Experiment 13
JTAG Attack

The objective of this experiment is to implement a JTAG attack in order to change the firmware stored inside the flash of the microcontroller.

Instructor: Dr. Swarup Bhunia
TAs: Shuo Yang, Reiner Dizon, and Miles F Mulet

Theory Background

1. JTAG

The Joint Test Action Group (JTAG) is an electronics industry association formed in 1985 for developing a method of verifying designs and testing printed circuit boards after manufacture. In 1990 the Institute of Electrical and Electronics Engineers codified the results of the effort in IEEE Standard 1149.1-1990, entitled Standard Test Access Port and Boundary-Scan Architecture.

JTAG implements standards for on-chip instrumentation in electronic design automation (EDA) as a complementary tool to digital simulation. It specifies the use of a dedicated debug port implementing a serial communications interface for low-overhead access without requiring direct external access to the system address and data buses. The interface connects to an on-chip test access port (TAP) that implements a state-based protocol to access a set of test registers that present chip logic levels and device capabilities of various parts.

1.1 Electrical characteristics

A JTAG interface is a special interface added to a chip. Depending on the version of JTAG, two, four, or five pins are added. The four and five pin interfaces are designed to support multiple chips so that they can have their JTAG lines daisy-chained together when specific conditions are met. The 2-pin interface is designed so that multiple chips can be connected in a star topology. In either case, a test probe is needed to connect to a single JTAG port in order to access to all chips on a circuit board.

The connector pins are:

TDI (Test Data In)

TDO (Test Data Out)

TCK (Test Clock)

TMS (Test Mode Select)

TRST (Test Reset), optional

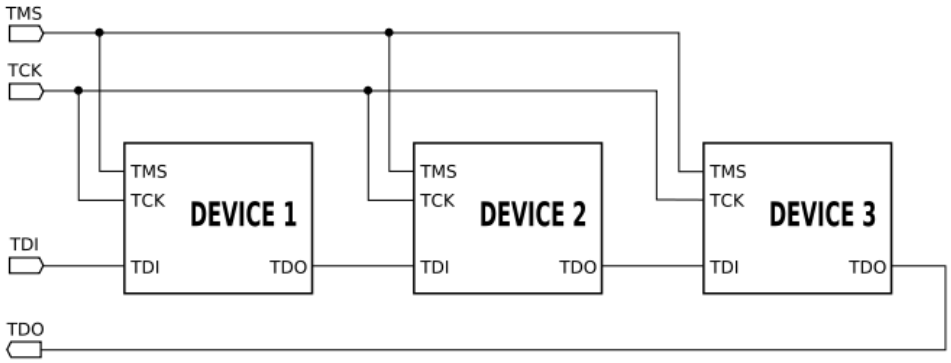

Figure 1 JTAG circuitry

1.2 TAP controller

The function of JTAG is controlled by a TAP controller. The TAP controller is a 16-state finite state machine that controls the operation of the Boundary-scan circuitry, JTAG programming circuitry, or On-chip Debug system. The state transitions depicted in Figure 2 depend on the signal present on TMS (shown adjacent to each state transition) during the rising edge of TCK. The initial state after a Power-on Reset is Test-Logic-Reset. Here is some typical scenario for using the JTAG interface:

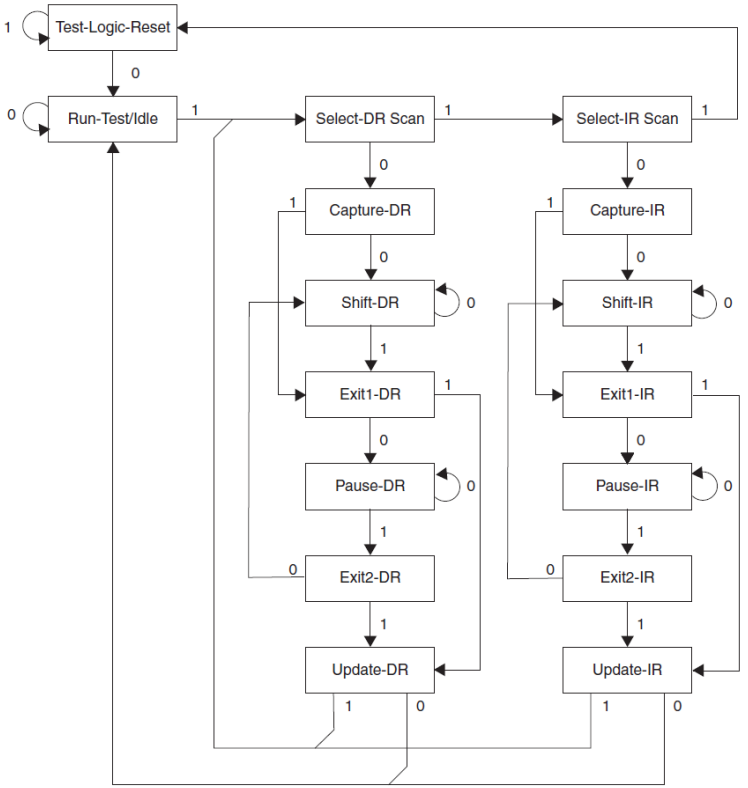

Figure 2 TAP controller state diagram

- No matter what is the present state, applying a sequence of five 1's or more at the TMS input (at the ***rising*** edges of TCK, the same below) will make the TAP to enter the Test-Logic-Reset state.
- Assuming Run-Test/Idle is the present state, applying the sequence 1, 1, 0, 0, at TMS to make TAP to enter Shift-IR. While in this state, shift the JTAG instructions (they are 4-bit instructions for our microcontroller, ***LSB*** shift in first) into the JTAG Instruction Register from the TDI input at the rising edge of TCK. The TMS input must be held ***low*** during input of the three LSBs for the state machine to remain in the Shift-IR state. The MSB of the instruction is shifted in when this state is left by setting TMS high.
- Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. The instruction is latched onto the parallel output from the Shift Register path in the Update-IR state.
- At the TMS input, apply the sequence 1, 0, 0 to enter the Shift Data Register – Shift-DR state. While in this state, upload the selected Data Register (selected by the present JTAG instruction in the JTAG Instruction Register) from the TDI input at the rising edge of TCK. To remain in the Shift-DR state, the TMS input must be held low during input of all bits except the MSB. The MSB of the data is shifted in when this state is left by setting TMS high.
- Apply the TMS sequence 1, 1, 0 to re-enter the Run-Test/Idle state. If the selected Data Register has a latched parallel output, the latching takes place in the Update-DR state.

1.3 JTAG instructions

In this part, we will introduce not only the instructions for boundary scan but also for JTAG programming. Note that different devices will support different instructions. The instructions below are the instructions for the microcontroller we are using. The Instruction Register is 4-bit wide, supporting up to 16 instructions. The LSB is shifted in and out first for all Shift Registers.

- **EXTEST**: 0x0. Mandatory JTAG instruction for selecting the Boundary-scan Chain as Data Register for testing circuitry external to the AVR package. The contents of the latched outputs of the Boundary-scan chain is driven out as soon as the JTAG IR-Register is loaded with the EXTEST instruction.
- **IDCODE**: 0x1. Optional JTAG instruction selecting the 32-bit ID-Register as Data Register. The ID-Register consists of a version number, a device number and the manufacturer code chosen by JEDEC. This is the default instruction after power-up.
- **SAMPLE_PRELOAD**: 0x2. Mandatory JTAG instruction for pre-loading the output latches and taking a screenshot of the input/output pins without affecting the system operation. However, the output latches are not connected to the pins. The Boundary-scan Chain is selected as Data Register.
- **AVR_RESET**: 0xC. The AVR specific public JTAG instruction for forcing the AVR device into the Reset mode or releasing the JTAG reset source. The TAP controller is not reset by this instruction. The one bit Reset Register is selected as Data Register. Note that the reset will be active as long as there is a logic “one” in the Reset Chain. The output from this chain is not latched.
- **BYPASS**: 0xF. Mandatory JTAG instruction selecting the Bypass Register for Data Register.
- **PROG_ENABLE**: 0x4. The AVR specific public JTAG instruction for enabling programming via the JTAG port. The 16-bit Programming Enable Register is selected as Data Register. The programming enable signature is 0b1010_0011_0111_0000.
- **PROG_COMMANDS**: 0x5. The AVR specific public JTAG instruction for entering programming commands via the JTAG port. The 15-bit Programming Command Register is selected as Data Register.
- **PROG_PAGELOAD**: 0x6. The AVR specific public JTAG instruction to directly load the Flash data page via the JTAG port. An 8-bit Flash Data Byte Register is selected as the Data Register.
- **PROG_PAGEREAD**: 0x7. The AVR specific public JTAG instruction to directly capture the Flash content via the JTAG port. An 8-bit Flash Data Byte Register is selected as the Data Register. This is physically the 8 LSBs of the Programming Command Register.

