

CRYPTOGRAPHY ON RISC-V SYSTEMS:
ASCON ALGORITHM ON CV32E40X CORE
CV-X-IF BASED ISE USING PYNQ-Z1 SOC FPGA



ALI RAZA
09-242242-001

BAHRIA UNIVERSITY ISLAMABAD

CRYPTOGRAPHY ON RISC-V SYSTEMS:
ASCON ALGORITHM ON CV32E40X CORE
CV-X-IF BASED ISE USING PYNQ-Z1 SOC FPGA



ALI RAZA
09-242242-001

A thesis submitted in fulfilment of the
requirements for the award of the degree of
Master of Science (Computer Engineering)

Department of Computer Engineering

BAHRIA UNIVERSITY ISLAMABAD

AUGUST 2026

MS-13 Thesis Completion Certificate

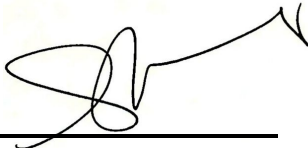
Student Name: **Ali Raza** Registration Number: **97825**

Program of Study: **Master of Science (Computer Engineering)**

Thesis Title: **Cryptography on RISC-V Systems: ASCON Algorithm on CV32E40X
Core CV-X-IF Based ISE Using PYNQ-Z1 SoC FPGA**

It is to certify that the above student's thesis has been completed to my satisfaction and, to my belief, its standard is appropriate for submission for evaluation. I have also conducted plagiarism test of this thesis using HEC prescribed software and found similarity index at **3%** that is within the permissible set by the HEC. for MS/MPhil/PhD.

I have also found the thesis in a format recognized by the BU for MS/MPhil/PhD thesis.

Principle Supervisor's Signature: 

Principle Supervisor's Name: **Dr. Syed Khawar Hussain Shah**

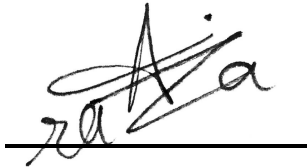
August 20, 2026

MS-14A

Author's Declaration

I, **Ali Raza** hereby state that my MS thesis titled “**Cryptography on RISC-V Systems: ASCON Algorithm on CV32E40X Core CV-X-IF Based ISE Using PYNQ-Z1 SoC FPGA**” is my own work and has not been submitted previously by me for taking any degree from “**Bahria University, Islamabad**” or anywhere else in the country / world.

At any time if my statement is found to be incorrect even after my Graduate the university has the right to withdraw cancel my MS degree.

A handwritten signature in black ink, appearing to read 'Ali Raza', is written over a solid black horizontal line.

Ali Raza

09-242242-001

August 20, 2026

MS-14B Plagiarism Undertaking

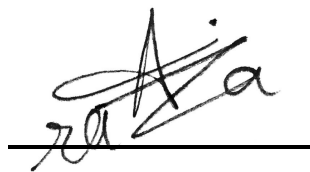
I, **Ali Raza** solemnly declare that research work presented in the thesis titled

**Cryptography on RISC-V Systems: ASCON Algorithm on CV32E40X Core
CV-X-IF Based ISE Using PYNQ-Z1 SoC FPGA**

is solely my research work with no significant contribution from any other person. Small contribution / help whenever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of Bahria University and the Higher Education Commission of Pakistan towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarised and any material used is properly referred / cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS degree, the university reserves the right to withdraw / revoke my MS degree and HEC and the university has the right to publish my name on HEC / University Website on which name of students who submitted plagiarised thesis are placed.

A handwritten signature in black ink, appearing to read 'Ali Raza', is written over a horizontal line.

Ali Raza

09-242242-001

August 20, 2026

DEDICATION

This thesis is dedicated to my parents for their unwavering love and prayers, my siblings for their constant support and encouragement, and my friends for their companionship, understanding, and motivation throughout this journey.

ACKNOWLEDGMENTS

I thank Almighty Allah for all His blessings and providing me strength and potential to complete my thesis.

I would like to pay gratitude to my supervisor Dr. Syed Khawar Hussain Shah for his continuous guidance, dedicated involvement and support throughout completing my thesis.

My sincere appreciation to all the Professors for their immense knowledge, without their teachings it wasn't possible to accomplish my work. My university fellows who always helped me whenever needed. I would like to thank my family for always supporting me.

ABSTRACT

In 2023 the National Institute of Standards and Technology (NIST) selected Ascon as the standard for lightweight cryptography (LWC), formalized as NIST SP 800-232. As Ascon becomes the basis for confidentiality, integrity and authentication on constrained devices, the cost of accelerating it on RISC-V is a pressing engineering question. RISC-V instruction set extension work has characterized Ascon authenticated encryption and hashing in detail, but three gaps remain: Ascon-MAC, Ascon-PRF and Ascon-PRFShort are largely unaddressed. Every comparable RISC-V work measures the CAESAR-era v1.2 specification rather than the standardized algorithms and none reports a single shared hardware permutation core measured on real silicon across the complete seven member family. This thesis therefore establishes what such an accelerator costs in performance, area and power on real hardware, and how its benefit is distributed across the seven members. A CV-X-IF tightly coupled Ascon-p accelerator was designed for a CV32E40X Embedded RISC-V core, synthesized and deployed on a Xilinx Zynq-7000 PYNQ-Z1 FPGA board. It occupies 597 LUTs and 368 flip-flops with no DSP or block RAM, and closes timing at 60 MHz with positive margin. The software baseline and the accelerated run execute on the same bitstream, differing only in one control register bit, and that baseline is hand-written RV32I assembly rather than the compiled C used in the comparable literature, making the ratios conservative. All 28 measurement points, spanning 0 to 1024 bytes and covering Ascon-AEAD128 under both associated-data conventions, passed NIST Known Answer Test verification in both modes. Speedups measured on hardware, not in simulation, range from 2.98x on the single call Ascon-PRFShort to 21.37x on Ascon-Hash256 at 1024 bytes. The area and timing are tool reports, power is a vectorless estimate, and no energy result is claimed. The benefit is not uniform. At a common 16 byte length the seven members span 3.41x to 11.12x from identical hardware, bounded not by the permutation they share but by the non-permutation work accompanying each call. To the author's knowledge, this is the first silicon measured characterization of a single shared Ascon accelerator serving the complete seven member SP 800-232 era family on a RISC-V core.

TABLE OF CONTENTS

DECLARATION	ii
DEDICATION	iv
ACKNOWLEDGMENTS	v
ABSTRACT	vi
TABLE OF CONTENTS	vii
LIST OF TABLES	xi
LIST OF FIGURES	xiii
LIST OF ABBREVIATIONS	xiv
1 INTRODUCTION	1
1.1 Background and Motivation	1
1.2 Overview of Ascon LWC	2
1.3 Problem Statement	4
1.4 Research Questions and Objectives	5
1.5 Contributions of This Thesis	6
1.6 Thesis Organization	8
2 LITERATURE REVIEW	9
2.1 Ascon as a Standard, and Version Gap	9
2.2 Software Baselines and Comparability	10
2.3 RISC-V ISEs: Coupling Style and Cost	11
2.4 Evaluation Grade and Coverage	14
2.5 Standalone Coprocessors	14
2.6 Adjacent Work: Security and Verification	16
2.7 Survey Literature and Metric Vocabulary	17
2.8 Research Gap and Positioning	17

3	METHODOLOGY	20
3.1	Overview and Research Approach	20
3.2	System Architecture	21
3.2.1	Target Platform and Block Design	21
3.2.2	Harvard Memory Architecture	22
3.2.3	SoC Integration and Performance Counter	23
3.3	The Ascon Permutation Accelerator	26
3.3.1	Design Rationale	26
3.3.2	The CV-X-IF Protocol	27
3.3.3	Instruction Routing	29
3.3.4	State Representation and FSM	30
3.3.5	The AXI-Lite Register File	35
3.4	Marshall et al. Compliance: Four Principles Verified	37
3.5	Firmware Implementation	38
3.5.1	The Bare-Metal Environment	38
3.5.2	The Permutation Assembly	40
3.5.3	Build System	41
3.6	Functional Verification Methodology	43
3.6.1	Verification Architecture	43
3.6.2	Unit and Integration Testbenches (Levels 1–3)	44
3.6.3	Full SoC Verification: Levels 4 to 7	44
3.7	FPGA Implementation and Vivado	47
3.7.1	The Zynq 7000 Block Design	47
3.7.2	Clock Configuration	48
3.7.3	Resource Utilisation and Bitstream Deployment	49
3.8	Performance Measurement Methodology	50
3.8.1	The Two Pass Measurement	50
3.8.2	Counter and KAT Verification	51

3.8.3	Derived Metrics and Traceability	51
3.9	Summary	53
4	RESULTS	54
4.1	Chapter Introduction	54
4.2	FPGA Implementation Results	56
4.3	Functional Verification Summary	57
4.4	Per-Family Results	58
4.4.1	Performance Metrics: Definitions	58
4.4.2	Ascon-AEAD128	59
4.4.3	Ascon-Hash256	61
4.4.4	Ascon-XOF128	62
4.4.5	Ascon-CXOF128	63
4.4.6	Ascon-MAC	63
4.4.7	Ascon-PRF	64
4.4.8	Ascon-PRFShort	64
4.5	Cross Family Analysis and Discussion	66
4.5.1	The Absorb-Fraction Argument	66
4.5.2	Why Hardware CPI Is Higher	67
4.5.3	CPB Comparison	67
4.5.4	PRFShort as a Special Case	67
4.5.5	CXOF128 as a Special Case	68
4.6	Comparison With Published Work	68
4.6.1	Direct Numeric Comparison	69
4.6.2	Context-Only Comparison	79
4.6.3	Survey and Methodology References	81
4.7	Chapter Summary	82
5	CONCLUSIONS	84
5.1	Summary of Findings	84

5.1.1	RQ1: PPA of CV-X-IF Accelerator	84
5.1.2	RQ2: How the Benefit	85
5.2	Contributions	86
5.3	Limitations and Constraints	88
5.4	Future Work	90
5.5	Concluding Remarks	91
REFERENCES		92
APPENDICES A – C		99

LIST OF TABLES

1.1	Ascon LWC Addressed in This Thesis	3
2.1	Comparability Across Closest Related Work	18
3.1	AXI Address Map	22
3.2	Module Instantiations <code>cv32e40x_system.sv</code>	24
3.3	Performance Counter and Boundary Conditions	26
3.4	CV-X-IF Six-Channel Protocol	28
3.5	CV-X-IF Channel Gating Summary	29
3.6	State Representation Correspondence	31
3.7	FSM State Summary	34
3.8	Nine-Step Measurement Sequence	37
3.9	Marshall et al. Compliance: Four Principles	38
3.10	Seven Level Verification Campaign	43
3.11	Verification Coverage by Length	46
4.1	Provenance of Reported Quantities	55
4.2	Fixed Configuration for All Measurements	55
4.3	ModelSim and FPGA CC at MSG=16B	57
4.4	Metric Definitions	58
4.5	AEAD128 Results: Cycle Counts	59
4.6	AEAD128 Results: Derived Metrics	59
4.7	AD Effect at Matched Plaintext Length	60
4.8	Hash256 Results: Cycle Counts	61
4.9	Hash256 Results: Derived Metrics	62
4.10	XOF128 Results: Cycle Counts	62

4.11	XOF128 Results: Derived Metrics	62
4.12	CXOF128 Results: Cycle Counts	63
4.13	CXOF128 Results: Derived Metrics	63
4.14	MAC Results: Cycle Counts	64
4.15	MAC Results: Derived Metrics	64
4.16	PRF Results: Cycle Counts	65
4.17	PRF Results: Derived Metrics	65
4.18	PRFShort Results: Cycle Counts	65
4.19	PRFShort Results: Derived Metrics	65
4.20	Absorb Fraction and Speedup	67
4.21	Direct Comparison with Bader (2024)	70
4.22	Comparison with Directly related Works	77
4.23	Survey and Methodology References	81

LIST OF FIGURES

1.1	ASCON-AEAD128 Algorithm	3
3.1	PYNQ Z1 FPGA [1]	22
3.2	CV32E40X Core from OpenHW	27
3.3	Decision Flow Chart	33
3.4	ASCON-AEAD128 SPN	35
3.5	Vivado Timing Constraints	49
4.1	This Thesis Vivado On Chip Power	71
4.2	Bader (2024) [2] Vivado On Chip Power	71

LIST OF ABBREVIATIONS

AEAD	Authenticated Encryption with Associated Data
AXI	Advanced eXtensible Interface
CPB	Cycles Per Byte
CPI	Cycles Per Instruction
CV-X-IF	CORE-V eXtension Interface
CXOF	Customized Extendable Output Function
DMA	Direct Memory Access
DSP	Digital Signal Processor
FSM	Finite State Machine
IoT	Internet of Things
ISA	Instruction Set Architecture
KAT	Known Answer Test
LWC	Lightweight Cryptography
MAC	Message Authentication Code
MMIO	Memory-Mapped Input/Output
NIST	National Institute of Standards and Technology
OBI	Open Bus Interface
RTL	Register Transfer Level
SVA	SystemVerilog Assertion
XOF	Extendable Output Function

LIST OF APPENDICES

A	Instruction Set, Registers and Source Listings	99
B	Verification and Benchmark Data	106
C	FPGA Implementation and Reproduction	112

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

The proliferation of Internet of Things (IoT) devices, wireless sensor networks and other resource constrained embedded systems has created a growing demand for cryptographic protection of data generated, stored and transmitted at the edge. These devices operate under tight constraints on silicon area, energy budget and memory, which makes conventional algorithms such as AES-GCM [3, 4] costly to implement efficiently. That gap motivated the field of lightweight cryptography, which seeks algorithms offering adequate security at a far lower implementation cost in both hardware and software.

In response, the National Institute of Standards and Technology (NIST) ran a multi-year Lightweight Cryptography standardization process, selecting in 2023 the Ascon family, designed by Dobraunig, Eichlseder, Mendel and Schl  ffer [5]. The selection was formalized in NIST Special Publication 800-232, published in 2025, which specifies four Ascon-based algorithms covering authenticated encryption, hashing and extendable output functions [6]. A companion specification adds dedicated message authentication and pseudorandom function modes [7]. Because one family is now expected to serve confidentiality, integrity and authentication across a wide range of constrained devices, how efficiently that family executes on real embedded processors is a pressing engineering question.

RISC-V, an open and modular instruction set architecture, is a natural

target for that question, since its extensibility allows cryptographic acceleration to be added without departing from a standard, toolchain-supported core. Attaching an accelerator to such a core can close much of the gap between pure software execution and a fully custom coprocessor while preserving portability. This thesis addresses exactly that problem: accelerating the Ascon family on a RISC-V core, and establishing what the acceleration costs and delivers when it is actually built and measured on hardware rather than estimated.

1.2 Overview of Ascon LWC

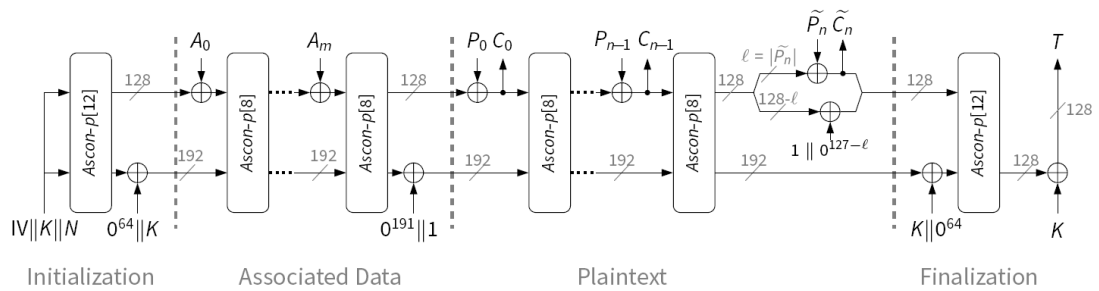
Ascon is not a single algorithm but a family of constructions built around one cryptographic permutation, *Ascon-p*, which operates on a 320-bit state organized as five 64-bit words [5]. That state size is a deliberate design choice: on a 32-bit processor such as the CV32E40X [8] each 64-bit word fits in a pair of registers, so the complete state occupies exactly ten general purpose registers. This property, revisited in Chapter 3, is part of what makes Ascon suited to tightly coupled, register file based acceleration rather than memory mapped coprocessor designs.

Every member reuses that same permutation, run for a chosen number of rounds, inside a sponge or duplex construction that absorbs input and, where required, squeezes out output. Varying the round count and the absorption pattern produces different functional members without changing the underlying primitive. NIST SP 800-232 specifies four members [6] and a companion specification defines three more [7] together these seven, listed in Table 1.1, form the complete scope of this thesis.

Because all seven share one permutation, a single hardware implementation of *Ascon-p* can in principle serve the whole family rather than requiring one accelerator per algorithm, which is one of the two questions this thesis investigates. Ascon's practical appeal for constrained deployment comes from three further properties. It uses only bitwise Boolean operations and

Table 1.1: Ascon LWC Addressed in This Thesis

Algorithm	Type	Functionality
Ascon-AEAD128	Authenticated Encryption	Confidentiality and integrity; 128-bit key, nonce and tag.
Ascon-Hash256	Hash Function	Fixed 256-bit digest.
Ascon-XOF128	Extendable Output Function	Arbitrary length digest.
Ascon-CXOF128	Customized XOF	XOF with a customization string for domain separation.
Ascon-MAC	Message Authentication Code	Authenticity without encryption; fixed 128-bit tag.
Ascon-PRF	Pseudorandom Function	Arbitrary length keyed pseudorandom output.
Ascon-PRFShort	Short Input PRF	Optimized for inputs and outputs up to 128 bits.

**Figure 1.1:** ASCON-AEAD128 Algorithm

rotations, needing no lookup tables, multiplication or division hardware. It requires only the forward direction of the permutation, since no mode needs an inverse and it processes data in a single pass without buffering a whole message. These are why it is being adopted for IoT sensor authentication, constrained network security protocols and embedded systems where AES-based [3, 4] alternatives are too costly.

1.3 Problem Statement

Ascon-AEAD128, the authenticated encryption member of the family, has already received considerable attention in the hardware and RISC-V integration literature. Closely related work, most notably Steinegger and Primas [9] on the RI5CY core and Bader [2] on the CV32E40X core [8] via the CV-X-IF coprocessor interface, has characterized the performance of instruction set extension accelerators for Ascon’s authenticated encryption and hashing modes. However, the three companion specification members now forming part of the Ascon family. Ascon-MAC, Ascon-PRF, and Ascon-PRFShort are largely absent from this RISC-V acceleration literature, even though they share the same underlying permutation and are expected to see comparable deployment in authentication only use cases such as pointer authentication and challenge response protocols. In addition, no published work known to the author reports a single shared hardware permutation core measured, on real FPGA silicon rather than in simulation, across all seven family members at matched message lengths, leaving open the question of whether one hardware core can serve the complete family efficiently or whether family specific accelerators are required in practice. This thesis addresses both gaps. It evaluates the performance, area, and power cost of a CV-X-IF tightly coupled Ascon accelerator on a real RISC-V SoC, and it characterizes how the benefit of that single accelerator is distributed across all seven Ascon family members.

1.4 Research Questions and Objectives

This thesis is guided by two research questions, addressed respectively in the two halves of Chapter 4.

RQ1: What are the performance, area, and power costs of a CV-X-IF tightly coupled Ascon permutation accelerator when implemented on FPGA and driven by RISC-V firmware?

To answer RQ1, the thesis pursues the following objectives: design and implement an *Ascon-p* permutation accelerator coupled to the CV32E40X [8] RISC-V Embedded core through the CV-X-IF coprocessor extension interface; synthesize and place-and-route the design for a Xilinx Zynq-7000 SoC [10] and report its resource utilization, timing closure, and power consumption; and measure its cycle accurate performance directly on FPGA hardware rather than through only ModelSim simulation estimates.

RQ2: How is the benefit of a single shared permutation accelerator distributed across the seven members of the Ascon family, and what determines that distribution?

RQ2 is deliberately not posed as a yes-or-no question about feasibility. Because all seven members are built on the same permutation, a shared accelerator is expected to serve them all, and demonstrating that alone would establish little. The open question is quantitative. Whether accelerating the shared primitive delivers a shared benefit, and if not, what governs the difference. To answer it, the thesis pursues the following objectives: extend the single accelerator to serve all seven family members through per family firmware drivers on one unmodified bitstream; verify the functional correctness of every family member against NIST Known Answer Test (KAT) [11] vectors at every measurement point; and measure cycle counts, cycles per instruction and speedup across family members, message lengths and, for Ascon-AEAD128, both associated-data conventions, isolating what bounds the benefit each member obtains.

1.5 Contributions of This Thesis

This thesis makes five contributions. The ASCON permutation accelerator itself is not among them. CV-X-IF and register-file coupled Ascon accelerators already exist, in Steinegger and Primas [9], Bader [2] and Dolmeta et al. [12], and the *Ascon-p* round function is fixed by the standard, so every correct implementation of it is identical. What is new is what was measured, how it was measured, and what the measurements show.

First, coverage of the standardised family on a general purpose core. Ascon-MAC, Ascon-PRF and Ascon-PRFShort are implemented and measured on a RISC-V core for the first time. Dolmeta et al. [12] name them but their evaluation reports only authenticated encryption, hashing and extendable output. The measurements also target NIST SP 800-232 rather than the CAESAR-era v1.2 specification that every comparable RISC-V work implements, a difference visible in those works' own artefact names and material to their results, since Ascon-128 absorbs at a 64-bit rate under p^6 where Ascon-AEAD128 absorbs at 128 bits under p^8 . Munir et al. [13] alone implement the standard, and do so with no host processor. The result is the first evaluation coupling the standardised Ascon family to a general purpose processor through instruction issue.

Second, a single-variable measurement methodology. The software baseline and the accelerated run execute on the *same* placed and routed bitstream, at the same clock, through the same memory subsystem, with byte-identical data memory contents, differing only in which permutation assembly is linked and one control-register bit that gates the CV-X-IF channels. Every comparable work measures its baseline on different hardware. Steinegger and Primas against an unmodified 45.6 kGE core rather than their extended 50.3 kGE one, and Cheng et al. [14] across three separately synthesized configurations of 3,303, 3,764 and 4,234 LUTs. Those comparisons vary two things at once. This one varies a single bit, and the measurements confirm the isolation, with stall counts identical between the two modes at 24 of the 28 points. Supporting this, a hardware counter-freeze

mechanism captures the cycle count at the exact cycle the firmware signals completion, making the measurement independent of host polling latency. In simulation, where the comparable literature obtains its cycle counts, such a mechanism is unnecessary, but on deployed silicon it has to be built.

Third, the accelerator organisation that makes one core serve seven algorithms. The round count is carried in the instruction encoding rather than fixed in hardware, indexing a shared round-constant table, so p^6 , p^8 and p^{12} execute on one datapath with no variant multiplexer and no re-synthesis, and a further depth would require only a firmware constant. The 320-bit state is held in the datapath’s own flip-flops and transits the host register file, with no storage dedicated to it. Both of the closest designs choose otherwise. Dolmeta et al. [12] carry a dedicated 320-bit register file and a separate interface controller, and Nguyen et al. [15] build around a dedicated 320-bit register. The measured consequence, against the nearest comparable design at the same one-round-per-cycle design point on a sibling core through the same interface, is 597 LUTs and 368 flip-flops against 1,321 LUTs and 810 registers.

Fourth, the finding that acceleration benefit is bounded by non-permutation work. Accelerating a shared primitive does not deliver a shared benefit. At a common message length the seven members range from 3.41x to 11.12x, and the ordering by speedup inverts against the ordering by absolute cost. Ascon-MAC runs at 5.5 cycles per byte against Ascon-Hash256’s 8.8 at 1024 bytes while showing less than half the speedup. What governs the difference is how much work accompanies each permutation call rather than the permutation itself. Two independent measurements establish this. In Ascon-AEAD128 an associated-data block, which is a load, an exclusive-or and a permutation, is accelerated by more than 10x, whereas a plaintext block, which must also store ciphertext, is accelerated by 3.92x. In Ascon-PRFShort, whose permutation call count is fixed by construction, the software and hardware cycle counts move by the same absolute amount at every input length, identifying the entire length dependence as work the

accelerator does not touch. Together these bound what any accelerator of this kind can deliver and explain every per-member profile reported here.

Fifth, reusable artefacts and a conservative baseline. Thirteen hand-written RV32I assembly implementations are original to this work. Six software baselines for family members that the reference distribution provides only in C, and seven accelerator-coupled cores. Steinegger and Primas identify precisely this absence as their reason for benchmarking against compiled C [9]. Closing it makes the baseline used throughout this thesis tighter than any in the comparable literature, which is why the speedups reported here are conservative rather than favourable, and no ordering against published figures is claimed.

1.6 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 reviews the published work on Ascon hardware acceleration, covering RISC-V instruction set extensions, standalone coprocessors and the survey literature that establishes the performance metrics used here, and identifies the gap this thesis addresses. Chapter 3 documents the methodology: the decision to use a CV-X-IF instruction set extension rather than a memory-mapped or DMA-coupled accelerator, the hardware design, the firmware and build system, the verification campaign, and the FPGA implementation and measurement protocol. Chapter 4 presents the results for all seven family members, distinguishing measured performance from tool-reported area and timing and from estimated power, analyses the cross-family differences and compares them against the most directly relevant published work. Chapter 5 closes the thesis by answering the two research questions, restating the contributions as established results, discussing the study’s limitations and proposing directions for future work.

CHAPTER 2

LITERATURE REVIEW

This chapter reviews the published work on hardware acceleration of the Ascon family on RISC-V processors. It is organised around a problem this literature does not itself confront. Its headline figures are not comparable with one another. Sections 2.1 to 2.4 take the four axes responsible in turn, treating individual works as evidence within each rather than reviewing them one by one. Sections 2.5 to 2.7 then cover standalone coprocessors, adjacent physical-security work, and the survey literature, before Section 2.8 states the gap that follows.

2.1 Ascon as a Standard, and Version Gap

Ascon originated as a CAESAR submission, progressing through v1 (2014), v1.1 (2015) and v1.2 (2016) under Dobraunig, Eichlseder, Mendel and Schl  ffer [16]. CAESAR recommended Ascon-128 and Ascon-128a for lightweight applications. The same v1.2 specification entered NIST’s Lightweight Cryptography process in 2019 and was published in the Journal of Cryptology [5]. NIST announced Ascon’s selection in February 2023 and published SP 800-232 in August 2025 [6].

The move from v1.2 to the standard was not a renaming, and the distinction is load-bearing for every comparison in this thesis. SP 800-232 switches byte ordering to little-endian, benefiting microcontrollers such as the cores used here. Updates the initial value format, adds truncation and nonce masking and consolidates Ascon-128, Ascon-128a and Ascon-80pq

into a single Ascon-AEAD128 built on the Ascon-128a construction [6]. That consolidation changes what a benchmark measures. Ascon-128 absorbs at a 64-bit rate using p^6 for data, Ascon-AEAD128 at a 128-bit rate using p^8 , so two implementations benchmarked at the same message length issue different numbers of permutation calls at different round depths.

This matters because almost the entire RISC-V acceleration literature below measures v1.2, as the authors’ own artefact names show: `ascon128v12` and `asconhashv12` in Dolmeta et al. [12], `ascon128v1.2` in Bader’s [2] comparison against Cheng et al. [14]. Figures reported for v1.2 are therefore not transferable to the standard, and this thesis does not treat them as though they were. Chapters 3 and 4 measure SP 800-232 throughout, the first respect in which their results are new rather than better.

2.2 Software Baselines and Comparability

A speedup is a property of a pair, the baseline and the accelerated implementation, not of the accelerator alone. Its usefulness as a comparison currency therefore depends on the baselines being alike, and here they are not.

Dobraunig et al. [7], the specification paper for Ascon-MAC, Ascon-PRF and Ascon-PRFShort, report software *cycles-per-byte* on ARM Cortex-M3 and Cortex-M4, establishing that metric as the field’s absolute currency precisely because it does not depend on a baseline. Campos et al. [17] find that compiled C trails hand-optimized assembly on RISC-V by a small but measurable margin.

That margin is not hypothetical, because every RISC-V work in Section 2.3 measures against compiled C. Steinegger and Primas [9] at -03 , Dolmeta et al. [12] at -02 , Moura [18] against C on an unmodified Ibex, Cheng et al. [14] against an unextended RV32GC baseline. Steinegger and Primas state why: they compare against C “mainly due to the lack of available RISC-V optimized implementations” [9]. Two things follow. The absence they identified in 2020 is still unfilled for six of the seven family

members, and closing it is what the assembly implementations of Chapter 3 do. More critically, their accelerated code is hand-written assembly while their baseline is compiled C, so their reported factors of roughly 50 and 80 vary coupling style and implementation language at once and cannot be decomposed into either.

The consequence is methodological rather than rhetorical because this thesis’s baseline is hand-optimized assembly, its ratios are not comparable with the published ones in either direction, so Chapter 4 claims no ranking by speedup and compares cycles-per-byte and per-call cost instead, both properties of a single implementation.

2.3 RISC-V ISEs: Coupling Style and Cost

The strongest predictor of performance across these works is not the permutation datapath, which is fixed by the specification and identical in every correct implementation, but how the accelerator is attached to the processor.

