POLITECNICO
MILANO 1863

Progetto di Ingegneria Informatica
FPGATriggerer

Anno Accademico 2023/2024

Periodo:	Marzo-Maggio
Tutor:	Prof. Luca Oddone Breveglieri
Azienda:	Micron Semiconductor
Supervisor:	Niccolò Izzo

Indice

1 Abstract

Le tecniche di iniezione a guasti fanno parte della categoria degli attacchi attivi di tipo *side-channel*, queste infatti, permettono di alterare il normale flusso di esecuzione di un dispositivo inducendo in esso delle interferenze, che possono essere sulla linea di alimentazione oppure al campo elettromagnetico circostante il dispositivo. Questo genere di attacchi, generalmente, hanno come obiettivo quello di saltare dei controlli di sicurezza o creare dei canali secondari che permettano, in una seconda fase, l'estrazione di dati sensibili, inclusi eventuali firmware. Per fare ciò andranno eseguite una serie di operazioni in modo molto preciso e per questo è necessario un dispositivo programmabile che orchestri tutto ciò.

2 Introduzione

2.1 Obiettivo

Lo scopo del progetto consiste nella ricerca e nello sviluppo di un sistema, che sia in grado di rispondere alle specifiche, utilizzando prodotti off-the-shelf in modo che si possano sensibilmente abbassare i costi del setup, che spesso è necessario parallelizzare su più unità.
Inoltre, essendo l'argomento ancora poco esplorato, principalmente per motivi economici dovuti ad alti costi di dispositivi e licenze, questo progetto è interamente composto da software libero, con un'occhio di riguardo al mondo accademico.

2.2 Specifiche

Il sistema deve ricevere in ingresso un segnale che possa essere digitale o analogico, che lo attivi per eseguire le varie operazioni.
Tra le operazioni che il sistema deve essere in grado di eseguire vi sono:

- Ritardo ad alta precisione: attendere un periodo specificato in cicli di clock che abbia una granularità di 5/6ns.
- Ritardo configurabile: il ritardo specificato precedentemente deve essere lungo almeno 100ms.
- Poche risorse logiche: nell'ottica dell'integrazione di questo sistema in uno più complesso si preferisce un'implementazione che usi poche risorse logiche(LUT, ALU, REGs).
- Uscita per reset del Device Under Test.
- Periodo di holdoff: dopo aver lanciato un guasto si vuole attendere che il sistema si ristabilizzi prima di rieseguire l'attacco.

Funzionalità avanzate:

- Convertitore Analogico Digitale: permette di leggere segnali di ingresso analogici e abilita la funzione successiva.
- Pattern Matching: permette di attivare il sistema a fronte di un segnale pattern rilevato in ingresso.

2.3 Lavori correlati

Le tecniche qui citate, in particolare iniezione di guasti sull'alimentazione, ma anche glitch sul segnale di clock e interferenze elettromagnetiche, sono tecniche per cui il nostro dispositivo può essere usato e che non vengono descritte approfonditamente in questo documento.
Dettagli in merito si possono trovare qui [1], [2], [3].

Il prototipo da me creato prende ispirazione da un prodotto open-source che si presta molto bene a questo genere di attacchi che è il ChipWhisperer-Husky[1], ma anche dal Riscure icWaves[2] a cui ho fatto riferimento per stilare le specifiche.

2.3.1 Prodotti a confronto

Per quanto riguarda i prodotti precedentemente citati, faccio un confronto tra di loro, in particolare per alcune metriche fondamentali.

Metrica	Nostro prototipo	Riscure icWaves	ChipWhisperer Husky
Sample to trigger	34ns	250ns	?
Delay	0-100ms	0-100ms	0-21s
Granularity	6ns	5ns	5ns
Jitter	± 1.2ns	± 120ns	?
Price	30$	**~46k$**	550$

