IIK-4100 Hardware Security of Embedded Systems

Fault Injection Experiment : 1-byte Fault Injection

NTNU
Norwegian University of Science and Technology

Table of Content

1. Introduction

Field-Programmable Gate Array (FPGA) is a special type of integrated circuit (IC) which allows the user to program his own custom digital circuit/architecture, after manufacturing, and change it at will [1]. This is by opposition to more "standard" computers where the logic architecture is fixed.

This exercise is made on the Hacking Hardware (Ha-Ha) board v3.0. It is a custom-designed FPGA board created for learning purposes [2]. It has been designed by Professor Swarup Bhunia and his collaborators and students, at the University of Florida.

The Ha-Ha v3.0 will allow us to experiment with fault injection. The latter is used in development and manufacturing by inducing fault to uncover and address potential faulty behaviors from the devices or software's. In hardware specifically, fault injection can take multiple forms with manipulation of voltage, clock or creating disturbances with flash, laser or even electromagnetic fields. But this concept can also be taken advantage of by attackers to alter normal functioning and/or steal data.

In this lab, we will focus on clock glitching with AES. Since some of the encryption operations are based on the clock to ensure proper coordination, interfering with the said clock will ultimately change the result of the encryption process.

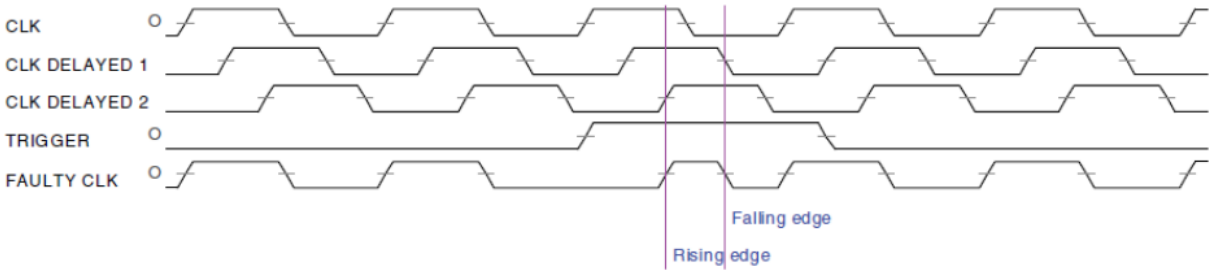
Our clock fault injection will consist of 1 bit switch on the cipher output of the 9^{th} round of AES. This change is small and localized enough to propagate and modify only 1 byte of the final cipher text. By implementing the fault in all 128 bits of the AES block, we can collect all the resulting "faulty cipher texts". With the genuine cipher text (without fault injection), and all the 128 faulty ones, it is possible to mathematically reconstruct the encryption key.

2. Background

2.1 Clock fault injection

The clock fault can be injected in multiple ways. In this lab, 2 delayed clock signals will be generated from the original one. A faulty clock signal will follow the genuine one but will be modified upon trigger. The trigger signal will modify the faulty clock one, the latter being modified by leveraging the 2 delayed clocks.

In our case, we will model the fault with the rising edge of the 2nd delayed clock, and the falling edge of the 1st one:



2.2 AES encryption

AES is a block cipher algorithm designed with the repetition of a set of fixed operations. The base algorithm has a block size of 128 bits and 10 rounds. But there are other versions with key sizes of 128/192/256 bits, and 10/12/14 rounds respectively.

Round 0	Round 1 to 9	Round 10
Plaintext block XORed with key to have starting value for round 1.	Each round has 4 operations: - Substitute bytes - Shift rows - Mix columns - Add round key	Same as before, but without mixing columns: - Substitute bytes - Shift rows - Add round key

In this lab, we will use the base 128 bits-key and 10 rounds version. Injecting a faulty bit in the output of the 9th round will propagate to 1 faulty byte in the final cipher output, after the 3 operations of the 10th round.

2.3 Differential Fault Analysis (DFA) concept

The differential fault analysis consists in injecting fault into cryptographic operations to reveal their internal states. In this lab, we will perform the single byte fault on AES.

For a given encryption, we will start by collecting the genuine cipher text. For the same encryption, we will then inject a single bit fault on the cipher output of the 9th round (called M9), which, by the design of AES, will result in a 1-byte difference for the final cipher text. After collecting all faulty cipher text from injecting the faulty bit in all 128 possible positions (AES block size), we can proceed with the mathematical reconstruction of the key.

The last AES round is only composed of the following operations:
SubBytes => Shiftrows => AddRoundKey

Thus, by definition, we have:

$$C_j = S(x) \oplus k_j \qquad \text{(correct encryption)}$$

$$D_j = S(x \oplus e_i) \oplus k_j \qquad \text{(faulty encryption)}$$

With…

C and **D** the genuine/faulty ciphertexts,
x the unknown byte of M9
k the round key,
j the byte index,
$\mathbf{e_i}$ the mask of the faulty bit,
S() the substitution operation.

N.B.: The ShiftRows operation is simply a byte permutation that is taken into account in "x". The given equations thus still hold true, x being of a different byte index than j.

By XORing both values, we have the following:

$$C_j \oplus D_j = \left(S(x) \oplus k_j\right) \oplus \left(S(x \oplus e_i) \oplus k_j\right)$$

Since XORing a value by itself cancels out, this trick allows us to get rid of the round key, effectively leaving:

$$C_j \oplus D_j = \mathrm{SubByte}(x) \oplus \mathrm{SubByte}(x \oplus e_i)$$

We have the left part of this equation from the collected data. As for the right part, we can simply brute-force x.

