A Comprehensive Framework for Fair and Efficient Benchmarking of Hardware Implementations of Lightweight Cryptography

Jens-Peter Kaps, Farnoud Farahmand, Kris Gaj

George Mason University, USA

William Diehl

Virginia Tech, USA

Michael Tempelmeier

Technische Universität München, Germany

CERG
Cryptographic Engineering Research Group
cryptography.gmu.edu

SAL
SIGNATURES ANALYSIS LAB

Lehrstuhl für Sicherheit in der Informationstechnik

TUM
Technische Universität München

Ekawat Homsirikamol, Independent Researcher

Acknowledgements

- This work is partially supported by the Department of Commerce (NIST) Grant no. 70NANB18H219

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

CERG
Cryptographic Engineering Research Group
cryptography.gmu.edu

SAL
SIGNATURES ANALYSIS LAB

Overview

- Introduction
- Proposed Hardware API for Lightweight Cryptography
- Development Package and Implementer’s Guide
- Conclusions

Introduction

- LWC HW API Team
- Previous Work

LWC HW API Team

Jens-Peter Kaps

CERG

GEORGE MASON UNIVERSITY

William Diehl

SAL

VIRGINIA TECH

Michael Tempelmeier

EI SEC

TUM

Farnoud Farahmand

CERG

GEORGE MASON UNIVERSITY

“Ice” Homsirikamol

Independent

Researcher

Kris Gaj

CERG

GEORGE MASON UNIVERSITY

Previous Work

- SHA-3 Contest (2007-2012)
  - 1st attempt at defining hardware API by CERG.
  - High-speed implementations of all 14 Round 2 and 5 Round 3 candidates and SHA-2 using API.
  - Lightweight implementations of 13 Round 2 and 5 Round 3 candidates using LW API.
  - API not endorsed by NIST.
- CAESAR Contest (2013-2019)
  - Hardware API proposed by CERG and endorsed by CAESAR committee in May 2016.
  - Development Package v1 released in Jun. 2016.
  - Implementer's Guide published at the same time.
  - Development Package v2 (incl LWC support) released Dec. 2017.

CAESAR (continued)

- Development Package
  - Non mandatory, not endorsed by CAESAR committee.
  - 32 out of 42 (76%) Round 2 implementations fully compliant with CAESAR HW API. All compliant used Development Package.
  - 23 out of 29 (79%) implementations of 15 Round 3 candidates were fully compliant. All compliant used Development Package.
  - Several LW implementations were also reported.
- CAESAR HW API and its endorsement had a major impact on fairness and comprehensiveness of HW benchmarking.
- Random Data Input (RDI) was added to facilitate benchmarking of implementations protected against Power Analysis.

Proposed Hardware API for LWC

- Minimum Compliance Criteria
- Interface
- Communications Protocol
- Support for Side-channel Resistant Implementations

Minimum Compliance Criteria (1)

- Authenticated encryption and decryption should be implemented within one LWC core.
  - If hashing is supported, an additional version for encryption, decryption, and hashing in one LWC core.
- Only one operation (enc/dec/hash) executed at a time.
- Key scheduling should be implemented in LWC core.
- LWC core should handle incomplete blocks.
  - Padding should be implemented in hardware.
- Decrypted plaintext blocks should be released immediately, before tag check.
  - Buffering handled by external HW or SW.

Minimum Compliance Criteria (2)

- LWC core should support only inputs composed of full bytes.
- Use of external memory only for two-pass algorithms.
- The LWC core should have only one clock input and internal clock signal.
- Inputs that are not changed should not be passed to the output, e.g., Npub, AD.
- Permitted data bus width are 8, 16, and 32 bits.

Minimum Compliance Criteria (3)

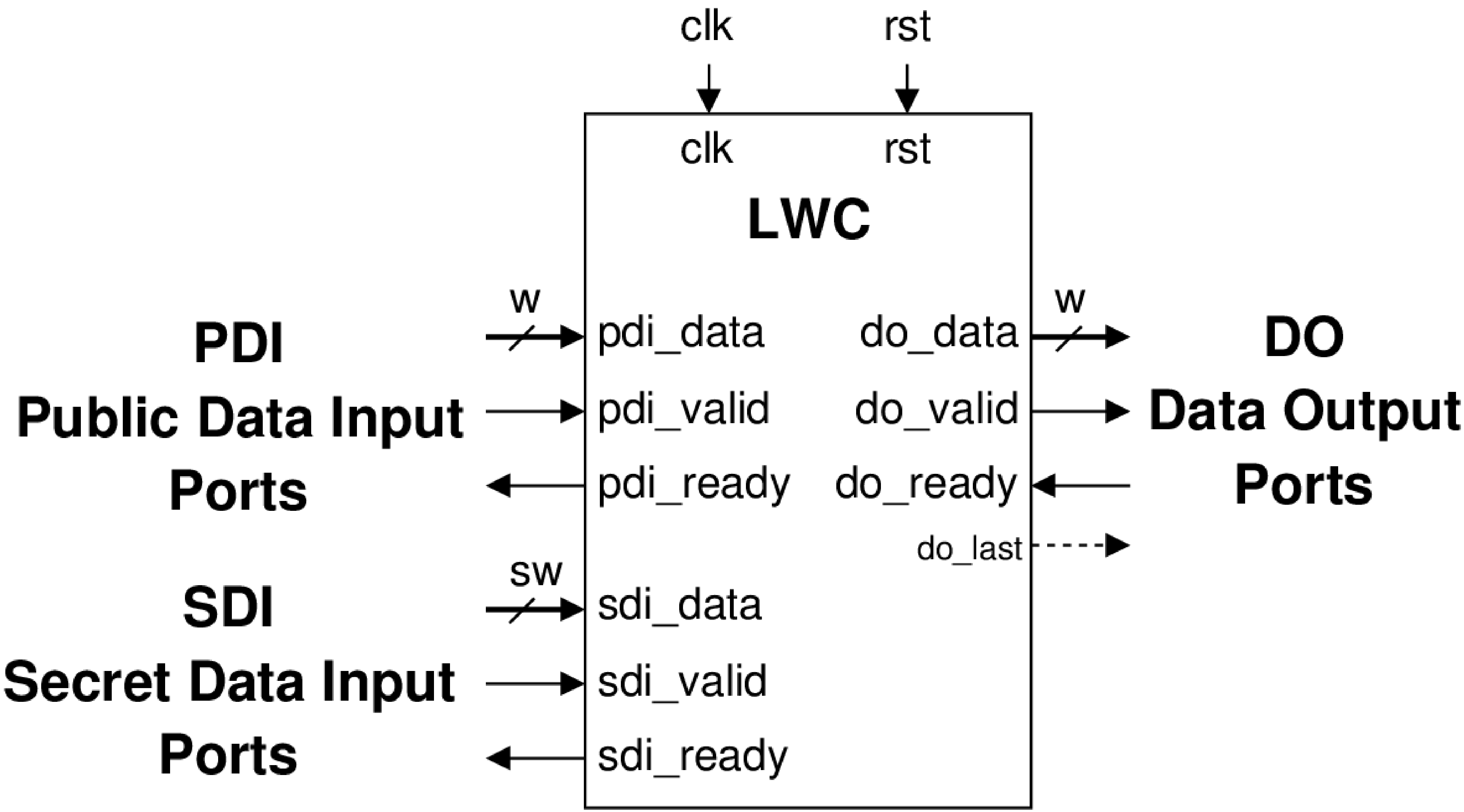
- LWC core should support following max sizes:

Single Pass	
2^{16}-1	Default
2^{32}-1	CAESAR API
2^{50}-1	NIST limit

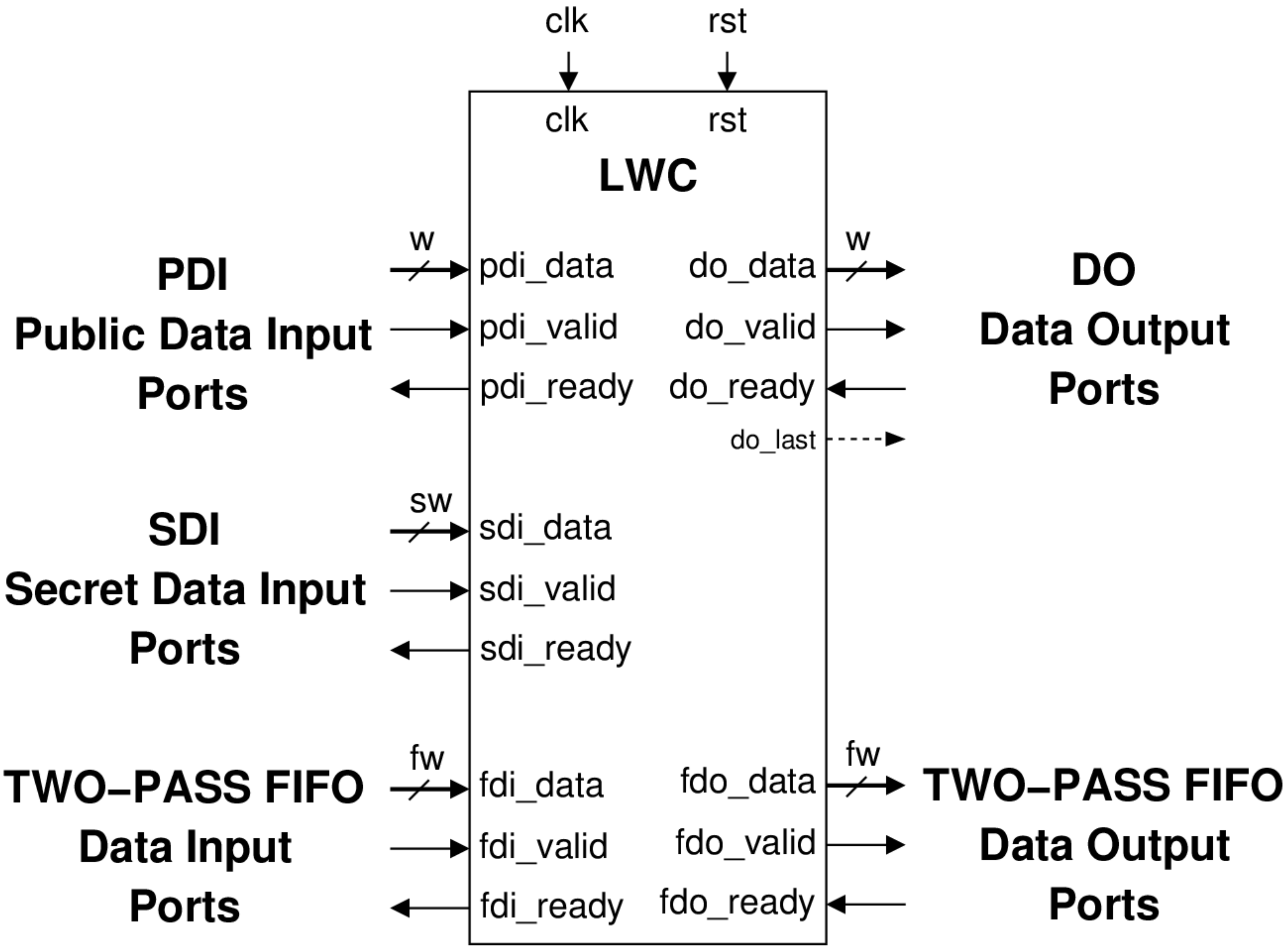
Two Pass	
2^{16}-1	Default
2^{11}-1	CAESAR API
2^{50}-1	NIST limit

- The size limit 2^{16}-1 should be sufficient for the majority of applications.
- Implementers should make sure that the remaining size limits do not influence
  - Maximum clock frequency,
  - Throughput for long messages.

