Experiment 4
Hardware Trojan Attack I
on HaHa Board v3.0

We describe an experiment on hardware Trojan attacks, in the form of malicious modifications of electronic hardware, that pose major security concerns in the electronics industry.

Instructor: Dr. Swarup Bhunia
Co-Instructors/TAs: Reiner Dizon-Paradis and Shuo Yang

Case Study

We describe an experiment on hardware Trojan attacks, in the form of malicious modifications of electronic hardware, that pose major security concerns in the electronics industry.

In this chapter, we describe an experiment on hardware Trojan attacks and countermeasures. Emerging trend of outsourcing the design and fabrication services to external facilities as well as increasing reliance on third-party Intellectual Property (IP) cores and electronic design automation (EDA) tools makes integrated circuits (ICs) increasingly vulnerable to hardware Trojan attacks at different stages of its life-cycle. The modern IC design, fabrication, test and deployment stages highlight the level of trust at each stage. This scenario raises a new set of challenges for trust validation against malicious design modification at various stages of an IC life-cycle, where untrusted components/personnel are involved. In particular, it emphasizes the requirement of reliable detection of malicious design modification made in an untrusted fabrication facility, during the post-manufacturing test. It also imposes a requirement for trust validation in IP cores obtained from untrusted third-party vendors.

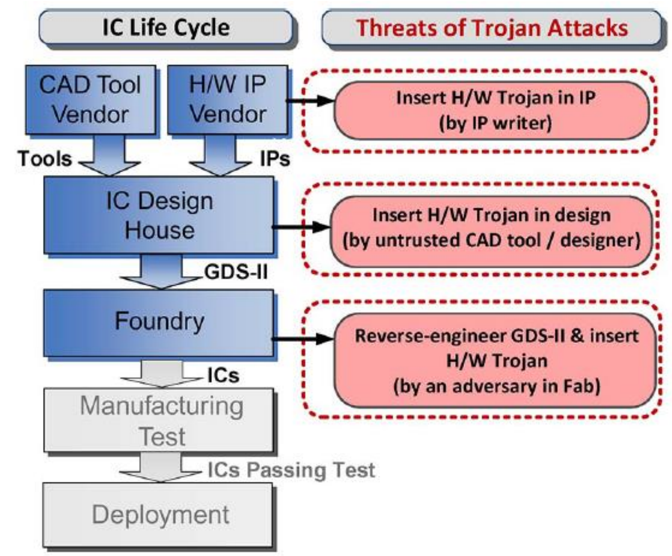


Figure 1 Hardware Trojan attacks by different parties at different stages of IC cycle.

Figure 1 illustrates different steps of a typical IC life cycle and the possibility of Trojan attacks in these steps. Each party associated with the design and fabrication of an IC can be a potential adversary who can tamper it. Such tampering can be accomplished through add/delete/alteration of circuit structure or through modification of manufacturing process steps that cause reliability issues in ICs. From an attacker's perspective, the objective of such attacks can be manifold, e.g., to malign the image of a company to gain competitive edge in the market; disrupt major national infrastructure by causing malfunction in electronics used in mission-critical systems; or leak secret information from inside a chip to illegally access a secure system.

Theory Background

Recently Intel announced a flaw in the implementation of the "TSX" instruction for its Haswell series of Central Processing Unit (CPU). This announcement came almost a year into the product's lifecycle and almost three years since the beginnings of Haswell's architecture was laid out. This is a legitimate mistake on Intel's part – there is no foul play or trickery here.

However, researchers at the University of Massachusetts were able to modify an Intel Ivy Bridge processor – the series that Haswell replaced – and significantly impair the Random Number Generator (RNG) of the processor. They did this by modifying the silicon that made up actual transistor. Their modification is completely undetectable without a Scanning Electron Microscope (SEM) and a known good chip to authenticate against. If the security of the RNG is compromised then everything generated from it is also compromised, for example, private encryption keys.

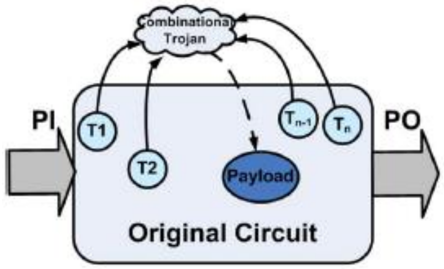


Figure 2 Combinational Trojan model

An intelligent adversary is expected to hide such tampering with an IC's behavior in a way that makes it extremely difficult to detect with conventional post-manufacturing testing. Intuitively, it means that the adversary would ensure that such tampering is manifested or triggered under very rare conditions at the internal nodes, which are unlikely to arise during testing but can occur during long hours of field operation.

Figure 2 and Figure 3 show general models of combinational and sequential Trojans, respectively. These abstract models of Trojans are useful for studying the space of possible Trojans, and, similar to fault models, help in test vector generation for Trojan detection.

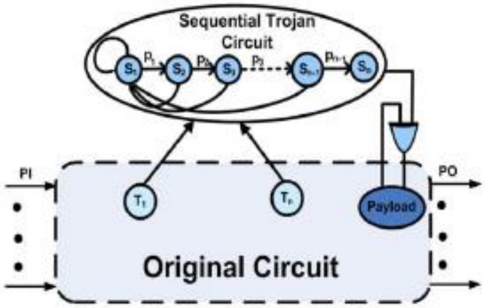


Figure 3 Sequential Trojan Model

💻Experiment Set-up: Configuration

1. The instruments needed for this experiment are the HaHa Board v3.0, a USB A to B cable, and a computer.
2. The software needed is GOWIN FPGA Designer, version 1.98 or higher.
3. Refer to the HaHa User Manual to see the steps of configuring the GOWIN GW1N-9 FPGA.

Instructions and Questions

In this experiment, you will need to implement a DES (Data Encryption Standard) into the GOWIN GW1N-9 FPGA, and then hack it by inserting two kinds of hardware Trojan into it.

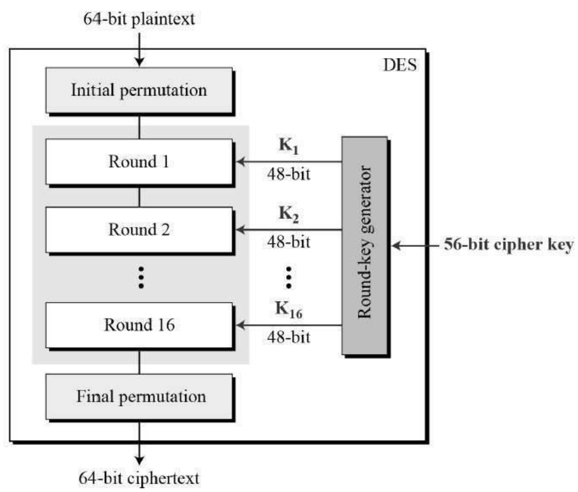


Figure 4 DES encryption illustration.