Multiple answers can verify this equation but by repeating it for all e_i's, only one value can verify all 8 equations at the same time: the right value of x. This process is to be repeated for all j's to reconstruct all bytes of x.

Once x is known, the output of the 9th round, we can easily retrieve the round 10 subkey. And by reversing the key scheduling, we can then retrieve the master key.

3. Laboratory experiment

3.1 Required material

- Ha-Ha v3.0 board
- USB A to B cable
- Computer with GOWIN FPGA Designer installed, version 1.98 or higher
- Python 3.x
- The provided Ha-Ha v3.0 config file
- The provided Python files (M9.py and key.py) and .txt template

3.2 Instructions

1) Download the .rao file and the configuration file which has your group number (ex: **group 3** downloads the **3.fs** file only, disregarding 1-2 and 4 to 8).

2) Follow along the tutorial in next section to retrieve the key

3) Write a report answering the following questions and upload it in pdf format on BlackBoard (named "Group_X_Fault_Injection.pdf", where X is your group number). Write all participants' names on the report. Here are the questions to answer:

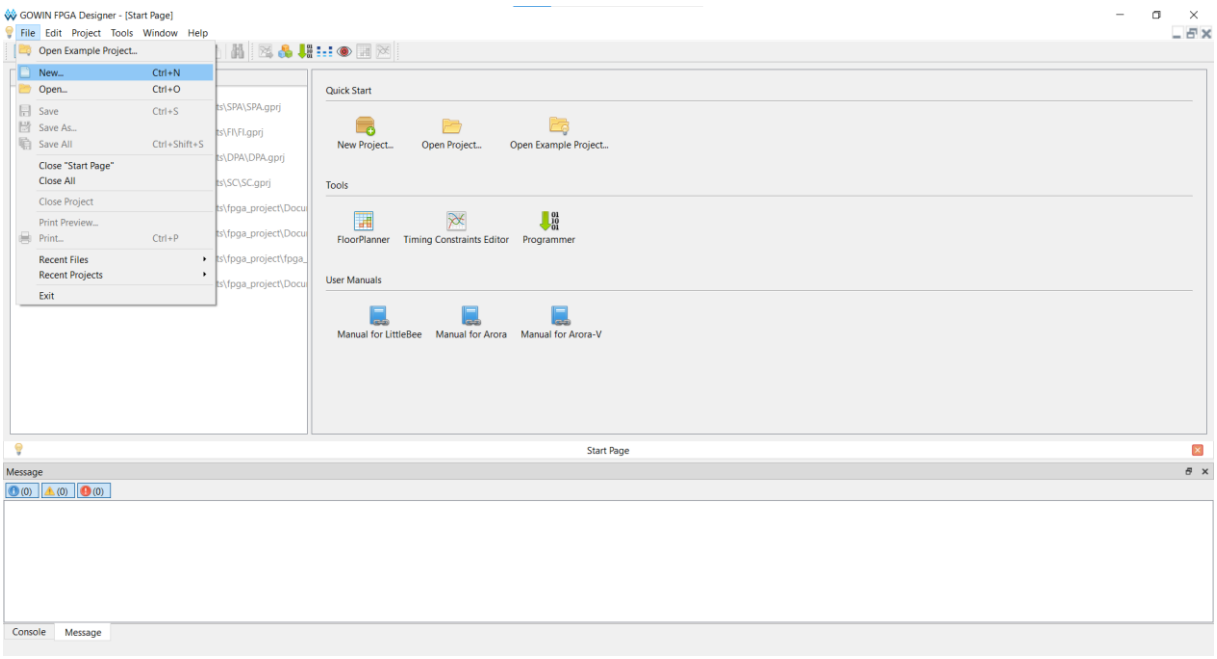
a. What is your genuine cipher text?
b. What is your M_9?
c. What is your K_{10}?
d. Submit the 128-bit AES key in hexadecimal format, no space.
e. In your own words, briefly explain why do we choose to inject a fault before the last encryption round? What's the difference between the last round and the other rounds?
f. List down your 128 faulty ciphertexts

