UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Inyección de fallos de voltaje para controlar el flujo de ejecución en microcontroladores.

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Blasco Pellicer, María Teresa

Tutor/a: Pérez Blasco, Pascual

CURSO ACADÉMICO: 2023/2024

Resum

El TFG se centra en l'estudi i utilització de tècniques de hardware hacking per a manipular el comportament d'un microprocessador i aconseguir controlar el seu flux d'execució. Així mateix, s'exposa la problemàtica que comporta la utilització d'estes tècniques per a eludir la seguretat d'una arrancada segura. Per a això, es realitza un experiment pràctic d'injecció de fallades de voltatge en un microcontrolador explicant els components necessaris, la configuració de l'entorn de treball i les limitacions trobades així com l'impacte en la seguretat i possibles contramesures per a mitigar el risc.

Paraules clau: microcontrolador, flux d'execució, injecció de fallades, ciberseguretat, arrancada segura, hardware hacking, sistemes embeguts

Resumen

El TFG se centra en el estudio y utilización de técnicas de hardware hacking para manipular el comportamiento de un microprocesador y conseguir controlar su flujo de ejecución. Así mismo, se expone la problemática que conlleva la utilización de estas técnicas para eludir la seguridad de un arranque seguro. Para ello, se realiza un experimento práctico de inyección de fallos de voltaje en un microcontrolador explicando los componentes necesarios, la configuración del entorno de trabajo y las limitaciones encontradas así como el impacto en la seguridad y posibles contramedidas para mitigar el riesgo.

Palabras clave: microcontrolador, flujo de ejecución, inyección de fallos, ciberseguridad, arranque seguro, hardware hacking, sistemas embebidos

Abstract

The TFG focuses on the study and use of hardware hacking techniques to manipulate the behavior of a microprocessor and control its execution flow. Likewise, the problems involved in using these techniques to circumvent the security of a secure boot are exposed. For this, a practical experiment of voltage faults injection in a microcontroller is carried out explaining the necessary components, the configuration of the working environment and the limitations encountered, as well as the impact on safety and possible countermeasures to mitigate the risks.

Key words: microcontroller, execution flow, fault injection, cybersecurity, secure boot, hardware hacking, embedded systems

Índice general

Índice de figuras

Índice de tablas

Índice de listados

CAPÍTULO 1
Introducción

Actualmente, la seguridad es una de las áreas de mayor importancia en sistemas empotrados y en la tecnología IoT (*Internet of Things*). Los fundamentos de la seguridad en los sistemas digitales es, en esencia, un conjunto de operaciones criptográficas ejecutadas en un hardware de confianza (*root of trust*). Sin embargo, los chips utilizados para la construcción de los sistemas de tecnología de la información son cada vez más baratos, más rápidos y más potentes y, a medida que el mundo se vuelve más dependiente de tecnologías avanzadas, la confianza implícita en el hardware se convierte en una opción insostenible. [1]

Dado que el acceso a los dispositivos es asequible, es común explorar las vulnerabilidades de los dispositivos IoT. Los chips semiconductores han sido objeto de ingeniería inversa, inserción maliciosa, ataques de canal lateral o ataques de inyección, entre otros. Estos ataques, las vulnerabilidades asociadas y las contramedidas se conoce como "Seguridad hardware".

La seguridad del hardware se ha convertido recientemente en un tema importante y cada vez se investiga en más profundidad. Sin embargo, la comprensión de la seguridad del hardware a menudo se mezcla con la informática, la electrónica, la criptografía, sistemas de comunicación, etc. A veces, requiere la construcción de equipos especializados y, por lo general, se necesita algo de práctica para adquirir habilidades.

Para probar la seguridad de un dispositivo se utilizan técnicas de hardware hacking, las cuales consisten en la manipulación o modificación de dichos dispositivos físicos para obtener información del equipo o alterar su funcionamiento. Las técnicas de hardware hacking pueden ser utilizadas por investigadores de seguridad para descubrir vulnerabilidades y mejorar la seguridad de los dispositivos, pero también por atacantes para comprometer la seguridad. [2, 3]

Una técnica de hardware hacking es la presentada en este documento: introducción de fallos de voltaje o, más comunmente conocido por su término en inglés, *voltage fault injection*. Los ataques por inyección de fallos buscan provocar un mal funcionamiento en el dispositivo atacado exponiéndolo a condiciones fuera de los límites especificados por el fabricante. Esto puede resultar en apagar o reiniciar un dispositivo, saltarse instrucciones, cambiar valores lógicos o acceder a zonas de código no autorizadas. [4]

1.1 Motivación

La seguridad informática es uno de los temas más preocupantes en la actualidad y la criptografía es esencial para la protección de la información. La idea inicial para el desarrollo del TFG era realizar un análisis comparativo entre el cifrado de direfentes ran-

somwares poniendo en práctica los conocimientos adquiridos en la asignatura de Criptografía. Esta idea quedó descartada tras hablar con mi tutor sobre la propuesta del proyecto que se ha llevado a cabo, "Inyección de fallos de voltaje para controlar el flujo de ejecución en microcontroladores".

Si bien muchas medidas e iniciativas de seguridad de las empresas se centran en peligros como el phishing, el ransomware o amenazas internas, otra forma menos conocida en la que los ciberdelincuentes pueden infiltrarse en una red es atacando el hardware. Cuando se trata de ataques basados en hardware, el riesgo aumenta significativamente debido a la falta de educación y conciencia entorno a este ámbito de seguridad y fue precisamente esa falta de información el factor decisivo para escoger el tema del proyecto.

Uno de los motivos esenciales que impulsó la propuesta de este TFG fue visibilizar la importancia de la seguridad hardware y dar a conocer las técnicas más utilizadas para atacar los dispositivos, al mismo tiempo que he adquirido conocimientos en la manipulación del hardware y las herramientas utilizadas en el proyecto.

1.2 Objetivos

El trabajo persigue los siguientes objetivos:

- Objetivo 1: Dar a conocer la importancia de la seguridad hardware.
- Objetivo 2: Aprender sobre técnicas de hardware hacking.
- Objetivo 3: Informar sobre la aplicación de estas técnicas en el arranque seguro de un dispositivo.
- Objetivo 4: Documentar el proceso de realización de un ataque por inyección de fallos de forma reproducible y asequible.

A través del primer objetivo se busca transmitir el impacto que conlleva un ataque hardware. Para ello, durante el documento se citan ataques reales que se han llevado a cabo utilizando las técnicas explicadas.

El segundo objetivo pretende despertar la curiosidad e informar al lector sobre métodos utilizados para evadir la seguridad en dispositivos físicos, inlcuyendo los fundamentos teóricos básicos para poder comprender el caso práctico realizado posteriormente.

Mediante el tercer objetivo se busca dar a conocer el concepto de arranque seguro y explicar cómo las técnicas de hardware hacking son utilizadas para evitar dicha medida de seguridad.

Finalmente, con el cuarto objetivo se pretende detallar el proceso realizado para producir un ataque por fallo de voltaje. Para que este objetivo sea cumplido, el TFG debe definir los elementos utilizados (microcontroladores y programas), incluir el proceso de configuración del entorno (conexiones entre dispositivos) y que éste sea repetible por el lector.

1.3 Estructura de la memoria

El capítulo 1, la introducción, ofrece una visión general de la importancia de la seguridad hardware. Incluye una sección de motivación donde se explica las razones que han llevado a realizar el TFG y, a continuación, se detallan los objetivos que persigue este documento y la estructura del mismo.

El capítulo 2, el estado del arte, explica la evolución tecnologíca de las técnicas de inyección de fallos y algunas herramientas que se han desarrollado para facilitar el uso de dichas técnicas. Además, analiza los trabajos realizados hasta la fecha en la UPV relacionados con el tema del TFG y justifica la aportación del presente trabajo respecto a lo ya existente.

El capítulo 3, fundamentos teóricos, aporta al lector los conocimientos base sobre las técnicas de hardware hacking y el impacto que conllevan, además de informar sobre ataques contra el arranque seguro.

El capítulo 4, análisis del problema, compara las herramientas actuales utilizadas para la inyección de fallos, y los diferentes componenetes a utilizar en el trabajo y justifica la elección de aquellos utilizados en las pruebas.

El capítulo 5, diseño de la solución, está dividido en varias secciones, una por cada prueba realizada. Cada una de ellas comienza con el cálculo de un presupuesto aproximado del material utilizado. Así mismo, se listan y muestran los componenetes y se explica, paso a paso, las conexiones entre ellos.

El capítulo 6, desarrollo de la solución, también se divide en varias secciones, correspondientes a las pruebas realizas y, en cada una, indica los aspectos más relevantes del código software ejecutado por las placas así como las configuraciones necesarias.

El capítulo 7, pruebas, se divide en secciones y explica las diferentes pruebas que se han realizado con cada uno de los sistemas configurados entre los capítulos 5 y 6, incluyendo las dificultades encontradas.

El capítulo 8, conclusiones, presenta un resumen del trabajo realizado a lo largo del documento. Se sintetizan las ideas clave y se evalúa el cumplimiento de los objetivos planteados.

El capítulo 9, trabajos futuros, sugiere posibles líneas de investigación y desarrollo que podrían continuarse para expandir y complementar el trabajo realizado en el presente documento.

El capítulo 10, glosario de términos, recopila y define algunos conceptos que pueden ser desconocidos por el lector, facilitando así la comprensión del documento.

El último capítulo, la bibliografía, expone las fuentes utilizadas para dar soporte a las definiciones, afirmaciones y análisis de todo el TFG.

Para terminar, al final de documento se encuentran los anexos, que contienen el código completo utilizado en las pruebas y otra información que puede resultar de interés.

CAPÍTULO 2
Estado del arte

En este capítulo se pretende mostrar una visión general del contexto tecnológico sobre el tema propuesto. Se exponen algunas herramientas y metodologías ya existentes y se explica la aportación del presente documento con respecto a otros trabajos realizados sobre la misma materia.

En la sección 2.1, evolución tecnológica, se explica el desarrollo de la técnica de inyección de fallos.

En la sección 2.2, aplicaciones existentes, se nombran algunas herramientas diseñadas para realizar inyecciones de fallos en microcontroladores.

En la sección 2.3, crítica al estado del arte, se comentan los diferentes trabajos relaccionados con el tema propuesto o aquellos que utilizan tecnologías parecidas a las usadas en este documento y cómo la seguridad en dichos proyectos podría verse afectada.

2.1 Evolución tecnológica

La inyección de fallos es una técnica ampliamente utilizada en la investigación y evaluación de la seguridad de sistemas embebidos, especialmente en microcontroladores. Esta técnica consiste en inducir errores controlados en un sistema para analizar su comportamiento bajo condiciones no ideales, como variaciones de voltaje, cambios de temperatura, radiación electromagnética, o perturbaciones de reloj. En este contexto, la inyección de fallos de voltaje es una de las metodologías más comunes, debido a su capacidad para alterar el flujo de ejecución de un microcontrolador de manera precisa y replicable.[3]

La técnica de inyección de fallos se remonta a la década de 1970 como un medio para probar la robustez del hardware. La técnica se convirtió rápidamente en una herramienta indispensable para los ingenieros, permitiéndoles identificar posibles debilidades en sus sistemas y verificar la corrección de sus diseños. A finales de los años 90, las técnicas de inyección de fallos comenzaron a utilizarse para evaluar la seguridad de tarjetas inteligentes (*smart cards*) en aplicaciones bancarias y de identificación. La inyección de fallos se ha utilizado para probar chips individuales y sistemas completos, dispositivos integrados y de IoT, así como circuitos de tarjetas de pago.

Con el tiempo, la técnica se ha sofisticado y extendido a una amplia gama de dispositivos embebidos, incluyendo microcontroladores utilizados en automoción, electrónica de consumo, dispositivos médicos o sistemas de control industrial. La evolución tecnológica ha permitido la creación de herramientas más precisas y económicas para llevar a cabo estos ataques, haciéndolos accesibles incluso para investigadores con recursos limitados.

2.2 Herramientas existentes

En el ámbito de la seguridad informática, existen diversas herramientas y metodologías diseñadas para realizar inyecciones de fallos en microcontroladores. A continuación, se describen algunas de las más relevantes que guardan relación con el presente trabajo.

- **ChipWhisperer**[1]: Es una plataforma de código abierto ampliamente utilizada para el análisis de seguridad de dispositivos embebidos. Se centra principalmente en ataques de análisis de energía y fallos de voltaje y reloj que interrumpen el suministro de energía o la señal del reloj de un dispositivo y causan un comportamiento no deseado del mismo.
- **VC Glitcher**: Herramienta desarrollada por Riscure[2] para realizar inyecciones de fallos mediante *glitcing* de voltaje y reloj. Fue diseñada para automatizar la búsqueda de vulnerabilidades en sistemas embebidos como microcontroladores o tarjetas inteligentes. Su versatilidad le permite ajustar múltiples parámetros, como la amplitud y la duración de los *glitches*, lo que facilita la creación de escenarios de prueba específicos y la reducción del tiempo de prueba.
- **Spider**: Es también una herramienta desarrollada por Riscure. Se utiliza para reducir la complejidad de la configuración en ataques de canal lateral o ataques por inyección de fallos además de para generar *glitches* variados que permiten pruebas muy específicas sobre los microcontroladores.
- **Raspberry Pi Glitching**: El uso de una Raspberry Pi junto con un circuito de inyección de fallos es una técnica documentada en la comunidad de seguridad de hardware. Aunque no es una solución comercial, se ha demostrado efectiva y se utiliza frecuentemente en entornos de aprendizaje y pruebas experimentales.

2.3 Crítica al estado del arte

Para la investigación del estado del arte de la temática presentada en este TFG se ha hecho uso de la base de datos RiuNet[3] , la cual recopila los Trabajos de Fin de Grado y Máster de la Universidad Politécnica de Valencia.

Tras analizar varios trabajos de la plataforma, se han encontrado dos tipos de proyectos que pueden relacionarse con la temática de este documento. Por un lado, aquellos que utilizan el mismo o un hardware parecido al presentado en este proyecto, ya que las vulnerabilidades de los dispositivos les afectan directamente y, por otro lado, aquellos proyectos que analizan los ataques por inyección de fallos.

Entre las obras que utilizan un hardware similar se encuentra la de Antonio Andreu Escrivá, "Domotización y control de una vivienda" o la de David Núñez Martínez, "Sistema de seguridad domótica para control de acceso mediante autenticación con huella dactilar". Estas obras hacen uso de placas Arduino para la domotización de una vivienda o para el acceso seguro a una estancia privada y, principalemente, utilizan esta tecnología por su bajo coste y por ser de software y hardware libre, razones por las cuales estos dipositivos son utilizados también en el proyecto presentado en este documento. [5, 6]

En el primer caso, la obra de Antonio Andreu Escrivá, la placa Arduino elegida no incorpora un módulo WiFi (aunque existen algunas placas Arduino que sí lo incluyen).

[1] https://rtfm.newae.com/
[2] https://www.riscure.com
[3] https://riunet.upv.es/

Si se introdujeran fallos controlados durante las operaciones del microcontrolador, se podría provocar una denegación de servicio o incluso acceder a la memoria interna del dispositivo y extraer datos de configuración sensibles, como la información de red WiFi. En esta obra, uno de los dispositivos configurados es un sensor de proximidad, lo cual dificultaría el acceso físico a los dispositivos hardware, inicialmente, pero sería posible la inyección de fallos electromagnéticos para provocar un mal funcionamiento del sensor.

En la obra realizada por David Núñez Martínez no existe este inconveniente. Al ser un sistema de control de acceso, es posible interactuar directamente con el hardware. En este proyecto se registran unas huellas autenticadas y, para conceder el acceso, se hacen comparaciones entre los registros y la huella detectada en el sensor. Es durante esas operaciones del microcontrolador donde pueden producirse ataques de inyección de fallos y provocar saltos de instrucciones críticas que otorgan o deniegan el acceso.

Entre las obras que analizan los ataques por inyección de fallos se encuentra la de Rafael José Boix Carpi, "Optimization of parameter settings search for a successful Fault Injection", el trabajo de Joaquín Gracia-Morán (como coautor designado), "A Comparative Study of the Effects of Intermittent Faults in a Microcontroller" o la obra de Gabriel Cobos Tello, "Fault Tolerance Techniques for FPGA-based Space Applications". [7, 8, 9]

En los dos últimos trabajos nombrados se estudian las técnicas de inyección de fallos para comprobar la robustez del hardware. Y en la obra de Rafael José Boix Carpi se propone un esquema para la automatización de la búsqueda de parámetros para lograr una inyección de fallos con éxito.

CAPÍTULO 3
Hardware hacking

Las técnicas de hardware hacking consisten en la manipulación o modificación de dispositivos físicos para alterar su funcionamiento o extraer información sensible. A diferencia del hacking de software, éste involucra la interacción directa con el hardware aprovechando debilidades físicas o características intrínsecas del dispositivo.

Este apartado pretende describir algunas de las técnicas más comunes de hardware hacking utilizadas hoy en día tanto para la investigación de la seguridad de los dispositivos como para realizar ataques dirigidos a sistemas embebidos y microcontroladores.

3.1 Tipos de ataques

Un ataque puede clasificarse en no invasivo, invasivo o semi-invasivo dependiendo de lo invasivo que sea (cúanto debe manipularse el dispositivo para violarlo) y de los parámetros que deben alterarse para causar el fallo. [4, 10]

Los ataques no invasivos consisten en observar el dipositivo o manipular señales externas pero no requieren modificaciones físicas del chip y no suelen dejar rastro. Además, el material para realizar estos ataques suele ser de bajo coste. Dentro de este grupo, los ataques pueden ser activos o pasivos dependiendo de si el ataque influye directamente en el dispositivo mediante estímulos internos o externos o si éste consiste en explotar la información que el dispositivo filtra espontáneamente. Los ataques de canal lateral son ataques no invasivos y pasivos y los ataques de inyección de fallos son no invasivos y activos.

Los ataques invasivos sí requieren la manipulación del dispositivo, ya sea agregando nuevas conexiones o quitando piezas. Se requiere un equipo sofisticado, por lo que el coste suele ser alto, y la complejidad del ataque a menudo causa que el dispositivo atacado deje de funcionar. Los ataques de ingeniería inversa son un ejemplo de esta categoría.

Los ataques semi-invasivos se sitúan en el medio de los dos mencionados anteriormente. En este tipo de ataque es necesario decapsular el chip pero la estructura interna permanece intacta. Aunque estos ataques suelen dejar rastros, en la mayoría de los casos el chip permanece en pleno funcionamiento. Los ataques por inyección de fallos ópticos se clasifican dentro de este grupo.

3.1.1. Ataques por inyección de fallos

Todo dispositivo informático requiere ciertas condiciones para funcionar con normalidad. Algunas de esas condiciones son: un suministro de voltaje estable, una señal de

reloj estable, o una temperatura de funcionamiento correcta. Si una o más de estas condiciones no se cumple, el dispositivo puede actuar de formas inesperadas y devolver resultados incorrectos.

