicas más usadas tradicionalmente es la de **blindar** directamente el propio circuito, o en su defecto, las partes más sensibles. Otra de las técnicas ya ha sido mencionada anteriormente, y es la de TMR, en el que se triplica gran parte del circuito para el caso en el que algo falle, haya otras dos partes del circuito exactamente iguales que puedan llevar a cabo esa función. El problema de estas técnicas es que aumentan de gran manera el coste de realizar estos circuitos, ya que es necesario triplicar todos los componentes del circuito, todas las conexiones entre dichos componentes...

Una de las técnicas con más fuerza en la actualidad, y sobre la que trataremos en este trabajo es la de proporcionar al circuito robustez por software. Esto quiere decir, que el circuito es suficientemente "inteligente" como para reprogramarse en caso de fallo (sensibilidad estática), además de de tener la capacidad de poder llegar a detectar algunas fuentes de errores y protegerse frente a ellos (sensibilidad dinámica). El problema de llevar a cabo esta técnica es que es necesario conocer la reacción del circuito frente a la radiación, además de conocer cómo reaccionará el circuito frente a un SEU.

Para llegar a conocer dicha reacción del circuito, se tienen dos tipos de posibilidades. La primera de ella son los llamados **test de radiación**, en los cuales, el circuito es expuesto de manera prolongada a radiación en centros especializados para ello, y una vez terminada la exposición, estudiar cómo se ha comportado el circuito frente a la radiación. Una vez obtenidos los resultados, se pueden llegar a sacar conclusiones acerca de la tolerancia frente a fallos del circuito (esto se llamará, vulnerabilidad del circuito frente a radiación) y conocer aquellas zonas más sensibles, y que necesitan de una mayor robustez. Además podemos llegar a saber cómo anticiparnos al fallo y dotar al circuito de una protección más específica. El principal problema de los test de radiación es su elevado precio para poder ser llevados a cabo, que ha hecho que se hayan buscado algunas alternativas a éstos test, además del gran tiempo que se invierte llevándolos a cabo. Otro problema, es que es necesario fabricar el circuito para poder irradiarlo.

Esta alternativa son las llamadas **plataformas de inyección de fallos** [4]. En ella, los SEUs son provocados de una manera controlada, además de darnos la posibilidad de conocer la salida cuando dicho SEU se ha producido. Por tanto estas técnicas nos ayudan a conocer la tolerancia frente a fallos de circuitos electrónicos. Una de las principales ventajas de esta técnica reside en su bajo precio frente a los test de radiación, y la rapidez para llevarlo a cabo. En cambio sabemos de distintas limitaciones de las plataformas de inyección de fallos.

Una de las dificultades para que la campaña llevada a cabo por la plataforma de inyección de fallos y los test de radiación sean lo más parecido posible, reside en la implementación del circuito que es radiado. Puede haber diferencias entre ambas implementaciones como puede ser compensaciones de fan-out, o simplemente diferencias en las posiciones de los pines. Otra posible diferencia reside en el número de bits que pueden llegar a sufrir un SEU, así, en las plataformas de inyección de fallos, pueden no incluir todos los posibles elementos del circuito donde un SEU puede llegar a ocurrir, y por tanto, y dependiendo de la importancia de dichos elementos del circuito, un mismo diseño puede tener una mayor o menor tolerancia a fallos con respecto al experimento llevado a cabo por la plataforma de inyección de fallos. Además, las plataformas de inyección de fallos inyectan un SEU por cada ejecución de los vectores de test[1], en cambio, la aleatoriedad de los test de radiación podrían inyectar uno, varios o ningún SEU por cada ejecución de los vectores de test. Existen otro tipo de limitaciones que serán comentadas con detalle más adelante. Estas limitaciones habrá que tenerlas en

[1] Se detallará más adelante el funcionamiento de la plataforma de inyección de fallos FTUNSHADES 2.

cuenta cuando llevemos a cabo la comparación entre los resultados proporcionados por los test de radiación y los proporcionados por la plataforma de inyección de fallos, teniendo especial cuidado a la hora de tratar los resultados.

En este trabajo, se ha llevado a cabo un determinado test de radiación usando la plataforma de inyección de fallos FTUNSHADES2 [5], en el que usaremos un "dummy mode", en el cual, no inyectamos ningún fallo, dejando que sea el haz de partículas el que produzca el SEU, que es capturado para su posterior análisis. Esto nos permitirá reproducir con exactamente la misma implementación, la misma FPGA, y el mismo software de control, para una vez obtenidos los resultados, éstos puedan ser comparados usando estadística.

Comprobaremos que, los resultados obtenidos por ambas partes son los esperados, teniendo en cuenta un determinado margen de error. La estadística usada para llevar a cabo la comparación y para calcular el margen de error se detallará más adelante.

2 TRABAJOS PREVIOS

"La ciencia es la progresiva aproximación del hombre al mundo real".

- Max Planck -

A lo largo de esta sección daremos a conocer ciertos trabajos previos que fueron realizados para que este trabajo pudiese llevarse a cabo. Estos trabajos son esenciales para lograr el fin de este proyecto y no fueron llevados a cabo por el autor de este documento. Los detallaremos a continuación:

2.1. Test de radiación en LANSCE

Como ya comentamos en la introducción, en este trabajo abordaremos la comparación de ciertos resultados proporcionados a partir de un test de radiación. Los resultados del test de radiación fueron facilitados al autor de este documento para la realización de este trabajo. El procesamiento de los resultados será comentado más adelante. Los datos técnicos son en base a una información dada por Yolanda Morilla García, miembro del CNA, y encargada de la realización del test de radiación, el cual fue llevado a cabo en Los Alamos Science Center (LANSCE) [8, 9] entre los días 18-24 de Septiembre de 2017.

2.1.1 Descripción del set-up

El haz de protones incide en un blanco de tungsteno donde se producen los neutrones (fuente n). El haz tiene un recorrido fijo hasta la línea de trabajo donde hay una cámara de ionización para medir la intensidad del haz. Las tarjetas con el DUT (Device Under Test) o dispositivo a irradiar se colocan a diferentes distancias desde ese punto, siempre en torno a 19.7 metros, como se muestra en la figura 2-1.

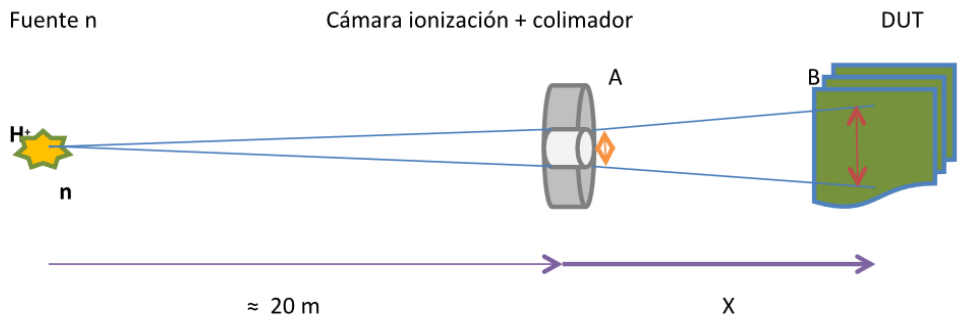


Figura 2-1. Representación del Set-Up. Fuente: Yolanda Morilla, CNA.

2.1.2 Cálculo del flujo y fluencia

En las hojas de parámetros del haz, se toma el valor "Sum above 10Mev" que indica la densidad de neutrones en **n/cm^2/pulso.** Teniendo en cuenta los datos reflejados en la tabla 2-1, se puede llegar a considerar un flujo constante, aunque en cada experimento habría que comprobar si ha habido interrupciones de duración considerable. Para el análisis de errores de cada ejecución del test de vectores habría que considerar el tiempo real de irradiación.

Septiembre 2017	DÍA				
Horario	19	20	21	22	23
Mañana	173165	171894	173165	171415	173523
Tarde	172900	172372	172900	172003	172857

Tabla 2-1. Flujos (n/cm^2/pulso) diarios aportados por LANSCE.

El valor medio de las 10 medidas es de 172619 con una dispersión <1%. A la vista de los resultados podemos pasar a realizar el cálculo de un flujo por unidad de tiempo. Tomamos el valor integrado en un minuto, y lo multiplicamos por el número de pulsos recogido en un minuto. Se ha comprobado que el número de pulsos por minuto está en torno 300 pulsos. Pero hay que tener en cuenta que la intensidad de corriente, en definitiva el flujo, es inversamente proporcional a la distancia al cuadrado. Por tanto, según la posición del DUT respecto al punto de integración de la corriente, tendremos que aplicar un factor de corrección.

La fluencia acumulada podrá ser determinada en función de la duración del mismo. Se muestra la fluencia acumulada por las tarjetas en la posición de los DUTs a lo largo de toda la campaña, en base al número total de pulsos registrados en el contador, y teniendo en cuenta el factor de corrección.

$$\boldsymbol{Fluencia\ total} = 0.56 * 2190060 * 172619 = \mathbf{2.1}\ (\boldsymbol{n/cm^2/pulsos})$$

2.1.3 Características a tener en cuenta de la realización del test de radiación

Como ya comentamos en la introducción, el experimento de radiación fue llevado a cabo con el mismo hardware y controlado por el mismo software que la plataforma de inyección de fallos. Como ya ha sido mencionado, fue desarrollado un "inyector dummy" únicamente con el fin de llevar a cabo la ejecución de los vectores de test sin introducir ningún tipo de SEU, así, de esta manera, el SEU detectado es el inducido por el haz de radiación. La fuente de alimentación era capaz de controlarse de manera remota.

El control del experimento se llevaba a cabo a partir de un script[2]. El script se basas en un bucle, que apaga y enciende la plataforma cada hora. Esto se lleva a cabo para evitar que un SEU, pueda llegar a convertirse en un fallo permanente, y que éste afecte al diseño durante más de una hora.

La plataforma ejecuta 750 runs[3] en una hora aproximadamente (más adelante se proporcionarán los resultados exactos), y hace una captura del estado interno de la FPGA cada 10 runs. Esta captura la llamaremos a partir de ahora **readback**, y pasaremos a comentarlo con más detalle en los siguientes capítulos. Después de llevarse a cabo el proceso de readback, la FPGA se reiniciaba y reconfiguraba con el fin de prevenir que el daño se fuera acumulando a lo largo del experimento. Esto podría llegar a provocar que todos los runs lleven consigo fallos, incluso cuando no se ha producido ningún SEU. Vemos por tanto, que tenemos dos líneas de defensa, la primera de ellas que consistía en encender y apagar la plataforma, y la segunda, en la que se reinicia y reconfigura la FPGA.