3.3 Environment setup

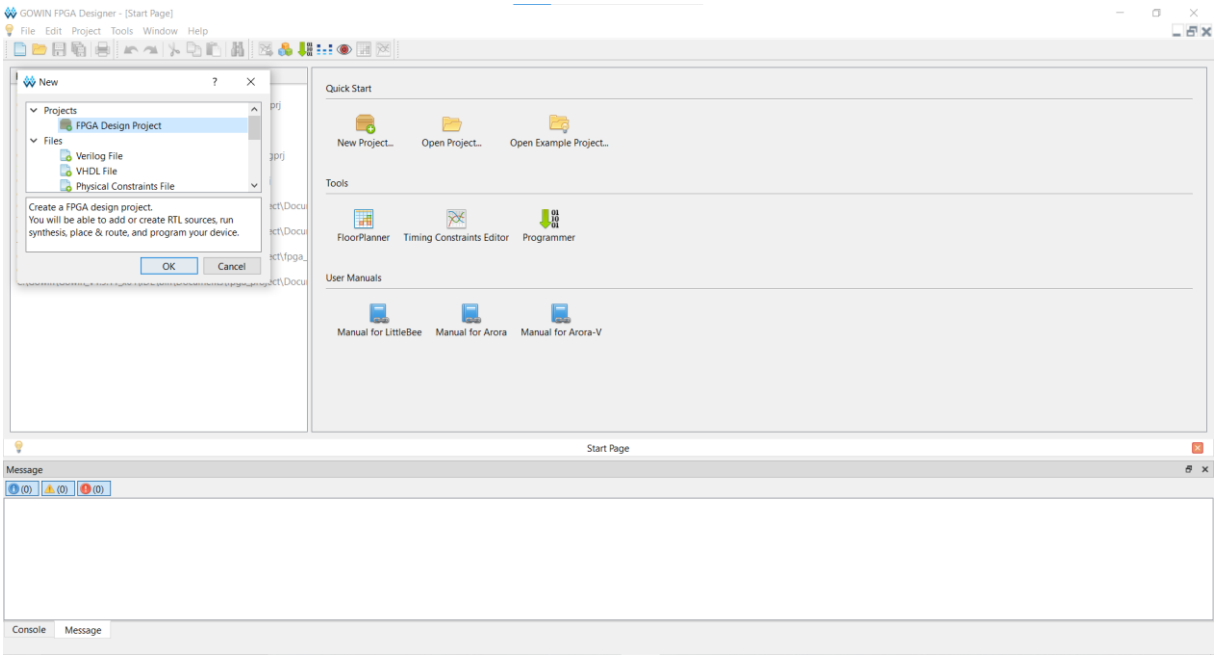
3.3.1 Setup the GAO

To proceed, start a new project on GOWIN FPGA Designer and follow along those steps:

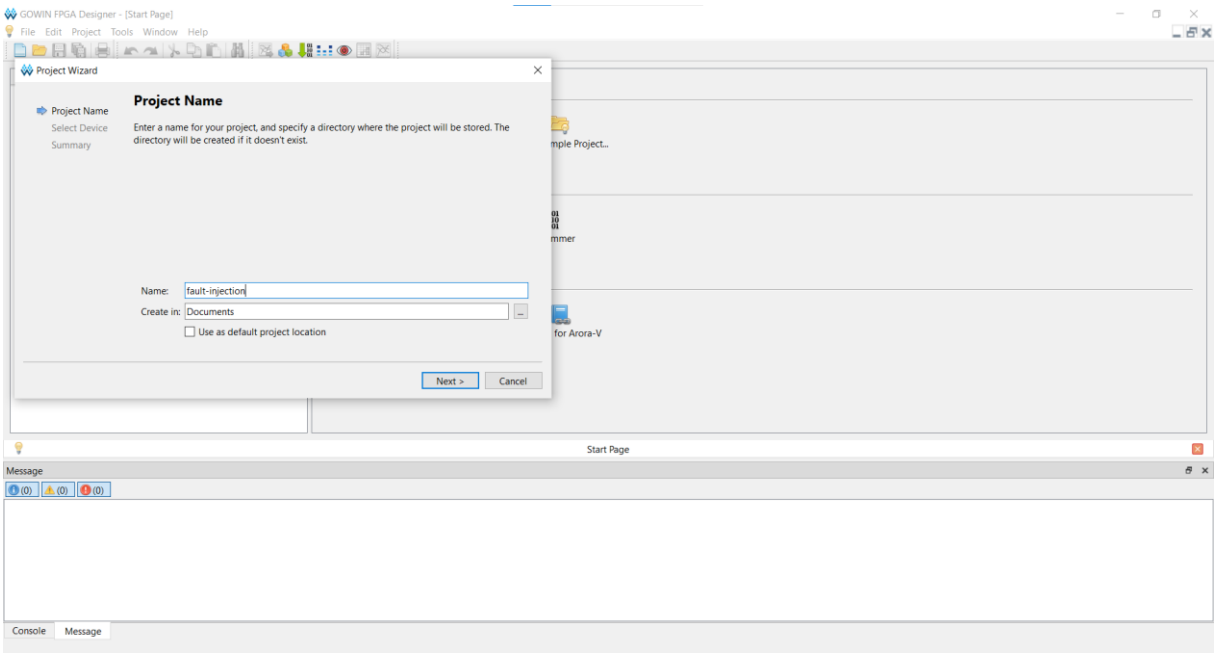
1) Go to File > **New**



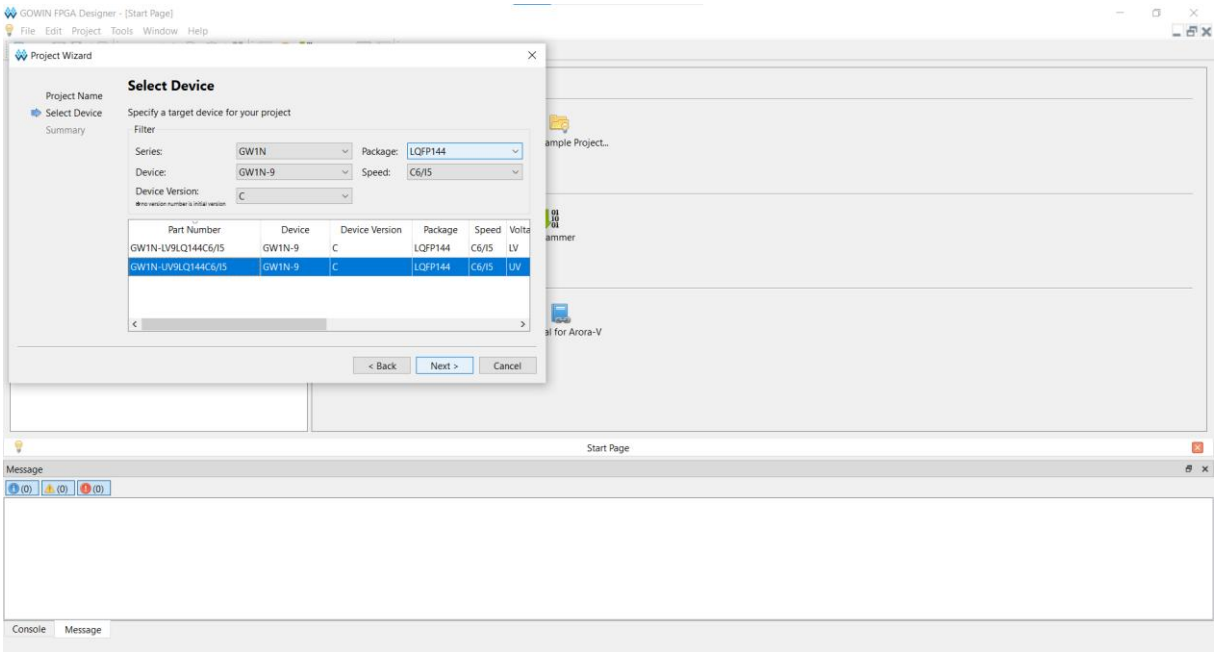
2) Select « FPGA Design Project ». Click **OK**.



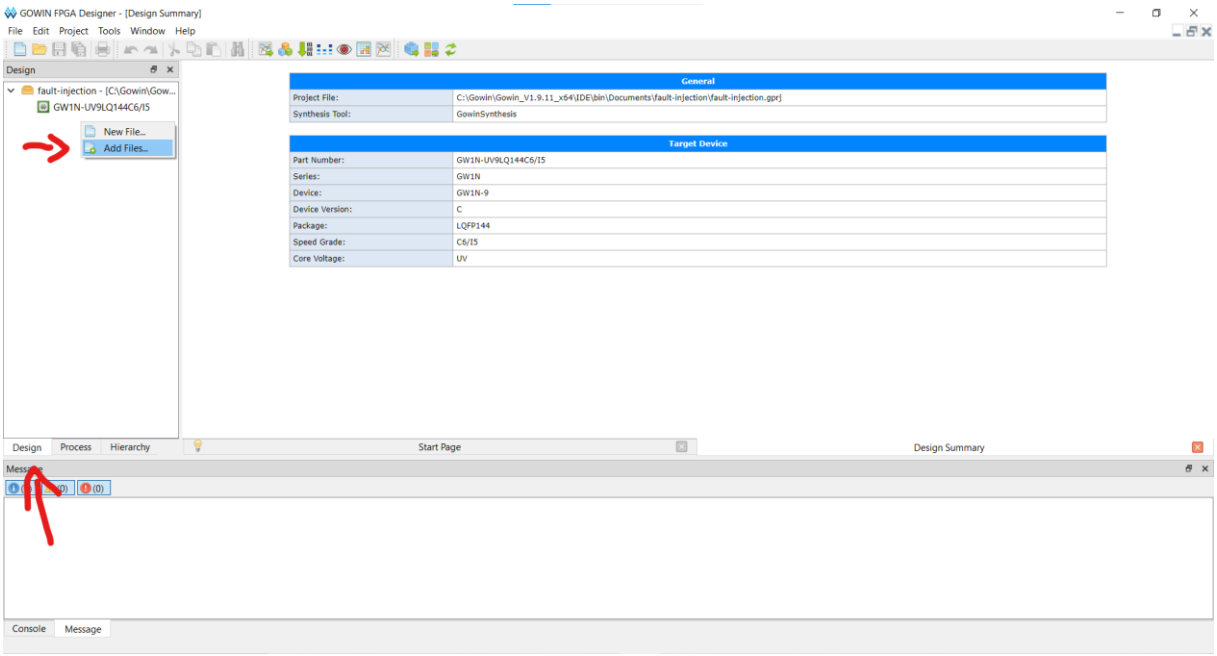
3) Type the name of the project and its location (here, “fault-injection”). Click **Next**.



