"El saber de mis hijos hará mi grandeza"

Universidad de Sonora

División de Ciencias Exactas y Naturales

Departamento de Investigación en Física

Doctorado en Ciencias en Electrónica

Diagnóstico de fallas del tipo Stuck-at en interconexiones para circuitos de pruebas implementados en un FPGA aplicando redes neuronales.

T E S I S

Presentada en cumplimiento de los requisitos para obtener el grado de
Maestría en Ciencias en Electrónica
presenta:

Yaikel Portales González

Directores: **Dr.Roberto Gómez Fuentes**
Dr.José Rafael Benito Noriega

Asesora: **Dr.Alicia Vera Marquina**

Hermosillo, Sonora, México, 1 de agosto de 2023

Índice general

Lista de Figuras

Lista de Tablas

Introducción

Con el pasar de los años el aumento significativo del número de transistores en el área del chip ha permitido el incremento de las funcionalidades de un *circuito integrado* (CI). Esto ha provocado la disminución de sus geometrías, extensión del área del chip, el aumento del número de pines de conexión entre otros. Como consecuencia, han aumentado la complejidad los métodos de detección y diagnóstico de fallas, los cuales son de vital importancia para garantizar el buen funcionamiento del CI y su salida al mercado. Anteriormente, cuando los circuitos integrados eran relativamente menos complejos se hacia más fácil la aplicación de estos métodos de detección, por ejemplo un contador binario de 4 bits puede probarse exhaustivamente con $2^4 = 16$ *vectores de prueba* [1]. Sin embargo, para un CI digital con 32 entradas que solo tiene una complejidad de diseño modesta, por estándares de integración (VLSI) se requieren $2^{32} = 4{,}294{,}967{,}296$ *vectores* para pruebas funcionales, estos son aplicado a razón de 10^6 vectores por segundo, tomará 71.58 minutos para probar un solo CI [1]. Debido a la necesidad de disminuir el tiempo de ejecución de prueba y aumentar la efectividad, ha sido necesario desarrollar algoritmos de detección de fallas que incorporen herramientas CAD, software de cálculo matemático e incluso algoritmos de inteligencia artificial, tales como las Redes Neuronales las cuales han venido aportando soluciones a múltiples problemas, además de su capacidad de construir, de manera eficiente, vectores de pruebas para el modelo de fallas Stuck-at. Estos algoritmos neuronales necesitan, para ser implementados, de dispositivos embebidos con grandes potencias de cálculos tales como DSPs y FPGAs [7] [8]. Precisamente éste proyecto plantea la implementación de un algoritmo de detección de fallas del tipo Stuck-at en circuitos digitales, basado en la aplicación de Redes Neuronales Artificiales implementadas en una Rasberry.

Agradecimientos

Quiero agradecer primeramente a mi Señor Jesús por su bondad y misericordia, por haberme dado la fuerza para terminar, de forma cabal, mi proceso en la Mestría. Por haberme dado sabiduría y discernimiento a la hora de cumplir mis metas, esta es tu victoria Padre y no la mía. Quisiera agradecer a la Universidad de Sonora, al Comité Académico por aceptarme en el programa de posgrado y a todos mis profesores por los cuales siento mucha admiración. Quiero reconocer de forma especial la guía y ayuda del Dr.Roberto Gómez Fuentes, el Dr.José Rafael Benito Noriega Luna y Dr.Dainet Berman Mendoza.

Dedicatoria

A mis padres por su apoyo incondicional, por sus maravillosos concejos y por confiar siempre en mí, a mi esposa por su inmenso amor y comprensión. También a mis Pastores, hermanos en Cristo y todos las amistades que siempre me tuvieron en sus oraciones. En general a todos lo que siempre creyeron en mí.

Capítulo 1

Introducción

Diversos defectos en la manufactura de los circuitos integrados pueden afectar su rendimiento, y en el peor de los casos su correcta funcionalidad. Es por ello que es posible afirmar que el **Testing** es actualmente un componente esencial tanto para el diseño como para la fabricación de circuitos integrados CIs [1] pues en dependencia de la naturaleza del defecto y la funcionalidad del CI es el tipo de Test que se realiza. En la actualidad existen numerosos algoritmos y metodologías, que han evolucionado con el transcurrir de los años, para potenciar las tareas de detección y diagnóstico de fallas y de esta forma garantizar la confiabilidad y el buen funcionamiento del circuito. Estas herramientas se basan en modelos de fallas, encontrándose como uno de los principales el modelo **Stuck-at**. Con el modelado de fallas se logra que los defectos físicos puedan ser analizados desde otro punto de vista, ya sea a nivel eléctrico, lógico o funcional, disminuyendo considerablemente la complejidad de análisis. En este capítulo se mencionarán los diferentes tipos de defectos que más afectan el proceso de fabricación, así como la relación defecto-falla. Además se exponen los niveles de Testing y los modelos más empleados durante el proceso de detección y diagnóstico de falla, destacando el modelo **Stuck-at**.

Hipótesis

La implementación de circuitos de pruebas estándar en un FPGA y la programación de un algoritmo de Redes Neuronales en una tarjeta microcomputadora diagnosticaría fallas del tipo Stuck-at en circuitos VLSI.

Justificación

El desarrollo de la tecnología de los circuitos electrónicos conlleva a un aumento en varios ordenes tanto de su rendimiento como de la complejidad. Un sistema en la actualidad puede constar de varios módulos, y cada uno de ellos contiene muchas placas de circuito impreso. Una placa de circuito impreso consiste en muchos circuitos integrados de aplicación específica y dispositivos de memoria. Cada dispositivo, a su vez, consta de cientos de entradas/salidas (E/S), millones de compuertas lógicas y varias decenas de millones de bits de memoria embebida. Además, las frecuencias operativas de los circuitos de alta velocidad de procesamiento se pueden encontrar por encima de 1 GHz, mientras que las tasas de transmisión de datos de las E/S de alta velocidad pueden alcanzar los 6 Gb/s [9]. A medida que aumenta la complejidad y aumenta la velocidad, las tasas de defectos por millones aumentan y las fallas funcionales sutiles son cada vez más difíciles de detectar y diagnosticar. Con la creación de una interfaz electrónica que nos permita comprobar, registrar y clasificar el tipo de falla en un determinado CI de forma rápida, eficiente y confiable, estaríamos contribuyendo al mejoramiento del diseño de estos CI en el mundo industrial.

Objetivo General

Implementar un algoritmo para la detección de fallas tipo Stuck-at empleando redes neuronales. El cual se integrará en una tarjeta Raspberry PI 4B para analizar circuitos lógicos combinacionales de pruebas ISCAS'85 implementados en un FPGA .

Objetivos Específicos

- Definir la metodología de prueba más conveniente y la aplicación de la redes neuronales al diagnóstico de fallas a partir del estudio de la bibliografía y su discusión.
- Implementar la metodología de la detección de fallas para circuitos convencionales de pruebas estándar.
- Simular circuitos con fallas del tipo Stuck-at en el FPGA mediante la programación en Verilog.
- Entrenar y validar la red neuronal en el software Scilab.
- Implementación de las Redes Neuronales en la Raspberry Pi para su funcionamiento.
- Obtener un modelo del circuito C17, C432 que permita la simulación de fallas en lenguaje Verilog
- Recabar y procesar los datos para el entrenamiento de la red.

Metas

- Obtener un modelo basado en redes neuronales para el diagnóstico de fallas en circuitos digitales.
- Crear un esquema de prueba con la Raspberry PI y dispositivos lógicos para la comprobación del algoritmo de la red Neuronal en la realización de la prueba.
- Programar en el FPGA Cyclon IV los circuitos ISCAS C17 y C432 para comprobar el funcionamiento del diagnóstico de falla.
- Publicar resultados obtenidos en el proyecto de tesis para que puedan servir de bases a investigaciones futuras en el tema.

1.1. Defectos en circuitos lógicos

Los defectos se pueden clasificar como locales o globales [1]. Esta última clasificación se refiere a las perturbaciones que afectan regiones completas de una oblea, mientras que el local se refiere a perturbaciones regionales aleatorias dentro de un CI. Bajo la clasificación global se encuentra los defectos como grabado excesivo (inferior), desalineaciones de máscara, falta de uniformidad de las dimensiones críticas, cambio de dopantes, etc. La clase local se ocupa principalmente de los defectos puntuales que son alteraciones locales de la estructura en la capa de silicio causada por el polvo, las variaciones del proceso, etc.

Los defectos mencionados anteriormente suelen ocurrir con cierta frecuencia en la fabricación de circuitos integrados [10]. En la Tabla 1.1 y la gráfica mostrada en la Figura 1.1 se presentan los porcentajes de ocurrencia:

Tipos	Defectos
Tipo 1	Burbuja redonda o de forma alargada
Tipo 2	Partículas grandes
Tipo 3	Copos
Tipo 4	Defectos de forma
Tipo 5	Defectos largos
Tipo 6	Residuos resistivos
Tipo 7	Pista de aterrizaje de caída
Tipo 8	Irregularidades pocos profundas
Tipo 9	Grandes partículas porosas
Tipo 10	Formas irregulares
Tipo 11	Partículas muy pequeñas encima del metal

Tabla 1.1: Tipos de defectos [1].

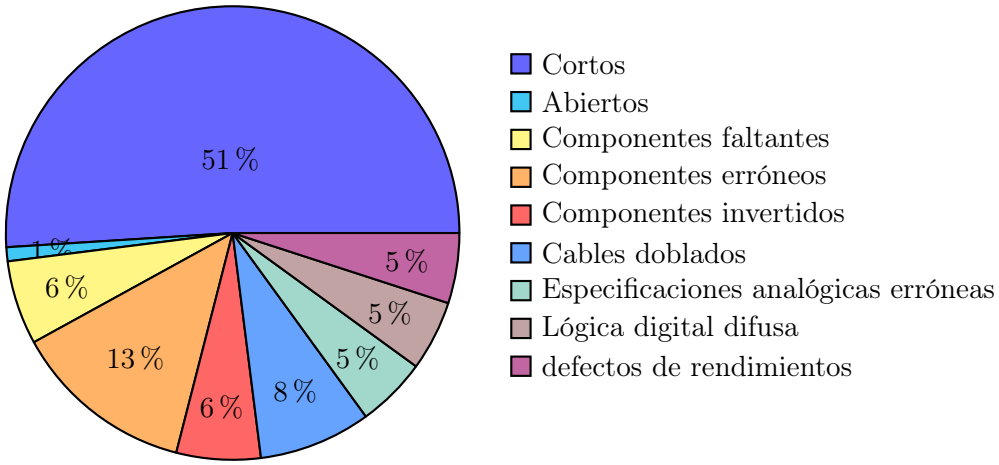


Figura 1.1: Porcentaje de defectos según la ocurrencia [1].

1.1.1. Relación Error-Defecto-Falla

El error es una acción que produce un resultado incorrecto, el defecto es la manifestación de ese error en el chip y la falla es la revelación eventual del defecto al comprobar su funcionalidad. Debemos tener en cuenta que entre defecto y falla existe una similitud pero la diferencia está en que un fallo es la operación anómala de un circuito integrado cuya causa puede tener origen en un defecto. Con la búsqueda cada vez mayor de más funciones en un solo CI ha llevado a la contracción de las geometrías de los dispositivos y el aumento del área del mismo. Desafortunadamente, ambos desarrollos han causado que los circuitos integrados se vuelvan susceptibles a diversos mecanismos de pérdida de rendimiento [1]. Por tanto, el rendimiento de un CI es un factor muy importante que determina si una mayor integración es o no económicamente una buena proposición. De ahí que se haya vuelto cada vez más relevante conocer diferentes mecanismos de pérdida de rendimiento que están muy vinculado con ciertos factores que producen errores durante el proceso de fabricación realizados en un lote de obleas [11]. El resultado de una operación de fabricación depende de tres factores principales: los parámetros de control del proceso, el diseño de la CI, y algunos factores ambientales que cambian aleatoriamente, llamados perturbaciones. El control de una operación de manufactura es el conjunto de parámetros que deben ser manipulado para los cambios deseados en la estructura del CI fabricado. El diseño de un CI es el conjunto de máscaras que distinguen las áreas del CI que necesitan ser procesados para cada paso de fabricación. Las perturbaciones son factores ambientales que influyen en el resultado de la operación de fabricación y por supuesto son las causantes de un alto por ciento del error de producción y a su vez de los defectos antes mencionados. Las perturbaciones del

proceso de fabricación se han estudiado con gran detalle y se clasifican como:

- Errores humanos y fallas de equipos.
- Inestabilidades en las condiciones del proceso.
- Inestabilidades materiales.
- Inhomogeneidades del sustrato o manchas de litografía.

1.2. Modelado de fallas Lógicas

El modelado de fallas permite que los defectos físicos, efectos del test, puedan ser representados en un nivel más alto de abstracción, dígase a nivel eléctrico, lógico o funcional; es así como la generación de los vectores de pruebas depende del modelo de fallas que halla sido seleccionado. Con este tipo de análisis se logra transformar el problema físico en un problema lógico, esto reduce enormemente la complejidad del análisis [12]. Los modelos de fallas lógicas son independientes de la tecnología, esto quiere decir que un mismo modelo puede ser aplicable en diferentes tecnologías, aunque éstas evolucionen.

Las pruebas derivadas de *fallas lógicas* se pueden usar para fallas físicas cuyo efecto en el comportamiento del circuito no se entiende completamente o es demasiado complejo para ser analizado. Los modelos lógicos de fallas pueden ser *explícitos* o *implícitos*. Un modelo explícito es aquel donde las fallas son detectadas de forma individual y por ende pueden ser analizadas de forma explícita [1] [12]. Este modelo es práctico en la medida en que el tamaño del universo de fallas no es prohibitivamente grande. El modelo implícito define un universo de fallas que son identificadas colectivamente permitiendo la identificación de sus propiedades características. Por tanto, si contamos con un modelo del sistema y una falla lógica, en principio podemos determinar la función lógica del sistema en presencia de una falla.

Las fallas definidas que guardan relación con el modelo estructural son conocidas como fallas estructurales, su efecto modifica las interconexiones entre componentes. Las fallas relacionadas con el modelo funcional pueden provocar un cambio en el funcionamiento del dispositivo por ejemplo en su la tabla de verdad. Por otro lado aunque las fallas intermitentes y transitorias ocurren con frecuencia, su modelado requiere datos estáticos sobre su probabilidad de ocurrencia. Estos datos son necesario para determinar cuántas veces debería de repetirse una prueba experimental sobre una línea desconectada para maximizar la probabilidad de detección de fallas que algunas veces se presentan en el circuito bajo prueba. Para un mejor análisis siempre se asume, a menos que no se especifique lo contrario, que tenemos solo una falla lógica en el sistema. Esta suposición se justifica por una estrategia de frecuencia de prueba que establece que debemos probar un sistema varias veces para que la probabilidad de que se desarrolle más de una falla entre dos experimentos de prueba consecutivos sea lo suficientemente pequeña. Por lo tanto, si los intervalos de mantenimiento de un sistema en funcionamiento son demasiado largos, es probable que encontremos múltiples fallas [12].

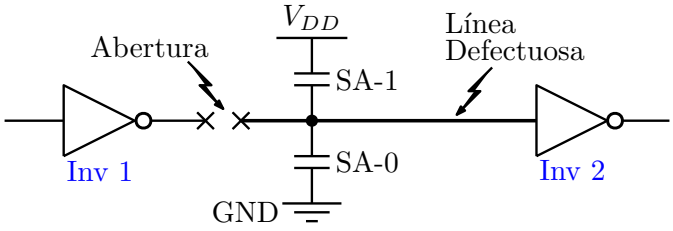


Figura 1.2: Representación de fallas tipo Stuck-at y Stuck open simple.

Si la comprobación no detecta todas las fallas simples en el circuito entonces puede existir una falla indetectable en un momento determinado lo que puede ocasionar, con la ocurrencia de una segunda entre el periodo de ejecución de dos pruebas, fallas múltiples. Pero aún cuando estamos en presencia de fallas múltiples, el método de detección derivado de la suposición de falla simple es empleado para detectarlas pues en la mayoría de los casos, una falla múltiple puede ser detectada por las pruebas diseñadas para las fallas individuales [12]. En general, el modelo de falla estructural asume que los componentes están libres de fallas y que solamente sus interconexiones son las afectadas [13]. Los tipos de fallas que afectan las interconexiones pueden ser cortos o aperturas. Un corto es formado cuando se conectan puntos no destinados a ser conectados, mientras que una apertura es el resultado del rompimiento de una conexión. Por ejemplo, en muchas tecnología, un corto a tierra o a Vcc en una línea, puede ocasionar que la señal permanezca con un nivel de tensión fijo. La falla lógica correspondiente consiste en que la señal está siendo atascada (Stuck) a un nivel lógico V ($v = \{0; 1\}$), y está denotada por S-a-v [12] [14] ver Figura 1.2. Un corto entre dos señales usualmente crean una nueva función. La falla lógica que representa este tipo de evento es conocida como puente de falla (BRIDGING FAULT).

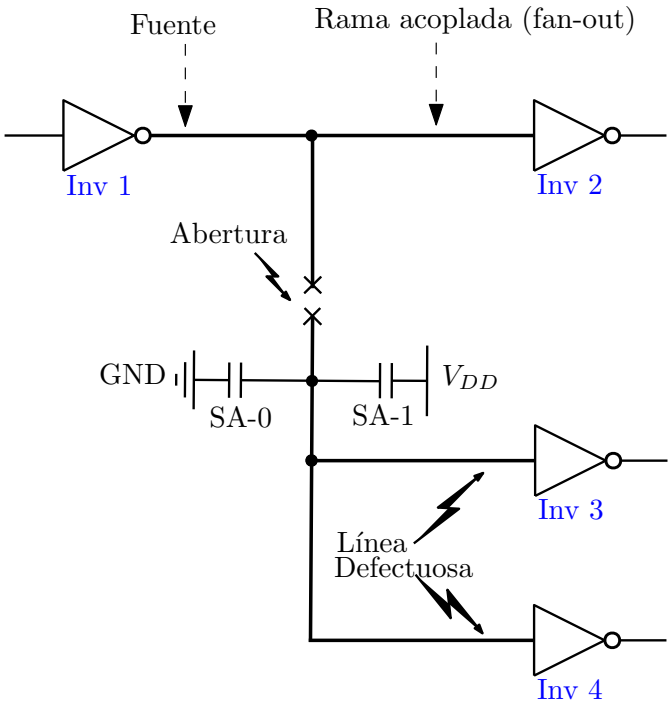


Figura 1.3: Representación de fallas tipo Stuck-at y Stuck open múltiples.

En muchas tecnologías, el efecto de una apertura sobre una línea de señal unidireccional con único fan-out provoca una desconexión en la entrada, del otro elemento conectado, permitiendo que dicha entrada asuma un valor lógico constante y por tanto se comporte como una falla Stuck Figura 1.2. Este efecto puede también resultar de una falla física interna de un componente conectado en la misma línea o a una fuente de alimentación (S-a-v), para este caso si no podemos comprobar los extremos, no podremos discernir entre los dos casos. Esta distinción no será necesaria, con este tipo de modelo pues podemos asumir simplemente que la línea está atascada.

Una apertura en una línea con una rama fan-out puede causar una múltiple falla tipos Stuck en las subramas de las conexiones Figura 1.3. Si emplearemos el modelo de fallas Stuck simples, entonces tenemos que considerar cualquier falla Stuck simple de la rama de ramificación por separado.

1.2.1. Modelo de fallas Stuck-at

El modelo de falla Stuck-at es el más ampliamente usado pues se abstrae de la implementación y los detalles tecnológicos de la representación de un circuito colocando la ocurrencia de falla directamente dentro de la representación del nivel de compuerta del circuito. Este fallo es modelado asignando un valor fijo (0 ó 1) a la línea de señal en el circuito que puede ser una entrada o salida de una compuerta lógica o un flip-flop. Este modelo asume que un nodo defectuoso se comporta como un nodo permanentemente conectado a una de las fuentes de voltajes, ya sea VDD o GND. En este modelo, SA0 (Stuck-at-0) y SA1 (Stuck-at-1) son usados para describir un nodo que exhibe una falla. A el nivel de compuerta, el número de fallas que pueden ocurrir para una compuerta combinacional con n entradas y 1 salida es 2n + 2. Cada nodo de las n entradas puede sufrir una falla SA0 ó SA1 [14], lo mismo es aplicable para los nodos de salida. Para facilitar el análisis lógico y funcional hemos considerado simular la ocurrencia de fallas simples en un solo instante de tiempo. El análisis para las fallas Stuck-at simultáneas es mucho más complejo, requiere considerar otros factores que aumenta la complejidad del análisis del modelo. El set de vectores es aplicado a las entradas primarias del circuito para someter la interconexión a un estímulo que permita revelar la existencia de una falla y el error sea propagado a la salida primaria. Un circuito con $\boldsymbol{n}$ líneas tendría $3\boldsymbol{n}$-1 estados posibles Stuck-at, el cual, es un número alto y computacionalmente caro. Por consiguiente, esto es común para modelar solo una falla Stuck-at al mismo tiempo (no múltiples fallas). De este modo un circuito con $\boldsymbol{n}$ líneas tendrá 2n fallas Stuck-at. Este número está reducido por el proceso de compresión debido a que existen fallas equivalentes. Algunas de las características de este modelo se describen a continuación:

- Muchos defectos físicos diferentes pueden ser modelados por la misma lógica.
- La complejidad se reduce grandemente con el análisis lógico.
- En comparación con otros modelos, la cantidad de fallas simples Stuck-at en un circuito es pequeña. Además el número de fallas a analizar explícitamente se puede reducir mediante técnicas de colapso de fallas.
- Es independiente de la tecnología, ya que el concepto de una línea de señal atascada en un valor lógico se puede aplicar a cualquier modelo estructural.

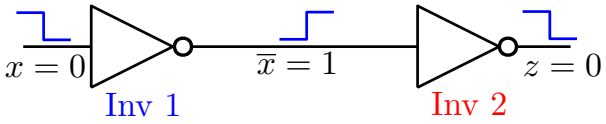


Figura 1.4: Circuito combinacional a probar.

La Figura 1.4 es un circuito combinacional formado por dos inversores conectados en serie donde la señal de entrada la llamamos $\boldsymbol{x}$ y la salida $\boldsymbol{z}$. Si aplicamos el álgebra de Boole podemos obtener la función lógica de salida muy sencilla que llamaremos f_z. En condiciones normales las señales de entrada y salida serán iguales o sea:

$$f_z = x \tag{1.1}$$

Esta función responde a la Tabla de la verdad 1.2:

En la Figura 1.5 se ha simulado una falla S-a-0 a la entrada del inversor 2, esto afecta la funcionalidad del circuito pues ya no existe la igualdad proporcionada por la ecuación 1.1 debido a que no se cumple la Tabla de la verdad 1.2. Para esta falla nuestra ecuación tiene la siguiente forma:

$$f_z = 1 \tag{1.2}$$

x	z
0	0
1	1

Tabla 1.2: Tabla de la verdad del circuito 1.4

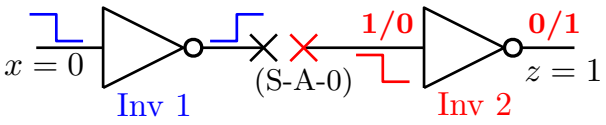


Figura 1.5: Representación de fallas tipo Stuck-at.

Esta ecuación nos dice que bajo este escenario la salida será invariante ante los cambios en la entrada x. La notación $\mathbf{0/1}$ hace referencia a la simbología $\boldsymbol{v}/v_f$, donde v es el valor lógico esperado por la función y v_f es el valor lógico causado por la falla. Si aplicamos los conceptos de modelos de fallas nos damos cuenta que en este pequeño circuito sólo pueden aplicarse 3 fallas S-a-0 y 3 S-a-1, pues tenemos que por cada compuerta se pueden generar un total de 4 fallas(2 dela entrada(n) y dos en la salida), esto responde a la ecuación $2n + 2$ donde n=1 para cada inversor, pero como están conectados en serie existen dos fallas que son redundantes por lo tanta tendremos 6 en total. A pesar de las grandes ventajas del modelo de falla Stuck-at, se ha encontrado que no es adecuado para representar algunos defectos en las tecnologías CMOS.

