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

A mi tutor, Hipólito, por guiarme en este proyecto, pero sobre todo, por darme la oportunidad de formar parte de su equipo desde tan temprano, otorgándome la plaza de alumno interno que me ha llevado a la realización de este TFG. A pesar de los momentos en los que ha escaseado el tiempo, ha sido una experiencia muy buena que reperitía sin dudarlo.

A María, por su ayuda cada vez que la he necesitado, a Luis, por guiarme en los inicios con la programación y solucionarme cada nuevo problema que me surgía, y a Dani, por las pequeñas ayudas que ha realizado.

A todos aquellos profesores que demostraban día a día la pasión por su trabajo y que hacían de aprender la mejor de las experiencias.

A mis maestros y maestras del Colegio Salesiano de Utrera, por su dedicación. Especialmente a mis profesores Elena y Fernando, por sus enseñanzas, pero sobre todo por su cariño, su comprensión y su paciencia conmigo. Y en particular a Eduardo, por todo lo que aprendí de informática y de física, pero sobre todo, por hablarme de los campus científicos que me han traído directamente hasta aquí.

A mis compañeros de clase, por acompañarme durante todo el camino.

A Damián, por su apoyo y amistad en todo momento durante este último año.

A mis amigos de Utrera, por tener paciencia y no haberme abandonado tras cuatro años desaparecido con mis estudios.

A mi familia, por su amor y apoyo incondicional.

A mi abuelo, que a pesar de no haber podido estar aquí, se ha ido dejándome la mejor herencia que se puede dejar, una buena educación.

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 se reduce.

La hipótesis de la que partimos para diseñar las técnicas de diagnóstico es que los SEU próximos entre si producen patrones de error 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 candidato 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.

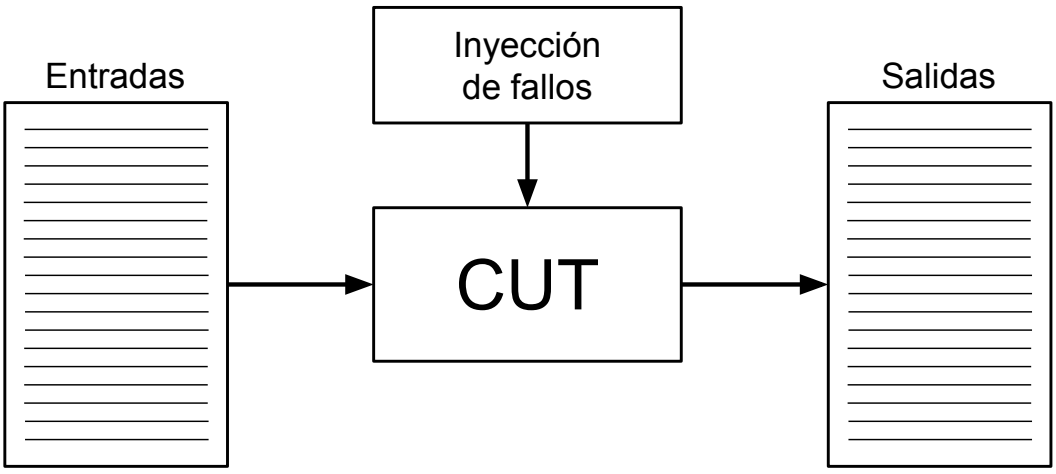


Figura 3.1 Ejecución de una campaña de inyección de fallos.

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.

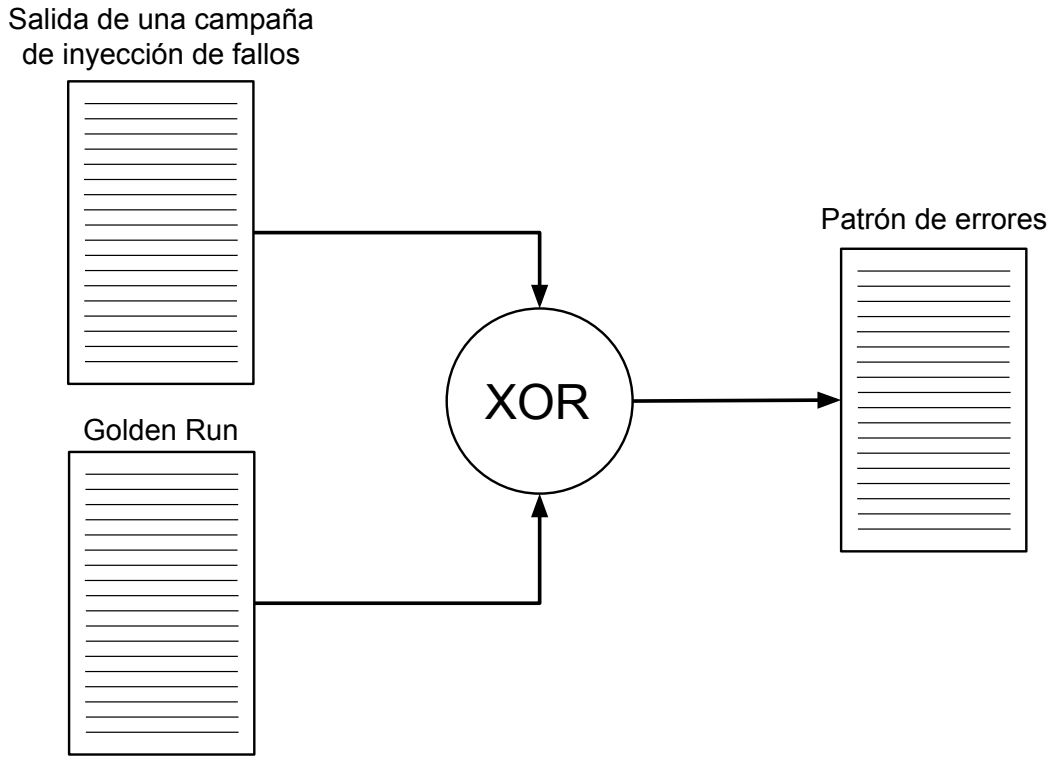


Figura 3.2 Postprocesado de la salida de una ejecución de la campaña.

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.

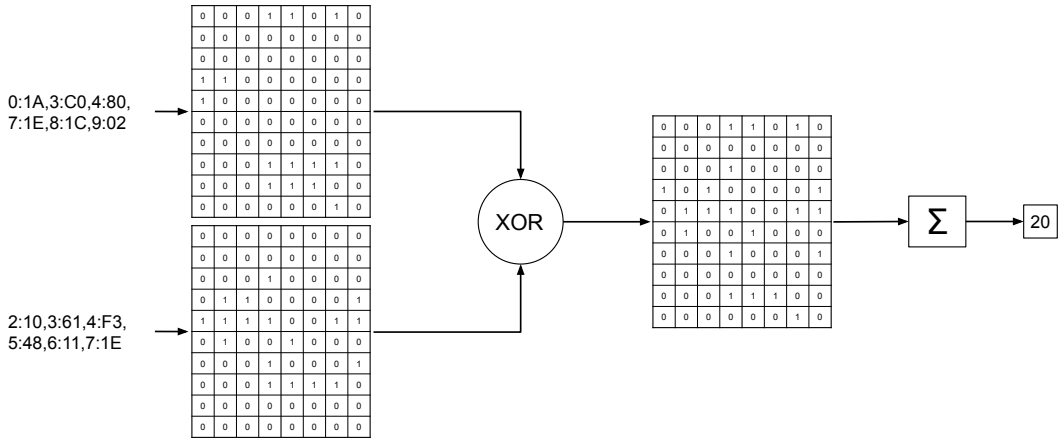


Figura 4.1 Cálculo de la distancia de Levenshtein.