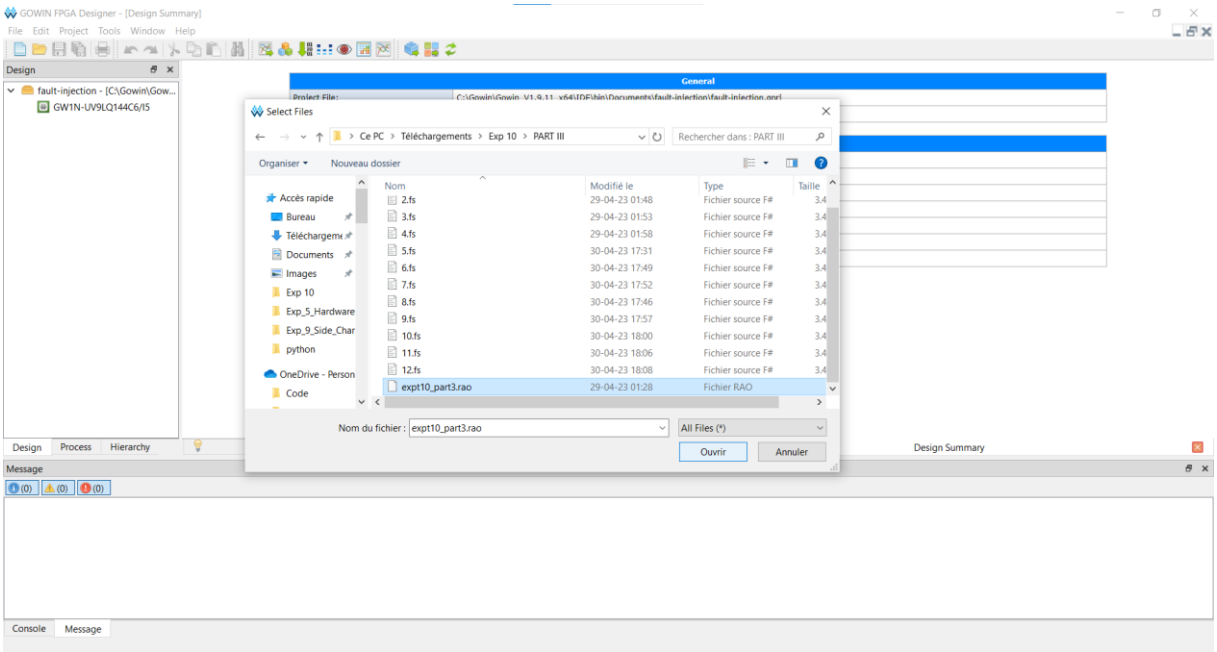
4) Select the following device: **GW1N-UV9LQ144C6/I5**. Click **Next**, then **Finish**.



5) In the **Design tab**, right-click and select "**Add Files…**"

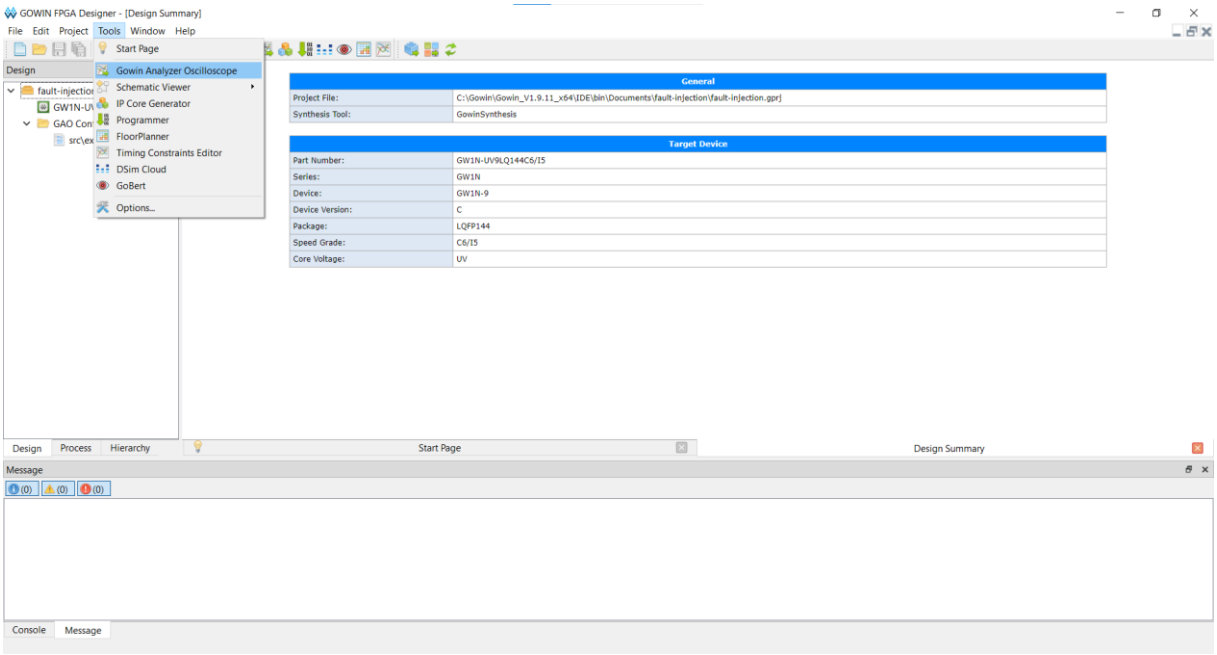


6) Select and upload the provide **.rao** file. Click **Yes** when prompted to copy the file.