1.3. Test de circuitos integrados

El objetivo de aplicar el test de circuitos integrados es discernir en un conjunto de circuitos fabricados e identificar los que no satisfacen las especificaciones iniciales de diseño. La metodología para aplicar el test a circuitos integrados se muestra a continuación [2]:

1. Aplicar un vector de test previamente definido a las entradas del circuito bajo prueba. El vector debe de tener la capacidad de estimular las interconexiones defectuosas y propagar el error a una salida observable.

2. Se realiza una medición para conocer el valor lógico en la salida.

3. Se procede a comparar el valor obtenido con una referencia. Si no existen diferencias el circuito es aceptado, de lo contrario es rechazado.

En el diagrama que se ilustra en la Figura 1.6 representa un método de verificación automático para la detección de fallas en un circuito bajo prueba. El método consiste en generar una secuencia de 1s y 0s como estímulo de entrada al çircuito bajo prueba" para comparar su respuesta con la respuesta correcta y producir una señal de aprobación o rechazo, dependiendo si el circuito funciona correctamente o está defectuoso [15] [16]. A cada combinación de 1s y 0s de la secuencia se le denomina "vector de prueba".

En la mayor parte de los sistemas automáticos de detección de fallas, las respuestas correctas a cada uno de los vectores aplicados al circuito bajo prueba están almacenadas en una memoria [17]. La comparación se efectúa recuperando de la memoria la respuesta correcta a cada vector de prueba aplicado y comparando la misma con la respuesta que da el circuito bajo verificación. Los tipos de fallas que pueden encontrarse en un circuito digital son muy variados, también podemos encontrarnos que los niveles de tensión del circuito pueden ser inadecuados, interconexiones se encuentren abiertas o en cortocircuito.

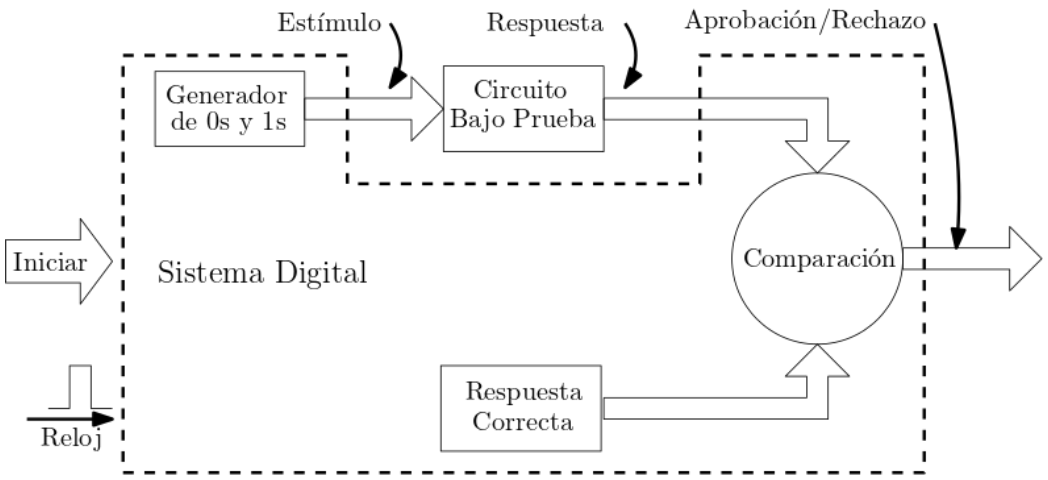


Figura 1.6: Esquema general para detección de fallas en Circuitos Integrados [2].

1.3.1. Tipos de Testing

Test lógico

El test lógico es usado para monitorear niveles lógicos (valores booleanos) del circuito bajo prueba (CUT) los cuales tienen una función lógica característica. El nodo de salida de un CUT muestra un valor lógico definido por una combinación de señales de entrada. El test lógico compara la respuesta del nodo de salida del CUT contra la respuesta esperada libre de falla del CUT. Si resulta que ambas no son iguales el CUT es defectuoso. En el test lógico, se asume que un tiempo suficiente es esperado después de que el vector es aplicado a las entradas para establecer los niveles estables.

Testing funcional

Este tipo de Testing está basado sobre el modelo funcional y es implementado fundamentalmente en circuitos menos complejos como son los SSI, circuitos de pequeña escala de integración o (Small Scale Integration), donde los métodos estructurales no son tan necesarios. Un modelo funcional refleja las especificaciones funcionales del sistema y en gran medida es independiente de su implementación. Esto permite que las pruebas funcionales derivadas puedan ser implementadas no sólo para verificar si hay fallas *físicas* presentes en el sistema fabricado, sino también como prueba de verificación de *diseño* para comprobar que la implementación está libre de errores. El objetivo del Testing funcional es validar la correcta operación de un sistema con respecto a sus especificaciones funcionales, esto tiende a tener dos enfoques: Un enfoque asume los modelos de fallas funcionales específicos e intenta generar pruebas que detecten las fallas definidas por estos modelos [11]. El otro no se preocupa por los posibles tipos de comportamiento defectuoso y trata de derivar pruebas basado únicamente en el comportamiento libre de fallas especificado. Se dice que las pruebas funcionales que detectan casi cualquier falla son *exhaustivas*, ya que deben ejercer completamente el comportamiento libre de fallas. Debido a la longitud de la resultante prueba, la prueba exhaustiva se puede aplicar en la práctica sólo a *circuitos pequeños*, pues los nodos internos son fácilmente accesibles a través de los pines de I/O y la generación de los vectores de prueba es fácil. Para circuitos combinacionales con n entradas, un test exhaustivo consiste de $2n$ vectores de prueba [18]. Para un circuito de lógica secuencial con m registros de un bit (elementos de memoria) y una relación entrada-salida en la cual las salidas dependen de las entradas y los valores de los registros, un test exhaustivo consistiría de 2^{m+n} vectores de prueba. Mediante el uso algún conocimiento sobre la estructura del circuito y estrechando ligeramente el universo de fallas garantizadas para ser detectadas, podemos obtener una prueba *pseudoexhaustiva* que pueden ser significativamente más cortos que los exhaustivos. Por esta razón, se emplearon mucho en los primeros años de la tecnología de los circuitos integrados de pequeña escala de

integración (SSI), pues la complejidad del circuito estaba limitada a compuertas sencillas [19].

Testing estructural

El Testing estructural funciona tomando como referencia el modelo estructural ya que utiliza, para el análisis, las estructuras internas del circuito. De esta forma la cantidad de vectores de prueba es reducido y se les conoces como vectores de pruebas estructurales. Con la evolución de los niveles de integración en los CI de SSI y circuitos de media escala de integración MSI a circuito de larga escala de integración (LSI) no era posible la implementación del Testing funcional debido a la costosa aplicación del patrón de vectores de pruebas [18]. Posteriormente para los circuitos LSI el concepto de Testing tuvo que ser desarrollado utilizando técnicas computacionales con el objetivo de determinar un numero mínimo de vectores necesarios para realizar una verificación estructural del CI. Con las pruebas estructurales existe la posibilidad que en un mismo recorrido (path) defectuoso no sólo pueda detectarse una falla sino varias a lo largo de ese camino. Existen variedades de algoritmos estructurales de generación automática de patrones orientadas sólo para circuitos combinacionales. Un circuito combinacional consiste de n entradas y m salidas que dependen únicamente del estado actual de las entradas en ese instante de tiempo. Dentro del circuito podemos encontrar compuertas lógicas, interconexiones definidas por ramas y nodos. El nodo se considera la el elemento mas pequeño del circuito a nivel compuerta, que puede asumir un valor lógico distinto. Las fallas son insertadas una en cada p nodo del circuito y el objetivo es encontrar un juego de p vectores que provoquen las expresiones de las fallas insertadas para que aparezcan en el vector de salida correspondiente [18]. El estado de comprobación se realiza comparando las funciones lógicas de salidas entre el circuito correcto y el circuito defectuoso tomando como referencia la tabla de verdad. Si existe deferencias entonces ese vector de pruebas es almacenado. El Testing estructural es muy efectivo sólo en circuitos combinacionales pero esa efectividad se reduce para el análisis de circuitos secuenciales [20].

1.3.2. Test de retrasos

Unos de los comportamientos incorrectos e inestables en el funcionamiento de los circuitos integrados son causados por las fallas de retrasos. Los casos de aumento de los retrasos en la propagación de la señal se modelan mediante lo que son llamadas fallas de retardo. Este se realiza por medio de retardo de compuertas y por retardo de rutas, en el modelo de falla de retardo de puerta, dos fallas de retardo, de aumento lento (STR) y caída lenta (STF), están asociados con cada entrada y salida de puerta [21]. Los modelos de fallas de ascenso lento (lento de caída) aumentan el retraso en la propagación de una transición ascendente o de 0 a 1 (transición descendente o de 1 a 0) a través de compuertas impulsadas por el lugar de la falla. Las pruebas de fallas de retardo de puerta requieren tener en cuenta el tamaño del defecto de retardo. Por ejemplo, si la dimensión del defecto en un conductor de circuito r es menor que la holgura de r, la falla puede no ser detectable por ninguna prueba. La holgura de una línea de circuito es la diferencia entre el período del reloj funcional y el retardo máximo de todos los caminos a través de r. El modelo de retraso de compuerta está basado en las especificaciones del testing de tiempo del dispositivo seleccionado. Además, el retardo de propagación de una entrada de puerta depende de los estados de las otras entradas a las compuertas y capacitancias de acoplamiento a otras líneas adyacentes. El modelo de retraso de rutas resuelve este problema [5] [18]. Una trayectoria es seleccionada para ser el objeto de medición del retraso, entonces las transiciones de 0-1 y 1-0 son propagadas a través de la trayectoria. Si el retraso medido se encuentra dentro de la ventana de observación, entonces la trayectoria esta libre de fallas, de lo contrario es defectuosa. La ventana de observación puede ser definida como cerrada para el tiempo funcional de la trayectoria en distribuciones estáticas de retrasos. Cuando el intervalo de la señal de frecuencia base es mayor que el retraso de propagación de la señal, a través de la ruta desde la entrada primaria hasta la salida primaria, se dice que el circuito se encuentra funcionando correctamente [22]. Los defectos que causan retrasos de propagación a lo largo de una o más trayectorias sea mayor que el intervalo de la señal de reloj funcional, puede resultar en la captura de valores lógicos incorrectos en registros internos o en el retraso de los valores del circuito funcional en las salidas primarias.

1.3.3. Test de *IDDQ*

El Testing de corriente de alimentación en reposo (IDDQ) es una técnica basada en la medición de la corriente en estado estable del dispositivo bajo test, es aplicable fundamentalmente a dispositivos que utilizan tecnología CMOS donde se presentan bajas corrientes de fuga. Esto se utiliza para medir la corriente de suministro del dispositivo bajo condiciones de estado estable. La idea básica detrás de las pruebas IDDQ es, cuando sus entradas son estables, los circuitos CMOS estáticos consumen potencia modesta, ya que no hay una ruta directa desde el VDD a tierra [23]. Los cortos (puentes) entre la fuente de alimentación, líneas o entre VDD y tierra y los dos nodos interruptores o una señal, son detectados por la prueba IDDQ. Un defecto pasivo aumenta la fuga para todos los patrones de entrada, mientras que un defecto activo aumenta la fuga para algunos patrones de entrada. El test de IDDQ es una técnica muy sensible, capaz de detectar problemas en estado temprano, incluso antes de que realmente dañen al circuito. Debido a esto, ofrece una ventana para el comportamiento futuro del dispositivo. También, es una propuesta alternativa para remplazar a otras técnicas, más costosas o que consuman más tiempo en la generación de vectores necesarios para garantizar la calidad y funcionalidad del dispositivo probado. En combinación con la espectroscopía de emisión y el análisis espectral, IDDQ es una técnica muy poderosa para detectar la ubicación y diagnosticar una falla [18].

1.4. Organización de los Capítulos

Este proyecto está conformado por 5 capítulos los cuales están organizados de la siguiente manera: en este primer capítulo se hizo una introducción a los diferentes tipos de defectos que provocan las fallas en los circuitos integrados. Se mencionaron los conceptos principales de los modelos de fallas destacando el Stuck-at como el más implementado en la detección y diagnóstico de fallas, así como, su implementación a través de los tipos de Testing utilizados por la industria, para garantizar el correcto funcionamiento de los CI. En el segundo capítulo se abordan conceptos fundamentales sobre la teoría de las Redes Neuronales Artificiales, el tipo de entrenamiento y los algoritmos matemáticos aplicados en nuestro proyecto, como es el caso del *gradiente en descenso* implícito en el Backpropagation que es el algoritmo empleado para entrenar la red neuronal artifical (ANN). El tercer capítulo se brinda detalles sobre la elaboración, organización y estructuración de la información que se emplearán en el entrenamiento y validación del algoritmo neuronal, tomando como base los circuitos ISCAS C17 y C432. Por otra parte se explica cómo se realiza el proceso de generación y transmisión de esta información en el hardware implementado. En el capítulo 4 se muestra el entrenamiento de la red en el software Scilab y su validación tanto para el circuito C17 como para el C432. Luego se demuestra cómo fue programado este algoritmo en el miniordenador Raspberry Pi 4B y como se establecieron las interfaces de conexión con el dispositivo bajo prueba, implementado en el FPGA cyclon IV. Mientras que el capítulo 5 se comparten los resultados de los diferentes aspectos que se necesitaron para concluir este proyecto, por ejemplo los resultados obtenidos del entrenamiento para el circuito C17 y C432. La validación e implementación de este algoritmo en el hardware propuesto. Finalmente, se concluye el trabajo de tesis en el capítulo 6 con las conclusiones y recomendaciones para trabajos futuros que puedan realizarse para mejorar aún más este proyecto.

Capítulo 2

Redes Neuronales

2.1. Introducción

Una red neuronal artificial (ANN) es un tipo de proceso de *Deep Learning* llamado aprendizaje profundo que funciona muy parecido al cerebro humano [24]. La ANN está compuesta por pequeñas unidades de procesamiento denominadas *neuronas*, que interconectadas entre sí, conforman estructuras funcionales llamadas *capas*. En estas interconexiones, denominadas *pesos sinápticos*, es donde se almacena gran parte de la información, fundamentalmente la relación existente entre los datos de entrada y salida. El aprendizaje de la red se realiza mediante un proceso de entrenamiento en el cual la información contenida en los pesos sinápticos es modificada teniendo en cuenta el error del entrenamiento y la razón de aprendizaje [25]. Las redes neuronales son muy importantes dentro de la inteligencia artificial y se emplean en casi todos los campos de la ciencia, fundamentalmente en aplicaciones donde se desee realizar una clasificación importante de datos cuya información de entrada y salida es prácticamente desconocida [26]. De esta manera, el modelo permite predecir cual es el valor de salida dado un valor de entrada no especificado en la etapa de entrenamiento. Las capacidades de la red van a depender en gran medida de la fase de entrenamiento a la que haya sido sometida [9].

2.2. Estructura de una red neuronal

Las *redes neuronales artificiales* están basadas en el funcionamiento de las *redes de neuronas biológicas*. Las neuronas que todos tenemos en nuestro cerebro están compuestas de dendritas, el soma y el axón, las dendritas se encargan de captar los impulsos nerviosos que emiten otras neuronas. Estos impulsos, se procesan en el soma y se transmiten a través del axón que emite un impulso nervioso hacia las neuronas contiguas.

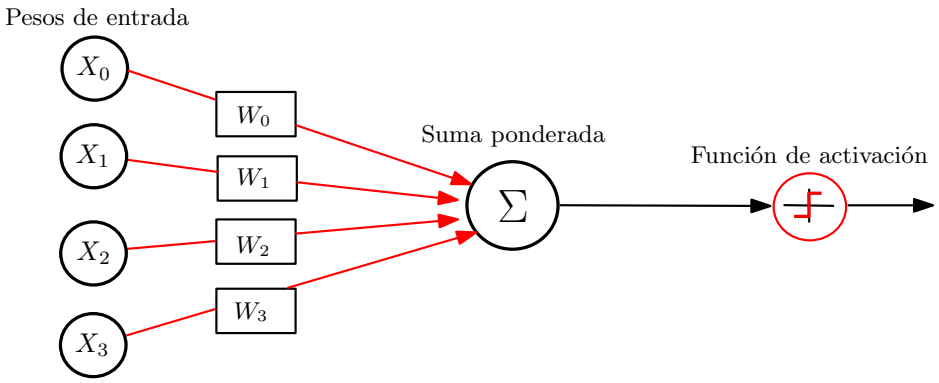


Figura 2.1: Diagrama del Perceptrón [3].

En el caso de las neuronas artificiales, la suma de las entradas multiplicadas por sus pesos asociados determina el impulso nervioso que recibe la neurona. Este valor se procesa en el interior de la célula mediante una función de activación que devuelve un valor que puede ser analógico o digital y constituye la salida de la neurona, esta estructura se conoce como *Perceptrón* y fue creada por Frank Rosenblatt en 1943 ver Figura 2.1. Del mismo modo que, nuestro cerebro está compuesto por neuronas interconectadas entre sí, una red neuronal artificial está formada por neuronas artificiales conectadas entre sí y agrupadas en diferentes niveles que denominamos **capas**, Figura 2.2. Una capa es un conjunto de neuronas cuyas entradas provienen de una capa anterior (o de los datos de entrada en el caso de la primera capa) y cuyas salidas son la entrada de una capa posterior [26].

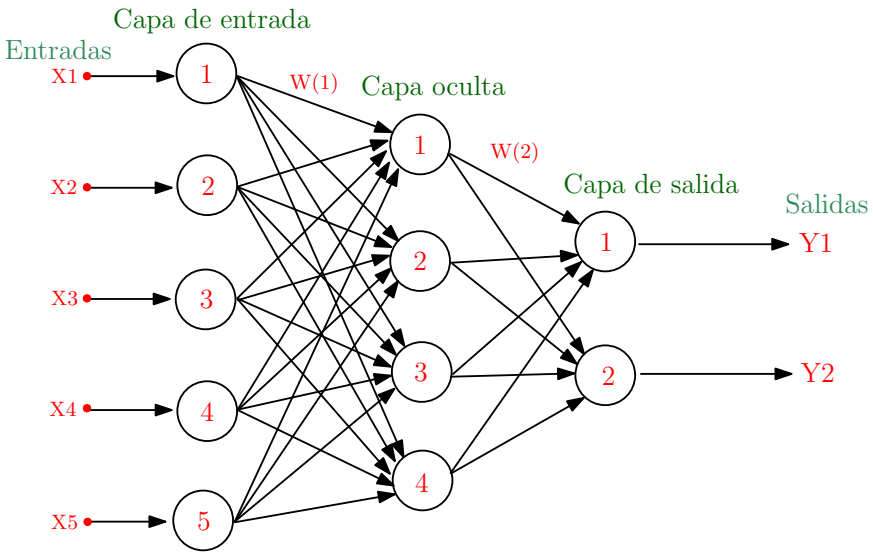


Figura 2.2: Diagrama de capas de la Red Neuronal.

Las neuronas de la primera capa reciben como entrada los datos reales que alimentan a la red neuronal. Es por eso, por lo que la primera capa se conoce como capa de entrada. La salida de la última capa es el resultado visible de la red, por lo que la última capa se conoce como la capa de salida. Las capas que se sitúan entre la capa de entrada y la capa de salida se conocen como capas ocultas ya que desconocemos tanto los valores de entrada como los de salida [27]. Una red neuronal, por lo tanto, siempre está compuesta por una capa de entrada, una capa de salida (si sólo hay una capa en la red neuronal, la capa de entrada coincide con la capa de salida) y puede contener 0 o más capas ocultas. El concepto de Deep Learning nace a raíz de utilizar un gran número de capas ocultas en las redes [28].

2.2.1. Tipos de Neuronas Artificiales

Las neuronas artificiales se pueden clasificar de acuerdo a los valores que pueden adquirir. Generalmente encontramos dos tipos principales:

- Neuronas binarias.
- Neuronas reales.

Las neuronas binarias solamente pueden tomar valores dentro del intervalo 0, 1 asociado a la salida de la función sigmoide o -1, 1 para la función tangente hiperbólica, mientras que las neuronas reales pueden hacerlo dentro del rango[0, 1] o [-1, 1]. Los pesos normalmente no están restringidos a un cierto intervalo,aunque para aplicaciones específicas puede ser esto necesario [27].

2.3. Entrenamiento de una red Neuronal

Entrenar una red neuronal consiste en ajustar cada uno de los pesos de las entradas de todas las neuronas que forman parte de la red neuronal para que las respuestas de la capa de salida se ajusten lo más posible a los datos que conocemos. Para lograr un entrenamiento exitoso es de vital importancia que nuestros datos se encuentre organizados correctamente para evitar problemas indeseados como son las correlaciones de datos [29]. Otro factor importante es tener definida la estructura de red aquí se incluye la cantidad de neuronas por capas, la función de activación en cada capa, la cantidad de capas oculta que tendrá la red y por último la capa de salida, en base en la tipología escogida se asignan valores iniciales a cada uno de los parámetros que conforma la red [25]. La experiencia del entrenador es otro punto que influye mucho, pues existen determinados parámetros que son definido de manera empírica, tales como la razón de entrenamiento, la cantidad de neuronas que tendrá cada capa, el número de capaz ocultas entre otros, pues no existen modelos específicos para calcular de forma exacta esta información. Para el entrenar una red también debemos definir la cantidad de iteraciones o número de épocas y el error de entrenamiento. En este punto se partirá de pesos sinápticos generados pseudoaleatoriamente, el proceso de aprendizaje se encargará entonces de modificar iterativamente estos pesos hasta encontrar el conjunto capaz de desarrollar la tarea programada con el menor error posible [9]. El tiempo del entrenamiento estará influenciado por la cantidad de información, la complejidad de la red, la cantidad de iteraciones, el error restablecido y las limitaciones computacionales que tenga el sistema. Después que la red se entrene debemos de comprobar su efectividad mediante un proceso conocido como *validación de datos*, donde se comprueban los resultados con datos que no formaron parte del conjunto de entrenamiento.

2.3.1. Tipos de aprendizajes

El aprendizaje es el proceso por el cual una red neuronal modifica sus pesos en respuesta a una información de entrada. Los cambios que se producen durante el mismo se reducen a la destrucción, modificación y creación de conexiones entre las neuronas. En los modelos de redes neuronales artificiales, la creación de una nueva conexión implica que el peso de la misma pasa a tener un valor distinto de cero. De la misma manera, una conexión se destruye cuando su peso pasa a ser cero. Durante el proceso de aprendizaje, los pesos de las conexiones de la red sufren modificaciones, por lo tanto, se puede afirmar que este proceso ha terminado (la red ha aprendido) cuando los valores de los pesos permanecen estables. Un aspecto importante respecto al aprendizaje de las redes neuronales es el conocer cómo se modifican los valores de los pesos, es decir, cuáles son los criterios que se siguen para cambiar el valor asignado a las conexiones cuando se pretende que la red aprenda una nueva información [27]. Otro criterio que se puede utilizar para diferenciar las reglas de aprendizaje se basa en considerar si la red puede aprender durante su funcionamiento habitual o si el aprendizaje supone la desconexión de la red, es decir, su inhabilitación hasta que el proceso termine. En el primer caso, se trataría de un aprendizaje *on line*, mientras que el segundo es lo que se conoce como *off line*. Debido precisamente a su carácter estático, estos sistemas no presentan problemas de estabilidad en su funcionamiento. Existen tres grupos de aprendizajes:

- Aprendizaje supervisado: se caracteriza porque el proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo que determina la respuesta que debería generar la red a partir de una entrada determinada. El supervisor controla la salida de la red y en caso de que ésta no coincida con la deseada, se procederá a modificar los pesos de las conexiones, con el fin de conseguir que la salida obtenida se aproxime a la deseada.

- Aprendizaje no supervisado: estas redes no requieren influencia externa para ajustar los pesos de las conexiones entre sus neuronas. La red no recibe ninguna información por parte del entorno que le indique si la salida generada en respuesta a una determinada entrada es o no correcta. Además deben encontrar las características, regularidades, correlaciones o categorías que se puedan establecer entre los datos que se presenten en su entrada. Existen varias posibilidades en cuanto a la interpretación de la salida de estas redes, que dependen de su estructura y del algoritmo de aprendizaje empleado [27].

- Aprendizaje por esfuerzo. Se presenta a la red un conjunto de datos de entrada nuevamente sin información de la salida esperada, pero es indicada si la salida obtenida es correcta o no.

