Title of the Experiment: Modchip Attacks

Group Members' Names: Sujan Kumar Saha, Pankaj Bhowmik

Date: 10/29/2019

Abstract

In this experiment, we perform modchip attack. The concept of modchip attack is to replace a chip on a PCB board to change the functionality or bypassing any secret information (e.g. any key-value). As we can't replace a chip on the HaHa board, we emulate the attack by bypassing a key value that is coming from EEPROM. Here, we set up our base system where a key value is stored in EEPROM and microcontroller read that value and send it to the FPGA. The FPGA uses that key to control the on and off operation of the LED. Then, we act as a manufacturer to store a functional key value in the EEPROM and check that the system can use that to control the LED. After that, we act as an attacker where we bypass the key value which is being sent from the EEPROM and control the LED using our own key which is known to the attacker.

Experimental Details

Goals: The goal of this experiment is to emulate the Modchip attack. We will bypass a key which is used to control a function in FPGA.

Experimental Setup: We used Quartus, ATMEL Studio and Analog discovery to do the experiments. We set up our system by following figure 1. A key is stored in the EEPROM. The microcontroller reads the key from the EEPROM through SPI communication. Then, the microcontroller send the key to FPGA. The FPGA uses that key to control the LED light-up functionality.

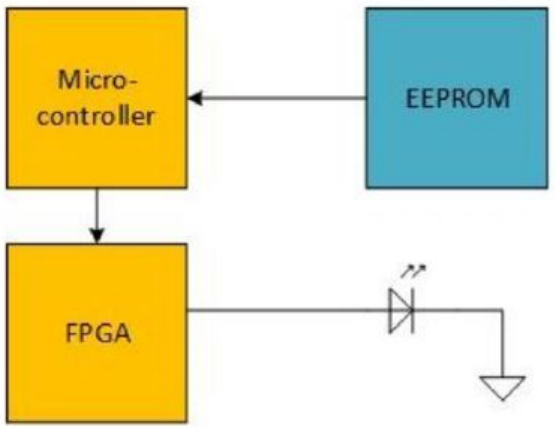

Figure 1: System Setup

Part 1

The objective of this part is to establish an SPI communication between external EEPROM with a microcontroller. SPI communication is operated by the SPI control register, status register, and a data register. To instantiate different modes of operation like read, write, etc, there are some opcodes READ, WRITE, WREN, RDSR and we obtain

those values from the EEPROM datasheet. When the microcontroller successfully establishes the SPI connection, then we send the data to the FPGA on the HAHA board. The FPGA checks the data and if the data is the desired key then it LEDs on the haha board (D1-D4) turns on and off otherwise. Besides, we keep the data to a RAM and check it through in-system memory content editor. And this memory content validates our claim. Figure 2 shows the experimental setup. (For detail please see the video demo)