La duración total del experimento fue de aproximadamente 27 horas, en el que se llevaron a cabo un total de 20250 runs.

[2] Se mostrará más adelante.

[3] Llamaremos runs a partir de ahora a la ejecución completa de los vectores de test.

2.2. Diseño bajo test (DUT)

Una vez comentado el test de radiación, pasaremos a detallar el diseño sobre el que se ha aplicado dicha radiación. Este diseño consiste en un transmisor Zigbee modelado en lenguaje VHDL[4]. El diseño tiene una particularidad, y es que se añaden cuatro variantes del transmisor que se detallan a continuación:

1. Plain: El diseño original sin ningún tipo de redundancia.
2. Selective: Una modificación del diseño original en el que únicamente tiene redundancia los dos módulos más sensibles[5].
3. TMR: El diseño original con triple redundancia en todos sus elementos.
4. Hamming: En este diseño, se ha añadido un bloque antes de la primera etapa que computa el código Hamming de los datos que envía. De esta manera y a partir del código Hamming, podremos llegar a detectar errores y corregirlos.

El diseño final incluye 4 copias de cada una de las variantes con el fin de tener una mayor área sensible en la FPGA y de esta manera asegurarnos de que el haz inducirá errores en el diseño. En total, llegamos a ocupar alrededor del 30% de toda la FPGA, frente al 1.8% que ocuparía en el caso de no introducir ninguna variante, ni cuadruplicar ninguna de ellas. De esta manera conseguimos aumentar de manera considerable el área sensible de la FPGA, pero sin llegar a ocupar el área total.

Es importante conocer cómo es el diseño bajo test ya que, aunque para este trabajo no sea muy necesario el conocimiento de dicho diseño, la idea para un futuro es la de implementar un receptor que sea capaz de interpretar los datos enviados por el transmisor.

[4] Este diseño es parte del proyecto Edelweiss [6]

[5] El comportamiento del transmisor frente a radiación ha sido estudiado previamente mediante simulación.

3 ALCANCE DEL TRABAJO

"Son vanas y están plagadas de errores las ciencias que no han nacido del experimento, madre de toda incertidumbre".

- Leonardo Da Vinci -

En esta sección pasaremos a hablar más específicamente de lo que se pretende conseguir con la realización de este trabajo. Como ya hemos detallado en la sección anterior, la idea es partir de ciertos resultados obtenidos a partir de un test de radiación.

Por otro lado, se pretenderá reproducir de la manera más fiel posible, el mismo experimento que se ha llevado a cabo bajo un test de radiación mediante la plataforma de inyección de fallos FTUNSHADES 2 (a partir de ahora se abreviará por FTU2). Esto llevará consigo numerosos detalles que serán comentados en las siguientes secciones.

Una vez tenemos los resultados de ambos experimentos, pasaremos a realizar un análisis más detallado en relación al contraste de ambos resultados. Éstos resultados serán tratados de la manera adecuada y se realizará una comparación entre ambos en base a un estudio estadístico[6]. Dicho estudio estadístico será utilizado además para calcular el número de runs del experimento de inyección de fallos, y se demostrará que lo resultados están dentro de un margen de error definido por la estadística (teniendo en cuenta un cierto intervalo de confianza).

Los detalles serán explicados en las siguientes secciones, así como los problemas encontrados durante la realización del trabajo, y la manera de abordarlos. El trabajo se finalizará con unas conclusiones en base a los resultados obtenidos y comentando ciertas líneas de futuro.

[6] Dicho estudio estadístico será referenciado bibliográficamente más adelante en el documento.

4 FT-UNSHADES 2

"la ciencia es el alma de la prosperidad de las naciones
y la fuente de vida de todo progreso".

- Louis Pasteur -

Una vez ya hemos presentado los trabajos realizados con anterioridad, además de dar a conocer lo que pretendemos con este trabajo, pasaremos a presentar la plataforma de inyección de fallos que se ha usado para la obtención de resultados.

Es importante dar a conocer cómo es la plataforma, para así tener una idea del formato de resultados que vamos a tener cuando realicemos la campaña de inyección o cuando pasemos a tratar los datos del test de radiación.

Aún así, los detalles de la campaña de inyección de fallos que se ha llevado a cabo para la realización de este trabajo, así como los resultados de esta, serán comentados en otra sección más adelante.

4.1. Presentación

Se trata de una plataforma que controla tanto la parte software como la parte hardware, en la que se puede inyectar no sólo en los bits de usuario, sino también, en los bits de configuración de la FPGA (simulando lo que ocurre en un experimento de radiación). La parte hardware está compuesta por una placa madre y dos placas hijas idénticas. Cada una de estas placas hijas incluyen una FPGA, cuya función es la de implementar el DUT. Mientras se ejecuta los vectores de test en ambas FPGA, **sólo** inyectamos fallos en una de ellas, dejando la otra libre de fallos con el fin de poder comparar ambas salidas. En los siguientes apartados comentaremos tanto los ficheros necesarios, como los ficheros generados por la plataforma para el análisis de resultados.

Para poder llevar a cabo la campaña, se ha desarrollado un servidor web, en el que, para tener acceso, únicamente hace falta una contraseña y un usuario proporcionados por la Universidad de Sevilla.

4.2. Ficheros necesarios

Explicaremos los ficheros necesarios para poder llevar a cabo una campaña. Es necesario trabajar con la herramienta ISE 14.7 de Xillinx y usaremos el simulador ISim de Xillinx. El lenguaje para desarrollar los diseños puede ser VHDL. El objetivo es llegar a tener 3 ficheros principalmente:

1. Bitstream (.bit): Contiene la información de la localización de los bits en la FPGA. El bitstream configura la FPGA para emular cualquier circuito, y es generado a partir del fichero .vhd.
2. Logic Location (.ll): Indica la posición donde se establecen elementos como latches, flip-flops…

3. I/O vectors (.dat): Se obtiene a partir de un fichero .vcd. Este fichero es un "dump" de los vectores de entrada y de salida de todos los pines obtenidos a partir de la simulación del circuito. Una vez tenemos el fichero .vcd, la plataforma FTU2 genera el fichero .dat.

Con todos los ficheros ya generados pasaremos a explicar cómo se lleva a cabo una campaña.

4.3. Realización de campaña y ficheros generados

Existen varias guías que nos ayudan a la ejecución de las campañas, y por tanto, no es necesario entrar en detalles de cómo se llevan a cabo dichas campañas. Se pueden realizar las campañas de dos modos distintos, uno denominado **Modo Campaña** y el otro, **Modo Debug**.

Nosotros nos centraremos en comentar el Modo Campaña que es el que se ha utilizado para realizar la campaña de inyección de fallos de este trabajo.

Para llevar a cabo una campaña en modo trabajo, primeramente será necesario cargar los ficheros anteriormente mencionados. También se puede cargar un fichero de pines, aunque esto es opcional. Una vez tenemos cargados el .bit, el .ll y el .dat, podemos elegir las características que tendrá nuestra campaña. Esto nos permite llevar a cabo la campaña de la manera que mejor se adapte a nuestras necesidades.

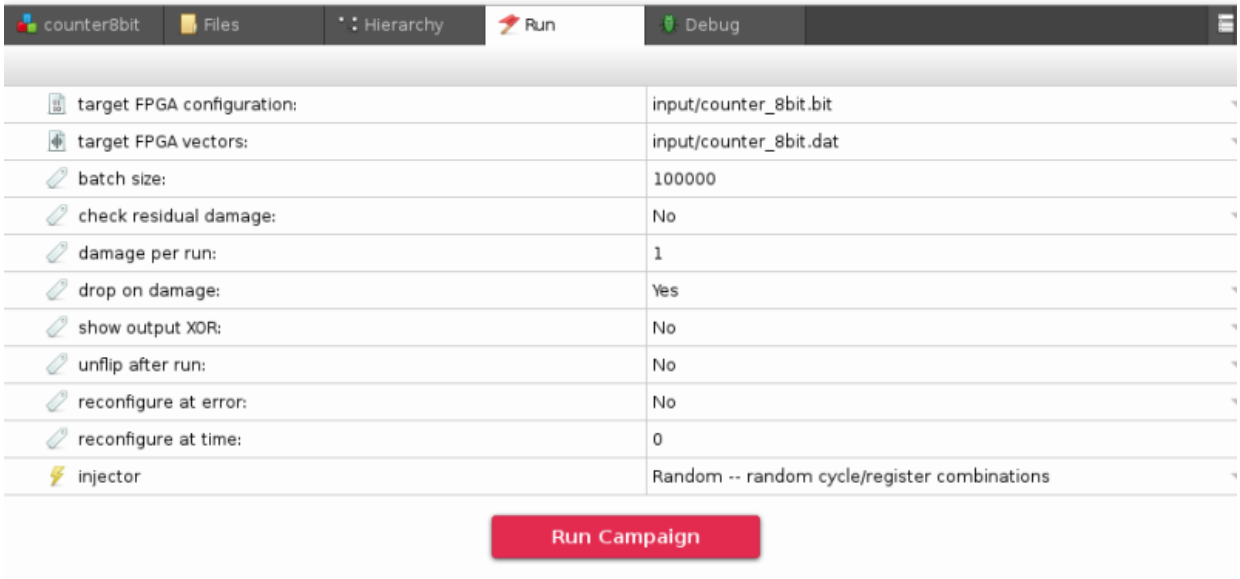


Figura 4-1. Parámetros de Campaña FTU2.

Como vemos en la figura 4.1, hay un gran número de parámetros que harán que nuestra campaña se amolde lo máximo posible a nuestros requerimientos. Los más interesantes de comentar son "**show output XOR"** y "**reconfigure at…"** El primero de ellos, nos dará en el fichero de damages.csv, que comentaremos más adelante, la XOR de las salidas sin fallos con las salidas con fallos. Si no se activa esta opción, únicamente aparecerá en el damages.csv el ciclo dónde se ha producido el fallo, sin dar más detalles. El segundo de los parámetros a destacar, nos permite reconfigurar la FPGA cuando queramos, con el fin de evitar que cualquier fallo se propague. Este parámetro de reconfiguración, es el que se ha configurado como una de las líneas de defensa comentadas en la sección de trabajos previos.

Una vez hemos configurado la campaña a partir de los parámetros deseados, pasaremos a configurar el inyector. **El inyector introduce un SEU por cada run**. Destacaremos dos opciones del inyector aunque tenga algunas más. La primera de ellas es el inyector **Random**, en el que elegiremos nosotros el número de SEUs que queremos inyectar, y se irá inyectando un SEU por ciclo elegido aleatoriamente, como podemos ver en la figura 4-2.

