Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Técnica de diagnóstico de SEU utilizando diccionarios de fallos incompletos

Autor: Álvaro Calvo Matos

Tutor: Hipólito Guzmán Miranda

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

Sevilla, 2020

Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Técnica de diagnóstico de SEU utilizando diccionarios de fallos incompletos

Autor:
Álvaro Calvo Matos

Tutor:
Hipólito Guzmán Miranda
Profesor Titular

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

Sevilla, 2020

Trabajo Fin de Grado: Técnica de diagnóstico de SEU utilizando diccionarios de fallos incompletos

Autor: Álvaro Calvo Matos
Tutor: Hipólito Guzmán Miranda

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Orden recomendado: - Comienza con los agradecimientos más formales, que suelen ir dirigidos a patrocinadores y/o al tutor del proyecto.

- Jerarquiza en función de su influencia en partes relevantes del proyecto, de mayor a menor.

- No uses frases largas, aunque cuando nombres a personas cercanas puedes hacer uso de dedicatorias en el TFG; te dejamos algunos ejemplos de cómo hacerlo más adelante.

- Las dedicatorias en el TFG pueden ser palabras tuyas, propias, o comenzar con un verso, un proverbio, etc.

Algunos ejemplos de dedicatorias: - … y particularmente agradezco a mi maestro D/Dª ....., por inculcarme el amor por las matemáticas cuando sólo era un niño de 7 años.

- También deseo agradecer el apoyo y la amistad demostrada en todo momento por …..., incluso cuando le llamaba, temeroso de no lograr terminar esta tesis, a altas horas de la madrugada.

- Gracias a mi familia por su amor y apoyo incondicional desde mi nacimiento, que se mantiene siendo un adulto.

- Y deseo agradecer de manera especial al profesor/a ..... de la asignatura ..... porque sin su buen hacer en la docencia no habría sido capaz de acometer el apartado ..... con facilidad.

- La vida es hermosa, y una de las formas en que se manifiesta esta hermosura es en el hecho de poder compartir y disfrutar con quienes amamos, ..., y con quienes nos ayudan en nuestro camino, como han hecho ..... en mi formación académica.

Poli María Dani Luis

Eduardo Elena Fernando

Damián Compañeros de clase

Familia

A mis profesores del Colegio Salesiano de Utrera, ... en especial a mis dos últimos tutores, Dª Elena Ojeda ¿Rodríguez? y D Fernando ¿? ¿? , por la formación que me dieron, pero sobre todo por entenderme, soportarme y apoyarme. Y a D Eduardo Pérez Prados, de quien adquirí mis primeros conocimientos en informática, y quién posteriormente me informó de la existencia de las becas científicas de verano, gracias a las cuales descubrí mi vocación por la robótica, llevándome directamente hasta donde estoy hoy.

Álvaro Calvo Matos
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Sevilla, 2020

Resumen

El diagnóstico de *Conmutaciones por evento único o Single Event Upset (SEU)* es un problema abierto sobre el que apenas se han realizado investigaciones previas. En este trabajo perseguimos diseñar una nueva técnica de diagnóstico que permita localizar un SEU a partir de la información de la que se disponga.

Es común disponer únicamente de diccionarios de fallos incompletos, ya que, en circuitos grandes, el tiempo necesario para obtener un diccinario de fallos completo lo hace inviable. Vamos a ver qué técnica usamos para diagnosticar en estas situaciones y cuándo se comienza a perder la capacidad de diagnóstico conforme la exhaustividad del diccionario de reduce.

La hipótesis de la que partimos para diseñar las técnicas de diagnóstico es que los SEU próximos entre si producen patrones similares a la salida. Estos pueden ser caracterizados de diferentes formas y usados para estimar la localización real del SEU que queremos localizar.

Combinando la información que obtenemos al aplicar distintas métricas sobre la información disponible, hemos conseguido unos resultados bastante buenos sobre los diseños en los que se ha probado la técnica. Incluso para aquellos circuitos en los que no se consigue acertar el biestable y ciclo exactos, la técnica, tras la primera iteración, acota la localización del SEU en un relativamente reducido rango de ciclos y a unos registros concretos. A partir de esta primera acotación podemos obtener un nuevo diccionario de fallos enfocado en las zonas del circuito señaladas por el algoritmo de diagnóstico y repetir con él el proceso, mejorando el resultado. El diagnóstico puede darse por finalizado cuando encontremos un candidatos que produzca exactamente el mismo patrón de salida que el SEU bajo diagnóstico.

Con este proceso iterativo, si el diccionario de partida es lo suficientemente completo para realizar correctamente la primera estimación, llegará un momento en el que podamos obtener un diccionario completo de la zona acotada. Si llegados a este punto aún no ha terminado el diagnóstico y las iteraciones han seguido el camino correcto, el siguiente diccionario contendrá al menos una entrada cuyo patrón de salida coincida con el patrón que produce el SEU bajo diagnóstico.

Esta técnica puede ser muy útil en el proceso de diseño de circuitos resistentes a radiación, ya que, por ejemplo, ante cualquier vulnerabilidad encontrada tras irradiar el circuito en el acelerador de partículas, evita repetir el proceso de diseño completo. Aplicando la técnica se puede saber en qué biestable se ha producido el SEU y reforzar la zona en caso de que fuera necesario.

Abstract

The Single Event Upset (SEU) diagnosis is an open problem that has hardly been investigated previously. In this work we seek to design a new diagnostic technique that allows locating a SEU from the information that is available.

It is common to have only incomplete fault dictionaries, since, on large circuits, the time required to obtain a complete fault dictionary makes it unfeasible. We are going to see what technique we use to diagnose in these situations and when the diagnostic capacity begins to lose, according to the exhaustiveness of the dictionary.

The hypothesis from which we start to design diagnostic techniques is that the SEUs close to each other produce similar patterns at the output. These can be characterized in different ways and used to estimate the actual location of the SEU that we want to locate.