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

La idea de implementar una distancia basada en ciclos surge de la observación realizada en los resultados de la distancia de Levenshtein (ver subsección 4.3.2). La también llamada *"distancia de ciclos"* se calcula a partir del primer ciclo en que la inyección se manifiesta a la salida, a partir de ahora *"first cycle"*. La distancia de ciclos se calcula como la diferencia entre el *first cycle* del target run y el *first cycle* de cada entrada del diccionario, pudiendo ser positiva, negativa o cero si el fallo del target run se manifiesta antes, despues o en el mismo ciclo que el de las entradas del diccionario.

La hipótesis que realizamos basándonos en las observaciones y la cual aplicamos para mejorar el diangóstico sería la siguiente:

Hipótesis 5.0.1 *"La distancia temporal, definida como la diferencia entre los primeros ciclos en que se manifiestan dos inyecciones, guarda relación directa con la diferencia entre los ciclos de inyección de dichas inyecciones".*

Puntualizar que, mientras una distancia de Levenshtein nula significa que producen el mismo fallo a la salida y por tanto, bien son el mismo SEU o bien colisionan; una distancia temporal nula no implica que nos encontremos ante la misma inyección. Se cumple que:

$$D_{Levenshtein} = 0 \Rightarrow D_{ciclo} = 0$$

Para generar la lista de distancias de ciclo entre el target run y el resto del diccionario primero leemos la información de ambos archivos tal y como explicabamos en la sección 4.1. Despues identificamos el primer ciclo en que se manifiestan las inyecciones para cada entrada del diccionario. Hacemos lo propio con el SEU que estamos diagnosticando. La distancia de ciclos se calcula como primer ciclo de la entrada menos primer ciclo del target run. Se hace el cálculo para cada entrada del diccionario y se almacenan las distancias temporales junto con las respectivas distancias de Levenshtein.

Al diagnosticar, suponemos que el target run presenta fallos a la salida, ya que debe existir un error que detectar antes de saber que tenemos algo que diagnosticar, por lo que siempre existirá primer ciclo para él. Esta afirmación no es cierta para el resto de entradas del diccionario. Todas se corresponden con una inyección, pero no todas producen errores a la salida. Esto se traduce en la existencia de runs vacíos en el archivo *"damages.csv"* y por tanto, en entradas del diccionario a las cuales no se le puede asignar un *first cycle* de forma inmediata.

Cuando una inyección no se manifiesta a la salida puede deberse a tres razones básicas:

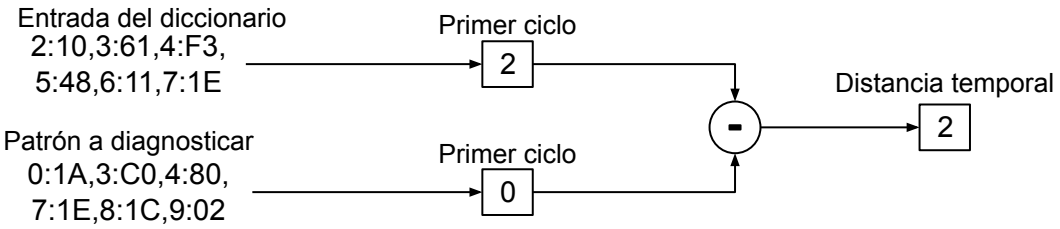


Figura 5.1 Cálculo de la distancia temporal.

1. El SEU se produjo durante el reset, por lo que automáticamente se restaura el sistema.
2. El SEU no ha tenido tiempo suficiente para manifestarse a la salida.
3. El SEU esta localizado (FF, ciclo) en un sitio donde no produce fallos a la salida.

Con la información que disponemos del propio diccionario, podemos identificar las inyecciones que se han producido durante un reset del circuito y las que se han producido muy próxima al fin de su ejecución, no teniendo tiempo de manifestarse.

Aunque podríamos aproximar estos dos casos a primer ciclo cero y último respectvamente, al final decidimos que lo mejor era descartar directamente los runs vacíos como candidatos, ya que el target run siempre presenta fallos a la salida y por tanto, en ningún caso puede corresponderse con una entrada del diccionario vacía.

5.1 Diagnóstico basado en la distancia temporal

La selección de candidatos, en su versión final, la realizamos de la misma forma que para la distancie de Levenshtein con un ligero matiz. Las entradas del diccionario se ordenan de menos a mayor según el valor absoluto de su distancia temporal al target run. Una vez hecho esto, se seleccionan los *"n"* primeros.

El FF de inyección de los candidatos seleccionados según la distancia temporal no aporta información. Sin embargo, observamos cómo, en circuitos simples, el ciclo donde se localiza el SEU bajo diagnóstico coincide con la diferencia entre el ciclo de inyección del candidato y la distancia temporal que este presente hasta el target run. Para circuitos más grandes, con un mayor número de biestables y que por tanto requieren ser ejecutados durante más ciclos para que el diccionario presente las diferencias suficientes entre inyección e inyección, observamos como este cálculo se aproxima bastante al valor del ciclo real del target run. En la *uart* por ejemplo, el margen de error oscila en torno a los 1000 ciclos de los 37000 ciclos que componen cada ejecución.

Observamos además cómo el error cometido disminuye si lo hacemos con candidatos que además dispongan de una distancia de Levenshtein relativamente baja.

5.2 Fusión de las distancias temporal y de Levenshtein

El objetivo que perseguimos al fusionar las dos distancias es sacar partido de los puntos fuertes de cada una de ellas.

Por un lado, la distancia de Levenshtein suele aproximarse más al registro e incluso FF correcto. Mientras que la distancia temporal resulta muy útil para localizar temporalmente al SEU.

El procedimiento que hemos seguido para diagnosticar conjuntamente con las dos distancias consistía en calcular todas las distancias y agruparlas en una sola lista, ordenar la lista de menor a mayor en función de la distancia de Levenshtein y seleccionar los *"n"* primeros.

Llegados a este punto, el algoritmo mostraba por pantalla, para cada candidato seleccionado, su distancia de Levenshtein, su distancia de ciclo y la información de su inyección. Además, dado

que para nosotros esa información era conocida, mostrabamos la localización correcta del SEU a diagnosticar. Podíamos comprobar cómo efectivamente, la inyección de los candidatos obtenido con la distancia de Levenshtein podía ser corregida con la distancia temporal.

Más adelante, para la vesión final de la técnica, contenida en el capítulo 7, veremos cómo hemos integrado el cómputo del ciclo de inyección completamente en el algoritmo.

5.3 Resultados experimentales

Dado que la distancia temporal no está lo suficientemente básada en la salida como para realizar un diagnóstico completo únicamente a partir de ella, es difícil mostrar unos resultados experiementales de este algortimo aislado. Además, no fue hasta más adelante, con la inclusión de dos aproximaciones más que aún faltan por explicar, que empezamos a registrar resultados donde la distancia temporal intervenía.

Es por eso que a continuación vamos a ver sólo unos datos que se obtienen previos a la fusión de las dos distancias.

5.3.1 Diccionarios no exhaustivos

Dado que el diagnóstico con diccionarios exhaustivos se resuelve únicamente con la distancia de Levenshtein, y esta investigación trata de desarrollar técnicas de diagnóstico de SEU con diccionarios incompletos, vamos a centarnos únicamenrte en los resultados que se obtienen con estos últimos.

Tabla 5.1 Resultados experimentales. Distancia de ciclos. Dic. incompletos ($\leq 5\,\%$).

Diseños	Registro	FF
adder_acum	100	37
counter	100	63
dual_counter	98	67
fifo	98	0
fir_ri (37'78 %)	21	4
pcm	67	52
shiftreg	100	52
simple_fsm	100	88
uart (0'87 %)	92	74

Los experimentos a partir de los cuales se han obtenido los datos de la tabla se han realizado de la misma forma que la explicada en el capítulo anterior (ver 4.3.2), y los números representan también el número de veces de las 100 simuladas que se encuentra el Registro/FF correcto entre los candidatos.

Observamos cómo efectivamente se tiene menos capacidad de diagnóstico para FF con la distancia de ciclo. Parece que además el porcentaje de acierto ronda en torno al 50 %, lo cual encaja con la hipótesis de que la ditancia temporal no sirve más que para localizar el SEU temporalmente, obteniendo este nivel de acierto fruto del azar, aunque con excepciones.

