UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE TELECOMUNICACIONES

TEMA:

EVALUACIÓN DE DISPOSITIVOS FIELD PROGRAMMABLE GATE ARRAYS (FPGA), COMO MEDIDA DE SEGURIDAD EN SISTEMAS DE INTERNET DE LAS COSAS A NIVEL DE CAPA FÍSICA.

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE

INGENIERÍA DE TELECOMUNICACIONES

AUTOR

BOLÍVAR SAUL BOLAÑOS CHAMORRO

DIRECTOR

ING. FABIAN GEOVANNY CUZME RODRIGUEZ, MSC.

Ibarra, 2023

UNIVERSIDAD TECNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN

A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DEL CONTACTO			
CÉDULA DE IDENTIDAD	1003863873		
APELLIDOS Y NOMBRES	Bolaños Chamorro Bolívar Saul		
DIRECCIÓN	Ibarra, C. Alfredo Pareja Diezcanseco y Los Galeanos		
E-MAIL	bsbolanosc@utn.edu.ec		
TELÉFONO FIJO	----------	**TELÉFONO**	0996864902

DATOS DE LA OBRA	
TÍTULO	"Evaluación de dispositivos field programmable gate arrays (FPGA), como medida de seguridad en sistemas de internet de las cosas a nivel de capa física"
AUTOR	Bolaños Chamorro Bolívar Saul
FECHA	13/12/2023
PROGRAMA	☒ **PREGRADO** ☐ **POSTGRADO**
TÍTULO	Ingeniero en Telecomunicaciones
DIRECTOR	Ing. Fabian Geovanny Cuzme Rodriguez, Msc.

UNIVERSIDAD TECNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 13 días del mes de diciembre de 2023

EL AUTOR

.....

Bolaños Chamorro Bolívar Saul

CI: 1003863873

UNIVERSIDAD TECNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN:

MAGISTER FABIAN CUZME, DIRECTOR DEL PRESENTE TRABAJO DE TITULACIÓN CERTIFICA:

Que, el presente trabajo de Titulación "EVALUACIÓN DE DISPOSITIVOS FIELD PROGRAMMABLE GATE ARRAYS (FPGA), COMO MEDIDA DE SEGURIDAD EN SISTEMAS DE INTERNET DE LAS COSAS A NIVEL DE CAPA FÍSICA" ha sido desarrollado por el señor Bolaños Chamorro Bolívar Saul bajo mi supervisión.

Es todo en cuanto puedo certificar en honor de la verdad.

Ing. Fabián Geovanny Cuzme Rodríguez, MsC.

C.I. 1311527012

DIRECTOR

DEDICATORIA

Dedico este trabajo de grado a mis amados padres, Sandra y Bolívar, y a mi querido hermano Cristian, quienes fueron mi mayor inspiración a lo largo de la elaboración de este trabajo de grado.

A mi madre, por su amor incondicional, su apoyo constante y sobre todo la confianza puesta en mí, siempre fue mi motivación.

A mi padre, por su ejemplo de trabajo duro y dedicación, ya que es mi inspiración para seguir adelante.

A mi hermano, por su amistad, apoyo incondicional y por estar siempre a mi lado, en las buenas y en las malas.

Sin ustedes, no sería la persona que soy hoy, este trabajo es el resultado de todo el esfuerzo y dedicación puesta en mí.

Bolaños Chamorro Bolívar Saul

AGRADECIMIENTO

Quiero expresar mi más sincero agradecimiento a mis padres, Sandra y Bolívar, y a mi hermano, Cristian. Gracias por su ejemplo, me han ayudado a ser la persona que soy. Gracias por creer en mí, incluso cuando yo no creía en mí mismo. Gracias por enseñarme el valor del esfuerzo, la perseverancia y el amor.

También quiero agradecer a mi tutor, el Ing. Fabián Cuzme por su paciencia, su guía y sus consejos fueron fundamentales para la realización de este trabajo. Gracias por creer en mí y por ayudarme a alcanzar mis objetivos.

Agradezco también a mis profesores y amigos que tuve la dicha de conocer a lo largo de este proceso, quienes me han brindado su apoyo y amistad durante estos años de estudio.

Este trabajo es el resultado del esfuerzo y la dedicación de muchas personas. A todos ellos, les doy las gracias de corazón.

(“El que persevera alcanza”)

Bolaños Chamorro Bolívar Saul

RESUMEN

El Internet de las Cosas (IoT) está transformando la forma en que vivimos y trabajamos. Los dispositivos IoT están cada vez más presentes en nuestras vidas, desde los hogares inteligentes hasta las ciudades inteligentes. Sin embargo, este crecimiento exponencial también plantea nuevos desafíos de seguridad.

Los dispositivos IoT suelen ser vulnerables a ataques físicos y cibernéticos. Los ataques físicos pueden incluir sobrecargas eléctricas, transitorios, sabotaje o manipulación física. Los ataques cibernéticos pueden incluir malware, ataques de denegación de servicio o acceso no autorizado.

Este trabajo proporciona una visión general de la gestión de riesgos y la seguridad en sistemas IoT a nivel de capa física. Se destaca la importancia de evaluar amenazas, vulnerabilidades y ataques, y se discuten las medidas de protección que pueden implementarse en la capa física de los sistemas IoT.

En particular, el trabajo analiza la protección de dispositivos FPGA contra sobrecargas eléctricas y transitorios. Se enfatiza la necesidad de implementar mecanismos de detección y mitigación de ataques de denegación de servicio. Las pruebas de funcionamiento se abordan como un elemento clave para evaluar la robustez y confiabilidad de estos dispositivos en entornos IoT.

ABSTRACT

The Internet of Things (IoT) is transforming the way we live and work. IoT devices are increasingly present in our lives, from smart homes to smart cities. However, this exponential growth also poses new security challenges.

IoT devices are often vulnerable to physical and cyber attacks. Physical attacks can include electrical surges, transients, sabotage or physical tampering. Cyber attacks can include malware, denial of service attacks, or unauthorized access.

This paper provides an overview of risk management and security in IoT systems at the physical layer level. It highlights the importance of assessing threats, vulnerabilities and attacks, and discusses protection measures that can be implemented at the physical layer of IoT systems.

In particular, the paper discusses the protection of FPGA devices against electrical surges and transients. The need to implement detection and mitigation mechanisms for denial-of-service attacks is emphasized. Performance testing is addressed as a key element to evaluate the robustness and reliability of these devices in IoT environments.

INDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. CAPÍTULO I: ANTECEDENTES

En el primer capítulo del presente tema, se llevará a cabo la identificación de los fundamentos que permiten presentar la propuesta, así como los objetivos, el alcance y el contexto en el que se aplicarán los conceptos relacionados con la ingeniería.

1.1 Definición del Problema

Uno de los principales avances dentro de las tecnologías ha sido el Internet de las Cosas (IoT), campo el cual ha ganado gran importancia dentro de la vida diaria, en los sistemas IoT se da a conocer que cada dispositivo electrónico y numerosos sistemas deben estar asociados, dentro de cada uno de estos sistemas la mayor falencia es la seguridad de la información, debido a que mediante la aplicación de cada uno de los aspectos de la vida diaria, con lo que se puede clasificar las distintas amenazas hacia los dispositivos IoT según el vector de impacto tanto en el entorno lógico como en el entorno físico (Monzón et al., 2019). Se sabe que la seguridad dentro de los dispositivos IoT hasta 2016 tenía un gasto significativo de $ 91 millones, lo que asegura el tener un crecimiento significativo dentro de los próximos diez años (Señor Sánchez, 2021).

Dentro de las amenazas se puede presentar la incrustación de software malicioso en las herramientas principales de los sistemas autónomos, que tiene como objetivo infiltrarse en un dispositivo sin consentimiento del usuario, de igual manera existen la denegación de servicio distribuidos, donde varios equipos atacan un objetivo común para saturarlo y al final dejar inoperativo el servicio. Cada uno de los aspectos mencionados provoca la desconfianza dentro del uso de estos sistemas, debido a que dentro del tratamiento de los datos se pueden vulnerar y posiblemente que exista una invasión de troyanos de hardware en los componentes (Calva et al., 2021).

Por tanto, dentro de la implementación de la seguridad en los sistemas IoT se tiene una opción muy factible que pertenece a los dispositivos FPGA proporcionando una lógica programable, además que se establece un circuito digital usando un lenguaje especifico (Marianetti et al., 2021). La solución de seguridad mediante un sistema FPGA tiene una curva creciente dentro del mercado, la razón es que se instaura la seguridad de acceso físico con el respectivo diseño de programación, además que facilitan el diseño en la aplicación de computación intensiva, aprendizaje automático, inteligencia artificial y la computación en la nube, finalmente mediante los FPGA se puede reconsiderar los nuevos problemas de seguridad que pueden surgir en el despliegue de su funcionamiento en un modelo establecido (Sunkavilli et al., 2021).

1.2 Objetivos

1.2.1 Objetivo General

Evaluar dispositivos FPGA (Field Programmable Gate Arrays) para la implementación de seguridad a nivel de capa física en sistemas autónomos de Internet de las cosas, para la mitigación de vulnerabilidades.

1.2.2 Objetivos Específicos

- Identificar las vulnerabilidades dentro de la capa física, mediante el análisis de las técnicas existentes dentro del campo estimado en los sistemas autónomos.
- Establecer requerimientos específicos de las FPGA con criterios de seguridad que puedan ser implementadas en entornos IoT.
- Evaluar la mitigación de los posibles ataques dentro de los sistemas autónomos IoT donde se implemente los dispositivos programables FPGA.

1.3 Alcance

En la presente idea de proyecto se propone la evaluación de los dispositivos FPGA como opción dentro de los sistemas de Internet de las cosas para la implementación de la seguridad a nivel de la capa física, con el que se establecerán requerimientos que permitan la identificación de los riesgos y el nivel de mitigación en los sistemas ya mencionados.

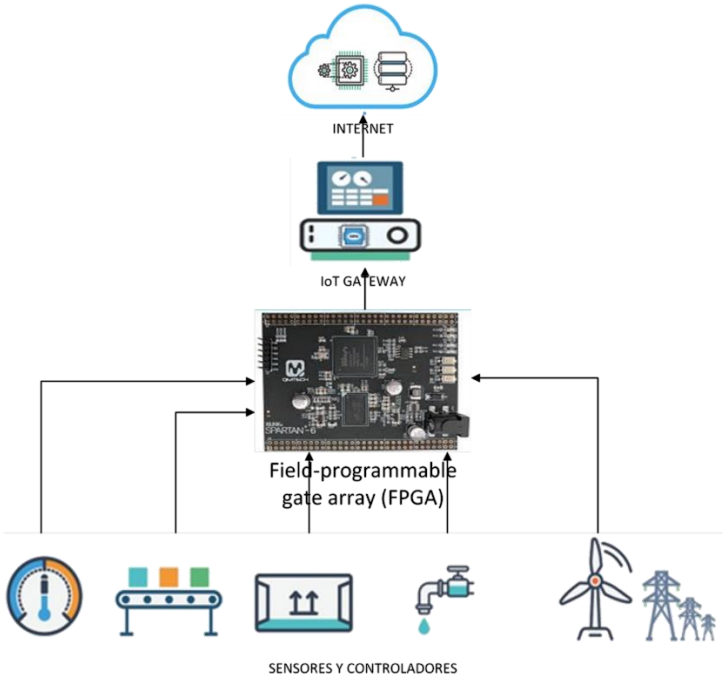
Como metodología base para el desarrollo del proyecto se tiene el modelo en espiral, este modelo se utiliza para el diseño de sistemas dentro del campo de la ingeniería donde el esfuerzo del desarrollo es iterativo, tan pronto culmina una etapa del desarrollo se debe empezar con otra, donde se tiene cuatro fases: Planificación, Ingeniería, Análisis del riesgo y Evaluación, cada una complementándose con su parámetro posterior (Espinoza Sánchez & Cisneros Vilca, 2013).

En primera instancia como parámetro de planificación se realizará el análisis de la fundamentación teórica, donde se abordan temas sobre los entornos IoT, la seguridad dentro estos sistemas y acerca de los FPGAs, además se realizará un análisis de vulnerabilidades a nivel de capa física en los entornos IoT, considerando herramientas y técnicas que permitan validar las vulnerabilidades encontradas.

El siguiente paso a realizar a nivel de la fase de ingeniería, mediante la comprensión de los requisitos y la fundamentación teórica se establecerán los requerimientos aplicables a la capa física que involucren las FPGA dentro de la seguridad de los sistemas IoT, además se propondrán diagramas de bloques, flujogramas, diagramas de casos o de secuencia que complementen la opción válida de implementación de las FPGA, además como se puede observar en la Figura 1 se expone la arquitectura de funcionamiento, de igual manera especificando cada uno de los dispositivos que se va a utilizar para el desarrollo y como se va a evaluar en estos sistemas.

Figura 1.

Arquitectura de un sistema IoT con FPGA



Nota. La figura presenta de manera conceptual como estaría establecido un sistema IoT con un FPGA implementado para proveer de seguridad a nivel de capa física.

Dentro del parámetro de análisis de riesgos y evaluación, se evaluará las medidas de mitigación en un proceso de selección y comparación que permitan establecer niveles de seguridad aceptables para entornos IoT que involucren las FPGA dentro de la capa física, por parte de los resultados obtenidos.

1.4 Justificación

En el diseño de un sistema de internet de las cosas (IoT), no se resalta la importancia de la gestión de riesgos, teniendo en cuenta la relevancia que va tomando en la vida cotidiana. La continua evolución del Internet de las cosas (IoT) obliga a las empresas a implementar medidas para reducir el impacto potencial de problemas de ciberseguridad y privacidad de la información

del usuario que puedan afectar la operación continua del negocio (García & Alberto, 2019). Se entiende por amenaza a la presencia de factores que pueden aprovechar las vulnerabilidades que se presentan, ya sea un sistema de información, personas o procesos, para la implementación en los sistemas IoT se deben considerar los siguientes factores de seguridad: las amenazas, vulnerabilidades, ataques y compromisos (Calva et al., 2021).

En cuanto a las FPGA se caracterizan por altas densidades de puerta, alto rendimiento, un número grande de entradas y salidas definibles por el usuario, un esquema de interconexión flexible, y un entorno de diseño similar al de matriz de puertas, asimismo en un FPGA es posible realizar modificaciones de último minuto sin que esto implique alteraciones, en el hardware o en el software (Barón Chacón, 2006). Con las capacidades que provee estos dispositivos se podría implementar en un sistema autónomo de Internet de las cosas, para proporcionar seguridad dentro de la capa física para cumplir con los requerimientos de los estándares dentro de los sistemas ya mencionados.

Con la evaluación de los dispositivos FPGA se pretende tener conocimiento sobre el impacto de las medidas de seguridad que se pueden aplicar a nivel de capa física como también permite la comparación con otros dispositivos dentro del contexto del nivel que se estableció, así como también la importancia y el rendimiento que nos proporciona los dispositivos mencionados para el diseño del sistema IoT seguros.

2. CAPITULO II: FUNDAMENTACIÓN TEÓRICA

En el segundo capítulo se recopila la fundamentación teórica que sustenta el desarrollo del Trabajo de Titulación. Esta recopilación comprende principalmente los temas relevantes relacionados con los dispositivos FPGA y su impacto en la seguridad de los sistemas de Internet de las Cosas. También se especifican los parámetros básicos para los dispositivos mencionados.

2.1 Internet de las Cosas (IoT)

La Internet de las cosas (IoT) se refiere a la interconexión de dispositivos a través de la red, lo que permite controlar el uso de dispositivos que antes no estaban conectados, como refrigeradores y televisores, mediante aplicaciones de Internet. Esta tendencia está evolucionando rápidamente y un ejemplo práctico se encuentra en la domótica, donde los usuarios pueden controlar la calefacción, las luces, las persianas y los accesos a la casa desde un dispositivo instalado en sus hogares. El IoT también ha dado lugar a la aparición de wearables, que son pequeños dispositivos electrónicos que las personas pueden llevar para capturar información sobre sus actividades (Romero & Stiven, 2020).

La finalidad de la Internet de las cosas (IoT) es establecer una red de dispositivos y sistemas interconectados que puedan colaborar de manera más inteligente y eficiente, con el propósito de mejorar la experiencia del usuario y resolver los desafíos de la vida diaria. Esto se logra mediante el uso de sensores, software y conectividad de red para recopilar y compartir información en tiempo real, lo que permite una mayor eficiencia y seguridad en diferentes áreas de la vida cotidiana. El objetivo último del IoT es mejorar la calidad de vida de las personas mediante la creación de soluciones innovadoras y efectivas (Cerquera, 2017).

Considerando así el (IoT) es una tecnología que tiene el potencial de transformar la manera en que interactuamos con los objetos y dispositivos a nuestro alrededor, mejorando la eficiencia y la calidad de vida. A medida que la tecnología continúa evolucionando, el IoT se está convirtiendo en una parte cada vez más importante de nuestra vida cotidiana, y es probable que continúe cambiando la forma en que vivimos y trabajamos en el futuro.

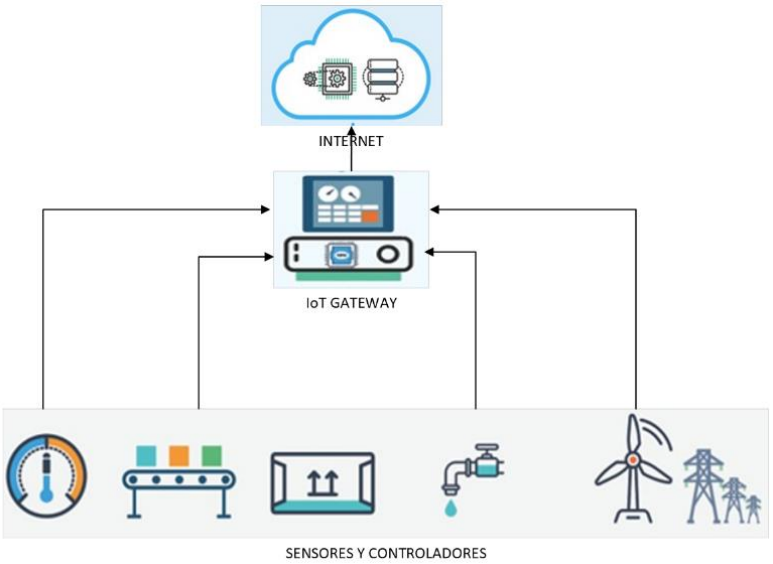
2.1.1 Arquitectura

La estandarización de una arquitectura para la Internet de las cosas (IoT) sigue siendo un proceso en evolución. En general, los esfuerzos se han enfocado en abordar dos problemas principales: establecer un estándar para el acceso a los dispositivos y medios, y desarrollar una forma de integrar los dispositivos a Internet.

Diferentes organizaciones han realizado contribuciones importantes en este ámbito, incluyendo el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE), que ha creado grupos de trabajo como el 802.15, y el protocolo 802.15.4, que permite la comunicación con baja tasa de transmisión para trabajar con dispositivos de bajo costo y limitados en recursos. La (IETF) ha presentado diversas propuestas, incluyendo el protocolo CoAP (Protocolo de aplicación limitada) y 6LoWPAN (IPv6 sobre redes personales inalámbricas de baja potencia) es un grupo de trabajo independiente que se enfoca en el desarrollo de un protocolo con el mismo nombre (González, 2013).

Figura 2.

Arquitectura simple de un sistema IoT.

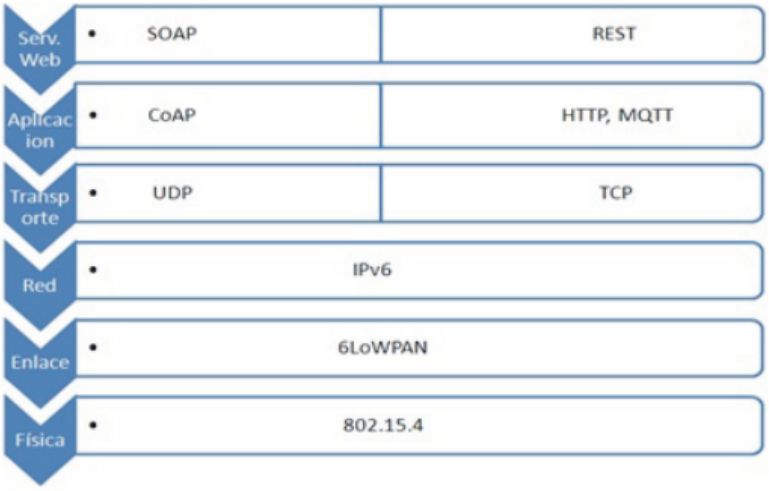


Nota. Se logra observar una forma simplificada de cómo se compone un sistema IoT en correcto funcionamiento.

A medida que se desarrollan nuevas tecnologías y avances en el campo de la IoT, es importante que las organizaciones continúen trabajando juntas para establecer estándares que puedan ser ampliamente adoptados para mejorar la experiencia del usuario y proporcionar soluciones innovadoras para los desafíos cotidianos.

Figura 3.

Stack de Protocolos de la Arquitectura de la IoT



Nota. La figura presente establece un modelo con las diferentes capas con los respectivos protocolos, en referencia al modelo OSI y TCP/IP. Adaptado de "Arquitectura y Gestión de la IoT" (p.51), por González, D. R. (2013).

La estandarización de la arquitectura de IoT puede resolver el problema de interoperabilidad entre las arquitecturas existentes en sectores específicos de la industria, mediante el uso de la Web como medio de comunicación entre los dispositivos. Además, esta estandarización permite la interconexión de los elementos de la IoT con la infraestructura de red existente, gracias al uso de protocolos de Internet estandarizados.

2.1.2 Características

Las características del Internet de las cosas (IoT) son importantes para entender cómo funciona esta tecnología y cómo puede ser aplicada en diferentes áreas de la vida. Algunas de estas características incluyen la conectividad, la interoperabilidad, la seguridad, la escalabilidad y la capacidad de análisis de datos. Estas características son fundamentales para el éxito de la IoT y su

aplicación en una amplia gama de industrias y áreas de la vida. Es importante tener en cuenta que la IoT sigue evolucionando y mejorando, lo que significa que es probable que surjan nuevas características y desarrollos en el futuro (Manotas Campos & Martinez Marín, 2018).

Tabla 1.

Características principales dentro del IoT.

Característica	Descripción
Conexión	Los dispositivos se conectan entre sí y con Internet para intercambiar datos y realizar acciones.
Automatización	Los dispositivos pueden tomar decisiones y ejecutar acciones automáticamente, sin intervención humana.
Interoperabilidad	Los dispositivos pueden trabajar juntos, independientemente de su fabricante o tecnología utilizada, gracias a estándares comunes de comunicación.
Sensores y Actuadores	Los dispositivos IoT tienen sensores para detectar su entorno y actuadores para realizar acciones en respuesta a esos datos.
Recopilación y Análisis de Datos	Los dispositivos IoT recopilan datos en tiempo real y los envían a la nube para su análisis, lo que permite tomar decisiones informadas y mejorar procesos.

Ubicuidad	Los dispositivos IoT pueden estar presentes en cualquier lugar y en cualquier momento, lo que permite monitorear y controlar objetos y procesos desde cualquier lugar del mundo.
Escalabilidad	El IoT puede manejar grandes cantidades de dispositivos y datos, lo que lo hace ideal para aplicaciones industriales y empresariales.
Seguridad	La seguridad de los datos y la privacidad son fundamentales en el IoT, por lo que se deben tomar medidas para proteger la información transmitida y almacenada.

Nota. Esta tabla muestra las características principales que se comprenden dentro de los sistemas de Internet de las Cosas.

2.1.3 Aplicaciones

El Internet de las cosas permite la creación de diversas aplicaciones para la industria y para los usuarios finales, y los dispositivos y redes aseguran una conectividad confiable y robusta entre ellos. En las aplicaciones de IoT, es crucial que los datos y mensajes sean recibidos y procesados de manera adecuada y oportuna. Aunque en algunas aplicaciones de dispositivo a dispositivo la visualización de los datos no es necesaria, cada vez más se enfocan en aplicaciones de IoT que permiten la visualización intuitiva de la información y la interacción con el entorno para los usuarios finales (Bonilla-Fabela et al., 2016).

Figura 4.

Diferentes aplicaciones del Internet de las Cosas

Nota. Se evidencia en la imagen las principales aplicaciones de los sistemas IoT para formar en conjunto una Smart City. Tomado de (cistec, 2018).

Es crucial que las aplicaciones de IoT sean desarrolladas con capacidad inteligente, para que los dispositivos puedan monitorear su entorno, detectar problemas, comunicarse entre sí y potencialmente resolver los problemas sin necesidad de intervención humana.

Tabla 2.

Aplicaciones del Internet de las Cosas

Aplicación	Descripción
Domótica	El control de los hogares inteligentes, que permite el control de la iluminación, la temperatura, la seguridad, los electrodomésticos, entre otros.

Industria	La automatización de los procesos de producción, la monitorización en tiempo real de las máquinas y la logística inteligente son algunas de las aplicaciones más relevantes de la industria 4.0.
Salud	La monitorización remota de la salud y el bienestar de los pacientes, la gestión de medicamentos, el seguimiento de los hábitos de sueño y el control del bienestar físico son algunas de las aplicaciones del IoT en el campo de la salud.
Ciudades Inteligentes	El IoT también se utiliza para gestionar el tráfico, mejorar la eficiencia energética de los edificios públicos, el control del alumbrado público, entre otros.
Agricultura de Precisión	El seguimiento y control de la humedad del suelo, la temperatura, la humedad del aire, la pluviometría, el control de plagas, la monitorización del crecimiento de las plantas y la optimización del riego son algunas de las aplicaciones del IoT en el campo de la agricultura de precisión.

Seguridad	El control de accesos y el monitoreo de cámaras de vigilancia son algunas de las aplicaciones del IoT en el campo de la seguridad.

Nota. La tabla presente muestra cada una de las aplicaciones donde más se concentra el propósito del Internet de las Cosas

2.1.4 Seguridad en los Sistemas de Internet de las Cosas

Uno de los principales desafíos en los sistemas IoT es la seguridad, debido a que estos sistemas se conectan a través de redes públicas y privadas que pueden estar expuestas a vulnerabilidades y ataques cibernéticos. Los dispositivos IoT a menudo tienen limitaciones en su capacidad de procesamiento y almacenamiento, lo que los hace vulnerables a ataques que aprovechan estas limitaciones para infiltrarse en el sistema (Gélvez-Rodríguez & Santos-Jaimes, 2020).

IoT permite transferir y compartir constantemente datos entre objetos y usuarios para alcanzar determinados objetivos. En un entorno de intercambio de este tipo, la autenticación, la autorización, el control de acceso y el no repudio son importantes para garantizar una comunicación segura, con los requisitos clave de la seguridad: autenticación, confidencialidad y control de acceso (Sicari et al., 2015).

- **Privacidad en IoT.** La aplicación del Internet de las Cosas tiene campos muy diversos, por ejemplo: monitorización remota de pacientes, control del consumo energético, control del tráfico, sistema de aparcamiento inteligente, gestión de inventarios, cadena de producción, personalización de la compra en el supermercado, protección civil. En todos ellos, los usuarios requieren la protección

de su información personal relacionada con sus movimientos, hábitos e interacciones con otras personas (Sanchez Alcon et al., 2015).

- **Confianza en IoT.** Uno de los principales problemas de muchos enfoques sobre la definición de la confianza es que no se prestan al establecimiento de métricas y metodologías de evaluación. Además, la satisfacción de los requisitos de confianza está estrictamente relacionada con la gestión de identidades y el control de acceso (Sicari et al., 2015).
- **Seguridad Móvil en IoT.** Los nodos móviles en IoT se desplazan a menudo de un clúster a otro, en los que se requieren protocolos basados en criptografía para proporcionar una rápida identificación, autenticación y protección de la privacidad (Mondragón & Guillén, 2018).

En esencia los sistemas IoT deben estar diseñados con medidas de seguridad adecuadas para proteger la privacidad y la integridad de los datos, así como para prevenir el acceso no autorizado a los dispositivos y redes. Las medidas de seguridad pueden incluir la autenticación y la autorización de los usuarios y dispositivos, el cifrado de datos, la segmentación de redes, la monitorización continua y la actualización frecuente de software y firmware para remediar vulnerabilidades conocidas.

2.2 Dispositivos Field Programmable Gate Arrays (FPGA)

Los dispositivos Field Programmable Gate Arrays (FPGA) son dispositivos electrónicos programables que consisten en una matriz de bloques lógicos configurables interconectados. Estos bloques pueden ser configurados y conectados para implementar circuitos lógicos complejos y adaptarse a diferentes necesidades.

En el mercado de los circuitos integrados, se pueden encontrar diferentes tipos de dispositivos, desde aquellos diseñados específicamente para problemas particulares hasta dispositivos reprogramables que se adaptan a diversas necesidades. Entre los dispositivos reprogramables, se encuentra la tecnología FPGA o Field Programmable Gate Array, que consta de bloques lógicos configurables que se pueden modificar según las necesidades del diseño, lo que la hace popular en diversos sectores de la industria debido a sus características (Sánchez-Solano et al., 2023).

2.2.1 Características

Al tratarse de la tecnología FPGA, se pueden identificar ciertas características que permiten compararla con otras arquitecturas existentes para determinar cuál es la más adecuada para resolver un problema específico.

- **Rendimiento.** En cuanto al rendimiento, éste varía según las necesidades del problema a resolver en cálculo o procesamiento en paralelo, tecnologías como, FPGA es de las más apropiadas.
- **Programabilidad.** La tecnología FPGA se beneficia de su configuración basada en instrucciones, aunque a cambio sacrifica algo de rendimiento en favor de una mayor programabilidad. Sin embargo, su uso todavía puede resultar complicado, ya que se requiere un amplio conocimiento del diseño y la configuración mediante máquinas de estados.
- **Adaptabilidad.** Los FPGA tienen una ventaja sobre otros microcontroladores en cuanto a flexibilidad debido a su capacidad de ser reconfigurables, lo que les permite adaptarse mejor a los cambios que se puedan presentar en un diseño o proyecto.

- **Eficiencia.** Los dispositivos FPGA pueden ser altamente eficientes en términos de rendimiento y consumo de energía debido a su capacidad de programación y configuración. Al poder personalizar y reconfigurar los bloques lógicos para adaptarse a tareas específicas, los FPGA pueden ofrecer un procesamiento altamente optimizado y eficiente para aplicaciones específicas. Además, a diferencia de los microcontroladores fijos, los FPGA son reprogramables, lo que les permite ser actualizados y adaptados para diferentes necesidades y tareas.
- **Aplicación en tiempo real.** Los dispositivos FPGA tienen una gran capacidad para ser utilizados en aplicaciones en tiempo real gracias a su alta velocidad de procesamiento y la capacidad de realizar múltiples tareas en paralelo. La capacidad de reconfiguración de los dispositivos FPGA también los hace ideales para aplicaciones que requieren actualizaciones frecuentes, permitiendo una rápida adaptación a los cambios en los requerimientos del sistema.

Aunque pueden sacrificar un poco el rendimiento a cambio de una mayor programabilidad, aún ofrecen un alto nivel de eficiencia energética y una capacidad para adaptarse a diferentes necesidades de diseño. Su capacidad de reconfiguración los hace más flexibles que los microcontroladores tradicionales y su uso se extiende a una amplia variedad de aplicaciones, desde sistemas embebidos hasta procesamiento de señales y aceleración de cálculos en computación de alto rendimiento (Juan Quevedo, 2022).

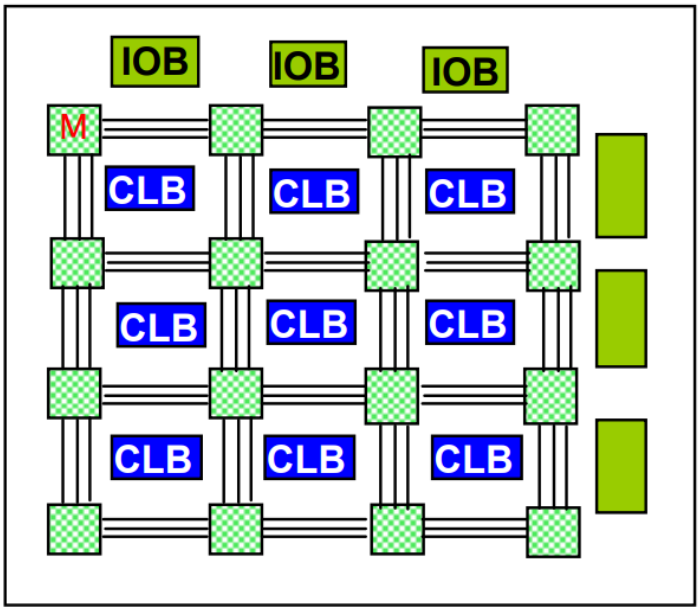
2.2.2 Funcionamiento

Los dispositivos FPGA funcionan a través de la reconfiguración de su hardware digital mediante la programación de una matriz de puertas programables (LUTs) y bloques de memoria RAM distribuidos en su interior. La configuración de estas LUTs permite la implementación de

funciones lógicas complejas y personalizadas, mientras que la memoria RAM permite almacenar datos y programación para su uso posterior (Castellanos Hernández et al., 2014).

Figura 5.

Arquitectura simplificada de un FPGA



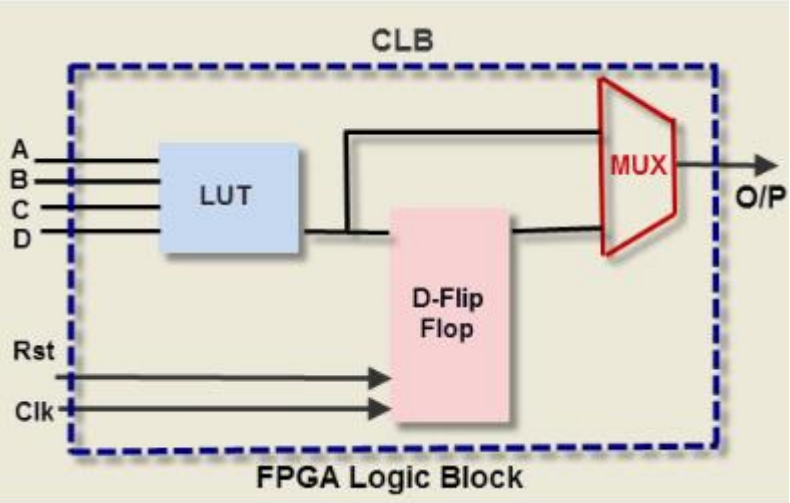
Nota. La estructura de un FPGA se compone de una serie de bloques lógicos configurables (CLB) conectados entre sí mediante canales horizontales y verticales. Esto da lugar a una matriz bidimensional, aunque la configuración de los bloques lógicos y la forma de enrutamiento puede variar según el tipo de FPGA utilizado. Adaptado de (López Echeverry & Santa V., 2011).

- **Cofigurable Logic Block.** Un Bloque Lógico Configurable (CLB) es el componente que se encarga de realizar las funciones lógicas específicas requeridas por el diseño. Está formado por varios elementos que permiten realizar estas funciones. El número de elementos que componen el CLB puede variar según las características de este, pero generalmente se compone de uno o varios LUTs cuyas salidas se encuentran multiplexadas

y se tratan según ciertos parámetros de configuración del CLB (Castellanos Hernández et al., 2014).

Figura 6.

Componentes del CLB



Nota. El CLB (Bloque Lógico Configurable) incluye lógica digital, entradas y salidas. Implementa la lógica de usuario. Tomado de (Agarwal, 2014)

- **Input/Output Block.** El bloque Input/Output (IOB) es responsable de la comunicación de la FPGA con dispositivos externos, lo que le permite enviar o recibir señales. Los IOBs son altamente adaptables, ya que pueden trabajar con diferentes tensiones y estándares digitales, lo que permite que la FPGA se adapte fácilmente a diferentes necesidades. La cantidad de bloques IOB en una FPGA depende del número de terminales de entrada/salida, lo que permite la configuración independiente de cada terminal y la existencia de un buffer único para cada uno de ellos (Juan Quevedo, 2022).