La inyección de fallos es una técnica no invasiva y activa utilizada para inducir errores controlados en un sistema con el fin de alterar su comportamiento o extraer información confidencial. Los métodos más utilizados son los siguientes:

- **Inyección de fallos de reloj** (*Clock fault injection*).

  Implica la insercción de flancos ascendentes adicionales en el reloj de entrada del dispositivo con el objetivo de violar las restricciones de tiempo. La figura 3.1 representa gráficamente la inyección de un ciclo de reloj adicional de duración reducida.

  Una de las limitaciones es que requiere un dispositivo que utilice una entrada de reloj externa, por lo que este tipo de ataque no será efectivo contra dispositivos que utilicen osciladores internos[1]. [11, 12]

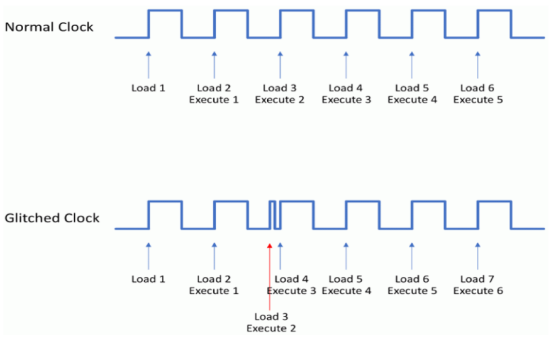

Figura 3.1: Clock fault injection

- **Inyección de fallos de voltaje** (*Voltage fault injection*).

  Implica la manipulación de la fuente de alimentación de un microcontrolador para provocar fallos en su funcionamiento normal. Las variaciones de voltaje deben controlarse con precisión y durante un tiempo determinado para tener éxito en el ataque. Reducir el voltaje durante demasiado tiempo puede provocar reinicios del chip por lo que los parámetros utilizados son esenciales.

  Uno de los problemas de este método es la presencia de componentes cuyo objetivo es especialmente mantener la alimentación en el voltaje correcto. [13, 12]

- **Inyección de fallos electromagnéticos** (*Electromagnetic fault injection*).

  En esta técnica se utilizan campos magnéticos para inducir cambios de voltaje o corriente en el objetivo sin necesidad de contacto físico. Para conseguir un campo magnético se necesita un inductor (una bobina de un material conductor) y una corriente que pasar por la bobina. Es especialmente útil en escenarios donde los dispositivos embebidos están encapsulados o el acceso físico es limitado. [11]

  La inyección de fallos electromagnéticos es un ataque muy poderoso, pero tiene la desventaja de requerir un entorno físico más complejo.

  El artículo "*Defeating Modern Secure Boot Using Second-Order Pulsed Electromagnetic Fault Injection*"[2] muestra un ataque efectivo contra una implementación de arranque seguro basado en TrustZone utilizando esta técnica.

[1] Osciladores internos y externos: https://www.youtube.com/watch?v=3Iz-etpMpUc
[2] https://www.usenix.org/system/files/conference/woot17/woot17-paper-cui.pdf

- **Inyección de fallos ópticos** (*Optical fault injection*).

  Consiste en provocar fallos en el dispositivo mediante pulsos de luz. Dichos pulsos inducen una corriente fotoeléctrica en el dispositivo generando distorsiones en los datos calculados. Para conseguir mayor precisión suelen utilizarse rayos láser. Esta técnica conlleva un alto coste del montaje y la decapsulación del chip.

3.1.2. Ataques de canal lateral

En seguridad informática, un ataque de canal lateral es un ataque pasivo y no invasivo en el cual se obtiene información filtrada espontáneamente por el propio dispositivo, como la energía, consumo, información de sincronización o emisiones electromagnéticas. Analizando esa información, un atacante puede descubrir datos confidenciales como claves criptográficas o información personal. Algunos tipos de ataques de canal lateral son los siguientes:

- **Electromagnético**: La medición de la radiación electromagnética puede utilizarse para inferir claves criptográficas u otro tipo de información. Un ejemplo del uso de esta técnica es el ataque de Van Eck[3] también conocido como TEMPEST[4].
- **Temporal**: Se mide el tiempo de computación para deducir cuando se realizan ciertas operaciones críticas, como comparaciones de claves o certificados. El tiempo total puede proporcional datos sobre el estado de un proceso o el tipo de proceso que se está ejecutando
- **Energético**: Se estudian las variaciones del consumo de energía del hardware durante la realización de cálculos u operaciones.
- **De caché**: Los sistemas modernos utilizan el almacenamiento en caché y la recuperación previa de datos para mejorar el rendimiento. Un atacante puede abusar de estos sistemas para acceder a información que debería bloquearse.
- **Acústico**: Se miden los sonidos producidos por los componenetes electrónicos durante un cálculo.
- **Análisis de potencia diferencial**: Se aprovechan las variaciones de consumo energético para obtener información sobre algortimos criptográficos. Mediante el uso de métodos estadísticos se derivan claves secretas con la información recogida.

Los ataques de canal lateral son más comunes hoy en día debido a dos factores principalmente. Por un lado, a la evolución de los equipos de medición, que hacen posible la recopilación de datos extremadamente detallados, y, por otro lado, al avance de la informática y el aprendizaje automático que facilitan la recopilación y comprensión de los datos. Algunos ejemplos son el ataque *Spectre*[5] en 2018, *Meltdown*[6] (una variación del anterior) o *Indirector*[7], este mismo año (2024).

Defenderse de los ataques de canal lateral es complicado ya que son difíciles de detectar en acción y no suelen dejar rastro. Estos ataques pueden ser eficaces contra sistemas aislados físicamente, contra máquinas virtuales o incluso en entornos de computación en la nube donde el atacante y el objetivo comparten el mismo hardware físico. [14, 15]

[3] https://es.wikipedia.org/wiki/Interferencia_de_Van_Eck

[4] https://www.ccn-cert.cni.es/publico/seriesCCN-STIC/series/400-Guias_Generales/401-glosario_abreviaturas/index.html?n=910.html

[5] https://en.wikipedia.org/wiki/Spectre_(security_vulnerability)

[6] https://en.wikipedia.org/wiki/Meltdown_(security_vulnerability)

[7] https://unaaldia.hispasec.com/2024/07/intel-se-enfrenta-a-un-ciberataque-similar-a-spectre-llamado-indirector-que-filtra-datos.html

3.1.3. Ingeniería inversa

La ingeniería inversa en seguridad hardware es el proceso de desmontar un dispositivo para analizar sus componentes, el diseño y la estructura con el objetivo de comprender cómo funciona. El proceso de ingienería inversa incluye tanto la decapsulaión de chips como la extracción de firmware.

Cuando se trata de ingeniería inversa en microcontroladores se requiere un análisis tanto estructural como del código de programa para poder comprender cómo funciona el dispositivo. Esta técnica permite a los ingenieros obtener información sobre el funcionamiento de un sistema, identificar vulnerabilidades potenciales, mejorar el desarrollo y la seguridad de productos, o incluso desarrollar otros dispositivos completamente nuevos.

Utilizando la ingeniería inversa se puede crear una réplica precisa del dispositivo e identificar cualquier defecto en el diseño. Esto resulta beneficioso para los desarrolladores y analistas de seguridad ya que permite encontrar puntos pontenciales de fallos y arreglarlos. Sin embargo, la ingeniería inversa puede ser utilizada por cualquiera que consiga tener acceso al código o al hardware y posea los conocimientos necesarios para realizar un análisis en busca de fallos. Una de las desventajas del hardware libre es que cualquier persona puede analizar el diseño en busca de fallos y puede ser más susceptible a vulnerabilidades de seguridad.

La plataforma *Semantic scholar*[8] recoge publicaciones académicas entre las cuales se encuentran casos efectivos de inyección de fallos de voltaje [16], de fallos ópticos [17], estudios de ataques de canal lateral sobre el consumo de energía [18], o sobre cómo esta técnica se utiliza para averiguar claves criptográficas [19]. También se encuentran trabajos relaccionados con la ingenieria inversa aplicada al hardware [20] y la importancia de investigar y mejorar los dispositivos para garantizar la seguridad [21].

3.2 Efectos de los *glitches*

Debido a la inyección de fallos, el comportamiento de un dispositivo puede verse afectado de varias maneras. Los ataques pueden influir en los datos almacenados en los registros, la caché o las operaciones realizadas por la CPU. Los resultados de los fallos no siempre se pueden predecir pero suelen manifestarse de las siguientes formas [22]:

- **Manipulación de instrucciones**.

  Consiste en modificar instrucciones al provocar los fallos generando comportamientos inesperados. Por ejemplo, el cambio de un bit en un código de operación de una instrucción puede convertir una suma en una operación de resta.

- **Omisión de instrucciones**.

  Es un caso especial de la manipulación de instrucciones cuyo resultado es convertir la instrucción a algo que no va a utilizarse por el programa como puede ser una instrucción nop. Los objetivos típicos suelen ser instrucciones de salto o comparaciones.

- **Corrupción de memoria**.

  Los fallos introducidos afectan a valores cargados de memoria o de registros, lo cual puede provocar efectos inesperados en la ejecución del programa.

[8] https://www.semanticscholar.org/

- **Obtención de datos incorrectos**.

  Si el fallo se inyecta durante el proceso de lectura de una palabra es posible invertir algunos de los bits leídos o que la palabra completa se lea como unos o ceros. Cuando los datos críticos de seguridad se corrompen de esta manera, el dispositivo afectado puede volver a un estado menos seguro.

- **Denegación de Servicio**.

  A diferencia de los anteriores, el objetivo de un ataque puede no ser la obtención de información o alterar el flujo de un programa, sino conseguir inhabilitar el dispositivo. Si los fallos inyectados provocan el reinicio constante del microcontrolador, éste dejará de proporcionar el servicio esperado.

3.3 Arranque seguro

El arranque seguro es un proceso mediante el cual un microcontrolador verifica la integridad y autenticidad del firmware antes de ejecutarlo. Es decir, se ejecuta un código para comprobar que el firmware no ha sido alterado antes de permitir el inicio. El código del arranque seguro es el único que se ejecuta en el reinicio y es inalterable.

La verificación se logra mediante la validación de firmas digitales de los componentes cargados durante el proceso de arranque. Si algún componente no se autentica durante el proceso, el arranque se detiene. El proceso de autenticación es parecido a la PKI (*Public Key Infrastructure*) utilizada en los navegadores. La diferencia es que la raíz de confianza en el arranque seguro está compuesta de código y clave inmutables. [23]

Las técnicas de inyección de fallos son utilizadas también para comprometer la seguridad durante el proceso de arranque interfiriendo en la verificación de firmas, permitiendo la ejecución de código no autorizado o incluso eludiendo completamente el arranque seguro.

Exiten pruebas de concepto que se han realizado utilizando varias de las técnicas de inyección de fallos comentadas anteriormente. Por ejemplo, el trabajo de Rick Housley [24], que genera fallos electromagnéticos en una arquitectura ARM, con una herramienta novedosa, para conseguir sobrepasar el arranque seguro basado en TrustZone. Otras pruebas de concepto a destacar son las presentadas en ponencias como Black Hat, en concreto y teniendo relacción con el presente trabajo, aunque en más profundidad y aplicado al arranque seguro, "Vlind Glitch: Blind VCC Glitching Technique to Bypass the Secure Boot of Qualcomm MSM8916 Mobile SoC"[9] y "Bypassing Secure Boot Using Fault Injection"[10].

3.4 Medidas de seguridad

Las técnicas de hardware hacking son herramientas poderosas en la evaluación de la seguridad de sistemas embebidos y microcontroladores pero, al mismo tiempo, proporcionan una amplia gama de métodos para comprometer a los dispositivos a aquellos con objetivos no éticos. Comprender y mitigar estos riesgos es esencial para el desarrollo de sistemas más seguros y resistentes.

Mediante un ataque hardware, el objetivo puede ser tanto hardware como software. Aunque el método de ataque sea a través de un dispositivo físico, el propósito puede ser,

[9] https://www.youtube.com/watch?v=zQKYazIlAgM
[10] https://www.youtube.com/watch?v=0fZdL3ufVOI

por ejemplo, alterar un programa. Por lo tanto, las medidas de seguridad deben aplicarse tanto en software como en hardware.

Algunos métodos para la mitigación de ataques hardware son los siguientes:

- **Evitar acceso a pines**.

  Los puertos de prueba en placas de producción (JTAG, SPI, UART) deben desaparecer si no son necesarios o no van a ser usados por el cliente final, ya que podrían ser aprovechados por los especialistas en hardware hacking.

  La lectura directa de señales en los pines de un chip puede ser evitada mediante el uso de resina epoxy cubriendo dichas patillas. Si se intenta quitar la resina para hacer uso de los pines y atacar el dispositivo, la placa se daña impidiendo establecer las comunicaciones. [25]

- **Protección física**.

  Aislar los dispositivos hardware críticos o usar encapsulados que dificulten el acceso físico al chip o la exposición a técnicas como láseres o ataques de luz intensa. Además, incorporar sensores que detecten intentos de acceso físico no autorizado, como la apertura del encapsulado.

- **Doble verificación**.

  Consiste en realizar la misma comprobación varias veces. Si el *glitch* provoca un fallo en la primera comprobación, la siguiente detectará el fallo. El pseudocódigo siguiente muestra un ejemplo de doble verificación:

```
1        if x == y then
2            if x != y then
3                return error
4            else
5                print "Access granted"
6        else
7            print "Access denied"
```

Listing 3.1: Pseudocódigo doble verificación

  Debe realizarse depuración del código si se aplica esta medida ya que los compiladores modernos tienden a optimizar el código y podrían realizar la comprobación solo una vez. Incluso forzar a que el compilador compruebe el valor de una variable declarándola *volatile* puede no ser suficiente para evitar las optimizaciones.

- **Comprobación de integridad del código**.

  Implementar verificaciones de integridad del código para detectar modificaciones no autorizadas o errores inducidos. El pseudocódigo siguiente muestra un ejemplo de comprobación del checksum.

```
1        var result = VALUE
2        var resultChecksum = ~VALUE
3        ...
4        if result XOR resultChecksum != 0xFF then
5            return error
```

Listing 3.2: Pseudocódigo comprobación checksum

- **Retrasos aleatorios**.

  Consiste en introducir paradas de duración aleatoria con el objetivo de evitar los ataques basados en el tiempo. Las paradas pueden generarse mediante bucles como el mostrado a continuación o provocando que la CPU se salte un ciclo de reloj.

```
1         var randomValue = random()
2         while randomValue > 0 do
3             randomValue--
```

Listing 3.3: Psudocódigo retrasos aleatorios

- **Monitorización**.

  Añadir condensadores para suavizar los *glitches* o implementar circuitos de monitoreo para detectar cambios anómalos en el voltaje o la frecuencia de reloj. Si se detectara, por ejemplo, una caída abrupta (un *glitch*), una forma de respuesta del sistema podría ser reiniciar el dispositivo.

  De la misma forma, pueden incorporarse sensores de temperatura y, cuando la temperatura supere un umbral seguro, el sistema puede apagarse o activar mecanismos de protección.

- **Introducción de ruido**.

  La relación entre la señal y el ruido producidos por un dispositivo determina la tasa de éxito de un ataque de canal lateral: cuanto menor sea la claridad de la señal producida, más mediciones serán necesarias.

  Para aumentar la cantidad de ruido se puede utilizar un generador de ruido basado en hardware o software pero, cabe señalar, que esta medida solo dificulta el ataque ya que el atacante puede ser capaz de eliminar el ruido nuevamente. [26]

Algunas de las medidas mencionadas pueden no ser lo suficientemente robustas por sí solas, pero es la combinación de varias medidas lo que hace al sistema más resistente y seguro.

Un aspecto a tener en cuenta, además de las medidas a implementar, es cómo debe reaccionar un dispositivo al detectar un posible fallo y el impacto en su rendimiento. Esto depende mucho de la aplicación y del nivel de seguridad requerido. Por ejemplo, una tarjeta bancaria (tarjeta inteligente o *smart card)* debe dejar de funcionar por completo después de detectarse un número bajo de fallos, pero un dispositivo con requisitos de alta confiabilidad puede continuar funcionando incluso si ocurren fallos. En este caso, se podría bloquear el acceso a operaciones criptográficas durante algún tiempo o al menos registrar el incidente.

Aunque el área de la seguridad hardware debe investigarse en mayor profundidad, ya existen regulaciones que abordan medidas de seguridad en componentes de sistemas de automatización y control industrial como la IEC 62443-4-2[11], publicada en 2019, que se enfoca en los requisitos que deben cumplirse para garantizar sistemas fiables y seguros.

[11] https://webstore.iec.ch/en/publication/34421

CAPÍTULO 4
Análisis del problema

En este capítulo se estudian algunas de las herramientas y componentes existentes para realizar inyecciones de fallos de voltaje y se escogen los materiales a utilizar en el proyecto haciendo un balance entre las ventajas y desventajas de cada uno.

En la sección 4.1, identificación y análisis de soluciones posibles, se exponen varias opciones para realizar el proyecto, tanto de microcontroladores, como herramientas para realizar el ataque.

En la seccion 4.2, solución propuesta, se justifica la metodología elegida para llevar a cabo el proyecto y la aportación con respecto a las tecnologías existentes.

4.1 Identificación y análisis de soluciones posibles

4.1.1. Chipwhisperer

Es una plataforma de código abierto ampliamente utilizada para el análisis de seguridad de dispositivos embebidos. Está diseñada específicamente para realizar ataques de análisis de potencia y fallos de voltaje y reloj, lo que proporciona una precisión y control superior a soluciones más genéricas como Arduino.

La plataforma incluye una amplia variedad de herramientas de software para análisis de señales, facilitando la interpretación de los resultados de los experimentos sin necesidad de configurar o desarrollar herramientas adicionales y cuenta con una comunidad activa y una extensa documentación, lo que facilita la resolución de problemas. Esta herramienta se ha convertido en una referencia en el campo de la seguridad hardware.

Una de las principales desventajas del ChipWhisperer es su elevado coste. Además, aunque el ChipWhisperer facilita la inyección de fallos y el análisis de señales, su configuración inicial y la curva de aprendizaje son más complejas, sobre todo si se quiere modificar o personalizar el entorno de ataque para un caso específico. [27]

4.1.2. Arduino

Arduino es una plataforma de código abierto basada en hardware y software de uso sencillo y, por ello, se ha utilizado en miles de proyectos y aplicaciones diferentes para construir dispositivos digitales e interactivos que puedan controlar objetos de la vida real.

Respecto al hardware, las placas Arduino implementan microcontroladores de la marca Atmel. Debido a que este tipo de placas se caracterizan por la integración de múltiples

entradas digitales y analógicas, son compatibles con una gran diversidad de componentes y periféricos.

Para la programación software, Arduino cuenta con una plataforma llamada Arduino IDE (Entorno de Desarrollo Integrado), el cual es un entorno de programación multiplatoafroma, es decir, se puede utilizar en los sistemas operativos más comunes hoy en día. La plataforma es fácil de usar para principiantes, pero lo suficientemente flexible como para que los usuarios avanzados también lo aprovechen.

