Hamming Distance in Ripes and its uses

Federico Villa
federico5.villa@mail.polimi.it

Abstract. Ripes is a visual computer architecture simulator built around the RISC-V ISA. In this work I extend the capabilities of Ripes by incorporating the calculation of Hamming Distance, a metric for analyzing data disparities, thus improving its usefulness for circuit simulation and allowing the simulator to be used to test side-channel attacks.

1 Introduction

Computer architecture simulators are very useful for understanding the functioning of extremely complicated components such as processors. By seeing a processor running you can better understand how each individual component that makes it up works. Among these there is *Ripes*, a visual microarchitecture simulator. Ripes is an open-source microarchitecture simulator built around the RISC-V ISA (*Instruction Set Architecture*).

Its source code can be found in the Ripes GitHub repository. It is developed primary in C++ and utilizes the Qt framework for its graphical user interface.

Ripes stands out as a notable open-source project due to its widespread adoption and active maintenance: it is a continuously expanding and evolving project. It is utilized in various educational institutions worldwide, as a tool in computer architecture courses and for academic research purposes.

2 RISC-V

RISC-V is an open source instruction set architecture designed to be simple, modular and easily extensible. The architecture is suitable and used in a wide range of computing devices, from embedded systems to high-performance servers. The RISC-V ISA follows the Reduced Instruction Set Computing (*RISC*) philosophy, that aims to provide a minimalistic instruction set with a small number of instructions that perform simple tasks.

"A RISC processor has an instruction set that is designed for efficient execution by a pipelined processor and for code generation by an optimizing compiler." — Michael Slater, Microprocessor Report.

RISC-V has gained significant importance in both academia and industry, with a growing ecosystem of tools, libraries, and community support. It is widely used as an excellent platform for teaching computer architecture, digital design, and low-level programming concepts in various universities around the world.

The choice of a RISC-V architecture simulator for this project is not casual. Its simple architecture, combined with a small set of basic instructions, facilitates the analysis of execution patterns and data flows.

This simplicity is not only useful because it makes integrating the Hamming Distance easier, but also because it is simpler to understand the behavior of an executed process, which is crucial for conducting side-channel attacks.

3 How does Ripes work?

Ripes supports different processor models with various instruction set architectures that vary from a single-stage pipeline processor to more complex models like a five-stage pipeline architecture with a 64-bit address space,forwarding and hazard detection units or a more complex six-stage pipeline with dual-issue execution.

Those processors are implemented in the simulator using *VSRTL - Visual Simulation of Register Transfer Logic*, an open source C++ framework that simulates digital circuits. The framework defines the logic at the base of each component of the processors.

Each component is simulated faithfully to the real component: from the processor handler, to the syscall execution. Each processor can interact with the memory, input and output through defined C++ classes that represent those entities.

In a now discontinued branch of the Ripes GitHub repository there was an implementation of a processor model that wasn't implemented through *VSRTL*, instead it employed a processor described in Verilog (*PicoRV32*) and then converted into C++ using Verilator.

4 Hamming Distance

The Hamming distance is a measure of the difference between two values and it is calculated as the number of positions at which the corresponding symbols of two values are distinct. The Hamming distance between two numbers consists of the number of digits in which their binary representation differs.

Given two integers `x` and `y`, defined with their binary representation:

$$x = \sum_{i=0}^{n-1} d_i \cdot 2^i \qquad\qquad y = \sum_{i=0}^{n-1} e_i \cdot 2^i$$

Their hamming distance is:

$$HammingDistance(x, y) = \sum_{i=0}^{n-1} d_i \oplus e_i$$

Let $x = 1011001001$ and $y = 1001000011$ be two binary numbers. To compute the hamming distance between them, we perform the operation bit by bit:

$$x = 10\underline{1}100\underline{1}0\underline{0}1 \qquad y = 10\underline{0}100\underline{0}0\underline{1}1$$

$$HammingDistance(x, y) = 3$$

In the modified version of the simulator each component contains two sets of data:

- The Hamming distance between the current port value and the value immediately preceding it.
- A histogram consisting of 65 values; each value in the histogram indicates the number of occurrences of the respective Hamming distance between two clock cycles in the component.
- A sequence of values that contains, for each component and each clock cycle, the Hamming distance.

The Hamming distance has significant application in the realm of side-channel attacks, a class of computer attacks that obtain sensible information by observing the physical or temporal behavior of a system during the execution of cryptographic operations. This class of attacks is widely used in embedded devices such as smart cards or IoT devices.