Combining the information we obtain by applying different metrics on the available information, we have achieved quite good results on the designs in which the technique has been tested. Even for those circuits in which it is not possible to hit the exact flip-flop and cycle, the technique, after the first iteration, limits the location of the SEU in a relatively reduced range of cycles and to specific registers. From this first dimension we can obtain a new fault dictionary focused on the areas of the circuit indicated by the diagnostic algorithm and repeat the process with it, improving the result. The diagnosis can be terminated when we find a candidate that produces the exact same output pattern as the SEU under diagnosis.

With this iterative process, if the starting dictionary is complete enough to make the first estimate correctly, there will come a time when we can obtain a complete dictionary of the bounded area. If at this point the diagnosis has not yet finished and the iterations have followed the correct path, the following dictionary will contain at least one entry whose output pattern matches the pattern that the diagnostic SEU produces.

This technique can be very useful in the process of designing radiation resistant circuits, since, for example, before any vulnerability found after irradiating the circuit in the particle accelerator, it avoids repeating the entire design process. By applying the technique, it is possible to know in which bistable the SEU has been produced and to reinforce the area if necessary.

... -translation by google-

Índice Abreviado

Índice

1 Introducción

La primera vez que se observaron los efectos de la radiación en satélites en órbita fue a mediados de la década de 1970. Desde entonces, los investigadores han estudiado sus efectos sobre diferentes circuitos y tecnologías. La radiación puede ser un problema para los circuitos destinados a trabajar en su presencia. Si esta es ionizante, puede dar lugar a un *Single Event Effect (SEE)*, o un efecto de evento único, provocando un error en el circuito. Los daños que provoca la radiación se clasifican en dos grandes grupos: *errores físicos ('hard errors')* y *errores lógicos ('soft errors')* [1]. Las *conmutaciones por evento único o Single Event Upset (SEU)* son errores lógicos inducidos por radiación en el circuito que consisten en el cambio de valor de un biestable del mismo. No son daños permanentes, pero si que pueden afectar al correcto funcionamiento del sistema.

Con la miniaturización de los circuitos, la dosis de radiación necesaria para provocar un SEU es cada vez menor, con la consiguiente aparición de sus efectos cada vez a menor altitud [2]. Esto acerca el problema de la radiación a aplicaciones más comunes como puede ser la aviación o las telecomunicaciones. A veces no importa, o es asumible, que un bit del circuito conmute a causa de radiación. Blindar un móvil frente a radiación o reforzar sus circuitos con técnicas como la *Redundancia Modular Triple o Triple Modular Redundancy (TMR)* [3] puede ser innecesario dado que el mayor riesgo al que nos exponemos es mínimo, pero esto no siempre es así. Cuando se trata de satélites, aviones, o incluso bases militares armadas, existen sistemas críticos cuyas misiones pueden ser el control orbital, la estabilización del vuelo o el lanzamiento de misiles, donde no son asumibles los errores que pueda provocar un SEU.

Diseñar circuitos resistentes a radiación puede ser un proceso costoso, complicado y lento. Además, aplicar técnicas de refuerzo contra radiación a circuitos completos puede ser una pérdida de recursos [4]. Uno de los pasos del diseño suele ser emplear una plataforma de inyección de fallos para estudiar qué zonas del circuito son críticas y cuáles no necesitan ser reforzadas [4]. Si todo ha ido bien, uno de los últimos pasos suele ser irradiar el circuito de forma real para verificar el diseño. Si se detectan irregularidades a la salida a causa de un SEU ocurrido durante la prueba, sería necesario rediseñar el circuito para reforzar aquellas zonas donde se hayan producido las conmutaciones. Este proceso se vería enormemente beneficiado de una técnica que permita localizar los SEU, es decir, calcular el ciclo de reloj y biestable en el que ha tenido lugar. Determinar la localización espacial y temporal de un SEU se denomina *SEU diagnosis* [5].

El problema al que nos enfrentamos al tratar de localizar un SEU a partir de la información de salida de un circuito crece exponencialmente con el tamaño del circuito. Las escasas técnicas de diagnóstico existentes hasta el momento requieren de diccionarios de fallos completos o exhaustivos, requisito que no siempre es posible cumplir.

En la presente investigación hemos desarrollado una nueva técnica de diagnóstico de SEU basada en diccionarios de fallos incompletos o no exhaustivos. La hipótesis de la que partimos es que:

Hipótesis 1.0.1 *"Dos SEU próximos entre sí provocarán patrones de error similares a la salida".*

La mayor parte de la investigación se ha centrado en obtener métricas para examinar los patrones de salida desde distintas aproximaciones, ya que el parecido o no de dos salidas depende mucho de cómo las observemos. En el desarrollo del presente documento analizaremos y compararemos las métricas desarrolladas. Veremos si alguna de ellas es mejor que otras, cómo podemos combinarlas en un único algoritmo que realice el diagnóstico, si este mejora o empeora al prescindir de alguna de las métricas fusionadas, y hasta qué punto podemos elevar la calidad del diagnóstico empleando esta técnica.

2 Estado del arte

2.1 Detección de fallos (*Fault Detection*)

Dado que no es posible realizar un diagnóstico de SEU sin detectarlo primero, numerosos estudios se centran en desarrollar técnicas que permitan detectarlos a tiempo para suprimir sus efectos. Por ejemplo, en 2014, un equipo chino presentó una técnica de detección de SEU basada en la *Máquina de Boltzman Restringida o Restricted Boltzmann Machine (RBM)*, bloque fundamental en muchos algoritmos de *Deep Learning* [6]. En [7] abordan el problema de *faul detection* por el modelo dinámico del sistema. Comparan las lecturas tomadas por los sensores con los valores teóricos que se obtienen del modelo dinámico del robot SCARA. De esta forma detectan anomalías debidas a radiación. En un estudio más reciente, enfocado a sistemas embebidos, emplean programas de detección por software. Multitud de hilos se ejecutan simultáneamente y se encargan de examinar el circuito con el objetivo de detectar alguna irregularidad causada por radiación [8].