En resumen, entre las ventajas que proporciona Arduino frente a otros sistemas destacan:

- Un bajo coste.

- Fácil de utilizar.

- Plataforma abierta.

- Gran variedad de placas.

- Hardware y software ampliable y de código abierto.

- Gran variedad de aplicaciones y usos.

4.1.3. STM32

El STM32 es una familia de microcontroladores basada en la arquitectura ARM Cortex-M, ampliamente utilizada en aplicaciones embebidas por su potencia, versatilidad y soporte de desarrollo.

Los microcontroladores STM32 ofrecen un rendimiento significativamente superior en comparación con el ATmega328p, lo que puede permitir un análisis más detallado de los efectos de la inyección de fallos. Además, cuenta con herramientas de desarrollo más avanzadas y un mejor soporte para depuración.

Sin embargo, estos microcontroladores suelen requerir una configuración inicial más compleja. Esto incluye la configuración del entorno de desarrollo, la selección del modelo adecuado dentro de la familia STM32 y la implementación de código más avanzado, es decir, el uso de STM32 implica una curva de aprendizaje más pronunciada en comparación con el Arduino. Además, aunque los microcontroladores STM32 son generalmente asequibles, pueden ser más costosos que otros chips. [28]

4.1.4. Transistor MOSFET

En la inyección de fallos de voltaje, los MOSFETs de canal N y canal P juegan roles cruciales, pero tienen características diferentes que los hacen más o menos adecuados para ciertos tipos de inyección de fallos. [29, 30]

MOSFET CANAL N

- **Conducción**: Para que un MOSFET de canal N conduzca, es necesario aplicar un voltaje positivo en la compuerta con respecto a la fuente. Esto hace que sea más fácil de controlar cuando la fuente está conectada a tierra (circuito común).

- **Rendimiento**: Los MOSFETs de canal N generalmente tienen un rendimiento mejor en términos de velocidad de conmutación y eficiencia energética.

- **Estructura**: En un MOSFET de canal N, los electrones son los portadores de carga mayoritarios, lo que suele resultar en una menor resistencia al paso de corriente y, por lo tanto, menor pérdida de potencia comparado con los MOSFETs de canal P.

- **Aplicación en inyección de fallos**: En la inyección de fallos de voltaje, los MOSFETs de canal N son útiles para controlar la conexión a tierra o al potencial negativo del circuito, lo que los hace más eficaces para cortocircuitar o interrumpir el suministro de energía en una sección del circuito.

MOSFET CANAL P

- **Conducción**: Para que un MOSFET de canal P conduzca, es necesario aplicar un voltaje negativo en la compuerta con respecto a la fuente. Esto lo hace más adecuado para aplicaciones donde la fuente está en un potencial más alto que el drenador (típicamente en circuitos de alimentación positiva).
- **Rendimiento**: Los MOSFETs de canal P suelen ser menos eficientes en términos de velocidad de conmutación y consumo de energía en comparación con los de canal N.
- **Estructura**: En un MOSFET de canal P, los agujeros son los portadores de carga mayoritarios, lo que suele llevar a una mayor resistencia al paso de corriente y, por lo tanto, mayor pérdida de potencia comparado con los MOSFETs de canal N.
- **Aplicación en inyección de fallos**: Los MOSFETs de canal P se utilizan para desconectar la alimentación positiva o para inyectar un voltaje positivo en el circuito. Son útiles cuando se necesita interrumpir o manipular la conexión al suministro de voltaje positivo.

4.2 Solución propuesta

Tras analizar diferentes herramientas y las ventajas y desventajas de cada una, para la realización de este TFG se ha optado por la utilización de una placa Arduino Uno equipada con un microcontrolador Atmega328, un transistor MOSFET de canal N y una placa Arduino Mega 2560. Esta elección se justifica por varias razones:

- **Accesibilidad y coste**: La plataforma Arduino roporciona hardware y software libre además de un entorno de desarrollo (Arduino IDE) sencillo de utilizar. Permite la programación en distintos sistemas operativos y su bajo costo es una ventaja significativa frente a herramientas comerciales más costosas como ChipWhisperer. Además, es posible separar el chip de la placa Arduino Uno y esto facilitará las pruebas y las conexiones entre dispositivos.
- **Popularidad**: Arduino es una de las plataformas de desarrollo más populares en educación e investigación. Debido a la variedad de placas existentes y, como ya se ha comentado, al fácil acceso y la sencillez de uso, este hardware es utilizado en multitud de proyectos. Es por eso que dar a conocer los posibles vectores de ataque hacia estos dispositivos es importante.
- **Compatibilidad**: El microcontrolador Atmega328 es lo suficientemente versátil para soportar la inyección de fallos de voltaje, permitiendo realizar experimentos controlados replicables y escalables a otros microcontroladores de la misma familia.

Respecto al transistor utilizado, se ha optado por utilizar un MOSFET de canal N ya que es más adecuado para inyectar fallos que involucren cortar o manipular conexiones a tierra o voltajes negativos, debido a su baja resistencia de encendido, alta velocidad de conmutación y mejor eficiencia.

Además de las razones mencionadas, en la prueba de concepto realizada no se utilizan las herramientas ya diseñadas que facilitan la inserción de fallos sino que se pretende realizar desde cero y comprobar si es posible generar este tipo de ataque entre microcontroladores con características similares. El trabajo también recopila en un solo documento información sobre las técnicas de hardware hacking más utilizadas ampliando el conocimiento del lector a otros vectores de ataque además del documentado en el caso práctico.

En resumen, las herramientas elegidas permiten cumplir con los objetivos del TFG de manera efectiva, aprovechando las ventajas de una plataforma económica y bien documentada, sin sacrificar la rigurosidad y relevancia de la investigación en el campo de la inyección de fallos.

CAPÍTULO 5
Diseño de la solución

En este capítulo se presentan los componentes utilizados en las pruebas. Para cada una de ellas, se describe la arquitectura detallando cada uno de los componentes que conforman el sistema y la interacción entre ellos. Además, se desglosa el coste asociado al desarrollo de cada prueba y se calcula un presupuesto aproximado.

5.1 Primera prueba. Código desconocido

La primera prueba realizada se basa en el artículo de Christoffer Claesson [31] y trata de replicar el ataque por inyección de fallos de voltaje presentado en él. Lo interesante de realizar esta prueba es que no se conce el código exacto ejecutado por el arduino target.

5.1.1. Presupuesto

Los componentes utilizados se muestran en la figura 5.1 y son los siguientes:

- Arduino Uno
- Microprocesador ATmega328p
- Arduino Mega 2560
- Protoboard
- MOSFET 2N7000
- Cables puente

Guiándose por los precios de la página oficial de Arduino[1] y Microchip[2] el presupuesto para realizar esta prueba es el mostrado en la Tabla 5.1.

La placa Arduino incorpora un microcontrolador ATmega328p extraíble fácilmente por lo que no es necesario un microcontrolador adicional.

Siguiendo los precios mostrados en la Tabla 5.1, el presupuesto global para la prueba asciende a los 89,02 €. Es un precio aproximado y basado en el importe de las páginas anteriormente mencionadas pero es posible obtener algunos materiales más baratos a través de otras compañías o reducir la cantidad, por ejemplo, de los cables puente, ya que tantos no son necesarios.

[1] https://www.arduino.cc/
[2] https://www.microchip.com/

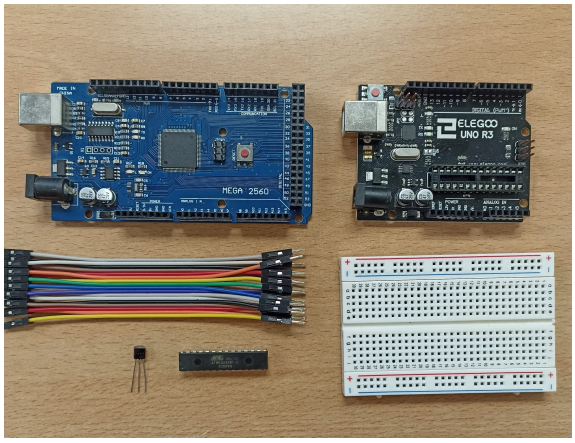

Figura 5.1: Materiales primer caso de prueba

Componenete	Cantidad	Coste
Arduino Uno Rev3	1	29,06 €
Microprocesador ATmega328p	1	-
Arduino Mega 2560 Rev3	1	50,78 €
Breadboard	1	4,86 €
MOSFET 2N7000	1	0,45 €
Cables puente	40	3,87 €

Tabla 5.1: Presupuesto primer caso de prueba

5.1.2. Arquitectura del sistema

Para poder llevar a cabo esta prueba se necesitan dos configuraciones diferentes. Por un lado, el Arduino Mega debe utilizarse como programador ISP [32] para programar el ATmega328p y, por otro lado, una vez el Arduino Uno contiene el programa correspondiente, hay que realizar las conexiones entre todos los componentes.

Arquitectura Programador ISP

Todos los microcontroladores AVR disponen de cuatro pines específicos para ser programados, MISO, MOSI, SCK y RESET. Estos pines, juntos al positivo y al negativo de la alimentación, sirven para conectar un programador externo. En la figura 5.2 se muestran los pines en un Arduino Uno.

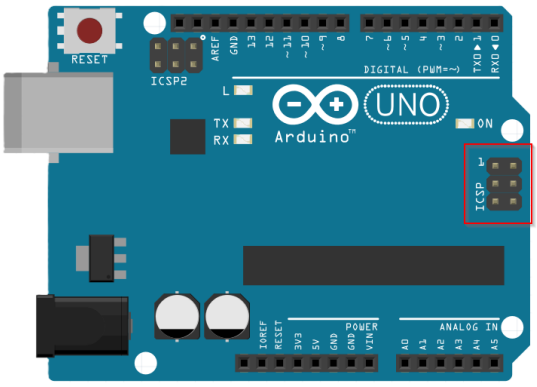

Figura 5.2: Pines ISP

Para esta prueba, el Arduino Mega será el programador y el Arduino Uno la placa a programar, de aquí en adelante referenciada como *Target*. Se ha consultado la página oficial de Arduino sobre los programadores ISP y las conexiones entre placas [32] y el esquema final es el mostrado en la figura 5.3 y en la figura 5.4.

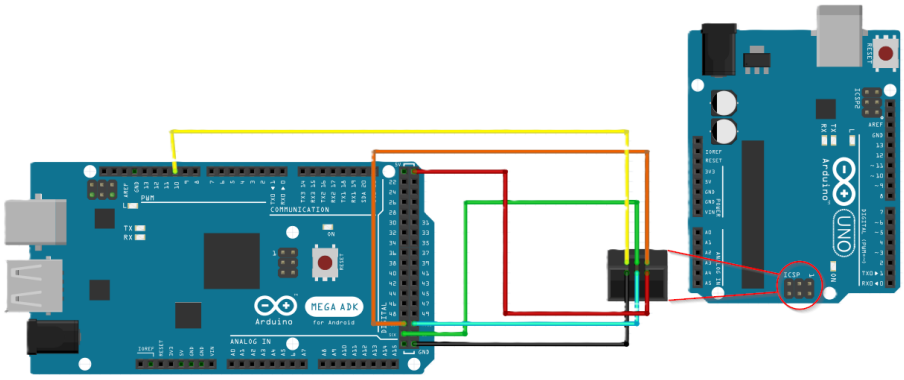

Figura 5.3: Esquema conexiones ISP

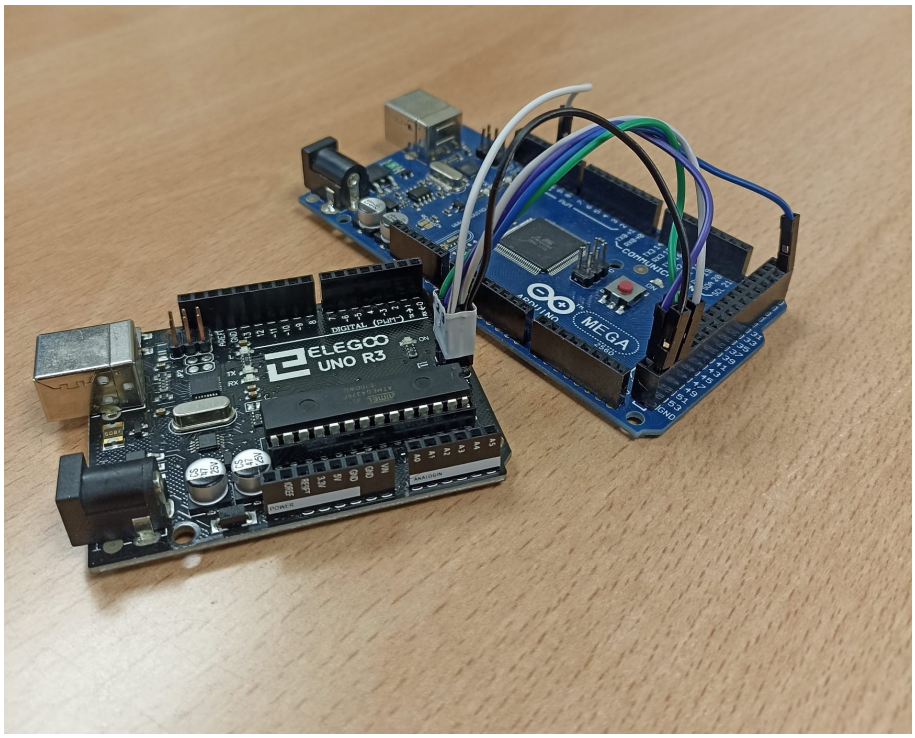

Figura 5.4: Conexiones ISP

Arquitectura pruebas

Para realizar la inyección de fallos, el microcontrolador se separa del Arduino Uno y se coloca en la protoboard. De esta manera, se evita que ningún condensador de la placa interfiera en las variaciones de voltaje.

Los pines que se deben conectar, como mínimo, son los pines 2 (RX), 3 (TX), 7 (VCC), 8 (GND), 9 y 10 (reloj), los cuales manejan la comunicación en serie con el microcontrolador, la energía y las operaciones básicas. También se ha conectado el pin 1 (RESET) que es necesario para enviar código al Arduino y que éste se reinicie. La figura 5.5 muestra la

funcionalidad de los pines del microcontrolador y en la figura 5.6 vemos cómo deben quedar las conexiones.

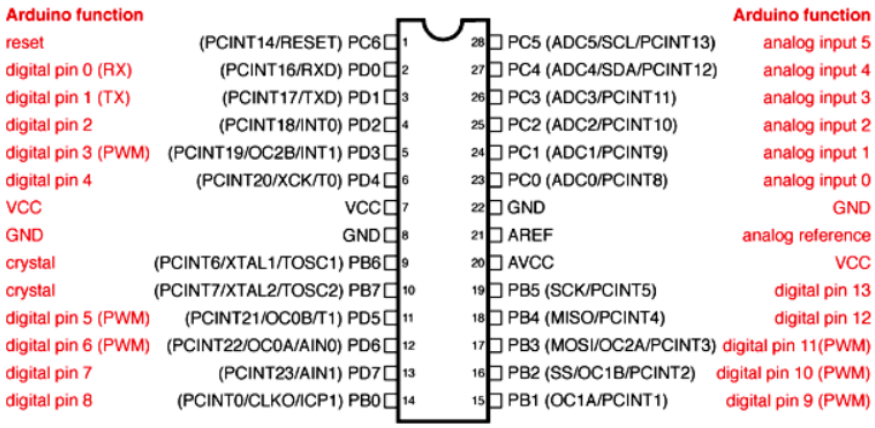

Figura 5.5: ATmega pines

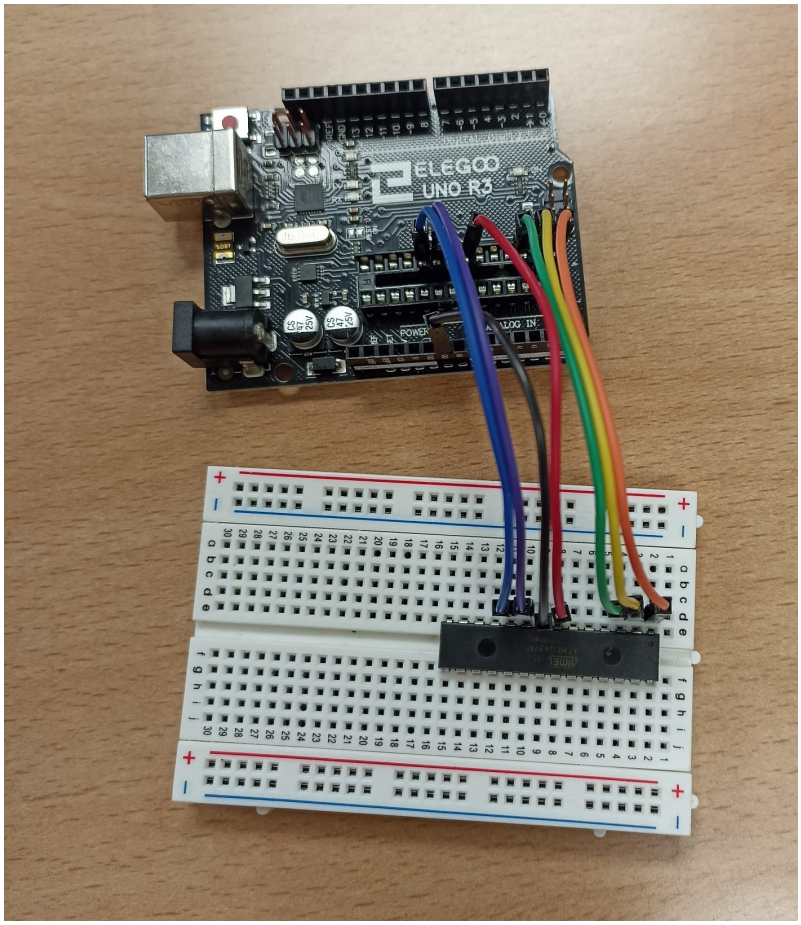

Figura 5.6: ATmega328p conexiones básicas

Una vez el microcontrolador se ha configurado en la protoboard, se debe conectar el MOSFET y el Arduino Mega, de aquí en adelante referenciado como *Host*. En esta prueba se realizará una subida de tensión controlada a través del MOSFET. Para ello, se ha conectado GND del Host al pin *Source* del MOSFET, GND del Target al pin *Drain*, y al *Gate*, un pin digital del Host que servirá para controlar el funcionamiento del componente. La

fuente de alimentación será el PC al que se conecta el Host, y el Target se alimentará con VCC del Host. En la figura 5.7 se muestra cómo debe quedar el sistema tras realizar todas las conexiones.

Para poder visualizar los datos enviados por el Target por el puerto serie en la consola, ha de conectarse también el pin de transmisión del Target (TX) con el pin de recepción del Host (RX0).