In these attacks, the Hamming distance can be used to analyze variations in power consumption associated with different operations. Because cryptographic operations usually involve a significant number of bitwise and arithmetic operations, variations in the input data bits can directly influence variations in the system's power consumption.

5 Implementation

The goal of this project is to increase the functionality of the Ripes simulator by integrating the Hamming Distance calculation for each component of the simulated processors. When integrating the logic for the Hamming distance, particular attention was given to placing the computation in an optimal point. I tried to modify the simulator as little as possible and at the same time be able to create modifications that could be regardless of the processor model and the selected ISA, so as to guarantee functionality even in the event of the addition of new processors or instruction sets architecture. This approach not only facilitated a standardized and modular implementation but also maximized the performance of the simulator.

Taking advantage of the modular architecture of the simulator, in which each processor is nothing more than a set of logical components created with the VSRTL framework, I implemented the Hamming distance calculation logic at the lowest possible level: directly in the VSRTL library.

A component created with the *VSRTL* framework consists of some input and output ports, connected to the input or output ports of other components. A specific internal function connects the input ports to the output ports of the same component.

There is a common design for all the components which is responsible for propagating the values in all the ports of each component at each clock cycle, thus allowing the present values to be updated.

The Hamming Distance calculation was added in the function that updates the port values at each clock cycle.

The computation is managed by a class created following the *Singleton* design pattern. Each simulated component is described in the class with a map,

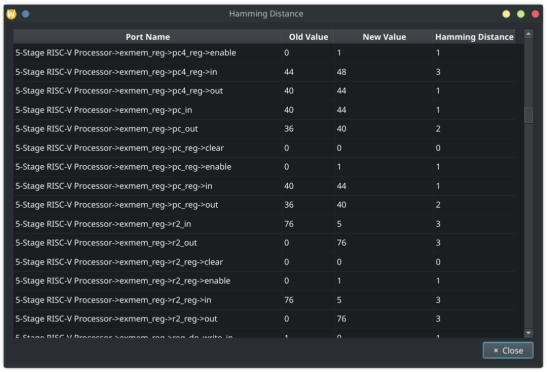
where each element is a pair of values: the key, represented by the component name, and the value, which consists of a C++ struct containing the current Hamming distance, a histogram and a vector with the previously calculated Hamming distances at each clock cycle and support values.

I implemented a *Singleton* to ensure that there is only one instance of the class throughout the program's execution. This ensures that the calculated Hamming values, which are shared among all components, remain consistent and accessible across different parts of the code, such as those related to the GUI and the CLI.

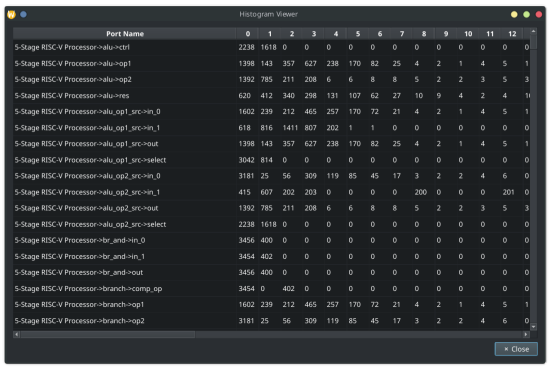
6 Integration of Hamming Distance Features into GUI and CLI

I have added a new entry in the simulator GUI main menu, "Hamming Distance". It includes three buttons:

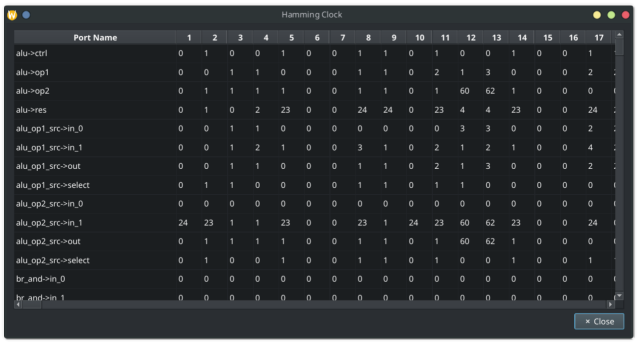
- A button that opens a new window in which it's displayed the Hamming distance of the processor components at the given instant.



- A button that opens a new window to show the histograms of the Hamming distances.



- A button that opens a new window to show the Hamming distances of each component at each clock cycle.



- A checkbox to indicate whether to clear the previously calculated values at every simulator reset.

The simulator also features a command-line interface (*CLI*), which offers a more efficient way to execute source code due to its lower computational resource requirements. Additionally, for conducting side-channel attacks, utilizing the CLI is preferable due to its capability to easily parallelize executions.