2.4. Algoritmo del Gradiente en Descenso (∇f)

El gradiente en descenso o gradiente descendente (**GD**) es un algoritmo de optimización genérico que hemos utilizado para desarrollar nuestro entrenamiento, éste es capaz de encontrar soluciones óptimas para una amplia gama de problemas. La idea del gradiente descendente es ajustar los parámetros de forma iterativa para minimizar una función. Es uno de los algoritmos de optimización más populares en aprendizaje automático, particularmente por su uso extensivo en el campo de las redes neuronales, pues es un método general de minimización para cualquier función [30] [31].
Concretamente, se tiene de una función diferenciable convexa $f : \Omega \subset \mathbb{R}^n \to \mathbb{R}$ el algoritmo GD permite encontrar un w en Ω tal que $f(w)$ es un mínimo, en otras palabras, GD se utiliza para determinar los elementos del siguiente conjunto:

$$w \in argmin f(w) \to (w \in \Omega) \tag{2.1}$$

Para determinar los valores de w que optimiza la función $f(w)$, GD hace uso de una serie de iteraciones que se hacen de acuerdo con la siguiente regla de actualización:

$$w_{t+1} = w_t - \eta_t \nabla f(w_t) \tag{2.2}$$

que usualmente se inicializa en cero y cada iteración, como se puede observar, se hace en la dirección negativa del gradiente. Recuerde que el gradiente se define como el vector:

$$\nabla f(w) = \left(\frac{\partial f}{\partial x_1}(w), \ldots, \frac{\partial f}{\partial x_n}(w) \right) \tag{2.3}$$

Un hiperparámetro importante en GD es el $\eta_t > 0$,denominado como tasa de aprendizaje. Si la tasa de aprendizaje es demasiado pequeña, entonces el algoritmo tendrá que pasar por muchas iteraciones para converger, lo que llevará mucho tiempo. Por otro lado, si la tasa de aprendizaje es demasiado alta, es posible que se salte el mínimo global y termine en otro lado, posiblemente incluso más alto que antes. Esto podría hacer que el algoritmo diverja, con valores cada vez mayores, sin encontrar una buena solución.

2.5. Algoritmo de Retropropagación (Backpropagation)

Este algoritmos esta basado en la regla de aprendizaje correlación-error. Se basa en dos etapas fundamentalmente, una es la propagación de la señal de entrada hacia delante conocido como (Forward Pass) y la otra es la propagación hacia atrás (Backward pass) del error producido por la señal a la salida. En el Forward Pass se aplica una señal a la entrada (vector de entrada) y el efecto es propagado capa por capa donde finalmente una respuesta es producida a la salida de la red. Durante el Forward Pass el valor de los pesos sinápticos se mantiene fijo. Por otra parte, en el Backward pass, todos los valores de los pesos son reajustados en dependencia de una regla de correlación-error, esto quiere decir que la respuesta de salida de la red es restada a la respuesta deseada para producir una señal de error. Esta señal de error se transmite a través de la red en dirección contraria a la conexión de los pesos sinápticos, este error es conocido como retropropagación de error o (error Back-propagation) [25]. El proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total.
Basándose en la señal de error percibida, se actualizan los pesos de conexión de cada neurona, para hacer que la red converja hacia un estado que permita clasificar correctamente todos los patrones de entrenamiento. La importancia de este proceso consiste en que, a medida que se entrena la red, las neuronas de las

capas intermedias se organizan a sí mismas de tal modo que las distintas neuronas aprenden a reconocer distintas características del espacio total de entrada. Después del entrenamiento, cuando se les presente un patrón arbitrario de entrada que contenga ruido o que esté incompleto, las neuronas de la capa oculta de la red responderán con una salida activa si la nueva entrada contiene un patrón que se asemeje a aquella característica que las neuronas individuales hayan aprendido a reconocer durante su entrenamiento. Y a la inversa, las unidades de las capas ocultas tienen una tendencia a inhibir su salida si el patrón de entrada no contiene la característica para reconocer, para la cual han sido entrenadas.
Para redes multicapa, el Backpropagation, es una generalización del algoritmo LMS (Least minimum squared), ambos algoritmos realizan su labor de actualización de pesos y ganancias con base en el error medio cuadrático. Como la red Backpropagation trabaja bajo aprendizaje supervisado necesita un conjunto de entrenamiento que le describa la salida para cada set de entrada [25].
Cada patrón de entrenamiento se propaga a través de la red y sus parámetros para producir una respuesta en la capa de salida, la cual se compara con los patrones objetivo o salidas deseadas para calcular el error en el aprendizaje, este error marca el camino mas adecuado para la actualización de los pesos y ganancias, que al final del entrenamiento producirán una respuesta satisfactoria a todos los patrones de entrenamiento, esto se logra minimizando el error medio cuadrático en cada iteración del proceso de aprendizaje [9].

El error medio a la salida de una neurona $\boldsymbol{j}$ para un conjunto de entrenamiento $\boldsymbol{p}$ es definido entonces de la siguiente manera:

$$E_p = \frac{1}{2}\Sigma_j \left(d_{pj} - y_{pj}\right)^2 \tag{2.4}$$

donde:
d_{pj}: representa la salida esperada para un conjunto de entrenamiento.
y_{pj}: representa la salida obtenida por la red.
Para actualizar los pesos sinápticos asociados a la capa de salida se requiere computar entonces la siguiente ecuación:

$$\Delta_p w_{ij} = \eta \left(d_{pj} - y_{pj}\right) x_{pi} = \eta \delta_{pj} x_{pi} \tag{2.5}$$

donde:
η: es el paso del vector gradiente o también conocido como factor de aprendizaje.
Cuando el modelo escogido posee varias capas, se parte de las funciones de activación con monotonía creciente y diferenciables. Es posible obtener por este método el error mínimo para el cual el modelo se comporta de forma satisfactoria.

$$\delta_{pj} = \frac{\partial E_p}{\partial z_{pj}} = -\frac{\partial E_p}{\partial y_{pj}} \frac{\partial E_p}{\partial z_{pj}} = -\frac{\partial E_p}{\partial y_{pj}} f'_j(z_{pj}) \tag{2.6}$$

Para las neuronas de las capas ocultas, como no se dispone de una expresión para el error, el cálculo no es inmediato. Nuevamente se procede a aplicar regla de la cadena, de forma que:

$$\sum_p \frac{\partial E_p}{\partial z_{pk}} \frac{\partial z_{pk}}{\partial y_{pj}} = \sum_p \frac{\partial E_p}{\partial z_{pk}} \frac{\partial}{\partial y_{pj}} \sum w_{ji} x_{pi} = \sum_p \frac{\partial E_p}{\partial z_{pk}} w_{kj} = -\sum_k \delta_{pk} w_{kj} \tag{2.7}$$

para las capas ocultas entonces el error se traduciría en:

$$\delta_{pj} = f'_j(z_{pj}) \sum_k \delta_{pk} w_{kj} v \tag{2.8}$$

Estas ecuaciones determinan la forma en que se actualizan los pesos sinápticos de la red y el aporte al error cometido por cada neurona, exceptuando su propia capa. En las técnicas de gradiente descendiente es conveniente avanzar por la superficie de error (es el hiperplano conformado por la función de coste y los pesos sinápticos de la red) con incrementos pequeños de los pesos. Esto se debe a que tenemos una información local de la superficie y no se sabe cuan lejos o cerca se está del punto mínimo. Si se toman

incrementos grandes, se corre el riesgo de pasar por encima del punto mínimo global. Con incrementos pequeños, por otra parte, aunque se tarde más en llegar, se evita que esto ocurra.
El elegir un incremento adecuado influye en la velocidad de convergencia del algoritmo, esta velocidad se controla a través de la tasa de aprendizaje, que por lo general se escoge como un número pequeño, para asegurar que la red encuentre una solución adecuada. Un valor pequeño de η significa que la red tendrá que hacer un gran número de iteraciones, si se toma un valor muy grande, los cambios en los pesos serán muy grandes, avanzando muy rápidamente por la superficie de error, con el riesgo de saltar el valor mínimo del error y estar oscilando alrededor de él, pero sin poder alcanzarlo [9].

Capítulo 3

Análisis, estructuración y organización de los datos

3.1. Introducción

La generación de los vectores de prueba juega un papel muy importante para el diagnóstico de fallas en circuitos integrados, entre mayor sea el número de entradas del circuito mayor será la cantidad de vectores de prueba que deberá generarse. En nuestro proyecto, los vectores se generan de dos formas, la primera mediante un algoritmo diseñado en lenguaje C para el circuito C17 a estos se les conoce como *vectores funcionales* y la otra es un conjunto de vectores predefinidos para el circuito C432, llamados *vectores estructurales*, que están expuesto en el sitio web de circuitos ISCAS de BENCHMARK [32]. Ambos grupos se encuentran programados en la tarjeta Raspberry Pi por un algoritmo implementado en lenguaje C. Estos son transmitidos al FPGA, donde se encuentra el circuito que se está probando, posteriormente la respuesta generada es enviada a la Raspberry Pi para ser procesada por la red neuronal. Es muy importante conocer el diseño del circuito lógico que se está probando porque entre más información se extraiga mejor será para el procesamiento en la red neuronal [33].

3.2. Circuitos combinacionales de pruebas estándar ISCAS'85

Los ISCAS'85 son circuitos combinacionales que fueron proporcionados a los autores del ***Simposio Internacional sobre Circuitos y Sistemas*** en el año 1985. Su complejidad va aumentado en dependencia de la cantidad de compuertas internas y el nivel de conexión que presenten, delimitando de esta forma su escala de integración. Estos circuitos son utilizados en diferentes ramas de investigación pero su mayor aplicación se encuentra en el área del *testing*. La gama comienza a partir de los modelos C1,C5,C17 hasta el C47552, en la Tabla 3.1 mencionamos algunos de los más complejos.

Para realizar la comprobación funcional de nuestro algoritmo hemos tomado como objeto de prueba los circuitos C17 y C432.

3.2.1. Circuito de prueba ISCAS'85 C17

El C17 es un circuito combinacional conformado por seis compuertas NAND ver Figura 3.1. Tiene un conjunto de 5 entradas (N1,N2,N3,N4 y N5) y 2 salidas (N22,N23) primarias por lo que solo pueden generarse 32 vectores de pruebas. Tiene 16 líneas de conexión, esto permite que pueden generarse 28 fallas del tipo Stuck-at simples. Es importante resaltar que sólo se han modelado las fallas Stuck-at simples pues esto simplifica el análisis para la detección de fallas [12]. Teniendo en cuenta la respuesta correcta de la tabla de verdad del C17 hemos representado los mintérminos para determinar las funciones de salidas F_{N22} y F_{N23} mediante el mapa de Karnaugh. Esto es fundamental para poder realizar la comprobación

Circuit Name	Circuit Function	Total Gates	Input Lines	Output Lines	Fault
C432	Priority Decoder	160	36	7	524
C499	ECAT	202	41	32	758
C880	ALU and Control	383	60	26	942
C1355	ECAT	546	41	32	1574
C1908	ECAT	880	33	25	1879
C2670	ALU and Control	1193	233	140	2747
C3540	ALU and Control	1669	50	22	3428
C5315	ALU and Selector	2307	178	123	5350
6288	16 bit Multiplier	2406	32	32	7744
C47552	ALU and Control	3512	207	108	7550

Tabla 3.1: Circuitos combinacionales ISCAS '85 Benchmark [6]

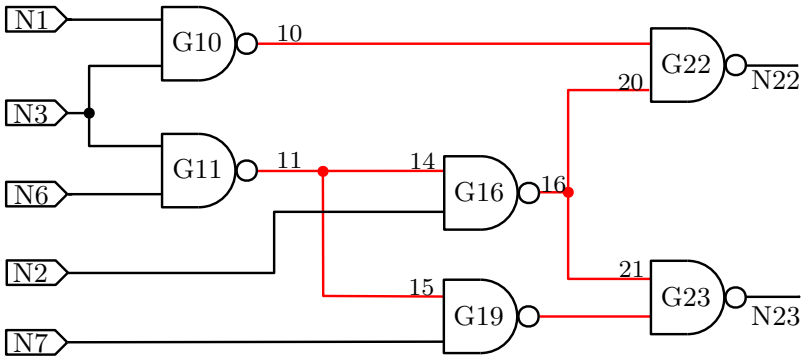
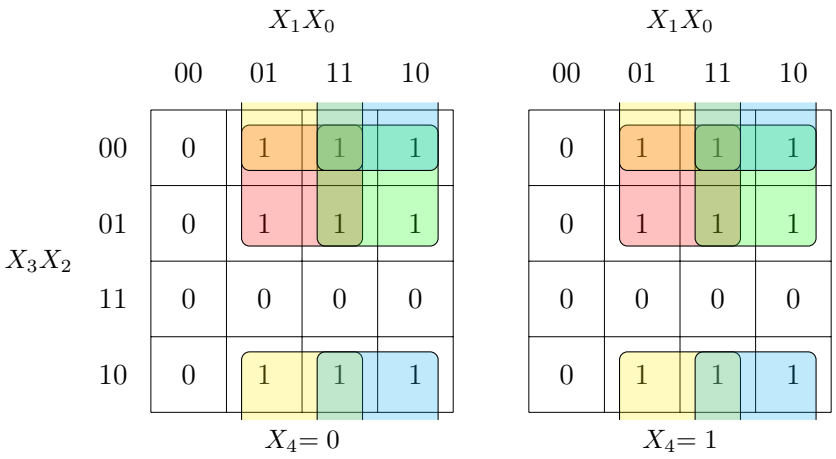


Figura 3.1: Circuitos combinacionales ISCAS '85 C17 [4]

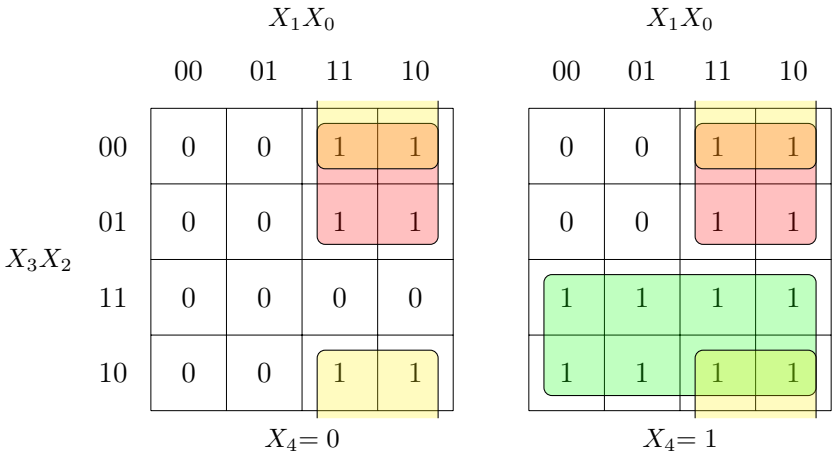
del funcionamiento del C17. Las salidas N23 y N22 serían las respuestas conocidas, la cuales se tendrían en cuenta, además de otras condiciones para determinar la funcionalidad del circuito. Con el objetivo de realizar las simulaciones de las fallas Stuck-at, este circuito fue implementado en el lenguaje descriptivo Verilog para ser programado en el FPGA Cyclon IV, aprovechando la capacidad de reconfiguración de estos dispositivos, que es muy ventajoso pues se pueden realizar 28 implementaciones en un sólo chip sin necesidad de tener 28 circuitos C17 físicos con una falla única diferente.

$$F_{N23} = \sum (1, 2, 3, 5, 6, 7, 9, 10, 11, 17, 18, 19, 21, 22, 23, 25, 26, 27) \tag{3.1}$$



$$F_{N23} = \overline{N3}N2 + \overline{N6}N2 + \overline{N3}N7 + N6N7 \tag{3.2}$$

$$F_{N22} = \sum (2, 3, 6, 7, 10, 11, 18, 19, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31) \tag{3.3}$$



$$F_{N22} = \overline{N3}N2 + \overline{N6}N2 + N1N3 \tag{3.4}$$

3.2.2. Circuito de prueba ISCAS'85 C432

El circuito combinacional C432 es un *controlador de interrupción* de 27 canales que tiene 36 entradas y 7 salidas. Las señales de entrada son agrupadas en paquetes de 9 bits formando de esta manera cuatro buses llamados A,B,C y E. Las peticiones de interrupción se realizan mediante los canales A,B y C donde el nivel de prioridad lo define la posición del bit dentro de cada bus. Por otro lado tenemos el bus de activación E que se encarga de habilitar o deshabilitar las solicitudes de interrupción dentro de la respectiva posición del bit. El régimen de prioridad se aplica como sigue: $A[i] > B[j] > C[k]$, para cualquier i, y k es decir, el bus A tiene la mayor prioridad y el bus C la menor. Dentro de cada bus, un canal con un índice alto tiene prioridad sobre uno con un índice más bajo; por ejemplo $A[i] > A[j]$, si es que $i > j$. Si $E[i] = 0$, entonces las entradas A[i], B[i] y C[i] se discriminan. Tenemos además cinco módulos representados por M1,M2,M3,M4 y M5, donde los tres primeros son controlados de forma independiente por los canales A,B y C respectivamente, todas las señales se concentran en el módulo M4. El bus E se encuentra conectado a los cuatro módulo porque es que el habilitador de las peticiones, de forma sencilla hemos representado el circuito en la Figura 3.2. Las sietes salidas PA, PB, PC y Chan[3:0] especifican que canales han reconocido las solicitudes de interrupción. Sólo el canal de la más alta prioridad en el bus de interés de más alta prioridad es reconocido. Una excepción es que si dos o más solicitudes producen interrupciones en el canal que se reconoce, cada bus se reconoce. Por ejemplo, si A[4], A[2], A[6] y C[4] tienen solicitudes pendientes, A[4] y C[4] son reconocidas. En modulo M5 es un codificador con prioridad de 9 a 4 líneas. La línea de salida número 421 actualmente produce la respuesta invertida Chan[3] [32]. De la misma forma que el C17, este circuito fue implementado en lenguaje Verilog para simular las fallas Stuck-at en el dispositivo FPGA antes mencionado.

3.3. Análisis de datos: Vectores de entrada

Como se mencionó anteriormente una de las características que presentan las **ANN** es su uso en aplicaciones donde se desconoce exactamente que relación existe entre las entradas y salidas en un conjunto

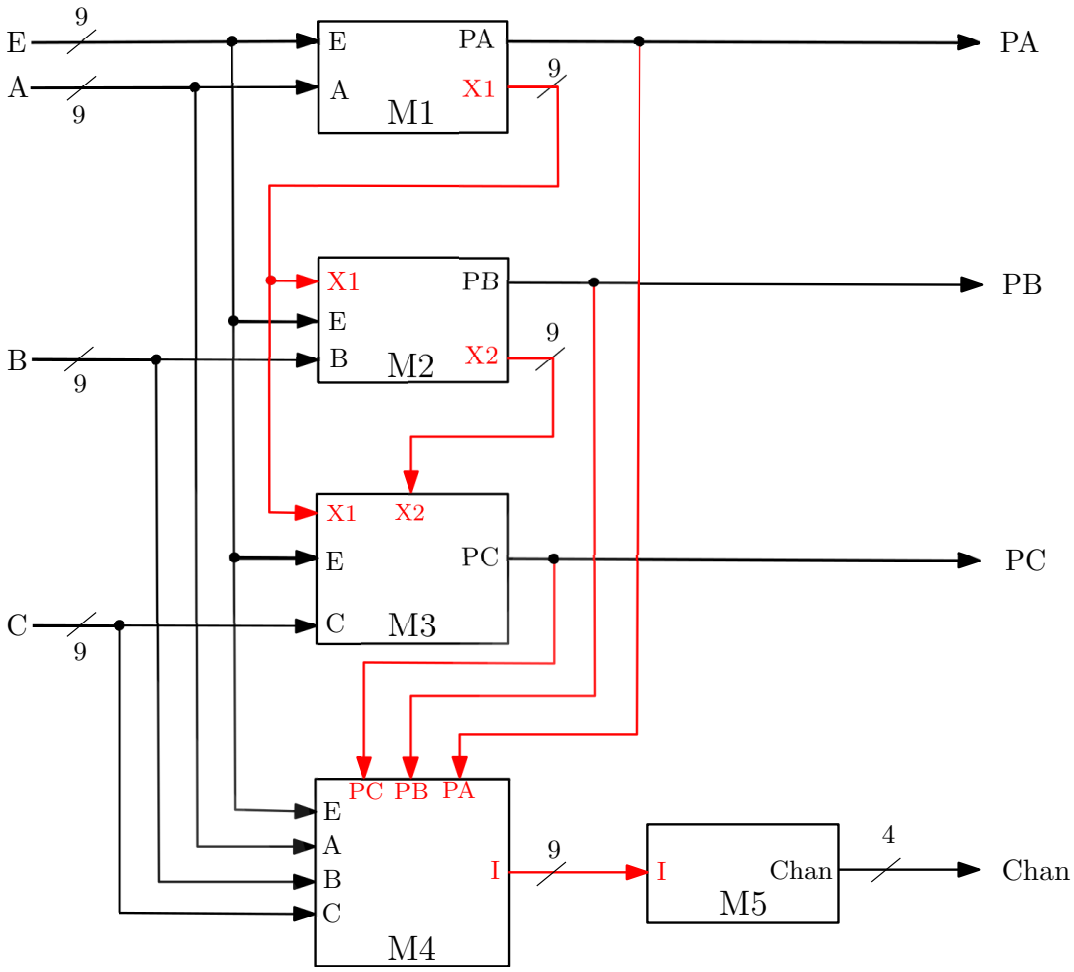


Figura 3.2: Modelo de nivel alto del controlador de interrupción C432 [5].

de datos. En nuestro caso queremos conocer que características presentan los vectores de prueba para poder diagnosticar las fallas en un circuito digital VLSI. Existe la posibilidad de que cierto grupo de vectores se encuentren apuntando hacia la misma región en el espacio, o sea que varios vectores detecten las misma fallas en el circuito de prueba y esto nos permitiría escoger sólo a un vector, de ese grupo particular, para detectar todas las fallas de esa región.

3.3.1. Clasificación de los vectores de prueba para el circuito C17

Como se mencionó anteriormente, los vectores de pruebas para el circuito C17 fueron generados por un algoritmo en lenguaje C. Para comprobar su funcionalidad se diseñó un circuito C17 en el software *Proteus* de Labcenter Electronics con todas las fallas simples en cada interconexión. Esta simulación se basa en interruptores que se encuentran conectados a una fuente de tensión ya sea Vcc o GND. Para simular la fallas S-a-0 se conecta a GND solamente el interruptor asociado a esa conexión y se observan los bit de salida, de igual manera sucede con las fallas S-a-1 pero estas se conectan a Vcc. Solamente se simuló una falla en ese instante de tiempo, los datos de salida de cada compuerta se almacenaron en una base de datos confeccionada en el programa *Excel*. De esta forma se logró confeccionar una matriz de 32 vectores con 28 fallas simuladas para cada vector o sea 14 fallas del tipo $S-a-1$ y las restantes $S-a-0$. Mediante un proceso de clasificación, se encontró un conjunto de *tres vectores* que podían detectar todas las fallas del circuito con la menor correlación posible. Esta información es mostrada en la Tabla 3.2.

En la Tabla 3.2 se observan los tres vectores que detectan el 99 % de todas las fallas del C17. La Tabla

Vector1	Vector2	Vector3
1 0 1 1 0	1 1 1 1 1	0 1 0 0 1
1 1	0 1	1 0
1 1	0 0	1 0
1 1	1 1	1 0
1 1	0 1	1 1
1 1	1 1	1 0
1 1	1 1	1 0
0 0	0 1	1 0
0 0	0 1	1 0
1 1	0 1	0 0
1 1	0 1	0 0
1 1	1 1	1 0
1 1	0 1	1 1
1 0	0 0	1 0
1 1	1 1	1 0
0 1	0 1	0 0

Vector1	Vector2	Vector3
1 0 1 1 0	1 1 1 1 1	0 1 0 0 1
1 1	0 1	1 0
1 1	0 1	1 1
0 1	0 1	1 0
1 1	0 0	1 0
0 0	0 1	1 0
1 1	0 1	1 0
1 1	1 1	1 0
1 1	0 1	1 0
1 1	1 1	1 0
1 1	0 1	1 0
1 1	0 1	0 0
1 0	0 1	1 0
1 1	0 1	1 1
0 1	0 1	1 0
1 1	1 1	1 0

Tabla 3.2: Tabla de vectores organizada por tipo de falla.

de la izquierda contiene todas las 14 fallas $S-a-0$ simuladas mientras que la otra Tabla nos muestra las $S-a-1$. La salida de cada vector se encuentra en su respectiva columna, las salidas que tienen el color verde son las respuestas correctas esperadas cuando el circuito está sano.
Por otra parte, las de color negro son aquellas que aún cuando hay fallas el circuito arroja una respuesta correcta. Las salidas de color rojos representan las fallas del tipo $S-a-0$ que se encuentran en la Tabla izquierda, en tanto color azul identifica las fallas Stuck-at a Vcc o sea $S-a-1$ visualizadas en la Tabla derecha. Por último, tenemos la falla enmascarada, color amarillo, que a diferencia de las de color negro, son fallas que sí producen cambios funcionales pero las salidas coinciden con la de un circuito sano y es la única falla que no se puede detectar. Estos son los datos que emplearemos para entrenar la red neuronal utilizando el método de aprendizaje supervisado para el circuito C17. La información que entra a la red es un vector con 7 bits donde los 2 bits más significativo representan las salidas N23 y N22, los bits restantes coinciden con el vector de prueba que se esté aplicando en ese momento.