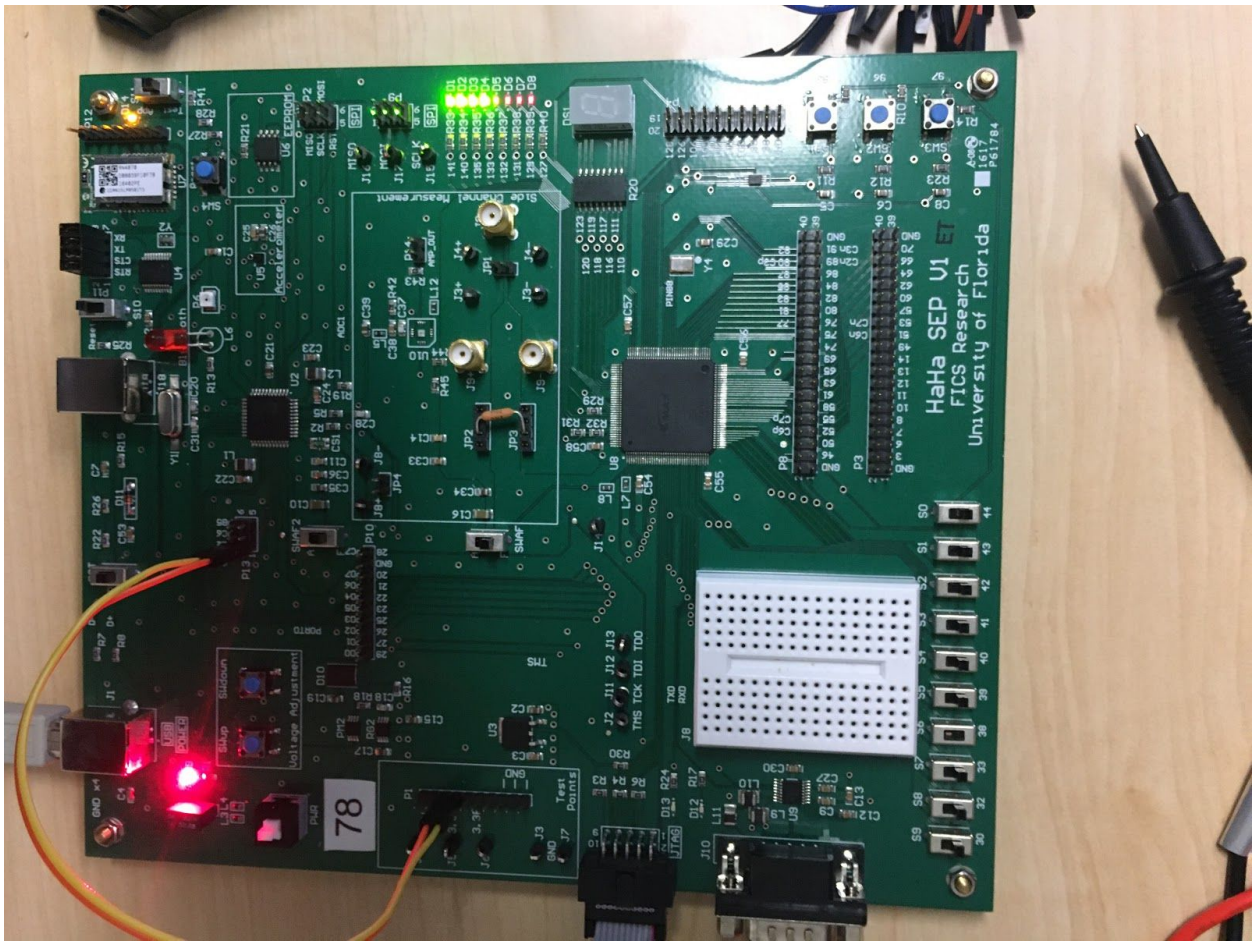

Figure2: Experimental setup. With the functional key D1, D2, D3, and D4 are in the ON state.

Part2

In this part, we set up this design from the manufacturer's point of view. Here, we preset the factory setting of the key to a full-functional key by setting the content of the chosen EEPROM address to be 00000000. After the key is written, restart the board and configure the system as we performed in PART I. The system reads the key from the EEPROM from the aforementioned address. When the FPGA reads the data from that address, it matches with the functional key and turns on the four LEDs on the HAHA board. (For detail please see the video demo)

Part 3

This part is also performed from the manufacturer's point of view. Here we preset the factory setting of the key to an 8-bit binary number different from 00000000. In this case, we used a hexadecimal value of

"**3D**". After writing the key-value to FPGA we observed that the LEDs on the HAHA board are in the off state. The reason is the functional key is different from the limited key. Hence, the limited key disables the function of the system. Besides, we read the content of the memory to assure that we are writing the limited key successfully in the EEPROM and sending the same key to the FPGA. The following figure shows that after asserting the limited key the LEDs are no set. (For detail please see the video demo)