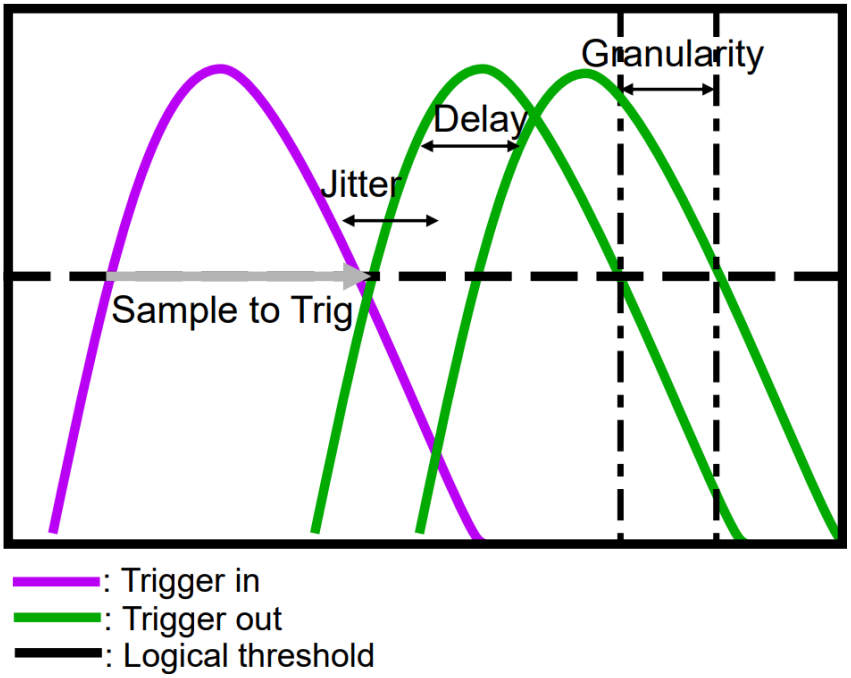

Figura 1: Legenda metriche

[1] https://rtfm.newae.com/Capture/ChipWhisperer-Husky/
[2] https://www.riscure.com/products/icwaves/

3 Fase di ricerca

La fase preponderante del progetto è stata quella di ricerca, ho impiegato infatti circa un mese e mezzo di lavoro part-time(20h/settimana) per studiare i prodotti già esistenti, fare delle bozze di architetture e cercare i componenti che ci avrebbero permesso di soddisfare le specifiche.

Uno dei problemi che si incontrano quando si vuole creare un prodotto a basso costo è che i componenti che si trovano non sempre sono ben supportati e documentati. Questo mi ha portato all'uso di una scheda che poi si è rivelata non compatibile con la toolchain open source che avremmo voluto usare.

Dopo un paio di settimane mi sono reso conto che sarebbe stato troppo oneroso aggiungere il supporto per la nostra scheda e ne ho procurata un'altra già supportata.

La fase di ricerca è stata fondamentale anche per quanto riguarda le architetture hardware da usare sull'FPGA per ottimizzare i tempi e le risorse.

Dopo aver definito le velocità raggiungibili e le architetture da usare ho iniziato a creare vari prototipi per capire effettivamente come rispondesse l'hardware, dato che, quando si lavora ad alte frequenze(necessarie per ottenere una maggiore precisione) si creano dei disturbi non graditi.

4 Progettazione

4.1 Hardware

Per soddisfare le prestazioni richieste si è pensato all'uso di un FPGA, inoltre per indurre il guasto alla tensione, è stata adottata la tecnica della *crowbar*[2], che permette di interrompere per un brevissimo periodo l'alimentazione al dispositivo.

4.2 Architettura del dispositivo

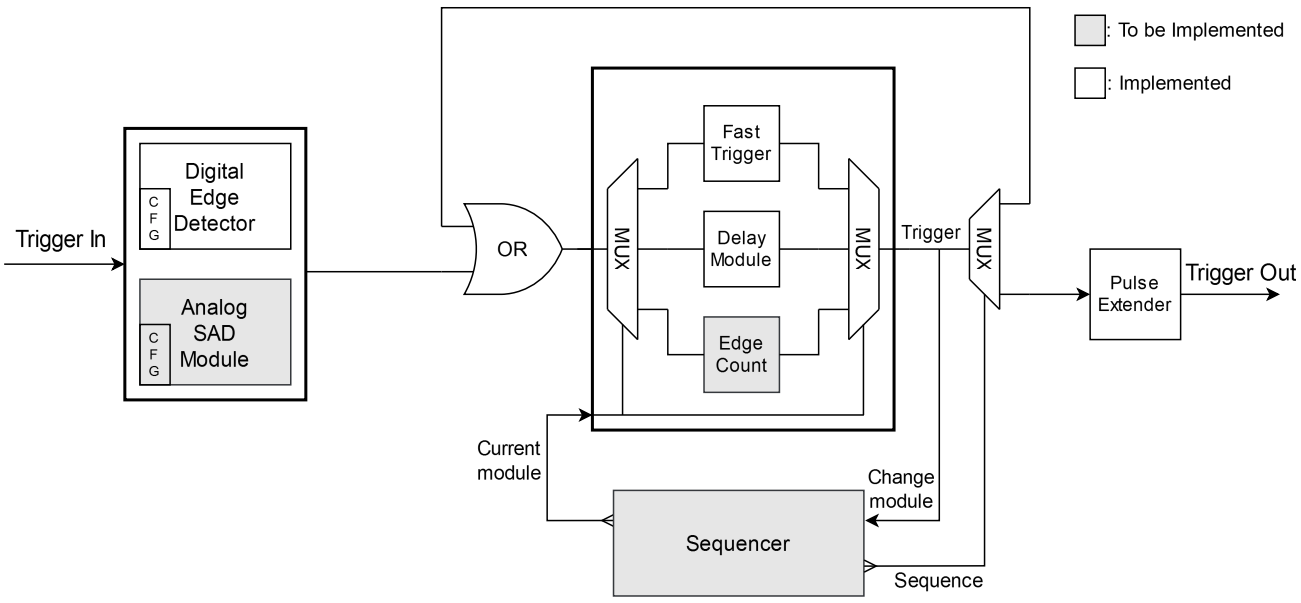

Figura 2: Architettura a blocchi del dispositivo

Quando si progetta un'architettura per FPGA si fa spesso uso di modelli a blocchi, anche in questo caso l'architettura è stata pensata a blocchi per vari motivi tra cui:
- facilità di integrazione in altre architetture già esistenti
- scalabilità e flessibilità
- compartimentalizzazione