2.2 Diagnóstico de fallos o localización de fallos

Hasta ahora, el diagnóstico de fallos ha sido poco estudiado, siendo los fallos de fabricación a los que más esfuerzos de investigación se les ha dedicado [9, 10, 11, 12, 13, 14, 15]. Estos no son el tipo de fallos que nos interesa diagnosticar en esta investigación, ya que no son causados por radiación, si no que se producen, como su nombre indica, en el momento de fabricación del circuito (*stuck-at-0, stuck-at-1*).

Las técnicas existentes para localización de fallos provocados por radiación se basan principalmente en el uso de diccionarios de fallos, aunque también se emplean vectores de prueba, listas de fallos, tabla de verdad de nodos (*"node truth table"*) y tabla de conexiones de pines (*pin connection table*) [16, 17, 18].

A excepción de contados estudios, la mayoría de los revisados modelan al circuito bajo prueba o *Circuit Under Test (CUT)* como una caja negra, es decir, el diseño del circuito no se conoce y solo las salidas pueden ser monitorizadas. Normalmente, el número de biestables del circuito es mucho mayor que el número de salidas, por lo que es necesario observar el circuito el suficiente tiempo como para detectar patrones que puedan ser asociados a la localización de un determinado SEU [5]. Estas huellas son registradas y almacenadas en un diccionario durante una fase previa al diagnóstico.

El diccionario de fallos se genera mediante inyección de fallos, en alguna plataforma que lo permita [19, 20, 21], y contiene información de la localización de los SEU inyectados y el patrón de salidas que produce. Si el diccionario recoge todas las posibilidades, se habla de diccionario completo o exhaustivo, tomando el nombre de la campaña de inyección de fallos necesaria para generarlo (*Campaña Exhaustiva*). En el caso contrario, es un diccionario incompleto o no exhaustivo,

es decir, no todas las posibles combinaciones de (biestable, ciclo) han sido inyectadas y almacenadas en el diccionario.

Durante la fase de diagnóstico, para localizar un SEU detectado, se compara el patrón de error que genera en las salidas del circuito con los patrones almacenados en el diccionario. Debido al largo tiempo de observación comentado, la información a comparar puede tener un tamaño considerable, y por tanto el tiempo necesario para procesar la comparación es alto. Una solución para reducir esta cantidad de información y por tanto, el tiempo, permitiendo incluso localización de SEU en tiempo real, es comprimirla. Un ejemplo sería el uso de códigos HASH [5].

Dada la gran cantidad de biestables existentes en comparación con el reducido número de salidas, no es dificil imaginar la posibilidad de que dos SEU localizados en biestables y/o ciclos distintos produzcan exactamente el mismo patrón de error a la salida, al menos durante el tiempo y test programados. Cuando esto ocurre, se habla de *"Colisión"*. Además, es posible que un SEU no produzca error alguno a la salida durante el test, siendo indistinguible de una situación libre de conmutaciones. Ante estas situaciones, existirá más de una entada del diccionario que conincida con la buscada. Como resultado del diagnóstico se obtienen no una si no una lista de posibles localizaciones para el SEU bajo diagnóstico.

Hasta ahora hemos hablado de diagnóstico empleando diccionarios de fallos completos, pero si el CUT es grande, obtener un diccionario exhaustivo es una operación inviable, ya que la cantidad de combinaciones biestable-ciclo a inyectar para ello se vuelve inabarcable. Si se intenta diagnosticar un SEU empleando un diccionario de fallos incompleto, aparecen nuevos problemas, ya que puede ocurrir que la ubicación correcta no se haya inyectado durante la prueba, y por tanto no se encuentre en el diccionario. Si además existe una colisión que sí se ha inyectado, el diagnóstico conlcuirá con una localización única aparentemente correcta que puede no se acerque nada a la real.

3 Inyección de fallos

La inyección de fallos es una técnica que permite recrear los efectos que produce la radiación sobre un circuito. Esta técnica es ampliamente usada ya que permite estudiar en qué parte de un circuito causa más efecto un error lógico y qué partes son más resistentes a este tipo de error. El diseño de circuitos destinados a trabajar en entornos hostiles, donde recibirán altas dosis de radiación ionizante, encuentra en esta técnica una gran ayuda, ya que permite estudiar la sensibilidad de sus módulos a este tipo de errores sin necesidad de fabricarlo y testearlo bajo radiación real. Con esto se acelera enormemente el proceso de diseño y refuerzo de circuitos resistentes a radiación.

En nuestro caso, hemos empleado técnicas de inyección de fallos como ayuda para el diagnóstico de SEU, es decir, tratar de localizar el error, determinar en qué biestable (*flip-flop (FF)*) se ha producido la conmutación lógica y durante qué ciclo de reloj ha ocurrido. El papel de la técnica en el diagnóstico es el de recrear los efectos que tendrían SEU concretos sobre el circuito, de forma que podamos generar una base de datos donde tener relacionados cada localización espacial y temporal de un error lógico con su consecuencia a la salida del circuito. Esta base de datos, en diagnóstico de SEU, toma el nombre de diccionario de fallos, y se obtienen mediante campañas de inyección de fallos.

Durante una campaña de inyección de fallos se ejecuta el CUT con las mismas señales de entrada una y otra vez, partiendo siempre desde el reset. En cada ejecución, se inyecta un fallo en alguno de las posibles localizaciones (FF, ciclo). Si el CUT está formado por *"n"* FF y las pruebas se ejecutan durante *"m"* ciclos de reloj, existirán *"n·m"* posibles combinaciones donde inyectar un error lógico.