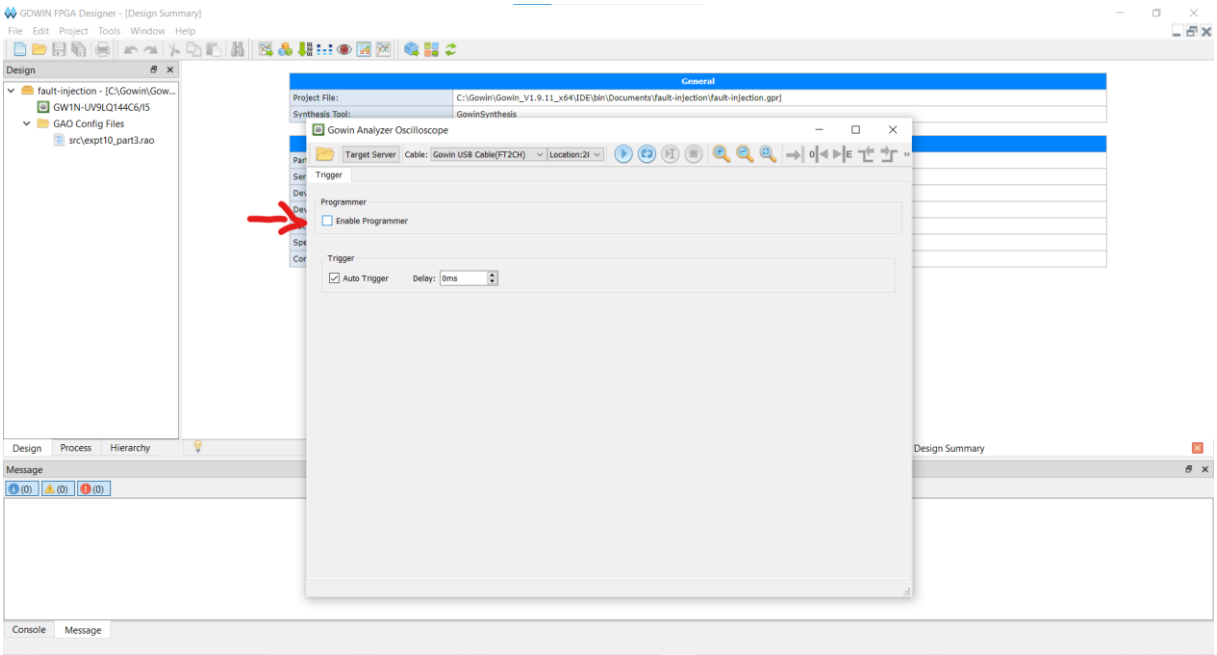


3.3.1: Program the FPGA

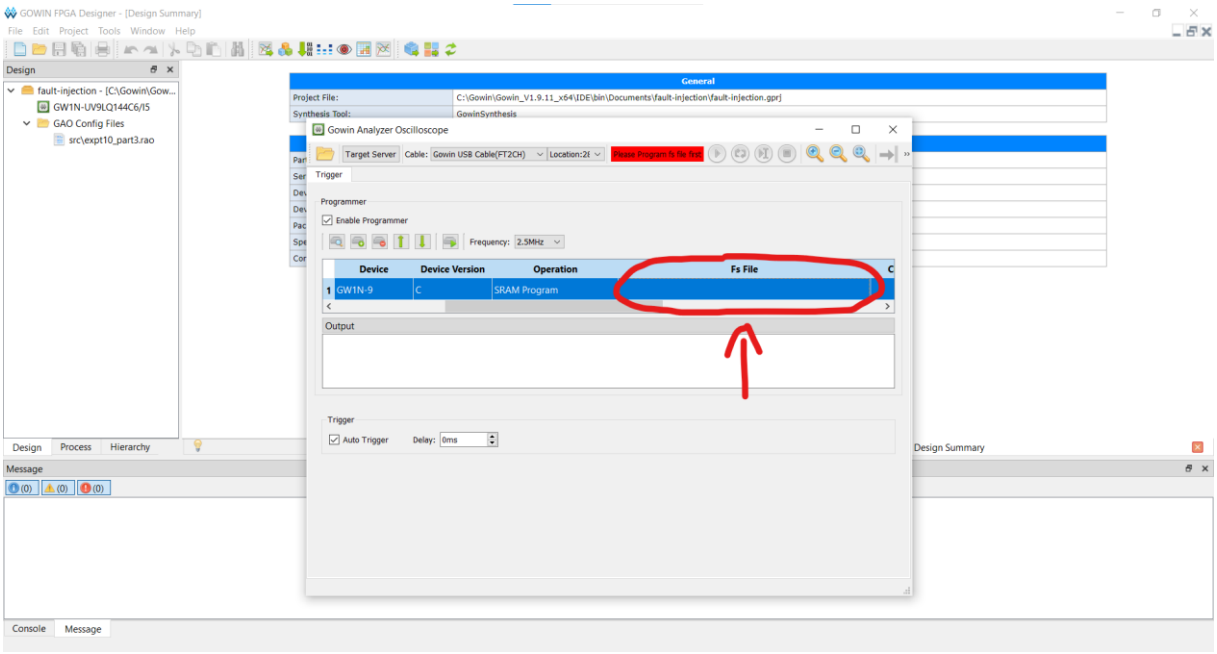
1) Go to Tools > **Gowin Analyzer Oscilloscope**.