L'architettura in Figura 2 è suddivisa in tre stadi:

1. Stadio di **acquisizione**: Riceve in ingresso un segnale digitale o analogico e lo passa ai moduli interni che dopo aver effettuato delle operazioni lanciano in uscita un segnale di trigger per il prossimo stadio.
   - **Digital Edge Detector**: In base alla configurazione interna, attiva l'uscita sul fronte di salita o sul fronte di discesa del segnale di ingresso.
   - **Analog SAD Module**: Si tratta di un modulo che al suo interno contiene un convertitore analogico-digitale ad alta frequenza in grado di elaborare, appunto, segnali analogici, ciò consente ad esempio di rilevare un livello specifico in ingresso oppure riconoscere un pattern(pattern matching)[4], attraverso la tecnica **SAD**(Sum of Absolute Difference).
2. Stadio di **elabolazione**: Lo stadio di elaborazione si trova nella parte centrale del diagramma, è anch'essa composta da dei moduli che prendono in ingresso il trigger e vi applicano un'operazione, ad esempio ritardare il segnale(delay module) o contare i fronti di salita(edge counter), prima di lanciare un'impulso in uscita. I moduli possono essere eseguiti in serie attraverso il **sequencer** che, previa configurazione, esegue in cascata i vari moduli.
3. Stadio di **uscita**: Si occupa di effettuare le ultime operazioni sul segnale proveniente dallo stadio di elaborazione. Nell'attuale architettura si occupa di estendere la durata del trigger in uscita, da un ciclo di clock a più cicli. Un'altra funzione di cui si potrebbe occupare è quella di generare un segnale o pattern analogico, attraverso l'uso di un convertitore digitale-analogico(DAC).

4.3 Software

La parte software del progetto consiste in un API Python, descritta nel paragrafo successivo, che nasconde all'utilizzatore finale la complessità di interazione attraverso il protocollo documentato nel paragrafo 4.4, utilizzato sulla porta seriale UART.

4.3.1 Python API

Per rendere più semplice la configurazione del dispositivo e per permettere di creare script Python che generino dei grafici con risultati o altre statistiche, è stata creata una mini libreria che può essere importata direttamente nel progetto corrente.
A seguire il diagramma UML delle classi.

