Experiment 8
Modchip Attack
on HaHa v3.0 Board

The objective of this experiment is to perform a Modchip attack on a key protected system.

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

Case Study

The objective of this experiment is to simulate a Modchip attack towards a system that is protected by a key.

Figure 1 shows a simplified schematic view of the Microcontroller and Flash memory which provides key to the system. Modchipping this FLASH will cause the system to get a different key (usually a more powerful key). In this experiment, we will change the key data that is communicated through the SPI bus without replacing the FLASH memory. We can force the data on the MISO wire to different values by connecting the Modchip soldering point (please find on P2 or P9) to another source. To make things simpler, we'll short it to ground in this experiment.

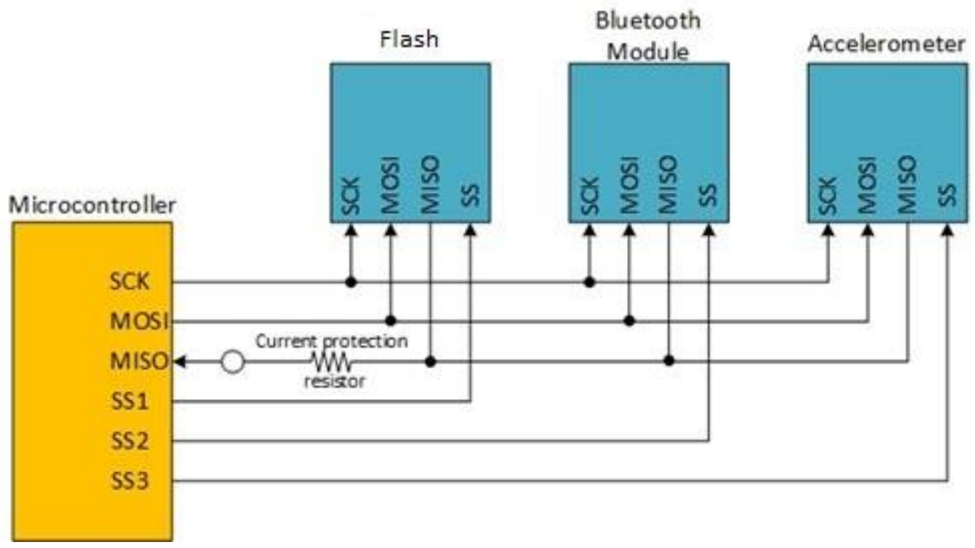


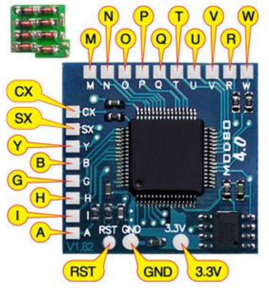
Figure 1 Circuit of the Microcontroller and FLASH

Theory Background

A Modchip is a small electronic device used to alter or disable artificial restrictions of computers or entertainment devices. Modchips are mainly used in video game consoles, DVD players, TV cables, etc. They introduce various modifications to its host system's function, including the circumvention of region coding, digital rights management, and copy protection checks for the purpose of using media intended for other markets, copied media, or unlicensed third-party software.

Modchips replace or override a system's hardware or software protection. They achieve this by either exploiting existing interfaces in an unintended or undocumented manner or by actively manipulating the system's internal communication, sometimes to the point of re-routing it to substitute parts provided by the Modchip.

Most of the Modchips consist of one or more integrated circuits (microcontrollers, FPGAs, or CPLDs), often complemented with discrete parts, usually packaged on a small PCB to fit within the console system it is originally designed for. Although there are Modchips that can be reprogrammed for different purposes, most of the Modchips are designed to work within only one console system or even only one specific hardware version.



Modchips typically require some degree of technical acumen to install since they must be connected to a console's circuitry, most commonly by soldering wires to select traces or chip legs on a system's circuit board. Some Modchips allow for installation by directly soldering the Modchip's contacts to the console's circuit ("quick solder"), by the precise positioning of electrical contacts ("solderless"). In rare cases, they can be plugged into a system's internal or external connector.

The diversity of hardware Modchips operates on varying methods. While Modchips are often used for the same goal, they may work in vastly different ways, even if they are intended for use on the same console. Some of the very first Modchips for the Nintendo Wii known as drive chips, modify the behavior and communication of the optical drive to bypass its security. While on the Xbox 360, a common Modchip took advantage of the fact short periods of instability in the CPU could be used to fairly reliably lead it to incorrectly compare security signatures. The precision required in this attack meant the Modchip made use of a CPLD. Other Modchips, such as the XenoGC and clones for the Nintendo GameCube, invoke a debug mode where security measures are reduced or absent, in this case, a stock Atmel AVR microcontroller was used. The most recent innovations are optical disk driven emulators or ODDE, these replace the optical disk drive and allow data to come from another source by bypassing the need to circumvent any security. These often make use of FPGAs to enable them to accurately emulate timing and performance characteristics of the optical drives.