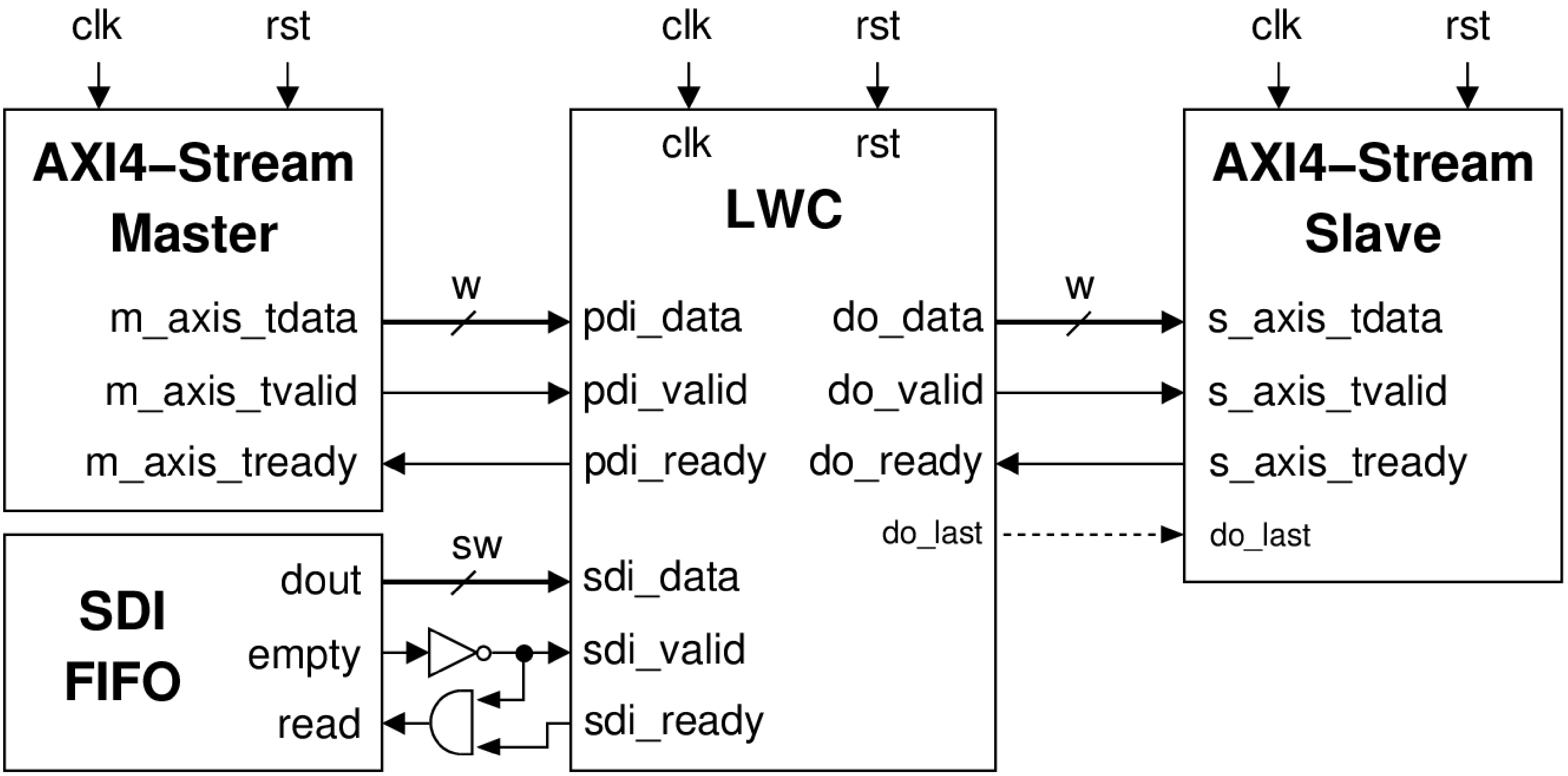
LWC Interface



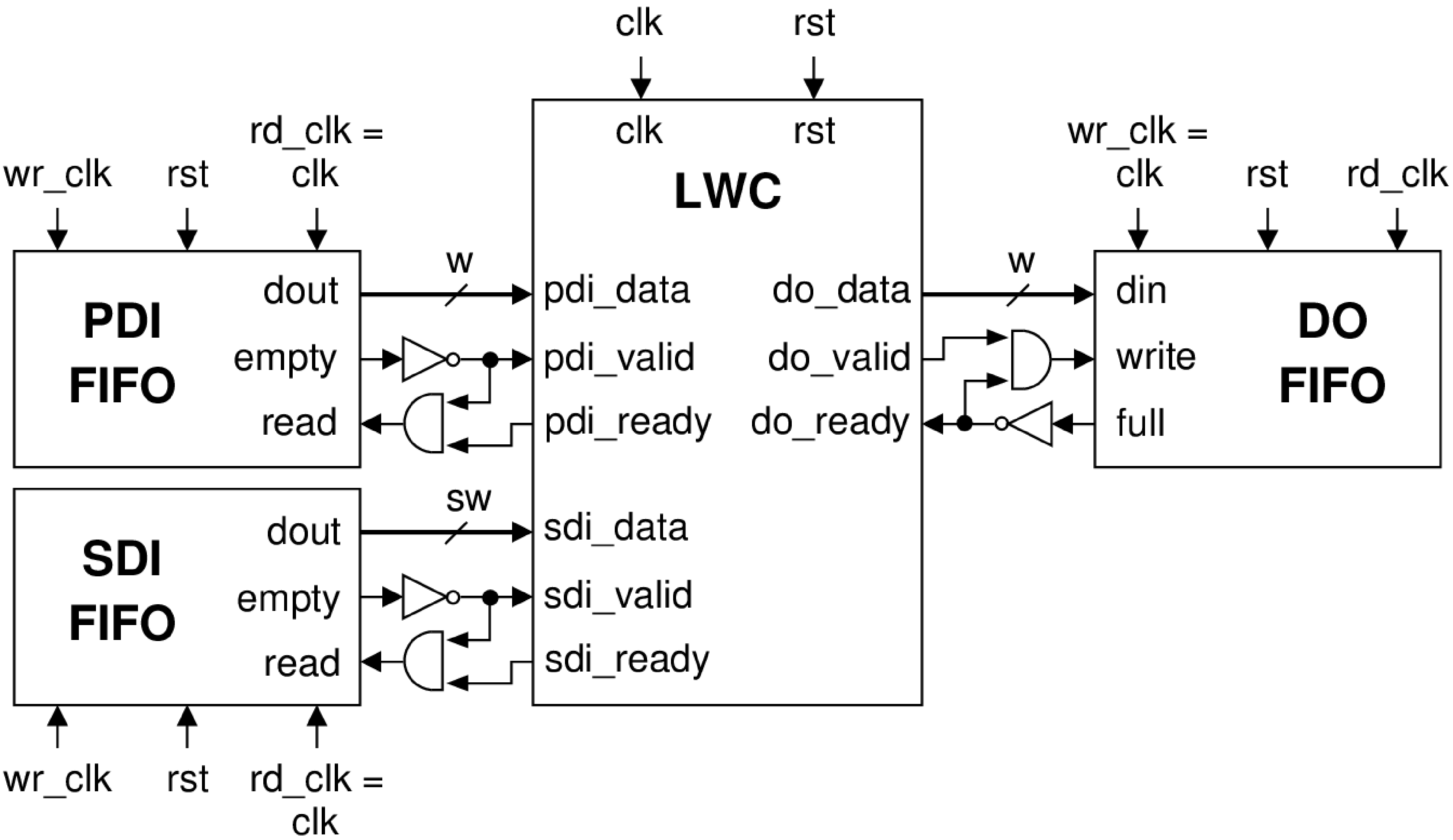
LWC Interface for Two-Pass Algorithms



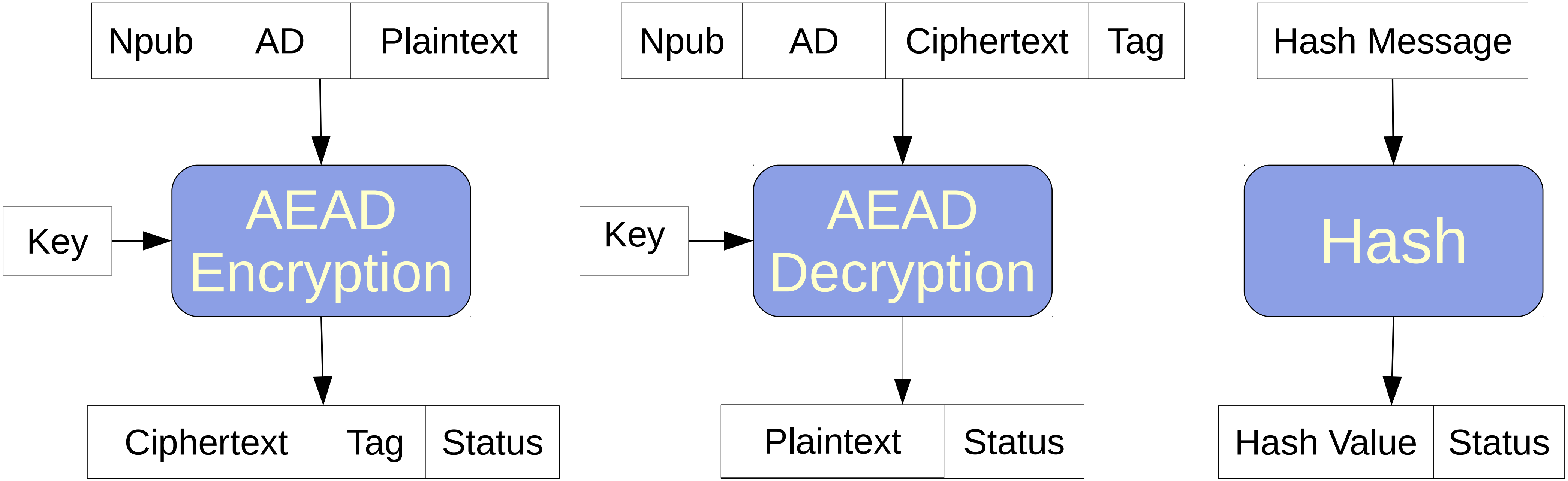
Typical External Circuits – AXI4 IPs



Typical External Circuits – FIFOs



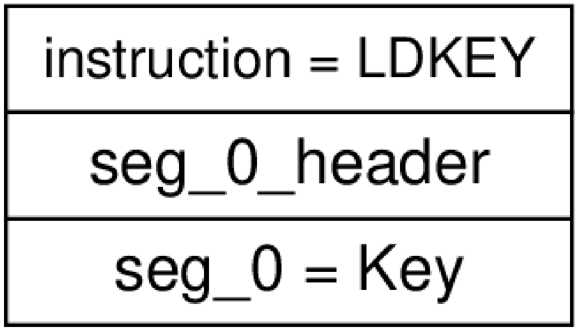
Input and Output of an LWC Core



- Npub – Public Message Number: Nonce
- AD – Associated Data
- Status: Success or Failure

Format of Secret Data Input

- All inputs start with an instruction.
- They are followed by segments.
- SDI has only one instruction and segment type.



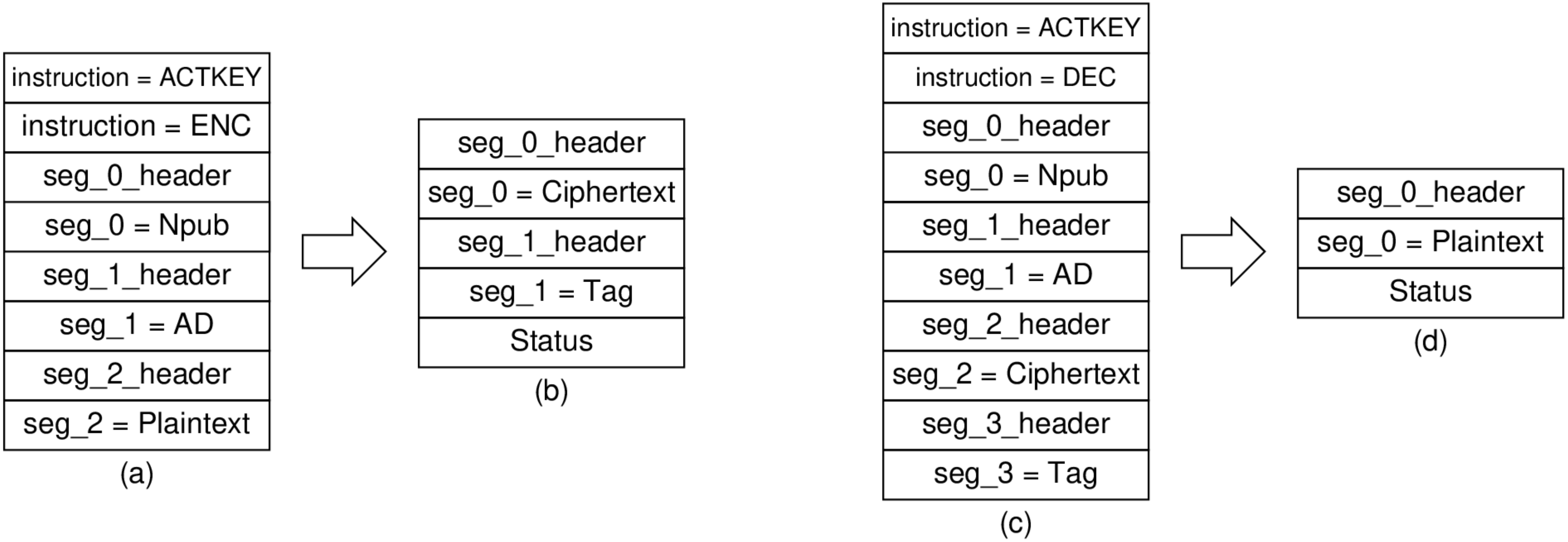