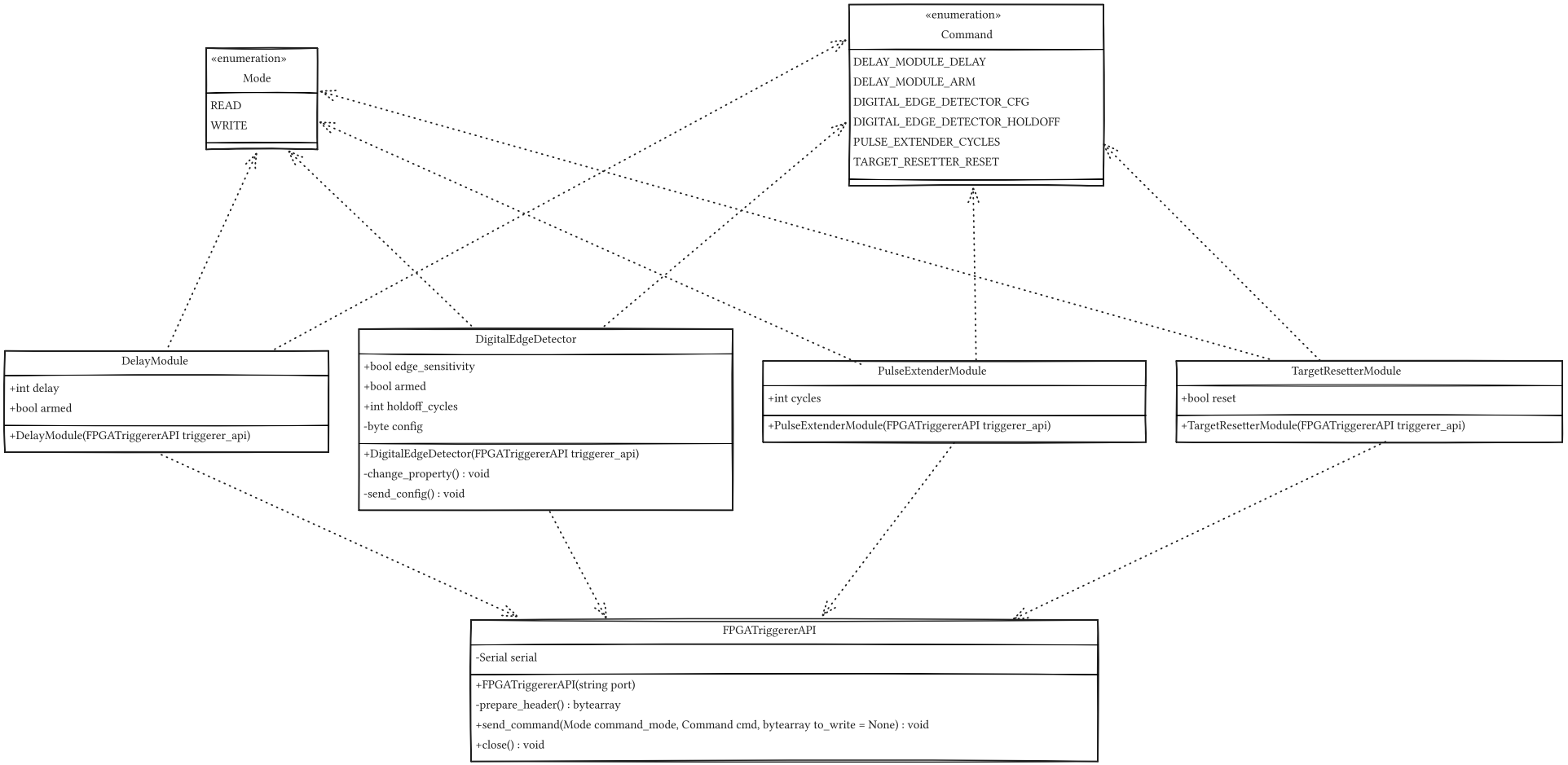

Figura 3: Diagramma delle classi dell'API

4.4 Protocollo di comunicazione

Quando si lavora con questo genere di dispositivi è necessario variare a runtime i parametri, ad esempio la durata del guasto oppure il periodo di holdoff.
Dato che bisogna variare questi parametri svariate volte al secondo non si possono usare pulsanti e manopole, ma è necessaria un'interazione dispositivo-macchina, si è pensato quindi all'uso di un protocollo ad-hoc molto semplice, trasmesso su porta seriale UART.

4.4.1 Documentazione

Il protocollo è composto da **comandi** che possono essere di **scrittura** o **lettura**.
Ogni comando legge o scrive il valore in un registro che può essere più o meno grande, in base alla specifica.

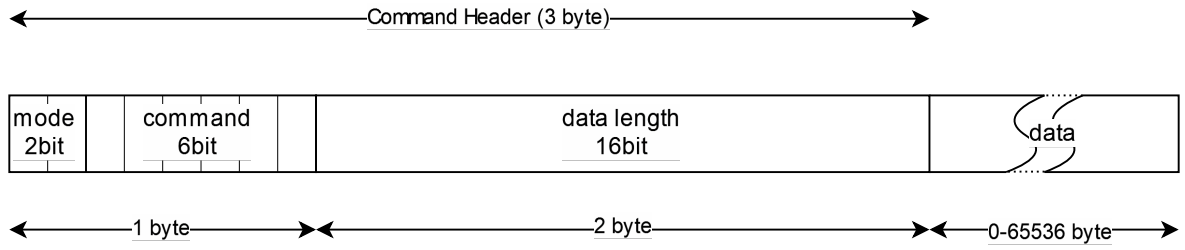

Figura 4: Struttura del pacchetto che viene trasmesso al dispositivo

- mode: Modalità del comando
  - 10: Lettura
  - 11: Scrittura
- command: Operazione da eseguire. Uno dei **Command ID** specificati nella tabella seguente
- data length: Lunghezza dei dati che il dispositivo legge(scrive) dal(sul) registro interno
- data: Dati codificati in **Little-endian** da scrivere nel registro, campo non presente in lettura

Ogni **"Command Name"** inizia con il nome del modulo che comanda.
Esempio: *DELAY_MODULE_DELAY* fa parte del modulo ***DELAY MODULE***.
I comandi attualmente implementati sono descritti nella seguente tabella.

Command ID (decimal)	**Command Name**	**Register Size**
`0d8`	DELAY_MODULE_DELAY	3 byte
`0d9`	DELAY_MODULE_ARM	1 byte
`0d16`	DIGITAL_EDGE_DETECTOR_CFG	1 byte
`0d17`	DIGITAL_EDGE_DETECTOR_HOLDOFF	2 byte
`0d24`	PULSE_EXTENDER_CYCLES	2 byte
`0d32`	TARGET_RESETTER_RESET	1 byte

4.4.1.1 DELAY_MODULE_DELAY

Registro da 3 byte contenente un numero intero codificato in big-endian rappresentante il numero di cicli di clock da attendere prima di lanciare un impulso in uscita.

4.4.1.2 DELAY_MODULE_ARM

Registro da 1 byte.

X	X	X	X	X	X	X	A

- X: Non utilizzato
- A: Armato
  - 1: Il dispositivo è armato e risponde agli ingressi.
  - 0: Il dispositivo è disarmato, valore di default dopo l'accensione.

4.4.1.3 DIGITAL_EDGE_DETECTOR_CFG

Registro da 1 byte. Contiene la configurazione del modulo.

X	X	X	X	X	X	X	E

- X: Non utilizzato
- E: Edge Sensitivity
  - 1: Da alto a basso, il modulo si attiva sul fronte di discesa del segnale di ingresso.
  - 0: Da basso ad alto, il modulo si attiva sul fronte di salita del segnale di ingresso.