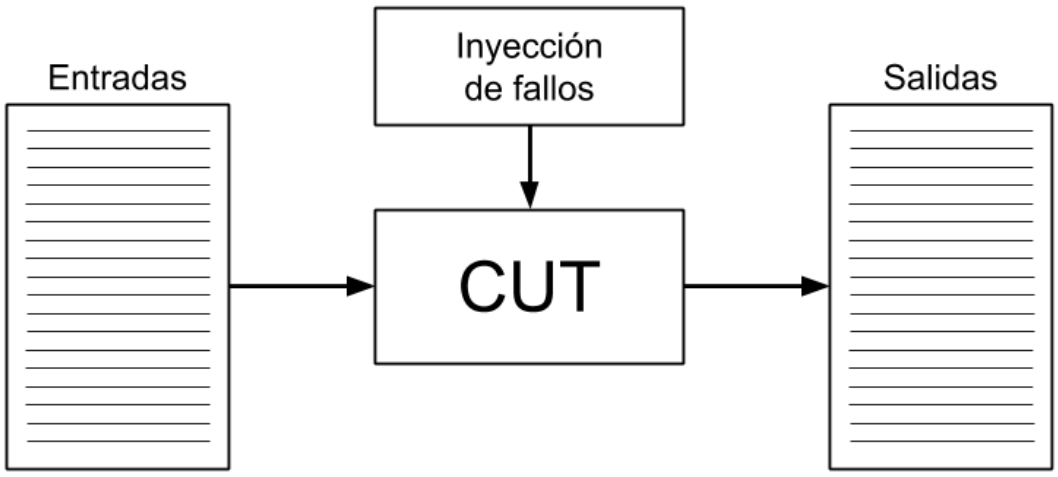

Figura 3.1 Ejecución de una campaña de inyección de fallos.

Las salidas que se obtienen, ciclo a ciclo, durante una ejecución de la campaña, se registran y almacenan junto con la información de la inyección que las ha causado. Dado que solo nos interesan los errores, no que se traten de ceros cuando deberían ser unos o viceversa, la salida del circuito (entendiendo "salida" como el conjunto de las salidas de los ciclos que dura la prueba) es comparada bit a bit con la salida que se obtendría si no existiese fallo alguno. Para llevar a cabo esta comparación se emplea la operación lógica *XOR*, ya que por definición, toma el valor 1 únicamente cuándo sus entradas son distintas.

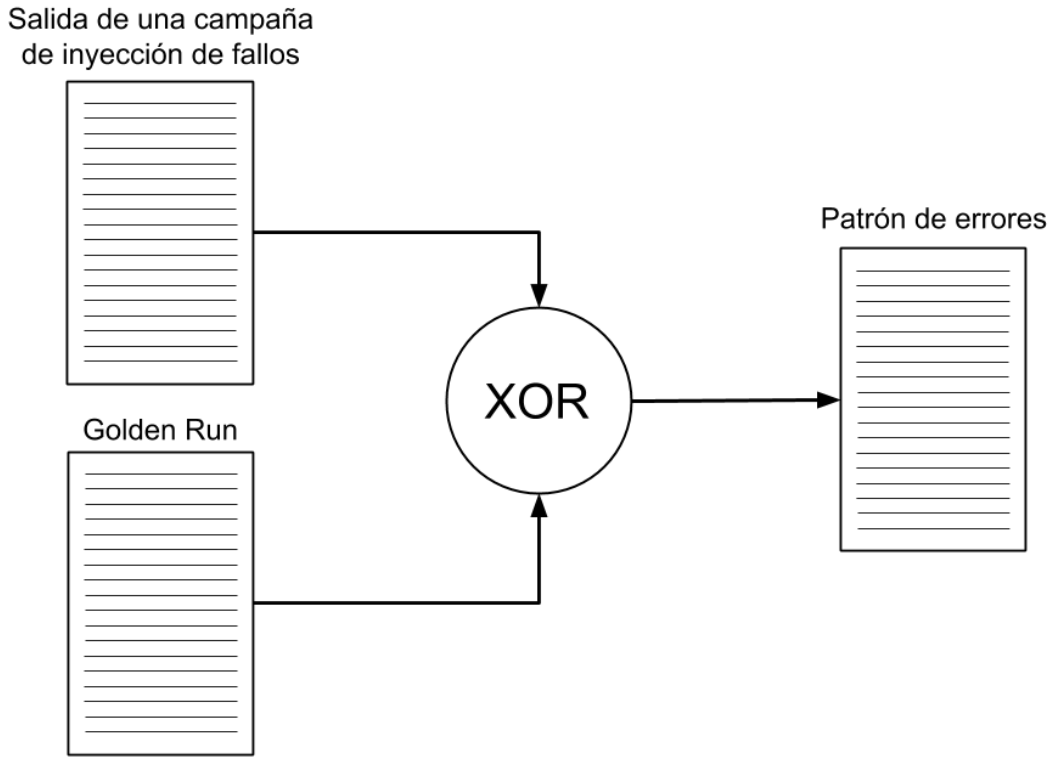

Figura 3.2 Postprocesado de la salida de una ejecución de la campaña.

Aplicandola para cada correspondiente pareja de bits entre una salida de la campaña y la salida del circuito sin inyecctar, conseguimos resaltar los errores que causa el SEU inyectado. A esta versión de la salida de una ejecución de la campaña donde solo toman valor lógico alto los errores causados por la inyección nos referiremos de ahora en adelante como *"Run"*. Así mismo, a la salida del circuito durante una ejecución libre de inyecciones la denominaremos *"Golden Run"*.

Cada run, junto con la información de la inyección que lo ha causado, constituye una entrada del diccionario. Es decir, cada entrada del diccionario está formado por la localización del SEU recreado (FF, ciclo) y las discrepancias que ha causado a la salida. El diccionario estará constituido por tantas entradas como inyecciones se hayan realizado durante la campaña.

Se dice que el diccionario de fallos es completo o exhaustivo cuando se genera a partir de una campaña de inyección de fallos exhaustiva, es decir, se inyectan todas las posibles combinaciones de FF y ciclo. Como hemos visto, existen *"n·m"* inyecciones posibles. Además, cada una de las ejecuciones tiene una extensión de *"m"* ciclos, lo que hace un total de *"n·m·m"* ciclos totales de ejecución necesarios para completar una campaña exhaustiva.