2. JTAG attacks

There is a set of possible attacks, each of which has attack requirements. For example, an attack to capture the configuration stream of an FPGA as the configuration bits are being sent over JTAG requires the ability to sniff the JTAG data line. Each potential attacker will have a set of attack capabilities not masked by defenses, which determines his or her set of possible attacks. In a given attack scenario, the attacker possesses a limited set of capabilities:

- Sniff the TDI/TDO signals,

- Modify the TDI/TDO signals,
- Control the TMS and TCK signals, or
- Access the keys used by testers.

2.1 Sniff secret data

Figure 3 shows a simple sniffing attack. The attacker's goal is to learn a secret that is being sent to a victim chip via JTAG. An additional requirement for this attack is that the victim chip is downstream from the attack chip on the same JTAG chain. As Figure 3 shows, the attack chip exhibits a false BYPASS mode that is externally identical to true BYPASS mode, but which parses JTAG signals. When the secret is sent to the victim chip, the attack chip captures a copy. The secret is then delivered to the attacker either through JTAG interrogation of the attack chip in the field or through a side channel. Alternatively, the attack chip might directly make use of the sniffed data.

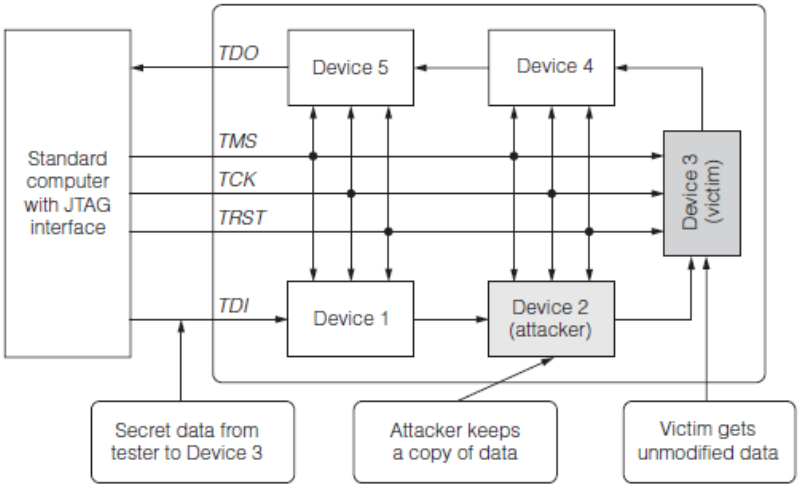

Figure 3 The attacker obtains secret data by sniffing the JTAG data path.

2.2 Read-out secret

In a read-out attack, the attacker's goal is to learn a secret that is contained in the victim chip. We assume the attacker is capable of using I/O drivers in the upstream attack chip to control the TMS and TCK lines. An additional requirement for this scenario is that the attack and victim chips are on the same JTAG chain with the victim chip sandwiched between the attack chips. The upstream attack chip forcefully acts as the JTAG bus master and performs a scan operation on the victim chip to access embedded secrets. The downstream attack chip collects the secret as it emerges from the TDO of the victim chip. This technique can be used by embedded attackers as described or by an attacker who can attach external hardware to the system under attack.

2.3 Obtain test vectors and responses

In the test vector collection attack, the attacker's goal is to learn both the vectors used to test the victim chip and the normal responses. This attack requires that the attack chip be downstream from the victim on the same JTAG chain. The upstream attack chip collects test vectors as they are sent to the victim chip. The downstream attack chip collects the responses as they propagate from the device under test back to the tester. Knowledge of the test vectors and responses helps an attacker further infiltrate a system by enabling the attacker to produce Trojans that pass normal tests.