Es importante destacar que, como no existe un solapamiento considerable entre fallas, se reducen los errores causados por *la correlación de datos*. El problema de *la correlación de datos* será tratado más adelante.

3.3.2. Clasificación de los vectores de prueba para el circuito C432

El circuito combinacional C432 es mucho mas complejo pues tiene mas compuertas internas e interconexiones que el C17. Además la longitud del vector de entrada es de 36 bits esto permite que sean generados 2^{36} vectores funcionales dando como resultado **68,719,476,736**. Este número es muy grande lo que complejiza el análisis lógico funcional, trayendo como consecuencia mayor cantidad de procesamiento de cómputo y demora. Un Test estructural es mucho más efectivo, pues sólo simularemos las fallas en interconexiones específicas del circuito. Para reducir nuestro análisis y concentrarnos solo en las interconexiones de mayor probabilidad de fallas en el C432 fue necesario la aplicación de algoritmo de **Dijkstra** y los valores de mayor capacitancia parásita para determinar las rutas más críticas que indican los por cientos de las probabilidades más altas de ocurrencia de fallas en el circuito [34]. Teniendo en cuenta la ruta se identificaron éstas interconexiones para confeccionar un circuito combinacional, también utilizando el software de diseño *Proteus*. La idea que se persigue es la misma que el circuito anterior solo que esta

vez comprobaremos los 32 vectores estructurales propuesto por BENCHMARK [32]. Sólo se simularon fallas simples en un instante de tiempo, los bits de salida también fueron almacenados en una base datos confeccionada en *Excel*. La cantidad de fallas Stuck-at realizadas fueron 72 en total, 36 $S-a-0$ y las restantes $S-a-1$, de manera similar encontramos un grupos de seis vectores que podían detectar solo el 90 % de las fallas simuladas pues con ese conjunto de vectores no fueron reconocidas las $S-a-1$ internas en el módulo 3 del circuito C432. A continuación mostramos la Tabla 3.3 con los vectores de pruebas seleccionados para el entrenamiento de la red neuronal artificial para el C432. Hemos realizado una conversión de notación binaria a decimal, Tabla 3.3, para entender mejor esta información, pues con la cantidad de 0 y 1, el lector tiende a confundirse. Las fallas S-a-0 y S-a-1 son reflejadas en las Tablas 3.4 y 3.5 respectivamente.

Número Vector	**Notación Binario**	**Notación Decimal**
1	**11111111111111111010011111101111111**	**68719296383**
2	**01111110101111110101111101011111100**	**34023537404**
3	**10000001010000001010000010100000000**	**34695939328**
4	**11111111111111111111111011111111111**	**68719474687**
5	**11111111111101111111101111111011111**	**68711071711**
6	**11111111111101101100010010000010010**	**68709795876**

Tabla 3.3: Equivalencia de vectores de prueba en notación binaria a decimal.

N_o	**Vector**	**Salidas**	**Lugar**
1	**68719296383**	**0111100**	**F00**
		1000000	**M11-F01**,**M12-F01** **M13-F01**
		0111110	**M55-F01**
2	**34023537404**	**1000001**	**F00**
		1100000	**M22-F01**,**M23-F01**, **M24-F01**
		1010000	**M33-F01**,**M35-F01**, **M36-F01**,**M37-F01**,**M38-F01**
			M39-F01,**M310-F01**
		1001001	**M47-F01**,**M54-F01**
		1000011	**M55-F01**,**M57-F01**
3	**34695939328**	**1111010**	**F00**
		1111011	**M58-F01**,**M56-F10**
4	**68719474687**	**0011011**	**F00**
		1000000	**M11-F01**,**M12-F01**,**M13-F01**,**M14-F01**, **M15-F01**
		0100000	**M22-F01**,**M23-F01**,**M24-F01**
		0011110	**M44-F01**
		0011000	**M56-F01**
5	**68711071711**	**0101001**	**F00**
		0110000	**M35-F01**,**M36-F01**,**M37-F01**,**M38-F01**, **M39-F01**,**M310-F01**
		0101011	**M55-F01**,**M57-F01**
6	**68709795876**	**1111100**	**F00**
		1111110	**M55-F01**,**M57-F01**
		1111101	**M58-F01**

Tabla 3.4: Tabla de vectores organizada por tipo de falla Stuck-at 0.

N_o	Vector	Salidas	Lugar
1	68719296383	0111100	F00
		0011100	M24-F10
		0110100	M53-F10,M54-F10
		0101100	M38-F10,M310-F10
2	34023537404	1000001	F00
		1100000	M21-F10
		1000000	M42-F10,M58-F10
		1000011	M56-F10
3	34695939328	1111010	F00
		0111110	M11-F10,M13-F10
		1001010	M24-F10
		1101010	M31-F10,M39-F10,M310-F10
		1110000	M310-F10
		1110010	M53-F10
		1110011	M56-F10
		1111000	M57-F10
4	68719474687	0011011	F00
		0100000	M21-F10
		0000111	M39-F10,M310-F10
		0011110	M45-F10
		0010011	M53-F10,M54-F10
5	68711071711	0101001	F00
		0011110	M24-F10
		0101111	M46-F10
		0100001	M53-F10,M54-F10
		0101011	M56-F10
		0101000	M58-F10
6	68709795876	1111100	F00
		0011110	M13-F10
		1001100	M24-F10
		1101100	M310-F10
		1110100	M53-F10,M54-F10

Tabla 3.5: Tabla de vectores organizada por tipo de falla Stuck-at-1.

Las fallas se han simulado teniendo en cuenta las rutas más críticas en el C432, éstas tienen sus orígenes en las entradas primarias y atraviesan los 5 módulos del circuito hasta llegar a las salidas primarias. La cantidad de fallas por módulo varía y está en dependencia de los datos extraído del layout de diseño como son las capacitancias parásitas y la cantidad de compuertas e interconexiones. En la Tabla 3.4 sólo se han representado las fallas S-a-0, el significado de los colores es el mismo que en la del C17 pero la simbología ha cambiado. Por ejemplo M14-F01 significa la falla S-a-0 número 4 en el módulo 1, los códigos F01 y F10 identifican el tipo de falla, F01 para las Stuck-at a 0 y F10 para las Stuck-at a 1. La Tabla 3.5 sólo muestra las S-a-1 y el significado sigue siendo el mismo.

3.3.3. Algoritmo Neuronal

La estructura de la **ANN** diseñada está conformada por dos capas: la de entrada o sensitiva y la otra de salida. La *tangente hiperbólica* ha sido seleccionada como función de activación de la primera capa debido al tipo de aplicación y a los resultados que se esperan durante el proceso de entrenamiento, lo mismo

sucede con la función *sigmoide* para la capa de salida. La cantidad de neuronas asociadas a la primera capa se ha escogido en dependencia de la longitud del vector de prueba como referencia. Se ha utilizado el entrenamiento supervisado teniendo en cuenta los resultados esperados y la información existente [35]. La misma topología es utilizada para ambos circuitos, solo varían algunos parámetros como son: la cantidad de neuronas en la capa de entrada y en la salida, la razón de entrenamiento, la cantidad de pesos sinápticos y bias.

La red prevista para el C17 contiene, en la primera capa, 7 neuronas que están conectadas a la misma cantidad de señales de entrada y *bias*. Estas son multiplicadas por sus respectivos pesos sinápticos obteniéndose una matriz de orden 7×7 ($W_1 = 49$) correspondiente a 7 neuronas. Los resultados de esta multiplicación serán las entradas de la primera capa, en esta se realizará una suma ponderada para luego ser multiplicadas por la función de activación. La *función de activación* que se utilizó fue la tangente hiperbólica para poder representar en todo el plano los valores que se obtienen de los vectores de prueba. Por otro lado tenemos la capa de salida con 4 neuronas que representan el código correspondiente a la región de falla detectada. Esta capa recibe la información de salida producida por la primera capa multiplicada por sus pesos, una matriz de 7×4 ($W_2 = 28$). En cada neurona se produce la suma ponderada y se multiplica por la segunda función de activación. En esta capa se utilizó una sigmoide como función de activación pues los códigos a representar oscilan entre 0 y 1. Otro parámetro de importancia es la razón de entrenamiento $\eta = 0{,}0035$, la selección de este valor depende mucho de la experiencia que tenga el diseñador.

La misma arquitectura de la ANN se utilizó para obtener un modelo neuronal basado en el circuito C432 pero cambiando la cantidad de neuronas en la capa sensitiva a 20 inicialmente, pues la señal de entrada tiene 43 bits donde los primeros 7 bits hacen referencia a los estado de las salida PA, PB, PC, Chan[4], Chan[3], Chan[2], Chan[1] y Chan[0], mientra que los bits restantes se corresponden a los 36 estados de los vectores de prueba esto trae como consecuencia que la red necesite más neuronas para realizar un buen proceso de clasificación. A diferencia del C17, en este circuito se extraer más información del vector de salida porque su longitud es mayor, 7 bits, esto también modifica la cantidad de neuronas en la capa de salida ajustándose a 6, este valor está en dependencia de la longitud del código de salida. La matriz de los pesos sinápticos de la entrada se reajustó a 7×7 ($W_1 = 79$), lo mismo sucedió con los pesos sinápticos de la capa de salida modificada a 7×7 ($W_1 = 79$). La Figura 3.3 muestra la propuesta de red neuronal para el entrenamiento de estos dos circuitos, solo visualizamos la configuración del C17 porque para el C432 es la misma solo con la variaciones antes mencionadas.

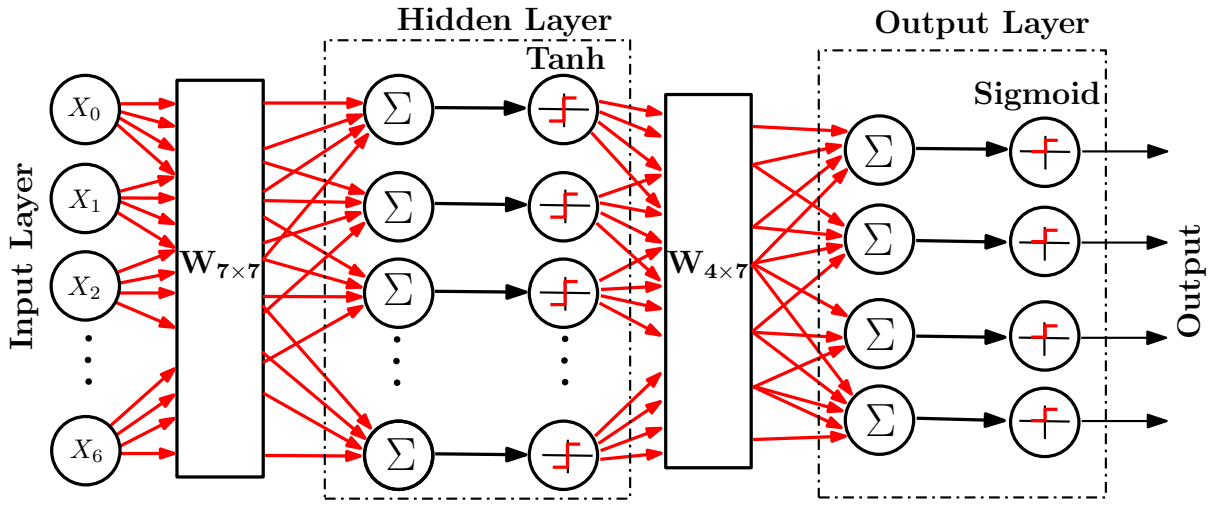


Figura 3.3: Red neuronal de dos capas para el circuito C17.

Interpretación de Datos

Se pudo comprobar que con solo 3 vectores pertenecientes a un conjunto de 32, es suficiente para diagnosticar las fallas Stuck-at simples en el circuito C17. Sin embargo debido a la correlación de datos no es posible determinar con exactitud el punto fallido, pues el circuito C17 solo presenta dos bits de salida los cuales no pueden representar toda la información de las 28 fallas simples presentes. Sin embargo en el C432 existe un poco más de flexibilidad porque el tamaño de bits en la salida primaria es mayor, aunque la cantidad de fallas que pueden generarse internamente es mucho mayor. Aun cuando se hayan utilizado 6 vectores estructurales en el C432 para diagnosticar 72 de fallas simuladas en la ruta más críticas la correlación persiste. El vector que emplea la red para el C17 tiene una longitud de 7 bits, utilicemos ese vector como ejemplo para explicar esta correlación de datos. En la Tabla 3.6 se muestra lo antes mencionado.

Vector	Código
1 0 0 1 0 0 1	**F00**
1 0 0 1 0 0 1	**F01-1**
1 0 0 1 0 0 1	**F01-2**
1 1 0 1 0 0 1	**F01-3**
1 0 0 1 0 0 1	**F01-4**
1 0 0 1 0 0 1	**F01-5**
1 0 0 1 0 0 1	**F01-7**
1 0 0 1 0 0 1	**F01-8**
0 0 0 1 0 0 1	**F01-10**
0 0 0 1 0 0 1	**F01-11**
1 0 0 1 0 0 1	**F01-12**
1 1 0 1 0 0 1	**F01-14**
1 0 0 1 0 0 1	**F01-15**
1 0 0 1 0 0 1	**F01-16**
0 0 0 1 0 0 1	**F01-18**
1 1 0 1 0 0 1	**F10-1**
1 0 0 1 0 0 1	**F10-2**
1 0 0 1 0 0 1	**F10-3**
1 0 0 1 0 0 1	**F10-4**
0 0 0 1 0 0 1	**F10-5**
1 0 0 1 0 0 1	**F10-7**
0 0 0 1 0 0 1	**F10-8**
1 0 0 1 0 0 1	**F10-10**
1 0 0 1 0 0 1	**F10-11**
0 0 0 1 0 0 1	**F10-12**
1 0 0 1 0 0 1	**F10-14**
1 1 0 1 0 0 1	**F10-15**
1 0 0 1 0 0 1	**F10-16**
1 0 0 1 0 0 1	**F10-18**

Vector	Código
1 0 0 1 0 0 1	**F00**
1 0 0 1 0 0 1	**F01-1**
1 0 0 1 0 0 1	**F01-2**
1 1 0 1 0 0 1	**F01-3**
1 0 0 1 0 0 1	**F01-4**
1 0 0 1 0 0 1	**F01-5**
1 0 0 1 0 0 1	**F01-7**
1 0 0 1 0 0 1	**F01-8**
0 0 0 1 0 0 1	**F01-10**
0 0 0 1 0 0 1	**F01-11**
1 0 0 1 0 0 1	**F01-12**
1 1 0 1 0 0 1	**F01-14**
1 0 0 1 0 0 1	**F01-15**
1 0 0 1 0 0 1	**F01-16**
0 0 0 1 0 0 1	**F01-18**
1 1 0 1 0 0 1	**F10-1**
1 0 0 1 0 0 1	**F10-2**
1 0 0 1 0 0 1	**F10-3**
1 0 0 1 0 0 1	**F10-4**
1 0 0 1 0 0 1	**F10-5**
1 0 0 1 0 0 1	**F10-7**
1 0 0 1 0 0 1	**F10-8**
1 0 0 1 0 0 1	**F10-10**
1 0 0 1 0 0 1	**F10-11**
0 0 0 1 0 0 1	**F10-12**
1 0 0 1 0 0 1	**F10-14**
1 1 0 1 0 0 1	**F10-15**
1 0 0 1 0 0 1	**F10-16**
1 0 0 1 0 0 1	**F10-18**

Tabla 3.6: Tabla de vectores de prueba a la izquierda. Correlación de datos a la derecha .

Las Tablas 3.6 mostradas hacen referencia a uno de los vectores que se utiliza para realizar entrenamiento de la red neuronal del C17. La Tabla de la izquierda muestra todas las fallas Stuck-at que puede detectar este vector. En la derecha solo hemos resaltado un vector "1101001"del subconjunto de vectores que detecta las fallas $S-a-0$ F01-3,F01-14 y las fallas $S-a-1$ F10-1 e F10-15. Como podemos ver existe una correlación de datos importante porque el mismo vector es capaz de detectar cuatro fallas diferentes

en total y esto para red es un problema porque no sabe cual de las fallas puede asociar al vector. Por tanto no es posible encontrar exactamente el lugar de la falla pero si *la región*.

Detección de fallas por grupos en el circuito

Los tres vectores de prueba propuestos para el C17 detectan diferentes fallas Stuck-at en lugares específico del circuito por lo que el nivel de solapamiento de zona es mínimo, de esta manera podemos agrupar ese conjunto de fallas y formar un grupo que solo será detectado por ese vector. Esto ayudaría considerablemente a resolver el problema de la correlación de datos en el entrenamiento de la red neuronal. Por otra parte podemos identificar ciertas regiones físicas en el circuito que corresponden con ese subconjunto de fallas y a ese grupo asignarle un código y de esa forma ubicaríamos las fallas. Si utilizáramos la misma Tabla 3.6 (izquierda) nos podemos dar cuenta que ese vector de prueba detecta 5 fallas S-a-0 y 5 fallas S-a-1 con dos respuestas distintas "1101001" y "0001001". Otra forma de interpretar lo mismo sería, decir que el vector "1101001" solo detecta 4 fallas entre S-a-0 y S-a-1, o sea que si ocurriera una falla en uno de estos 4 lugares el circuito arrojaría la misma respuesta. Esto no especifica el lugar de donde ocurrió la falla pero si que puede estar en uno de esos 5 lugares. Si mapeáramos estos puntos de detección obtendríamos la Figura 3.4, también mostramos la Tabla 3.7 para que nos ayude a entender mejor.

Vector	Código
1 0 0 1 0 0 1	F00
1 1 0 1 0 0 1	F01-3,F01-14,F10-1,F10-15
0 0 0 1 0 0 1	F01-10, F01-11,F01-18,F10-5,F10-8,F10-12

Tabla 3.7: Tabla que hace referencia a los grupos de fallas del vector 1001001.

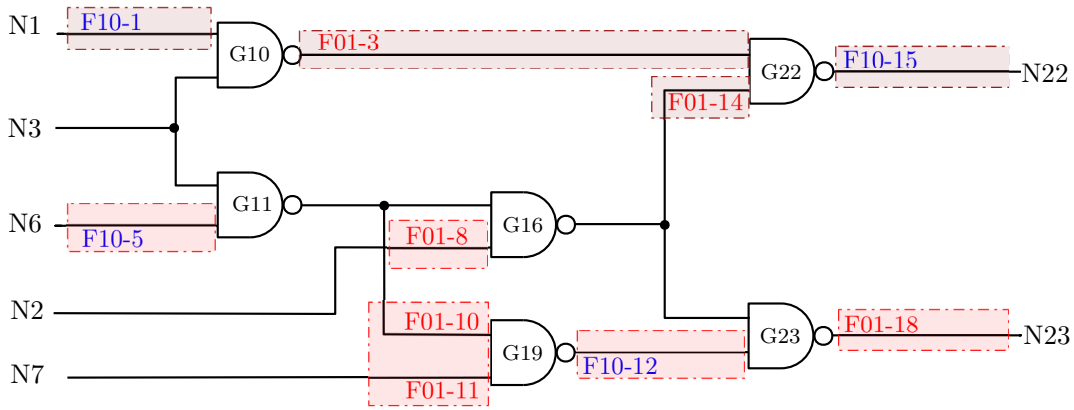


Figura 3.4: Regiones de fallas para el vector de entrada 01001.

También existe una zona de fallas común entre los 3 vectores de prueba, lo que implica una detección por redundancia. El solapamiento es ventajoso para el algoritmos porque la falla será detectada por el primer vector que se envíe.

El mismo análisis se aplica al C432 pero los vectores utilizados tienen mayor zona de solapamiento porque la cantidad de fallas y la longitud de los vectores es mucho mayor. Para ello seleccionaremos el vector de prueba 4 en las Tablas 3.4 y 3.5 ya que es uno de los vectores que más fallas detecta y por lo tanto mayor es el grupo que presenta. Solo hemos representado una parte de las zonas detectadas específicamente en el módulo *M1*.

La representación de las fallas sigue teniendo el mismo significado, pero como el C432 tiene 5 módulos hemos identificado primeramente el número del módulo seguido del número de la falla, finalizando con el tipo de falla. Por ejemplo **M11-F01** significa que es la primera falla simulada del tipo S-a-0 en el módulo

N_o	Vector	Salidas	Identificador
4	**68719474687**	**0011011**	**F00**
		1000000	**M11-F01**,**M12-F01**,**M13-F01**,**M14-F01**, **M15-F01**

Tabla 3.8: Ejemplo de regiones de fallas para el circuito C432.

1 ver Tabla 3.8, también seguimos diferenciando las fallas por el color o sea *rojo* para las S-a-0 y *azul* para las S-a-1. Seguidamente, mostramos donde se encuentran ubicadas estas fallas en el módulo 1 ver la Figura 3.5. A modo de ejemplo tomamos el módulo 1 como una muestra de todas las fallas simuladas en el C432.

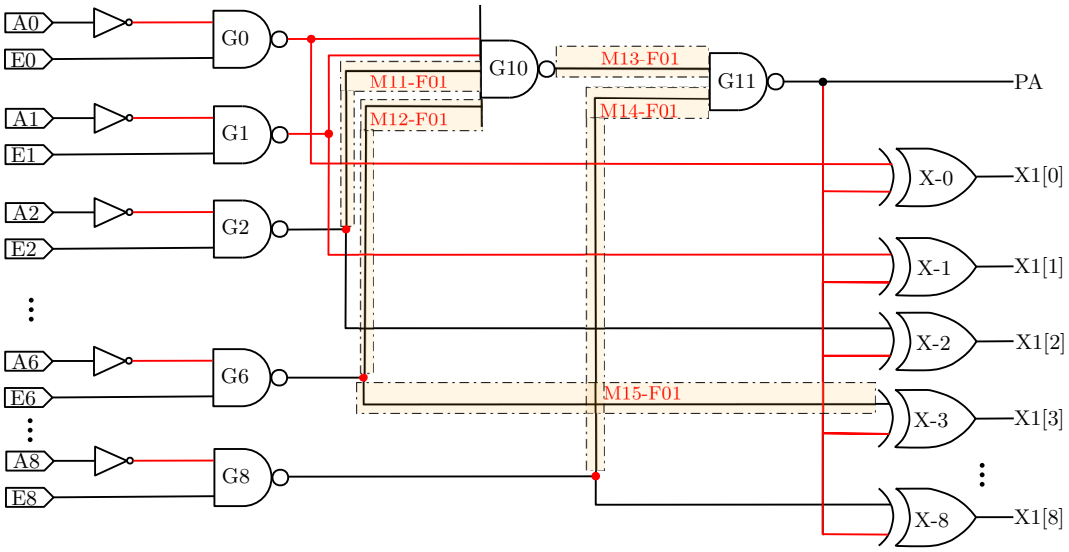


Figura 3.5: Regiones de fallas para el vector de entrada **68719474687**.