4.4.1.4 DIGITAL_EDGE_DETECTOR_HOLDOFF

Registro da 3 byte contenente un numero intero codificato in big-endian rappresentante il numero di cicli di clock per cui ignorare il segnale di ingresso.

4.4.1.5 PULSE_EXTENDER_CYCLES

Registro da 2 byte contenente un numero intero codificato in big-endian rappresentante il numero di cicli per cui il segnale in ingresso al modulo deve essere copiato sull'uscita.

4.4.1.6 TARGET_RESETTER_RESET

Registro da 1 byte.

X	X	X	X	X	X	X	R

- X: Non utilizzato
- R: Reset
  - 1: L'uscita di reset del dispositivo viene alzata ad 1 logico.
  - 0: L'uscita di reset del dispositivo viene abbassata a 0 logico.

5 Implementazione

5.1 Hardware

Trattandosi di un progetto di ricerca, per ridurre i costi di prototipazione, si è fatto uso di una scheda pronta all'uso(Tang Nano 9K)[3] che contiene un FPGA Gowin GW1NR-9, un converti-

[3] https://wiki.sipeed.com/hardware/en/tang/Tang-Nano-9K/Nano-9K.html

tore USB-JTAG per la programmazione dell'FPGA e una Flash da 32Mbit in cui memorizzare la bitstream ed eventuali impostazioni.

Come *crowbar*, è stato usato un N-Channel MOSFET(IRF8714)[4] ad alta velocità di risposta.

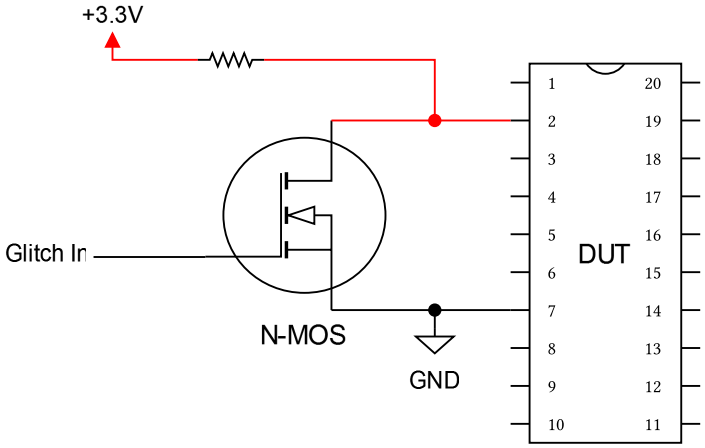

Figura 5: Circuito con DUT e Crowbar sulla sinistra

Quando il MOS viene stimolato crea un canale a bassa resistenza tra il polo positivo e il polo negativo, la corrente perciò preferisce percorrere la strada meno resistente e per questo il dispositivo non viene più alimentato. Per evitare di creare un corto circuito è stata inserita una resistenza a monte.

5.2 Software

La parte software specificata nella Sezione 4.3 è stata implementata seguendo il diagramma UML in Figura 3. Essendo molto semplice viene descritta con un esempio.

5.2.1 Esempio

Per interagire con il dispositivo, dopo averlo connesso al computer e ricavato la porta seriale, bisogna instanziare l'API nel seguente modo:

```
1 triggerer_serial_port = '/dev/ttyUSB1' #Linux
2 #triggerer_serial_port = 'COM5'        #Windows
3 triggerer = FPGATriggererAPI(triggerer_serial_port)
```

Dopo di che si potranno istanziare tutte le classi che rappresentano i vari moduli del dispositivo cioè:

- DelayModule
- DigitalEdgeDetector
- PulseExtenderModule
- TargetResetterModule

Ognuno di essi espone gli attributi documentati in Figura 3, leggendo o cambiando il loro valore, attraverso le *@property* di Python verranno inviati i corrispettivi comandi documentati nella Sezione 4.4, ciò rende molto comodo e semplice l'utilizzo del dispositivo.

[4] https://www.infineon.com/dgdl/irf8714pbf.pdf?fileId=5546d462533600a40153560d6f0f1d71

Dopo aver dichiarato la seguente classe **Manager**, che si occupa di condurre l'attacco:

```python
class Manager:
    def __init__(self, triggerer: FPGATriggererAPI, reset_enable_high: bool) -> None:
        self._triggerer = triggerer
        self._reset_enable_high = reset_enable_high

        self._edge_detector   = DigitalEdgeDetector(triggerer)
        self._delay_module    = DelayModule(triggerer)
        self._pulse_extender  = PulseExtenderModule(triggerer)
        self._target_resetter = TargetResetterModule(triggerer)

        self._edge_detector.edge_sensitivity = False
        # We don't want to reset the device on power up
        self._target_resetter.reset = not self._reset_enable_high

    def set_arm_value(self, value: bool):
        self._delay_module.armed = value

    def reset_dut(self, reset_duration: float):
        self._target_resetter.reset = True if self._reset_enable_high else False
        time.sleep(reset_duration)
        self._target_resetter.reset = not self._target_resetter.reset

    def set_delay(self, delay: int):
        self._delay_module.delay = delay

    def set_pulse_width(self, width: int):
        self._pulse_extender.cycles = width

    def set_holdoff_cycles(self, cycles: int):
        self._edge_detector.holdoff_cycles = cycles
```