- **Block Random Access Memory.** Los bloques de RAM (BRAM) son unidades en los dispositivos FPGA utilizados para almacenar grandes cantidades de datos. Estos bloques tienen un tamaño limitado que puede ser configurado en términos de ancho y profundidad, y generalmente tienen tamaños predefinidos como 4, 8, 16 o 32 kb. El número de bloques de RAM en un dispositivo FPGA dependerá de la necesidad específica de la aplicación (Castellanos Hernández et al., 2014).
- **Encaminamiento.** Los enrutadores en las FPGA son responsables de establecer las conexiones necesarias entre los diferentes bloques lógicos, creando una red de caminos para permitir la comunicación y el intercambio de datos. Estos caminos están formados por canales de comunicación horizontales y verticales, que se conectan entre sí mediante bloques especiales llamados bloques de conexiones. De esta manera, se asegura una conexión adecuada y eficiente entre los diferentes bloques de la FPGA (Cayssials, 2014).

2.2.3 Aplicaciones

Los FPGAs son dispositivos altamente versátiles debido a su naturaleza programable, lo que los hace adecuados para una amplia variedad de aplicaciones en diferentes mercados, desde automotriz hasta comunicaciones inalámbricas y procesamiento de video e imagen.

AMD es líder en la industria de FPGA, proporcionando soluciones completas que incluyen dispositivos FPGA, software avanzado y núcleos IP configurables y listos para usar para una variedad de aplicaciones. La flexibilidad, la rápida implementación y la capacidad de adaptarse a las cambiantes demandas del mercado son solo algunas de las ventajas clave que los FPGAs ofrecen a los diseñadores de sistemas (Sánchez Narváez, 2018).

- **Aeroespacial y Defensa.** FPGAs tolerantes a la radiación junto con propiedad intelectual para procesamiento de imágenes, generación de formas de onda y reconfiguración parcial para SDRs.
- **Automotriz.** Soluciones de silicio e IP automotrices para sistemas de puerta de enlace y asistencia al conductor, comodidad, conveniencia y entretenimiento en el vehículo.
- **Radiodifusión y AV Profesional.** Adaptarse a los requisitos cambiantes de manera más rápida y prolongar los ciclos de vida de los productos con plataformas de diseño dirigidas a la radiodifusión y soluciones para sistemas de radiodifusión profesionales de alta gama.
- **Electrónica de Consumo.** Soluciones rentables que permiten aplicaciones de consumo de próxima generación y con todas las funciones, como teléfonos convergentes, pantallas planas digitales, electrodomésticos de información, redes domésticas y decodificadores residenciales.
- **Centro de Datos.** Diseñados para servidores de alta velocidad, aplicaciones de redes y almacenamiento para proporcionar un mayor valor en las implementaciones en la nube.
- **Industrial.** FPGA y plataformas de diseño dirigidas al mercado industrial, científico y médico (ISM) que permiten un mayor grado de flexibilidad, un tiempo de comercialización más rápido y costos totales de ingeniería no recurrente (NRE) más bajos para una amplia gama de aplicaciones, como la imagen y la vigilancia industrial, la automatización industrial y equipos de imagen médica.

- **Medicina.** Para aplicaciones de diagnóstico, monitoreo y terapia, las familias de FPGA Virtex y Spartan™ se pueden utilizar para satisfacer una variedad de requisitos de procesamiento, visualización e interfaz E/S.
- **Procesamiento de Video e Imágenes.** Los FPGA y las plataformas de diseño dirigidas permiten un mayor grado de flexibilidad, un tiempo de comercialización más rápido y costos totales de ingeniería no recurrente (NRE) más bajos para una amplia gama de aplicaciones de video e imágenes.
- **Comunicaciones con Cable.** Soluciones de extremo a extremo para el procesamiento de paquetes de tarjetas de línea de red reprogramables, marcos/MAC, retro planes en serie y más.
- **Comunicaciones Inalámbricas.** Soluciones de RF, banda base, conectividad, transporte y redes para equipos inalámbricos, que abordan estándares como WCDMA, HSDPA, WiMAX y otros.

2.2.4 Tipos de FPGA

Los diferentes tipos de FPGAs disponibles en el mercado proporcionan a los diseñadores una amplia gama de opciones para elegir la solución óptima para su aplicación específica.

Los FPGAs de baja densidad son ideales para aplicaciones con menor complejidad y menor número de I/O, mientras que los FPGAs de alta densidad son ideales para aplicaciones de mayor complejidad y mayor número de I/O. Los FPGAs de alta velocidad están diseñados específicamente para aplicaciones que requieren altas velocidades de procesamiento y un alto rendimiento. En general, la elección del tipo de FPGA como se observa en la Figura dependerá de los requisitos específicos de la aplicación y del presupuesto disponible (Sisterna, 2010).

Tabla 3.

Tipos de FPGA

Tipo	Características
Basado en Antiguas Tecnologías	Son los FPGA más antiguos y se basan en tecnologías como EEPROM, SRAM, Flash, etc.
Matriz de Puertas Programables	Esta es una arquitectura de FPGA más antigua en la que los bloques lógicos y la interconexión están predeterminados.
Basado en Celdas Programables	Es una arquitectura de FPGA más reciente en la que se utilizan celdas programables para la implementación de circuitos lógicos.
Gran Capacidad	Son FPGA de alta gama que ofrecen una capacidad de procesamiento de alta velocidad y gran cantidad de recursos lógicos.
Baja Potencia	Son FPGA que se han diseñado para reducir el consumo de energía y aumentar la eficiencia energética.
Procesamiento de Señal	Estos FPGA se utilizan específicamente para el procesamiento de señales digitales y analógicas.

Alto Rendimiento	Son FPGA que se han diseñado para ofrecer un alto rendimiento y velocidad de procesamiento.
Integrado en Sistemas SoC	Estos FPGA se integran en un sistema SoC (System on Chip) y ofrecen una amplia gama de funciones y capacidades.

Nota. Se puede observar los tipos de FPGA los cuales se pueden clasificar en diferentes categorías según su arquitectura, tamaño, capacidad, etc.

2.3 Fabricantes

Las implementaciones de seguridad en hardware son atractivas para dispositivos con recursos limitados de procesamiento, ya que es fácil incorporar aceleradores de cómputo en sistemas existentes. A diferencia de una solución basada en software, que no requiere modificaciones en la plataforma física del dispositivo, los procesadores utilizados en estos sistemas pueden no ser adecuados para ejecutar operaciones criptográficas complejas (Lara-Nino et al., 2020).

2.3.1 Xilinx

Xilinx es reconocido por ser uno de los líderes en el mercado de FPGA. Ofrece una amplia gama de dispositivos FPGA que se pueden utilizar en sistemas IoT. Xilinx integra funciones de seguridad en sus FPGA para proteger tanto la configuración como los datos. Sus dispositivos cuentan con características como cifrado de configuración, lo que garantiza que solo la configuración autorizada pueda cargarse en el FPGA. Además, ofrecen mecanismos de autenticación para verificar la integridad de la configuración y detectar cualquier manipulación no

autorizada. También se incluyen funciones de cifrado de datos para proteger la confidencialidad de los datos transmitidos o almacenados en el FPGA (Rodríguez Valido et al., 2012).

2.3.2 Intel

Intel, otro importante fabricante de FPGA, también se enfoca en la seguridad de los sistemas IoT. Sus dispositivos FPGA incluyen medidas de seguridad, como la protección de la configuración mediante el cifrado y la firma digital. También ofrecen funciones de seguridad adicionales, como cifrado de datos, protección de la integridad de los datos y mecanismos para proteger contra ataques de inyección de fallas. Además, Intel proporciona herramientas y soluciones de software para ayudar en el desarrollo y la implementación de sistemas IoT seguros basados en FPGA (Charte et al., 2017).

2.3.3 Microchip

Microchip es reconocido por sus FPGA y soluciones de seguridad integradas. Sus dispositivos FPGA están diseñados para abordar los desafíos de seguridad en sistemas IoT. Proporcionan protección de configuración mediante cifrado y firmado digital para garantizar la autenticidad de la configuración. También incluyen funciones criptográficas, como aceleradores de hardware para algoritmos criptográficos, para proteger la confidencialidad y la integridad de los datos. Además, Microchip ofrece soluciones de seguridad física en sus FPGA para proteger contra ataques físicos, como manipulación de voltaje o análisis de fallas (Piñal et al., 2009).

2.3.4 Lattice Semiconductor

Lattice Semiconductor se centra en ofrecer soluciones de seguridad para sistemas IoT basados en FPGA. Sus dispositivos FPGA incluyen características de seguridad, como protección de configuración mediante cifrado y firma digital. También ofrecen funciones criptográficas para proteger los datos transmitidos o almacenados en el FPGA. Lattice Semiconductor tiene en cuenta

la seguridad física y proporciona mecanismos para proteger contra ataques físicos, como la clonación del dispositivo o el análisis de fallas (Viñé Viñuelas, 2012).

Estos fabricantes, junto con otros en el mercado, están comprometidos en ofrecer soluciones de seguridad en FPGA para abordar las necesidades de protección de los sistemas IoT. Cada uno de ellos ofrece características y enfoques específicos en términos de seguridad, por lo que es importante evaluar y seleccionar el fabricante y el dispositivo FPGA que mejor se adapte a los requisitos de seguridad de un sistema IoT en particular.

2.4 Seguridad en Sistemas IoT con FPGA

La seguridad en sistemas de Internet de las cosas (IoT) es un tema crítico debido a la gran cantidad de datos que se transmiten y procesan. Los dispositivos FPGA han demostrado ser una solución efectiva para mejorar la seguridad en estos sistemas. Debido a su naturaleza programable, los FPGA permiten la implementación de algoritmos criptográficos personalizados y la integración de funciones de seguridad en el hardware mismo, lo que hace que sea más difícil para los atacantes comprometer el sistema (Barrios Alfaro, 2017).

La detección de intrusiones en la red es tan importante como el concepto de integridad de los datos. La intrusión en la red, que es la activación no autorizada de una red, puede poner en peligro no solo los datos, sino también los nodos y sistemas enteros. La principal mejora que se podría hacer en este enfoque es entrenar una red más precisa, que tenga una tasa de éxito superior al 90% (Magyari & Chen, 2022).

Además, los FPGA también permiten la detección y respuesta a intrusiones, mediante la implementación de técnicas como la monitorización del tráfico y la verificación de la integridad

del sistema. Esto hace que los dispositivos FPGA sean una herramienta valiosa para mejorar la seguridad en sistemas IoT críticos, como la salud y la seguridad pública.

2.4.1 Protección de los Sistemas IoT con los Dispositivos FPGA

Es esencial proteger los sistemas IoT para evitar posibles riesgos de seguridad y privacidad de los datos de los usuarios que podrían afectar a la continuidad del negocio. Los dispositivos FPGA son ideales para este fin, ya que permiten una gran flexibilidad y personalización en el diseño e implementación de soluciones de seguridad para adaptarse a las necesidades específicas de cada sistema IoT. Además, los FPGA pueden procesar grandes cantidades de datos en tiempo real, lo que es fundamental para detectar y prevenir posibles amenazas (BARBOSA, 2017).

Los dispositivos FPGA ayudan a la seguridad de los sistemas IoT en estos aspectos de la siguiente manera:

- **Rendimiento y eficiencia energética.** Los dispositivos FPGA pueden procesar los datos en paralelo y con bajo consumo de energía, lo que les permite realizar tareas complejas y sensibles al tiempo, como el cifrado, la compresión y el filtrado de datos.
- **Flexibilidad y escalabilidad.** Los dispositivos FPGA pueden ser reprogramados y reconfigurados para adaptarse a diferentes funciones y requisitos, lo que les permite responder a los cambios en las necesidades y las amenazas de los sistemas IoT.
- **Resistencia a los ataques físicos y lógicos.** Los dispositivos FPGA pueden implementar mecanismos de cifrado, autenticación y verificación para proteger la integridad y la confidencialidad de los datos, así como detectar y prevenir intentos de manipulación o alteración del dispositivo.

2.4.2 Vulnerabilidades dentro los sistemas IoT

Las vulnerabilidades en los dispositivos FPGA (Field Programmable Gate Arrays) para sistemas IoT (Internet of Things) se refieren a las debilidades o puntos débiles que pueden ser explotados por atacantes para comprometer la seguridad y privacidad de los sistemas y dispositivos conectados a la red. Estas vulnerabilidades pueden permitir a los atacantes acceder, modificar o interrumpir el funcionamiento de los dispositivos FPGA, lo que puede tener consecuencias graves en términos de integridad, confidencialidad y disponibilidad de los datos y servicios.(Aisawa, 2020).

Algunas de las vulnerabilidades comunes que pueden estar presentes incluyen:

2.4.2.1 Ataques de Canal Lateral

El ataque del canal lateral es una vulnerabilidad que se produce en la capa física del modelo OSI. En este tipo de ataque, un adversario intenta obtener información confidencial a través de la monitorización y el análisis de señales no deseadas generadas por el dispositivo objetivo. El ataque del canal lateral se basa en aprovechar las fugas de información que se producen durante la ejecución de un sistema, como la radiación electromagnética, el consumo de energía, el tiempo de respuesta y el ruido. Estas fugas de información pueden revelar detalles sobre las operaciones internas del dispositivo, como claves criptográficas, datos sensibles o patrones de uso. (He, 2014).

2.4.2.2 Inyección de Fallas

La inyección de fallas es una vulnerabilidad que puede ocurrir en varias capas del modelo OSI, incluyendo la capa física, la capa de enlace de datos y la capa de software. Este tipo de ataque consiste en alterar deliberadamente el funcionamiento normal de un dispositivo o sistema introduciendo fallos controlados. En la capa física, la inyección de fallas puede implicar la manipulación de señales eléctricas, como aumentar o disminuir la tensión de alimentación, generar

ruido electromagnético o irradiar radiación electromagnética. Estos ataques pueden provocar errores en la transmisión de datos, dañar componentes electrónicos o interrumpir el funcionamiento normal del dispositivo. En la capa de enlace de datos, la inyección de fallas puede tener lugar mediante la manipulación de paquetes de datos, como modificar bits específicos, eliminar o duplicar paquetes, o alterar el orden de entrega. Esto puede causar la corrupción de los datos, la interrupción de la comunicación o la ejecución de acciones no deseadas en el sistema.(Alaminos Benéitez, 2012).

2.4.2.3 Intercepción y Manipulación de Comunicaciones

La vulnerabilidad de intercepción y manipulación de comunicaciones puede ocurrir en la capa de red y la capa de aplicación del modelo OSI. Consiste en la captura y alteración de los datos que se transmiten entre los dispositivos de comunicación, lo que permite a un atacante obtener información confidencial o modificar los datos para su beneficio. En la capa de red, un atacante puede interceptar los paquetes de datos que se transmiten a través de la red. Esto puede lograrse mediante técnicas como la captura de paquetes (sniffing) o la suplantación de identidad (spoofing) para redirigir el tráfico a través de su propio dispositivo. Una vez que los datos son interceptados, el atacante puede leer o modificar su contenido, lo que puede comprometer la privacidad y la integridad de la comunicación. En la capa de aplicación, los ataques de intercepción y manipulación de comunicaciones se centran en las aplicaciones y los protocolos utilizados para la transmisión de datos. Un ejemplo común es el ataque Man-in-the-Middle (MITM), donde un atacante se sitúa entre el remitente y el destinatario y puede interceptar, leer y modificar los datos que se transmiten. Esto puede permitir al atacante robar información confidencial, como credenciales de inicio de sesión, o modificar los datos para realizar acciones maliciosas.(Potestad Ordoñez, 2019b).

2.4.2.4 Debilidades en el Firmware

Las debilidades en el firmware se refieren a vulnerabilidades y fallos en el software de bajo nivel que controla y opera los dispositivos electrónicos, como los dispositivos FPGA. El firmware es un software embebido en el hardware del dispositivo y es responsable de gestionar sus funciones y operaciones. Estas debilidades pueden existir en cualquier capa del firmware, desde el bootloader y los controladores de dispositivos hasta los componentes de aplicación específicos. Pueden ser el resultado de errores de programación, falta de validación de datos, implementaciones inseguras de protocolos de comunicación o puertas traseras involuntarias o intencionadas dejadas por los desarrolladores.(Mayo Vilches, 2021).

2.4.2.5 Escalamiento de Privilegios

El escalamiento de privilegios es una vulnerabilidad que permite a un atacante obtener niveles de acceso o privilegios superiores a los que le corresponden inicialmente en un sistema o dispositivo. Esto significa que un usuario con privilegios limitados o incluso sin autenticación puede obtener acceso y control completo sobre el sistema, lo que le permite realizar acciones no autorizadas o comprometer su seguridad. Esta vulnerabilidad puede manifestarse en diferentes capas del sistema, dependiendo de la configuración y las medidas de seguridad implementadas. A nivel de software, el escalamiento de privilegios puede aprovechar debilidades en la gestión de permisos, control de acceso o validación de datos. A nivel de sistema operativo, puede implicar la explotación de vulnerabilidades en el kernel, en servicios privilegiados o en componentes de administración de usuarios. El impacto del escalamiento de privilegios puede ser significativo, ya que un atacante con privilegios elevados puede realizar acciones maliciosas, como acceder a información confidencial, modificar archivos del sistema, instalar software malicioso, eliminar o corromper datos, o incluso tomar el control completo del sistema.(Tenelema Arias, 2020a).

Es importante tener en cuenta que la evaluación y mitigación de estas vulnerabilidades es fundamental para garantizar la seguridad de los sistemas y dispositivos IoT. Esto implica la implementación de medidas de seguridad adecuadas, como el uso de algoritmos de cifrado robustos, la detección y mitigación de ataques de canal lateral, la aplicación de técnicas de verificación de integridad del firmware y la protección de las comunicaciones (Potestad Ordoñez, 2019a).

2.4.3 Evaluación de Seguridad en Sistemas IoT con Dispositivos FPGA

La evaluación de seguridad en sistemas IoT es un proceso crítico para garantizar la seguridad y privacidad de los datos transmitidos y almacenados en estos sistemas. Este proceso involucra la identificación de posibles vulnerabilidades en la red y dispositivos IoT, así como la evaluación de la efectividad de las medidas de seguridad implementadas. Puede involucrar diferentes métodos, como pruebas de penetración, análisis de vulnerabilidades y revisiones de código, entre otros. Estos métodos pueden ser realizados tanto de manera manual como automatizada, y pueden requerir herramientas y tecnologías específicas para su implementación (Calva et al., 2021b).

En particular, en el caso con dispositivos FPGA, la evaluación de seguridad también puede incluir pruebas específicas para detectar vulnerabilidades en los dispositivos FPGA, como la lectura no autorizada de la configuración de la FPGA o la inyección de fallas. La evaluación de seguridad en sistemas IoT es un proceso continuo que evoluciona con el tiempo debido a que las amenazas y vulnerabilidades cambian constantemente (BARBOSA, 2017).

2.4.4 Metodologías de Evaluación de Seguridad en Dispositivos FPGA

La mayoría de las familias de FPGA permiten la lectura de su configuración para facilitar el proceso de depuración, pero esto también puede ser explotado por los atacantes para acceder a

información confidencial mediante la lectura de la configuración a través del JTGo. Para evitar esta posibilidad, los fabricantes de dispositivos FPGA proporcionan elementos de seguridad en su diseño, pero estos pueden ser comprometidos por los atacantes mediante técnicas de inyección de fallos (López Echeverry & Santa V., 2011).

Existen varias metodologías de evaluación de seguridad en sistemas IoT con FPGA, que pueden ayudar a identificar posibles vulnerabilidades y riesgos de seguridad en estos dispositivos. Algunas de estas metodologías incluyen:

- **Análisis de vulnerabilidades.** Esta metodología implica identificar las vulnerabilidades existentes en los sistemas IoT, así como las posibles amenazas y ataques que podrían explotar estas vulnerabilidades. Para llevar a cabo este análisis, se pueden utilizar herramientas de escaneo de vulnerabilidades y exploración de puertos para detectar posibles vulnerabilidades en la red.
- **Pruebas de penetración.** Esta metodología implica simular ataques reales para evaluar la seguridad de los sistemas IoT con FPGA. Las pruebas de penetración pueden incluir ataques de denegación de servicio (DoS), ataques de fuerza bruta, explotación de vulnerabilidades conocidas y otros tipos de ataques.
- **Criptoanálisis.** Esta metodología implica analizar la criptografía utilizada en los sistemas IoT con FPGA para identificar posibles debilidades en la implementación de la criptografía. Esto puede ayudar a identificar posibles vulnerabilidades en los sistemas IoT, como la filtración de datos sensibles.
- **Monitoreo de redes.** Esta metodología implica monitorear el tráfico de red en los sistemas IoT para identificar posibles actividades maliciosas. Esto puede incluir la

detección de patrones de tráfico inusual, como tráfico que proviene de direcciones IP sospechosas o tráfico que utiliza puertos no autorizados.

La evaluación de seguridad en sistemas IoT con FPGA es un proceso crítico que puede ayudar a identificar posibles riesgos de seguridad y a garantizar la integridad y confidencialidad de los datos en estos dispositivos. Es importante utilizar una combinación de metodologías de evaluación para obtener una imagen completa de la seguridad de los sistemas IoT.

2.4.5 Herramientas de Evaluación de Seguridad en Dispositivos FPGA

Existen diversas herramientas para la evaluación de seguridad en dispositivos FPGA, como analizadores de protocolos, depuradores, emuladores y simuladores, entre otros. También se pueden utilizar herramientas de criptografía y autenticación para garantizar la integridad y autenticidad de los datos transmitidos (Magyari & Chen, 2022).

Existen diversas herramientas de evaluación de seguridad que se pueden utilizar en sistemas IoT con FPGA, algunas de ellas son:

2.4.5.1 ChipWhisperer.

ChipWhisperer es una plataforma de código abierto diseñada para realizar ataques y pruebas de seguridad en sistemas embebidos y dispositivos electrónicos. Específicamente, se centra en la evaluación de la seguridad de chips de microcontroladores y FPGA (Field-Programmable Gate Array). ChipWhisperer ofrece una combinación de hardware y software que permite a los investigadores y profesionales de seguridad llevar a cabo diferentes tipos de ataques, como ataques de inyección de fallas, ataques de desbordamiento de búfer y análisis de lado del canal. Estos ataques se realizan con el objetivo de identificar y explotar posibles vulnerabilidades en los dispositivos y sistemas bajo evaluación (O'Flynn & Chen, 2014).

La plataforma ChipWhisperer se compone de un hardware llamado Capture Rev. 2, que se conecta al dispositivo objetivo y permite la monitorización y manipulación del tráfico y señales eléctricas. Además, cuenta con una interfaz de software que proporciona herramientas para controlar y analizar los datos capturados, así como para desarrollar y ejecutar diferentes técnicas de ataque. El objetivo principal de ChipWhisperer es facilitar la evaluación de la seguridad en sistemas embebidos y dispositivos electrónicos, permitiendo a los investigadores y profesionales identificar y mitigar posibles vulnerabilidades. Al ser una plataforma de código abierto, fomenta la colaboración y el intercambio de conocimientos en el campo de la seguridad de los sistemas embebidos (Dewar et al., 2020).

2.4.5.2 OpenOCD.

Es una herramienta de depuración y programación de código abierto ampliamente utilizada en el desarrollo de sistemas embebidos. Proporciona una interfaz de software que permite la comunicación y el control de dispositivos de depuración y programación, como sondas JTAG (Joint Test Action Group), para interactuar con los microcontroladores y FPGA. OpenOCD ofrece funcionalidades como la programación y depuración de código en tiempo real, el acceso a registros internos del dispositivo, la inspección y modificación de la memoria, y el seguimiento de eventos y trazas de ejecución. Además, admite una amplia gama de dispositivos y arquitecturas, lo que lo convierte en una herramienta versátil y compatible con múltiples plataformas de desarrollo (Camacho Olarte, 2020).

Esta herramienta es utilizada por desarrolladores y profesionales de sistemas embebidos para depurar y programar firmware en microcontroladores y FPGA durante el proceso de desarrollo y pruebas. Permite el control completo sobre el hardware objetivo, lo que facilita la detección y resolución de errores, la optimización del rendimiento y la realización de pruebas

exhaustivas en sistemas embebidos. Al ser una herramienta de código abierto, OpenOCD cuenta con una comunidad activa de usuarios y desarrolladores que contribuyen con mejoras, correcciones de errores y nuevas características. Esto proporciona una base sólida para el desarrollo de proyectos y la integración con otras herramientas de desarrollo de sistemas embebidos (Del Barrio García et al., 2017).

2.4.5.3 Xilinx Vivado Design Suite.

Xilinx Vivado Design Suite es un conjunto de herramientas de desarrollo de diseño de hardware proporcionadas por Xilinx, una empresa líder en tecnología de dispositivos lógicos programables (FPGA) y sistemas en chip programables (SoC). Vivado Design Suite es una plataforma integral que abarca desde la creación de diseños hasta su implementación y verificación en dispositivos Xilinx (Llorente Aragón, 2017).

Vivado Design Suite ofrece un entorno de diseño integrado y fácil de usar que permite a los ingenieros de hardware crear y optimizar diseños complejos de FPGA y SoC. Proporciona herramientas de síntesis, implementación, verificación y depuración, así como una interfaz gráfica intuitiva que facilita la visualización y el control de los flujos de diseño (Juan Quevedo, 2022).

Las características clave de Vivado Design Suite incluyen:

- Síntesis y optimización de diseño: Permite convertir la descripción del diseño en un circuito lógico optimizado para la implementación en dispositivos FPGA y SoC de Xilinx.
- Implementación y Enrutamiento: Realiza la colocación de los componentes del diseño en el dispositivo objetivo y establece las conexiones necesarias para la funcionalidad deseada.

- Verificación y Depuración: Ofrece herramientas para la simulación y verificación del diseño, así como capacidades de depuración para detectar y solucionar problemas en el circuito.
- Soporte de Lenguajes de Descripción de Hardware: Admite lenguajes de descripción de hardware populares como VHDL (VHSIC Hardware Description Language) y Verilog, lo que permite a los diseñadores expresar el comportamiento y la estructura del circuito.
- Integración de IP: Facilita la integración de componentes predefinidos y reutilizables en el diseño, lo que acelera el proceso de desarrollo y reduce la complejidad.
- Soporte para Dispositivos Xilinx: Vivado Design Suite está específicamente diseñado para trabajar con los dispositivos FPGA y SoC de Xilinx, lo que garantiza una compatibilidad y optimización óptimas.

2.4.5.4 JTAGulator.

Es una herramienta utilizada en el campo de la electrónica y la seguridad informática para el análisis y la depuración de dispositivos electrónicos que admiten la interfaz JTAG (Joint Test Action Group). La interfaz JTAG se utiliza comúnmente para la depuración y el acceso a nivel de hardware de dispositivos como microcontroladores, FPGA y otros componentes integrados (Davidson Tremblay, 2014).

La herramienta JTAGulator utiliza técnicas de búsqueda y prueba para identificar los pines de acceso JTAG de un dispositivo. Estas técnicas implican enviar señales específicas a los pines sospechosos y observar las respuestas del dispositivo para determinar si se trata de pines JTAG válidos. Una vez que se han identificado los pines JTAG, JTAGulator también puede usarse en

conjunto con otras herramientas y software para interactuar con el dispositivo a través de la interfaz JTAG. Esto permite realizar tareas como la depuración, el análisis de firmware, la extracción de datos y la manipulación de la memoria del dispositivo (Gallissot et al., 2020).

2.4.5.5 IDA Pro.

IDA Pro es una poderosa herramienta de análisis y desensamblado de código binario utilizada en el campo de la ingeniería inversa y la seguridad informática. Desarrollada por Hex-Rays, IDA Pro es ampliamente utilizada por investigadores de seguridad, analistas de malware y expertos en ingeniería inversa para analizar y comprender el funcionamiento interno de programas y sistemas de software. IDA Pro permite desensamblar programas ejecutables y examinar el código de bajo nivel en diferentes arquitecturas de procesadores, como x86, ARM, MIPS, PowerPC, entre otros. La herramienta proporciona una interfaz gráfica que muestra el código desensamblado, permitiendo a los analistas explorar y entender la estructura y el flujo de un programa (Eagle, 2011).

Una de las características destacadas de IDA Pro es su capacidad para realizar análisis estático y dinámico de código. El análisis estático implica examinar el código sin ejecutarlo, mientras que el análisis dinámico implica ejecutar el programa en un entorno controlado para observar su comportamiento en tiempo real. Esto permite identificar funciones, estructuras de datos, vulnerabilidades y comportamientos maliciosos en el código analizado. IDA Pro también cuenta con una amplia gama de complementos y scripts que amplían su funcionalidad y permiten automatizar tareas de análisis, realizar búsqueda de patrones, realizar análisis de malware y mucho más. Además, IDA Pro se utiliza comúnmente en la industria de la seguridad informática para el desarrollo de exploits, ingeniería inversa de firmware, análisis de vulnerabilidades y auditorías de seguridad (Liu et al., 2017).

3. CAPÍTULO III: DESARROLLO EXPERIMENTAL

En este capítulo se llevará a cabo el diseño de la propuesta como estrategia para mitigar las vulnerabilidades presentes en los sistemas de Internet de las Cosas mediante el uso de dispositivos FPGA. Asimismo, se establecerán los requisitos necesarios para garantizar la seguridad en los sistemas de IoT.

3.1 Análisis de la Situación Actual

En los últimos años, el uso de los sistemas de Internet de las Cosas (IoT) se ha extendido en diversos ámbitos, como la industria, el hogar y la salud. Sin embargo, la creciente adopción de estos sistemas también ha llevado a una mayor preocupación por la seguridad y privacidad de los datos que se manejan en ellos.

Los dispositivos IoT se han vuelto más accesibles y económicos, también se han vuelto más vulnerables a posibles ataques. Además, muchos de estos dispositivos fueron diseñados sin tener en cuenta la seguridad, lo que los convierte en un objetivo fácil para los ciberdelincuentes. Los ataques a sistemas IoT pueden tener consecuencias graves, como el robo de información personal, el control remoto de los dispositivos y el daño a la infraestructura. Estos riesgos pueden ser particularmente preocupantes en ámbitos como la salud, donde los dispositivos IoT se utilizan para monitorear pacientes y entregar medicamentos (Kumar.V.G et al., 2017).

En respuesta a esta situación, se han desarrollado diversas iniciativas y regulaciones para mejorar la seguridad en sistemas IoT. Las empresas están invirtiendo en soluciones de seguridad y en el diseño de dispositivos más seguros. Las organizaciones gubernamentales también están implementando regulaciones para garantizar que los dispositivos IoT cumplan con ciertos estándares de seguridad.

Tabla 4.

Tendencias generales sobre la utilización de los FPGA (Field-Programmable Gate Arrays) en la industria

Tendencia	Descripción
Tamaño del Mercado	Según varios informes de mercado, se espera que el mercado global de los FPGA continúe creciendo en los próximos años. Se estima que el mercado de los FPGA alcanzará un valor de varios miles de millones de dólares para el año 2026.
Sectores de Aplicación	Los FPGA se utilizan en una amplia variedad de sectores y aplicaciones. Algunos de los sectores más destacados incluyen las telecomunicaciones, la automoción, la aeroespacial y defensa
Evolución Tecnológica	La tecnología FPGA ha experimentado avances significativos en los últimos años, lo que ha permitido el desarrollo de dispositivos más potentes y eficientes.
Uso en sistemas de alto rendimiento	Los FPGA se utilizan en sistemas de alto rendimiento, como la computación de alto rendimiento (HPC), donde se utilizan para

	acelerar cálculos complejos y tareas intensivas en computación.

Nota. Es importante destacar que estos datos y tendencias son generales y pueden variar dependiendo de la región geográfica, la industria y otros factores.

A medida que los sistemas IoT se vuelven más comunes, la necesidad de una seguridad sólida se vuelve cada vez más crítica. La industria está trabajando en soluciones para mejorar la seguridad de los sistemas IoT, incluyendo el uso de dispositivos FPGA y otras tecnologías avanzadas de seguridad. Se desarrollan nuevas soluciones de seguridad para los sistemas de IoT con FPGA, es importante tener en cuenta que la seguridad no es un problema que pueda resolverse de manera definitiva. Los ataques y las vulnerabilidades evolucionan constantemente, por lo que se necesitan soluciones de seguridad que puedan adaptarse y actualizarse continuamente para garantizar la protección a largo plazo de los sistemas de IoT (Acar & Ors, 2017).

3.2 Establecimiento de requerimientos

La presente sección abordará los requerimientos necesarios para garantizar el cumplimiento de los criterios fundamentales en la evaluación de dispositivos FPGA como medida de seguridad en sistemas de Internet de las Cosas (IoT). En primer lugar, se realizará un análisis exhaustivo de los requerimientos a través de la revisión de documentación bibliográfica relevante. Esto incluirá información relacionada con el uso de softwares de simulación específicos, así como vulnerabilidades conocidas y métodos de mitigación aplicables al proyecto. Para este apartado se lo lleva a cabo siguiendo las directrices establecidas en la norma ISO/IEC/IEEE 29148, la cual proporciona un marco sólido para la definición y documentación de requerimientos en proyectos de ingeniería de sistemas y desarrollo de software. Esto garantizará que los requerimientos

identificados estén alineados con los objetivos y restricciones particulares del sistema IoT que involucra dispositivos FPGA.

3.2.1 Nomenclatura de Requerimientos

Con el fin de agilizar y facilitar la identificación de los diversos requerimientos, se utilizarán nomenclaturas específicas que se encuentran detalladas en la Tabla 5. Estas nomenclaturas hacen referencia a diferentes secciones que deben ser consideradas en el proyecto, permitiendo una distinción más rápida y eficiente de los requerimientos correspondientes.

Tabla 5.

Abreviaturas para los Requerimientos

Requerimiento	Abreviatura
Requerimientos de Stakeholders	StSR
Requerimientos de Sistema	SySR
Requerimientos de Arquitectura	SrSH

Nota. La tabla muestra la nomenclatura que se utilizará a lo largo del documento en base a lo determinado.

3.2.2 Stakeholders

En este apartado se menciona al conjunto de usuarios que se benefician del proyecto y están involucrados en sus actividades y decisiones, lo que afecta su rendimiento. Los detalles de las partes interesadas en este proyecto se presentan en la Tabla 6, donde se enumeran y describen las diferentes personas, grupos o entidades que tienen un interés y participación en el proyecto.

Tabla 6.

Actores involucrados

N°	Stakeholders	Descripción
StSR1	Ing. Fabian Cuzme, MSc	Director
StSR2	Ing. Daniel Jaramillo, MSc	Asesor
StSR3	Bolívar Bolaños	Desarrollador

Nota. Se presentan los involucrados dentro del trabajo de titulación propuesto.

3.2.3 Requerimientos de Sistema

En esta sección, se centra en la especificación de los requisitos del sistema. Para ello, la Tabla 7 que se realizó en base al **ANEXO A**. proporciona una descripción detallada de los requisitos de hardware y software del sistema, junto con las restricciones que están asociadas a su funcionamiento. Estos valores se refieren específicamente al entorno de simulación en software, ya que no se dispone de la infraestructura necesaria para implementar estos dispositivos en un sistema IoT de forma física.

Tabla 7.

Requerimientos del Sistema

N°	Requerimientos	Prioridad		
		Alta	Media	Baja
SySR1	El sistema debe ser capaz de simular el procesamiento interno que realiza un FPGA	X		

SySR2	El sistema debe contar con la posibilidad de la reprogramabilidad.	X	
SySR3	El sistema debe mitigar en cierto grado las vulnerabilidades presentes.	X	
SySR4	El sistema debe tener la posibilidad de ser implementado en un sistema IoT.	X	
SySR5	El sistema debe contar con mecanismos de monitoreo y auditoría para registrar eventos de seguridad.	X	
SySR6	El sistema debe tener las consideraciones del consumo energético.		X
SySR7	El sistema debe cumplir con estándares de seguridad y seguir buenas prácticas de diseño seguro.	X	
SySR8	El sistema debe permitir la elección de pines tanto de entrada de datos como de salida de datos.		X
SySR9	El sistema de simulación debe contar con la capacidad de incorporar configuraciones específicas para la implementación del ataque deseado.		X
SySR10	El sistema debe ser capaz de realizar pruebas de operatividad.	X	

Nota. En la tabla se presentan los requisitos necesarios para determinar la opción más adecuada de software de simulación para el desarrollo del proyecto.