Part I: Implement a DES

The DES is a symmetric-key algorithm for encrypting electronic data. Although it is now considered insecure, it was highly influential in the advancement of modern cryptography. It uses 16 round Feistel structure. The block size is 64-bit, of which DES has an effective key length of 56 bits since 8 of the 64 bits of the key are not used by the encryption algorithm. The encryption steps are illustrated in Figure 4.

You have been provided several 12 Verilog files related to the DES implementation on the FPGA. If you prefer VHDL, you can use any open-source module online. Create a subdirectory in your lab3 folder, called "part1". Within this, create a new GOWIN project called `part1` with top module called `top`, that instantiates `des` module from `des.v` file. Refer to Figure 5. There are two modules inside this file: `des` and `des_o`. `des_o` contains the DES implementation, and `des` is the wrapper top module that instantiates `des_o`. Plaintext and key are given in des. There is a RAM module called ram1 instantiated, which is a 64-bit wide, 32-word deep RAM. It will store all the encryption results of the 16 rounds. Refer to the appendix for starting this project.

You should use the Gowin Analyzer Oscilloscope to view the DES encryption. The encryption process happens repeatedly, so you can use GAO to view the signals repeatedly perform the encryption; however, after the first encryption cycle (which happens very fast, before you can start capture with GAO), the result in the RAM should not change.

If you look at des.v, you will see the clock is being slowed down by 50x, resulting in a 1MHz clock. You may want to clock the logic analyzer using this slower clock signal so that you are not viewing the same data repeatedly. You can shorten the capture window down to 32, which will be large enough to capture all rounds of encryption at least once.

See the following pages for details on the questions you need to answer for Part 1.

Part II: Insert a combinational Trojan

Create a new folder, **part2**, that contains a new Gowin project. Start with a working DES circuit, identical to what you did in Part 1 (You can copy all of the files into the *src* directory for this new project).

Insert a combinational Trojan into the DES circuit. The trigger condition of the Trojan is when the output of the F function (Figure 6) satisfies for some value of the least significant 4 bits (see next section). When the Trojan is triggered, the LSB of the input key (NOT round keys) for the DES is inverted (ie. invert key56[0]). When the trigger condition is not true, the key becomes the original input key.

You will want to register your trigger signal to prevent a combinational loop from occurring. That means that in any cycle where the trigger is true, the following cycle will have key56[0] inverted.

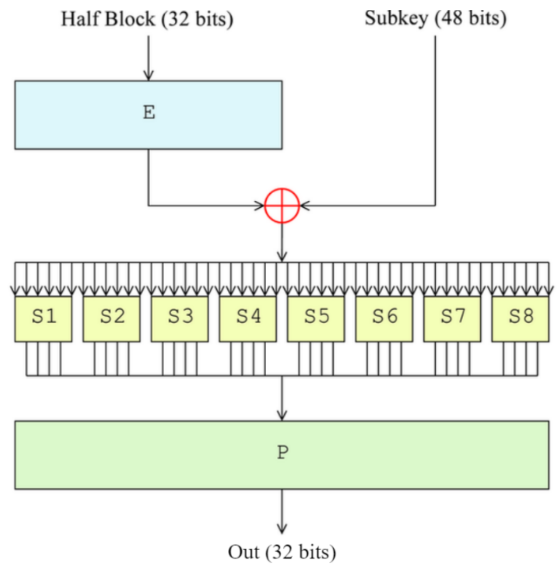


Figure 5 Feistel function (F function) of DES

Part III Insert a sequential Trojan

Create a new folder, **part3**, that contains another new Gowin project. Once again, copy over your working DES design from Part 1.

Insert a sequential Trojan into the DES circuit. Use the same clock as the DES circuit. The trigger condition of the Trojan is when the least significant 2 bits of the F function output in order go through some order of three values at the negative edge of the clock (see next section).

Like Part II, when the Trojan is triggered, the LSB of the input key (NOT round keys) for the DES is inverted (ie. invert key56[0]). The key only needs to be inverted for one cycle, and can then revert to the original value.

(The lab previously suggested inverting the key permanently; however, it then becomes difficult to observe the trigger activating in the initial encryption pass. You are welcome to implement it the 'old' way if you prefer, although it is harder to properly observe the trigger).

Measurement, Calculation, and Question

Answer the following questions.

Part I: Implement a DES

1) Use the plaintext and key provided and add them in the des.v Verilog file:

    `desIn` = 64'hA42F891BD376CE05
    `key64` = 64'h0123456789ABCDEF

    Store all the encryption results for the 16 rounds in an implemented RAM and show the result of the final round (upper 64 bits of the 1024-bit RAM). You can verify the correct encryption using an online tool, such as http://des.online-domain-tools.com/
2) Which module is creating the key for each round? How?
3) Which module is doing the Feistel function?
4) How many Logic Elements are used?

Part II: Insert a combinational Trojan

1) When the trigger condition is 4’b0110, will the Trojan be triggered? How many times is it triggered in the 16 rounds? Turn in a screenshot of the GAO window.
2) When the condition is 4’b1001, repeat answering question 1) again. Take another screenshot of the GAO window.
3) When the condition is 4’b1010, repeat answering question 1) again. Take another screenshot of the GAO window.

Part III: Insert a sequential Trojan

1) Consider the sequential trigger 2’b01→2’b10→2’b11:
    a. How many states are needed in total? How many additional registers have you implemented?
    b. When the trigger condition is 2’b01→2’b10→2’b11, will the Trojan be triggered? How many times is it triggered in the 16 rounds? Turn in a screenshot of the GAO window.
2) When the condition is 2’b11→2’b10→2’b00, repeat answering question 1) again.
3) When the condition is 2’b11→2’b01→2’b00, repeat answering question 1) again.

References and Further Reading

[1] http://securityaffairs.co/wordpress/17875/hacking/undetectable-hardware-trojan-reality.html

[2] Bhunia, Swarup, et al. "Hardware Trojan attacks: threat analysis and countermeasures." Proceedings of the IEEE 102.8 (2014): 1229-1247.