2.4 Modify state of authentic part

The attacker's goal is to modify the state of victim chip. We assume that the attacker can insert strong I/O drivers in the attack chip to forcefully control the TMS and TCK lines. An additional requirement for this attack is that the victim chip is downstream from the attack chip on the same JTAG chain. The attack chip takes control of the TMS and TCK lines and puts the JTAG test access port of the victim chip into a state where it can shift data in, thereby setting the state of registers within the victim chip, including registers that affect its normal operation.

2.5 Return false responses to test

In a false responses attack, the attacker's goal is to deceive the tester about the victim chip's true state. An additional requirement for this attack is that the victim chip is downstream from the attack chip on the same JTAG chain. The tester attempts to apply test vectors to the victim chip, which is not the first chip in the JTAG chain. To do this, the tester tries to place the other chips into BYPASS mode. The attack chip ignores this request and intercepts the test vectors while instructing the victim chip and other downstream chips to enter the BYPASS mode. The attack chip can then transmit the bogus test responses back to the tester.

Experiment Set-up: Configuration

Two groups will join to finish this experiment. The hardware and software needed for this experiment include:

1. Two HAHA Boards and USB blasters.
2. A computer.
3. Wires to build the JTAG chain between two HAHA Boards.
4. Quartus to program the FPGA.
5. Matlab to process data.

The JTAG header pin assignment are given below in the table.

Table 1 JTAG header pin assignment.

Pin Number	Function
1	TCK
3	TDO
5	TMS
8	JTAGEN (not used in this experiment)
9	TDI

Instructions and Questions

In this experiment, we will first build a hacking tool, which is a JTAG programmer, into one of the FPGAs. Then use it to prove that we can use the JTAG scan chain to shift out data while the chip is working. In the third part, we will use the hacking tool to hack the micro controller on the **other** HAHA Board. For one HAHA Board, the FPGA and the micro controller are in the same JTAG chain. If we want to use the FPGA to hack the micro controller which is in the same chain; the JTAG ports of the FPGA will not only transfer instructions data from the micro controller, but also communicate with a PC through the USB blaster for us in order to collect data, which is not possible. That's why we need at least two HAHA boards for this experiment.

PART I Implement a hacking tool in the FPGA

In this part, we will build a hacking tool, which is a JTAG programmer, into the FPGA. The hacking tool will need two functions: interfacing with the victim chip and collecting responses from the victim chip.

As shown in Figure 4, the programmer should have the four JTAG ports: TDI, TDO, TMS, and TCK in order to interface with the victim chip. However, here the TDI is sending data to the TDI port of the victim chip, it should be an output port. Similarly, the TDO will be an input port. Also, data input from TDO will feed the RAM that built in the programmer. You should allow In-System Memory Content Editor (the ***Editor***) to capture and update the content of the RAM through the USB blaster so that you can monitor what has been shifted out from the victim chip.

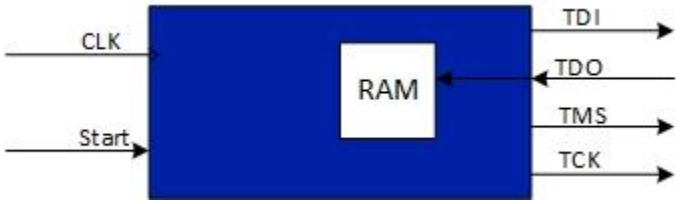

Figure 4 JTAG Programmer.

In order to get what you want from the TDO port, you should send instructions properly at port TMS and TDI on the positive edge of TCK. Refer to the TAP state diagram and instruction set to see what should be shifted out and finish the questions below.

1. Put nothing in the chain. By connecting the TDI and TDO directly together, we put nothing in the chain. Keep sending pattern '101010…' from the TDI port and see what's been received from TDO and stored in the RAM. Turn in your code and screenshot of the Editor.
2. Put the bypass register in the chain. By connecting all the four ports to the victim chip and sending BYPASS instruction through TMS, we put the bypass register in the chain. Keep sending pattern '101010…' from the TDI port and monitor the content of RAM. Turn in your code for this step.