instruction = LDKEY

seg_0_header

seg_0 = Key

Format of Public Data Input for AEAD

- Encryption
  - (a) Public Data Input
  - (b) Data Output
- Decryption
  - (c) Public Data Input
  - (d) Data Output



instruction = ACTKEY
instruction = ENC
seg_0_header
seg_0 = Npub
seg_1_header
seg_1 = AD
seg_2_header
seg_2 = Plaintext

(a)

seg_0_header
seg_0 = Ciphertext
seg_1_header
seg_1 = Tag
Status

(b)

instruction = ACTKEY
instruction = DEC
seg_0_header
seg_0 = Npub
seg_1_header
seg_1 = AD
seg_2_header
seg_2 = Ciphertext
seg_3_header
seg_3 = Tag

(c)

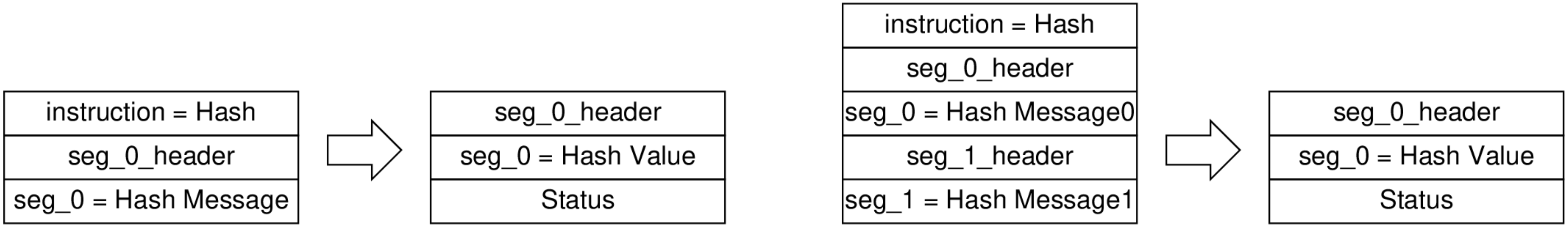
seg_0_header
seg_0 = Plaintext
Status

(d)