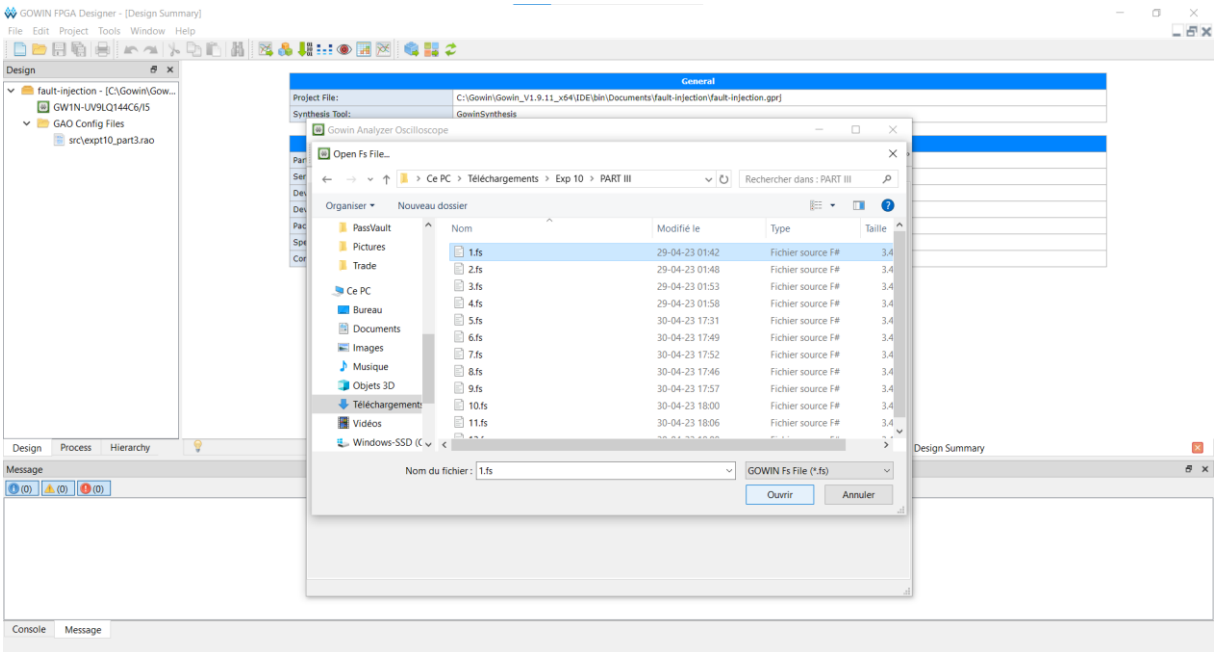


2) Click on **Enable Programmer**.

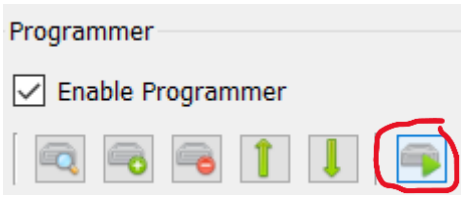


3) Double click in the “**Fs file**” column area and select the provided .fs file.

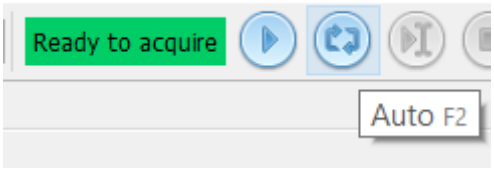




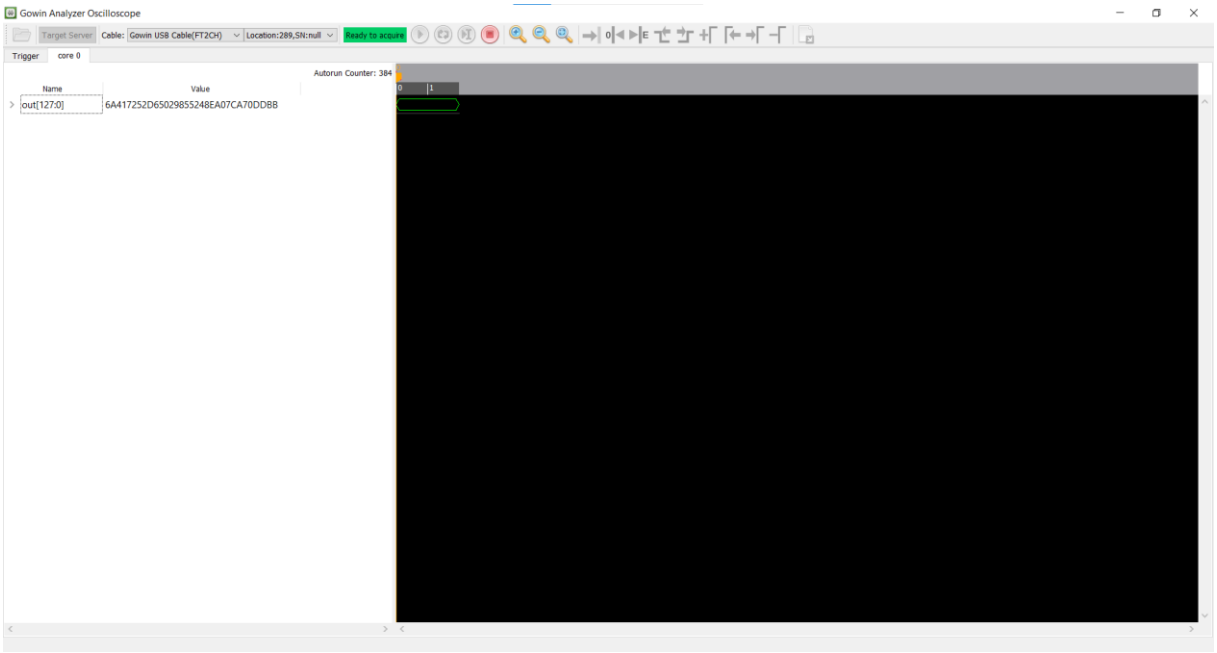
3) Click on the **play** button.



4) Click on the **Auto** button to launch the oscilloscope and start capturing signals.



5) Resize the columns to display the entire cipher text output value. Setup is done.