3.2.4 Requerimientos de Arquitectura

El SrSH proporciona la estructura y organización de los requisitos del sistema. Su objetivo es definir la arquitectura del sistema, establecer las relaciones entre los diferentes componentes y especificar los requisitos de comportamiento y rendimiento. El SrSH se utiliza como una herramienta de planificación, gestión y control en el proceso de desarrollo de requisitos, los cuales se definieron en base al **ANEXO B**. Además, también se emplea para verificar y validar los requisitos del sistema, así como para identificar posibles problemas o riesgos durante el desarrollo del sistema.

Tabla 8.

Requerimientos de Arquitectura.

Nro.	**Requerimientos**	**Prioridad**		
		Alta	**Media**	**Baja**
SrSH1	El sistema debe ser capaz de soportar algoritmos criptográficos robustos para garantizar la seguridad de la comunicación en los sistemas IoT.	X		
SrSH2	El sistema debe contar con mecanismos de autenticación y autorización para garantizar el acceso seguro a los recursos del sistema.	X		

SrSH3	El dispositivo FPGA debe contar con mecanismos de detección y mitigación de ataques de denegación de servicio para asegurar la disponibilidad de los servicios y aplicaciones.			X
SrSH4	El sistema debe implementar políticas de control de acceso y auditoría para monitorear y registrar las actividades de los usuarios y detectar posibles amenazas.	X		
SrSH5	Las pruebas sobre el sistema deben reflejar el nivel de mitigación para las vulnerabilidades.	X		
SrSH6	El dispositivo FPGA debe contar con protección contra sobrecargas eléctricas y transitorios para evitar daños físicos y garantizar la integridad del sistema.		X	
SrSH7	El dispositivo FPGA debe cumplir con estándares de seguridad reconocidos y ser sometido a pruebas y evaluaciones de seguridad independientes para garantizar su robustez y confiabilidad en entornos IoT.	X		

Nota. La integración de todos los componentes asegura que la arquitectura esté diseñada de manera compatible con el software de simulación, permitiendo que todos los procesos operen de manera eficiente y efectiva.

3.3 Selección del Software de Simulación

La elección de un software de simulación apropiado es de gran importancia debido a que permite recrear y analizar el comportamiento de los dispositivos FPGA en diferentes escenarios y condiciones. Esto nos ayuda a identificar posibles vulnerabilidades y evaluar la efectividad de las medidas de seguridad implementadas.

Es importante considerar aspectos como la disponibilidad de bibliotecas de componentes y modelos predefinidos, la compatibilidad con los estándares y protocolos de seguridad, la facilidad de uso y la capacidad de integración con otras herramientas de desarrollo y evaluación. Además, la reputación y el soporte técnico del proveedor del software son aspectos para tener en cuenta.

Tabla 9.

Evaluación de los criterios establecidos para el sistema a implementar.

Requerimientos del sistema (SyRS)	**ChipWisperer**	**Vivado**	**Quartus Prime**
SySR1	X	X	X
SySR2	X	X	X

SySR3	X	X	X
SySR4	X	X	X
SySR5	X	X	X
SySR6	-	X	X
SySR7	-	X	X
SySR8	X	X	X
SySR9	X	X	X
SySR10	X	X	X
Total	8	10	10

Nota. En la tabla se puede observar que cada "X" indica el cumplimiento de cada parámetro mencionado, por lo que el software elegido en este caso será Quartus Prime por su compatibilidad al momento de establecer parámetros específicos.

3.4 Selección de Hardware Simulado

La selección adecuada del hardware para la simulación es de vital importancia en la evaluación de dispositivos FPGA como medida de seguridad en sistemas de Internet de las Cosas (IoT). El hardware utilizado en la simulación debe ser capaz de emular las características y funcionalidades del dispositivo FPGA en cuestión de manera precisa y confiable. Esto garantiza que los resultados obtenidos durante la simulación reflejen de manera precisa el comportamiento y desempeño del dispositivo real.

Al elegir el hardware para la simulación, es necesario considerar factores como la capacidad de procesamiento, la capacidad de memoria, la compatibilidad con el software de simulación utilizado y la capacidad de emulación de las interfaces y protocolos específicos del dispositivo FPGA. Además, es importante asegurarse de que el hardware seleccionado sea lo suficientemente escalable para adaptarse a futuras necesidades de simulación y pruebas.

Una selección cuidadosa del hardware de simulación garantiza una evaluación más precisa y confiable de las vulnerabilidades de seguridad en los dispositivos FPGA. Esto permite identificar y mitigar posibles riesgos de seguridad de manera más efectiva, contribuyendo así a la protección de los sistemas de Internet de las Cosas y a garantizar su correcto funcionamiento.

Tabla 10.

Evaluación de los criterios para la arquitectura establecidos para el sistema a implementar.

Requerimientos de la Arquitectura (SrSH)	**Max 10**	**Cyclone V**	**Arria 10**
SrSH1	X	X	X
SrSH2	X	X	X
SrSH3	X	X	X
SrSH4	X	X	X
SrSH5	X	X	X

SrSH6	-	X	-
SrSH7	X	X	X
Total	6	7	6

Nota. En la tabla se puede observar que cada "X" indica el cumplimiento de cada parámetro mencionado, por lo cual para ser simulado el hardware elegido es el Cyclone V.

3.5 Selección de Vulnerabilidades

La identificación de estas vulnerabilidades en sistemas IoT con dispositivos FPGA requiere un enfoque exhaustivo y metodologías de evaluación de seguridad adecuadas. Esto puede incluir pruebas de penetración, análisis de código, revisión de configuraciones, monitoreo de tráfico de red y análisis de protocolos de comunicación. Es fundamental realizar evaluaciones periódicas de seguridad y mantener actualizados los dispositivos FPGA con las últimas actualizaciones y parches de seguridad. Además, es importante seguir las mejores prácticas de seguridad, como implementar políticas de acceso seguro, cifrado de datos y autenticación fuerte para minimizar los riesgos de vulnerabilidades.

El análisis de vulnerabilidades en sistemas IoT se refiere al proceso de identificar y evaluar las debilidades de seguridad en los dispositivos y la infraestructura utilizados en entornos de Internet de las cosas. Consiste en examinar los componentes del sistema IoT, como sensores, dispositivos de comunicación, plataformas en la nube y software, para identificar posibles puntos de vulnerabilidad que podrían ser explotados por actores malintencionados.

Tabla 11.

Vulnerabilidades presentes en un sistema IoT con dispositivos FPGA

Vulnerabilidad	Capa Afectada	Impacto		
		Alta	**Media**	**Baja**
Ataques de Canal Lateral	Física		X	
Inyección de Fallas	Física	X		
Intercepción y Manipulación de Comunicaciones	Red, Aplicación		X	
Debilidades en el firmware	Red, Aplicación	X		
Escalamiento de privilegios	Enlace de Datos, Red, Aplicación	X		

Nota. Se presentan las vulnerabilidades principales en este tipo de dispositivos con su impacto, los cuales se especificaron en el Capítulo 2, en este caso para el presente trabajo se trabajará sobre las vulnerabilidades que se presentan en la capa física. Fuente: Autoría.

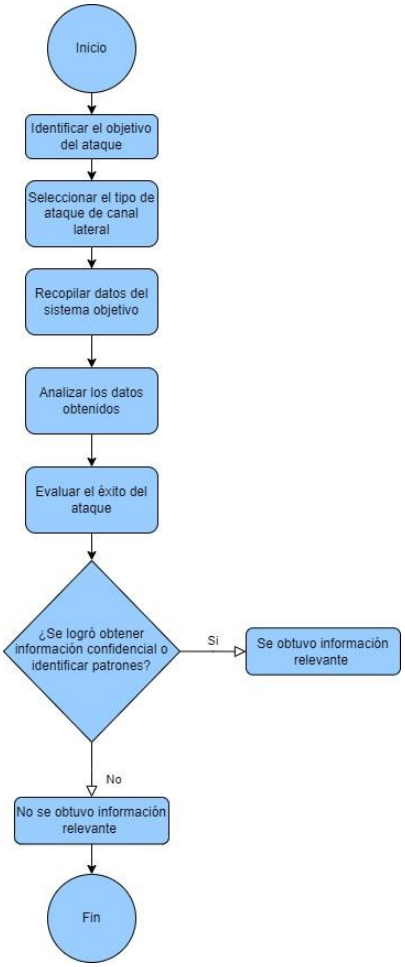
3.5.1.1 Ataques de Canal lateral

Un ataque de canal lateral en un FPGA (Field Programmable Gate Array) es una técnica utilizada por un atacante para extraer información confidencial o secretos mediante el análisis de las fugas de información que se producen durante el funcionamiento del dispositivo. Los ataques de canal lateral, conocidos como Side-Channel Attacks, aprovechan las fugas de información accidental o no intencional que se producen durante el funcionamiento de un sistema. Estas fugas pueden estar relacionadas con diferentes aspectos físicos, como el consumo de energía, el tiempo de respuesta, las emisiones electromagnéticas o las fluctuaciones en el voltaje. Los atacantes pueden analizar estas fugas de información para obtener datos sensibles o secretos, como claves de cifrado o patrones de uso, comprometiendo así la confidencialidad y la integridad de los datos.

- **Impacto.** Los ataques de canal lateral pueden tener un impacto significativo en la seguridad de un sistema. Al obtener información confidencial, los atacantes pueden llevar a cabo acciones maliciosas, como el descifrado de comunicaciones seguras, el acceso no autorizado a datos confidenciales o incluso la manipulación de transacciones. Esto puede conducir a graves consecuencias, como pérdida de datos, robo de información sensible, violación de la privacidad o daño a la reputación de una organización.

Figura 7.

Diagrama de flujo para el Ataque del Canal Lateral



Nota. Fuente: Autoría.

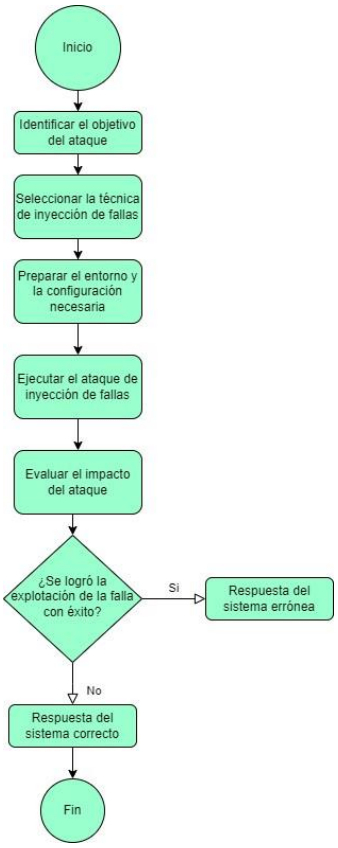
3.5.1.2 Inyección de Fallas

Los ataques de inyección de fallas pueden dirigirse a los dispositivos FPGA para alterar su funcionamiento normal. Esto puede incluir la manipulación de señales eléctricas o el cambio de valores en registros y bloques lógicos, lo que puede llevar a resultados inesperados o incluso a la ejecución de instrucciones maliciosas.

- **Impacto.** La inyección de fallas puede llevar a resultados inesperados en el funcionamiento del dispositivo FPGA, comprometiendo la integridad y confidencialidad de los datos procesados por el sistema IoT.

Figura 8.

Diagrama de flujo para el Inyección de Fallas



Nota. Fuente: Autoría

3.6 Comparación de los sistemas de mitigación para cada uno de los ataques

En la evaluación de sistemas de mitigación para diferentes tipos de ataques en dispositivos FPGA, es fundamental realizar una comparación exhaustiva para determinar cuál es la opción más adecuada en función de los requisitos específicos de seguridad y las características del sistema. la elección de la técnica de mitigación dependerá de la aplicación específica y de la priorización de

los requisitos de seguridad, el costo, la eficiencia y otros factores. Es común implementar múltiples capas de protección para abordar diferentes amenazas. Además, es importante seguir buenas prácticas de seguridad en el diseño de FPGA y mantenerse al tanto de las últimas investigaciones en seguridad en FPGA para adaptar y mejorar continuamente las estrategias de mitigación.

La razón principal por la que es importante realizar una evaluación de amenazas y vulnerabilidades específicas para su aplicación antes de seleccionar y aplicar mitigaciones de seguridad es que cada sistema y contexto y en base a la norma ISO 27001 es único en términos de sus riesgos y requerimientos de seguridad. Aquí hay algunas razones clave por las que este enfoque es esencial:

- **Riesgos Variables:** Los sistemas pueden enfrentar una amplia gama de riesgos y amenazas, desde ataques físicos hasta ataques cibernéticos. Identificar los riesgos específicos que enfrenta su sistema le permite priorizar adecuadamente qué amenazas son las más críticas y deben abordarse primero.
- **Requisitos de Seguridad:** No todos los sistemas tienen los mismos requisitos de seguridad. Por ejemplo, un dispositivo médico implantable tiene requisitos de seguridad mucho más altos que un dispositivo de entretenimiento en el hogar. Conocer sus requisitos de seguridad es esencial para seleccionar las mitigaciones adecuadas.
- **Limitaciones y Recursos:** Cada sistema tiene limitaciones en términos de recursos, como potencia, rendimiento y costo. Debe seleccionar mitigaciones que sean prácticas y viables dentro de estas limitaciones.

- **Impacto en el Rendimiento:** Algunas mitigaciones pueden afectar el rendimiento del sistema, como aumentar la latencia. Es importante equilibrar la seguridad con el rendimiento para garantizar que su sistema funcione de manera eficiente.
- **Cumplimiento Regulatorio:** Dependiendo de la industria y la ubicación, es posible que deba cumplir con regulaciones específicas de seguridad. Esto puede requerir enfoques de seguridad específicos.
- **Evolución de las Amenazas:** Las amenazas y los ataques evolucionan con el tiempo. Lo que funcionó como una mitigación efectiva en el pasado puede no ser tan efectivo contra amenazas emergentes. Una evaluación continua es esencial.
- **Costo-Beneficio:** No todas las amenazas merecen una inversión significativa en seguridad. Debe evaluar el costo-beneficio de cada mitigación para asegurarse de que esté gastando recursos de manera eficaz.

Las características base para la elección del sistema de mitigación se derivan de los principios de seguridad generalmente aceptados. Aquí hay una breve explicación de cómo se relacionan estas sugerencias con esas características:

- **Efectividad:** Las técnicas de detección y corrección de errores, así como el enmascaramiento, son enfoques probados para hacer frente a la inyección de fallas y mejorar la efectividad de la seguridad.
- **Compatibilidad:** La elección de estándares abiertos de seguridad, como AES y RSA, garantiza que las mitigaciones sean compatibles con otros sistemas y puedan intercambiar información de manera segura.

- **Robustez:** El control de acceso físico y la protección de la configuración del FPGA son medidas que mejoran la robustez al proteger el hardware contra manipulaciones maliciosas.

- **Bajo Costo:** La optimización de recursos y la implementación eficiente pueden ayudar a reducir los costos asociados con la seguridad del FPGA.
- **Estándares de Seguridad:** Cumplir con estándares reconocidos proporciona una base sólida para garantizar que las mitigaciones sean confiables y efectivas.
- **Latencia:** Optimizar el rendimiento y utilizar la paralelización son estrategias clave para mantener la latencia baja mientras se aplica la seguridad.

3.6.1 Mitigación Inyección de Fallas

La mitigación de la inyección de fallas implica la detección, localización, aislamiento y corrección de fallas en un sistema, así como la implementación de medidas de seguridad para prevenir futuras fallas. Lo cual mediante la comparación con los diferentes sistemas proporcionados en base al **ANEXO C** se lo podrá determinar en la medida de las características proporcionadas. Este proceso es esencial para garantizar la confiabilidad y la seguridad de los dispositivos electrónicos, como las FPGAs, especialmente en entornos críticos donde las fallas pueden tener consecuencias graves.

Tabla 12.

Mitigación para una inyección de fallas dentro de un sistema IoT con dispositivos FPGA

Mitigación	Característica					
	Efectividad	**Compatibilidad**	**Robustez**	**Bajo Costo**	**Estándares Seguridad**	**Latencia**
Detección y Corrección de Errores (ECC)	Alta	Alta	Alta	Media	Cumple	Baja
Temporización segura	Alta	Variable	Alta	Bajo	Variable	Baja
Redundancia	Alta	Variable	Alta	Variable	Variable	Baja
Checksum	Alta	Alta	Alta	Bajo	Cumple	Baja
Cifrado y autenticación	Alta	Alta	Alta	Medio	Cumple	Baja
Pruebas de integridad	Alta	Variable	Alta	Variable	Cumple	Baja

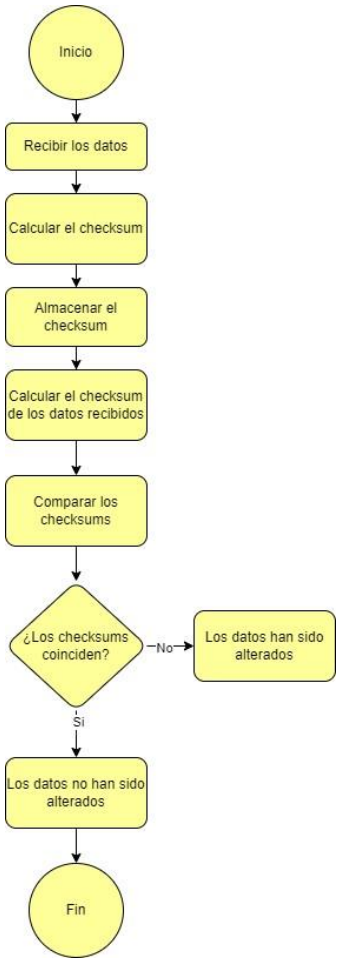
Nota. Se presentan mitigaciones principales para la inyección de fallas en este tipo de dispositivos con su impacto, en este caso para el presente trabajo se trabajará con la mitigación de Checksum ya que cumple con la mayoría de las características propuestas. Fuente: Autoría.

3.6.1.1 Checksum

El proceso de checksum implica realizar una operación matemática, como una suma de verificación o una función hash, sobre los datos originales para generar un valor de verificación. Este valor se compara con un checksum previamente calculado para determinar si los datos han sido modificados o corrompidos durante la transmisión o almacenamiento. Si los valores coinciden, se considera que los datos son válidos y no se ha producido una inyección de fallas. Si los valores no coinciden, indica que los datos han sido alterados y se puede inferir que ha ocurrido una inyección de fallas.

Figura 9.

Diagrama de flujo Checksum



Nota. El checksum es una técnica que verifica la integridad de los datos. Se calcula un valor de suma de verificación y se compara con un valor conocido. Si difieren, se detecta la falla. Es una técnica eficiente y ampliamente compatible. Fuente: Autoría.

3.6.2 Mitigación Ataque del Canal Lateral

La mitigación del ataque del canal lateral es esencial en aplicaciones donde la confidencialidad de la información es crítica, como en sistemas de cifrado de datos y dispositivos de acceso seguro, de igual manera que para la vulnerabilidad anterior la comparación se lo realiza en base al **ANEXO C**. La combinación de varias de estas medidas puede fortalecer significativamente la seguridad y dificultar que los atacantes utilicen el canal lateral para obtener información confidencial.

Tabla 13.

Mitigación para el ataque del canal lateral dentro de un sistema IoT con dispositivos FPGA

Mitigación	**Característica**					
	Efectividad	**Compatibilidad**	**Robustez**	**Bajo Costo**	**Estándares Seguridad**	**Latencia**
Enmascaramiento aleatorio	Alta	Media	Alta	Bajo	Cumple	Baja
Diversificación de Claves	Alta	Variable	Alta	Variable	Cumple	Baja

Control de Acceso Físico	Alta	Alta	Alta	Variable	Cumple	Baja
Cifrado de Lado del Cliente	Alta	Alta	Alta	Variable	Cumple	Baja

Nota. Se presentan mitigaciones principales para el ataque del canal lateral en este tipo de dispositivos con su impacto, en este caso para el presente trabajo se trabajará con la mitigación de Enmascaramiento Aleatorio ya que cumple con la mayoría de las características propuestas. Fuente: Autoría.

3.6.2.1 Enmascaramiento Aleatorio

La máscara aleatoria tiene la propiedad de que cambia en cada ciclo de reloj, lo que dificulta la extracción de información por parte de un atacante a través del canal lateral. Al aplicar una operación XOR entre la máscara aleatoria y el dato original en cada ciclo de reloj, se asegura que el dato mitigado esté constantemente enmascarado con una máscara aleatoria diferente, lo que dificulta la correlación entre el dato original y los posibles rastros o fugas de información. Esta técnica de enmascaramiento aleatorio es comúnmente utilizada en la mitigación de ataques de canal lateral, ya que proporciona una forma eficaz de proteger los datos sensibles contra la extracción de información a través de medidas criptográficas y de enmascaramiento.

Figura 10.

Diagrama de flujo para el Enmascaramiento Aleatorio

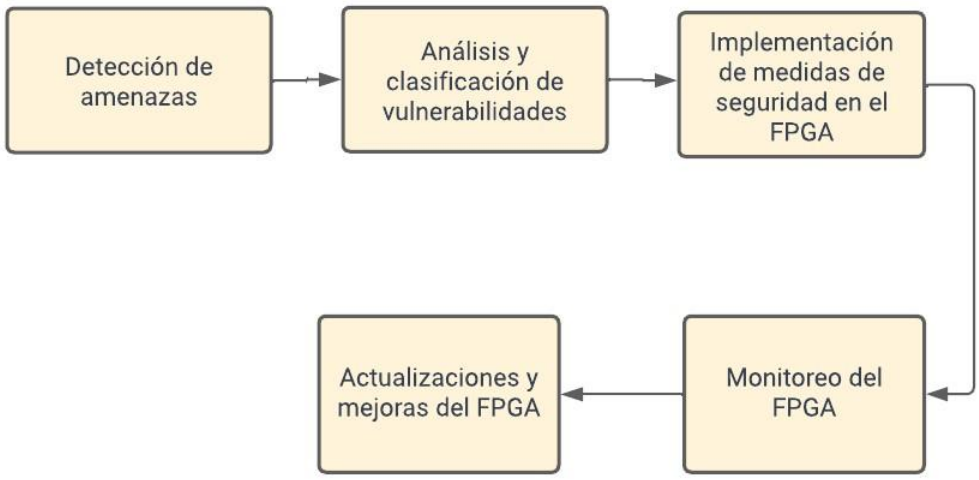
Nota. Fuente: Autoría.

3.7 Diseño del Sistema de Mitigación Preliminar

Con el análisis realizado anteriormente se procede con el diseño del sistema de mitigación de vulnerabilidades para dispositivos FPGA, en cumplimiento de los requerimientos establecidos para la evaluación de seguridad en sistemas de Internet de las Cosas (IoT). Para el diseño se toma en cuenta la detección, prevención y mitigación posibles ataques y vulnerabilidades en los dispositivos FPGA, garantizando así la integridad y confidencialidad de los datos en entornos IoT, para mayor detalle en el **ANEXO D** se encuentra desplegado el diagrama de flujo completo acerca del diseño.

Figura 11.

Diagrama de Bloques de Etapas del Sistema de Mitigación



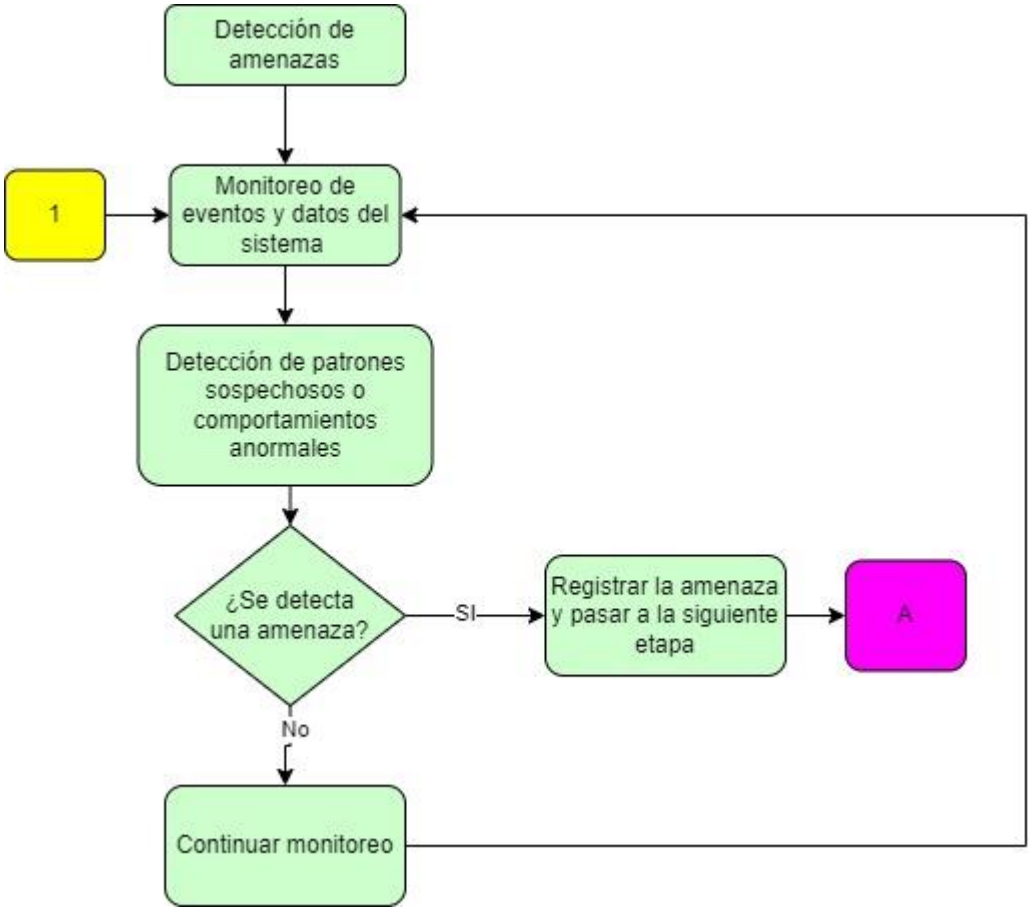
Nota. Se puede observar las etapas para el correcto funcionamiento del sistema de mitigación. Fuente: Autoría.

3.7.1 Detección de Amenazas

En esta etapa, se implementan técnicas y algoritmos para detectar posibles amenazas y vulnerabilidades en el dispositivo FPGA. Esto puede incluir el monitoreo constante de eventos y actividades dentro del FPGA, como cambios en el flujo de datos, accesos no autorizados a recursos críticos o comportamientos anómalos lo cual se puede observar en la Figura 12. Se utilizan técnicas de análisis de firmas, detección de anomalías y monitoreo de tráfico para identificar posibles amenazas y activar las medidas de mitigación correspondientes.

Figura 12.

Diagrama de flujo para la Detección de Amenazas.



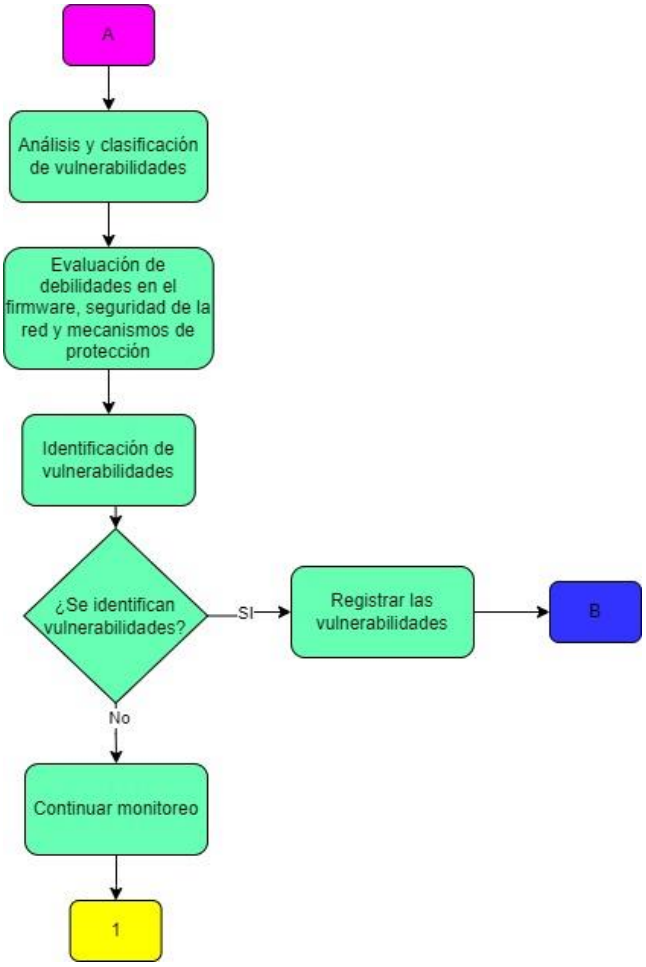
Nota. Se pueden observar los pasos para realizar la detección de las amenazas para el sistema de mitigación y para continuar con la siguiente etapa se la expresa con el literal A.

3.7.2 Análisis y Clasificación de Vulnerabilidades

Una vez que se detectan las posibles amenazas, se realiza un análisis detallado para evaluar su gravedad y nivel de riesgo. Esto implica examinar la naturaleza de la vulnerabilidad, su impacto potencial en el sistema y la probabilidad de explotación lo cual se puede observar en la Figura 13. Las vulnerabilidades se clasifican en función de su gravedad, lo que permite establecer prioridades en la implementación de medidas de mitigación.

Figura 13.

Diagrama de flujo para el Análisis y Clasificación de Vulnerabilidades.



Nota. Están plasmados los pasos para realizar la Análisis y Clasificación de Vulnerabilidades para el sistema de mitigación y para dar continuidad se dan a conocer los literales A y B referentes a las etapas tanto la anterior como posterior respectivamente.

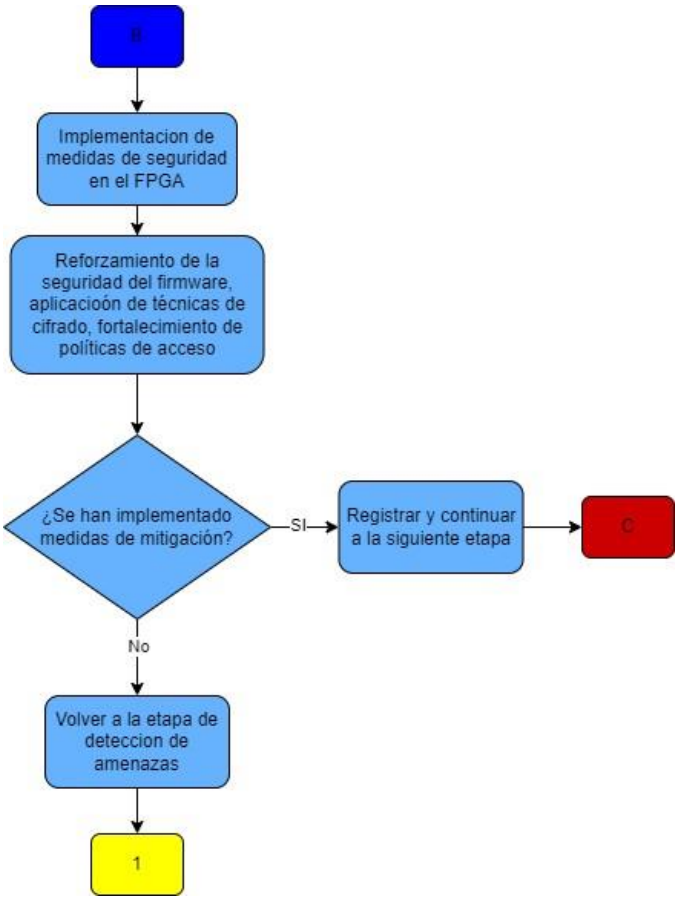
3.7.3 Implementación de Medidas de Seguridad en el FPGA

Para esta etapa, se aplican medidas de seguridad específicas en el diseño y configuración del FPGA para contrarrestar las vulnerabilidades identificadas. Esto puede incluir la utilización de técnicas de encriptación para proteger los datos almacenados y transmitidos por el FPGA.

La implementación de mecanismos de control de acceso para restringir el acceso no autorizado a recursos críticos del FPGA y la integración de algoritmos de detección y corrección de errores en el diseño del FPGA este proceso se observa en la Figura 14.

Figura 14.

Diagrama de flujo para la Implementación de Medidas de Seguridad en el FPGA.



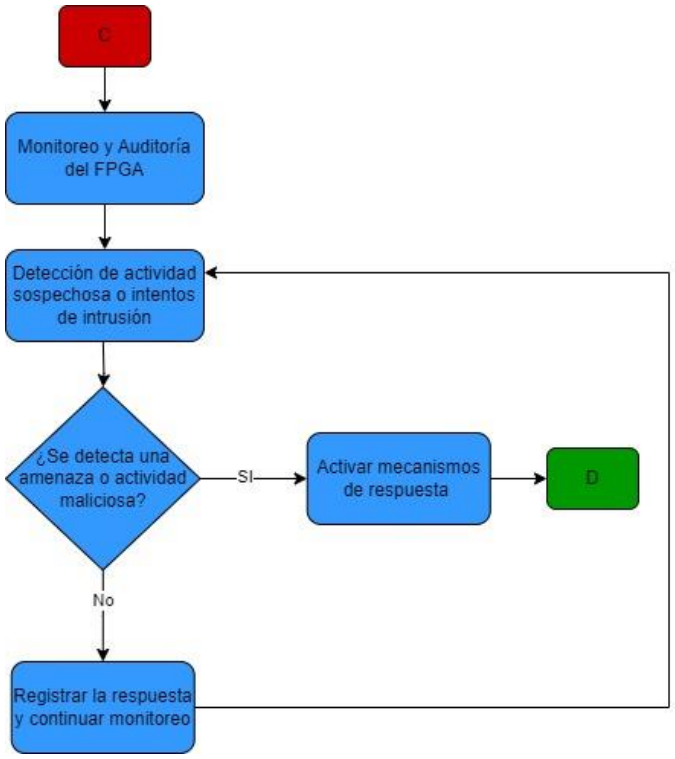
Nota. Se muestran los pasos para realizar la Implementación de Medidas de Seguridad en el FPGA para el sistema de mitigación y para dar continuidad se dan a conocer los literales A, B y C referentes a las etapas tanto la anterior como posterior respectivamente.

3.7.4 Monitoreo y Auditoría del FPGA

El sistema de mitigación incluye una etapa de monitoreo y auditoría continua del FPGA para evaluar la efectividad de las medidas de seguridad implementadas. Esto implica la recopilación de registros de actividad del FPGA, el seguimiento de eventos relevantes dentro del FPGA y la generación de informes de seguridad específicos del FPGA, los pasos se pueden observar en la Figura 15 con el diagrama de flujo. El monitoreo constante permite detectar posibles intentos de explotación o comportamientos anómalos, y la auditoría permite identificar áreas de mejora y optimización en las medidas de seguridad implementadas.

Figura 15.

Diagrama de flujo para la Monitoreo y Auditoría del FPGA.



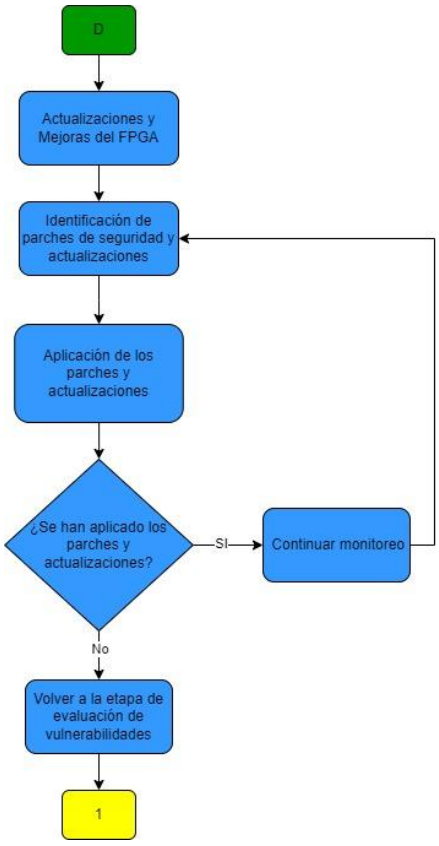
Nota. Se muestran los pasos para realizar el Monitoreo y Auditoría del FPGA para el sistema de mitigación y para dar continuidad se dan a conocer los literales C y D referentes a las etapas tanto la anterior como posterior respectivamente.