Steinegger and Primas [9] occupy the tightly coupled extreme and remain the foundational reference. They target RI5CY, the direct architectural predecessor of the CV32E40X [8], writing results straight into the register file with no handshake, at 4.2 kGE for the accelerator itself and roughly 2 cycles per byte asymptotically. Their own headline figure of 4.7 kGE adds the 0.5 kGE of register-file multiplexing and decode logic that integration into the core requires [9]. The cost is rarely stated. Direct coupling requires modifying the core’s register file and decode stage, forfeiting attachment to an unmodified, vendor-maintained core. Their figures are an upper bound on what coupling can deliver, not a target a portable design should expect to match.

Bader [2] is the most architecturally similar work, implementing an Ascon ISE on the CV32E40X [8] through the same CV-X-IF interface, evaluated on X-HEEP. Its most valuable contribution is not its speedup but its diagnosis of the interface. Having built both a true ISE and a CV-X-IF

coprocessor, Bader observes that with the latter “many cycles are wasted for loading and storing data”, that only one 32-bit word can be written back per clock cycle, and that “each time an instruction is dispatched, there is a handshake protocol to respect”, concluding that the effect is “a reduction of the speedup by a factor of 2” [2]. That independently identifies the overhead measured in Section 4.5.2, and is the closest any prior work comes to isolating the interface cost, though it stops at a ratio rather than a cycle figure.

Dolmeta et al. [12] make the same point in area. CHIMERA offers two CV-X-IF parametrizations on the CV32E40PX. A Complete Round version at 1,321 LUTs and 810 registers, dominated by a dedicated 320-bit state register file and a 408-LUT interface controller, and a Bitwise Rotation Unit version accelerating only the diffusion rotations at 573 LUTs and 109 registers. Their speedups across all reported algorithms, 2.91x to 6.04x against 1.41x to 1.56x, track that split closely. Restricted to authenticated encryption the Complete Round range is 2.98x to 3.22x. These two entries are also less independent than their separate citations imply. CHIMERA-CR is the published form of the design first documented in Bader’s [2] thesis. Both originate at Politecnico di Torino, both attach an Ascon coprocessor to an X-HEEP host through CV-X-IF, and both report the same coprocessor decomposition into an interface controller, a permutation datapath and a dedicated 320-bit state file with identical register counts at every level: 810 flip-flops for the wrapper, 123 for the interface controller, 687 for the datapath and 320 for the state file. The lookup-table counts differ by a single unit in two rows across the two synthesis runs, 1,322 against 1,321 for the wrapper and 409 against 408 for the interface controller. Both are retained in this review because they report different benchmark points, but they are one line of work rather than two independent confirmations of it. Read that way they still establish the substantive point, that within a fixed interface the decisive variables are where the state is stored and how often it crosses that interface, not how the round function is computed. What they

do not supply is corroboration by a second group. Neither quantifies the per-call cost in cycles.

Cheng et al. [14] evaluate one ISE for each of the ten LWC finalists on the Rocket RV32GC core. Ascon is reported at three message lengths, 16, 128 and 1024 bytes, so unlike the works above its scaling behaviour is visible. What the breadth costs is depth of analysis rather than coverage. Their results also need a decomposition the paper does not perform. The Ascon ISE raises the core from 3,764 to 4,234 LUTs, an increment of 470, but the step from the unextended 3,303-LUT core to that 3,764-LUT base is the generic $Zbkb/x$ bit-manipulation extension, which alone accounts for 1.89x of the reported 3.65x, slightly under half their headline figure is therefore not Ascon-specific. Their benchmark also sets associated data length equal to plaintext length throughout, following the NIST microcontroller benchmarking framework, a convention Steinegger and Primas do not share and which the two works never reconcile.

Moura [18] implements Ascon ISEs on Ibex in 28 nm alongside AES-128-CCM and ChaCha20-Poly1305, reporting a Power-Performance-Area framework. Two features matter beyond the headline numbers. Its elevated hardware CPI relative to software CPI, attributed to long-latency accelerator instructions, independently confirms a pattern this thesis explains in Section 4.5.2. And its energy gain differs from its performance gain, because the former derives from measured switching activity per mode rather than a static estimate scaled by cycle count, a distinction Section 4.6.3.1 returns to.

Gewehr, Luza and Moraes [19] accelerate Crystals-Kyber on Ibex, with one variant instantiating Kyber’s symmetric primitive using the Ascon permutation. Because their extension targets the surrounding polynomial arithmetic rather than the permutation, it supplies no data on Ascon acceleration and is cited only as evidence that the permutation is being adopted as a reusable primitive in adjacent post-quantum contexts.

2.4 Evaluation Grade and Coverage

Two further axes separate these works, and both are obscured when their figures are tabulated together. The first is the grade of evidence behind a cycle count. Bader's [2] performance figures come from QuestaSim simulation driven, in his own account, by a testbench performing a single test rather than by the Known Answer Test file, his FPGA work supplying utilisation and power but not cycle counts. Dolmeta et al. [12] implement on a Zynq UltraScale+ device at 50 MHz with binaries loaded over JTAG. Nguyen et al. [15] report from a fabricated 180 nm ASIC, the highest grade here. The remainder report synthesis estimates. A cycle count from one simulated vector and one from a deployed bitstream verified against the official vectors at every measurement point are not the same kind of claim, yet the literature presents them in identical units and the surveys of Section 2.7 aggregate them without distinction. Power reporting is weaker still and estimated by different methods, at different confidence levels, and seldom labelled as an estimate.

The second axis is family coverage, narrower than it first appears. Bader [2] covers authenticated encryption, hashing and the extendable-output function at two lengths. Cheng et al. [14] authenticated encryption alone, though at three lengths. Dolmeta et al. [12] name message authentication codes and pseudorandom functions in their abstract, yet their evaluation tables report only the v1.2 AEAD, hash and extendable-output variants. The distinction between naming a family member and implementing one is worth drawing precisely, since on abstracts alone the coverage gap looks smaller than it is. Munir et al. [13] do cover four SP 800-232 members, the closest scope match here, but as a standalone ASIC with no host.

2.5 Standalone Coprocessors

A second cluster implements Ascon as a standalone coprocessor or bus-attached peripheral. None couples through instruction issue, so none sup-

plies integration data, their figures serve as area and throughput context only.

Wei et al. [20] present RECO-HCON, a reconfigurable six-mode Ascon processor developed in Chisel, evaluated in 28/32 nm and also implemented in Skywater 130 nm through OpenLane, reaching over 667 MHz with a reported 29% power saving against state of the art lightweight cryptography hardware. Neither headline figure transfers here. A throughput an order of magnitude above the 60 MHz PYNQ-Z1 [1] operating point says nothing about relative efficiency at this operating point, and the design implements the v1.2 parameters, absorbing at a 64-bit rate, rather than the standardised Ascon-AEAD128, which absorbs at a 128-bit rate under p^8 .

Khairallah and Yadhunathan [21] study six ways of constructing message authentication codes from the Ascon permutation on a Xilinx Artix-7 FPGA, the same family as the PYNQ-Z1 [1]. Because no ISE work covers MAC or PRF hardware, this is the only published hardware reference for those members, being standalone, it establishes what the modes cost in isolation, not coupled to a core, which is the question this thesis asks.

Nguyen et al. [15] attach Ascon as a memory-mapped peripheral to a SERV core over a Wishbone bus, on FPGA and a fabricated 180 nm ASIC, reporting 1.75 Mbps per LUT and 417 Gbits per joule at 1.0 V. Theirs are actual on-chip measurements, the strongest evidence grade in this review, and they identify the same absence this thesis does. The lack of real-world on-chip evaluation of Ascon. Two qualifications bound what transfers. Their crypto core occupies 1,424 LUTs on FPGA and 17.4 kGE in 180 nm, and is built around a dedicated 320-bit state register, the same architectural choice Dolmeta et al. [12] make and the opposite of the approach taken here. And their measurements are taken at 32 kHz and 2 MHz on a bit-serial host over a bus, so neither their throughput nor their cycle counts are commensurable with an instruction coupled design on a pipelined core. It remains the closest analogue to the memory-mapped attachment that Section 3.3.1 considered and rejected. **Munir et al.** [13] explore round

based architectures performing one, two and four rounds per cycle against operation based variants with 64-bit and 32-bit datapaths, synthesized with a commercial 65 nm library. Alone among the works reviewed here they implement the standardised specification, covering Ascon-AEAD128, Hash256, XOOF128 and CXOOF128, but as a standalone design with no host processor.

The recurring pattern is a throughput against area trade-off, the highest raw throughput costing LUT or gate counts in the low thousands to tens of thousands against this thesis's 597 LUTs. That is context, not an efficiency claim in either direction. None of these designs must fit inside a coprocessor interface, and none carries a host's decode, issue and writeback logic in its accounting.

2.6 Adjacent Work: Security and Verification

A further body of work addresses Ascon's resistance to implementation level attacks, or the formal correctness of its hardware. Ramadass and Paramasivam [22] recover a 128-bit key from 460 traces of an unprotected FPGA implementation. You et al. [23] demonstrate low-trace-count template attacks on 32-bit software implementations. Das and Mazumdar [24] show that a single fault in the finalization round's substitution layer recovers a 128-bit key with between 63 and 451 chosen plaintexts. Saravanan et al. [25] verify control and data path correctness of an Ascon-128/128a implementation by assertion-based formal verification.

These works answer a different question than **RQ1** and **RQ2**, but their volume is informative. A community turns to hardening an implementation once it regards performance as settled, and this cluster's growth indicates the field considers authenticated encryption and hashing on RISC-V to be characterised. That cuts against the completeness of the literature rather than for it. The members left unmeasured have been passed over, not resolved. The accelerator presented here is itself unprotected, and these

attacks define the class of question a deployment ready successor must answer, as Section 5.3 acknowledges.

2.7 Survey Literature and Metric Vocabulary

Three surveys establish the metric vocabulary used across this field. Kaur et al. [26], in ACM Computing Surveys and describing itself as the first survey written against the finalized 2023 standard, summarizes FPGA and ASIC implementations across area, power, throughput, energy and efficiency. Konstantopoulou et al. [27] cover all ten LWC finalists and introduce Throughput/Area alongside Throughput/Power. Mirigaldi et al. [28] review implementation strategies alongside side channel countermeasures.

Their shared limitation is consequential here. All three survey standalone FPGA and ASIC implementations. None covers instruction set extension integration with a general purpose core. The metrics they consolidate therefore assume the accelerator *is* the system, so that dividing throughput by total area or power attributes the whole cost to the cryptographic function. That fails for a coupled design, where the host dominates both terms. The accelerator measured in Chapter 4 is 597 of the SoC's 6,200 LUTs, and the ARM processing system accounts for 1.520 W of a 1.723 W total. Applying Throughput/Area unmodified would either charge the accelerator for the host or, by excluding it, describe a system that cannot execute anything. Chapter 4 therefore reports cycles-per-byte and per-call cost as primary metrics, invoking Throughput/Area only against the standalone designs for which it was devised.

2.8 Research Gap and Positioning

The four axes converge. On specification, this literature measures v1.2 almost without exception, so its figures do not describe the algorithms NIST standardized. On baselines, every work measures against compiled C, and the one that states its reason names an absence, RV32 assembly, still unfilled

for six of the seven members. On coupling, Bader [2] and Dolmeta et al. [12] both identify the interface as the dominant cost, neither quantifying it in cycles. On evaluation grade and coverage, cycle counts from single vector simulation are tabulated beside synthesis estimates, and no work implements and measures Ascon-MAC, Ascon-PRF or Ascon-PRFShort on any RISC-V core. Table 2.1 summarizes the pattern. The columns most often empty are the argument. The Bader [2] and Dolmeta et al. [12] rows appear separately because they report different benchmark points and different evaluation bases, but for the purpose of counting independent prior designs they are one entry, not two.

Table 2.1: Comparability Across Closest Related Work

Work	Platform / Core	Interface	Baseline / Spec.	Family Coverage	Evaluation Basis
Steinegger & Primas (2020) [9]	RI5CY, ASIC	Direct reg-file; core modified	Compiled C -03 / v1.2	AEAD, Hash, XOF	Simulation / ASIC synthesis estimate
Cheng et al. (2023) [14]	Rocket RV32GC	In-pipeline FU (Zbkb/x + ISE)	Unextended RV32GC / v1.2	AEAD only; 16, 128, 1024 B	FPGA synthesis estimate
Bader (2024) [2]	CV32E40X (X-HEEP)	CV-X-IF (same interface)	Compiled C / v1.2	AEAD, Hash, XOF; 16, 32 B	QuestaSim, single test vector
Dolmeta et al., CHIMERA (2025) [12]	CV32E40PX (X-HEEP)	CV-X-IF	Compiled C -02 / v1.2	AEAD, Hash, XOF	FPGA implementation (Zynq Ultra-Scale+)
Munir et al. (2026) [13]	Standalone ASIC (65 nm), no host	Not applicable	None / SP 800-232	AEAD, Hash, XOF, CXOF	Synthesis estimate
This Work	CV32E40X on PYNQ-Z1, Zynq-7000	CV-X-IF	Hand-written RV32I assembly / SP 800-232	All 7 members	Deployed bitstream, KAT-verified at every point

This thesis is positioned against that pattern rather than against any figure within it. It implements the standardized specification rather than

v1.2. Measures against a hand-written assembly baseline, making its ratios conservative rather than favourable, quantifies the CV-X-IF per-call cost in cycles, converting the penalty Bader [2] identified as a ratio into a measured quantity. Reports every performance figure from a deployed bitstream verified against the official vectors at each measurement point.

CHAPTER 3

METHODOLOGY

3.1 Overview and Research Approach

This chapter describes the complete methodology employed to design, implement, verify, and evaluate a hardware accelerator for the ASCON family of lightweight cryptographic algorithms on an FPGA based RISC-V platform. The research follows the applied engineering paradigm for hardware accelerator development. A system design phase establishes the target architecture and justifies every major design decision. An implementation phase realizes the design in synthesizable RTL and in bare-metal firmware. A verification phase establishes functional correctness using NIST Known Answer Tests (KATs) [11]. An integration phase synthesizes and implements the design on target silicon, and a measurement phase collects quantitative performance data on the physical hardware.

The methodology is structured into **eight sections** and a closing summary, corresponding to the eight aspects of the complete research artefact.

- Section 3.1 introduces the platform and the overall research approach.
- Section 3.2 describes the SoC system architecture.
- Section 3.3 covers the ASCON permutation accelerator design and its justification against the Marshall et al. [29] ISE design requirements.
- Section 3.4 gives the RTL-level evidence that the accelerator satisfies the four applicable Marshall et al. [29] principles.
- Section 3.5 describes firmware implementation for seven ASCON family members.

- Section 3.6 documents the multi-level functional verification campaign.
- Section 3.7 describes the FPGA implementation and Vivado integration process.
- Section 3.8 specifies the performance measurement methodology that produced the quantitative results presented in Chapter 4.

3.2 System Architecture

3.2.1 Target Platform and Block Design

The target platform is the Digilent PYNQ-Z1 board [1], built around a Xilinx Zynq-7000 XC7Z020CLG400-1 SoC [10] that combines a dual-core ARM Cortex-A9 Processing System (PS) with 7-series Artix Programmable Logic (PL) on one die. The PS runs PetaLinux and hosts the Python measurement scripts. The PL holds the complete RISC-V SoC, comprising the CV32E40X core, the ASCON accelerator, the memory subsystem and the performance counter infrastructure. The whole design runs from a single 60.002 MHz clock derived from the PS FCLK0 output, and closes timing with a Worst Negative Slack of +2.145 ns.

The PL design is a Vivado 2023.2 IP Integrator block design named `RISC_V`, containing twelve IP instances: the `Zynq_processing_system7_0` and its reset controller, the `axi_interconnect_0` crossbar, paired AXI-BRAM controllers and block memory generators forming a 64 KB IMEM and a 64 KB DMEM, the two user-authored module reference wrappers, and three `xlconstant` tie-off blocks. Section 3.7.1 walks through the role of every block in full.

The canvas itself is reproduced full page in Appendix C as Figure C.1 and Figure C.2. These are a single screenshot split across two figures purely because the full canvas exceeds one page at a readable resolution, they are not separate designs. The first shows the control-plane side and the second the memory and interconnect side.

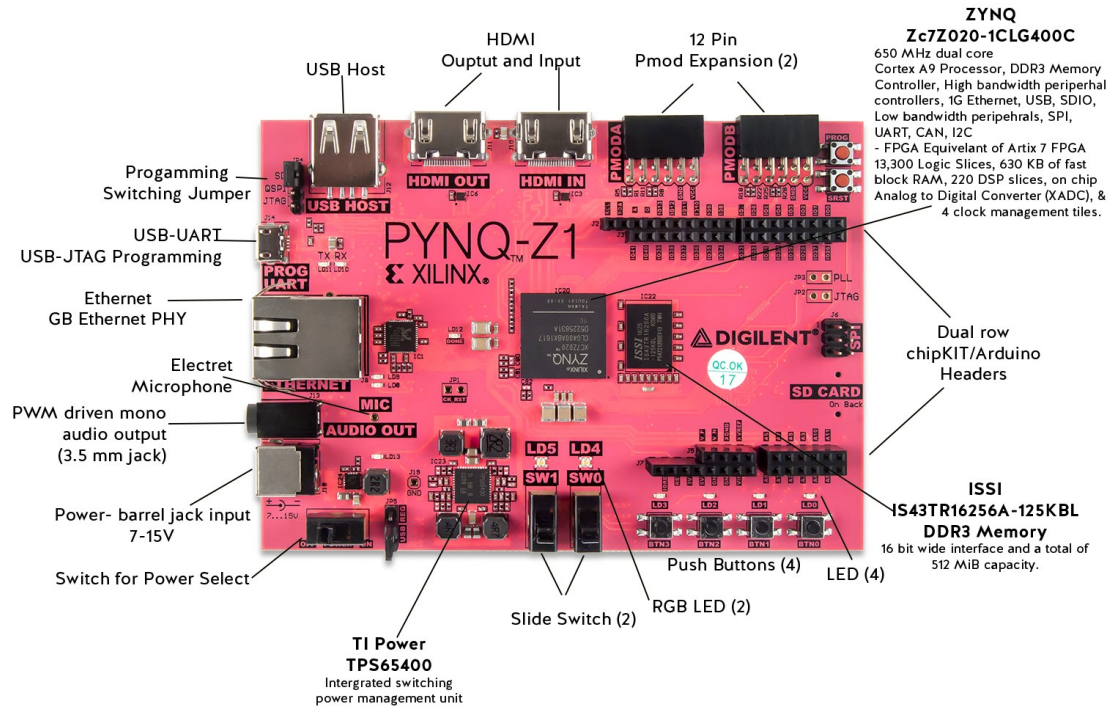


Figure 3.1: PYNQ Z1 FPGA [1]

The PS reaches the PL through the AXI GP0 master port. Because the PS issues AXI3 and the BRAM controllers expect AXI4, the interconnect inserts an automatic AXI3-to-AXI4 converter. A further AXI4-to-AXI4LITE converter sits on the path to `accel_ctrl_regs`, which implements AXI4LITE only. The three fixed address windows are given in Table 3.1.

Table 3.1: AXI Address Map

Peripheral	Base Address	Size	Function
<code>axi_bram_ctrl_imem_0</code>	0x40000000	64 KB	RISC-V Instruction BRAM
<code>axi_bram_ctrl_dmem_0</code>	0x40010000	64 KB	RISC-V Data BRAM
<code>accel_ctrl_regs_wrap_0</code>	0x40020000	4 KB	Control/Status Registers

3.2.2 Harvard Memory Architecture

The CV32E40X has structurally separate instruction and data OBI ports and cannot be configured as Von Neumann, so the design is Harvard by necessity: `.text` and `.rodata` live in IMEM, the remaining sections in

DMEM, and the PS loads both BRAMs over separate AXI slaves before releasing the core. One consequence matters downstream: const data would land in `.rodata/IMEM`, unreachable from the data bus, so every benchmark input array is declared non-const. The separation also means no single binary holds the full image, so objcopy runs twice per ELF, yielding 56 IMEM/DMEM pairs across the project.

The core speaks OBI while the Xilinx BRAM primitives use a different protocol with a one-cycle registered read. Three authored modules bridge them. `instr_mem_cv32e40x` is stateless, since instruction fetch is single-master, read-only and always grantable. The OBI contract expects zero-wait-state instruction memory, and any added latency would inflate the CPI figures in Chapter 4. `data_mem_cv32e40x` is a pure combinational passthrough, because all DMEM arbitration timing belongs to the arbiter and buffering here would desynchronize the grant signal and corrupt `stall_count`.

BRAM read output is registered, so data presented at cycle N appears at $N+1$ and a three-state design would capture stale data. The arbiter therefore uses four states, giving every DMEM transaction a fixed four-cycle latency of one grant cycle and three stall cycles. This is the entire source of every `stall_count` value in Chapter 4, since *total stalls* = $3 \times \text{DMEM accesses}$. Arbitration is fixed-priority with the CPU winning, and because `ascon_permutation.sv` drives `mem_req = 0` to satisfy Marshall et al. Requirement 4, the accelerator side is never exercised during a benchmark.

3.2.3 SoC Integration and Performance Counter

`cv32e40x_system` is the top-level shell instantiating every module above plus the core and accelerator, synthesized as the `cv32e40x_system_wrap_0` block of Figure C.1 in Appendix C, so every RTL module described here sits inside that one IP Integrator block. Table 3.2 lists the seven sub-modules, connected through six CV-X-IF interface instances

because the routing is fixed, swapping in a different accelerator requires changing only `accel_wrapper.sv` and the accelerator itself.

Table 3.2: Module Instantiations `cv32e40x_system.sv`

Instance	Module	Role
u_core	cv32e40x_core	4-stage RV32IMC core, X_EXT=1 for the coprocessor channel
u_xif_mux	cv32e40x_xif_mux	Gates all six CV-X-IF channels when <code>accel_enable=0</code>
u_accel	accel_wrapper, ascon_permutation	CV-X-IF coprocessor: the ASCON accelerator
u_imem	instr_mem_cv32e40x	IMEM adapter; source of <code>instr_rvalid</code>
u_dmem	data_mem_cv32e40x	DMEM passthrough to the arbiter
u_arbiter	memory_arbiter	DMEM arbitration and timing
u_perf	performance_counters	The five hardware counters

IMEM_BASE and DMEM_BASE default to `0x00000000` here, since the CPU-side adapters work in a zero-based domain while the Vivado wrapper passes the PS-side AXI bases as generics.

Five derivations feed `performance_counters.sv`, four of them single assignments: `instr_valid`, `core_stall`, `accel_active` and `accel_op_complete`. Their simplicity is deliberate, since added logic here would risk sampling error in every Chapter 4 number. The fifth, `halt_count`, is the module's only non-structural sequential logic.

```
// halt_count: freezes cycle_count at firmware completion boundary
// Trigger: CPU store of 0xDEADBEEF to DMEM address 0x8200
always_ff @(posedge clk or negedge rst_n) begin
    if (!rst_n) halt_count <= 1'b0;
    else if (counter_reset) halt_count <= 1'b0; // pre-pass reset
    else if (cpu_mem_we &&
             (cpu_mem_addr == 32'h00008200) &&
             (cpu_mem_wdata == 32'hDEADBEEF))
        halt_count <= 1'b1; // latched until the next pass
end
```

It fires identically regardless of `accel_enable`, since the trigger signals sit on the CPU side of the arbiter and the accelerator never writes to

DMEM. Table 3.3 in Section 3.2.3 gives the source and stop condition of each counter these signals drive.

Three reset levels exist: `rst_n` from the Zynq reset IP and the `core_reset` from the `CONTROL[0]`, already inverted in `accel_ctrl_regs.sv` so that 1 means running and this module passes it through unchanged and `accel_enable` from `CONTROL[2]`, gating the CV-X-IF channels. Twelve simulation-only assertions enforce IMEM timing, DMEM response latency, address range legality and alignment, and were the primary means of catching integration bugs. `cv32e40x_system_wrapper.v` exists only because IP Integrator requires a pure-Verilog top level, mapping 1:1 with no added logic.

Five hardware counters are instantiated in this same shell and are defined here once. Section 3.8 refers back to them rather than repeating the definitions. All five share one enable gate, `core_reset`, and are cleared by a `counter_reset` write before every pass. At 60 MHz the 32-bit `cycle_count` would need 71.6 seconds to overflow against a longest benchmark of 197,918 cycles, so overflow never occurs.

The critical boundary is `cycle_count`'s stop condition. Every `bench_main` writes `0xDEADBEEF` to `0x8200` as its last action, and the `halt_count` flip-flop latches on that write and gates the counter, so it freezes on the clock edge at which the firmware completes rather than when Python's polling detects it. The other four keep running, but the sentinel write is the last instruction, so none changes appreciably first. Because the trigger sits on the CPU side of the arbiter and the accelerator never writes to `0x8200`, the boundary behaves identically in both modes. Two counters also serve as correctness checks: `accel_ops` should equal the algorithm's known call count, and `accel_cycles/accel_ops` should equal the round count.

Table 3.3: Performance Counter and Boundary Conditions

Counter	Source signal	Stop condition	SW/HW differs?
cycle_count	One increment per clock while core_reset=1	halt_count=1, latched on the sentinel write	Yes – lower in HW; the speedup numerator
instr_count	instr_rvalid from u_imem; one pulse per instruction delivered to decode	Firmware halts	Yes – fewer instructions in HW
stall_count	data_req & ~data_gnt; high for the 3 stall cycles of every DMEM access	Firmware halts	No – identical at 24 of 28 points; PRF differs by 3
accel_ops	op_complete; one pulse per completed $p[r]$ call	Algorithm completes	Yes – 0 in SW, > 0 in HW
accel_cycles	accel_busy; high while the FSM is in ST_PERMUTE	Algorithm completes	Yes – 0 in SW, > 0 in HW

3.3 The Ascon Permutation Accelerator

3.3.1 Design Rationale

The accelerator connects through the CV-X-IF coprocessor interface rather than as a memory-mapped peripheral or DMA-coupled unit because the 320-bit ASCON state maps exactly onto ten RV32 registers, so it moves through the register file at zero transfer cost instead of through DMEM. This also satisfies Marshall et al. Requirement 4 directly: `ascon_permutation.sv` drives `mem_req` permanently to 0, and CV-X-IF stalls the pipeline automatically while the accelerator runs, so no firmware polling loop contaminates the cycle count.

The resulting design costs 597 LUTs and 368 flip-flops with no DSP or BRAM on the `xc7z020clg400-1`. Zero BRAM because no memory is allocated for the state, zero DSP because the S-box and diffusion layer are purely bitwise. Every call is fixed and data-independent, a $p[12]$ taking exactly 27 cycles (5 WS writes, 12 rounds, 10 RS reads), so `accel_cycles` always reads the round count regardless of data.

3.3.2 The CV-X-IF Protocol

CV-X-IF lets the core accept custom instructions without any modification to its own RTL. It exposes the instruction, its source register values and its transaction ID at the core boundary, and an external coprocessor claims the opcodes it wants. Because none of the roughly 40 OpenHW core files are touched, a future core update drops in unchanged and the accelerator is portable to any system built on the same core with `X_EXT=1`.

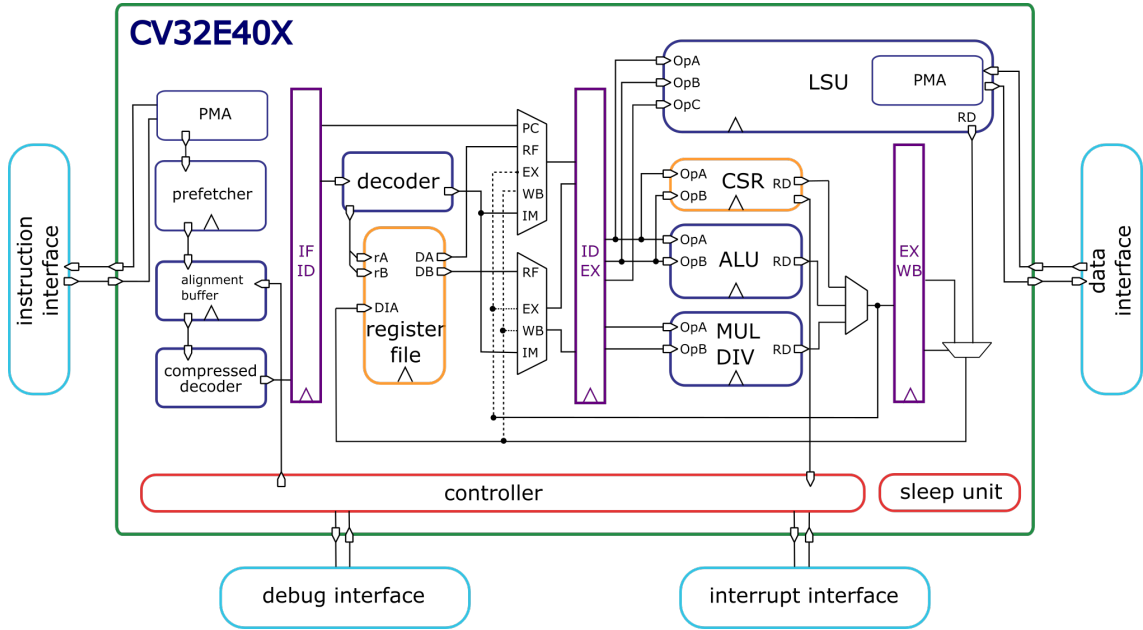


Figure 3.2: CV32E40X Core from OpenHW

CV-X-IF is enabled by setting `X_EXT=1` with the interface width parameters consistently across the core instantiation, all six interface declarations and the coprocessor. `X_NUM_RS=2` matters most. It gives each custom instruction two 32-bit operands, exactly enough to carry one 64-bit state word's halves in a single `ASCON_WS`. Of the six handshake channels the design uses two, as summarised in Table 3.4.

