Experiment 18
Hardware Watermarking

We describe an experiment to implement IP watermarking to protect circuits from confidentiality and integrity attacks.

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

Theory Background

In images and papers, a watermark is unique pattern that can only be observed in certain conditions to prevent counterfeiters from making recreations of these products. An example of this is the faint text peppered throughout a copyrighted image that prevents others from using the image without paying license and/or attributing the artists or their companies. The same principle can be applied to hardware IPs, where additional circuitry can be used to produce a pattern of signals, states, transitions, etc. that is unique and identifiable by the IP owner. Hardware watermarking helps protect against forms of confidentiality and integrity attacks. The additional circuitry for an IP watermark is expected to produce a hidden and unique pattern that is specific to design and its owner. In this experiment, we will explore two types of watermarking: 1) FSM-Based and 2) Combinational-Based. The goal of this experiment is to give introduction to these techniques in order to better understand ways to prove authorship of an IP as well as protect against attacks, like counterfeiting and others.

1. FSM-Based Watermarking

FSM-Based Watermarking embeds the signature inside the finite state machine of a design. If there are n flip-flops used for the finite state machine, there is a maximum of 2^n FSM states, but in many designs, not all these states are part of the FSM. For example, to implement a simple traffic light controller, you need three states: Red, Yellow, and Green, so in order to implement this FSM, you need at least two FFs, which can implement a max of four states. Thus, there is one unused state in this minimum configuration. A simple FSM-based watermark for this controller is to use the lone unused state that the IP owner can uniquely transitioned to in order to prove they own the IP as shown in Figure 1. Ideally, more unused states can be beneficial in constructing a more robust FSM-based watermark.

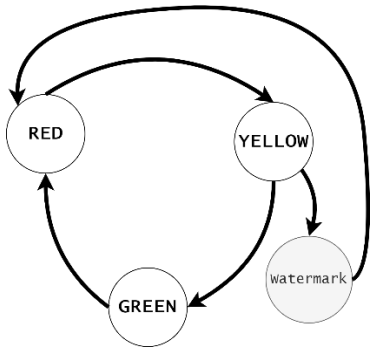

Figure 1 – Simple Traffic Light Controller FSM with watermark state.

2. Combinational-Based Watermarking

Most designs can be broken down into two major elements: combinational logic and sequential elements as shown in Figure 2. In the previous part, we explored watermarking based on the FSM, which only makes modifications to some sequential elements and connected combinational logic to the same. Another technique for hardware watermarking to select nets from the combinational logic and feed them into a digest circuit, which produces the watermark signature outputs. The selection of these nets is crucial to creating a signature that changes when tampering occurs in the IP while producing a unique watermark for a particular primary input pattern only known to the IP owner. These nets are obtained using analysis of various features of the design, such as static probability and toggle rate. The digest circuit helps create the combinational watermark from these selected nets. A simple example of the digest circuit is an XOR tree. The watermark outputs should differ if any of the gates in the design is altered.

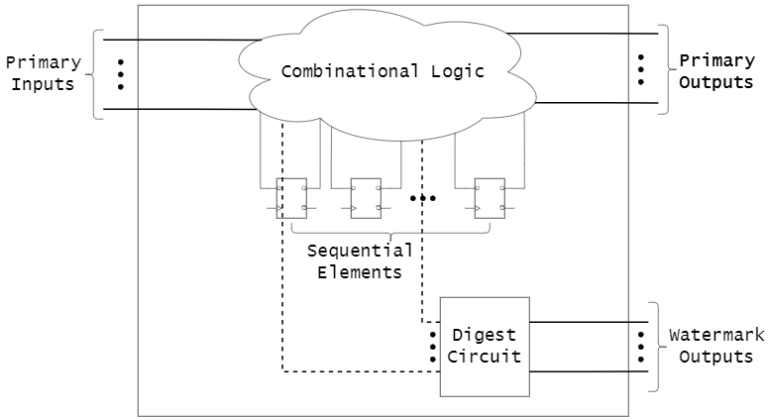

Figure 2 – Overview of Combinational-based hardware watermarking

Experiment Set-up: Configuration

The hardware and software needed for this experiment include:

1. The HaHa Board and USB blaster
2. ECE Linux Server
   a) Design Compiler
3. Ubuntu OS (virtual machine, dual boot, or primary OS)
   a) iVerilog
   b) Python 3

Instructions and Questions

In this experiment, you will insert an FSM-based watermark on a sequential design and a combinational-based watermark with digest circuit on a combinational design.