3.7.5 Actualizaciones y Mejoras del FPGA

El sistema de mitigación se mantiene actualizado y adaptable a medida que surgen nuevas amenazas y vulnerabilidades en el contexto del FPGA. Se implementan actualizaciones periódicas en el diseño y configuración del FPGA para incorporar parches de seguridad, mejoras en algoritmos de detección y corrección, y nuevas técnicas de mitigación específicas del FPGA los pasos que se pueden observar en el diagrama de flujo de la Figura 16. Esto garantiza que el sistema de mitigación siga siendo efectivo y capaz de hacer frente a las amenazas emergentes.

Figura 16.

Diagrama de flujo para la Actualizaciones y Mejoras del FPGA.



Nota. Se muestran los pasos para realizar la Actualizaciones y Mejoras del FPGA para el sistema de mitigación.

4. CAPITULO IV: Implementación, Pruebas y Resultados

En este capítulo, se presentarán de manera detallada y exhaustiva los resultados obtenidos en relación con la mitigación de vulnerabilidades en sistemas de Internet de las Cosas (IoT) mediante el uso de dispositivos FPGA. Se expondrán los diferentes escenarios y casos que pueden surgir en el contexto de los sistemas IoT, con el objetivo de resaltar las ventajas y beneficios que los dispositivos FPGA ofrecen como una solución altamente eficaz y segura.

Además, se abordarán los casos específicos de vulnerabilidades identificadas y evaluadas en los sistemas IoT, y se describirán las estrategias de mitigación aplicadas utilizando dispositivos FPGA. Se proporcionarán detalles sobre los enfoques y técnicas utilizadas para enfrentar cada vulnerabilidad, destacando los resultados obtenidos y su impacto en la seguridad general del sistema. Esta información será fundamental para comprender la eficacia de los dispositivos FPGA como medida de seguridad en sistemas IoT y su capacidad para abordar de manera efectiva diversos escenarios de vulnerabilidad.

4.1 Implementación

En este apartado, se presenta la implementación del sistema de mitigación de amenazas para sistemas IoT basado en FPGA presentado de manera gráfica en el apartado del Capítulo 3. El sistema consta de cinco etapas: detección de amenazas, análisis y clasificación de vulnerabilidades, implementación de medidas de seguridad, monitoreo del FPGA y actualizaciones y mejoras del FPGA.

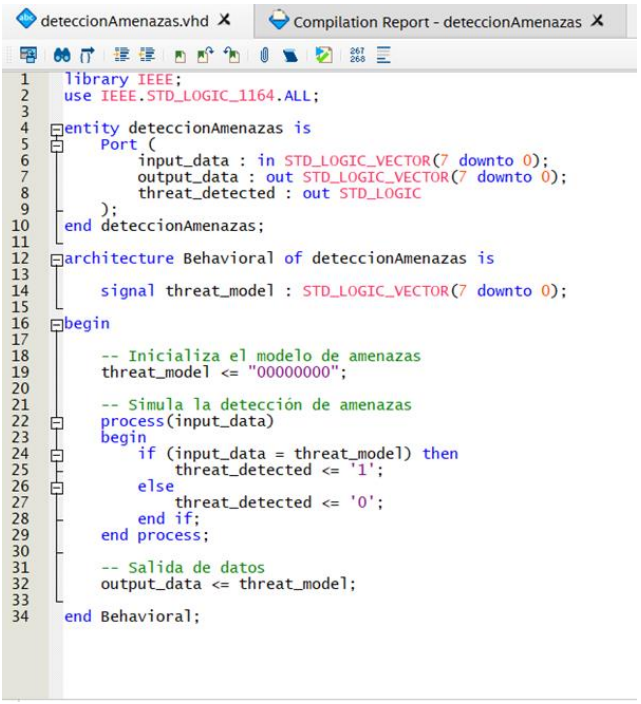
La implementación del sistema se ha realizado utilizando la herramienta Quartus Prime. El código del sistema se ha diseñado teniendo en cuenta las recomendaciones específicas que se han mencionado anteriormente.

4.1.1 Detección de Amenazas

En esta sección, se presenta la simulación de la etapa de detección de amenazas. Esta etapa se encarga de identificar las amenazas que pueden afectar al sistema IoT. Para ello, el sistema utiliza un modelo de amenazas que define las amenazas que se deben detectar.

Figura 17.

Código para la implementación de la Detección de Amenazas



Este código define un módulo detección amenazas que tiene dos puertos:

- input_data: Un puerto de entrada que recibe los datos que se van a analizar.
- output_data: Un puerto de salida que emite los datos que se han analizado.

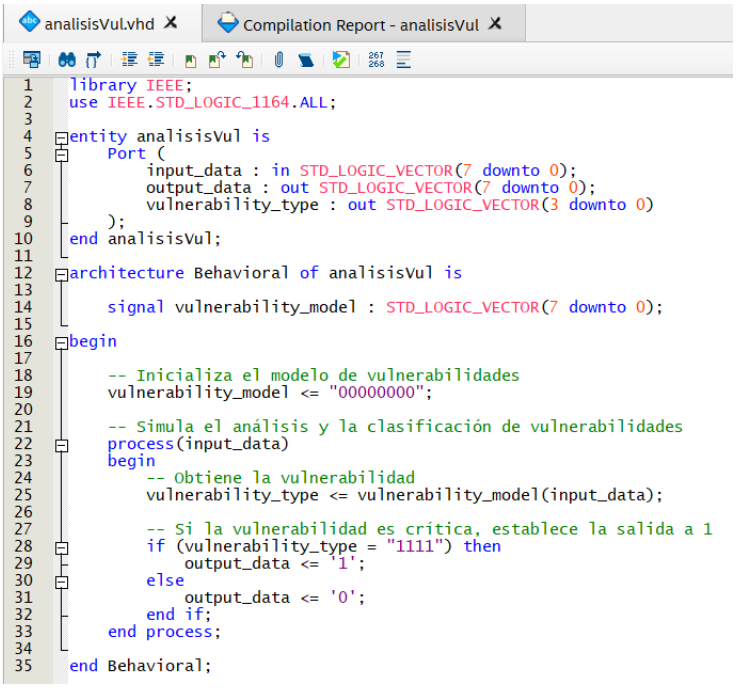
El módulo también tiene una señal interna threat_model que representa el modelo de amenazas. El proceso principal del módulo simula la detección de amenazas comparando los datos de entrada con el modelo de amenazas. Si los datos de entrada coinciden con el modelo de amenazas, entonces se establece la señal threat_detected a 1.

4.1.2 Análisis y Clasificación de Vulnerabilidades

En esta sección, se presenta la simulación de la etapa de análisis y clasificación de vulnerabilidades. Esta etapa se encarga de analizar las amenazas detectadas para identificar las vulnerabilidades que pueden aprovecharlas. Para ello, el sistema utiliza un modelo de vulnerabilidades que define las vulnerabilidades que se deben analizar.

Figura 18.

Código para la implementación de la Análisis y Clasificación de Vulnerabilidades



Este código define un módulo analisisVul que tiene dos puertos:

- input_data: Un puerto de entrada que recibe los datos que se van a analizar.
- output_data: Un puerto de salida que emite los datos que se han analizado.

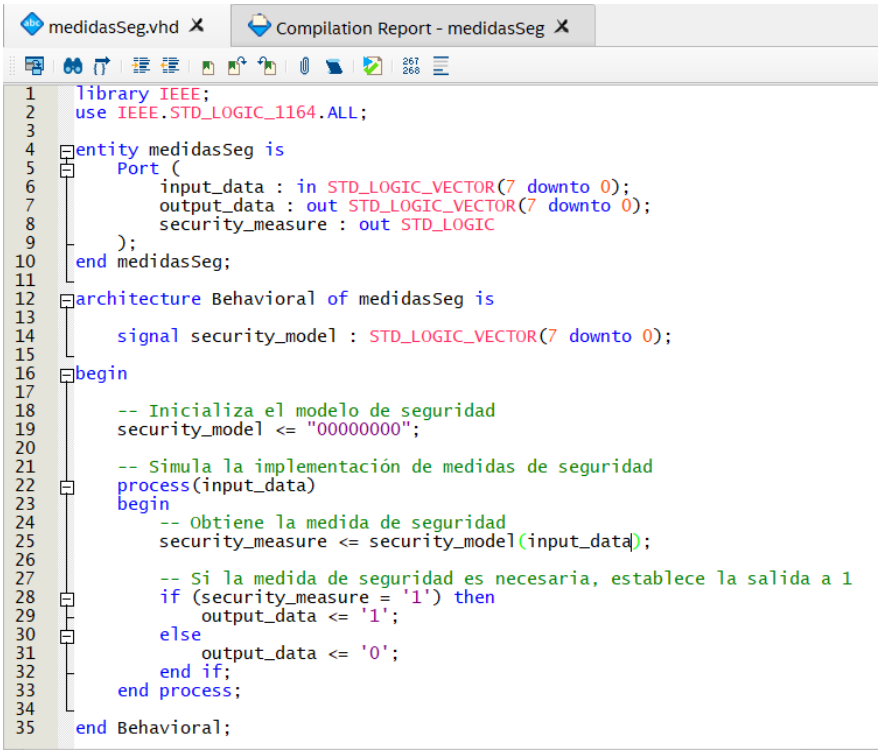
El módulo también tiene una señal interna vulnerability_model que representa el modelo de vulnerabilidades. La señal vulnerability_type representa el tipo de vulnerabilidad que se ha encontrado. Si la vulnerabilidad es crítica, entonces se establece la señal output_data a 1.

4.1.3 Medidas de Seguridad

En esta sección, se presenta la simulación de la etapa de implementación de medidas de seguridad. Esta etapa se encarga de implementar las medidas de seguridad necesarias para mitigar las vulnerabilidades identificadas. Para ello, el sistema utiliza un modelo de seguridad que define las medidas de seguridad que se deben implementar.

Figura 19.

Código para la implementación de la Medidas de Seguridad



Este código define un módulo medidasSeg que tiene tres puertos:

- input_data: Un puerto de entrada que recibe los datos que se van a analizar.
- output_data: Un puerto de salida que emite los datos que se han analizado.
- security_measure: Un puerto de salida que indica si la medida de seguridad es necesaria.

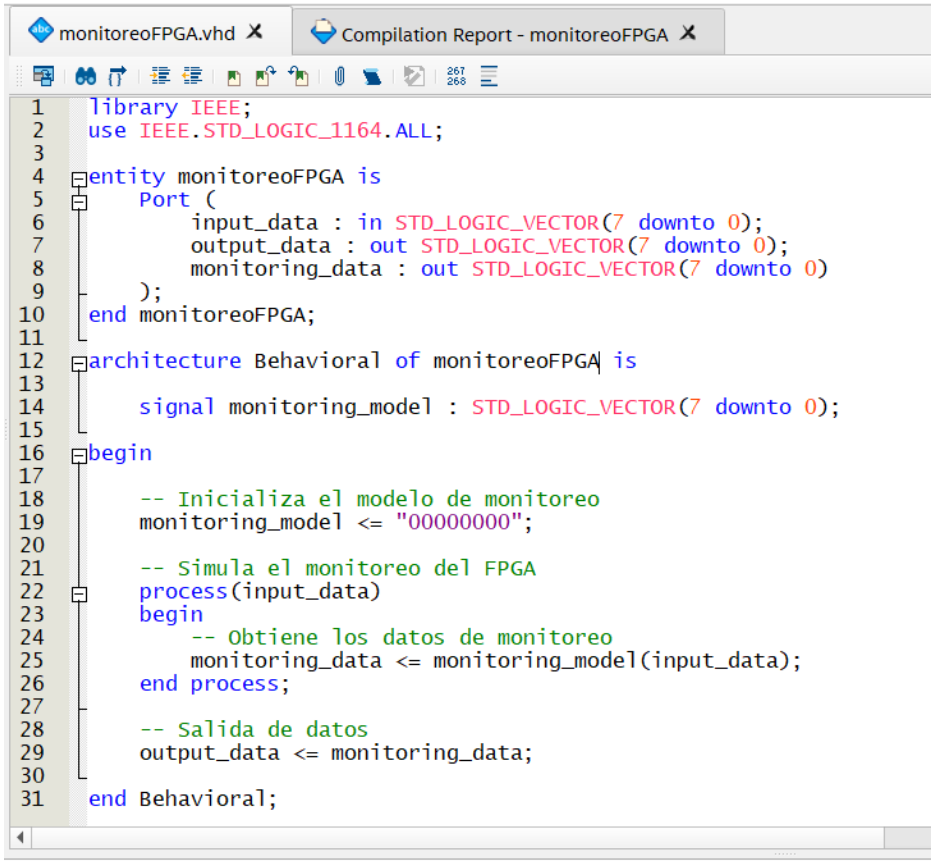
El módulo también tiene una señal interna security_model que representa el modelo de seguridad. La señal security_measure indica si la medida de seguridad es necesaria. Si la medida de seguridad es necesaria, entonces se establece la señal output_data a 1.

4.1.4 Monitoreo

En esta sección, se presenta la simulación de la etapa de monitoreo del FPGA. Esta etapa se encarga de monitorizar el FPGA para detectar cualquier cambio que pueda afectar a la seguridad del sistema. Para ello, el sistema utiliza un modelo de monitoreo que define los datos que se deben monitorizar.

Figura 20.

Código para la implementación del Monitoreo del FPGA



Este código define un módulo monitoreoFPGA que tiene tres puertos:

- input_data: Un puerto de entrada que recibe los datos que se van a analizar.
- output_data: Un puerto de salida que emite los datos que se han analizado.
- monitoring_data: Un puerto de salida que contiene los datos de monitoreo.

El módulo también tiene una señal interna monitoring_model que representa el modelo de monitoreo. El proceso principal del módulo simula el monitoreo del FPGA utilizando el modelo de monitoreo. La señal monitoring_data contiene los datos de monitoreo.

4.1.5 Actualizaciones y Mejoras del FPGA

En esta sección, se presenta la simulación de la etapa de actualizaciones y mejoras del FPGA. Esta etapa se encarga de actualizar o mejorar el FPGA para mejorar la seguridad del sistema. Para ello, el sistema utiliza un modelo de actualización que define las actualizaciones o mejoras que se deben realizar.

Figura 21.

Código para la implementación de las Actualizaciones y Mejoras del FPGA

actualizacionFPGA.vhd | Compilation Report - actualizacionFPGA

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity actualizacionFPGA is
    Port (
        input_data : in STD_LOGIC_VECTOR(7 downto 0);
        output_data : out STD_LOGIC_VECTOR(7 downto 0);
        update_data : out STD_LOGIC_VECTOR(7 downto 0)
    );
end actualizacionFPGA;

architecture Behavioral of actualizacionFPGA is

    signal update_model : STD_LOGIC_VECTOR(7 downto 0);

begin

    -- Inicializa el modelo de actualización
    update_model <= "00000000";

    -- Simula las actualizaciones y mejoras del FPGA
    process(input_data)
    begin
        -- Obtiene los datos de actualización
        update_data <= update_model(input_data);
    end process;

    -- Salida de datos
    output_data <= update_data;

end Behavioral;
```

Este código define un módulo simulation_update que tiene tres puertos:

- input_data: Un puerto de entrada que recibe los datos que se van a analizar.
- output_data: Un puerto de salida que emite los datos que se han analizado.
- update_data: Un puerto de salida que contiene los datos de actualización.

El módulo también tiene una señal interna update_model que representa el modelo de actualización. El proceso principal del módulo simula las actualizaciones y mejoras del FPGA utilizando el modelo de actualización. La señal update_data contiene los datos de actualización.

4.2 Pruebas de Funcionamiento

En este apartado se presentan las pruebas de funcionamiento realizadas para evaluar la seguridad de los sistemas IoT mediante dispositivos FPGA. Las pruebas se basaron en los parámetros definidos para el software de simulación y las vulnerabilidades con su respectiva mitigación. Los dispositivos FPGA fueron vulnerables a una serie de ataques, incluyendo ataques de inyección de fallas, ataques de canal. Las pruebas demostraron que las medidas de mitigación implementadas en los dispositivos FPGA fueron eficaces para mitigar los ataques. En ningún caso se logró explotar una vulnerabilidad para comprometer la seguridad del sistema. En base a los resultados de las pruebas se desea demostrar que los dispositivos FPGA sean una solución eficaz para mejorar la seguridad de los sistemas IoT.

4.2.1 Plan de Pruebas

Para que las pruebas seas efectivas con los parámetros ya establecidos se a realizado un cronograma de pruebas el cual va a permitir validar el funcionamiento del sistema en lo que se sería un entorno simulado la cual se puede observar en la Tabla 14.

Tabla 14.

Plan de pruebas.

Plan de Pruebas		
Tipo de Prueba	**Entorno de la Prueba**	**Resultado esperado**
Prueba Preliminar	Simulado	Demostrar que el simulador funciona correctamente y de forma fiable en base a los parámetros establecidos en un

		entorno controlado.
Prueba 1: Caso Ideal	Simulado	Evidenciar los datos que deben ser transmitidos de manera correcta para su próxima comparación.
Prueba 2: Vulnerabilidad	Simulado	Implantar los ataques establecidos a un sistema funcional para evaluar la consecuencia en los resultados.
Prueba 3: Mitigación de la Vulnerabilidad	Simulado	Instaurar la mitigación en la vulnerabilidad especifica y lograr comparar el rango de fidelidad con el caso ideal.

El plan de pruebas se ha elaborado para cubrir todos los aspectos del sistema, con el fin de evaluar su rendimiento y funcionalidad en condiciones reales.

4.2.2 Pruebas Preliminares

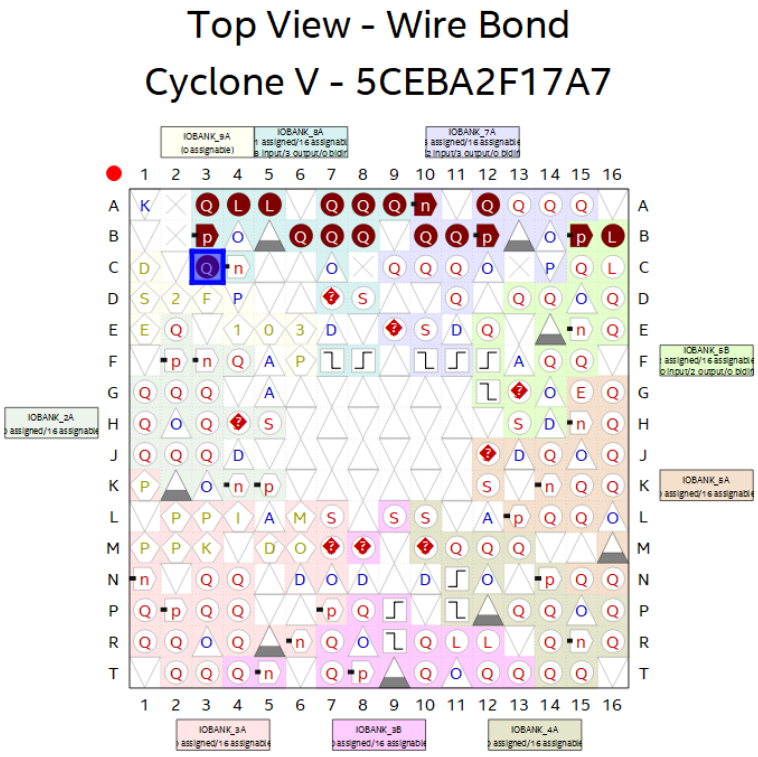
En este apartado se llevó a cabo el establecimiento de parámetros iniciales para la simulación del apartado específico con cada una de las vulnerabilidades establecidas dentro del Capítulo 3, para este caso se establece la placa con la que se van a realizar las pruebas específicas al igual que se presentan el diagrama de compuertas lógicas como también el entorno de simulación con el que se va a trabajar más adelante.

4.2.2.1 Plano de pines

El plano de pines es esencial para el diseño de sistemas con dispositivos FPGA, ya que permite establecer la conexión adecuada entre el FPGA y otros componentes del sistema, como periféricos, módulos de memoria, interfaces de comunicación, entre otros. Al visualizar el plano de pines, es posible identificar rápidamente qué pines del FPGA se utilizan para cada función específica del diseño.

Figura 22.

Plano de pines para el sistema



Nota. Se proporciona la especificación de los pines que se utilizan tanto de entrada, salida, reseteo y para el reloj.

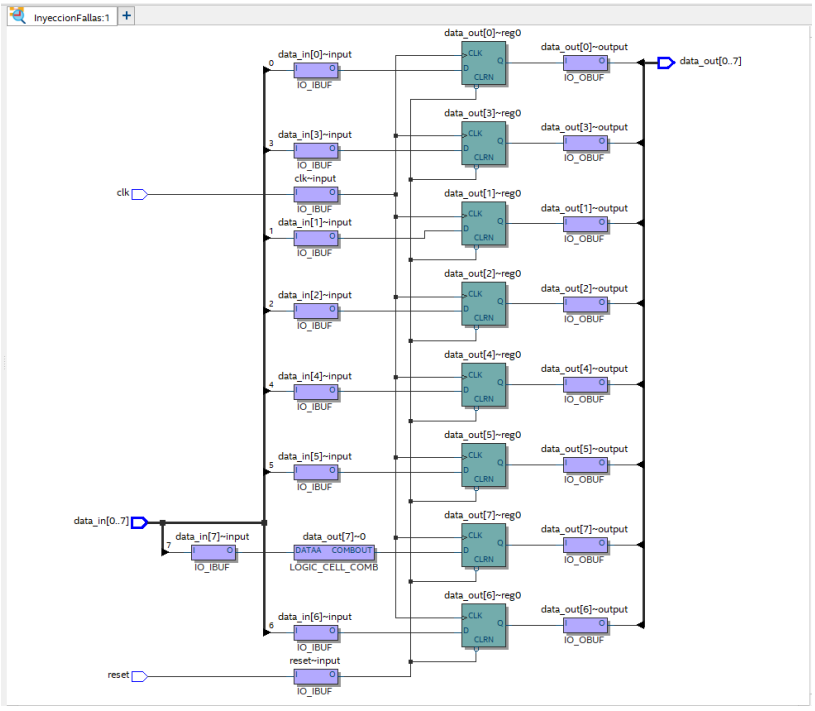
4.2.2.2 Diagrama de Compuertas Lógicas

Una vez que el diseño ha sido compilado y sintetizado en Quartus Prime, es posible visualizar el resultado en forma de diagrama de niveles lógicos. Este diagrama muestra cómo las compuertas lógicas se han implementado y conectado dentro del FPGA, proporcionando una representación gráfica del circuito lógico resultante.

Es importante destacar que el diseño y visualización de compuertas lógicas en Quartus Prime se basa en el enfoque de descripción en lenguaje de hardware, donde las compuertas se definen mediante código en VHDL o Verilog. Esto proporciona una mayor flexibilidad y capacidad de abstracción en el diseño de circuitos digitales.

Figura 23.

Apartado de las compuertas lógicas utilizadas para la simulación



Nota. En el diagrama de igual manera se observa cómo están distribuidos tanto las entradas como las salidas de los pines hacia las compuertas

4.2.3 Pruebas Específicas

Todas estas pruebas, se considera tanto el comportamiento esperado del FPGA bajo condiciones normales como la respuesta ante situaciones anómalas y ataques. Es importante llevar a cabo un enfoque completo y exhaustivo para evaluar las vulnerabilidades y las medidas de mitigación implementadas, garantizando así la seguridad y confiabilidad del sistema de Internet de las Cosas (IoT) en el que se integra el FPGA.

4.2.3.1 Pruebas en Funcionamiento Caso Ideal

Para este apartado de las pruebas se lo implementa en la medida de un ambiente ideal en donde se especifican las entradas y salidas con su respectivo procesamiento de los datos donde no se tiene ningún caso de intrusión, vulnerabilidad o ataque dentro de su funcionamiento por tanto las respuestas al procesamiento sometido deberán ser las reales en cuanto al sistema se refiere.

Figura 24.

Código Caso Ideal 8 bits

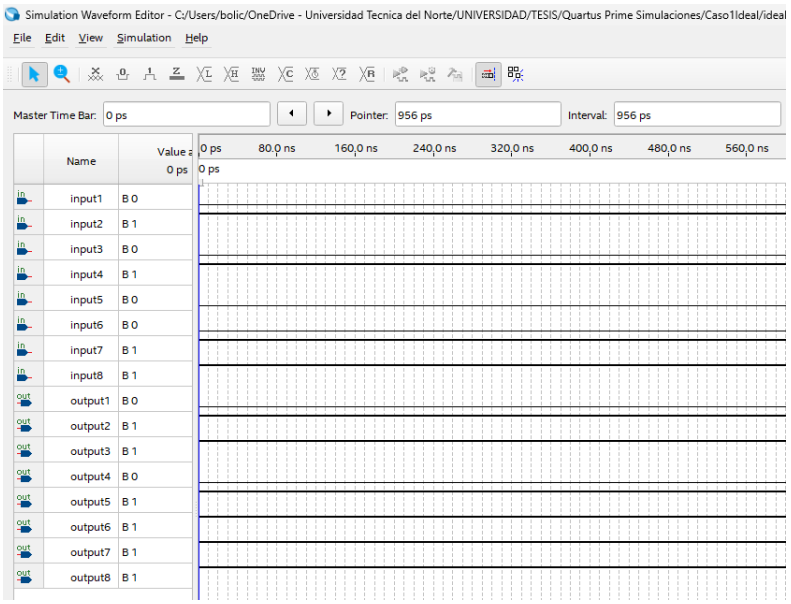
idealInyeccion.vhd | Compilation Report - idealInyeccion

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity idealInyeccion is
    Port ( input1 : in STD_LOGIC;
           input2 : in STD_LOGIC;
           input3 : in STD_LOGIC;
           input4 : in STD_LOGIC;
           input5 : in STD_LOGIC;
           input6 : in STD_LOGIC;
           input7 : in STD_LOGIC;
           input8 : in STD_LOGIC;
           output1 : out STD_LOGIC;
           output2 : out STD_LOGIC;
           output3 : out STD_LOGIC;
           output4 : out STD_LOGIC;
           output5 : out STD_LOGIC;
           output6 : out STD_LOGIC;
           output7 : out STD_LOGIC;
           output8 : out STD_LOGIC);
end idealInyeccion;

architecture Behavioral of idealInyeccion is
begin
    -- Procesamiento de las entradas utilizando compuertas lógicas
    output1 <= (input1 AND input2) OR (input3 AND input4);
    output2 <= NOT (input5 AND input6) OR (input7 AND input8);
    output3 <= NOT (input1 AND input3) OR (input5 AND input7);
    output4 <= NOT (input2 AND input4) OR (input6 AND input8);
    output5 <= NOT (input1 AND input8) OR (input2 AND input7);
    output6 <= NOT (input3 AND input6) OR (input4 AND input5);
    output7 <= NOT input1 OR input2 OR input3;
    output8 <= NOT input4 OR input5 OR input6 OR input7 OR input8;
end Behavioral;
```

Nota. Código para la simulación para del caso ideal a evaluar dentro de las pruebas de funcionamiento en 8 bits.

Figura 25.

Resultado Código Caso Ideal 8 bits.



Nota. Resultado del caso ideal para ser evaluado en los siguientes casos con 8 bits. Fuente: Autoría.

Tabla 15.

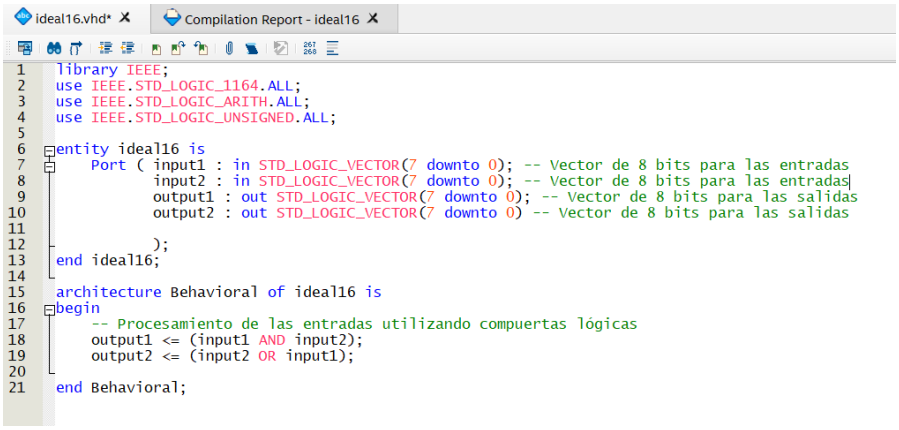
Tabla de especificación valores de entrada y salida para el caso ideal de 8 bits.

#	Valor Entrada	Valor Salida
1	0	0
2	1	1
3	0	1
4	1	0
5	0	1
6	0	1
7	1	1
8	1	1

Nota. Se especifican tanto el número de los valores como el estado tanto en la entrada como en la salida del procesamiento realizado por el código para 8 bits.

Figura 26.

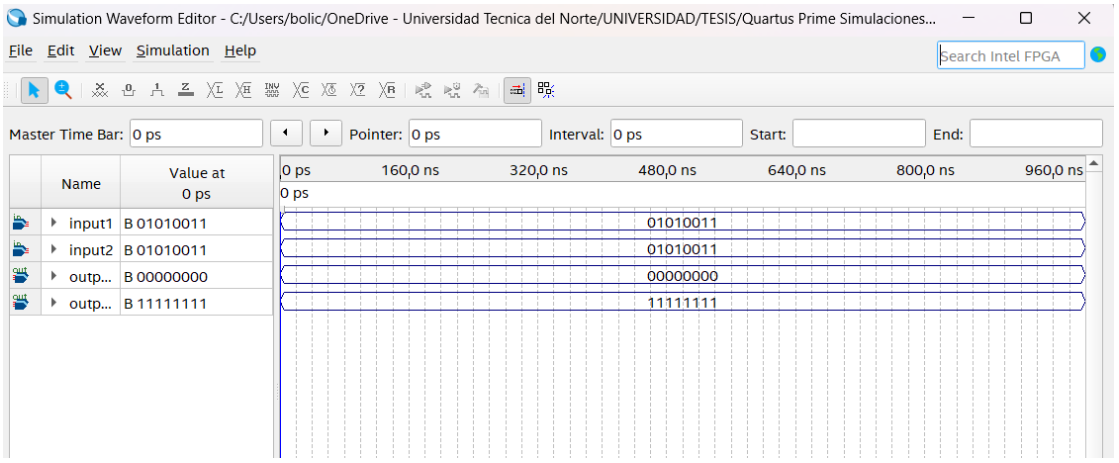
Código Caso Ideal 16 bits



Nota. Código para la simulación para del caso ideal a evaluar dentro de las pruebas de funcionamiento en 16 bits.

Figura 27.

Resultado Código Caso Ideal 16 bits.



Nota. Resultado del caso ideal para ser evaluado en los siguientes casos con 16 bits. Fuente: Autoría.

Tabla 16.

Tabla de especificación valores de entrada y salida para el caso ideal de 16 bits.

#	Valor Entrada	Valor Salida
1	01010011	00000000
2	01010011	11111111

Nota. Se especifican tanto el número de los valores como el estado tanto en la entrada como en la salida del procesamiento realizado por el código para 16 bits.

Figura 28.

Código caso ideal 32 bits

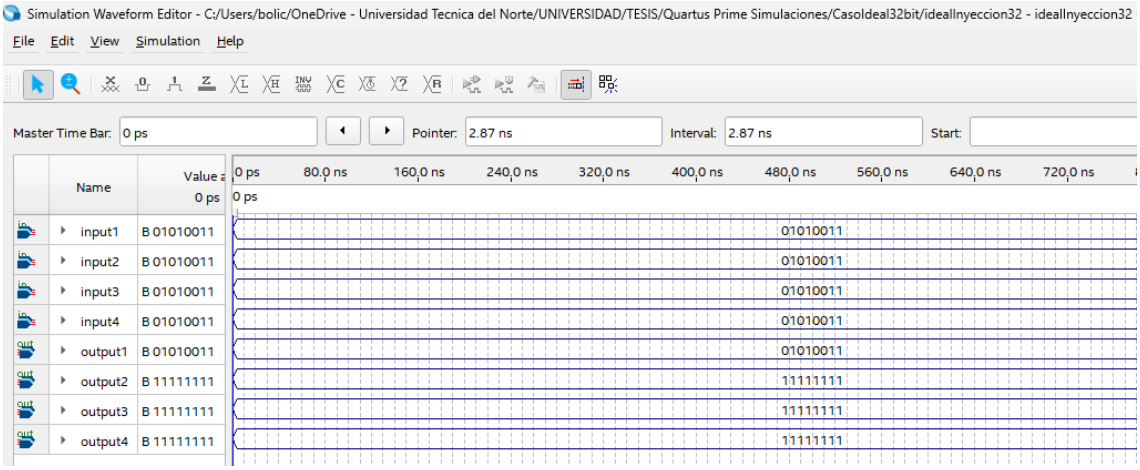
idealInyeccion32.vhd | Compilation Report - idealInyeccion32

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity idealInyeccion32 is
    Port ( input1 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input2 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input3 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input4 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           output1 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output2 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output3 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output4 : out STD_LOGIC_VECTOR(7 downto 0)-- Vector de 8 bits para las salidas
           );
end idealInyeccion32;

architecture Behavioral of idealInyeccion32 is
begin
    -- Procesamiento de las entradas utilizando compuertas lógicas
    output1 <= (input1 AND input2) OR (input3 AND input4);
    output2 <= NOT (input4 AND input1) OR (input2 AND input3);
    output3 <= NOT (input2 AND input4) OR (input3 AND input1);
    output4 <= NOT (input1 AND input2) OR (input3 AND input4);

end Behavioral;
```

Nota. Código para la simulación para del caso ideal a evaluar dentro de las pruebas de funcionamiento en 32 bits.

Figura 29.

Resultado Código Caso Ideal 32 bits.



Nota. Resultado del caso ideal para ser evaluado en los siguientes casos con 32 bits. Fuente: Autoría.

Tabla 17.

Tabla de especificación valores de entrada y salida para el caso ideal de 32 bits.

#	Valor Entrada	Valor Salida
1	01010011	01010011
2	01010011	11111111
3	01010011	11111111
4	01010011	11111111

Nota. Se especifican tanto el número de los valores como el estado tanto en la entrada como en la salida del procesamiento realizado por el código para 32 bits.

Figura 30.

Código caso ideal 64 bits

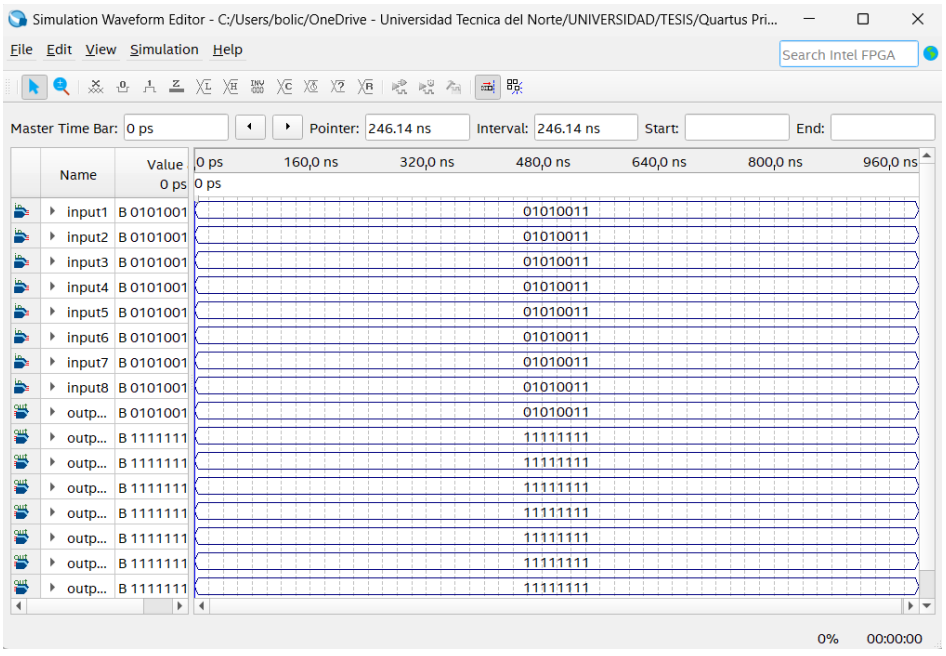
idealInyeccion64.vhd | Compilation Report - idealInyeccion64

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity idealInyeccion64 is
    Port ( input1 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input2 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input3 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input4 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input5 : in STD_LOGIC_VECTOR(7 downto 0);
           input6 : in STD_LOGIC_VECTOR(7 downto 0);
           input7 : in STD_LOGIC_VECTOR(7 downto 0);
           input8 : in STD_LOGIC_VECTOR(7 downto 0);
           output1 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output2 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output3 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output4 : out STD_LOGIC_VECTOR(7 downto 0);-- Vector de 8 bits para las salidas
           output5 : out STD_LOGIC_VECTOR(7 downto 0);
           output6 : out STD_LOGIC_VECTOR(7 downto 0);
           output7 : out STD_LOGIC_VECTOR(7 downto 0);
           output8 : out STD_LOGIC_VECTOR(7 downto 0)
           );
end idealInyeccion64;

architecture Behavioral of idealInyeccion64 is
begin
    -- Procesamiento de las entradas utilizando compuertas lógicas
    output1 <= (input1 AND input2) OR (input3 AND input4);
    output2 <= NOT (input5 AND input6) OR (input7 AND input8);
    output3 <= NOT (input1 AND input3) OR (input5 AND input7);
    output4 <= NOT (input2 AND input4) OR (input6 AND input8);
    output5 <= NOT (input1 AND input8) OR (input2 AND input7);
    output6 <= NOT (input3 AND input6) OR (input4 AND input5);
    output7 <= NOT input1 OR input2 OR input3;
    output8 <= NOT input4 OR input5 OR input6 OR input7 OR input8;

end Behavioral;
```

Nota. Código para la simulación para del caso ideal a evaluar dentro de las pruebas de funcionamiento en 64 bits.