Para el *fifo* y el *fir_ri* el bajo nivel de acieto se explica debido a una cantidad muy alta de colisiones. Además de los *"n"* candidatos seleccionados hay muchos otros que son equidistantes a ellos y que no se seleccionan, quedando atrás las inyecciones correctas a causa de las colisiones.

Por el contrario, si que observamos circuitos donde se supera claramente la mitad de aciertos. El caso de la máquina finita de estados se explica porque existen muy pocos FF donde seleccionar, por

lo que es muy probable que uno de los correctos se encuentre entre los candidatos. La UART no tiene una explicación tán sencilla. Puede deberse a la propia ejecución que está ejecutando. Las zonas del circuito podrían estar activandose alternadamente por periodos, con lo que al detectar los más próximos temporalmente, estaríamos seleccionando además inyecciones a los mismos registros o incluso biestables.

6 Técnicas de diagnóstico auxiliares

Aunque la distancia de Levenshtein funciona muy bien una vez llegamos a las soluciones de descartar los runs vacíos del diccionario y controlar el número de candidatos seleccionados mediante la ordenación y selección de las *"n"* menores distancias en lugar de establecer un corte según el rango; podemos pensar que si el primer intento ha dado tan buenos resultados, existirán otras métricas incluso mejores aún por explorar.

Con este pensamiento en mente, tras convertir las entradas del diccionario en imágenes binarias, observamos patrones característicos para cada circuito. Cómo además las inyecciones de los diccionarios exhaustivos estaban ordenadas en la medida de lo posible, ocurría que a simple vista se podía observar relación entre una inyección y las que la rodeaban. Podemos observar incluso un efecto tipo *gif* si pasamos las imágenes ordenadamente con cierta velocidad.

Esta observación corroboraba la hipótesis de partida (hipótesis 1.0.1), pero además nos llevó a pensar que quizás se podrían aplicar algoritmos del campo de la percepción al diagnóstico de SEU.

En total probamos cuatro distancias diferentes con las que seleccionar candidatos.

6.1 Diagnóstico basado en el análisis de imágenes

En percepción, el procedimiento estudiado para identificar objetos en imágenes consistía en binarizar la imagen RGB, aplicar técnicas para que cada objeto quede separado del resto (los pixeles blancos de cada objeto estén separados del resto de objetos) y luego caracterizar las imágenes binarias de cada objeto de forma que puedan ser posteriormente identificados o que incluso se pueda entrenar un clasificador de objetos en base a ello.

El diagnóstico basado en imágenes consiste en tratar cada run como una imagen, y todo lo contenido en una imagen como un único objeto, sin importar que los fallos produzcan un patrón completamente unido en la imagen o no.

Para realizar la conversión de las entradas del diccionario en imágenes escribimos un pequeño programa en Python que se encargaba de leer la información del fichero *"damages.csv"* y generar un *.png* con ella.

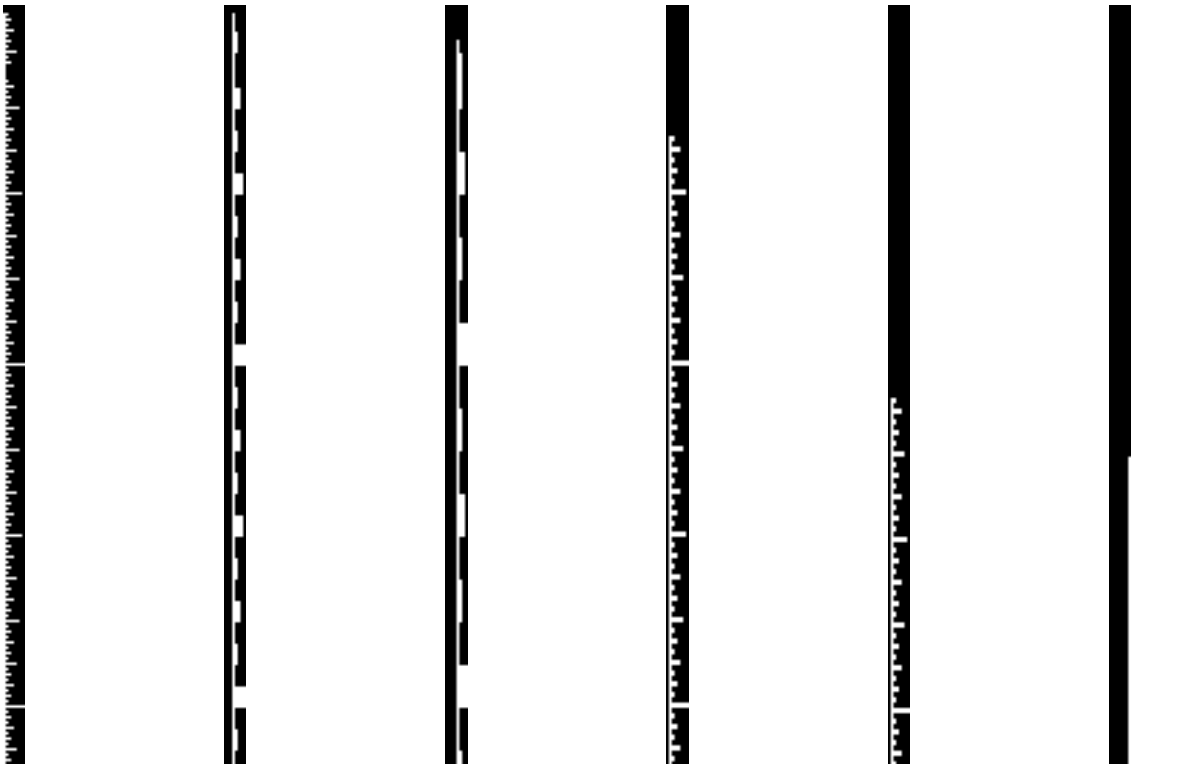
La caracterización escogida para posteriormente identificar a los objetos son los Momentos invariantes de Hu. Estos presentan la propiedad de ser invariantes a rotación, traslación y cambios de escala. Esta medida quizás es excesiva dada la naturaleza de los patrones que queremos identificar en imágenes, ya que no rotan, aunque si nos fijamos por ejemplo en la imagen 6.2 podemos observar cierto cambio de escala.

Los 7 momentos de Hu se calculan a partir de los momentos normales. Sus expresiones pueden consultarse en [27].

Las distancias basadas en estas expresiones se calculan de forma vectorial. Se construye un vector de 7 componentes para cada run con sus momentos de Hu; lo mismo para el target run. La distancia

(a) run 40. **(b)** run 42. **(c)** run 949. **(d)** run 1842.

Figura 6.1 Visualización de algunas inyecciones del adder_acum como imágenes.



(a) run 32. **(b)** run 35. **(c)** run 116. **(d)** run 401. **(e)** run 1185. **(f)** run 1367.

Figura 6.2 Visualización de algunas inyecciones del counter como imágenes.

será el módulo de la diferencia entre ambos vectores.

$$\overrightarrow{Hu_{run}} = (I_1, I_2, I_3, I_4, I_5, I_6, I_7) \tag{6.1}$$

$$D_{Hu} = |\overrightarrow{Hu_{run}} - \overrightarrow{Hu_{taget\_run}}| \tag{6.2}$$

La selección de candidatos en su versión final se realiza como hemos explicado anteriormente: ordenamos las distancias de Hu de menos a mayor y seleccionamos como candidatos los *"n"* primeros.

6.2 Diagnóstico por coincidencias

La última de las distancias que probamos fué la que hemos llamado *"distancia de coincidencia"*. Esta consiste en contabilizar los fallos a la salida que tienen en común dos inyeciones distintas, ignorando cuántos no. En la distancia de Levenshtein sucedía todo lo contrario: contabilizábamos las diferencias ignorando cuánto tenían en común.