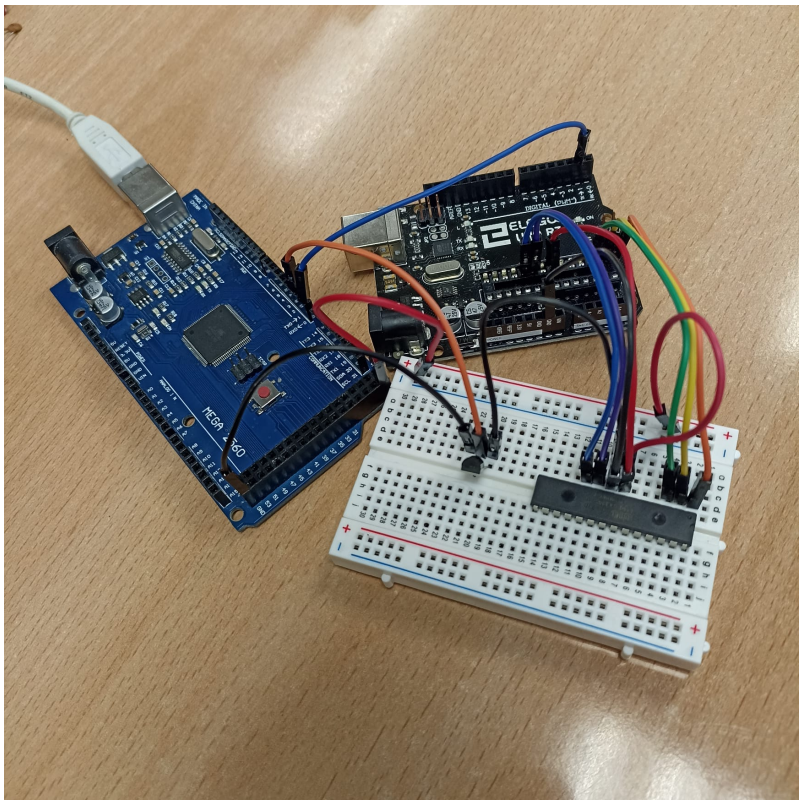

Figura 5.7: Arquitectura completa primera prueba

5.2 Segunda prueba. Estructura if-else

Esta prueba es similar a la anterior pero, en este caso, sí se conoce el código ejecutado por el arduino target (listado C.1). Los materiales utilizados son los mismos que en la primera prueba, pero se ha añadido otro Arduino Mega para la recepción de los datos enviados por el Target.

5.2.1. Presupuesto

Los componentes utilizados se muestran en la figura 5.8 y son los siguientes:

- Arduino Uno
- Microprocesador ATmega328p
- Arduino Mega 2560 (x2)
- Protoboard
- MOSFET 2N7000

- Cables puente

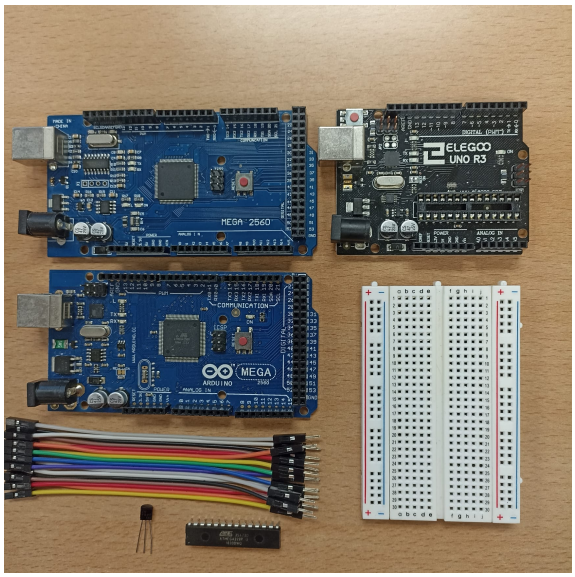

Figura 5.8: Materiales segundo caso de prueba

La tabla 5.2 muestra los precios de cada componente guiándose por los importes de las páginas de Arduino y Microchip, tal y como se hizo en el presupuesto para el primer caso de prueba (tabla 5.1). Los precios son los mismos pero la cantidad de Arduino Mega es 2, en este ejemplo.

Teniendo en cuenta que el Arduino añadido no es necesario, ya que se puede modificar el código para que un mismo Arduino produzca el *glitch* y lea los datos enviados por el Target, el presupuesto para la realización de esta prueba puede variar entre 89,02€ y 139,90 €. Es un precio aproximado ya que es posible obtener algunos materiales más baratos a través de otras compañías.

Componenete	Cantidad	Coste
Arduino Uno Rev3	1	29,06 €
Microprocesador ATmega328p	1	-
Arduino Mega 2560 Rev3	2	101,56 €
Breadboard	1	4,86 €
MOSFET 2N7000	1	0,45 €
Cables puente	40	3,87 €

Tabla 5.2: Presupuesto segundo caso de prueba

5.2.2. Arquitectura del sistema

Tal y como se ha hecho para la primera prueba, se extrae el microcontrolador del Arduino Uno y se coloca en la protoboard realizando las conexiones necesarias para su funcionamiento (figura 5.6).

El MOSFET se conecta de la misma manera que anteriormente, a GND del Target y GND del Host, y es controlado por el Host a través de un pin digital conectado con *Gate*. El Arduino Mega que actúa como Host, conecta VCC a la protoboard para alimentar al Target. El Arduino Mega que actúa como Reader (receptor de los datos del Target), se

conecta también al PC, siendo ésta su fuente de alimentación, y su pin RX0 recibe los datos enviados por el Arduino Uno a través del puerto serie (TX).

El resultado final del sistema se muestra en la figura 5.9.

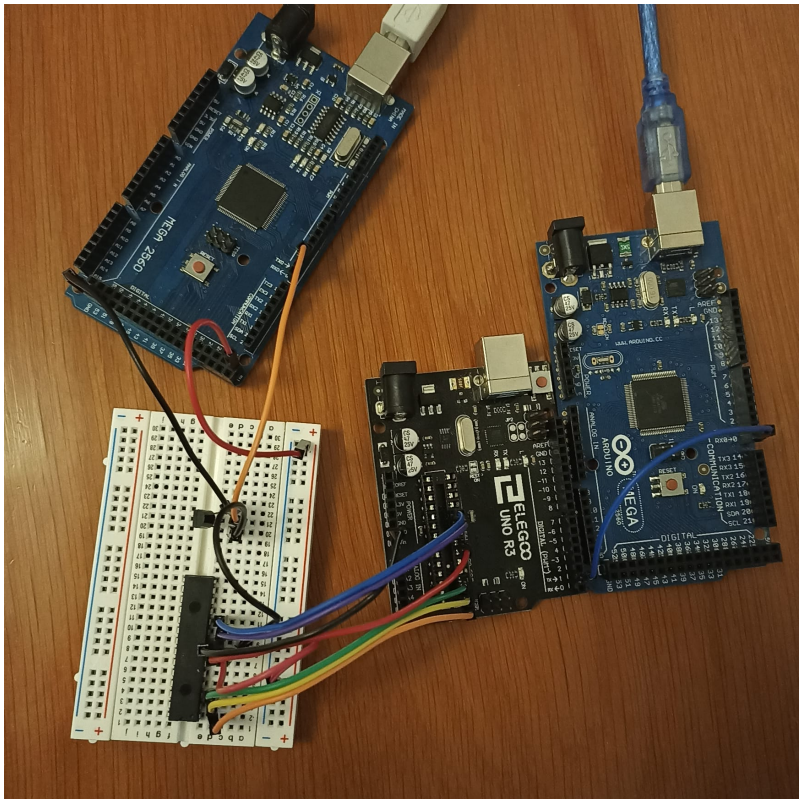

Figura 5.9: Arquitectura completa segunda prueba

5.3 Tercera prueba. Bucle for

La arquitectura en esta prueba es similar a las anteriores, el único cambio es que se utiliza el LED incorporado en el Arduino Uno y para ello es necesario conectar el pin 19 del microcontrolador, que se corresponde con el pin digital 13 (figura 5.5). El LED será la referencia para saber si los fallos inyectados tienen éxito ya que en este ejemplo no se transmitirán datos por el puerto serie.

5.3.1. Presupuesto

Los componentes utilizados son los mostrados en la figura 5.1, los de la primera prueba, y, por lo tanto, el presupuesto también es el mismo, 89,02 € (tabla 5.1). Los materiales son:

- Arduino Uno
- Microprocesador ATmega328p
- Arduino Mega 2560
- Protoboard
- MOSFET 2N7000

- Cables puente

5.3.2. Arquitectura del sistema

Tal y como ya se ha hecho en los apartados anteriores, se extrae el microcontrolador del Arduino Uno y se coloca en la protoboard realizando las conexiones necesarias para su funcionamiento, en este caso, incluyendo el pin del LED (figura 5.10).

Una vez el microcontrolador se ha configurado en la protoboard, se debe conectar el MOSFET y el Arduino Mega (Host) quedando como en la figura 5.11.

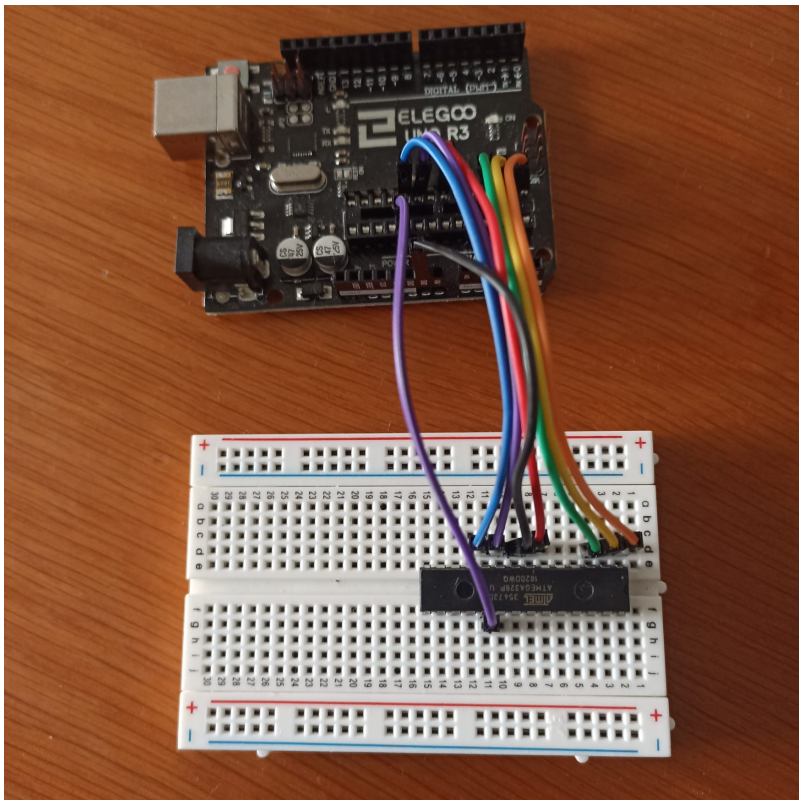

Figura 5.10: ATmega328p conexiones básicas y LED

5.4 Cuarta prueba. Bucle for sin Arduino Uno

La arquitectura de esta prueba es diferente a las anteriores ya que se prescinde del Arduino Uno y se configura el ATmega328p directamente en la protoboard con todos los componentes necesarios para su funcionamiento. El sistema final es muy similar a los ya presentados anteriormente pero no utiliza la placa del Arduino Uno, sino tan solo el microcontrolador, que es sobre el que se producen los fallos. Este escenario es intersante ya que hasta el momento todas las pruebas han sido dependientes de las conexiones con el Arduino Uno, pero este microcontrolador es utilizado en otras placas.

5.4.1. Presupuesto

Los componentes utilizados se muestran en la figura 5.12 y son los siguientes:

- Microprocesador ATmega328p

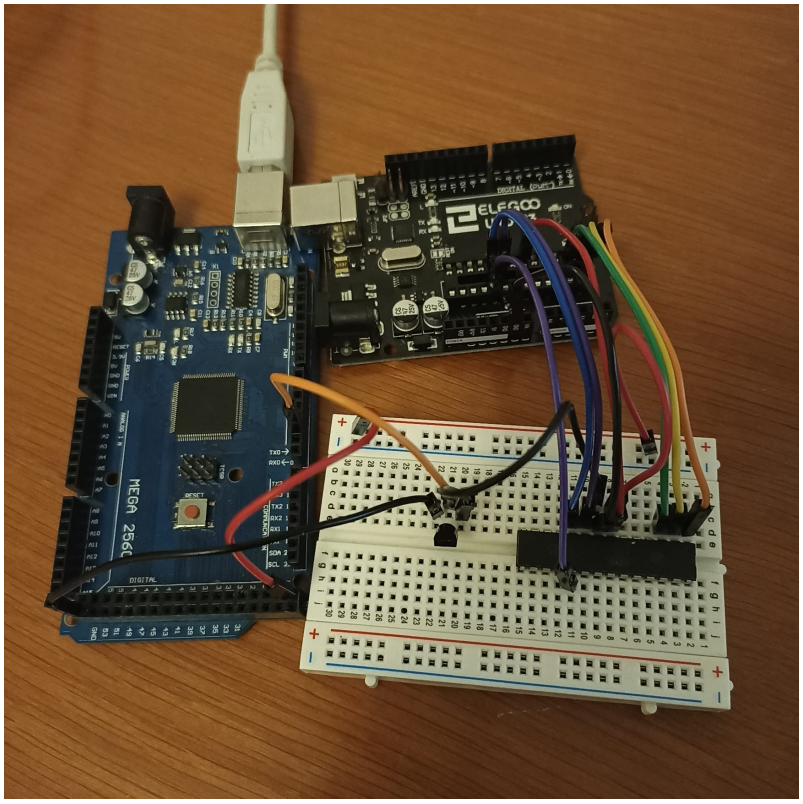

Figura 5.11: Arquitectura completa tercera prueba

- Arduino Mega 2560
- Protoboard
- MOSFET 2N7000
- Cables puente
- Oscilador de cristal de 16 MHz
- Condensadores 15 p (x2)
- Resistencias (100 Ω, 330 Ω, 4,7k Ω)
- LED

Los precios de los componentes se muestran en la tabla 5.3. El presupuesto asciende a los 66,68 €. La realización de esta prueba conlleva menos gasto en los materiales pero requiere más conexiones y trabajo en el procedimiento.

5.4.2. Arquitectura del sistema

En esta prueba el microcontrolador ATmega328p debe ser programado a través del Arduino Host. Para ello, es necesario conectar los cuatro pines de la conexión ISP (MOSI, MISO, RESET, SCK), que se corresponden con los pines 1, 17, 18 y 19 del chip.

Para el funcionamiento básico del chip, es necesario incluir en la arquitectura un oscilador conectado a VCC y GND (pines 7 y 8) y en paralelo a dos condensadores. Además se ha incluido un led para poder realizar el mismo experimento que en la prueba anterior y tener visibilidad de los *glitches*. Las conexiones entre el MOSFET, el Arduino y el chip, son las mismas que anteriormente. El esquema final es el mostrado en la figura 5.13.

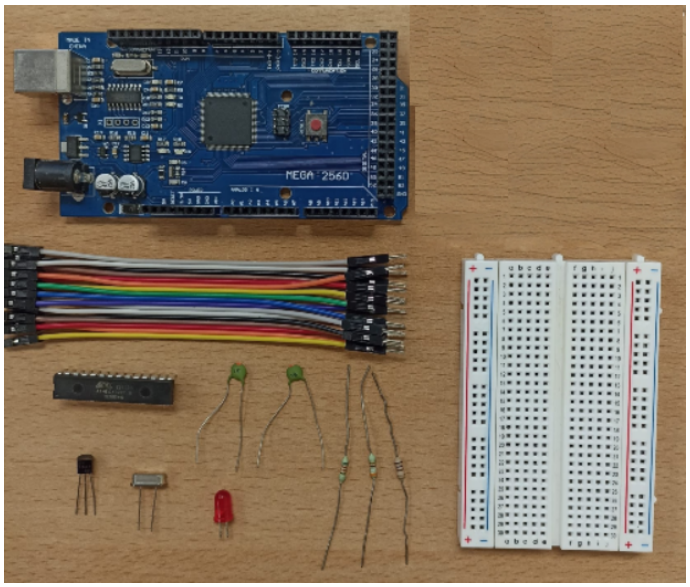

Figura 5.12: Materiales cuarto caso de prueba

Componenete	Cantidad	Coste
Microprocesador ATmega328p	1	2,62€
Arduino Mega 2560 Rev3	2	50,78 €
Breadboard	1	4,86 €
MOSFET 2N7000	1	0,45 €
Cables puente	40	3,87 €
Oscilador de cristal (16 MHz)	1	0,28 €
Condensador 15 pF	2	0,12 €
Resistencias	3	3,70 €

Tabla 5.3: Presupuesto cuarto caso de prueba

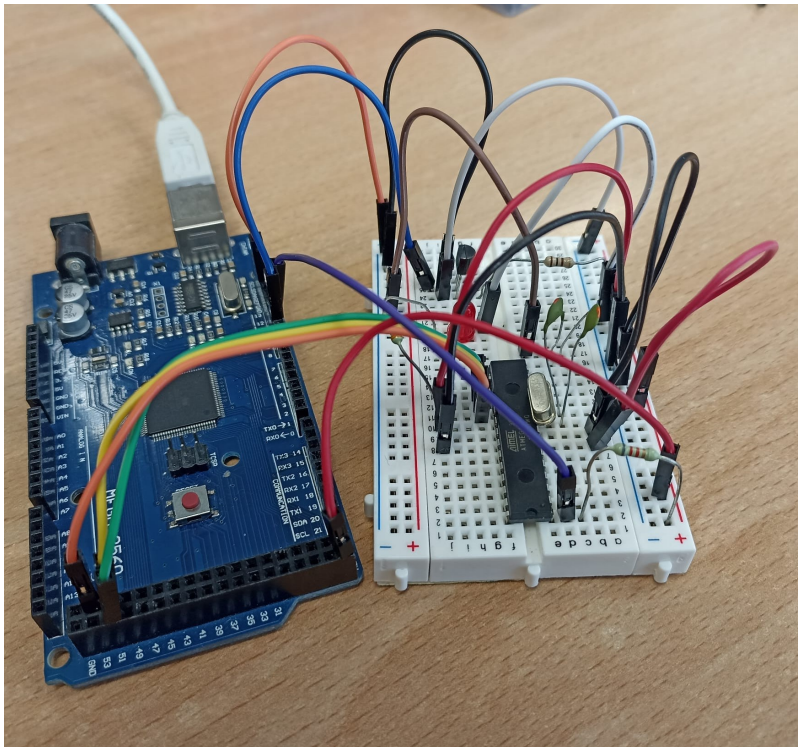

Figura 5.13: Arquitectura completa cuarta prueba

CAPÍTULO 6
Desarrollo de la solución

6.1 Primera prueba. Código desconocido

Tal y como se ha expuesto en el apartado anterior, esta prueba consiste en dos partes. La primera parte conlleva la programación del Arduino Uno (Target) a través de una conexión ISP con el objetivo de programar el bootloader y no solo subir un programa. Esto define un escenario en el cual el atacante no conoce el código ejecutado por el microcontrolador objetivo (Target).