Figura 31.

Resultado Código Caso Ideal 64 bits



Nota. Resultado del caso ideal para ser evaluado en los siguientes casos con 64 bits. Fuente: Autoría.

Tabla 18.

Tabla de especificación valores de entrada y salida para el caso ideal de 64 bits.

#	Valor Entrada	Valor Salida
1	01010011	01010011
2	01010011	11111111
3	01010011	11111111
4	01010011	11111111
5	01010011	11111111
6	01010011	11111111
7	01010011	11111111
8	01010011	11111111

Nota. Se especifican tanto el número de los valores como el estado tanto en la entrada como en la salida del procesamiento realizado por el código para 64 bits.

Para este caso en los códigos se puede observar cómo se definen tanto las entradas como las salidas solo con la peculiaridad de la diferencia de los bits a evaluar, para lo cual se utiliza el lenguaje solicitado para el entorno de Quartus Prime, de igual manera se puede visualizar el procesamiento que se realiza para las entradas y vinculándolos a cada una de las salidas presentes para esta ocasión.

En las figuras se observan los resultados como parte de la simulación en cuanto el código presentado con anterioridad, se puede observar la especificación de los valores estado de los pines de entrada como también los valores estado de los pines de salida. Para una correcta comparación con los siguientes casos de las pruebas de funcionamiento se procede con la presentación de todos los valores tanto de entrada como de salida en la Tablas presentes.

4.2.4 Pruebas con Vulnerabilidades

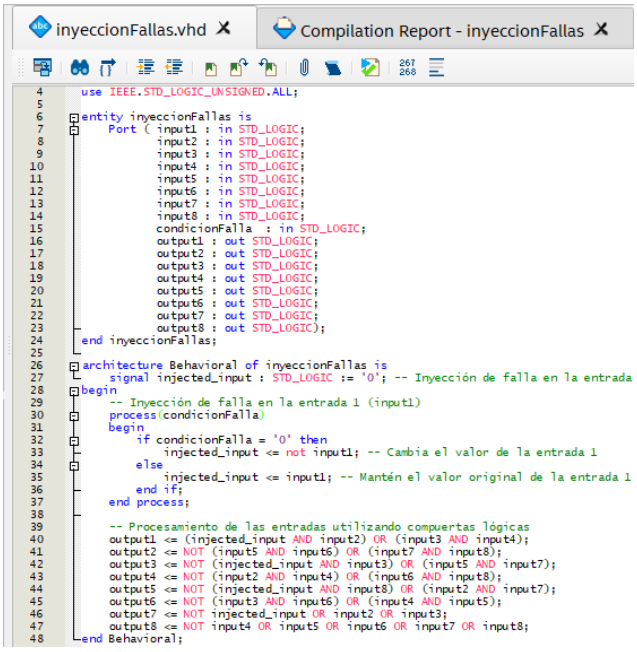
Para las pruebas de funcionamiento relacionadas con la inyección de fallas y el ataque del canal lateral, se han diseñado procedimientos exhaustivos que permiten evaluar la resistencia y robustez del sistema ante estas vulnerabilidades potenciales. Estas pruebas buscan garantizar la integridad, seguridad y fiabilidad del sistema, asegurando su correcto funcionamiento incluso en escenarios adversos.

4.2.4.1 Inyección de Fallas

En esta prueba, se simula la inyección de fallas en el FPGA para evaluar cómo responde el sistema ante estas situaciones. Se seleccionan puntos específicos en el diseño donde se introducen errores en los datos y se observa cómo afecta al sistema FPGA para el tratamiento de datos presente en un sistema IoT.

Figura 32.

Código para la inyección de fallas 8 bits

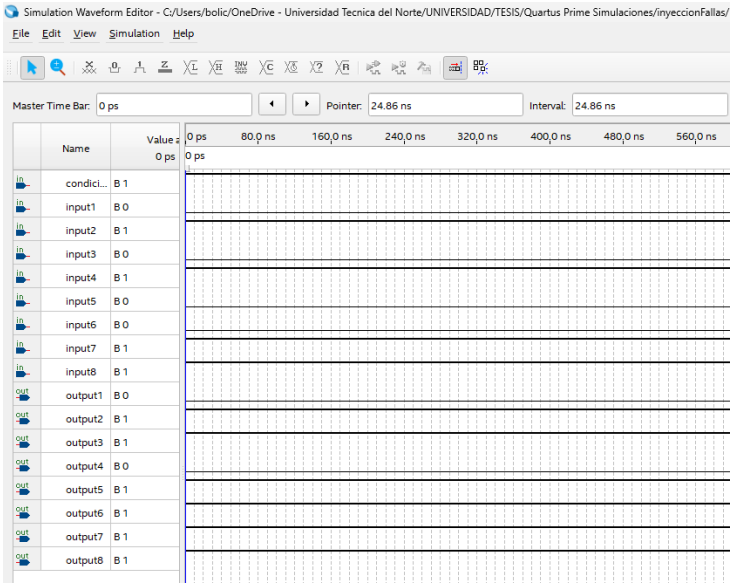


```
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity inyeccionFallas is
    Port ( input1 : in STD_LOGIC;
           input2 : in STD_LOGIC;
           input3 : in STD_LOGIC;
           input4 : in STD_LOGIC;
           input5 : in STD_LOGIC;
           input6 : in STD_LOGIC;
           input7 : in STD_LOGIC;
           input8 : in STD_LOGIC;
           condicionFalla  : in STD_LOGIC;
           output1 : out STD_LOGIC;
           output2 : out STD_LOGIC;
           output3 : out STD_LOGIC;
           output4 : out STD_LOGIC;
           output5 : out STD_LOGIC;
           output6 : out STD_LOGIC;
           output7 : out STD_LOGIC;
           output8 : out STD_LOGIC);
end inyeccionFallas;

architecture Behavioral of inyeccionFallas is
    signal injected_input : STD_LOGIC := '0'; -- Inyección de falla en la entrada
begin
    -- Inyección de falla en la entrada 1 (input1)
    process(condicionFalla)
    begin
        if condicionFalla = '0' then
            injected_input <= not input1; -- Cambia el valor de la entrada 1
        else
            injected_input <= input1; -- Mantén el valor original de la entrada 1
        end if;
    end process;

    -- Procesamiento de las entradas utilizando compuertas lógicas
    output1 <= (injected_input AND input2) OR (input3 AND input4);
    output2 <= NOT (input5 AND input6) OR (input7 AND input8);
    output3 <= NOT (injected_input AND input3) OR (input5 AND input7);
    output4 <= NOT (input2 AND input4) OR (input6 AND input8);
    output5 <= NOT (injected_input AND input8) OR (input2 AND input7);
    output6 <= NOT (input3 AND input6) OR (input4 AND input5);
    output7 <= NOT injected_input OR input2 OR input3;
    output8 <= NOT input4 OR input5 OR input6 OR input7 OR input8;
end Behavioral;
```

Nota. Código para la simulación con la inyección de fallas dentro de las pruebas de funcionamiento en 8 bits.

Figura 33.

Comportamiento de la inyección de fallas para la salida en el software de simulación 8 bits



Nota. Resultado del caso con la inyección de fallas para ser mitigado en el siguiente apartado con 8 bits. Fuente: Autoría.

Tabla 19.

Valores de Entrada y Salida para la inyección de fallas 8 bits.

#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	0	1	0	X
2	1	1	1	✓
3	0	1	1	✓
4	1	1	0	X
5	0	0	1	X
6	0	1	1	✓
7	1	1	1	✓
8	1	1	1	✓
		Total Errores		3

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores con 8 bits.

Figura 34.

Código para la inyección de fallas 16 bits.

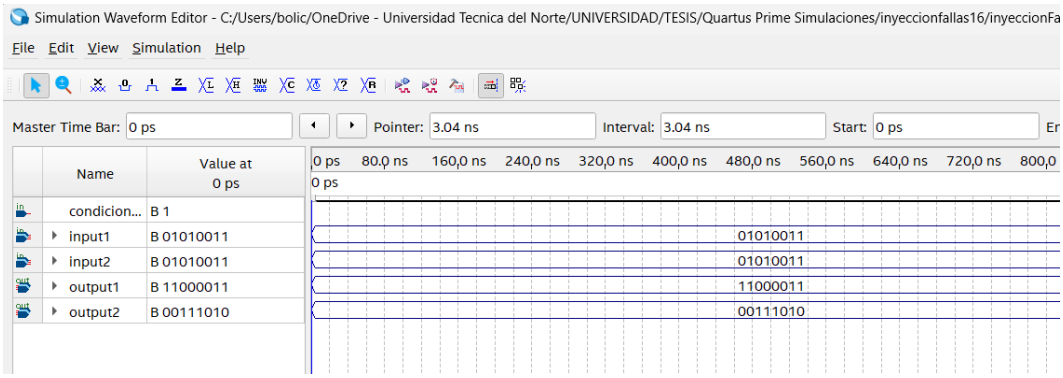
inyeccionFallas32.vhd | Compilation Report - inyeccionFallas32

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity inyeccionFallas32 is
    Port ( input1 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input2 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input3 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input4 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           condicionFalla  : in STD_LOGIC;
           output1 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output2 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output3 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output4 : out STD_LOGIC_VECTOR(7 downto 0)-- Vector de 8 bits para las salidas
           );

end inyeccionFallas32;

architecture Behavioral of inyeccionFallas32 is
    signal injected_input : STD_LOGIC_VECTOR(7 downto 0) := "00000000"; -- Inyección de fall
begin
    -- Inyección de falla en la entrada 1 (input1)
    process(condicionFalla)
    begin
        if condicionFalla = '0' then
            injected_input <= not input1; -- Cambia el valor de la entrada 1
        else
            injected_input <= input1; -- Mantén el valor original de la entrada 1
        end if;
    end process;

    -- Procesamiento de las entradas utilizando compuertas lógicas
    output1 <= (injected_input AND input2) OR (input3 AND input4);
    output2 <= NOT (input4 AND injected_input) OR (input2 AND input3);
    output3 <= NOT (input2 AND input4) OR (input3 AND injected_input);
    output4 <= NOT (injected_input AND input2) OR (input3 AND input4);
end Behavioral;
```

Nota. Código para la simulación con la inyección de fallas dentro de las pruebas de funcionamiento en 16 bits.

Figura 35.

Comportamiento de la inyección de fallas para la salida en el software de simulación 16 bits



Nota. Resultado del caso con la inyección de fallas para ser mitigado en el siguiente apartado con 16 bits. Fuente: Autoría.

Tabla 20.

Valores de Entrada y Salida para la inyección de fallas 16 bits.

#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	01010011	11000011	00000000	X
2	01010011	00111010	11111111	X
		Total Bits Erróneos		8

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores con 16 bits.

Figura 36.

Código para la inyección de fallas 32 bits.

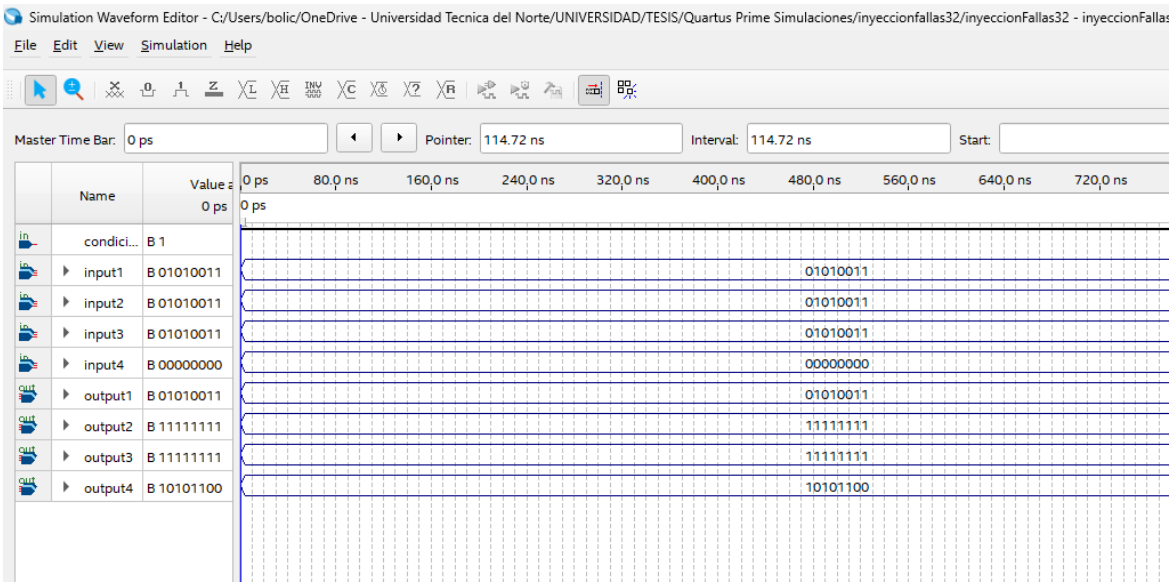
inyeccionFallas32.vhd | Compilation Report - inyeccionFallas32

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity inyeccionFallas32 is
    Port ( input1 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input2 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input3 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input4 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           condicionFalla  : in STD_LOGIC;
           output1 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output2 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output3 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output4 : out STD_LOGIC_VECTOR(7 downto 0)-- Vector de 8 bits para las salidas
           );

end inyeccionFallas32;

architecture Behavioral of inyeccionFallas32 is
    signal injected_input : STD_LOGIC_VECTOR(7 downto 0) := "00000000"; -- Inyección de falla en la entrada 1
begin
    -- Inyección de falla en la entrada 1 (input1)
    process(condicionFalla)
    begin
        if condicionFalla = '0' then
            injected_input <= not input1; -- Cambia el valor de la entrada 1
        else
            injected_input <= input1; -- Mantén el valor original de la entrada 1
        end if;
    end process;

    -- Procesamiento de las entradas utilizando compuertas lógicas
    output1 <= (injected_input AND input2) OR (input3 AND input4);
    output2 <= NOT (input4 AND injected_input) OR (input2 AND input3);
    output3 <= NOT (input2 AND input4) OR (input3 AND injected_input);
    output4 <= NOT (injected_input AND input2) OR (input3 AND input4);
end Behavioral;
```

Nota. Código para la simulación con la inyección de fallas dentro de las pruebas de funcionamiento en 32 bits.

Figura 37.

Comportamiento de la inyección de fallas para la salida en el software de simulación 32 bits



Nota. Resultado del caso con la inyección de fallas para ser mitigado en el siguiente apartado con 32 bits. Fuente: Autoría.

Tabla 21.

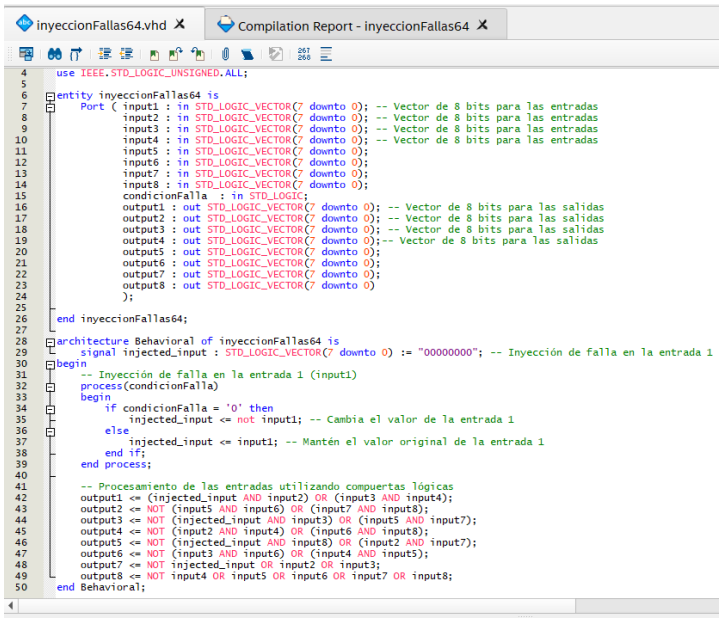
Valores de Entrada y Salida para la inyección de fallas 32 bits.

#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	01010011	01010011	01010011	✓
2	01010011	11111111	11111111	✓
3	01010011	11111111	11111111	✓
4	01010011	01010011	11111111	X
		Total Bits Erróneos		4

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores con 32 bits.

Figura 38.

Código para la inyección de fallas 64 bits.

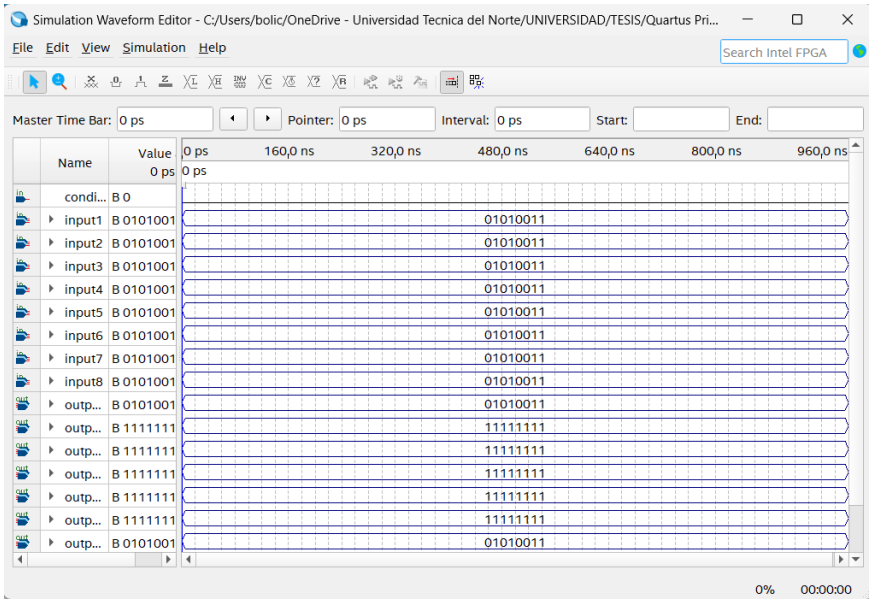


```
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity inyeccionFallas64 is
    Port ( input1 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input2 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input3 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input4 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input5 : in STD_LOGIC_VECTOR(7 downto 0);
           input6 : in STD_LOGIC_VECTOR(7 downto 0);
           input7 : in STD_LOGIC_VECTOR(7 downto 0);
           input8 : in STD_LOGIC_VECTOR(7 downto 0);
           condicionFalla  : in STD_LOGIC;
           output1 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output2 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output3 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output4 : out STD_LOGIC_VECTOR(7 downto 0);-- Vector de 8 bits para las salidas
           output5 : out STD_LOGIC_VECTOR(7 downto 0);
           output6 : out STD_LOGIC_VECTOR(7 downto 0);
           output7 : out STD_LOGIC_VECTOR(7 downto 0);
           output8 : out STD_LOGIC_VECTOR(7 downto 0)
           );

end inyeccionFallas64;

architecture Behavioral of inyeccionFallas64 is
    signal injected_input : STD_LOGIC_VECTOR(7 downto 0) := "00000000"; -- Inyección de falla en la entrada 1
begin
    -- Inyección de falla en la entrada 1 (input1)
    process(condicionFalla)
    begin
        if condicionFalla = '0' then
            injected_input <= not input1; -- Cambia el valor de la entrada 1
        else
            injected_input <= input1; -- Mantén el valor original de la entrada 1
        end if;
    end process;

    -- Procesamiento de las entradas utilizando compuertas lógicas
    output1 <= (injected_input AND input2) OR (input3 AND input4);
    output2 <= NOT (input5 AND input6) OR (input7 AND input8);
    output3 <= NOT (injected_input AND input3) OR (input5 AND input7);
    output4 <= NOT (input2 AND input4) OR (input6 AND input8);
    output5 <= NOT (injected_input AND input8) OR (input2 AND input7);
    output6 <= NOT (input3 AND input6) OR (input4 AND input5);
    output7 <= NOT injected_input OR input2 OR input3;
    output8 <= NOT input4 OR input5 OR input6 OR input7 OR input8;
end Behavioral;
```

Nota. Código para la simulación con la inyección de fallas dentro de las pruebas de funcionamiento en 64 bits.

Figura 39.

Comportamiento de la inyección de fallas para la salida en el software de simulación 64 bits



Nota. Resultado del caso con la inyección de fallas para ser mitigado en el siguiente apartado con 64 bits. Fuente: Autoría.

Tabla 22.

Valores de Entrada y Salida para la inyección de fallas 64 bits.

#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	01010011	01010011	01010011	✓
2	01010011	11111111	11111111	✓
3	01010011	11111111	11111111	✓
4	01010011	11111111	11111111	✓
5	01010011	11111111	11111111	✓
6	01010011	11111111	11111111	✓
7	01010011	11111111	11111111	✓
8	01010011	01010011	11111111	X
		Total Bits Erróneos		4

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores con 64 bits.

En los códigos, se definen los puertos: Inputs y Outputs, todos como vectores de 8 bits (STD_LOGIC_VECTOR(7 downto 0)). Luego, se realiza el procesamiento de las entradas utilizando compuertas lógicas y asignado los resultados a los elementos correspondientes del vector de salida Outputs. Cada elemento de Outputs se calcula a partir de elementos correspondientes en Inputs.

Como se puede observar se tiene en cuenta la base del caso ideal para la salida del caso con vulnerabilidad establecida en este caso al activar la vulnerabilidad se tiene la salida errónea en los pines. En este caso se puede observar en el panel de los flancos de subida y de bajada como se va dando un error en el apartado de la salida con el que se establece la falla definida en este caso con un valor incorrecto al que se debe tener como se observa en la Figuras específicamente en la primera entrada. Se observa los valores erróneos de salida con la inyección de fallas corriendo para lo cual se enumera los errores para este caso y se tiene el total de los valores erróneos que como se sabe en datos binarios es muy significativo el hecho que existan estos errores para la transmisión de datos en los diferentes sistemas existentes.

- ✓ **Comportamiento inyección de fallas**

El comportamiento de la inyección de fallas se refiere a cómo se manifiestan las alteraciones o errores introducidos de manera controlada en el sistema durante las pruebas. Este comportamiento puede variar dependiendo de varios factores, como el tipo de falla inyectada, la ubicación en el sistema donde se realiza la inyección y las características del sistema en sí. Cuando se realiza una inyección de fallas, se espera que el sistema reaccione de alguna manera ante la alteración introducida. Algunos posibles comportamientos que pueden observarse incluyen:

Error en el funcionamiento: El sistema puede presentar errores en la ejecución de instrucciones, cálculos incorrectos o comportamientos inesperados en la lógica de control.

Fallo del sistema: En casos más extremos, la inyección de fallas puede llevar al sistema a un estado de fallo completo, donde deja de funcionar correctamente y se detiene por completo.

4.2.4.2 Ataque Canal Lateral

Se simulan ataques de canal lateral para intentar obtener información sensible del sistema, como claves de cifrado o datos confidenciales. Se analiza cómo el sistema responde a estos ataques en el caso de que se lo implemente en base a un sistema IoT.

Figura 40.

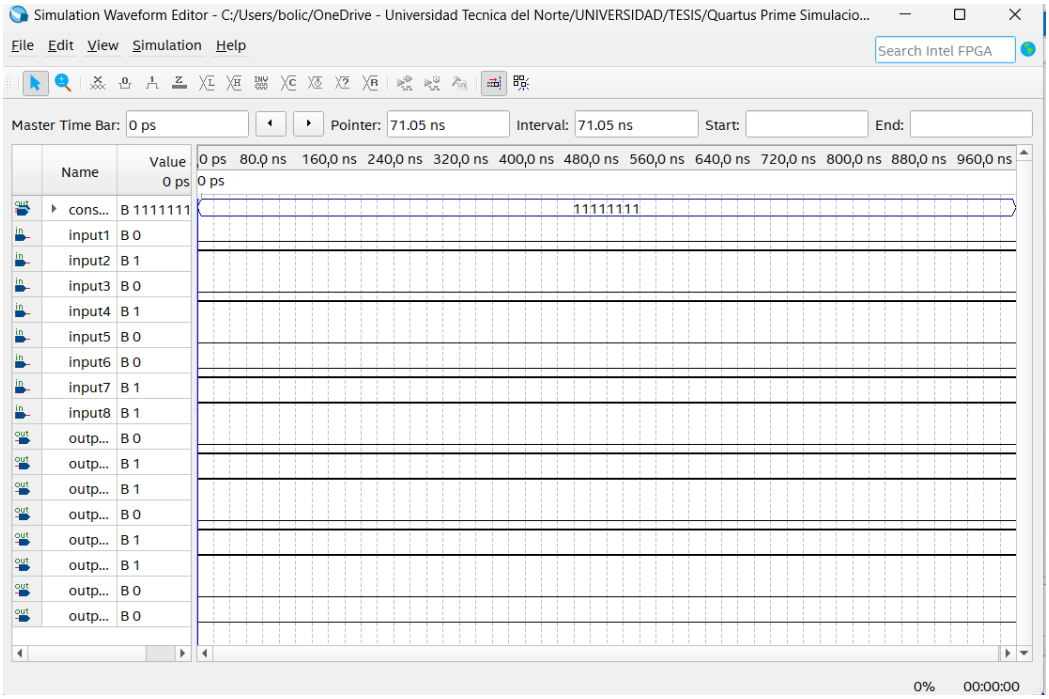
Código para la simulación del Ataque del Canal Lateral con 8 bits.

```
13              input7 : in STD_LOGIC;
14              input8 : in STD_LOGIC;
15              output1 : out STD_LOGIC;
16              output2 : out STD_LOGIC;
17              output3 : out STD_LOGIC;
18              output4 : out STD_LOGIC;
19              output5 : out STD_LOGIC;
20              output6 : out STD_LOGIC;
21              output7 : out STD_LOGIC;
22              output8 : out STD_LOGIC;
23              ataqueCanal : out STD_LOGIC); -- Se agrega una señal de consumo de energía
24  end canalLateral8;
25
26  architecture Behavioral of canalLateral8 is
27  begin
28      -- Procesamiento de las entradas utilizando compuertas lógicas
29      output1 <= (input1 AND input2) OR (input3 AND input4);
30      output2 <= NOT (input5 AND input6) OR (input7 AND input8);
31      output3 <= NOT (input1 AND input3) OR (input5 AND input7);
32      output4 <= NOT (input2 AND input4) OR (input6 AND input8);
33      output5 <= NOT (input1 AND input8) OR (input2 AND input7);
34      output6 <= NOT (input3 AND input6) OR (input4 AND input5);
35      output7 <= NOT input1 OR input2 OR input3;
36      output8 <= NOT input4 OR input5 OR input6 OR input7 OR input8;
37
38      -- Simulación de un ataque de canal lateral (consumo de energía)
39      process(input1, input2, input3, input4, input5, input6, input7, input8)
40      begin
41          if (input1 = '1' and input2 = '1') then
42              ataqueCanal <= '1';  -- Alto consumo de energía
43              output8 <= ataqueCanal;
44              output6 <= ataqueCanal;
45          else
46              ataqueCanal <= '0';  -- Bajo consumo de energía
47              output5 <=  ataqueCanal;
48              output7 <= ataqueCanal;
49          end if;
50      end process;
51  end Behavioral;
```

Nota. Código para la simulación con el ataque del canal lateral dentro de las pruebas de funcionamiento en 8 bits. Fuente: Autoría

Figura 41.

Comportamiento Ataque canal Lateral 8 bits



Nota. Resultado del caso con el ataque del canal lateral, esto para ser mitigado en el siguiente apartado con 8 bits. Fuente: Autoría.

Tabla 23.

Valores de Entrada y Salida para el Ataque Canal Lateral 8 bits.

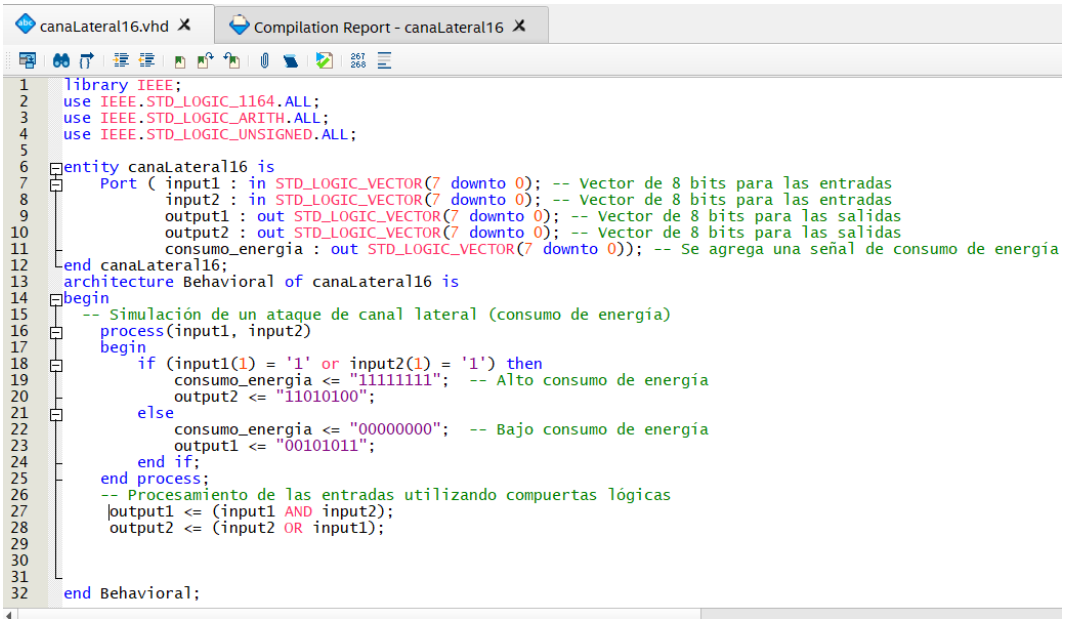
#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	0	0	0	✓
2	1	1	1	✓
3	0	1	1	✓
4	1	0	0	✓

5	0	1	1	✓
6	0	1	1	✓
7	1	0	1	X
8	1	0	1	X
		Total Errores		2

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores en el ataque del canal lateral con 8 bits.

Figura 42.

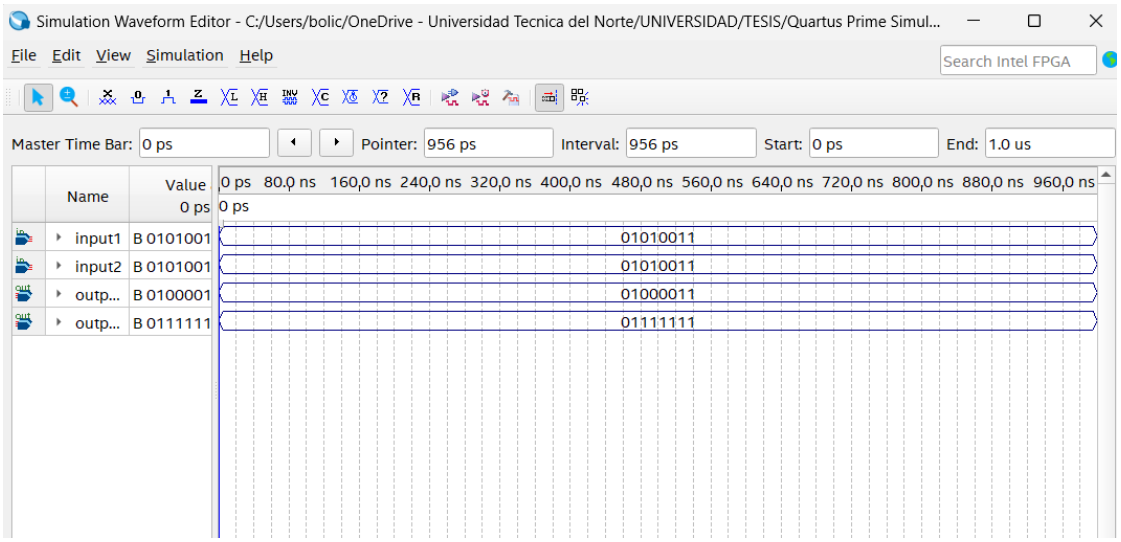
Código para la simulación del Ataque del Canal Lateral con 16 bits.



Nota. Código para la simulación con el ataque del canal lateral dentro de las pruebas de funcionamiento en 16 bits. Fuente: Autoría

Figura 43.

Comportamiento Ataque canal Lateral 16 bits



Nota. Resultado del caso con el ataque del canal lateral, esto para ser mitigado en el siguiente apartado con 16 bits. Fuente: Autoría.

Tabla 24.

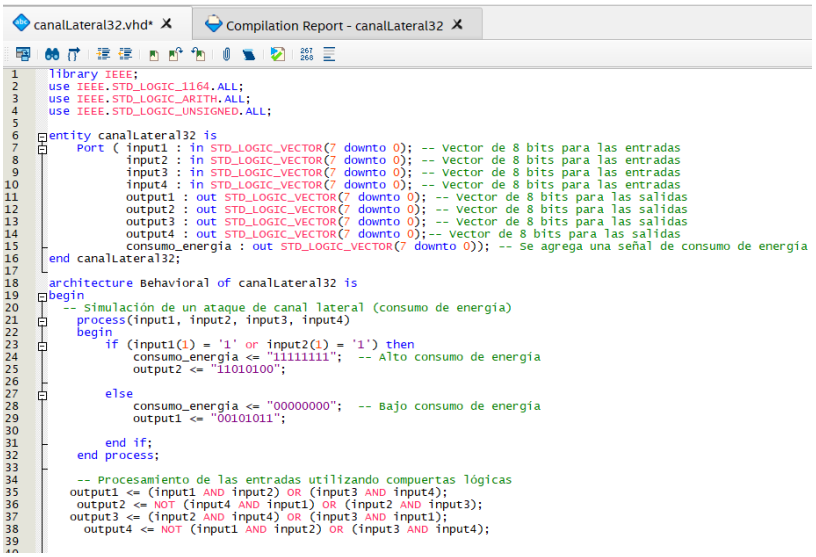
Valores de Entrada y Salida para la Ataque del Canal Lateral 16 bits.

#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	01010011	10000011	00000000	X
2	01010011	01111111	11111111	X
		Total Bits Erróneos		4

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores en el ataque del canal lateral con 16 bits.

Figura 44.

Código para la simulación del Ataque del Canal Lateral con 32 bits.