Si può creare una funzione che si occupa di enumerare i vari parametri, eventualmente plottando i risultati su un grafico o usando un euristica per ridurre le combinazioni.

```python
def launch_fault_injection(triggerer: FPGATriggererAPI, manager: Manager, dut: DUT):
    manager.set_arm_value(True)
    manager.set_holdoff_cycles(1000)

    for delay in range(1, 20):
        manager.set_delay(delay)

        for pulse_width in range(1, 10, 2):
            manager.set_pulse_width(pulse_width)

            print('Trying parameters, delay={}, width={}'.format(delay, pulse_width))
            for retry in range(5):
                if check_dut_password(manager, dut):
                    print('Found correct parameters, delay={}, width={}'
                            .format(delay, pulse_width))

    manager.set_arm_value(False)
```

Quando verranno trovati i parametri corretti essi verranno stampati e il dispositivo verrà disarmato.
Si consiglia di usare il codice precedente solo come esempio trattandosi in alcuni casi di pseudo-codice.

5.3 Strumenti utilizzati

5.3.1 Software

Come citato nel Paragrafo 2.1(Obiettivo), si è preferito l'uso di strumenti FOSS(Free and Open Source Software), perciò abbiamo composto la seguente toolchain:

1. Verilog HDL: Linguaggio usato per descrivere a livello hardware il nostro dispositivo.
2. Yosys[5]: Suite di strumenti utili nella fase di sintesi, cioè la trasformazione del codice Verilog in un'insieme di componenti logici che realizzano la nostra funzione.
3. NextPnR[6]: Strumento che effettua la fase di place-and-route, cioè il piazzamento dei componenti logici e la connessione di essi.
4. Project Apicula[7]: Progetto che tramite reverse-engineering e fuzzing ha documentato e creato un software per mappare i vari componenti logici alle primitive offerte dagli FPGA Gowin.
5. openFPGALoader[8]: Strumento necessario a caricare sull'FPGA, tramite JTAG, la bitstream creata dal software precedente.
6. API Python: l'API descritta nel Paragrafo 4.3 per interagire con il dispositivo.

Attraverso l'uso di questi strumenti si è in grado di compilare il firmware del dispositivo, caricarlo su di esso e interagirci.

Un'altra strada alternativa, che è stata utile in alcuni casi in cui le primitive utilizzate nel codice Verilog non fossero state implementate da Project Apicula, è quella dell'utilizzo del Gowin IDE, cioè un ambiente di sviluppo ufficiale, creato per programmare gli FPGA Gowin. Purtroppo tale software non è FOSS, ma offre una licenza "educational".

5.3.2 Hardware

Durante la fase di sviluppo del prototipo sono stati condotti vari test atti a verificare il corretto funzionamento del codice e misurare le performance di esso, tutto ciò è stato condotto in un laboratorio protetto da ESD(Scariche elettrostatiche) usando vari strumenti tra cui:

- Alimentatore da banco Keysight EDU36311A: Per alimentare il DUT
- Generatore di funzioni Keysight 33511B: Usato per simulare il trigger-in tramite la forma d'onda PRBS(PseudoRandomBitStream)
- LeCroy WaveRunner 104XI-A: Oscilloscopio 4 Canali, 1GHz di banda, usato per misurare i tempi di risposta del dispositivo e la sincronizzazione dei vari segnali.
- Stazione saldante Weller: usata per saldare i pin sul dispositivo e aggiungere un filo per il reset del DUT.
- Multimetri vari

[5] https://github.com/YosysHQ/yosys
[6] https://github.com/YosysHQ/nextpnr
[7] https://github.com/YosysHQ/apicula
[8] https://github.com/trabucayre/openFPGALoader

6 Risultati

In Figura 6 si possono vedere i segnali coinvolti durante l'attacco, nello specifico il trigger-in dato dal dispositivo, il trigger-out che comanda l'N-MOS e la tensione di ingresso al dispositivo. Come si può vedere dalla fase in cui il regolatore di tensione del DUT cerca di compensare la caduta di tensione, il circuito con la *crowbar* ha funzionato.

È stato verificato l'effettivo funzionamento del **Delay Module** e del **Pulse Extender** ma anche del **Digital Edge Detector** che riconosce il trigger-in e del **Target Resetter** che resetta il DUT quando esso non risponde più ai comandi.

Purtroppo però, per ritardi nel reperimento di MOSFET effettivamente veloci, non è stato possibile saltare il controllo della password che era stato programmato sul DUT, inoltre a causa degli elevati picchi di tensione che possono essere raggiunti(20+ V) uno dei DUT utilizzati è stato danneggiato.
Si tratta infatti di un'attacco abbastanza invasivo che, enumerando svariate coppie di parametri aumenta la possibilià che il dispositivo si danneggi dato che lavora fuori dagli "Absolute Maximum Rating".