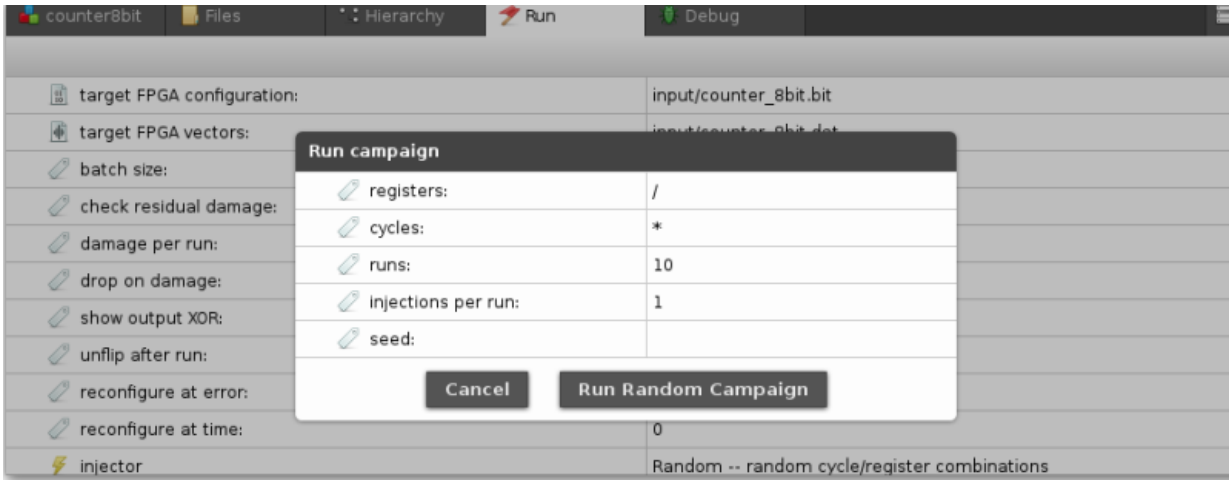


Figura 4-2. Campaña Random.

La otra opción a destacar del inyector es **Exhaustive Campaign,** en el que el número total de runs es el equivalente a inyectar en cada uno de los bits del diseño, y para cada bit inyectar en cada uno de los ciclos. Este tipo de campaña nos proporciona un conocimiento completo de la reacción del circuito frente a SEUs, y por tanto frente a una posible radiación. El problema de este tipo de campañas reside en la duración, llegando a ser inviables en algunos casos[7].

Una vez completada la campaña, la plataforma nos generará una serie de ficheros que nos permitirá analizar los resultados de dicha campaña. Estos ficheros se enumeran a continuación:

1. damages.csv: Es el resultado de la campaña. Dependiendo de las opciones que hayamos seleccionado tenemos un tipo de fichero u otro.
2. injections.csv: Una descripción de todos los runs ejecutados durante la campaña. Básicamente indica el bit y el ciclo donde se ha inyectado el SEU en cada run, con el formato: **ciclo:reg;**
3. reg_names.txt: Es una lista de todos los bits donde se han introducido SEU. El propósito de este fichero es el de asignarle un número a cada bit, que será usado en el injections.csv. El primero de la lista corresponde con el número uno, el segundo con el dos… Esto lo podemos ver en la figura 4-3.

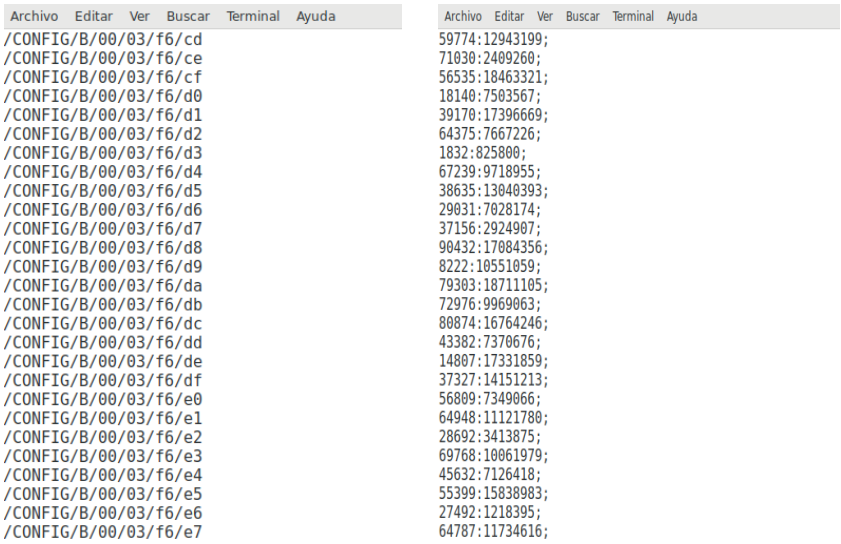


Figura 4-3. Ejemplo de reg_names.txt (izquierda) y de injections.csv (derecha).

4. stats.txt: Algunas estadísticas tomadas mientras se ejecutaba la campaña.

[7] Más adelante en el documento se darán más detalles sobre este tipo de campañas.

5 PROCESADO Y ANÁLISIS DE LOS DATOS OBTENIDOS DEL TEST DE RADIACIÓN

"La ciencia está hecha de datos, como una casa de piedras. Pero un montón de datos no es ciencia más de lo que un montón de piedras es una casa".

- Henri Poincaré -

Antes de pasar a comentar los detalles del procesado de los datos obtenidos del test de radiación, vamos a explicar un parámetro que ya fue mencionado en la sección 2, y que detallaremos a continuación. Este parámetro es el readback.

Los readbacks, y como ya dijimos, no es más que una captura del estado interno de la FPGA. Esta captura estará reflejada en un archivo binario, cuyo contenido no es más que el estado de los bits que forman una FPGA. Por tanto, y analizando el fichero de readback podemos sacar conclusiones de qué partes del circuito están a nivel alto, y cuáles están a nivel bajo, y a partir de ahí, llegar a conocer, si ha habido algún SEU, únicamente viendo si se ha producido algún "flip" en algún bit. A partir de un fichero de readback podemos llegar a ver el número total de bits que componen una FPGA, simplemente examinando su tamaño. Como ya comentamos en la sección 2, se ejecuta una campaña por cada hora, en la que se llevan a cabo un total de 750 runs. Esto implica que, si realizamos un readback cada 10 runs, tenemos un total de 75 readbacks por cada campaña.

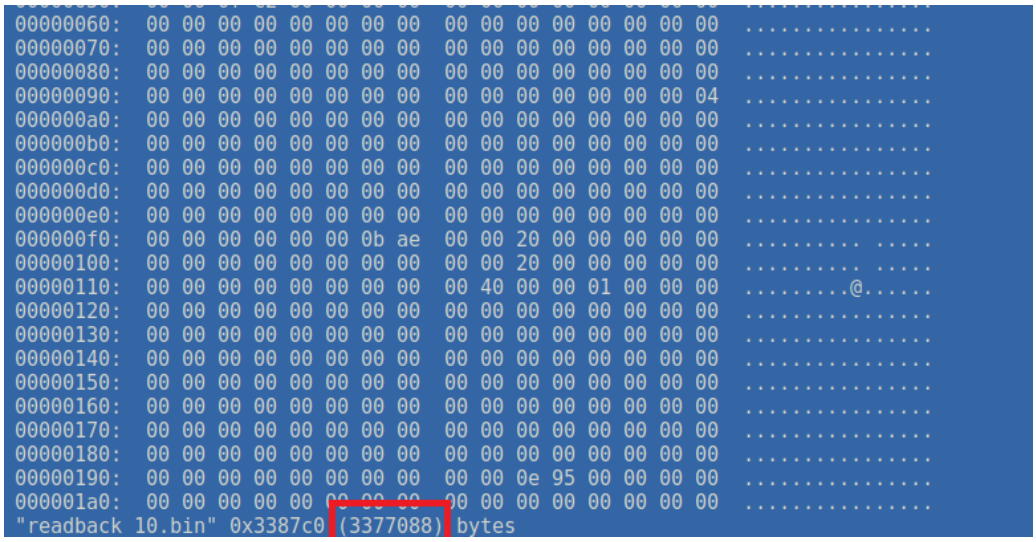


Figura 5-1. Análisis de un fichero de readback.

En la figura 5-1, podemos ver cómo es un fichero de readback, en este caso, el que corresponde después de los 10 primeros runs (readback_10.bin). Se puede comprobar que este fichero muestra únicamente el estado de una gran cantidad de bits y aunque visto de esta forma es difícil sacar ningún tipo de conclusión, podemos ver el tamaño, que es 3377088 bytes, que da un total de **27016704** bits.

5.1. Presentación del formato de resultados obtenidos del test de radiación

Una vez ya hemos presentado el concepto de readback, indispensable para poder continuar, pasaremos a comentar con detalle la estructura de cada campaña de radiación con el fin de dar a conocer qué resultados tenemos, y qué podemos hacer con ellos.

Como ya hemos comentado anteriormente, el software para obtener los datos de radiación es el mismo que para obtener los datos de la plataforma de inyección de fallos FTU. Como también se ha comentado, se deja que sea el propio haz el que introduzca el SEU. Así los ficheros generados son los mismos que generaríamos a partir de FTU2, y que han sido descritos en la sección **4.3**.

Para llevar a cabo el test de radiación, se han realizado 27 campañas, con 750 runs cada campaña, que hacen un total de 20250 runs[8]. El nombre de la carpeta donde está cada una de las 27 campañas corresponde con la fecha y hora de realización de la campaña, como podemos ver en la figura 5-2.

```
danvelcal@walle:~/EXP2_1_14/results$ ls
2017-09-19-22-15-29  2017-09-20-05-26-11  2017-09-20-12-34-21  2017-09-20-19-41-09
2017-09-19-23-16-31  2017-09-20-06-26-33  2017-09-20-13-35-30  2017-09-20-20-46-15
2017-09-20-00-17-32  2017-09-20-07-27-44  2017-09-20-14-36-12  2017-09-20-21-48-08
2017-09-20-01-18-25  2017-09-20-08-28-50  2017-09-20-15-37-05  2017-09-20-22-49-16
2017-09-20-02-18-58  2017-09-20-09-29-26  2017-09-20-16-38-14  2017-09-20-23-51-23
2017-09-20-03-20-47  2017-09-20-10-31-26  2017-09-20-17-39-00  2017-09-21-00-52-26
2017-09-20-04-21-27  2017-09-20-11-32-19  2017-09-20-18-40-37
```

Figura 5-2. Campañas del Test de Radiación.