[3] http://www.emvlab.org/descalc/

[4] https://www.pantechsolutions.net/matlab-code-for-des-algorithm

[5] http://www.tutorialspoint.com/cryptography/data_encryption_standard.htm

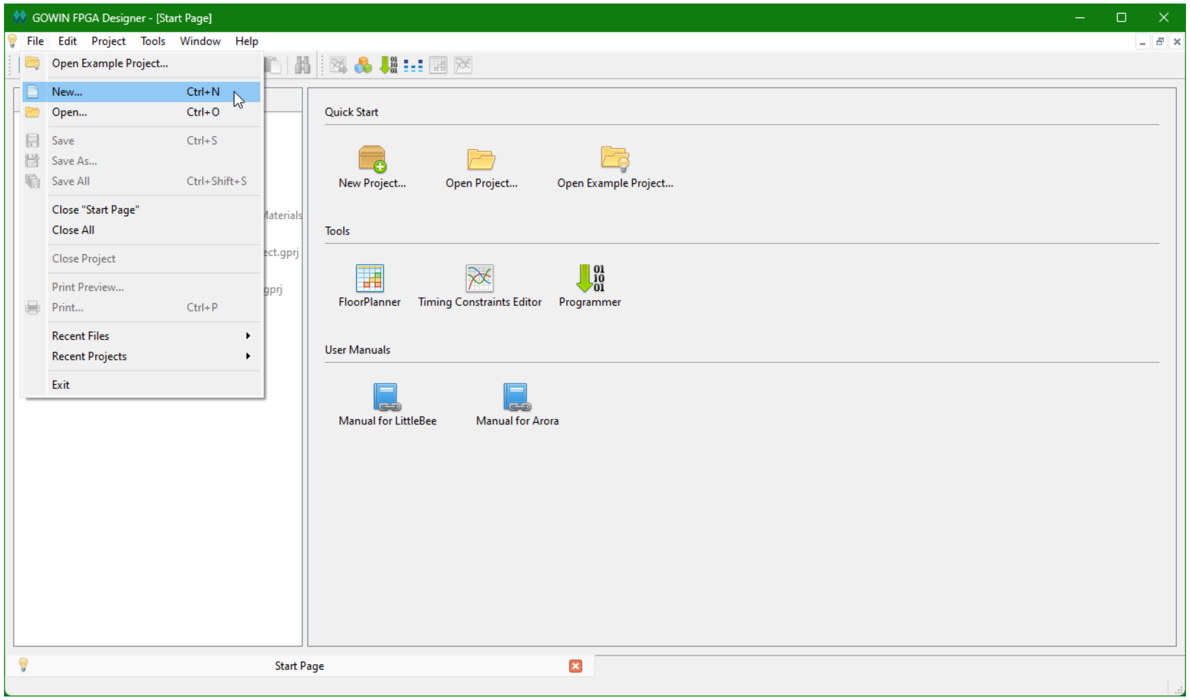
[6] https://en.wikipedia.org/wiki/Data_Encryption_Standard

[7] https://en.wikipedia.org/wiki/Feistel_cipher

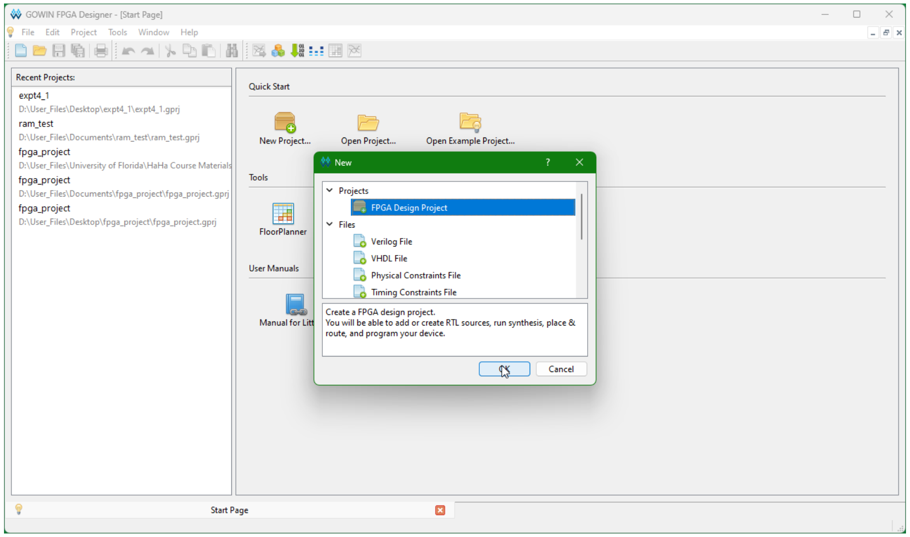
Appendix A: Starting a GOWIN FPGA Designer Project

The following are the steps to start a project on the GOWIN FPGA Designer software for part I of this experiment:

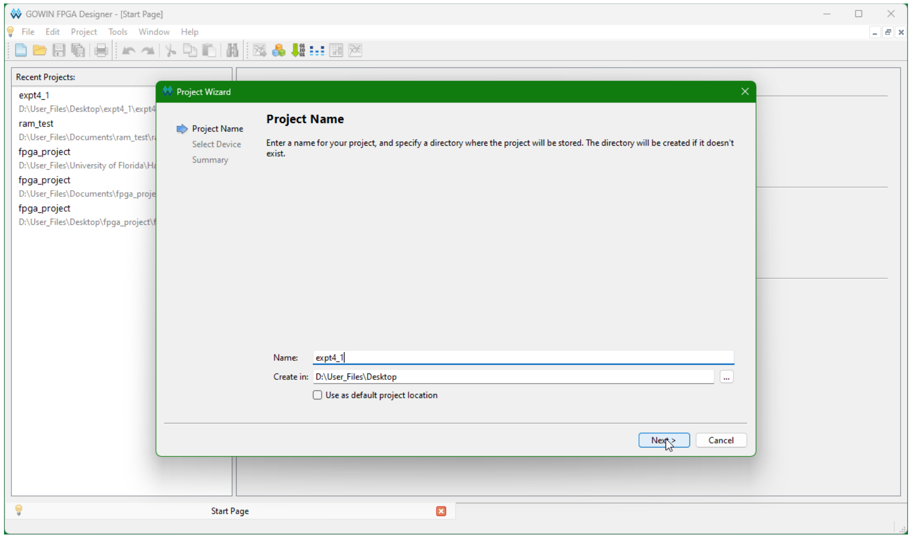
1. Go to **File** > **New**.



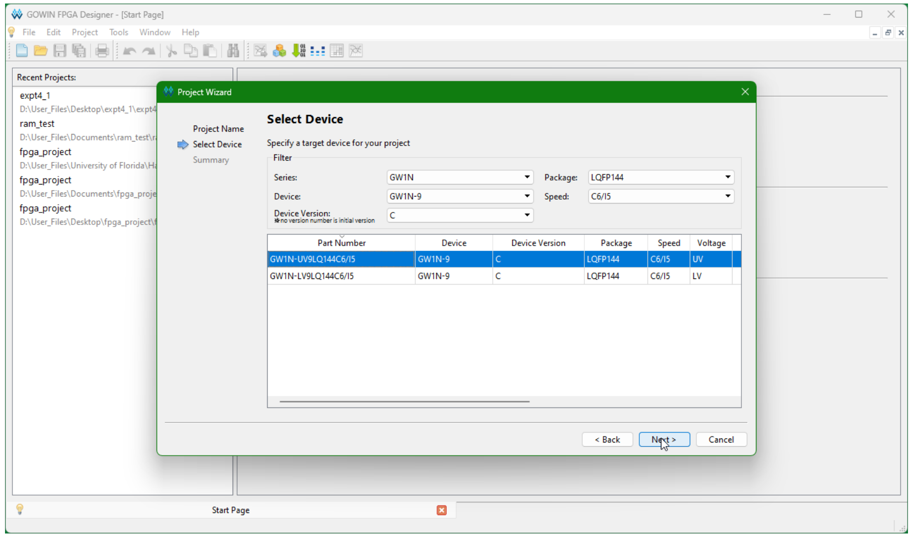
2. Select **FPGA Design Project**. Click **OK**.