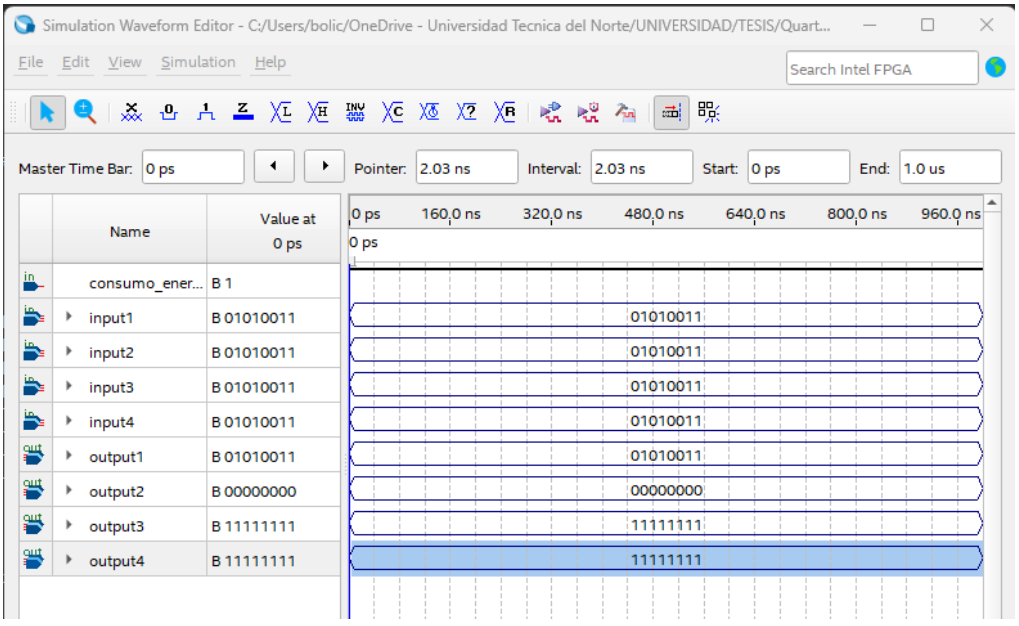


```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity canalLateral32 is
    Port ( input1 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input2 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input3 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input4 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           output1 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output2 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output3 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output4 : out STD_LOGIC_VECTOR(7 downto 0);-- Vector de 8 bits para las salidas
           consumo_energia : out STD_LOGIC_VECTOR(7 downto 0)); -- Se agrega una señal de consumo de energía
end canalLateral32;

architecture Behavioral of canalLateral32 is
begin
    -- Simulación de un ataque de canal lateral (consumo de energía)
    process(input1, input2, input3, input4)
    begin
        if (input1(1) = '1' or input2(1) = '1') then
            consumo_energia <= "11111111";  -- Alto consumo de energía
            output2 <= "11010100";

        else
            consumo_energia <= "00000000";  -- Bajo consumo de energía
            output1 <= "00101011";

        end if;
    end process;

    -- Procesamiento de las entradas utilizando compuertas lógicas
    output1 <= (input1 AND input2) OR (input3 AND input4);
    output2 <= NOT (input4 AND input1) OR (input2 AND input3);
    output3 <= (input2 AND input4) OR (input3 AND input1);
    output4 <= NOT (input1 AND input2) OR (input3 AND input4);
```

Nota. *Código para la simulación con el ataque del canal lateral dentro de las pruebas de funcionamiento en 32 bits.* Fuente: Autoría

Figura 45.

Comportamiento Ataque canal Lateral 32 bits



Nota. Resultado del caso con el ataque del canal lateral, esto para ser mitigado en el siguiente apartado con 32 bits. Fuente: Autoría.

Tabla 25.

Valores de Entrada y Salida para el Ataque Canal Lateral 32 bits.

#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	01010011	01010011	01010011	✓
2	01010011	11111111	11111111	✓
3	01010011	11111111	11111111	✓
4	01010011	01010011	11111111	X
		Total Bits Erróneos		4

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores en el ataque del canal lateral con 32 bits.

Figura 46.

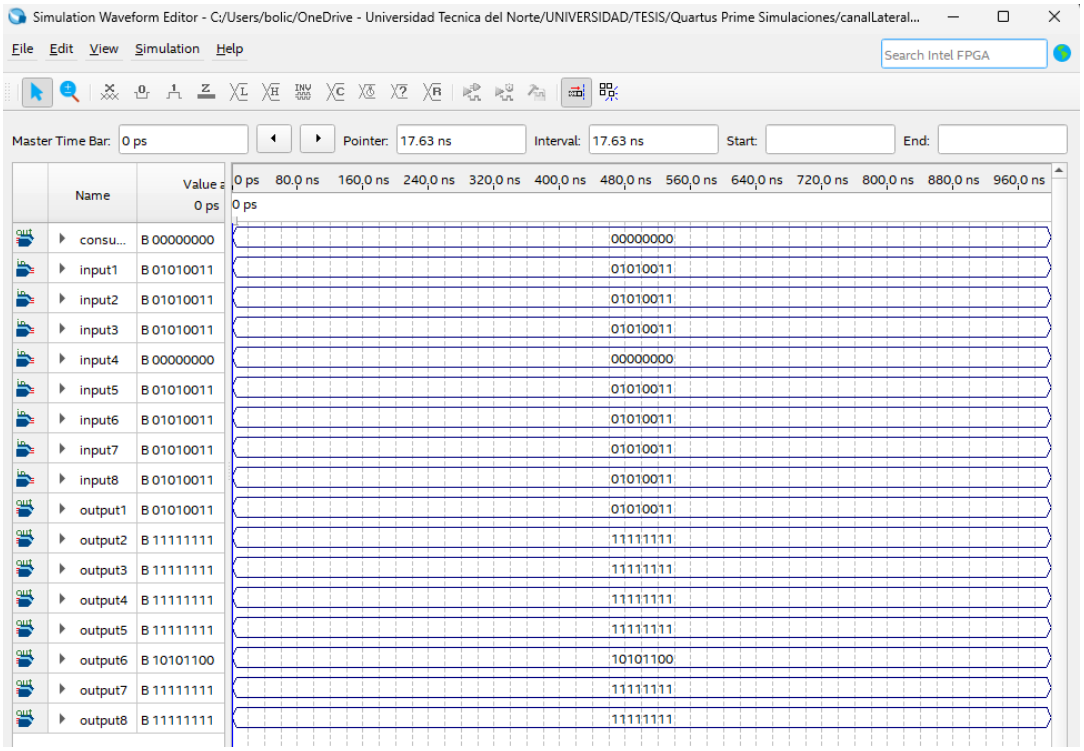
Código para la simulación del Ataque del Canal Lateral con 64 bits.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity canalLateral64 is
    Port ( input1 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input2 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input3 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input4 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
      input5 : in STD_LOGIC_VECTOR(7 downto 0);
           input6 : in STD_LOGIC_VECTOR(7 downto 0);
           input7 : in STD_LOGIC_VECTOR(7 downto 0);
           input8 : in STD_LOGIC_VECTOR(7 downto 0);
           output1 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output2 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output3 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output4 : out STD_LOGIC_VECTOR(7 downto 0);-- Vector de 8 bits para las salidas
           output5 : out STD_LOGIC_VECTOR(7 downto 0);
           output6 : out STD_LOGIC_VECTOR(7 downto 0);
           output7 : out STD_LOGIC_VECTOR(7 downto 0);
           output8 : out STD_LOGIC_VECTOR(7 downto 0);
           consumo_energia : out STD_LOGIC_VECTOR(7 downto 0)); -- Se agrega una señal de consumo de energía
end canalLateral64;

architecture Behavioral of canalLateral64 is
begin
    -- Procesamiento de las entradas utilizando compuertas lógicas
    output1 <= (input1 AND input2) OR (input3 AND input4);
    output2 <= NOT (input5 AND input6) OR (input7 AND input8);
    output3 <= NOT (input1 AND input3) OR (input5 AND input7);
    output4 <= NOT (input2 AND input4) OR (input6 AND input8);
    output5 <= NOT (input1 AND input8) OR (input2 AND input7);
    output6 <= NOT (input3 AND input6) OR (input4 AND input5);
   output7 <= NOT input1 OR input2 OR input3;
    output8 <= NOT input4 OR input5 OR input6 OR input7 OR input8;

    -- Simulación de un ataque de canal lateral (consumo de energía)
    process(input1, input2, input3, input4, input5, input6, input7, input8)
    begin
        if (input1 = "1" or input2 = "1") then
            consumo_energia <= "11111111";  -- Alto consumo de energía
        else
            consumo_energia <= "00000000";  -- Bajo consumo de energía

        end if;
    end process;
```

Nota. Código para la simulación con el ataque del canal lateral dentro de las pruebas de funcionamiento en 64 bits. Fuente: Autoría.

Figura 47.

Comportamiento Ataque canal Lateral 64 bits



Nota. Resultado del caso con el ataque del canal lateral, esto para ser mitigado en el siguiente apartado con 64 bits. Fuente: Autoría.

Tabla 26.

Valores de Entrada y Salida para el Ataque Canal Lateral 64 bits.

#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	01010011	01010011	01010011	✓
2	01010011	11111111	11111111	✓

3	01010011	11111111	11111111	✓
4	01010011	11111111	11111111	✓
5	01010011	11111111	11111111	✓
6	01010011	01010011	11111111	✗
7	01010011	11111111	11111111	✓
8	01010011	11111111	11111111	✓
		Total Bits Erróneos		4

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores en el ataque del canal lateral con 64 bits.

- ✓ **Comportamiento Ataque canal Lateral**

El comportamiento de un ataque de canal lateral en un dispositivo FPGA se refiere a la forma en que se aprovecha la información filtrada a través de canales no deseados, como la potencia consumida, las emisiones electromagnéticas o el tiempo de ejecución, para obtener información confidencial o realizar ataques en el sistema. El objetivo principal de un ataque de canal lateral es explotar las fugas de información involuntarias que se producen durante la ejecución del sistema. Estas fugas de información pueden revelar datos confidenciales, como claves criptográficas, patrones de acceso a memoria o información sobre algoritmos implementados.

En la simulación, se agregó la señal “consumo_energia” como una salida para representar la medición del consumo de energía. El proceso en la arquitectura simula un ataque de canal lateral basado en el consumo de energía, donde el consumo de energía se establece en un valor alto cuando input1 y input2 son '1', lo que podría indicar la presencia de cierta información sensible en las entradas. Esto es solo un ejemplo simplificado de un ataque de canal lateral y cómo se podría implementar en el código VHDL. En la práctica, los ataques de canal lateral pueden ser mucho más complejos y requieren medidas de mitigación adecuadas.

4.2.5 Pruebas con la Mitigación Sugerida

La mitigación de las vulnerabilidades se refiere a la implementación de medidas y controles para reducir el riesgo y minimizar el impacto de las vulnerabilidades identificadas. En este caso el objetivo es prevenir o limitar la explotación de las vulnerabilidades y fortalecer la seguridad del sistema.

4.2.5.1 Checksum para la mitigación de la inyección de fallas

Para este apartado se realiza las pruebas de funcionamiento con la medida de mitigación dentro de la inyección de fallas mediante el cual se toma la base del código donde se implementa la vulnerabilidad para el sistema propuesto.

Figura 48.

Código para la mitigación de la inyección de fallas 8 bits.

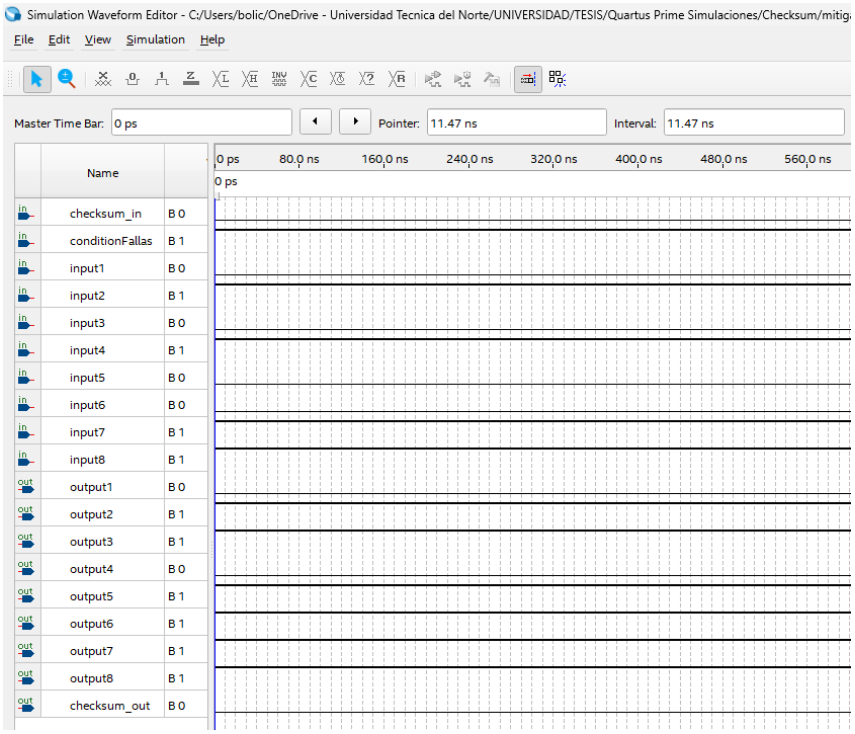
mitigacionChecksum.vhd | Compilation Report - mitigacionChecksum

```
5
6   entity mitigacionChecksum is
7       Port ( input1 : in STD_LOGIC;
8              input2 : in STD_LOGIC;
9              input3 : in STD_LOGIC;
10             input4 : in STD_LOGIC;
11             input5 : in STD_LOGIC;
12             input6 : in STD_LOGIC;
13             input7 : in STD_LOGIC;
14             input8 : in STD_LOGIC;
15             checksum_in : in STD_LOGIC;
16             conditionFallas : in STD_LOGIC;
17             output1 : out STD_LOGIC;
18             output2 : out STD_LOGIC;
19             output3 : out STD_LOGIC;
20             output4 : out STD_LOGIC;
21             output5 : out STD_LOGIC;
22             output6 : out STD_LOGIC;
23             output7 : out STD_LOGIC;
24             output8 : out STD_LOGIC;
25             checksum_out : out STD_LOGIC);
26  end mitigacionChecksum;
27
28  architecture Behavioral of mitigacionChecksum is
29      signal injected_input : STD_LOGIC := '0'; -- Inyección de falla en la entrada 1
30      signal calc_checksum : STD_LOGIC;
31      signal stored_checksum : STD_LOGIC := '0'; -- Valor almacenado del checksum
32  begin
33      -- Inyección de falla en la entrada 1 (input1)
34      process(conditionFallas)
35      begin
36          if conditionFallas = '0' then
37              injected_input <= not input1; -- Cambia el valor de la entrada 1
38          else
39              injected_input <= input1; -- Mantén el valor original de la entrada 1
40          end if;
41      end process;
42
43      -- Calcula el checksum
44      calc_checksum <= (injected_input XOR input2) XOR input3;
45      stored_checksum <= checksum_in when rising_edge(calc_checksum);
46
47      -- Procesamiento de las entradas utilizando compuertas lógicas
48      output1 <= (injected_input AND input2) OR (input3 AND input4);
49      output2 <= NOT (input5 AND input6) OR (input7 AND input8);
50      output3 <= NOT (injected_input AND input3) OR (input5 AND input7);
51      output4 <= NOT (input2 AND input4) OR (input6 AND input8);
52      output5 <= NOT (injected_input AND input8) OR (input2 AND input7);
53      output6 <= NOT (input3 AND input6) OR (input4 AND input5);
54      output7 <= NOT injected_input OR input2 OR input3;
55      output8 <= NOT input4 OR input5 OR input6 OR input7 OR input8;
56
57      -- Comparación del checksum calculado con el almacenado
58      checksum_out <= '1' when calc_checksum = stored_checksum else '0';
59  end Behavioral;
60
```

Nota. Código para la simulación con la mitigación de la inyección de fallas dentro de las pruebas de funcionamiento en 8 bits.

Figura 49.

Comportamiento mitigación de la vulnerabilidad de inyección de fallas para la salida en el software de simulación 8 bits.



Nota. Resultado de la mitigación en el caso de la inyección de fallas con 8 bits donde se puede observar la fidelidad con el caso ideal. Fuente: Autoría.

Tabla 27.

Comparación de los valores de entrada y salida en la mitigación con el caso ideal 8 bits.

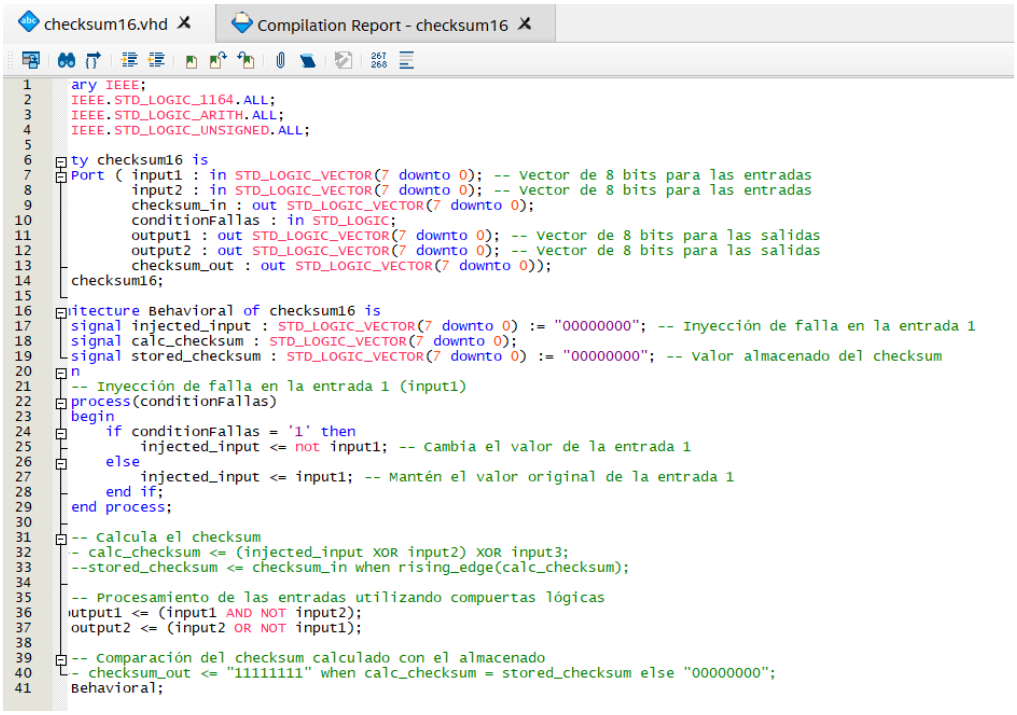
#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	0	0	0	✓
2	1	1	1	✓
3	0	1	1	✓

4	1	0	0	✓
5	0	1	1	✓
6	0	1	1	✓
7	1	1	1	✓
8	1	1	1	✓
		Total Errores		0

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores, se tiene el total de 0 valores erróneos comprobando el correcto funcionamiento de este apartado.

Figura 50.

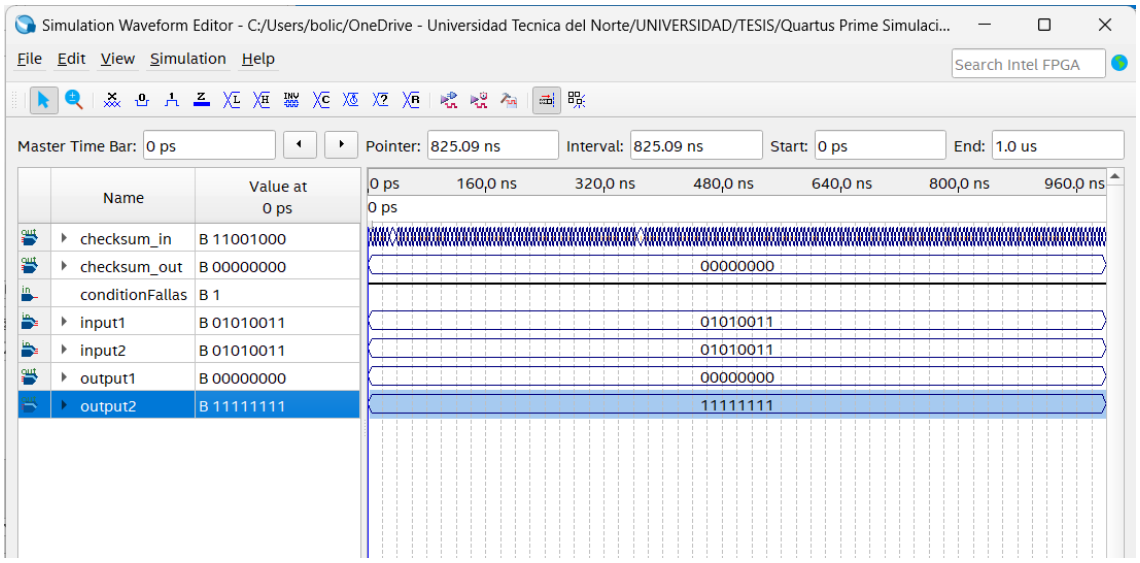
Código para la mitigación de la inyección de fallas 16 bits.



Nota. Código para la simulación con la mitigación de la inyección de fallas dentro de las pruebas de funcionamiento en 16 bits.

Figura 51.

Comportamiento mitigación de la vulnerabilidad de inyección de fallas para la salida en el software de simulación 16 bits.



Nota. Resultado de la mitigación en el caso de la inyección de fallas con 16 bits donde se puede observar la fidelidad con el caso ideal. Fuente: Autoría.

Tabla 28.

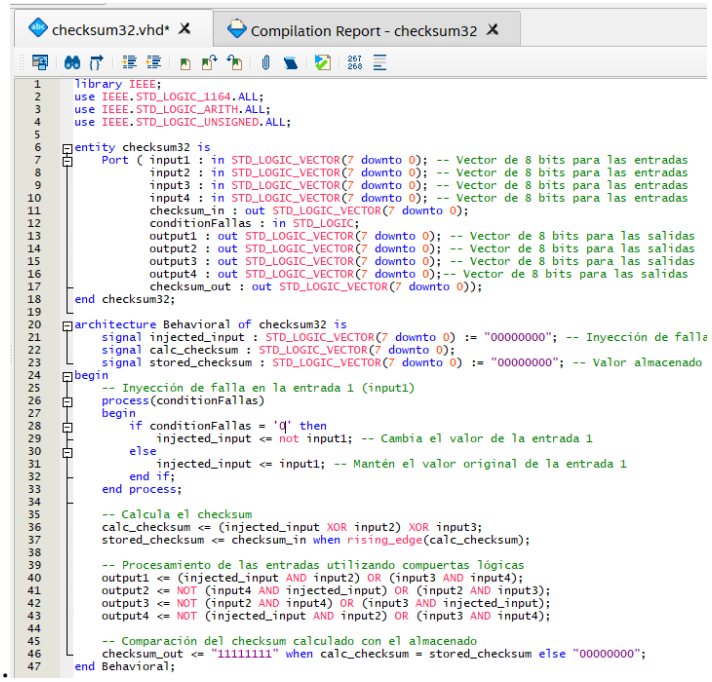
Comparación de los valores de entrada y salida en la mitigación con el caso ideal 16 bits.

#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	01010011	00000000	00000000	X
2	01010011	11111111	11111111	X
		Total Bits Erróneos		0

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores, se tiene el total de 0 valores erróneos comprobando el correcto funcionamiento de este apartado.

Figura 52.

Código para la mitigación de la inyección de fallas 32 bits

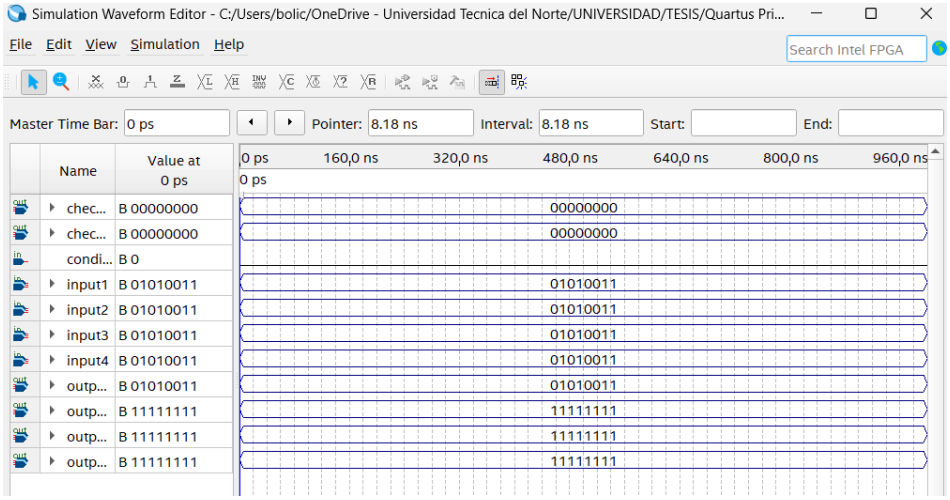


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity checksum32 is
    Port ( input1 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input2 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input3 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           input4 : in STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las entradas
           checksum_in : out STD_LOGIC_VECTOR(7 downto 0);
           conditionFallas : in STD_LOGIC;
           output1 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output2 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output3 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para las salidas
           output4 : out STD_LOGIC_VECTOR(7 downto 0);-- Vector de 8 bits para las salidas
           checksum_out : out STD_LOGIC_VECTOR(7 downto 0));
end checksum32;

architecture Behavioral of checksum32 is
    signal injected_input : STD_LOGIC_VECTOR(7 downto 0) := "00000000"; -- Inyección de falla
    signal calc_checksum : STD_LOGIC_VECTOR(7 downto 0);
    signal stored_checksum : STD_LOGIC_VECTOR(7 downto 0) := "00000000"; -- Valor almacenado
begin
    -- Inyección de falla en la entrada 1 (input1)
    process(conditionFallas)
    begin
        if conditionFallas = '0' then
            injected_input <= not input1; -- Cambia el valor de la entrada 1
        else
            injected_input <= input1; -- Mantén el valor original de la entrada 1
        end if;
    end process;

    -- Calcula el checksum
    calc_checksum <= (injected_input XOR input2) XOR input3;
    stored_checksum <= checksum_in when rising_edge(calc_checksum);

    -- Procesamiento de las entradas utilizando compuertas lógicas
    output1 <= (injected_input AND input2) OR (input3 AND input4);
    output2 <= NOT (input4 AND injected_input) OR (input2 AND input3);
    output3 <= NOT (input2 AND input4) OR (input3 AND injected_input);
    output4 <= NOT (injected_input AND input2) OR (input3 AND input4);

    -- Comparación del checksum calculado con el almacenado
    checksum_out <= "11111111" when calc_checksum = stored_checksum else "00000000";
end Behavioral;
```

Nota. Código para la simulación con la mitigación de la inyección de fallas dentro de las pruebas de funcionamiento en 32 bits.

Figura 53.

Comportamiento mitigación de la vulnerabilidad de inyección de fallas para la salida en el software de simulación 32 bits.



Nota. Resultado de la mitigación en el caso de la inyección de fallas con 32 bits donde se puede observar la fidelidad con el caso ideal. Fuente: Autoría.

Tabla 29.

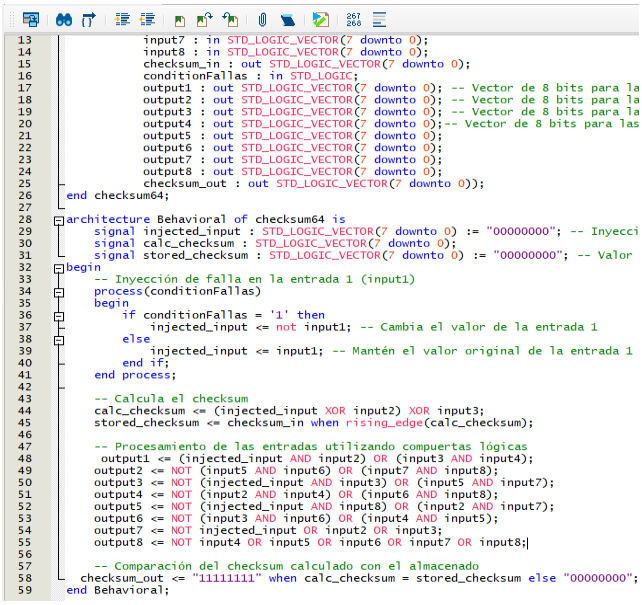
Comparación de los valores de entrada y salida en la mitigación con el caso ideal 32 bits.

#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	01010011	01010011	01010011	✓
2	01010011	11111111	11111111	✓
3	01010011	11111111	11111111	✓
4	01010011	11111111	11111111	✓
		Total Bits Erróneos		0

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores, se tiene el total de 0 valores erróneos comprobando el correcto funcionamiento de este apartado.

Figura 54.

Código para la mitigación de la inyección de fallas 64 bits.

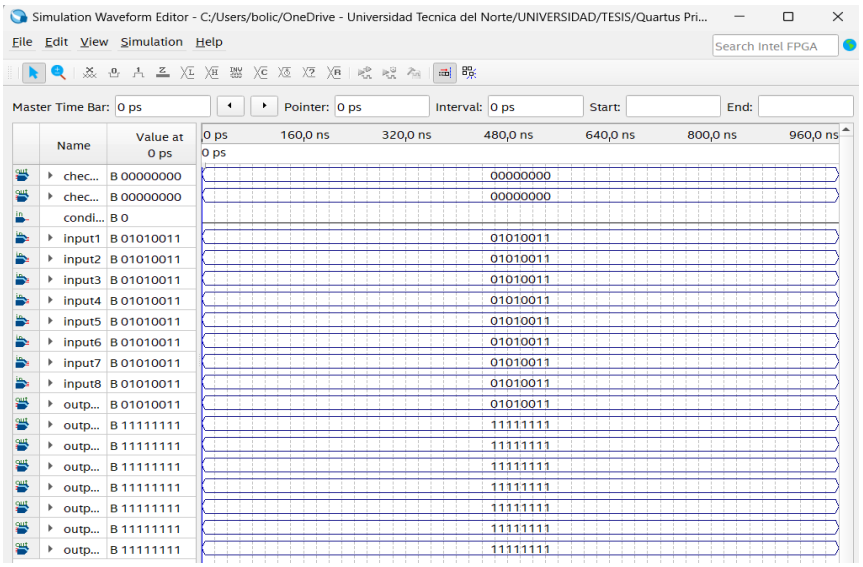


```
            input7 : in STD_LOGIC_VECTOR(7 downto 0);
            input8 : in STD_LOGIC_VECTOR(7 downto 0);
            checksum_in : out STD_LOGIC_VECTOR(7 downto 0);
            conditionFallas : in STD_LOGIC;
            output1 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para la
            output2 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para la
            output3 : out STD_LOGIC_VECTOR(7 downto 0); -- Vector de 8 bits para la
            output4 : out STD_LOGIC_VECTOR(7 downto 0);-- Vector de 8 bits para las
            output5 : out STD_LOGIC_VECTOR(7 downto 0);
            output6 : out STD_LOGIC_VECTOR(7 downto 0);
            output7 : out STD_LOGIC_VECTOR(7 downto 0);
            output8 : out STD_LOGIC_VECTOR(7 downto 0);
            checksum_out : out STD_LOGIC_VECTOR(7 downto 0));
end checksum64;

architecture Behavioral of checksum64 is
    signal injected_input : STD_LOGIC_VECTOR(7 downto 0) := "00000000"; -- Inyecci
    signal calc_checksum : STD_LOGIC_VECTOR(7 downto 0);
    signal stored_checksum : STD_LOGIC_VECTOR(7 downto 0) := "00000000"; -- Valor
begin
    -- Inyección de falla en la entrada 1 (input1)
    process(conditionFallas)
    begin
        if conditionFallas = '1' then
            injected_input <= not input1; -- Cambia el valor de la entrada 1
        else
            injected_input <= input1; -- Mantén el valor original de la entrada 1
        end if;
    end process;

    -- Calcula el checksum
    calc_checksum <= (injected_input XOR input2) XOR input3;
    stored_checksum <= checksum_in when rising_edge(calc_checksum);

    -- Procesamiento de las entradas utilizando compuertas lógicas
     output1 <= (injected_input AND input2) OR (input3 AND input4);
    output2 <= NOT (input5 AND input6) OR (input7 AND input8);
    output3 <= NOT (injected_input AND input3) OR (input5 AND input7);
    output4 <= NOT (input2 AND input4) OR (input6 AND input8);
    output5 <= NOT (injected_input AND input8) OR (input2 AND input7);
    output6 <= NOT (input3 AND input6) OR (input4 AND input5);
    output7 <= NOT injected_input OR input2 OR input3;
    output8 <= NOT input4 OR input5 OR input6 OR input7 OR input8;

    -- Comparación del checksum calculado con el almacenado
  checksum_out <= "11111111" when calc_checksum = stored_checksum else "00000000";
end Behavioral;
```

Nota. Resultado de la mitigación en el caso de la inyección de fallas con 64 bits donde se puede observar la fidelidad con el caso ideal. Fuente: Autoría.

Figura 55.

Comportamiento mitigación de la vulnerabilidad de inyección de fallas para la salida en el software de simulación 64 bits.



Nota. Código para la simulación con la mitigación de la inyección de fallas dentro de las pruebas de funcionamiento en 64 bits. Fuente: Autoría.

Tabla 30.

Comparación de los valores de entrada y salida en la mitigación con el caso ideal 64 bits.

#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	01010011	01010011	01010011	✓
2	01010011	11111111	11111111	✓
3	01010011	11111111	11111111	✓
4	01010011	11111111	11111111	✓
5	01010011	11111111	11111111	✓
6	01010011	11111111	11111111	✓
7	01010011	11111111	11111111	✓
8	01010011	11111111	11111111	✓
		Total Bits Erróneos		0

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores, se tiene el total de 0 valores erróneos comprobando el correcto funcionamiento de este apartado.

En las Figuras se observa cómo se agregan una señal checksum_in como entrada y checksum_out como salida para el checksum. El cálculo del checksum se realiza utilizando una operación XOR de algunas de las entradas. Luego, el checksum calculado se compara con el checksum almacenado y se establece la salida checksum_out en '1' si coinciden y en '0' si no coinciden. Esto puede ayudar a detectar posibles inyecciones de fallas en las entradas.

Se comprueba el funcionamiento de la mitigación la cual utiliza la suma de todos los valores de datos en un conjunto de datos y la adición de ese valor de suma como un bit adicional. Cuando los datos se transmiten o almacenan, se calcula un checksum antes y después de la transmisión o almacenamiento. Luego, el receptor o el sistema que almacena los datos pueden calcular el checksum nuevamente y compararlo con el valor de checksum original.

- ✓ **Comportamiento Checksum**

El checksum es utilizado como una medida complementaria en la detección de inyección de fallas, pero su efectividad dependerá de la naturaleza de la inyección y la implementación específica del sistema. La mitigación efectiva de la inyección de fallas requiere el uso de múltiples técnicas y un enfoque integral para garantizar la seguridad y la confiabilidad del sistema.

- **Cálculo del checksum:** Para calcular el checksum, se aplica una función matemática o algoritmo a los datos originales. Este algoritmo suele ser una operación aritmética simple, como una suma o una operación XOR (OR exclusiva) de los bytes de los datos. El resultado del cálculo es el valor de comprobación o checksum.
- **Transmisión o almacenamiento del checksum:** El valor de checksum se transmite junto con los datos originales o se almacena en un lugar específico junto con los datos.
- **Recepción o recuperación de los datos:** Cuando los datos se reciben o se recuperan, se aplica nuevamente el mismo algoritmo de checksum a los datos recibidos para calcular un nuevo valor de checksum.

- **Verificación del checksum:** El nuevo valor de checksum se compara con el valor de checksum transmitido o almacenado. Si ambos valores coinciden, se considera que los datos son íntegros y no han sufrido alteraciones. Si los valores no coinciden, se detecta una alteración o corrupción en los datos.

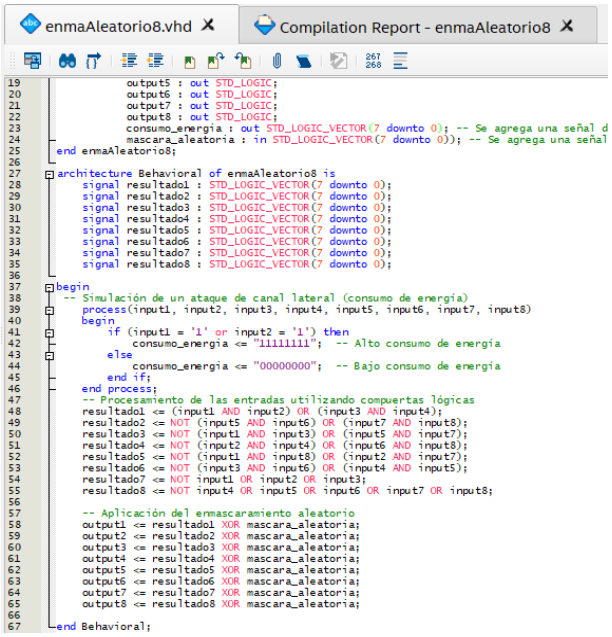
En este caso con la simulación en funcionamiento se verifica los valores iguales que en el caso ideal, a pesar de que se tiene también la vulnerabilidad dentro del código presentado en las Figuras.