La extensión de cada una de las ejecuciones de una campaña, en ciclos, debe ser tal que el error lógico inyectado tenga el tiempo suficiente para producir un patrón de salidas identificable y diferenciable del resto de inyecciones. Si el circuito es pequeño, estas diferencias se harán notables de forma más temprana. Cuando el circuito tiene un número de FF elevado, el número de ciclos

necesario para que los patrones de salida se diferencien lo suficiente unos de otros, y por tanto, podamos identificar distintos SEU, también lo es. Esto hace que ejecutar una campaña de inyección de fallos exhaustiva, y por tanto, obtener un diccionario de fallos exhaustivo sea inviable para circuitos grandes.

Cuando esto sucede, no queda más opción que la de trabajar con diccionarios de fallos incompletos. Las campañas de inyección de fallos no exhaustivas pueden enfocarse a un subconjunto de biestables del circuito, limitar las inyecciones en tiempo, o realizarse de forma aleatoria. El diccionario de partida con el que hemos diagnosticado en circuitos grandes es incompleto y aleatorio, aunque plantearemos también el uso de diccionarios específicos para subconjuntos de FF y ciclos en otras fases del diagnóstico cuándo la primera no es suficiente.

Es importante destacar que el estudio aquí realizado requiere de una plataforma de inyección de fallos que funcione correctamente. La verificación del correcto funcionamiento de esta plataforma no es ámbito de esta investigación.

3.1 FT-Unshades2

Esta es la plataforma de inyección de fallos con la que hemos trabajado durante toda la investigación. Fué diseñada por un equipo del departamento de Ingeniería Electrónica de la Universidad de Sevilla en el año 2011 [21] y ha sido utilizada por multitud de equipos para el desarrollo de sus proyectos, incluidos algunos pertenecientes a la Agencia Espacial Europea (*ESA*).

Está basada en las FPGA (*Field Programmable Gate Array*) avanzadas de *Xilinx*. Concretamente, hace uso de una de sus características, llamada *"Capture and ReadBack"*, mediante la cual tienen acceso a partes del esquema de configuración. De esta forma pueden realizar cambios de valor en cualquier registro de usuario.

Partiendo de esta funcionalidad de las tarjetas de *Xilinx* consiguen controlar todo el proceso de inyección. Manejan la señal de reloj precisamente para poder inyectar en el ciclo y biestable deseado de forma que el CUT no sea capaz de percibir el proceso de inyección.

"FTU2 puede ser una de las plataformas de inyección de fallos basada en hardware más poderosas que existe para la evaluación de SEE. La flexibilidad y recursos disponibles hacen a FTU2 apropiada para diferentes usos, no solo como emulador de hardware para técnicas de inyección de fallos".

- J. M. Mogollon, H. Guzmán-Miranda, J. Nápoles, J. Barrientos, M. A. Aguirre, 2011, pág. 5 [21]

La plataforma FTU2 tiene potencia y flexibilidad suficiente para llevar a cabo todos las labores de inyección de fallos que se han necesitado para este trabajo.

4 Primera aproximación a una métrica apropiada. Distancia de Levenshtein

La distancia de Levenshtein recibe su nombre del científico ruso Vladimir Levenshtein, quién la creó en 1965.

"La distancia de Levenshtein, distancia de edición o distancia entre palabras es el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra".

- Wikipedia, 2020, párrafo 1, [22]

Aunque este algoritmo esté concebido como métrica de la diferencia entre dos cadenas de caracteres, podemos aplicar el concepto básico a dos salidas de la campaña de inyección de fallos. Redefiniendo la distancia de Levenshtein para nuestro caso en particular:

"La distancia de Levenshtein es el número mínimo de bits que hay que conmutar de la salida de una campaña de inyección de fallos para transformarla en otra".

Existe una operación lógica ya mencionada anteriormente que permite comparar dos salidas obteniendo como resultado ceros para aquellos bits en los que son iguales y unos en los diferentes. La operación *XOR* bit a bit permite obtener tantos unos como diferencias existen entre las dos salidas. La distancia de levenshtein entre dos salidas de una campaña es el sumatorio de todos estos bits con valor lógico alto.

De esta forma, según la hipótesis inicial 1.0.1, dos SEU que estén próximos entre sí tendrán una distancia de Levenshtein relativamente baja entre ellos. Cuando el diccionario de fallos contiene el error lógico a diagnosticar, existirá al menos una entrada con la cual la distancia de Levenshtein será cero. Cuando esto sucede, damos al diagnóstico por terminado, aunque en el capítulo 7 veremos una ampliación de la técnica que permite continuar un poco más y aumentar la confianza en los resultados obtenidos. En caso contrario, las entradas a menor distancia del run a diagnosticar (de aquí en adelante *"target run"*) serán utilizadas para localizarlo.

4.1 Elaboración de la base de datos de distancias

Antes de explicar más concretamente cómo se calculan las distancias entre runs del diccionario y se elabora la tabla de distancias, es necesario comentar cómo está expresada inicialmente la información de las salidas una vez son comparadas con el *Golden run* (ver figura 3.2). Un detalle hasta ahora no mencionado es que el diccionario se divide en dos archivos, *"damages.csv"* e *"injections.csv"*. Cada run divide su información entre estos dos ficheros, ubicando ciclo y FF de la inyección en *injections.csv* y los errores que esta genera a la salida del circuito, en *"damages.csv"*. Para explicar

como está dispuesta la información dentro de este último documento vamos a suponer que el CUT tiene 8 salidas y cada ejecución de la campaña dura 10 ciclos de reloj. Un ejemplo de salida contenida en el diccionario de este circuito puede ser:

0:1A,3:C0,4:80,7:1E,8:1C,9:02

Cada pareja de números separados por dos puntos significan *"Ciclo:Fallos"*, donde *Fallos* es una representación hexadecimal de las salidas del CUT en el ciclo *Ciclo*. Los fallos de distintos ciclos están separados entre sí mediante comas, y cada run contenido en el diccionario se encuentra en una línea independiente. Podemos observar en el ejemplo anterior que ciertos ciclos no aparecen. Estos son los ciclos de la ejecución donde la salida no presenta discrepancias con el *Golden run*. De esta forma, las inyecciones que no generen fallos a la salida del circuito se corresponderán con líneas vacías del diccionario.

