Hardware Implementation of the ChaCha20 Stream Cipher in Verilog

A Parameterized, Reproducible, and Open Hardware Core

Abhilash M.
Kalasalingam Academy of Research and Education
OpenSiliconHub

December 2025

Abstract

ChaCha20 is a widely used ARX (Add–Rotate–XOR) stream cipher standardized in RFC 8439. This paper presents a compact, fully reproducible, and parameterized Verilog hardware implementation supporting ChaCha8, ChaCha12 and ChaCha20. The implementation uses an iterative datapath controlled by a finite-state machine (FSM) to trade area for throughput and to provide a compact, synthesizable RTL suitable for FPGA and prototyping ASIC flows. Validation is performed using RFC official test vectors and full scripts are provided for reproducibility. This version omits DOI metadata; DOI will be added after archival on a long-term repository (Zenodo) when you trigger the release.

1 Introduction

Cryptographic primitives implemented in hardware bring performance, robustness, and reduced exposure of secret material compared to software implementations. ChaCha20's ARX structure (Add–Rotate–XOR) is particularly amenable to hardware due to its simple combinational operations and the absence of S-boxes.

This work presents a synthesizable, parameterized Verilog implementation of the ChaCha block function (keystream block generator). The goal is a small-area, well-documented, reproducible core that can be used in embedded/IoT devices or as a research baseline.

Contributions.

- A compact iterative ChaCha hardware core parameterized by number of double rounds (ChaCha8/12/20).
- Full verification against RFC 8439 test vectors with reproducible scripts.
- Performance characterization on the Lattice iCE40 flow (Yosys/nextpnr).
- Detailed documentation, diagrams, and an appendix with build commands.

2 Background and Related Work

ChaCha was introduced by Bernstein [1] as an ARX variant of Salsa20. Implementations range from small iterative cores (area-efficient) to fully unrolled/pipelined designs (throughput-optimized). Notable open efforts include academic FPGA cores and larger projects (OpenTitan, CrypTech) — this work aims to be a compact, reproducible baseline designed for easy archival and citation.

3 Algorithm Overview

ChaCha operates on a 4×4 matrix of 32-bit words (512-bit state). The quarterround applies four ARX operations that mix four words; a double-round combines a column round and a diagonal round.

Figure 1 shows the word layout.

w_{0*4+0}	w_{0*4+1}	w_{0*4+2}	w_{0*4+3}
w_{1*4+0}	w_{1*4+1}	w_{1*4+2}	w_{1*4+3}
w_{2*4+0}	w_{2*4+1}	w_{2*4+2}	w_{2*4+3}
w_{3*4+0}	w_{3*4+1}	w_{3*4+2}	w_{3*4+3}

Figure 1: ChaCha state matrix indexed by word.

After the configured number of double-rounds, the working state is added back to the original state and serialized (little-endian) to the 512-bit output block.

4 Hardware Architecture

The RTL is split into:

- **State register file:** 16 × 32-bit registers.
- **Quarterround unit:** combinational ARX engine (adds, XORs, barrel rotates).
- **FSM controller:** sequences column and diagonal quarterrounds and tracks iterations.
- **Serializer:** outputs 512-bit block in little-endian order.