Format of Public Data Input for Hash

- One Segment
  - (a) Public Data Input
  - (b) Data Output

- Multiple Segments
  - Allowed for AD, Plaintext, Ciphertext, Hash Message
  - (c) Public Data Input
  - (d) Data Output



instruction = Hash
seg_0_header
seg_0 = Hash Message

(a)

seg_0_header
seg_0 = Hash Value
Status

(b)

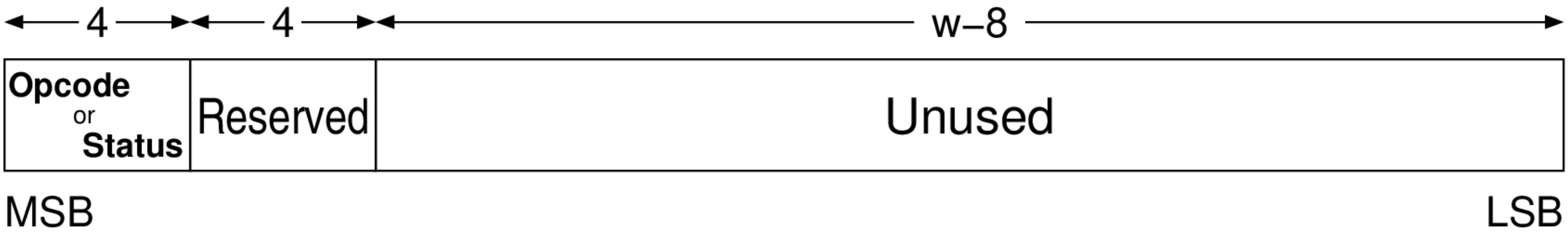
instruction = Hash
seg_0_header
seg_0 = Hash Message0
seg_1_header
seg_1 = Hash Message1

(c)

seg_0_header
seg_0 = Hash Value
Status

(d)

Format of Instruction/Status Word



4	4	w–8
Opcode or **Status**	Reserved	Unused

MSB LSB

Opcode:

0010 – Authenticated Encryption (ENC)

0011 – Authenticated Decryption (DEC)

0100 – Load Key (LDKEY)

0111 – Activate Key (ACTKEY)

1000 – Hash

Status:

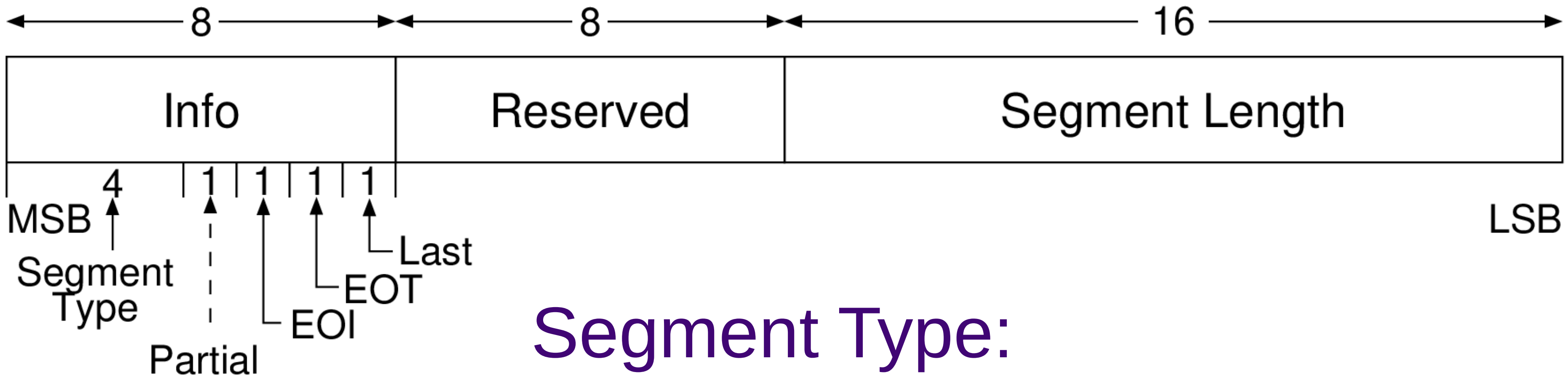
1110 – Success

1111 – Failure

Others – Reserved

- Word size w can be 8, 16, or 32

Format of Segment Header



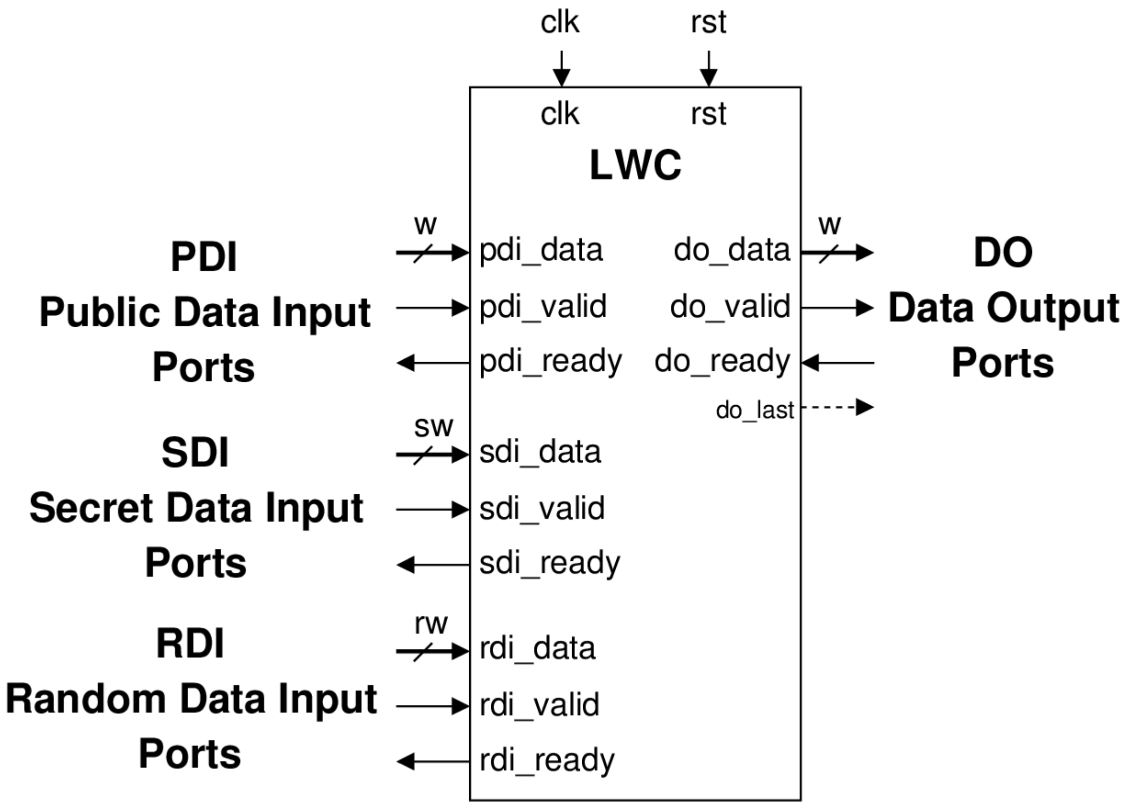
Segment Type:

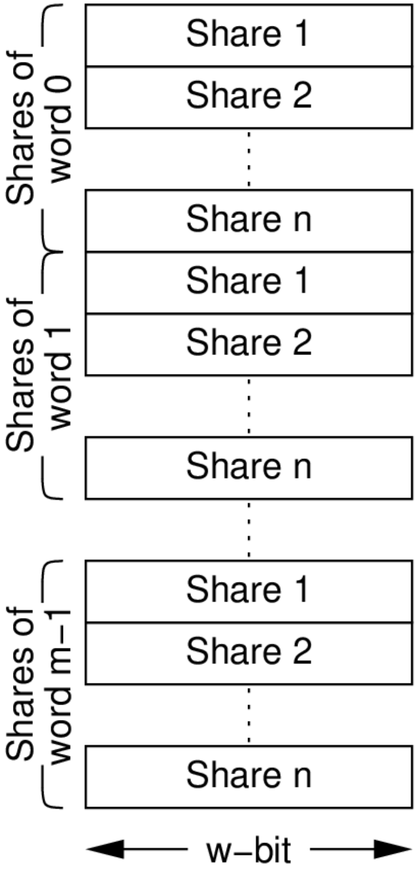
Encoding	Type
0000	*Reserved*
0001	AD
0010	Npub||AD
0011	AD||Npub
0100	Plaintext
0101	Ciphertext
0110	Ciphertext||Tag
0100	Hash Message

Encoding	Type
1000	Tag
1001	Hash Value
1010	Length
1011	*Reserved*
1100	Key
1101	Npub
1110	Nsec
1111	Enc Nsec

Support for Side-channel Resistant Implementations

- Added Random Data Input (RDI) bus
- No header or instruction words, no segments
- Sets rdi_ready, checks rdi_valid and reads *rw* bits of random data.

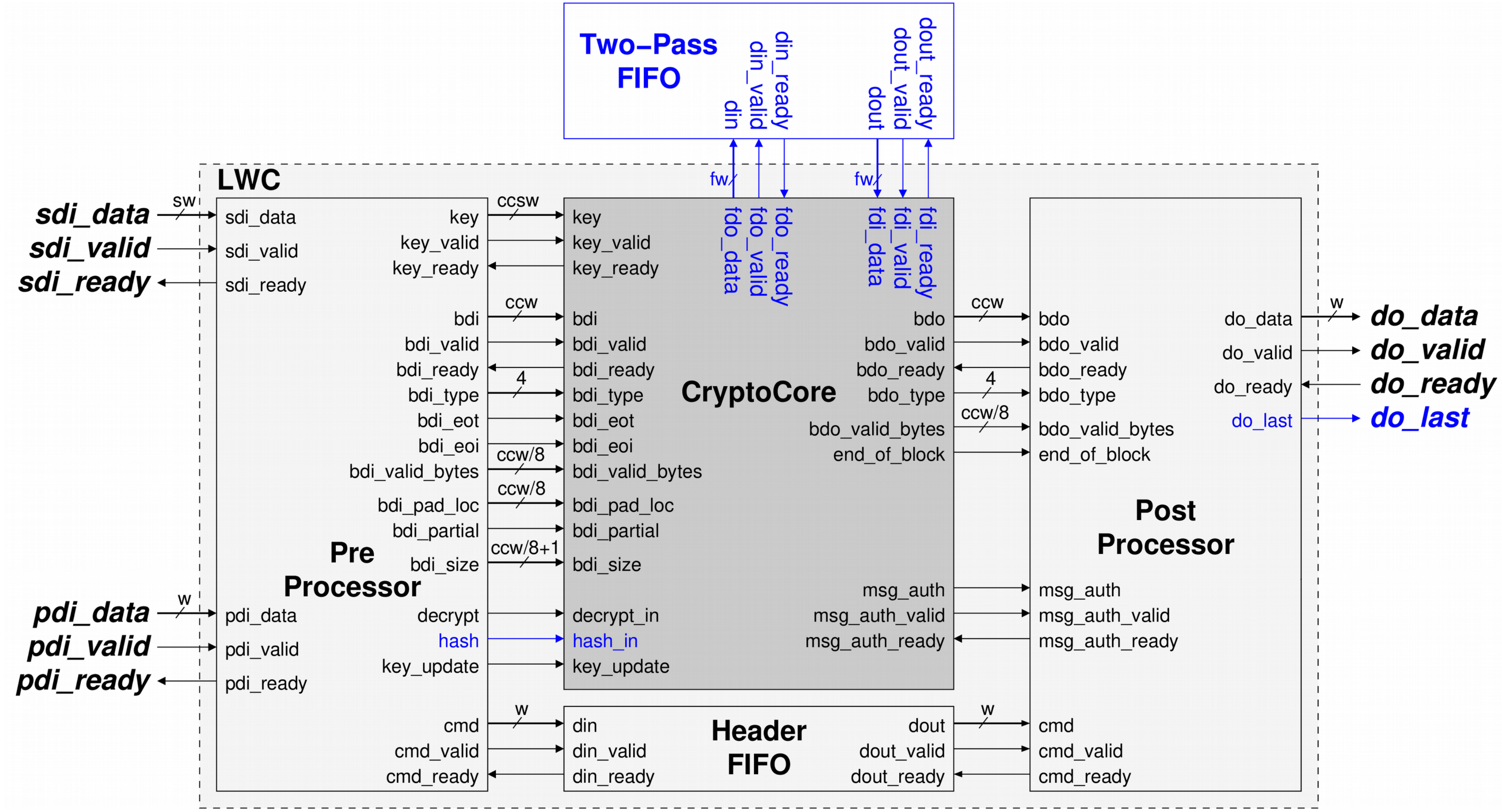




Development Package and Implementer’s Guide

- Block Diagram and Design Methodology
- Test Vector Generator and Universal Testbench
- Experimental Testing