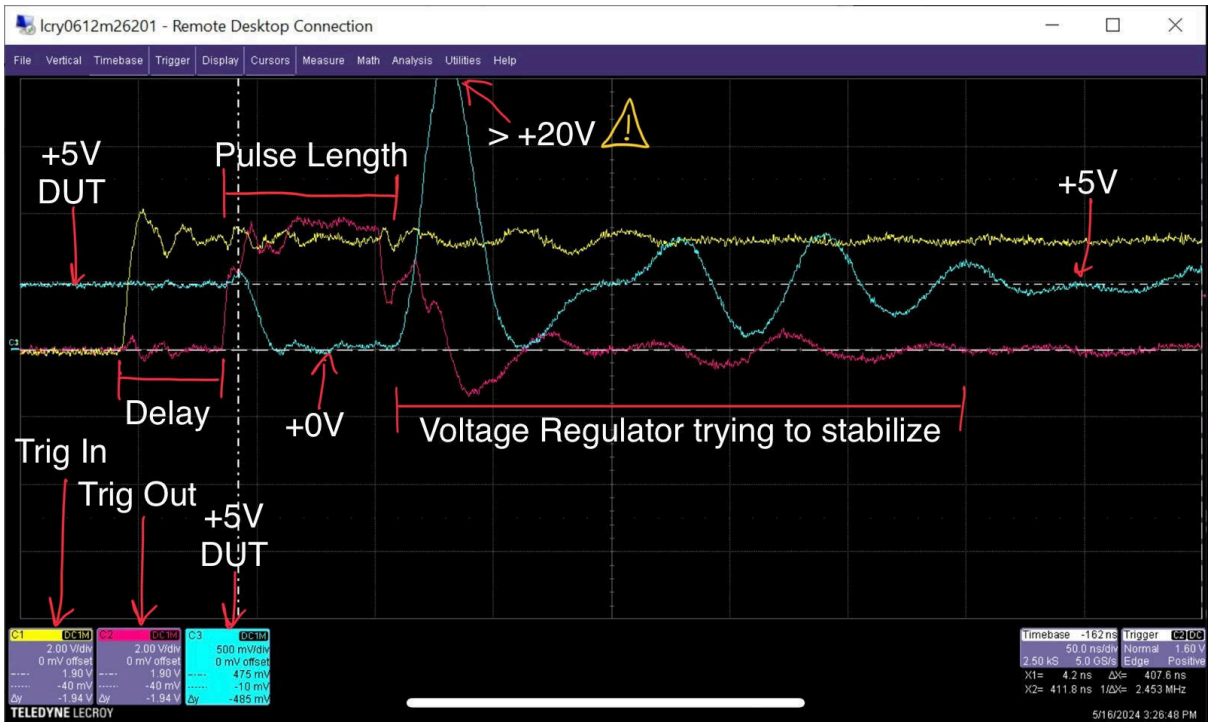

Figura 6: Esecuzione di un test sul setup documentato sotto

6.1 Setup

Il setup di prova, durante le varie misurazioni era composto da:

- Tang Nano 9K: Board con FPGA
- WaveShare ESP32-S3-Zero[9]: Il DUT, dispositivo sotto attacco
- N-MOS IRF8714
- Convertitore USB-Seriale CP2102: Per comunicare con il DUT
- LeCroy WaveRunner 104XI-A[10]: Oscilloscopio 4 Canali, 1GHz di banda
- Alimentatore da banco Keysight EDU36311A
- Generatore di funzioni Keysight 33511B