3. Put the 32-bit ID-Register in the chain. Send instruction IDCODE to the victim chip to put the ID register in the chain. Shift whatever you want from TDI and shift out the 32-bit ID from the target register. What is the 32-bit number you shift out? Turn in your code and the screenshot of the Editor.

PART II

In this part, we will prove that the JTAG scan chain can be used to scan in and out data even when the chip is working and using the IO ports. As shown in Figure 5, there is a microcontroller and an FPGA in the same JTAG chain on the HaHa Board. While here we only want to use one of them. Therefore, here we can just connect a wire between the pin 18 and 19 of the FPGA to make a short circuit.

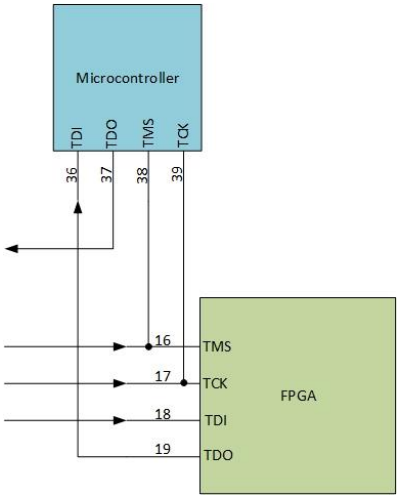

Figure 5 JTAG circuit.

Use another four normal IOs of the FPGA to connect to the four JTAG IOs of the microcontroller. Normal IOs means IOs that you can find on the headers and you can choose any four of them. Do not use the four JTAG IOs of the FPGA for the reason that these four pins need to be used as a programming method. This is very important. Otherwise, the FPGA may be unprogrammable in the future. Use the tool that you built in PART I to scan in and out data and store the data in the RAM.

Program the microcontroller with the hex file you used in the bus snooping attack so that the chip is communicating with the accelerometer through its IOs. Repeat using the tool to scan and store data to prove that it will not be hindered even when the microcontroller is using its IOs.

PART III Hack the victim chip with your hacking tool

In this part, we will first read out what is the firmware that is stored in the victim chip, and then modify the firmware with our JTAG hacking tool so that the victim chip will work differently.

Table 2 Firmware for programming

Firmware 1 (F1)	:020000020000FC :0800000000C0279A2F9AFFCFE0 :00000001FF
Firmware 2 (F2)	:020000020000FC :0800000000C0279A2F98FFCFE2 :00000001FF

1. Create a HEX file and copy F1 as shown in the table above into it. Program the flash of the victim chip with created HEX file. Reboot the HAHA board so that the victim chip will start executing the firmware. What is the function of F1? What is the phenomenon?
2. Make your JTAG hacking tool to use JTAG programming instructions to 'PROG_PAGEREAD' the content of the flash of the victim chip. Store what is shifted out into the RAM of the tool. Compare the content of the RAM with the original code of F1 to see if they are same. Turn in your code of the hacking tool for this step. Obviously, it will show all the procedures and instructions you implemented to get this done. Also, turn in a screenshot of the Editor.
3. (Optional and extra point) Modify the firmware from F1 to F2 with your hacking tool. This requires the tool to write F2 into the flash of the victim chip. Follow the procedure to program the flash with JTAG and change the code for your hacking tool according to that. After modifying the firmware from F1 to F2, you can leave the programming mode and then reboot the board. How the board is functioning now? How is that different from the function of F1? Turn in your code of the hacking tool for this step.

☞ Lab Report Guidelines

1. Follow the steps in part I to build a JTAG hacking tool. Make sure it works by succeeding in putting whatever register in the chain and getting what you want from the output.
2. Follow the steps in part II to hack a microchip with the tool you built. First, program the microchip in the normal way so that it can run an unknown program. Then hack it with the JTAG hacking tool so that it runs the way you want. Attach pictures showing the HAHA boards before and after your hacking.
3. For both the parts you did, submit all the code that's needed.

References and Further Reading

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

[2] Atmel ATmega16U4/ATmega32U4 Datasheet.

[3] Rosenfeld, Kurt, and Ramesh Karri. "Attacks and Defenses for JTAG." IEEE Design and Test of computers 27.1 (2010): 36-47.

[4] http://blog.senr.io/blog/jtag-explained

[5] https://en.wikipedia.org/wiki/Intel_HEX