BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2026
http://excel.fit.vutbr.cz

Efficient Hashing Algorithms for High-Speed FPGA Applications

Ondřej Schwarz

Abstract

High-speed networking heavily relies on hashing algorithms for hardware lookup tables, packet inspection, and checksum calculations. While traditional algorithms like CRC and Toeplitz are fast and resource-efficient, they lack strong cryptographic security and collision resistance. The aim of this paper is to design and evaluate efficient, modern hashing architectures in VHDL that are optimized for direct FPGA deployment.

Four distinct hashing algorithms—SpookyHash, SipHash, Chaskey, and an experimental parallel function named PCARX—were implemented with a focus on maximum configurability. The designs utilize variable pipelining with support for deep pipelining, allowing registers to be inserted between individual logical operations. This methodology enables a precise, application-specific balance between maximum operating frequency, latency, and logic resource consumption.

The highly pipelined architectures achieved exceptional performance, with maximum frequencies (f_{max}) exceeding 700 MHz for most configurations, and reaching up to 1 GHz for specific Chaskey setups. SpookyHash delivered the best metrics for non-cryptographic use cases. SipHash and Chaskey provided robust security with efficient ARX structures, while the experimental PCARX demonstrated superior collision resistance and much smaller latency when processing longer keys then it's counterparts for the price of higher logic utilization. These scalable implementations provide hardware engineers with a comprehensive toolkit to select and configure the optimal hashing function based on strict design constraints. The detailed analysis of the throughput-latency-area trade-offs facilitates better architectural decisions in high-speed network security and data processing applications.

*xschwao00@vut.cz, *Faculty of Information Technology, Brno University of Technology*

In high-speed networking, hashing algorithms have many critical use cases, including hardware lookup tables, packet inspection, and checksum calculations. These algorithms must be deployed directly on hardware. Today, CRC and Toeplitz hashes are typically used. Although they are fast and highly resource-efficient, they lack strong security and resistance to collisions.

The goal of my work is to implement efficient, modern hashing algorithms in VHDL that can be deployed directly on an FPGA chip. My primary focus was maximum configurability—allowing adjustments, such as the number of compression rounds, so the algorithms can be easily adapted to specific use cases. I also implemented variable pipelining with support for deep pipelining, which means registers can be inserted between every logical operation. This approach allows developers to find the perfect balance between maximum frequency, latency, and resource consumption based on their specific application. Thanks to this variable pipelining, we can achieve extremely high frequencies: over 700 MHz for most settings, and up to 1 GHz for specific configurations of the Chaskey hash function.

For this project, I implemented four specific algorithms: SpookyHash, SipHash, Chaskey, and an experimental hash function I call PCARX.

Let's start with SpookyHash [1]. It is a blazing fast, lightweight algorithm generating a 128-bit hash. It is not cryptographic, however, and I implemented the short version of the algorithm, meaning it supports keys up to 191 bytes. The block diagram is shown in Figure 1. As you can see in Figures 2 and 3, it offers incredibly high throughput with relatively low latency and resource consumption compared to the cryptographic functions, at least for the shorter keys it sup-

ports. The statistical results are also highly satisfactory, marginally beating even the cryptographic SipHash. Figure 4 demonstrates a very healthy avalanche effect: when one bit of the message changes, the number of changed bits in the resulting hash converges perfectly around 64, which is exactly the ideal 50% mark for a 128-bit output.

Next is SipHash [2]. This is a highly popular cryptographic algorithm based on an ARX (Addition, Rotation, XOR) network. It is ideal for hardware implementation because it is lightweight and easily parallelizable. I implemented four variants in total:

- **Regular SipHash:** 64-bit state variables and a 64-bit output for general use.
- **Extended SipHash:** Generates a 128-bit hash for applications requiring higher collision resistance.
- **HalfSipHash:** Uses 32-bit state variables and generates a 32-bit hash. Ideal for short messages where severe resource constraints apply.
- **Extended HalfSipHash:** Combines the small hardware footprint of HalfSipHash with a more collision-resistant 64-bit output. However, it should only be used for small keys, as it processes data in 4-byte blocks instead of 8-byte blocks.

Figure 5 shows the block diagram of the regular SipHash; the extended versions simply add extra constants and finalization rounds. Figures 6 and 7 show that SipHash is roughly twice as resource-intensive as SpookyHash (these results are for the highly secure SipHash-4-8 variant with 4 compression and 8 finalization rounds). Statistical results are slightly lower than SpookyHash but still robust, and the avalanche effect, shown in Figure 8, remains excellent.

The third implemented algorithm is Chaskey [3], a lightweight cryptographic MAC also based on an ARX network. Designed to be highly efficient, it uses 32-bit state variables, making its ARX network very similar to HalfSipHash. However, unlike SipHash, it utilizes its entire state to generate the final hash, producing a full 128-bit output. In theory, this makes it a cheaper alternative for generating a 128-bit cryptographic hash compared to the extended SipHash. Figure 9 outlines its architecture. Figures 10 and 11 compare throughput, latency, and logic utilization across different pipeline configurations. The benefits and downfalls of 32-bit ARX are clear to see—for shorter keys, Chaskey performs better then SipHash in both metrics, however, the longer the key, the smaller the gap gets. The statistical metrics for the Chaskey-LTS variant with 12 rounds are very strong. Collision resistance is similar to the previous algorithms, sitting only about 300 points away from the ideal mathematical value. Its information entropy, however, is the best among the algorithms discussed so far, and the avalanche effect performs flawlessly.

The final hash function is an experimental design I named PCARX. It is inspired by the PCASD [4] function developed by Chinese researchers a few years ago, but introduces key architectural changes: it reduces the block width from 512 to 256 bits and replaces the SHA-2 style compression with a hardware-friendly ARX structure. It relies on a parallel hash tree and cellular automata, both of which are exceptionally well-suited for FPGAs. The general structure is depicted in Figure 13. Figures 14 and 15 clearly demonstrate the trade-off that parallel hash trees offer compared to sequential hash functions—the hardware resource consumption is much greater and the throughput is lower because of the lower operating frequencies compared to the functions discussed previously, but the latency is much more stable, as it increases only when the tree gains another level. It is then clear that the latency of sequential hash functions will grow linearly, while in the case of PCARX it will increase stepwise, and the gap will widen the longer the key is, in favor of the parallel structure. Statistically, the PCARX-8-8 variant is outstanding. It surpasses all previously mentioned functions in both collision resistance and information entropy, while maintaining ideal avalanche performance as indicated by Figure 16.

To conclude: SpookyHash is the ultimate choice for applications where cryptographic strength is not required. If security is a concern, either a variant of SipHash or Chaskey should be deployed, depending on the key width and target hash size. Finally, PCARX is at its best in scenarios involving larger keys, where minimizing processing latency takes precedence over logic resource consumption.

References

[1] Bob Jenkins. SpookyHash: a 128-bit noncryptographic hash. https://burtleburtle.net/bob/hash/spooky.html, 2012. Accessed: 2026-04-19.

[2] Jean-Philippe Aumasson and Daniel J. Bernstein. SipHash: a fast short-input prf. Cryptology ePrint Archive, Paper 2012/351, 2012.

[3] Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. Chaskey: An efficient MAC algorithm for 32-bit microcontrollers. Cryptology ePrint Archive, Paper 2014/386, 2014.

[4] Yijun Yang, Huan Wan, Xiaohu Yan, and Ming Zhao. Parallel hash algorithm based on cellular automata and stochastic diffusion model. ResearchGate Preprint, 2024.