Experiment Set-up: Configuration

The instruments needed for this experiment are the HaHa v3.0 Board, a USB A to B cable, a USB Blaster, a computer, and wires.

The software needed for this experiment are Microchip Studio, Atmel Flip, and GOWIN FPGA Designer.

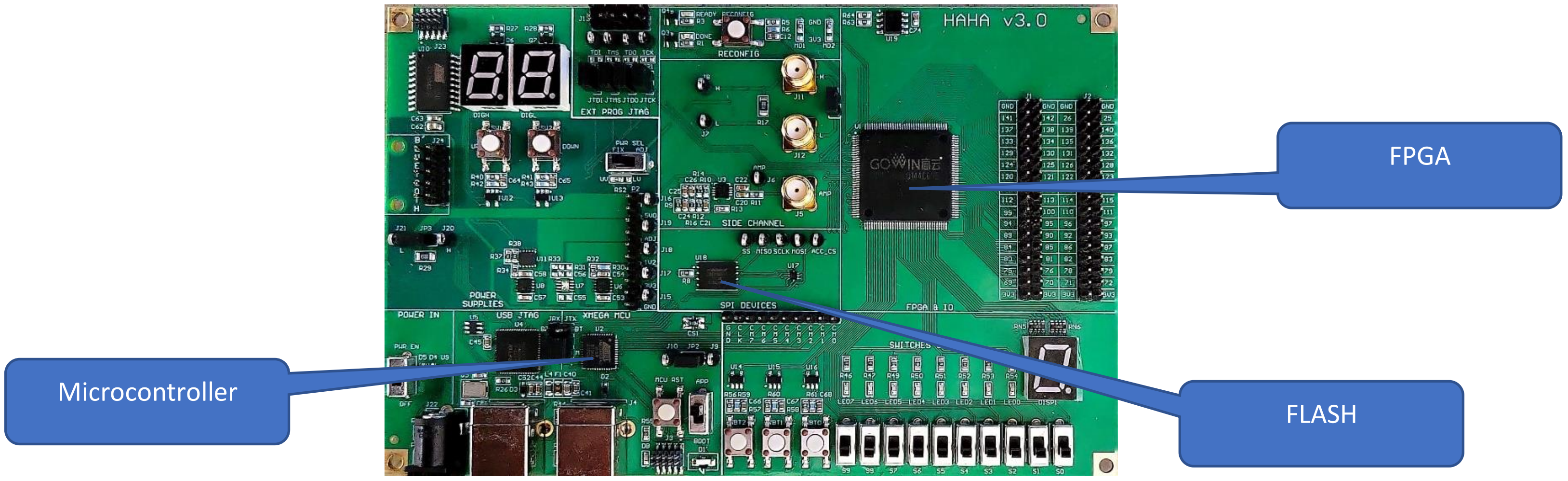


Figure 2 HaHa v3.0 Board

Experiment Set-up: Instructions

In this lab, you will design a simple system whose function is to light up an LED. First, you will play the role of a manufacturer of the system where you will store a limited key into the system's ROM so that the function is inaccessible. Next, you will play the role of a hacker and apply a mod-chip attack toward the system to bypass key protection block and make the system functional. In order to facilitate the SPI programming, please use the *HaHav3_helpful_codes.zip* file from Canvas and include them in your Atmel project. Refer to the Xmega Programming Guide and the Appendix on how to use these files.

Part I: Design a computer system with locking function

Design a system that will have the following properties. The microcontroller reads the 8-bit key from the FLASH and sends the key to FPGA. The FPGA checks whether the key is a full-functional key. Only a key with permission (here we assume only a 00000000 has permission) can enable the function of lighting up a LED. If the key is a limited key (not equal to 00000000), it will disable the function, and nothing will happen. The diagram of the system is shown in Figure 3.

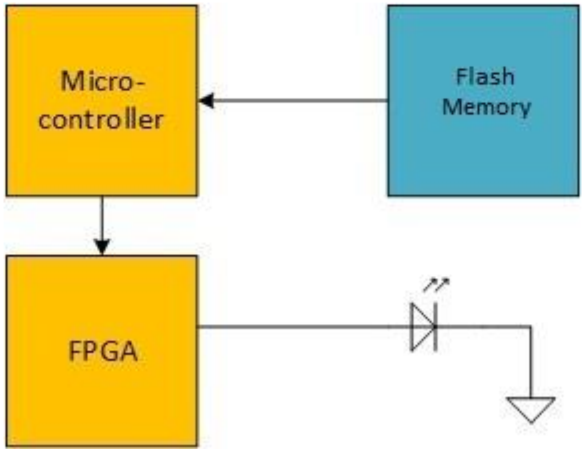


Figure 3 System Structure.