Block Diagram of LWC

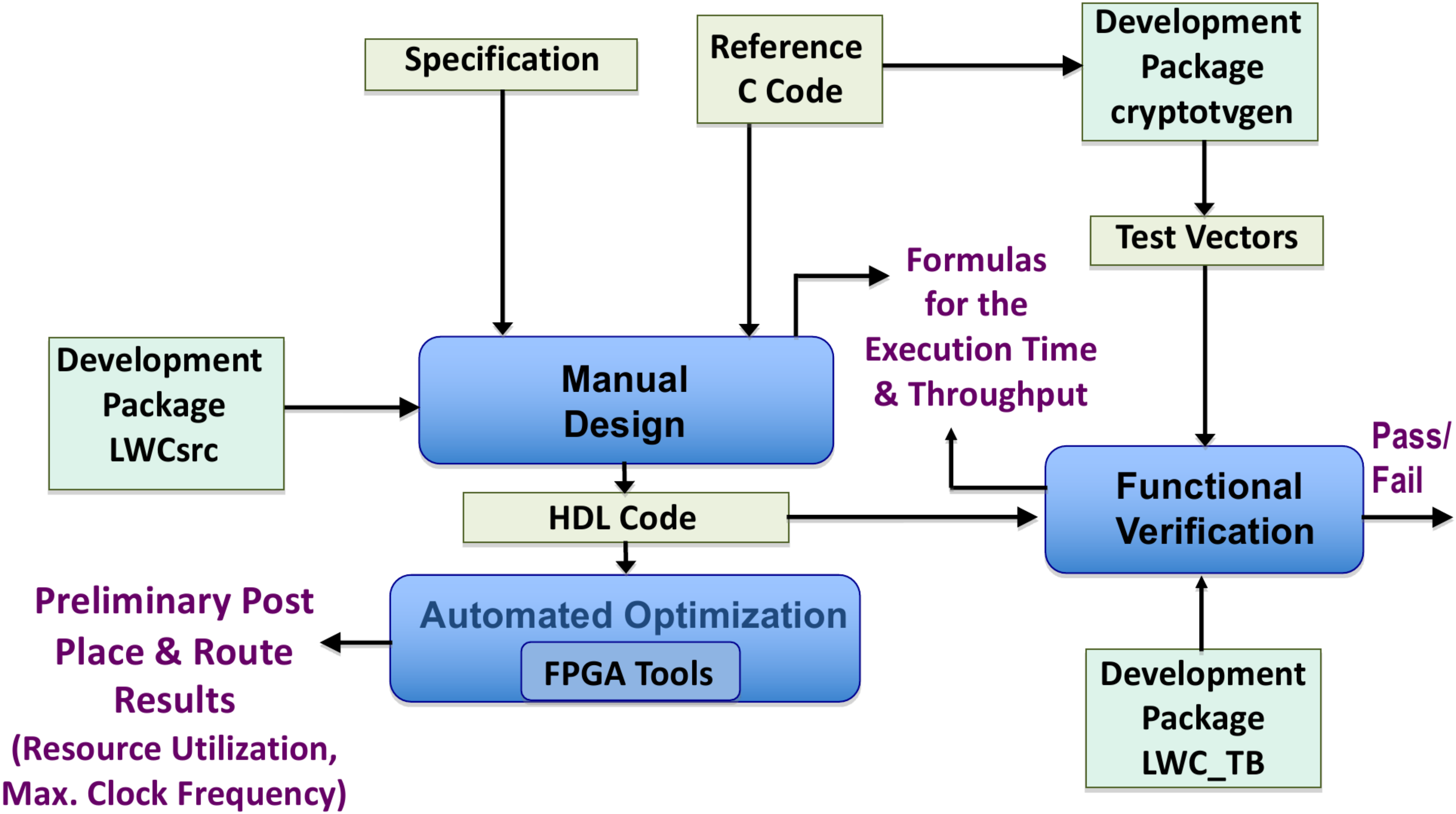


Development Package Source Code

- PreProcessor
  - Parsing segment headers
  - Loading keys
  - Passing input blocks to CryptoCore
  - Keeping track of number of data bytes left to process
- PostProcessor
  - Clearing any portions of output words not belonging to ciphertext or plaintext
  - Generating the header for output data blocks
  - Generating the status block with results of authentication
- VHDL code of the PreProcessor, PostProcessor, and Header FIFO is provided in Development Package
- Development Packages supports bus widths of
  - Input width *w* vs internal width *ccw*:
  - *sw* = *w* (for *w* = 8, 16, 32)

External *w*	Internal *ccw*
8	8
16	16
32	8, 16, 32

Design Methodology



Specification

Reference C Code

Development Package cryptotvgen

Test Vectors

Development Package LWCsrc

Manual Design

Formulas for the Execution Time & Throughput

HDL Code

Functional Verification

Pass/ Fail

Automated Optimization

FPGA Tools

Preliminary Post Place & Route Results (Resource Utilization, Max. Clock Frequency)

Development Package LWC_TB

Dummy CryptoCore

- Example design of a lightweight dummy authenticated cipher

$$CT_i = PT_i \oplus i \oplus Key \oplus Npub \quad CT_m = \mathrm{Trunc}(PT_m \oplus i \oplus Key \oplus Npub, PT_m)$$
$$PT_i = CT_i \oplus i \oplus Key \oplus Npub \quad PT_m = \mathrm{Trunc}(CT_m \oplus i \oplus Key \oplus Npub, CT_m)$$
$$\text{for } i = 1 \ldots m-1$$

$$Tag = Key \oplus Npub \oplus Len \oplus \bigoplus_{i=1}^{n-1} AD_i \oplus \mathrm{Pad}(AD_n) \oplus \bigoplus_{i=1}^{m-1} PT_i \oplus \mathrm{Pad}(PT_m)$$

- Example design of a lightweight dummy hash function

$$Hash\_Value = \bigoplus_{i=1}^{m-1} HASH\_MSG_i \oplus \mathrm{Pad}(HASH\_MSG_m)$$

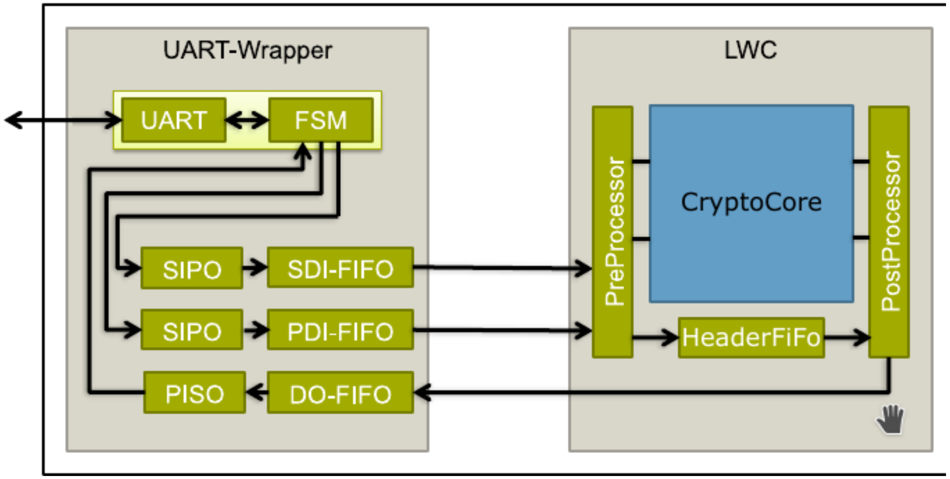
- Dummy CryptoCore supports cww=ccsw=8, 16, 32