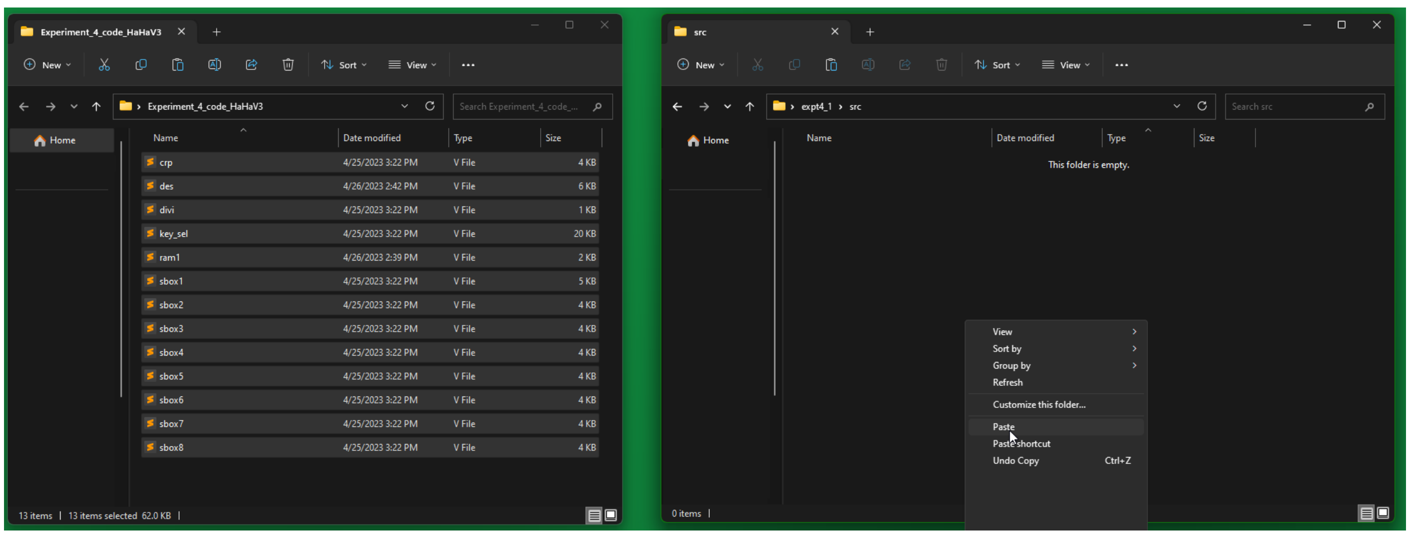
3. Type the name of the project and its location. Click **Next**.



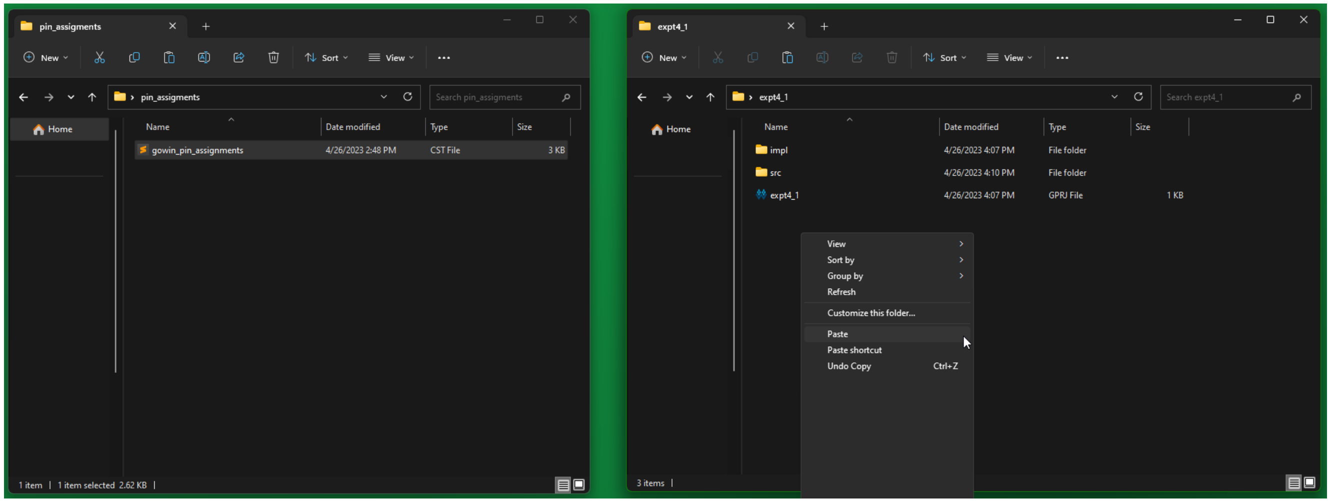
4. Select the device according to the screenshot below. Click **Next** and then **Finish**.



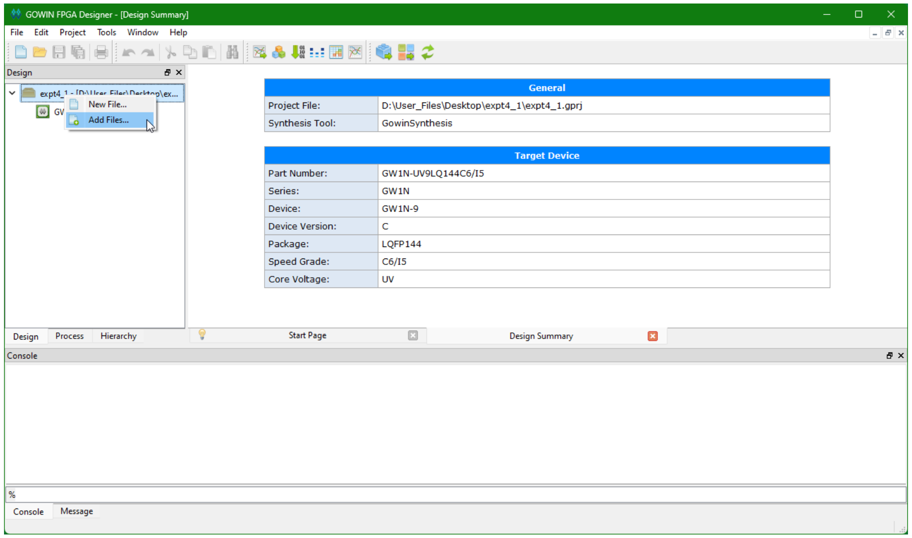
5. Copy the experiment codes from your **lab3** to **lab3/*part1/src*** directory.