4.1 Top-level

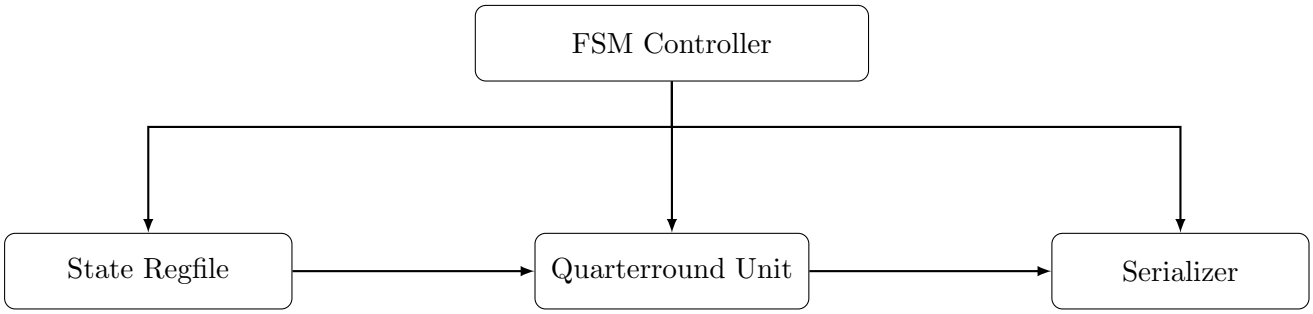

Figure 2: Top-level block diagram of the ChaCha20 hardware core.

4.2 Quarterround detail

The quarterround implements the ARX sequence; a typical pseudocode is:

```
a += b; d ^= a; d = (d <<< 16);
c += d; b ^= c; b = (b <<< 12);
a += b; d ^= a; d = (d <<< 8);
c += d; b ^= c; b = (b <<< 7);
```

All arithmetic is 32-bit modulo 2^{32}. Rotations are performed by barrel shifters for a fixed rotation amount.

4.3 Controller (FSM)

States: `IDLE`, `LOAD`, `ROUND_LOOP`, `DOUBLE_ROUND_STEP`, `FINAL_ADD`, `SERIALIZE`, `DONE`. The FSM ensures the correct sequence per RFC 8439 (columns then diagonals) and counts double-rounds via a parameterizable counter.

5 Design Rationale and Trade-offs

We chose an iterative datapath to maximize reusability and minimize area. Unrolled designs provide lower latency (higher throughput) at much larger area; pipelining offers throughput improvements but complicates verification and increases register pressure.

For a student / academic baseline and small-FPGA targets, the iterative implementation reduces verification surface and eases reproducibility.

6 Verification

Functional verification uses a self-checking Verilog testbench that runs the RFC 8439 official test vectors (sections cited in the repository) and asserts equality on the produced 512-bit blocks. The testbench can produce VCD waveforms for visual inspection.

Example:

```
iverilog -g2012 -o tb.out chacha20.v chacha20_tb.v
vvp tb.out
```

The repository contains the full testbench sources and sample VCD outputs.

7 Performance Evaluation

Synthesis was performed using Yosys with the `synth_ice40` backend targeting iCE40. Table 1 summarizes area and latency. Throughput numbers use a conservative Fmax of 100 MHz for reporting; please run post-route timing for exact device-specific Fmax.

Table 1: ChaCha performance on iCE40 (Yosys/nextpnr flow)

Variant	LUT4	FFs	Carry	Latency (cycles)	Throughput
ChaCha8	2949	1928	1493	9	5.69 Gbps
ChaCha12	2949	1928	1493	11	4.65 Gbps
ChaCha20	2949	1928	1493	15	3.41 Gbps

Notes: numeric values depend on synthesis options and device/package.

8 Limitations and Future Work

Limitations:

- No side-channel countermeasures (masking/hiding) in this release.
- Iterative design exhibits increased latency vs unrolled designs.

- ASIC post-layout PPA data not included yet.

Planned future work:

- Masked/hardened versions and SCA evaluation.
- Optional unrolled/pipelined variants geared for high-throughput platforms.
- End-to-end ChaCha20-Poly1305 AEAD module and system-level reference designs.
- ASIC flow results (synthesis, place-and-route, power estimates).

9 Release and Reproducibility

The repository will include:

- `SRC/Chacha20/chacha20.v` — RTL
- `tests/*` — testbenches RFC test vectors

10 Conclusion

We present a compact, parameterized, and documented ChaCha hardware core with reproducible artifacts aimed at students, researchers, and embedded developers. By releasing code, verification, and synthesis scripts, we provide a baseline core that can be extended and cited.

Acknowledgements

OpenSiliconHub contributors and the open-source toolchain projects (Yosys, nextpnr, Icarus Verilog) for enabling accessible hardware development.

References

[1] D. J. Bernstein, "ChaCha, a variant of Salsa20," 2008.

[2] RFC 8439, "ChaCha20 and Poly1305 for IETF Protocols," 2018.