Ahora que conocemos cómo está contenida la información en el diccionario de fallos, podemos proceder al cálculo de las distancias. El primer paso es leer la información de ambos archivos y cargarlos en memoria, rellenando los ciclos sin fallos con ceros. Hemos decidido almacenar la información en forma de enteros en base 10, aunque estos sean tratados como binario. A contiuación se realiza, bit a bit, la *XOR* de cada run con el resto. Como último paso en el cálculo de las distancias de Levenshtein, se toman los resultados de la operación lógica y se suman los bits con valor lógico alto. El resultado de esta suma es la distancia en cuestión.

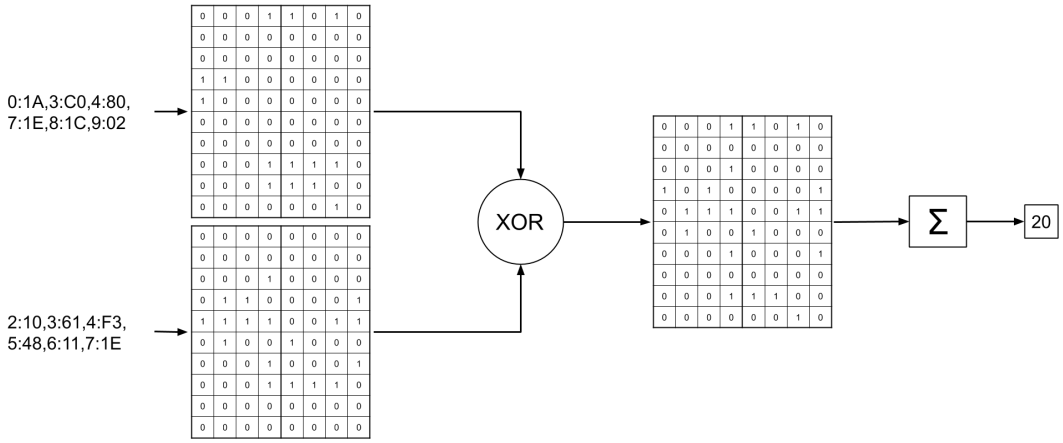

Figura 4.1 Cálculo de la distancia de Levenshtein.

Podemos elaborar una lista con las distancias entre el *target run* y todas las entradas del diccionario de forma que la componente *"i"* sea la distancia de Levenshtein entre el SEU a diagnosticar y la entrada *"i"*. Acompañando a esta lista tendríamos otra en la que se encuentra la información de las inyecciones.

Por último, notar que al tratarse de distancias, se cumplen las propiedades típicas de una distancia:

$$D(i,i) == 0$$
$$D(i,j) == D(j,i)$$

4.2 Diagnóstico basado en la distancia de Levenshtein

Una vez hayamos calculado las distancias de Levenshtein entre el *target run* y todas las entradas del dicionario y hayamos elaborado las dos listas mencionadas tendremos todo lo necesario para llevar a cabo el diagnóstico con esta primera versión de la técnica.

Tal y como hemos explicado en el último párrafo del apartado 4, las entradas del diccionario cuyas inyecciones se encuentren más próximas a la que habría probocado al *target run* presentarán las menores distancias hasta el mismo. Diagnósticar consiste en seleccionar estas entradas, las cuales pasarán a llamarse *"Candidatos"*.

Originalmente, el proceso que seguíamos para seleccionar candidatos consistía en establecer una distancia de corte en función de las distancias máxima y mínima. El corte se establecía según el porcentaje especificado del rango sobre el mínimo. De esta forma, eran seleccionados todos runs cuya distancia era menor o igual que el valor establecido:

$$D(i) \leq tolerancia \times \frac{(max - min)}{100} + min \tag{4.1}$$

Donde tolerancia establecía qué porcentaje del rango de distancias se toma. De esta forma, todos aquellos runs cuyas distancias cumplían la ecuación 4.1 pasaban a ser candidatos.

El problema de este sistema es que no controlabamos cuántos candidatos seleccionabamos en cada ocasión. Para valores de tolerancia de 1 a 5, se obtenían cantidades de candidatos bastante dispares en los diseños de circuitos en que se probó. Los causantes son valores de distancias aislados y alejados del resto, es decir, un mínimo o un máximo aislado mucho menor o mayor que el resto respectivamente. Esto proboca que el rango de distancias se alargue y se concentren todas muy próximas al extremo, seleccionando bien muy pocos o bien demasiados candidatos respectivamente.

Este problema puede solucionarse si se identifican y descartan estas distancias antes de calcular el valor de corte, pero si analizamos bien, veremos que hay una solución mejor: ordenar la lista de distancias de menor a mayor y seleccionar las entradas correspondientes a las *"n"* primeras distancias, siendo *"n"* el número de candidatos que deseemos seleccionar, especificado como argumento de entrada.

4.3 Resultados experimentales

Aunque se realizaron bastantes experimentos hasta llegar a la solución última de seleccionar los *"n"* primeros runs, hablaremos solo de aquellos resultados correspondientes a la versión última de este algoritmo.

Para probar la técnica disponíamos de una serie de circuitos con sus respectivos diccionarios de fallos. Algunos de estos eran exhaustivos, mientras que de otros, debido al tamaño del circuito, solo diaponíamos de uno incompleto fruto de una campaña de inyección de fallos aleatoria. Concretamente disponíamos de un diccionario del 0'005 % de exhaustividad para el diseño de la *FFT*, 0'87 % para la *UART* y 37'78 % para el *FIR_RI*.

En total, hemos probado el algoritmo en 10 diseños distintos:

- Sumador (*adder_acum*): suma acumuladamente los 8 bits de entrada en un registro de 20 bits.
- Contador (*counter*): contador de 8 bits.
- Doble contador (*dual_counter*): dos contadores de 8 bits conectados formando uno de 16 bits.
- FIFO (*fifo*): memoria de 32 bits del tipo "Primero que entra, Primero que sale" (First in, First out (FIFO)) [23].
- Filtro de respuesta de impulso finito (*FIR_RI*) [24]
- PCM (*pcm*): implementa una interfaz I²C para el códec PCM3168 [25]
- Registro de desplazamiento (*shiftreg*): registro de desplazamiento de 8 bits.