Partiendo del esquema de la figura 5.4, el primer paso es convertir el Arduino Mega en un programador como tal. Para eso, el IDE de Arduino dispone de un programa específico que puede abrirse desde el menú **Archivo - Ejemplos - ArduinoISP** como se muestra en la figura 6.1.

Como cualquier otro programa, se ha de conectar la placa al PC mediante un cable USB y seleccionar la placa correcta en el menú. También es necesario marcar que se quiere utilizar como programador, como se muestra en la figura 6.2. Hecho esto, se carga el programa en la placa y el programador queda configurado.

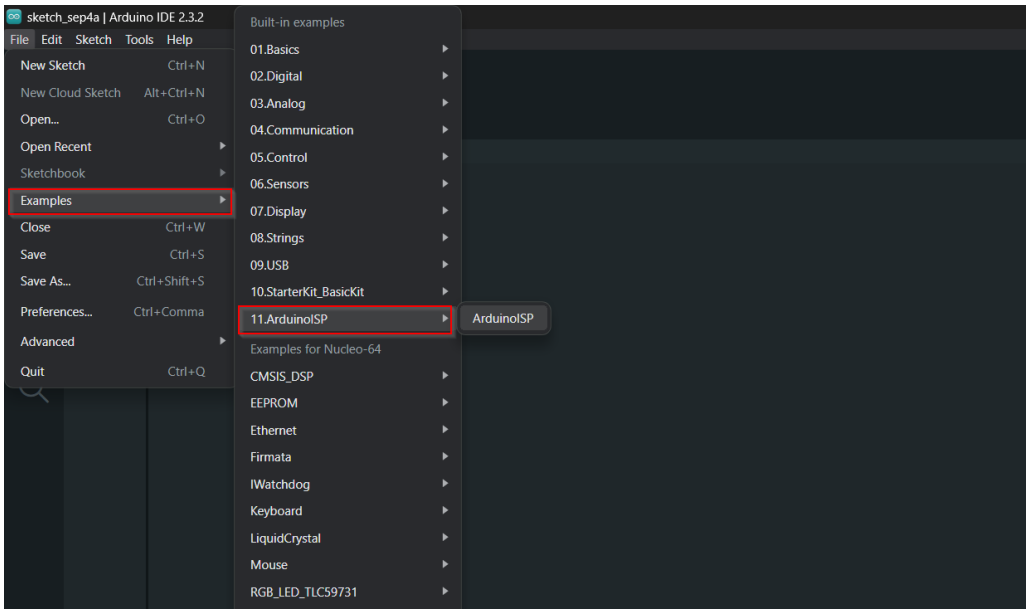

Figura 6.1: Ejemplo ArduinoISP

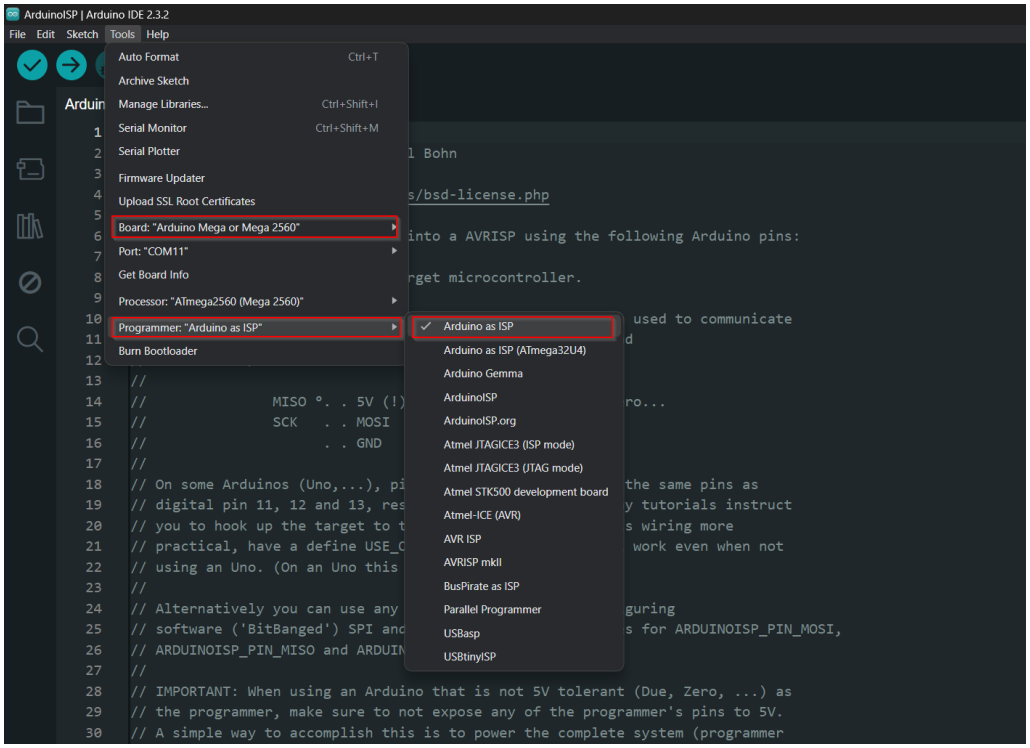

Figura 6.2: Arduino as ISP

El programa que debe contener el Arduino Uno, fiesta.hex, se ha obtenido del repositorio de la página en la cual se basa esta prueba [33]. Para cargarlo, se ha de utilizar avrdude[1] tal y como se muestra en la figura 6.3. Si se ha cargado correctamente veremos el mensaje "*avrdude.exe: 2188 bytes of flash written*".

```
PS C:\Users\mtbla\AppData\Local\Arduino15\packages\arduino\tools\avrdude\6.3.0-arduino17> .\bin\avrdude -C .\etc\avrdude.conf
c arduino -p atmega328p -P COM4 -b19200 -u -V -U flash:w:fiesta.hex
avrdude.exe: stk500_recv(): programmer is not responding
avrdude.exe: stk500_getsync() attempt 1 of 10: not in sync: resp=0x9a

avrdude.exe: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.02s

avrdude.exe: Device signature = 0x1e950f (probably m328p)
avrdude.exe: NOTE: "flash" memory has been specified, an erase cycle will be performed
             To disable this feature, specify the -D option.
avrdude.exe: erasing chip
avrdude.exe: reading input file "fiesta.hex"
avrdude.exe: input file fiesta.hex auto detected as Intel Hex
avrdude.exe: writing flash (2118 bytes):

Writing | ################################################## | 100% 2.55s

avrdude.exe: 2118 bytes of flash written

avrdude.exe done.  Thank you.

PS C:\Users\mtbla\AppData\Local\Arduino15\packages\arduino\tools\avrdude\6.3.0-arduino17>
```

Figura 6.3: Programación con avrdude

Tras este paso se ha configurado el código del arduino Target y concluimos la primera parte de la prueba. Para comprobar que se ha cargado correctamente el código en el Arduino Uno, lo conectamos al PC y podemos observar que imprime constantemente "Lock" (figura 6.4).

[1]https://aprendiendoarduino.wordpress.com/tag/avrdude/

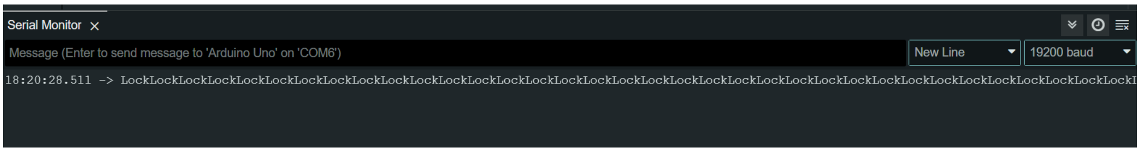

Figura 6.4: Mensaje Lock por consola

Para poder cargar programas en el Arduino Uno en un futuro, es necesario restaurar el bootloader. Para ello, teniendo conectados el programdor y el Target con los pines ISP ha de seleccionarse la placa Target en el Arduino IDE (Arduino Uno en este caso), el programador (Arduino as ISP) y pulsar en la opción "Burn bootloader" (figura 6.5). Este paso no debe realizarse para esta prueba pero sí será necesario para poder realizar las siguientes y cargar los programas en el Target.

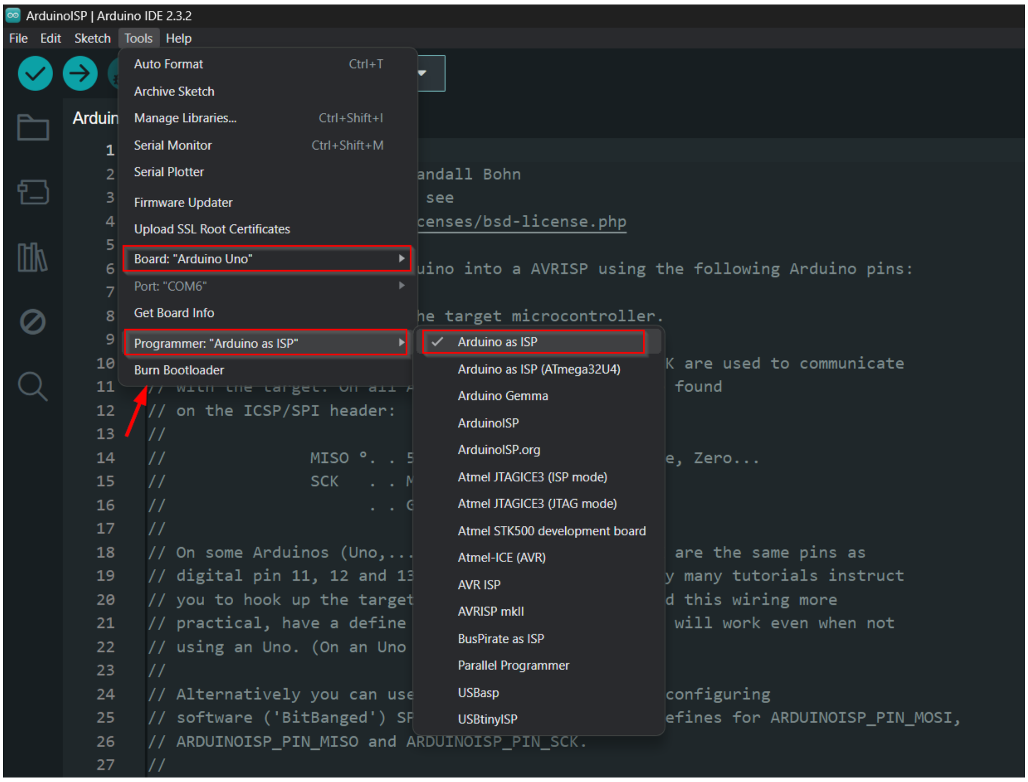

Figura 6.5: Restaurar bootloader

La segunda parte de esta prueba consiste en generar los fallos y para ello, el Arduino Mega se encarga de activar y desactivar el MOSFET para generar variaciones en el voltaje utilizando la función *glitch* mostrada en el listado 6.1. Esta función configura a LOW el pin de control de *Gate* del MOSFET (definido como *GLITCHING_PIN*), es decir, corta la conexión a tierra del Arduino Uno y, antes de restaurarla, espera un tiempo, incrementado en 10 cada vez que se llama a la función.

Para realizar la espera se define un bucle que incrementa una variable. Para evitar optimizaciones del compilador y que no se ejecute el bucle es necesario añadir el modificador *volatile* a la variable incrementada dentro del bucle.

```
void glitch(){
  volatile int waste = 0;

  digitalWrite(GLITCHING_PIN, LOW);
  for (int i = 0; i<glitchDelay; i++){ waste++; }
```

```
  digitalWrite(GLITCHING_PIN, HIGH);

  glitchDelay +=10;
}
```

Listing 6.1: Función glitch (Host)

Además de generar los cambios de voltaje, el código del Host lee los datos enviados por el arduino Target y los muestra por la consola para que, en caso de éxito, se vea por pantalla un mensaje. El código completo puede consultarse en el listado B.2 del anexo B.

6.2 Segunda prueba. Estructura if-else

En la primera prueba no se conoce el código ejecutado por el Arduino Target aunque se conjetura que está compuesto por una estructura if-else en la cual se comprueba el valor de una variable y dependiendo de éste, se transmite un mensaje u otro. El mensaje recibido es "Lock" y, si al producirse un *glitch* genera un fallo exitoso, se recibe el mensaje "unlocked".

Partiendo de esa suposición, en esta prueba se intentan generar los fallos de voltaje teniendo un código basado en una estructura if-else cargado en el Arduino Target, con el fin de manipular la instrucción de condición y acceder a una parte del código que no debería ser accesible.

El código del Arduino Uno, el Target, es muy sencillo, ya que tan solo contiene una estructura if-else (listado 6.2). Como la variable *cond* siempre es 1, no debería accederse a la zona de código del else y la ejecución resultaría en escribir "Lock" indefinidamente por el puerto serie.

```
int cond = 1;

void loop() {
  if(cond){
    Serial.write("Lock");
  } else {
    Serial.write("DESBLOQUEADO");
    delay(5000);
  }
  delay(50);
}
```

Listing 6.2: Estructura if-else código Target

El código del Arduino Mega, el Host, se centra en controlar el funcionamiento del MOSFET generando subidas de tensión de diferentes duraciones. En los listados 6.3 y 6.4 se muestra cómo se escribe a cero el pin de control del MOSFET (produciendo una subida de tensión), se realiza una espera (GLITCHING_DURATION) y se retorna al estado inicial con el MOSFET activo. En el listado 6.3 el retardo siempre es el mismo pero en el listado 6.4, éste varía en cada repetición de loop().

```
void loop() {
    for(int i = 0 ; i < GLITCHING_ITERATIONS ; i++) {
      digitalWrite(GLITCHING_PIN, LOW);
      delayMicroseconds(GLITCHING_DURATION);        //Glitch duration
      digitalWrite(GLITCHING_PIN, HIGH);
    }
    delay(GLITCHING_INTERVAL);
}
```

Listing 6.3: Generación de fallos de duración fija

```
void loop() {
    rand = random(1, MAX_GLITCHING_DURATION);

    for(int i = 0 ; i < GLITCHING_ITERATIONS ; i++) {
      digitalWrite(GLITCHING_PIN, LOW);
      delayMicroseconds(rand);                //Glitch duration
      digitalWrite(GLITCHING_PIN, HIGH);
    }
    delay(GLITCHING_INTERVAL);
}
```

Listing 6.4: Generación de fallos de duración aleatoria

El propóstito del otro Arduino Mega (Reader) es imprimir por pantalla los datos enviados por el Target. Se ha decidido separarlo para que el Host se centre solo en provocar el *glitch* y no sobrecargarlo, pero poder leer los datos enviados por el Target y tener visibilidad de los errores provocados.

La función del Reader consiste en leer los datos recibidos por el puerto serie constantemente (listado 6.5). Si el carácter recibido es una 'D', significa que alguno de los *glitches* ha tenido éxito y se ha accedido a la zona *else* en el código del Target y, por lo tanto, éste envía "DESBLOQUEADO".

```
void loop() {
  if (Serial1.available() > 0) {
    incomingByte = Serial1.read();
    Serial.print(char(incomingByte));

    //Glitch exitoso
    if (char(incomingByte) == 'D'){ //*DESBLOQUEADO*
      Serial.println("\nConseguido");
      delay(10000);
    }
  }
}
```

Listing 6.5: Lectura de datos

6.3 Tercera prueba. Bucle for

En esta prueba se ha cambiado el escenario de inyección de fallos. En vez de tener una estructura if-else, el Target ejecuta un bucle for de 10000 iteraciones incrementando el valor de una variable contador (listado 6.6). El objetivo de la inyección de fallos es provocar que el microprocesador se salte alguna instrucción y el valor del contador no se corresponda con el que debería tener.

```
void loop() {
  digitalWrite(LED_BUILTIN, LOW);

  for(i=0; i<10000; i++){
    ctr++;
  }

  if(ctr == 10000){
    ctr = 0;
  } else {
    digitalWrite(LED_BUILTIN, HIGH);
  }
}
```

Listing 6.6: Bucle for código Target

Para la programación del Host se utilizan los programas de la prueba anterior. Pueden consultarse en los listados C.2 y C.3 del anexo C.

6.4 Cuarta prueba. Bucle for sin Arduino Uno

Aunque la arquitectura de esta prueba cambia bastante respecto a las otras, el código, tanto del Target como del Host, puede ser reutilizado. Se utilizarán los listados D.1, C.2 y C.3 mencionados anteriormente.

La programación del ATmega no puede hacerse a través del Arduino Uno conectado el cable USB, se ha de cargar el programa utilizando el Host como programador y, para ello, el Arduino IDE proporciona una opción.

Partiendo del esquema de la figura 5.13, el primer paso es convertir al Arduino Mega en programador. Los pasos para realizar esta tarea se han explicado en el apartado 6.1 de este documento. A continuación, ha de abrirse el fichero que contenga el código del Target y seleccionar la placa destino como si fuera a subirse el programa tal y como lo haremos para el Host. Sin embargo, en este caso utilizaremos la opción "Subir usando programador" (figura 6.6) tras haber marcado la opción "Arduino as ISP" en **Herramientas - Programador**.

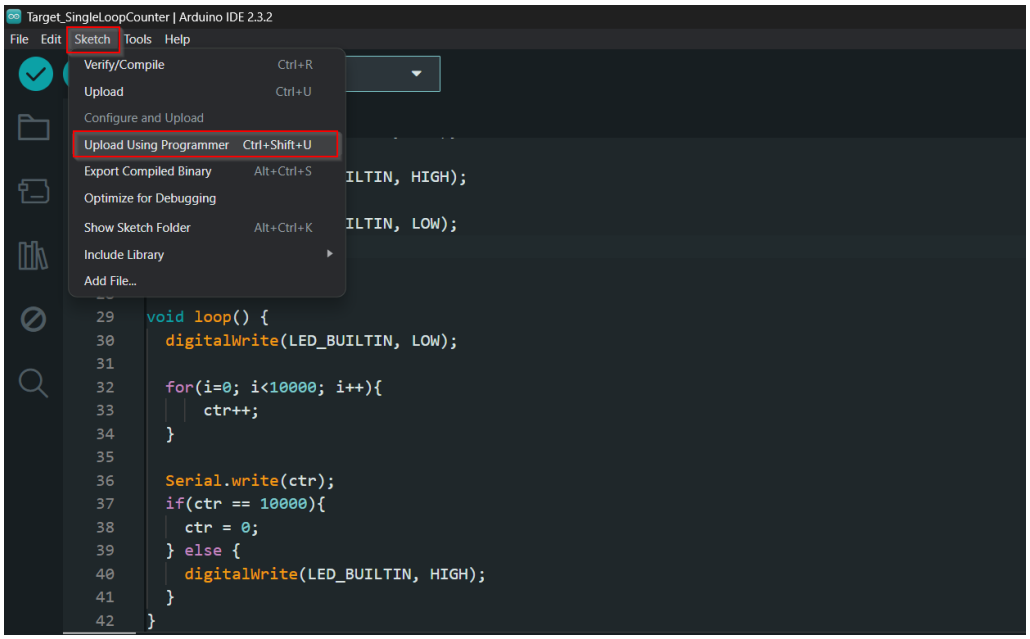

Figura 6.6: Subir utilizando programador

CAPÍTULO 7
Pruebas

7.1 Primera prueba. Código desconocido

Al conectar el Arduino Mega a la fuente de alimentación (PC) comienza la ejecución del programa del Host y del Target. En la consola observamos el inicio de los mensajes (figura 7.1) y cómo se producen reinicios en algunos momentos a causa de los *glitches* introducidos (figura 7.2). Cuando la subida de tensión se produce por un tiempo prolongado, el microcontrolador comienza a reiniciarse continuamente (figura 7.3). A partir de este momento, sabemos el rango de tiempo en el cual la inyección de fallos puede ser efectiva sin llegar a provocar un reinicio y será en este espacio de tiempo donde se realizarán las pruebas.