La razón por la que se ha realizado en 27 campañas y no en una sólo, es por lo que comentamos en la sección de trabajos previos. La plataforma se reinicia cada hora, con el fin de evitar que cualquier error producido en la placa pueda provocar un fallo que se repita en todos los runs, provocando resultados que no se correspondan con lo que estamos buscando.

Los ficheros de cada campaña se muestran en la figura 5-3.

```
danvelcal@walle:~/EXP2_1_14/results/2017-09-19-22-15-29$ ls
damages.csv            readback_250.bin  readback_440.bin  readback_630.bin
injections.csv         readback_260.bin  readback_450.bin  readback_640.bin
readback_0.bin         readback_270.bin  readback_460.bin  readback_650.bin
readback_100.bin       readback_280.bin  readback_470.bin  readback_660.bin
readback_10.bin        readback_290.bin  readback_480.bin  readback_670.bin
readback_10.bin.hexer  readback_300.bin  readback_490.bin  readback_680.bin
readback_110.bin       readback_30.bin   readback_500.bin  readback_690.bin
readback_120.bin       readback_310.bin  readback_50.bin   readback_700.bin
readback_130.bin       readback_320.bin  readback_510.bin  readback_70.bin
readback_140.bin       readback_330.bin  readback_520.bin  readback_710.bin
readback_150.bin       readback_340.bin  readback_530.bin  readback_720.bin
readback_160.bin       readback_350.bin  readback_540.bin  readback_730.bin
readback_170.bin       readback_360.bin  readback_550.bin  readback_740.bin
readback_180.bin       readback_370.bin  readback_560.bin  readback_80.bin
readback_190.bin       readback_380.bin  readback_570.bin  readback_90.bin
readback_200.bin       readback_390.bin  readback_580.bin  readback_golden.bin
readback_20.bin        readback_400.bin  readback_590.bin  run.tcl
readback_210.bin       readback_40.bin   readback_600.bin  stats.txt
readback_220.bin       readback_410.bin  readback_60.bin
readback_230.bin       readback_420.bin  readback_610.bin
readback_240.bin       readback_430.bin  readback_620.bin
```

Figura 5-3. Ficheros de cada campaña.

[8] Este número de runs es importante, y se comentará más adelante.

Podemos ver los ficheros comentados en la sección **4.3** (damages.csv, injections.csv, stats.txt…), además de los 75 readbacks de esta campaña. Cabe destacar que el fichero injections.csv está vacío, ya que, como dijimos, el inyector es "dummy", y se deja que sea el haz el que introduzca el SEU.

En los próximos apartados comentaremos el tratamiento de dichos ficheros. La idea es que los mismos scripts usados para tratar estos resultados, sean los que se usan para tratar los resultados del experimento de inyección de fallos.

5.2. Conocimiento del número de SEUs provocados por el haz a partir de fichero de readback

La principal utilidad de los ficheros de readbacks es la de poder conocer el número de SEU que se han producido cada 10 runs. Por tanto, y con el fin de contabilizar el número de SEU que han sido inducidos por el haz, se creó un nuevo tipo de readback, llamado **golden_readback**. El golden_readback corresponde con la configuración inicial de la FPGA, además, el estado de los bits que contiene este fichero, deberían ser los que tuvieran los bits de los readbacks al final de cada run. De esta manera las discrepancias que haya entre los readbacks y el golden_readback corresponden con daños causados por radiación. Entre esas discrepancias podemos distinguir dos tipos: aquellas que dan lugar a errores en la salida por cada lote[9] y aquellas que no. Este último tipo es importante, en el sentido que, si esas discrepancias no producen errores en la salida, eso significa que no se propaga internamente por el plano de usuario de la FPGA, o por lo menos, no se propaga de manera significativa.

5.2.1 Obtención de la posición de los SEU en la FPGA

Si algo nos permite el golden_readback es conocer la posición dónde se ha producido el SEU. Simplemente realizando una XOR entre ambos ficheros (readback_x y golden_readback), se puede llegar a conocer la posición en la FPGA de los bits dónde se ha producido un SEU y por consiguiente, contar cuantos SEUs se han producido. Esto se ha llevado a cabo a partir del script que se muestra en la Figura 5-4.

```
#!/bin/bash


file_count=0
dir_count=0

for i in results/*
do

        mkdir diff_readbacks_v3/campaing_$((++dir_count))
        for j in `ls $i/ | sort -t _ -nk 2 | grep "^readback_[0-9]."`
        do
                echo $(tnt-mask  --xor $i/$j $i/readback_golden.bin | tnt-dump -1) > di
ff_readbacks_v3/campaing_$dir_count/$j
        done
        file_count=0
done
```

Figura 5-4. Script en Shell para obtener posición de los SEUs en la FPGA.

Como consecuencia de ejecutar este script se obtiene, a partir de los readback, la posición dónde se han producido los SEU, además de contabilizar cuántos SEUs se han producido, únicamente contando el número de posiciones.

Una vez ejecutado el script, el resultado no fue el que se preveía en primera instancia. Cada readback tenía alrededor de 10 mil discrepancias respecto al golden_readback, lo que resultó confuso ya que el resultado esperado, y teniendo en cuenta estudios de radiación anteriores, era de en torno a 12. Por lo que se llevó a cabo un conteo de frecuencia de las posiciones, por si algunas se repitieran más que otras, con el fin de aclarar la procedencia del elevado número de discrepancias.

[9] Denominaremos lote a partir de ahora a cada uno de los conjunto de 10 runs.

Para ello, se llevó a cabo el desarrollo de un script, para contar todas las posiciones que coincidieran de todos los readbacks, y las escribiera en otro fichero con el formato: **posición : número de veces que se repite la posición**. Este script se detalla en la figura 5-5.

```
import sys

foo = {}
files=[]


def read_files(file):
        for l in file:
                for j in l.strip("\n").split(" "):
                        if j in foo:
                                foo[j] = foo[j] + 1
                        else:
                                foo[j] = 1


with open(sys.argv[1]) as f:
        for l in f:
                files.append(l.strip("\n"))

for i in files:
        with open(i) as f:
                read_files(f)

with open("readback_analysis", "w") as f_out:
        for k,v in foo.iteritems():
                f_out.write(str(k) + ":" +str(v) + "\n")
```

Figura 5-5. Script en Python para conteo de frecuencia de posiciones.

En la figura 5-6, se muestra una parte del fichero generado:

```
17817475:2
24117119:1
24117118:1
7189628:1050
7189626:1050
24117110:1
24117117:1
24117116:1
7189623:1050
9549799:1
15025474:1
24619882:1
24619883:1
24619880:1
24619881:1
19539071:1
```

Figura 5-6. Porción del fichero de salida del conteo de frecuencia de posiciones.

Los resultados de este análisis fueron bastante esclarecedores. Vemos en la figura 5-6, como hay algunas posiciones que se repiten a lo largo de los 1050 readback que se han realizado durante todas las campañas. Por tanto, se concluyó que estos errores sistemáticos provienen de la propia FPGA y que no son producidos por el

haz de radiación. Por tanto, dichas posiciones fueron quitadas para el análisis final, dando como resultado unos números más cercanos a los esperados.

5.2.2 Calculo de SEU/lote y de SEU/s del test de radiación

Aunque el SEU/lote y el SEU/s no sean datos que puedan ser comparados con los resultados obtenidos a partir de la plataforma de inyección de fallos, sí son de interés para conocer la resistencia del diseño frente a radiación.

Primeramente, los bits sistemáticos fueron eliminados de los ficheros con las posiciones. Esto se llevó a cabo mediante el script que se muestra en la Figura 5-7.

```
#!/bin/bash

cuenta=0
dir_count=0

for j in `ls ~/EXP2_1_14/diff_readbacks_v4/ | sort -t _ -nk 2 | grep "^campaing"`
do
        mkdir campaing_$((++dir_count))
        for i in `ls ~/EXP2_1_14/diff_readbacks_v4/$j | sort -t _ -nk 2`
        do
        diff -y <(sort ~/EXP2_1_14/diff_readbacks_v4/$j/$i) <(sort ../readbacks/systema
tic/systematic_RT) | grep "<" | grep -o "^[0-9].\{1,15\}" >  campaing_$dir_count/$i
        #$((cuenta++))
        cuenta=$((cuenta+1))
        done
        cuenta=0
done
```

Figura 5-7. Script en Shell para eliminar los bits sistemáticos.

Una vez eliminados los bits sistemáticos, únicamente quedan las posiciones donde realmente se han producido un SEU, y por tanto el número de SEUs producidos se puede obtener de manera sencilla.

Primeramente se llevó a cabo de SEUs por segundos. Para ello, partiremos el fichero **stats.txt**, para calcular la duración exacta de cada campaña y, una vez sumada la duración de cada campaña, obtener la duración total del test de radiación. Por otro lado, los ficheros con las posiciones donde se han producido SEU fueron usados para hacer un recuento total de los SEUs. Para llevar todo esto a cabo, se ha usado el script que se muestra en la figura 5-8.

```
import sys

files=[]
seg=[]
seg_totales=0
seu_totales=0
contador=0

def read_files(file_x, contador):
        global seu_totales
        global seg
        seus=0
        for l in file_x:
                seus=seus+float(l.strip("\n"))
                seu_totales=seu_totales+float(l.strip("\n"))
        #print str(seus), str(seu_totales)
        print "El numero de seus/s es de: " + str(float(seus)/float(seg[contador]))


with open(sys.argv[1]) as f:
        for l in f:
                x=l.strip("\n").split(" ")[3].split(":")
                seg.append(float(x[0])*3600 + float(x[1])*60 + float(x[2]))
                seg_totales=seg_totales+(float(x[0])*3600 + float(x[1])*60 + float(x[2]
))
```

```
with open(sys.argv[2]) as f:
        for l in f:
                files.append(l.strip("\n"))



for i in files:
        with open("seu_campaing/" + i) as f:
                read_files(f, contador)
        contador=contador + 1


print "El numero de seus/seg totales es de: " + str(float(seu_totales)/float(seg_totale
s))
print seg_totales
print seu_totales
```

Figura 5-8. Script en Python para calcular el SEU/s.

Se ha obtenido, una vez ejecutado el script, un valor de **47.789 seg/lote**, y un resultado de **0.29 SEU/s**, para los datos del test de radiación**.** Además, se ha obtenido un valor de **12.78 SEU/lote**, o lo que es lo mismo **1.28 SEU/run**.