4.2.5.2 Enmascaramiento Aleatorio Mitigación para el Ataque del Canal Lateral

El enmascaramiento aleatorio es una técnica crucial para mitigar los ataques de canal lateral en sistemas y dispositivos críticos. Este enfoque de seguridad se basa en la idea de introducir ruido adicional en las señales emitidas por el dispositivo durante su funcionamiento normal. Este ruido hace que sea extremadamente difícil para un atacante discernir información confidencial a partir de las señales físicas o eléctricas observadas, para esta ocasión se utilizó para simulación un ataque del canal lateral en base al consumo.

Figura 56.

Código para la mitigación del Ataque del Canal Lateral 8 bits.

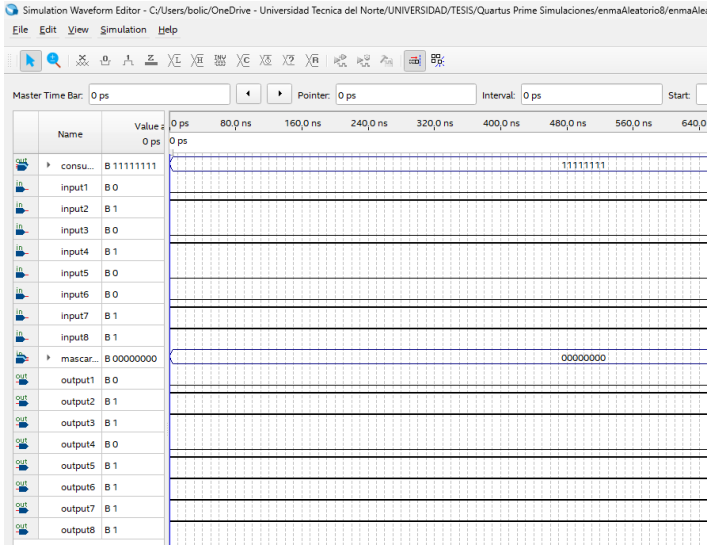


```
enmaAleatorio8.vhd  |  Compilation Report - enmaAleatorio8

19          output5 : out STD_LOGIC;
20          output6 : out STD_LOGIC;
21          output7 : out STD_LOGIC;
22          output8 : out STD_LOGIC;
23          consumo_energia : out STD_LOGIC_VECTOR(7 downto 0); -- Se agrega una señal d
24          mascara_aleatoria : in STD_LOGIC_VECTOR(7 downto 0)); -- Se agrega una señal
25  end enmaAleatorio8;
26
27  architecture Behavioral of enmaAleatorio8 is
28      signal resultado1 : STD_LOGIC_VECTOR(7 downto 0);
29      signal resultado2 : STD_LOGIC_VECTOR(7 downto 0);
30      signal resultado3 : STD_LOGIC_VECTOR(7 downto 0);
31      signal resultado4 : STD_LOGIC_VECTOR(7 downto 0);
32      signal resultado5 : STD_LOGIC_VECTOR(7 downto 0);
33      signal resultado6 : STD_LOGIC_VECTOR(7 downto 0);
34      signal resultado7 : STD_LOGIC_VECTOR(7 downto 0);
35      signal resultado8 : STD_LOGIC_VECTOR(7 downto 0);
36
37  begin
38      -- Simulación de un ataque de canal lateral (consumo de energía)
39      process(input1, input2, input3, input4, input5, input6, input7, input8)
40      begin
41          if (input1 = '1' or input2 = '1') then
42              consumo_energia <= "11111111";  -- Alto consumo de energía
43          else
44              consumo_energia <= "00000000";  -- Bajo consumo de energía
45          end if;
46      end process;
47      -- Procesamiento de las entradas utilizando compuertas lógicas
48      resultado1 <= (input1 AND input2) OR (input3 AND input4);
49      resultado2 <= NOT (input5 AND input6) OR (input7 AND input8);
50      resultado3 <= NOT (input1 AND input3) OR (input5 AND input7);
51      resultado4 <= NOT (input2 AND input4) OR (input6 AND input8);
52      resultado5 <= NOT (input1 AND input8) OR (input2 AND input7);
53      resultado6 <= NOT (input3 AND input6) OR (input4 AND input5);
54      resultado7 <= NOT input1 OR input2 OR input3;
55      resultado8 <= NOT input4 OR input5 OR input6 OR input7 OR input8;
56
57      -- Aplicación del enmascaramiento aleatorio
58      output1 <= resultado1 XOR mascara_aleatoria;
59      output2 <= resultado2 XOR mascara_aleatoria;
60      output3 <= resultado3 XOR mascara_aleatoria;
61      output4 <= resultado4 XOR mascara_aleatoria;
62      output5 <= resultado5 XOR mascara_aleatoria;
63      output6 <= resultado6 XOR mascara_aleatoria;
64      output7 <= resultado7 XOR mascara_aleatoria;
65      output8 <= resultado8 XOR mascara_aleatoria;
66
67  end Behavioral;
```

Nota. Código para la simulación con la mitigación de la inyección de fallas dentro de las pruebas de funcionamiento en 8 bits.

Figura 57.

Comportamiento mitigación de la vulnerabilidad del ataque del canal lateral para la salida en el software de simulación 8 bits.



Nota. Resultado de la mitigación en el caso del ataque del canal lateral con 8 bits donde se puede observar la fidelidad con el caso ideal. Fuente: Autoría.

Tabla 31.

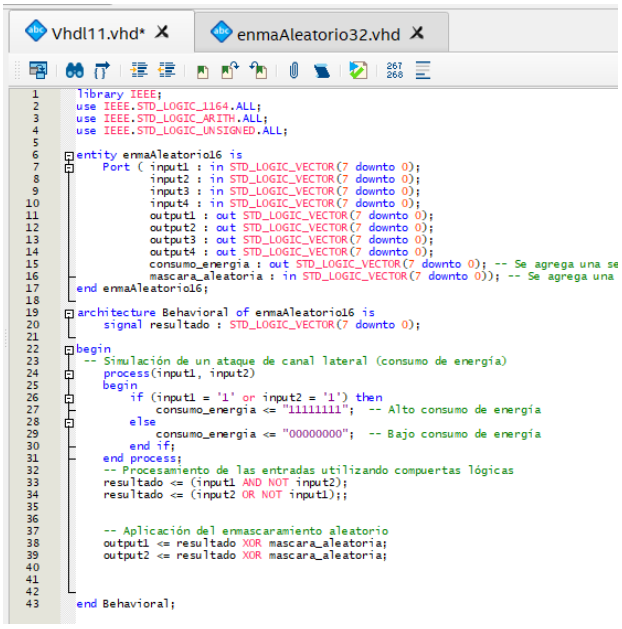
Comparación de los valores de entrada y salida en la mitigación con el caso ideal 8 bits.

#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	0	0	0	✓
2	1	1	1	✓
3	0	1	1	✓
4	1	0	0	✓
5	0	1	1	✓
6	0	1	1	✓
7	1	1	1	✓
8	1	1	1	✓
		Total Errores		0

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores, se tiene el total de 0 valores erróneos comprobando el correcto funcionamiento de este apartado.

Figura 58.

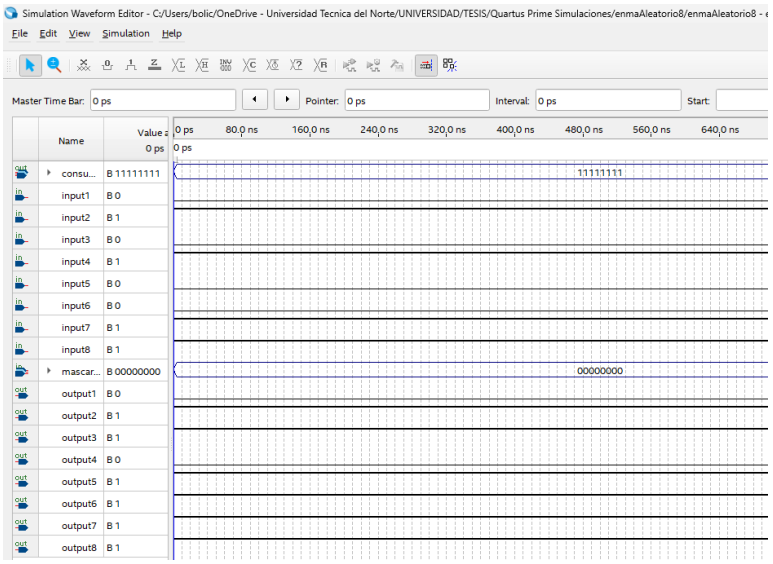
Código para la mitigación del Ataque del Canal Lateral 16 bits.



Nota. Código para la simulación con la mitigación de la inyección de fallas dentro de las pruebas de funcionamiento en 16 bits.

Figura 59.

Comportamiento mitigación de la vulnerabilidad del ataque del canal lateral para la salida en el software de simulación 16 bits.



Nota. Resultado de la mitigación en el caso del ataque del canal lateral con 16 bits donde se puede observar la fidelidad con el caso ideal. Fuente: Autoría.

Tabla 32.

Comparación de los valores de entrada y salida en la mitigación con el caso ideal 16 bits.

#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	01010011	00000000	00000000	X
2	01010011	11111111	11111111	X
		Total Bits Erróneos		0

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores, se tiene el total de 0 valores erróneos comprobando el correcto funcionamiento de este apartado.

Figura 60.

Código para la mitigación del Ataque del Canal Lateral 32 bits.

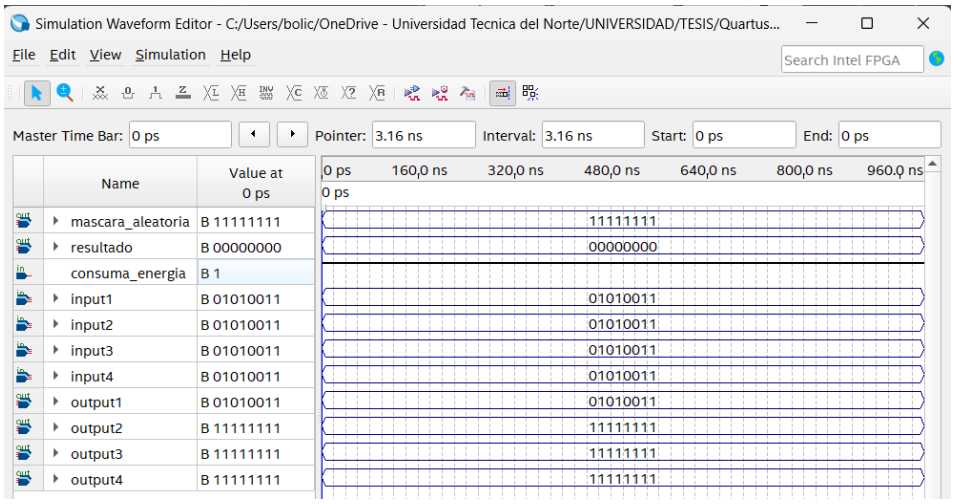
enmaAleatorio32.vhd | Compilation Report - enmaAleatorio32

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity enmaAleatorio32 is
    Port ( input1 : in STD_LOGIC_VECTOR(7 downto 0);
           input2 : in STD_LOGIC_VECTOR(7 downto 0);
           input3 : in STD_LOGIC_VECTOR(7 downto 0);
           input4 : in STD_LOGIC_VECTOR(7 downto 0);
           output1 : out STD_LOGIC_VECTOR(7 downto 0);
           output2 : out STD_LOGIC_VECTOR(7 downto 0);
           output3 : out STD_LOGIC_VECTOR(7 downto 0);
           output4 : out STD_LOGIC_VECTOR(7 downto 0);
           consumo_energia : out STD_LOGIC_VECTOR(7 downto 0); -- Se agrega u
           mascara_aleatoria : in STD_LOGIC_VECTOR(7 downto 0)); -- Se agrega
end enmaAleatorio32;

architecture Behavioral of enmaAleatorio32 is
    signal resultado : STD_LOGIC_VECTOR(7 downto 0);

begin
  -- Simulación de un ataque de canal lateral (consumo de energía)
    process(input1, input2, input3, input4)
    begin
        if (input1 = '1' or input2 = '1') then
            consumo_energia <= "11111111";  -- Alto consumo de energía
        else
            consumo_energia <= "00000000";  -- Bajo consumo de energía
        end if;
    end process;
    -- Procesamiento de las entradas utilizando compuertas lógicas
    resultado <= (input1 AND input2) OR (input3 AND input4);
    resultado <= NOT (input4 AND input1) OR (input2 AND input3);
    resultado <= NOT (input2 AND input4) OR (input3 AND input1);
    resultado <= NOT (input1 AND input2) OR (input3 AND input4);