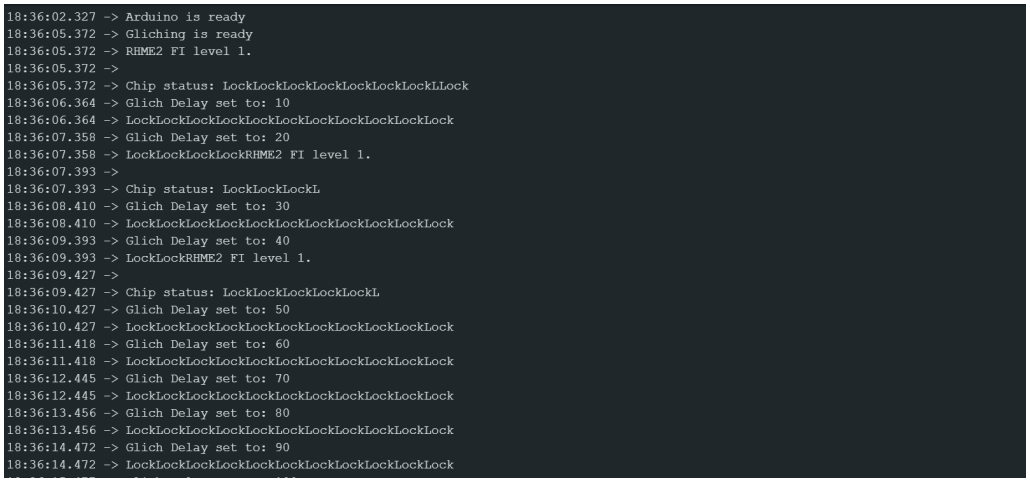

Figura 7.1: Mensajes consola

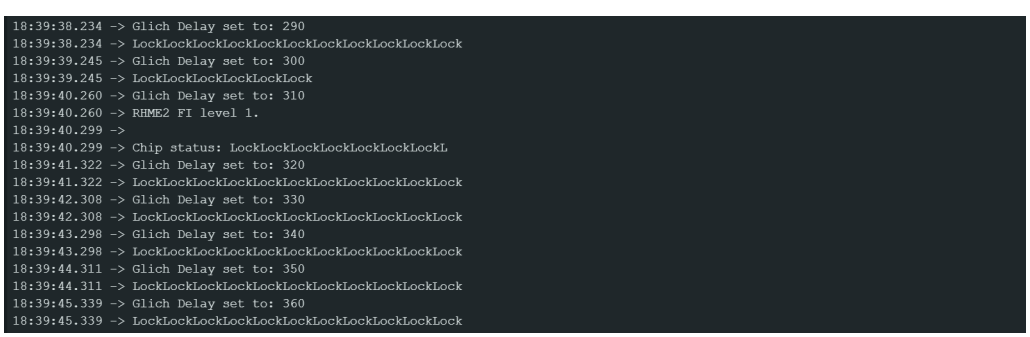

Figura 7.2: Efecto glitch

```
18:41:10.373 -> LockLockLockLockLockLockLockLockLockLockLock
18:41:11.372 -> Glich Delay set to: 1210
18:41:11.372 -> LockLockLockLockLockLockLockLockLockLo
18:41:12.411 -> Glich Delay set to: 1220
18:41:12.411 -> RHME2 FI level 1.
18:41:12.411 ->
18:41:12.411 -> Chip status: LockLockLockLockLockLockL
18:41:13.410 -> Glich Delay set to: 1230
18:41:13.410 -> LockLockLockLockLockLockLockLockLockLock
18:41:14.434 -> Glich Delay set to: 1240
18:41:14.434 -> RHME2 FI level 1.
18:41:14.471 ->
18:41:14.471 -> Chip status: LockLockLockLockLockLockL
18:41:15.444 -> Glich Delay set to: 1250
18:41:15.444 -> LockLockLockLockLockLockLockLockLockLock
18:41:16.456 -> Glich Delay set to: 1260
18:41:16.456 -> RHME2 FI level 1.
18:41:16.489 ->
18:41:16.489 -> Chip status: LockLockLockLockLockLockL
18:41:17.489 -> Glich Delay set to: 1270
18:41:17.489 -> LockLockLockLockLockLockLockLockLockLock
18:41:18.526 -> Glich Delay set to: 1280
18:41:18.526 -> RHME2 FI level 1.
18:41:18.526 ->
18:41:18.526 -> Chip status: LockLockLockLockLockLockL
18:41:19.522 -> Glich Delay set to: 1290
18:41:19.522 -> RHME2 FI level 1.
```

Figura 7.3: Reinicio target

En las figuras 7.4 y 7.5 se muestran tres sondas en el osciloscopio. La línea amarilla representa la transmisión de datos del Target por el puerto serie. En la figura 7.6 se amplía esta línea y si se analiza bit a bit, puede comprobarse que el mensaje enviado se corresponde con la secuencia "Lock".

La línea de color azul muestra el pin del Host que controla el MOSFET. En la figura 7.4 los cursores del osciloscopio muestran con exactitud el tiempo del *glitch* en ese instante. Y la línea verde representa GND del Arduino Target. Según los resultados obtenidos, la desactivación del MOSFET no produce una subida de tensión abrupta, sino que ésta aumenta conforme el tiempo de el *glitch* es mayor. En la figura 7.5 los cursores muestran que la subida de tensión al final del *glitch* es de 2 voltios. Si se supera este voltaje empiezan a producirse reinicios en el microcontrolador.

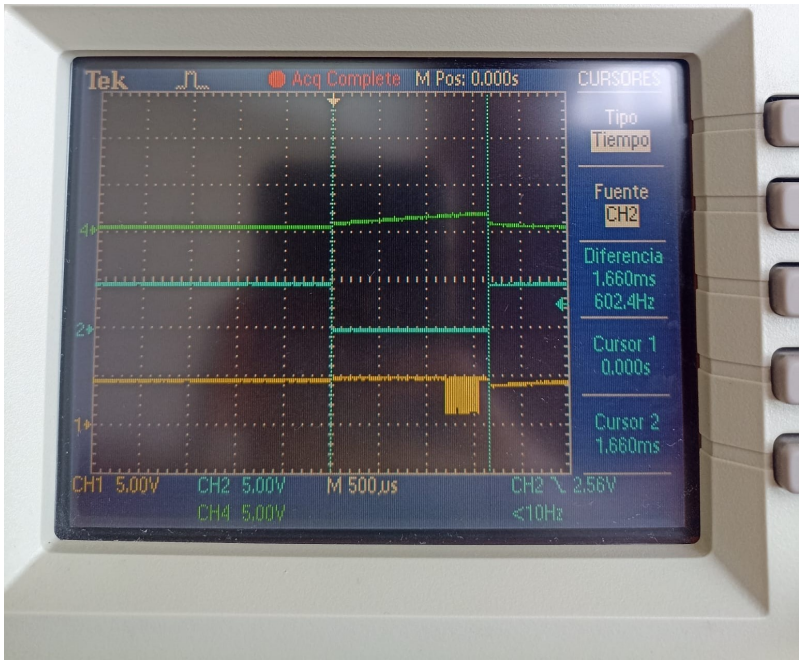

Figura 7.4: Tiempo de glitch

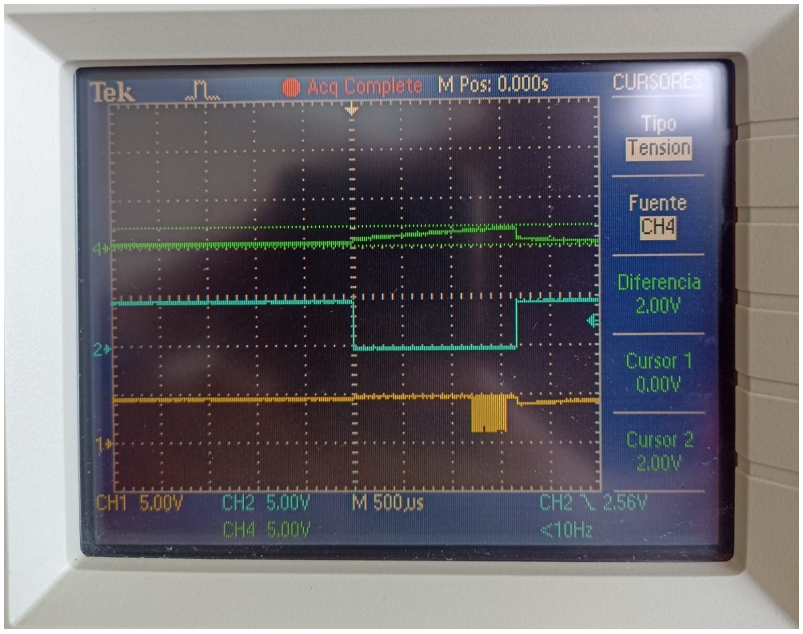

Figura 7.5: Voltaje de glitch

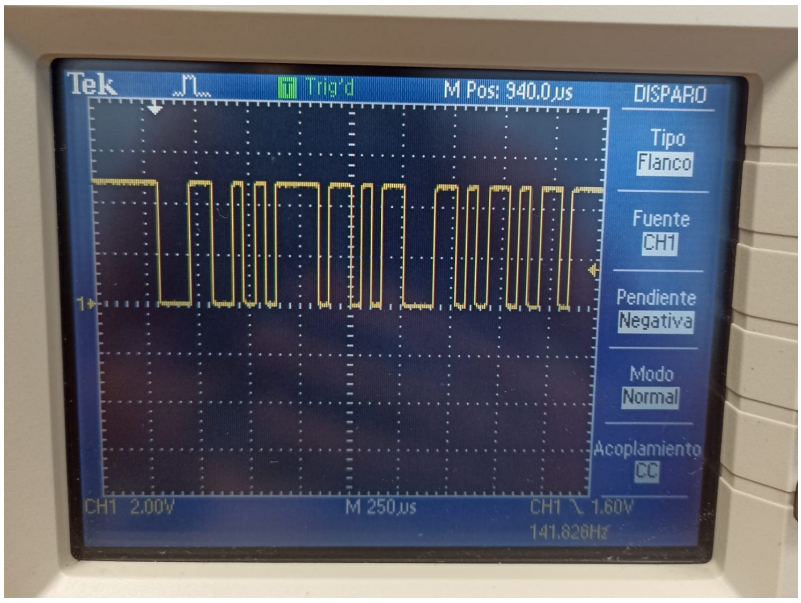

Figura 7.6: Mensaje Lock osciloscopio

Tras varias pruebas modificando la duración del *glitch* se han conseguido fallos exitosos que producen reinicios en el microcontrolador pero no se ha conseguido controlar los fallos para alterar el flujo de ejecución del programa evitando el reinicio.

7.2 Segunda prueba. Estructura if-else

El primer experimento realizado con el código de la estructura if-else en el Target utilizaba el programa del Host de la primera prueba (B.2) pero debido a que los reinicios del microcontrolador se producían muy rápido (figura 7.7) se decidió utilizar el Arduino que actúa como Reader y cambiar el programa del Host al listado C.2.

```
19:13:56.605 -> Glich Delay set to: 830
19:13:56.605 -> LockLockLockLockLockLockLockLockLockLockLockLockLockLock
19:13:57.635 -> Glich Delay set to: 840
19:13:57.635 -> LockLockLockLockLockLockLockLockLockLockLockLockLockLocck
19:13:58.673 -> Glich Delay set to: 850
19:13:58.673 -> LockLockLockLockLockLockLockLockLockLockLockLockLockLock
19:13:59.673 -> Glich Delay set to: 860
19:13:59.673 -> LockLockLockLockLockLockLockLockLockLockLockLockLockLoc
19:14:00.673 -> Glich Delay set to: 870
19:14:00.673 ->
19:14:00.673 -> INICIO
19:14:01.686 -> Glich Delay set to: 880
19:14:02.669 ->
19:14:02.669 -> Glich Delay set to: 890
19:14:02.669 ->
19:14:02.669 -> INICIO
19:14:03.702 -> Glich Delay set to: 900
19:14:04.721 ->
19:14:04.721 -> Glich Delay set to: 910
19:14:04.721 ->
19:14:04.721 -> INICIO
19:14:05.723 -> Glich Delay set to: 920
```

Figura 7.7: Reinicio target

Para realizar la prueba con los tres Arduinos, el Host y el Reader deben conectarse a la fuente de alimentación (PC). Hecho esto, comienza la ejecución y podemos ver por la consola la transmisión de mensajes del Target, que consisten en una secuencia "INICIO" y la secuencia "Lock" reiteradamente hasta que el Target se reinicia o el *glitch* provoca el acceso a la zona del else.

En las figuras 7.8, 7.10 y 7.9 se muestran tres sondas en el osciloscopio. La línea verde representa la transmisión de datos y como en la figura 7.6 de la prueba anterior, si se analiza bit a bit puede comprobarse que el mensaje enviado se corresponde con la secuencia "Lock".

La línea amarilla muestra el pin del Host que controla el MOSFET y la línea azul representa GND del Target. Como puede verse en las figuras 7.10 y 7.9, en algunas ocasiones la subida de voltaje es muy abrupta y son esos puntos en los que se puede producir un *glitch* exitoso.

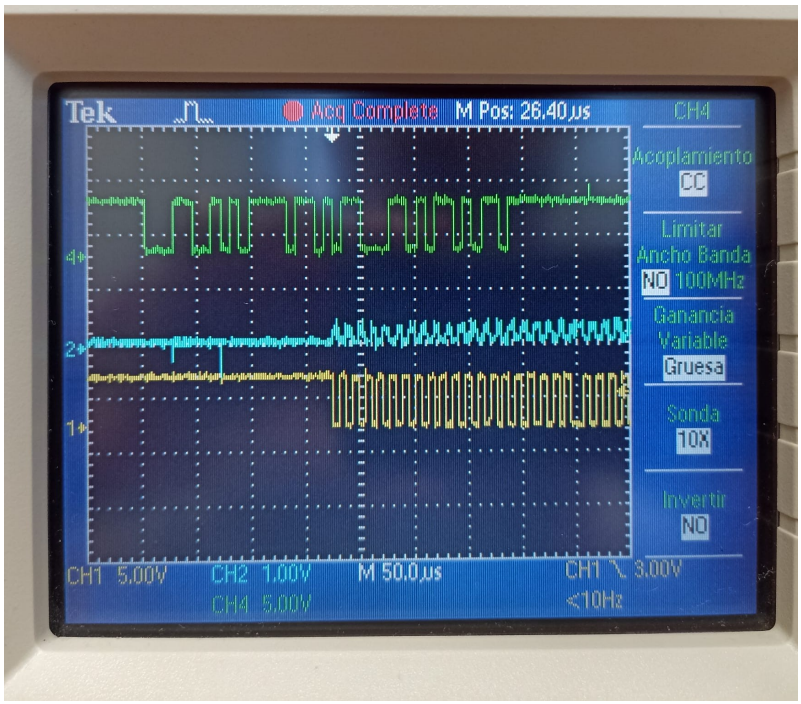

Figura 7.8: Transmisión de datos vista en el osciloscopio

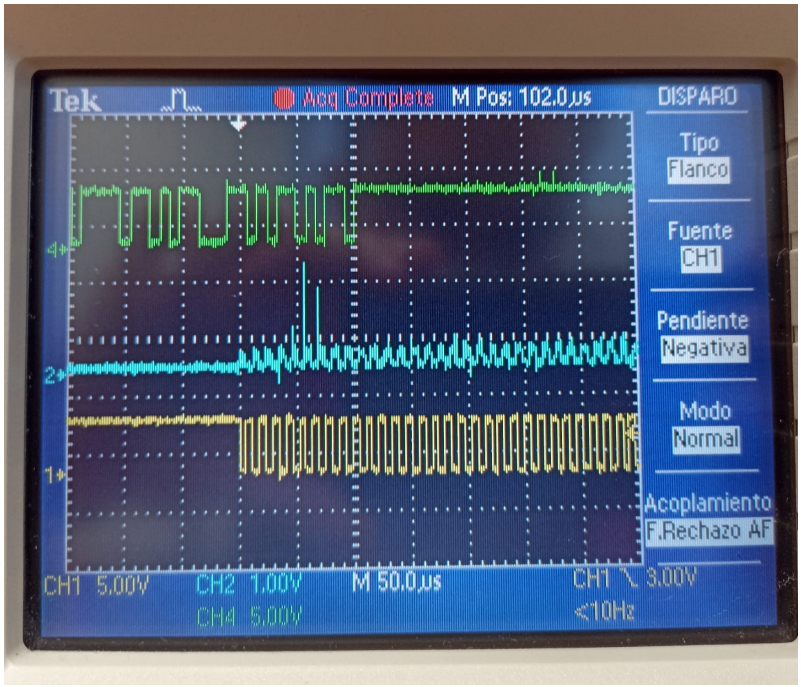

Figura 7.9: Subidas de tensión provocadas por el glitch (1)

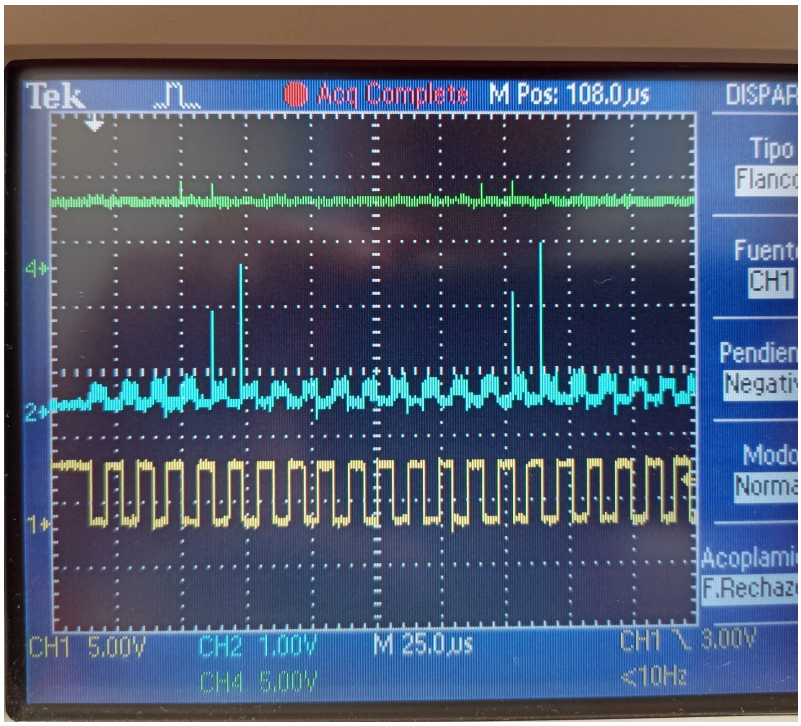

Figura 7.10: Subidas de tensión provocadas por el glitch (2)

Después de varias pruebas, los *glitches* han producido bastantes reinicios en el microcontrolador pero han habido algunos fallos, como se muestra en las figuras 7.11, 7.12 y 7.13, que han alterado el flujo de ejecución accediendo a la parte el else en el código del Target.