Part II: Observing circuit functionality in full-functional mode

Pretend that you are a manufacturer, and the system is your product. Preset the factory setting of the key to a full-functional key by setting the content of the chosen FLASH address to be 00000000. After the key is written, restart the board and configure the system that you designed in PART I. See how the key with permission can enable the function of the system.

Part III: Observing circuit functionality in limited mode

Pretend you are still the manufacturer, and you are producing the same system but with a limited key. Preset the factory setting of the key to an 8-bit binary number different from 00000000. After the key is written, restart the board and configure the system that you designed in PART I. See how the limited key will disable the function of the system.

Part IV: Perform Modchip attack on limited mode circuit

Pretend you are a hacker, and you only have a system with limited function. Carry out a Modchip attack by connecting the Modchip soldering point (find on header P2 or P9) to GND so that only a low voltage can be read on MISO of the microcontroller. Reading in a serial of low voltage, the Microcontroller is convinced that the key from the FLASH is 00000000. See how the function of the system is enabled by modifying the chip which does not have a full-functional key.

Measurement, Calculation, and Question

1. In this experiment, we need to use the Microcontroller to write data to the FLASH and read data out from the FLASH.
    a. The Microcontroller and the FLASH will communicate through Serial Peripheral Interface (SPI).
    b. Refer to the datasheet [6] of the Microcontroller. There are detailed instructions for programming using SPI.
    c. Turn in your assembly code (or C code) for the Microcontroller.
2. The FPGA will receive the key from the Microcontroller and see if the key has permission for the functions.
    a. How does the FPGA in your design compare keys?
    b. Turn in your Verilog or VHDL description for the FPGA.

✐Optional Follow-up

1) In this experiment, we are hacking the MISO net on the HaHa v3.0 Board by shorting it to the ground.
    a. Are there other nets on the HaHa v3.0 Board that can be Modchip hacked to achieve the same effect?
    b. If yes, where are they?
    c. What is the difference between these two methods?
2) If the full function key is 8’b10101010, how can you perform a mod-chip attack on it?
    a. Demonstrate this attack on your video report

☞ Lab Report Guidelines

Deliverables:

1. In your report, answer ALL the questions.
2. Attach a photo of your experiment set up.
3. Prove you succeed in the attack by giving all your code and the photos of the phenomenon when the attack works.

Demonstration:

Submit videos that fulfill the requirements below.

1. Show the normal function of your system when there is a functional key stored in the FLASH.
2. Show that the system is not functional when there is a wrong key stored in the FLASH and the system will work as soon as you Modchip attack it.

Your videos should follow the format below.

1. start with a commentary (3-4 sentences) on what this experiment is about and what are the goals.
2. followed by a commentary (3-4 sentences) on the steps you follow to do the experiments.
3. Next, you move the camera to show the experiment setup (the HaHa v3.0 board, connection with Analog Discovery 2 and computer), go through the steps one by one on the setup - how you apply the inputs, what components of the HaHa v3.0 board are involved, what you expect to see - and then show the outputs - e.g., recorded signal (if any), output LED or seven segment displays, etc.
4. Next, add a commentary on the key lessons learned through this experiment. If you've done any advanced options, optional follow-up, or explored anything beyond that is covered by the lab instructions, describe them too.
5. The video should have no mistakes.

References and Further Reading

[1] https://en.wikipedia.org/wiki/Modchip

[2] http://www.modchipcentral.com/store/modbo-4.0.html

[3] https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

[4] http://www.avrbeginners.net/

[5] http://ww1.microchip.com/downloads/en/devicedoc/spi.pdf

[6] http://ww1.microchip.com/downloads/en/DeviceDoc/22064D.pdf

Appendix: Notes on the provided files

- Please review the provided files to familiarize yourself with the SPI protocol in the microcontroller.
- For your reference, here are the pin names for FLASH pins:
  - SS (Slave Select): PORTC4
  - HOLD_N: PORTE1
  - WP_N: PORTE2
- Set those pins appropriately for your application.
- Make sure to add the following include statement:

```
#include "haha_v3_xmega.h"
```

- Make sure to uncomment the following line inside haha_v3_xmega.h file:

```
#include "spi_driver.h"
```

- Initialize your SPI master using the following:

```
haha_v3_SPIBegin();
```

- Make sure to set SS low before sending/reading data from SPI and then high after the intended operation.
- Send 1 byte of data to SPI slave using the following:

```
SPI_MasterTransceiveByte(&spiMasterC, <send_data>);
```

- Read 1 byte of data to SPI slave using the following:

```
read_data = SPI_MasterTransceiveByte(&spiMasterC, <send_data>);
```