6. Copy GOWIN pin assignment file (**gowin_pin_assignments.cst**) to the ***src*** directory.

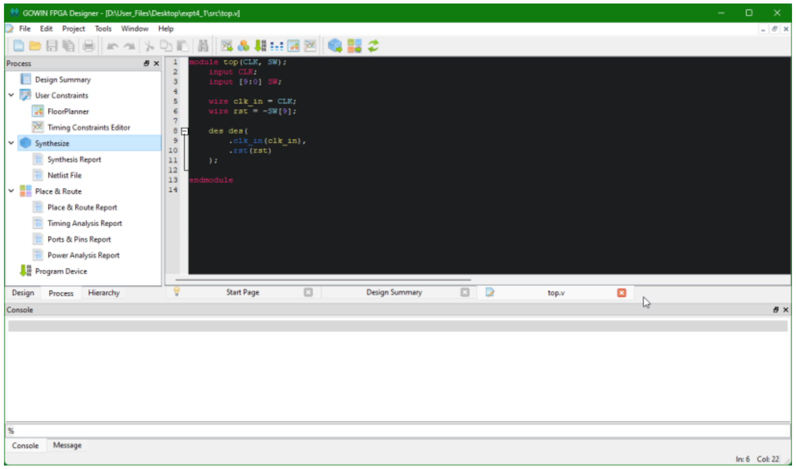


7. In the GOWIN FPGA Designer window, add existing files by right clicking on the left sidebar. Click **Add Files...** button. Select all the files inside ***src*** directory (including all .v files and the .cst file). Click **Open**.

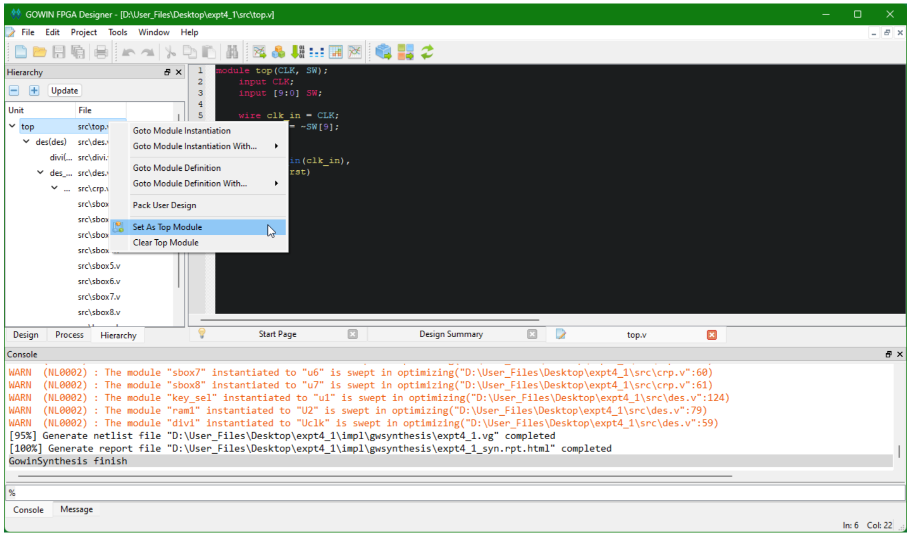


Experiment 4

8. **IT IS HIGHLY RECOMMENDED TO COPY THE *sample_top.v* INSIDE THE *hw_resources* TO THE MODULE BELOW**. This file ensures the pin assignments file map to the correct port names in this file. The module below is a slightly modified version of the *sample_top.v* file. DO NOT USE THE EXACT VERILOG CODE BELOW. You need to instantiate the DES module similar to the code below.



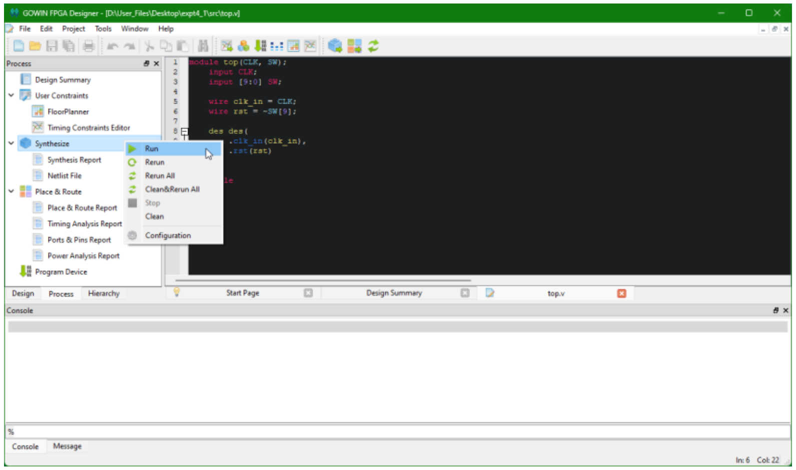
9. Change the left sidebar to **Hierarchy** tab. Right click on **top** and select **Set As Top Module**.



Before synthesizing your design, go to Project->Configuration->Dual-Purpose Pin. Check "Use SSPI as regular IO"

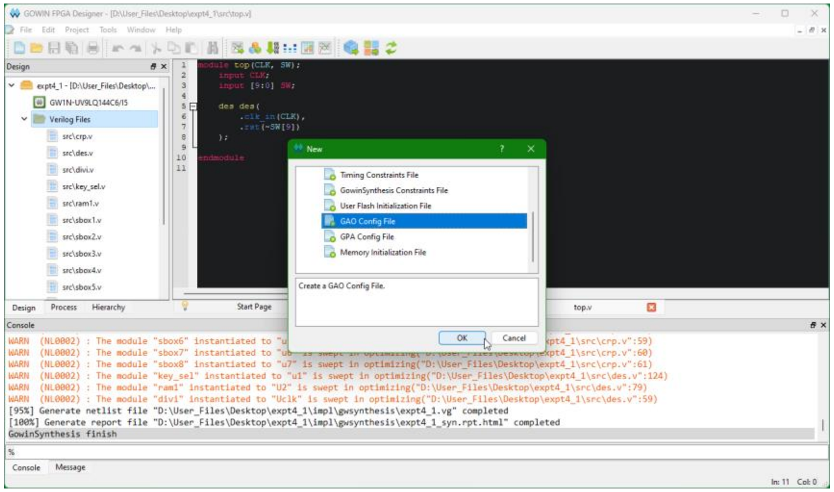
Appendix B: Synthesizing a circuit

1. Run the synthesis tool by changing the left sidebar to **Process**. Right click on **Synthesize** and Click **Run**.

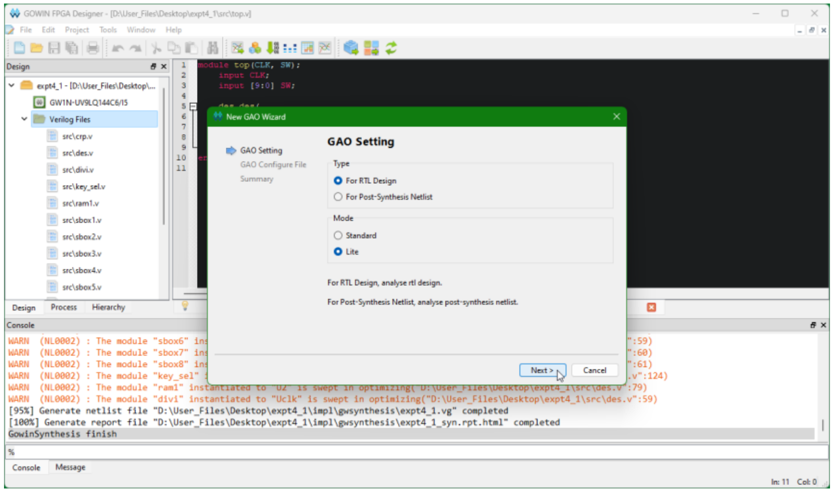


Appendix C: Setup GOWIN Analyzer Oscilloscope file (*Optional*)