In a transaction the core asserts `issue_valid` with the instruction word and two operands. The accelerator asserts `issue_ready` and `accept` if the opcode is `custom-0` and the FSM is ready, with `writeback` set only for `ASCON_RS`. A trigger `WS` moves the FSM to `ST_PERMUTE` and drops `issue_ready`, stalling the core for the round count. All `ASCON`

Table 3.4: CV-X-IF Six-Channel Protocol

#	Channel	ASCON Usage
1	Compressed	Tied off – all custom instructions are 32-bit R-type
2	Issue	Active – carries all 15 custom <code>.word</code> instructions per call
3	Commit	Tied off – ASCON permutations are never speculative
4	Memory Request	Tied off – <code>mem_req</code> permanently 0 (Marshall Req. 4)
5	Memory Result	Tied off – no memory requests are ever issued
6	Result	Active – carries all 10 ASCON_RS results

instructions use `custom-0`, one of four opcode spaces the ISA guarantees no standard extension will claim because it is not decoded internally the core defers entirely to the coprocessor’s `accept` signal, which is the mechanism the bypass mux exploits.

Every ASCON custom instruction uses the standard RISC-V R-type format, encoded as raw `.word` directives since the toolchain has no built-in knowledge of the ISE. The field layout is given in Table A.1 of Appendix A: `funct7` carries the instruction class and round count, `funct3` the lane index 0–4, and `rs1/rs2` the high and low halves of the state word being written.

`rs1` and `rs2` are not decoded from the instruction word but arrive as live register contents forwarded by the core, unlike `funct7`, `funct3` and `rd`, which are literal bit fields. Six instruction variants exist across two classes, latching writes and result reads, tabulated in Table A.2.

The lower four bits of `funct7` encode the round count directly, so the decoder needs no case statement and a further round count would require only a new firmware constant. Restricting the trigger to lane 4 is what guarantees all five state words are loaded before a permutation can start: `is_ws_latch` requires `dec_funct3 != 3'b100` so lanes 0–3 can only latch, while the trigger wires require `dec_funct3 == 3'b100`, and the `ST_LOAD` to `ST_PERMUTE` transition can therefore occur only on a lane-4 write. The decode logic is described in Appendix A.4 and the macro itself is listed in Appendix A.3. In outline the macro issues five ASCON_WS instructions, the fifth carrying the trigger, then ten ASCON_RS reads. The WS instructions always read `x28/x29` as `rs1/rs2`, hard-coded into the `.word`

encoding, so each state word is first copied there with `mv`. RS instructions need no staging, their `rd` field varying per instruction.

3.3.3 Instruction Routing

`cv32e40x_xif_mux` sits between the core and `accel_wrapper`, gating all six CV-X-IF channels on `accel_enable` deasserted, the accelerator is combinationally invisible to the core, asserted, all six pass through unchanged. This lets one bitstream and one boot session produce both measurement passes with no reprogramming.

Only the valid and ready handshakes are gated. The data and request structs pass through unconditionally, since a receiver ignores them whenever valid is low. One asymmetry is deliberate, `result_ready` is left ungated even at `accel_enable=0`, because `result_valid` is already forced low on that path, and gating it risked blocking a legitimate handoff on the exact cycle `accel_enable` changes.

Table 3.5: CV-X-IF Channel Gating Summary

Channel	Gated: <code>accel_enable=0</code>	Always passes through
Compressed	valid, ready, resp	req (data struct)
Issue	valid, ready, resp	req (instr, rs1, rs2, id)
Commit	valid	commit struct (one-way, no return path)
Memory Req	valid	req (permanently idle – <code>mem_req=0</code>)
Memory Result	valid	result (never arrives; no request ever issued)
Result	valid, result data	<code>result_ready</code> (ungated by design)

In practice the gate never fires during a measurement run, since the SW binary issues no `custom-0` instruction at all, so the mux adds zero latency to any SW-mode measurement, were one issued while unaccepted, the core would raise an illegal instruction exception and the trap handler would record it, making the failure self diagnosing rather than silent. Because the mux is the only hardware difference between the two passes of Section 3.8.1, any cycle-count difference is attributable purely to the firmware’s instruction stream rather than to a reset transient, which makes it a correctness property of the measurement methodology rather than a convenience.

`accel_wrapper` sits between the mux and `ascon_permutation`, absorbing all six CV-X-IF channels and presenting a flat, protocol unaware interface to the accelerator underneath. The boundary was proven before ASCON existed, the wrapper's first child was `matrix_multiply_accel`, and swapping it for `ascon_permutation` required changing only the instantiation and the accelerator file, with no changes elsewhere in the SoC.

Only the Issue and Result channels carry live traffic. The other four are tied off. One tie-off is functional rather than cosmetic. `result.we` must be asserted whenever a result is presented, separately from `issue_resp.writeback`. The first tells the core a result is coming, the second tells it to commit the write and omitting it would silently discard every computed result with no error. The accept and writeback logic lives here rather than in `ascon_permutation` since both depend on fields already visible at this boundary (Appendix A.4), accept requires `issue_ready` to prevent a new instruction being taken mid-permutation, while writeback declares a property of the instruction type rather than a live handshake.

The accelerator's reset is `rst_n & ~accel_reset`, and since `accel_reset` is itself already inverted from `CONTROL[3]` the two inversions cancel, so `0x0E` releases core and FSM on the same clock edge, eliminating any race with the first ASCON_WS instruction. `accel_op_complete` passes through unmodified to increment `accel_ops`, which is why that counter equals the exact number of $p[r]$ calls executed.

3.3.4 State Representation and FSM

The `funct3` field is used directly, with no translation, as the array index into the hardware state registers, which is what connects the instruction encoding to the datapath. NIST SP 800-232 specifies a 320-bit state as five 64-bit words on the 32-bit CV32E40X each splits into high and low halves, held as the register arrays `S_H[0:4]` and `S_L[0:4]`.

These ten are the only storage for the state between the WS writes and the RS reads, no BRAM or DMEM is ever involved, and of the module's

368 flip-flops 320 are the state itself. They map one-to-one to the ten software registers of Section 3.3.2, the same funct3 lane index applying at every layer.

Table 3.6: State Representation Correspondence

NIST Word	Hardware	Lane (funct3)	ascon.S [11] Alias	ABI Reg	Physical Reg
S0	S_H[0]/S_L[0]	0	x0h / x0l	s0 / s1	x8 / x9
S1	S_H[1]/S_L[1]	1	x1h / x1l	s2 / s3	x18 / x19
S2	S_H[2]/S_L[2]	2	x2h / x2l	s4 / s5	x20 / x21
S3	S_H[3]/S_L[3]	3	x3h / x3l	s6 / s7	x22 / x23
S4	S_H[4]/S_L[4]	4	x4h / x4l	s8 / s9	x24 / x25

Callee-saved registers are used rather than temporaries because the permutation is called from a C-level NIST LWC API function and the ILP32 convention requires them to survive the call boundary. Marshall et al. Principle 3 requires that an ISE introduce no new architectural state, which S_H and S_L satisfy. They are invisible to every standard instruction and to debug infrastructure, their only access path is the WS/RS sequence, and they are not preserved across a context switch, making them an implementation detail of an operation that appears atomic to the programmer.

Three registers drive the iterative round engine: `round_count` indexes the round constant table, `num_rounds` latches the round count at the trigger so one FSM handles all three variants, and `start_idx` sets the starting offset, making the constant lookup the single expression `ROUND_CONST[start_idx + round_count]` with no case statement.

The shared suffix is not incidental. The ASCON specification defines shorter round variants as the final rounds of the 12-round sequence, which the `start_idx` formula implements directly. Two smaller registers complete the module: `prev_done`, which makes `op_complete` a true single-cycle pulse so `accel_ops` cannot over-count, and `xif_result_id_r`, which latches the transaction ID at the trigger for echo back during the RS phase.

The transaction protocol around these registers is implemented as a finite state machine, which has four states:

```
typedef enum logic [1:0] {
    ST_IDLE    = 2'b00,    // Waiting for the first ASCON_WS
    ST_LOAD    = 2'b01,    // Receiving state words for lanes 0-3
    ST_PERMUTE = 2'b10,    // Executing permutation rounds
    ST_RESULT  = 2'b11     // Permutation complete; serving reads
} fsm_state_t;
```

Table 3.7 summarises the four states. Two properties drive everything else: ST_LOAD accepts no RS reads, since reading mid-load would return a mix of old and new state, and `issue_ready` is held at 0 throughout ST_PERMUTE, which is the direct cause of the pipeline stall. `issue_ready` is the module's most important single signal, and is asserted only when two conditions hold together. The first is that the FSM is in ST_IDLE, ST_LOAD or ST_RESULT, which is what stalls the pipeline for the duration of ST_PERMUTE. The second is a separate guard that holds `issue_ready` low for one more cycle if the core has not yet accepted the previous RS result, which would otherwise create two overlapping `result_valid` assertions on the same channel. The complete state transition diagram is given as Figure A.1 in Appendix A.5.

ST_RESULT can transition straight to ST_LOAD or ST_PERMUTE without passing through ST_IDLE, which is what allows back-to-back calls with zero idle cycles, as the duplex $p[8]$ calls in AEAD128 rely on. A complete $p[12]$ call therefore runs 5 WS cycles, 12 stalled cycles in ST_PERMUTE and 10 RS cycles, 27 in total, with $p[8]$ and $p[6]$ following the same $5 + r + 10$ structure. The count is fixed and data independent, giving the constant time property expected of a cryptographic permutation.

What the accelerator computes while the FSM is busy is the round function itself. A full round applies three layers in sequence, $p = pC \circ pS \circ pL$, entirely combinational with no clocked elements and no data dependent branches, so it completes within one clock period. pC XORs an 8-bit round constant into the low 8 bits of word S2 alone; every other bit

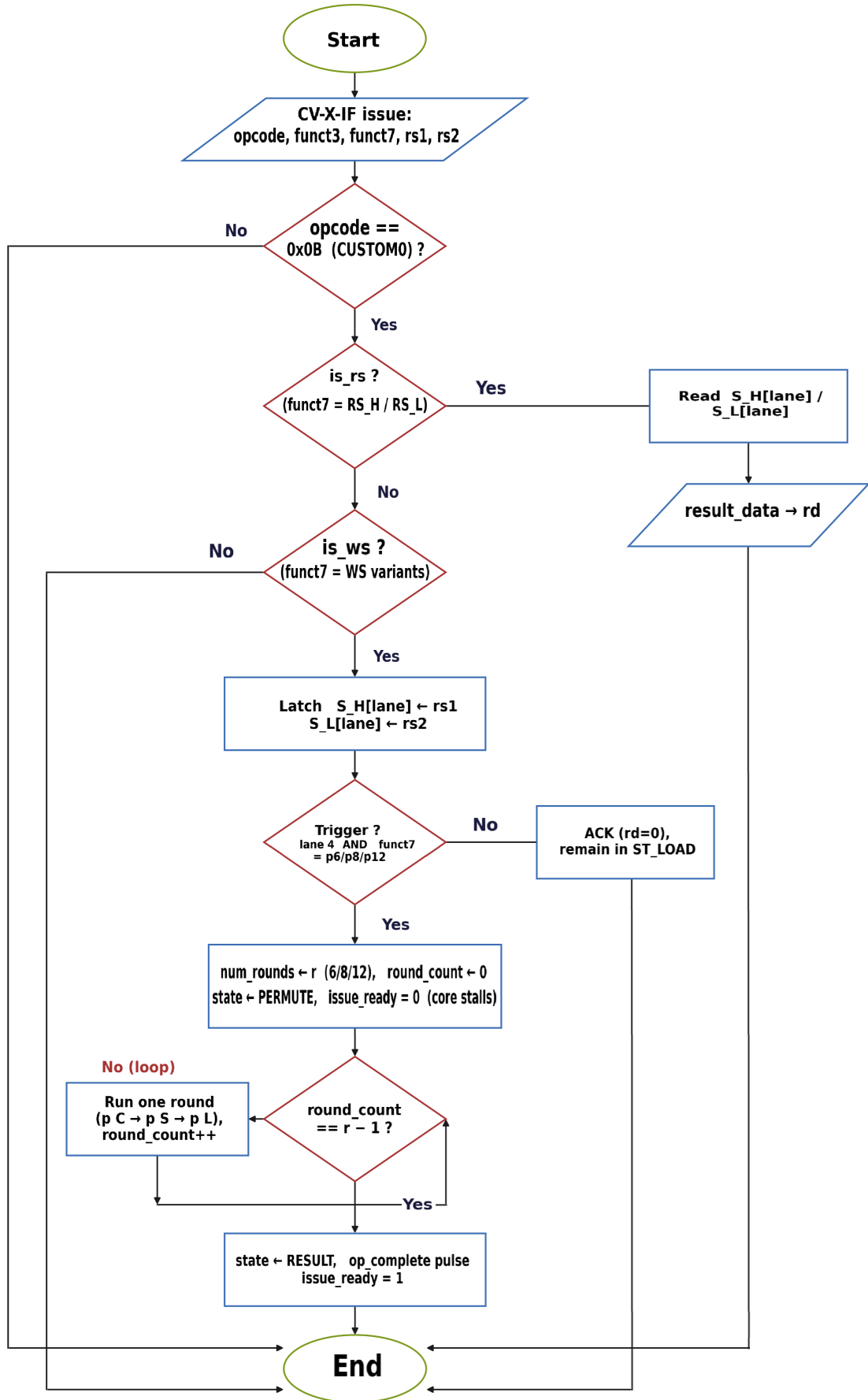


Figure 3.3: Decision Flow Chart

Table 3.7: FSM State Summary

State	issue_ready	busy/done	Key transitions
ST_IDLE	High*	0/0	ST_LOAD on WS latch; ST_PERMUTE on WS trigger if pre-loaded
ST_LOAD	High*	0/0	Remains on WS latch; ST_PERMUTE on lane-4 trigger; no RS accepted
ST_PERMUTE	Low	1/0	One round/cycle; ST_RESULT when round_count==num_rounds-1
ST_RESULT	High*	0/1	Serves RS reads; op_complete fires on entry; ST_LOAD or ST_PERMUTE for next call

* High only when no result_valid is still awaiting the core's acknowledgment.

passes through unchanged. A 16-entry table holds the NIST constants, the standard's sixteen-round sequence 0x3C, 0x2D, 0x1E, 0x0F, 0xF0, ..., 0x4B, indexed by `start_idx + round_count` so that $p[12]$ begins at index 4, $p[8]$ at 8 and $p[6]$ at 10, all three ending on 0x4B at index 15.

Changing the constant every round is what stops rounds looking identical to an attacker, which is what a slide attack exploits. pS then replaces each of the state's 64 bit-columns with the output of the same 5-bit S-box, applied in parallel across all columns. It is built from four fixed stages using only XOR, AND and NOT. An affine mixing stage, a nonlinear stage forming $T_i = \overline{X_i} \wedge X_{i+1}$, a stage recombining $Y_i = X_i \oplus T_{i+1}$, and a final output stage.

Because no stage branches on the data, the S-box's delay of 3–4 LUT levels is the same regardless of input, which is the constant-time property a cryptographic permutation needs. Only one of the four stages is nonlinear. Without it the round would collapse into an easily invertible linear map. pL diffuses each word by combining it with two rotated copies of itself,

$$S_i \leftarrow S_i \oplus \text{ROR64}(S_i, r_0) \oplus \text{ROR64}(S_i, r_1)$$

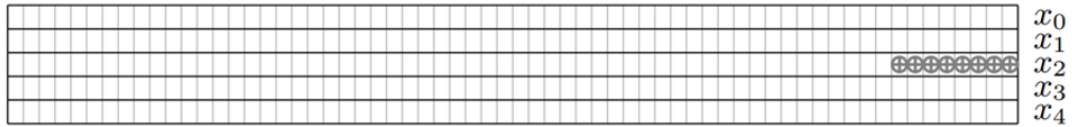
where r_0 and r_1 are fixed per-word rotation amounts. RV32 has no native 64-bit rotate, so `ror64` is built from the two 32-bit halves in three cases. The below 32 each half is shifted right and refilled from the other at exactly 32 the halves simply swap and above 32 the same construction applies with the halves exchanged and the amount reduced by 32.

Each word's halves are updated identically, its S-box output rotated by r_0 and r_1 and XORed. The five fixed pairs are (19, 28) for S0, (61, 39) for S1, (1, 6) for S2, (10, 17) for S3 and (7, 41) for S4, so S4 is the only word whose two amounts straddle the 32-bit boundary and therefore exercises both halves of the `ror64` construction.

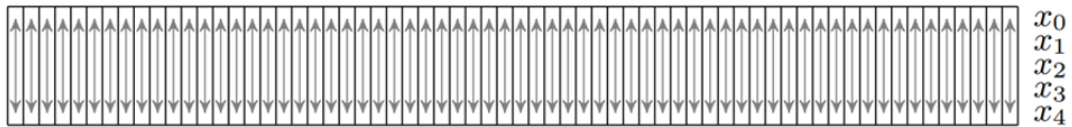
One clock cycle carries the state through the full pipeline, constant addition then substitution then diffusion, producing the next state:

$$S_H/S_L \rightarrow pC \rightarrow pS \rightarrow pL \rightarrow \text{next } S_H/S_L$$

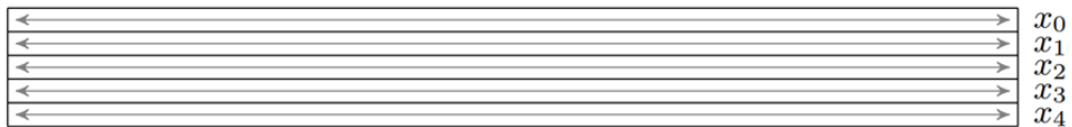
This is roughly 12 LUT levels, and Vivado attributes the design's critical path not to it but to the core's own multiplier (Section 3.7.2). Running the chain `num_rounds` times over the same registers is what makes the architecture iterative rather than unrolled, unrolling would replicate the round hardware r times at r times the LUT cost.



(a) Round constant addition p_C



(b) Substitution layer p_S with 5-bit S-box $\mathcal{S}(x)$



(c) Linear layer with 64-bit diffusion functions $\Sigma_i(x_i)$

Figure 3.4: ASCON-AEAD128 SPN

3.3.5 The AXI-Lite Register File

Every interaction between the Python host and the SoC passes through this module, an AXI-Lite slave at `0x40020000` that starts and stops the core,

selects SW or HW mode, resets the counters and returns the measurement data. Ten offsets are defined, of which five are read-only counters and the rest control and status registers, and the full map is given in Table A.3. Unrecognised offsets are ignored on write and read as zero. Two of the four CONTROL bits are active high in software convention but are inverted once, here, into active low hardware signals:

Specifically, `core_reset` and `accel_reset` are the inverted forms of `CONTROL[0]` and `CONTROL[3]` while `fetch_enable` and `accel_enable` map directly, which gives the three CONTROL values used throughout, `0x01` holds the core for safe memory loading, `0x02` selects SW mode and `0x0E` selects HW mode with accelerator and FSM released. A three state sequence handles the AXI-Lite write handshake, with only three offsets writable, and because a default assignment runs before the case statement every cycle, `counter_reset` is high for exactly one clock cycle however long the surrounding handshake takes.

Two implementation details are worth recording, both of which produce a silent hang rather than an error. The ready signals `s_axi_awready` and `arready` must be initialised to 1 at reset, not 0. The Zynq reset IP takes about 16 ms to release `peripheral_aresetn` while Python issues its first write within a few milliseconds, so a slave that starts not-ready never acknowledges that write. The AXI specification permits initialising them high, and an assertion now checks it. Second, in Vivado 2023.2 the `X_INTERFACE_PARAMETER` declaring `PROTOCOL = AXI4LITE` must be attached to the `AWADDR` port specifically, on any other signal Vivado synthesises an empty AXI slave, the bitstream builds normally, and every transaction hangs with no diagnostic.

Every SW/HW measurement pair executes the same nine register accesses:

`cycle_count` freezes at Step 7 while the other four run on until Step 9, a few hundred cycles at most against benchmarks of tens of thousands, which is why `cycle_count` is the primary metric.

Table 3.8: Nine-Step Measurement Sequence

Step	Access	Purpose
1	CONTROL=0x01	Hold core in reset for safe IMEM / DMEM loading
2	Load IMEM/DMEM	Write firmware and input data binaries
3	Clear 0x8200, 0x8000	Remove stale sentinel / result from the previous pass
4	COUNTER_CTRL=0x01	Pulse counter_reset; zero all five counters (50 ms settle)
5	Read CYCLE/INSTR_COUNT	Optional pre-flight check that counters read zero
6	CONTROL=0x02 (SW) or 0x0E (HW)	Release core in the desired mode
7	Poll DMEM 0x8200 for 0xDEADBEEF	Wait for completion; cycle_count freezes here
8	Read all five counters	Read immediately to minimize post-completion drift
9	CONTROL=0x01	Re-assert reset before the next pass

3.4 Marshall et al. Compliance: Four Principles Verified

Section 3.1 claims the accelerator satisfies four applicable RISC-V ISE design principles drawn from Marshall et al. [29]. This section gives the RTL-level evidence for each. The paper’s AES-specific requirements [3] do not apply to a permutation based primitive and are not assessed.

Three of the four carry verification beyond the static RTL evidence in the table. P1 is enforced by the X_NUM_RS=2 synthesis parameter, a hard interface width constraint, P3 by the absence of any RTL path from any other instruction to S_H/S_L and P4 by the p_no_mem_req assertion at runtime, not merely by the constant zero assignment. Only P2 rests on architectural convention rather than a hardware interlock, though the encoding still makes it structurally correct since the result channel can only write to a standard register. Two further requirements from the same paper are also met, data oblivious execution, the round count being fixed by the trigger encoding and the S-box purely combinational (Section 3.3.4), and practical implementability at 597 LUTs and 368 flip-flops with 2.145 ns of slack.

Table 3.9: Marshall et al. Compliance: Four Principles

#	Principle	RTL Evidence	Status
P1	Two source registers, one destination register per instruction	X_NUM_RS=2; R-type rs1/rs2/rd fields fixed by the instruction format	SATISFIED
P2	Use the general purpose scalar register file; no new architectural register file	State lives in s0–s9 (x8–x25), the same registers <code>ascon.S</code> [11] uses; results write back through the standard <code>result.rd</code> path	SATISFIED
P3	No new architectural state visible to ISA, compiler, or OS	S_H / S_L are unaddressed, not debug visible, not context-switched; module header documents this explicitly, citing the paper’s own partial product register analogy	SATISFIED
P4	No new caches, scratchpads, or special purpose memory accessible by ISE instructions	<code>mem_req / mem_addr / mem_we / mem_be / mem_wdata</code> permanently tied to 0 in continuous assigns; <code>p_no_mem_req</code> assertion never fired in any simulation run	SATISFIED

3.5 Firmware Implementation

3.5.1 The Bare-Metal Environment

On reset the core has no stack, no zeroed memory and no exception handler. It simply begins executing whatever sits at IMEM address `0x00000000`. `startup.S` is that first code, shared identically across all seven family members since it operates only on linker provided symbols. It initialises `sp`, verifies 16-byte alignment, and zeroes `.bss`, which matters because `.bss` holds the output buffers and stale data would otherwise be compared against the wrong KAT [11] reference.

The Harvard .data bypass. A standard bare-metal startup copies `.data`’s initial values from their load address to their runtime address at boot. That is invalid here. The source address falls in IMEM, but a load always goes through the data bus to DMEM, so the copy would read back `0x00000000` and silently zero every key, nonce and plaintext, with no exception and no diagnostic. The fix is an unconditional jump around the copy loop, Python pre-loading DMEM instead. The firmware consequence

is that every `bench_main` input array must be static and non-const, so it lands in `.data/DMEM` rather than `.rodata/IMEM`, which the data bus cannot reach. A trap handler at `mtvec` writes diagnostic state to `RESULT[]` without touching the stack, and boot markers distinguish the three terminal outcomes Python must tell apart: completion, a trap, and a hang inside `main()`.

No standard linker script can express this memory model, since IMEM and DMEM both appear at address `0x00000000` and are distinguished only by which bus reaches them, so `cv32e40x.ld` was authored from scratch and is shared unmodified across all seven family members.

```
MEMORY {
    IMEM (rx)      : ORIGIN = 0x00000000, LENGTH = 64K
    DMEM (rw)      : ORIGIN = 0x00000000, LENGTH = 0x8000
    RESULTMEM (rw) : ORIGIN = 0x00008000, LENGTH = 0x200
    STACKMEM (rw)  : ORIGIN = 0x00008200, LENGTH = 0x7E00
}
```

IMEM and DMEM sharing `ORIGIN = 0x00000000` would be illegal in a Von Neumann script but is correct here, the linker accepting it only because every DMEM section's load address is redirected into IMEM via `AT >`. DMEM is split into three sub-regions so that `ASSERT` statements catch `.data` or `.bss` growing into the fixed-address `RESULT[]` buffer.

`_result_start(0x8000)` and `_sig_addr(0x8200)` are fixed regardless of any family member's section sizes, being defined at region `ORIGINs`, and both are hardcoded in all seven Python scripts. The directive `.data : { ... } > DMEM AT> IMEM` puts the runtime address in DMEM but the load address in IMEM, to control where `objcopy` finds the initial values because that address sits at a nonzero offset, the `objcopy` pass must include `-change-section-lma .data=0x0`, or the binary would begin with zero padding that Python would write into DMEM ahead of the real key and nonce.

3.5.2 The Permutation Assembly

`ascon.S` [11], used for AEAD128, is the official NIST LWC `asm_rv32i` reference implementation. The other six are original contributions of this project, since `ascon-c` publishes no RV32 assembly for Hash256, XOF128, CXOF128, MAC, PRF or PRFShort, only portable C. They establish the software baseline every speedup figure is measured against.

All seven share the same register convention: state words `S0–S4` occupy the callee-saved pairs `s0/s1` through `s8/s9`, as the ILP32 ABI requires. Those pairs match the `funct3` lane numbering of the ISE encoding and the hardware array indexing (Sections 3.3.2 and 3.3.4), giving a three-way identity in which the same integer is the `funct3` lane, the hardware index and the software register ordinal.

The `sbox` macro implements the S-box over one 32-bit half at a time and `linear` implements `pL` as two 64-bit rotations XORed together. The round loop XORs the constant into `S2`, calls `sbox` twice and `linear` five times, then advances the constant. The structure is identical in all seven files, differing only in starting constant, rate and phase count.

Two structural details matter for Chapter 4. CXOF128’s customization phase adds a fixed four-call overhead before any message processing, proportionally largest at short lengths, which is why it shows the highest speedup ratios at small inputs. PRFShort, at the other extreme, uses one `p[12]` call with no absorb loop, so its permutation *call count* is constant across all four lengths while its cycle count is not, every cycle by which it varies with input length being non-permutation work the accelerator never touches.

The hardware variants are derived from these files directly. Every `_accel.S` file is byte-for-byte identical to its software counterpart except at the permutation call site, sharing signature, register convention, stack frame and phase structure, with only the software call replaced by one macro expansion. This was a deliberate requirement. A firmware author should be able to convert a verified software file into a working hardware

one by swapping a single call pattern. All seven use exactly 25 instructions per call (10 mv, 5 WS, 10 RS), a uniformity that follows from the state always being 320 bits, only `trigger_funct7` varies.

Consecutive calls need no register saves and no status polling. State persists in `s0–s9` across the macro expansion, leaving a four-instruction gap between one permutation and the next. A memory-mapped or DMA-coupled design would instead need explicit write-state, start, poll and read-back steps between every call, adding several hundred DMEM-latency instructions for Hash256 at 1024 B, since here the pipeline stall itself functions as the implicit poll.

3.5.3 Build System

The toolchain is `riscv32-unknown-elf-gcc 15.2.0`, a bare-metal RV32 cross-compiler. The environment and flag set are listed in Appendix C.4. Every flag follows from a hardware constraint rather than preference, `-mstrict-align` because the load-store unit faults on unaligned access and `-mno-relax` because GP is never initialised in this startup.

Every measurement point produces an SW and an HW target, identical except in which permutation assembly is linked. Each ELF passes through the two-pass objcopy extraction of Section 3.2.2 to produce its IMEM and DMEM binaries (Appendix C.4). One consequence of `-nostdlib` deserves note: it removes `libc`'s `memcpy`, which `encrypt.c` still calls, so a minimal byte-by-byte replacement is linked and `-fno-builtin` forces every call through it rather than GCC's inlined version, which may assume unaligned-access support the core lacks.

There are 28 benchmark driver files, one per family member and measurement point. Each is self-contained, embedding the KAT [11] vectors for exactly one measurement point, calling the NIST LWC API function, packing output into `RESULT[]` and writing the completion sentinel. The

28 points and their permutation call counts are listed in Table B.1. A parameterized design would need fewer files but opens a specific failure mode. A Makefile mismatch between the `-D` value and the output filename yields a binary that runs correctly yet is labelled wrong, with no compile-time, link-time or runtime error to catch it. The dedicated file makes the filename itself the authoritative identifier.