3.4. Generación, trasmisión y recepción de datos en el hardware propuesto.

La transferencia de datos se realiza en dos etapas, en la primera, el sentido de transferencia de la información viaja del sistema SBC (single board computer) hacia el FPGA como se observa en la Figura 3.6, pues los vectores de prueba son generados en la Raspberry Pi, que es el dispositivo donde se ha implementado el algoritmo neuronal. Dentro del algoritmo programado en lenguaje C, existe una función que genera los vectores de prueba en dependencia del tipo de circuito ISCAS que se esté comprobando. Para el C17 los vectores de pruebas tienen una longitud de 5 bits mientras que los del C432 son de 36 bits. Cada vector de prueba tiene una función asociada, la hemos declarado *Vector A*, *Vector B*, *Vector C*, *Vector D*, *Vector E* y *Vector F* respectivamente, estas funciones se envían secuencialmente para después ejecutar el algoritmo neuronal. Se procede de la siguiente forma: se envía primeramente la función Vector A, después que el dispositivo responde se ejecuta el algoritmo neuronal, si no se encuentra la falla se procede con la próxima función Vector B y así con las demás funciones hasta encontrar la falla, salvo en algunas excepciones que no será posible el diagnóstico. En el diagrama de flujo mostrado en la Figura 3.7 se explica antes mencionado.

Una vez enviado los datos hacia el FPGA existe una demora de 100 milisegundo para leer la respuesta, todo este proceso se ejecuta en la misma función. La otra etapa consiste en la transferencia de datos del FPGA hacia la Raspberry Pi. De la misma forma que sucede con los vectores de prueba, la salida también depende de las características del circuito a comprobar. Para la transferencia de datos inicialmente fue

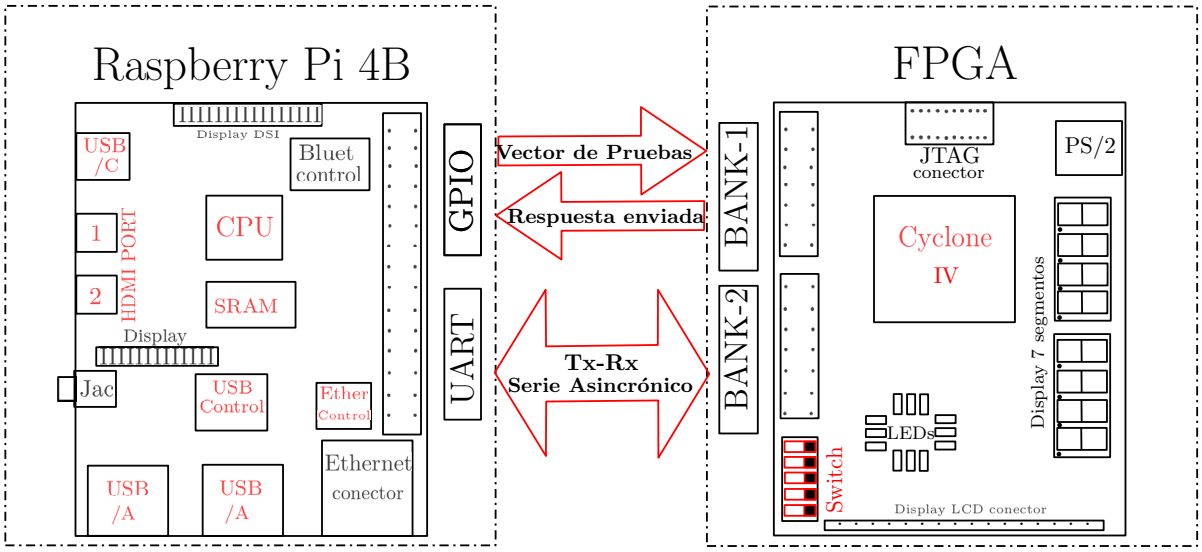


Figura 3.6: Transmisión de datos entre el FPGA y la Raspberry Pi 4B.

implementada vía paralela utilizando el GPIO de la Raspberry Pi específicamente los pines de propósito general 13,14,15,17 y 11, tratándose del la entrada del C17 es de 5 bits. Para el C432 se propone transferir de forma paralela un dato de 9 bits agregando 4 pines más del GPIO 21,22,23 y 24. En el FPGA también se configuraron los pines PG 1,2,3,7,10,28,30,31 y 32 para recibir la información enviada de la Pi y ser procesada en el circuito implementado. Para el C17 solo se utilizarían los primeros 5 bits en cambio para el C432 todo el bus de datos que físicamente serían los bancos 1 y 2 del FPGA. Existe un bloque implementado en Verilog llamado *buffer circular* que se encarga de organizar los datos cuando se está recibiendo la información, este buffer tiene dos rutinas una de lectura y otra de escritura, que funcionan de manera independiente y secuencial [36]. Cuando el Buffer se llena, se envía la información a un registro que tiene un tamaño máximo de 36 bits, posteriormente se procesa la información en el circuito implementado. La respuesta generada en el FPGA es almacenada en un registro de salida con longitud máxima de 7 bits y posteriormente es enviada por los pines 38,39,42,43,44,46 y 49 pertenecientes al banco 3. Para la salida del circuito C17 solo se utilizarían los dos primeros pines 38 y 39. La información es recibida por las Raspberry a través de las entradas 1,4,5,6,26,27 y 28, de igual manera los primeros dos bits son los que se utilizarían para el C17.

3.5. Organización y distribución de los datos en el algoritmo

Como se mencionó en el sección 3.3.1 una vez recibida la respuesta enviada del FPGA se conforma la información que será empleada por la **ANN**. El dato de entrada a la red está estructurado de la siguiente manera, los bits más significativos hacen referencia a los estados de las salidas del circuito que se esté diagnosticando en ese momento. Los bits restantes identifican el vector de prueba correspondiente a la salida producida, por ejemplo el dato de entrada a la red del C17 tiene la forma (N23 N22 N1 N3 N6 N2 N7). Los dos bits más significativos del dato están conformado por la respuesta y los bits restantes lo integran el vector de prueba que se envía.
El dato para la red del C432 estaría formado N223 N329 N370 N421 N430 N431 N432 N1 N4 N8 N11 N14 N17 N21 N24 N27N30 N34 N37 N40 N43 N47 N50 N53 N56 N60 N63 N66 N69 N73 N76 N79 N82 N86 N89 N92, N95,N99,N102,N105, N108,N112,N115. Los bit de salida se corresponden con las señales (PA,PB,PC, Chan[3], Chan[2],Chan[1],Chan[0]) y las entradas con los buses A,E,C y D. Con esta información podemos conformar un conjunto de datos, donde a cada vector de prueba tiene un grupo de fallas asociadas que puede detectar exceptuando las compartidas y las indetectables. Precisamente esa matriz de los datos de

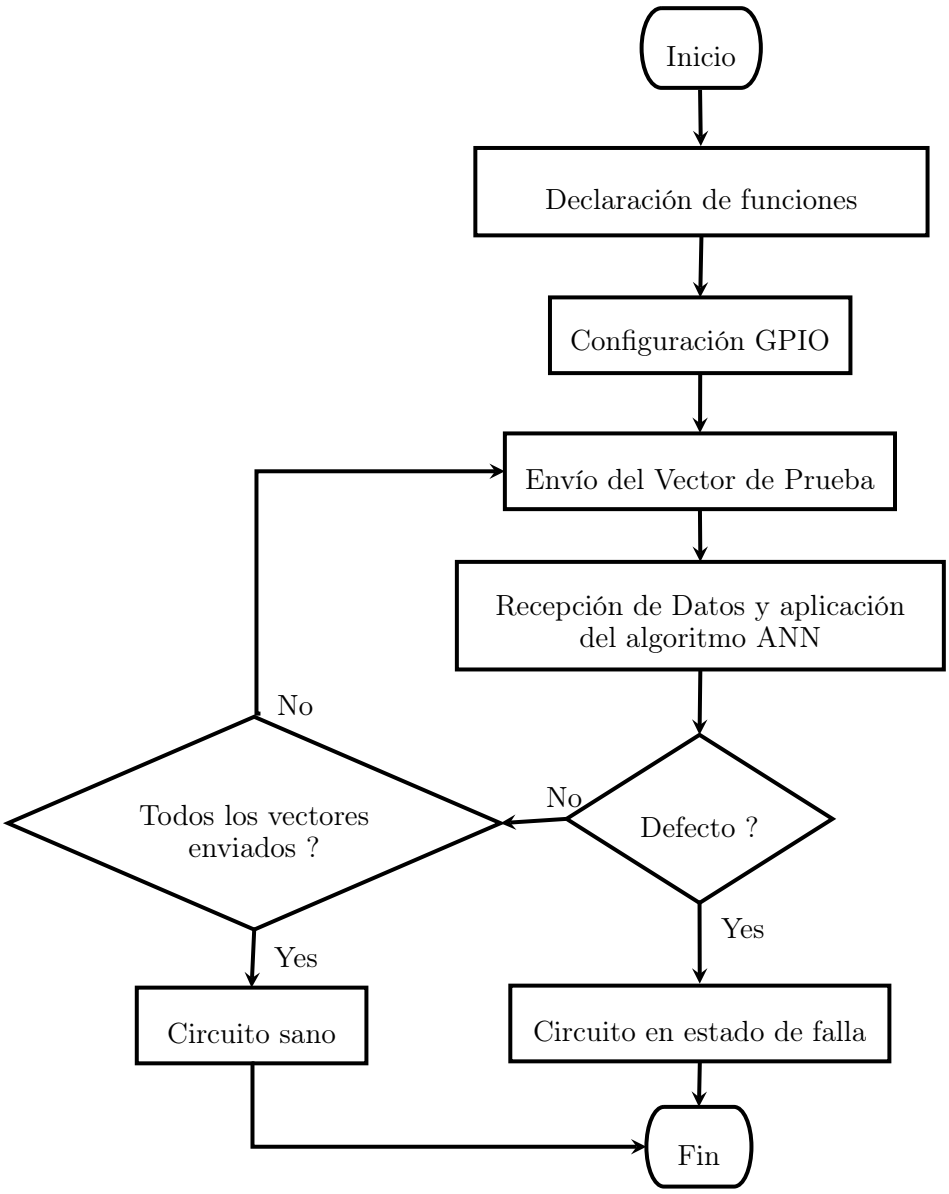


Figura 3.7: Diagrama de flujo del funcionamiento del programa de Testing en la Rapsberry Pi.

entrada se le llamó matriz SAM. En las siguientes tablas se muestran cómo fue organizada esa información para cada circuito, primeramente se muestra la matriz de entrada y luego la salida que se espera después que la red se entrene.

Los identificadores de fallas se han definido teniendo en cuenta el lugar y el tipo de circuito a comprobar. El identificador F00 se corresponde con la respuesta correcta del circuito para cada vector de prueba, el F01 hace referencia a la falla Stuck-at 0 y el F10 a la Stuck-at 1. El número que sigue nos indica el lugar en el circuito, para un mejor entendimiento ver Figura 3.8. La matriz **SAM** cuyo rango en de 10×7, hace referencia a 10 grupos de falla, cuya información de entrada es de 7 bits ver Tabla 3.9. Se ha implementado un codificador que asocia un código a cada zona detectada, esta sería la información que nos aporta la red neuronal. Seguidamente mostramos la matriz de salida de la ANN para el C17.

En la Tabla 3.10, el código 0000 hace referencia a la salida sana, los números del 0001 al 0011 en binario indican las regiones de fallas detectadas por el vector de prueba 10110, por otra parte los códigos 0100 e 0101 nos expresan las fallas detectadas por el vector 11111 y por último tenemos los códigos restantes correspondientes al vector 01001. Esta matriz de códigos representa la salida esperada por la red. En la

Matriz **SAM**	**Vectores**	**Identificadores de fallas**
1 1 1 0 1 1 0	Vector 10110	faultless F00
0 1 1 0 1 1 0	Vector 10110	faults: F10-1, F10-2, F01-18
0 0 1 0 1 1 0	Vector 10110	faults: F10-4, F01-7, F01-8
1 0 1 0 1 1 0	Vector 10110	faults: F10-14, F01-15, F10-16
0 1 1 1 1 1 1	Vector 11111	faultless F00
0 0 1 1 1 1 1	Vector 11111	faults: F01-1, F10-3, F01-15
1 1 1 1 1 1 1	Vector 11111	faults: F01-2, F01-4, F01-5, F10-7, F10-10, F01-12, F01-16, F10-18
1 0 0 1 0 0 1	Vector 01001	faultless F00
1 1 0 1 0 0 1	Vector 01001	fault: F10-1, F01-3, F10-8, F01-14, F01-15
0 0 0 1 0 0 1	Vector 01001	fault: F10-5, F01-10, F01-11, F10-12, F01-18

Tabla 3.9: Tabla que relaciona la información de entrada con el lugar de la falla en el C17.

Código	**Grupos de fallas**
0 0 0 0	F00 Healthy circuit
0 0 0 1	Faults: F10-1, F10-2, F01-18
0 0 1 0	Faults: F10-4, F01-7, F01-8
0 0 1 1	Faults: F10-14, F01-15, F10-16
0 0 0 0	F00, Healthy circuit
0 1 0 0	Faults: F01-1, F10-3, F01-15
0 1 0 1	Faults: F01-2, F01-4, F01-5, F10-7, F10-10, F01-12, F01-16, F10-18
0 0 0 0	F00, Healthy circuit
0 1 1 0	Faults: F10-1, F01-3, F10-8, F01-14, F01-15
0 1 1 1	Faults: F10-5, F01-10, F01-11, F10-12, F01-18

Tabla 3.10: Tabla con la información de los datos de salida a la red neuronal del circuito C17.

Figura 3.8 podemos observar las zonas de fallas en el C17.

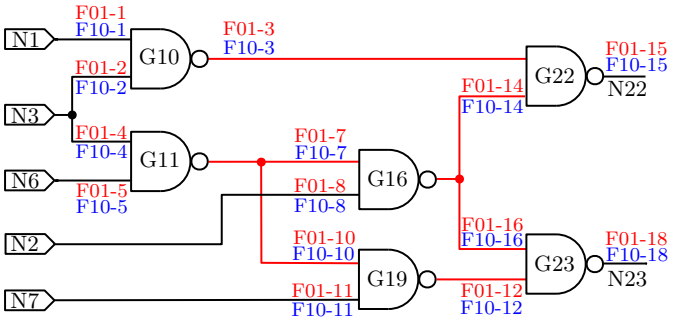


Figura 3.8: Fallas Stuck-at por secciones en el circuito C17.

En la Tabla 3.11 observamos los valores en sistema decimal para un mejor entendimiento, los datos que se emplearan en la red son los que se encuentran en la columna SAM. Como se aprecia es una matriz mucho mas grande, la primera columna nos dice el numero del vector de prueba asociado, en la segunda se muestra el dato de entrada a las red (43 bits). Seguidamente se indica el vector de pruebas correspondiente y por ultimo el identificador de la falla. La nomenclatura se mantiene en el C432, el color rojo hace referencia a las fallas S-a-0 y el azul a las S-a-1. Anteriormente, explicamos el significado del identificador para este circuito ver apéndice 3.3.2. Por otro lado en la Tabla 3.12 se muestra la matriz de salida, el código es de 6 bits por la cantidad de fallas asociadas. Los código tienen la misma interpretación que en el C17. Esta matriz de orden 6×41 contiene todas las zonas de fallas simuladas en el C432.

N_o	SAM	vector	Identificador
1	4191887900543	68719296383	F00
1	4466765807487	68719296383	M11-F01,M12-F01, M13-F01
1	4329326854015	68719296383	M55-F01
1	1992864644991	68719296383	M24-F10
1	3092376272767	68719296383	M53-F10,M54-F10
1	3642132086655	68719296383	M53-F10,M54-F10
2	4500789525244	34023537404	F00
2	6631093304060	34023537404	M22-F01,M23-F01, M24-F01,M21-F10
2	5531581676284	34023537404	M33-F01,M35-F01, M36-F01,M37-F01
			M38-F01, M39-F01,M310-F01
2	5050545339132	34023537404	M47-F01,M54-F01
2	4638228478716	34023537404	M55-F01,M57-F01,M56-F10
2	4432070048508	34023537404	M42-F10,M58-F10
3	8418472101120	34695939328	F00
3	8487191577856	34695939328	M58-F01,M56-F10
3	4295303496960	34695939328	M11-F10,M13-F10
3	5119937217792	34695939328	M24-F10
3	7318960473344	34695939328	M31-F10,M39-F10,M310-F10
3	7731277333760	34695939328	M310-F10
3	7868716287232	34695939328	M53-F10
3	8281033147648	34695939328	M57-F10
4	1924145346559	68719474687	F00
4	4466765985791	68719474687	M11-F01,M12-F01,M13-F01,M14-F01, M15-F01
4	549755811839	68719474687	M39-F10,M310-F10
4	2267742730239	68719474687	M21-F10
4	2130303776767	68719474687	M45-F10
4	1717986916351	68719474687	M56-F01
4	1374389532671	68719474687	M53-F10,M54-F10
5	2886209617887	68711071711	F00
5	3367245955039	68711071711	M35-F01,M36-F01,M37-F01,M38-F01
			M39-F01,M310-F01
5	3023648571359	68711071711	M55-F01,M57-F01
5	2130295373791	68711071711	M24-F10
5	3298526478303	68711071711	M46-F10
5	2336453803999	68711071711	M53-F10,M54-F10
5	2817490141151	68711071711	M58-F10
6	8589924911140	68709795876	F00
6	8727363864612	68709795876	M55-F01,M57-F01
6	8658644387876	68709795876	M58-F01
6	2130294097956	68709795876	M13-F10
6	5291390027812	68709795876	M24-F10
6	7490413283364	68709795876	M310-F10
6	8040169097252	68709795876	M53-F10,M54-F10

Tabla 3.11: Tabla que relaciona cada respuesta generada con el lugar de la falla.

A continuación mostramos las matrices de entrada y salida a la ANN del C432.

Código	Grupos de fallas
0 0 0 0 0 0	**F00**
0 0 0 0 0 1	**M11-F01**,**M12-F01**, **M13-F01**
0 0 0 0 1 0	**M55-F01**
0 0 0 0 1 1	**M24-F10**
0 0 0 1 0 0	**M53-F10**,**M54-F10**
0 0 0 1 0 1	**M53-F10**,**M54-F10**
0 0 0 0 0 0	**F00**
0 0 0 1 1 0	**M22-F01**,**M23-F01**,**M24-F01**,**M21-F10**
0 0 0 1 1 1	**M33-F01**,**M35-F01**,**M36-F01**,**M37-F01** **M38-F01**, **M39-F01**,**M310-F01**
0 0 1 0 0 0	**M47-F01**,**M54-F01**
0 0 1 0 0 1	**M55-F01**,**M57-F01**,**M56-F10**
0 0 1 0 1 0	**M42-F10**,**M58-F10**
0 0 0 0 0 0	**F00**
0 0 1 0 1 1	**M58-F01**,**M56-F10**
0 0 1 1 0 0	**M11-F10**,**M13-F10**
0 0 1 1 0 1	**M24-F10**
0 0 1 1 1 0	**M31-F10**,**M39-F10**,**M310-F10**
0 0 1 1 1 1	**M310-F10**
0 1 0 0 0 0	**M53-F10**
0 1 0 0 0 1	**M57-F10**
0 0 0 0 0 0	**F00**
0 1 0 0 1 0	**M11-F01**,**M12-F01**,**M13-F01**,**M14-F01** **M15-F01**
0 1 0 0 1 1	**M39-F10**,**M310-F10**
0 1 0 1 0 0	**M21-F10**
0 1 0 1 0 1	**M45-F10**
0 1 0 1 1 0	**M56-F01**
0 1 0 1 1 1	**M53-F10**,**M54-F10**
0 0 0 0 0 0	**F00**
0 1 1 0 0 0	**M35-F01**,**M36-F01**,**M37-F01**,**M38-F01** **M39-F01**,**M310-F01**
0 1 1 0 0 1	**M55-F01**,**M57-F01**
0 1 1 0 1 0	**M24-F10**
0 1 1 0 1 1	**M46-F10**
0 1 1 1 0 0	**M53-F10**,**M54-F10**
0 1 1 1 0 1	**M58-F10**
0 0 0 0 0 0	**F00**
0 1 1 1 1 0	**M55-F01**,**M57-F01**
0 1 1 1 1 1	**M58-F01**
1 0 0 0 0 0	**M13-F10**
1 0 0 0 0 1	**M24-F10**
1 0 0 0 1 0	**M310-F10**
1 0 0 0 1 1	**M53-F10**,**M54-F10**

Tabla 3.12: Tabla con la información de los datos de salida a la red neuronal del circuito C432.

Capítulo 4

Programación e implementación del algoritmo de la red neuronal

4.1. Introducción

El objetivo principal de este proyecto no es sólo que la red sea capaz de discernir entre un circuito sano y otro defectuoso sino que detecte, lo más exacto posible, en que región se pueda encontrar la falla. Como se explicó, en el capítulo anterior, los datos que se le proporcionan a la ANN tienen una longitud desde 7 a 43 bits y son introducidos vector por vector. La salida entrega un código que se corresponde con el grupo de falla asociado al vector que se esté probando en ese instante. Para que la red funcione correctamente debe pasar por un proceso de entrenamiento donde el valor de los pesos sinápticos se calculan de tal manera que la salida de la red sea la correcta, teniendo en cuenta el error esperado. El entrenamiento fue realizado con el software libre **Scilab** de Scilab Enterprise y la programación del algoritmo fue llevado a cavo en *Lenguaje C* debido a las excelentes propiedades que nos brinda por ser compatible con nuestro dispositivo de cómputo Raspberry Pi para la implementación de algoritmos neuronales [37] [38].

4.2. Entrenamiento de la red neuronal en Scilab

Primeramente, es necesario la organización de la información de los datos de entrada - salida, esto garantiza que la red no se equivoque durante el proceso de entrenamiento. La información de entrada es una matriz llamada *SAM*, que contiene toda la información correspondiente a los vectores de prueba, la respuesta sana y todas las fallas simuladas del circuito. Por otra parte tenemos la matriz *Zout* con los códigos que serán entregados por la red para determinar si el circuito está sano o si está fallido y a qué región de fallas pertenece. Las matrices de entradas y salida de ambos circuitos ISCAS'85 fueron mostradas en el capítulo anterior, en la sección 3.5. Para facilitar el entrenamiento de la red, teniendo en cuenta los valores límites de la función de activación de salida (sigmoide) que se ha empleado, fue necesario convertir la representación de nivel lógico **1** a **0.8** y el nivel **0** a **0.2**. Esto es necesario debido a que el algoritmo de Backpropagation es *recursivo* y emplea el método de derivación en cadena. Por esta razón, una de las condiciones necesarias es que la función de activación sea derivable en todos sus puntos y n veces, de esta forma se garantizan los ciclos recursivos de derivación, si los valores son cercanos a 0 y 1. Los datos son generados y almacenado en un documento creado por el Scilab que posteriormente será utilizado en el programa principal.

4.2.1. Estructura del algoritmo neuronal

El programa principal se encuentra dividido en tres partes fundamentalmente, en la primera se declara el número de iteraciones que tendrá nuestro entrenamiento, esto también se conoce como *épocas*. Se cargan

los datos que serán empleados por la red, entre los que se encuentran la matriz de entrada (SAM) y la de salida(Zout) que fueron explicadas en la sección 3.5. Por otra parte se declaran las funciones de activación que se utilizarán en el algoritmo del Backpropagation, para nuestra aplicación utilizaremos las funciones *Tangente Hiperbólica* y *sigmoide*. Es importante aclarar que las funciones de activación se declaran de forma diferente, por ejemplo el código de la función sigmoide que se emplea en el algoritmo del *Forward Pass* no es el mismo que se emplea en el *Backward Pass* pues en este último se incluye el algoritmo del gradiente en descenso y esto requiere de otros parámetros. Estos algoritmos son aplicables tanto para circuito C17 como para el C432.