PART I: FSM-Based Watermarking

1. Create a new Quartus project called **fsm_watermarking**
    a. Reminder: The device model number is: 10M50SAE144C8G
2. Download the provided file **mips_multi_cycle_controller.v**
    a. Include this file by going to **Project** > **Add/Remove Files in Project…**
3. Create a top module called **fsm_watermarking.v** as this will serve as your top module
    a. Instantiate the **mips_multi_cycle_controller** module in this top module.
    b. Expose its inputs and outputs as the primary inputs/outputs for now. This will change later.
4. Before we synthesize the original design, let's change some settings on Quartus to better observe the FSM.
    a. Go to **Assignments** > **Settings**
    b. In the left sidebar, click on Compiler Settings
    c. Locate the button called **Advanced Settings (Synthesis)** on the right side.
    d. Find or filter for "State Machine Processing" and change that setting to "User-Encoded"
    e. Find or filter for "Safe State Machine" and change that setting to "On"
    f. Press OK for both pop-up windows.
5. Synthesize the design by clicking on **Processing** > **Start** > **Start Analysis & Synthesis**
6. Go to **Tools** > **Netlist Viewers** > **State Machine Viewer**
7. Analyze the **mips_multi_cycle_controller.v** file.
    a. Take note of number of used states
    b. Calculate the number of unused states
8. Add a watermark output signal called **watermark_out**

9. Insert an FSM-based watermark that outputs ‘1’ with the following characteristics:
    a. From **State S4_MemWriteback**, add three unused states (S12, S13, S14).
    b. S4 goes to S12 if the Opcode = 6’b110011 else it returns to S0
    c. S12 goes to S13 if the Opcode = 6’b000111 else it returns to S0
    d. S13 goes to S14 if the Opcode = 6’b110001 else it returns to S0
    e. S14 stays in the same state if the Opcode = 6’b001110 else it returns to S0
        i. Output ‘1’ for the watermark output at this state.
10. Wire LEDs and SWs of the FPGA to monitor outputs and provide inputs to the watermarked circuit.
    a. Use LED[3:0] to show the current state of the FSM.
    b. Use LED[7] to show the value of the **watermark_out** output signal.
    c. Use SW[9] to control the clock signal of the MIPS controller.
    d. Use SW[8] to control the reset signal of the MIPS controller.
        i. Note: The reset signal is active low, so plan accordingly.
    e. Use SW[5:0] to control the Opcode input signal of the MIPS controller.
11. Observe the watermark output by toggling the correct state transitions using the toggle switches.
12. Repeat Step 9 with a different set and ordering of watermark states with corresponding different primary input values (i.e., Opcode).
    a. Use three watermark states that are not used by the original FSM logic.
    b. Observe the watermark output by toggling the correct state transitions using the toggle switches.

PART II: Combinational-Based Watermarking

Step I: Analyze the c7552 circuit using Design Compiler on the ECE Linux Server

1. Download the provided file **c7552_analysis.zip**
2. Place this file on your account on the ECE Linux Server and unzip it to a folder **c7552_analysis/**
3. Navigate to that directory. In the terminal, run the following commands:

```
source /apps/settings

dc_shell -f c7552_analysis.tcl
```

4. Analyze the circuit and pick 8 nets for watermarking digest circuit based on either of the following features:
    a. Static Probability: Probability of net being at logic '1'
    b. Toggle Rate: % of transitions ('0' to '1' and '1' to '0') observed at that net

    Note: You are not allowed to select nets within a depth of 1 unit from PI and PO.