AEAD128 is limited to two plaintext lengths, and so to four measurement points across the two associated-data conventions, because its KAT file’s plaintext and associated-data grid tops out at 32 bytes. CXOF128’s 0/8/16/32 B progression isolates its fixed four call customization overhead, MSG=0B measuring that overhead in pure form.

The measurement scripts run on the PYNQ-Z1’s [1] own ARM Cortex-A9 Processing System under PetaLinux rather than on a separate PC over JTAG or UART, which makes the PS-PL link a direct memory-mapped AXI bus with deterministic latency. A JTAG over USB approach was set aside for its unpredictable 1–50 ms per-register latency. All seven scripts share five engine functions differing only in per-family configuration, so a correctness fix to one applies to all seven, and each of the 28 expected output sets was cross checked on the Spike ISA simulator before deployment (Appendix C.5).

Binaries follow the pattern `{family}_{mode}_{unit}{length}` for IMEM, and the same with a `_dmem.bin` suffix for DMEM. For a given family member and length the SW and HW DMEM binaries are byte-identical, since both builds compile the same `bench_main` file and the DMEM binary derives purely from its `.data` section, while only `.text` differs between the two modes and `.text` is not part of the DMEM binary at all. The Python scripts nonetheless load each pass’s DMEM binary independently, keeping every measurement pass self-contained.

All 56 targets build on a development machine, producing 112 binaries which are transferred to the PYNQ-Z1 alongside the single shared bitstream. The 56 firmware runs, 28 measurement points in two modes, complete in

roughly 5–10 minutes in total, and the resulting seven ASCII tables are the complete measurement dataset Chapter 4 presents.

3.6 Functional Verification Methodology

The design was verified across seven progressive integration levels, from individually tested RTL modules up to a complete SoC running real compiled firmware for all seven family members against the official NIST LWC KAT vectors. Twenty one testbench files carry roughly 350 assertion checks, per-level results are tabulated in Appendix B.

3.6.1 Verification Architecture

Table 3.10: Seven Level Verification Campaign

Level	Scope	Tool
1	Each RTL module individually, no CPU core	Vivado XSim
2–3	Module pairs/triples, then full memory subsystem and accelerator chain, behavioral CPU driver	Vivado XSim
4	Complete <code>cv32e40x_system</code> , synthetic instruction injection (no real binary)	Vivado XSim
5	Complete SoC, real compiled firmware: non-ASCON (matrix multiply) infrastructure test	ModelSim
6	Full SoC, first ASCON binary, SW-only, KAT-verified	ModelSim
7	Full SoC, two-run SW / HW comparison, all seven family members	ModelSim

XSim handles Levels 1–4; ModelSim takes over for Levels 5–7, where simulating a pipelined core over thousands of real instructions needs faster large design simulation. No level was run until every testbench below it had passed, which makes root causing unambiguous, and one deliberate discontinuity supports this. Levels 3–4 verify the CV-X-IF infrastructure using `matrix_multiply_accel` rather than `ascon_permutation`, so that when the real datapath arrives at Level 7 any CV-X-IF failure is attributable to the new permutation RTL.

Why one message length in simulation. Levels 6–7 use a single 16-byte message. The standard comparison point in the literature, the shortest length still exercising every code path, and the one length all seven KAT files define.

3.6.2 Unit and Integration Testbenches (Levels 1–3)

Six unit testbenches cover the authored modules in isolation behind behavioral stubs, `ascon_permutation.sv` being exercised through its `accel_wrapper` parent rather than in a testbench of its own. The roughly 45 SVA properties are listed in Appendix B.2. Because they live in the RTL rather than the testbenches, a property verified at Level 1 is still watching at Level 7.

Two caught real bugs, the alignment property on `data_mem_cv32e40x` fired on an early build whose default memcpy produced unaligned halfword accesses, leading to the word-aligned replacement of Section 3.5.3, and the arbiter’s CPU-priority property caught an FSM holding `accel_req` high after grant. Two `performance_counters` properties reach beyond their module, the `halt_count` freeze mechanism (Section 3.2.3), and `instr_count ≤ cycle_count`, a hardware enforced guarantee that $CPI \geq 1.0$.

Above the unit level, three XSim testbenches connect groups of verified modules and test the interfaces between them (Appendix B.2). These levels do not instantiate the core. SystemVerilog tasks mimic the OBI and AXI-Lite handshakes, allowing stimulus a real core could not produce, such as simultaneous CPU and accelerator requests. The AXI-Lite write task’s poll on `awready` is what exposed the bug of Section 3.3.5, the testbench hanging exactly as Python’s first `ctrl.write()` later hung on the board.

3.6.3 Full SoC Verification: Levels 4 to 7

Levels 4 through 7 all exercise the complete SoC, differing in what they feed it and what they compare against. Level 4 instantiates the complete

`cv32e40x_system` as one DUT with the real core executing synthetic instruction words from a behavioral BRAM model rather than compiled firmware, so that the integration wiring is confirmed independently of the firmware toolchain and a failure here is unambiguously a wiring bug. The BRAM model reproduces RAMB36's one-cycle registered read latency, which a combinational model would hide. Three patterns are preloaded. A NOP sled, a store-word sled giving countable DMEM accesses, and a synthetic `custom-0` instruction exercising the CV-X-IF path with a real core for the first time. Thirty-four checks across seven groups then establish the BRAM wiring, address translation, counter connectivity, CV-X-IF gating and clean mid-execution reset that Level 5 depends on.

Level 5 is the first level to run real compiled firmware, loading an 8x8 matrix-multiplication binary and waiting for the completion sentinel exactly as the Python scripts do on hardware. Its checks fall into six groups: boot, fetch and retirement, DMEM integrity, functional correctness, completion, and baseline recording. A non-ASCON program is used first because a wrong ciphertext could indicate a pipeline, memory or algorithm bug indifferently, whereas a wrong matrix element cannot be an algorithm bug, so once all 64 match Spike any Level 6 failure is attributable to ASCON alone.

Level 6 runs the compiled AEAD128 software binary with `accel_enable = 0`, so the permutation executes entirely through `ascon.S` [11], establishing the software baseline every later speedup divides against. The most common source of false KAT failures in development was byte ordering rather than any algorithm bug, the firmware packing `RESULT[]` little-endian against the KAT file's hex string. Switching only the loaded binary also allowed `opt32` and `asm_rv32i` to be compared: both produce identical ciphertext, but `opt32`'s `C uint64_t` costs measurably more cycles than the register-pair assembly of Section 3.5.2, which is the basis for the baseline choice.

Level 7 is the final pre-FPGA gate. Each of six testbenches runs two

simulations in one invocation. Run A loads the software binary, then Run B reloads the BRAM with the hardware binary, both checked against the same KAT output (Appendix B.2). The testbench must clear the sentinel between runs, or Run A's `0xDEADBEEF` would still be present when Run B begins polling, reporting completion at zero cycles and a speedup of infinity. Group 7 carries the campaign's only non-trivial performance assertion. Hardware cycles must be strictly fewer than software cycles, or Level 7 does not certify. CXOF128 is the exception, added after the framework was complete and verified instead on the FPGA at Level 8.

No testbench simulates a message beyond 16 bytes, on three grounds. Only the absorb loop's iteration count changes with message length, and a wrong count would itself fail the KAT comparison. Simulation cannot establish timing at all, since ModelSim treats every combinational path as completing in zero time and the FPGA verifies every length independently, each run performing its own KAT comparison before any performance number is recorded.

Table 3.11: Verification Coverage by Length

Family	16B	Longer lengths
AEAD128, Hash256, XOF128, MAC, PRF	ModelSim (Levels 6–7)	FPGA Python KAT
CXOF128	FPGA Python (no L7 testbench)	FPGA Python KAT
PRFShort	ModelSim (Level 7)	FPGA Python KAT (0 / 1 / 8B)

Every one of the 28 measurement points is KAT verified by one method or the other before its cycle count is recorded. Simulation verified that the ISE functions correctly; the FPGA verifies that it is fast. Neither alone would suffice, and the full chain from RTL assertions through ModelSim KAT checks and Vivado timing closure to FPGA KAT checks is a defence-in-depth strategy in which each layer catches a distinct class of failure.

3.7 FPGA Implementation and Vivado

With the RTL verified through Level 7, this section documents its translation into a deployed PYNQ-Z1 bitstream. The process is not synthesis alone. The `xc7z020`'s hard Processing System can only be configured through a Vivado Block Design, not from HDL.

3.7.1 The Zynq 7000 Block Design

The `xc7z020` is a heterogeneous SoC. A hard ARM Cortex-A9 Processing System (PS) alongside 7-series Programmable Logic (PL), sharing one die but communicating only through AXI. The PS is fixed silicon and cannot be configured from HDL. It's DDR3 timing, peripherals, PLL dividers and AXI port wiring are set exclusively through the `processing_system7_0` IP block, which is what makes a Block Design mandatory for any project using the PS. The PL clock path originates at the PS7's `FCLKCLK[0]` pin and passes through a global clock buffer to every clocked register in the PL.

Twelve IP instances fall into five groups across Figure C.1 and Figure C.2 of Appendix C. The PS and its reset controller, the two authored wrappers, three tie-off blocks, the AXI crossbar, and the memory subsystem. The wrappers appear as `xilinx.com: module_ref` rather than catalogued IP because IP Integrator requires a pure-Verilog top level (Section 3.2.3). The dual-port BRAM structure is what makes simultaneous PS access and CPU execution possible: Port A belongs to the PS-facing AXI controller and Port B exclusively to the CV32E40X, on the same clock with no domain crossing.

Three elements are specific to this board. The PS7 hard silicon, the CV32E40X's 7-series BUFGCE clock gate primitive, for which OpenHW Group provides target specific equivalents, and the three AXI base addresses. Everything else, `ascon_permutation.sv` included, uses only standard synthesizable SystemVerilog and carries over to any FPGA or ASIC target unchanged. This matters for Chapter 4. The absolute cycle counts and

throughput figures are specific to this xc7z020 at 60 MHz, but the speedup ratio is independent of clock frequency and target platform, being a property of the ISA-level implementation rather than of the die.

3.7.2 Clock Configuration

The 60 MHz clock is generated inside the hard PS by its internal ARM PLL. The frequency was determined by timing analysis rather than caution. The critical path consumes 13.911 ns of the 16.666 ns period, leaving +2.145 ns of slack, whereas at 100 MHz the 10.000 ns period falls below that fixed delay, giving roughly -4.5 ns and a hard timing failure. Every PL register runs off the same `clk_fpga_0` edge, so `cycle_count` increments unambiguously, and PLL jitter cancels out of the speedup ratio.

Synthesis used Vivado’s default strategy, and post route utilization is low enough to leave ample free fabric for short, direct paths. The Vivado project summary for the completed run is reproduced as Figure C.3 in Appendix C.

DRC reports 48 warnings and zero errors, none in an authored accelerator module. Eleven concern the core’s un-pipelined DSP48E1 multiplier, the same structure that sets the 60 MHz ceiling. The remaining 37 are 36 on the arbiter’s asynchronous-reset BRAM control registers, accepted because reset asserts only before firmware is loaded so no access is ever in flight, and one CHECK-3 notice that the report rule limit was reached (Appendix C.3).

The result is unambiguous. The fully routed design closes timing completely: WNS = +2.145 ns with TNS = 0.000 ns, zero failing setup paths across all 14,775 endpoints, and WHS = +0.043 ns on the hold side. Both being positive together is the complete closure criterion.

The critical path runs from `mulh_state_reg[0]` in the core’s integer multiplier to `S_H_reg[0][26]` in the accelerator, since it is the CV-X-IF forwarding path itself. Two cascaded un-pipelined DSP48E1 blocks account for 63% of its logic delay, so the bottleneck is the core’s multiplier rather than the accelerator (Appendix C.1).

Design Timing Summary			
Setup	Hold	Pulse Width	
Worst Negative Slack (WNS): 2.145 ns	Worst Hold Slack (WHS): 0.043 ns	Worst Pulse Width Slack (WPWS): 7.353 ns	
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns	
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	
Total Number of Endpoints: 14775	Total Number of Endpoints: 14775	Total Number of Endpoints: 5044	
All user specified timing constraints are met.			

Figure 3.5: Vivado Timing Constraints

3.7.3 Resource Utilisation and Bitstream Deployment

The design occupies 11.65% of LUTs, 4.59% of flip-flops, 22.86% of RAMB36 blocks and 1.36% of DSP. The complete SoC accounts for 4,510 of the 6,200 LUTs, and within it the ASCON accelerator is 597 LUTs and 368 flip-flops with zero BRAM and zero DSP (Appendix C.2). Total on-chip power is estimated at 1.723 W, of which 1.520 W is the PS7 hard IP, unlike the utilization and timing figures above, which are static analyses of the routed netlist, this is a vectorless `report_power` estimate at Medium confidence, and Section 4.6.3.1 sets out what it does and does not support. The zero-BRAM, zero-DSP result follows from the register file coupling decision of Section 3.3.1. Holding the 320-bit state in flip-flops rather than a minimum 4 KB BRAM allocation keeps the footprint proportional to the state size. Figure C.4 in Appendix C shows the resulting post-implementation device floorplan.

Deployment follows directly from these results. Vivado produces the bitstream files `RISC_V_wrapper.bit` alongside `RISC_V_wrapper.hwh`, the handoff file PYNQ's `Overlay` class reads to discover the topology at load time. The build and deployment steps are given in Appendix C.5.

One bitstream serves all seven family members and all 28 measurement points with no re-synthesis, because the hardware is algorithm-agnostic: `ascon_permutation.sv` implements *Ascon-p[r]* generically for r in $\{6, 8, 12\}$ and is always present in the PL whichever firmware is loaded, the round count coming from the `funct7` field of the issuing `ASCON_WS` instruction. The only difference between a SW and a HW pass is a single

CONTROL register write (0x02 versus 0x0E). With `accel_enable` deasserted, `cv32e40x_xif_mux` gates every CV-X-IF valid and ready handshake, so the permutation FSM never leaves `ST_IDLE` and its datapath registers do not toggle. That is an argument from the RTL, not a power measurement. The estimate of Section 4.2 is vectorless and identical for both modes, so it can neither confirm nor refute it. What the measurements do establish is that the accelerator issues no memory traffic in either mode, since stall counts are identical between the two passes at 24 of the 28 measurement points.

3.8 Performance Measurement Methodology

3.8.1 The Two Pass Measurement

Every family member is benchmarked twice per message length: Pass 1 (SW) releases the core with `CONTROL=0x02`, accelerator bypassed, running the software permutation from `ascon.S` [11]. Pass 2 (HW) releases with `CONTROL=0x0E`, accelerator enabled, running the `ASCON_HW_PERM` macro. Both use identical input and expected output, so their cycle-count ratio is the speedup reported in Chapter 4. Each pass follows the nine-step AXI sequence of Section 3.3.5, loading IMEM and DMEM with the core held in reset, when the OBI fetch interface is silent and BRAM Port A writes cannot collide with Port B.

`stall_count` is identical between the two passes at the same message length at 24 of the 28 points, the four Ascon-PRF points differing by three cycles, which serves as a cross check. Stalls arise only from `data_req` & `~data_gnt` on the CPU's own DMEM accesses, the accelerator never issues a memory request (Marshall Principle 4), and both builds issue the same DMEM sequence.

Firmware completion and Python's detection of it occur at different times, so that the `cycle_count` is frozen in hardware. The `halt_count` mechanism of Section 3.2.3 latches at the exact cycle the firmware writes

0xDEADBEEF to 0x8200, not when Python's reset write arrives, so no polling latency enters the measurement. The other four counters run on through the post-completion WFI loop, which is negligible against benchmark durations of thousands to hundreds of thousands of cycles. This is what validates the comparison: `halt_count` fires on CPU-side data bus signals identical in the SW and HW builds of `bench_main`, so both passes freeze under the same condition with no systematic bias.

3.8.2 Counter and KAT Verification

All five counters are defined in Section 3.2.3. Three properties govern how the Chapter 4 figures should be read. `instr_count` is a fetch-side count, not a retirement count, since the CV32E40X exposes no retirement signal. On this in-order core the two differ only by pipeline fill and drain. `stall_count` gives 3 x (number of DMEM accesses) and excludes instruction fetch and accelerator traffic. `accel_ops` and `accel_cycles` read zero in every SW pass, and `accel_cycles` counts ST_PERMUTE occupancy alone, 12 cycles per `p[12]` call rather than 27, since the WS and RS phases are separate issue-channel transactions.

No cycle count is recorded until its run has passed KAT verification, since a run can complete and write the sentinel while still producing wrong output. Verification compares `RESULT[20 onward]` word by word against KAT-derived values, and any single mismatch excludes the run. Across the full 56-run campaign, zero KAT failures were recorded.

3.8.3 Derived Metrics and Traceability

Metric formulas are defined in Table 4.4 of Chapter 4. Two special cases apply. XOF128 and CXOF128 have a fixed 64-byte output independent of input length, so both an input CPB, the cross-family comparable figure, and an output CPB (`cc/64`) are reported, coinciding exactly at MSG=64B. CPB and throughput are undefined at MSG=0B for CXOF128 and PRFShort, where the cycle count remains meaningful but no per-byte rate exists.

The selection logic for the 28 measurement points is given in Section 3.5.3. Two constraints on Ascon-AEAD128 are worth separating, because they are of different kinds and the distinction matters for how its results are read.

The first is a genuine limit imposed by the KAT file. `LWC _AEAD _KAT _128 _128 .txt` is a 33×33 grid of plaintext length against associated-data length, indexed as $\text{Count} = 33|PT| + |AD| + 1$, and both axes stop at 32 bytes. AEAD128 therefore cannot be measured beyond 32 bytes without generating vectors outside the official file, which would forfeit the NIST verification standard applied throughout this work.

The second is a benchmarking convention, not a constraint. Nonzero associated-data vectors exist at every plaintext length in that grid, so the choice of $AD = 0$ was made rather than forced. It follows Steinegger and Primas [9], who benchmark “messages of various lengths (0 bytes of associated data)”, and it keeps the cycles-per-byte denominator unambiguous while making AEAD128 comparable at matched lengths with the other six members, none of which has an associated-data input. It is not the only convention in the literature. Cheng et al. [14] set associated-data length equal to plaintext length throughout, following the NIST microcontroller benchmarking framework, and the two works never reconcile the difference.

This thesis therefore measures AEAD128 under *both* conventions, at four points rather than two: Counts 529 and 1057 at $AD = 0$, and Counts 545 and 1089 at $AD = |PT|$. The four builds differ only in `BENCH_AD_LEN` and the associated-data array, so the cycle difference between a matched pair isolates the cost of associated-data processing exactly. A secondary benefit is coverage: with $AD = 0$ the associated-data absorption loop inside `ascon_core()` is compiled into every AEAD binary but never entered at run time, so the $AD = |PT|$ points are what exercise and KAT-verify that path on hardware. Section 4.4.2 reports the consequence.

Each family’s script prints one ASCII table per message length, SW and HW side by side. Every number in Chapter 4 traces to exactly one cell,

addressed by family, message length and metric row. All seven results files are archived with the firmware and scripts as the reproducible record.

3.9 Summary

This chapter documented the accelerator's design (Section 3.3), its Marshall et al. compliance (Section 3.4), the firmware and build system (Section 3.5), the seven level verification campaign (Section 3.6), the FPGA implementation and timing closure (Section 3.7), and the two pass measurement methodology (Section 3.8) together the complete, reproducible basis for the results Chapter 4 presents.

CHAPTER 4

RESULTS

4.1 Chapter Introduction

Every performance figure in this chapter is measured on real hardware. A synthesized, placed and routed bitstream running on the PYNQ-Z1 at 60 MHz, not a simulation estimate. Not every *quantity* reported here is of that kind, however, and the distinction is stated once, up front, rather than left to the reader to infer. Area and timing figures are post-implementation report values, produced by static analysis of the routed netlist rather than by observation of the running device. Power is a tool estimate, and it is the only estimated quantity in this chapter. Table 4.1 records the provenance of each class of number, and quantities marked [†] throughout the chapter are estimates in the sense defined there.

With that established, the chapter answers two questions: what a CV-X-IF coupled ASCON accelerator costs in area, timing and power, and how well a single shared permutation core serves all seven ASCON family members.

Tool-reported is not the same as estimated. Utilization and timing are deterministic, reproducible analyses of the actual routed netlist. No probe was placed on the device, but no probabilistic modelling is involved either. Power is different in kind, and Section 4.2 states exactly what its estimate rests on.

Table 4.1: Provenance of Reported Quantities

Quantity	Provenance	Instrument or source
cc, ic, sc, accel_ops, accel_cycles	Measured	RTL performance counters on the deployed bitstream, frozen by halt_count
KAT pass / fail at each point	Measured	Firmware output buffer read back over AXI-Lite and compared to the NIST vector [11]
CPI, Stall %, CPB, Speedup, Throughput	Derived	Arithmetic on the measured counters (Table 4.4); exact given their inputs
LUT, FF, RAMB36, DSP	Tool-reported (static)	report_utilization, post-route, Vivado 2023.2
WNS, TNS, WHS	Tool-reported (static)	report_timing_summary, post-route
On-chip power, junction temperature [†]	Tool-estimated	report_power, post-route, <i>vectorless</i> , overall confidence Medium
Figures attributed to published work	Third-party reported	As cited; marked (est.) where read from a figure rather than a table

Table 4.2: Fixed Configuration for All Measurements

Parameter	Value
FPGA	Xilinx [10] Zynq-7000 xc7z020clg400-1 (PYNQ-Z1)
Clock	60 MHz
Core	CV32E40X RV32IMC via CV-X-IF
Permutation hardware	ascon_permutation.sv – iterative, one round per clock cycle (not pipelined)
SW / HW assembly	asm_rv32i / per-family *_accel.S
Cycle accuracy	halt_count RTL latch, freezing cc on the completion sentinel write
AEAD128 associated data	Both conventions measured: $AD = 0$ and $AD = PT $ (Section 4.4.2)

4.2 FPGA Implementation Results

The full methodology and reasoning behind these figures is in Section 3.7. This is the results summary. The three groups below are reported separately because, per Table 4.1, they are three different kinds of number.

Timing (tool-reported, `report_timing_summary`, post-route). WNS = +2.145 ns, TNS = 0.000 ns, WHS = +0.043 ns. Fully closed at 60 MHz, with the critical path running from the CV32E40X’s multiplier into the accelerator’s state register.

Resources (tool-reported, `report_utilization`, post-route). The complete design uses 6,200 LUTs / 4,888 FFs / 32 RAMB36 / 3 DSP (11.65% / 4.59% / 22.86% / 1.36% of device capacity), of which the CV32E40X SoC block `cv32e40x_system_wrap_0` accounts for 4,510 LUTs and the remainder is the AXI interconnect, the two BRAM controllers and the processing-system glue. The ASCON accelerator itself is 597 LUTs and 368 flip-flops, 13.2% of that SoC block and 9.6% of total device usage, with zero BRAM and zero DSP, since the permutation is pure XOR/AND/rotation logic with no multiplication.

Power (tool-estimated[†], `report_power`, post-route). Total on-chip power is 1.723 W, of which 1.583 W is dynamic and 0.140 W device static. Of the dynamic total, 1.520 W is the PS7 hard IP and only 0.063 W is PL dynamic power. Within the PL, block RAM accounts for 0.031 W against 0.008 W for all slice logic, and the `cv32e40x_system_wrap_0` instance, core and accelerator together, draws 0.021 W. Two structural points follow and are worth stating because they bound how much this figure can say about ASCON at all. The PS7 total is dominated by the ARM core supply (0.718 W on `Vccpint`) and the DDR3 interface (0.686 W on `Vcco_ddr`), neither of which participates in the cryptographic computation and inside the PL, the memories draw roughly four times what the entire logic fabric does. The accelerator is therefore not separable from the core in this report, and core-plus-accelerator is 1.2% of the SoC total.

This estimate carries three qualifications that Section 4.6.3.1 develops

and Appendix C.2 documents in full. It is *vectorless*. The report records no setting file, no simulation activity file and no matched design nets, so internal switching activity is modelled from tool defaults rather than derived from a simulation of the firmware. Its overall confidence is **Medium**, and the single term responsible is internal node activity, for which Vivado records that under 25% of internal nodes are user-specified, against over 95% for both clock and I/O activity and High confidence on implementation state and device models. And its thermal figures rest on default environment assumptions, including 250 LFM airflow and a 10-inch board, that the PYNQ-Z1 as deployed does not satisfy. The reported 44.9 °C junction temperature should be read accordingly.

4.3 Functional Verification Summary

Every cycle count in Section 4.4 is only meaningful if the computation producing it is correct.

KAT coverage. All 28 measurement points map to a specific NIST LWC KAT file [11] and Count entry, with the same permutation call breakdown already given in Table B.1. The KAT source file follows the algorithm in each case.

Simulation to FPGA consistency. Level 7 ModelSim testbenches passed 56 assertion checks for five family members and 60 for XOF128 additional output word checks before any FPGA deployment.

Table 4.3: ModelSim and FPGA CC at MSG=16B

Family	L7 SW / L8 SW	Δ SW	L7 HW / L8 HW	Δ HW
AEAD128	4,957 / 4,931	−26 (0.5%)	1,270 / 1,244	−26 (2%)
Hash256	9,358 / 9,355	−3	1,046 / 1,043	−3
XOF128	15,473 / 15,470	−3	1,621 / 1,618	−3
MAC	3,462 / 3,459	−3	690 / 687	−3
PRF	8,356 / 8,353	−3	1,414 / 1,411	−3
PRFShort	1,964 / 1,961	−3	579 / 576	−3
CXOF128	− / 21,313	N/A	− / 1,917	N/A

Six of seven families agree to within 26 cycles under 2%, confirming the `halt_count` mechanism is cycle accurate on real silicon, the small residual gap is boot sequence and BRAM timing model differences between ModelSim and real hardware.

Full campaign result. All 56 individual runs (28 points x 2 modes) passed KAT verification, 28/28 SW, 28/28 HW, zero failures. Verified output ranges: RESULT[20–27] for AEAD128 at 16B (RESULT[20–31] at 32B, since ciphertext is longer), RESULT[20–27] for Hash256 / MAC / PRFShort’s digest or tag, and RESULT[20–35] for the 64 byte outputs of XOOF128 / CXOOF128 / PRF.

4.4 Per-Family Results

4.4.1 Performance Metrics: Definitions

Three hardware counters drive every metric below:

- CC – cycle count, frozen by `halt_count` at the completion sentinel
- IC – instruction count, fetch-side
- SC – stall count, pipeline cycles waiting on BRAM or the accelerator

Table 4.4: Metric Definitions

Metric	Formula	Notes
CPI	CC / IC	SW > 1 from BRAM stalls; HW is still higher since CV-X-IF wait cycles inflate CC without inflating IC (explained in Section 4.5.2)
Stall %	$SC / CC \times 100$	SW: BRAM latency only. HW: also includes accelerator wait cycles
CPB	$CC / \text{message_bytes}$	Primary comparison metric; lower is better; N/A at MSG=0B
Speedup	SW_CC / HW_CC	The primary acceleration result
Throughput (Mbps)	$\text{message_bytes} \times 8 \text{ bits} \times 60 \text{ MHz} / CC$	Secondary only – scales directly with clock frequency, not valid for cross-platform comparison

4.4.2 Ascon-AEAD128

AEAD128 encrypts a plaintext into an equal length ciphertext plus a 16-byte tag. It is the only family member with an associated-data input, and for the reasons set out in Section 3.8.3 it is measured under both benchmarking conventions in the literature: at $AD = 0$, following Steinegger and Primas [9], and at $AD = |PT|$, following Cheng et al. [14] and the NIST microcontroller benchmarking framework. The 32-byte ceiling on both axes of the KAT grid limits the family to two plaintext lengths.

The call counts follow directly from the specification: p^8 is applied after every associated-data block *including* the mandatory padded final block, but after every plaintext block *except* the last, which is followed by finalisation's p^{12} . With $AD = 0$ the associated-data phase is skipped entirely. The `accel_ops` counter read on each hardware pass matched the predicted count at all four points.

Table 4.5: AEAD128 Results: Cycle Counts

PT	AD	KAT	Calls	CC SW	CC HW	SC
16B	0B	529	3	4,931	1,244	274
16B	16B	545	5	6,959	1,444	315
32B	0B	1057	4	6,158	1,557	318
32B	32B	1089	7	9,178	1,834	369

Table 4.6: AEAD128 Results: Derived Metrics

PT	AD	CPI SW	CPI HW	CPB SW	CPB HW	Speedup
16B	0B	1.136	2.152	308.2	77.8	3.964x
16B	16B	1.104	2.158	434.9	90.2	4.819x
32B	0B	1.134	2.130	192.4	48.7	3.955x
32B	32B	1.097	2.130	286.8	57.3	5.004x

All KAT checks PASS at all four points. The CPB columns normalise by plaintext length alone, so at the $AD > 0$ rows the numerator includes associated-data absorption while the denominator does not. Those two CPB figures are therefore inflated and are not comparable with the $AD = 0$ rows or with the other family members. Speedup and the raw counts carry no

such distortion, because the identical associated-data work appears in both the software and the hardware run of the same binary.

4.4.2.1 The Cost of Associated Data

Because the $AD = 0$ and $AD = |PT|$ builds at a given plaintext length differ only in `BENCH_AD_LEN` and the contents of the associated-data array, the difference between a matched pair isolates the cost of associated-data processing exactly. Table 4.7 gives those differences.