Algorithm 1 Algoritmo para la organización de los datos de la ANN en Pseudocódigo

```
//Definición de variables
Matriz Zout //Matriz de salida de datos
Matriz SAM // Matriz de entrada de datos
ri          // número de filas de Matriz
cj          // número de columnas de Matriz
conti, cont_j   // contadores
//Implementación del algoritmo
r_i=número de fila
c_j=número de columna
cont_i = 0
for_1( cont_j = 0 to r_i do)
  for_2(cont_j = 0 to c_j do)
    if(elemento SAM==1)
      elemento SAM=0.8
    else
      elemento SAM=0.2
    endif
   cont_j = cont_j + 1
  endfor2
 cont_i = cont_i + 1
endfor1
Guardar la Matriz SAM y Zout en fichero
```

En la segunda parte se define un prototipo de función para actualizar la tasa de aprendizaje de adaptación. Si está activada, cambiará la tasa de aprendizaje y acelerará el tiempo de convergencia o reducirá el número de épocas. Se debe de tener cuidado al elegir las constantes y el valor inicial de eta (η), razón de aprendizaje, ya que el algoritmo puede volverse inestable. Además, se configuran los parámetros ANN como, retrasos de entrada, número de neuronas de entrada, número de neuronas de salida, número de señales de entrada, número señales de salida, tamaño de matrices de peso y matrices de polarización. Nuestro algoritmo para el circuito de prueba ISCAS C17 cuenta con dos capas, la primera está conformada por 7 neuronas y la segunda por 4. Para el C432 tenemos 24 neuronas en la capa de entrada y 6 en la capa de salida, esto es teniendo en cuenta la base de datos con la que se cuenta. Por último tenemos la implementación del algoritmo *Backpropagation* conformado por dos rutinas la primera es el *Forward Pass* y la segunda el *Backward Pass*. En el Forward Pass se actualizan los valores de los pesos sinápticos, bias y los datos de entradas, el sentido del procesamiento es de la entrada hacia la salida. Una vez terminada el Forward se calcula el error de salida y concluido pasamos a la rutina del *Backward Pass*, aquí es donde se aplica el gradiente en descenso para retropropagar ese error calculado y puedan ser actualizados los parámetros de la red, el sentido del procesamiento es de la salida hacia la entrada. Al terminar las iteraciones se calcula el error total mediante la *función de coste* y éste se compara con el error definido, el algoritmo concluye cuando se cumple esta condición. Por último se guardan los resultados de los parámetros calculados tales

como: los pesos sinápticos los valores de los bias de la primera y segunda etapa y el eta. A continuación, en el algoritmo 2, mostramos el Pseudocódigo del programa de entrenamiento:

Algorithm 2 Algoritmo de la red neuronal en Pseudocódigo

//Definición de variables y Funciones
Función **afht** //Función de activación tangente h. para el Forward Pass
Función **afsig** //Función de activación sigmoide para el Forward Pass
Función **BPtanhInputLayer** //Función de activación tangente h.para el Backward Pass
Función **BPSigmOutputLayer** //Función de activación sigmoide para el Backward Pass
Cargar el archivo donde se almacenan los Datos para el entrenamiento de la red(SAM y Zout)
//variables
TraininSetSize = el número de filas de la matriz SAM
InputVectorSize = número de columnas de la matriz SAM
*CodePerFaul*t = número de columnas de la matriz Zout
CodeLength = número de columnas de la matriz Zout
e= ceros de *CodePerFault* y *TraininSetSize*
esum= ceros de *TraininSetSize*
Y = ceros de *CodePerFault* y *TraininSetSize*
eta=0.035 hspace1.5 //razón de entrenamiento
Definición de variable para el entrenamiento: *nin*, *non*, *nbi*, *nbo*, *nin* y *nos* ,**epochs**
//Inicialización
W1= valor aleatorio, *W2*= valor aleatorio, *B1*= valor aleatorio, *B2*= valor aleatorio,
for_1($k = 1$ **to** *epochs* **do**)
  for_2($i = 1$ **to** *TraininSetSize* **do**)
//Forward Pass
    P= carga vector de SAM; *FLout*= Función **afht**; *SLout*= Función **afsig**
    SLout= Función **afsig**; *Y*= SLout; *e*= *SLout*- *Y*
//Backward Pass
    $[W2, B2, deltao]$= Función **BPSigmOutputLayer**
    $[W1, B1, deltai]$= Función **BPtanhInputLayer**
    i=i+1
 endfor_2
 esum= cálculo del error por iteración
  if(esum<**0.001**
    Salir del ciclo
  end_if
 $k = k + 1$
endfor_1
Guardar "*W1*","*W2*","*B1*","*B2*","*eta*.en fichero

4.2.2. Descripción del algoritmo neuronal

El tipo de entrenamiento utilizado es el *supervisado* pues conocemos los datos de entrada y de salida. Tomando como referencia el diagrama de flujo anterior, explicaremos el funcionamiento de la red ANN durante el proceso de entrenamiento. Iniciamos con la definición de las funciones de activación tangente hiperbólica y sigmoide para la rutina del *Fordward Pass* y *Backward Pass*.

Definición de funciones en Scilab

La función tangente hiperbólica definida **afht**, recibe tres parámetros estos son: el vector del sesgo o *bias* de la primera capa llamado **b**, el vector de entrada **p** perteneciente a la matriz SAM y la matriz de los pesos sinápticos **w** que dependerá de la cantidad de neuronas en la primera capa. Se calcula la tangente hiperbólica a la suma ponderada de las funciones lineales mediante la función *tanh* de la librería de Scilab. El resultado es almacenado en la variable ***y*** mediante la instrucción $y = tanh(w * p + b)$;. A continuación mostramos su implementación en el algoritmo 3:

Algorithm 3 Función de activación Tangente Hiperbólica implementada en Scilab

```
//Forward Pass Tangente Hiperbólica
function[y]=afht(w,b,p)
    y=tanh(w*p+b);
end function
```

De la misma forma, definimos la sigmoide **afsig** que recibe la misma cantidad de parámetros pero referidos a la segunda capa, serían los vectores del *bias* **b2**, **FLout**(salida de la primera capa *y*, los pesos sinápticos **w2**. El cálculo de los valores de realiza con la instrucción $y = 1/(1 + exp(-(w * p + b)))$;. Mostramos su implementación en lenguaje de Scilab:

Algorithm 4 Función de activación Sigmoide implementada en Scilab

```
//Forward Pass Sigmoide
function [y]=afsig(w,b,p)
    y=1./(1+exp(-(w*p+b)));
end function
```

Por otra parte, la implementación de las funciones para el ***Backward Pass*** tienen en cuenta el algoritmo del **gradiente en descenso**, por lo que requieren de más parámetros de entrada específicamente 6 para la sigmoide, que definimos como ***BPSigmOutputLayer***, y 7 parámetros para la tangente hiperbólica llamada ***BPtanhInputLayer***. La función sigmoide recibe como entrada los vectores de salida de las funciones del *Forward Pass* ***SLout*** y ***FLout***. También la matriz de los pesos sinápticos y el vector del bias de la segunda capa, por otra parte se necesita el valor la razón de aprendizaje (η) y la matriz de errores calculada durante cada iteración. Devuelve tres parámetros, el primero es el valor de la derivada de la función de salida de la última capa multiplicada por la matriz de error(***err***), este resultado es almacenado en una variable llamada ***deltao*** que será empleada en el algoritmo del gradiente en descenso para calcular los nuevos valores de los pesos sinápticos (W_2) y los bias (B_2) de esa capa y será la señal de entrada de la próxima función. Recordar que en el Backward Pass el sentido de la transferencia es de la capa de salida hacia la de entrada. Las variable (W_2) y (B_2) serían el segundo y tercer parámetro de salida de esta función, los cuales serán también utilizada por la tangente hiperbólica que se explicará a continuación. Seguidamente mostramos el código de la función.

Algorithm 5 Función de activación Tangente hiperbólica implementada en Scilab

```
//Backward Pass Tangente Hiperbólica
function [wo, bo, deltao]=BPSigmOutputLayer(wo,bo,err,eta,slout,flout)
    deltao=(err.*(slout.*(1-slout)));
    wo=wo+eta*(deltao*flout');
    bo=bo+eta*deltao;
end function
```

Para la función tangente hiperbólica (***BPtanhInputLayer***) se necesitan los pesos sinápticos y bias de

la primera capa (w_1) y (b_1), los pesos, la salida de la segunda capa y el resultado de la función sigmoide en el Backward Pass, la salida de la tangente hiperbólica, además del coeficiente de aprendizaje y el vector de entrada a la red. El proceso es parecido a la sigmoide ya que se está aplicando la derivación en cadena a la respuesta proporcionada por la función (**deltao**) pero se debe de multiplicar por la derivada de la salida de la tangente en el Forward Pass (***FLout***), esta es la línea de código correspondiente $deltai = (1 - flout.^2). * (wo' * deltao)$, nuevamente se actualizan los pesos y el bias de la primera capa (W1) y (B1). Esta función también entrega tres resultados, la variable *deltai* que contiene la derivada resultante por la derivación en cadena de función de la capa anterior. Los otros parámetros son los pesos (W1) y los bias (B1) que serán nuevamente utilizados por la tangente hiperbólica en el Forward Pass. A continuación mostramos el código:

Algorithm 6 Función de activación Sigmoide implementada en Scilab

```
//Backward Pass sigmoide
function [wi,bi,deltai]=BPtanhInputLayer(wi,wo,bi,eta,flout,deltao,P)
    deltai = (1 - flout.^2). * (wo' * deltao);
    wi=wi+eta*(deltai*P');
    bi=bi+eta*deltai;
end function
```

Por otra parte, se cargan los parámetros de entrada provenientes del archivo generado por Scilab que se mencionó en el apéndice 4.2, esta será el vector de entrada que almacena la variable P.

Inicialización y configuración de variables

Primeramente, se obtienen las dimensiones para las variables que definen el tamaño del conjunto de entrenamiento (**TraininSetSize**) y el tamaño de los vectores de entradas (**InputVectorSize**), estos datos son obtenidos de la matriz de vectores *SAM* empleando el comando ***size*** de Scilab. De la misma forma, se determina la longitud de las variables de salida en el entrenamiento que son (**CodePerFault**) y (**CodeLength**), esta longitud se obtiene a partir de la matriz de salida *Zout* mencionada anteriormente. La matriz que contiene el error de salida por cada iteración durante el proceso del Forward Pass, se almacena en la variable **e** cuyo orden está definido por los zeros de (**CodePerFault**) y (**CodeLength**). La matriz **Y** muestra los códigos de salida que entrega la red y el vector **esum** almacena la sumatoria de los errores durante el entrenamiento. En esta parte del programa se inicializan las variables que se emplearán en el entrenamiento, tales como la cantidad de neuronas de la capa de entrada (**nin**), las neuronas de la capa de salida(**non**). Además, se definen la cantidad de elementos que tendrán los vectores del bias para la primera y segunda capa (**nbi**) e (**nbo**). Por último, se de definen la cantidad de se señales de entrada para la primera y segunda capa (**nis**) e (**nos**). Además, se inicializan los valores iniciales de los pesos sinápticos de las capas conjuntamente con los valores de los bias, estos valores se seleccionan de forma aleatoria utilizando la función ***rand*** de la librería Scilab. Este proceso sólo se realiza en la primer entrenamiento porque los valores de los pesos y los bias, cuando se actualizan, almacenan en un archivo generado por Scilab, el cual será el punto de partida para el próximo entrenamiento y así sucesivamente hasta que la red arroje los resultados esperados.

Funcionamiento del algoritmo Backpropagation implementado en Scilab

El Backpropagation es un algoritmo recursivo que depende de la cantidad de interacciones que se definan para el entrenamiento. En este algoritmo contamos con dos ciclos *for* anidados, en el for interno es donde se ejecutan las rutinas del Forward y el Backward Pass. El Forward es el primero, comenzando con la actualización del vector **P**, durante cada iteración en **P** se carga solo un vector de la matriz *SAM* el cual será la información de entrada de la primera capa. Seguidamente, se calcula la tangente hiperbólica con el código $FLout = afht(W1, B1, P)$ y ésta será la salida de la primera capa almacenada en **FLout**.

Posteriormente, se calcula la salida de la segunda capa mediante la función sigmoide aplicando la sentencia $SLout = afsig(W2, B2, FLout)$. El dato que proporciona la variable ***SLout*** se almacena en la matriz **Y** que será empleada durante el cálculo del error para ser restada a la matriz ***Zout*** mediante el comando $e(:, i) = Zout(:, i) - Y(:, i)$. Recordemos que ***Zout*** es nuestra matriz de referencia de salida, de esta forma concluye la rutina del Forward Pass.
En el Backward Pass solo intervienen dos funciones de activación, primero se ejecuta la sigmoide tomando los datos proporcionados durante el Forward Pass y el error de salida para actualizar el valor de los pesos sinápticos mediante el algoritmo del gradiente en descenso y para ello debemos de obtener la derivada de la función de salida que entrega ***SLout*** por lo que emplearemos la instrucción $[W2, B2, deltao] = BPSigmOutputLayer(W2, B2, e(:, i), eta, SLout, FLout)$. Recordemos que el Backward Pass fluye de la salida hacia la entrada, por lo que la salida ***deltao*** será utilizada por la tangente hiperbólica para actualizar el valor de los pesos sinápticos de la primera capa. Esto se logra mediante el código $[W1, B1, deltai] = BPtanhInputLayer(W1, W2, B1, eta, FLout, deltao, P)$ y aquí concluye el *for* interno.
En el ciclo externo además de ejecutarse el *for* interno, se calcula el error total de cada iteración y se compara con un ajuste predefinido que se escoge según el criterio del entrenador. El ciclo concluirá cuando se termine la cantidad de iteraciones programados para el entrenamiento (número de épocas) ó se cumpla la condición de que el error total sea menor al error esperado. Cuando termina el *for* externo se almacenan los datos actualizados de los pesos sinápticos, del bias y del coeficiente de entrenamiento en un archivo. Este fichero será cargado en el próximo entrenamiento y así se ejecutará el algoritmo nuevamente. El entrenamiento será llevado a cabo una y otra vez hasta que se cumpla la condición de que el error total sea menor que el ajustado.

4.3. Implementación del algoritmo en la Raspberry Pi

La Raspberry Pi es el dispositivo donde se ha implementado el algoritmo de la ANN pues presenta características importantes, para nuestra aplicación, que permiten el buen funcionamiento de la red, tales como la velocidad de procesamiento. El lenguaje de programación utilizado es el ***C*** y el entorno de desarrollo integrado es el *Geany* proporcionado por el propio sistema operativo Raspbian. El programa de la red tiene 3 partes, el primero es la declaración de variables y definición de funciones. La segunda es la implementación del algoritmo de diagnóstico de fallas y la tercera es la implementación de las funciones definidas.

Definición de variables y funciones

En esta sección sólo se declaran las variables que serán empleadas en el programa de ejecución, éstas se dividen en dos grupos. El primero tiene que ver con la configuración del GPIO, en esta parte se definen los terminales de entrada y salidas que serán utilizados para la transmisión y recepción de datos. Todas las variables de este grupo usan la directiva $\#define$ para utilizar etiquetas que se corresponden con los números de los terminales por ejemplo: LED_0, LED_1, LED_2, LED_3, LED_INT1, LED_INT2, size_D1 entre otras. El segundo grupo contiene variables que se utilizan en las operaciones matriciales fundamentalmente del tipo apuntador **double**. También, tenemos variables tipo constantes que almacenan valores utilizados para la operación del algoritmo neuronal, tales como los pesos sinápticos, los bias, los vectores de pruebas, los grupos de fallas entre otros, aquí tenemos: **B1**, **B2**, **W1** y **W2** y los arreglos tipo **int** G0, G1, G2 hasta G35 contienen los grupos de vectores. Además contamos con **F_lout** y **S_lout** que son variables de tipo *double* que almacenan los resultados de las funciones de activación tangente y sigmoide. Se definieron variables del tipo apuntador para el trabajo con las operaciones matriciales funcionalmente resaltamos: **prt_w1, prt_w2,prt_B1,prt_B2,prt_P, prt_F_lout, prt_S_lout , prt_vect_A, rt_vect_B, prt_vect_C, prt_Vec_out,prt_w1,prt_w1**. Por otra parte, tenemos la declaración de funciones que también están definidas por categorías. Existe un grupo que se encarga de las operaciones con matrices tales como multiplicación, suma y transposición, estas funciones tienen

la características de no retornar valores porque modifican los resultados por mediación de apuntadores. Estas funciones son: **Trans_Matrix**, **Mult_Matrix** y **Sum_Matrix**. También tenemos las funciones que son utilizadas en el algoritmo neuronal éstas son **afht** función tangente hiperbólica, **afsig** sigmoide. Estas operan de la mismas forma que las funciones de activación utilizadas durante el *Fordward Pass*. Por último, tenemos las funciones que se encargan de acondicionar los datos para el algoritmo de la red éstas son tipo **void** y estas son: **encoder_vector**, **decoder_vector**, **identifier_code**.

Algorithm 7 Algoritmo de la red neuronal en Pseudocódigo implementado en la Raspberry Pi 4B

//***Definición de variables y Funciones***
1: Función **afht**,**afsig**,**Text_Forward Pass** //Funciones de activación para el Forward Pass
2: Función **encoder_vector**,textbfdecoder_vector //Función que convierte los valores 1 a 0.8 y 0 a 0.2
3: Función **Identifier_code**,**compare_vector** //Función que realiza el proceso de identificación de código
4: Función **compare_vector** //Compara la respuesta de la red con el código de salida definido
5: Función **vector_A**,**vector_B**, **vector_E** //Envío de los vectores de prueba
6: Función **Trans_Matrix**, **Mult_Matrix** y **Sum_Matrix** // Funciones que se utilizan para las operaciones con matrices
7: //***variables globales***
8: **LED_0**, **LED_1**, **LED_2**, **LED_3**, **LED_4**, **LED_INT1** y **LED_INT2** // Conf del **GPIO**
9: *W1* y *W2* = cantidad de pesos sinápticos de la primera capa y segunda capa
10: *B1* y *B2* = cantidad de bias de la primera y segunda capa
11: **F_lout** y **S_lout**// almacenamiento de los resultados de salida de primera y segunda capa
12: **Vect_in_A**, **Vect_in_B** hasta **Vect_in_E**// contienen las repuestas de los vectores de prueba
13: G0,G1, G2, G3, G4, G5, G6 hasta G35// constantes que tienen los códigos de salidas
14: variables del tipo apuntador definidas para poder manipular arreglos bidimensionales en las operaciones con matrices:
15: **prt_w1, prt_w2,prt_B1,prt_B2,prt_P, prt_F_lout, prt_S_lout , prt_vect_A, rt_vect_B, prt_vect_C, prt_Vec_out,prt_w1,prt_w1**
16: //**Implementación del algoritmo**
17: configuración del **GPIO**
18: **vector_A**// Envío del primer vector de prueba
19: **Text_Forward Pass**// aplica el algoritmo de la red neuronal
20: **tiempo**
21: **Identifier_code**
22: **while(prt_Vec_out == G0)**
23: **vector_VP**// Envío vector de prueba
24: **Text_Forward Pass**// aplica el algoritmo de la red neuronal
25: **tiempo**
26: **Identifier_code**
27: **contador++**
28: **end_while**// aplica el algoritmo de la red neuronal
29: **if**(contador==total)
30: Imprime *CIRCUITO SANO*
31: **else**
32: Imprime *CIRCUITO DEFECTUOSO*
33: **end_if**
34: **Fin**

Implementación de funciones

Debido a la importancia que tienen las operaciones matriciales en nuestro algoritmo se han implementado funciones para realizar la suma, multiplicación y transposición de matrices. Para facilitar el trabajo con arreglos dimensionales se ha aprovechado la aplicación de los apuntadores, ya que no es posible el retorno de arreglos en lenguaje C. Los métodos encargados para estas operaciones son:**Trans_Matrix**, **Mult_Matrix** y **Sum_Matrix**. Estas funciones no retornan los resultados, simplemente los modifican mediante punteros, para el caso de la suma y la multiplicación reciben la misma cantidad de parámetros punteros tipo double. Para la transferencia de los vectores de prueba tenemos las funciones **vector_A**,**vector_B** hasta **vector_E**, todas funcionan de la misma forma pero se diferencian en la configuración de los terminales de salida del GPIO. Existe una demora de 100 milisegundo para garantizar la recepción de la respuesta del FPGA, para esto se emplea la función **read_vector**. Para el almacenamiento de la recepción de datos se utilizan las variables **Vect_in_A**, **Vect_in_B** hasta**Vect_in_E** las cuales son modificadas por la función **encoder_vector** que tiene como objetivo convertir los datos de 1 a 0.8 y de 0 a 0.2. A diferencia del proceso de entrenamiento, una vez entrenada la red sólo se utiliza para el diagnóstico de la falla el Forward Pass y de esto se encarga la función **Text_Forward Pass**. Dentro de **Text_Forward Pass**, se ejecuta la función tangente hiperbólica **afht**, cuyo resultado se almacena en la variable **F_lout**. La función sigmoide **afsig** encargada de calcular la salida de la segunda capa que se almacena en **S_lout**. Este resultado es procesado por la función **decoder_vector** que convierte los valores de 0.8 a 1 ó 0.2 a 0 y los almacena en ***vect_out***. Finalizada esta operación se procede a comparar la salida con los códigos que se almacenan en los arreglos G0 hasta G7 mediante el algoritmo **Identifier_code**.

4.4. Programación de fallas para el circuito ISCAS'85 C17 en el FPGA

El C17 es un circuito combinacional de cinco entradas y dos salidas primarias que es fácil de implementar en el lenguaje descriptivo ***Verilog***. Para simular las fallas Stuck-at se programó un multiplexor de cinco señales de selección donde cada combinación representa una s-a-v. Físicamente estas señales son controladas por un switch de cinco selectores que se encuentra conectado al FPGA. Como en nuestro proyecto no se contemplaron las Stuck-at múltiples solo se simularon 28 fallas entre S-a-0 y S-a-1. La información de salida se envía a la Raspberry Pi de forma alámbrica por los terminales 34 y 38 del banco 0 del FPGA, además se muestra en dos display siete segmentos, de la tarjeta Altera, el vector de entrada con su salida. El módulo principal recibe una señal de reloj y dos vectores de entrada de cinco bits llamados **vector** y **fault_c**, el primero contiene la longitud del vector de prueba y el segundo el código referente a la falla en el circuito. Como señales de salida tenemos tres vectores (**out_c17,out_seg y out_seg1**), el primero contiene la salida que se enviará a la Raspberry Pi, y los restantes se utilizan en la visualización en display siete segmentos. Tenemos variables tipo wire y reg que se utilizan en la implementación del circuito por ejemplo *W10*,*W11*,*W16*,*W19*. Tenemos dos constantes tipo *paramenter* que son *Fstk_0* y *Fstk_1* encargadas de introducir S-a-0 o S-a-1 según sea el caso, el pseudocódigo de este programa es mostrado en el algoritmo 8. Una vez que se recibe el vector de prueba, en dependencia de cómo estén configurado el switch, la salida se genera con la falla programada y es enviada a la Raspberry.

4.5. Programación de fallas para el circuito ISCAS'85 C432 en el FPGA

El C432 fue programado, teniendo en cuenta su complejidad, mediante dos niveles de descripción: el nivel estructural y el nivel de comportamiento. El programa se conforma por siete módulos, el primero es el módulo *Circuit432*, tenemos el *TopLevel 432*, los módulos que controlan las prioridades de los buses A,B y C que son: *PriorityA*, *PriorityB* y *PriorityC*, por último los dos módulos restantes que son *DecodeChan*, *EncodeChan*. El primero es el módulo principal donde se conecta funcionalmente las señales de entrada salida físicas con las internas empleadas por los módulos restantes y de esta forma realizar la función del C432. El módulo *TopLevel432* tiene el objetivo probar de forma concurrente la funcionalidad de los

módulos de prioridad de los buses (A,B y C), el codificador y decodificador de canales. Esta estructura recibe como entrada cuatro registros de 9 bits y da salida a siete señales PA, PB, PC, y los cuatro bit del canal o sea la salida del C432 de forma interna. Los módulos *PriorityA*, *PriorityB* y *PriorityC* funcionan de forma idéntica pero su diferencia está en las señales que reciben y dan salida. Cada uno recibe el bus E y su respectivo bus por ejemplo para *PriorityA* se le asigna el bus A. Entregan dos salidas de un bit, una es la señal de salida primaria por ejemplo PA para *PriorityA* y la otra es utilizada por los módulo de menor prioridad. La estructura del decodificador recibe todas las señales primarias como entrada conjuntamente con todos los buses de entadas A, B, C y D como salida tenemos el bus I que será utilizado en el módulo codificador. Este codificador de prioridad es de 9 a 4 bits, estos bits conforman la parte menos significativa de la salida primaria. La simulación de fallas se produce de la misma manera que en el C17 por medio de un multiplexor pero de mayor señales de selección, éstas son controladas por el arreglo de switch que tiene la tarjeta de desarrollo. La información de salida es mostrada en los display de siete segmentos y enviada a la Raspberry por los pines:38, 39, 42, 43, 44, 46 y 49 para ser procesada por el algoritmo neuronal. Para proporcionar los datos de entrada al circuito se ha implementado un módulo que se encarga de organizar los datos enviados por la Raspberry, éstos son organizados de 9 bits en un registro de 36 bits, de forma tal que se realizan cuatro desplazamiento, partiendo del bits menos significativo del dato. La entrada se produce de forma sincrónica y es la Raspberry la que indica en el último envío del paquete de datos cuando debe de comenzar el programa descrito en el FPGA. Esta información es recibida por el FPGA en los pines 1, 2, 3, 7, 10, 28, 30, 31 y 32 Se ha mostrado parte del funcionamiento del programa descriptivo en pseudocódigo mostrado en el algoritmo 9.

