Experiment 17
Scan Chain Protection
on HaHa v3.0 Board

We describe an experiment to introduce the Design for Test methodology called scan-chain, its advantages, disadvantages, and a method to protect the scan-chain.

Instructor: Dr. Swarup Bhunia
Co-Instructors/TAs: Pravin Gaikwad and Reiner Dizon-Paradis

Theory Background

The modern day SoC has become much more complicated with reduction in transistor size. The high complexity increases questions about the reliability of the SoC. Scan-chain based testing is one of the powerful and popular techniques that can be used to verify the functionality of the chip and it can be designed to cover 100% of faults in the complex design. The scan-chain enables access to the internal signal of the design which are hard to control and observe from primary input and primary outputs.

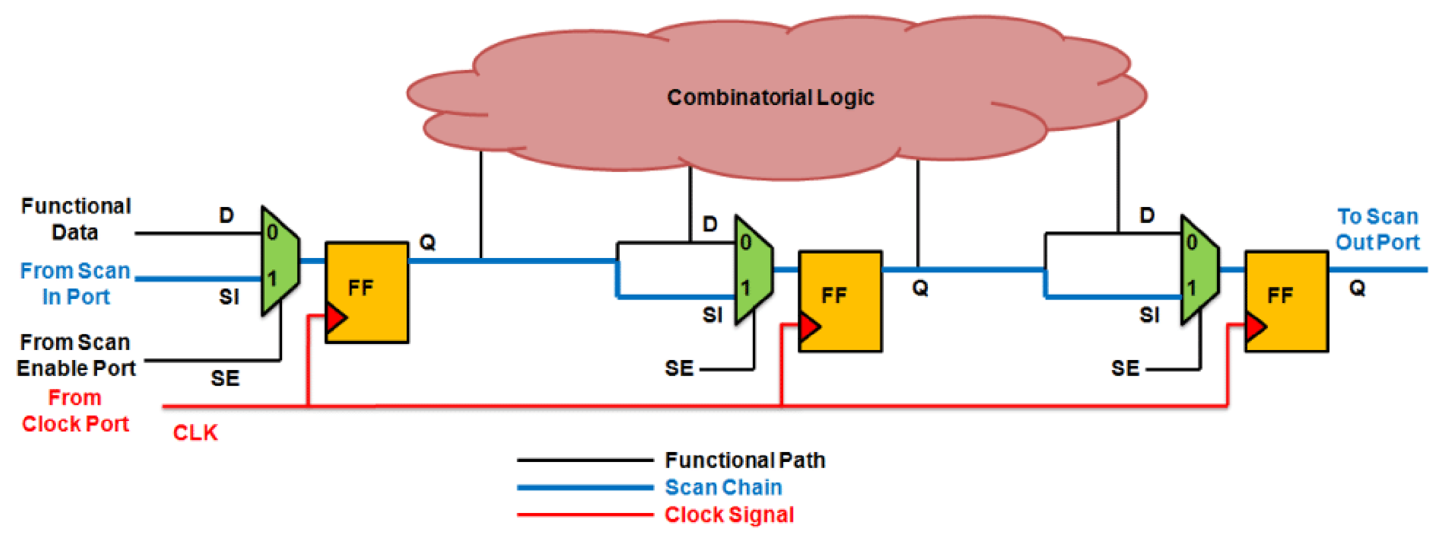

Figure 1 – A typical sequential design.

Figure 1 shows a typical sequential design with registers/Flop-flops connected to combinational elements to achieve some desired functionality based on the specifications. Here, it becomes hard to access the certain internal state of the flip-flops or to access the certain internal state of the certain wires only from primary inputs. To overcome this problem, researchers have proposed a testing methodology called scan-chain to make the entire design easily accessible. In typical scan-chain architectures, a set of scan flip-flops are used which are nothing but a normal flip-flop with muxed input. These scan flip-flops are connected to form a chain called scan-chain. The access to the chain is granted by a scan enable, scan input and scan output signals. These sets of signals are typically added to the primary inputs and primary outputs of the design. This

architecture helps in easily accessing the internal states of the design as the contents of the flip-flop can be changed directly from the primary inputs and the contents of the internal wires / flip-flops can easily be observed with the help of the added functionality in front of the flip-flop. This functionality is disabled in-field. If the adversary gets access to the scan-chain in-field, then sensitive information can get leaked. Hence, a couple of efforts are taken in the research community to protect the scan chain as presented in cite VIm-scan, etc.