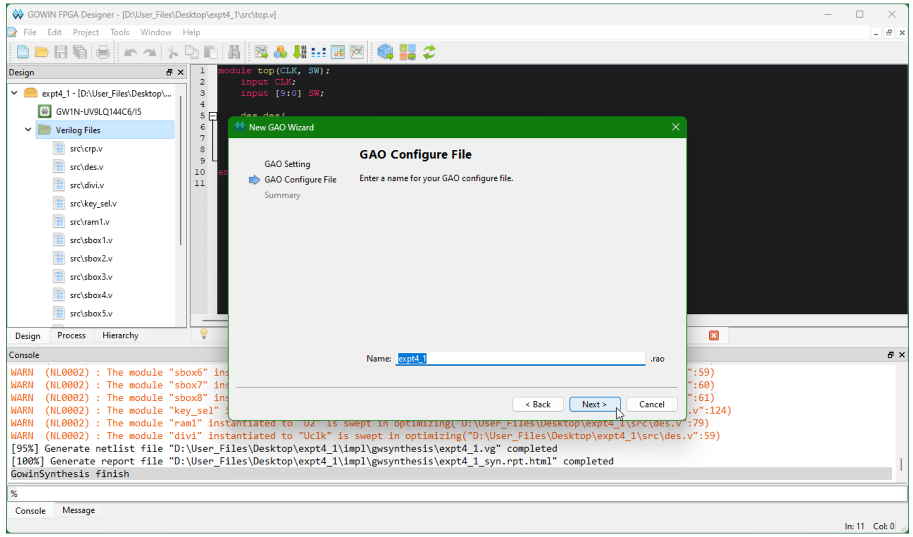
1. Create a new file with type **GAO Config File**. Click **OK**.



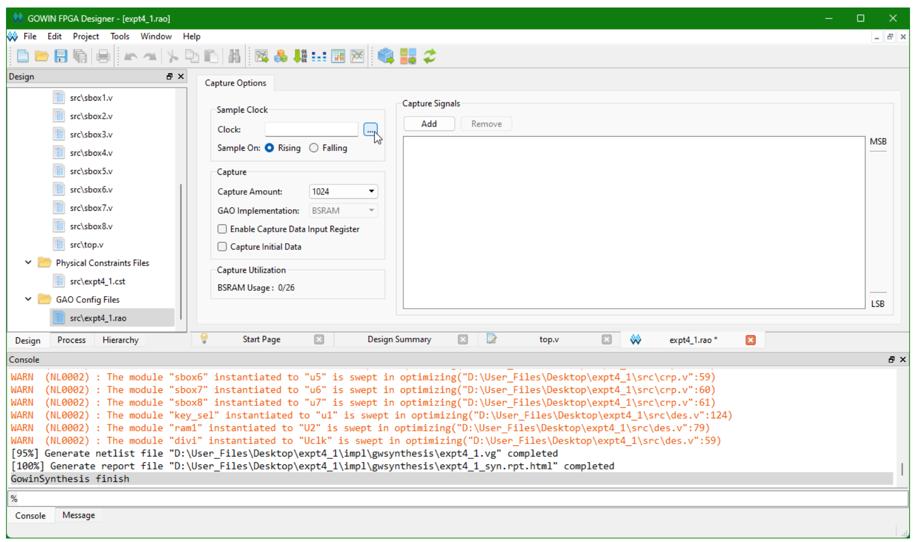
2. Change the mode to **Lite**. Click **Next**.



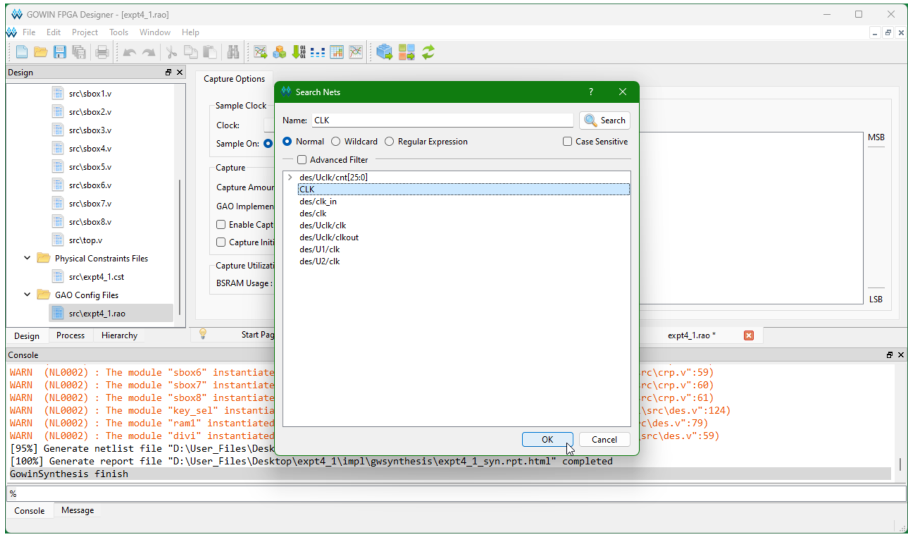
3. Enter a name or leave it as it is. Click **Next** and then **Finish**.



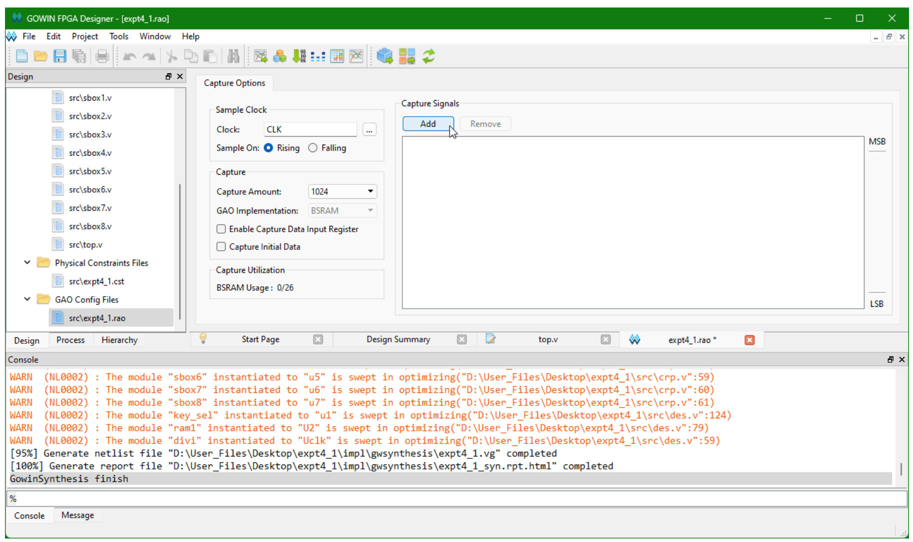
4. In the Design tab, open the **src\part1.rao** file. Click on **…** button next to **Clock**.