Para calcuar las coincidencias sólo hay que realizar una pequeña modificación en el algoritmo encargado de calcular la distancia de Levenshtein. La única diferencia es que se emplea la operación lógica *"AND"* en lugar de la *"XOR"*. La distancia de coincidencia será la inversa del resultado obtenido tras la suma de todos los bits altos.

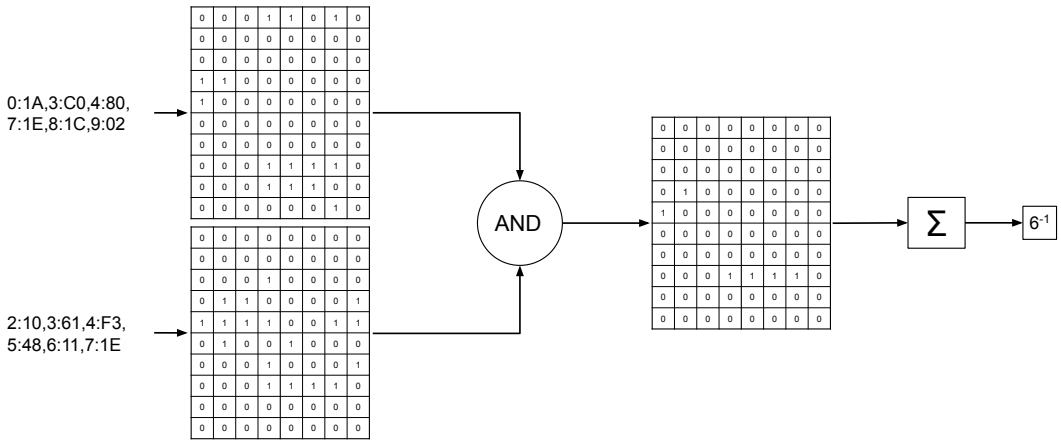


Figura 6.3 Cálculo de la distancia de coincidencia.

La selección de candidatos se realiza del mismo modo que anteriormente: ordenamos las distancias de menor a mayor y seleccionamos los *"n"* primeros.

6.3 Resultados experimentales

Con la finalidad de comprobar si podemos prescindir de algunos de los cuatro algoritmos como técnica de diagnóstico debido a que alguno de los demás siempre lo supere, hemos realizado un estudio en el que, para cada circuito, seleccionábamos 100 inyecciones a diagnósticar y calculábamos candidatos independientemente con las cuatro distancias.

Respecto al método de selección de candidatos explicado hasta ahora, de este estudio en adelante realizamos modificaciones, de forma que no tienen por qué seleccionarse *"n"* candidatos exactamete si se dan ciertas situaciones.

- Si el diccionario contiene menos de *"n"* runs no vacíos, la cantidad de candidatos seleccionados será menor que *"n"* e igual a la cantidad de runs no vacíos.
- Si, tras ordenar las distancias, al candidato número *"n"* le siguen inyecciones equidistantes a él, estas serán también seleccionadas cómo candidatos (con un máximo de $2 \times n$ candidatos).

El primer caso se explica solo. Respecto al segundo, decidimos ampliar la selección de esta manera para no descartar candidatos que según la distancia son tan válidos como el número *"n"*. El tope de $2 \times n$ va en contra de este razonamiento, pero ocurría que circuitos con muchas colisiones seleccionaban muchísimos candidatos.

El primer detalle a destacar de ambas tablas (6.1 y 6.2) es el elevado números de veces que se aciertan tanto el registro como el FF de inyección agrupando los candidatos de los cuatro algoritmos y seleccionando únicamente 5 candidatos con cada uno. Cabe resaltar que en las situaciones concretas

Tabla 6.1 Resultados experimentales. Diagnóstico del registro de inyección con los 4 algoritmos. Dicccionarios incompletos ($\leq 5\,\%$).

Diseños\Algoritmos	Levenshtein	Ciclos	Hu	Coincidencias	Todos
adder_acum	100	100	100	100	100
counter	100	100	100	100	100
dual_counter	99	98	99	100	100
fifo	98	98	97	95	99
fir_ri (37'78 %)	29	21	55	30	78
pcm	82	67	82	73	82
shiftreg	100	100	100	32	100
simple_fsm	100	100	100	100	100
uart (0'87 %)	97	92	96	84	97

Tabla 6.2 Resultados experimentales. Diagnóstico del FF de inyección con los 4 algoritmos. Dicccionarios incompletos ($\leq 5\,\%$).

Diseños\Algoritmos	Levenshtein	Ciclos	Hu	Coincidencias	Todos
adder_acum	98	37	35	100	100
counter	96	63	72	100	100
dual_counter	99	67	74	94	100
fifo	1	0	0	1	1
fir_ri (37'78 %)	3	4	4	10	16
pcm	69	52	68	56	69
shiftreg	82	52	53	4	91
simple_fsm	88	88	88	88	88
uart (0'87 %)	93	74	91	79	93

mencionadas, pueden ser más o menos de 5 candidatos por algoritmo. También es muy posible que los candidatos estén repetidos, ya que un buen candidato tendrá simultáneamente distancias pequeñas en todos los algoritmos, con el límite de 0 para las colisiones (excepto la distancia de coincidencia, que funciona de forma diferente).

Como habíamos comentado anteriormente, los datos parecen indicar que la distancia de Levenshtein siempre tiene más acierto que la distancia en ciclos. Sin embargo, esto no siempre se cumple con el resto. El primer ejemplo de ello es el *dual_counter* de la tabla 6.1, donde por primera vez el algoritmo de Levenshtein es superado por el de coincidencias. Esto mismo vuelve a suceder de forma más acusada en la tabla 6.2 con los algoritmos de Hu y Coincidencia.

Son especialmente interesante las filas donde la columna *"Todos"* es mayor estricto al resto. El número contenido en la columna *"Todos"* representa el número de veces que obtenemos el registro y FF correcto con los cuatro algoritmos conjuntamente. El hecho de que esta columna a veces sea mayor que las anteriores demuestra que no hay algoritmo mejor, sino que dependiendo del caso concreto, funcionará mejor un algoritmo u otro. Los casos más claros son el *fir_ri* de la tabla 6.1 y los casos *fir_ri* y *shiftreg* de la tabla 6.2.

Cabe mencionar que los resultados obtenidos con el *simple_fsm* no son válidos, ya que el diccionario exhaustivo tiene 64 entradas y este fue reducido hasta un diccionario del 5 % de exhaustividad, haciendo un total de 5 entradas de las cuales 1 además es vacía. Dado que la *"n"* se fijó a 5, los algoritmos estaban seleccionando siempre los únicos 4 candidatos disponibles. Esto lo supimos posteriormente a obtener los resultados de los experimentos.

Así mismo, comprobamos que los malos resultados obtenidos tanto en *fifo*, como en *fir_ri*, como en *pcm*, únicos circuitos en los que el diagnóstico se aleja del 90 % de aciertos espaciales, se deben a la presencia de un alto número de colisiones. Dada la limitación de $2 \times n$ candidatos impuesta para runs equidistantes, los candidatos correctos están permaneciendo fuera de la selección.

En el siguiente capítulo explicaremos en que consiste una *Campaña Iterativa*. Para realizar la primera iteración, es necesario realizar también un primer diagnóstico. Basándonos en los resultados presentes en las tablas 6.1 y 6.2, hemos decidido seleccionar $4 \times n$ candidatos, es decir, *"n"* candidatos con cada algoritmo (cantidad variable en función de las situaciones especiales comentadas). Es de esperar que ciertos candidatos aparezcan repetidos, cada vez que esto ocurra se contabilizará, siendo una medida extra de la calidad de cada uno de los candidatos, ya que habrá sido seleccionado por más de un algoritmo.