5.3. Cálculo del Architecture Vulneraribliity Factor (AVF)

A continuación, procederemos al detallar el cálculo del parámetro fundamental que será usado para comparar con los resultados del experimento de inyección de fallos, y este es el Architecture Vulnerability Factor (AVF), que no es más que la relación entre el número de runs con fallos y el número total de runs:

$$AVF = \frac{Número\ de\ runs\ \text{con}\ errores\ en\ la\ salida}{Número\ total\ de\ runs}$$

A partir de este valor nos podremos hacer una idea de la vulnerabilidad del diseño frente a radiación. Así, un diseño con un AVF mayor será más sensible a la radiación que otro con un menor AVF.

Para el cálculo de este valor, únicamente será necesario el fichero damages.csv de cada campaña, en el que cada run sin errores en la salida se indica con un "**;**".

Para el cálculo de este parámetro se ha usado el script que se muestra en la figura 5-9.

Una vez ejecutado el script, podremos el AVF de cada campaña puede ser calculado, y de manera muy sencilla, llegar conocer el AVF del test de radiación completo.

Como detalle comentar, que dicho script es capaz de coger todos los ficheros damages.csv de todas las campañas, incluso estando estos en distintos directorios. Esto es debido a un pequeño script, que no pondremos debido a su baja importancia, que consigue volcar la ruta de todos los damages.csv en otro fichero. El script de python es capaz de, a partir de ese fichero con las rutas, procesar los damages.csv de tal forma que pueda calcular el AVF.

A continuación, se detallan los resultados del AVF por campaña y total, en forma de tabla:

Campaña	Fecha/Hora	AVF	Campaña	Fecha/Hora	AVF
Campaña_1	19-09-17/22:15:29	3.067%	Campaña_2	19-09-17/23:26:31	1.87%
Campaña_3	20-09-17/00:17:32	2.133%	Campaña_4	20-09-17/01:18:25	0.8%
Campaña_5	20-09-17/02:18:58	3.2%	Campaña_6	20-09-17/03:20:47	1.6%
Campaña_7	20-09-17/04:21:27	4.53%	Campaña_8	20-09-17/05:26:11	1.06%
Campaña_9	20-09-17/06:26:33	1.87%	Campaña_10	20-09-17/07:27:44	1.47%
Campaña_11	20-09-17/08:28:50	0.93%	Campaña_12	20-09-17/09:29:26	5.47%
Campaña_13	20-09-17/10:31:26	2.8%	Campaña_14	20-09-17/11:32:19	0.8%
Campaña_15	20-09-17/12:34:21	2.67%	Campaña_16	20-09-17/13:35:30	1.6%
Campaña_17	20-09-17/14:36:12	1.73%	Campaña_18	20-09-17/15:37:05	1.6%
Campaña_19	20-09-17/16:38:14	3.6%	Campaña_20	20-09-17/17:39:00	2.67%
Campaña_21	20-09-17/18:40:37	0.4%	Campaña_22	20-09-17/19:41:09	6%
Campaña_23	20-09-17/20:46:15	3.87%	Campaña_24	20-09-17/21:48:08	2.8%
Campaña_25	20-09-17/22:49:16	5.87%	Campaña_26	20-09-17/23:51:23	2%
Campaña_27	21-09-17/00:52:26	7.06%	**TOTAL**	-	**2.72%**

Tabla 5-1. Resultados AVF por campaña y AVF total.

Como se puede observar en la tabla 5-1, el AVF del test completo es de **2.72%**. Este valor será usado más adelante para poder llevar a cabo la comparación con el AVF obtenido de la campaña de inyección de fallos, por lo que será un valor importante a tener en cuenta.

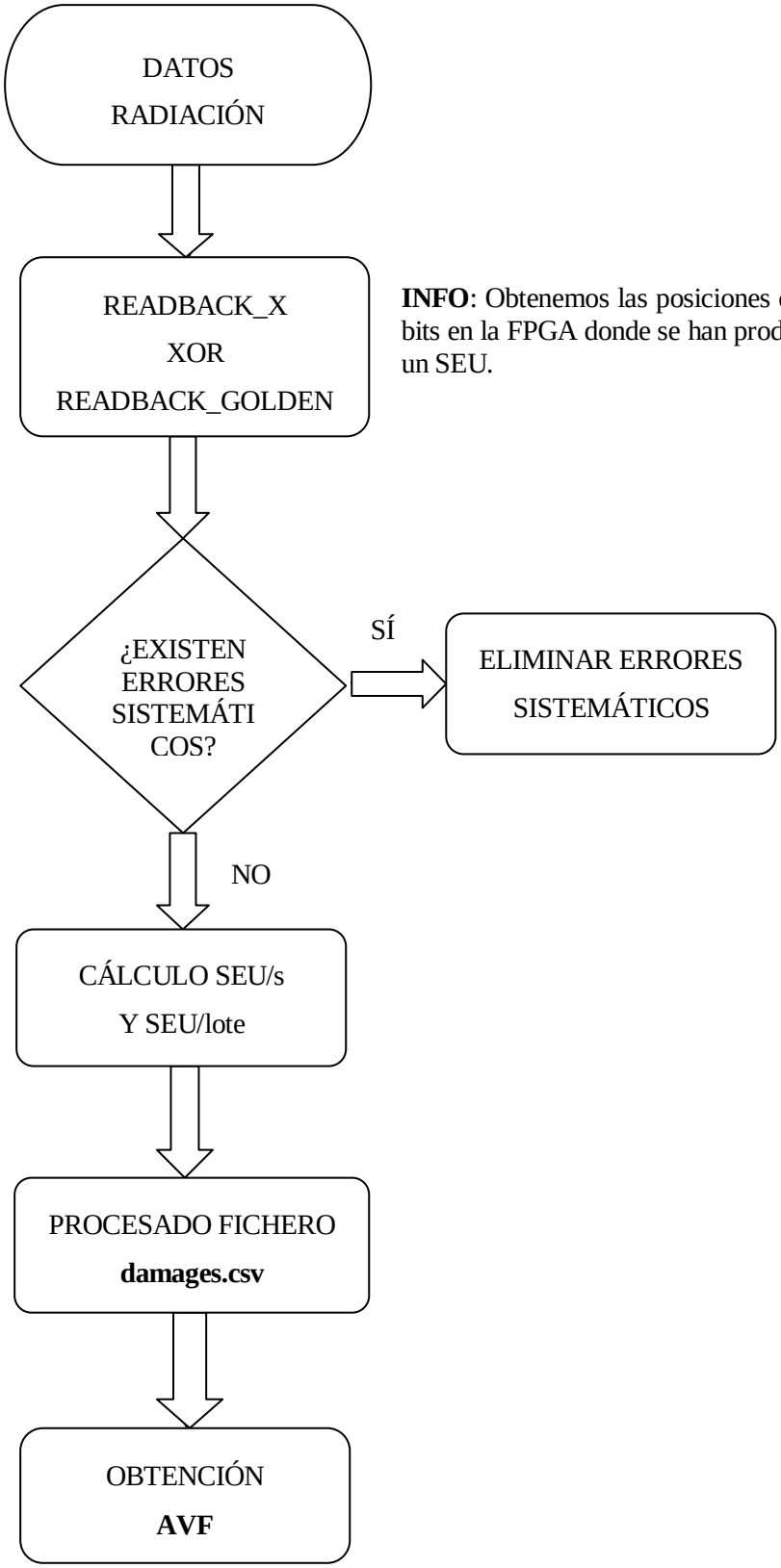
Cada experimento por separado, tiene, como ya ha sido comentado anteriormente, un total de 750 runs. Por tanto, el margen de error entre experimentos, y si tenemos en cuenta una serie de fórmulas que detallaremos más adelante, es de **9.2045%**. Es por eso que nos salen valores tan distantes, pero siempre dentro de ese margen de error.

```
import sys

line_with_error=0
line=0

with open(sys.argv[1]) as f:
        for l in f:
                line += 1
                x=l.strip("\n")
                if x[0] != ";":
                        line_with_error += 1
                        #print "Fail in pos " + str(line)
                #else:
                        #print "not"

#print float(10)/float(8)
print "The Architecture Vulneravity Factor is " + str((float(line_with_error)/float(lin
e))*100) + "%"
```

Figura 5-9. Script en Python para cálculo de AVF.

5.4. Diagrama de flujo del procesado de datos

En esta sección, se detallará en forma de diagrama de flujo, el proceso seguido para poder llevar a cabo el procesado de datos con el fin de esclarecer dicho procedimiento.



6 REPRODUCCIÓN DEL EXPERIMENTO DE RADIACIÓN CON FTU2

"Lo más incomprensible acerca de este mundo es que es comprensible".

- Albert Einstein -

En este apartado se explicará como se ha llevado a cabo la reproducción del test de radiación mediante la plataforma de inyección de fallos FTU2. Se intentará dar explicación a ciertos fenómenos que han surgido a lo largo de la realización del mismo.

Como ya ha sido comentado al principio de este documento, para llevar a cabo la comparación entre ambos resultados, se hará uso de la estadística. Dicha estadística será presentada a continuación.

5.5. Estadística usada para este trabajo

Es sabido que una de las principales limitaciones de las plataformas de inyección de fallos para diseños que son mediamente complejos, reside en el número de inyecciones que se tienen que llevar a cabo. Para poder llevar a cabo un estudio completo del efecto de la radiación sobre cualquier circuito electrónico, será necesario realizar una **campaña exhaustiva**, esto es, inyectar en todos los bits, en todos los ciclos de nuestro diseño. Por tanto, habrá que inyectar en todos los ciclos de test de vectores y en todas las localizaciones posibles:

$$c_{total} = (c_{clocks})^2 * i_{locations}$$

Donde c_{clocks} es el número de ciclos del test de vectores y $i_{locations}$ es el número de posibles inyecciones. Podemos ver que, conforme aumentar la complejidad del diseño, la duración de la campaña exhaustiva incrementa linealmente con el número de posibles inyecciones y cuadráticamente con el número de ciclos. Así por ejemplo, un diseño con 10 000 bits y un test de vectores de 100 000 ciclos de reloj da como resultado una campaña exhaustiva de 10^9 ejecuciones. Para un caso como el nuestro en el que tenemos millones de elementos, una campaña exhaustiva se vuelve inviable.

Habrá que considerar por tanto, una campaña de inyección de fallos que sea lo más fiel posible a una campaña exhaustiva, y para ello aplicaremos la estadística con el fin de **reducir** el número de inyecciones, pero pagando el precio de considerar un cierto margen de error.

El estudio estadístico llevado a cabo para calcular dicho número de inyecciones es en base a una publicación de unos investigadores del TIMA Laboratory en Grenoble [7]. Se trata de un estudio estadístico para

plataformas de inyección de fallos, en el que, a partir de ciertas fórmulas, podemos obtener el número de inyecciones para un diseño dado un cierto margen de error, o de manera inversa, calcular el margen de error de nuestros resultados en base a un determinado número de inyecciones.