For those reasons I decided to implement the Hamming features in the command line interface too. The CLI requires command-line options to function; these options allow users to specify parameters such as the processor model, ISA extensions, or even the source file to execute. I have created new options that, when set, allows printing the previously calculated values the end of execution.

7 Python CLI Script

To facilitate the extraction of hamming distance metrics from the CLI, a python script was created that calculates and extracts the hamming distances from different runs in parallel. The script allows you to run the same program with different values by searching in the source code to be executed the `$VAL` keyword and replacing it with values randomly generated or taken from a given file. The script performs a parallel execution of the CLI Ripes simulator with the option to print the Hamming values enabled. To handle parallel execution the python library *multiprocessing* is used: a parallel task is generated for each value that replaces keywords in the source file.

The data extracted is stored in a "*.dat*" file. A DAT file comprises a fixed-length metadata header followed by a simple arrangement of binary values. All multi-byte values are recorded in little-endian order, facilitating buffer filling on little endian devices using C *read()*.

A *Trace* in this context is a set of hamming distance measurement that gives information about the microcontroller internal state.

The DAT structure consists of:

- Fixed-length Metadata Header
- 1^{st} trace
- 1^{st} input to the device, corresponding to the side channel behaviour of the 1^{st} trace

- 2^{nd} trace
- 2^{nd} input to the device, corresponding to the side channel behaviour of the 2^{nd} trace
- ...
- ...
- n^{th} trace
- n^{th} input to the device, corresponding to the side channel behaviour of the n^{th} trace

The DAT header comprises:

- A 32 bit unsigned integer, representing the number of traces.
- A 32 bit unsigned integer, representing the number of samples of a trace.
- A single character, indicating the data type of the single trace sample, according to the following convention:
  - '*c*': 8-bit signed integer (`int8_t`).
  - '*h*': 16-bit signed integer (`int16_t`).
  - '*f*': 32-bit IEEE-754 float.
  - '*d*': 64-bit IEEE-754 double.
- A 8 bit signed integer, representing the length of the input in *bytes*.

8 Side Channel

As mentioned above, the changes were made to allow the simulator to be used to conduct side-channel attack analysis. The simulator serves as an ideal microcontroller model, where the Hamming distance measurements are clean, noise-free, precise and separated for each component.

The simulator creates traces similar to those of a real microcontroller in which a small resistor is placed in series with the power or ground inputs. Naturally the data extracted from the microcontroller, usually recorded by a digital oscilloscope that captures the voltage drop across the resistor, must be cleaned through the application of low-pass filters, high-pass filters or statistical functions to be comparable to the ideal values.

A *Trace* in this context is a set of power consumption measurement.

This traces are the base of a class of side channel attacks: *Power Attacks*, first introduced in 1999 by Paul Kocher in the work Differential Power Analysis [2].

Side-channel power attacks work because circuits executing cryptographic operations are eventually implemented using logic gates composed of transistors. In a transistor the current flow is dependent on the charge applied or removed from it. The variation of charge determines a power consumption that can be observed with laboratory equipment and that leaks internal information of the executing algorithm, therefore creating a potential side channel resource.

Two energy consumption dependencies are typically exploited: data dependency and operation dependency.

Based on the findings of references [3], [4] and [5], buses are commonly recognized as the most power-intensive components within a microcontroller. The modifications made to the simulator involve the calculation of the Hamming Distance on input or output ports immediately after the signal propagates through the buses. These propagations represent the main sources of power consumption in the microcontroller. As a result, the data extracted from my modifications provides crucial information for side-channel attacks, based on the assumption that power consumption is directly proportional to the number of changed data values in the processed data.

In his work Kocher [2] described two Power Analysis techniques:

- Simple Power Analysis (*SPA*), that consists of directly interpreting power consumption measurements during the execution of a microarchitecture program. This method can reveal the sequence of instructions executed, information that is really useful in all the algorithms in which the execution path depends on the data being processed, such as the *Data Encryption Standard* subkeys bits shifting. SPA attacks rely on the condition that the secret key must have a significant impact on the power consumption.

- Differential Power Analysis (*DPA*) is a method that exploits the correlation between power consumption and the data being processed. It's essentially a statistical technique used to perform attacks on ciphers. DPA operates under the assumption that power consumption depends on the hamming distances of the cryptographic device components. Differential power analysis (DPA) attacks are widely favoured due to their accessibility, as attackers do not require intricate knowledge of the targeted device. However, these attacks rely heavily on mathematical methodologies and power consumption models to be effective.