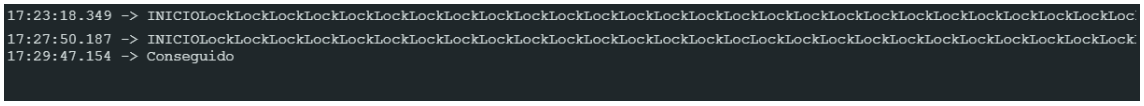

Figura 7.11: Éxito glitch (1)

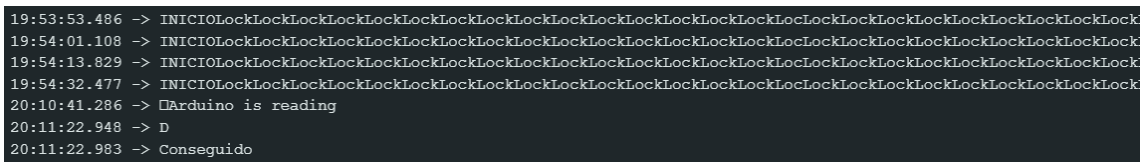

Figura 7.12: Éxito glitch (2)

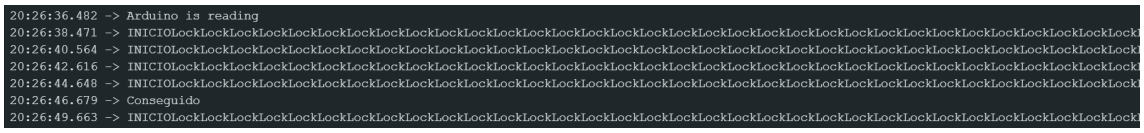

Figura 7.13: Éxito glitch (3)

7.3 Tercera y cuarta prueba. Bucle for

Los experimentos realizados en estas prueba son similares a los de la prueba anterior. En este caso no hay comunicación a través del puerto serie pero se muestra la línea del LED en el osciloscopio. Esta línea, la verde, se mantiene a cero si el *glitch* no tiene efecto. En caso de producir saltos en instrucciones o cualquier tipo de fallo que resulte en provocar que el valor de la variable contador sea diferente de 10000, el LED se enciende. Como puede verse en la figura 7.15, algunos *glitches* provocan el reinicio del microcontrolador y, como la secuencia de inicio son dos parpadeos, se puede ver en el osciloscopio que la línea del LED se pone a uno.

La línea amarilla representa el control del MOSFET y y la azul GND del Target. Tal y como ha ocurrido en las pruebas anteriores, hay momentos en los que la subida de tensión es más pronunciada y es en estos momentos donde el *glitch* tiene más probabilidad de éxito.

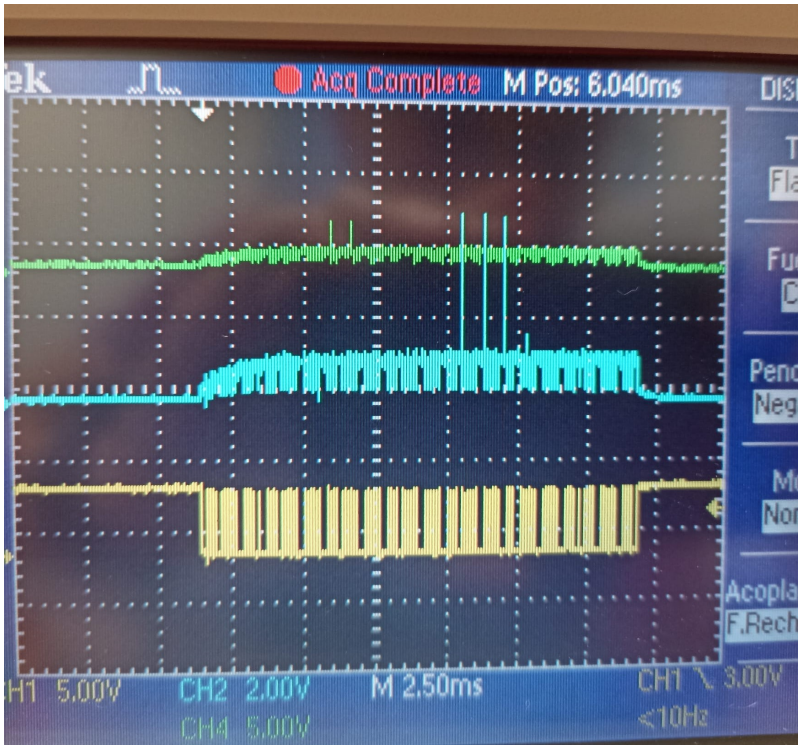

Figura 7.14: Subidas de tensión provocadas por el glitch

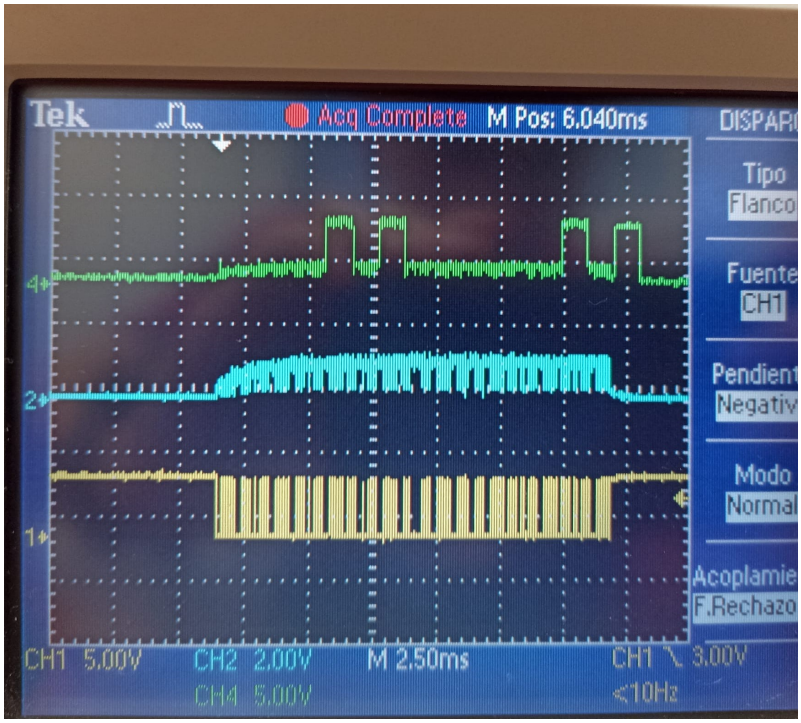

Figura 7.15: Reinicio microcontrolador. Parpadeo del LED

Tras varias pruebas y tiempo analizando los *glitches* producidos se han conseguido fallos exitosos que producen reinicios en el microcontrolador pero no se ha conseguiso controlar estos fallos para alterar el flujo de ejecución del programa evitando el reinicio.

7.4 Análisis de los resultados

El objetivo de las pruebas era inducir alteraciones en el flujo de ejecución del programa mediante la manipulación del suministro de voltaje al dispositivo. Sin embargo, a pesar de los esfuerzos realizados y de haber ajustado diferentes parámetros, no se ha logrado obtener los resultados esperados. A continuación, se describen los motivos que pueden haber contribuido al resultado no exitoso.

Se han realizaron múltiples intentos de inyección de fallos de voltaje, con distintas combinaciones de los parámetros descritos anteriormente y se ha observado que, conforme se aumentaba la duración del *glitch*, el microcontrolador simplemente se reiniciaba más continuamente, pero no mostraba ningún comportamiento anómalo durante su ejecución posterior al reinicio.

Los resultados obtenidos indican que es posible que los fallos inyectados no se aplicaran en momentos del ciclo de reloj donde podrían generar un fallo significativo o que el rango de variación de voltaje no haya sido lo suficientemente amplio como para introducir fallos críticos sin comprometer el funcionamiento del dispositivo

A pesar de los múltiples intentos, solo se ha logrado alterar el flujo de ejecución del microcontrolador en uno de los casos presentados. Estos resultados sugieren la necesidad de explorar en más profundidad los parámetros para ajustar el momento exacto del *glitch* o probar otras metodologías, como utilizar diferentes microcontroladores.

CAPÍTULO 8
Conclusiones

A lo largo del TFG se ha recalcado la importancia de la seguridad hardware. Se han estudiado las técnicas más comunes para la inyección de fallos en dispositivos físicos y otras técnicas de hardware hacking cuyo objetivo es obtener o manipular información de un sistema. Además, se han realizado varios casos prácticos en los que se ha experimentado, en concreto, con la técnica de inyección de fallos de voltaje y se ha expuesto las dificultades de realizar un ataque exitoso.

Para poner en contexto al lector, se ha realizado un estudio de mercado de algunas de las herramientas ya utilizadas para la inyección de fallos de voltaje pero se ha optado por realizar las pruebas con una arquitectura más económica y sencilla para experimentar las dificultades que conlleva realizar este ataque en la práctica y valorar las herramientas que facilitan estas tareas.

Para cada prueba realizada se ha obtenido el presupuesto y se han realizado las configuraciones necesarias para inyecctar fallos de voltaje en entornos con diferentes características. Para terminar la parte práctica, se han realizado pruebas que demuestran los éxitos de los fallos inyecctados.

Con todo lo expuesto, se analiza el cumplimiento de los objetivos propuestos al inicio del proyecto:

OBJETIVO 1 - Dar a conocer la importancia de la seguridad hardware: A lo largo del trabajo, se ha destacado la relevancia de la seguridad hardware en un mundo cada vez más conectado y dependiente de dispositivos electrónicos. Se ha mostrado cómo un ataque exitoso a nivel de hardware puede comprometer la seguridad completa de un sistema, lo que subraya la necesidad de diseñar microcontroladores y otros dispositivos hardware con mecanismos de protección robustos frente a ataques físicos, como la inyección de fallos.

OBJETIVO 2 - Aprender sobre técnicas de hardware hacking: Durante el desarrollo del proyecto, se han adquirido conocimientos prácticos y teóricos sobre diferentes técnicas de hardware hacking. No solo se ha hecho un estudio sobre estas técnicas sino que el proceso de preparación y ejecución de los ataques ha proporcionado una comprensión profunda de la complejidad involucrada en la manipulación física de los sistemas.

OBJETIVO 3 - Informar sobre la aplicación de estas técnicas en el arranque seguro de un dispositivo: El presente trabajo también explora la aplicación de técnicas de hardware hacking en el contexto del arranque seguro de dispositivos. Se han discutido los riesgos que presentan los ataques de inyección de fallos, especialmente en los procesos de arranque, donde se espera que un dispositivo inicie de manera segura y verificada.

OBJETIVO 4: Documentar el proceso de realización de un ataque por inyección de fallos de forma reproducible y asequible: Se ha descrito en detalle el material utilizado,

los pasos seguidos, y las configuraciones implementadas, con el objetivo de que otros investigadores puedan replicar los experimentos o adaptar las metodologías a sus propios proyectos. A pesar de haber realizado alguna prueba con éxito, la inyección de fallos presentada no es reproducible, por lo que no se han obtenido los resultados esperados.

Finalmente, se puede afirmar que, a pesar de no haber conseguido documentar un ataque de inyección de fallos de voltaje reproducible, ya que no se puede concretar el momento exacto en el cual el *glitch* ha tenido éxito, sí se ha trasladado al lector la importancia de un sistema hardware seguro.

CAPÍTULO 9
Trabajos futuros

La técnica de inyección de fallos de voltaje puede llevarse a cabo en múltiples escenarios y contra muchos dispositivos hardware diferentes. En este documento se ha presentado la técnica utilizando un microcontrolador y unas herramientas específicas que conforman un entorno sencillo para llevar a cabo el ataque. Algunas vías de investigación posibles se comentan a continuación.

En primer lugar, el estudio podría extenderse a otros modelos de microcontroladores o incluso a diferentes arquitecturas de procesadores para evaluar si los resultados obtenidos son generales o específicos de un modelo en particular. Además de la inyección de fallos de voltaje, se podrían realizar casos prácticos de otras técnicas de hardware hacking presentadas en el trabajo.

En segundo lugar, el trabajo podría extenderse para investigar cómo la inyección de fallos de voltaje puede comprometer el proceso de arranque seguro en los microcontroladores. Esto incluiría analizar si los fallos de voltaje pueden alterar la autenticación del código de arranque o permitir la ejecución de código no autorizado. En concreto, el estudio del arranque seguro podría centrarse en dispositivos IoT, que a menudo tienen limitaciones en cuanto a recursos y seguridad.

Finalmente, la parte práctica podría extenderse para realizar otros tipos de fallos en el mismo microcontrolador y con las arquitecturas presentadas y observar si los resultados obtenidos en estos casos son más exitosos. Algunas técnicas que podrían probarse serían la inyección de fallos de reloj o fallos electromagnéticos. Esto permitiría un análisis más completo de la resiliencia del microcontrolador ante diferentes amenazas.

CAPÍTULO 10
Glosario de términos

IoT: *Internet of Things* o Internet de las cosas. Hace referencia a objetos físicos con sensores, capacidad de procesamiento y software que se conectan e intercambian datos con otros dispositivos. [34]

Glitch: Es un error que no puede considerarse un fallo, sino más bien una característica no prevista.

ISP o **ISCP**: *In-System Programming* o *In-Circuit Serial Programmer*. Es la habilidad de algunos dispositivos lógicos programables, microcontroladores y otos circuitos electrónicos de ser programados mientras están instalados en un sistema completo, en lugar de requerir que el chip sea programado antes de ser instalado dentro del sistema. [35]

MISO: *Master Input Slave Output*.Transmite datos desde el esclavo hacia el máster.

MOSI: *Master Output Slave Input*. Transmite datos desde el máster hacia el exclavo.

SCK: *Serial Clock*. Hace referencia a los pulsos de reloj que sincronizan la transmisión de datos generada por el Controlador y una línea específica para cada dispositivo.

USB: *Universal Serial Bus*. Es un bus de comunicaciones que sigue un estándar industrial que permite el intercambio de datos y la entrega de energía entre muchos tipos de dispositivos electrónicos.

Bibliografía

[1] N. Mark Tehranipoor; Nalla Anandakumar; Farimah Farahmandi. *Hardware Security Training, Hands-on!* Springer Nature Switzerland, 2023.

[2] Mario Micucci. Hardware hacking: introducción al análisis de un firmware. *WeLiveSecurity*, 2023. [Citado el 8 de agosto de 2024]. Disponible en `https://www.welivesecurity.com/la-es/2023/04/21/hardware-hacking-introduccion-analisis-firmware/`.

[3] Jasper van Woudenberg y Colin O'Flynn. *The Hardware Hacking Handbook*. William Pollock, 2022.

[4] Nicolás Ruminot; Claudio Estevez; Samuel Montejo-Sánchez. A novel approach of a low-cost voltage fault injection method for resource-constrained iot devices: Design and analysis. *Sensors*, 2023.

[5] Antonio Andreu Escrivá. *Domotización y control de una vivienda*. Universidad Politécnica de Valencia, 2020.

[6] David Núñez Martínez. *Sistema de seguridad domótica para control de acceso mediante autenticación con huella dactilar*. Universidad Politécnica de Valencia, 2021.

[7] Rafael José Boix Carpi. *Optimization of parameter settings search for a successful Fault Injection*. Universidad Politécnica de Valencia, 2013.

[8] Daniel Gil Tomás; Joaquin Gracia-Morán; Juan Carlos Baraza Calvo; Luis Saiz-Adalid; Pedro Joaquin Gil Vicente. *Studying the effects of intermittent faults on a microcontroller*. Universidad Politécnica de Valencia, 2012.

[9] Gabriel Cobos Tello. *Fault Tolerance Techniques for FPGA-based Space Applications*. Universidad Politécnica de Valencia, 2019.

[10] Sergei P. Skorobogatov. Semi-invasive attacks – a new approach to hardware security analysis. Technical report, University of Cambridge, 2005.

[11] Colin O'Flynn. Fault injection using crowbars on embedded systems. *Cryptology ePrint*, 2016.

[12] Stefanos Koffas; Praveen Kumar Vadnala. On the effect of clock frequency on voltage and electromagnetic fault injection. *Delft University of Technology*, 2022.

[13] Théo Gordyjan. How to voltage fault injection. *Synacktiv*, Théo Gordyjan. [Citado el 10 de agosto de 2024]. Disponible en `https://www.synacktiv.com/en/publications/how-to-voltage-fault-injection#clock_glitching`.

[14] Gavin Wright. side-channel attack. *TechTarget*, 2021. [Citado el 14 de agosto de 2024]. Disponible en https://www.techtarget.com/searchsecurity/definition/side-channel-attack.

[15] Rambus. Side-channel attacks explained: everything you need to know. *Rambus Press*, 2021. [Citado el 14 de agosto de 2024]. Disponible en https://www.rambus.com/blogs/side-channel-attacks/.

[16] Niek Timmers; Cristofaro Mune. Escalating privileges in linux using voltage fault injection. *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2017. [Citado el 22 de agosto de 2024]. Disponible en https://api.semanticscholar.org/CorpusID:41621870.

[17] Aurélien Vasselle; Hugues Thiebeauld; Quentin Maouhoub; Adèle Morisset; Sébastien Ermeneux. Laser-induced fault injection on smartphone bypassing the secure boot-extended version. *IEEE Trnasactions on Computers*, 2020. [Citado el 22 de agosto de 2024]. Disponible en https://api.semanticscholar.org/CorpusID:69211347.

[18] Yingchen Wang; Riccardo Paccagnella; Elizabeth Tang He; Hovav Shacham; Christopher W. Fletcher; David Kohlbrenner. Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86. *IEEE Micro*, 2023. [Citado el 22 de agosto de 2024]. Disponible en https://api.semanticscholar.org/CorpusID:251793871.

[19] Catinca Mujdei; Lennert Wouters; Angshuman Karmakar; Arthur Beckers; Jose Maria Bermudo Mera; Ingrid M R Verbauwhede. Side-channel analysis of lattice-based post-quantum cryptography: Exploiting polynomial multiplication. *ACM Transactions on Embedded Computing Systems*, 2022. [Citado el 22 de agosto de 2024]. Disponible en https://api.semanticscholar.org/CorpusID:248754789.