El Doctor Hipólito Guzmán Miranda realizó unas tablas en Excel en base a las fórmulas de dicho estudio estadístico. Dichas tablas las podemos ver en la figura 6-1.

	Fill the data inside the boxes		Based on the paper ->	https://ww					Temporary variables (hidden) ->			Temporary variables (hidden) ->	
	Design name	Number of Flip-flops (F)	Number of clock cycles (C)	Standard error (p)	Confidence level (%)	Confidence level cut-off point (t)	Population size (N = F*C)	Desired error margin (e)	->	Number of faults to inject (n)	Number of injected faults (n)	->	Actual error margin (e)
4	8-bit counter	8	300	0.5	99	2.58	2400	0.01	->	2097.605042	2098	->	0.009992526665
5	Edelweiss TX FPGA	19000000	100000	0.5	99	2.58	1900000000000	0.01	->	16640.99985	1500	->	0.03330765676
6	Test (delete this)	100000	100000	0.5	90	1.645	10000000000	0.01	->	6765.057924	6766	->	0.009999303791
7	Counter 8 bit user FFs	8	285	0.5	99	2.58	2280	0.01	->	2005.363636		->	
8	Counter 8 bit Config Bits	2770	285	0.5	99	2.58	789450	0.01	->	16297.48223	8887	->	0.01360673922
9	Counter 8 bits all bits	2778	285	0.5	99	2.58	791730	0.01	->	16298.45112		->	
10	MSP430 FR5969	720	1203	0.5	95	1.96	866160	0.1	->	96.0294631	100	->	0.09799439925
11	TX_Zigbee	11032	100000	0.5	99	2.58	1103200000	0.01	->	16640.749		->	
12	Edelweiss TX FPGA radiati	26400000	100000	0.5	99	2.58	2640000000000	0.01	->	16640.9999	17250	->	0.009821892108
13		10000	100000	0.5	99	2.58	1000000000	0.0125	->	10650.12658		->	
14							0		->			->	
15							0		->			->	

Figura 6-1. Tabla Excel en base a las fórmulas de [7]

La base de cada una de las fórmulas es la llamada “Population size”, que no es no es más que el producto del número total de bits que pueden llegar a sufrir un SEU (F:Number of Flip-flops) por el número total de ciclos que componen nuestro diseño y que viene determinado por el test de vectores (C:Number of clock cycles).

$$N = F * C$$

Por suerte, estos investigadores pudieron demostrar que el número de inyecciones satura en torno a un valor. Por tanto no importa lo grande que sea “N” o lo mucho que aumente, que el número de inyecciones se mantiene siempre en torno a un valor. Esto lo podemos ver en la figura 6-2.

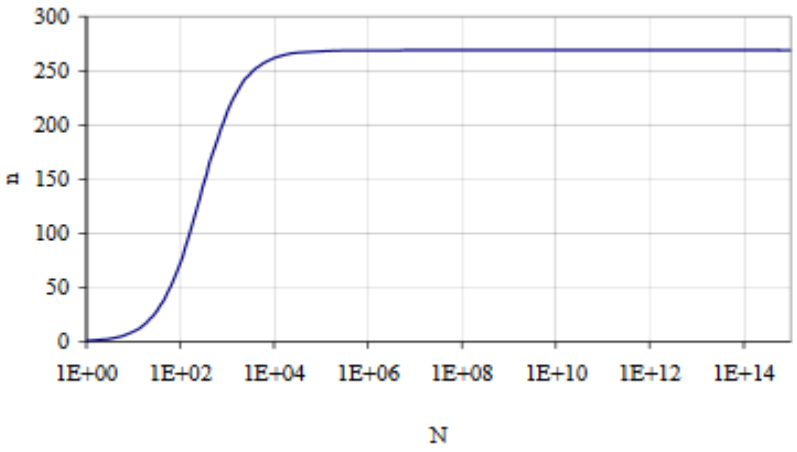


Figura 6-2. Imagen extraída de [7]

Gracias a esto podemos obtener márgenes de error muy pequeños, sin necesidad de realizar un gran número de inyecciones.

5.6. Preparación del experimento

Como ya se ha comentado, se procederá a tratar de reproducir el test de radiación mediante la plataforma de inyección de fallos FTU.

Se van a usar tanto el mismo fichero VHDL y el mismo test de vectores que se usó para el test de radiación con el fin de que los resultados sean lo más parecido posible. Como es sabido, el haz de radiación es totalmente aleatorio y puede afectar a todos los bits de la FPGA. Para intentar inyectar en todos los bits de la FPGA, se ha desarrollado un fichero llamado **full.ll**. Este fichero se ha creado a partir de una opción de Xillinx que nos proporciona los "bits esenciales" de un diseño. Modificando algunos parámetros, llegamos a conocer que la FPGA tiene un total de **18936096** bits esenciales. Estos son los bits sobre los que nuestra plataforma provocará los SEUs. Destacar que este número es bastante menor que los 27016704 bits que pueden llegar a sufrir SEU en el test de radiación, y que ya ha sido calculado anteriormente. La explicación para cubrir esa diferencia se detallará más adelante.

Por tanto, usando los ficheros explicados en la sección 4, podremos reproducir el test de radiación mediante la plataforma de inyección de fallos. Una vez cargados los ficheros, tendremos que seleccionar el tipo de campaña, que como ya hemos comentado y teniendo en cuenta que una campaña exhaustiva es algo inviable, se llevará a cabo una campaña "random". El número de inyecciones ha sido calculado usando las tablas de Excel que se muestran la Figura 6-1.

Design name	Number of Flip-flops (F)	Number of clock cycles (C)	Standard error (p)	Confidence level (%)	Confidence level cut-off point (t)	Population size (N = F*C)	Desired error margin (e)	->	Number of faults to inject (n)
Edelweiss TX FPGA	18936096	100000	0.5	99	2.58	1893609600000	0.01	->	16640.99985

Figura 6-3. Cálculo para el número de inyecciones.

Podemos ver en la figura 6-3 que para nuestro diseño, que tiene un total de casi 19 millones de bits, tendremos que inyectar **16641** fallos para obtener un margen de error del 1%, con un intervalo de confianza del 99%. Cabe destacar que, como hemos comentado en la figura 6-2, a pesar de tener una "N" muy grande (1893609600000), el número de inyecciones es un valor viable.

Comentar también, y aunque entraremos con más detalle en las próximas secciones, el cálculo del margen de error que tenemos de los resultados del test de radiación. Usando las mismas fórmulas, y teniendo en cuenta que se lleva cabo un total de 20250 runs, el margen de error calculado para un intervalo de confianza del 99% es de **0.9065%**. Es un margen de error un poco menor que el 1%, ya que realizamos más de 16641 runs.

Una vez ha sido calculado el número de inyecciones que se debería realizar, sólo basta con ejecutar el script que se muestra en la Figura 6-4.

```
dev_open
dev_timeout 60
dev_panic
load_pin exp.pin
#load_ll exp.ll
load_ll full.ll
load_config exp.bit
load_vectors exp.dat
#bit_ls
#run_set batch_size 10
run_set check_residual_damage 1
run_set show_output_xor 1
run_set damage_per_run 100000
run_set readback_at_time 10
run_set reconfigure_at_time 10
#run_clean 750
#for { set i 0 } { $i < 14 } { incr i } {
run_random * * 16641 1.4039 3.4781 10
```

Figura 6-4. Script en tcl para campaña de inyección de fallos.

Dicho script es el mismo que se ejecutó para llevar a cabo el test de radiación. Puede verse en el script la opción de realizar un readback cada 10 runs, o la de reconfigurarse la FPGA cada 10 runs entre otras. Como ya hemos comentado varias veces a lo largo del documento, en el test de radiación se ejecutaba un inyector "dummy" que no realizaba ninguna inyección, dejando que fuera el haz de radiación el que introdujera el SEU. En este caso, sólo habrá que comentar esa línea (rectángulo rojo) y añadir una nueva que no sea "dummy" (rectángulo azul), para así poder llevar a cabo un total de 16641 inyecciones.

7 PROCESADO Y ANÁLISIS DE LOS RESULTADOS DE INYECCIÓN DE FALLOS

"La verdadera ciencia enseña, sobre todo, a dudar y a ser ignorante".

- Miguel de Unamuno -

El principal objetivo de usar el mismo script, y la misma plataforma, tanto para el test de radiación como para el experimento de inyección de fallos, lo vamos a apreciar en esta sección. Una vez se ha terminado de ejecutar la campaña de inyección de fallos, el número de ficheros y el contenido de estos son exactamente iguales que los del test de radiación. Por tanto, podemos reutilizar todos los script presentados en la sección 5.

6.1. Reutilización de script para análisis de resultados

Para las campañas de inyección de fallos, es totalmente innecesario calcular el número de SEUs por segundo o el número de SEUs por lotes. Es sabido ya que en los experimentos de inyección de fallos nos aseguramos un total de 1 SEU por run, por tanto hemos introducido un total de 16641 SEUs.

Lo que sí puede ser interesante, son las posiciones donde han sido provocado los SEUs para así poder determinar zonas más o menos sensibles de la FPGA. Para ello, únicamente se ejecuta el script de la **Figura 5-4**. Al igual que antes, se obtuvieron una serie de errores sistemáticos. Por tanto, se volvió a ejecutar el script de la **Figura 5-5**. Una vez conocida las posiciones de los errores sistemáticos, se procedió a compararlos con los errores sistemáticos del test de radiación, y curiosamente, no se produjo ninguna coincidencia. Se ha dejado como posible trabajo para el futuro, intentar descubrir la procedencia de dichos errores sistemáticos.

El parámetro que ha sido comentado y que sí que nos interesa calcular, es el Architecture Vulnerabity Factor o AVF. Como hemos hecho antes, para hallar este parámetro basta únicamente con ejecutar el script de la **Figura 5-9**. Todos estos trabajos ha sido llevado a cabo de manera muy simple y rápida únicamente ejecutando los mismos scripts que para el test de radiación.

Por tanto, y una vez ha sido ejecutado dicho script, el AVF resultante fue de **6.41%**. Este resultado, queda muy lejos de los **2.72%** que se calculó del test de radiación, y evidentemente queda fuera de los márgenes de error calculados. Eso se debe a que es necesario hacer una serie de ajustes, que se detallaremos a continuación.

6.2. Ajustes sobre el AVF del experimento de inyección de fallos

Hay una serie de factores que necesitan ser considerados con el fin de conseguir un mejor ajuste en el cálculo

del AVF. Primeramente, hay que tener en cuenta la sensibilidad del diseño frente a errores, para así intentar ser lo más fiel posible a lo que ocurre en la realidad.