Step II: Add watermark based on the circuit analysis and test the watermarked design using Ubuntu machine

1. Download the provided file **c7552_ubuntu.zip**
2. Place this file on your Ubuntu machine and unzip it to a folder **c7552_ubuntu/**
3. Open a Terminal window and run the following commands:

```
chmod +x ./install_watermarking_deps.sh

./install_watermarking_deps.sh
```

Note: You will need to type in your password before the programs are installed in your Ubuntu OS.

4. Add the nets to the digest circuit on **c7552_gsc_w_digest.v**
    a. **Note:** They start with "XOR2X1 dig_" on the Verilog file.
    b. Replace nets with ports '.A' and '.B' with the nets you chose.
5. Pick three 207-bit primary input patterns such that the watermark can identify if the design gets tampered and place them into the **./tb/c7552_tb.v** file. Make sure that there 1:1 mapping between PIs presents in the test bench and patterns that you are providing. Report the methodology used to select the PIP.

6. Make a Python script to generate 100 tamper designs using **wm_tamper_sets.txt** and **wm_tamper_sets_type.txt**
    a. **wm_tamper_sets.txt** indicates the four gates to be replaced in the c7552 watermarked design per tamper design.
    b. **wm_tamper_sets_type.txt** are the replacement gate types.
7. Make a Python script to run iVerilog for the untampered and 100 tampered watermark designs. Below are the sample commands:

```
iverilog -s tb -o {sim_out_location} {watermarked design} {testbench file} {library file}
vvp {sim_out_location} > {vcd_log_out}
```

Note: This will create a file called **tb.log** which gets overwritten every time you run these commands. Make sure to make copies of the log file per each simulation.

8. Make a Python script to parse each output log file per design and score your watermark per each primary input pattern (PIP).
    a. If the resulting watermark output for a tampered design differs from the original watermarked design, then increment the score for that primary input pattern. Sample code is below:

```
score0 = score0 + 1 if W_out_tampered(PIP0/1/2) != W_out_original(PIP0/1/2) else score0
```

    b. Also, calculate the overall score accounting for all selected PIPs and all tamperings. Maximum total score will be out of [# of PIPs x # of tamperings].
9. Repeat Part 4 of Step I as needed to improve your score. A portion of your points will be derived based on the performance of the digest on the instructors set of tampered designs.

☞ Lab Report Guidelines

Deliverables:

In your report, answer ALL the questions and provide the following:

1. For Part I:
    a. Include a screenshot of the State Machine Viewer of the original FSM.
    b. Include a screenshot of the State Machine Viewer of the watermarked design.
    c. Include a picture of the HaHa board to show the watermark transition states and correct input/output.
    d. Report new watermark states and their corresponding inputs of your created FSM watermark.
    e. Include a screenshot of the State Machine Viewer of the new watermarked design.
    f. Include a picture of the HaHa board to show the watermark transition states and correct input/output for your created FSM watermark.
2. For Part II:
    a. Report Static probability and Toggle rate for the design.
    b. Report depth of nets from PI and PO.
    c. Report and explain the process/analysis/methodology used to select the nets for digest.
    d. Report the process/analysis/methodology used to select primary input patterns for the watermark.
    e. Python script that generates tampered designs based on tamper set and tamper set type files.
    f. Python script used to run iVerilog for untampered and 100 tampered benchmarks.
    g. Python script used to test the watermark (script that gives out score for the sample tamperings).
    h. Report scores for the watermark digest.

Demonstration:

1. Demonstrate successful transition to the watermark states and observe a ‘1’ on the watermark output signal on the HaHa board
2. Demonstrate the entire process of combinational-based watermarking with high level details.

References and Further Reading

[1] A. Cui, C. Chang, L. Zhang, “A hybrid watermarking scheme for sequential functions”

[2] A. Nair, P. SLPSK, C. Rebeiro, S. Bhunia, “SIGNED: A Challenge-Response Based Interrogation Scheme for Simultaneous Watermarking and Trojan Detection”