Hardware Security group #1

Paolo Monti – Alessandro Salvato – Michele Simili

This project originates from the courses of Computer Architectures and Specification and Simulation of Digital Systems, and the reference professors are Ernesto Sanchez and Luca Sterpone, respectively.

Project specifications

Due to the increasing costs of manufacturing chips smaller and smaller transistors, the vast majority of companies is forced to employ and trust a third party to fabricate their design. This makes them very susceptible to malicious circuitry injected into their chip at fabrication time. We want to focus on digital hardware that can cause unintended behavior (trojan) and possible ways to

The project is divided into two parts: understanding how a digital hardware trojan works by developing an example, and how to mitigate or deny its malicious effects. This file will focus on the first part only.

For the first part, the assignment consists in a module that simulates a hardware attack on the miniMIPS processor. This module, that we will call Sniffer from now on, has to implement the following features:

- Detect when a certain sequence of specific instructions has been executed and deploy a payload by activating a signal. Three variants are proposed.
    1. When, e.g., the sequence I1-I2-I3 is detected.
    2. When the sequence I1-any instruction(s)-I2-any instruction(s)-I3 is detected.
    3. When any permutation of the sequence I1-I2-I3 is detected.
- Count how many times each instruction is executed and store each value for future inspection. Thanks to a control signal, the possible operations are two:
    1. Dump all the data gathered up to this point.
    2. Configure a register so that the counting will stop when a certain instruction is executed a certain amount of times, then dump all data.
- Generate a digital signature based on the instructions that get executed.

Each of these tasks will be carried out by modules that we will call Trojan, Profiler and MISR, respectively.