Test Vector Generator and Universal Testbench

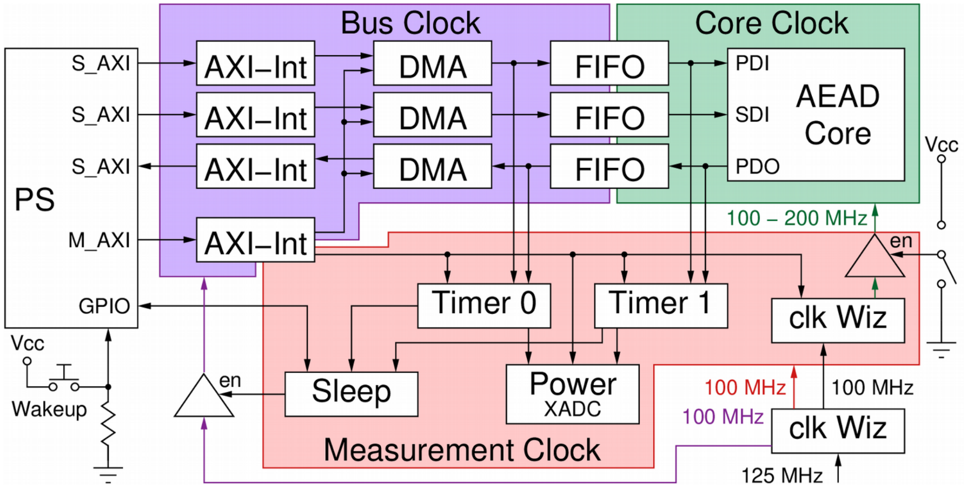
- *cryptotvgen* is a Python app that lets users easily generate test vectors for multiple test cases:
  - Single AD/Plaintext/Ciphertext/Hash Message block
  - Random inputs with custom selected sizes
  - Empty AD/Plaintext/Ciphertext/Hash Message
  - Various, randomly selected sizes of AD, Plaintext, Ciphertext, and Hash Message.
- Universal Testbench *LWC_TB*
  - supports any LWC core following the LWC HW API, and
  - allows simulation of wait states on inputs.

Experimental Testing

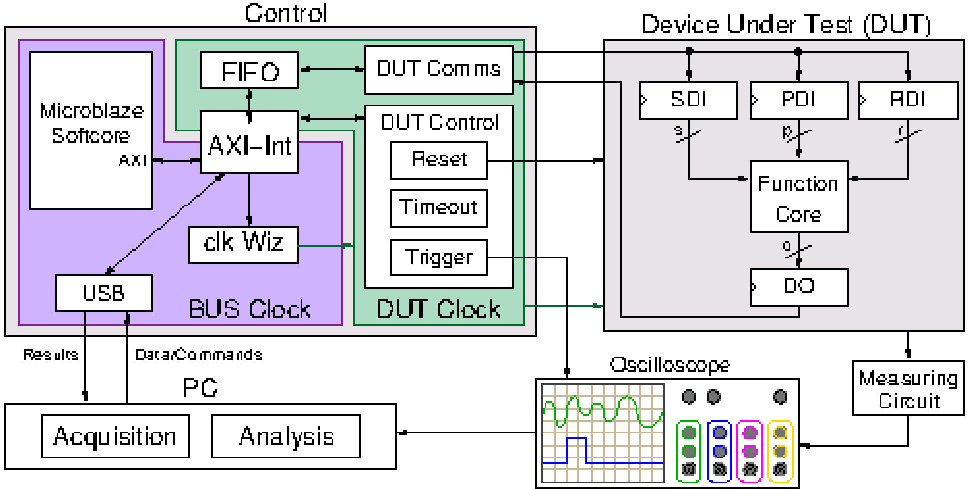
- UART based Framework

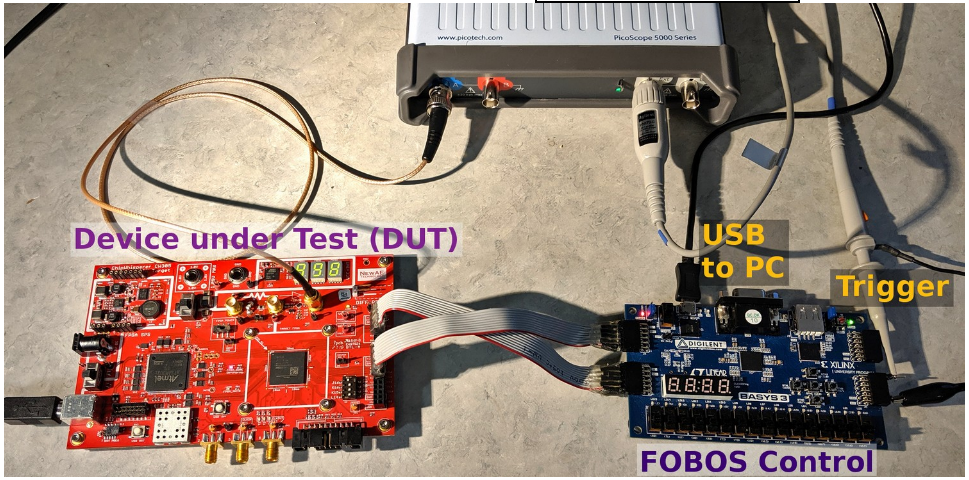


- PYNQ based Framework



- Side-Channel Analysis Framework (FOBOS 2)





Conclusions

Conclusions

- Complete Hardware API for lightweight cryptography including
  - Interface
  - Communications Protocol
- Comments from lwc-forum were incorporated.
- LWC Hardware API, Development Package, and Implementer’s Guide publicly available since October 14th, 2019.
  - Validated with implementations, e.g., Gimli, COMET CHAM 128, SpoC, Spook, GIFT-COFB
- Design with LWC Hardware API supported through:
  - Detailed specification,
  - Universal testbench and test vector generation,
  - ProProcessor and PostProcessor in VHDL,
  - Dummy cipher core,
  - Availability of experimental testing platforms.

Recommendation

- We would like to kindly ask NIST for the endorsement of the proposed hardware benchmarking framework.
- We suggest that NIST should
  - Enforce the submission of hardware description language code compliant with the proposed API.
  - Set the deadline for submissions to middle of Round 2.
- We would be happy to
  - Provide technical support to any Round 2 submission team regarding the Development Package and its documentation.
  - Take responsibility for benchmarking compliant implementations using Xilinx and Intel FPGAs.

Questions? Comments? Suggestions?

All resources available at

https://cryptography.gmu.edu/athena