Table 4.7: AD Effect at Matched Plaintext Length

PT	Extra calls	Δ CC SW	Δ CC HW	Cycles/call SW : HW	Marginal speedup
16B	+2	+2,028	+200	1,014 : 100	10.14x
32B	+3	+3,020	+277	1,007 : 92	10.90x

The marginal speedup of an associated-data block, above 10x at both lengths, is close to three times the 3.92x marginal speedup of a plaintext block, obtained from the 16B to 32B step at $AD = 0$ (+1,227 SW against +313 HW cycles for one added call). The two block types invoke the same permutation on the same hardware, so the difference is not in the accelerator but in what accompanies each block. Absorbing an associated-data block is a load, an XOR and a permutation. Absorbing a plaintext block is a load, an XOR, a *ciphertext store* and a permutation, and it also lengthens the output buffer, which adds DMEM traffic at three stall cycles per access and further iterations of the result-packing loop. The accelerator replaces the permutation and nothing else. In software the permutation dominates both paths, so their costs differ by only 17%. In hardware the permutation cost collapses and what remains is the residual software work, 100 cycles against 313. Associated-data absorption is very nearly pure permutation, which is the accelerator’s best case, whereas plaintext absorption carries irreducible ciphertext production with it.

Three consequences follow. **First**, the $AD = 0$ figures are a floor rather than a representative value: associated data raises AEAD128’s measured speedup from 3.964x to 4.819x at PT=16B and from 3.955x to 5.004x at

PT=32B, increases of 21.6% and 26.5%. Any comparison drawn against the $AD = 0$ rows understates this design by roughly a quarter.

Second, the direction of the length trend reverses between the two conventions. At $AD = 0$ speedup falls slightly with length, 3.964x to 3.955x, because the marginal plaintext block at 3.92x sits just below the running ratio and drags it down. At $AD = |PT|$ speedup rises, 4.819x to 5.004x, because each length step adds one associated-data block and one plaintext block whose blended marginal ratio is 5.69x, above the running ratio. The same accelerator produces opposite trends under the two conventions, and both are explained by the same marginal-cost model.

Third, normalising by the bytes actually processed rather than by plaintext alone shows the effect from another direction. Counting $|PT| + |AD|$, the hardware cost falls from 77.8 to 45.1 cycles per byte at PT=16B and from 48.7 to 28.7 at PT=32B once associated data is present, because the additional bytes are absorbed through the cheapest path the algorithm has.

The fixed initialisation and finalisation calls, two of the three to seven issued here, still dominate at these lengths, which is why AEAD128 remains the second-lowest speedup in the family behind PRFShort. The absorb fraction ranges from 33% at Count 529 to 57% at Count 1089, a wider span than the $AD = 0$ points alone could show, and its effect is consistent with the model of Section 4.5.1.

4.4.3 Ascon-Hash256

Hash256 absorbs at an 8-byte rate and squeezes a fixed 32-byte digest through 4 calls, giving it the highest permutation density of any family member relative to input size (132 total calls at 1024B).

Table 4.8: Hash256 Results: Cycle Counts

MSG	KAT	Calls	CC SW	CC HW	SC
16B	17	6	9,355	1,043	213
32B	33	8	12,252	1,168	223
64B	65	12	18,049	1,421	237
1024B	1025	132	191,929	8,981	717

Table 4.9: Hash256 Results: Derived Metrics

MSG	CPI SW	CPI HW	CPB SW	CPB HW	Speedup
16B	1.047	2.253	584.7	65.2	8.969x
32B	1.037	2.242	382.9	36.5	10.490x
64B	1.027	2.227	282.0	22.2	12.702x
1024B	1.007	2.181	187.4	8.8	21.371x

All KAT checks PASS. Clearest demonstration of the absorb fraction model from Section 4.5.1: speedup climbs monotonically from 8.969x to 21.371x as the absorb fraction grows from 33% to 97% of total calls.

4.4.4 Ascon-XOF128

XOF128 shares Hash256's 8 byte absorb rate but squeezes a full 64 bytes through 8 calls instead of 4, giving it a heavier fixed squeeze overhead (10 total calls at 16B versus Hash256's 6).

Table 4.10: XOF128 Results: Cycle Counts

MSG	KAT	Calls	CC SW	CC HW	SC
16B	17	10	15,470	1,618	293
32B	33	12	18,366	1,742	301
64B	65	16	24,157	1,989	319
1024B	1025	136	197,918	9,430	797

Table 4.11: XOF128 Results: Derived Metrics

MSG	CPI SW	CPI HW	In.CPB SW	In.CPB HW	Speedup
16B	1.044	2.325	966.9	101.1	9.561x
32B	1.038	2.313	574.0	54.4	10.543x
64B	1.030	2.303	377.5	31.1	12.145x
1024B	1.008	2.232	193.3	9.2	20.988x

All KAT checks PASS. Output CPB (CC/64) equals input CPB exactly at MSG=64B. XOF128's speedup progression tracks Hash256's almost exactly, running slightly lower at each length because its larger fixed squeeze cost (8 calls vs. 4) is a bigger fraction of the total.

4.4.5 Ascon-CXOF128

CXOF128 extends XOF128 with a fixed 16 byte customization string Z, adding a mandatory Z-phase of 4 permutation calls to every run regardless of message content. 1 call for the length prefix, 2 for the two full Z blocks, and 1 for the final partial Z block.

Table 4.12: CXOF128 Results: Cycle Counts

MSG	KAT	Calls	CC SW	CC HW	SC
0B	17	12	18,441	1,817	317
8B	281	13	19,882	1,872	320
16B	545	14	21,313	1,917	326
32B	1073	16	24,225	2,057	332

Table 4.13: CXOF128 Results: Derived Metrics

MSG	CPI SW	CPI HW	In.CPB SW	In.CPB HW	Speedup
0B	1.040	2.288	N/A	N/A	10.149x
8B	1.037	2.291	2485.2	234.0	10.621x
16B	1.035	2.312	1332.1	119.8	11.118x
32B	1.031	2.283	757.0	64.3	11.777x

All KAT checks PASS. Input CPB undefined at MSG=0B. Output CPB (CC/64) is 288.1/28.4 there instead. CXOF128's speedup curve is flatter and starts much higher than Hash256/XOF128's, because the fixed 4 call Z phase gives the accelerator a substantial workload floor even with zero message bytes.

4.4.6 Ascon-MAC

MAC uses a 32-byte absorb rate, the widest of any family member, so it needs zero absorb calls at MSG=16B (just init + final = 2 calls) and only engages its absorb loop past 32 bytes.

All KAT checks PASS. MAC achieves the lowest CPB of any family member at every length. Its speedup is correspondingly lower than Hash256's at matching lengths since there are fewer permutation calls over to offload.

Table 4.14: MAC Results: Cycle Counts

MSG	KAT	Calls	CC SW	CC HW	SC
16B	17	2	3,459	687	179
32B	33	3	4,943	785	185
64B	65	4	6,484	940	201
1024B	1025	34	52,714	5,590	681

4.4.7 Ascon-PRF

PRF shares MAC’s 32-byte absorb rate but adds a 3 call squeeze phase to produce 64 bytes of output.

All KAT checks PASS. PRF’s absorb fraction trails MAC’s at matching lengths, but at 1024B its 9.121x speedup converges toward MAC’s 9.430x.

4.4.8 Ascon-PRFShort

PRFShort is architecturally unique: exactly one $p[12]$ call regardless of input length, up to its 16 byte maximum. A deliberate constant time design against length based timing leakage.

That fixed call count makes PRFShort the cleanest available probe of non-permutation cost. Because the accelerated work is identical at every length, all length variation must come from software the accelerator never touches, and the measurements confirm it: taking the 16B point as reference, the software and hardware cycle counts move by almost exactly the same absolute amount at every other length — -15 and -16 at 0B, $+21$ and $+20$ at 1B, $+126$ and $+125$ at 8B. A shared increment of that kind can only be work present in both modes.

Table 4.15: MAC Results: Derived Metrics

MSG	CPI SW	CPI HW	CPB SW	CPB HW	Speedup
16B	1.099	2.127	216.2	42.9	5.035x
32B	1.072	2.088	154.5	24.5	6.297x
64B	1.060	2.013	101.3	14.7	6.898x
1024B	1.029	1.749	51.5	5.5	9.430x

Table 4.16: PRF Results: Cycle Counts

MSG	KAT	Calls	CC SW	CC HW	SC SW/HW
16B	17	5	8,353	1,411	306/303
32B	33	6	9,839	1,511	314/311
64B	65	7	11,379	1,665	332/329
1024B	1025	37	57,610	6,316	810/807

Table 4.17: PRF Results: Derived Metrics

MSG	CPI SW	CPI HW	CPB SW	CPB HW	Speedup
16B	1.087	2.268	522.1	88.2	5.920x
32B	1.075	2.232	307.5	47.2	6.512x
64B	1.068	2.171	177.8	26.0	6.834x
1024B	1.033	1.806	56.3	6.2	9.121x

Table 4.18: PRFShort Results: Cycle Counts

MSG	KAT	Calls	CC SW	CC HW	SC
0B	1	1	1,946	560	168
1B	2	1	1,982	596	177
8B	9	1	2,087	701	191
16B	17	1	1,961	576	177

Table 4.19: PRFShort Results: Derived Metrics

MSG	CPI SW	CPI HW	CPB SW	CPB HW	Speedup
0B	1.171	2.240	N/A	N/A	3.475x
1B	1.183	2.258	1982.0	596.0	3.326x
8B	1.210	2.240	260.9	87.6	2.977x
16B	1.177	2.268	122.6	36.0	3.405x

All KAT checks PASS. This is the project’s speedup floor with only one permutation call, the 25 instruction CV-X-IF handshake overhead has nothing to amortize over, unlike every other family member.

4.5 Cross Family Analysis and Discussion

4.5.1 The Absorb-Fraction Argument

Speedup increases with message length wherever an absorb loop exists, because total cycles split into a fixed component F (init, finalize, boot, identical in both modes) and a variable absorb component that only hardware accelerates. Throughout, the absorb fraction is the number of permutation calls that absorb a full message or associated-data block, divided by the total number of calls:

$$SW_CC = F + n \cdot A_sw$$

$$HW_CC = F + n \cdot A_hw$$

As n grows, the ratio approaches A_sw/A_hw roughly 23x for Hash256, matching the observed 21.371x at 1024B.

The model treats every permutation call as carrying the same absorb cost A , which holds within a family member but not across the different phases of one algorithm. Section 4.4.2.1 measures the two absorption paths of AEAD128 separately and finds A_sw/A_hw of 10.1–10.9x for an associated-data block against 3.92x for a plaintext block. The difference is the non-permutation work each block carries. A plaintext block must also store ciphertext and lengthen the output buffer, and the accelerator replaces none of that. The absorb fraction therefore predicts the *direction* of the trend reliably, but the asymptote A_sw/A_hw is a property of a specific phase, not of the family member as a whole.

Table 4.20: Absorb Fraction and Speedup

Family	16B Absorb% / Speedup	1024B Absorb% / Speedup
Hash256	33% / 8.969x	97% / 21.371x
XOF128	20% / 9.561x	94% / 20.988x
MAC	0% / 5.035x	94% / 9.430x
PRF	0% / 5.920x	86% / 9.121x
AEAD128 (AD=0)	33% / 3.964x	measured to 32B only
AEAD128 (AD= PT)	40% / 4.819x	57% / 5.004x at 32B
PRFShort	0% / 3.475x	constant

4.5.2 Why Hardware CPI Is Higher

HW CPI (1.749–2.325) consistently exceeds SW CPI (1.007–1.210). In software, the permutation is a tightly pipelined loop near CPI 1.0. In hardware, each 25 instruction macro includes a fixed 12 cycle wait that counts in CC but not IC, mechanically inflating CPI. The comparison that matters is absolute cycles. HW CPI 2.3 with CC=1,043 is still 8.97x faster than SW CPI 1.05 with CC=9,355, since the 12 cycle hardware wait replaces roughly 1,449 software cycles per call.

4.5.3 CPB Comparison

At 1024B, absorb rate dominates. MAC (5.5 HW CPB) and PRF (6.2) beat Hash256 (8.8) and XOF128 (9.2), directly because MAC / PRF absorb 32 bytes per call against 8, and the accelerator costs the same 12 cycles per call regardless of algorithm.

4.5.4 PRFShort as a Special Case

Its single call structure is a NIST specification requirement for constant time execution, and it produces the lowest speedups in the family. 2.977x to 3.475x across the 0–16B range. Its HW stall percentage (27–31%) is the highest of any family member because the single 12 cycle wait is a larger share of an already small total, with no absorb loop to dilute it.

The non-monotonic profile deserves comment, since 8B is slower than both 0B and 16B in both modes and yields the lowest speedup measured anywhere in this work. It is not an anomaly but the same marginal-cost effect identified for AEAD128 in Section 4.4.2.1, seen in its purest form. Of the four lengths, 16B is exactly one full aligned rate block and 0B is empty. Both are handled by aligned word accesses. 1B and 8B are partial blocks requiring byte-wise padding, and 8B has eight bytes to place rather than one, which costs the +126 SW / +125 HW cycles measured above. Because that increment is fixed software work added to both a large software total and a small hardware total, it depresses the ratio far more than it depresses either count: $(1961 + 126)/(576 + 125) = 2.977$ against $1961/576 = 3.405$. The accelerator is no slower at 8B. The surrounding firmware is more expensive, and with only one permutation call there is nothing to amortise that cost against.

This is the general principle behind every profile in this chapter, isolated by a family member whose accelerated work is constant. What the accelerator delivers is bounded not by the permutation but by how much non-permutation work travels with it.

4.5.5 CXOF128 as a Special Case

The fixed 4 call Z-phase creates a computational floor even at MSG=0B. Isolating the Z-phase cost: at MSG=16B, CXOF128 costs 299 more HW cycles and exactly 4 more calls than XOF128 at the same length about 75 cycles per Z-phase call, consistent with the 62 cycle CV-X-IF handshake cost plus setup.

4.6 Comparison With Published Work

This section compares the measured results of this work against published ASCON implementations on two dimensions. Performance (cycle counts,

speedup) and area / throughput efficiency. The comparison is organized into three tiers.

Section 4.6.1 covers direct numeric comparison against RISC-V instruction set extension implementations that report cycle counts or speedup at matched or matchable message lengths, where a side-by-side numeric comparison is meaningful.

Section 4.6.2 covers context only comparison against standalone hardware coprocessors and adjacent RISC-V+ASCON integrations, whose cycle counts are not commensurable with a software / hardware coprocessor system, but whose area and throughput figures provide useful upper bound and efficiency context.

Section 4.6.3 cites survey references that establish the metric vocabulary used throughout. Work on formal verification of ASCON's correctness or on physical attack resistance answers a different research question than the PPA characterization and integration patterns of RQ1 and RQ2, and is outside this section's scope.

4.6.1 Direct Numeric Comparison

The six works compared here report cycle counts, speedup or area for RISC-V ISE implementations of ASCON at lengths that match or can be aligned with this work, that shared basis is what makes a numeric comparison defensible. Table 4.22 consolidates the headline figures.

One convention applies throughout. Every one of these works measures speedup against a compiled-C baseline, whereas this work measures against hand-optimized assembly, so its ratios are systematically conservative relative to the published figures. Units also differ across technologies, ASIC gate equivalents against FPGA LUTs, and are reported side by side rather than converted.

4.6.1.1 Bader (2024) – CV-X-IF on CV32E40X: Direct Architectural Comparison

Bader [2] is the most architecturally comparable published work. It implements an ASCON ISE on the same CV32E40X core family using the same CV-X-IF interface, evaluated on the X-HEEP microcontroller platform. Bader reports speedup, cycle counts at MSG=16B and MSG=32B, per-level FPGA LUT and flip-flop counts for the coprocessor hierarchy in his Table 5.4, and ASIC synthesis area. The comparison at the same two message lengths on the same core family and the same interface protocol is the strongest available benchmark against prior art. This is also the design later published as CHIMERA-CR by Dolmeta et al. [12], examined in Section 4.6.1.5. The two sections read one accelerator at two reporting points rather than two separate comparators.

Table 4.21: Direct Comparison with Bader (2024)

Metric	Bader (2024)	This Work	Notes
Specification measured	Ascon v1.2	NIST SP 800-232	Different algorithms; see text
Software baseline	Compiled C	Hand-written RV32I asm	Ratios not commensurable
AEAD128 speedup @ 16B	3.56x	3.964x (AD=0)	Same order; not a ranking
Hash256 speedup @ 16B	6.08x	8.969x	Same order; not a ranking
FPGA LUT (coprocessor wrapper)	1,322 LUTs	597 LUTs	Matched scope; both include interface logic
Clock Frequency	40 MHz	60 MHz	PYNQ-Z1
MAC, PRF, PRFShort	Not reported	All 7 members	Novel contribution
Message length sweep	16B, 32B only	Up to 1024B	Extended range

No ranking is drawn between these figures, and the reason is not caution but arithmetic. Bader’s 3.56x is measured on `ascon128v1.2`, as his own comparison against Cheng et al. [14] states. Ascon-128 absorbs at a 64-bit rate using p^6 for data, whereas the standardised Ascon-AEAD128 measured here absorbs at a 128-bit rate using p^8 . At a given message length the two do not issue the same number of permutation calls at the same round depth, so they are different algorithms rather than two measurements of one. Independently, his baseline is compiled C and this work’s is hand-written

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power: 1.723 W
Design Power Budget: Not Specified
Process: typical
Power Budget Margin: N/A
Junction Temperature: 44.9°C
 Thermal Margin: 40.1°C (3.3 W)
 Ambient Temperature: 25.0 °C
 Effective θ_{JA} : 11.5°C/W
 Power supplied to off-chip devices: 0 W
 Confidence level: Medium

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

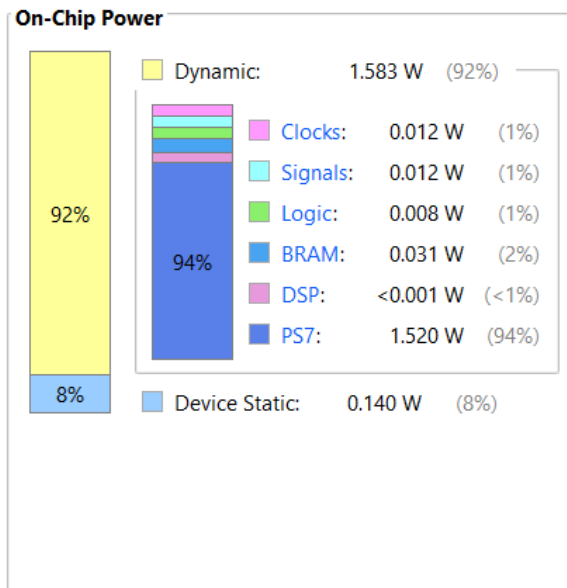


Figure 4.1: This Thesis Vivado On Chip Power

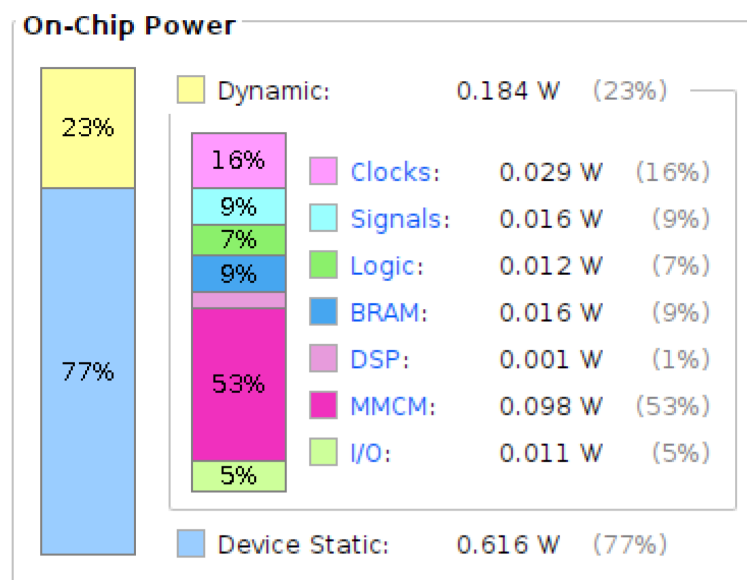


Figure 4.2: Bader (2024) [2] Vivado On Chip Power

assembly, and his cycle counts come from a single-vector QuestaSim simulation on the X-HEEP platform rather than from a deployed bitstream on this SoC, whose four-cycle DMEM arbiter latency differs from his memory subsystem. Three confounds separate the two ratios, and a percentage difference between them would quantify none of them.

What the comparison does establish is worth more than an ordering. Bader, having built both a true instruction set extension and a CV-X-IF coprocessor, attributes to the interface a “reduction of the speedup by a factor of 2”, arising from state transfer at one 32-bit word per cycle and a handshake on every instruction dispatch [2]. That is an independent identification of the overhead measured directly in Section 4.5.2, from a different platform and a different specification, and it corroborates that both designs pay the same structural price for attaching to an unmodified core. The substantive differences are elsewhere. This evaluation covers MAC, PRF, PRFShort and CXOF128, reaches 1024 B, measures both associated-data conventions, and reports the standardised specification, none of which any prior CV-X-IF ASCON study has done.

4.6.1.2 Steinegger and Primas (2020) – RI5CY/CV32E40P: Predecessor Core

Steinegger and Primas [9] implemented an ASCON ISE on the RI5CY core, the direct architectural predecessor of the CV32E40X, same 4 stage in order pipeline, originally from the PULP. They report speedups of 50x for AEAD and 80x for Hash compared to pure C software, at 50 MHz. These numbers are substantially higher than the results in this work and require careful contextualization before any comparison is meaningful.

The primary difference is integration style. Steinegger and Primas used tighter, direct register file coupling. Their ISE writes results straight to the general purpose register file with no CV-X-IF handshake, so each permutation call costs fewer overhead instructions than the 25-instruction sequence used here. That, together with the C baseline, inflates their speedup figures relative to those reported here.

A cycles-per-byte comparison is more informative than a speedup comparison, and it explains the apparent 50x versus 4x gap. Steinegger and Primas report roughly 2 cycles/byte at the asymptotic length for both AEAD and Hash with acceleration. This work measures 8.8 cycles/byte for Hash256 and 5.5 for MAC at 1024 B. The difference is the CV-X-IF handshake overhead per permutation call, which is the price of attaching an accelerator without modifying the CV32E40X core at all. The same overhead identified in Section 4.5.2 as the reason HW CPI exceeds SW CPI.

4.6.1.3 Moura (2024) – Ibex RISC-V: Same Pipeline Class

Moura [18] implements ASCON ISEs on the RISC-V Ibex core. RV32I, also a 4 stage in order pipeline, different from CV32E40X in ISA extensions and memory interface, synthesized in 28nm FDSOI ASIC technology. They report a performance gain of 60.3% (approximately 2.5x speedup) for ASCON compared to the unaccelerated C baseline, at a 128-bit (16B) message, the same message length as this work’s primary AEAD128 measurement point. The lower speedup relative to this work reflects the different interface. Moura uses a direct instruction extension without the CV-X-IF coprocessor protocol, and measures with a C baseline rather than assembly.

The Ibex comparison is valuable primarily for its CPI analysis. Moura reports CPI, instruction count and stall count using the same metric structure applied here, and their HW CPI values are elevated above 1.0 for the same reason as this work’s. Long-latency accelerator instructions inflate CPI without inflating the cycle count. Their energy figures are derived from each mode’s simulated switching activity rather than from a static estimate scaled by cycle count, a distinction discussed in Section 4.6.3.1.

4.6.1.4 Cheng et al. (2023, TCHES) – Rocket RV32GC: NIST LWC-Standard Methodology

Cheng et al. [14], published in IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES), design one instruction set extension for each of the ten NIST LWC final round submissions, including ASCON, evaluated on the Rocket RV32GC RISC-V core on FPGA at 50 MHz. For ASCON specifically, they report cycle counts and speedup at three message lengths. 16B, 128B, and 1024B across two accelerated configurations measured against the same unextended software baseline. A generic bit manipulation extension (Zbkb/x, not Ascon specific) and an extended configuration that adds a custom Ascon ISE on top of Zbkb/x. Their Table 2 reports three separately synthesized core configurations. The unextended Rocket core at 3,303 LUTs, a base core adding the generic Zbkb/x bit-manipulation extension at 3,764, and the extended core adding the Ascon ISE on top at 4,234. The Ascon-specific extension therefore costs 470 LUTs over the base core, or 931 LUTs cumulatively over the unextended core once Zbkb/x is included.

Two properties of their benchmark govern how it may be compared. Their message-length points set associated-data length equal to plaintext length throughout, following the NIST microcontroller benchmarking framework, so their 16 B figure describes 16 bytes of plaintext *and* 16 bytes of associated data. Until the measurements of Section 4.4.2.1 this thesis had no point under that convention, and the AD=0 figure of 3.964x was not its counterpart. The comparable measurement is Count 545, at 4.819x. Second, their headline 3.65x is not wholly attributable to Ascon-specific hardware. Their own Table 4 shows the generic Zbkb/x extension alone taking the unextended core from 14,801 to 7,839 cycles, a factor of 1.89x, with the Ascon ISE carrying it the remaining 1.93x to 4,059 cycles. Slightly under half of the reported speedup comes from a general-purpose bit-manipulation extension rather than from the accelerator.

Neither the cycle nor the area figures support an ordering. Their implementation is `ascon128v1.2`, absorbing at a 64-bit rate under p^6 , against the standardised Ascon-AEAD128 measured here at a 128-bit rate under p^8 . Their baseline is compiled C against this work’s hand-written assembly and their ISE is stateless by design, their Section 4 stating that all their extensions require combinational logic only, whereas this accelerator holds the full 320-bit state across instructions. On area, 597 LUTs against their 470 compares a stateful core serving seven family members with a stateless functional unit serving authenticated encryption alone, on a different host microarchitecture. What the two works share is the NIST benchmarking convention, which is why the associated-data measurements were made, and a broadly consistent order of magnitude at the one point where that convention now aligns.

4.6.1.5 Dolmeta et al. (2025) – CHIMERA: CV-X-IF on CV32E40PX

Dolmeta et al. [12], published at ISVLSI, introduce CHIMERA, a RISC-V accelerator for ASCON built on the same CV-X-IF interface used in this work, integrated into the X-HEEP microcontroller platform on the CV32E40PX core, a CV32E40P variant extended for CV-X-IF, distinct from but closely related to this work’s CV32E40X. CHIMERA-CR is the published form of the accelerator first documented in Bader’s [2] thesis at the same institution. The two reports give the same design hierarchy with matching register counts at every level, 810 flip-flops for the coprocessor wrapper, 123 for the interface controller, 687 for the permutation datapath and 320 for the dedicated state file. Section 4.6.1.1 and this section therefore compare against one accelerator reported at two benchmark points, not against two independent designs, and the two are counted once wherever this chapter reasons about how many prior works agree. CHIMERA’s own related work discussion identifies Cheng et al., Gewehr et al., and Steinegger & Primas as its closest prior art, the same cluster already covered

in Sections 4.6.1.2–4.6.1.4 which independently confirms the grouping used throughout this section.

CHIMERA is presented in two versions. The Complete Round (CR) version is a tightly coupled coprocessor with a dedicated state register file, architecturally the closest published analogue to this work’s own accelerator. The Bitwise Rotation Unit (BRU) version is a minimal ISE that accelerates only the linear diffusion rotation operations using the scalar register file, trading performance for area. Both versions cover AEAD-128, AEAD-128a, AEAD-80pq, Hash, Hasha, XOF, and XOFa, like Bader [2], neither version covers MAC, PRF, or PRFShort, so this work’s coverage advantage holds against CHIMERA as well.

The area comparison has to be drawn at a matched level of the hierarchy, and the level is not obvious from either report. This work’s 597 LUTs and 368 flip-flops are the `accel_wrapper` subtree (Appendix C.2), which carries the CV-X-IF handshake and decode logic as well as the permutation datapath and the 320-bit state. The matching scope in CHIMERA is the Chimera Wrapper at 1,321 LUTs and 810 registers, which likewise carries its 408-LUT XIF Controller alongside the datapath and its 320-register state file. The inner `Chimera` block of 913 LUTs and 687 registers excludes that interface logic and is therefore not comparable with a figure that includes this work’s. At matched scope this work is 597 LUTs and 368 flip-flops against 1,321 and 810, smaller on both counts while covering more of the family, consistent with the area pattern already seen against Bader and Cheng et al.: sharing one permutation core across the whole family costs less than duplicating logic per algorithm. CHIMERA-BRU is the instructive exception, its 573-LUT wrapper containing no dedicated permutation datapath at all, the accelerating logic instead sitting in the core’s decode and commit stages, at the cost of far lower speedup. That CR/BRU split is an independent illustration, on a different core, of the same coprocessor versus ISE trade-off argued in Section 3.3.1. It is not, however, independent of Bader [2]: CHIMERA-CR and Bader’s coprocessor are

the same accelerator, so the agreement between those two rows is internal consistency within one line of work rather than replication across groups.

Gewehr et al. [19], which CHIMERA also cites, is excluded as a comparison target: it uses the Ascon-Hash and Ascon-XOF permutation as one interchangeable building block inside a post-quantum Kyber implementation rather than evaluating ASCON as a standalone primitive.