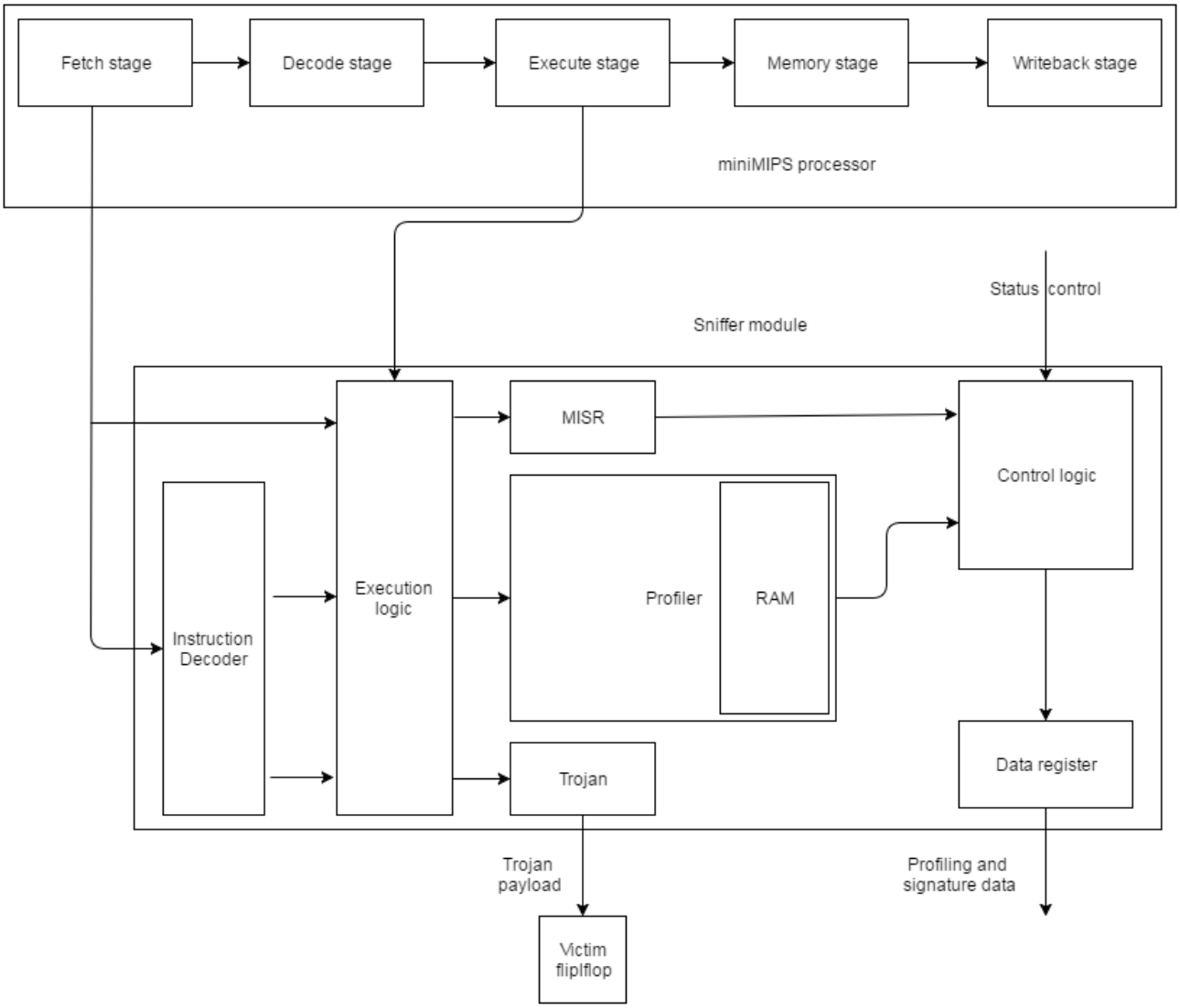
Implementation

After receiving the specifications, we retrieved some documentation about the miniMIPS and agreed upon some design details.

The miniMIPS is a 32-bit pipelined processor, with 32 32-bit registers and a fixed instruction length of 32 bits. This makes it very easy to spy the data coming from the instruction register into the fetch stage. On the other hand, since it is a pipelined processor, it’s not as easy to detect whether or not a fetched instruction is actually executed of not. In fact, in case of a branch, the instruction

following that branch could be fetched, but not executed since the branch could move the Program Counter to a different location in the code.

This is a simplified block diagram of our design.



The instruction decoder takes the 32 bits that encode an instruction, including the operands, and retrieves its corresponding opcode on 6 bits, identifying the instruction itself. The MISR module will account for the operands as well, so no decoding is performed.

The so called Execution logic is a module that will hold the data coming from the instruction decoder (or not) and hold it until the execution of that instruction is confirmed, to avoid profiling instructions that are fetched but not executed.

The MISR is a simple register that performs mathematical operations at each load cycle, generating a (almost) unique sequence of 32 that can only be replicated by loading the same data in the same order.

The Trojan is a Finite State Machine that will change state depending on the instruction that gets executed, and activates the payload signal when the target sequence is detected.

Hardware Security group #1

Paolo Monti – Alessandro Salvato – Michele Simili

The Profiler is essentially composed by a RAM and some logic to control it. When a new instruction is executed, it should retriever the value of the counter for that instruction, increment it by 1, and store it back into the memory. When designing the RAM, we had to decide its dimensions. Since the possible instructions of the miniMIPS are 51, no more than 51 words are needed. About the width of each word, we tried to choose a reasonable number of bits to effectively count how many times each instruction is executed. Since 8 bits seemed too few and 16 too many, we agree upon having a 16-bit word that stores, on the 10 most significant bits, the counter of an instruction, and on the 6 least significant bits, the opcode for that instruction. This implementation makes use of the 6 extra" bits for a safer data dump. In fact, when the profiling data will be sent to the outside world, having the opcode next to the counter of an instruction will allow for much easier handling of eventual packet loss.

Finally, the control logic allows, by the use of an external signal, to configure the module's behavior and state. Possible states are Off, Idle, Active, Output profile data, Output signature data, and so on.

When sending signature data, the module simply writes the output of the MISR module on the 32-bit register that interfaces with the outside world. Instead, when dumping profile data, this operation must be carried out in the least amount of clock cycles as possible. Our proposed solution is to implement a "read 2 words" operation on the RAM so that we can output the count of two instructions at once. This way, we are able to begin and end the communication in 26 clock cycles at most. Then the Profiler can start again, or stay idle until a new control command is issued.