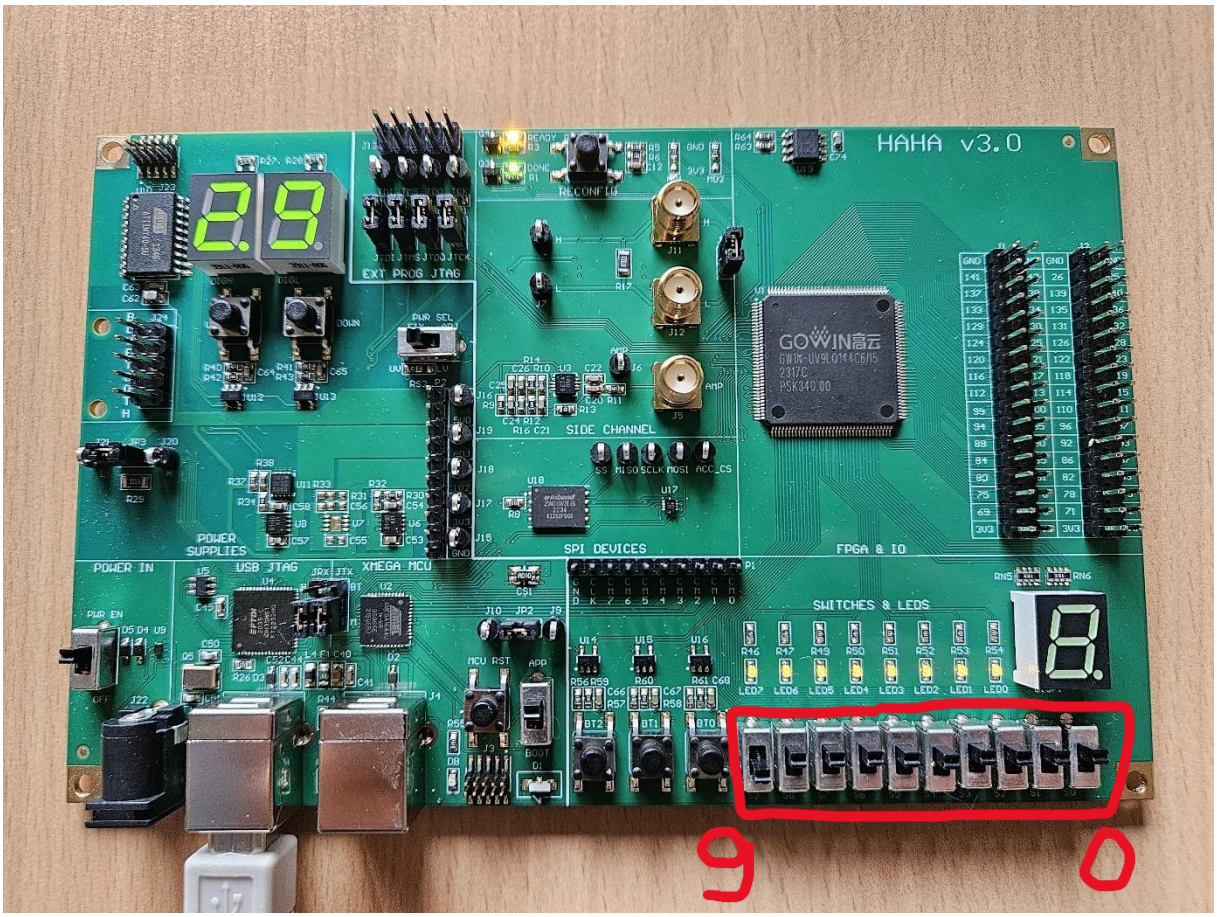
3.4 Cipher text collection

Now that the configuration file is loaded on the board, it runs the AES encryption process in loop, displaying the final cipher text output in the Gowing oscilloscope. The key is unknown and specific to your configuration file. The task is to retrieve it with differential fault analysis.

On the bottom right part of the board are 9 switches (from 9 to 0 from left to right). When they are all OFF (switched down), you get the normal encryption result. However, when the switch n°9 is turned ON, the faulty bit is injected. The position of the faulty bit is determined by switches nb 6 to nb 0. The decimal value from those 7 bits will indicate the position of the faulty bit in the 128 possible ones in M_9. Switches nb 8 and 7 don't do anything in this lab.

Example:

- Switch 9 turned OFF: genuine cipher text
- Switch 9 turned ON and switches 0 to 6 turned off: faulty bit at position 0
- Switch 9 turned ON and switches 0 to 1 turned OFF: faulty bit at position 1
- ...
- Switch 9 turned ON and switches 0 to 6 turned ON: faulty bit at position 128



With switch 9 OFF

Value

6A417252D65029855248EA07CA70DDBB

With switch 9 ON

Value

6A4172B6D65029855248EA07CA70DDBB

In the provided template.txt file, collect the genuine and all faulty cipher texts, starting with the faulty bit from position 0 to 128, one cipher text per line. The provided m9.py will allow you to retrieve the 9th round cipher output from the faulty cipher text file. And the key.py file will allow to retrieve the master key from M9 and the genuine cipher text (those values are to be hard coded at the end of the script).

4. References

[1] Schneider, J., & Smalley, I. (2025, July 22). Field programmable gate array. *Think*. https://www.ibm.com/think/topics/field-programmable-gate-arrays

[2] Yang, S., Paul, S. D., & Bhunia, S. (2021). Hands-On Learning of Hardware and Systems Security. *Advances in Engineering Education*, *9*(2), n2.