- Maquina de estados finita (*simple_fsm*): maquina de estados (Finite State Machine (FSM)) con 4 estados.
- UART (*uart*): Transmisor y receptor asíncrono universal (Universal Asynchronous Receiver and Transmitter (UART)), dispositivo para comunicaciones serie [26]

4.3.1 Diccionarios exhaustivos

Con el uso de diccionarios completos para el diagnóstico se obtiene siempre al menos un candidato con el que los patrones de error a la salida coinciden al completo, ya que todas las combinaciones posibles de *(FF, ciclo)* han sido inyectadas, y por tanto, el target run también.

Como el algoritmo de selección obtiene *"n"* candidatos, también es posible que obtengamos candidatos a distancia mayor que cero. Como el algoritmo nos muestra a la salida la distancia de Levenshtein que tiene cada candidato al target run, podemos descartar manualmente los candidatos que no esten a distancia cero cuando existan otros que si lo estén como es el caso.

En algunos circuitos obtenemos de hecho varios candidatos a distancia nula. Esto es debido a colisiones, y es indistinguible de cuál de ellas se trata el target run realmente.

4.3.2 Diccionarios no exhaustivos

Para simular diccionarios no exhaustivos en todos los circuitos, realizamos un pequeño script que eliminaba aleatoriamente entradas del diccionario con una probabilidad concreta en función del porcentaje de exhaustividad que deseasemos. La mayor parte de los experimentos se ha realizado con diccionarios del 5 % de exhaustividad, siendo inferior en aquellos diccionarios que ya lo eran.

Para diccionarios originalmente exhaustivos, seleccionamos aleatoriamente 100 entradas del diccionario original a modo de objetivos a diagnosticar. Como el nuevo diccionario reducido es también aleatorio, podían estar contenidos o no en el diccionario. Por el contrario, en diccionarios ya muy incompletos, el procedimiento que seguimos fue extraer 100 targets del diccionario original, obteniendo uno nuevo con 100 entradas menos. Para estos circuitos, el target run no se encontraba en el diccionario.

Tabla 4.1 Resultados experimentales. Distancia de Levenshtein. Dic. incompletos ($\leq 5\,\%$).

Diseños	Registro	FF
adder_acum	100	98
counter	100	96
dual_counter	99	99
fifo	98	1
fir_ri (37'78 %)	29	3
pcm	82	69
shiftreg	100	82
simple_fsm	100	88
uart (0'87 %)	97	93

En la tabla 4.1 se muestran los resultados de las 100 simulaciones para cada circuito. La columna *Registro* indica cuántas de las 100 veces se encontraba el registro correcto entre los *"n"* candidatos seleccionados, en este caso 5 candidatos. Análogamente, la columna *FF* es cuántas de las 100 veces se encontraba el FF correcto entre los candidatos. Cuando entre los 5 candidatos de una ejecución estaban más de una vez, se contaba como uno solo para el total de las 100 ejecuciones. La tabla es

una medida de la efectividad de esta distancia como método de diagnóstico aislado (aunque no se ha tenido en cuenta el ciclo de inyección para realizar el recuento de aciertos).

Cabe mencionar que *FF* es el biestable de inyección y *Registro* el resto de la dirección de inyección en caso de existir. En el caso del *simple_fsm*, no existen registros, y tanto *adder_acum* como *counter* y *shiftreg* tienen un único registro. El 100 % de aciertos en estos tres casos no es significativo.

Observando los candidatos que seleccionaba el algoritmo y comparando sus distancias de Levenshtein y la información de sus inyecciones con la inyección del target run, la cual conocíamos (por supuesto el algoritmo no tenía acceso a ella) se observaba relación directa entre la diferencia de los ciclos de las inyecciones y la distancia de Levenshtein. Mostrando un número mayor de candidatos pudimos comprobar como efectivamente, cuando la distancia de Levenshtein aumentaba o disminuía, también lo hacía la diferencia entre los ciclos de inyección. Tras esta observación decidimos implementar una nueva métrica con el objetivo de mejorar los resultados del diagnóstico.

5 Inclusión de la distancia temporal en el algoritmo de selección de candidatos

5.1 Diagnóstico basado en la distancia temporal

5.2 Fusión de las distancias temporal y de Levenshtein

5.3 Resultados experimentales

5.3.1 Diccionarios exhaustivos

5.3.2 Diccionarios no exhaustivos

6 Técnicas de diagnóstico auxiliares

6.1 Diagnóstico basado en el análisis de imágenes

6.2 Diagnóstico por coincidencias

6.3 Resultados experimentales

7 Campañas iterativas a partir de los candidatos seleccionados

7.1 Estudio preliminar sobre el porcentaje de acierto de los algotirmos

7.2 Obtención de la lista de candidatos

7.3 Extracción de la información para la siguiente campaña de inyección de fallos

7.4 Resultados experimentales

8 Distancia en flip-flops. Mejora de la distancia temporal

8.1 Inclusión de la distancia en flip-flops en el algoritmo

8.2 Resultados experimentales

8.2.1 Diccionarios exhaustivos

8.2.2 Diccionarios no exhaustivos

9 Conclusiones y trabajos futuros

9.1 Conclusiones

9.2 Trabajos futuros

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] H. G. Miranda, "Aportaciones a las técnicas de emulación y protección de sistemas microelectrónicos complejos bajo efectos de la radiación," Ph.D. dissertation, Universidad de Sevilla, May 2010.

[2] M. Santarini, "Cosmic radiation comes to asic and soc design," May 2005. [Online]. Available: https://www.edn.com/cosmic-radiation-comes-to-asic-and-soc-design/

[3] C. Carmichael, "Triple module redundancy design techniques for virtex fpgas," *Xilinx Application Note XAPP197*, vol. 1, 2001.