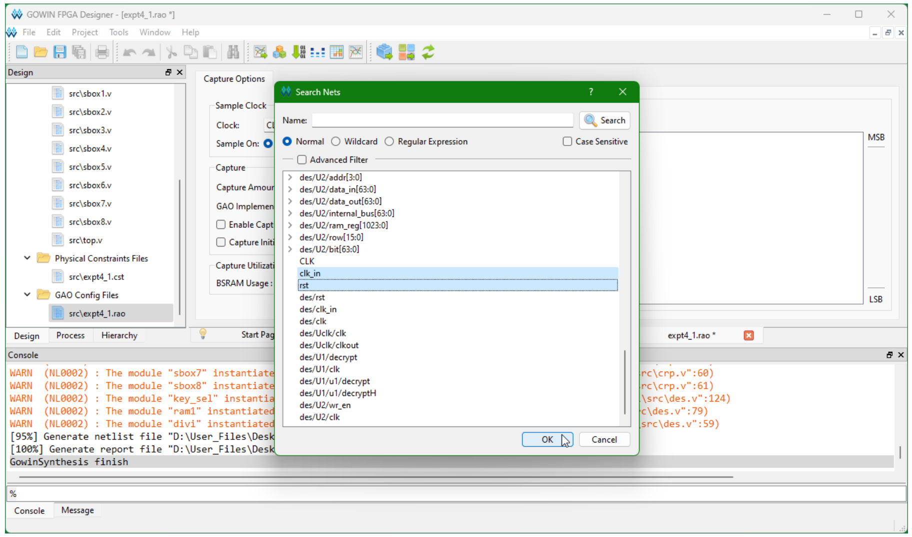
5. Search for **CLK** or *whatever clock signal you used*. Click **OK**.



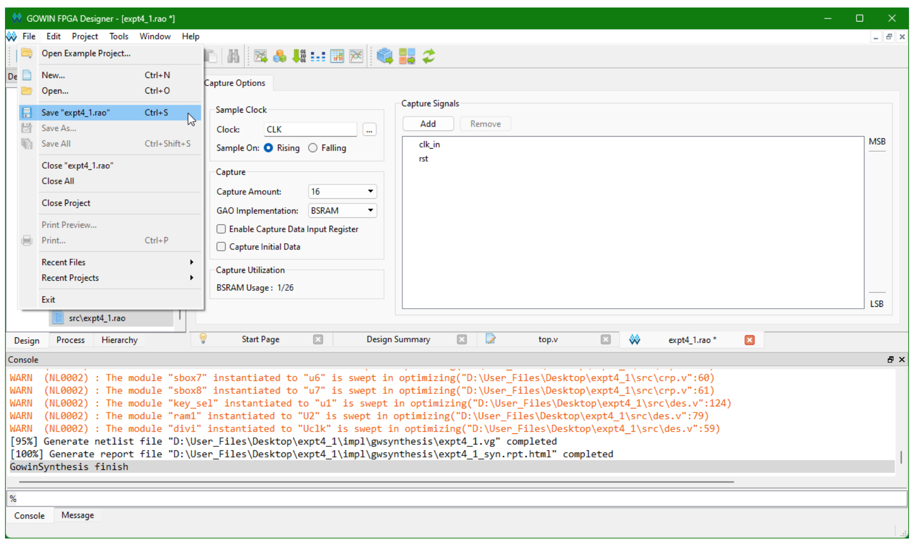
6. Click on **Add** below Capture Signals.



7. Search for any signal you want to monitor (you must choose at least one). Click **OK**.

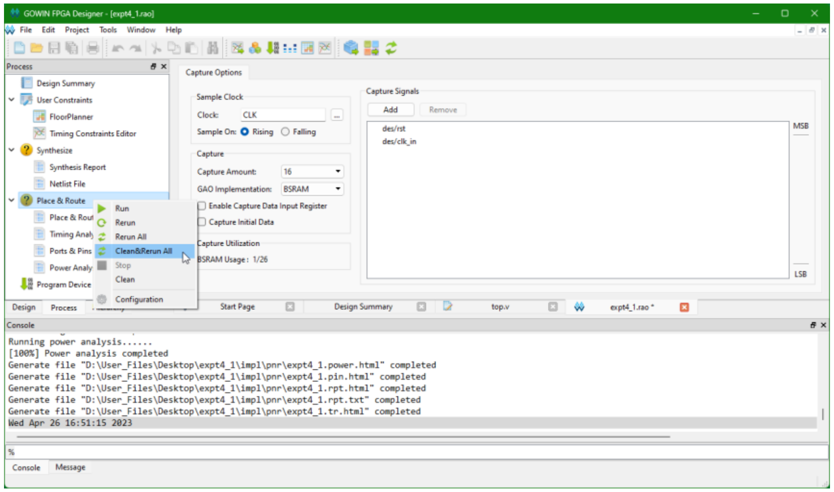


8. Save the **part1.rao** file. If you want to run the GAO tool, run Step 1 of Appendix D. Then, go right into Appendix E afterwards.

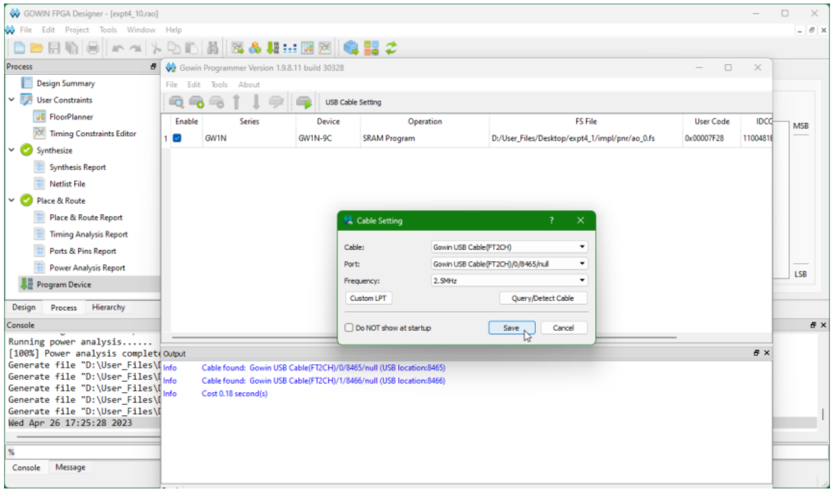


Appendix D: Running Place/Route tool & Programming the FPGA

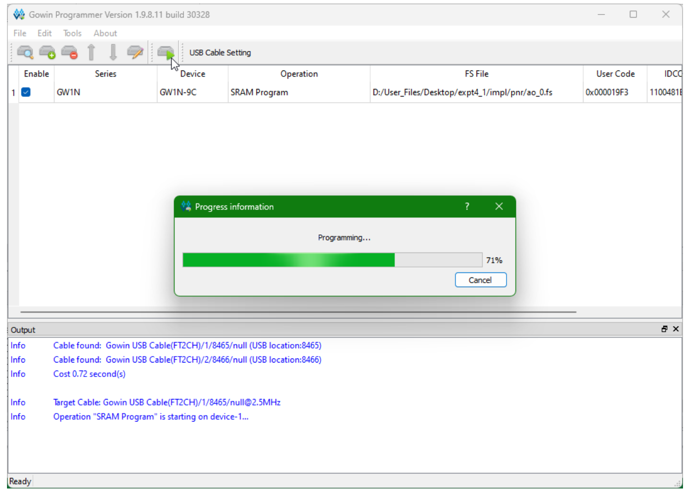
1. Go to Process tab on the left sidebar. Right click on **Place & Route** and click **Clean&Rerun All**.