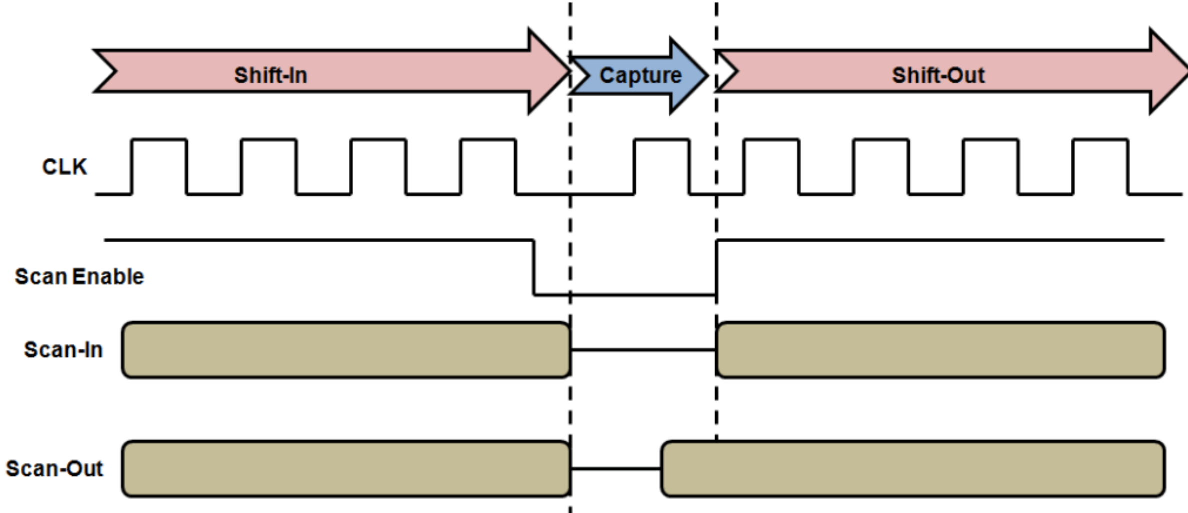

Figure 2 – Scan chain operation

Figure 2 shows timing diagram when scan-chain operations are performed. As shown in the figure, with scan enable user can shift-in or load the data in the flip-flops based on the order of the flip-flop. Once the data is loaded in the flip-flop, a normal operation is performed as in the capture cycle and finally, internal signal values are captured and shifted-out through scan-chain from scan output port. Notice that while unloading the data new data will get shifted in simultaneously.

💻Experiment Set-up: Configuration

The hardware and software needed for this experiment include:

1. The HaHa v3.0 Board.
2. ECE Linux Server
    a) Design Compiler
    b) iVerilog

Instructions and Questions

Please start early! This experiment has four required parts. The first two parts are required for the last two parts. While Parts I and II are relatively easy, there are lots of setup that is required to get working correctly. Please make sure to do the first two parts thoroughly before proceeding to the last parts. The last two parts require knowledge from the Hardware Obfuscation experiment. Feel free to refer to your code, designs, etc. from that experiment to help you with Parts III and IV.

PART I: Simulate Sequential Design

1. Download the provided file **01_simulate_design.zip**
2. Unzip this into the ECE Linux Server in **01_simulate_design/**. Inside this created directory, create a subdirectory called **tb**
3. Create a testbench file called **tb.v** (inside the **tb** directory) that writes to a file the primary outputs and flip-flop outputs given random primary input patterns.
    a. Refer to the Appendix on how to read internal signals inside a testbench module.
    b. Feel free to use the provided **tb.v** as a starting point.
4. Open **s298.v** design. Note of the following inside this Verilog file:
    a. D flip-flops are instantiated as:

```
dfxtp_1 DFF_<x>_Q_reg ( .D(), .CLK(), .Q() );
```

    b. <x> stands for a numerical position in the register.
    c. There should be a couple of these flip-flops in the design. How many flip-flops are in this design?
5. Use random values (e.g., $random, $urandom) to vary the primary input signals and run for a total of 1000 cycles.
6. Run iVerilog to simulate the design. Below are the sample commands:

```
iverilog -s tb -o iv_sim {original design} {testbench file} {library file}
vvp iv_sim > vcd.log
```

7. Find an FSM state that triggers rarely. Report the contents of the flip-flops (FFs) at this state. Record the next three states, including FF contents and primary inputs that trigger that state.

PART II: Add Scan Chain to connect all the flip-flops

1. Download the provided file **02_add_scan_chain.zip**
2. Unzip this into the ECE Linux Server in **add_scan_chain/** and navigate to this directory.