Como valor añadido, la selección simultánea de candidatos con varias distancias puede ser una posible medida para evitar mínimos locales. El algoritmo que mejor encaja como forma de abordar los mínimos locales es el de la distancia temporal, ya que, como concluimos anteriormente, selecciona inyecciones que desde el punto de vista de los biestables, con excepciones, son más aleatorias. Abordaremos con más detalle el problema de los mínimos locales en el capítulo 7.

7 Campañas iterativas a partir de los candidatos seleccionados

Las campañas iterativas serían el recurso propuesto para diagnósticos que no encuentran candidatos a distancias cero, es decir, cuando el diccionario no contiene al SEU que queremos diagnósticar. Esta tecnica aplica para diccionarios incompletos, ya que un diccionario completo siempre contendrá por definición al target run.

Esta técnica se emplea en situaciones en las que el diccionario es extremadamente pequeño pero sin llegar a perder completamente la capacidad de diagnóstico. El concepto es sencillo. Se obtiene un diccionario incompleto del circuito mediante inyecciones aleatorias (*"Random Sampling"*), a partir de él, realizamos el primer diagnóstico. Obtenemos una lista preliminar de candidatos a partir de la cual extraemos la información necesaria para una siguiente inyección, que ya no será aleatoria, al menos en su mayoría. El proceso de obtención de diccionario, diagnóstico, y obtención de un nuevo diccionario a partir del el cual volver a diagnosticar puede repetirse tantas veces como sea necesario, finalizando con la detección de al menos una inyección que colisione con el target run.

Según la información de salida del algoritmo que selecciona los candidatos, podemos realizar nuevas campañas desde distintos enfoques.

- Campaña enfocada en ciclos: si la información de salida no deja duda sobre en qué ciclo se localiza el SEU, podemos realizar una campaña de inyección de fallos en la que sólo inyectemos durante ese ciclo en concreto. El espacio total a inyectar se reduce al número de biestables, y la longitud de la simulación se divide por la cantidad de ciclos que componían cada ejecución.
- Campaña enfocada en registros: cuando la información de la salida apunta claramente a un registro o un conjunto de ellos, la campaña de inyección de fallos a realizar podría centrarse en inyectar esas zonas del circuito, reduciendo el espacio de inyección al descartar de la campaña el resto de registros del CUT.
- Campaña enfocada en biestables: del mismo modo, podríamos inyectar FF concretos del CUT en todos los ciclos posibles.
- Combinación de las restricciones anteriores: si el circuito es tan grande que toda reducción del espacio de inyección es poca, podemos acotar la siguiente campaña combinando restricciones. Inyectar un registro durante un rango de ciclos, un ciclo pero un conjunto de FF, etc.

El código que selecciona y procesa los candidatos devuelve una lista de biestables concretos, donde cada biestable va acompañado del número de veces que ha sido seleccionado y de un rango de ciclos que se calcula en función de la distancia en ciclos. También se indica el ciclo central del rango, siendo posible, cuando no cabe ninguna duda, de que el rango sea un único ciclo. Por separado, se

muestran, de existir, las inyecciones del diccionario que producen exactamente el mismo patrón de fallos.

Con esta información, es el usuario el que decide cuál de las 4 campañas descritas aplicar, siendo una buena opción inyectar tal cual esos biestables en el rango de ciclos que los acompañan. En cualquier caso, si la duración de la campaña lo permite, realizar una campaña menos acotada evitará malos diagnósticos y mínimos locales.

Alcanzar un mínimo local en este contexto sería como "seguir una pista falsa". Corremos el peligro de caer en un mínimo local cuando el candidato que presenta la menos de las distancias no apunta a donde debería. Esto es más probable conforme reducimos la exhaustividad del diccionario. De esta forma, campaña tras campaña, acotaríamos el espacio de inyección en torno a una zona que presenta aparentemente menos distancia al taget run, pero en la que nunca detectaremos una colisión.

Este problema se podría evitar aumentado el número de candidatos que seleccionamos (*"n"*) o tratando de acotar más progresivamente. Otra posibilidad es la incluir inyecciones aleatoriamente en cada nueva campaña, de forma que se aumenten las probabilidades de estimular la zona correcta si previamente esta no estaba contenida en el diccionario.

Las características y puntos fuertes de cada técnica en particular también pueden propiciar la caída en un mínimo local. Esta es otra razón por la que hemos decidido mantener los cuatro algoritmos a la hora de extraer candidatos. El objetivo es que, si uno de los algoritmos se ve especialemente influenciado por un mínimo local, los otros tres incluyan candidatos en la campaña que pertenezcan a otra zona del circuito. Como mencionamos en los resultados del capítulo anterior, la distancia en ciclo cumple especialemente bien este objetivo al no tener capacidad para diagnosticar espacialmente por si sola.

7.1 Obtención de la lista de candidatos

Primeramente leemos la información tanto de *"dammages.csv"* como de *"injections.csv"* y leemos el patrón de fallos a diagnósticar. En este momento calculamos todas las distancias y agrupamos la información de las distancias y las inyecciones en una lista. Tal y como hemos explicado para cada distancia, realizamos la selección de los *"n"* primeros candidatos con cada una (máximo de $2 \times n$ como hemos comentado).

Agrupamos los $4 \times n$ candidatos en una sola lista, manteniendo aún todas las distancias para cada uno de ellos. A parir de este momento comenzamos a preparar la información necesaria para la siguiente iteración.

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

Antes que nada, comprobamos si el diagnóstico ha concluido, es decir, si alguno de los candidatos presenta distancia de Levenshtein, ciclo y Hu igual a cero. Tanto en caso negativo como en positivo, procedemos a calcular la información de inyección recomendada para la siguiente campaña de inyección de fallos.

Aunque tras detectar candidatos a distancia cero pueda parecer que el diagnóstico no puede mejorar, tenemos que tener en cuenta la posibilidad de encontrarnos ante una colisión, por lo que la localización correcta del SEU no sería esa. Realizar una iteración extra y obtener una nueva lista de candidatos puede mejorar el diagnóstico en este sentido, ya que podría llegar a localizar más inyecciones que generen exactamente el mismo patrón, consiguiendo un diagnóstico más completo.

La información recomendada para la siguiente inyección es de la forma que hemos explicado hace un momento, biestables acompañados del número de veces que han sido seleccionados y el

rango de ciclos en el que podría estar, centrado en un ciclo concreto e incluyendo márgenes de error. Para extraer esta información, primero separamos las inyecciones de los candidatos en ciclo, registro y FF. En este punto disponemos de dos posibilidades, agrupar inyecciones por FF o por registros repetidos, realizando el recuento de cuántas veces se repiten. En nuestro caso, dado el alto porcentage de acierto inicial, hemos decidido agrupar los candidatos repetidos por biestables.

Por último, recorremos de nuevo la lista de candidatos que contiene las distancias calculando el rango de ciclos en los que el algoritmo selecciona cada uno de ellos y estimando la posición central. Esta se calcula mediante la ecuación 7.1, y el margen de error que compone el rango se ha establecido como dos veces la distancia en ciclos para cada sentido a partir del ciclo central.

$$Ciclo\_Central = Ciclo\_de\_inyeccin - Distancia\_en\_ciclos \tag{7.1}$$

Como resultado de este proceso obtenemos una lista con toda la información necesaria para realizar la siguiente campaña de inyección de fallos. Esta información se muestra con el siguiente formato:

[ciclo inferior, ciclo central, ciclo superior] :/ registro/ FF nº de repeticiones

Aunque esta información tal cual se correspondería con la próxima campaña recomendada, aún podemos decidir manualmente, tras observar los rangos de ciclos y la variedad de registros o FF, si realizaremos una campaña especialmente enfocada en alguna parte del circuito tal y como hemos comentado anteriormente.