4.6.1.6 Baldit, Dehez-Clementi and Dion (2026) – CVA6 on Zynq-7000

Baldit, Dehez-Clementi and Dion [30], published in IEEE Open Journal of the Solid State Circuits Society, target the CVA6 RISC-V core through the same CV-X-IF interface used in this work, deployed on a Xilinx Zynq-7000 FPGA, the same FPGA family as the PYNQ-Z1 board used here, and a closer platform match than CHIMERA’s Zynq UltraScale+. Unlike every other work in this section, which builds a single purpose ASCON accelerator, Baldit et al. [30] design reusable instruction level extensions shared between ChaCha20 and ASCON, explicitly prioritizing modularity and minimal hardware overhead over peak throughput. Their own comparison table benchmarks only against Steinegger & Primas [9] for ASCON, reporting their coprocessor as approximately twice its size in Gate Equivalents, an independent corroboration of the Steinegger & Primas [9] area figure already used in Table 4.22.

Table 4.22: Comparison with Directly related Works

Work	Platform	Interface	AEAD Speedup	Accel. Area	Energy Eff.
Steinegger & Primas (2020) [9]	RI5CY @ 50 MHz, ASIC	Direct reg-file coupling	50x (C baseline)	4.2 kGE (ASIC)	Not re- ported

Table 4.22 – continued

Work	Platform	Interface	AEAD Speedup	Accel. Area	Energy Eff.
Bader (2024) [2]	CV32E40X @ FPGA (area); Ques-taSim (cycles)	CV-X-IF (same)	3.56x (C baseline)	1,322 LUTs (wrapper)	Not re-ported here
Moura (2024) [18]	Ibex @ 28nm ASIC	Direct ISE	2.5x (C baseline)	+8.1% (NAND2 gates)	65.6% gain (measured)
Cheng et al. (2023) [14]	Rocket RV32GC @ 50 MHz, FPGA	Stateless FU (Zbkb/x + ISE)	3.65x @16B, AD=16B (C base-line)	470 LUTs (ISE over base)	Not re-ported
CHIMERA-CR (2025) [12]	CV32E40PX @ 50 MHz, FPGA	CV-X-IF (dedicated regfile)	2.98–3.22x @128B (C baseline)	1,321 LUTs (wrapper)	Not re-ported
CHIMERA-BRU (2025) [12]	CV32E40PX @ 50 MHz, FPGA	CV-X-IF (scalar ISE)	1.42x @128B (C baseline)	573 LUTs (wrapper)	Not re-ported
Baldit et al. (2026) [30]	CVA6 @ Zynq-7000, FPGA	CV-X-IF (shared, 2 ciphers)	2–3x (C baseline)	10.2 kGE (shared, ASIC-equiv.)	Not re-ported

Table 4.22 – continued

Work	Platform	Interface	AEAD Speedup	Accel. Area	Energy Eff.
This Work	CV32E40X @ 60 MHz, FPGA	CV-X-IF (coprocessor)	3.96x @16B AD=0; 4.82x AD=16B (asm baseline)	597 LUTs (13.2% of the CV32E40X SoC block)	Not independently measured – see Section 4.6.3.1

Speedup figures are only comparable within the same message length – see the individual subsections for which length each figure is measured at. Units differ across rows (ASIC kGE / NAND2 gates vs. FPGA LUTs) and are not converted between technologies. No energy-efficiency figure is quoted for this work: as Section [4.6.3.1](#) shows, any such figure derivable from the available power estimate would restate the measured speedup rather than add to it. Cheng et al.’s 3.65x includes a factor of 1.89x contributed by the generic z_{bkb}/x bit-manipulation extension before any Ascon-specific hardware. Only the residual 1.93x is attributable to their ISE. The Bader [2] and CHIMERA-CR [12] rows report the same accelerator at two different benchmark points and should be read as one comparator rather than two.

4.6.2 Context-Only Comparison

The works in this section fall into **three** groups that share a common property. Their reported numbers are not directly comparable to this work’s cycle counts or speedups.

The first group implements ASCON as a standalone hardware coprocessor, without a RISC-V host, firmware overhead, or memory-interface latency, for example Khairallah and Yadhunathan [21], Athanasiou et al. [31], the seven stage pipelined design of Bhuvaneshwari et al. [32], a re-configurable single core design supporting five AEAD / Hash variants, the throughput optimized design of Malal [33], a split state design optimized

purely for maximum clock frequency, Gross et al.'s [34] family of application tuned ASIC designs, Wei et al.'s [20] reconfigurable RECO-HCON processor, Khan et al.'s [35] round parallel Ascon-Hash design, Elsadek et al.'s [36] unified methodology energy evaluation of all ten NIST LWC finalists, Elsadek and Tawfik's [37] programmable multi-variant core, and Munir et al.'s [13] design space exploration of the complete NIST standard ASCON family.

The second group integrates ASCON with a RISC-V host but for a different purpose or through a different attachment style than the CV-X-IF instruction level acceleration this thesis focuses on. A transparent memory encryption engine on the AXI bus of a CVA6 SoC [38], and Nguyen et al.'s [15] ASCON peripheral attached to a SERV-core SoC via the Wishbone bus, the closest published analogue in this literature to a memory mapped attachment style.

The third group integrates ASCON into a specific system or application context rather than evaluating it as a general purpose accelerator. A real time video encryption application on two PYNQ boards, a medical IoT (IoMT) encryption design by Raj and Bodapati [39], and an IPSec/ESP network security core by Tran et al. [40].

Three factors make these numbers non-comparable to Table 4.22. They are standalone coprocessors or bus-attached peripherals with no instruction-level coupling, so they carry none of the cost of a processor fetching and retiring accelerator instructions. They report peak streaming throughput rather than the end-to-end latency measured here and throughput in Mbps scales with clock frequency, so a design at several hundred MHz reports proportionally more than a 60 MHz one at equal cycle efficiency.

The defensible thing to draw is therefore an area bound rather than a throughput figure. Across this group the reported footprints run from roughly 1,300 LUTs up to more than 22,000 on FPGA, and from 2.6 kGE to 25 kGE on ASIC, every one of them well above this accelerator's 597 LUTs which is precisely the trade the tightly coupled CV-X-IF design of

Section 3.3.1 set out to make. Nguyen et al. [15] stands out on evidence quality rather than area, being among the few verified on fabricated silicon. Cagua et al. [41] is excluded on the same grounds as Gewehr et al., being a software feasibility study on Raspberry Pi hardware with no synthesis and no RISC-V integration.

4.6.3 Survey and Methodology References

Three survey papers inform the metric choices used throughout this section without serving as numeric comparison targets, since a survey aggregates other authors' implementation numbers rather than reporting their own. Table 4.23 summarizes their scope and why each is cited here.

Table 4.23: Survey and Methodology References

Survey	Why cited here
Kaur et al. [26]	ASCON-specific FPGA and ASIC survey; establishes area, power, throughput, energy and efficiency as the standard reporting vocabulary used in this comparison
Konstantopoulou et al. [27]	Covers all ten NIST LWC finalists; introduces the Throughput/Area convention used when positioning this work against the standalone literature
Mirigaldi et al. [28]	Throughput-versus-area framework across roughly 40 implementations; independently validates the RISC-V comparison cluster of Section 4.6.1

4.6.3.1 A Note on Power and Energy Comparison

Several works in Section 4.6.2, and Moura [18], report energy efficiency as a headline result. This thesis reports none, and the reason is worth setting out rather than leaving as an omission.

The power figures available here come from a vectorless Vivado estimate at Medium overall confidence, not from a bench measurement. Three properties of that estimate, all recorded in the report itself and reproduced in Appendix C.2, jointly prevent an energy result. **First**, it does not separate the accelerator from the core: the hierarchical breakdown resolves only as far as `cv32e40x_system_wrap_0` at 0.021 W, so no figure for

`ascon_permutation.sv` alone exists. **Second**, and decisively, it is a single static estimate that is independent of which firmware runs. The report lists its simulation activity file as absent and its matched design nets as not applicable, meaning no switching activity from either the SW or the HW binary informed it. Multiplying one mode-independent power figure by the two measured cycle counts therefore yields an energy ratio that is arithmetically identical to the speedup. The resulting “energy gain” would be the speedup restated in different units, not independent evidence, which is why Table 4.22 leaves that cell empty rather than filling it with a caveated number. **Third**, the estimate’s environment settings are Vivado defaults, including 250 LFM forced airflow, which the deployed board does not have.

Moura [18] avoids the second problem by deriving energy from each mode’s measured switching activity, which is why that work’s energy and performance figures differ from one another rather than tracking exactly. Obtaining a comparable result here would require a post-implementation simulation of one SW and one HW pass, a switching activity file from each, and a re-run of `report_power` against them. Section 5.4 records this as the first step any energy characterization of this design would have to take. A board level measurement would not substitute for it. The PYNQ-Z1 exposes no programmable logic supply rail for instrumentation, so a shunt on the board input would capture the ARM processing system and DDR interface as well, and those account for roughly 88% of the estimated total.

4.7 Chapter Summary

Across all 28 measurement points and 56 firmware runs, every KAT check passed, ModelSim and FPGA cycle counts agreed to within 2%, and the design closed timing at 60 MHz with +2.145 ns of margin using 597 LUTs (9.6% of the SoC’s LUT usage) and zero BRAM/DSP for the accelerator itself. Measured speedups ranged from 2.98x on PRFShort, whose single permutation call leaves the CV-X-IF handshake nothing to amortise against,

up to 21.4x on Hash256 at 1024B, with every family's profile fully explained by its absorb rate and phase structure, answering both research questions. The accelerator's area and timing cost is small and precisely characterized, its power cost is bounded but estimated rather than measured, and a single shared permutation core serves all seven ASCON family members correctly with performance differences attributable entirely to their NIST specified structural differences.

CHAPTER 5

CONCLUSIONS

5.1 Summary of Findings

5.1.1 RQ1: PPA of CV-X-IF Accelerator

RQ1 asked what a CV-X-IF tightly coupled Ascon permutation accelerator costs in performance, area and power when implemented on a real FPGA and driven by RISC-V firmware. The answer, established in Chapter 4, is concrete. The accelerator occupies 597 LUTs, 368 flip-flops, zero DSP blocks and zero BRAM primitives on a Xilinx Zynq-7000 xc7z020 [10] device, and closes timing at 60 MHz with a Worst Negative Slack of +2.145 ns, implying a theoretical ceiling near 68.9 MHz and therefore headroom above the measured operating point.

The power third of RQ1 admits a weaker answer than the other two, and the difference is one of evidence rather than of magnitude. Total on-chip power is estimated at 1.723 W, of which 1.520 W is the ARM processing system and only 0.063 W the whole programmable logic; the core-and-accelerator instance draws an estimated 0.021 W, roughly 1.2% of the total. Those figures come from a vectorless `report_power` estimate at Medium confidence, not from measurement, and the hierarchy does not resolve the accelerator separately from the core. The honest answer to RQ1 is therefore that the accelerator’s area and timing costs are measured and small, while its power cost is bounded and small but estimated. Section 4.6.3.1 sets out why no energy figure is claimed at all.

The zero-BRAM and zero-DSP results follow directly from the register

file coupled ISE decision of Section 3.3.1. The Ascon state travels entirely through the CV32E40X register file rather than through dedicated memory, and the permutation’s substitution and diffusion layers require only bitwise Boolean operations, never multiplication. Measured on FPGA hardware rather than estimated from simulation, the accelerator delivers speedups from 2.98x on the single-call Ascon-PRFShort to 21.37x on Ascon-Hash256 at 1024 bytes, with the asymptotic per-call ratio approaching roughly 23x as the absorb loop comes to dominate runtime.

5.1.2 RQ2: How the Benefit

RQ2 asked how the benefit of a single shared permutation accelerator is distributed across the seven family members, and what determines that distribution. That one accelerator can serve them all is confirmed and is the least interesting part of the answer. Ascon-AEAD128, Ascon-Hash256, Ascon-XOF128, Ascon-CXOF128, Ascon-MAC, Ascon-PRF and Ascon-PRFShort share the identical `ascon_permutation.sv` module and the identical bitstream, differing only in their firmware drivers, and every member passed NIST Known Answer Test [11] verification at every one of the 28 measurement points. The substantive answer is that accelerating a shared primitive does *not* deliver a shared benefit.

At a common 16-byte message length the seven members range from 3.41x to 11.12x, a spread of more than three to one from identical hardware. The ordering by speedup also inverts against the ordering by absolute cost. At 1024 bytes Ascon-MAC runs at 5.5 hardware cycles per byte against Ascon-Hash256’s 8.8, and is therefore the faster of the two, while showing less than half the speedup. A designer reading only the authenticated-encryption and hashing literature would draw the opposite conclusion about which mode this accelerator serves best.

What governs the distribution is not the permutation, which is common to all seven, but how much non-permutation work accompanies each

permutation call. Two independent measurements establish this. In Ascon-AEAD128 an associated-data block, being a load, an exclusive-or and a permutation, is accelerated by more than 10x, whereas a plaintext block, which must additionally store ciphertext and lengthen the output buffer, is accelerated by 3.92x. The two invoke the same hardware and differ only in what surrounds it. In Ascon-PRFShort, whose call count is fixed by construction at one, the software and hardware cycle counts move by the same absolute amount at every input length, which identifies the whole of its length dependence as work the accelerator never touches and explains why it records the lowest speedup measured anywhere in this study.

Absorption rate remains the useful first-order predictor, and the per-member structural properties of Section 4.4 still account for the broad ordering. But rate alone is incomplete, because it counts permutation calls as though they were interchangeable, and the measurements show they are not.

Taken together the two findings support one conclusion from two directions. A single, standards compliant, register file coupled accelerator is an efficient way to accelerate the complete Ascon family on a RISC-V core (RQ1), and the benefit it delivers to any given member is bounded by that member's ratio of permutation to surrounding work rather than by the accelerator (RQ2). The second is the more useful result for anyone selecting an Ascon mode for a constrained device.

5.2 Contributions

The five contributions set out in Section 1.5 can now be restated as established results rather than intentions. The accelerator itself is not among them, since CV-X-IF and register-file coupled Ascon accelerators already exist and the round function is fixed by the standard.

Coverage. Ascon-MAC, Ascon-PRF and Ascon-PRFShort have been implemented and measured on a RISC-V core for the first time, and the algo-

rithms measured are those of NIST SP 800-232 rather than the CAESAR-era v1.2 specification implemented by every comparable RISC-V work. This is the first evaluation coupling the standardised Ascon family to a general purpose processor through instruction issue.

Methodology. The software baseline and the accelerated run were executed on the same placed and routed bitstream, at the same clock, through the same memory subsystem, differing only in the linked permutation assembly and one control-register bit; stall counts identical between the two modes at 24 of 28 points confirm the isolation. Every comparable work measures its baseline on separately synthesized hardware. A hardware counter freeze mechanism made the cycle counts independent of host polling latency, which is what allows cycle-accurate measurement on a deployed device rather than in simulation.

Organisation. Carrying the round count in the instruction encoding and holding the state in the datapath’s own flip-flops, rather than in storage dedicated to it, produced an accelerator of 597 LUTs and 368 flip-flops against the 1,321 LUTs and 810 registers of the nearest comparable design at the same design point through the same interface.

Finding. Accelerating a shared primitive does not deliver a shared benefit. The seven members span more than three to one at a common message length, and what bounds each member’s benefit is the non-permutation work accompanying its permutation calls, established independently by the associated-data and PRFShort measurements.

Artefacts and positioning. Thirteen hand-written RV32I assembly implementations are original to this work, closing an absence that Steinegger and Primas [9] name as their own reason for benchmarking against compiled C, and making the baseline used here tighter than any in the comparable literature. The results are positioned against that literature without being ranked against it. The prior CV-X-IF studies measure v1.2 against compiled C in simulation, so their ratios are not commensurable in either direction, and what the comparison establishes is that Bader [2]

independently identifies the same CV-X-IF per-call overhead that Chapter 4 quantifies.

One qualification applies throughout. All *performance* results reported here come from a synthesized, placed and routed bitstream running on a PYNQ-Z1 board at 60 MHz. Area and timing are post-implementation report values obtained by static analysis of the routed netlist and power is a tool estimate, as Section 4.6.3.1 sets out in full. The distinction is drawn wherever a figure appears, and no power or energy result is claimed as measured.

Collectively these contributions close the gap identified in Section 2.8. The absence, until this work, of the standardised Ascon family measured on deployed hardware, through instruction issue, on a general purpose RISC-V core.

5.3 Limitations and Constraints

This work has five limitations worth stating plainly.

First, all measurements were taken at a single clock frequency, 60 MHz, on a single FPGA family. Although the timing analysis implies a ceiling near 68.9 MHz, no result here empirically demonstrates performance, power or timing behaviour at a higher frequency or on a different device family, so generalization beyond the PYNQ-Z1 rests on the RTL's portability rather than on further measurement.

Second, this thesis does not evaluate resistance to side channel or fault injection attacks. Section 2.6 reviewed published power analysis, template attack and persistent fault work targeting Ascon hardware and placed that class of question outside the scope of **RQ1** and **RQ2**. The accelerator presented here is an unprotected implementation, and its physical attack resistance is unknown.

Third, the decision to use a CV-X-IF instruction set extension rather than a memory mapped or DMA coupled accelerator, argued in Section 3.3.1

on the basis of BRAM usage, stall cycle contribution and compliance with Marshall et al.’s ISE design principles, was never built and measured as an alternative. That comparison is therefore a design rationale argument rather than an empirical one.

Fourth, message length coverage is uneven across family members. Ascon-AEAD128 is limited to two plaintext lengths because both axes of its KAT file’s plaintext / associated-data grid stop at 32 bytes; Ascon-CXOF128 likewise tops out at 32 bytes, and Ascon-PRFShort at 16 bytes by construction, since PRFShort accepts no longer input. Only Hash256, XOF128, MAC and PRF reach 1024 bytes. This limitation, justified on KAT availability grounds in Section 3.8.3, means the asymptotic behaviour of the other three is extrapolated from a narrower range.

That constraint should not be confused with the associated-data convention, which was a choice rather than a limit. Nonzero associated-data vectors exist at every plaintext length in the KAT grid, and Section 4.4.2.1 reports AEAD128 under both the $AD = 0$ convention of Steinegger and Primas [9] and the $AD = |PT|$ convention of Cheng et al. [14]. The associated-data absorption path is therefore exercised and KAT-verified on hardware, not merely compiled. What remains uncharacterised is its behaviour beyond 32 bytes, where no official vector exists.

Fifth, the power results are estimated rather than measured. The `report_power` run behind them records no simulation activity file and no matched design nets, so internal switching activity is modelled from tool defaults. Its overall confidence is Medium, limited by that one input class. Its hierarchy stops at the core plus accelerator instance, so the accelerator is not separately resolved and its thermal model assumes 250 LFM forced airflow, which the deployed board does not have. One consequence is decisive because the estimate is identical whichever firmware runs, any energy figure formed from it would reproduce the measured speedup rather than add to it, which is why none is reported. This is a limitation of evidence, not a claim that the figures are wrong.

5.4 Future Work

Six directions follow directly from those limitations.

First, synthesizing the same RTL for an ASIC flow would permit direct numeric comparison against the standalone ASIC designs reviewed in Section 2.5, rather than the area efficiency only comparison this thesis was limited to.

Second, building and measuring the memory mapped or DMA coupled alternative would convert the design rationale argument of Section 3.3.1 into an empirical one, replacing engineering estimates of BRAM usage and stall cycle cost with hardware measurements.

Third, extending the accelerator with side channel or fault injection countermeasures, following the threshold implementation and masking approaches surveyed in Section 2.6, would allow this design to be evaluated for physical attack resistance, a question any real deployment must eventually answer.

Fourth, characterizing the design at higher clock frequencies, up to the 68.9 MHz ceiling implied by its own timing margin, and on additional FPGA families, would test whether the speedup and CPI relationships established in Chapter 4 hold at other operating points or are specific to the configuration measured here.

Fifth, exploring a pipelined or multi-round-per-cycle permutation datapath, in place of the current single-round-per-cycle iterative design, would trade LUT area for throughput and extend the performance-area trade-off space this work occupies rather than replacing it.

Sixth, characterizing power from measured switching activity rather than tool defaults would convert the weakest evidence in this thesis into a result. A post-implementation timing simulation of one software and one hardware pass, a switching activity file captured from each, and a re-run of `report_power` against them would raise the internal-node activity input from Medium to High confidence and, decisively, yield two *different* power figures for the two modes, so that an energy comparison would no longer

be the speedup restated. Preserving the accelerator’s hierarchy through synthesis would additionally resolve it separately from the core. This is the first step any energy characterization of this design would have to take, and it requires neither new hardware nor board time. A board-level measurement would not substitute for it. The PYNQ-Z1 exposes no programmable logic supply rail for instrumentation, so a shunt on the board input would capture the processing system and DDR interface as well, and those account for roughly 88% of the estimated total.

5.5 Concluding Remarks

This thesis began from a simple observation, as Ascon becomes the standard cryptographic primitive for resource constrained devices, understanding what it costs to accelerate in hardware, and whether that cost can be paid once rather than once per algorithm, is a practical question with a measurable answer. The work presented here shows that a single, standards compliant, register file coupled accelerator, built using an interface already available on an unmodified CV32E40X core, can serve the complete seven member Ascon family at an area cost of under 600 LUTs, with silicon measured speedups that are not uniform across it. What each member gains is set by how much non-permutation work accompanies its permutation calls. For a lightweight cryptography standard whose stated purpose is efficient security for constrained IoT devices, an accelerator this compact, measured this rigorously, is a meaningful step toward making that efficiency real rather than theoretical.

REFERENCES

- [1] Digilent Inc., “PYNQ-Z1 board reference manual,” Digilent Inc., Tech. Rep., 2026. [Online]. Available: <https://digilent.com/reference/programmable-logic/pynq-z1/reference-manual> Cited on pp. [xiii](#), [15](#), [21](#), [22](#), and [42](#).
- [2] Bader, F. Design and Integration of a RISC-V Based Accelerator for ASCON. Master’s Thesis. Politecnico di Torino; 2024. Cited on pp. [xiii](#), [4](#), [6](#), [10](#), [11](#), [12](#), [14](#), [18](#), [19](#), [70](#), [71](#), [72](#), [75](#), [76](#), [78](#), [79](#), and [87](#).
- [3] National Institute of Standards and Technology. Advanced Encryption Standard (AES). FIPS 197-upd1. 2023. Originally published 2001; this 2023 update makes no technical changes to the algorithm, editorial revision only. Cited on pp. [1](#), [4](#), and [37](#).
- [4] Dworkin, M. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. National Institute of Standards and Technology, NIST SP 800-38D. 2007. Cited on pp. [1](#) and [4](#).
- [5] Dobraunig, C., Eichlseder, M., Mendel, F. and Schl  ffer, M. Ascon v1.2: Lightweight Authenticated Encryption and Hashing. Journal of Cryptology. 2021. 34(3): Article 33. Cited on pp. [1](#), [2](#), and [9](#).
- [6] Turan, M. S., McKay, K., Chang, D., Kang, J. and Kelsey, J. Ascon-Based Lightweight Cryptography Standards for Constrained Devices: Authenticated Encryption, Hash, and

- Extendable Output Functions. National Institute of Standards and Technology, NIST SP 800-232. 2025. [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-232.pdf> Cited on pp. 1, 2, 9, and 10.
- [7] Dobraunig, C., Eichlseder, M., Mendel, F. and Schl  ffer, M. Ascon MAC, PRF, and Short-Input PRF: Lightweight, Fast, and Efficient Pseudorandom Functions. Topics in Cryptology – CT-RSA 2024, LNCS, vol. 14643. Springer. 2024. 381–403. Cited on pp. 1, 2, and 10.
- [8] OpenHW Group, “CV32E40X RISC-V core,” GitHub repository, 2026, user manual available at <https://docs.openhwgroup.org/projects/cv32e40x-user-manual/en/latest/intro.html>. Accessed 2026-07-25. [Online]. Available: <https://github.com/openhwgroup/cv32e40x> Cited on pp. 2, 4, 5, and 11.
- [9] Steinegger, S. and Primas, R. A Fast and Compact RISC-V Accelerator for Ascon and Friends. International Conference on Smart Card Research and Advanced Applications. 2020. 53–67. Cited on pp. 4, 6, 8, 10, 11, 18, 52, 59, 72, 77, 87, and 89.
- [10] Xilinx, Inc., “Zynq-7000 SoC data sheet: Overview,” Xilinx, Inc., Tech. Rep. DS190 (v1.11.1), 2018. [Online]. Available: <https://docs.amd.com/v/u/en-US/ds190-Zynq-7000-Overview> Cited on pp. 5, 21, 55, and 84.
- [11] Dobraunig, C., Eichlseder, M., Mendel, F., and Schl  ffer, M., “ascon-c: Ascon – lightweight authenticated encryption & hashing (reference and optimized C/assembly implementations),” GitHub repository, 2026. [Online]. Available: <https://github.com/ascon/ascon-c> Cited on pp. 5, 20, 31, 38, 40, 41, 45, 50, 55, 57, and 85.

- [12] Dolmeta, A., Piscopo, V., Martina, M. and Masera, G. CHIMERA: Cryptographic Hardware for Integrated Multipurpose Engine on RISC-V with ASCON. Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI). 2025. Cited on pp. 6, 7, 10, 12, 14, 15, 18, 70, 75, 78, and 79.
- [13] Munir, M. S., Dordoni, T., Aksoy, L. and Akleylek, S. From Authenticated Encryption to Hash: A Comprehensive Design Space Exploration of the NIST Standard Ascon Family. Proceedings of the Great Lakes Symposium on VLSI 2026 (GLSVLSI). 2026. 998–1005. Cited on pp. 6, 14, 15, 18, and 80.
- [14] Cheng, H., Großschädl, J., Marshall, B., Page, D. and Pham, T. RISC-V Instruction Set Extensions for Lightweight Symmetric Cryptography. IACR Transactions on Cryptographic Hardware and Embedded Systems. 2023. 2023(1): 193–237. Cited on pp. 6, 10, 13, 14, 18, 52, 59, 70, 74, 78, and 89.
- [15] Nguyen, K. D., Dang, T.-K., Kieu-Do-Nguyen, B., Le, D.-H., Pham, C.-K. and Hoang, T.-T. ASIC Implementation of ASCON Lightweight Cryptography for IoT Applications. IEEE Transactions on Circuits and Systems II: Express Briefs. 2025. 72(1): 278–282. Cited on pp. 7, 14, 15, 80, and 81.
- [16] Dobraunig, C., Eichlseder, M., Mendel, F. and Schläffer, M. Ascon v1.2. Submission to Round 3 of the CAESAR Competition. 2016. [Online]. Available: <https://competitions.cr.yp.to/round3/asconv12.pdf> Cited on p. 9.
- [17] Campos, F., Jellema, L., Lemmen, M., Müller, L., Sprenkels, A. and Viguier, B. Assembly or Optimized C for Lightweight Cryptography on RISC-V? Cryptology and Network Security (CANS 2020), LNCS, vol. 12579. Springer. 2020. Cited on p. 10.

- [18] Moura, N. S. Acceleration of AEAD Algorithms for Resource-Constrained Embedded Devices. Master's Thesis. Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS); 2024. Cited on pp. 10, 13, 73, 78, 81, and 82.
- [19] Gewehr, C., Luza, L. and Moraes, F. G. Hardware Acceleration of Crystals-Kyber in Low-Complexity Embedded Systems with RISC-V Instruction Set Extensions. IEEE Access. 2024. 12: 94477–94495. Cited on pp. 13 and 77.
- [20] Wei, X., El-Hadedy, M., Mosanu, S., Zhu, Z., Hwu, W.-M. and Guo, X. RECO-HCON: A High-Throughput Reconfigurable Compact ASCON Processor for Trusted IoT. International System on Chip Conference (SOCC). 2022. Cited on pp. 15 and 80.
- [21] Khairallah, M. and Yadhunathan, S. FPGA Implementations of Message Authentication Codes Based on Ascon-p. NIST Lightweight Cryptography Workshop 2023. 2023. Cited on pp. 15 and 79.
- [22] Ramadass, P. and Paramasivam, S. An Optimized Hardware Implementation of ASCON With Power Analysis Attack Resistance. International Journal of Circuit Theory and Applications. 2025. 54(5): 2864–2880. Cited on p. 16.
- [23] You, S.-C., Kuhn, M. G., Sarkar, S. and Hao, F. Low Trace-Count Template Attacks on 32-Bit Implementations of ASCON AEAD. IACR Transactions on Cryptographic Hardware and Embedded Systems. 2023. 2023(4): 344–366. Cited on p. 16.
- [24] Das, M. and Mazumdar, B. Security Analysis of ASCON Cipher Under Persistent Faults. Security, Privacy, and Applied Cryptography Engineering (SPACE 2024), LNCS, vol. 15351. Springer. 2024. Cited on p. 16.