2. Double click on **Program Device** on the Process tab. Make sure the Port is set to "*Gowin USB Cable(FT2CH)/0/…*" If this is not the case, unplug all USBs and plug in the HaHa FPGA again. Then, click **Save** on the new window.

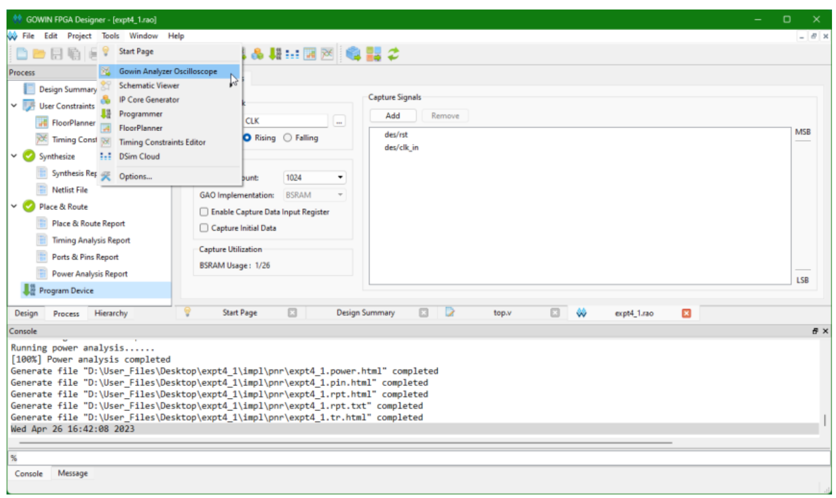


3. Click on the play button. The bitstream will then be programmed to the FPGA.

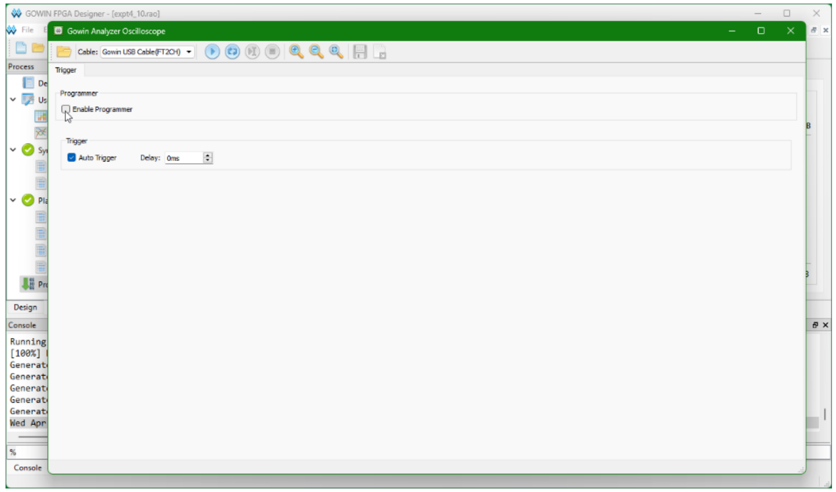


Appendix E: Running the GOWIN Analyzer Oscilloscope tool

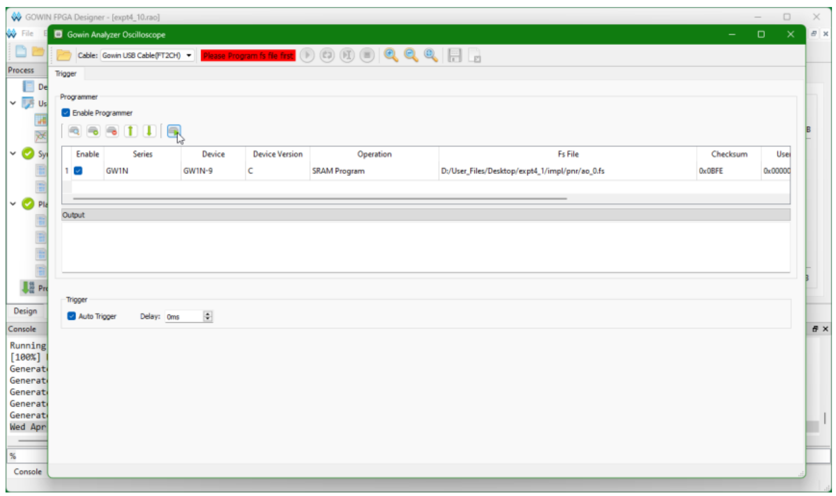
1. Go to **Tools** > **Gowin Analyzer Oscilloscope**.



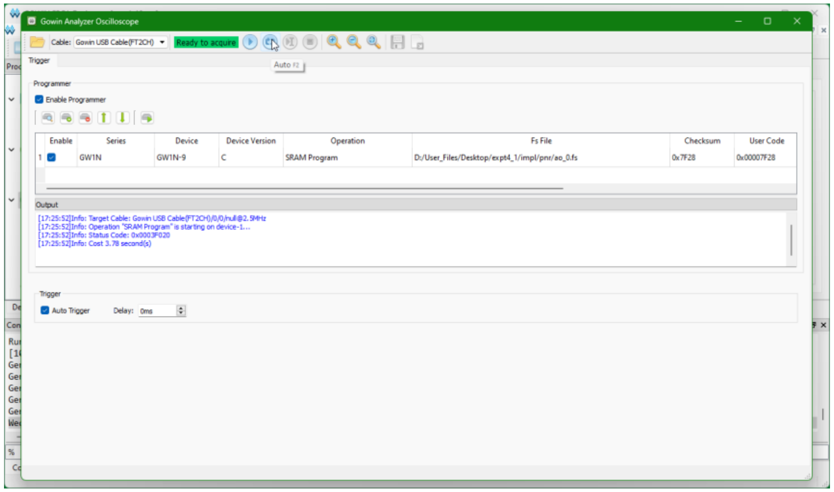
2. Click on **Enable Programmer**.



3. Click on the play button.



4. Click on the **Auto** button to start capturing signals.



5. You will now see the captured signals that you set up in the RAO file.