3. Open the file called **add_scan_chain.tcl** and pay attention to the two "set" commands at top of the tcl script. Make the following changes and save the script.
    a. Set *design_name* to *s298*
    b. Set *locked* to *false*
4. In the terminal, run the following commands. The first command only applies to UF ECE students.

```
source /apps/settings

dc_shell -f add_scan_chain.tcl
```

Note: This will create a circuit called **s298_scan_chain.v** and other debugging files.

5. Open **scan_chain_order.txt** file. Note the scan chain ordering based on the instance names.
6. Open **s298_scan_chain.v** design. Observe the changes to the D flip-flops
    a. What is the module name of the scan chain FFs?
        i. This does not refer to the instance names, which should remain the same after the insertion of scan chain FFs.
    b. What are the additional signals for these FFs? Read the background section for additional information.
    c. Confirm the scan chain ordering from **scan_chain_order.txt** file. Are the scan chain signals of the scan chain flip-flops connected correctly? Explain the connections of these flip-flops.
7. Using a similar set-up from Part I, simulate this circuit with scan chain. Note of any new signals added as primary inputs or outputs.
    a. First, load the rare state that you found from Part I via the scan chain flip-flops.
    b. Load the primary inputs that transition the flip-flops to the next three states as you note in Part I
    c. Compare the primary outputs and scan chain FF outputs from Part I
    d. What are the advantages of using scan chain as demonstrated in this part?
    e. What are other advantages of using scan chain on a design?

PART III: Add combinational obfuscation with scan chain and observe scan chain vulnerabilities

Please complete Part I and Part II thoroughly before proceeding to Parts III and IV. The next two parts of the experiment will heavily refer to the previous parts, especially their setups. Parts III and IV require knowledge from Hardware Obfuscation experiments.

1. Insert 16-bit combinational obfuscation on the original s298 design on Part I (no scan chain added). The primary key inputs should be loaded into registers of D flip-flops, which then connects to the XOR/XNOR locking gates.

   a. Use the Part III Key from Table 1 and a selected primary input pattern to be loaded into the design.
   b. XOR gates in Skywater library is:

```
xor2_1 <instance name>( .A(), .B(), .X() );
```

   c. XNOR gates in Skywater library is:

```
xnor2_1 <instance name>( .A(), .B(), .Y() );
```

   d. D flip-flops in Skywater library is:

```
dfxtp_1 <instance name>( .D(), .CLK(), .Q() );
```

   e. Please pay special attention to the input/output signal names.

2. Create a testbench and simulate the design using iVerilog to verify the locked designs
   a. Compare the outputs of unlocked and locked designs.
3. Using the setup from Part II, insert scan chain FFs to this locked design.
   a. Confirm the scan chain ordering from **scan_chain_order.txt** file. Are the scan chain signals of the scan chain flip-flops connected correctly? Explain the connections of these flip-flops.
4. Using a similar set-up from Part I, create a testbench and simulate this locked circuit with scan chain. Note of any new signals added as primary inputs or outputs.
   a. Unlock the design by loading the correct key. Make sure to disable scan chain by setting `test_se` to 0.
   b. After loading the key, enable the scan chain. Read the scan chain output for at least 30 cycles.
   c. What do you observe on the scan chain output?
   d. Explain the vulnerabilities of the scan chain as demonstrated in this part.
   e. What are other disadvantages or possible vulnerabilities of using scan chain on a design?

PART IV: Protect the scan chain using sequential obfuscation

1. Design a separate obfuscation FSM module with five states. Refer to Table 1 for the unlocking key sequence specific to your group.
   a. Add a scan chain output unlock signal.
   b. At the last state of the obfuscation FSM, the scan chain output unlock signal should be ‘1’ otherwise it’s ‘0’.
2. Compile the obfuscation FSM module into Skywater library using **convertToSkywater.tcl** script into Design Compiler.
3. Create a testbench and simulate the obfuscation FSM module using iVerilog to verify the working locking module.
4. Instantiate the obfuscation FSM module onto the s298 design with scan chain from Part II.
   a. Lock the `test_so` signal using AND gate and the scan chain output unlock signal from the obfuscation FSM.
   b. Feel free to copy the compiled obfuscation FSM module underneath the s298 module.