El primer factor a tener en cuenta reside en el número de bits que pueden llegar a sufrir un SEU y que no son otros que sobre los que inyecta nuestra plataforma de inyección de fallos. Como ya hemos comentado anteriormente, hay que considerar que el haz de radiación puede provocar SEU en todos los bits de la FPGA, esto es, en un total de **27016704**, de los cuales, y como ya ha sido mencionado, sólo **18936096** son esenciales. La plataforma de inyección de fallos FTU inyectará únicamente sobre estos bits esenciales, y considerando, que únicamente estos bits pueden llegar a provocar errores en la salida, será necesario hacer un ajuste para corregir el AVF, con el fin de tener un AVF lo más fiel posible al AVF del test de radiación. Por tanto, y añadiendo runs sin fallos que nos ayude a "eliminar" aquellos SEUs producidos sobre bits no esenciales, se obtiene un factor de escala de **18 936 096/26 400 000**. Hay que tener en cuenta que algunos de estos bits no esenciales pueden llegar a provocar un error permanente en la FPGA, que aunque no provoquen errores en la salida de manera directa, sí lo pueden provocar de manera indirecta. Pero se considerará que esto pasará con una probabilidad muy baja, y si por un casual llegara a ocurrir, al apagar/encender la plataforma cada hora, estos errores permanentes serán corregidos. Por tanto, se considerará que no contribuye de manera significante al cálculo del AVF.

El segundo factor a tener en cuenta tiene relación con la aleatoriedad del test de radiación. Como ya se ha comentado, puede haber runs con un fallo, con más de un fallo, o incluso **runs sin fallo**. Si tenemos en cuenta que en el experimento de inyección de fallos, todos los runs tienen un SEU, habría que calcular un segundo factor para ajustar el AVF. Como antes, puede haber runs sin errores en la salida, pero en este caso son provocados porque no se ha producido ningún SEU. Por tanto y para el cálculo de este segundo factor de ajuste, se ha tenido en cuenta un análisis del fichero damages.csv y de los readbacks por lotes, mediante el siguiente script:

```
import sys
ten_run_seus=[]
run_damages=[]
ten_run_counter=0
ten_run_damages=[]
ten_writer=0
global_counter=0
run_without_seus_counter=0
total_seu_lots_without_error=0

def recursive(index, global_counter):
        counter = 0
        global ten_run_counter
        global total_seu_lots_without_error
        for i in range(ten_run_counter, ten_run_counter+10):
                if run_damages[i] != ";":
                        counter += 1
        if counter == 0:
                total_seu_lots_without_error += int(ten_run_seus[global_counter])


        ten_run_damages.append(counter)
        ten_run_counter += 10
        global_counter += 1
        if ten_run_counter != 750:
                recursive(ten_run_counter, global_counter)
```

```
with open(sys.argv[2]) as f:
        for l in f:
                ten_run_seus.append(l.strip("\n"))
                if int(l.strip("\n")) < 10:
                        run_without_seus_counter+=(10 - int(l.strip("\n")))

with open(sys.argv[1]) as f:
        for l in f:
                run_damages.append(l.strip("\n"))

recursive(ten_run_counter, global_counter)

for j in range(0, len(ten_run_damages)):
        print "Los runs [" + str(ten_writer) + "-" + str(ten_writer+9) + "] tienen " +
str(ten_run_seus[j]) + " seus y " + str(ten_run_damages[j]) + " runs con errores"

        ten_writer += 10

print "The total of SEUs in lots without output errors is " + str(total_seu_lots_withou
t_error)
#print run_without_seus_counter
```

Figura 7-1. Script en python para cálculo del segundo factor de ajuste.

Este script fue creado primeramente, para llevar a cabo un análisis bastante completo por lotes, como se muestra a continuación:

```
Los runs [0-9] tienen 1 seus y 0 runs con errores
Los runs [10-19] tienen 8 seus y 0 runs con errores
Los runs [20-29] tienen 8 seus y 0 runs con errores
Los runs [30-39] tienen 12 seus y 0 runs con errores
Los runs [40-49] tienen 10 seus y 0 runs con errores
Los runs [50-59] tienen 6 seus y 0 runs con errores
Los runs [60-69] tienen 8 seus y 0 runs con errores
Los runs [70-79] tienen 5 seus y 0 runs con errores
Los runs [80-89] tienen 10 seus y 0 runs con errores
Los runs [90-99] tienen 8 seus y 0 runs con errores
Los runs [100-109] tienen 11 seus y 0 runs con errores
Los runs [110-119] tienen 12 seus y 0 runs con errores
Los runs [120-129] tienen 9 seus y 0 runs con errores
Los runs [130-139] tienen 8 seus y 0 runs con errores
Los runs [140-149] tienen 10 seus y 0 runs con errores
Los runs [150-159] tienen 10 seus y 0 runs con errores
Los runs [160-169] tienen 2 seus y 0 runs con errores
Los runs [170-179] tienen 10 seus y 0 runs con errores
Los runs [180-189] tienen 11 seus y 0 runs con errores
Los runs [190-199] tienen 8 seus y 0 runs con errores
Los runs [200-209] tienen 15 seus y 0 runs con errores
Los runs [210-219] tienen 9 seus y 0 runs con errores
Los runs [220-229] tienen 12 seus y 0 runs con errores
Los runs [230-239] tienen 12 seus y 0 runs con errores
Los runs [240-249] tienen 10 seus y 0 runs con errores
Los runs [250-259] tienen 8 seus y 0 runs con errores
Los runs [260-269] tienen 11 seus y 0 runs con errores
```

Figura 7-2. Análisis completo de los primeros lotes de la campaña 1.

En la figura 7-2 se puede observar como tenemos un análisis de los SEUs y de los runs con errores de cada lote de cada una de las campañas. Pero para obtener el valor que estamos buscando, basta con eminiar el comentario que viene en el recuadro rojo de la figura 7-1.

Por tanto, se ha podido demostrar que el número medio de runs en los que, al menos, se ha producido un SEU es **609**. Por lo tanto, y teniendo en cuenta que tenemos un total de 750 runs por experimento, el segundo factor

de escala que debe ser introducido es **609/750**. Cuanto más cerca este factor de 1, más próximo será al experimento de inyección de fallos. Hay que tener en cuenta también, que un valor alto de SEUs por run, puede provocar que la FPGA falle más de lo esperado debido al daño acumulado.

Considerando estos fallos, se puede predecir el AVF con la siguiente fórmula:

$$AVF_{predicted} = Sensitivity_{raw} * F_{coverage} * F_{noSEU} \qquad (7.1)$$

Donde $F_{coverage}$ es el factor de escala debido a los bits que no son esenciales, y que sí se tienen en cuenta en el test de radiación pero no en el experimento de inyección de fallos, y F_{noSEU} es el factor de escala procedente de los runs sin SEU provocados por el haz.

Por otro lado, y de manera general, podemos computar sendas fórmulas para los factores de escala, que se detallan a continuación:

$$F_{coverage} = \frac{Número\ bits\ esenciales\ de\ la\ FPGA}{Número\ total\ de\ bits\ de\ la\ FPGA} \qquad (7.2)$$

$$F_{noSEU} = \frac{Valor\ medio\ de\ runs\ \text{sin}\ SEU}{Número\ total\ de\ runs} \qquad (7.3)$$

Por lo tanto, y computando las fórmulas (7.2) y (7.3), obtenemos:

$$F_{coverage} = \frac{18\ 936\ 096}{26\ 400\ 000} = \mathbf{0.7173}$$

$$F_{noSEU} = \frac{609}{750} = \mathbf{0.812}$$

Finalmente, si se calcula el AVF con la fórmula (7.1), y a partir de los factores de escala computados anteriormente, se obtiene un valor final de **3.65%**.

8 CONTRASTE DEL EXPERIMENTO DE RADIACIÓN FRENTE A INYECCIÓN DE FALLOS

"La ciencia se compone de errores, que a su vez, son los pasos hacia la verdad".

- Julio Verne -

Uno de los objetivos de este trabajo, era el de contrastar los resultados del test de radiación frente los del experimento de inyección de fallos. Aunque ya se ha ido calculando algunos de los parámetros a lo largo de las secciones, en esta sección lo que se hará será poner todos los resultados en común, para así dar sentido a todo lo que hemos ido realizando. Se irá usando muchos de los parámetros calculados y muchas de las fórmulas usadas a lo largo de las secciones anteriores.

7.1. Contraste de los resultados

Como ha sido comentado, una medida de la vulnerabilidad frente a radiación de cualquier diseño es el AVF. Este parámetro ya ha sido calculado en secciones anteriores, tanto para el test de radiación como para el experimento de inyección de fallos. Para el test de radiación, se tenía un total de 20250 runs, que usando las tablas de Excel de la figura 6-1, se obtiene un margen de error de **0.9065%**, respecto de la campaña exhaustiva, con intervalo de confianza del 99%. En la tabla 8-1, podemos ver más detallado los resultados del AVF de ambos experimentos.

$runs_{FI}$	$runs_{rad}$	F_{noSEU}	AVF_{FI}	AVF_{rad}	e_{FI}	e_{rad}
16641	20250	0.812	3.65%	2.96%	1%	0.9065%

Tabla 8-1. Resultados de AVF de ambos experimentos más margen de error en tanto por ciento.

Como se puede observar, el estudio estadístico ha sido usado en ambas direcciones. Primeramente para hallar un determinado número de inyecciones dado un cierto margen de error (1%), y por otro lado, para determinar el margen de error dado un determinado número de runs.

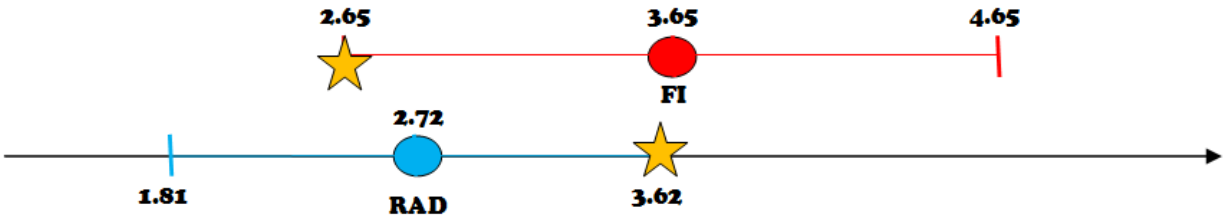


Figura 8-1. Gráfico de comparación de AVF de ambos experimentos

En la figura 8-1se puede ver de manera gráfica lo mostrado en la tabla 8-1. Lo único que ha sido puesto en la gráfica son los resultados obtenidos de AVF, con sus respectivos márgenes de error. Se puede comprobar que existe una zona de intersección de ambas rectas, que no es otra, que la zona comprendida entre las dos estrellas (entre 2.65% y 3.62%). Es necesario tener en cuenta este margen de error, ya que no se realizan campañas exhaustivas, y se tiene un número menor de inyecciones. En teoría si se llevase a cabo una campaña exhaustiva, debería de salir el mismo AVF. Aunque llevar a cabo una campaña exhaustiva, tanto en radiación, como en inyección de fallos, para un diseño de estas características, se vuelve algo absolutamente inviable.