For instance, consider the *Advance Encryption Standard* (AES), a widely used symmetric encryption algorithm. AES involves various operation such as *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*. Each of these layers affects the power consumption and consequently the hamming distance differently.

Focusing on the AddRoundKey (ARK) layer, that involves performing a bitwise XOR operation between the round key and the processed data, it can leak lots of useful information about the processed data and the key.

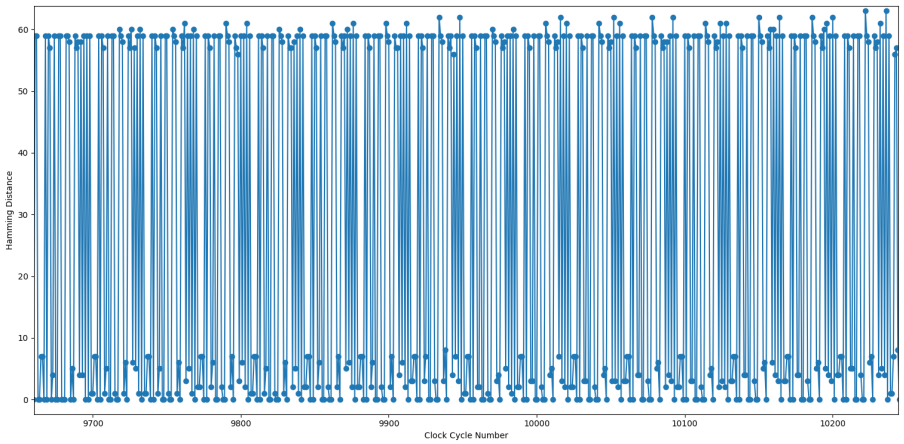
In the AES NIST standard the input, output and each key are processed as groups of eight bits. As each round starts with 128 bits of input, the block size is equal to 16 elements of 8 bits each. Following the NIST standard the ARK layer can be implemented as follows:

```
for (int i = 0; i < AES_SBOX_BLOCK_SIZE; ++i)
        state[i] ^= round_key[AES_SBOX_BLOCK_SIZE * round + i];
```

The lines of code consists of four main operations: calculating the required address to access the data, retrieving the data from the memory, performing the XOR operation and saving the new state in the memory.

By analyzing the hamming distance values obtained from the created script, I extracted the graph below which represents the hamming distances of the

ALU output during the execution of the 16 bitwise XOR operations of the Add Round Key layer.



It can be seen that there are 16 distinct clusters of operations with similar Hamming distance values, corresponding to the hamming distance of each Add Round Key bitwise XOR operation. The high hamming distance values in each cluster are due to the high values of the memory addresses from which to retrieve the round key and where to store the result of the bitwise operation.

This data provides insight into the underlying cryptographic processes and highlights the importance of the Hamming distance model in analyzing power consumption patterns in side-channel attacks.

9 Conclusion

This article describes how I modified an open source RISC-V simulator by adding the hamming distance calculation for each component simulated. The modification was made to create a tool for simulate side channel attacks. Those attacks expose vulnerabilities that traditional security measures may overlook, underscoring the critical need for robust hardware-level defense mechanisms. In the realm of smart cards, where secure operations are paramount, power analysis attacks emerge as a powerful tool to extract sensitive information by exploiting variations in power consumption during cryptographic operations.

The Python script I developed proves to be very useful for conducting power analysis attacks, especially differential power analysis (DPA). This automated tool allows you to run numerous Ripes simulator simulations in parallel, each with different source code constants, to generate a large set of power traces.

This efficient and automated approach greatly simplifies the collection of data necessary to carry out a DPA attack, making it more feasible to analyze and identify vulnerabilities in the cryptographic system.

References

[1] M. B. Petersen, *Ripes: A Visual Computer Architecture Simulator* 2021 ACM/IEEE Workshop on Computer Architecture Education (WCAE).

[2] P. Kocher, J. Jaffe and B. Jun, *Differential Power Analysis*. In: Advances in Cryptology - CRYPTO' 99.

[3] Z. Martinasek, V. Clupek and T. Krisztina, *General scheme of differential power analysis* 2013 36th International Conference on Telecommunications and Signal Processing (TSP).

[4] E. Brier, C. Clavier, F. Olivier, *Correlation Power Analysis with a Leakage Model* Cryptographic Hardware and Embedded Systems - CHES 2004.

[5] S. Mangard, E. Oswald, T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer-Verlag, NJ, USA 2007.