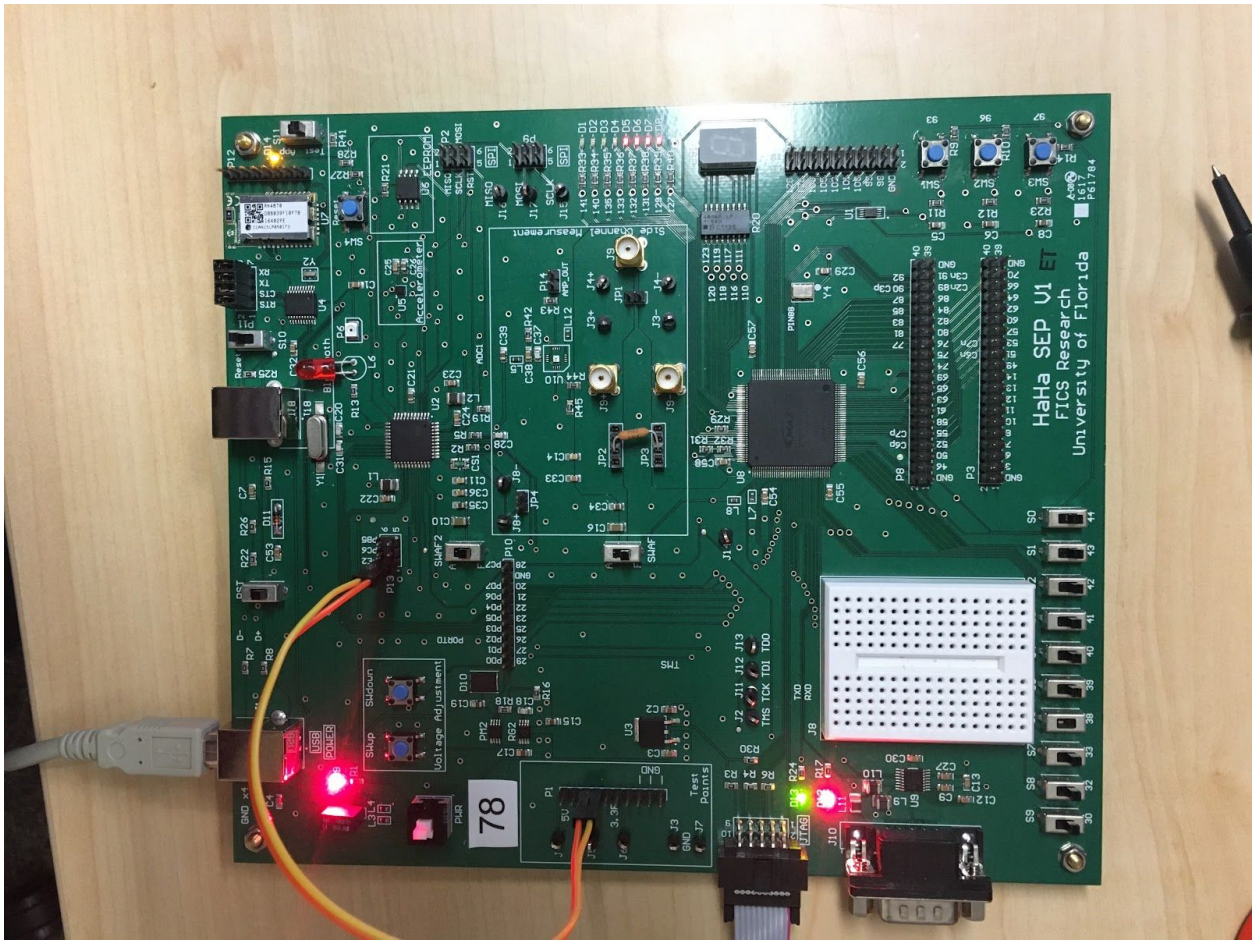

Figure 3: Experimental setup. With the limited key D1, D2, D3, and D4 are in the OFF state.

Part4:

This part is also performed from the hackers' point of view. Here we want to enable the disabled function. This implies the limited key will enable the function in the FPGA. Since the functional key is 00000000, and MISO sends the signal to the microcontroller and FPGA, we will force the signal to zero. In P2 there is MISO and we connected this pin with the ground of the haha board. This operation forces the data on MISO to be zero always and bypasses the limited key. Then it enables the function on the FPGA and turns the LEDs in on state. We added a video demo to show how LEDs are light up by doing this.

Optional Follow-up

Part 1

The question here is whether there are other nets on the HaHa board that can be Madchip hacked to achieve the same effect. Yes, the is another net which is PD0-PD7. We can ground these pins to make the key value to 00000000. The difference with the previous parts is here we are changing the key values while sending from microcontroller to the FPGA. In the previous parts, we changed the value while sending from EEPROM to microcontroller by sorting the MISO pin. The following figure shows the setup for this part.