7.2. Conclusiones finales sobre el contraste de los resultados

En este caso, el AVF obtenido es de, 3.65% para el experimento de inyección de fallos, y de 2.72% para el test de radiación. Si se volviese a realizar el mismo experimento, se obtendrían unos resultados comprendidos entre 2.65% y 4.65% para el experimento de inyección de fallos, y entre 1.81% y 3.62% para un test de radiación.

Como es evidente, y teniendo en cuenta que hay tramos donde coinciden, podría ocurrir el caso en el que incluso llegaran a coincidir los AVF de ambos experimentos.

La conclusión que se puede llegar a sacar de estos resultados es que, para cualquier caso general, el AVF "experimental" (obtenido a partir de un test de radiación), y el AVF predicho (obtenido a partir de un experimento de inyección de fallos), están dentro de unos determinados márgenes de error conocidos, lo que significa que esta plataforma de inyección de fallos tiene las capacidades suficientes para predecir la sensibilidad de un diseño frente a radiación, siempre teniendo en cuenta un cierto margen de error.

Estas conclusiones a las que se ha llegado tienen un gran valor. En primer lugar, se puede llegar a predecir el AVF de un diseño, sin la necesidad de llevar a cabo un test de radiación, esto es, podemos ser capaces de conocer la sensibilidad de cualquier diseño frente a radiación, sin la necesidad de llevar a cabo el propio experimento de radiación. En segundo lugar, y siguiendo la línea de lo ya mencionado, se ahorraría realizar el test de radiación. Esto lleva consigo un gran ahorro de dinero y sobre todo de tiempo. Como ejemplo, el tiempo para realizar el test de radiación, teniendo en cuenta traslado de placa a Los Álamos, preparación del set-up…, ha sido en torno a 3 meses, mientras que para el experimento de inyección bastó sólo con un día para obtener los resultados.

9 CONCLUSIONES Y TRABAJOS FUTUROS

"La ciencia de hoy es la tecnología del mañana".

- Edward Teller -

Por último y para finalizar este trabajo, se comentará las conclusiones a las que se ha podido llegar una vez obtenido y analizado los resultados. Además se comentarán posibles líneas de futuro que pueden ser de interés para mejorar este trabajo o para comenzar otros nuevos.

Como conclusiones podemos comentar que, el objetivo de este trabajo era el de comparar mediante la estadística, los datos de un determinado test de radiación con los de un experimento de un inyección de fallos. El objetivo era intentar demostrar que los datos obtenidos a partir de una plataforma de inyección de fallos no invasiva, son iguales, teniendo en cuenta un determinado margen de error, que otros datos de radiación, obtenidos a partir de unos centros especializados en crear ambientes hostiles, como puede ser el espacio.

Con el fin de obtener una serie de resultados que puedan llegar a ser comparados, se ha usado tanto para la plataforma de inyección de fallos, como para el test de radiación, el mismo hardware, diseño, test de vectores y software de control.

Se ha podido demostrar que, los resultados obtenidos están dentro del margen de error esperado. Incluso se ha podido mostrar el cálculo de dicho margen de error. Por tanto, hemos podido demostrar el alto poder predictivo que tiene la plataforma de inyección de fallos FTU2.

Esto, y como ya hemos comentado, es de gran interés, ya que refuerza la idea del uso de plataforma de inyección de fallos frente a test de radiación, que la usarían como predicción y herramienta de análisis prioritaria, debido a sus grandes ventajas.

Es obvio que, no hay nada que nos proporcione un mejor análisis de la vulnerabilidad de un diseño frente a radiación, que someter dicho diseño a un test de radiación y analizar los resultados. Pero esto supone un gran gasto tanto de dinero como de tiempo, en cambio, y como se ha podido demostrar, mediante una plataforma de inyección de fallos, puede llegar a obtener resultados bastante fiables, ahorrando una gran cantidad de tiempo y dinero.

Por tanto, se quería buscar un análisis de los resultados de ambos experimentos, y dicho análisis no ha podido ser más positivo.

Para finalizar, comentar que para este trabajo ha sido preparado un manuscrito con el fin de publicar los resultados en conferencias y revistas.

8.1. Trabajos Futuros

Por último, se detallará algunos de los trabajos futuros que están pendientes y que no han podido ser desarrollados todavía. Muchos de ellos son de gran interés, y se espera que puedan ser desarrollados en un futuro.

9.1.1 Faulty Outputs

Una de las principales vías abiertas, es la de las llamadas "fauly outputs". Consiste básicamente en pasar por un receptor las salidas erróneas del transmisor Zigbee. Como ya se ha comentado en la sección de trabajos previos, los datos se han obtenido a partir de radiar un transmisor Zigbee. Por tanto, al igual que se desarrolló un transmisor, también fue desarrollado un receptor.

Partimos por tanto de los ficheros damages.csv, que como ya dijimos, son máscaras que indican las diferencias entre las salidas correctas y las erróneas.

Se ha desarrollado una forma de obtener una secuencia de las salidas erróneas, aplicando las máscaras a las salidas correctas.

Por ejemplo, si la salida correcta en el ciclo "x" es:

A23FEA65

Y tenemos, que en el ciclo "x" la máscara es[10]:

00FF008F

Tendremos por tanto, una salida en el ciclo "x" de:

A2C0EAEA

El cálculo de estas salidas erróneas se ha llevado a cabo mediante un script que no se mostrará debido a no considerarse necesario.

Este trabajo llevó bastante tiempo, pero será de mucha utilidad en el futuro. Tal es así, que ha sido incorporado como mejora de la plataforma de inyección de fallos de FTU2.

Por tanto, y procesando los ficheros damages.csv de cada campaña, se pueden obtener las salidas erróneas provocadas por la radiación sobre el transmisor. Dichas salidas serán pasadas al receptor, y estudiaremos la respuesta de éste. Esto nos llevará al estudio de un caso real, en el que tenemos un transmisor que genera unas salidas erróneas provocadas por la radiación, y un receptor que es capaz de transformar dichas salidas erróneas en salidas correctas. De esta manera podremos hacer la comunicación más robusta, al conseguir diseñar un receptor que se capaz de procesar una serie de datos procedentes de un transmisor bajo los efectos de la radiación, lo que puede ser de mucho interés.

9.1.2 Inyección gaussiana

Otra de las posibles líneas de trabajo que están abiertas es la de la inyección gaussiana. Es una nueva herramienta que ha sido añadida a la plataforma de inyección de fallos FTU2, y que podría llegar a dar un análisis mucho más realista a las simulaciones de inyección de fallos.

Esta herramienta no ha sido usada para obtener los datos de inyección de fallos de este trabajo, ya que se trata de una herramienta que está todavía en estado de desarrollo.

Se trata de una herramienta, que es capaz de simular de una manera mucho más precisa un test de radiación. Es capaz de provocar más de un SEU por run, o incluso runs sin SEU. Únicamente habrá que calcular la media y varianza con que se producen los SEUs en los test de radiación, y este inyector gaussiano será capaz de reproducir una campaña que tenga dicha media y dicha varianza en la inyección de SEUs.

Es evidente que habrá que reconsiderar los factores de ajuste y las fórmulas, pero una vez desarrollada la herramienta por completo, se puede llegar a conseguir datos de inyección de fallos, bastante parecidos a datos

[10] Esta información ha sido obtenida del damages.csv

de radiación, lo que puede servir de mucha utilidad para el estudio de la vulnerabilidad de los diseños, además de dar otro punto a favor para dar validez a los resultados proporcionados por una plataforma de inyección de fallos.

9.1.3 Errores sistemáticos

Como ya hemos comentado en las secciones anteriores, se descubrieron una serie de errores sistemáticos a la hora de hacer los readbacks. Como también se ha mencionado, esos errores sistemáticos no coincidían en los resultados del test de radiación y en los de inyección de fallos.

La idea es estudiar la procedencia de dichos errores sistemáticos, ya que nos proporcionaría mucha información sobre la FPGA, y sobre sus "debilidades".

Esto puede ser ayudar para encontrar zonas más sensibles de la FPGA, y así poder realizar un estudio de la vulnerabilidad del diseño, además de proponer nuevas medidas de protección.

REFERENCIAS

[1] «https://es.wikipedia.org/wiki/Single_event_upset,» [En línea].

[2] «https://www.jedec.org/standards-documents/dictionary/terms/single-event-latch-sel,» [En línea].

[3] «https://www.jedec.org/standards-documents/dictionary/terms/single-event-gate-rupture-segr,» [En línea].

[4] H. M. Quinn, D. A. Black, W. H. Robinson and S. P. Buchner, "Fault Simulation and Emulation Tools to Augment Radiation-Hardness Assurance Testing," in IEEE Transactions on Nuclear Science, vol. 60, no. 3, pp.2119-2142,June2013.

[5] J. M. Mogollon, H. Guzmán-Miranda, J. Nápoles, J. Barrientes, M. A. Aguirre, "FTUNSHADES2: A novel platform for early evaluation of robustness against SEE," *2011 12th European Conference on Radiation and Its Effects on components and Systems*, Sevilla, 2011, pp. 169-174.

[6] H. Guzman-Miranda, J. Barrientos-Rojas, P. Lopez-Gonzalez, V. Baena-Lecuyer, M.A. Aguirre, "On the structural robustness assessment of wireless communication systems for intra-satellite applications", *IEEE Trans. Nuclear Science*, vol. 61, issue: 6, pp. 3244-3249, Dec. 2014.

[7] R. Leveugle, A. Calvez, P. Maistri and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," *2009 Design, Automation, & Test in Europe Conference & Exhibition*, Nice, 2009, pp. 502-506. doi: 10.1109/DATE.2009.5090716.

[8] S.A. Wender and P.W. Liswoski, "A white neutron source from 1 to 400 MeV", Nuclear Instruments and Methods in Physics Research B, vol. 24-25, pp 897-900, 1987.

[9] P.W. Liwoski and K.F. Schoenberg, "The Alamos Neutron Science Center", Nuclear Instruments and Methods in Physics Research A, vol. 562, pp. 910-914, 2006.