- [25] Saravanan, P., Parthasarathy, R., Sona, B. and Fathima, N. A. A. Formal Verification of ASCON Cipher Using JasperGold. 2024 International Conference on Smart Electronics and Communication Systems (ISENSE). IEEE. 2024. 1–6. Cited on p. 16.
- [26] Kaur, J., Cintas Canto, A., Mozaffari Kermani, M. and Azarderakhsh, R. A Survey on the Implementations, Attacks, and Countermeasures of the NIST Lightweight Cryptography Standard: ASCON. ACM Computing Surveys. 2025. 58(1): Article 7. Cited on pp. 17 and 81.
- [27] Konstantopoulou, E., Athanasiou, G. and Sklavos, N. Review and Analysis of FPGA and ASIC Implementations of NIST Lightweight Cryptography Finalists. ACM Computing Surveys. 2025. 57(10): Article 246. Cited on pp. 17 and 81.
- [28] Mirigaldi, M., Piscopo, V., Martina, M. and Masera, G. The Quest for Efficient ASCON Implementations: A Comprehensive Review of Implementation Strategies and Challenges. Chips. 2025. 4(2): 15. Cited on pp. 17 and 81.
- [29] Marshall, B., Newell, G. R., Page, D., Saarinen, M.-J. O. and Wolf, C. The Design of Scalar AES Instruction Set Extensions for RISC-V. IACR Transactions on Cryptographic Hardware and Embedded Systems. 2021. 109–136. Cited on pp. 20 and 37.
- [30] Baldit, T., Dehez-Clementi, M. and Dion, A. Hardware Optimization of Cryptographic Algorithms on RISC-V for Embedded Applications. IEEE Open Journal of the Solid-State Circuits Society. 2026. Cited on pp. 77 and 78.
- [31] Athanasiou, G. S., Boufeas, D. and Konstantopoulou, E. A Robust ASCON Cryptographic Coprocessor for Secure IoT Applications. 2024 Panhellenic Conference on Electronics and Telecommunications (PACET). 2024. Cited on p. 79.

- [32] Bhuvaneshwari, A. J., Kaythry, P., Kumar, K. J. J. and Sachin, D. 7-Stage Pipelined Architecture of ASCON for Resource Constraint Devices. *IEEE Embedded Systems Letters*. 2025. Cited on p. 79.
- [33] Malal, A. High-Performance FPGA Implementations of Lightweight ASCON-128 and ASCON-128a with Enhanced Throughput-to-Area Efficiency. 2024 17th International Conference on Information Security and Cryptology (ISCTürkiye). 2024. Also available as IACR ePrint 2025/825. Conference-year (2024) used here. Cited on p. 79.
- [34] Groß, H., Wenger, E., Dobraunig, C. and Ehrenhöfer, C. Ascon Hardware Implementations and Side-Channel Evaluation. *Microprocessors and Microsystems*. 2017. 52: 470–479. Cited on p. 80.
- [35] Khan, S., Lee, W.-K., Karmakar, A., Mera, J. M. B., Majeed, A. and Hwang, S. O. Area–Time Efficient Implementation of NIST Lightweight Hash Functions Targeting IoT Applications. *IEEE Internet of Things Journal*. 2022. 10(9): 8083–8095. Cited on p. 80.
- [36] Elsadek, I., Aftabjahani, S., Gardner, D., MacLean, E., Wallrabenstein, J. R. and Tawfik, E. Y. Hardware and Energy Efficiency Evaluation of NIST Lightweight Cryptography Standardization Finalists. 2022 IEEE International Symposium on Circuits and Systems (ISCAS). 2022. 133–137. Cited on p. 80.
- [37] Elsadek, I. and Tawfik, E. Optimized and Reconfigurable Hardware Design for ASCON AEAD and Hash Standards. 2025 IEEE International Symposium on Circuits and Systems (ISCAS). London, UK. 2025. 1–5. Cited on p. 80.

- [38] Xu, D., Wang, X., Hao, Q., Wang, J., Cui, S. and Liu, B. A High-Performance Transparent Memory Data Encryption and Authentication Scheme Based on Ascon Cipher. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. 2024. 32(5): 925–937. Cited on p. 80.
- [39] Raj, K. and Bodapati, S. FPGA Based Light Weight Encryption of Medical Data for IoMT Devices Using Ascon Cipher. 2022 IEEE International Symposium on Smart Electronic Systems (ISES). 2022. 196–201. Cited on p. 80.
- [40] Tran, S. N., Hoang, V. T. and Bui, D. H. A Hardware Architecture of NIST Lightweight Cryptography Applied in IPSec to Secure High-Throughput Low-Latency IoT Networks. *IEEE Access*. 2023. Cited on p. 80.
- [41] Cagua, G., Gauthier-Umaña, V. and Lozano-Garzon, C. Implementation and Performance of Lightweight Authentication Encryption ASCON on IoT Devices. *IEEE Access*. 2025. 13: 16671–16682. Cited on p. 81.

Syed Khawar shah

CRYPTOGRAPHY ON RISC-V SYSTEMS

Document Details

Submission ID

trn:oid::3618:153501627

Submission Date

Oct 2, 2026, 7:47 AM GMT+0

Download Date

Oct 2, 2026, 7:52 AM GMT+0

File Name

CRYPTOGRAPHY ON RISC-V SYSTEMS.pdf

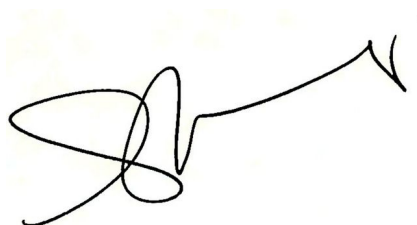
File Size

2.8 MB

143 Pages

35,387 Words

178,682 Characters



3% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Submitted works

Match Groups

-  **67 Not Cited or Quoted 3%**
Matches with neither in-text citation nor quotation marks
-  **23 Missing Quotations 1%**
Matches that are still very similar to source material
-  **1 Missing Citation 0%**
Matches that have quotation marks, but no in-text citation
-  **2 Cited and Quoted 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

- 3%  Internet sources
- 2%  Publications
- 0%  Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

APPENDIX A

INSTRUCTION SET, REGISTERS AND SOURCE LISTINGS

This appendix is the design reference for the accelerator. The custom instruction encoding and the sequence that drives one permutation call, the AXI-Lite control and status registers, and short excerpts of the source behind specific claims made in Chapter 3. Full source files are retained in the project repository.

A.1 Custom Instruction Encoding and Sequencing

The accelerator attaches through the CV-X-IF interface, of whose six handshake channels the design drives only Issue and Result; the remaining four are tied off.

All ASCON instructions use the RISC-V `custom-0` opcode (0x0B), one of four spaces the ISA guarantees no standard extension will claim since it is not decoded internally, the core defers entirely to the coprocessor's `accept` signal. Every instruction uses the standard R-type format of Table A.1.

Table A.1: R-Type Instruction Format

Bits	[31:25]	[24:20]	[19:15]	[14:12]	[11:7]	[6:0]
Field	funct7	rs2	rs1	funct3	rd	opcode
ASCON usage	class + round count	state word low (WS) / x0 (RS)	state word high (WS) / x0 (RS)	lane 0–4	x0 (WS) / dest reg (RS)	0x0B

Six instruction variants exist across two classes, listed in Table A.2.

Table A.2: ASCON ISE Instruction Taxonomy

Mnemonic	funct7	funct3	Behaviour
ASCON_WS	0x00	0–3	Latch state word S_i ; no trigger
ASCON_WS_P6/P8/P12	0x06/0x08/0x0C	4	Latch S_4 , then trigger $p[6]/p[8]/p[12]$
ASCON_RS_H	0x01	0–4	Read high 32 bits of $S[\text{funct3}]$ to rd
ASCON_RS_L	0x02	0–4	Read low 32 bits of $S[\text{funct3}]$ to rd

The lower four bits of funct7 encode the round count directly ($\text{dec_rounds} = \text{dec_funct7}[3:0]$), so the decoder needs no case statement, and a future round count would require only a new firmware constant. The trigger is restricted to lane 4 by design. This is the mechanism that guarantees all five state words are loaded before a permutation can start. The `rs1/rs2` fields are not decoded from the instruction word. They arrive as the live register contents forwarded by the core on the CV-X-IF issue channel, unlike `funct7/funct3/rd`, which are literal bit fields.

A single permutation call is issued as a fixed 15-instruction sequence. Five `ASCON_WS` instructions load the 320-bit state two 32-bit halves at a time, with lane index in `funct3` and the register pair supplying the high and low halves. The fifth, on lane 4, carries the trigger `funct7` (0x0C for $p[12]$) and starts the permutation. The CPU then stalls automatically through CV-X-IF until the rounds finish, so no explicit polling is needed. Ten `ASCON_RS` instructions read the result back, alternating high and low halves across the five lanes, each naming its destination register directly in `rd`.

Family members that use only $p[12]$ throughout (Hash256, XOF128, MAC, PRF, PRFShort) issue this exact sequence at every permutation call. AEAD128 additionally uses the $p[8]$ trigger (`funct7=0x08`) in place of `ASCON_WS_P12` on line 5 for its associated-data and plaintext duplexing rounds. The remaining fourteen instructions are unchanged.

A.2 AXI-Lite Control and Status Registers

Software on the Zynq PS side (Python, running under Jupyter on PYNQ Linux) controls and monitors the CV32E40X system and the accelerator through a single AXI-Lite slave, `accel_ctrl_regs`, mapped at base address `0x40020000`. Table A.3 lists every defined offset.

Table A.3: AXI Lite Control

Offset	Register	Access	Description
0x00	CONTROL	R/W	System lifecycle control; see Table A.4
0x04	STATUS	R	<code>accel_busy</code> , <code>accel_done</code>
0x08	PC	R	Current instruction fetch address
0x0C	CONFIG	R/W	Accelerator configuration (reserved)
0x10	CYCLE_COUNT	R	Primary timing metric; frozen by <code>halt_count</code> on completion
0x14	INSTR_COUNT	R	Instructions retired
0x18	STALL_COUNT	R	Pipeline stall cycles
0x20	COUNTER_CTRL	W	Write 0x01 to pulse <code>counter_reset</code> (self-clearing)
0x30	ACCEL_OPS	R	Permutation calls completed
0x34	ACCEL_CYCLES	R	Cycles spent in the <code>ST_PERMUTE</code> FSM state

The CONTROL register (offset 0x00) is the only register whose individual bits require separate documentation. Table A.4 lists them as observed from the software side, i.e. the effect a Python register write actually has.

Table A.4: CONTROL Register

Bit	Name	Function
0	<code>core_reset_reg</code>	Write 1 to hold the CV32E40X core in reset; 0 to release it.
1	<code>fetch_enable</code>	Write 1 to allow instruction fetch. Direct mapping.
2	<code>accel_enable</code>	Write 1 to select hardware (CV-X-IF) mode; 0 selects software-only. Direct mapping.
3	<code>accel_reset_reg</code>	Write 1 to release the permutation FSM from reset; 0 holds it. See note below.

Valid write values

0x01	0001	Hold core in reset. Safe state for loading IMEM/DMEM over AXI.
0x02	0010	Software-only mode: core running, fetch enabled, accelerator and FSM disabled.
0x0E	1110	Hardware mode: core running, fetch enabled, accelerator and FSM released.

Note on bit 3. Bit 3 passes through two inversions before reaching the permutation core's reset input, and they cancel. `accel_ctrl_regs.sv` computes `accel_reset = ~control_reg[3]`, and `accel_wrapper.sv` then forms the reset as `rst_n & ~accel_reset`. Each file labels its own intermediate signal active-low, which makes bit 3 look inverted if only one is consulted. Tracing through both confirms it behaves as a direct enable.

Warning. `0x04` and `0x06` are *not* valid substitutes for `0x0E` in hardware mode. Both leave bit 3 at 0, which holds the permutation FSM in reset while `accel_enable` (bit 2) is already set. The core then stalls indefinitely at the first `ASCON_WS` instruction. This was observed during bring-up as a fixed instruction-count freeze at `ic=176`. Bit 3 must be set to 1 whenever bit 2 is set.

A.3 The ASCON_HW_PERM Macro

Listing A.1 is the macro issuing that 15-instruction sequence, shown for lane 0 and the lane-4 trigger only. The other lanes differ solely in the `funct3` index and destination register. The state aliases `x0h–x4l` map to `s0–s9`, and the staging pair to `t3/t4`, which the `.word` encoding hard-codes as `rs1` and `rs2`.

Listing A.1: `ASCON_HW_PERM` macro (`ascon_core_accel.S`), lane 0 and trigger shown

```
.macro ASCON_HW_PERM trigger_funct7

    /* WRITE PHASE: 5 x ASCON_WS, one per lane */
    mv     t0h, x0h                /* stage S0 high into x28 */
    mv     t0l, x0l                /* stage S0 low into x29 */
    .word  ((0x00<<25)|(29<<20)|(28<<15)|(0<<12)|(0<<7)|0x0B)
    /* lanes 1-3 identical, funct3 = 1, 2, 3 */

    mv     t0h, x4h
    mv     t0l, x4l
    .word  ((funct7<<25)|(29<<20)|(28<<15)|(4<<12)|(0<<7)|0x0B)
    /* lane 4 carries the trigger; CPU stalls here until done */
```

```

/* READ PHASE: 10 x ASCON_RS, rd names the final register */
.word ((0x01<<25)|(0<<20)|(0<<15)|(0<<12)|(8<<7)|0x0B)
.word ((0x02<<25)|(0<<20)|(0<<15)|(0<<12)|(9<<7)|0x0B)
/* lanes 1-4 follow the same pair, funct3 = 1..4 */
.endm

```

All seven hardware variants invoke this macro with only the trigger argument changing.

A.4 Instruction Decode and Logic

The combinational instruction decoder in `ascon_permutation.sv` is the hardware counterpart to Table A.2. It classifies every incoming CV-X-IF instruction into one of the six variants from `funct7` and `funct3` alone, with no case statement, recovering the round count directly from the low four bits of `funct7`. Two details are worth pointing out that are easy to miss reading the encoding table alone. First, an `is_custom0` term gates every other classification, so any instruction that is not RISC-V `custom-0` is decoded as none of the six ASCON variants regardless of its other fields, consistent with the core deferring entirely to the accelerator’s `accept` signal (Section A.1). Second, the round count is derived combinatorially from `funct7[3:0]` rather than looked up, which is why the taxonomy’s three write-trigger variants (`0x06/0x08/0x0C`) require no dedicated round-count logic elsewhere in the design.

The read-side address decode in `accel_ctrl_regs.sv` is the hardware counterpart to Table A.3. Every offset in that table corresponds to one arm of a single case statement on `s_axi_araddr[7:0]`. Two things follow that are worth stating explicitly rather than left implicit. First, STATUS (offset `0x04`) is not a stored register at all. It is assembled on every read from the live `accel_busy/accel_done` signals, so it always reflects the accelerator’s current state rather than a value that could go stale between reads. Second, any offset not listed there, including the unused range between `0x18` and `0x30`, reads back as zero rather than triggering

an AXI error response, consistent with `accel_ctrl_regs.sv`'s own documentation. No error responses are generated by this slave.

As an aside, this file's own register-map comment block states CONTROL bit 3 as "1 = release accel from reset, 0 = hold accel in reset," which is the same behaviour independently derived by tracing the double inversion through `accel_ctrl_regs.sv` and `accel_wrapper.sv` in the note under Table A.4. The two agreeing, one from reading the code's own documentation and one from tracing the actual logic, is additional confidence that the resolution reached there is correct.

The accept and writeback conditions in `accel_wrapper.sv` are two combinational assignments. Accept requires `issue_valid`, `issue_ready` and a `custom=0` opcode together, so no instruction is taken while a permutation is still running. Writeback deliberately omits `issue_valid` and tests only the opcode and an `ASCON_RS funct7`, since it declares a property of the instruction type whether this opcode ever returns a value rather than a live handshake state.

A.5 Permutation FSM

Figure A.1 is the main state machine from `ascon_permutation.sv`, whose states and transitions are described in Section 3.3.4 and summarised in Table 3.7. Note in particular that `ST_RESULT` can transition straight to `ST_LOAD` or `ST_PERMUTE` without passing through `ST_IDLE`, which is what allows back-to-back permutation calls with zero idle cycles, and that `op_complete` is derived as a single-cycle pulse via a `prev_done` term so that `accel_ops` counts one per call rather than one per `ST_RESULT` cycle.

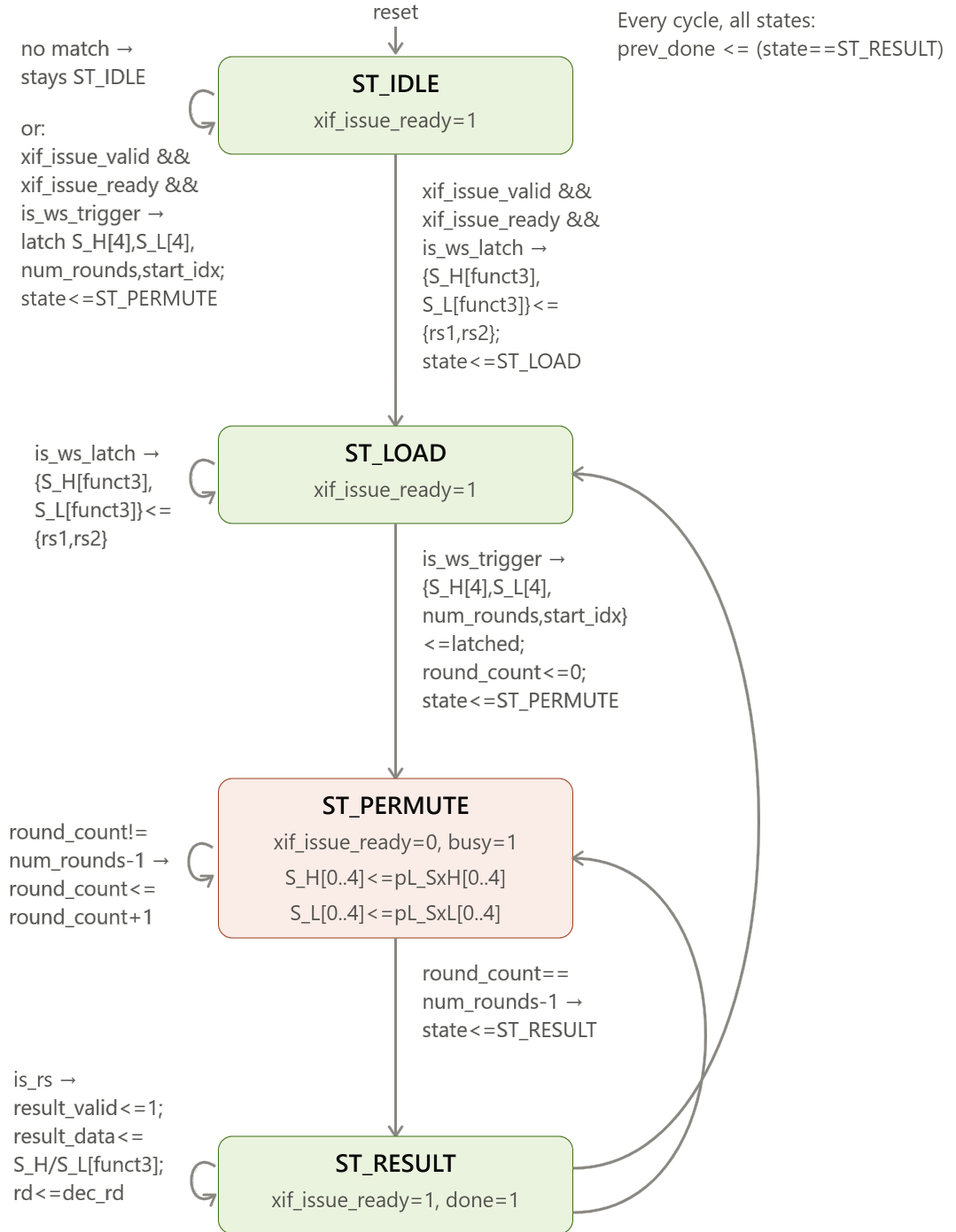


Figure A.1: Finite State Machine

APPENDIX B

VERIFICATION AND BENCHMARK

This appendix holds the correctness evidence and the raw measurement record behind Chapter 4. KAT coverage for all seven family members, the testbench results, and the cycle, instruction and stall counts for both passes at every one of the 28 measurement points. Every value is taken directly from that algorithm's results log; the derived metrics appear per family in Chapter 4 and follow from these counts by the formulas of Table 4.4.

B.1 KAT Coverage

Every measurement point is checked against a NIST Known Answer Test vector before its cycle counts are accepted. Table B.1 gives, per algorithm and length, the vector's Count and the number of permutation calls issued.

Table B.1: KAT Coverage per Measurement Point

Algorithm	Length	KAT Count	Permutation Calls
AEAD128	16 B, AD=0	529	$3 (1 \times p8 + 2 \times p12)$
AEAD128	16 B, AD=16 B	545	$5 (3 \times p8 + 2 \times p12)$
AEAD128	32 B, AD=0	1057	$4 (2 \times p8 + 2 \times p12)$

Algorithm	Length	KAT Count	Permutation Calls
AEAD128	32 B, AD=32 B	1089	7 ($5 \times p8 + 2 \times p12$)
Hash256	16 B	17	6 (2 absorb + 4 squeeze)
Hash256	32 B	33	8 (4 absorb + 4 squeeze)
Hash256	64 B	65	12 (8 absorb + 4 squeeze)
Hash256	1024 B	1025	132 (128 absorb + 4 squeeze)
XOF128	16 B	17	10 (2 absorb + 8 squeeze)
XOF128	32 B	33	12 (4 absorb + 8 squeeze)
XOF128	64 B	65	16 (8 absorb + 8 squeeze)
XOF128	1024 B	1025	136 (128 absorb + 8 squeeze)
CXOF128	0 B	17	12
CXOF128	8 B	281	13
CXOF128	16 B	545	14
CXOF128	32 B	1073	16
MAC	16 B	17	2
MAC	32 B	33	3
MAC	64 B	65	4
MAC	1024 B	1025	34
PRF	16 B	17	5
PRF	32 B	33	6
PRF	64 B	65	7
PRF	1024 B	1025	37
PRFShort	0 B	1	1
PRFShort	1 B	2	1
PRFShort	8 B	9	1

PRFShort	16 B	17	1
----------	------	----	---

PRFShort’s call count is fixed at 1 regardless of length, a consequence of its single-pass design and the structural reason for its constant cycle-count floor. AEAD128 splits its calls between $p[8]$ for duplexing and $p[12]$ for initialization and finalization. Every other member uses $p[12]$ exclusively.

B.2 Testbench Results and Cross-Check Method

Six unit testbenches exercise the accelerator’s supporting RTL modules in isolation, and three integration testbenches exercise the interactions between them. The CPU/accelerator memory path, the CV-X-IF handshake with the accelerator attached, and the AXI-Lite control and status path. Table B.2 summarizes all nine, run under Vivado xsim.

Table B.2: Unit and Integration Testbench Results

Test ID	Module / Scope Under Test	Assertions Passed	Result
<i>Unit level</i>			
1.2	instr_mem_cv32e40x	10	PASSED
1.3	data_mem_cv32e40x	9	PASSED
1.4	memory_arbiter	16	PASSED
1.5	performance_counters	12	PASSED
1.6	cv32e40x_xif_mux	10	PASSED
1.7	accel_wrapper	17	PASSED
<i>Integration level</i>			
2.1	Data path (tb_data_path_integration.sv)	9	PASSED
2.2	XIF + accelerator (tb_xif_accel_integration.sv)	13	PASSED
2.3	Control and status (tb_ctrl_stat_integration.sv)	23	PASSED
Total	9 of 9 testbenches	119	0 failed

The Level 7 testbenches instantiate the complete SoC and run each algorithm’s firmware twice, once against its software permutation and once against the accelerated one, checking both against the same KAT vector and confirming that `accel_ops` matches the expected call count.

Table B.3: Level 7 Results, MSG = 16 B

Algorithm	Checks	accel_ops	Result
AEAD128	56	3	PASSED
Hash256	56	6	PASSED
MAC	56	2	PASSED
PRF	56	5	PASSED
PRFShort	56	1	PASSED
XOF128	60	10	PASSED
Total	340	—	0 failed

XOF128 carries four checks beyond the common 56, verifying that the unused portion of the result buffer stays zero. Each `accel_ops` value matches its 16 B row in Table B.1, which is an internal consistency check in its own right. The call count predicted from the algorithm’s structure and the count recorded by hardware agree for every member. Ascon-CXOF128 has no Level 7 testbench, having been added after this framework was complete, and its correctness rests on board-level KAT comparison alone.

Every KAT determination in this appendix is made the same way. The expected output for each vector is stored as a `KAT_EXPECTED` table inside the corresponding Python host script, derived from the NIST LWC KAT file. After each run the host reads the result words back from data memory over AXI, repacks them, and compares byte-for-byte against the expected value. The same comparison is applied to both the software-only and the hardware-accelerated run, so a pass confirms three things at once. The software baseline is correct, the hardware path is bit-identical to it, and both agree with the NIST reference vector.

B.3 Raw Measurement Data

Table B.4: Raw Measurement Counts: AEAD128

Length	cc (SW)	cc (HW)	ic (SW)	ic (HW)	sc (SW)	sc (HW)
16 B, AD=0	4,931	1,244	4,340	578	274	274
16 B, AD=16 B	6,959	1,444	6,304	669	315	315
32 B, AD=0	6,158	1,557	5,430	731	318	318
32 B, AD=32 B	9,178	1,834	8,370	861	369	369

Table B.5: Raw Measurement Counts: Hash256

Length	cc (SW)	cc (HW)	ic (SW)	ic (HW)	sc (SW)	sc (HW)
16 B	9,355	1,043	8,936	463	213	213
32 B	12,252	1,168	11,818	521	223	223
64 B	18,049	1,421	17,583	638	237	237
1024 B	191,929	8,981	190,503	4,118	717	717

Table B.6: Raw Measurement Counts: XOF128

Length	cc (SW)	cc (HW)	ic (SW)	ic (HW)	sc (SW)	sc (HW)
16 B	15,470	1,618	14,816	696	293	293
32 B	18,366	1,742	17,697	753	301	301
64 B	24,157	1,989	23,456	864	319	319
1024 B	197,918	9,430	196,257	4,225	797	797

Table B.7: Raw Measurement Counts: CXOF128

Length	cc (SW)	cc (HW)	ic (SW)	ic (HW)	sc (SW)	sc (HW)
0 B	18,441	1,817	17,739	794	317	317
8 B	19,882	1,872	19,174	817	320	320
16 B	21,313	1,917	20,598	829	326	326
32 B	24,225	2,057	23,494	901	332	332

Table B.8: Raw Measurement Counts: MAC

Length	cc (SW)	cc (HW)	ic (SW)	ic (HW)	sc (SW)	sc (HW)
16 B	3,459	687	3,148	323	179	179
32 B	4,943	785	4,613	376	185	185
64 B	6,484	940	6,116	467	201	201
1024 B	52,714	5,590	51,206	3,197	681	681

Table B.9: Raw Measurement Counts: PRF

Length	cc (SW)	cc (HW)	ic (SW)	ic (HW)	sc (SW)	sc (HW)
16 B	8,353	1,411	7,684	622	306	303
32 B	9,839	1,511	9,151	677	314	311
64 B	11,379	1,665	10,653	767	332	329
1024 B	57,610	6,316	55,744	3,498	810	807

Table B.10: Raw Measurement Counts: PRFShort

Length	cc (SW)	cc (HW)	ic (SW)	ic (HW)	sc (SW)	sc (HW)
0 B	1,946	560	1,662	250	168	168
1 B	1,982	596	1,676	264	177	177
8 B	2,087	701	1,725	313	191	191
16 B	1,961	576	1,666	254	177	177

The four Ascon-AEAD128 rows include the two associated-data points added under the $AD = |PT|$ convention. Those two were verified on the FPGA at Level 8 only, since the Level 7 ModelSim framework predates them, exactly as Ascon-CXOF128 was. Stall counts are identical between the software and hardware pass at 24 of the 28 points, which confirms that the accelerator issues no DMEM traffic of its own. The four Ascon-PRF points are the exception, showing a consistent three-cycle difference; this is discussed in Section 3.2.3.

Ascon-XOF128 and Ascon-CXOF128 produce a fixed 64-byte output regardless of input length, so a second CPB convention is meaningful for them. Cycles per *output* byte, relevant when comparing against stream ciphers. Input CPB, the convention used for every other family member and reported in Chapter 4, is not repeated here. For Ascon-XOF128 at $MSG = 64$ B the two conventions coincide by construction.

Table B.11: Output Byte CPB XOF128 and CXOF128

Family	Length	Out-CPB (SW)	Out-CPB (HW)
XOF128	16 B	241.7	25.3
	32 B	287.0	27.2
	64 B	377.5	31.1
	1024 B	3092.5	147.3
CXOF128	0 B	288.1	28.4
	8 B	310.7	29.2
	16 B	333.0	30.0
	32 B	378.5	32.1

APPENDIX C

FPGA IMPLEMENTATION AND REPRODUCTION

This appendix reproduces the load-bearing extracts from the Vivado implementation reports behind the timing, resource and power figures of Chapter 4, and then gives the path from a clean checkout of the sources to a reproduction of the measurements. It draws only on the project’s own files. The reports themselves, the Makefile targets, and the register sequence documented in `accel_ctrl_regs.sv`.

C.1 Timing Closure

Table C.1 reproduces the post-route Design Timing Summary together with the worst setup path. Vivado’s summary line reads “All user specified timing constraints are met”. The basis for that is zero failing endpoints across setup, hold and pulse-width checks over 14,775 endpoints, not the WNS figure alone.

Table C.1: Design Timing Summary

Metric	Value
Setup (WNS / TNS)	2.145 ns / 0.000 ns, 0 of 14,775 endpoints failing
Hold (WHS / THS)	0.043 ns / 0.000 ns, 0 of 14,775 failing
Pulse width (WPWS / TPWS)	7.353 ns / 0.000 ns, 0 of 5,044 failing
Worst path source	u_core/.../mulh_state_reg[0]
Worst path destination	u_accel/u_accel/S_H_reg[0][26]
Delay vs requirement	13.911 ns of 16.666 ns (61% logic, 39% route)
Logic levels	12, including 2 DSP48E1 and 4 CARRY4

All five top setup paths share this pattern, originating at the multiplier's output register and terminating at an accelerator state register. This is a static timing result rather than a statement about firmware behaviour, but it does confirm that the accelerator's timing margin is coupled to the core's multiplier rather than being a property of the accelerator alone.