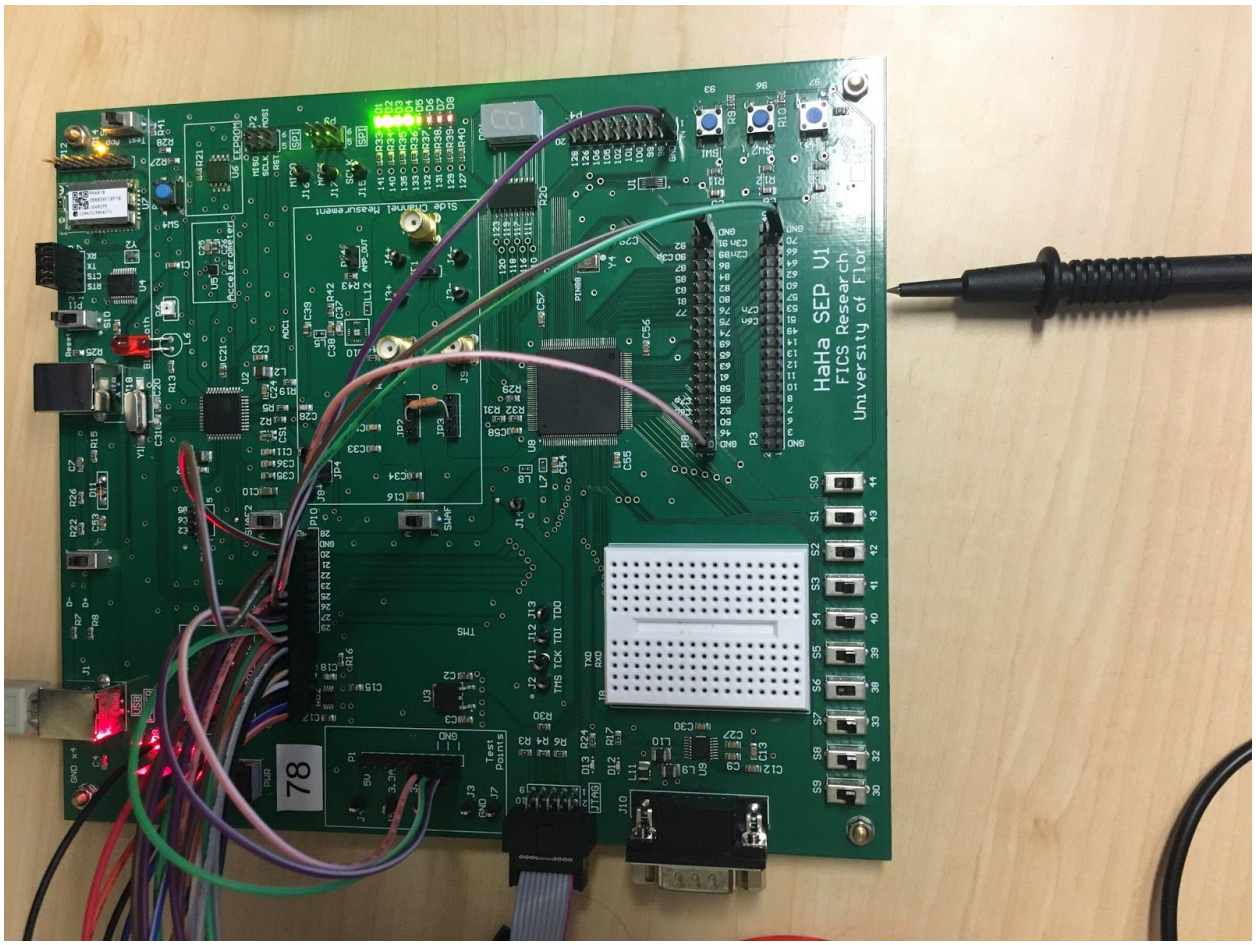

Figure 4: Experimental setup. D1, D2, D3, and D4 are in the ON state. PD0 ~PD7 pins are connected to ground and force the key to be always zero.

Part 2

If the full functional key is 8'b10101010, we can perform a modchip attack by setting the PD0-PD7 pins to VDD and GND. we need to connect PD7, PD5, PD3, and PD1 to VDD on the other hand PD6, PD4, PD2, and PD0 to GND. The experimental setup is almost the same as Figure 3 instead of the VDD connections. We added a video demo to show how LEDs are light up by doing this.

Summary

We perform a modchip attack in this experiment. We acted as a manufacturer and attacker both. We saved a personalized key in the EEPROM as a manufacturer and used that key to control LEDs from FPGA. Then, we acted as an attacker to change the key by bypassing the key-value and control LEDs by the attacker's choice.