    -- Aplicación del enmascaramiento aleatorio
    output1 <= resultado XOR mascara_aleatoria;
    output2 <= resultado XOR mascara_aleatoria;
    output3 <= resultado XOR mascara_aleatoria;
    output4 <= resultado XOR mascara_aleatoria;
```

Nota. Código para la simulación con la mitigación del Ataque del Canal Lateral dentro de las pruebas de funcionamiento en 32 bits.

Figura 61.

Comportamiento mitigación de la vulnerabilidad del ataque del canal lateral para la salida en el software de simulación 32 bits.



Nota. Resultado de la mitigación en el caso del ataque del canal lateral con 32 bits donde se puede observar la fidelidad con el caso ideal. Fuente: Autoría.

Tabla 33.

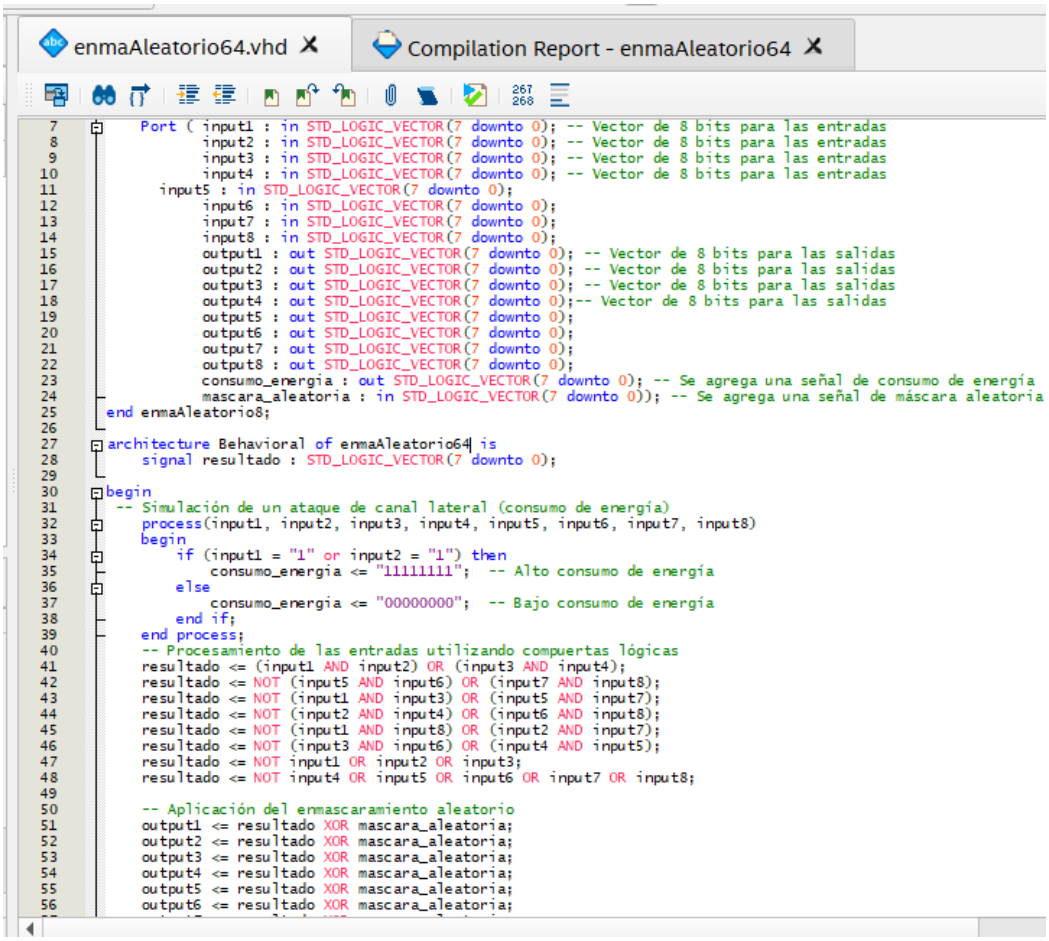
Comparación de los valores de entrada y salida en la mitigación con el caso ideal 32 bits.

#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	01010011	01010011	01010011	✓
2	01010011	11111111	11111111	✓
3	01010011	11111111	11111111	✓
4	01010011	11111111	11111111	✓
		Total Bits Erróneos		0

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores, se tiene el total de 0 valores erróneos comprobando el correcto funcionamiento de este apartado.

Figura 62.

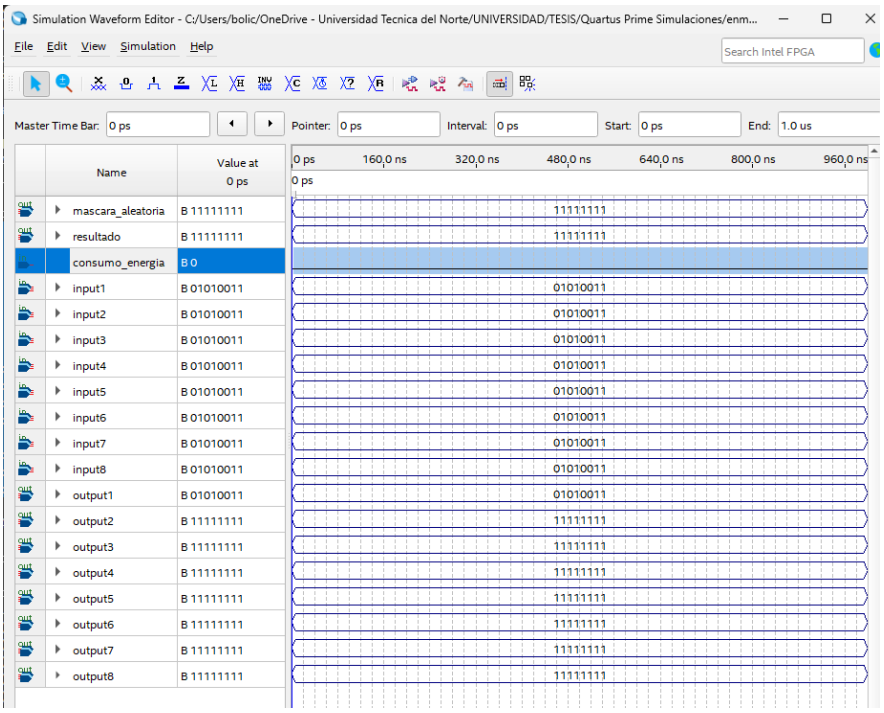
Código para la mitigación del Ataque del Canal Lateral 64 bits.



Nota. Código para la simulación con la mitigación del Ataque del Canal Lateral dentro de las pruebas de funcionamiento en 64 bits.

Figura 63.

Comportamiento mitigación de la vulnerabilidad del ataque del canal Lateral para la salida en el software de simulación 64 bits.



Nota. Resultado de la mitigación en el caso del ataque del canal lateral con 64 bits donde se puede observar la fidelidad con el caso ideal. Fuente: Autoría.

Tabla 34.

Comparación de los valores de entrada y salida en la mitigación con el caso ideal 64 bits.

#	Valor Entrada	Valor Salida	Valor Ideal	Comparación
1	01010011	01010011	01010011	✓
2	01010011	11111111	11111111	✓
3	01010011	11111111	11111111	✓
4	01010011	11111111	11111111	✓

5	01010011	11111111	11111111	✓
6	01010011	11111111	11111111	✓
7	01010011	11111111	11111111	✓
8	01010011	11111111	11111111	✓
		Total Bits Erróneos		0

Nota. Especificación de los valores de salida con la comparación y la enumeración de los errores, se tiene el total de 0 valores erróneos comprobando el correcto funcionamiento de este apartado.

En esta instancia, mediante la simulación en ejecución, se confirma que los valores coinciden con los del escenario ideal, incluso cuando el código incluye la vulnerabilidad como se muestra en las Figuras. A continuación, se detallan los valores tal como se presentan en la propuesta de mitigación.

Es importante destacar que la implementación adecuada del enmascaramiento aleatorio y la elección de las técnicas de enmascaramiento adecuadas son aspectos fundamentales para su éxito. Además, el enmascaramiento aleatorio debe ir acompañado de otras medidas de seguridad, como el control de acceso físico y la diversificación de claves, para lograr una protección integral contra los ataques de canal lateral.

- ✓ **Comportamiento del Enmascaramiento Aleatorio**

El enmascaramiento aleatorio es una técnica de mitigación utilizada para proteger sistemas y dispositivos electrónicos, incluidos los dispositivos FPGA, contra ataques de canal lateral y ataques de inyección de fallas. El objetivo principal de esta técnica es dificultar el acceso a información confidencial o sensibles por parte de posibles atacantes.

- **Selección aleatoria:** Se elige aleatoriamente un valor secreto o clave de enmascaramiento para cada operación crítica que involucre datos sensibles o información confidencial.
- **Enmascaramiento de datos:** Antes de realizar una operación crítica, los datos se combinan con el valor de enmascaramiento aleatorio utilizando una operación aritmética, como una suma o una operación XOR. Esto enmascara los datos originales y produce una versión enmascarada de los datos.
- **Ejecución de la operación:** La operación crítica, como una operación aritmética o criptográfica, se realiza utilizando los datos enmascarados en lugar de los datos originales.
- **Desenmascaramiento:** Después de completar la operación crítica, los resultados se desenmascaran al aplicar una operación inversa utilizando el mismo valor de enmascaramiento aleatorio. Esto permite obtener los resultados correctos de la operación.

4.3 Evaluación del Grado de Mitigación, Mediante los FPGA.

Este proceso de evaluación es de vital importancia, ya que proporciona una visión crítica sobre cuán efectivas son las medidas de seguridad adoptadas en la protección contra ataques, como la inyección de fallas y el ataque del canal lateral. A través de rigurosas pruebas y evaluaciones, se determinará en qué medida se cumplen los objetivos de seguridad y se identificarán posibles áreas de mejora. Este apartado se centrará en la metodología utilizada, los resultados obtenidos y las conclusiones derivadas de la evaluación de mitigación, brindando una visión integral de la robustez y confiabilidad del sistema implementado en los FPGA.

A continuación, se proporciona las métricas resumidas acerca de lo realizado en los casos presentados:

Tabla 35.

Tasa de Error presente en la inyección de fallas

Caso	Bits Enviados	Bits Errados	Tasa de Error
8 bits	5	3	0,6
16 bits	8	8	1
32 bits	28	4	0,125
64 bits	60	4	0,0625

Nota. Estos datos están dados por las simulaciones realizadas anteriormente en comparación del caso ideal.

Tabla 36.

Tasa de Error presente en la Ataque del Canal Lateral

Caso	Bits Enviados	Bits Errados	Tasa de Error
8 bits	6	2	0,33
16 bits	12	4	0,33
32 bits	28	4	0,125
64 bits	60	4	0,0625

Nota. Estos datos están dados por las simulaciones realizadas anteriormente en comparación del caso ideal.

Estos datos pueden cambiar a medida que tratamiento se haga a los datos proporcionados lo cual también influirá en este apartado, finalmente para cumplir con las características dadas en para el diseño del sistema se logra comprobar el cumplimiento de estas.

Tabla 37.

Características presentes en las mitigaciones presentes.

Mitigación	Característica					
	Efectividad	**Compatibilidad**	**Robustez**	**Bajo Costo**	**Estándares Seguridad**	**Latencia**
Checksum	✓	✓	✓	✓	✓	✓
Enmascaramiento Aleatorio	✓	✓	✓	✓	✓	✓

Nota. Características principales para las mitigaciones aplicadas en los diferentes casos establecidos en la sección anterior

4.4 Discusión

En García & Alberto (2019) destaca la relevancia de la gestión de riesgos, dado el creciente impacto que estos sistemas tienen en nuestra vida cotidiana. La constante evolución del Internet de las Cosas (IoT) ha impulsado a las organizaciones a implementar medidas destinadas a mitigar el potencial impacto de problemas relacionados con la ciberseguridad y la privacidad de los datos del usuario, los cuales podrían afectar la continuidad de las operaciones comerciales.

De igual manera en Calva et al. (2021), determina que es crucial comprender el concepto de amenaza, que hace referencia a la existencia de elementos que pueden aprovechar las vulnerabilidades presentes en sistemas de información, personas o procesos. Al diseñar sistemas IoT, es fundamental considerar aspectos de seguridad que aborden estos factores, incluyendo las amenazas, vulnerabilidades, posibles ataques y compromisos.

En Barón Chacón (2006) menciona que las FPGA se caracterizan por su alta densidad de puertas lógicas, rendimiento elevado, capacidad para configurar un gran número de entradas y salidas según las necesidades del usuario, una estructura de interconexión altamente flexible y un entorno de diseño que se asemeja al de una matriz de puertas. Estas características hacen que las FPGA sean una opción atractiva para su implementación en sistemas autónomos de Internet de las Cosas, donde pueden desempeñar un papel esencial en la capa física para cumplir con los estándares de seguridad exigidos en estos sistemas.

Finalmente, el enfoque que se quiere dar en el trabajo de titulación fue la evaluación de dispositivos FPGA, ya que permite comprender el impacto de las medidas de seguridad que pueden aplicarse a nivel de capa física. Además, facilita la comparación de estos dispositivos con otros en el contexto del nivel específico que se ha establecido. Esta evaluación resulta fundamental para determinar la idoneidad y el rendimiento de los dispositivos FPGA en el diseño de sistemas IoT seguros.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

- Las vulnerabilidades en los dispositivos FPGA, como los ataques de canal lateral, la inyección de fallas, la interceptación y manipulación de comunicaciones, las debilidades en el firmware y el escalado de privilegios, representan riesgos significativos que deben ser evaluados y mitigados adecuadamente.
- Se logró una identificación exhaustiva de las vulnerabilidades presentes en la capa física de sistemas autónomos IoT. El análisis detallado de las técnicas existentes proporcionó una comprensión profunda de los riesgos potenciales, permitiendo una respuesta proactiva y específica a cada amenaza identificada.
- Los requerimientos específicos de seguridad para las FPGA se definieron con precisión, considerando las particularidades de los entornos IoT. Estos criterios proporcionarán una guía clara para el diseño e implementación de medidas de seguridad, asegurando la adaptabilidad y eficacia de los dispositivos FPGA en este contexto dinámico.
- La evaluación de la mitigación de posibles ataques en sistemas autónomos IoT, mediante la implementación de dispositivos FPGA, demostró ser efectiva. Las técnicas de enmascaramiento y desordenamiento adoptadas no solo fortalecieron la seguridad de la capa física, sino que también ofrecieron una resistencia dinámica a las amenazas emergentes.

- Las FPGA se destacan por su flexibilidad y capacidad para adaptarse a los requisitos de hardware en sistemas IoT. Pueden desempeñar un papel clave en la capa física para garantizar la seguridad y cumplir con los estándares de estos sistemas.

Recomendaciones

- Es recomendable que se realicen evaluaciones de seguridad periódicas y se sigan las buenas prácticas de diseño seguro para mitigar las vulnerabilidades en los dispositivos FPGA utilizados en sistemas IoT. Además, es importante estar al tanto de las actualizaciones de seguridad proporcionadas por los fabricantes y seguir las recomendaciones de seguridad establecidas por los organismos reguladores y estándares de la industria.
- Implementar controles de acceso y autenticación sólidos en los dispositivos FPGA y en los sistemas IoT en general. Esto incluye el uso de contraseñas seguras, autenticación de múltiples factores y la asignación adecuada de permisos y privilegios.
- Realizar pruebas de estrés y pruebas de resistencia para evaluar la capacidad de los dispositivos FPGA y los sistemas IoT de manejar cargas de trabajo intensivas y situaciones adversas. Esto ayudará a identificar posibles puntos débiles y áreas de mejora en términos de seguridad y rendimiento.
- Establecer un proceso de gestión de vulnerabilidades efectivo que incluya la monitorización constante, la aplicación oportuna de parches de seguridad y la comunicación proactiva con los proveedores de dispositivos FPGA para estar al tanto de las actualizaciones de seguridad disponibles.

- Al simular en Quartus Prime se debe tener en cuenta la validación de todos los datos necesarios sea los pines, las librerías, la placa y el motor de ejecución ya que este simulador incluye una amplia gama de funciones, como un editor gráfico y un motor de síntesis.

BIBLIOGRAFÍA

Acar, B., & Ors, B. (2017). Hardware / Software Co-Design of a Lightweight Crypto Algorithm BORON on an FPGA. *2017 10th International Conference on Electrical and Electronics Engineering (ELECO), IEEE*, 1-5. https://doi.org/10.1126/science.1127119

Agarwal, T. (2014, septiembre 18). *Know about FPGA Architecture and thier Applications*. ElProCus - Electronic Projects for Engineering Students. https://www.elprocus.com/fpga-architecture-and-applications/

Aisawa, W. A. A. (2020). *Técnicas para identificação de funções de bibliotecas em binários vinculados estaticamente*.

Alaminos Benéitez, V. (2012). *Inyección de errores sobre FPGAS tipo Virtex-5*.

Aldaya, A. C., & Sarmiento, A. J. C. (2013). Diseño e integración de algoritmos criptográficos en sistemas empotrados sobre FPGA. *Revista Ingeniería Electrónica, Automática y Comunicaciones ISSN: 1815-5928*, *34*(3), 41-51.

BARBOSA, G. B. N. (2017). *Sistema de segurança para IoT baseado em agrupamento de smart cards gerenciados por FPGA*. Universidade Federal de Pernambuco.

Barón Chacón, G. D. (2006). *Diseño y construcción de un sistema de seguridad para un recinto cerrado implementando FPGA*. https://repository.unab.edu.co/handle/20.500.12749/1586

Barrios Alfaro, Y. (2017). *Análisis y diseño de aceleradores hardware sobre SoC basados en FPGA orientados a aplicaciones de seguridad en Internet de las cosas*. https://accedacris.ulpgc.es/jspui/handle/10553/77432

Benso, A., & Prinetto, P. (Eds.). (2004). *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation* (Vol. 23). Kluwer Academic Publishers. https://doi.org/10.1007/b105828

Bonilla-Fabela, I., Tavizon-Salazar, A., Morales-Escobar, M., Guajardo-Muñoz, L. T., & Laines-Alamina, C. I. (2016). IoT, el internet de las cosas y la innovación de sus aplicaciones. *Vinculatégica efan*, *2*(1), 2313-2340.

Calva, J. J. C., Rojas, D. L. H., Román, R. F. M., & García, C. D. R. (2021). Seguridad IoT: Principales amenazas en una taxonomía de activos. *HAMUT'AY*, *7*(3), Article 3. https://doi.org/10.21503/hamu.v7i3.2192

Camacho Olarte, J. F. (2020). *Uso de herramientas libres para diseñar un sistema de monitoreo de variables físicas de bajo costo basado en sistemas embebidos*.

Castellanos Hernández, J. A., Sandoval Ruiz, C. E., & Azpúrua Auyanet, M. A. (2014). Implementación sobre FPGA de un Algoritmo LMS para un arreglo de antenas inteligentes. *Revista Técnica de la Facultad de Ingeniería Universidad del Zulia*, *37*(3), 270-278.

Cayssials, R. (2014). *Sistemas embebidos en FPGA.* Alpha Editorial.

Cerquera, M. R. P. (2017). Internet de las cosas, la próxima evolución de internet, una oportunidad para mejorar la competitividad de Colombia en la región. *Encuentro Internacional de Educación en Ingeniería*. https://doi.org/10.26507/ponencia.585

Charte, F., Espinilla, M., Rivera, A., & Pulgar, F. (2017). Uso de dispositivos FPGA como apoyo a la enseñanza de asignaturas de arquitectura de computadores. *Enseñanza y Aprendizaje de Ingeniería de Computadores. Universidad de Granada*, *7*, 37-52.

cistec. (2018, junio 27). Las mil y una aplicaciones del Internet de las Cosas—IoT. *CISTEC Technology*. https://www.cistec.es/las-mil-y-una-aplicaciones-del-internet-de-las-cosas-iot/

Davidson Tremblay, P. (2014). *Protection et intégrité des systèmes embarqués réseautés.*

Del Barrio García, A. A., Roa Romero, C., Recas Piorno, J., Botella Juan, G., Tenllado Van Der Reijden, C. T., & Piñuel Moreno, L. (2017). *Framework libre para el desarrollo de aplicaciones en el escenario de "Internet of Things"*.

Dewar, A., Thibault, J.-P., & O'Flynn, C. (2020). *NAEAN0010: Power Analysis on FPGA Implementation of AES Using CW305 & ChipWhisperer*. October.

Eagle, C. (2011). *The IDA pro book*. no starch press.

Espinoza Sánchez, D. L., & Cisneros Vilca, V. M. (2013). *Diseño de un modelo para la toma de decisiones para los proyectos de softwareque utilizan la metodología espiral*. https://renati.sunedu.gob.pe/handle/sunedu/2699120

Gallissot, M., Puys, M., & Thevenon, P.-H. (2020). WonderCloud, une plateforme pour l'analyse et l'émulation de micrologiciels ainsi que la composition de pots de miels. *Proc. of the 27th C&esar conference on Deceptive Security (C&esar 2020), Rennes, France*.

García, M., & Alberto, J. (2019). *La importancia de la gestión de riesgos y seguridad en el internet de las cosas (IOT)*. http://repository.unipiloto.edu.co/handle/20.500.12277/6754

Gélvez-Rodríguez, L. F., & Santos-Jaimes, L. M. (2020). Internet de las Cosas: Una revisión sobre los retos de seguridad y sus contramedidas. *Revista Ingenio*, *17*(1), Article 1. https://doi.org/10.22463/2011642X.2370

Goméz González, C. (2016). *Implementación de algoritmos SHM en una plataforma embebida en FPGA*.

González, D. R. (2013). Arquitectura y Gestión de la IoT. *Telemática*, *12*(3), Article 3.

He, W. (2014). *Side-Channel Attack Protection Techniques in FPGA Systems using Enhanced Dual-Rail Solutions*. Industriales.

Juan Quevedo, V. (2022). *Guía para la definición y adaptación de algoritmos para su implementación en FPGAs mediante High Level Synthesis* [Proyecto/Trabajo fin de carrera/grado, Universitat Politècnica de València]. https://riunet.upv.es/handle/10251/179220

Kumar.V.G, K., Poojary, A., Rai, C. S., & Nagesh, H. R. (2017). Implementation of lightweight cryptographic algorithms in FPGA. *2017 International Conference on Circuits, Controls, and Communications (CCUBE)*, 232-235. https://doi.org/10.1109/CCUBE.2017.8394141

Lara-Nino, C. A., Diaz-Perez, A., & Morales-Sandoval, M. (2020). *Métodos de Criptografía Ligera para brindar Servicios de Seguridad al Internet de las Cosas*.

Liu, L., Wang, B., Yu, B., & Zhong, Q. (2017). Automatic malware classification and new malware detection using machine learning. *Frontiers of Information Technology & Electronic Engineering*, *18*(9), 1336-1347.

Llorente Aragón, R. (2017). *Configuración y puesta en marcha de módulos de comunicación inalámbrica en un sistema reconfigurable*.

López Echeverry, A. M., & Santa V., J. A. (2011). Ataques de seguridad en las FPGAs y cómo prevenirlos. *Scientia et Technica*, *2*(48), 105-110.

Magyari, A., & Chen, Y. (2022). Review of State-of-the-Art FPGA Applications in IoT Networks. *Sensors*, *22*(19), Article 19. https://doi.org/10.3390/s22197496

Manotas Campos, J. J., & Martinez Marín, N. (2018). *Exploración de las plataformas IOT en el mercado para fomentar el conocimiento, buen uso y efectividad de los dispositivos IOT creados en la facultad de ingeniería y ciencias básicas de la Institución Universitaria Politécnico Grancolombiano.* https://alejandria.poligran.edu.co/handle/10823/1215

Marianetti, O. L., Godoy, P. D., Chediak, E. E., Fontana, D. S., & García Garino, C. (2021). *Tecnología de dispositivos de lógica reconfigurable aplicada en la implementación segura de sistemas de IoT*. XXIII Workshop de Investigadores en Ciencias de la Computación (WICC 2021, Chilecito, La Rioja). http://sedici.unlp.edu.ar/handle/10915/120032

Mayo Vilches, J. (2021). *Diseño y simulación firmware de una transmisión de squitters en modo S encriptados mediante RSA*.

Mondragón, M. V. P., & Guillén, E. P. (2018). Servicios de autenticación y autorización orientados a internet de las cosas. *Telemática*, *17*(2), Article 2.

Monzón, G., Todt, C. M., Bolatti, D., Gramajo, S. D., & Scappini, R. J. R. (2019). *Modelo de seguridad IoT*. XXV Congreso Argentino de Ciencias de la Computación (CACIC) (Universidad Nacional de Río Cuarto, Córdoba, 14 al 18 de octubre de 2019). http://sedici.unlp.edu.ar/handle/10915/91363

O'Flynn, C., & Chen, Z. (David). (2014). ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. En E. Prouff (Ed.), *Constructive Side-Channel Analysis and Secure Design* (pp. 243-260). Springer International Publishing. https://doi.org/10.1007/978-3-319-10175-0_17

Ordaz-García, O. O., Ortiz-López, M., Quiles-Latorre, F. J., Olague, J. G. A., & Bellido-Outeiriño, F. J. (2018). Implementación del protocolo DALI en FPGAs de bajo consumo de energía para uso en redes inalámbricas de sensores. *Research in Computing Science*, *147*, 253-263.

Pardo Mesa, A. F., & Ardila Rodríguez, F. A. (2019). *Blockchain Applied to Cybersecurity Challenges*.

Piñal, J. F., Álvarez, R., & Sánchez, A. M. (2009). Implementación en hardware del estándar de encriptación avanzado (aes), en una plataforma fpga, empleando el microcontrolador picoblaze^{TM}. *e-Gnosis*, *7*.

Potestad Ordoñez, F. E. (2019a). *Vulnerabilidad y análisis diferencial mediante inserción de fallos de cifradores Trivium en FPGA y ASIC.*

Potestad Ordoñez, F. E. (2019b). *Vulnerabilidad y análisis diferencial mediante inserción de fallos de cifradores Trivium en FPGA y ASIC.*

Rodríguez Valido, M., Gutiérrez Castañeda, M., Magdaleno Castelló, E., Hernández Expósito, D., Pérez Nava, F., & Guerrero Vidal, L. (2012). Metodología de diseño en FPGA usando Xilinx System Generator. *Tecnologías Aplicadas en la Enseñanza de la Electrónica: TAEE 2012 : Actas del X Congreso de Tecnologías Aplicadas en la Enseñanza de la Electrónica, Escuela de Ingeniería Industrial Universidad de Vigo Vigo, España 13 al 15 Junio de 2012*, *2012*, *ISBN 978-84-8158-588-9*, 50. https://dialnet.unirioja.es/servlet/articulo?codigo=8615751

Romero, B., & Stiven, D. (2020). *Retos en la seguridad de dispositivos para el Internet de las Cosas (IoT)*. http://repository.unad.edu.co/handle/10596/35737

Sanchez Alcon, J. A., Lopez Santidrian, L., & Martínez Ortega, J. F. (2015). Solución para garantizar la privacidad en el internet de las cosas. *El Profesional de la Informacion*, *24*(1), Article 1.

Sánchez Narváez, R. (2018). *Implementación de aplicaciones en FPGA*.

Sánchez-Solano, S., Cabrera, A., Jiménez, C., Brox, P., Baturone, I., & Barriga, A. (2023). *IMPLEMENTACIÓN SOBRE FPGAS DE SISTEMAS DIFUSOS PROGRAMABLES*.

Señor Sánchez, J. (2021). *Aplicación de sistemas post-cuánticos a la seguridad en nodos de internet of things*. Industriales.

Serrano Santos, F. (2017). *Emulación basada en FPGA de los efectos de los single event upsets ocasionados por la radiación en circuitos digitales tolerantes a fallos*.

Sicari, S., Rizzardi, A., Grieco, L. A., & Coen-Porisini, A. (2015). Security, privacy and trust in Internet of Things: The road ahead. *Computer Networks, 76*, 146-164. https://doi.org/10.1016/j.comnet.2014.11.008

Sisterna, C. (2010). Field programmable gate arrays (FPGAS). *Electrónica Digital II, Facultad de Ingeniería, Universidad Nacional de San Juan (Argentina)*.

Sunkavilli, S., Zhang, Z., & Yu, Q. (2021). New Security Threats on FPGAs: From FPGA Design Tools Perspective. *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 278-283. https://doi.org/10.1109/ISVLSI51109.2021.00058

Tenelema Arias, E. N. (2020a). *Implementación de un modelo de seguridad para mitigación de vulnerabilidades en ambientes de almacenamiento en la nube con base en las normas ISO 27017 y 27018.*

Tenelema Arias, E. N. (2020b). *Implementación de un modelo de seguridad para mitigación de vulnerabilidades en ambientes de almacenamiento en la nube con base en las normas ISO 27017 y 27018.*

Ugarte Hurtado, J. A. (2021). *Diseño de una aplicación para la medida del consumo de potencia de una FPGA.*

Verdasco Ayala, B. (2019). *Implementación de función física inclonable (PUF) en FPGA.*

Villalta Bustillo, I. (2019). *Caracterización de la tolerancia a fallos de circuitos implementados en FPGAs*.

Viñé Viñuelas, M. (2012). *Implementación de un VHDL de un decodificador Viterbi y su integración en un prototipo de un sistema WiMAX*.

Zerbini, C. A. (2015). *Arquitecturas reprogramables de procesamiento de flujos de paquetes en redes de datos*.

ANEXOS

ANEXO A.

Objetivo del Análisis

Establecer una definición clara de todos los requerimientos necesarios para asegurar la protección y alcanzar los objetivos establecidos en el trabajo de investigación.

Requerimientos del Sistema

Nomenclatura	Requerimiento	Descripción	Prioridad
SySR1	El sistema debe ser capaz de simular el procesamiento interno que realiza un FPGA	Un FPGA realiza diversas operaciones lógicas y de procesamiento de señales dentro de su estructura interna, como la configuración de las celdas lógicas, el enrutamiento de las señales y la ejecución de algoritmos específicos. La simulación del procesamiento interno del FPGA permite evaluar el comportamiento y el rendimiento del diseño implementado en el FPGA, así como identificar	**Alta**

		posibles problemas o fallos en la lógica del sistema (Benso & Prinetto, 2004).	
SySR2	El sistema debe contar con la posibilidad de la reprogramabilidad.	La reprogramabilidad se refiere a la capacidad de modificar la configuración de un FPGA después de su fabricación inicial. Esto permite actualizar o modificar el diseño y la funcionalidad del FPGA sin tener que reemplazar el hardware físico. La reprogramabilidad es esencial en el contexto de la evaluación de seguridad de dispositivos FPGA, ya que permite realizar cambios en el diseño del FPGA para mitigar posibles vulnerabilidades o aplicar medidas de seguridad adicionales. Además, la reprogramabilidad también facilita la adaptación del FPGA a nuevas funciones o requisitos del sistema sin la necesidad de cambiar el hardware subyacente (Zerbini, 2015).	**Alta**
SySR3	El sistema debe mitigar en cierto grado las	La mitigación de vulnerabilidades implica la implementación de medidas de seguridad y técnicas de	**Alta**

	vulnerabilidades presentes.	protección que reduzcan el riesgo de explotación de las vulnerabilidades identificadas. Esto puede incluir el uso de algoritmos criptográficos robustos, técnicas de detección y prevención de ataques, controles de acceso, autenticación y encriptación de datos, entre otras estrategias de seguridad. El grado de mitigación dependerá de la efectividad de las medidas implementadas y su capacidad para abordar las vulnerabilidades específicas identificadas en el contexto de los sistemas IoT con dispositivos FPGA (López Echeverry & Santa V., 2011).	
SySR4	El sistema debe tener la posibilidad de ser implementado en un sistema IoT.	Los sistemas IoT se caracterizan por su amplia gama de dispositivos interconectados que recopilan y comparten datos en tiempo real. Para que el sistema de mitigación sea efectivo, debe ser compatible con los protocolos de comunicación utilizados en los sistemas IoT y ser capaz de integrarse sin problemas con la	**Alta**

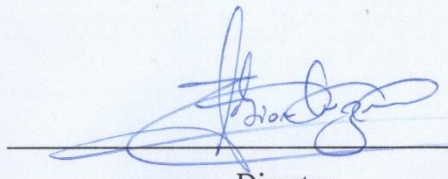
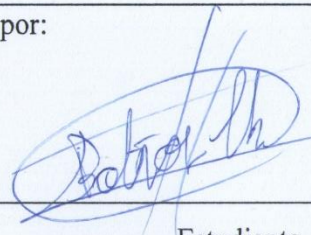
		infraestructura existente. Esto implica tener en cuenta los requisitos de conectividad, escalabilidad, interoperabilidad y eficiencia energética propios de los sistemas IoT. Al cumplir con este requerimiento, el sistema de mitigación podrá adaptarse a diferentes aplicaciones y entornos de IoT, proporcionando una capa adicional de seguridad para los dispositivos FPGA utilizados en dichos sistemas (Kumar.V.G et al., 2017).	
SySR5	El sistema debe contar con mecanismos de monitoreo y auditoría para registrar eventos de seguridad.	Los mecanismos de monitoreo y auditoría son fundamentales para tener un control efectivo sobre las acciones realizadas en el sistema, identificar posibles amenazas o intrusiones y permitir una respuesta rápida ante eventos de seguridad. El monitoreo y la auditoría pueden incluir la captura de registros de eventos, seguimiento de accesos y actividades, detección de anomalías y generación de informes de seguridad. Al	**Alta**

		contar con estos mecanismos, se obtiene una visibilidad completa de la seguridad del sistema y se facilita la detección temprana y la respuesta ante posibles incidentes de seguridad (Pardo Mesa & Ardila Rodríguez, 2019).	
SySR6	El sistema debe tener las consideraciones del consumo energético.	Es importante tener en cuenta que los dispositivos FPGA se utilizan en sistemas IoT que pueden estar alimentados por fuentes de energía limitadas, como baterías. Por lo tanto, es fundamental optimizar el consumo de energía del sistema para garantizar una operación eficiente y prolongar la vida útil de la fuente de energía (Ordaz-García et al., 2018).	**Media**
SySR7	El sistema debe cumplir con estándares de seguridad y seguir buenas prácticas de diseño seguro.	La seguridad es un aspecto fundamental en los sistemas IoT, especialmente cuando se trata de proteger la integridad, confidencialidad y disponibilidad de los datos y la comunicación. Es importante considerar y aplicar estándares de	**Alta**

		seguridad ampliamente aceptados, como ISO/IEC 29148. Estos estándares proporcionan pautas y mejores prácticas para identificar y abordar las vulnerabilidades de seguridad en los sistemas de IoT (Zerbini, 2015).	
SySR8	El sistema debe permitir la elección de pines tanto de entrada de datos como de salida de datos.	Los pines son los puntos de conexión física en un dispositivo FPGA que se utilizan para la entrada y salida de señales. La elección de pines es importante para garantizar una correcta interconexión entre el dispositivo FPGA y otros componentes del sistema IoT. Al permitir la elección de pines, el sistema ofrece flexibilidad en la configuración de las interfaces de comunicación y la integración con otros dispositivos. Esto implica que el sistema debe contar con una interfaz o herramienta que permita al usuario especificar los pines de entrada y salida deseados (Ugarte Hurtado, 2021).	**Media**

SySR9	El sistema de simulación debe contar con la capacidad de incorporar configuraciones específicas para la implementación del ataque deseado.	Cada tipo de ataque puede requerir diferentes configuraciones, como la selección de puntos de inyección de fallas, la duración y el momento de la inyección, o los patrones de manipulación de datos. Al contar con la capacidad de incorporar configuraciones específicas para la implementación del ataque deseado, el sistema de simulación ofrece flexibilidad y adaptabilidad para evaluar diferentes escenarios de vulnerabilidad (López Echeverry & Santa V., 2011).	**Media**
SySR10	El sistema debe ser capaz de realizar pruebas de operatividad.	Las pruebas de operatividad pueden incluir pruebas funcionales, pruebas de rendimiento, pruebas de estabilidad y otras pruebas que evalúen el comportamiento del sistema en diferentes escenarios y condiciones. Estas pruebas permiten detectar posibles errores, fallas o limitaciones en el diseño del sistema y brindan la oportunidad de corregirlos antes de la implementación en un entorno de producción. Esto	**Alta**

puede incluir la capacidad de generar casos de prueba, ejecutar las pruebas, recopilar y analizar los resultados, y generar informes de prueba detallados (Goméz González, 2016).

Revisado por:	Elaborado por:
	
Director	Estudiante
Msc. Fabián Geovanny Cuzme Rodríguez	Bolívar Saul Bolaños Chamorro

ANEXO B.

Objetivo del Análisis

El objetivo principal es identificar y describir de manera precisa todos los requerimientos de arquitectura relevantes que son necesarios para garantizar la seguridad y lograr los objetivos planteados en el trabajo de investigación.

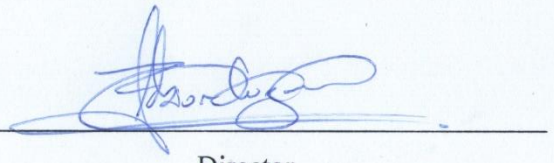
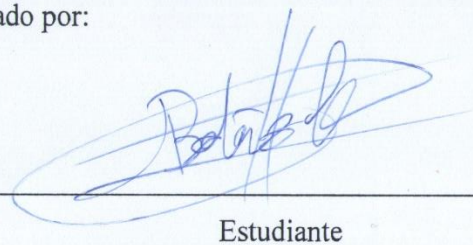
Requerimientos del Sistema			
Nomenclatura	**Requerimiento**	**Descripción**	**Prioridad**
SrSH1	El sistema debe ser capaz de soportar algoritmos criptográficos robustos para garantizar la seguridad de la comunicación en los sistemas IoT.	La capacidad del sistema para implementar algoritmos criptográficos que brinden seguridad en la comunicación de los sistemas IoT. Los algoritmos criptográficos son fundamentales para proteger la confidencialidad, integridad y autenticidad de los datos transmitidos en un entorno IoT. Estos algoritmos pueden incluir cifrado simétrico, cifrado asimétrico, funciones hash y protocolos de autenticación, entre otros. La implementación de algoritmos criptográficos robustos es esencial para	**Alta**

		asegurar que la información transmitida en los sistemas IoT no sea comprometida por ataques maliciosos (Aldaya & Sarmiento, 2013).	
SrSH2	El sistema debe contar con mecanismos de autenticación y autorización para garantizar el acceso seguro a los recursos del sistema.	La necesidad de implementar mecanismos de autenticación y autorización en el sistema con el fin de controlar y gestionar el acceso a los recursos. La autenticación se refiere al proceso de verificar la identidad de un usuario o dispositivo antes de permitir el acceso, mientras que la autorización se encarga de definir los privilegios y permisos que tiene cada usuario o dispositivo una vez que ha sido autenticado. Estos mecanismos son esenciales para garantizar que solo los usuarios autorizados puedan acceder y manipular los recursos del sistema IoT, protegiendo así la integridad y confidencialidad de la información (Verdasco Ayala, 2019).	**Alta**

SrSH3	El dispositivo FPGA debe contar con mecanismos de detección y mitigación de ataques de denegación de servicio para asegurar la disponibilidad de los servicios y aplicaciones.	Los mecanismos de detección permiten identificar patrones y comportamientos anómalos que puedan indicar un ataque de DoS en curso, mientras que los mecanismos de mitigación se encargan de tomar medidas para mitigar o contrarrestar estos ataques y mantener la disponibilidad del sistema (Villalta Bustillo, 2019).	**Baja**
SrSH4	El sistema debe implementar políticas de control de acceso y auditoría para monitorear y registrar las actividades de los usuarios y detectar posibles amenazas.	La necesidad de incorporar en el sistema IoT políticas de control de acceso y auditoría para garantizar la seguridad y la integridad de los datos. Las políticas de control de acceso permiten gestionar y regular el acceso a los recursos del sistema, asegurando que solo los usuarios autorizados puedan acceder a la información y funcionalidades pertinentes. Por otro lado, la auditoría implica el monitoreo y registro de las actividades de los usuarios, creando un registro detallado de las	**Alta**

		acciones realizadas en el sistema. Esto facilita la detección de comportamientos anómalos o potencialmente maliciosos, permitiendo una respuesta oportuna ante posibles amenazas o incidentes de seguridad (López Echeverry & Santa V., 2011).	
SrSH5	Las pruebas sobre el sistema deben reflejar el nivel de mitigación para las vulnerabilidades.	Las pruebas deben ser diseñadas de manera que permitan simular diferentes escenarios y situaciones de ataque, con el objetivo de comprobar la efectividad de las medidas de seguridad implementadas en el sistema. El resultado de estas pruebas debe reflejar claramente el grado de mitigación alcanzado, brindando información sobre la capacidad del sistema para resistir y responder adecuadamente a los ataques (Tenelema Arias, 2020b).	**Alta**

SrSH6	El dispositivo FPGA debe contar con protección contra sobrecargas eléctricas y transitorios para evitar daños físicos y garantizar la integridad del sistema	La protección contra sobrecargas y transitorios puede incluir la utilización de componentes de protección, como diodos de supresión de voltaje, filtros de línea, aisladores ópticos, entre otros. Estos componentes ayudan a limitar los niveles de voltaje y corriente que llegan al FPGA, evitando daños físicos en los circuitos y garantizando la integridad del sistema (Serrano Santos, 2017).	**Media**
SrSH7	El dispositivo FPGA debe cumplir con estándares de seguridad reconocidos y ser sometido a pruebas y evaluaciones de seguridad independientes para garantizar su robustez y confiabilidad en entornos IoT.	La importancia de que el dispositivo FPGA cumpla con estándares de seguridad reconocidos, como los establecidos por organizaciones y organismos internacionales, entre los que se pueden mencionar ISO, NIST, IEC, entre otros. Estos estándares proporcionan directrices y mejores prácticas en materia de seguridad que ayudan a asegurar la robustez y confiabilidad del dispositivo (Potestad Ordoñez, 2019a).	**Alta**

Revisado por:	Elaborado por:
	
Director	Estudiante
Msc. Fabián Geovanny Cuzme Rodríguez	Bolívar Saul Bolaños Chamorro

ANEXO C.

Objetivo del Análisis

El objetivo principal es realizar una comparación para los diferentes sistemas de mitigación para los ataques respectivos en base a un sistema IoT con dispositivos FPGA.

Inyección de Fallas							
Mitigación	**Referencias bibliográficas**	**Efectividad**	**Compatibilidad**	**Robustez**	**Bajo costo**	**Estándares de seguridad**	**Latencia baja**
Checksum	Washbrum, I. M. (2019).	El checksum es un método eficaz	El checksum es ampliamente compatible y se	El checksum es robusto en	Implementar el	El checksum es una técnica	El cálculo del

	Metodología de gestión de seguridades informáticas para internet de las cosas. *Espirales Revista Multidisciplinaria de Investigación*, *3*(24).	para verificar la integridad de los datos. Al agregar un valor de checksum a los datos transmitidos o almacenados, es posible detectar errores o alteraciones en los datos. Si se produce una inyección de fallas en los datos, el checksum no coincidirá y se detectará un error.	puede implementar en una variedad de sistemas y plataformas. Es una técnica comúnmente utilizada en la transmisión de datos y en el almacenamiento de datos para garantizar su integridad.	la detección de errores, siempre y cuando se utilice correctamente. Puede detectar errores tanto aleatorios como sistemáticos en los datos, lo que lo hace efectivo contra inyecciones de fallas.	checksum generalmente tiene un bajo costo computacional y no requiere recursos significativos, lo que lo hace asequible en términos de recursos.	ampliamente aceptada en términos de seguridad y protección de datos. Se utiliza en una variedad de aplicaciones, desde la verificación de archivos hasta la transmisión segura de datos.	checksum generalmente introduce una latencia insignificante en la operación, lo que significa que es adecuado para aplicaciones en tiempo real.

Detección y corrección de errores (ECC)	Alvarado Escamilla, R. (2004). Códigos para detección y corrección de errores en comunicaciones digitales. *Ingeniérías, 7*(25), 51-60.	La ECC es altamente efectiva para detectar y corregir errores en datos almacenados o transmitidos. Puede detectar errores de un solo bit y corregirlos, así como identificar errores más graves. La efectividad de ECC depende del código utilizado y su capacidad para detectar y corregir errores.	Se utiliza en la mayoría de las RAM, en sistemas de almacenamiento y en la transmisión de datos críticos. Su compatibilidad depende de la implementación específica y de la capacidad de hardware para admitirla.	Puede tolerar y corregir múltiples errores, lo que la hace adecuada para aplicaciones críticas. Sin embargo, la robustez también depende de la implementación y de cuántos errores puede detectar y corregir el código ECC utilizado.	La implementación de ECC puede aumentar el costo de hardware, ya que requiere hardware adicional para realizar cálculos de corrección de errores. Sin embargo, dado su valor en la protección de datos críticos,	La ECC se basa en estándares bien definidos, como los códigos Hamming, Reed-Solomon, etc. Estos estándares brindan confianza en la seguridad y la integridad de los datos en sistemas críticos.	Es generalmente baja, especialmente en comparación con otros métodos de corrección de errores más avanzados. Puede detectar y corregir errores en tiempo real, lo que la hace adecuada para aplicaciones

					muchas aplicaciones lo consideran una inversión necesaria.		en las que la latencia es crítica.
Redundancia	Cañas Palencia, J. A. (2021). Mecanismos de control de redundancia en percepción cooperativa para el vehículo autónomo conectado.	Al duplicar o triplicar componentes críticos, se pueden detectar y corregir errores mediante comparaciones entre las réplicas. La efectividad depende de la implementación específica y del grado de	La compatibilidad de la redundancia depende de cómo se implemente en un sistema. No es una solución universalmente compatible.	Es un sistema sea más robusto al proporcionar una tolerancia a fallos mejorada. Puede continuar funcionando incluso si una réplica experimenta un fallo. La robustez	La redundancia tiende a aumentar el costo de hardware y mantenimiento debido a la necesidad de componentes adicionales. El costo puede variar	Los estándares de seguridad pueden requerir redundancia en sistemas críticos, como en la aviación o la industria nuclear. Cumplir con estos estándares	La latencia en sistemas redundantes puede ser baja si se implementa adecuadamente. En algunos casos, como en sistemas de tolerancia a fallos en tiempo real, la latencia

		redundancia utilizado.		aumenta con el número de réplicas y la capacidad de detectar fallos.	según el grado de redundancia y el tipo de componentes duplicados.	puede ser una consideración importante para aplicaciones de alta seguridad.	debe mantenerse lo más baja posible para garantizar un rendimiento adecuado.
Temporización segura	Campo Giralte, L. (2014). Una arquitectura distribuida para la detección, comunicación y mitigación de la denegación de servicio.	Se centra en reducir la variabilidad en el tiempo de ejecución de operaciones críticas. Esto dificulta que un atacante infiera información mediante mediciones precisas de tiempos de respuesta.	Esta técnica puede depender de la infraestructura y el hardware existentes. Puede requerir modificaciones en el código y ajustes en el sistema para implementar correctamente medidas de	La robustez de un sistema al dificultar la extracción de información a través de ataques de canal lateral. Sin embargo, su eficacia depende de la implementación precisa y de la capacidad de	La implementación de medidas de temporización segura puede ser relativamente económica desde una perspectiva de hardware y software, ya que generalmente	En sistemas de alta seguridad, como los utilizados en el gobierno, la industria de defensa y la banca. Cumplir con estándares de seguridad rigurosos puede requerir su	Generalmente busca minimizar la latencia tanto como sea posible, ya que el objetivo es garantizar operaciones dentro de un marco de tiempo específico. La latencia

			temporización segura.	mantener una temporización consistente.	e no requiere componentes adicionales.	implementación.	baja es esencial para evitar retrasos innecesarios en operaciones críticas.
Pruebas de integridad	López Naranjo, M. A. (2022). *Análisis de amenazas IOT en un sistema domótico* (Master's thesis, Pontificia Universidad Católica del Ecuador).	Altamente efectivas para evaluar la capacidad de un sistema para mantener su integridad frente a posibles amenazas y ataques. Estas pruebas pueden identificar vulnerabilidades y debilidades, lo	La compatibilidad de las pruebas de integridad depende de cómo se integren en el entorno de desarrollo y evaluación del sistema. Deben ser diseñadas para ser compatibles con el hardware y el	Las pruebas de integridad mejoran la robustez de un sistema al identificar y abordar las vulnerabilidades y debilidades. La robustez se logra mediante la detección	Aunque pueden requerir inversión en términos de recursos y tiempo, a menudo son más rentables que enfrentar incidentes de	Las pruebas de integridad son fundamentales para el cumplimiento de estándares de seguridad, ya que ayudan a demostrar que se han tomado medidas para	La latencia asociada con las pruebas de integridad puede variar según el tipo de prueba y la frecuencia con la que se realizan. Sin embargo, es importante minimizar la interrupción

		que permite tomar medidas correctivas.	software utilizados.	temprana de posibles problemas y la implementación de medidas correctivas.	seguridad o fallos graves en el futuro.	garantizar la integridad de un sistema.	del sistema durante las pruebas para mantener una latencia baja en operaciones críticas.
Cifrado y autenticación	Moral de Aguilar, N. (2021). Seguridad en la Internet of Things: Ataques y medidas de seguridad en modelos Cloud de IoT.	El cifrado convierte la información en un formato ilegible para terceros no autorizados, mientras que la autenticación verifica la identidad de las partes involucradas en	La compatibilidad de estas técnicas depende de cómo se implementen en el sistema. Deben ser compatibles con los estándares y protocolos utilizados en el entorno, como TLS/SSL para la	Tanto el cifrado como la autenticación mejoran la robustez de un sistema al protegerlo contra amenazas como el espionaje, la interceptación de datos y el	La implementación de cifrado y autenticación puede requerir inversión en términos de hardware y software. Sin embargo, el costo se	Estas técnicas están respaldadas por estándares de seguridad bien definidos. El cifrado puede seguir estándares como AES (Advanced Encryption	La latencia asociada con cifrado y autenticación puede variar según la complejidad de los algoritmos y el rendimiento del hardware.

		una comunicación.	autenticación en comunicaciones web.	acceso no autorizado. Utilizan algoritmos y métodos de seguridad bien establecidos.	justifica por la protección proporciona da frente a posibles pérdidas de datos o intrusiones.	Standard) y la autenticación puede utilizar métodos como HMAC (Hash-based Message Authenticatio n Code).	Sin embargo, los avances tecnológicos han reducido significativa mente la latencia en las operaciones de cifrado y autenticació n.
Ataque del Canal Lateral							
Mitigación	**Referencias bibliográficas**	**Efectividad**	**Compatibilidad**	**Robustez**	**Bajo costo**	**Estándares de seguridad**	**Latencia baja**
Enmascaramie nto Aleatorio	Escalante Quimis, O. A. (2021). *Proto tipo de sistema de seguridad de*	Es altamente efectivo en la mitigación de ataques de canal lateral. Al introducir ruido	Puede requerir cambios en el software y hardware existentes para implementar	La robustez del enmascarami ento aleatorio depende en gran medida	Requiere recursos computacio nales adicionales para generar	Debe seguir los estándares criptográficos reconocidos para garantizar	La latencia depende del tamaño de las máscaras aleatorias y la velocidad

	base de datos en organizaciones públicas para mitigar ataques cibernéticos en Latinoamérica (Bachelor's thesis).	aleatorio en las operaciones, dificulta la correlación de datos internos con mediciones de canal lateral.	adecuadamente el enmascaramiento aleatorio. La compatibilidad depende de la arquitectura del sistema.	de la calidad de la máscara aleatoria y la implementación precisa. Si se aplica correctamente, es robusto contra una amplia gama de ataques de canal lateral.	máscaras aleatorias, pero no implica un costo significativo en términos de hardware o software adicional.	que las máscaras aleatorias sean seguras.	de generación. En muchas aplicaciones, la latencia resultante es aceptable.
Diversificación de Claves	Calatayud, A., & Ketterer, J. A. (2016). Gestión integral de riesgos para cadenas de valor. *Nota Técnica Núm.*	Es altamente efectiva para proteger las claves, al generar claves diferentes para cada instancia o transacción, se hace más difícil para un atacante	La compatibilidad depende de cómo se implementa la diversificación de claves en un sistema. En algunos casos, puede requerir	La diversificación de claves es robusta contra muchos tipos de ataques de canal lateral, ya que dificulta que	El costo de la diversificación de claves puede variar según la implementación. Puede ser bajo si se realiza de	La diversificación de claves suele cumplir con estándares de seguridad y criptográficos reconocidos. La elección	La latencia resultante depende de cómo se gestionen las claves diversificadas. En muchos casos, la latencia

	IDB-TN-922. Washington, DC: BID.	obtener información sensible.	modificaciones significativas en el hardware y el software para admitir múltiples claves.	un atacante correlacione múltiples mediciones con una sola clave.	manera eficiente, pero en sistemas altamente especializados podría requerir hardware adicional.	de algoritmos y protocolos seguros es fundamental.	sigue siendo aceptable para aplicaciones de seguridad.
Control de Acceso Físico	Fagan, M., Megas, K. N., Scarfone, K., & Smith, M. (2021). Actividades fundamentales de ciberseguridad para los fabricantes de dispositivos	Es altamente efectivo para proteger sistemas y dispositivos contra ataques que implican el acceso físico directo al hardware. Al restringir el acceso físico, se	La compatibilidad depende de cómo se implementa el control de acceso físico en un sistema. En algunos casos, puede requerir inversiones significativas en	El control de acceso físico es robusto contra una amplia gama de ataques de canal lateral que requieren acceso directo al dispositivo. Si se	El costo puede ser variable dependiendo de la implementación específica. Puede requerir inversiones en sistemas	El control de acceso físico a menudo se rige por estándares y mejores prácticas reconocidos en seguridad física y de acceso.	El control de acceso físico generalmente no agrega latencia significativa a las operaciones normales del sistema. Sin embargo, puede haber

	de IoT. *Gaithersburg, MD, Mar*.	reduce en gran medida la probabilidad de que un atacante pueda realizar mediciones directas.	hardware y procedimientos de seguridad.	implementa correctamente, puede proporcionar una defensa sólida.	de seguridad física, como cerraduras, tarjetas de acceso, sistemas de videovigilancia, etc.		retrasos mínimos al autenticar y permitir el acceso.
Cifrado de Lado del Cliente	Burgos Yar, V. V. (2021). *Cifrado de datos usando Cadena de Bloques (BlockChain) como tecnología de convergencia para dispositivos*	Los datos se cifran en el dispositivo del cliente antes de ser transmitidos o almacenados en el servidor, lo que proporciona una capa adicional de seguridad.	La compatibilidad depende de la implementación específica. Requiere que las aplicaciones o servicios sean capaces de cifrar y descifrar datos en el cliente. Es ampliamente compatible con	El cifrado de lado del cliente es robusto y resistente a varios tipos de ataques, incluyendo la interceptación de datos en tránsito y el acceso no autorizado a	El costo puede variar dependiendo de la implementación. El cifrado de lado del cliente puede requerir recursos adicionales	El cifrado de lado del cliente sigue estándares y protocolos de seguridad reconocidos, como TLS (para datos en tránsito) y AES (para datos en reposo).	La latencia puede aumentar ligeramente debido a la necesidad de cifrar y descifrar datos en el cliente. Sin embargo, con hardware y

	móviles asociados con IoT (Internet of Things), en la capa de aplicación del modelo de capas IoT (Master's thesis).	sistemas modernos.	datos almacenados en el servidor.	en términos de desarrollo de software y gestión de claves.	algoritmos adecuados, la latencia suele ser aceptable.

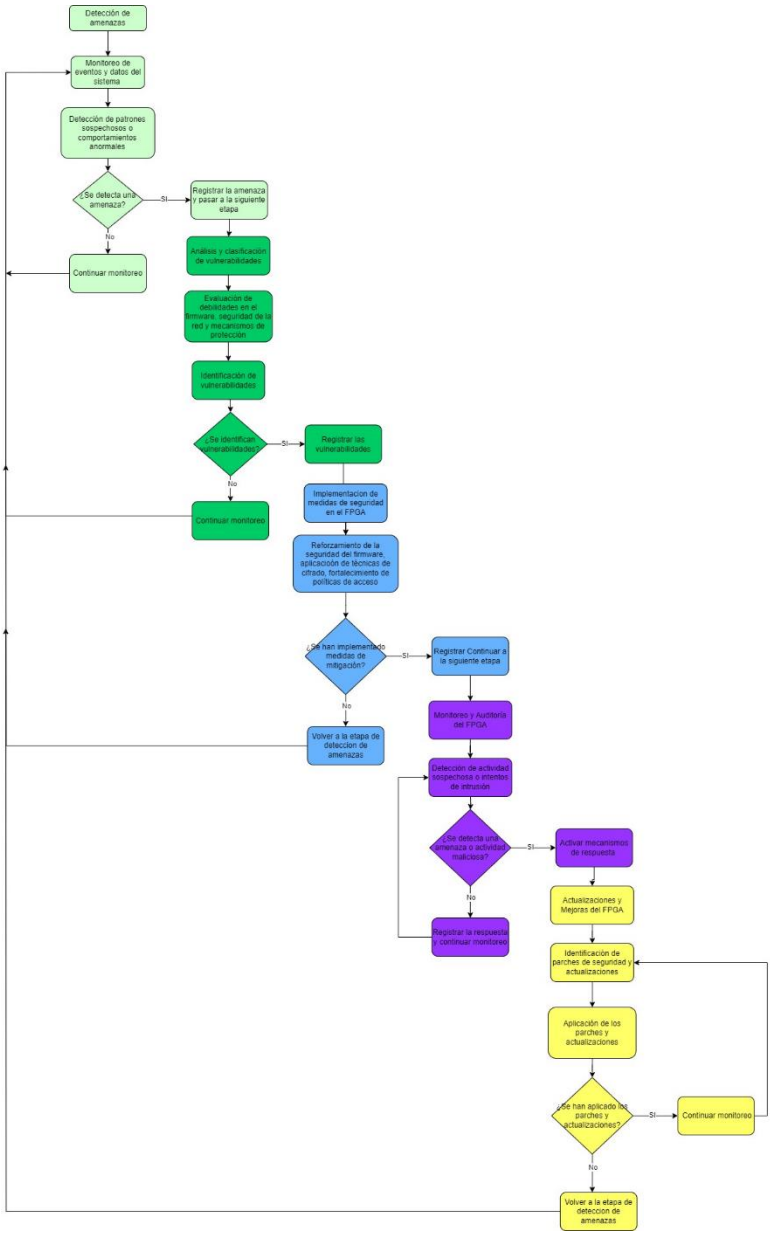
Revisado por:	Elaborado por:
Director	Estudiante
Msc. Fabián Geovanny Cuzme Rodríguez	Bolívar Saul Bolaños Chamorro

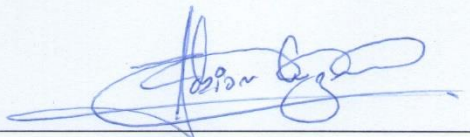
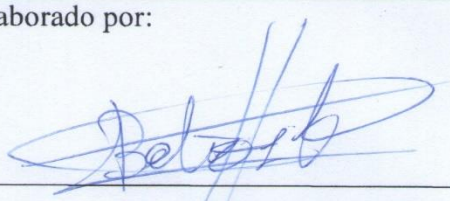
ANEXO D.

Objetivo del Análisis

El objetivo principal es describir las etapas para el diseño del sistema de mitigación presente en el trabajo de titulación de manera precisa.

Para una amplia visualización visitar el enlace: https://github.com/BolivarSE/Trabajo-de-Grado/blob/main/DiagramaCompleto.drawio.png



Revisado por:	Elaborado por:
	
Director Msc. Fabián Geovanny Cuzme Rodríguez	Estudiante Bolívar Saul Bolaños Chamorro