C.2 Resource Utilization and Power

Table C.2 combines whole design totals with hierarchical breakdown, resolved one level inside cv32e40x system wrap 0 so the accelerator is isolated from the core it shares a parent with. Block RAM and DSP are omitted from the table because their distribution is trivial: all 32 RAMB36 tiles belong to the two block memory generators, and all three DSP48E1 slices to u_core; the accelerator uses neither.

Table C.2: Device Totals and Hierarchical Breakdown

Resource / Instance	LUTs	FFs
Device used (of 53,200 / 106,400)	6200 (11.65%)	4888 (4.59%)
cv32e40x_system_wrap_0	4510	2965
u_core (cv32e40x_core)	3703	2296
u_accel (accel_wrapper)	597	368
u_perf (performance_counters)	162	160
u_arbiter (memory_arbiter)	38	139
u_imem / glue	13	2
axi_interconnect_0	1021	1268
axi_bram_ctrl_dmem_0 / _imem_0	282 / 228	261 / 248
accel_ctrl_regs_wrap_0	146	113
rst_ps7_0_60M	17	33

The 597 LUT / 368 FF figure Chapter 4 cites is u_accel's row. Its module is accel_wrapper rather than ascon_permutation directly, the latter sitting one level deeper, but Vivado rolls all usage below the requested depth into the totals shown, so the figure is the correct standalone cost for the whole accelerator subtree. The indented children sum to three LUTs more than the parent total, which the report attributes to cross-hierarchy LUT combining. The flip-flop totals, not subject to combining, sum exactly.

Every figure in this section is a *tool estimate*, not a measurement, in the sense set out in Table 4.1. It is reproduced in more detail than the other reports precisely because it is the one quantity in this thesis that rests on modelling rather than on observation, and a reader is entitled to see what the modelling assumed. Table C.3 reproduces the summary and hierarchical sections of `power_report.txt`; Table C.4 reproduces the estimation basis and confidence breakdown that qualify it; Table C.5 gives the on-chip breakdown by resource class.

Table C.3: Power Summary and Dynamic Power

Metric / Instance	Value
Total on-chip power	1.723 W
Dynamic	1.583 W
Device static	0.140 W
Junction temperature (default thermal model)	44.9 °C
Overall confidence level	Medium
Simulation activity file	— (none; vector-less)
<i>Dynamic power by hierarchy</i>	
processing_system7_0 (Zynq PS)	1.520 W
cv32e40x_system_wrap_0 (core + accelerator)	0.021 W
bram_gen_dmem / bram_gen_imem	0.018 W / 0.014 W
axi_interconnect_0	0.006 W
axi_bram_ctrl_dmem_0 / _imem_0	0.002 W / 0.002 W

Table C.4: Estimation Basis and Confidence

<i>Estimation basis, as recorded by the tool</i>		
Field	Value	Consequence
Command	report_power -file power_report.txt; Vivado 2023.2, design state <i>routed</i>	
Setting file	—	No user switching-activity settings applied
Simulation activity file	—	No SAIF or VCD; estimation is <i>vectorless</i>
Design nets matched	NA	No net activity derived from firmware execution
<i>Confidence by input class</i>		
User input data	Confidence	Tool detail
Design implementation state	High	Design is routed
Clock nodes activity	High	More than 95% of clocks user-specified
I/O nodes activity	High	More than 95% of inputs user-specified
Internal nodes activity	Medium	Fewer than 25% of internal nodes user-specified
Device models	High	Production characterization
Overall	Medium	Limited by internal node activity alone

Table C.5: On Chip Power by Resource Class

On-chip component	Power (W)	Note
Clocks	0.012	3 clock networks
Slice logic	0.008	13,204 cells; LUT-as-logic 0.007 W
Signals	0.012	10,668 nets
Block RAM	0.031	32 RAMB36; IMEM and DMEM
DSPs	<0.001	3 blocks, core multiplier only
PL dynamic subtotal	0.063	Sum of the five rows above
PS7 (Zynq processing system)	1.520	Vccpint 0.718 W; Vcco_ddr 0.686 W; Vccpaux 0.090 W; Vccpll 0.025 W
Device static	0.140	Leakage at the modelled 44.9 °C junction
Total	1.723	

Three qualifications follow from these tables and are carried into Chapter 4 wherever a power figure appears.

The accelerator is not separately resolved. The hierarchical section descends only to `cv32e40x_system_wrap_0`, so the 0.021 W figure covers core and accelerator together and no estimate exists for `ascon_permutation .sv` alone. Nothing in this report supports apportioning that 0.021 W between them, and none is attempted. An area-proportional split would assume uniform switching activity across two very differently exercised modules.

The estimate is mode-independent. With no simulation activity file and no matched nets, the same figure applies whether the SW or the HW binary is loaded. Any energy figure formed as power $\times$ measured cycles

would therefore differ between the two modes only by the cycle counts, reproducing the speedup exactly. This is why Section 4.6.3.1 declines to report one.

The environment is Vivado’s default, not the deployed one. The report assumes an ambient of 25 °C, θ_{JA} of 11.5 °C/W, 250 LFM airflow, no heat sink, and a medium (10 inch) 8-to-11-layer board. The PYNQ-Z1 as used here is a smaller board in still air, so the 44.9 °C junction temperature and 65.1 °C maximum ambient are properties of the default thermal model rather than of the deployed system. The dynamic figures are computed from activity and capacitance before the thermal model is applied and are unaffected. Only the temperature dependent static term and the reported thermal margins move.

C.3 Design Rule Checks

Table C.6: Design Rule Check Summary

Rule	Severity	Description	Count
REQP-1839	Warning	RAMB36 asynchronous control check	20
RBOR-1	Warning	RAMB output registers	16
DPIP-1	Warning	DSP input pipelining	5
DPOP-1	Warning	DSP PREG output pipelining	3
DPOP-2	Warning	DSP MREG output pipelining	3
CHECK-3	Warning	Report rule limit reached	1
Total		48 violations, all Warning severity	

A separate `report_methodology` run records two further findings, neither of which appears in the DRC summary above and one of which lies inside an authored module. LUTAR-1 (Warning) reports that a LUT in the accelerator’s state machine, `u_accel/FSM_sequential_state[1]_i_2`, drives the asynchronous clear pins of the `xif _result _data` result registers; LATCH-1 (Advisory) reports one inferred latch in the design. The LUTAR-1 finding is accepted on the same grounds as the arbiter’s RAMB asynchronous-control warnings. `accel_reset` is asserted only

by the host before firmware is loaded and is never toggled while a permutation is in flight, so no reset is ever applied mid-transaction. It is recorded here rather than omitted, since the DRC summary alone would suggest the authored modules are entirely clean.

All 48 violations are Warning severity, with zero Critical Warnings and zero Errors. All six rules are performance advisory checks on DSP pipelining and Block RAM control timing, consistent with the design's DSP-based multiplier and BRAM-backed memories, rather than functional or connectivity errors. `report_ip_status` additionally confirms all twelve IP instances as up to date.

The design uses a single clock domain sourced from the Zynq PS7's `FCLK_CLK0` output rather than a fabric MMCM or PLL, which is why the frequency is not an exact 60.000 MHz.

Table C.7: Clock Domain and AXI Map

Clock / Segment	Value	Source / Size
<code>clk_fpga_0</code>	60.0024 MHz	<code>PS7_i/FCLKCLK[0]</code>
AXI-Lite registers	0x40020000	4 KB
DMEM	0x40010000	64 KB
IMEM	0x40000000	64 KB

C.4 Toolchain and Firmware Build

Table C.8 lists the tool versions. Reproducing the cycle counts exactly does not require them, since the ISA and accelerator are deterministic, but reproducing without re-verifying does. A different compiler changes code size and scheduling enough to shift cycle counts, though not correctness.

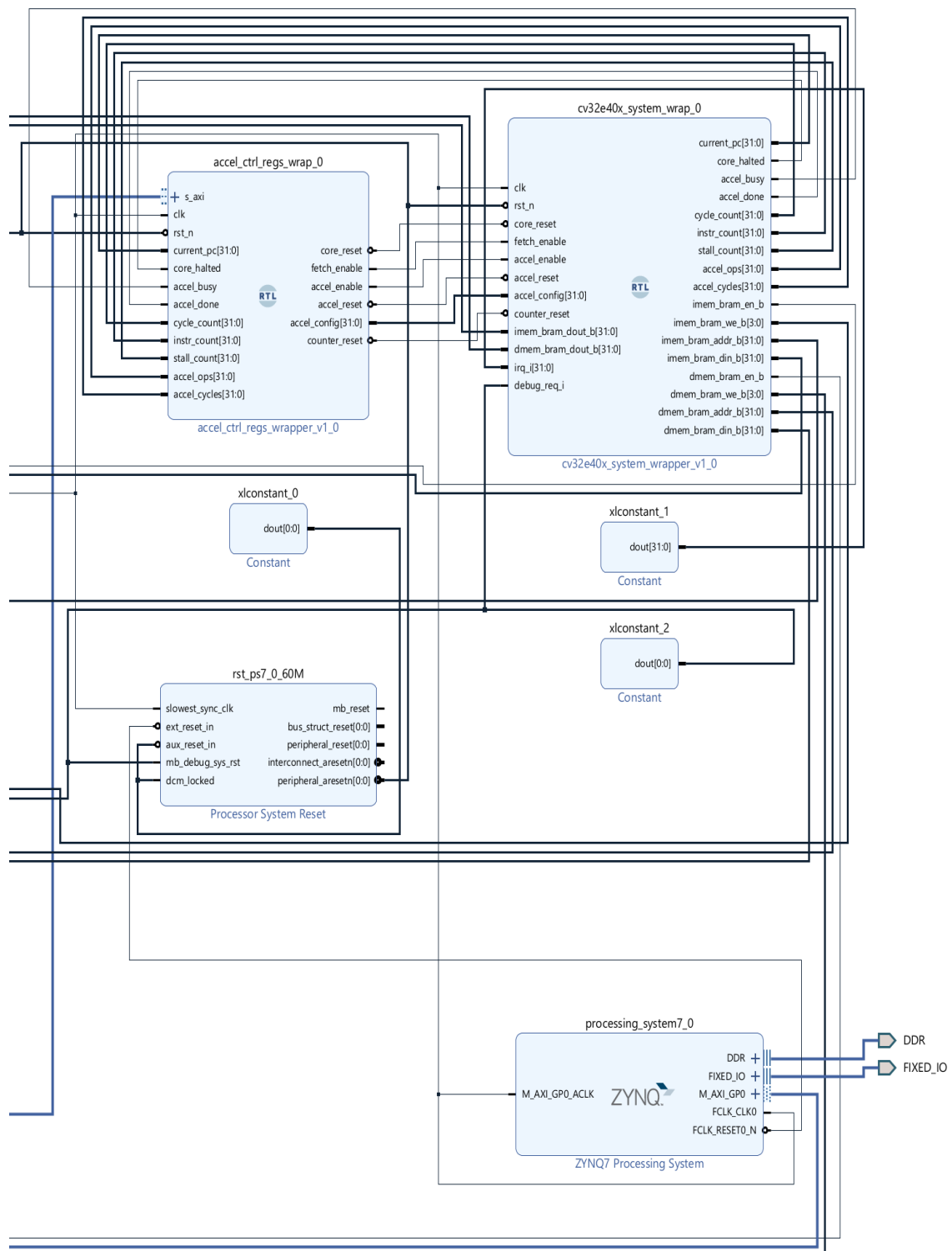


Figure C.1: Vivado IP Blocks 01

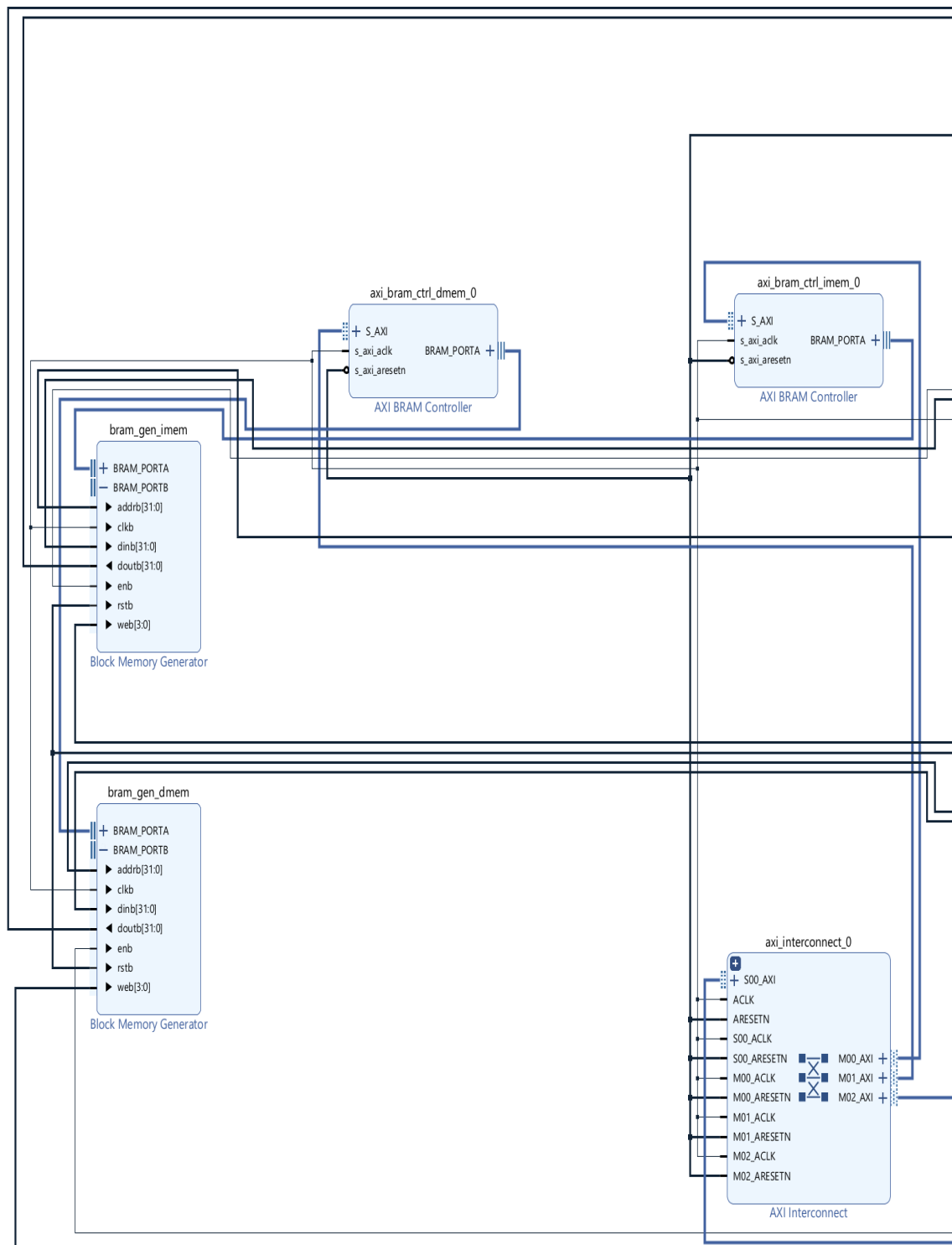


Figure C.2: Vivado IP Blocks 02

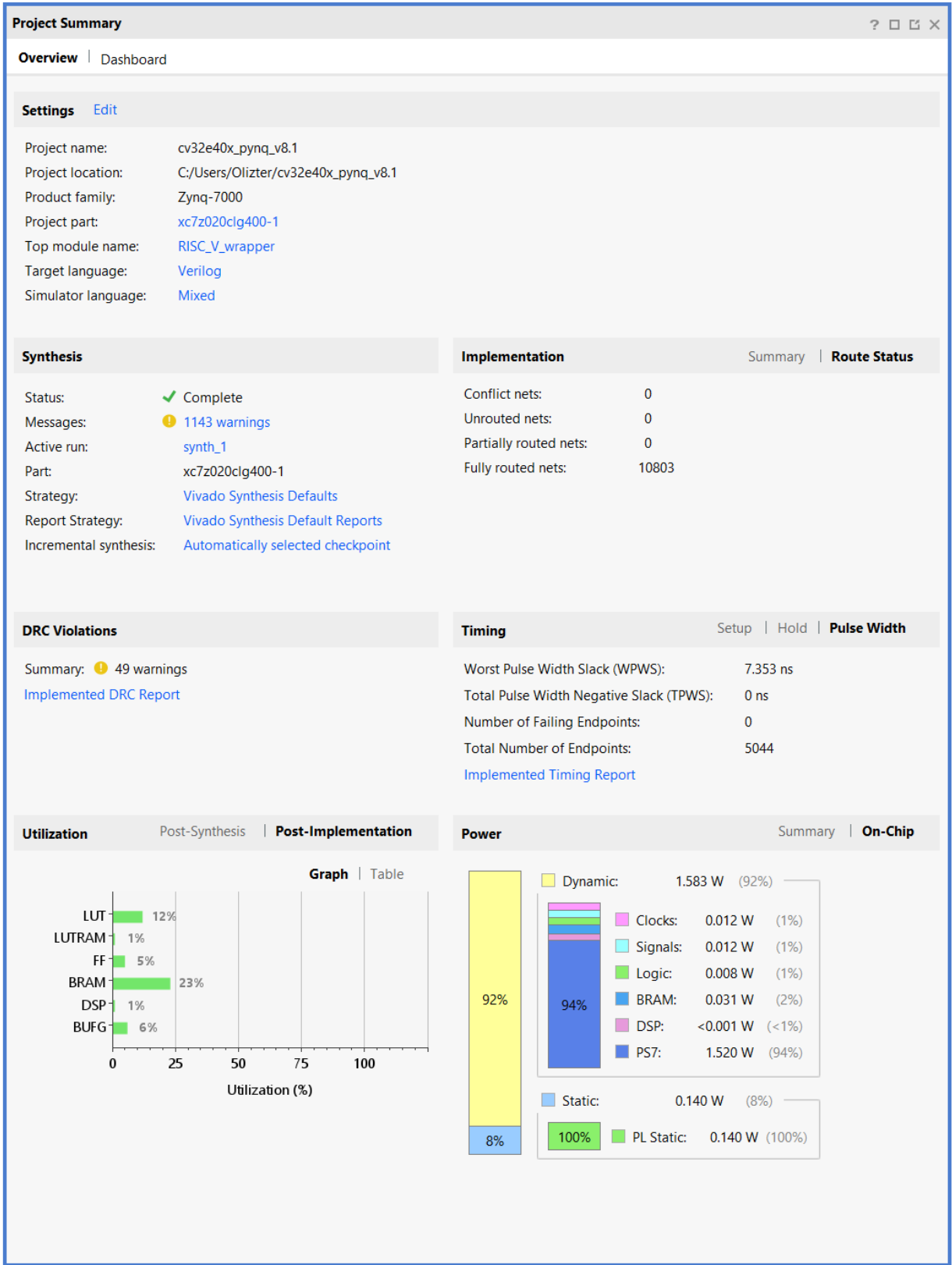


Figure C.3: Vivado Project Summary

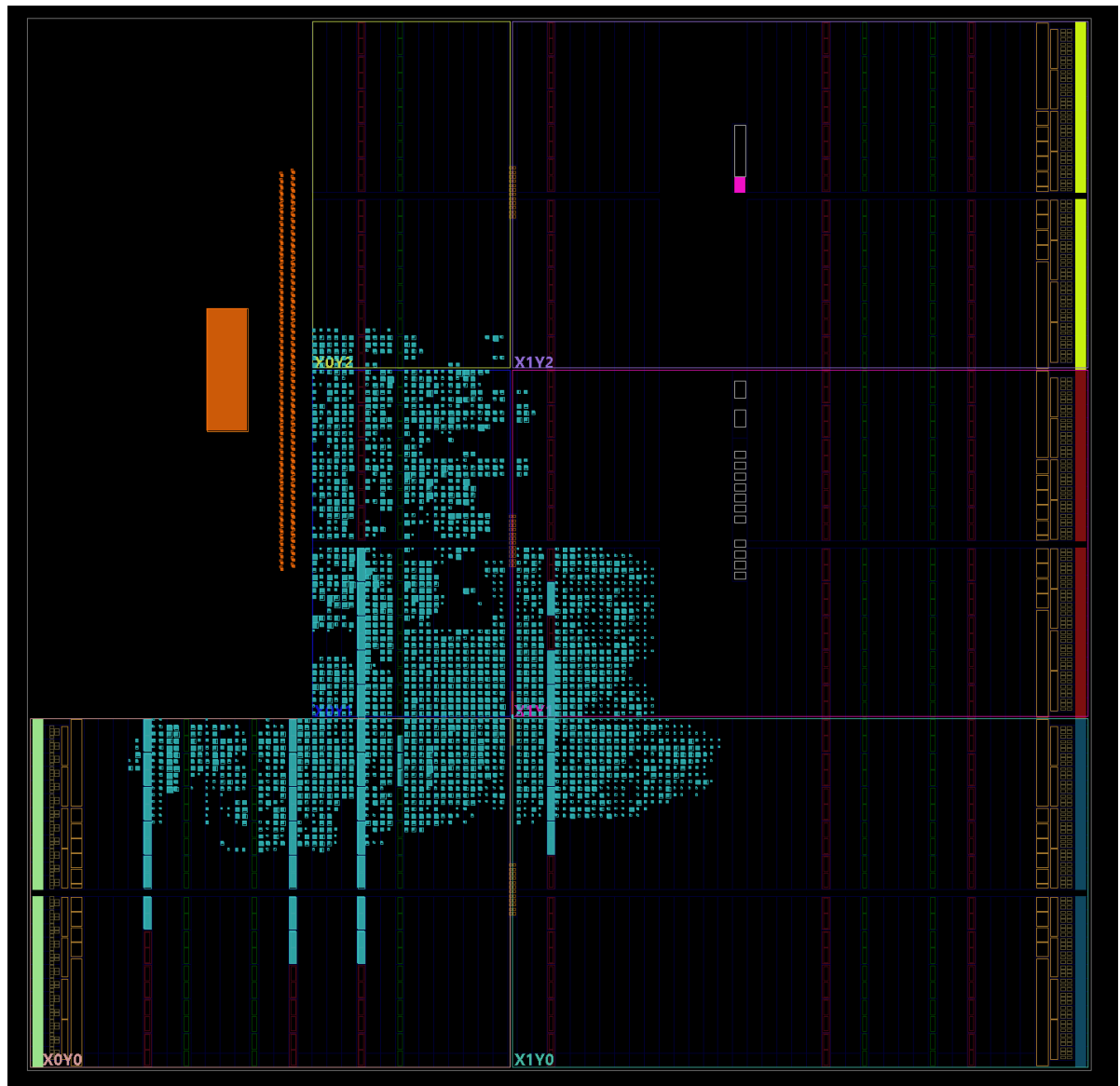


Figure C.4: Vivado Floor Planning

Table C.8: Toolchain and Environment

Component	Version / Value
FPGA board	Xilinx PYNQ-Z1 (Zynq-7000, xc7z020clg400-1)
PS-side OS	PYNQ Linux (Ubuntu 22.04 base), kernel 6.6.10-xilinx-v2024.1
Vivado	2023.2 (64-bit)
RTL simulator	ModelSim / Questa Intel FPGA Edition, vlog/vmap 2020.02; Vivado xsim (bundled with 2023.2) also used for unit-level testbenches
RISC-V toolchain	riscv32-unknown-elf-gcc 15.2.0
ISA / ABI	rv32imc_zicsr / ilp32
Default optimization	-O1 -g (Makefile OPT_FLAGS; -O0/-O2/-O3 variants available as separate targets, not used for the results in this thesis)
Python (board side)	3.12.3

The board binaries are produced by the top-level `Makefile`. The ModelSim hex files used by the Level 7 testbenches come from the same targets with a different output format.

Every Level 8 measurement point has its own named target, following the pattern `l8- // <algorithm>-<sw|hw>-<length>`. For example, AEAD128 at PT=16 B (the first row of Table B.4) is built with:

Listing C.1: Level 8 board binary build, AEAD128 PT=16B, software baseline

```
riscv32-unknown-elf-gcc -march=rv32imc_zicsr -mabi=ilp32 -O1 -g \
  -ffreestanding -nostdlib -fno-stack-protector \
  -fno-common -fno-builtin -fno-PIE \
  -mstrict-align -mno-relax \
  -Wl,-Map=output/aead128_sw_pt16.map \
  -Wl,--no-relax -Wl,--gc-sections \
  -T cv32e40x.ld \
  startup.S bench_main_aead128_16.c ascon.S encrypt.c memcpy.c \
  -o output/aead128_sw_pt16.elf

riscv32-unknown-elf-objcopy -O binary \
  --only-section=.text --only-section=.rodata \
  output/aead128_sw_pt16.elf output/aead128_sw_pt16.bin

riscv32-unknown-elf-objcopy -O binary \
  --only-section=.data --change-section-lma .data=0x0 \
```

```
output/aead128_sw_pt16.elf output/aead128_sw_pt16_dmem.bin
```

or, more simply, make `l8-aead128-sw-pt16`. The hardware counterpart make `l8-aead128-hw-pt16` is identical except that `ascon.S` is replaced by `ascon_core_accel.S`. The same pattern, with only the source files and the length parameter changing, produces every other row across the per-family tables of Section B.3, following the naming convention `<algorithm>_sw.S / <algorithm>_accel.S`. The resulting `.bin` files are what Step 2 of Section C.5 loads onto the board.

C.5 Bitstream Deployment and Measurement

The block design's IP instance list and address map are given in Appendix C, and `ip_status.rpt` records every IP version, but the TCL script that assembles the block design is not reproduced here. The reliable route is therefore to export a fresh TCL record from the working Vivado project with `write_project_tcl`, which captures it exactly as built. Failing that, the design can be rebuilt by hand: target `xc7z020clg400-1`, add each IP instance at its recorded version, wire the AXI and BRAM connections per the address map and the port list in `cv32e40x_system_wrapper.v`, and configure the PS7's `FCLK_CLK0` for 60 MHz. The clock is PS7-derived rather than set by an XDC constraint, which is why no constraints file accompanies the timing reports.

Either route should be checked against Appendix C before the bitstream is trusted. If the regenerated utilization and timing reports do not closely match Tables C.2 and C.1, the rebuild differs from the original and the measurements should not be assumed to carry over.

Once a bitstream and the firmware binaries are available, every measurement follows the same nine-step register sequence, driven from Python over the AXI-Lite interface of Appendix A.

1. **Hold reset.** `ctrl.write(0x00, 0x01)` Hold the core in reset, the safe state for loading memory contents.

2. **Load firmware.** Write the IMEM .bin and DMEM .bin files from Section C.4 into the design's BRAM over their AXI BRAM controllers, at the base addresses given in Table C.7 (IMEM 0x40000000, DMEM 0x40010000).
3. **Clear completion flags.** Clear the DMEM locations the firmware uses as completion sentinels `dmem.write(0x8200, 0)`, `dmem.write(0x8000, 0)`, so a stale value from a previous run cannot be mistaken for a fresh completion signal.
4. **Reset counters.** `ctrl.write(0x20, 0x01)` pulses `counter_reset` (offset 0x20, self-clearing); wait briefly (on the order of 50 ms was used in this project) for the pulse to take effect.
5. **Verify zero.** Read back `CYCLE_COUNT`, `INSTR_COUNT`, and `STALL_COUNT` (offsets 0x10/0x14/0x18) and confirm all three are zero before proceeding.
6. **Release the core.** Write 0x02 for a software-only pass or 0x0E for a hardware-accelerated pass (Table A.4); the core begins fetching immediately.
7. **Poll for completion.** Repeatedly read the DMEM completion sentinel (`dmem.read(0x8200)`) until it equals 0xDEADBEEF; the firmware's `halt_count` freezes the cycle counter at the moment it signals completion, so the poll interval does not affect the measured cycle count.
8. **Read the counters.** Read `CYCLE_COUNT`, `INSTR_COUNT`, `STALL_COUNT`, and, for hardware accelerated passes, `ACCEL_OPS` and `ACCEL_CYCLES` (offsets 0x10/0x14/0x18/0x30/0x34). These raw values are exactly the `cc/ic/sc` figures reported in Appendix B.
9. **Re-assert reset.** `ctrl.write(0x00, 0x01)` before loading the next binary or ending the session.

The derived metrics reported in Chapter 4 follow directly from the Step 8 readings:

$$\text{CPI} = \frac{CC}{IC},$$

$$\text{Stall \%} = \frac{SC}{CC} \times 100,$$

$$\text{CPB} = \frac{CC}{\text{message length in bytes}},$$

$$\text{Speedup} = \frac{CC_{\text{SW}}}{CC_{\text{HW}}},$$

$$\text{Throughput (Mbps)} = \frac{\text{message length in bytes} \times 8 \times 60}{CC}.$$

A pass through Steps 1–9 with Step 6 value `0x02`, followed by a second pass with `0x0E` on the same firmware source compiled twice, produces one complete row of the relevant per-family table in Section B.3, cross-checked against the KAT comparison of Section B.2.