[4] M. G. Valderas, M. P. García, C. López, and L. Entrena, "Extensive seu impact analysis of a pic microprocessor for selective hardening," in *2009 European Conference on Radiation and Its Effects on Components and Systems*, 2009, pp. 333–336.

[5] J. M. Mogollón, J. Nápoles, H. Guzmán-Miranda, and M. A. Aguirre, "Real time seu detection and diagnosis for safety or mission-critical ics using hash library-based fault dictionaries," in *2011 12th European Conference on Radiation and Its Effects on Components and Systems*, 2011, pp. 705–710.

[6] S. Jian, J. Jiang, K. Lu, and Y. Zhang, "Seu-tolerant restricted boltzmann machine learning on dsp-based fault detection," in *2014 12th International Conference on Signal Processing (ICSP)*, 2014, pp. 1503–1506.

[7] W. Tao and W. Xingsong, "Fault diagnosis of a scara robot," in *2008 15th International Conference on Mechatronics and Machine Vision in Practice*, 2008, pp. 352–356.

[8] R. Pettit and A. Pettit, "Detecting single event upsets in embedded software," in *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, 2018, pp. 142–145.

[9] B. K. Sikdar, N. Ganguly, and P. P. Chaudhuri, "Fault diagnosis of vlsi circuits with cellular automata based pattern classifier," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 7, pp. 1115–1131, 2005.

[10] S. S. Yau and Yu-Shan Tang, "An efficient algorithm for generating complete test sets for combinational logic circuits," *IEEE Transactions on Computers*, vol. C-20, no. 11, pp. 1245–1251, 1971.

[11] S. S. Yau and M. Orsic, "Fault diagnosis and repair of cutpoint cellular arrays," *IEEE Transactions on Computers*, vol. C-19, no. 3, pp. 259–262, 1970.

[12] V. Amar and N. Condulmari, "Diagnosis of large combinational networks," *IEEE Transactions on Electronic Computers*, vol. EC-16, no. 5, pp. 675–680, 1967.

[13] D. R. Schertz and G. Metze, "A new representation for faults in combinational digital circuits," *IEEE Transactions on Computers*, vol. C-21, no. 8, pp. 858–866, 1972.

[14] J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits," *IEEE Transactions on Electronic Computers*, vol. EC-16, no. 5, pp. 567–580, 1967.

[15] A. D. Friedman, "Fault detection in redundant circuits," *IEEE Transactions on Electronic Computers*, vol. EC-16, no. 1, pp. 99–100, 1967.

[16] Su Wei, Fan Tongshun, and Du Mingfang, "Research for digital circuit fault testing and diagnosis techniques," in *2009 International Conference on Test and Measurement*, vol. 1, 2009, pp. 330–333.

[17] S. Wei, Z. Shide, and X. Lijun, "Research on digital circuit fault location procedure based on lasar," in *2008 ISECS International Colloquium on Computing, Communication, Control, and Management*, vol. 2, 2008, pp. 322–326.

[18] N. Naber, T. Getz, Y. Kim, and J. Petrosky, "Real-time fault detection and diagnostics using fpga-based architectures," in *2010 International Conference on Field Programmable Logic and Applications*, 2010, pp. 346–351.

[19] R. Zhang, L. Xiao, J. Li, X. Cao, C. Qi, and M. Wang, "A fast fault injection platform of multiple seus for sram-based fpgas," in *2017 Prognostics and System Health Management Conference (PHM-Harbin)*, 2017, pp. 1–5.

[20] A. da Silva and S. Sanchez, "Leon3 vip: A virtual platform with fault injection capabilities," in *2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, 2010, pp. 813–816.

[21] J. M. Mogollon, H. Guzmán-Miranda, J. Nápoles, J. Barrientos, and M. A. Aguirre, "Ftunshades2: A novel platform for early evaluation of robustness against see," in *2011 12th European Conference on Radiation and Its Effects on Components and Systems*, 2011, pp. 169–174.

[22] Wikipedia, "Distancia de levenshtein — wikipedia, la enciclopedia libre," 2020, [Internet; descargado 15-junio-2020]. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Distancia_de_Levenshtein&oldid=125248609

[23] "Vhdl standard fifo," Available online: http://www.deathbylogic.com/2013/07/vhdl-standard-fifo/, accessed on 17 June 2020.

[24] "Fpga4student. a low pass fir filter for ecg denoising in vhdl," Available online: https://www.fpga4student.com/2017/01/a-low-pass-fir-filter-in-vhdl.html, accessed on 17 June 2020.

[25] "I^2s interface designed for the pcm3168 audio interface from texas instruments," Available online: https://github.com/wklimann/PCM3168, accessed on 17 June 2020.

[26] "Simple uart controller for fpga written in vhdl," Available online: https://github.com/jakubcabal/uart-for-fpga, acceded on 17 June 2020.

[27] Zhou Jing, Liu Zengrong, Chen Lei, Wang Shuo, Wen Zhiping, Chen Xun, and Qi Chang, "An accurate fault location method based on configuration bitstream analysis," in *NORCHIP 2012*, 2012, pp. 1–5.

[28] M. Muñoz-Quijada, S. Sanchez-Barea, D. Vela-Calderon, and H. Guzman-Miranda, "Fine-grain circuit hardening through vhdl datatype substitution," *Electronics*, vol. 8, no. 1, p. 24, 2019.

[29] "Vhdl implementation of fft algorithm(s)," Available online: https://github.com/thasti/fft, accessed on 17 June 2020.

Glosario

CUT Circuit Under Test. 3–5, 7, 10

ESA European Space Agency. 7

FF flip-flop. 5–7, 9, 12, 13

FIFO First in, First out. 11

FPGA Field Programmable Gate Array. 7

FSM Finite State Machine. 12

RBM Restricted Boltzmann Machine. 3

SEE Single Event Effect. 1, 7

SEU Single Event Upset. III, 1, 3–7, 9, 10

TMR Triple Modular Redundancy. 1

UART Universal Asynchronous Receiver and Transmitter. 12