Algorithm 8 Programa para la generación de fallas Stuck-at en el C17 en lenguaje Verilog

```
//Definición  módulo principal
// Definición de variables
señales de entradas: vector (5 bits), fault_c(5 bits) y reloj (1 bit)
señales de salidas: out_c17 (2 bits) ,out_seg (7 bits) y out_seg1 (7 bits)
// variables internas
registro 1bits: W10,W11,W16,W19
constante:Fstk_0=0, Fstk_1=1
// programa principal
delay // demora para el refrescamiento de los 7 segmentos
Generación de la falla // activación de los switchs de la tarjeta
case (fault_c="00000")
   algoritmo de circuito sano// Circuito sano código F00
case (fault_c="00001")
   algoritmo de circuito defectuoso// Circuito defectuoso código F01_1
⋮
case (fault_c="01110")
   algoritmo de circuito defectuoso// Circuito defectuoso código F01_18
case (fault_c="01111")
   algoritmo de circuito defectuoso// Circuito defectuoso código F10_1
⋮
case (fault_c="11100")
   algoritmo de circuito defectuoso// Circuito defectuoso código F10_18
Envío de datos// Envío de datos al Raspberry Pi
Visualizar// visualización en 7 segmentos
```

[1]

Algorithm 9 Programa para la generación de fallas Stuck-at en el C432 en lenguaje Verilog

//***Definición*** *módulo principal*

1: // Definición de variables
2: señales de entradas: **BUS A,B,C y E** (9 bits), **fault_c**(6 bits) y reloj (1 bit)
3: señales de salidas: **out_C432 (5 bits)** ,**out_seg (7 bits)** y **out_seg1 (7 bits)**
4: //Declaración e Implementación de los módulos internos
5: *módulos*: **Circuit432**,**TopLevel432**,**PriorityA**,**PriorityB**,**PriorityC**
6: **DecodeChan**,**EncodeChan**,**Circular buffer**
7: *constante*:Fstk_0=0, Fstk_1=1
8: // programa principal
9: *lectura de los datos de entada* // aplicación del modulo circular Buffer
10: *Generación de la falla* // activación de los switchs de la tarjeta
11: **delay** // demora para el refrescamiento de los 7 segmentos
12: **case** (fault_c="000000")
13: **algoritmo de circuito sano**// Circuito sano código F00
14: **case** (fault_c="000001")
15: **algoritmo de circuito defectuoso**// Circuito defectuoso código M11-F01
16: ⋮
17: **case** (fault_c="001110")
18: **algoritmo de circuito defectuoso**// Circuito defectuoso código M55-F01
19: **case** (fault_c="100010")
20: **algoritmo de circuito defectuoso**// Circuito defectuoso código M310-F10
21: ⋮
22: **case** (fault_c="100011")
23: **algoritmo de circuito defectuoso**// Circuito defectuoso código M53-F10
24: **Envío de datos**// Envío de datos al Raspberry Pi
25: **Visualizar**// visualización en 7 segmentos

[1]

Capítulo 5

Resultados y Conclusiones

5.1. Resultados

Una vez que la red neuronal entrega los parámetros deseados con un error menor al esperado entonces podemos decir que se encuentra entrenada. En la Tabla 5.1, el parámetro de salida, para el circuito C17, es la matriz cuyo orden es de 10×4 que está representada por la columna *Out ANN* y contiene los resultados del entrenamiento que hacen referencia al código entregado por la red. También observamos los parámetros esperados que están identificados por la columna *Zout*, por último tenemos la columna *Código*.

OutANN	**Zout**	**Código**
0.1976759 0.1931429 0.2027852 0.1977035	0.2 0.2 0.2 0.2	0 0 0 0
0.2066807 0.195903 0.2016048 0.7977912	0.2 0.2 0.2 0.8	0 0 0 1
0.1997988 0.1910622 0.8002545 0.2009248	0.2 0.2 0.8 0.2	0 0 1 0
0.1999317 0.2106897 0.7981111 0.7973188	0.2 0.2 0.8 0.8	0 0 1 1
0.1917958 0.2188909 0.1934185 0.2028462	0.2 0.2 0.2 0.2	0 0 0 0
0.2069694 0.7960799 0.2033667 0.2005791	0.2 0.8 0.2 0.2	0 1 0 0
0.1986161 0.7910827 0.2013166 0.8027812	0.2 0.8 0.2 0.8	0 1 0 1
0.2020604 0.1926994 0.2022503 0.2008438	0.2 0.2 0.2 0.2	0 0 0 0
0.2009751 0.8002008 0.8000727 0.2009045	0.2 0.8 0.8 0.2	0 1 1 0
0.1959904 0.8093458 0.7967115 0.7969283	0.2 0.8 0.8 0.8	0 1 1 1

Tabla 5.1: Tabla con la información de los datos de salida a la red neuronal del C17.

Este código corresponde a un grupo de fallas que se encuentran en lugares específicos del circuito y que solamente son detectadas por un vector de prueba, de esta forma podemos lograr que nuestro algoritmo pueda diagnosticar fallas Stuck-at simples.

Código	**Grupos de fallas**
0 0 0 0	F00 Healthy circuit
0 0 0 1	Faults: F10-1, F10-2, F01-18
0 0 1 0	Faults: F10-4, F01-7, F01-8
0 0 1 1	Faults: F10-14,F01-15, F10-16
0 1 0 0	Faults: F01-1, F10-3, F01-15
0 1 0 1	Faults: F01-2, F01-4, F01-5, F10-7, F10-10, F01-12, F01-16, F10-18
0 1 1 0	Faults: F10-1, F01-3, F10-8, F01-14, F01-15
0 1 1 1	Faults: F10-5, F01-10, F01-11, F10-12, F01-18

Tabla 5.2: Tabla que relaciona el código de salida con el grupo de falla del C17.

OutANN	Zout
0.1997264 0.2000016 0.2001314 0.2000904 0.2001197 0.1999761	0.2 0.2 0.2 0.2 0.2 0.2
0.1999827 0.200044 0.2000049 0.2000174 0.19996 0.8000325	0.2 0.2 0.2 0.2 0.2 0.8
0.2002679 0.2000356 0.1999375 0.1999897 0.8000212 0.2000111	0.2 0.2 0.2 0.2 0.8 0.2
0.1995468 0.2 0.199961 0.2000785 0.7999436 0.7999159	0.2 0.2 0.2 0.2 0.8 0.8
0.200451 0.2001546 0.1999068 0.7998888 0.1999426 0.1998675	0.2 0.2 0.2 0.8 0.2 0.2
0.2001081 0.1998472 0.2000427 0.7999337 0.1999994 0.8000974	0.2 0.2 0.2 0.8 0.2 0.8
0.2002988 0.2000083 0.1999781 0.2000362 0.1998876 0.1999761	0.2 0.2 0.2 0.2 0.2 0.2
0.1990772 0.2000588 0.2000702 0.8000595 0.7998668 0.1999605	0.2 0.2 0.2 0.8 0.8 0.2
0.200218 0.2000202 0.2000059 0.7999968 0.8000464 0.8000826	0.2 0.2 0.2 0.8 0.8 0.8
0.2001243 0.199902 0.8000164 0.20004 0.2000951 0.2001481	0.2 0.2 0.8 0.2 0.2 0.2
0.199957 0.2000333 0.7999938 0.1999473 0.1999916 0.7999682	0.2 0.2 0.8 0.2 0.2 0.8
0.2005333 0.1999622 0.7999118 0.1999868 0.8001119 0.1999647	0.2 0.2 0.8 0.2 0.8 0.2
0.1982958 0.1999028 0.2001231 0.1998088 0.200151 0.1999547	0.2 0.2 0.2 0.2 0.2 0.2
0.2000773 0.2000638 0.800045 0.1999772 0.8000096 0.7999188	0.2 0.2 0.8 0.2 0.8 0.8
0.2005177 0.200054 0.7999883 0.8000308 0.1999311 0.1999873	0.2 0.2 0.8 0.8 0.2 0.2
0.2002715 0.2000848 0.8000126 0.8000652 0.2000283 0.8000317	0.2 0.2 0.8 0.8 0.2 0.8
0.2001352 0.1999417 0.7999723 0.8000329 0.7999571 0.2001259	0.2 0.2 0.8 0.8 0.8 0.2
0.1994034 0.1999647 0.8000213 0.7999337 0.8000012 0.7999901	0.2 0.2 0.8 0.8 0.8 0.8
0.2002911 0.7999942 0.199946 0.2000445 0.2000004 0.1999962	0.2 0.8 0.2 0.2 0.2 0.2
0.2011324 0.8000101 0.1999178 0.2001564 0.1999314 0.800068	0.2 0.8 0.2 0.2 0.2 0.8
0.1983818 0.1999923 0.1997072 0.1999642 0.1998215 0.2001988	0.2 0.2 0.2 0.2 0.2 0.2
0.1998442 0.7999761 0.2000358 0.1998635 0.8000652 0.1998988	0.2 0.2 0.2 0.2 0.2 0.2
0.1994958 0.7999368 0.2001716 0.2001304 0.799967 0.8001338	0.2 0.8 0.2 0.2 0.8 0.2
0.1991226 0.8000548 0.1997705 0.8000158 0.1999068 0.2000538	0.2 0.8 0.2 0.2 0.8 0.8
0.200685 0.8000209 0.2001546 0.7999771 0.2000683 0.7999654	0.2 0.8 0.2 0.8 0.2 0.2
0.2001134 0.7999855 0.200066 0.7999995 0.799959 0.1999958	0.2 0.8 0.2 0.8 0.2 0.8
0.2020224 0.2000227 0.2000838 0.2000096 0.2001544 0.1997221	0.2 0.8 0.2 0.8 0.8 0.2
0.2003451 0.7999052 0.199915 0.800071 0.8001228 0.7999639	0.2 0.2 0.2 0.2 0.2 0.2
0.1996476 0.8000586 0.7999305 0.1999991 0.1999123 0.1999966	0.2 0.8 0.2 0.8 0.8 0.8
0.1996965 0.7998355 0.7999617 0.2000291 0.1999236 0.7998614	0.2 0.8 0.8 0.2 0.2 0.2
0.2001358 0.7999601 0.7999843 0.2000411 0.8000024 0.2000564	0.2 0.8 0.8 0.2 0.2 0.8
0.1998876 0.8001378 0.7999868 0.1999016 0.7999892 0.8000564	0.2 0.8 0.8 0.2 0.8 0.2
0.1998202 0.8001376 0.8001518 0.7999589 0.2000828 0.2000533	0.2 0.8 0.8 0.8 0.2 0.2
0.2005098 0.199938 0.2001348 0.1999602 0.2000293 0.2000431	0.2 0.2 0.2 0.2 0.2 0.2
0.1997542 0.7999212 0.7999664 0.7999671 0.1999847 0.8000181	0.2 0.8 0.8 0.8 0.2 0.8
0.2001086 0.7999182 0.8000543 0.8000914 0.8000366 0.1999954	0.2 0.8 0.8 0.8 0.8 0.2
0.1999341 0.8001177 0.7999642 0.7999302 0.7999715 0.7999365	0.2 0.8 0.8 0.8 0.8 0.8
0.799786 0.200006 0.1999714 0.1999226 0.1999987 0.1999686	0.8 0.2 0.2 0.2 0.2 0.2
0.7993855 0.1999909 0.1999812 0.1999347 0.2000453 0.7999355	0.8 0.2 0.2 0.2 0.2 0.8
0.8006631 0.1999307 0.1999561 0.2001071 0.7999523 0.2001255	0.8 0.2 0.2 0.2 0.8 0.2
0.8002095 0.2000806 0.200063 0.2000259 0.8000343 0.7999796	0.8 0.2 0.2 0.2 0.8 0.8

Tabla 5.3: Tabla con la información de los datos de salida a la red neuronal de C432.

Zout	Código	Grupos de fallas
0.2 0.2 0.2 0.2 0.2 0.2	0 0 0 0 0 0	G0, Healthy Circuit
0.2 0.2 0.2 0.2 0.2 0.8	0 0 0 0 0 1	G1, **M11-F01**,**M12-F01**, **M13-F01**
0.2 0.2 0.2 0.2 0.8 0.2	0 0 0 0 1 0	G2, **M55-F01**
0.2 0.2 0.2 0.2 0.8 0.8	0 0 0 0 1 1	G3, **M24-F10**
0.2 0.2 0.2 0.8 0.2 0.2	0 0 0 1 0 0	G4, **M53-F10**,**M54-F10**
0.2 0.2 0.2 0.8 0.2 0.8	0 0 0 1 0 1	G5, **M53-F10**,**M54-F10**
0.2 0.2 0.2 0.2 0.2 0.2	0 0 0 0 0 0	G0, Healthy Circuit
0.2 0.2 0.2 0.8 0.8 0.2	0 0 0 1 1 0	G6, **M22-F01**,**M23-F01**,**M24-F01**,**M21-F10**
0.2 0.2 0.2 0.8 0.8 0.8	0 0 0 1 1 1	G7, **M33-F01**,**M35-F01**,**M36-F01**,**M37-F01** **M38-F01**, **M39-F01**,**M310-F01**
0.2 0.2 0.8 0.2 0.2 0.2	0 0 1 0 0 0	G8, **M47-F01**,**M54-F01**
0.2 0.2 0.8 0.2 0.2 0.8	0 0 1 0 0 1	G9, **M55-F01**,**M57-F01**,**M56-F10**
0.2 0.2 0.8 0.2 0.8 0.2	0 0 1 0 1 0	G10,**M42-F10**,**M58-F10**
0.2 0.2 0.2 0.2 0.2 0.2	0 0 0 0 0 0	G0, Healthy Circuit
0.2 0.2 0.8 0.2 0.8 0.8	0 0 1 0 1 1	G11,**M58-F01**,**M56-F10**
0.2 0.2 0.8 0.8 0.2 0.2	0 0 1 1 0 0	G12,**M11-F10**,**M13-F10**
0.2 0.2 0.8 0.8 0.2 0.8	0 0 1 1 0 1	G13,**M24-F10**
0.2 0.2 0.8 0.8 0.8 0.2	0 0 1 1 1 0	G14,**M31-F10**,**M39-F10**,**M310-F10**
0.2 0.2 0.8 0.8 0.8 0.8	0 0 1 1 1 1	G15,**M310-F10**
0.2 0.8 0.2 0.2 0.2 0.2	0 1 0 0 0 0	G16,**M53-F10**
0.2 0.8 0.2 0.2 0.2 0.8	0 1 0 0 0 1	G17,**M57-F10**
0.2 0.2 0.2 0.2 0.2 0.2	0 0 0 0 0 0	G0, Healthy Circuit
0.2 0.8 0.2 0.2 0.8 0.2	0 1 0 0 1 0	G18,**M11-F01**,**M12-F01**,**M13-F01**,**M14-F01** **M15-F01**
0.2 0.8 0.2 0.2 0.8 0.8	0 1 0 0 1 1	G19,**M39-F10**,**M310-F10**
0.2 0.8 0.2 0.8 0.2 0.2	0 1 0 1 0 0	G20,**M21-F10**
0.2 0.8 0.2 0.8 0.2 0.8	0 1 0 1 0 1	G21,**M45-F10**
0.2 0.8 0.2 0.8 0.8 0.2	0 1 0 1 1 0	G22,**M56-F01**
0.2 0.8 0.2 0.8 0.8 0.8	0 1 0 1 1 1	G23,**M53-F10**,**M54-F10**
+ − qw0.2 0.2 0.2 0.2 0.2 0.2	0 0 0 0 0 0	G0, Healthy Circuit
0.2 0.8 0.8 0.2 0.2 0.2	0 1 1 0 0 0	G24,**M35-F01**,**M36-F01**,**M37-F01**,**M38-F01** **M39-F01**,**M310-F01**
0.2 0.8 0.8 0.2 0.2 0.8	0 1 1 0 0 1	G25,**M55-F01**,**M57-F01**
0.2 0.8 0.8 0.2 0.8 0.2	0 1 1 0 1 0	G26,**M24-F10**
0.2 0.8 0.8 0.2 0.8 0.8	0 1 1 0 1 1	G27,**M46-F10**
0.2 0.8 0.8 0.8 0.2 0.2	0 1 1 1 0 0	G28,**M53-F10**,**M54-F10**
0.2 0.8 0.8 0.8 0.2 0.8	0 1 1 1 0 1	G29,**M58-F10**
0.2 0.2 0.2 0.2 0.2 0.2	0 0 0 0 0 0	G0,Healthy Circuit
0.2 0.8 0.8 0.8 0.8 0.2	0 1 1 1 1 0	G30,**M55-F01**,**M57-F01**
0.2 0.8 0.8 0.8 0.8 0.8	0 1 1 1 1 1	G31,**M58-F01**
0.8 0.2 0.2 0.2 0.2 0.2	1 0 0 0 0 0	G32,**M13-F10**
0.8 0.2 0.2 0.2 0.2 0.8	1 0 0 0 0 1	G33,**M24-F10**
0.8 0.2 0.2 0.2 0.8 0.2	1 0 0 0 1 0	G34,**M310-F10**
0.8 0.2 0.2 0.2 0.8 0.8	1 0 0 0 1 1	G35,**M53-F10**,**M54-F10**

Tabla 5.4: Tabla que relaciona el código de salida con el grupo de falla del C432.

En la Tabla 5.2 se muestra una correspondencia entre el código generado representado por la columna *Código* y el campo que incluye los grupos de fallas asociados para el circuito C17.

También en la primera columna de la Tabla 5.3 obtuvimos la matriz de salida, en el campo *Out ANN*, para el C432. De la misma forma que el circuito C17 podemos apreciar que existe una gran semejanza entre el código arrojado por la red y el esperado que se encuentra representado por la columna *Zout*. Los valores que entrega la red se encuentran cerca de 0.2 para 0 y 0.8 para el 1, esto fue necesario para facilitar el entrenamiento debido al tipo de algoritmo utilizado, *Backprogation*, y para evitar una posible saturación ya que son valores cercanos a los puntos límites de la función de activación de salida, *sigmoide*. Los grupos de fallas correspondientes a cada código son mostrados en el campo *Grupos de falla* en la Tabla 5.4, cada grupo representa las fallas que solo pueden ser detectadas por un vector de prueba determinado. Estas fallas son detectadas en interconexiones específicas y son únicas para cada vector. Los grupos de fallas que detecta un vector no son detectables por otro salvo algunas zonas que son compartidas. Recordamos que todos los grupos de fallas fueron simulados teniendo en cuenta la ruta más critica según los por cientos de capacitancias parásitas existentes.

5.2. Conclusiones

Con el objetivo de diagnosticar fallas del tipo Stuck-at en circuitos combinacionales ISCAS'85, se diseñó un algoritmo basado en redes neuronales artificiales el cual fue implementado en un microordenador Raspberry Pi. El modelo Stuck-at fue utilizado para comprender como se genera este tipo de falla en las interconexiones de los circuitos integrados y al mismo tiempo como realizar un correcto análisis lógico y funcional del mismo. Otro punto importante fue saber como funcionan los diferentes tipos de testing y cual es su funcionalidad en dependencia del tipo de circuito que se esté analizando, por ejemplo el testing que se aplicó al circuito C432 fue estructural debido a las complejidades que presenta este circuito en cuanto al número de compuertas e interconexiones. Estos conocimientos sirvieron de base para la construcción de los circuito simuladores de fallas realizados con la herramienta CAD de Labcenter Electronic llamada Proteus. Estos simuladores de fallas fueron muy importantes en la investigación ya que nos permitieron generar una base de datos con la que se construyeron los modelos neuronales para cada circuito de prueba.

Además los datos extraídos fueron de vital importancia para entender cual era la relación que existía entre los vectores de prueba y las fallas Stuck-at simuladas. Al observar este conjunto de datos nos dimos cuenta de que existían fallas internas que no provocaban cambios en la salida del circuitos y tenían que ser descartadas para el análisis de detección, a estas fallas se les conoce con el nombre de enmascaradas. Además se observó que existen subconjuntos de vectores de pruebas que son capaces de detectar las mismas fallas en una zona determinada del circuito y que un solo vector de prueba no es capaz de detectar una única falla. Esto fue muy ventajoso pues con un solo vector, de ese subconjunto, se pudo diagnosticar la mayor cantidad de fallas posibles con la menor cantidad de vectores de pruebas reduciendo el tiempo de ejecución del programa. De esta manera se logró identificar un grupo específico de falla con cada vector de prueba. Aunque existen zonas que son compartidas por estos vectores ya que son capaces de detectar las mismas fallas.

Para poder cumplir con el objetivo general fue fundamental diseñar la estructura de la red neuronal partiendo de la base de datos construida y el tipo de aprendizaje que sería aplicado en dependencia de los resultados que se esperaban. La red se entrenó mediante el aprendizaje supervisado empleando la herramienta de software libre Scilab, para ello fue necesario comprender como funcionaba el Backprogation que es el algoritmo fundamental que se aplica en este tipo de entrenamiento y el algoritmo del gradiente en descenso. Esto también era necesario para poder implementar el programa en lenguaje C una vez que se validara el entrenamiento. El modelo obtenido para cada circuito guarda mucha relación con los vectores de pruebas seleccionados para dicho entrenamiento, para el circuito C17 se escogieron tres vectores de

pruebas sin embargo para el circuito C432 se obtuvieron seis. Esto no quiere decir que esta sea la cantidad máxima de vectores a escoger sin embargo debe existir una relación entre sensibilidad y rapidez del algoritmo, además que se deben evitar ciertos problemas de como por ejemplo la correlación de datos. El modelo obtenido de la red se formó por dos capas y las funciones de activación empleadas fueron la tangente hiperbólica y la sigmoide. Otro punto importante fue que se utilizó la misma topología de la red neuronal para diagnosticar las fallas Stuck-at tanto en el circuito C17 como en el C432, la diferencia está que no se utilizaron los mismos datos para realizar el entrenamiento debido a que existen diferencias tanto funcionales como estructurales.

Mediante la programación en lenguaje C se pudo implementar este algoritmo neuronal en la Raspberry Pi donde se validó de forma satisfactoria y pudo comprobarse su correcto funcionamiento. Además fue necesario la implementación de funciones matemáticas, lógicas y de comunicación para garantizar la transferencia de datos entre la Raspberry y el FPGA. El medio físico que se utilizó para garantizar la transmisión fue el cobre y la transferencia de datos se realizó de forma paralela. Por otra parte fue necesario programar en el FPGA , en lenguaje Verilog, los circuitos combinacionales ISCAS'85 C17 y C432 que fueron utilizados como circuitos de prueba. Las simulaciones de fallas Stuck-at en el FPGA fueron realizadas utilizando un arreglo de switch que responde a un tipo de falla Stuck-at en un lugar específico en el circuito, de esta manera se comprobó de forma eficiente el funcionamiento del algoritmo en el hardware propuesto.

El algoritmo implementado es capaz de diagnosticar fallas Stuck-at simples en los circuitos ISCAS'85 antes mencionado, además almacena los resultados para que puedan ser posteriormente analizados. De esta manera se pudo demostrar que las redes neuronales artificiales pueden ser aplicadas perfectamente para diagnosticar fallas del tipo Stuck-at en circuitos VLSI.

Para trabajos futuros recomendamos que se realice un algoritmo que sea capaz de detectar de forma automática la cantidad de vectores de pruebas mínimos necesarios para entrenar la red neuronal pues este proceso actualmente lo realizamos desde el punto de vista manual. Además Se sugiere que se implemente una función en la Raspberry Pi que logre transferir los datos en serie para vectores de prueba cuya longitud de bits sea considerable debido a que existen circuitos ISCAS'85 con mayor cantidad de números de entradas por lo que es recomendable utilizar un protocolo de comunicación serie. Es necesario entrenar la red con mas vectores de prueba para lograr un mayor por ciento de generalización y por último recomendamos aplicar el algoritmo neuronal a circuitos ISCAS más complejos.