5. Using a similar set-up from Part I, create a testbench and simulate this circuit with locked scan chain. Note of any new signals added as primary inputs or outputs.
    a. Without unlocking the scan chain, load a random set of 10 primary input patterns (please note of these patterns as they will be used again). What do you observe at the scan output?
    b. Report the selected 10 primary input patterns.
    c. Now unlock the scan chain, verify the working design using same 10 primary inputs patterns. Provide the screenshot comparing the locked and unlocked designs.
    d. What are the advantages of locking the scan chain as demonstrated in this part?
    e. What are other advantages of using this kind of scan chain locking on a design?

Table 1: Required unlocking keys for each group

Group No.	PART III Key	PART IV Key
1	16'b0110110100100101	5'b10001
2	16'b0111100101011110	5'b11110
3	16'b1111101110101110	5'b01101
4	16'b0001101011010101	5'b11101
5	16'b1011111010100100	5'b01110
6	16'b1110101100100110	5'b01011
7	16'b0100001011010101	5'b10011
8	16'b1110000010100111	5'b01001
9	16'b0000011000111010	5'b00011
10	16'b1010011000010100	5'b11010
11	16'b0010001010010010	5'b00110
12	16'b0010100001101110	5'b11001
13	16'b1100100001000101	5'b10110
14	16'b1000100010010100	5'b00101
15	16'b1000100110101100	5'b10100

☞ Lab Report Guidelines

Deliverables:

In your report, answer ALL the questions.

1. For Part I:
    a. Include the created testbench file to simulate the original s298 design.
    b. What is the rare FSM state from your simulation? Report the contents of the flip-flops (FFs) at this state.
    c. Record the next three states after the rare FSM state, including FF contents and primary inputs that trigger that state.
2. For Part II:
    a. Verilog file of s298 design with scan chain.
    b. What is the module name of the scan chain FFs?
    c. What are the additional signals for these FFs? Read the background section for additional information.
    d. Are the scan chain signals of the scan chain flip-flops connected correctly? Explain the connections of these flip-flops.
    e. Include the created testbench file to simulate the s298 design with scan chain.
    f. What do you observe between the output from Part I and Part II?
    g. What are the advantages of using scan chain as demonstrated in this part?
    h. What are other advantages of using scan chain on a design?
3. For Part III:
    a. Verilog file of s298 design with 16-bit combinational obfuscation.
    b. Include the created testbench file to simulate the s298 design with 16-bit combinational obfuscation.
    c. Provide the screenshot comparing the locked and unlocked designs.
    d. Verilog file of locked s298 design with scan chain.
    e. Include the created testbench file to simulate the locked s298 design with scan chain.
    f. After loading the key, what do you observe on the scan chain output?
    g. Explain the vulnerabilities of the scan chain as demonstrated in this part.
    h. What are other disadvantages or possible vulnerabilities of using scan chain on a design?
4. For Part IV:
    a. Verilog file of five-state obfuscation FSM, compiled to Skywater library.
    b. Include the created testbench file to simulate the five-state obfuscation FSM.

c. Verilog file of s298 design with locked scan chain.
d. Include the created testbench file to simulate the s298 design with locked scan chain.
e. Without unlocking the scan chain, what do you observe at the scan output?
f. Report the selected 10 primary input patterns.
g. Provide a screenshot comparing the locked and unlocked designs.
h. What are the advantages of locking the scan chain as demonstrated in this part?
i. What are other advantages of using this kind of scan chain locking on a design?

Demonstration:

1. Demonstrate the simulation of the original s298 design using iVerilog as well as finding the rare FSM state and its subsequent states.
2. Demonstrate the insertion and simulation of scan chain onto the s298 design as well as the loading of and continuation from a rare FSM state to its subsequent states.
3. Demonstrate a working 16-bit combinational obfuscation with registers onto the s298 design. Show the vulnerability of inserting scan chain on this locked design.
4. Demonstrate a working s298 design with locked scan chain using obfuscated FSM.

References and Further Reading

[1] https://anysilicon.com/overview-and-dynamics-of-scan-testing/
[2] S. Paul, R. S. Chakraborty, and S. Bhunia, “VIm-Scan: A Low Overhead Scan Design Approach for Protection of Secret Key in Scan-Based Secure Chips”

Appendix

Accessing internal nets for simulation using iVerilog

- Let’s assume there is the following internal net declaration inside some design:

```
module some_design(...);
      ...
      wire net0, net1, net2, net3; // internal nets
      ...
endmodule
```

- Inside the testbench, read the internal signals inside an $fwrite function call as shown below.

```
module tb();
...
some_design dut(...);
...
$fwrite(dut.net0, dut.net1, dut.net2, dut.net3);
...
endmodule
```