[20] Marc Fyrbiak; Sebastian Strauss; Christian Kison; Sebastian Wallat; Malte Elson; Nikol Rummel; Christof Paar. Hardware reverse engineering: Overview and open challenges. *2017 IEEE 2nd International Verification and Security Workshop (IVSW)*, 2017. [Citado el 22 de agosto de 2024]. Disponible en https://api.semanticscholar.org/CorpusID:29756857.

[21] Arne Roar Nygård; Sokratis K. Katsikas. Leveraging hardware reverse engineering to improve the cyber security and resilience of the smart grid. In *International Conference on Security and Cryptography*, 2023. [Citado el 22 de agosto de 2024]. Disponible en https://api.semanticscholar.org/CorpusID:259871923.

[22] Jeremy Boone; Sultan Qasim Khan. An introduction to fault injection (part 1/3). *nccgroup*, 2021. [Citado el 20 de agosto de 2024]. Disponible en https://research.nccgroup.com/2021/07/07/an-introduction-to-fault-injection-part-1-3/.

[23] Mouser Electronics. Sistemas embebidos de seguridad, con la seguridad no se juega. *convertronic*. [Citado el 24 de agosto de 2024]. Disponible en https://convertronic.net/mas-sistemas/sistemas-embebidos/9730-sistemas-embebidos-de-seguridad-con-la-seguridad-no-se-juega.html.

[24] Rick Housley. Badfet: Defeating modern secure boot using second-order pulsed electromagnetic fault injection. *Red Balloon Security*, 2017.

[25] Incibe. Hardware hacking en sistemas de control industrial. *Incibe-cert*, 2017. [Citado el 20 de agosto de 2024]. Disponible en https://www.incibe.es/incibe-cert/blog/hardware-hacking-sistemas-control-industrial.

[26] David Oswald. Lecture notes: Hardware and embedded systems security, 2023.

[27] NewAE Technology. [Citado el 15 de agosto de 2024]. Disponible en https://rtfm.newae.com/Capture/.

[28] Ernesto Tolocka. La familia de microcontroladores stm32, 2021. [Citado el 15 de agosto de 2024]. Disponible en https://www.profetolocka.com.ar/2021/03/29/la-familia-de-microcontroladores-stm32/.

[29] Vicente García. El transistor mosfet. *Electrónica Práctica Aplicada*, 2012. [Citado el 15 de agosto de 2024]. Disponible en https://www.diarioelectronicohoy.com/blog/el-transistor-mosfet.

[30] Ariat. Estructura, características y usos de mosfet de canal p en el campo electrónico. *Ariat Technology*, 2016. [Citado el 15 de agosto de 2024]. Disponible en https://www.ariat-tech.es/blog/structure,characteristics-and-uses-of-p-channel-mosfet-in-the-electronic-field.html#t3.

[31] Christoffer Claesson. Voltage glitching on the cheap. *Securebits.io*, 2019. [Citado el 23 de agosto de 2024]. Disponible en https://blog.securitybits.io/2019/06/voltage-glitching-on-the-cheap/.

[32] Arduino, 2024. [Citado el 30 de julio de 2024]. Disponible en https://docs.arduino.cc/built-in-examples/arduino-isp/ArduinoISP/.

[33] Riscure. Rhme-2016. [Citado el 23 de agosto de 2024]. Disponible en https://github.com/Riscure/Rhme-2016/blob/main/challenges/binaries/fiesta/fiesta.hex/.

[34] Wikipedia. Internet de las cosas. [Citado el 3 de septiembre de 2024]. Disponible en https://es.wikipedia.org/wiki/Internet_de_las_cosas.

[35] Wikipedia. Programación en el sistema. [Citado el 3 de septiembre de 2024]. Disponible en https://en.wikipedia.org/wiki/In-system_programming.

ANEXO A
Objetivos de desarrollo sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**			X	
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**	X			
ODS 17. **Alianzas para lograr objetivos.**				X

Tabla A.1: Objetivos de desarrollo sostenible

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

El 25 de septiembre de 2015, la Asamblea General de Naciones Unidas aprobó un conjunto de objetivos globales para proteger el planeta y asegurar la prosperidad y la

paz para todos como parte de una nueva agenda de desarrollo sostenible. Cada objetivo tiene metas específicas que deben alcanzarse en los próximos 15 años.

De los objetivos expuestos en la Tabla A.1, el proyecto de este TFG se relaciona con los siguientes:

- **Industria, innovación e infraestructuras**.

  La investigación de técnicas de inyección de fallos en microcontroladores busca mejorar la seguridad y la fiabilidad de los sistemas embebidos. Estos sistemas son fundamentales en muchas infraestructuras como automóviles, dispositivos médicos, sistemas industriales o redes de telecomunicaciones.

  Además, mejorar la detección y mitigación de fallos en microcontroladores ayuda a desarrollar sistemas más resilientes y asegura el correcto funcionamiento de los sistemas embebidos proporcionando la seguridad y robustez necesaria para la adopción masiva de tecnologías avanzadas en la industria.

- **Paz, justicia e instituciones sólidas**.

  La inyección de fallos de voltaje es una técnica utilizada en la seguridad informática para probar la robustez de los microcontroladores frente a posibles ataques. Asegurar que estos dispositivos sean seguros y confiables ayuda a proteger la información crítica y garantizar la seguridad de las instituciones y las personas.

  Al contribuir al desarrollo de sistemas más seguros se está ayudando a prevenir acciones que podrían desestabilizar instituciones, adectando la paz y la justicia en las sociedades. Un microcontrolador comprometido podría tener consecuencias graves en procesos electorales, redes de comunicación seguras o sistemas financieros. Al mejorar la seguridad de estos dispositivos se fortalece la confianza en las instituciones y se protege el bienestar social.

- **Igualdad de género**.

  A causa de la gran desigualdad de género experimentada actualmente en el sector informático, cualquier proyecto de divulgación de información dentro del sector, que pueda resultar interesante tanto para mujeres como a hombres, resulta en una pequeña contribución a reducir la brecha de género.

ANEXO B
Código primera prueba

Código Arduino Uno (Target), fiesta.hex:

```
:100000000C9434000C9451000C9451000C94510049
:100010000C9451000C9451000C9451000C9451001C
:100020000C9451000C9451000C9451000C9451000C
:100030000C9451000C9451000C9451000C945100FC
:100040000C9451000C9451000C9451000C945100EC
:100050000C9451000C9451000C9451000C945100DC
:100060000C9451000C94510011241FBECFEFD8E026
:10007000DEBFCDBF11E0A0E0B1E0E8EFF7E002C0E5
:1000800005900D92AE34B107D9F721E0AEE4B1E0AE
:1000900001C01D92A63CB207E1F70E94CD000C946E
:1000A000FA030C940000CF93DF9300D0CDB7DEB7F6
:1000B000CE0109969A83898329813A818F81988517
:1000C000AC0168E770E08EE491E00E9419018EE4D3
:1000D00091E00E94AF000F900F90DF91CF910895B3
:1000E000CF93DF9300D0CDB7DEB783E390E09A8360
:1000F000898385EC90E029813A81232F3327FC0105
:10010000208384EC90E02981FC01208381EC90E045
:1001100028E1FC01208382EC90E026E0FC012083B2
:100120000F900F90DF91CF910895CF93DF931F929F
:10013000CDB7DEB78983000080EC90E0FC018081C0
:10014000882F90E080729927892BB1F386EC90E09C
:100150002981FC0120830F90DF91CF910895CF93E7
:10016000DF9300D0CDB7DEB79A8389830BC0898136
:100170009A819C012F5F3F4F3A832983FC01808144
:100180000E94950089819A81FC018081882379F7FA
:100190000F900F90DF91CF910895CF93DF9300D010
:1001A0001F92CDB7DEB71B821A8219820E9470009F
:1001B00080E091E0892F8F9380E091E08F930E94FF
:1001C00053000F900F9014C089819A8101969A83F1
:1001D000898389819A81892B59F483E291E0892F5F
:1001E0008F9383E291E08F930E9453000F900F90C2
:1001F0008B81882349F388E291E0892F8F9388E2ED
:1002000091E08F930E9453000F900F9088E391E04C
:10021000892F8F9388E391E08F930E9453000F9072
:100220000F9080E090E00F900F900F90DF91CF91B2
:100230000895AEE0B0E0EFE1F1E00C94D1038C0161
:10024000CA0146E04C831A83098377FF02C060E04D
:1002500070E8FB013197FE83ED83A901BC01CE015B
:1002600001960E9445014D815E8157FD0AC02F8194
:10027000388542175307 0CF49A01F801E20FF31F77
:1002800010822E96E4E00C94ED03ACE0B0E0EBE4D9
:10029000F1E00C94C3037C016B018A01FC0117821D
:1002A0001682838181FFBDC1CE0101964C01F70109
:1002B0009381F60193FD859193FF81916F018823CE
:1002C00009F4ABC1853239F493FD859193FF819197
:1002D0006F01853229F4B70190E00E942D03E7CF2A
```

```
:1002E000512C312C20E02032A0F48B3269F030F414
:1002F000803259F0833269F420612CC08D3239F09C
:10030000803339F4216026C02260246023C0286035
:1003100021C027FD27C030ED380F3A3078F426FF92
:1003200006C0FAE05F9E300D1124532E13C08AE000
:10033000389E300D1124332E20620CC08E3221F4F1
:1003400026FD6BC1206406C08C3611F4206802C003
:10035000883641F4F60193FD859193FF81916F01F9
:100360008111C1CF982F9F7D9554933028F40C5F55
:100370001F4FFFE3F9830DC0833631F0833771F0EF
:10038000833509F05BC022C0F801808189830E5F4C
:100390001F4F44244394512C540115C03801F2E0FE
:1003A0006F0E711CF801A080B18026FF03C0652D7F
:1003B00070E002C06FEF7FEFC5012C870E9422031F
:1003C0002C0183012C852F77222E17C03801F2E0F3
:1003D0006F0E711CF801A080B18026FF03C0652D4F
:1003E00070E002C06FEF7FEFC5012C870E941703FA
:1003F0002C012C852068222E830123FC1BC0832D19
:1004000090E048165906B0F4B70180E290E00E94EF
:100410002D033A94F4CFF50127FC859127FE8191B5
:100420005F01B70190E00E942D0331103A94F1E092
:100430004F1A51084114510471F7E5C0843611F088
:10044000893639F5F80127FF07C060817181828103
:1004500093810C5F1F4F08C060817181072E000CD3
:10046000880B990B0E5F1F4F2F76722E97FF09C0D6
:1004700090958095709561957F4F8F4F9F4F206825
:10048000722E2AE030E0A4010E946503A82EA8186D
:1004900044C0853729F42F7EB22E2AE030E025C0F3
:1004A000F22FF97FBF2E8F36C1F018F4883579F01E
:1004B000B4C0803719F0883721F0AFC02F2F2061EA
:1004C000B22EB4FE0DC08B2D8460B82E09C024FF5F
:1004D0000AC09F2F9660B92E06C028E030E005C004
:1004E00020E130E002C020E132E0F801B7FE07C0B1
:1004F000608171818281938 10C5F1F4F06C0608192
:10050000718180E090E00E5F1F4FA4010E9465039F
:10051000A82EA818FB2DFF777F2E76FE0BC0372D57
:100520003E7FA51450F474FE0AC072FC08C0372D3B
:100530003E7E05C0BA2C372D03C0BA2C01C0B52CA5
:1005400034FF0DC0FE01EA0DF11D8081803311F4EE
:10055000397E09C032FF06C0B394B39404C0832F20
:10056000867809F0B39433FD13C030FF06C05A2CCF
:10057000B31418F4530C5B18B32CB31468F4B7011C
:1005800080E290E03C870E942D03B3943C85F5CF38
:10059000B31410F43B1801C0312C34FF12C0B70162
:1005A00080E390E03C870E942D033C8532FF17C01A
:1005B00031FD03C088E790E002C088E590E0B70114
:1005C0000CC0832F867859F031FF02C08BE201C046
:1005D00080E237FD8DE2B70190E00E942D03A51463
:1005E00038F4B70180E390E00E942D035A94F7CFCE
:1005F000AA94F401EA0DF11D8081B70190E00E94F8
:100600002D03A110F5CF332009F451CEB70180E2BC
:1006100090E00E942D033A94F6CFF70186819781EE
:1006200002C08FEF9FEF2C96E2E10C94DF03FC01F8
:10063000059061507040011 0D8F7809590958E0F0D
:100640009F1F0895FC0161507040019001 10D8F780
:10065000809590958E0F9F1F08950F931F93CF93B2
:10066000DF93FB01238121FD03C08FEF9FEF28C0A3
:1006700022FF16C0468157812481358142175307D6
:1006800044F4A081B1819D012F5F3F4F31832083CE
:100690008C93268137812F5F3F4F3783268310C08D
:1006A000EB01092F182F0084F185E02D0995892B86
:1006B000E1F68E819F8101969F838E83812F902FFB
:1006C000DF91CF911F910F910895FA01AA27283049
:1006D00051F1203181F1E8946F936E7F6E5F7F4F0F
```

```
:1006E0008F4F9F4FAF4FB1E03ED0B4E03CD0670F8B
:1006F000781F891F9A1FA11D680F791F8A1F911DDE
:10070000A11D6A0F711D811D911DA11D20D009F42D
:1007100068943F912AE0269F11243019305D31936F
:10072000DEF6CF010895462F4770405D4193B3E058
:100730000FD0C9F7F6CF462F4F70405D4A3318F0FF
:10074000495D31FD4052419302D0A9F7EACFB4E0B0
:10075000A695979587957956795BA95C9F7009768
:1007600061057105089590B01AC010A2E0694579509
:10077000479537952795BA95C9F7620F731F841F60
:10078000951FA01D08952F923F924F925F926F92F6
:100790007F928F929F92AF92BF92CF92DF92EF9211
:1007A000FF920F931F93CF93DF93CDB7DEB7CA1B92
:1007B000DB0B0FB6F894DEBF0FBECDBF09942A88BD
:1007C000398848885F846E847D848C849B84AA8465
:1007D000B984C884DF80EE80FD800C811B81AA81F2
:1007E000B981CE0FD11D0FB6F894DEBF0FBECDBFBD
:0807F000ED010895F894FFCF1C
:1007F80052484D4532204649206C6576656C20315B
:100808002E0D0A0D0A436869702073746174757 33C
:100818003A20004C6F636B004368697020756E6CFA
:100828006F636B65640D0A00464C41473A20576870
:0E083800795F346D5F495F686572333F000081
:00000001FF
```

Listing B.1: Código Arduino Uno (Target)

Código Arduino Mega (Host):

```
#define GLITCHING_PIN 2
int glitching_duration =  0;
int incomingByte = 0;

void setup() {
  Serial.begin(19200);
  Serial.println("Arduino is ready");

  pinMode(GLITCHING_PIN, OUTPUT);
  digitalWrite(GLITCHING_PIN, HIGH);
  delay(5000);

  Serial.println("Gliching is ready");
}

void glitch(){
  volatile int waste = 0;

  digitalWrite(GLITCHING_PIN, LOW);
  for (int i = 0; i<glitching_duration; i++){ waste++; }
  digitalWrite(GLITCHING_PIN, HIGH);

  glitching_duration +=10;
  Serial.println();
  Serial.print("Glich Delay set to: ");
  Serial.print(glitching_duration);
  Serial.println();
}

void loop() {
  for (int i = 0; i<200;i++){
      if (Serial.available() > 0) {
        incomingByte = Serial.read();
        Serial.print(char(incomingByte));
      }
    }
```

```
37      delay(1000);
38      glitch();
39  }
```

Listing B.2: Código Arduino Mega (Host)

ANEXO C
Código segunda prueba

Código Arduino Uno (Target):

```
#define cond 1

void setup() {
  Serial.begin(115200);
  Serial.write("INICIO");
  delay(1000);
}

void loop() {
  if(cond){
    Serial.write("Lock");
    delay(10);
  } else {
    Serial.write("DESBLOQUEADO");
    delay(5000);
  }
}
```

Listing C.1: Código Arduino Uno (Target)

Código Arduino Mega (Host):

```
#define GLITCHING_PIN 2
#define GLITCHING_ITERATIONS 500
#define GLITCHING_DURATION 20
#define GLITCHING_INTERVAL 1000

void setup() {
  pinMode(GLITCHING_PIN, OUTPUT);
  digitalWrite(GLITCHING_PIN, HIGH);
}

void loop() {
    for(int i = 0 ; i < GLITCHING_ITERATIONS ; i++) {
      digitalWrite(GLITCHING_PIN, LOW);
      delayMicroseconds(GLITCHING_DURATION);        //Glitch duration
      digitalWrite(GLITCHING_PIN, HIGH);
    }
    delay(GLITCHING_INTERVAL);
}
```

Listing C.2: Código Arduino Mega (Host)

Código Arduino Mega (Host). Tiempo de *glitch* aleatorio:

```
#define GLITCHING_PIN 2
```

```
#define GLITCHING_ITERATIONS 500
#define MAX_GLITCHING_DURATION 25
#define GLITCHING_INTERVAL 1000

int rand = 1;

void setup() {
  pinMode(GLITCHING_PIN, OUTPUT);
  digitalWrite(GLITCHING_PIN, HIGH);
}

void loop() {
    rand = random(1, MAX_GLITCHING_DURATION);

    for(int i = 0 ; i < GLITCHING_ITERATIONS ; i++) {
      digitalWrite(GLITCHING_PIN, LOW);
      delayMicroseconds(rand);                //Glitch duration
      digitalWrite(GLITCHING_PIN, HIGH);
    }
    delay(GLITCHING_INTERVAL);
}
```

Listing C.3: Código Arduino Mega (Host)

Código Arduino Mega (Reader):

```
int incomingByte = 0;

void setup() {
  Serial.begin(19200);
  Serial1.begin(115200);

  Serial.println("Arduino is reading");
  delay(2000);
}

void loop() {
  if (Serial1.available() > 0) {
    incomingByte = Serial1.read();
    Serial.print(char(incomingByte));

    //Glitch exitoso
    if (char(incomingByte) == 'D'){ //*DESBLOQUEADO*
      Serial.println("\nConseguido");
      delay(10000);
    }

    //Reinicio
    if(char(incomingByte)=='O'){ //*INICIO*
      delay(2000);
    }
  }
}
```

Listing C.4: Código Arduino Mega (Reader)

ANEXO D
Código tercera y cuarta prueba

Código Arduino Uno (Target):

```
int i, ctr=0;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  start();
}

void start(){
  //Parpadeo que indica el reinicio
  digitalWrite(LED_BUILTIN, LOW);
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);
  delay(100);
  digitalWrite(LED_BUILTIN, HIGH);
  delay(100);
  digitalWrite(LED_BUILTIN, LOW);
  delay(100);
}

void loop() {
  digitalWrite(LED_BUILTIN, LOW);

  for(i=0; i<10000; i++){
    ctr++;
  }

  if(ctr == 10000){
    ctr = 0;
  } else {
    digitalWrite(LED_BUILTIN, HIGH);
  }
}
```

Listing D.1: Código Arduino Uno (Target)

Código Arduino Mega (Host): listado C.2 y listado C.3.