7.3 Resultados experimentales

Los experimentos realizados para validar esta técnica se basan en la siguiente hipótesis:

Hipótesis 7.3.1 *"Si el espacio de inyección propuesto por el algoritmo es lo suficientemente pequeño como para realizar en él una campaña exhaustiva, y en en él se encuentra la inyección correcta; la siguiente campaña detectará al menos una colisión".*

Conociendo de antemano la inyección del SEU que se quiere diagnosticar, se puede saber si el proceso iterativo terminará detectando colisiones o no. Podemos pues ahorrarnos el proceso de inyección de cara a validar la técnica, ya que los experimentos de diagnóstico se están realizando para conmutaciones lógicas simuladas, y por tanto conocemos la información de la inyección. Esta información nos permite tanto validar la técnica como saber de antemano cuál es la mejor campaña que podemos aplicar dada la salida del algoritmo.

Hemos fijado *"n"* a 5. Esto hace un total de 20 candidatos (con un máximo de 40) en caso de que no se repita ninguno. Si además tenemos en cuenta que el algoritmo nos acota también el espacio de inyección temporalmente, podemos afirmar que este es lo suficientemente pequeño como para aplicar una campaña de inyección de fallos exhaustiva.

Por supuesto, aún tenemos libertad para decidir a qué ciclos aplicamos la campaña, pudiendo ampliarlos o reducirlos, y si la aplicamos a biestables concretos o si por el conrtrario inyecctamos en todos los FF de los registros seleccionados. Hemos programado un código que analiza la salida del algoritmo, la compara con la información de la inyección y predice el resultado que se obtendría al realizar realmente las campañas de inyección de fallos. Las posibles salidas de este código son:

- *"Next campaign will fail"*: en caso de que entre los candidatos no se encuentre ni el FF, ni el registro, ni el ciclo correcto.
- *"Will need more than one iteration to hit the target"*: cuando entre los candidatos se encuentra el FF correcto, pero el ciclo está fuera del rango. Los nuevos candidatos extraidos del diccionario resultado de la siguiente inyección afinarán la predicción temporal. Una campaña exhaustiva

para los FF seleccionados no acotada temporalmente solo necesitaría una campaña extra para colisionar.

- "If next campaign is oriented to registers, will hit the target": tanto el registro como el ciclo correcto están en el espacio de inyección acotado, pero el biestable concreto no aparece. Si la campaña la enfocamos a registros, sin concretar en cuáles de sus biestables se inyecta, localizará al target run.
- "Next campaign will hit the target": tanto ciclo, como registro, como biestable están en el espacio de inyección seleccionado. Las campañas iterativas finalizarían en la siguiente iteración.
- "Next campaign will contain only right cycle": sólo una campaña acotada temporalmente aseguraría mejorar el diagnóstico en la siguiente iteración.
- "Will need a register oriented campaign and more than one iteration": ni ciclo ni biestable correcto se encuentran en el espacio de inyección seleccionado. A base de iterar podría mejorar el diagnóstico temporal empleando la información de las nuevas distancias en ciclo, pero sería necesario no enfocar las campañas a biestables concretos.

Evidentemente, en una aplicación real no dispondríamos de la información correcta de la inyección, por lo que no se sabría qué tipo de campaña realizar en la proxima iteración para mejorar el diagnóstico. La información del número de veces que se repite cada biestable entre los candidatos puede orientarnos a la hora de saber si realmente el FF correcto se encuentra entre ellos o no. Aún así, es posible que se necesiten varios intentos de selección de tipo de campaña en caso de que, debido al gran tamaño inicial del espacio de inyección, sea necesario acotar lo máximo posible.

En líneas generales, los resultados muestran como el candidato que se repide un mayor número de veces es el que apunta al biestable correcto si la diferencia de repeticiones respecto al resto es notable. A pasar de lo claro que sea este resultado, se debe inyectar también en el resto de candidatos, en caso de que no se haya detectado colisión, para evitar caer en un mínimo local.

He seleccionado los resultados más significativos de todos los experimentos realizados, mostrando algún ejemplo para cada posible caso de los anteiormente enumerados.

Salida 7.1 Adder Acum: diagnóstico de la inyección 11:/i_valor_acum/10.

```
[-62, 11, 84] :/ i_valor_acum/ 10 x8
[-77, 11, 99] :/ i_valor_acum/ 15 x2
[-58, 11, 80] :/ i_valor_acum/ 5 x2
[11, 11, 11] :/ i_valor_acum/ 9 x1
[11, 11, 11] :/ i_valor_acum/ 17 x1
[10, 11, 12] :/ i_valor_acum/ 16 x1
[9, 11, 13] :/ i_valor_acum/ 8 x2
[8, 11, 14] :/ i_valor_acum/ 6 x1
[5, 11, 17] :/ i_valor_acum/ 1 x1
[5, 11, 17] :/ i_valor_acum/ 0 x1

11  :/  i_valor_acum/ 10
Next campaign will hit the target
```

En la salida 7.1 no se produce colisión, pero en la información de salida vemos que todos los ciclos centrales apuntan al ciclo 11 (el correcto) y que 8 de los 20 candidatos apuntan hacia el FF correcto. Se podría realizar una campaña de una única inyección al (11, i_valor_acum/10) para comprobar que efectivamente, esa localización produce el mismo patrón de fallos a la salida que el target run, o se podría simplemente lanzar la siguiente campaña sugerida al completo.

Salida 7.2 Adder Acum: diagnóstico de la inyección 63:/i_valor_acum/4.

```
Iterative campaign its over. Colision founded:
63  :/  i_valor_acum/ 4

To assure the result, this is the sugested campaign:

[35, 63, 91] :/ i_valor_acum/ 4 x8
[27, 63, 99] :/ i_valor_acum/ 15 x1
[28, 63, 98] :/ i_valor_acum/ 13 x1
[63, 63, 63] :/ i_valor_acum/ 0 x1
[63, 63, 63] :/ i_valor_acum/ 6 x2
[63, 63, 63] :/ i_valor_acum/ 17 x1
[62, 63, 64] :/ i_valor_acum/ 8 x2
[62, 63, 64] :/ i_valor_acum/ 9 x1
[50, 63, 76] :/ i_valor_acum/ 14 x1
[59, 63, 67] :/ i_valor_acum/ 10 x1

63  :/  i_valor_acum/ 4
Next campaign will hit the target
```

La salida 7.2 ha detectado una colisión, que además sabemos es la correcta. La información de la campaña sugerida para afianzar los resultados apunta además al ciclo y registro corretos en su mayoría. Este es el mejor resultado que se puede conseguir.

Salida 7.3 Dual Counter: diagnóstico de la inyección 187:/counter1/reg_i/5.

```
[26, 187, 348] :/ counter1/reg_i/ 5 x16
[186, 187, 188] :/ counter1/reg_i/ 4 x2
[186, 187, 188] :/ counter1/reg_i/ 1 x2
[182, 187, 192] :/ counter1/reg_i/ 0 x1
[182, 187, 192] :/ counter1/reg_i/ 2 x1
[67, 187, 307] :/ counter1/reg_i/ 7 x2
[73, 187, 301] :/ counter1/reg_i/ 3 x1

187 :/ counter1/reg_i/ 5
Next campaign will hit the target
```

El resultado de la salida 7.3 es idéntico al resultado 7.1, se aciertan ciclo y biestable correctos, con el valor añadido de que este circuito tiene más de un registro.

Salida 7.4 Dual Counter: diagnóstico de la inyección 157:/counter0/reg_i/4.

```
Iterative campaign its over. Colision founded:
27  :/  counter0/reg_i/ 4
121 :/ counter0/reg_i/ 4
159 :/ counter0/reg_i/ 4
161 :/ counter0/reg_i/ 4
223 :/ counter0/reg_i/ 4
231 :/ counter0/reg_i/ 4
251 :/ counter0/reg_i/ 4
```

```
To assure the result, this is the sugested campaign:

[27, 150, 251] :/ counter0/reg_i/ 4 x16
[14, 14, 14] :/ counter0/reg_i/ 1 x1
[39, 39, 39] :/ counter0/reg_i/ 2 x1
[53, 57, 63] :/ counter0/reg_i/ 5 x6
[54, 63, 70] :/ counter0/reg_i/ 0 x3
[68, 68, 68] :/ counter0/reg_i/ 3 x1
[12, 264, 516] :/ counter1/reg_i/ 0 x5
[86, 86, 86] :/ counter0/reg_i/ 6 x1

157 :/ counter0/reg_i/ 4
Next campaign will hit the target
```

La salida 7.4 es un ejemplo de campaña finalizada en múltiples colisiones. El diagnóstico termina pero no se sabe con certeza la localización del SEU. De hecho, todas las colisiones apuntan al biestable correcto pero entre ellas no se encuentra la localización real. Una campaña extra, como hemos explicado, mejoraría los resultados al encontrar más colisiones, estando ahora sí, la localización correcta entre ellas.