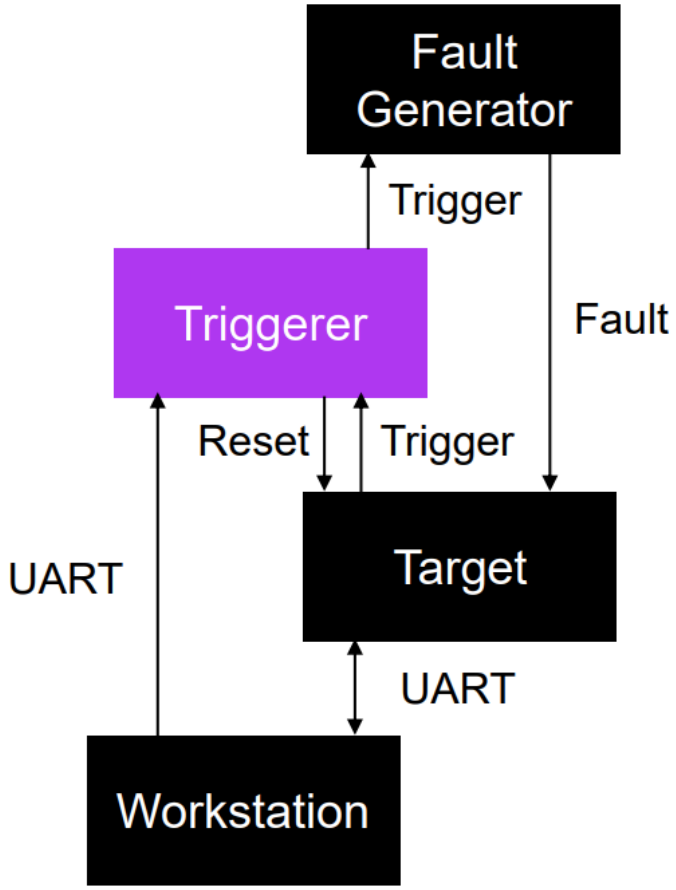

Figura 7: Setup schematico delle connessioni

[9] https://www.waveshare.com/wiki/ESP32-S3-Zero

[10] https://cdn.teledynelecroy.com/files/pdf/lecroy_waverunner_xi-a_specs.pdf

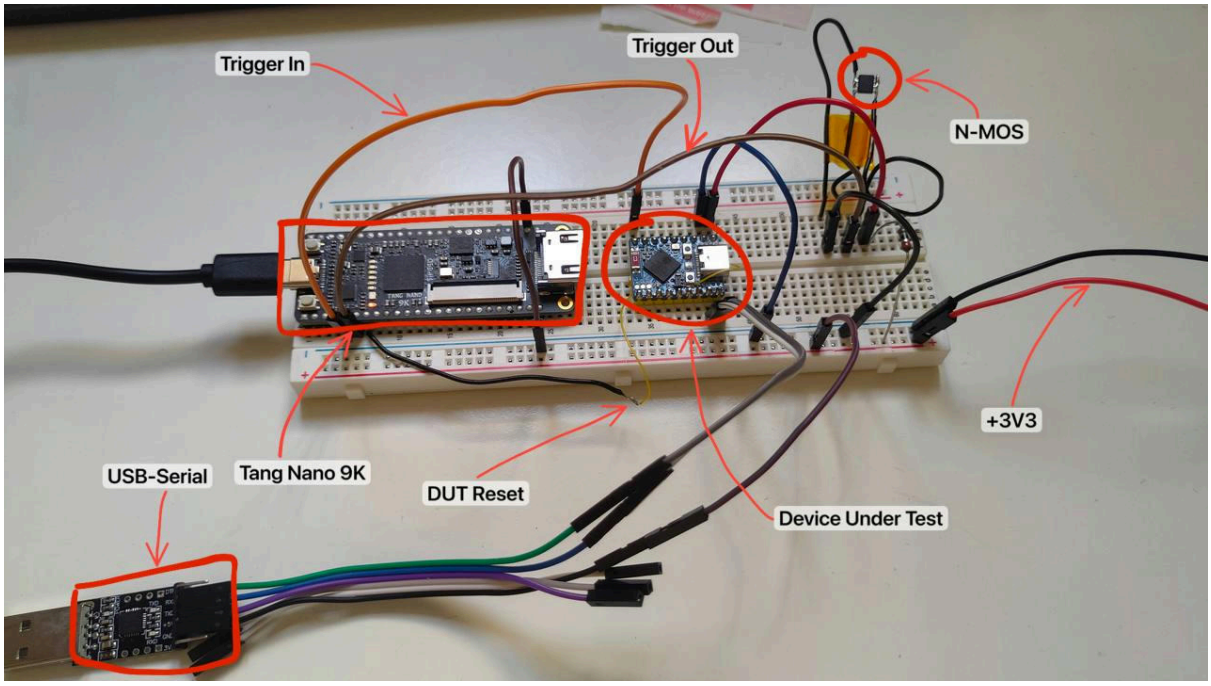

Figura 8: Setup con DUT

7 Conclusioni

L'obiettivo era quello di creare un dispositivo che fosse in grado di supportare la ricerca nell'ambito degli attacchi ad iniezione di guasti. Per fare ciò era necessario progettare un dispositivo che si basasse su hardware e strumenti che costassero poco e non avessero troppe licenze a pagamento.

Dopo aver stilato una lista di specifiche competitive con i prodotti attualmente presenti sul mercato è stato possibile ottenere il risultato desiderato utilizzando componenti off-the-shelf e software libero.

Come definito precedentemente, per rendere il dispositivo utilizzabile anche nell'ambito di attacchi side-channel sarebbe necessario integrare nel progetto, un convertitore analogico-digitale, ciò è già stato previsto ma per questioni di tempo non è stato implementato.

Il progetto infatti è ben lontano dall'essere completo, anche per questo esso sarà open-source e open-hardware, in modo che anche piccoli ricercatori indipendenti si possano interessare a queste tecniche, rendendo più sicure le architetture di dispositivi a rischio.

Bibliografia

[1] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures," *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056–3076, 2012, doi: 10.1109/JPROC.2012.2188769.

[2] C. O'Flynn, "Fault Injection using Crowbars on Embedded Systems." [Online]. Available: https://eprint.iacr.org/2016/810

[3] *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 2, pp. 199–224, Feb. 2019, doi: 10.13154/tches.v2019.i2.199-224.

[4] Y. Fukuda, K. Yoshida, and T. Fujino, "Fault Injection Attacks Utilizing Waveform Pattern Matching against Neural Networks Processing on Microcontroller," *IEICE TRANSACTIONS on Fundamentals*, no. 3, pp. 300–310, Mar. 2022, doi: 10.1587/transfun.2021CIP0015.