Capítulo 6

Anexos

A. Código del algoritmo desarrollado en el software Scilab

```
//Forward Pass hyperbolic-Tangent Activation Function
    function [y]=afht(w,b,p)
        y=tanh(w*p+b);
    endfunction
//Forward Pass Sigmoid Activation Function
    function [y]=afsig(w,b,p)
       y=1./(1+exp(-(w*p+b)));
    endfunction
```

Listing 6.1: Forward pass functions.

```
//Backpropagation for a Hyperbolic-Tangent Activation Function (output layer)
    function [wo,bo,deltao]=BPtanhOutputLayer(wo,bo,err,eta,slout,flout)
        deltao=err.*(1-(slout.^2));
        wo=wo+eta*(deltao*flout');
        bo=bo+eta*deltao;
    endfunction
//Backpropagation for a Sigmoid Activation Function (output layer)
    function [wo, bo, deltao]=BPSigmOutputLayer(wo,bo,err,eta,slout,flout)
        deltao=(err.*(slout.*(1-slout)));
        wo=wo+eta*(deltao*flout');
        bo=bo+eta*deltao;
    endfunction
```

Listing 6.2: Backward pass functions.

```
        [TrainingSetSize,InputVectorSize]=size(SAM);
        [CodePerFault,CodeLength]=size(Zout);
        e=zeros(CodePerFault,TrainingSetSize); // rows=CodePerFault, cols=
    TrainingSetSize
        esum=zeros(1,TrainingSetSize);
        Y=zeros(CodePerFault,TrainingSetSize);

        //Definiton of the ANN wights, inputs and outputs
        nin=24;                          // Number of Input Neurons de entrada
        non=CodePerFault;                // Number of Output Neurons de salida
        nbi=nin;                         // Number of Input bias
        nbo=non;                         // Number of Output bias
        nis=InputVectorSize;             // Number of Input Signals
        nos=nin;                         // Number of Input Signals to the Output Layer
        for k=1:epochs
            //  Forward pass
            for i=1:TrainingSetSize
```

```
                P=[SAM(i,:)'];
                FLout=afht(W1,B1,P);
                SLout=afsig(W2,B2,FLout);
                Y(:,i)=SLout;                // ANN model output
                e(:,i)=Zout(:,i)-Y(:,i);        // modeling error
            //  Backward pass - Back Propagation
                [W2,B2,deltao]=BPSigmOutputLayer(W2,B2,e(:,i),eta,SLout,FLout);
                [W1,B1,deltai]=BPtanhInputLayer(W1,W2,B1,eta,FLout,deltao,P);
            end
        //  Sum of the error
        esum(1,k)=sqrt(norm(e(:,:)/TrainingSetSize));
```

Listing 6.3: Programa Principal.

B. Código del algoritmo desarrollado en lenguaje C

```
#ifndef VALIDATION_C432
#define VALIDATION_C432

//**functions Matrix
void Trans_Matrix(double* ptr_M , double *ptr_N,int fila_M,int colum_M);
void Mult_Matrix(double*ptr_M ,double*ptr_N ,double*ptr_Z,int fila_M,int colum_M,int
    fila_N,int colum_N);
void Sum_Matrix(double*ptr_M,double *ptr_N ,double* ptr_Z  ,int fila_M,int colum_M ,int
    fila_N,int colum_N);
void Print_Matrix (double * ptr,int fila,int columna);
void Print_Matrix_int (int * ptr,int fila,int columna);
void initialitation_M (double*ptr_Matriz,int fila,int colum);
//***Ativation function code
void afht (double*ptr_W1 ,double *ptr_P,double*ptr_B1 ,double*ptr_F_lout,int m_W1, int
    n_W1,int m_P, int n_P,int m_B1, int n_B1,int m_Flout, int n_Flout);
void afsig (double*ptr_W2 ,double *ptr_F_lout,double*ptr_B2 ,double*ptr_S_lout, int m_W2
    , int n_W2,int m_Flout, int n_Flout,int m_B2, int n_B2,int m_Slout, int n_Slout);
//uncode-decode functions.
void encoder_vector(int* ptr_in,double *ptr_out, int fila,int colum);
void decoder_vector(double* ptr_in,int *ptr_out, int m_in,int n_in, int m_out,int n_out)
    ;
//bool compare_code(int*,const int[],int,int,int);
//void identifier_code(int*,int,int,FILE*,int*,int,int,double,int,int);
//Definition of functions of Hardware
void Vector_A(int*,int);
void Vector_B(int*,int);
void Vector_C(int*,int);
void Vector_D(int*,int);
void Vector_E(int*,int);
void Vector_H(int*,int);
void read_vector( int*,int);

#endif
```

Listing 6.4: Definición de las funciones principales del algoritmo de C432.

```
int main(int argc, char *argv[])
{
  double F_lout[size_D24][size_D1];
    double S_lout[size_D6][size_D1];
    bool t;
    int Vect_out[size_D1][size_D6];
    double P[size_D1][size_D43];
    double* ptr_W1     = &W1[0][0];
```

```
    double* ptr_W2     = &W2[0][0];
    double* ptr_B1     = &B1[0][0];
    double* ptr_B2     = &B2[0][0];
    double* ptr_P      = &P[0][0];
    double* ptr_F_lout = &F_lout[0][0];
    double* ptr_S_lout = &S_lout[0][0];
    int* ptr_Vect_in_A = &Vect_in_A[0][0];
    int* ptr_Vect_in_B = &Vect_in_B[0][0];
    int* ptr_Vect_in_C = &Vect_in_C[0][0];
    int* ptr_Vect_out  = &Vect_out[0][0];

  //Vector_A(ptr_Vect_in_A,size_D43);
  Print_Matrix_int(ptr_Vect_in_B,size_D1,size_D43);
  printf("\n\n");
  encoder_vector(ptr_Vect_in_B,ptr_P,size_D1,size_D43);
  Print_Matrix(ptr_P,size_D1,size_D43);
    printf("\n");
    afht (ptr_W1,ptr_P,ptr_B1,ptr_F_lout,size_D24,size_D43,size_D1,size_D43,size_D24,
    size_D1,size_D24,size_D1);
  //Print_Matrix(ptr_F_lout,size_D24,size_D1);
    afsig(ptr_W2,ptr_F_lout,ptr_B2,ptr_S_lout,size_D6,size_D24,size_D24,size_D1,size_D6,
    size_D1,size_D6,size_D1);
  //Print_Matrix(ptr_S_lout,size_D6,size_D1);
  decoder_vector(ptr_S_lout, ptr_Vect_out,size_D6,size_D1,size_D1,size_D6);
  Print_Matrix_int(ptr_Vect_out,size_D1,size_D6);
    identifier_code(ptr_Vect_out,size_D1,size_D6,ptr_Vect_in_B,size_D1,size_D43,size_D1,
    size_D6);
    return 0;
}
void Trans_Matrix(double* ptr_M , double *ptr_N,int fila_M,int colum_M)
{
  for(int i=0;i<colum_M;i++)
  {
    for(int j=0;j<fila_M;j++)
      *(ptr_N+i*fila_M+j)=*(ptr_M+j*colum_M+i);
  }
}

void Print_title(void)
{
  printf(" ------------------------------------------------------------------------------
    \n");
  printf("| ,___,  |
    |");
  printf("\n");
  printf("| [O.o]  |CIRCUITS OF TESTING OF ISCAS'85  C432 | UNIVERSITY OF SONORA
     |");
  printf("\n");
  printf("| /)__)  |-------------------------------------------  -----------------
    |");
  printf("\n");
  printf("| _\"_\"_  |***ARTIFICIAL NEURAL NETWORK  FOR DIGITAL CIRCUIT FAULT DIAGNOSIS
    ***|");
    printf("\n");
    printf("|________|
    ------------------------------------------------------------------|");
  printf("\n");
}
```

Listing 6.5: Parte del programa principal del algoritmo de C432.

C. Descripción de los circuitos de pruebas en lenguaje Verilog

```
module Fault_c17(clk,select,fault_c,out_seg,vector,out_c17,out_pi);
      input  [4:0]vector;
      input  clk;
      input  [4:0]fault_c;
      output [6:0]select;
      output [6:0]out_seg;
      output [1:0]out_c17;
      output [1:0]out_pi;
    //variable locales
    integer cont;
      wire Temp;
      reg [2:0]sel;
      reg W10,W11,W16,W19;
      reg [2:0]out_temp;
      parameter Fstk_0=1'b0,Fstk_1=1'b1;
    //Delay para el sincronismo
      always @(posedge clk)begin
          if(cont<100000)
          cont<=cont+1;
          else begin
          sel=sel+1;
          if(sel>6)sel<=0;
              cont<=0;
          end
          end
      always @(fault_c,vector) begin
        case(fault_c)
                        //Fallas Stuck_at_0 en el circuito Iscas C17
            5'b00001: begin //Falla Stuck-at_0 en la nand 10_1
                  W10= ~(Fstk_0 & vector[1]);
                  W11= ~(vector[1] & vector[2]);
                  W16= ~(W11  &  vector[3]);
                  W19= ~(W11 & vector[4]);
                  out_temp[0]=~(W10 & W16);
          out_temp[1]=~(W16 & W19);
        end

    //Decodificador decimal a binario
      assign select=(sel==0)?7'b1111110:
                    (sel==1)?7'b1111101:
                    (sel==2)?7'b1111011:
                    (sel==3)?7'b1110111:
                    (sel==4)?7'b1101111:
                    (sel==5)?7'b1011111:
                             7'b0111111;

    //Multiplexor  de 7 por 1
      assign Temp = (select==7'b1111110)? vector[4]://out_c17[0]
                    (select==7'b1111101)? vector[3]://out_c17[1]
                    (select==7'b1111011)? vector[2]:
                    (select==7'b1110111)? vector[1]:
                    (select==7'b1101111)? vector[0]:
                    (select==7'b1011111)? out_c17[0]
                                        :out_c17[1];
      assign out_seg=(Temp==1'b1)? 7'b1001111:7'b000_0001;
      assign out_pi=out_c17;
endmodule
```

Listing 6.6: Parte del programa principal del circuito C17.

D. Imágenes del hardware implementado

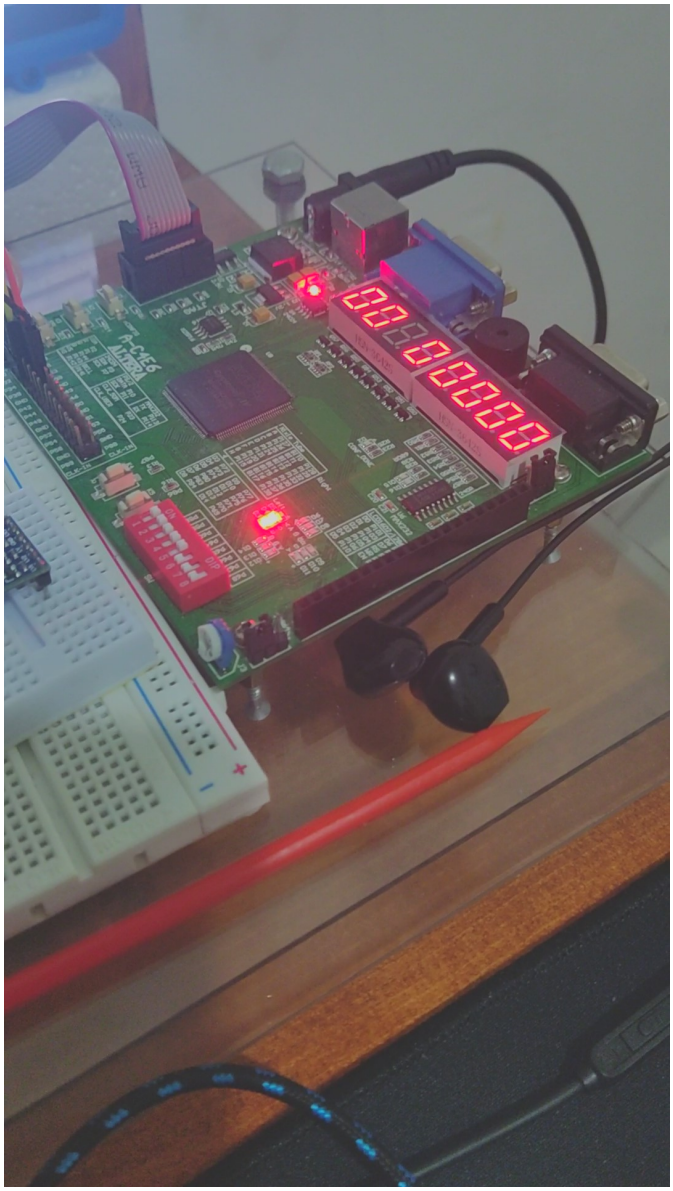


Figura 6.1: Funcionamiento del FPGA.

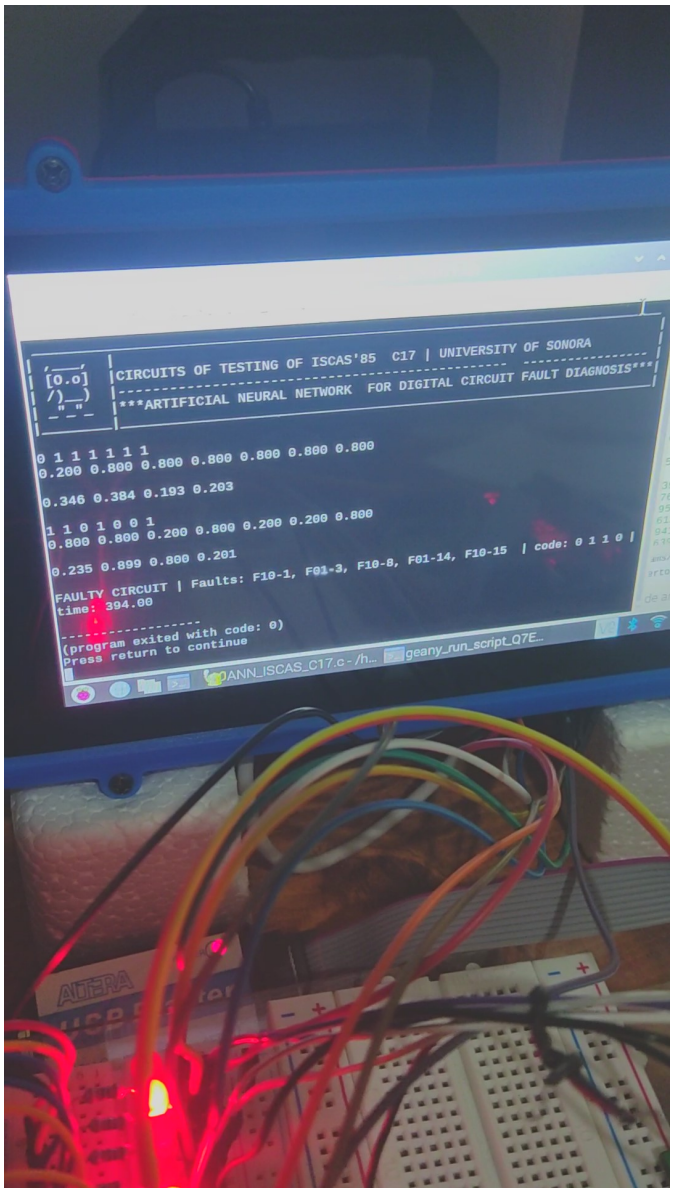


Figura 6.2: Funcionamiento del algoritmo neuronal.

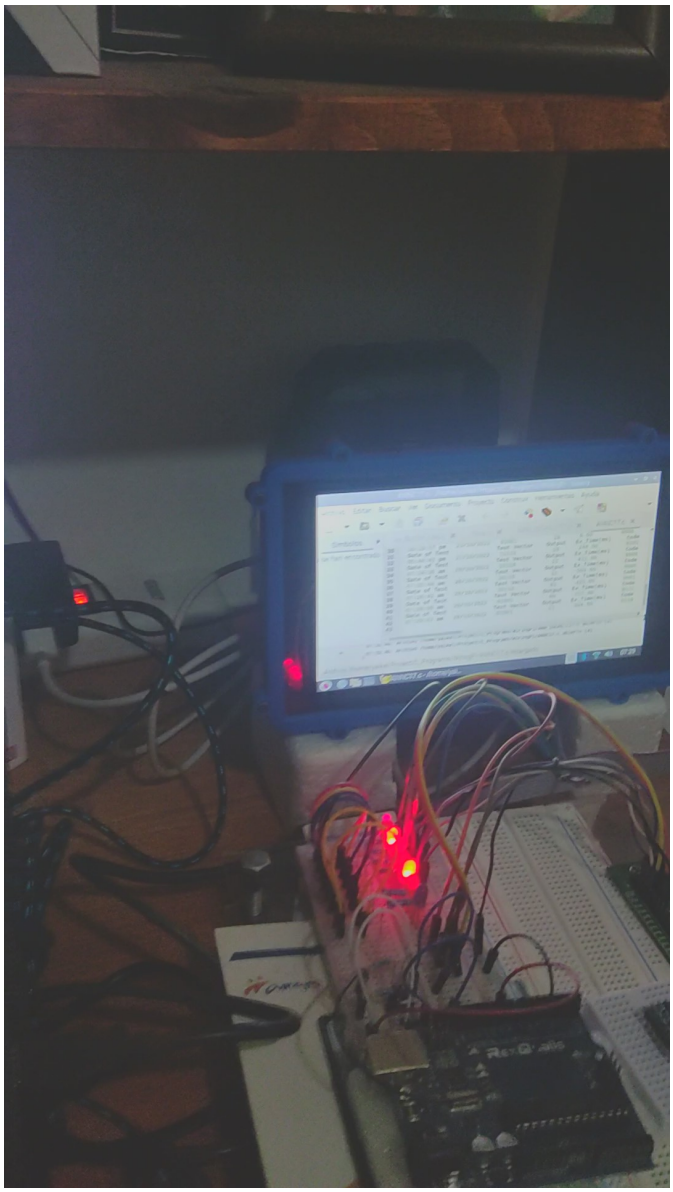


Figura 6.3: Funcionamiento del hardware propuesto.

Bibliografía

[1] Manoj Sachdev and Jose Pineda De Gyvez. *Defect-oriented testing for nano-metric CMOS VLSI circuits*, volume 34. Springer Science & Business Media, 2007.

[2] ALBERTO CUERVO. Guias únicas de laboratorio.

[3] Chris M Bishop. Neural networks and their applications. *Review of scientific instruments*, 65(6):1803–1832, 1994.

[4] Santosh Kumar, Suruchi Sharma, and Baljit Kaur. Leakage power estimation for iscas c17 benchmark circuit. In *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 108–112. IEEE, 2019.

[5] Daniel Brand and Vijay S. Iyengar. Timing analysis using functional analysis. *IEEE Transactions on Computers*, 37(10):1309–1314, 1988.

[6] David Bryan. The iscas'85 benchmark circuits and netlist format. *North Carolina State University*, 25:39, 1985.

[7] Eddy Morales-Rodriguez. Diseño de las estructuras de prueba para un microprocesador risc de 32 bits. 2005.

[8] Ilber Adonayt Ruge Ruge and José David Alvarado. Sistema basado en fpga para la evaluación de redes neuronales orientadas al reconocimiento de imágenes. *Tecnura*, 17(36):87–95, 2013.

[9] Gustavo Javier Aguirre Soler. Aplicación de redes neuronales artificiales a un sistema de detección de fallas en circuitos digitales. 2020.

[10] Michael Bushnell and Vishwani Agrawal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, volume 17. Springer Science & Business Media, 2004.

[11] Wojciech Maly, Andrzej J Strojwas, and Stephen W Director. Vlsi yield prediction and estimation: A unified framework. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 5(1):114–130, 1986.

[12] Miron Abramovici, Melvin A Breuer, Arthur D Friedman, et al. *Digital systems testing and testable design*, volume 2. Computer science press New York, 1990.

[13] AARÓN ALEJANDRO LÓPEZ CARVAJAL. Detección de fallas en circuitos vlsi con el uso de algoritmos shortest path. 2013.

[14] ÁNGEL FRANCISCO CARREÑO ACUÑA. Cálculo probabilístico en detección de fallas en circuitos vlsi. 2010.

[15] Geovanny Delgado Cascante. Deteccion de fallas con el algoritmo. *Ingeniería. Revista de la Universidad de Costa Rica*, 2(2):31–42, 1992.

[16] AARÓN ALEJANDRO LÓPEZ CARVAJAL. Detección de fallas en circuitos vlsi con el uso de algoritmos shortest path. 2013.

[17] Perelroyzen Evgeni. *Digital Integrated Circuits: Design-for-Test Using Simulink® and Stateflow®*. CRC Press, 2018.

[18] Luis Fernando Pérez Jiménez. Influencia del acoplo capacitivo en el test de fallas struck-open. 2008.

[19] Charles F Hawkins, H Troy Nagle, Ronald R Fritzemeier, and John R Guth. The vlsi circuit test problem-a tutorial. *IEEE Transactions on Industrial Electronics*, 36(2):111–116, 1989.

[20] MTM Rene Segers. The impact of testing on vlsi design methods. *IEEE Journal of solid-state circuits*, 17(3):481–486, 1982.

[21] Sandeep K Goel and Krishnendu Chakrabarty. *Testing for Small-Delay Defects in Nanoscale CMOS Integrated Circuits*. CRC Press, 2013.

[22] Scott F Midkiff and S Wayne Bollinger. Circuit-level classification and testability analysis for cmos faults. In *Digest of Papers 1991 VLSI Test Symposium'Chip-to-System Test Concerns for the 90's*, pages 193–198. IEEE, 1991.

[23] Patrick Girard. Survey of low-power testing of vlsi circuits. *IEEE Design & test of computers*, 19(3):82–92, 2002.

[24] Vidushi Sharma, Sachin Rai, and Anurag Dev. A comprehensive study of artificial neural networks. *International Journal of Advanced research in computer science and software engineering*, 2(10), 2012.

[25] Robert D Dony and Simon Haykin. Neural network approaches to image compression. *Proceedings of the IEEE*, 83(2):288–303, 1995.

[26] John Gerald Taylor. Neural networks and their applications. 1996.

[27] Damián Jorge Matich. Redes neuronales: Conceptos básicos y aplicaciones. *Universidad Tecnológica Nacional, México*, 41:12–16, 2001.

[28] Neha Sharma, Reecha Sharma, and Neeru Jindal. Machine learning and deep learning applications-a vision. *Global Transitions Proceedings*, 2(1):24–28, 2021.

[29] Elisa Zambrano Gómez, Luis Martín Torres Treviño, Gina María Idárraga Ospina, Carlos Gaytán Cavazos, and Juan José Saldívar Hinojosa. Uso de redes neuronales para reducir la dispersión de cálculos empíricos. *Ingenierías*, 18(69):18–21, 2015.

[30] Miguel Sotaquirá. `www.codificandobits.com`, 2 de Jgosto de 2018.

[31] Alejandro Sanchez Yali. `www.asanchezyali.com`, 7 de Enero de 2020.

[32] D Bryan. The iscas '85 benchmark circuits and netlist format [online] available: https://ddd. fit. cvut. cz/prj. *Benchmarks/iscas85. pdf.[Accessed Jul. 10, 2019]*.

[33] Luini Leonardo Hurtado Cortes, Edwin Villarreal-López, and Luís Villarreal-López. Detección y diagnóstico de fallas mediante técnicas de inteligencia artificial, un estado del arte. *Dyna*, 83(199):19–28, 2016.

[34] IVAN DOSTOYEWSKI MEZA IBARRA et al. Identificación de rutas lógicas influenciadas por acoplamientos capacitivos severos. 2022.

[35] Zaifu Zhang, Robert D Mcleod, and Witold Pedrycz. A neural network algorithm for testing stuck-open faults in cmos combinational circuits. *Journal of Electronic Testing*, 4(3):225–235, 1993.

[36] Julio Pérez Acle. Prototipado en fpgas para inyección de fallas. aplicación a sistemas distribuidos sobre bus can. 2005.

[37] Alberto Martínez Contreras, David Tinoco Varela, and Fernando Gudiño-Peñaloza. Diseño de una red neuronal distribuida entre dispositivos raspberry pi conectados a internet por medio de xmpp. *Res. Comput. Sci.*, 148(7):197–211, 2019.

[38] Marcela Belén Vallejos Calderón. Módulo didáctico de entrenamiento de redes neuronales para el reconocimiento de patrones de imágenes y voz con raspberry pi. B.S. thesis, 2018.