Salida 7.5 PCM: diagnóstico de la inyección 31:/I2S_IN_1/DATA_L/1.

```
Iterative campaign its over. Colision founded:
9 :/ I2S_IN_1/DATA_L/ 1
10 :/ I2S_IN_1/DATA_L/ 1
11 :/ I2S_IN_1/DATA_L/ 1
12 :/ I2S_IN_1/DATA_L/ 1
13 :/ I2S_IN_1/DATA_L/ 1
14 :/ I2S_IN_1/DATA_L/ 1
15 :/ I2S_IN_1/DATA_L/ 1
16 :/ I2S_IN_1/DATA_L/ 1
17 :/ I2S_IN_1/DATA_L/ 1
18 :/ I2S_IN_1/DATA_L/ 1

To assure the result, this is the sugested campaign:

[9, 13, 18] :/ I2S_IN_1/DATA_L/ 1 x36
[12, 14, 15] :/ CLK_96k/s_counter_lr/ 1 x4

31 :/ I2S_IN_1/DATA_L/ 1
Will need more than one iteration to hit the target
```

En la salida 7.5 volvemos a tener colisión múltiple. Esta vez, la campaña sugerida apunta por mayoría absoluta al FF correcto, pero el ciclo se encuentra fuera del rango. Las distancias temporales de los nuevos candidatos tras una inyección extra es posible que nos reorienten el rango de ciclos, pero dado que existen colisiones, en un caso real no realizaríamos el mínimo de dos campañas más que se necesitarían para encontrar primero el ciclo correcto y luego inyectarlo.

Salida 7.6 FIFO: diagnóstico de la inyección 3:/Memory_149_25.

```
Iterative campaign its over. Colision founded:
3 :/ Memory_1_6
3 :/ Memory_1_10
3 :/ Memory_1_27
3 :/ Memory_2_12
3 :/ Memory_2_26
3 :/ Memory_4_15
3 :/ Memory_4_28
3 :/ Memory_5_11
3 :/ Memory_5_12
3 :/ Memory_5_20

To assure the result, this is the sugested campaign:

[3, 3, 3] :/ Memory_1_6 x3
[3, 3, 3] :/ Memory_1_10 x3
[3, 3, 3] :/ Memory_1_27 x3
[3, 3, 3] :/ Memory_2_12 x3
[3, 3, 3] :/ Memory_2_26 x3
[3, 3, 3] :/ Memory_4_15 x3
[3, 3, 3] :/ Memory_4_28 x3
[3, 3, 3] :/ Memory_5_11 x3
[0, 3, 6] :/ Memory_5_12 x4
[3, 3, 3] :/ Memory_5_20 x3
[0, 3, 6] :/ Memory_0_2 x1
[0, 3, 6] :/ Memory_1_13 x1
[0, 3, 6] :/ Memory_2_8 x1
[0, 3, 6] :/ Memory_2_22 x1
[0, 3, 6] :/ Memory_3_6 x1
[0, 3, 6] :/ Memory_3_11 x1
[0, 3, 6] :/ Memory_3_16 x1
[0, 3, 6] :/ Memory_4_16 x1
[0, 3, 6] :/ Memory_6_11 x1

3 :/ Memory_149_25
Next campaign will contain only right cycle
```

Otra situación extra de colisión múltiple se produce en la salida 7.6. Todos los candidatos de la campaña sugerida están de acuerdo en el ciclo, el cual resulta ser el correcto. Se podría realizar una inyección abierta a todo el circuito pero centrada únicamente en ese ciclo con tal de detectar otras posibles colisiones que completen aún más el diagnóstico. Notar que son tantas las colisiones, que se han seleccionado el máximo impuesto de 40 candidatos y el target run no es uno de ellos.

Salida 7.7 UART: diagnóstico de la inyección 5804:/uart_i/uart_rx_i/rx_data/7.

```
Iterative campaign its over. Colision founded:
8426 :/ uart_i/uart_rx_i/rx_data/ 1
6131 :/ uart_i/uart_rx_i/rx_data/ 6
6750 :/ uart_i/uart_rx_i/rx_data/ 5
7210 :/ uart_i/uart_rx_i/rx_data/ 4
```

```
6407 :/ uart_i/uart_rx_i/rx_data/ 6
9079 :/ uart_i/uart_rx_i/rx_data/ 0
6453 :/ uart_i/uart_rx_i/rx_data/ 6
6864 :/ uart_i/uart_rx_i/rx_data/ 5
8470 :/ uart_i/uart_rx_i/rx_data/ 1
6172 :/ uart_i/uart_rx_i/rx_data/ 6

To assure the result, this is the sugested campaign:

[8426, 8444, 8470] :/ uart_i/uart_rx_i/rx_data/ 1 x5
[6131, 6273, 6453] :/ uart_i/uart_rx_i/rx_data/ 6 x9
[6750, 6807, 6864] :/ uart_i/uart_rx_i/rx_data/ 5 x4
[7210, 7210, 7210] :/ uart_i/uart_rx_i/rx_data/ 4 x2
[9079, 9079, 9079] :/ uart_i/uart_rx_i/rx_data/ 0 x2
[1146, 5118, 9564] :/ uart_i/uart_tx_i/tx_bit_count/ 1 x13
[7014, 7782, 8550] :/ uart_i/uart_rx_i/rx_bit_count/ 2 x2
[4061, 9498, 14969] :/ uart_i/uart_rx_i/ rx_pstate_FSM_FFd2 x2
[6483, 6969, 7455] :/ uart_i/uart_rx_i/rx_bit_count/ 1 x1

5804 :/ uart_i/uart_rx_i/rx_data/ 7
```

La salida 7.7 es previa a que programáramos el código que predice los resultados de la siguiente campaña, por eso no se muestra la última línea como es habitual. Tras analizar muchas salidas de este tipo de experimentos, este es el peor diagnóstico que he encontrado, y aún así detecta múltiples colisiones, todas apuntando al registro correcto aunque en diferentes biestables, si ser el correcto uno de ellos. Entre la información de la campaña sugerida no encontramos el ciclo correcto, ni el FF correcto (realmente el cilo correcto sí que se encuentra entre los rangos, pero al no ir compañando al registro correcto, consideramos que no aparece). El FF más votado se encuentra en un registro diferente (mínimo local), pero aún así, el registro correcto es votado 22 veces en total frente a las 18 restantes. Adicionalmente, el registro erróneo más votado, está centrado muy aproximadamente alrededor del ciclo correcto. Si no hubieran existido colisiones, incluso en estas condiciones, con un poco de atención podríamos haber detectado estas pista y programado una nueva campaña enfocada al registro y rango de ciclos correcto.

Salida 7.8 UART: diagnóstico de la inyección 9179:/uart_i/uart_clk_cnt/2.

```
Iterative campaign its over. Colision founded:
10180 :/ uart_i/uart_clk_cnt/ 2
10153 :/ uart_i/uart_clk_cnt/ 2
10327 :/ uart_i/uart_clk_cnt/ 2
9287 :/ uart_i/uart_clk_cnt/ 2
9954 :/ uart_i/uart_clk_cnt/ 2
10270 :/ uart_i/uart_clk_cnt/ 2
10216 :/ uart_i/uart_clk_cnt/ 2

To assure the result, this is the sugested campaign:

[147, 9487, 15662] :/ uart_i/uart_clk_cnt/ 2 x34

9179 :/ uart_i/uart_clk_cnt/ 2
```

```
Next campaign will hit the target
```

Por último, la salida 7.8 muestra un nuevo caso de colisión múltiple entre los candidatos, con la diferencia de que todas apuntan por unanimidad al biestable correcto, como el resto de la campaña sugerida. La localización temporal exacta no se encuentra entre los candidatos, pero si en el rango de ciclos propuesto. Respecto al rango de ciclos, comentar que, al incluirle un margen de seguridad de el doble de la distancia en ciclo máxima para cada sentido, que hacen un rango total de cuatro veces la distancia máxima, este abarca prácticamente todo el espacio de inyección. Podemos notar que el ciclo central se aproxima mucho al ciclo correcto. Una buena campaña para detectar más colisiones podría ser inyecctar ese biestable en todos los ciclos de la ejecución.

Cabe destacar que los diccionarios presentaban una exhaustividad del 5 %, pero sobre todo, que de aquellos originalmente no exhaustivos se han eliminado las entradas correspondientes a los 100 SEU que se estaban diagnosticando, explicando que en ninguna de las pruebas se haya detectado a la primera la colisión correcta.

Intentos de reducir aún más tanto los diccionarios completos como los diccionarios originalmente incompletos demostraron que la reducción afecta más a aquellos diccionarios que originalmente eran de por sí muy pequeños. Esto se debe a que, para el *adder_acum* por ejemplo, las 2000 entradas del diccionario exhaustivo original se traducen en 100 entadas para el diccionario reducido al 5 %. Reducirlo al 1 % significaría diagnosticar con un diccionario de 20 muestras, que lo hace estadísticamente insuficiente.

Sin embargo, en circuitos grandes como la *uart*, el 0’87 % de exhaustividad inicial se traduce en un diccionario de 16496 entradas. Reducirlo al 5 %, resultando en un diccionario exhaustivo al 0’04 %, no elimina completamente nuestra capacidad de diagnóstico.

8 Conclusiones y trabajos futuros

8.1 Conclusiones

El propósito de esta investigación era desarrollar técnicas de diagnóstico de SEU utilizando diccionario de fallos incompletos. Para ello, partíamos de la hipótesis 1.0.1, marcando la línea de investigación en encontar una métrica adecuada que permita sacar partido de las similitudes en los patrones que generan dos conmutaciones lógicas próximas entre sí.

Hemos realizado experimentos en los que se diagnosticaba aplicando métricas cuyos enfoques eran diferentes, todos ellos devolviendo unos resultados muy buenos. Además, hemos establecido las bases de una técnica de diagnóstico consistente en la realización de numerosas iteraciones de *"Campaña de inyección de fallos para generar un diccionario de fallos"*, *"Diagnóstico empleando el diccionario de fallos obtenido"*, *"Extracción de candidatos que acoten el espacio de inyección"*, con la cuál hemos podido realizar diagnósticos exitosos con diccionarios de fallos muy incompletos en circuitos de grandes dimensiones. Además, observamos como en muchas ocasaiones, el diagnóstico se resuelve con la campaña inicial, ya que los candidatos obtenidos no dejan duda.

Se ha propuesto además que siempre se realice una campaña extra tras finalizar el diagnóstico, ya que los resultados apuntan a que iteraciones extras obtendrían, de darse el caso, otras colisiones extras que enriquecerían el diagnóstico.

La técnica de campañas iterativas puede llevarnos hacia un mínimo local donde no se encuentre el SEU bajo diagnóstico, fallando su misión. Hemos propuesto soluciones a este problema tales como la inclusión de inyecciones aleatorias en cada iteración.

Hemos comprobado que la distancia temporal es especialmente útil para localizar temporalmente el SEU bajo diagnóstico. Para circuitos grandes, con diccionarios muy reducidos, la exactitud de la distancia en ciclos se reduce, pero los resultados demuestran que es capaz de acotar bastante la localización temporal del SEU.

La distancia de Levenshtein funciona especialmente bien como métrica de diagnóstico. Hemos comprobado que a distancias de Levenshtein bajas, el cálculo de ciclo realizado con la distancia temporal (ecuación 7.1) mejora, aunque no es un requisito imprescindible para que este cálculo funcione con exactitud.

Los experimentos encaminados a determinar cuándo se pierde la capacidad de diagnóstico sobre un circuito a base de reducir progresivamente la exhaustividad del diccionario muestran que ésta no se pierde instantáneamente. Primero perdemos la capacidad de localizar el biestable exacto, y posteriormente se van encontrando más dificultades para determinar el registro donde se localiza el SEU, siendo la capacidad de diagnóstico temporal lo último que se pierde. Es en este punto de pérdida total donde deja de tener potencial de diagnóstico completo la técnica iterativa, ya que hasta entonces, existirán campañas que mejoren el resultado.

El hecho de que el primer intento de encontar una distancia útil para diagnóstico haya resultado muy bien, nos lleva a pensar que podrían existir muchas otras métricas aún por explorar y que funcionen incluso mejor. Podemos concluir la investigación afirmando que estas técnicas tienen capacidad de diagnóstico con diccionarios de fallos incompletos; especialmente la técnica de campañas iterativas, la cual mantiene capacidad de diagnóstico incluso con diccionarios de fallos muy incompletos.

8.2 Trabajos futuros

Las mejoras más inmediatas que se podrían realizar sobre la técnica son optimizaciones computacionales. El código esta realizado completamente en Python, por ser un lenguaje de programación con el que se hacen scripts muy rápidamente. Traducir el lenguaje a código C reduciría significativamente el coste computacional y los tiempos de cómputo.

Para optimizar el uso de la memoria, se podría reemplazar la forma de almacenar los daños de cada entrada del diccionario por tablas dispersas.

Como trabajo futuro más inmediato, se podría aplicar la técnica a circuitos de mayor extensión. Además sería interesante aplicar la técnica en circuitos tras realizarles tests de radiación.

Por último, se podrían investigar otras formas de explotar las similitudes existentes entre patrones cercanos.

8.2.1 Otras técnicas de tratamiento de imágenes

El campo de la percepción por ordenador está muy desarrollado actualmente y avanza muy rápido. Sería interesante explorar qué otras opciones existen para diagnosticar basándonos en el tratamiento de imágenes.

Además, el deep learning está experimentando grandes avances que podrían ser aplicados para entrenar una red neuroal capaz de localizar un SEU del circuito para la que se le ha entrenado.

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

Una posible mejora del diagnóstico, basada en la topología del circuito, puede ser la que hemos llamado *"Distancia en FF"*.

La distancia en FF de un biestable a otro sería el número mínimo de ciclos de reloj que tomaría propagar un SEU de uno a otro. Esta distancia puede verse afectada por la dirección:

$$D(A-B) \neq D(B-A)$$

Calculamos que esta distancia puede ser útil para afinar el diagnóstico una vez ya se haya acotado bastant al SEU.

Para calcular la distancia en FF habría que contar cuántos elementos de lógica combinacional separan dos biestables. Esto podría realizarse empleando la suite de Yosys por ejemplo [28].

Índice de Figuras

Índice de Tablas

Índice de Salidas

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] Wikipedia, "Momentos de imagen — wikipedia, la enciclopedia libre," 2020, [Internet; descargado 18-junio-2020]. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Momentos_de_imagen&oldid=124767713

[28] C. Wolf, J. Glaser, and J. Kepler, "Yosys-a free verilog synthesis suite," in *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, 2013.

[29] 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.

[30] 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.

[31] "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, 25

ESA European Space Agency. 7

FF flip-flop. 5–7, 9, 12, 13, 16, 17, 21, 22, 25, 27, 28, 32, 36

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, 15–17, 19, 25–27, 30, 33, 35, 36

TMR Triple Modular Redundancy. 1

UART Universal Asynchronous Receiver and Transmitter. 12, 18